Aim:

To implement SVM binary classification for linearly and non linearly separable data

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 linear 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.

SVM - Binary Classification for Lineary Separable Data

Algorithm:

1. Generate a linearly separable dataset using `make_blobs`.

2. Split the dataset into training and testing sets.

3. Train an SVM classifier with a linear kernel.

4. Predict labels for the test set.

5. Evaluate performance using a confusion matrix.

6. Identify and visualize support vectors.

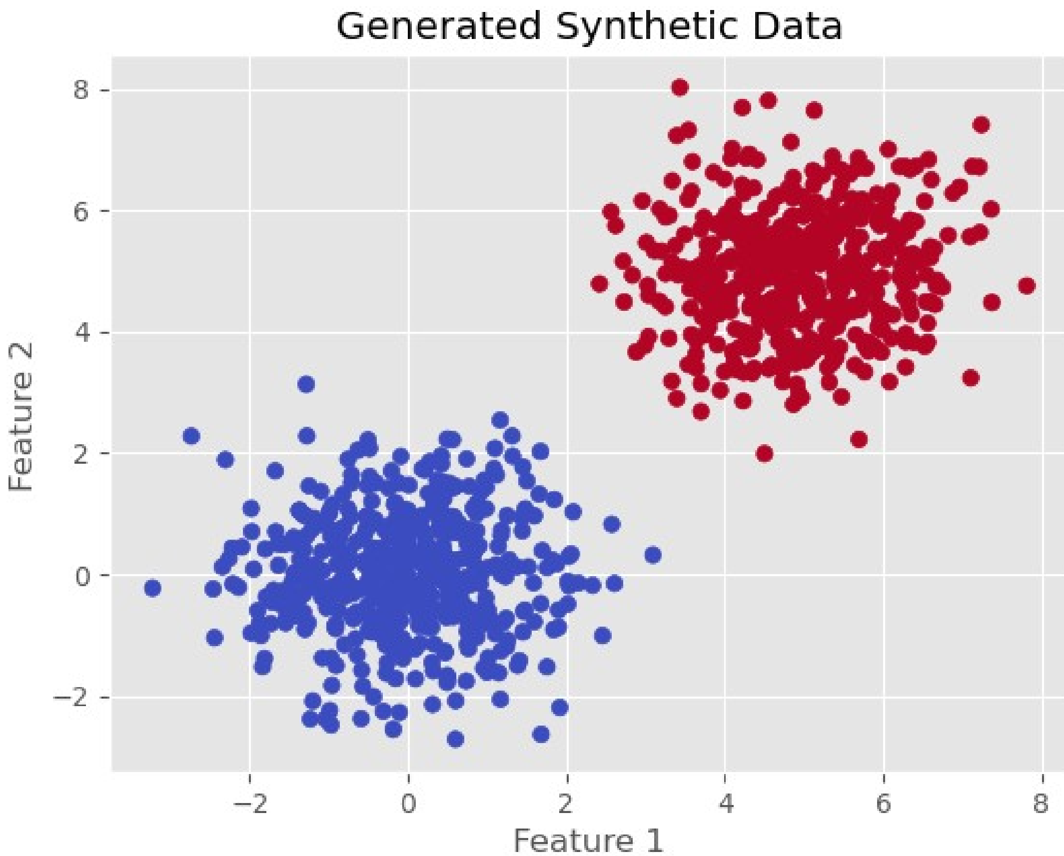
7. Plot the decision boundary with margin lines.

```
import pandas as pd
import numpy as np
```

