SVM - Multi Class Classification

Aim:

To implement SVM multi class classification

SVM Classification:

Support Vector Machine (SVM) is a supervised learning technique used for binary classification. It finds the optimal hyperplane that best separates data points into two classes. SVM aims to maximize the margin between the two classes, ensuring better generalization. The classification process involves the following steps:

1. **Initialization**: Define the dataset and choose a kernel for classification.

2. **Hyperplane Selection**: Identify the optimal decision boundary that maximizes the margin between the two classes.

3. **Support Vectors Identification**: Determine the critical data points (support vectors) that define the margin and influence the hyperplane's position.

4. **Optimization**: Use an optimization algorithm, such as Sequential Minimal Optimization (SMO), to find the best hyperplane.

5. **Prediction**: Classify new data points based on their position relative to the hyperplane.

Algorithm:

1. Load the dry bean dataset.
2. Plot the class distribution.
3. Split the dataset into training and testing sets.

4. Train an OvO and OvR SVM classifiers with a linear kernel.

5. Predict labels for the test set.

6. Evaluate performance using a confusion matrix and print the accuracy.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split as tts
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix,
```

```
ConfusionMatrixDisplay
from mlxtend.plotting import plot_decision_regions as pdr

# Loading the dataset
df = pd.read_excel('Dry_Bean_Dataset.xlsx')
df
```

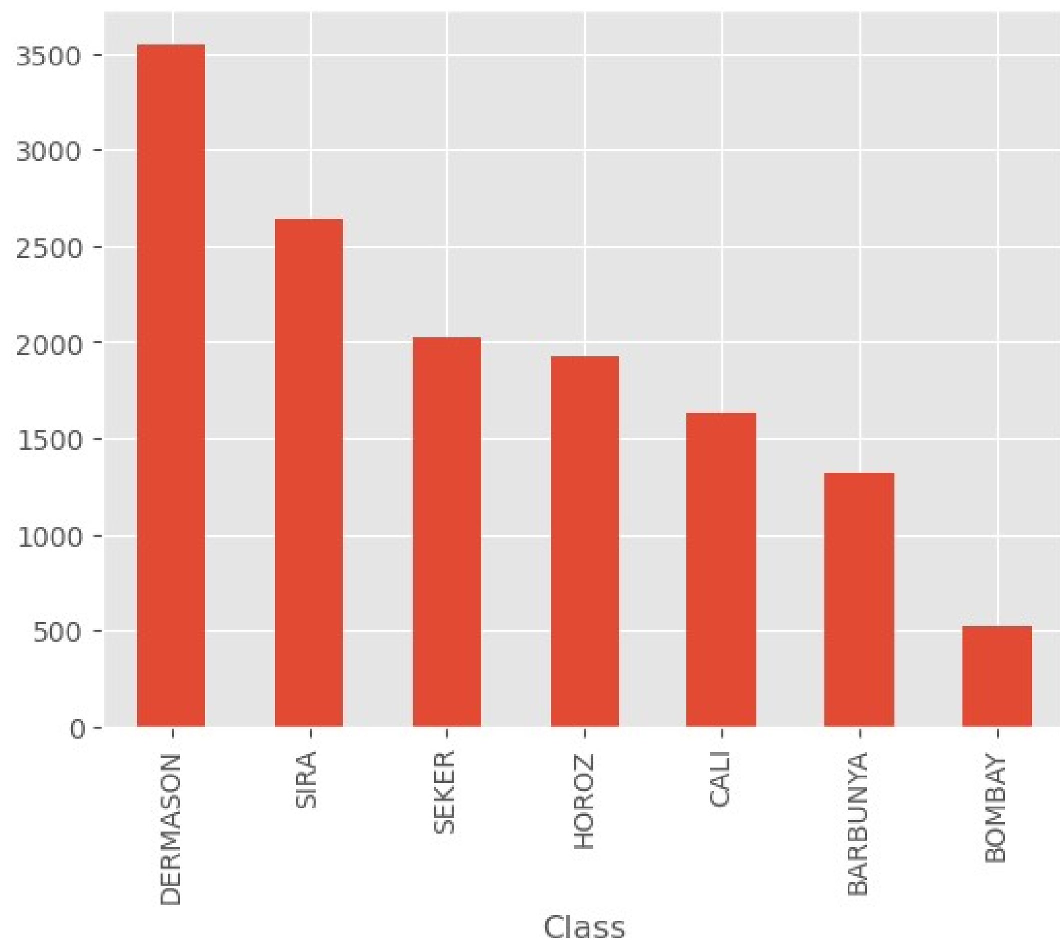
```
        Area  Perimeter  MajorAxisLength  MinorAxisLength  
AspectRation  \
0      28395    610.291       208.178117       173.888747      
1.197191   
1      28734    638.018       200.524796       182.734419      
1.097356   
2      29380    624.110       212.826130       175.931143      
1.209713   
3      30008    645.884       210.557999       182.516516      
1.153638   
4      30140    620.134       201.847882       190.279279      
1.060798   
...      ...        ...              ...              ...            ..
.   
13606  42097    759.696       288.721612       185.944705      
1.552728   
13607  42101    757.499       281.576392       190.713136      
1.476439   
13608  42139    759.321       281.539928       191.187979      
1.472582   
13609  42147    763.779       283.382636       190.275731      
1.489326   
13610  42159    772.237       295.142741       182.204716      
1.619841   

