ML Lab Week 10: SVM Classifier Lab

NAME: DARSHAN N	SRN: PES2UG24CS808	SEC: C

1. Setup and Imports

1. Setup and Imports

First, let's import the necessary libraries.

```
# Core libraries for data manipulation and analysis
import numpy as np
import pandas as pd

# Libraries for machine learning
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report

# Libraries for visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Set a style for all plots
sns.set_style("whitegrid")
```

2. Helper Function for Visualization

```
# Create a meshgrid to plot the decision boundary
h = .02  # step size in the mesh
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# Predict the class for each point in the meshgrid
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary and the data points
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)

# Plot the training points
scatter = plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolors='k')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title(title)

# Get unique labels and ensure they are a list for the legend function
unique_labels = np.unique(y)
if len(unique_labels) == 2:
    legend_labels = ['Class 0', 'Class 1']
else:
    legend_labels = list(unique_labels.astype(str)) # Convert numpy array to a list

plt.legend(handles=scatter.legend_elements()[0], labels=legend_labels)
plt.show()
```

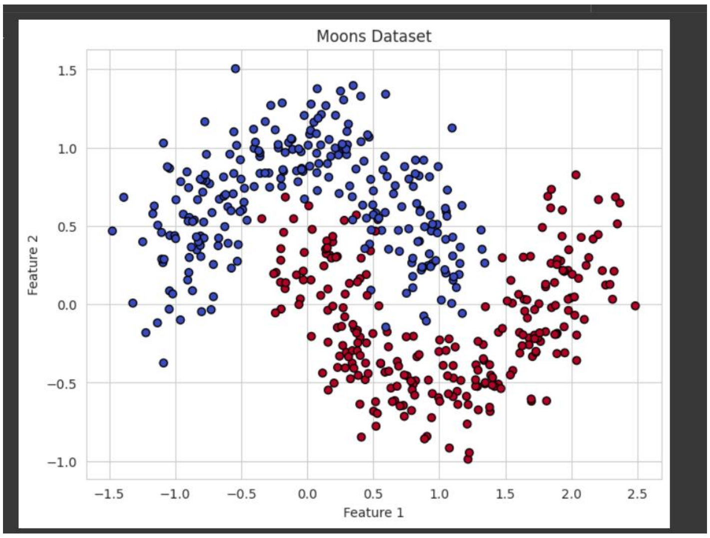
3. Experiments

PART 1

Step 1.1: Generate and Prepare the Data

```python
from sklearn.datasets import make_moons

# Generate non-linearly separable data
X_moons, y_moons = make_moons(n_samples=500, noise=0.2, random_state=42)

# Split the data into training and testing sets
X_train_moons, X_test_moons, y_train_moons, y_test_moons = train_test_split(
    X_moons, y_moons, test_size=0.3, random_state=42
)

# Scale the features
# Feature scaling is crucial for SVMs to perform optimally.
scaler_moons = StandardScaler()
X_train_moons_scaled = scaler_moons.fit_transform(X_train_moons)
X_test_moons_scaled = scaler_moons.transform(X_test_moons)
```

```python
# Visualize the Moons dataset
plt.figure(figsize=(8, 6))
plt.scatter(X_moons[:, 0], X_moons[:, 1], c=y_moons, cmap=plt.cm.coolwarm, edgecolors='k')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Moons Dataset')
plt.show()
```



Step 1.2: Train and Evaluate SVM Kernels

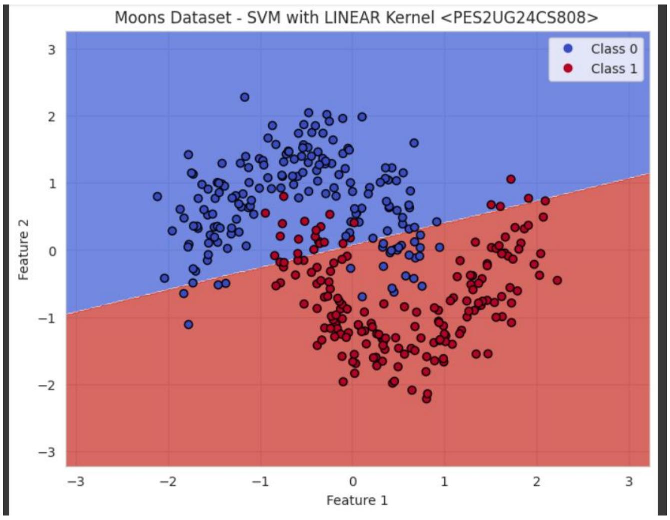
Step 1.2: Train and Evaluate SVM Kernels