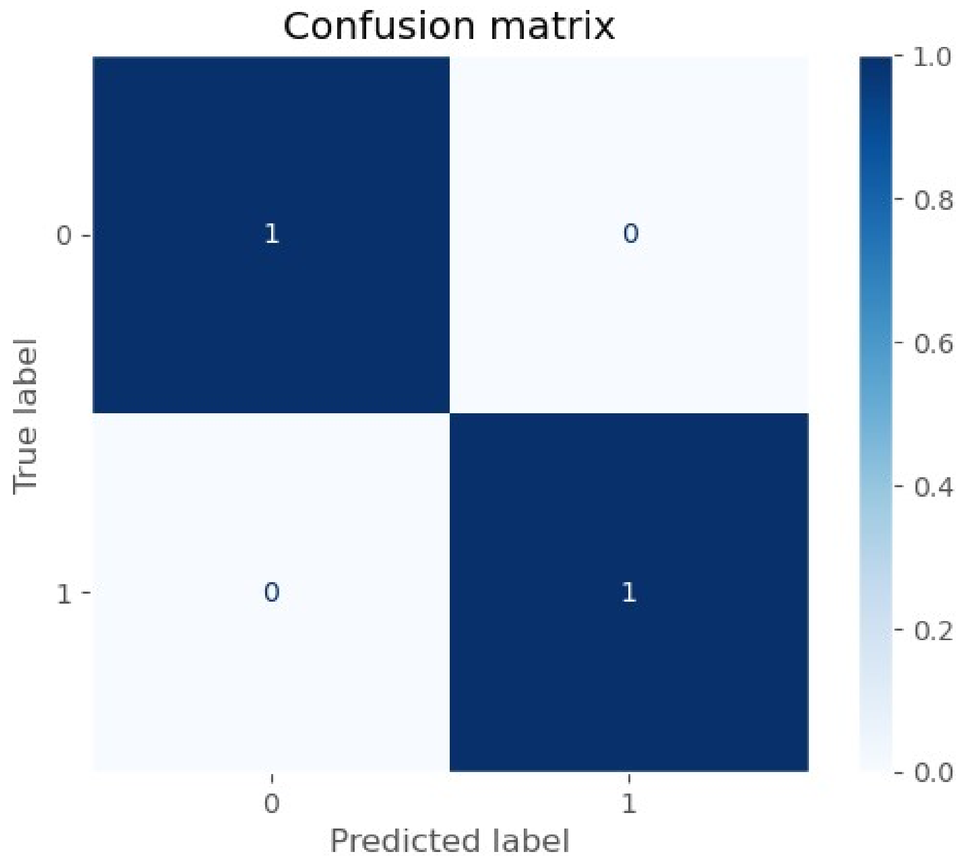
```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split as tts
from sklearn import svm
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from mlxtend.plotting import plot_decision_regions as pdr

# Generating a linearly separable dataset
inputs,targets = make_blobs(n_samples=1000, centers=[(0,0), (5,5)],
n_features=2, cluster_std=1)
df = pd.DataFrame(inputs, columns=['Feature 1', 'Feature 2'])
df['Cluster'] = targets
df
```

```
     Feature 1  Feature 2  Cluster
0     4.272317   6.851848        1
1     5.424737   4.077764        1
2     0.987841   1.081010        0
3     6.306403   4.907130        1
4     4.001077   6.510621        1
..         ...        ...      ...
995   5.191342   4.342608        1
996   0.487113  -0.713754        0
997   7.099982   5.562009        1
998  -0.484090   2.072306        0
999   5.657146   3.943541        1

[1000 rows x 3 columns]
```

