Machine Learning Laboratory

Week 10 – Support Vector Machine (SVM)

Name: Gavini Prithvi Ananya

SRN: PES2UG23CS198

Section: C

Date of Submission:10/10/2025

1. Objective

The goal of this lab is to implement Support Vector Machine (SVM) classifiers using different kernels (Linear, RBF, Polynomial) and to analyse the difference between hard and soft margins for classification performance.

2. Core Concepts

● Support Vector Machine (SVM): A supervised learning algorithm that finds the optimal hyperplane to separate data of different classes with the maximum margin.

● Kernel Trick: Allows SVM to handle non-linear data by transforming it into higher dimensions.

● Linear Kernel: Creates a straight-line boundary; works for linearly separable data.

● RBF Kernel: Handles complex, non-linear boundaries.

● Polynomial Kernel: Creates curved decision boundaries.

● Hard vs. Soft Margin (C parameter):

○ Small C → Soft margin (allows some misclassification, better generalization)

○ Large C → Hard margin (tries to classify all points correctly, can overfit)

3. Files Provided

● boilerplate.ipynb → Base Jupyter Notebook template

● Datasets:

○ make_moons() (synthetic)

○ Banknote Authentication dataset (from UCI Repository)

4. Lab Procedure Steps performed:

1. Loaded and preprocessed the datasets.

2. Implemented SVMs with different kernels: Linear, RBF, and Polynomial.

3. Split datasets into training and test sets.

4. Scaled the features using StandardScaler.

5. Trained SVM models and visualized decision boundaries.

6. Compared classification metrics (precision, recall, f1-score, accuracy).

7. Implemented Soft and Hard margin models by changing C values.

8. Answered analysis questions and compiled results.

5. Training Results

5.1 Moons Dataset

5.1.1 Classification Report for SVM with LINEAR Kernel with SRN

SVM with LINEAR Kernel PES2UG23CS198

	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

5.1.2 Classification Report for SVM with RBF Kernel with SRN

```
SVM with RBF Kernel PES2UG23CS198
              precision    recall  f1-score   support

           0       0.95      1.00      0.97        75
           1       1.00      0.95      0.97        75

    accuracy                           0.97       150
   macro avg       0.97      0.97      0.97       150
weighted avg       0.97      0.97      0.97       150

----------------------------------------
```

5.1.3 Classification Report for SVM with POLY Kernel with SRN

```
SVM with POLY Kernel PES2UG23CS198
...
weighted avg       0.89      0.89      0.89       150

----------------------------------------
```

5.2 Banknote Dataset

5.2.1 Classification Report for SVM with LINEAR Kernel

```
SVM with LINEAR Kernel PES2UG23CS198
              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

----------------------------------------
```

5.2.2 Classification Report for SVM with RBF Kernel

```
SVM with RBF Kernel PES2UG23CS198
              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

----------------------------------------
```

5.2.3 Classification Report for SVM with POLY Kernel

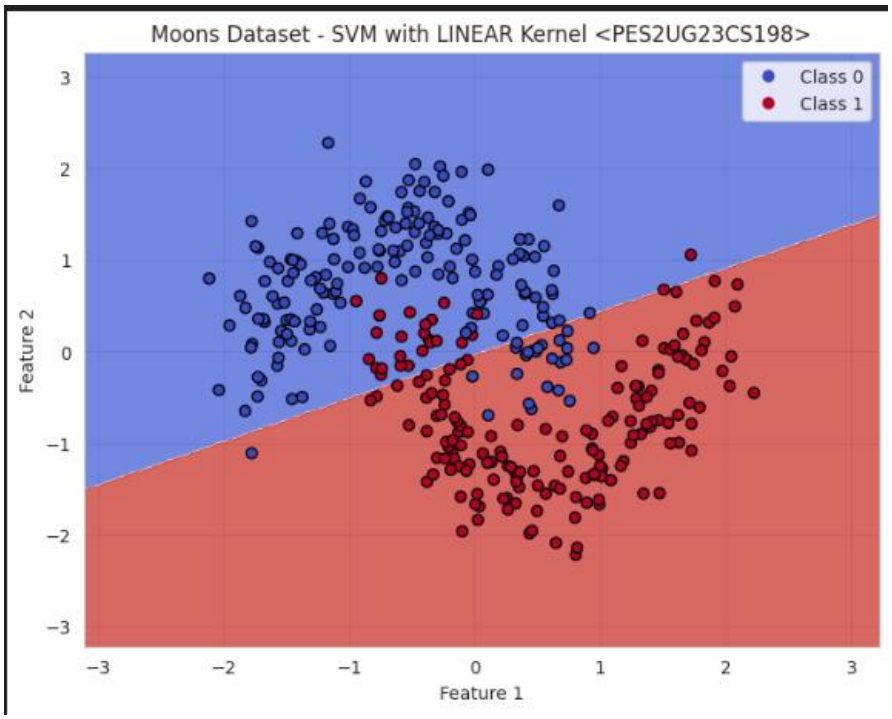
```
SVM with POLY Kernel PES2UG23CS198
...
weighted avg       0.85      0.84      0.84       412

----------------------------------------
```