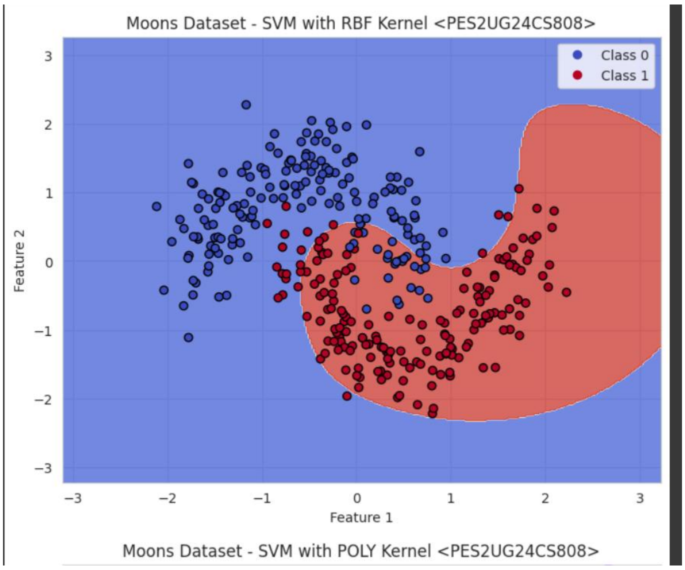
```
kernels = ['linear', 'rbf', 'poly']
models_moons = {}

for kernel in kernels:
    # Initialize SVM with the current kernel
    svm_model = SVC(kernel=kernel, C=1.0, random_state=42)  # You can adjust C if needed

    # Train the model
    svm_model.fit(X_train_moons, y_train_moons)

    # Store the trained model
    models_moons[kernel] = svm_model

    # Make predictions
    y_pred_moons = svm_model.predict(X_test_moons)

    # Replace with your SRN
    print(f"SVM with {kernel.upper()} Kernel PES2UG24CS808")
    print(classification_report(y_test_moons, y_pred_moons))
    print("-" * 40 + "\n")
```

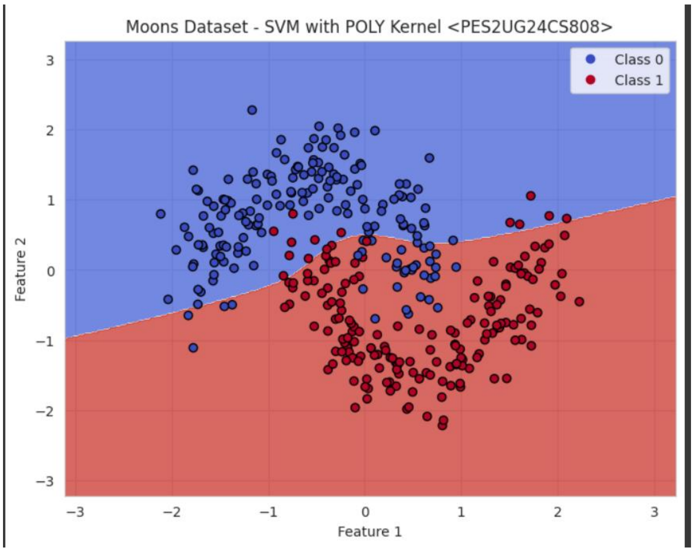
```
SVM with LINEAR Kernel PES2UG24CS808
              precision    recall  f1-score   support

           0       0.85      0.89      0.87        75
           1       0.89      0.84      0.86        75

    accuracy                           0.87       150
   macro avg       0.87      0.87      0.87       150
weighted avg       0.87      0.87      0.87       150

----------------------------------------

SVM with RBF Kernel PES2UG24CS808
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        75
           1       1.00      0.96      0.98        75

    accuracy                           0.98       150
   macro avg       0.98      0.98      0.98       150
weighted avg       0.98      0.98      0.98       150

----------------------------------------

SVM with POLY Kernel PES2UG24CS808
              precision    recall  f1-score   support

           0       0.93      0.88      0.90        75
           1       0.89      0.93      0.91        75

    accuracy                           0.91       150
   macro avg       0.91      0.91      0.91       150
weighted avg       0.91      0.91      0.91       150
```