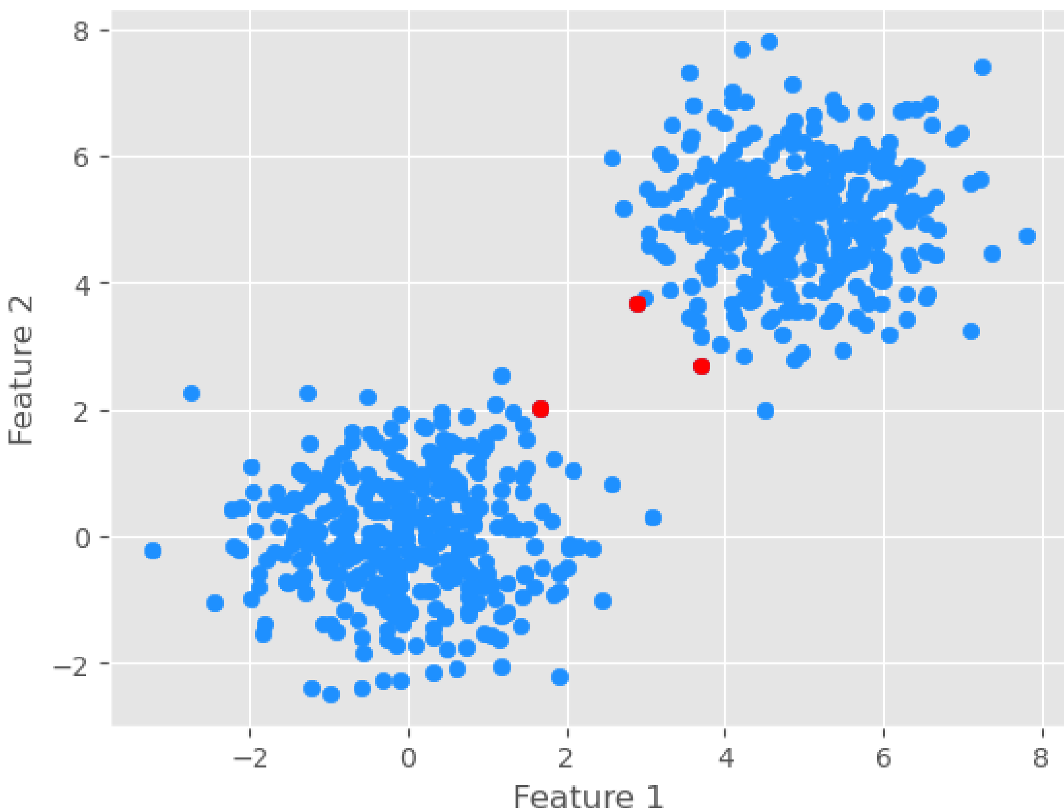
```
# Data Visualisation
plt.style.use('ggplot')
plt.scatter(inputs[:, 0], inputs[:, 1], c=targets, cmap='coolwarm')
plt.title('Generated Synthetic Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



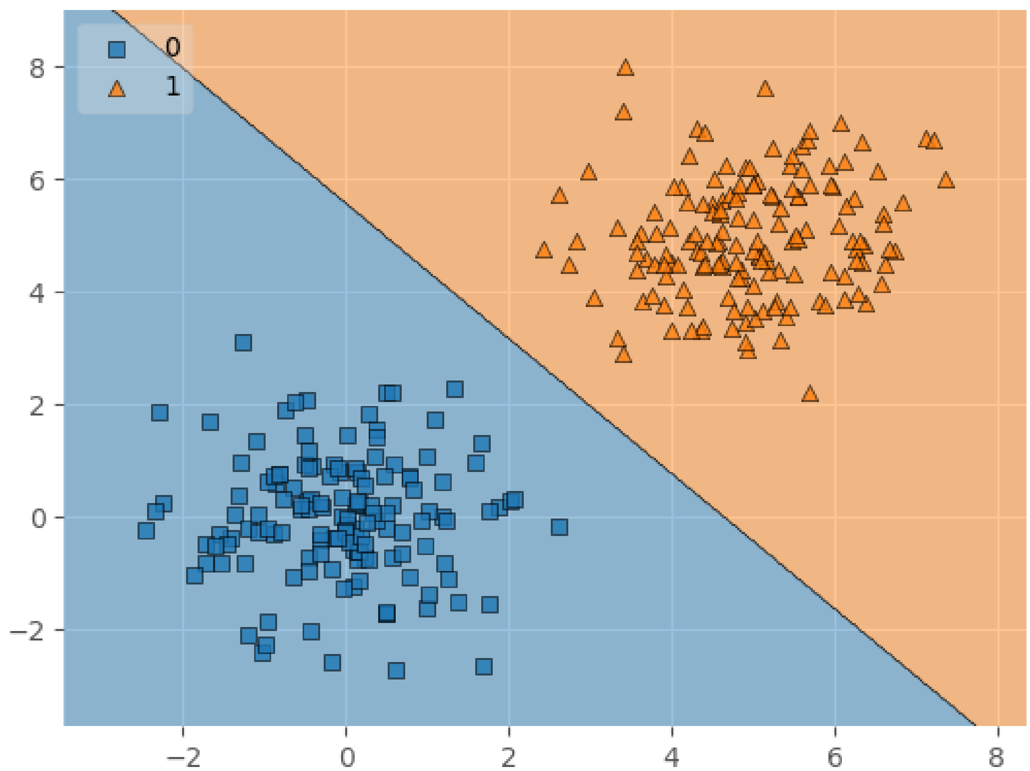