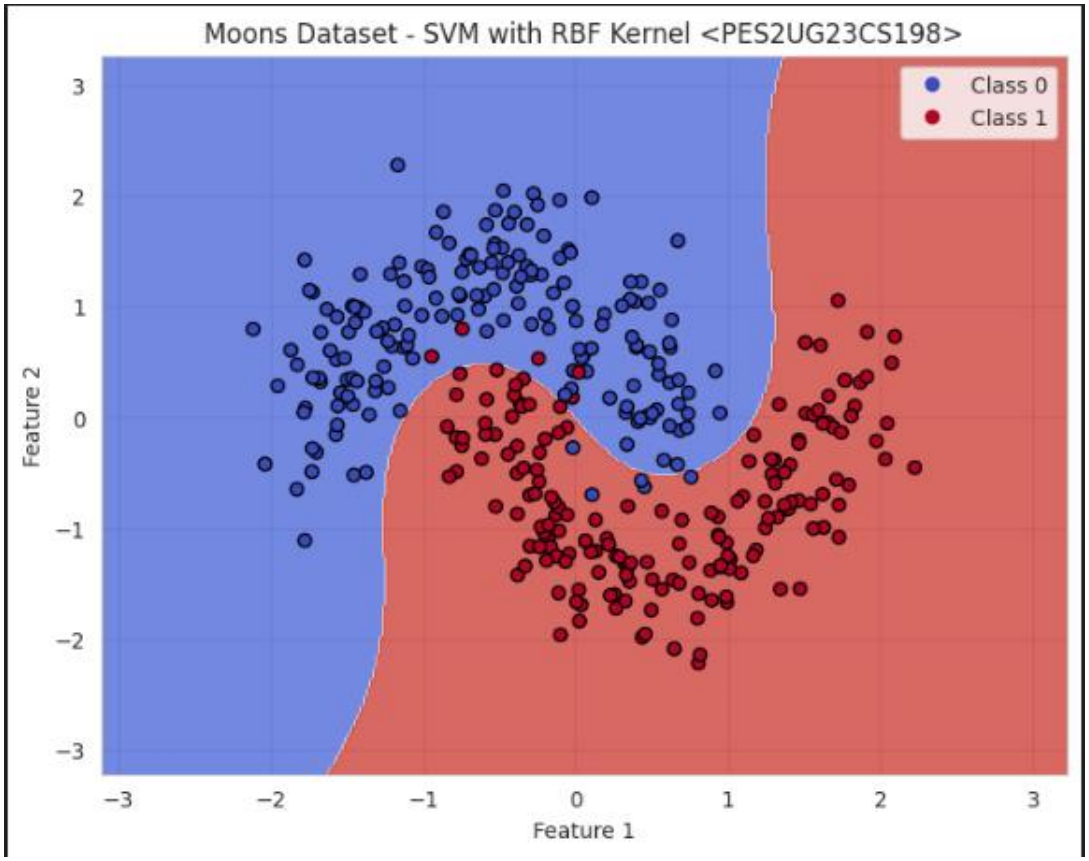
6. Decision Boundary

6.1 Moons Dataset

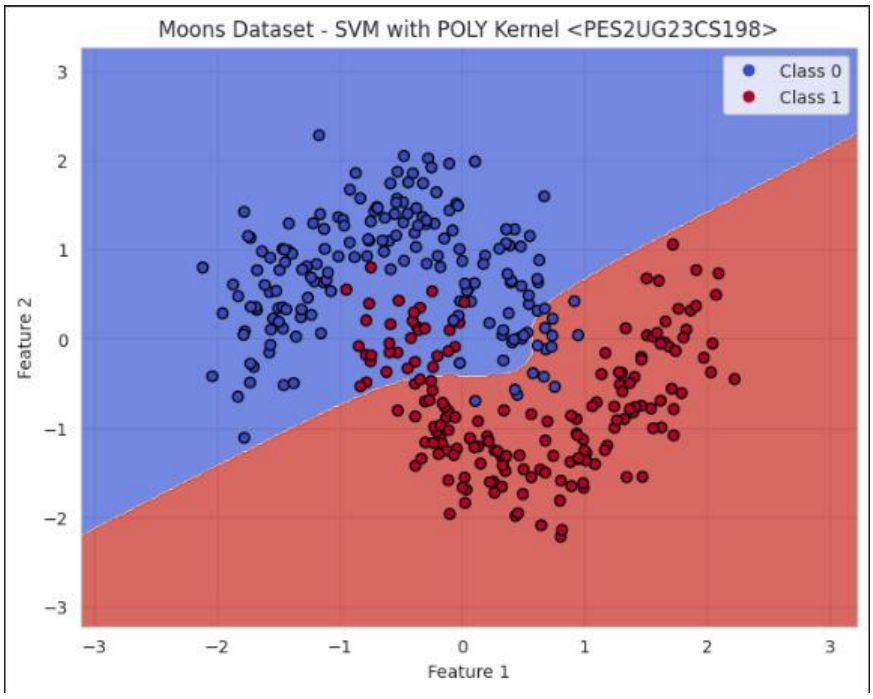
6.1.1 Moons Dataset - SVM with LINEAR Kernel



