Exploration of Support Vector Machines: From Linear Margins to RBF Decision Surfaces

Data Science and Machine Learning Module

December 2025

Abstract

This report details the implementation and analysis of Support Vector Machines (SVM). We explore the geometric "widest street" strategy, the impact of the slack penalty C, and the transition from linear hyperplanes to non-linear kernels. Using datasets ranging from perfectly separable and overlapping to interlocking "two moons" and "concentric circles" we demonstrate how RBF kernels and grid search optimization (for C and γ) allow for complex manifold classification.

1 Introduction

The objective of this project was to transition from probabilistic classification models (Logistic Regression) to a geometric framework using Support Vector Machines (SVM). In standard Logistic Regression, the decision boundary is derived from maximizing the likelihood of class membership. In contrast, SVM implements the "Widest Street Strategy," seeking a hyperplane that maximizes the margin—the physical distance between the boundary and the nearest training samples, known as **Support Vectors**.

This report synthesizes the following key project deliverables:

- **Linear SVM Implementation (Assignment 8):** Solving the dual optimization problem to find the optimal weights (w) and bias (b) for perfectly separable data (`linear_separable.csv`) and overlapping data (`linear_overlapping.csv`).
- **Soft-Margin Analysis:** Evaluating the impact of the slack penalty C. We observed that as $C \to \infty$, the model enforces a "Hard Margin," which often leads to **Overfitting** as the boundary becomes overly sensitive to noise or outliers. Conversely, a small C allows misclassifications to achieve a wider margin, effectively preventing overfitting by increasing bias to reduce variance.
- **Kernelized SVM (Assignment 9):** Extending the model to non-linear manifolds using Polynomial and Radial Basis Function (RBF) kernels. We explored how the RBF parameter γ controls the "reach" of individual points.
- **Hyperparameter Optimization:** Utilizing a 3D Grid Search to visualize the Validation Error Surface, identifying the specific (C, γ) coordinates that minimize generalization error.

2 Code Methodology

The implementation was conducted in Python, leveraging `scikit-learn` for model training and `Plotly` for interactive 3D visualization. The methodology followed a consistent pipeline:

2.1 Data Pre-processing and Scaling

SVMs calculate distances (Euclidean or via Kernels); therefore, features must be on the same scale. We used `StandardScaler` to ensure x_1 and x_2 contributed equally to the margin calculation.

2.2 The Training-Validation Split

To assess true performance, data was split (80/20). The `model.fit()` function was strictly applied to the training set. The model's "error" was then measured on the unseen validation set to simulate real-world generalization.

2.3 Hyperparameter Tuning Reflection

The code implemented a nested loop (Grid Search) to iterate through ranges of C and γ.

- **Underfitting Detection:** Observed when γ and C were too low, resulting in a boundary that was too simple (nearly linear) to capture the curvature of the `concentric_circles` or `two_moons`.
- **Overfitting Detection:** Observed when γ was high (e.g., $\gamma = 50$). The RBF kernel created tight "islands" around specific data points, leading to perfect training accuracy but high validation error.

2.4 Visual Analytics

For 2D data, we used `matplotlib.contourf` to visualize the non-linear decision regions. For hyperparameter analysis, we transformed C and γ into log_{10} space for plotting on a 3D Surface, allowing us to pinpoint the "optimal valley" where the error is minimized.

3 Theory

3.1 Separating Boundary and Gutter in linear 2D

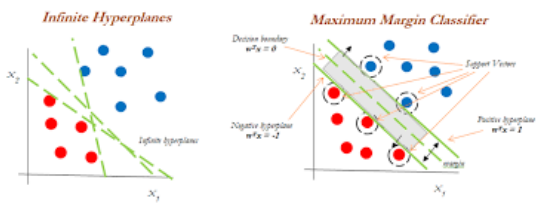


Figure 1: Linear data classified by Linear SVM

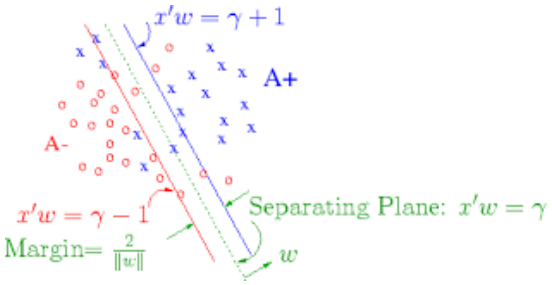


Figure 2: Non-linear data transformed and classified by Polynomial SVM

If the distance is greater than 1 or -1 the points are labeled as 1 and -1 respectively. Let us assume the separating boundary as ax+by+c=0. The model adjusts the coefficients in such a manner that both the gutter lines are 1 unit distance apart and touches the nearest labeled points.

