| SDS | Page No. |
|---|---|
| Date | / / |

# ML

Name :- PREKSHA A. PATEL
Sapid :- 60004210126
Branch :- Computer Engineering
Div :- C2 , Batch :- 1

## Experiment no. 8

**Aim :-** To implement Support Vector Machine

**Theory :-**

SVM or Support Vector Machine is a linear model for classification and regression problems. It can solve linear and non-linear problems and work well for many practical problems. The idea of SVM is simple: The algorithm creates a line or a hyperplane which separates the data into classes. At first approximation what SVMs do is to find a separating line (or hyperplane) between data of two classes. SVM is an algorithm that takes the data as an input and outputs a line that separates those classes if possible. According to the SVM algorithm we find the points closest to the line from both the classes. These points are called support vectors. Now, we compute the distance between the line and the support vectors. The distance is called the margin. Our goal is to maximize the margin. The hyperplane for which the margin is maximum is the optimal hyperplane. Thus, SVM tries to decide boundary in such a way that the separation between the two classes (that street) is as wide as possible.
For non-linear data, we can classify data by adding an extra dimension to it so that it becomes linearly separable and then projecting the decision boundary back to original dimensions using mathematical transformation.

P.T.O.

SVM has several advantages over other classification algorithms such as logistic regression, decision trees, and random forests. It can handle high-dimensional data efficiently, has a strong theoretical foundation, and can be used for both classification and regression tasks. SVM can also handle non-linear data efficiently using kernel functions.

Kernels in SVMs are functions that allow SVMs to operate in high-dimensional spaces without explicitly computing the coordinates of the data in the space. Various kernels in SVMs enable the algorithm to capture complex relationships in the data and make non-linear decision boundaries. Some commonly used kernels include:

1. Linear Kernel :-

The linear kernel computes the dot product between feature vectors in the original feature space, efficiently creating a linear decision boundary. It is suitable for linearly separable data.

2. Polynomial Kernel :-

The polynomial kernel maps the input features into a higher-dimensional space using polynomial functions. It allows SVMs to capture non-linear relationships between features and is suitable for data with non-linear decision boundaries.

3. Radial Basis Function (RBF) Kernel :-

The RBF kernel (also known as Gaussian kernel) maps data into an infinite-dimensional space based on the similarity between feature vectors. It is capable of capturing complex patterns in the data and is widely used due to its flexibility.

4. Sigmoid Kernel :-

The sigmoid kernel is based on the hyperbolic tangent function and is suitable for data that is not linearly separable. It maps the input features into ~~an~~ a higher-dimensional space and can handle non-linear decision boundaries.

Conclusion :

In conclusion, the experiment aimed to implement a support vector Machine (SVM) for classification tasks. By utilizing different kernels such as linear, polynomial, radial basis function (RBF), and sigmoid, the SVM was able to effectively learn complex patterns and create decision boundaries in the feature space. Through experimentation and parameter tuning, the SVM demonstrated its versatility and robustness in handling various types of data, achieving high accuracy in classifying different classes. The implementation of SVM highlights its capability of as a powerful machine learning algorithm for classification tasks, offering flexibility, and performance in real-world applications.