```
# Building the classifier
x_train, x_test, y_train, y_test = tts(inputs, targets, test_size=0.3,
random_state=42)
clf = svm.SVC(kernel='linear')
clf = clf.fit(x_train, y_train)

y_pred = clf.predict(x_test)
cm = confusion_matrix(y_test, y_pred, normalize='true')
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=clf.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.grid(False)
plt.title('Confusion matrix')
plt.show()
```




```python
# Finding the support vectors of the SVM
support_vectors = clf.support_vectors_
plt.scatter(x_train[:,0], x_train[:,1], color='dodgerblue')
plt.scatter(support_vectors[:,0], support_vectors[:,1], color='red')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

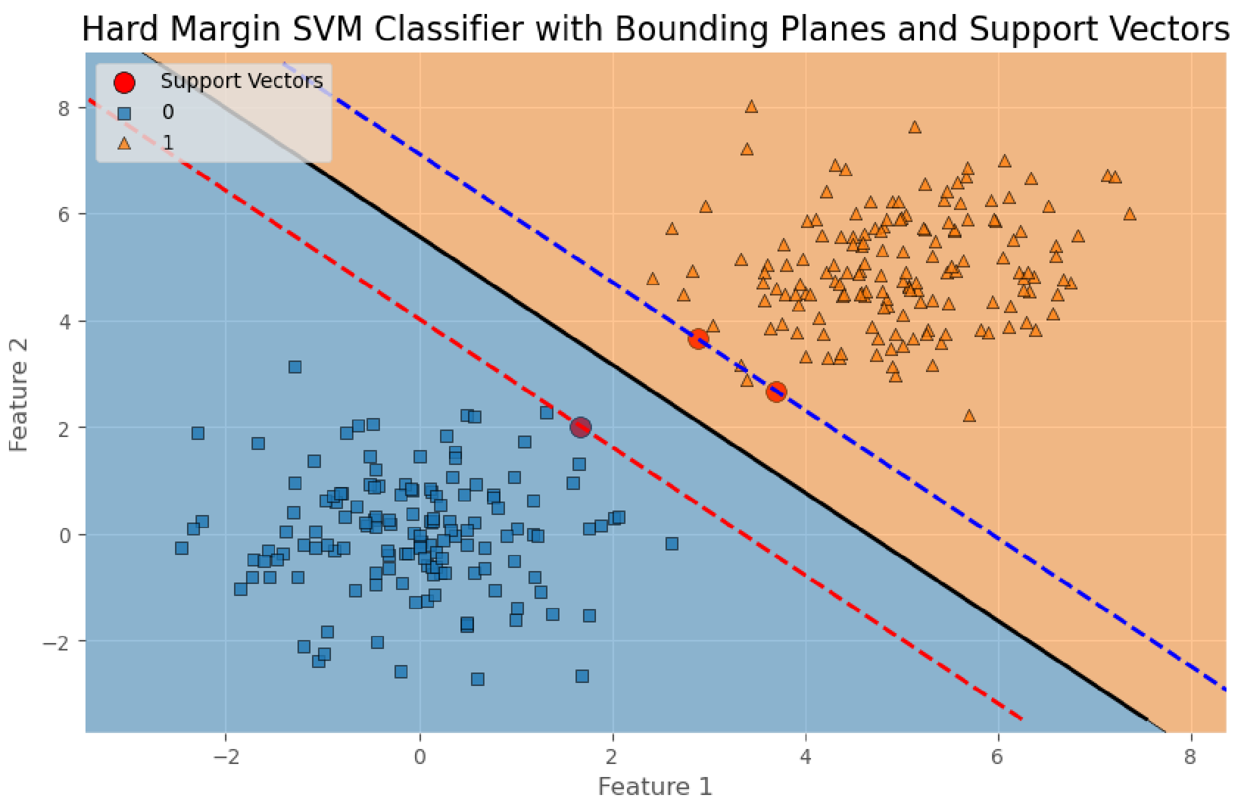



```
# Visualising the decision boundary
pdr(x_test, y_test,clf=clf, legend=2)
plt.show()
```




```
# Plotting the hard margin classifier and the bounding planes

# Get the separating hyperplane and margins
w = clf.coef_[0]
b = clf.intercept_[0]

# Calculate the margin lines (bounding planes) using w and b
xx, yy = np.meshgrid(np.linspace(x_train[:, 0].min() - 1, x_train[:,
0].max() + 1, 300),
                     np.linspace(x_train[:, 1].min() - 1, x_train[:,
1].max() + 1, 300))

# Decision function (hyperplane)
Z = w[0] * xx + w[1] * yy + b
Z = Z.reshape(xx.shape)

# Margin planes (offset by +1 and -1 from the decision boundary)
Z_margin_plus = Z + 1
Z_margin_minus = Z - 1

# Plot decision boundary and margin boundaries
plt.figure(figsize=(10, 6))

# Plot the decision boundary (0 level)
plt.contour(xx, yy, Z, levels=[0], colors='black', linewidths=2)
```

```
# Plot the margin boundaries (+1 and -1 levels)
plt.contour(xx, yy, Z_margin_plus, levels=[0], colors='red',
linestyles='--', linewidths=2)
plt.contour(xx, yy, Z_margin_minus, levels=[0], colors='blue',
linestyles='--', linewidths=2)

# Plot the support vectors (yellow 'X' markers)
plt.scatter(support_vectors[:, 0], support_vectors[:, 1], color='red',
s=100, label='Support Vectors', edgecolors='black')

# Plot the decision regions (already visualized)
pdr(x_test, y_test, clf=clf, legend=2)

# Labels, title, and legend
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("Hard Margin SVM Classifier with Bounding Planes and Support
Vectors", fontsize=16)
plt.legend(loc="best")
plt.show()
```