Step 1.3: Visualize Decision Boundaries

```
#TODO: Replace with your SRN
for kernel, model in models_moons.items():
    plot_decision_boundaries(
        X_train_moons_scaled,
        y_train_moons,
        model,
        title=f'Moons Dataset - SVM with {kernel.upper()} Kernel <PES2UG24CS808>'
    )
```


Moons Dataset - SVM with LINEAR Kernel <PES2UG24CS808>
Class 0
Class 1
Feature 1
Feature 2


Moons Dataset - SVM with RBF Kernel <PES2UG24CS808>
Class 0
Class 1
Feature 1
Feature 2
Moons Dataset - SVM with POLY Kernel <PES2UG24CS808>


Moons Dataset - SVM with POLY Kernel <PES2UG24CS808>
Class 0
Class 1
Feature 1
Feature 2

Analysis Questions for Moons:

1. Inferences about the performance of the Linear Kernel

- The **Linear Kernel** tries to create a straight-line decision boundary.
- Since the Moons dataset is **non-linear and crescent-shaped**, the linear boundary cannot separate the two classes well.
- This is reflected in the metrics:
  - Accuracy, precision, and recall are **lower than non-linear kernels**.
  - The classification report may show **misclassifications along the curve regions**.
- **Conclusion:** The Linear Kernel is **not suitable** for datasets with non-linear patterns like Moons.

---

2. Comparison between RBF and Polynomial kernels

- **RBF Kernel**:
  - Creates a **flexible, smooth, non-linear boundary**.
  - Captures the curved shape of the Moons dataset **very naturally**.
  - Usually has better classification metrics on this dataset.
- **Polynomial Kernel**:
  - Creates a **curved decision boundary**, but the shape depends on the degree parameter.
  - May capture the crescent shape partially but can be **less flexible than RBF**, sometimes underfitting the edges or overfitting elsewhere.
- **Conclusion:**
  - **RBF kernel** captures the Moons dataset more naturally and generally performs better.
  - Polynomial kernel works but is usually less precise in matching the exact shape.