       Eccentricity  ConvexArea  EquivDiameter    Extent  Solidity  
roundness  \
0          0.549812       28715     190.141097  0.763923  0.988856   
0.958027   
1          0.411785       29172     191.272750  0.783968  0.984986   
0.887034   
2          0.562727       29690     193.410904  0.778113  0.989559   
0.947849   
3          0.498616       30724     195.467062  0.782681  0.976696   
0.903936   
4          0.333680       30417     195.896503  0.773098  0.990893   
0.984877   
...             ...         ...            ...       ...       ...   
...   
13606      0.765002       42508     231.515799  0.714574  0.990331   
0.916603   
13607      0.735702       42494     231.526798  0.799943  0.990752   
0.922015   
```

```
13608      0.734065       42569     231.631261  0.729932  0.989899
0.918424
13609      0.741055       42667     231.653248  0.705389  0.987813
0.907906
13610      0.786693       42600     231.686223  0.788962  0.989648
0.888380

       Compactness  ShapeFactor1  ShapeFactor2  ShapeFactor3
ShapeFactor4  \
0         0.913358      0.007332      0.003147      0.834222
0.998724
1         0.953861      0.006979      0.003564      0.909851
0.998430
2         0.908774      0.007244      0.003048      0.825871
0.999066
3         0.928329      0.007017      0.003215      0.861794
0.994199
4         0.970516      0.006697      0.003665      0.941900
0.999166
...            ...           ...           ...           ...
...
13606     0.801865      0.006858      0.001749      0.642988
0.998385
13607     0.822252      0.006688      0.001886      0.676099
0.998219
13608     0.822730      0.006681      0.001888      0.676884
0.996767
13609     0.817457      0.006724      0.001852      0.668237
0.995222
13610     0.784997      0.007001      0.001640      0.616221
0.998180

          Class
0         SEKER
1         SEKER
2         SEKER
3         SEKER
4         SEKER
...         ...
13606  DERMASON
13607  DERMASON
13608  DERMASON
13609  DERMASON
13610  DERMASON

[13611 rows x 17 columns]

df['Class'].unique()
```

```
array(['SEKER', 'BARBUNYA', 'BOMBAY', 'CALI', 'HOROZ', 'SIRA',
'DERMASON'],
      dtype=object)
```

```
# Data Visualisation
df['Class'].value_counts().plot(kind='bar')
```

```
&lt;Axes: xlabel='Class'&gt;
```




```
# Splitting features and target
x = df.iloc[:, :-1].values
y = df['Class'].values