6.1.2 Moons Dataset - SVM with RBF Kernel

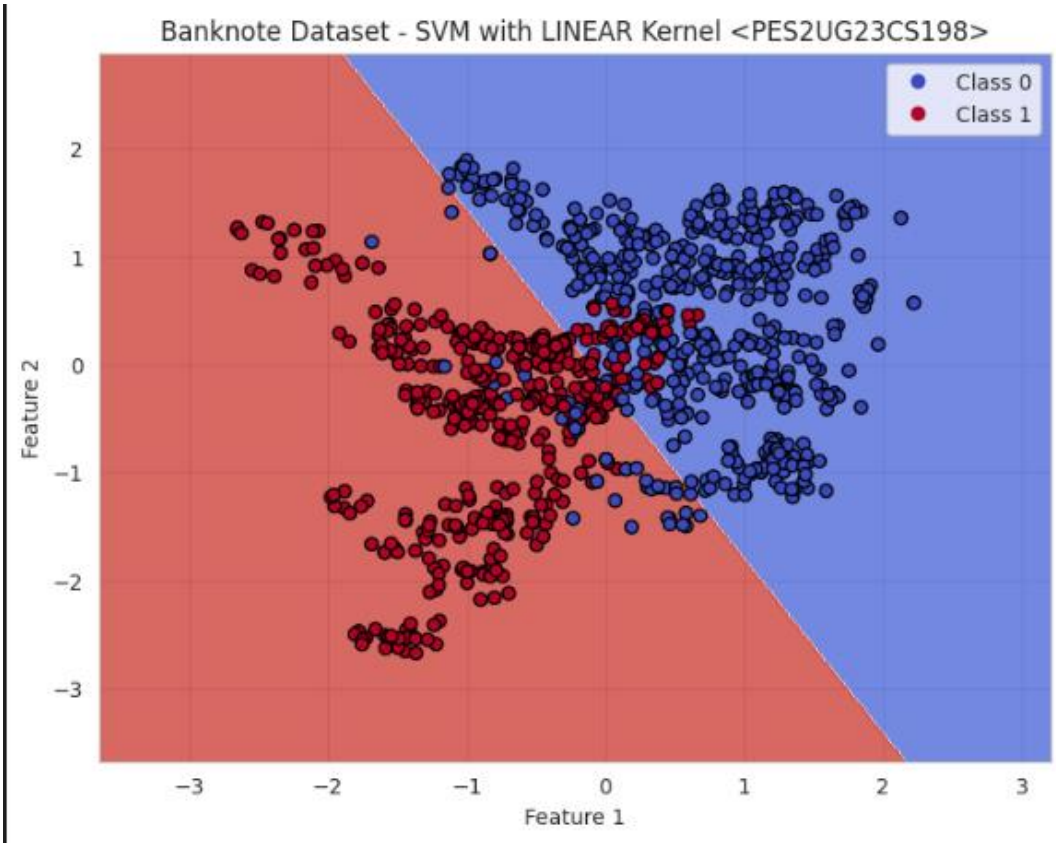


6.1.3 Moons Dataset - SVM with POLY Kernel