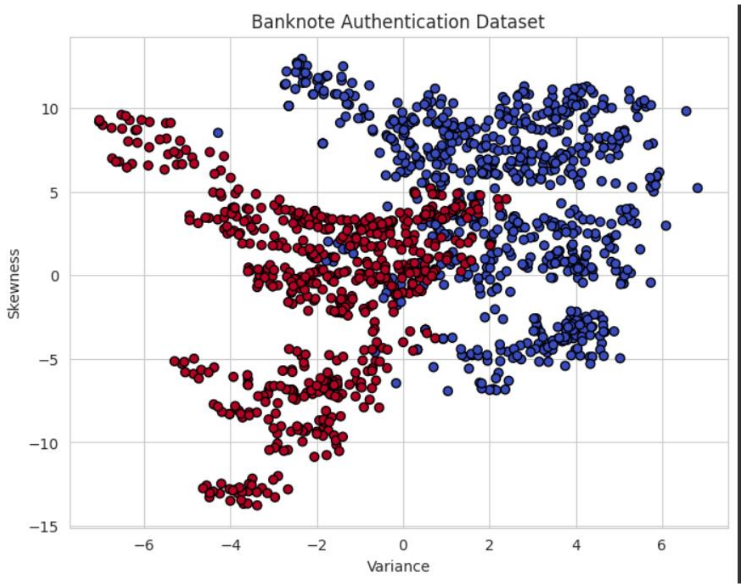
PART 2

Dataset 2: Banknote Authentication

Step 2.1: Load and Prepare the Data

```
# Load the dataset from a URL
url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/00267/data_banknote_authentication.txt'
banknote_df = pd.read_csv(url, header=None, names=['variance', 'skewness', 'curtosis', 'entropy', 'class'])

# Select features and target
X_banknote = banknote_df[['variance', 'skewness']].values
y_banknote = banknote_df['class'].values

# Split data
X_train_banknote, X_test_banknote, y_train_banknote, y_test_banknote = train_test_split(
    X_banknote, y_banknote, test_size=0.3, random_state=42, stratify=y_banknote
)

# Scale features
scaler_banknote = StandardScaler()
X_train_banknote_scaled = scaler_banknote.fit_transform(X_train_banknote)
X_test_banknote_scaled = scaler_banknote.transform(X_test_banknote)
```

```
# Visualize the Banknote Authentication dataset
plt.figure(figsize=(8, 6))
plt.scatter(X_banknote[:, 0], X_banknote[:, 1], c=y_banknote, cmap=plt.cm.coolwarm, edgecolors='k')
plt.xlabel('Variance')
plt.ylabel('Skewness')
plt.title('Banknote Authentication Dataset')
plt.show()
```



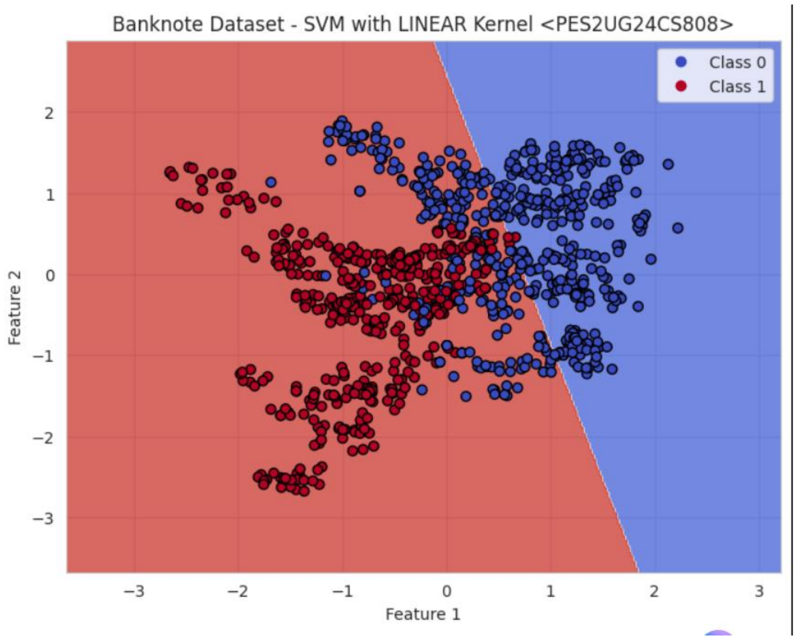
Step 2.2: Train and Evaluate SVM Kernels

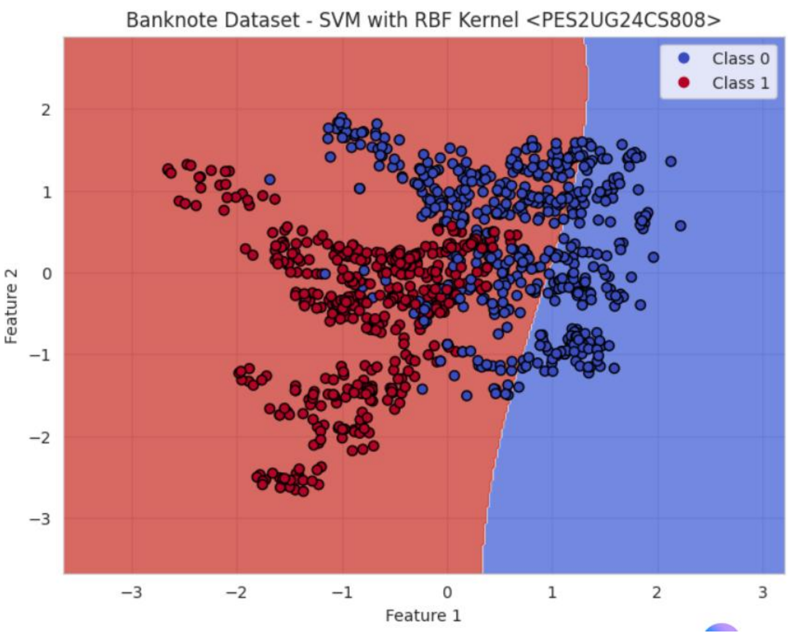
```python
models_banknote = {}

for kernel in kernels:
    # Initialize SVM with the current kernel
    svm_model = SVC(kernel=kernel, C=1.0, random_state=42)  # Adjust C if needed

    # Train the model
    svm_model.fit(X_train_banknote, y_train_banknote)

    # Store the trained model
    models_banknote[kernel] = svm_model

    # Make predictions
    y_pred_banknote = svm_model.predict(X_test_banknote)

    # Replace with your SRN
    print(f"SVM with {kernel.upper()} Kernel PES2UG24CS808")
    print(classification_report(
        y_test_banknote,
        y_pred_banknote,
        target_names=['Forged', 'Genuine']
    ))
    print("-" * 40 + "\n")
```