3.2 SVM Loss Function Derivation

Below is the flowchart representing the SVM Loss Function's implementation:

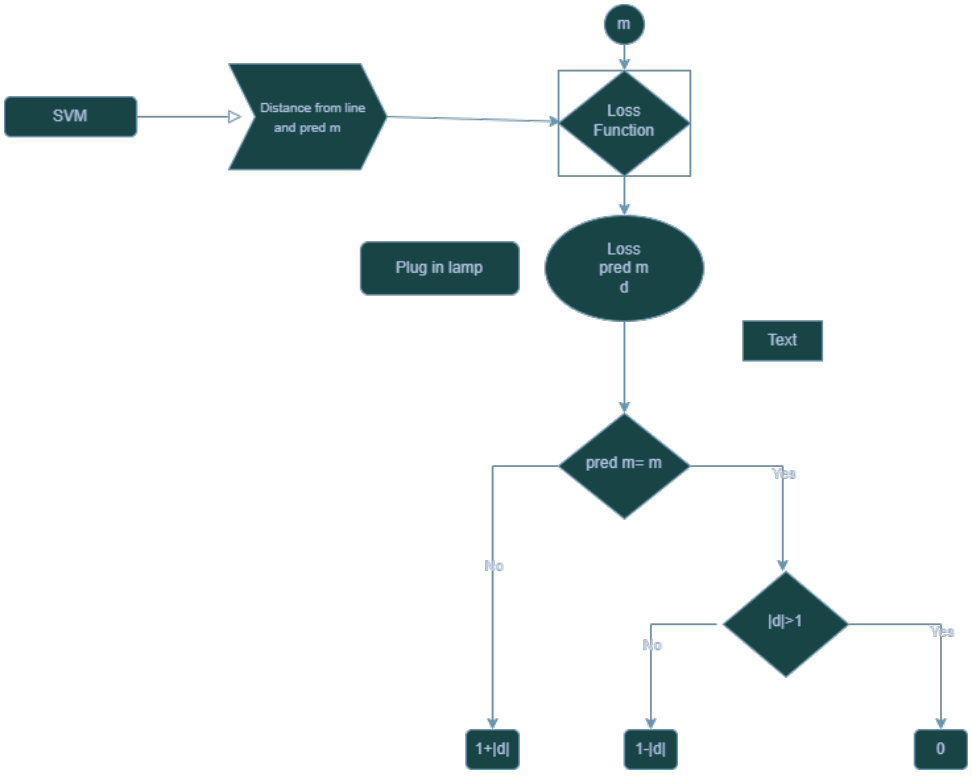


Figure 3: Flowchart of the implemented algorithm.

The Support Vector Machine aims to maximize the margin while minimizing classification error. We define the **Hinge Loss** as:

$$L_{hinge} = \max(0, 1 - m \cdot d) \tag{1}$$

To account for the geometric margin, we consider the distance from a point (x_1, y_1) to the decision boundary $ax + by + c = 0$. Setting the distance to the "gutters" as 1, we have:

$$\frac{|ax_1 + by_1 + c|}{\sqrt{a^2 + b^2}} = \frac{1}{\sqrt{a^2 + b^2}} \tag{2}$$

To maximize the margin, we must minimize the denominator's reciprocal. In optimization terms, this is equivalent to minimizing $\frac{1}{2}(a^2 + b^2)$. Combining these terms, the **Overall Loss Function** (Primal Form) becomes:

$$LF = \max(0, 1 - m \cdot d) + \frac{1}{2}(a^2 + b^2) \tag{3}$$

3.3 Extending to Multidimensional Data

The decision boundary is defined by the hyperplane $w \cdot x + b = 0$. The margin is the distance between two parallel "gutters":

$$w \cdot x + b = 1 \quad \text{and} \quad w \cdot x + b = -1 \tag{4}$$

The data points lying exactly on these gutters are the **Support Vectors**.

