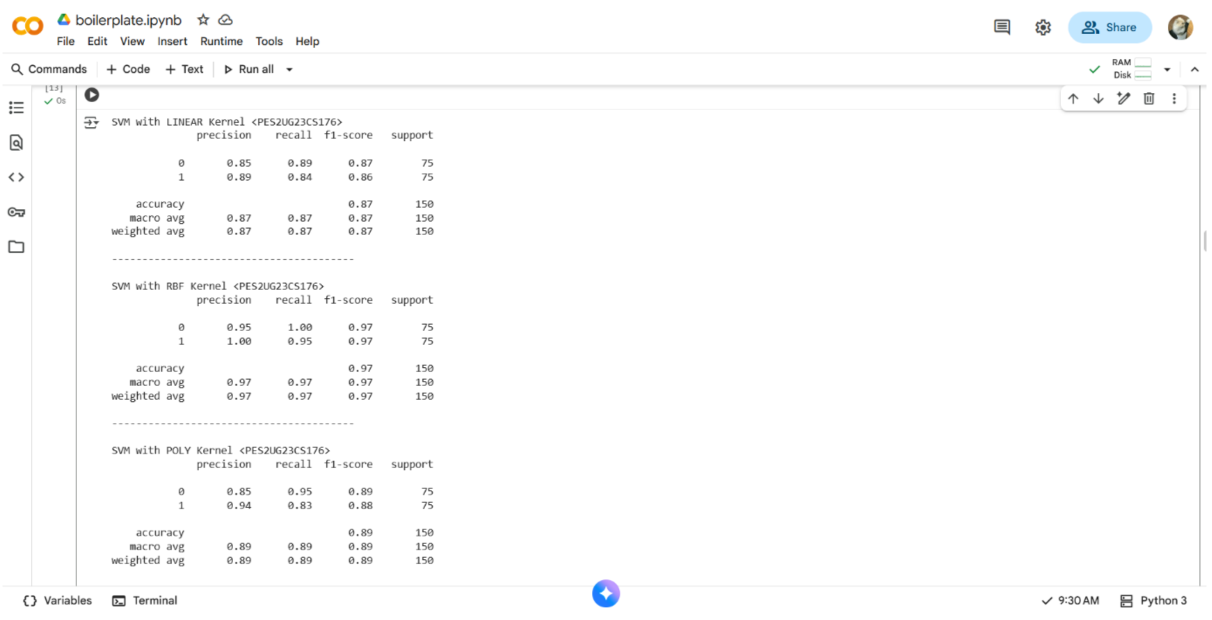
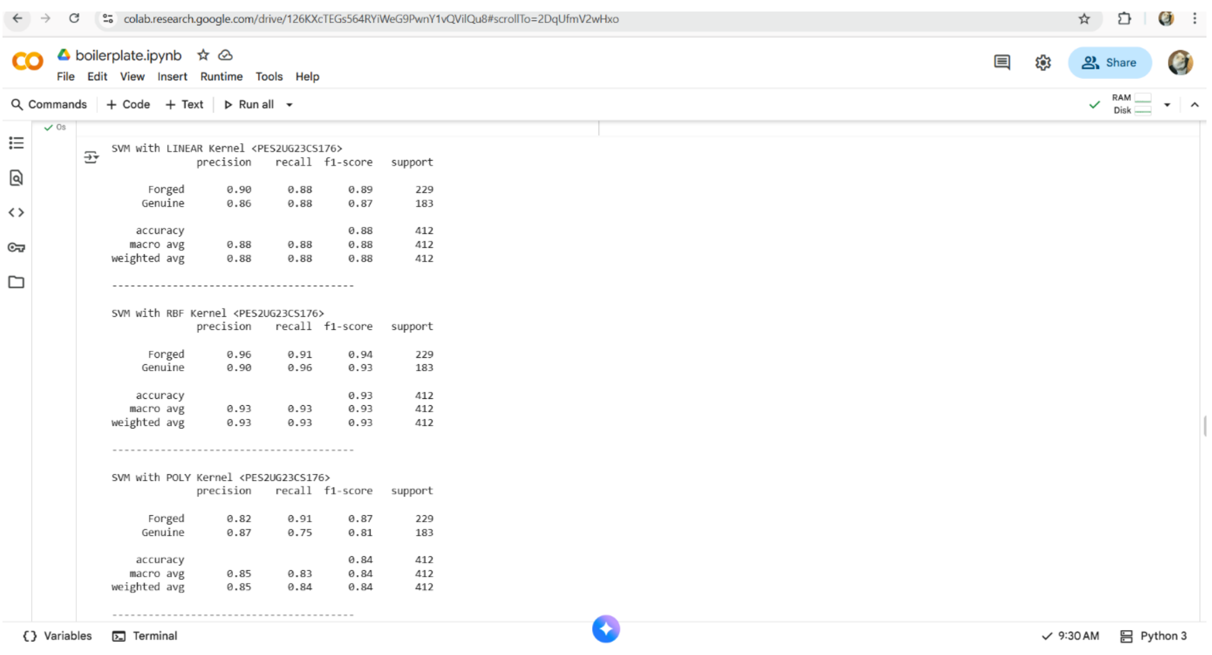
MACHINE LEARNING LAB

SCREENSHOTS:-

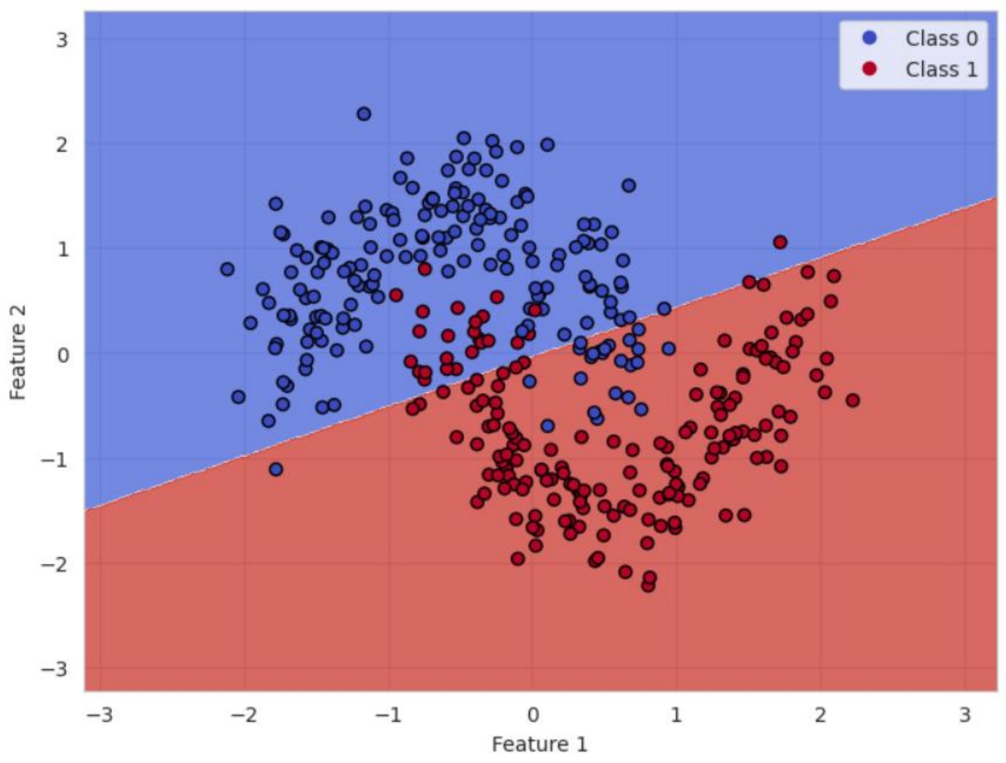

boilerplate.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM Disk
Variables Terminal
9:30 AM Python 3

```
SVM with LINEAR Kernel <PES2UG23CS176>
              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 <PES2UG23CS176>
              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

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

SVM with POLY Kernel <PES2UG23CS176>
              precision    recall  f1-score   support

           0       0.85      0.95      0.89        75
           1       0.94      0.83      0.88        75

    accuracy                           0.89       150
   macro avg       0.89      0.89      0.89       150