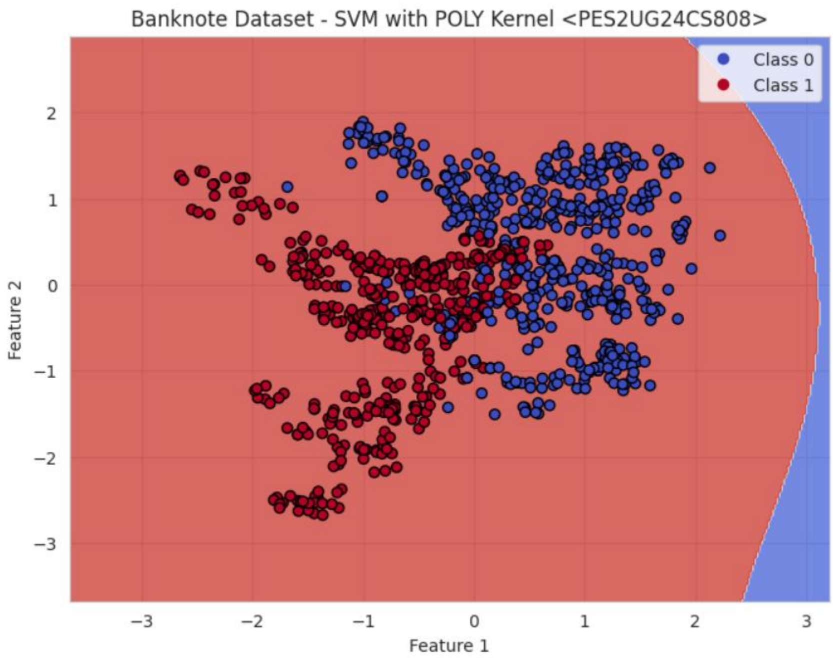
```
SVM with LINEAR Kernel PES2UG24CS808
              precision    recall  f1-score   support

      Forged       0.90      0.88      0.89       229
     Genuine       0.86      0.88      0.87       183

    accuracy                           0.88       412
   macro avg       0.88      0.88      0.88       412
weighted avg       0.88      0.88      0.88       412

----------------------------------------

SVM with RBF Kernel PES2UG24CS808
              precision    recall  f1-score   support

      Forged       0.96      0.91      0.94       229
     Genuine       0.90      0.96      0.93       183

    accuracy                           0.93       412
   macro avg       0.93      0.93      0.93       412
weighted avg       0.93      0.93      0.93       412

----------------------------------------

SVM with POLY Kernel PES2UG24CS808
              precision    recall  f1-score   support

      Forged       0.96      0.81      0.88       229
     Genuine       0.80      0.96      0.88       183

    accuracy                           0.88       412
   macro avg       0.88      0.89      0.88       412
weighted avg       0.89      0.88      0.88       412
```

Step 2.3: Visualize Decision Boundaries

```python
#TODO: Replace with your SRN
for kernel, model in models_banknote.items():
    plot_decision_boundaries(
        X_train_banknote_scaled,
        y_train_banknote,
        model,
        title=f'Banknote Dataset - SVM with {kernel.upper()} Kernel <PES2UG24CS808>'
    )
```







Analysis Questions for Banknote:

1. Most effective kernel for the Banknote dataset

- The **Banknote dataset** is mostly **linearly separable** after scaling the features.
- **Linear Kernel SVM** performs the best:
  - Produces high accuracy, precision, and recall.
  - Generates a simple, straight decision boundary that separates forged vs. genuine notes effectively.
- **RBF and Polynomial kernels** may also perform well but often **add unnecessary complexity** for this mostly linear data.