NAME :-PREKSHA ASHOK PATEL

BRANCH:-COMPUTER ENGINEERING

SAPID:-60004210126

DIV:-C2 ; BATCH:-1

MACHINE LEARNING

EXPERIMENT NO.8

USING LINEAR DATASET:-

CODE:-

```
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

from sklearn.datasets import load_breast_cancer

df = load_breast_cancer()

X = df.data[:, [0, 1]]
y = df.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

svm_linear = SVC(kernel='linear')
svm_poly = SVC(kernel='poly', degree=3)
svm_rbf = SVC(kernel='rbf')
svm_sigmoid = SVC(kernel='sigmoid')

svm_linear.fit(X_train, y_train)
svm_poly.fit(X_train, y_train)
svm_rbf.fit(X_train, y_train)
svm_sigmoid.fit(X_train, y_train)

y_pred_linear = svm_linear.predict(X_test)
y_pred_poly = svm_poly.predict(X_test)
y_pred_rbf = svm_rbf.predict(X_test)
y_pred_sigmoid = svm_sigmoid.predict(X_test)


print("Linear Kernel:")
print("Accuracy:", accuracy_score(y_test, y_pred_linear))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_linear))
print("Classification Report:\n", classification_report(y_test, y_pred_linear))

print("\nPolynomial Kernel:")
print("Accuracy:", accuracy_score(y_test, y_pred_poly))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_poly))
print("Classification Report:\n", classification_report(y_test, y_pred_poly))
```

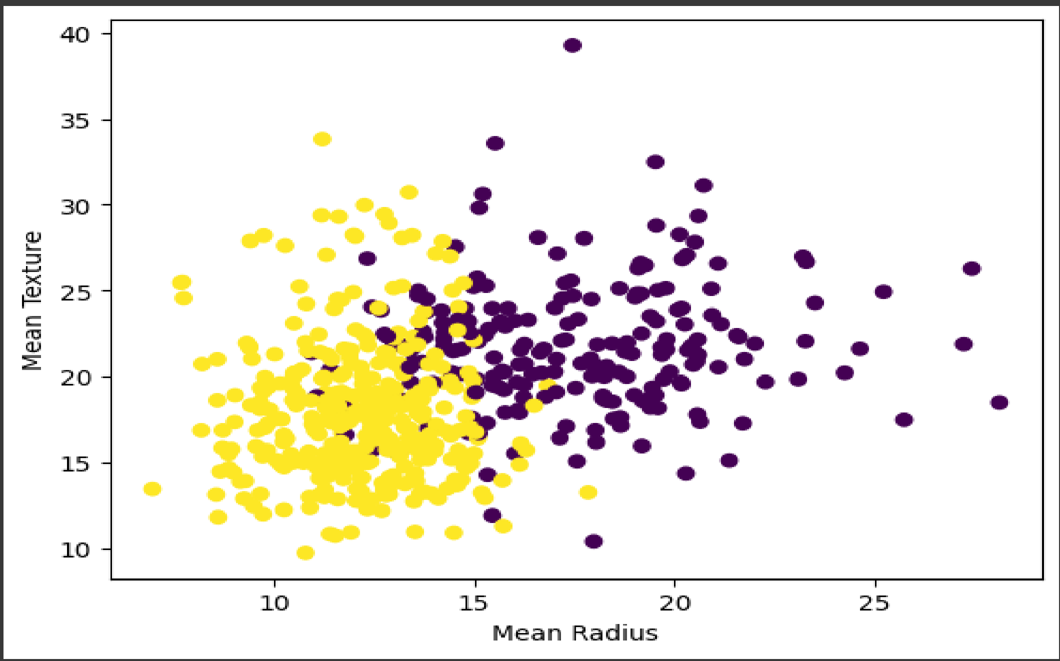
```
print("\nRBF Kernel:")
print("Accuracy:", accuracy_score(y_test, y_pred_rbf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rbf))
print("Classification Report:\n", classification_report(y_test, y_pred_rbf))

print("\nSigmoidal Kernel:")
print("Accuracy:", accuracy_score(y_test, y_pred_sigmoid))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_sigmoid))
print("Classification Report:\n", classification_report(y_test, y_pred_sigmoid))
```