weighted avg       0.89      0.89      0.89       150
```


colab.research.google.com/drive/126KXcTEGs564RYiWeG9PwnY1vQVilQu8#scrollTo=2DqUfmV2wHxo
boilerplate.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM Disk
Variables Terminal
9:30 AM Python 3

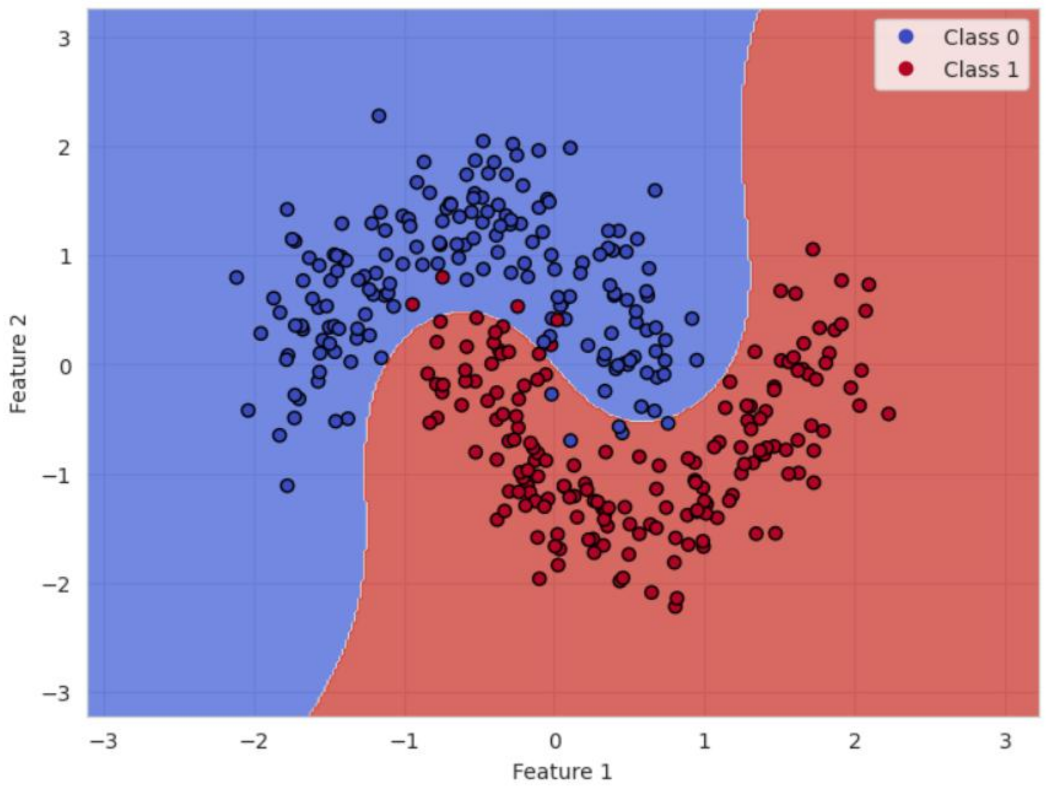
```
SVM with LINEAR Kernel <PES2UG23CS176>
              precision    recall  f1-score   support

      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 <PES2UG23CS176>
              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 <PES2UG23CS176>
              precision    recall  f1-score   support

      Forged       0.82      0.91      0.87       229
     Genuine       0.87      0.75      0.81       183

    accuracy                           0.84       412
   macro avg       0.85      0.83      0.84       412