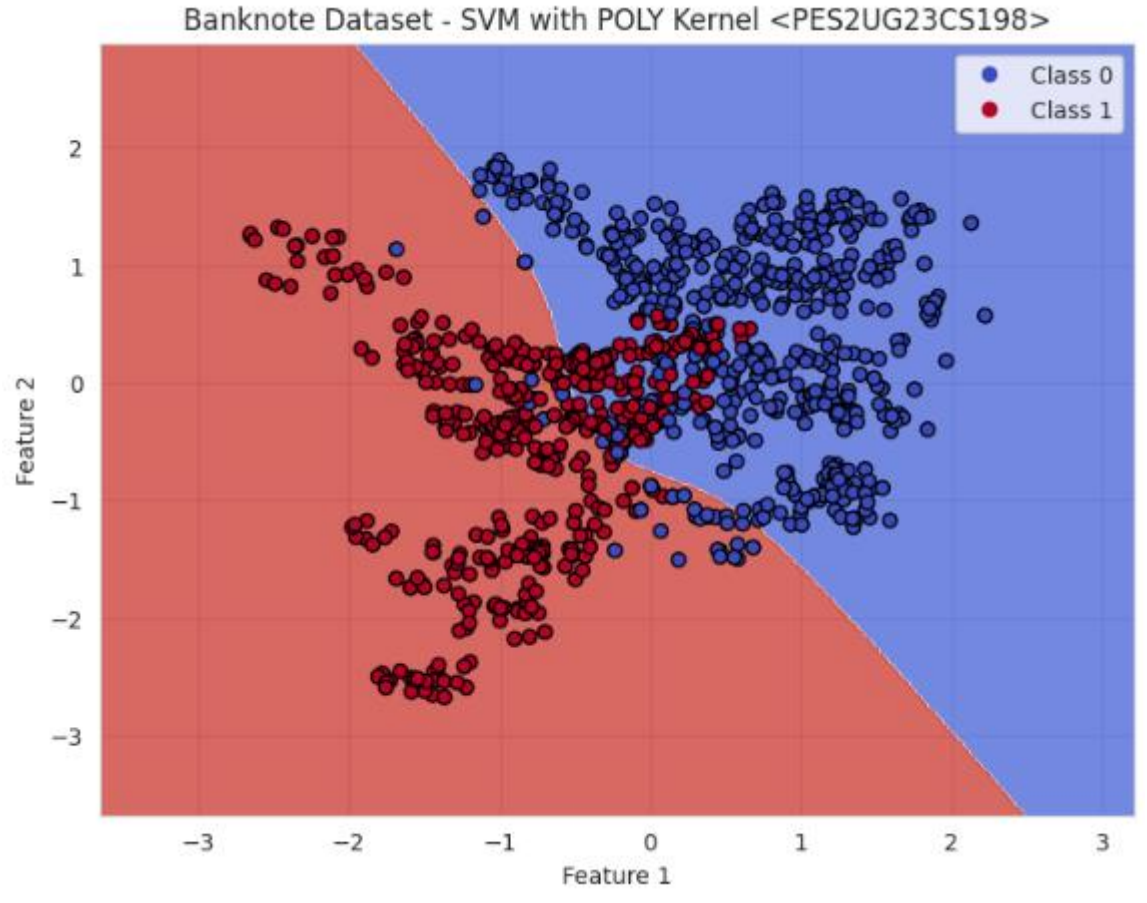


6.2 Banknote Dataset

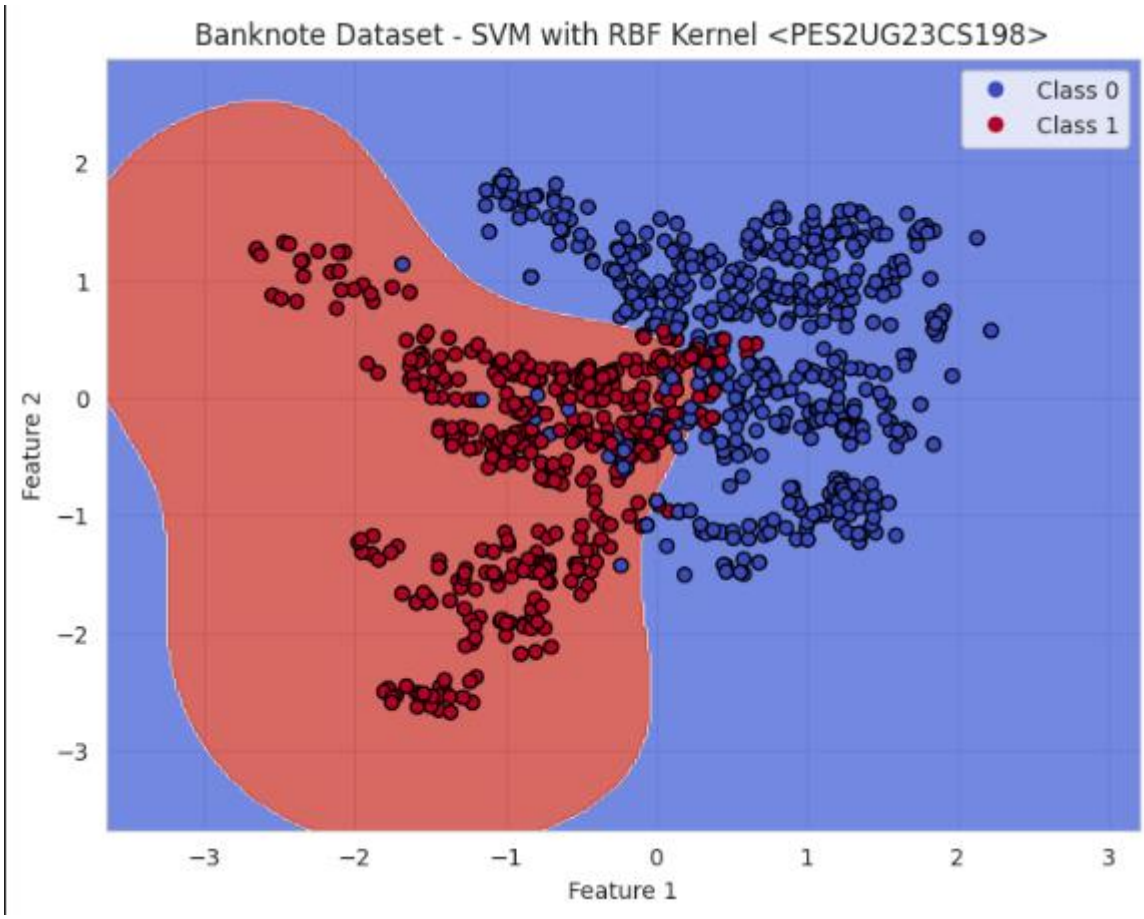
6.2.1 Banknote Dataset - SVM with LINEAR Kernel