To handle overlapping data, we use the Soft-Margin approach. The objective function incorporates **Hinge Loss**:

$$L = \max(0, 1 - y_i(w \cdot x_i + b)) \tag{5}$$

The hyperparameter C controls the trade-off:

- **Small C:** High error tolerance, wider margin (Lower variance).
- **Large C:** Low error tolerance, narrower margin (Higher variance).

3.4 The Kernel Trick: From Linear to RBF

To solve non-linear problems, we move from the Primal form to the **Dual Form**, which uses dot products. We replace these dot products with Kernel functions.

3.4.1 Polynomial Kernel Dynamics

The Polynomial Kernel facilitates non-linear separation by projecting input features into a higher-dimensional space where a linear hyperplane can be established. It is defined as:

$$K(x_i, x_j) = (\gamma x_i^\top x_j + r)^d \tag{6}$$

Key Tuning Parameters:

- **Degree (d):** Controls boundary complexity. While $d = 2$ (Quadratic) is optimal for radial patterns like `concentric_circles`, higher degrees ($d > 3$) increase the risk of **overfitting** by creating overly sensitive decision boundaries.
- **Independent Term (r):** Also known as `coef0`, it balances the influence of high-degree versus low-degree terms. A higher r ensures the model does not ignore lower-order feature interactions.
- **Scale (γ):** Adjusts the magnitude of the feature interactions within the polynomial expansion.

3.4.2 Impact of Tuning γ on Data Fitting in RBF Kernels

The hyperparameter γ controls the radius of influence of the support vectors. Tuning this parameter is a balancing act between capturing global patterns and memorizing local noise:

- **Low γ (Underfitting):** When γ is small, each support vector has a wide "reach." The model considers distant points, resulting in a smooth, simplified decision boundary. On complex manifolds like `two_moons`, this often fails to capture the curvature, leading to high bias.
- **High γ (Overfitting):** A large γ creates a localized, narrow reach. The boundary must twist sharply to accommodate individual training points, often resulting in "islands of influence." While this yields high training accuracy, it leads to high variance and poor generalization on validation data.

Table 1: Summary of γ Effects

γ Value	Influence Radius	Boundary Topology	Model State
Low	Large/Far	Smooth/Simplified	Underfitting
Optimal	Balanced	Manifold-Adaptive	Generalization
High	Small/Narrow	Jagged/Islands	Overfitting

4 Implementation

The following implementation demonstrates the transition from linear classification to non-linear manifold estimation using the RBF kernel.

4.1 Linear SVM and Hyperparameter Sweep

The code below initializes a linear SVM and performs a "C-Sweep" to evaluate the impact of the slack penalty on validation accuracy.

```
# 1. Linear SVM Training
X_train = np.column_stack((train_df['x1'].values, train_df['x2'].values
    ))
y_train = train_df['y'].values.astype(float)

model = SVC(kernel='linear', C=1.0)
model.fit(X_train, y_train)

# 2. Validation Accuracy Sweep (C-Sweep)
C_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
valid_acc = []

for c in C_values:
    model = SVC(kernel='linear', C=c)
    model.fit(X_train, y_train) # Training on train set
    y_val_pred = model.predict(X_val)
    acc = np.sum(y_val_pred == y_val) / len(y_val)
    valid_acc.append(acc)
```