SVM - Binary Classification for Non Linearly Separable Data

Aim:

To implement SVM binary classification for non linearly separable data

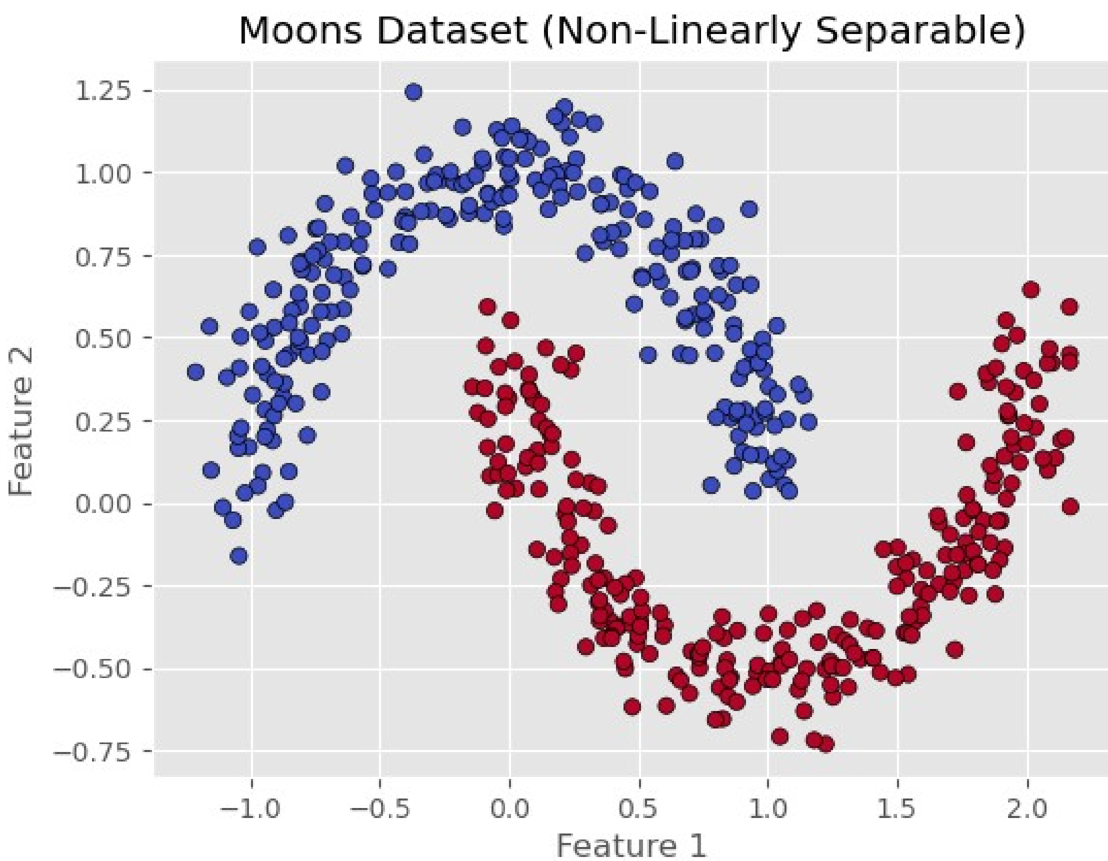
Algorithm:

1. Generate a non linearly separable dataset using make_moons.
2. Split the dataset into training and testing sets.
3. Train an SVM classifier with an RBF kernel.
4. Predict labels for the test set.
5. Evaluate performance using a confusion matrix.
6. Identify and visualize support vectors.
7. Plot the decision boundary with margin lines.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split as tts
from sklearn import svm
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from mlxtend.plotting import plot_decision_regions as pdr

# Generate a non-linearly separable dataset (moons dataset)
inputs, targets = make_moons(n_samples=500, noise=0.1,
random_state=42)

# Convert to DataFrame
df = pd.DataFrame(inputs, columns=['Feature 1', 'Feature 2'])
df['Class'] = targets
df
```

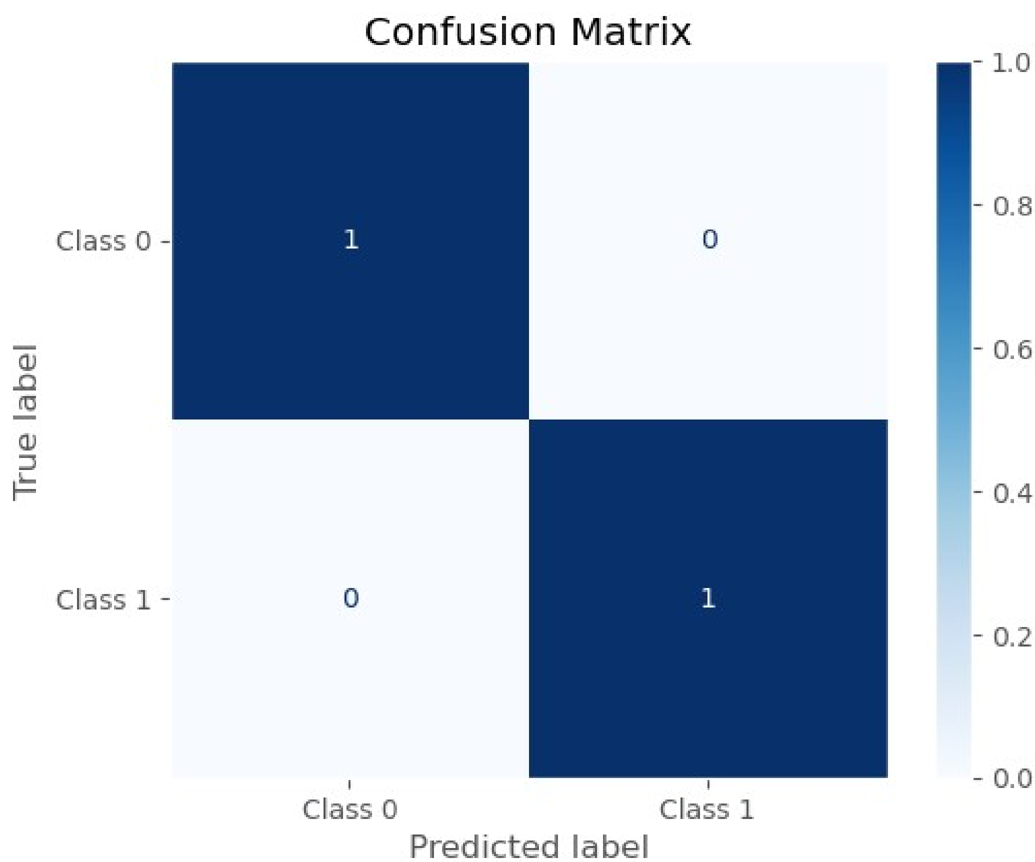
	Feature 1	Feature 2	Class
0	0.830676	-0.409936	1
1	0.798355	0.837612	0
2	1.050468	-0.485162	1
3	-0.258143	0.980008	0
4	0.330682	1.147633	0
..	...	...	...
495	0.248210	0.998040	0