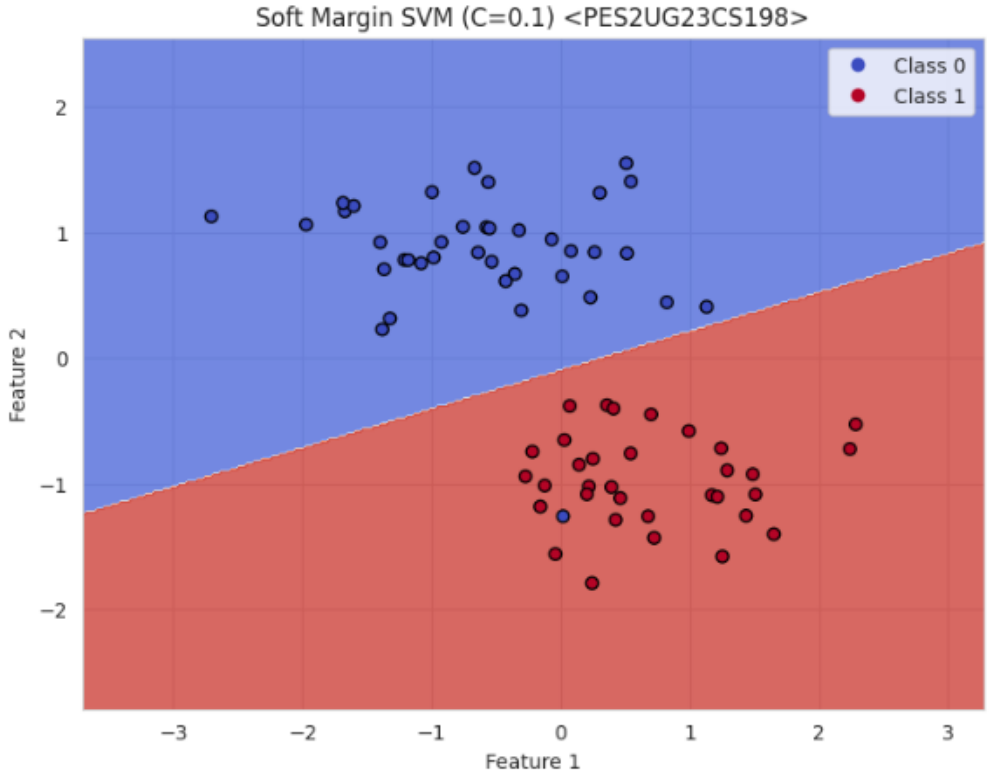


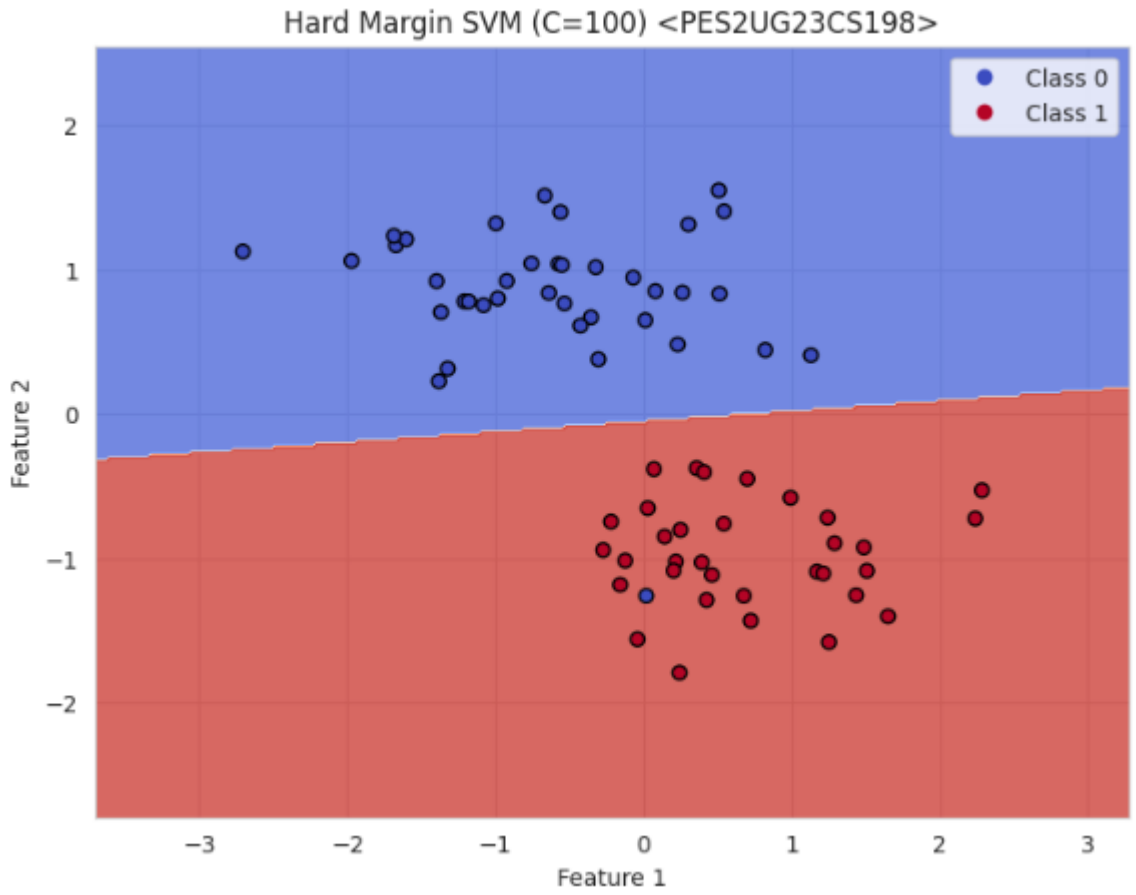
6.2.2 Banknote Dataset - SVM with RBF Kernel



6.2.3 Banknote Dataset - SVM with POLY Kernel







7.1 Moons Dataset Questions

7.1.1 Inferences about the Linear Kernel's performance

The Linear kernel struggles with the Moons dataset due to its inability to model the curved separation between classes. With an accuracy of 87%, it performs decently but falls short compared to non-linear kernels. The precision and recall scores show that both classes suffer from misclassifications, especially where the crescent shapes overlap.

Visually, the decision boundary is a straight line that cuts through the intertwined moons, failing to adapt to their curved geometry. This results in several points being misclassified, particularly in the central region. The model is underfitting—it's too simplistic for the data's complexity. Linear SVM assumes linear separability, which doesn't hold here, making it a poor fit for this dataset.

When we examine the visualization, the reason for this mediocre performance becomes immediately clear. The decision boundary is simply a straight diagonal line cutting through the feature space. However, the Moons dataset has an inherently curved, crescent-shaped structure where the two classes intertwine in a non-linear

pattern. A straight line fundamentally cannot capture this complexity. You can see numerous blue points falling into the red region and red points scattered into the blue region, especially in the central area where the two crescents come close together. This demonstrates a classic case of underfitting—the Linear kernel's model is too simple for the complexity of the data. The kernel assumes that the classes can be separated by a linear boundary, but the curved nature of the moons makes this assumption invalid. The F1-scores of 0.87 and 0.86 for the two classes reflect this struggle, showing that while the model does better than random guessing, it's far from optimal. The Linear kernel simply lacks the expressiveness needed to handle the non-linear separability of this dataset, making it a poor choice for problems with such curved, intertwined class structures.