Listing 1: Linear SVM and Validation Sweep

4.2 RBF Kernel and 3D Decision Surface Visualization

To visualize how the RBF kernel creates a non-linear decision boundary, we construct a mesh grid over the feature space and calculate the decision scores (Z) for every coordinate.

```
# 1. Train RBF Model
model = SVC(kernel='rbf', C=10, gamma=0.01)
model.fit(X_train, y_train)

# 2. Mesh Grid Formation for Plotting
x_min, x_max = X_train[:, 0].min() - 0.5, X_train[:, 0].max() + 0.5
y_min, y_max = X_train[:, 1].min() - 0.5, X_train[:, 1].max() + 0.5
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                     np.linspace(y_min, y_max, 100))

# 3. Calculate Decision Scores
grid_points = np.c_[xx.ravel(), yy.ravel()]
Z = model.decision_function(grid_points)
Z = Z.reshape(xx.shape)

# 4. 3D Plotting with Matplotlib
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(xx, yy, Z, cmap='coolwarm', alpha=0.7)
ax.contour(xx, yy, Z, levels=[0], colors='black', linestyles='dashed')
```

Listing 2: RBF Surface and Mesh Grid Plotting

4.3 Interactive Visualization

To better analyze the manifold, an interactive 3D surface was generated using `Plotly`, mapping the decision function as the vertical dimension (Z) against the two features.

```
import plotly.graph_objects as go

fig = go.Figure(data=[go.Surface(z=Z, x=xx, y=yy, colorscale='RdBu')])
fig.update_layout(title='Interactive RBF Gaussian Surface',
                  scene=dict(xaxis_title='X1', yaxis_title='X2',
   zaxis_title='Score'))
fig.show()
```

Listing 3: Interactive Plotly Surface

5 Results

5.1 Linear Experiments

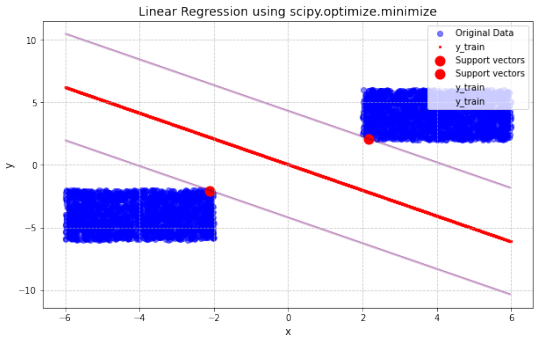


Figure 4: Linear Regression using scipy.optimize.minimize

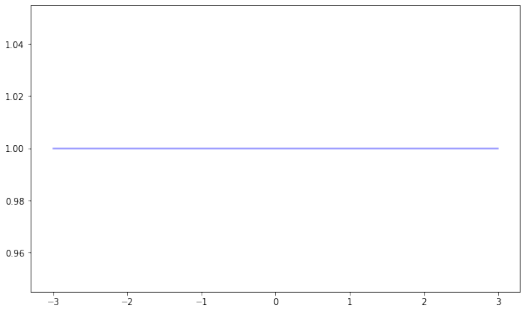
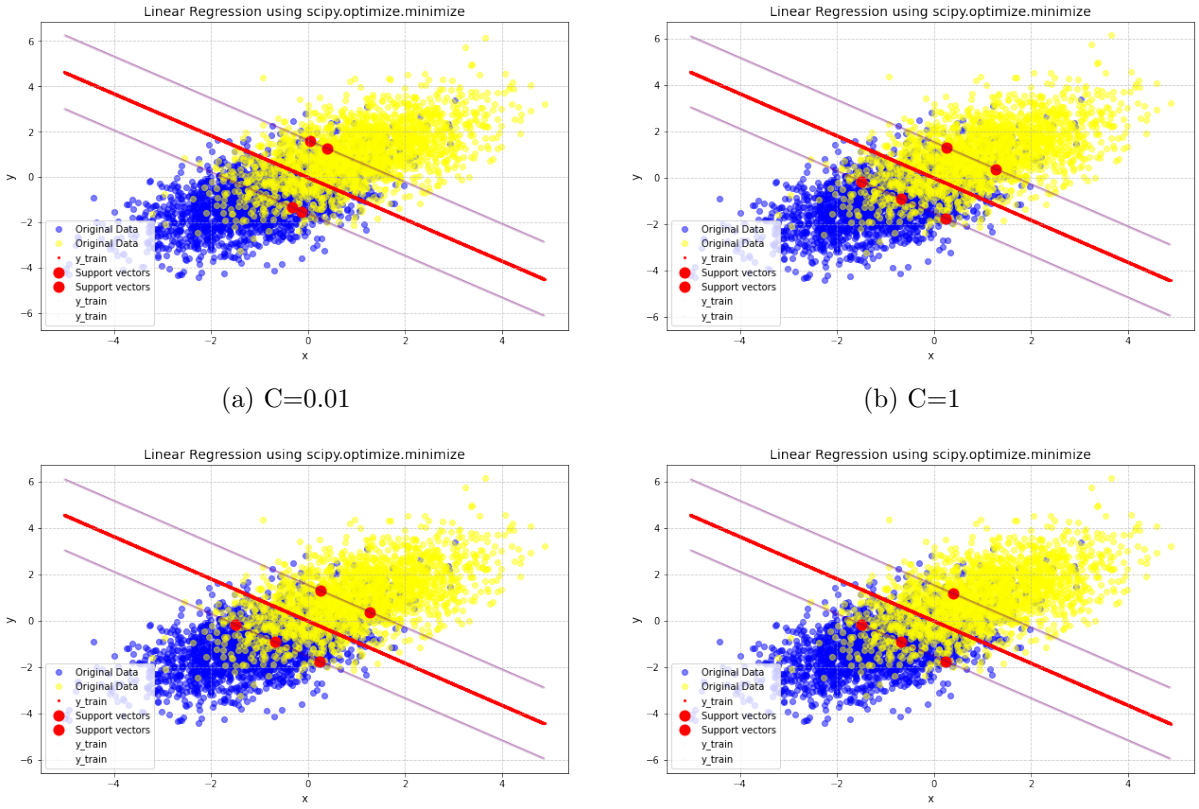