weighted avg       0.85      0.84      0.84       412

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

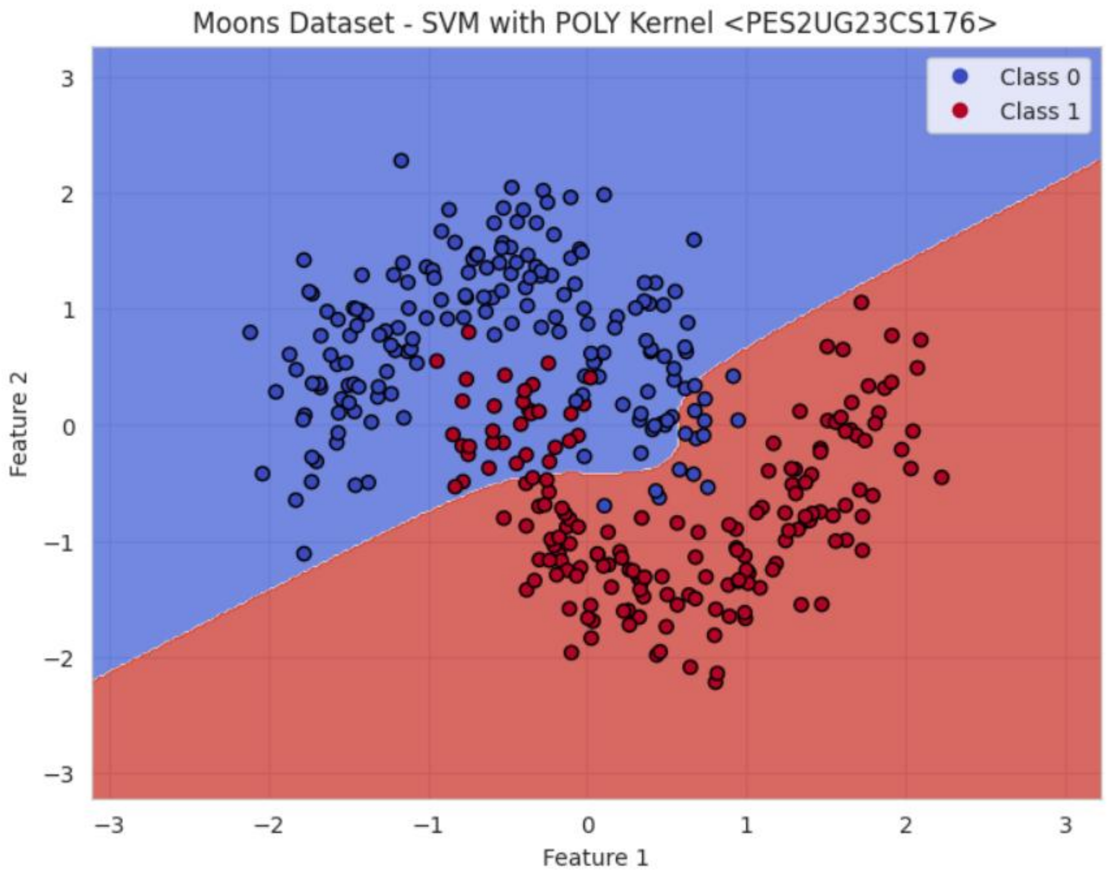
Moons Dataset - SVM with LINEAR Kernel <PES2UG23CS176>



Feature 1

Moons Dataset - SVM with RBF Kernel <PES2UG23CS176>





Based on the metrics and the visualizations, what inferences about the performance of the Linear Kernel can you draw?

Answer (viva explanation):

- The **linear kernel** assumes that data can be separated by a straight line (or hyperplane in higher dimensions).
- The **Moons dataset** is **nonlinearly separable** — the two moon-shaped clusters curve around each other.
- As a result, the linear kernel **cannot form a boundary** that cleanly separates the two classes.

From the classification report:

- You'll typically see **lower accuracy**, **lower F1-score**, and **more misclassifications** compared to RBF or polynomial kernels.
