VIVEKANAND EDUCATION SOCIETY'S INSTITUTE OF TECHNOLOGY

Hashu Advani Memorial Complex, Collector's Colony, R C Marg, Chembur, Mumbai-400074

V.E.S.
Since 1962

Department of Artificial Intelligence and Data Science

Subject: **Class:** D11AD **Semester:** VI

Roll No.: 26	Name: Dyotak Kachare			
Exp No.: 5	Title: Non - linear SVM			
DOP:	14/02		DOS:	21/02
GRADE		LAB OUTCOME:	SIGNATURE: [illegible]	

Name: ______ Class & Div. : ______ Page No.: ______

Subject: ______ Topic: ______ Date: ______

ML experiment - 5

Aim -
Kernel trick for non linear SVM.

Theory -

When we can easily seperate data with hyperplane by drawing a straight line is called linear SVM.
when we cannot seperate data with a straight line we use Non-linear SVM. In this, we have kernel functions.
It transforms data into another dimension so that the data can be classified.

Kernel tricks

Polynomial kernel:
A polynomial kernel is a kind of SVM kernel that uses a polynomial function to map the data into a higher dimensional space.
It does this by taking the dot product of the data points in the original space and the polynomial function in the new space.

$K(x_1, x_2) = (x_1^T x_2 + c)^d$

SIDDHII

RBF Kernels

The squared euclidian distance is multiplied by the gama parameter and then finding the exponent of the whole.

where
1. σ is the variance, & hyperparameter
2. $\|X_1 - X_2\|$ is the Euclidian (L_2 Norm) distance between two points X_1 & X_2

$$K(X_1, X_2) = \exp\left(-\frac{\|X_1 - X_2\|^2}{2\sigma^2}\right)$$

[illegible]

Conclusion

Thus, we have sucessfully implemented Non linear SVM.

[14]:
```python
from sklearn.datasets import load_breast_cancer
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

[3]:
```python
data = load_breast_cancer()
X, y = data['data'], data['target']
```

[4]:
```python
print(X)
print(y)
```

```
[[1.799e+01 1.038e+01 1.228e+02 … 2.654e-01 4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 … 1.860e-01 2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 … 2.430e-01 3.613e-01 8.758e-02]
 …
 [1.660e+01 2.808e+01 1.083e+02 … 1.418e-01 2.218e-01 7.820e-02]
 [2.060e+01 2.933e+01 1.401e+02 … 2.650e-01 4.087e-01 1.240e-01]
 [7.760e+00 2.454e+01 4.792e+01 … 0.000e+00 2.871e-01 7.039e-02]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1]
```

[5]:
```python
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