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

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

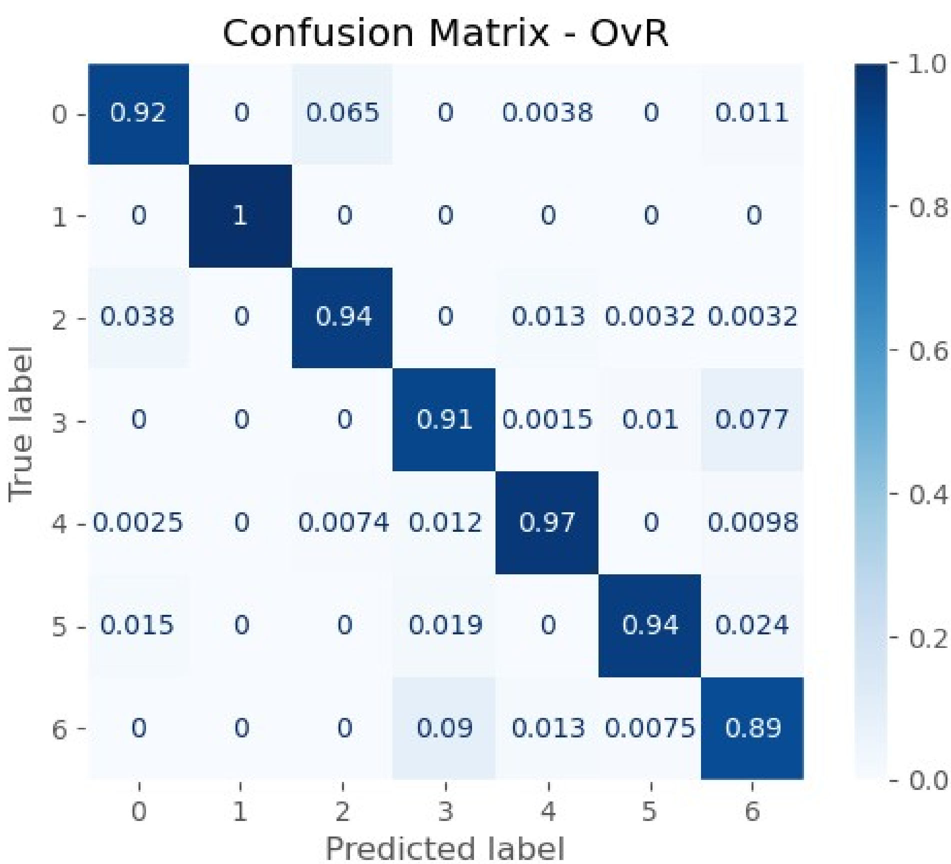
```
y_train = encoder.fit_transform(y_train)
y_test = encoder.transform(y_test)

# Training One-vs-All SVM classifier with a linear kernel
svm_ovr = SVC(kernel='linear', decision_function_shape='ovr')
svm_ovr.fit(x_train, y_train)
```

```
SVC(kernel='linear')
```

```
# Predict using OvA model
y_pred_ovr = svm_ovr.predict(x_test)
accuracy_ovr = accuracy_score(y_test, y_pred_ovr)
accuracy_ovr*100
```

```
92.83878075651855
```

```
# Confusion Matrix for OvA
cm_ovr = confusion_matrix(y_test, y_pred_ovr, normalize='true')
disp_ovr = ConfusionMatrixDisplay(confusion_matrix=cm_ovr,
display_labels=svm_ovr.classes_)
disp_ovr.plot(cmap=plt.cm.Blues)
plt.grid(False)
plt.title('Confusion Matrix - OvR')
plt.show()
```



```
# plt.figure(figsize=(12, 5))
# plt.subplot(1, 2, 1)
# pdr(x_test, y_test.astype(np.int_), clf=svm_ovr, legend=2)
# plt.title("OvR Decision Boundary")

# Train One-vs-One SVM classifier with a linear kernel
svm_ovo = SVC(kernel='linear', decision_function_shape='ovo')
svm_ovo.fit(x_train, y_train)
```

```
SVC(decision_function_shape='ovo', kernel='linear')
```

```
# Predict using OvO model
y_pred_ovo = svm_ovo.predict(x_test)
accuracy_ovo = accuracy_score(y_test, y_pred_ovo)
accuracy_ovo*100
```

```
92.83878075651855
```

```
# pdr(x_test[:, :2], y_test, clf=svm_ovo, legend=2)
# plt.title('Decision Boundary - OvO SVM')
# plt.show()
```