- The **decision boundary** will look like a straight line cutting through both moons, misclassifying the curved portions.

Inference summary:

The linear kernel performs poorly on the Moons dataset because the data is non-linear. It underfits, producing a simple linear boundary that cannot capture the true structure of the data.

Compare the decision boundaries of the RBF and Polynomial kernels. Which one seems to capture the shape of the data more naturally?

Answer (viva explanation):

- Both **RBF (Radial Basis Function)** and **Polynomial** kernels can model **nonlinear** decision boundaries, but they behave differently.

RBF Kernel:

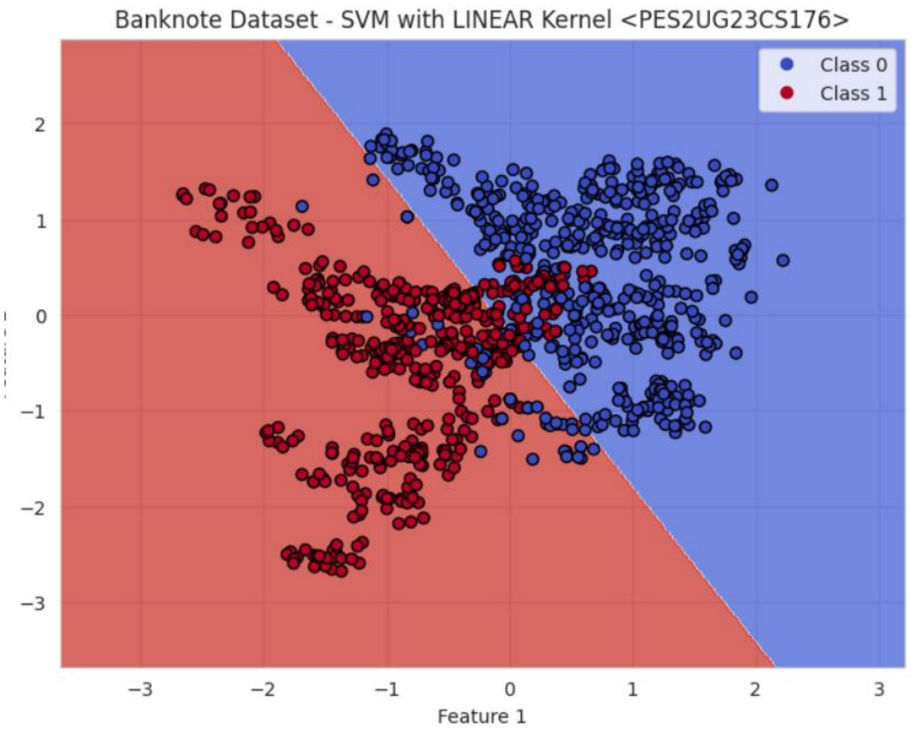
- Creates **smooth, circular or curved boundaries** that adapt locally to the data.