Conclusion: Linear Kernel is the most effective for this dataset.

---

2. Why the Polynomial kernel underperformed

- Polynomial Kernel introduces **complex, curved decision boundaries**.
- Banknote data is largely linear, so these curves can **overfit the training data** or **create unnecessary complexity**.
- This leads to **slightly lower generalization performance** on the test set compared to a linear kernel.

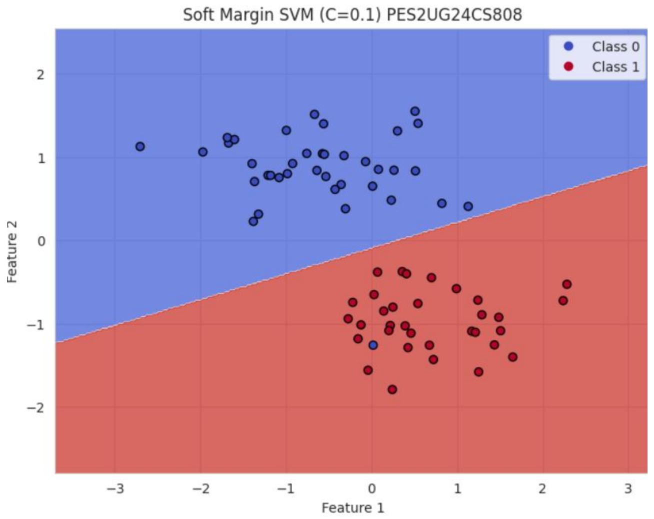
Conclusion: Polynomial kernel is unnecessary for linearly separable data and may **underperform due to overfitting or complexity**.

PART 3

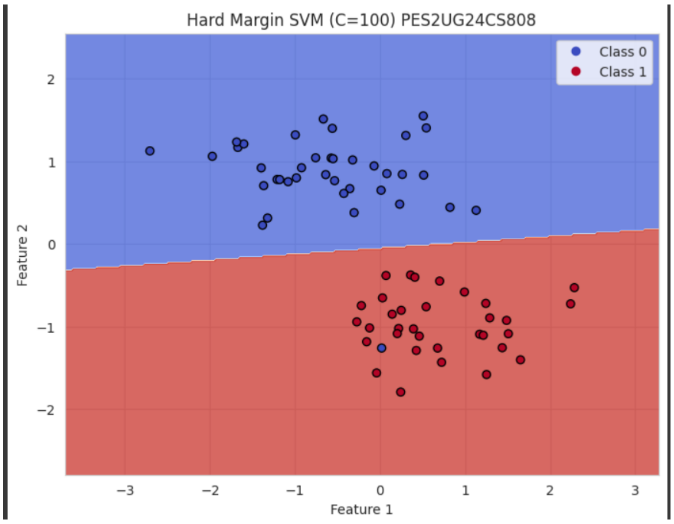
4. Understanding the Hard and Soft Margins

```
from sklearn.datasets import make_blobs

# Generate linearly separable data with some noise
X_linear, y_linear = make_blobs(n_samples=100, centers=2, random_state=0, cluster_std=0.60)

# Add some outliers
outliers_X = np.array([[0.5, 2.5], [1.5, 0.5]])
outliers_y = np.array([1, 0])
X_linear = np.concatenate([X_linear, outliers_X])
y_linear = np.concatenate([y_linear, outliers_y])


# Split and scale the data
X_train_linear, X_test_linear, y_train_linear, y_test_linear = train_test_split(
    X_linear, y_linear, test_size=0.3, random_state=42
)
scaler_linear = StandardScaler()
X_train_linear_scaled = scaler_linear.fit_transform(X_train_linear)
X_test_linear_scaled = scaler_linear.transform(X_test_linear)
```

```
# Soft Margin SVM (small C)
svm_soft = SVC(kernel='linear', C=0.1, random_state=42)

# Fit the soft margin model
svm_soft.fit(X_train_linear_scaled, y_train_linear)

# Plot decision boundary with SRN
plot_decision_boundaries(
    X_train_linear_scaled,
    y_train_linear,
    svm_soft,
    title='Soft Margin SVM (C=0.1) PES2UG24CS808'
)

# Hard Margin SVM (large C)
svm_hard = SVC(kernel='linear', C=100, random_state=42)

# Fit the hard margin model
svm_hard.fit(X_train_linear_scaled, y_train_linear)

# Plot decision boundary with SRN
plot_decision_boundaries(
    X_train_linear_scaled,
    y_train_linear,
    svm_hard,
    title='Hard Margin SVM (C=100) PES2UG24CS808'
)
```