Figure 5: Validation Accuracy v/s C



(a) C=0.01 (b) C=1

(c) C=10 (d) C=100

Figure 6: Comparison of SVM decision boundaries on overlapping dataset with different C values.

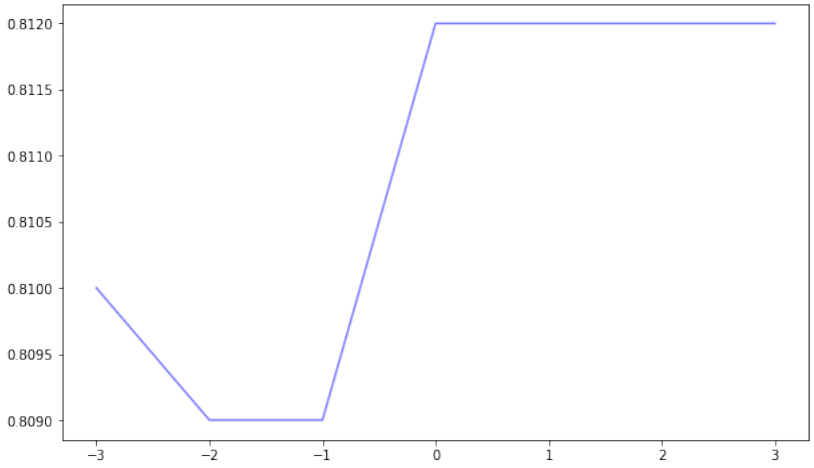


Figure 7: Validation Accuracy v/s C

Using `linear_separable.csv`, the model successfully identified the widest margin. In `linear_overlapping.cs` increasing C from 0.01 to 100 resulted in a significant tightening of the margin and an increase in training accuracy, though validation accuracy peaked at $C = 1$ with validation accuracy = 0.8090.