```
496   0.112622   0.119335      1
497   0.508698  -0.286530      1
498   1.547606  -0.344769      1
499   0.177838   1.167971      0

[500 rows x 3 columns]

# Data Visualization
plt.style.use('ggplot')
plt.scatter(inputs[:, 0], inputs[:, 1], c=targets, cmap='coolwarm',
edgecolors='k')
plt.title('Moons Dataset (Non-Linearly Separable)')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



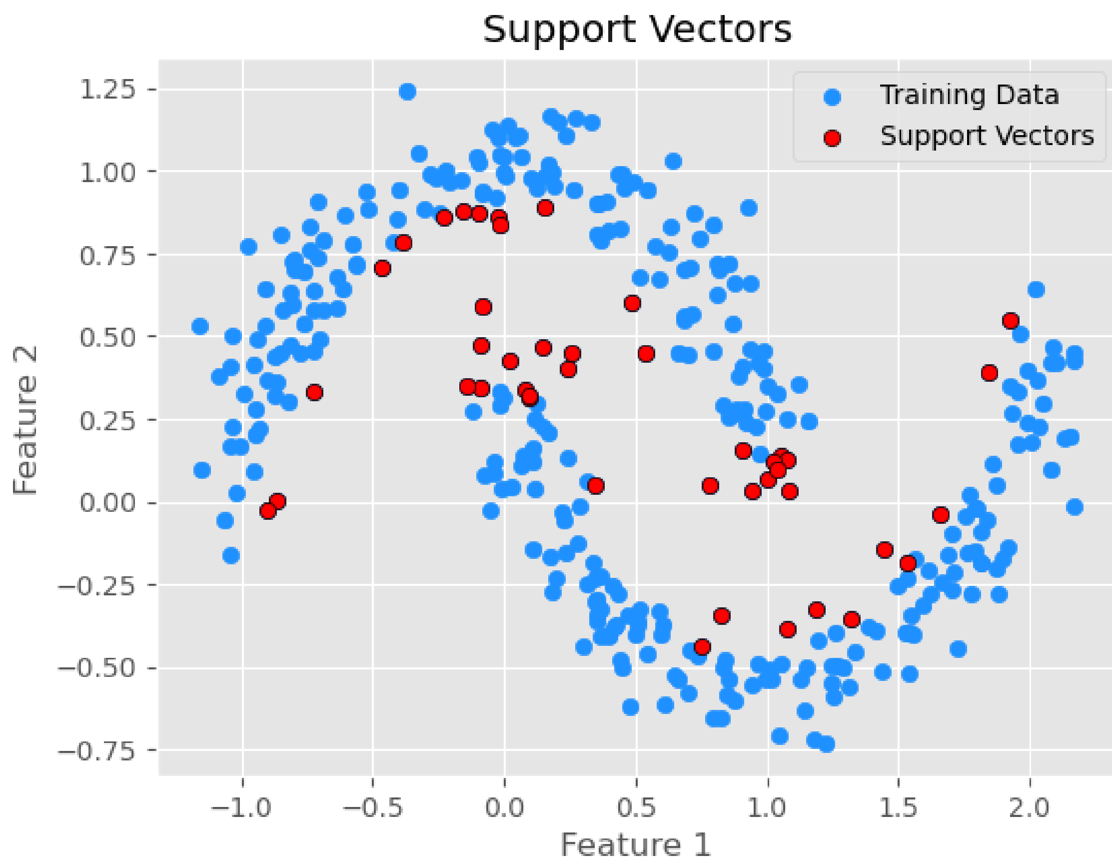

```
# Split dataset
x_train, x_test, y_train, y_test = tts(inputs, targets, test_size=0.3,
random_state=42)

# Train SVM classifier with RBF kernel
clf = svm.SVC(kernel='rbf', C=1, gamma='scale')
clf.fit(x_train, y_train)
```

```
SVC(C=1)

# Predictions and Confusion Matrix
y_pred = clf.predict(x_test)
cm = confusion_matrix(y_test, y_pred, normalize='true')
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=['Class 0', 'Class 1'])
disp.plot(cmap=plt.cm.Blues)
plt.grid(False)
plt.title('Confusion Matrix')
plt.show()
```