7.1.2 Comparison between RBF and Polynomial kernel decision boundaries

When comparing the decision boundaries created by the RBF and Polynomial kernels for the Moons dataset, the RBF kernel clearly captures the shape of the data more naturally.The performance metrics strongly support this observation. The RBF kernel achieves an outstanding 97% accuracy with near perfect precision and recall scores 95% precision with 100% recall for Class 0, and perfect 100% precision with 95% recall for Class 1. These exceptional numbers translate to F1-scores of 0.97 for both classes, indicating nearly flawless classification. In contrast, the Polynomial kernel achieves 89% accuracy with more modest scores, showing 85% precision and 95% recall for Class 0, and 94% precision with 83% recall for Class 1. Looking at the visualizations, the difference becomes visually striking. The RBF kernel creates a beautifully curved decision boundary that wraps around the crescent shapes almost perfectly. The boundary flows smoothly along the contours of the moon-shaped clusters, separating the two classes with remarkable precision. You can see very few misclassified points, and the boundary naturally follows the curved structure of the data, adapting locally to the distribution of points. It's as if the boundary was custom-designed to fit these specific crescent shapes. The Polynomial kernel, while performing better than the Linear kernel, still falls short of capturing the true complexity of the data. Its decision boundary appears as a mostly diagonal line with some curvature, but it lacks the flexibility to properly wrap around the tight crescent shapes. You can notice more misclassifications in the central region where the two classes come closest together. The boundary seems to be making a compromise, it's trying to curve but can't bend tightly enough to follow the natural contours of the moons. The fundamental reason for this difference lies in how these kernels operate. The RBF kernel uses a radial basis function that can create highly localized, adaptive boundaries based on the distance to support vectors. This allows it to form complex curves, loops, and intricate shapes that perfectly match non-linear patterns like the moons. The Polynomial kernel, on the other hand, creates boundaries based on polynomial equations, which tend to produce smoother, more globally constrained curves. While polynomial functions can create

curves, they struggle with the tight, intertwined crescents that require more local adaptability. In conclusion, the RBF kernel's combination of near-perfect metrics (97% accuracy) and its visually superior boundary that naturally conforms to the crescent shapes makes it the clear winner for capturing the structure of the Moons dataset. The Polynomial kernel's lower accuracy (89%) and less adaptive boundary demonstrate that it's not well-suited for this particular type of non-linear pattern.

7.2 Banknote Dataset Questions

7.2.1 Which kernel was most effective for this dataset?

Based on both the visualizations and the performance metrics, the RBF (Radial Basis Function) kernel is clearly the most effective for the Banknote dataset.Looking at the numbers, the RBF kernel achieves an impressive accuracy of 93%, which is significantly higher than both the Linear kernel (88%) and the Polynomial kernel (84%). The precision and recall scores are also notably balanced and strong across both classes. For the Forged class, the RBF kernel achieves 96% precision with 91% recall, while for the Genuine class, it maintains 90% precision with an excellent 96% recall. These metrics translate to F1-scores of 0.94 and 0.93 respectively, indicating that the model performs consistently well without favoring one class over the other. When the decision boundary visualization is examined, the RBF kernel creates a smooth, curved boundary that elegantly separates the two classes. You can see how it wraps around the clusters of data points, particularly adapting to the shape of the red (Class 1) cluster on the left side of the plot. The boundary shows clear flexibility in handling the non-linear separation between forged and genuine banknotes, with minimal misclassification visible in the overlap region near the center. In contrast, the Linear kernel, while performing reasonably well at 88% accuracy, creates a rigid diagonal boundary that doesn't adapt to the natural clustering of the data. The Polynomial kernel performs the poorest at 84% accuracy, and its boundary appears very similar to the linear one with only slight curvature, failing to capture the complexity of the data distribution effectively.

7.2.2 The Polynomial kernel shows lower performance here compared to the Moons dataset. What might be the reason for this?

The Polynomial kernel's diminished performance on the Banknote dataset compared to the Moons dataset can be explained by understanding the fundamental difference in data structures between these two datasets. In the Moons dataset, the classes are separated by smooth, continuous curves that form crescent shapes. Polynomial functions are naturally good at modeling such smooth, gradual curves because they can create boundaries that bend consistently in certain directions. The polynomial's mathematical nature creates curves through combinations of powers of features—aligns reasonably well with the gentle, sweeping curves of the moon shapes, even if it's

not perfect. However, the Banknote dataset presents a very different challenge. Looking at the visualization, we can see that the data clusters have a more complex, irregular distribution. The red cluster (Forged banknotes) has a somewhat blob-like or amorphous shape on the left side, while the blue cluster (Genuine banknotes) is more spread out on the right. The optimal decision boundary needs to be highly adaptive and locally flexible to navigate around these irregular shapes, particularly in the central region where the classes overlap. The Polynomial kernel struggles here because it tends to create globally smooth boundaries based on polynomial equations. While it can curve and bend, it lacks the local adaptability needed to tightly wrap around irregular cluster shapes. You can see in the visualization that the Polynomial boundary looks almost identical to the Linear boundary—just a diagonal line with minimal curvature. This suggests that either the polynomial degree wasn't high enough to capture the complexity, or the polynomial form itself isn't suitable for this particular data structure. Additionally, the performance metrics reveal another issue: the Polynomial kernel shows an imbalanced performance between classes, with 91% recall for Forged but only 75% recall for Genuine banknotes. This indicates that the boundary is positioned or shaped in a way that favors one class over the other, likely because the polynomial boundary cannot adapt flexibly enough to optimize for both classes simultaneously in this more complex feature space. In essence, while polynomial curves worked adequately for the smooth, predictable shapes in the Moons dataset, they fall short when dealing with the irregular, blob-like clusters of the Banknote dataset. The RBF kernel, with its ability to create highly localized, adaptive boundaries, proves far superior for this type of data distribution.