5.2 Non-Linear RBF Surface

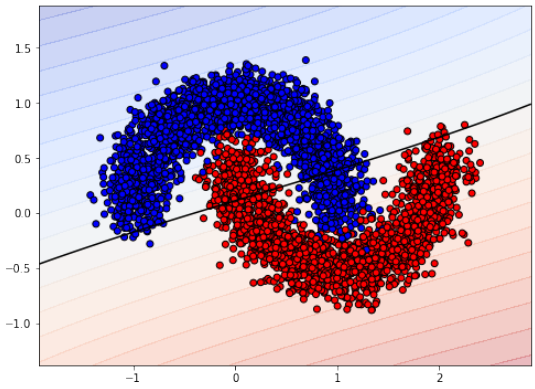


Figure 8:
$\gamma = 0.01$ and C= 10

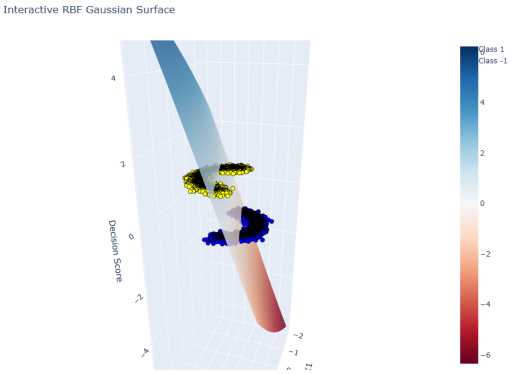


Figure 9: RBF Planar surface

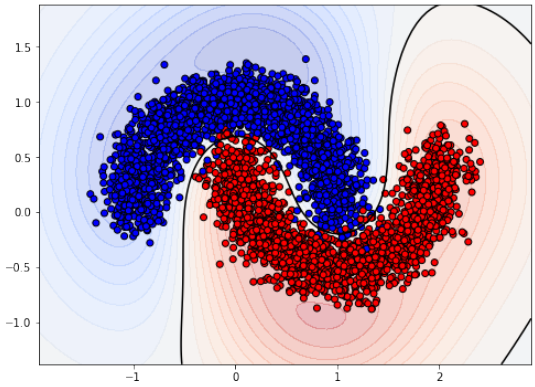


Figure 10:
$\gamma = 1$ and C= 10

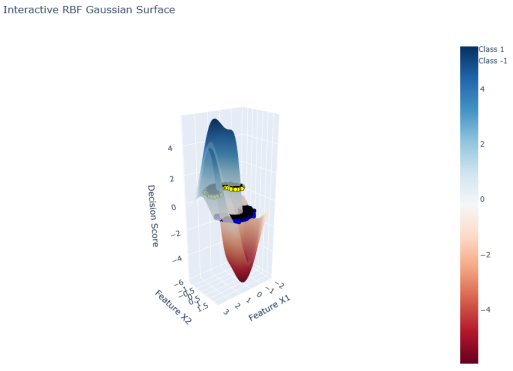


Figure 11: Smooth RBF hills

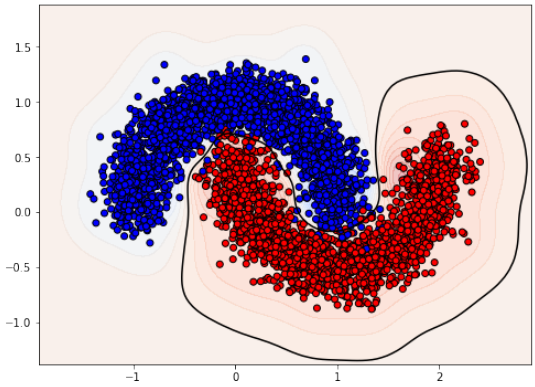


Figure 12:
$\gamma = 50$ and C= 10

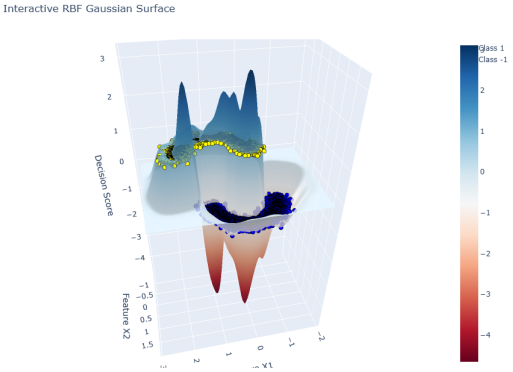


Figure 13: Pointy RBF Peaks

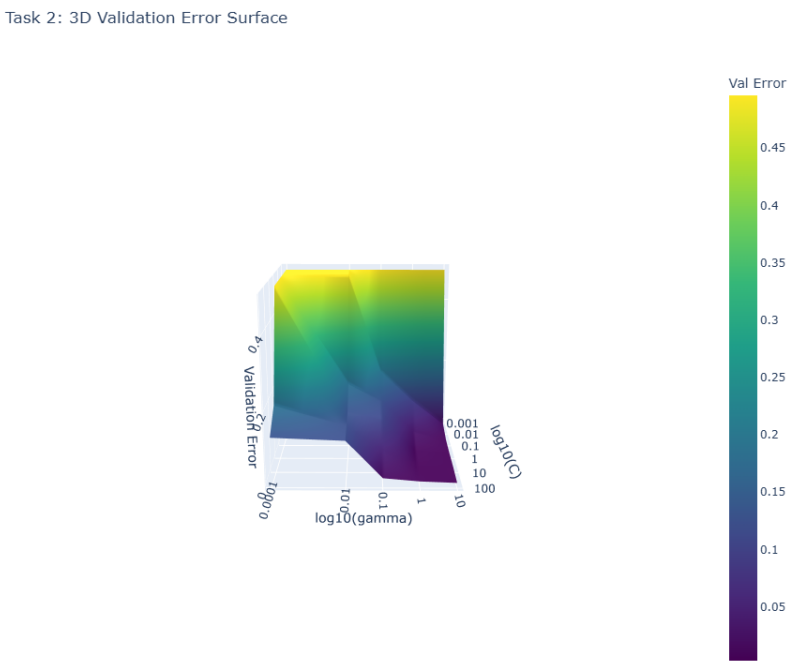


Figure 14: Validation Error v/s γ and C