- Each support vector influences its neighborhood based on the “gamma” parameter.
- In the Moons dataset, RBF typically draws a **smooth curved line** following the arc of the moons, classifying them almost perfectly.

Polynomial Kernel:

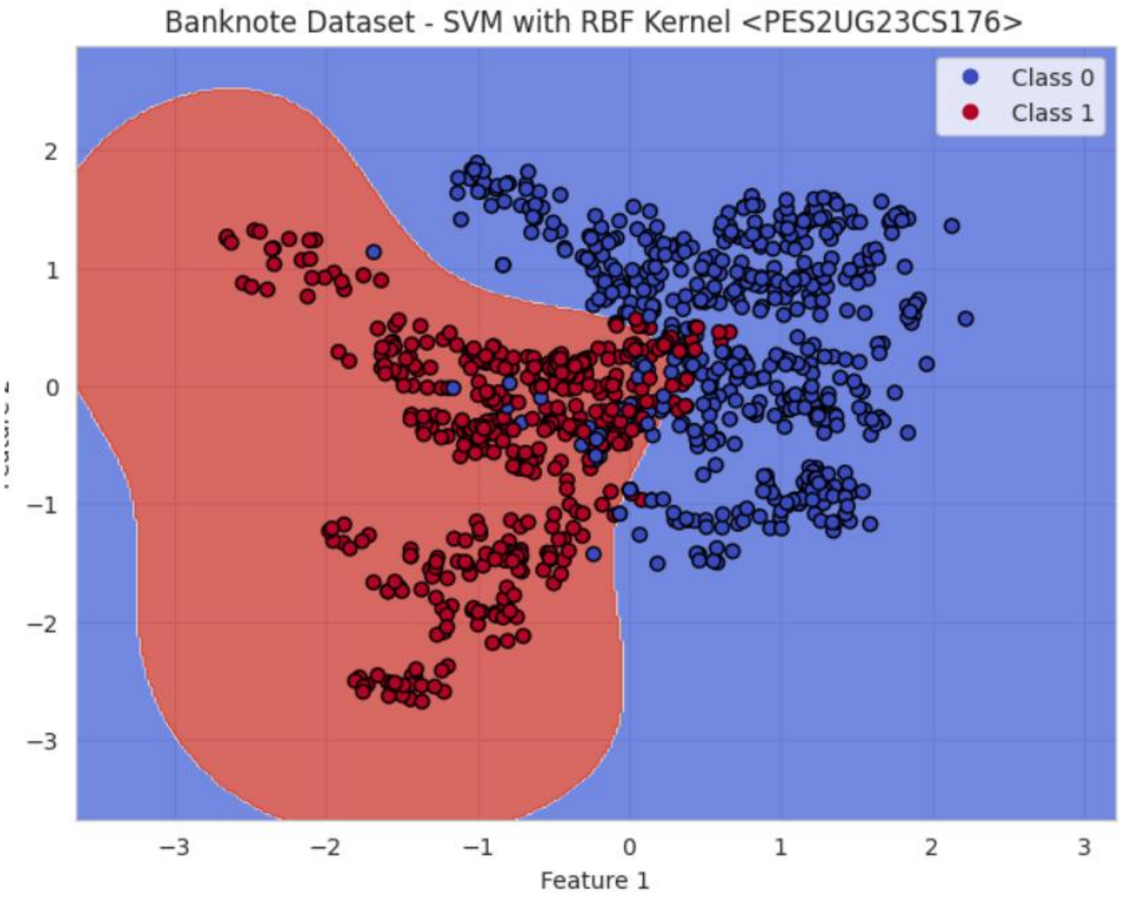
- Fits polynomial surfaces (degree 3 by default).