OUTPUT:-

```
Linear Kernel:
Accuracy: 0.9064327485380117
Confusion Matrix:
 [[ 52  11]
 [  5 103]]
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.83      0.87        63
           1       0.90      0.95      0.93       108

    accuracy                           0.91       171
   macro avg       0.91      0.89      0.90       171
weighted avg       0.91      0.91      0.91       171


Polynomial Kernel:
Accuracy: 0.9064327485380117
Confusion Matrix:
 [[ 54   9]
 [  7 101]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.86      0.87        63
           1       0.92      0.94      0.93       108

    accuracy                           0.91       171
   macro avg       0.90      0.90      0.90       171
weighted avg       0.91      0.91      0.91       171
```

```
RBF Kernel:
Accuracy: 0.9005847953216374
Confusion Matrix:
 [[ 51  12]
 [  5 103]]
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.81      0.86        63
           1       0.90      0.95      0.92       108

    accuracy                           0.90       171
   macro avg       0.90      0.88      0.89       171
weighted avg       0.90      0.90      0.90       171


Sigmoidal Kernel:
Accuracy: 0.631578947368421
Confusion Matrix:
 [[  0  63]
 [  0 108]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        63
           1       0.63      1.00      0.77       108

    accuracy                           0.63       171
   macro avg       0.32      0.50      0.39       171
weighted avg       0.40      0.63      0.49       171
```

```python
import matplotlib.pyplot as plt

plt.scatter(X[:, 0], X[:, 1], c=y)
plt.xlabel('Mean Radius')
plt.ylabel('Mean Texture')
plt.show()
```




USING NON-LINEAR DATASET:-

```python
[12] import sklearn
     from sklearn.datasets import make_moons
     from sklearn.metrics import confusion_matrix
     from sklearn.svm import SVC
     from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

     # Generate moons data
     X, y = make_moons(n_samples=200, noise=0.1)

     # Define kernels to test
     kernels = ['linear', 'poly', 'rbf', 'sigmoid']

     # Loop through each kernel
     for kernel in kernels:
       # Create SVC model with chosen kernel
       clf = SVC(kernel=kernel)

       # Train the model
       clf.fit(X, y)

       # Make predictions
       y_pred = clf.predict(X)

       # Evaluate model performance
       accuracy = accuracy_score(y, y_pred)
       confusion_matrix = sklearn.metrics.confusion_matrix(y_true=y, y_pred=y_pred)
       classification_report = sklearn.metrics.classification_report(y, y_pred)

       # Print results for each kernel
       print(f"\nKernel: {kernel}")
       print(f"Accuracy: {accuracy:.4f}")
       print(f"Confusion Matrix:\n{confusion_matrix}")
       print(f"Classification Report:\n{classification_report}")
```

```
Kernel: linear
Accuracy: 0.8850
Confusion Matrix:
[[91  9]
 [14 86]]
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.91      0.89       100
           1       0.91      0.86      0.88       100

    accuracy                           0.89       200
   macro avg       0.89      0.89      0.88       200
weighted avg       0.89      0.89      0.88       200


Kernel: poly
Accuracy: 0.9450
Confusion Matrix:
[[90 10]
 [ 1 99]]
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.90      0.94       100
           1       0.91      0.99      0.95       100

    accuracy                           0.94       200
   macro avg       0.95      0.95      0.94       200
weighted avg       0.95      0.94      0.94       200
```

```
Kernel: rbf
Accuracy: 0.9900
Confusion Matrix:
[[ 98   2]
 [  0 100]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       100
           1       0.98      1.00      0.99       100

    accuracy                           0.99       200
   macro avg       0.99      0.99      0.99       200
weighted avg       0.99      0.99      0.99       200


Kernel: sigmoid
Accuracy: 0.6500
Confusion Matrix:
[[64 36]
 [34 66]]
Classification Report:
              precision    recall  f1-score   support

           0       0.65      0.64      0.65       100
           1       0.65      0.66      0.65       100

    accuracy                           0.65       200
   macro avg       0.65      0.65      0.65       200
weighted avg       0.65      0.65      0.65       200
```

```
# Generate moons data
X, y = make_moons(n_samples=200, noise=0.1)

# Get feature names
feature_names = ['x1', 'x2']

# Create scatter plot
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.xlabel(feature_names[0])
plt.ylabel(feature_names[1])
plt.show()
```