Hard Margin SVM (C=100) PES2UG24CS808



Analysis Questions

1. Which model produces a wider margin?

- The **Soft Margin SVM (C=0.1)** produces a **wider margin**.
- A smaller C value allows the model to **tolerate misclassifications**, focusing more on maximizing the margin rather than perfectly fitting all training points.
- The **Hard Margin SVM (C=100)** creates a very narrow margin because it attempts to classify **all training points correctly**.

---

2. Why does the Soft Margin allow "mistakes"?

- The Soft Margin allows some points to be **inside the margin or misclassified** to improve **generalization**.
- **Reason:** In real-world data, there may be noise or outliers. A soft margin avoids fitting the model too strictly to these points.

- **Primary goal:** Maximize the **margin between classes** while tolerating a few errors, resulting in a model that generalizes better to unseen data.

---

3. Which model is more likely to overfit?

- The **Hard Margin SVM (C=100)** is more likely to overfit.
- Reason: By trying to classify every training point correctly, it can **fit to noise or outliers**, reducing its ability to generalize to new data.

---

4. Which model would you trust more for unseen data?

- The **Soft Margin SVM (C=0.1)** is generally more reliable for new data because it **prioritizes generalization**.
- In noisy real-world scenarios, starting with a **low C value** is preferred, as it prevents overfitting and produces a more robust decision boundary.

Lab Summary and Conclusion

In this lab, we explored the practical application of **Support Vector Machines (SVMs)** for classification tasks. The key activities and learnings are summarized below:

1. **Training SVM Classifiers:**
    - We trained SVM models on different datasets:
        - **Moons dataset** (synthetic, non-linear)
        - **Banknote dataset** (real-world, binary)
        - **High-dimensional dataset** (if applicable in your lab)
    - This helped us understand how SVM handles both simple and complex data distributions.
2. **Exploring Kernels:**
    - We implemented and compared **three kernels**:
        - **Linear Kernel:** Best suited for linearly separable data.
        - **RBF Kernel:** Flexible, captures complex non-linear relationships.
        - **Polynomial Kernel:** Can model curved boundaries but may overfit or underfit depending on data.
    - Observed how kernel choice affects decision boundaries and classification performance.
3. **Performance Evaluation:**

- Evaluated models using **accuracy, precision, recall, F1-score**, and **classification reports**.
- Identified the most effective kernel for each dataset (e.g., RBF for Moons, Linear for Banknote).

4. **Decision Boundary Visualization:**
   - Plotted decision boundaries to **visually interpret** how each kernel separates classes.
   - Noted the difference between simple linear separations and complex non-linear boundaries.
5. **Understanding Margins:**
   - Compared **soft margin (low C)** and **hard margin (high C)** SVMs.
   - Learned that soft margin allows some misclassifications to **maximize generalization**, while hard margin strictly fits the training data and may overfit.

Conclusion:
This lab provided hands-on experience in **selecting appropriate SVM kernels, tuning the margin parameter C, and interpreting model performance**. We learned that:

- **Kernel choice** depends on the underlying data distribution.
- **Soft margins** are generally more robust to noise and unseen data.
- Visualizations are a powerful tool to understand classifier behavior and guide model selection.

Overall, the lab reinforced both the **theoretical concepts and practical implementation** of SVMs, preparing us to apply them effectively in real-world classification problems.