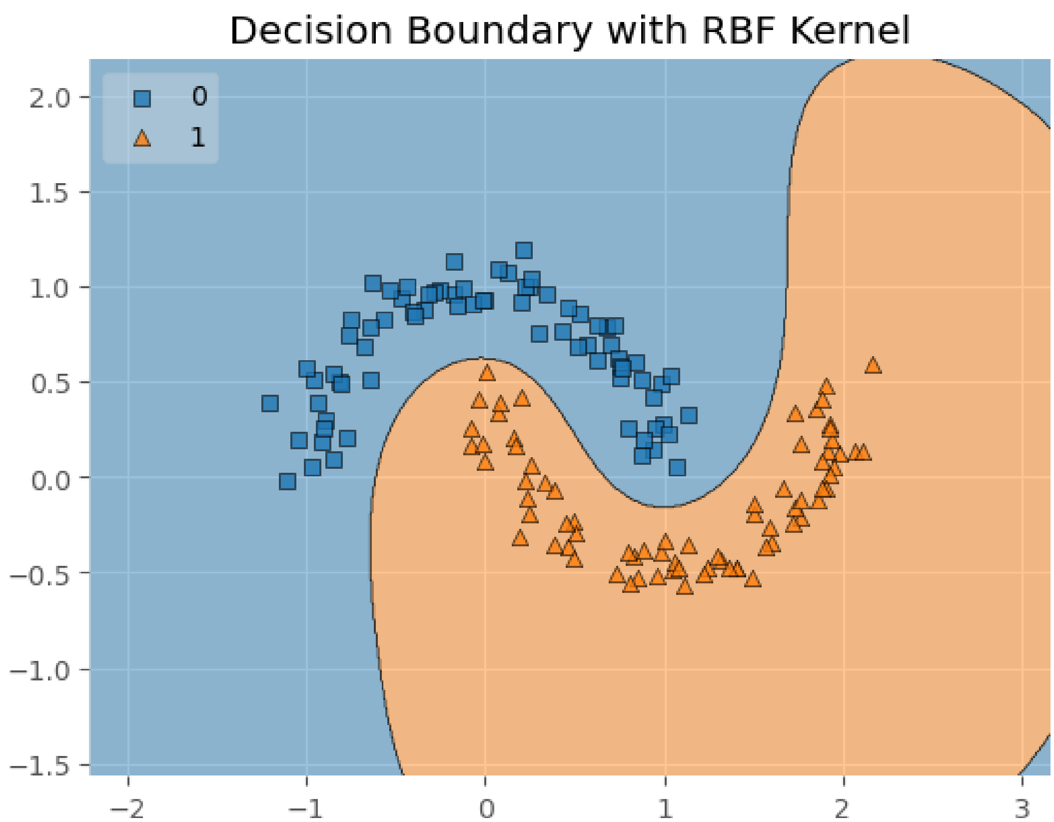



```
# Identify and plot support vectors
support_vectors = clf.support_vectors_
plt.scatter(x_train[:, 0], x_train[:, 1], color='dodgerblue',
label='Training Data')
plt.scatter(support_vectors[:, 0], support_vectors[:, 1], color='red',
label='Support Vectors', edgecolors='black')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
```

```
plt.title('Support Vectors')
plt.show()
```




```
# Visualizing the Decision Boundary
pdr(x_test, y_test, clf=clf, legend=2)
plt.title('Decision Boundary with RBF Kernel')
plt.show()
```



Decision Boundary with RBF Kernel
0
1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2
−1
0
1
2
3