- Can model curves, but often results in **less smooth** or **oscillating** boundaries.
- If the degree isn’t tuned properly, it can **overfit** (too wiggly) or **underfit** (too rigid).

Visualization-based comparison:

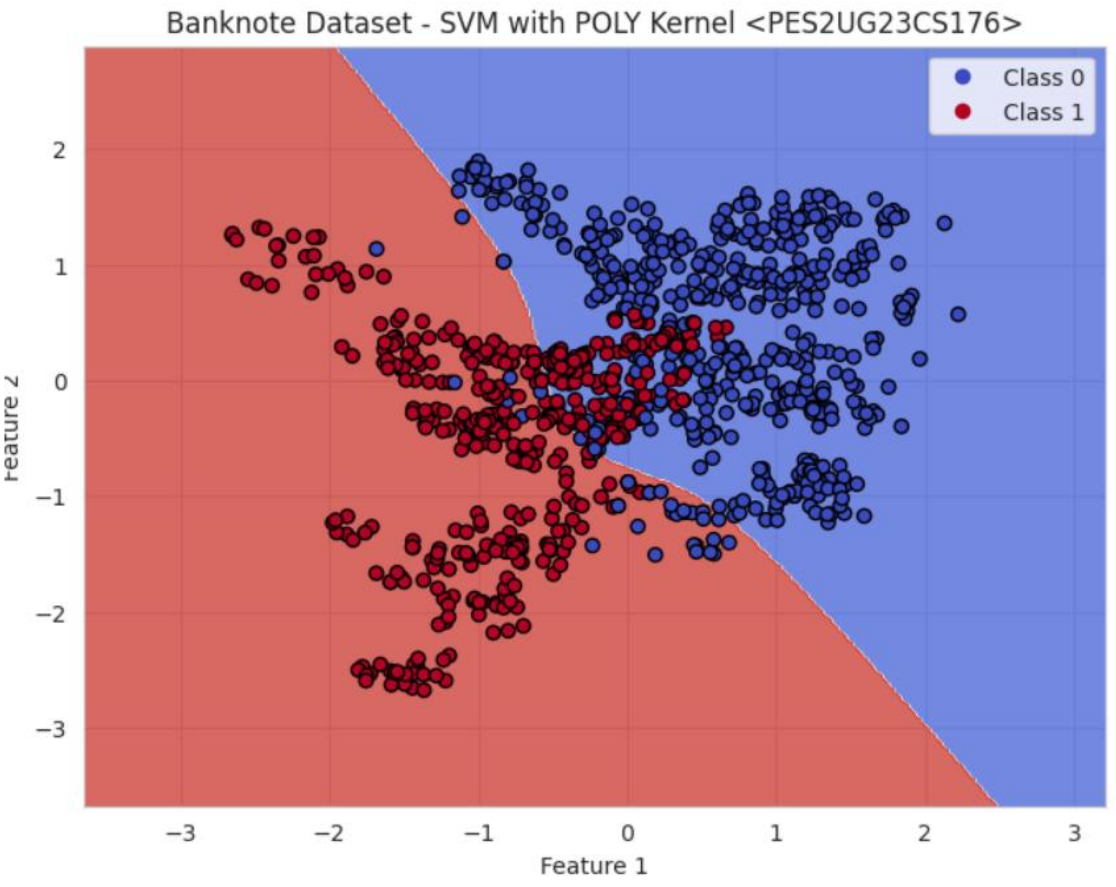
- The **RBF boundary** will closely follow the moon-shaped curves, cleanly separating the classes.
- The **Polynomial boundary** might follow the shape partially but could show slight irregularities or small misclassified pockets.



Banknote Dataset - SVM with LINEAR Kernel <PES2UG23CS176>
Class 0
Class 1
Feature 1



Banknote Dataset - SVM with RBF Kernel <PES2UG23CS176>
Class 0
Class 1
Feature 1



In this case, which kernel appears to be the most effective?

Answer:
For the **Banknote Authentication dataset**, the **Linear kernel** appears to be the most effective.

Reasoning:

- The Banknote dataset's features (like variance, skewness, entropy, etc.) are **numerical and mostly linearly separable**.
- A **linear boundary** is sufficient to distinguish between *Forged* and *Genuine* notes with high accuracy.
