LAB # 06

SUPPORT VECTOR MACHINE (SVM) CLASSIFIER

LAB TASKS

1. Load a dataset for classification (e.g., Parkinson disease, Breast Cancer dataset).

```python
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
```

```python
data = load_breast_cancer()
x = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)
```

```python
x.shape
```

```
(569, 30)
```

```python
y.shape
```

```
(569,)
```

```python
x.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...

	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
0	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
1	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
2	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758
3	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300
4	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678

5 rows × 30 columns

2. Apply data preprocessing (handle missing values, encode categorical data).

```python
x.isnull().sum()
```

	0
mean radius	0
mean texture	0
mean perimeter	0
mean area	0
mean smoothness	0
mean compactness	0
mean concavity	0
mean concave points	0
mean symmetry	0
mean fractal dimension	0
radius error	0
texture error	0
perimeter error	0
area error	0
smoothness error	0
compactness error	0
concavity error	0
concave points error	0
symmetry error	0
fractal dimension error	0
worst radius	0

3. Split the dataset into training and testing sets.

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

```
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

4. Apply Grid search to find the optimal parameters

```
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': ['scale', 'auto']
}
```

```
grid = GridSearchCV(SVC(), param_grid, cv=5, scoring='accuracy')
grid.fit(x_train, y_train)
```

```
GridSearchCV
best_estimator_:
SVC
SVC
```

```
print("\nBest Parameters found by Grid Search:")
print(grid.best_params_)
```

```
Best Parameters found by Grid Search:
{'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
```

```
best_model = grid.best_estimator_
y_pred = best_model.predict(x_test)
```

5. Use those parameters to make predictions on the test set.

```
print("\nBest Parameters found by Grid Search:")
print(grid.best_params_)
```

```
Best Parameters found by Grid Search:
{'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
```

```
best_model = grid.best_estimator_
y_pred = best_model.predict(x_test)
```

6. Evaluate performance using accuracy, precision, recall, and F1-score.

```
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

```
print("\nModel Evaluation Metrics:")
print('Accuracy:',acc)
print('Precision:',prec)
print('Recall:',rec)
print('F1-Score:',f1)
```

```
Model Evaluation Metrics:
Accuracy: 0.9824561403508771
Precision: 0.9726027397260274
Recall: 1.0
F1-Score: 0.9861111111111112
```

Visualize the Confusion Matrix