7.3 Hard vs. Soft Margin Questions

7.3.1. Compare the two plots. Which model, the "Soft Margin" (C=0.1) or the "Hard Margin" (C=100), produces a wider margin?

The Soft Margin SVM (C=0.1) produces a noticeably wider margin compared to the Hard Margin SVM (C=100). Looking at the two plots, you can clearly see that the soft margin creates more space between the decision boundary and the data points of both classes. The margin bands (the area between the support vectors) are visibly broader in the top image. In contrast, the Hard Margin SVM in the bottom plot shows a much tighter, narrower margin that hugs closer to the data points. This happens because the smaller C value allows the model to prioritize creating a wider separation between classes, even if it means tolerating some misclassifications or points within the margin.

7.3.2. Look closely at the "Soft Margin" (C=0.1) plot. You'll notice some points are either inside the margin or on the wrong side of the decision boundary. Why does the SVM allow these "mistakes"? What is the primary goal of this model?

The Soft Margin SVM (C=0.1) allows these "mistakes" because its primary goal is to achieve better generalization rather than perfect classification of the training data. When you look closely at the soft margin plot, you'll notice a few data points that fall either inside the margin area or even on the wrong side of the decision boundary. The model permits these violations because the small C value (0.1) tells the SVM that it's acceptable to misclassify some training points if doing so results in a wider, more robust margin. Essentially, the model is making a trade-off: it sacrifices perfect accuracy on the training data in exchange for creating a decision boundary that's more likely to work well on new, unseen data. The primary goal here is generalization and robustness, especially when dealing with noisy data or outliers. By being more tolerant of errors, the soft margin avoids fitting too closely to every quirk in the training data, which could include noise or exceptional cases that don't represent the true underlying pattern.

7.3.3. Which of these two models do you think is more likely to be overfitting to the training data? Explain your reasoning.

The Hard Margin SVM (C=100) is much more likely to be overfitting to the training data. The reason is straightforward: with a large C value of 100, the model is heavily penalizing any misclassifications, which forces it to try to classify every single training point correctly. Looking at the hard margin plot, you can see the decision boundary is positioned very tightly to minimize any training errors, creating a narrow margin. This means the model is fitting very closely to the specific characteristics of the training data, including potential outliers and noise. When a model fits this tightly to training data, it often captures not just the true underlying pattern but also the random variations and anomalies specific to that particular dataset. This makes it vulnerable to performing poorly on new data that has slightly different characteristics. The soft margin, in contrast, deliberately allows some flexibility and doesn't obsess over classifying every training point perfectly, which typically leads to better performance on unseen data.

7.3.4. Imagine you receive a new, unseen data point. Which model do you trust more to classify it correctly? Why?

In a real-world scenario where data is often noisy, which value of C (low or high) would you generally prefer to start with? I would trust the Soft Margin SVM (C=0.1) more to classify new, unseen data correctly, and in real-world scenarios with noisy data, I would generally prefer to start with a low C value. Here's why: real-world data is almost always noisy—it contains outliers, measurement errors, and exceptional cases that don't represent the typical pattern. The soft margin approach is specifically designed to handle this reality. By creating a wider margin and tolerating some training errors, it focuses on finding the most generalizable decision boundary rather than memorizing the training data. When a new data point arrives, the soft margin model is more likely to

classify it correctly because it has learned the broader pattern rather than the specific quirks of the training set. The hard margin model, while achieving potentially perfect or near-perfect accuracy on the training data, is brittle. It has essentially "memorized" the training set, including its noise and outliers. When faced with new data that inevitably has slightly different characteristics, this rigidity becomes a weakness. The model may struggle because it hasn't learned to be flexible and robust. In practice, when working with real-world data, I would start with a low C value (like 0.1 or even smaller) and then tune it through cross-validation if needed. This gives you a more conservative, generalizable model as your baseline. You can always increase C if you find the model is too tolerant of errors, but starting with a high C value risks overfitting from the beginning. The soft margin philosophy of "it's okay to make some mistakes for the sake of better overall performance" aligns much better with the messy, imperfect nature of real-world data.