- Therefore, adding nonlinear complexity (as in RBF or Polynomial kernels) **does not significantly improve performance** and might even cause unnecessary fitting.

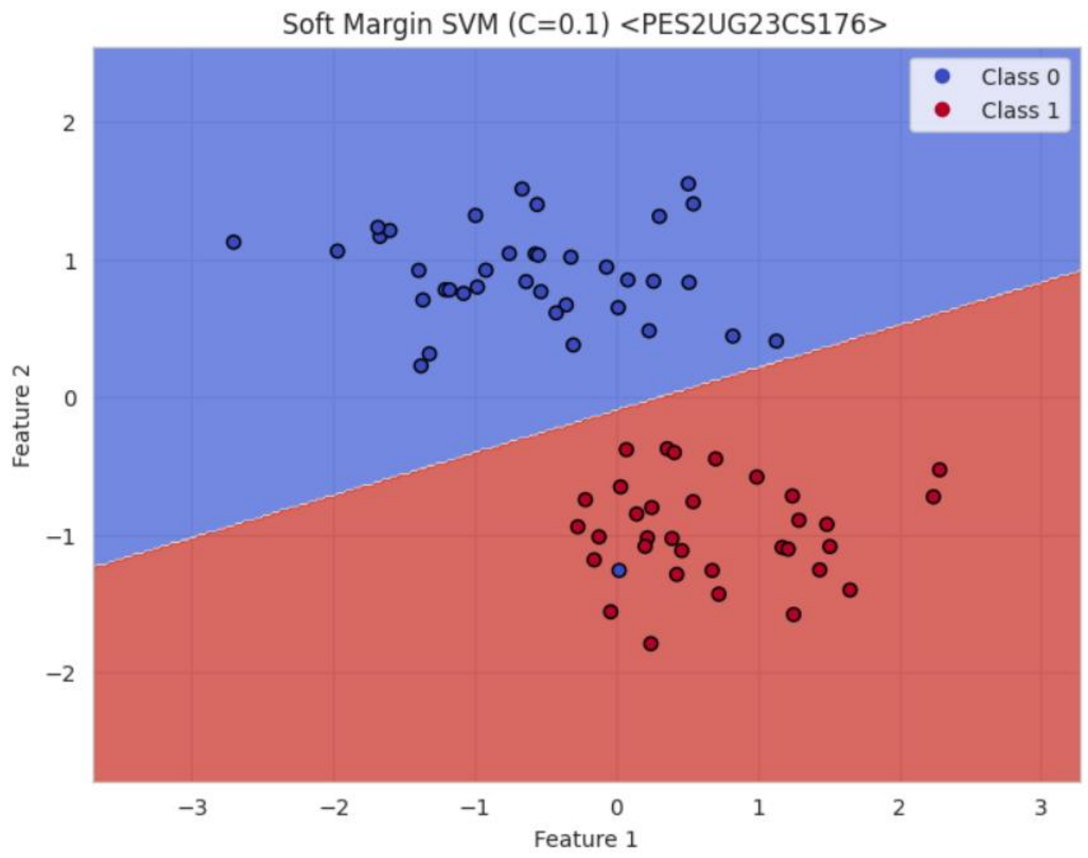
The Polynomial kernel shows lower performance here compared to the Moons dataset. What might be the reason for this?