[6]:
```python
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2)
```

[22]:
```python
model = SVC(probability=True)
model.fit(X_train, y_train)
```

[22]:
```
SVC(probability=True)
```

[9]:
```python
print(model.score(X_test, y_test))
```

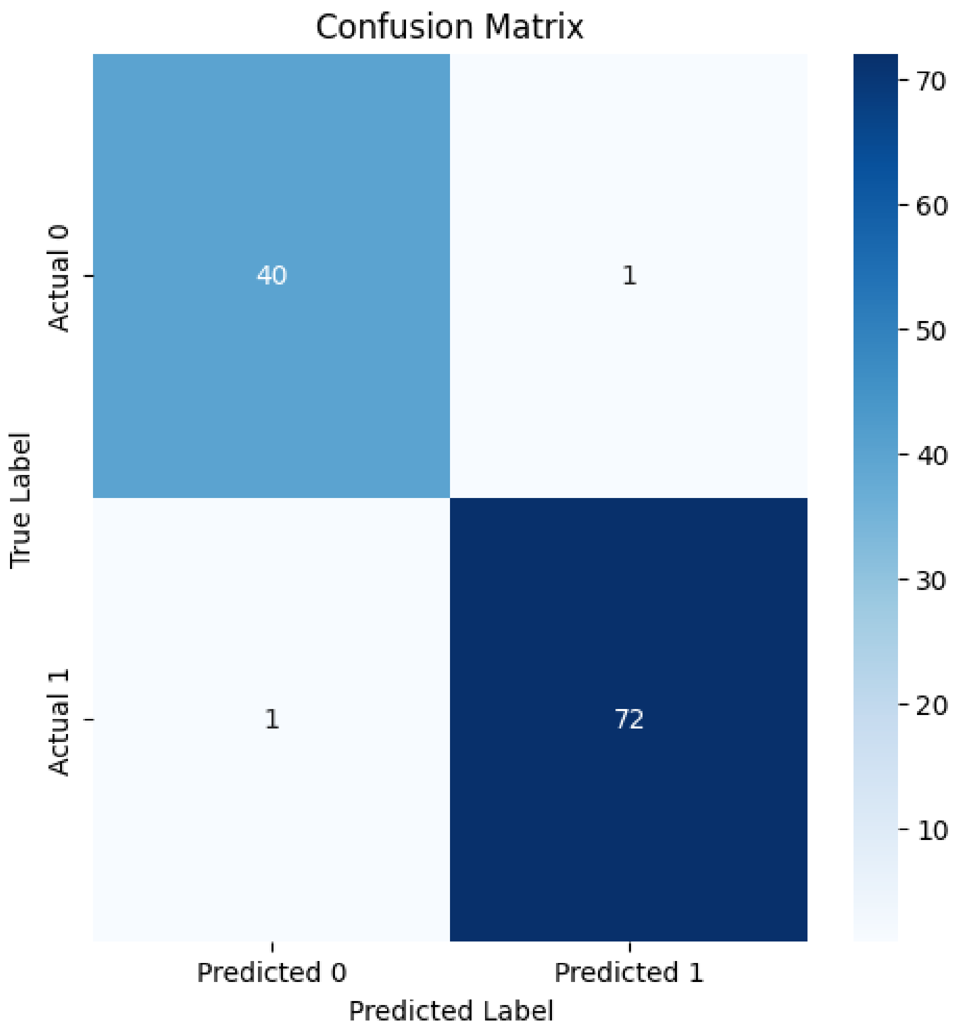
```
0.9824561403508771
```

[15]:
```python
y_predict = model.predict(X_test)
```

```
[32]: for i in range(20):
          print(f"Actual value: {y_test[i]} | Predicted value: {y_predict[i]}")
```

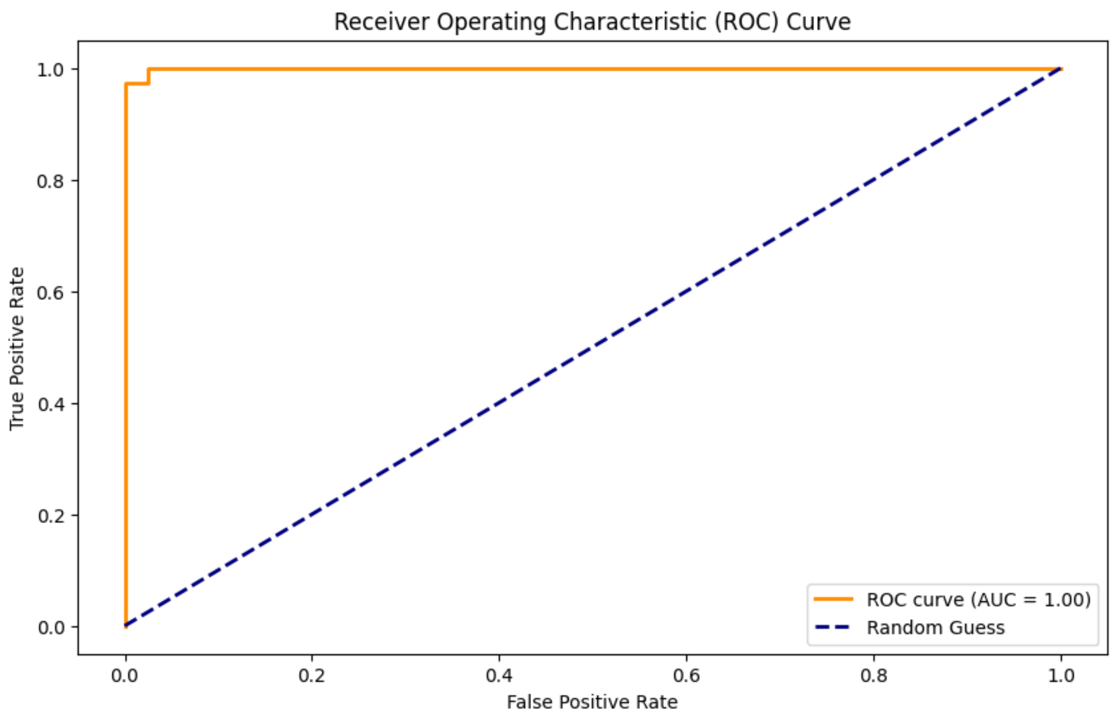
```
Actual value: 1 | Predicted value: 1
Actual value: 1 | Predicted value: 1
Actual value: 1 | Predicted value: 1
Actual value: 1 | Predicted value: 1
Actual value: 1 | Predicted value: 1
Actual value: 0 | Predicted value: 0
Actual value: 1 | Predicted value: 1
Actual value: 1 | Predicted value: 1
Actual value: 1 | Predicted value: 1
Actual value: 1 | Predicted value: 1
Actual value: 1 | Predicted value: 1
Actual value: 0 | Predicted value: 0
Actual value: 0 | Predicted value: 0
Actual value: 0 | Predicted value: 0
Actual value: 0 | Predicted value: 0
Actual value: 1 | Predicted value: 1
Actual value: 1 | Predicted value: 1
Actual value: 0 | Predicted value: 0
Actual value: 0 | Predicted value: 0
Actual value: 0 | Predicted value: 0
```

```
[19]: cm = confusion_matrix(y_test, y_predict)

      plt.figure(figsize=(6, 6))
      sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted 0',
        ↪'Predicted 1'], yticklabels=['Actual 0', 'Actual 1'])
      plt.xlabel('Predicted Label')
      plt.ylabel('True Label')
      plt.title('Confusion Matrix')
      plt.show()
```



```
[25]: y_probs = model.predict_proba(X_test)[:, 1]

      fpr, tpr, thresholds = roc_curve(y_test, y_probs)
      roc_auc = auc(fpr, tpr)
```

```
[26]: plt.figure(figsize=(10, 6))
      plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = {:.2f})'.
        ↪format(roc_auc))
      plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Guess')
      plt.xlabel('False Positive Rate')
      plt.ylabel('True Positive Rate')
      plt.title('Receiver Operating Characteristic (ROC) Curve')
      plt.legend()
      plt.show()
```


Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (AUC = 1.00)
Random Guess

```
[ ]:
```