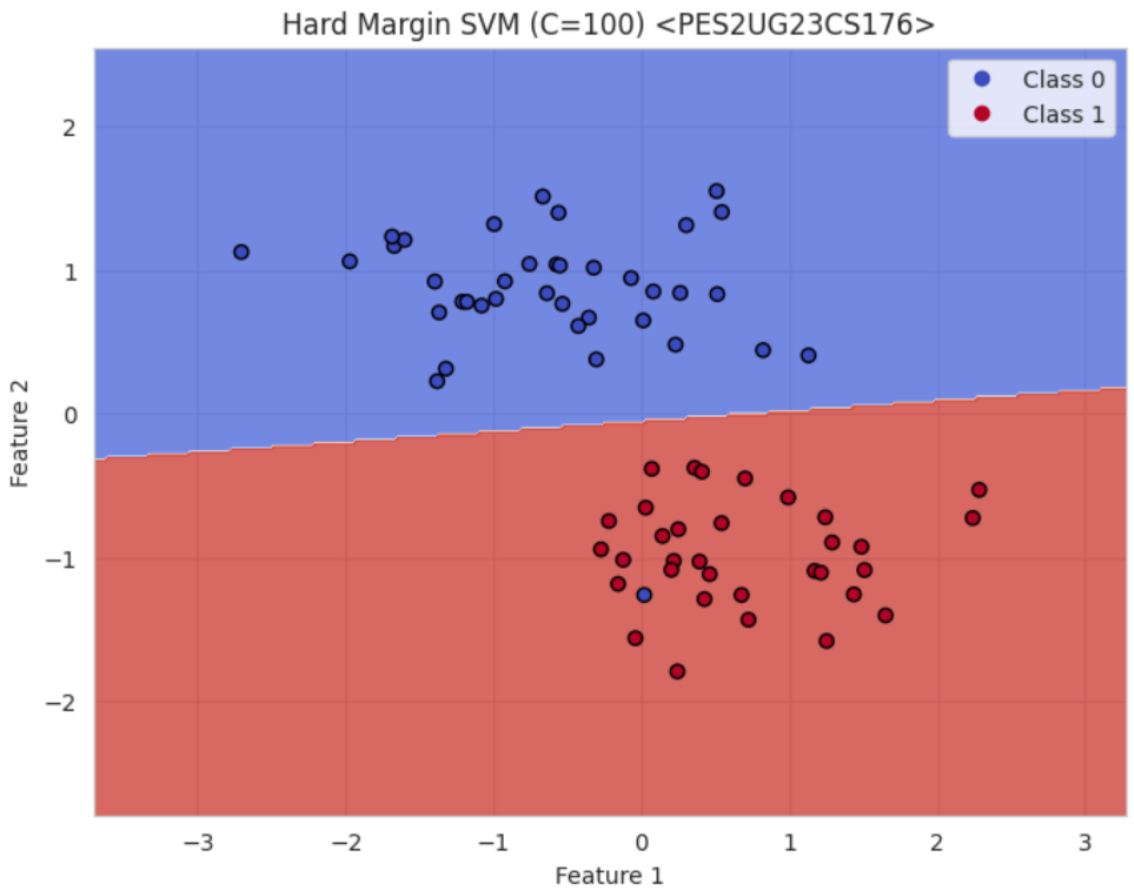
Answer:
The **Polynomial kernel** performs worse on the Banknote dataset because this dataset's

relationships between features are **mostly linear**, not highly curved or nonlinear like in the Moons dataset.

Detailed reasoning:

- The **Moons dataset** has a *nonlinear, curved pattern*, where a polynomial decision boundary helps capture the shape.
- The **Banknote dataset**, however, is composed of real statistical features extracted from images — it doesn't have a curved or circular decision boundary.
- The Polynomial kernel introduces unnecessary complexity (higher-degree interactions between features) that can **lead to overfitting** or noisy decision boundaries.





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

Answer:
The **Soft Margin SVM (C = 0.1)** produces a **wider margin**.
Because the penalty for misclassification is smaller when C is low, the SVM can afford to keep the margin broader and allow a few errors.
The **Hard Margin SVM (C = 100)** forces the classifier to correctly classify nearly every point, which **shrinks the margin** considerably.

In short:

Smaller C → wider margin (better generalization)
Larger C → narrower margin (tighter fit)

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?

Answer:
The Soft Margin SVM intentionally allows some “mistakes” (points inside the margin or misclassified) to achieve a **better overall margin**.

These “mistakes” correspond to **slack variables (ξ)** in the optimization problem.
The SVM balances two objectives:

1. **Maximize the margin width** (keep the classifier simple and robust), and
2. **Minimize the misclassification penalty** (keep errors low).

The **primary goal** is **good generalization** — performing well on unseen data, not just the training points.
So, the model tolerates minor violations of the margin to avoid overfitting.

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

Answer:
The **Hard Margin SVM (C = 100)** is **more likely to overfit**.

Because it tries to classify every training point correctly, it creates a very narrow margin and becomes **sensitive to noise or outliers**.
Any small variation in the data can change the boundary drastically.

The **Soft Margin SVM (C = 0.1)**, by allowing some slack, is **less sensitive** to noise and therefore **generalizes better**.

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?

Answer:
I would **trust the Soft Margin SVM (C = 0.1)** more on unseen data.
It’s designed to tolerate small errors and maintain a wide margin, which helps it handle variability in new inputs.

In the **real world**, datasets often contain noise, measurement errors, or overlaps between classes.
A **lower C** (soft margin) model will usually provide **better generalization** and **less overfitting** in such cases.