In the `two_moons` dataset, we observed:

- $\gamma = 0.01$**:** Decision boundary was nearly linear (Underfitting).
- $\gamma = 1$**:** A smooth curve separating the two moons.
- $\gamma = 50$**:** The model created tight circles around each point (Overfitting).
- Lowest Validation Error: **0.0030** for **C=100** and **gamma=10**

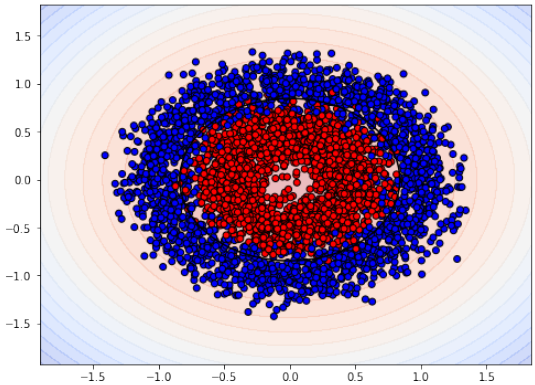


Figure 15:
$\gamma = 0.01$ and C= 10

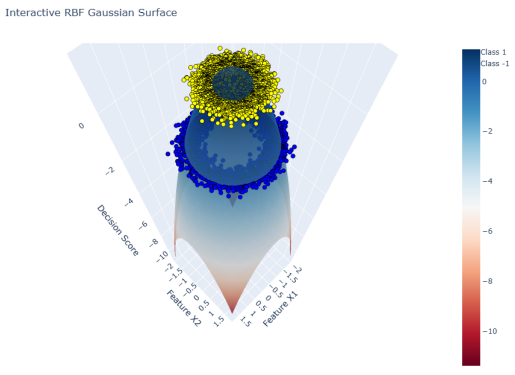


Figure 16: RBF surface

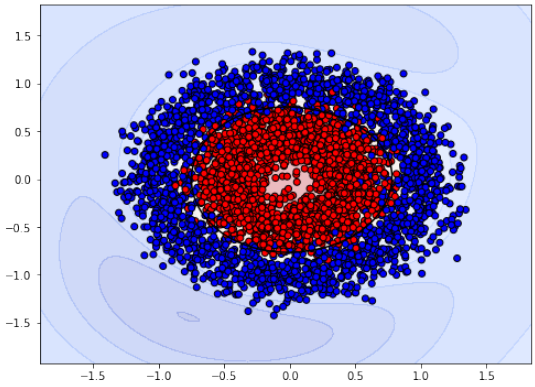


Figure 17:
$\gamma = 1$ and C= 10

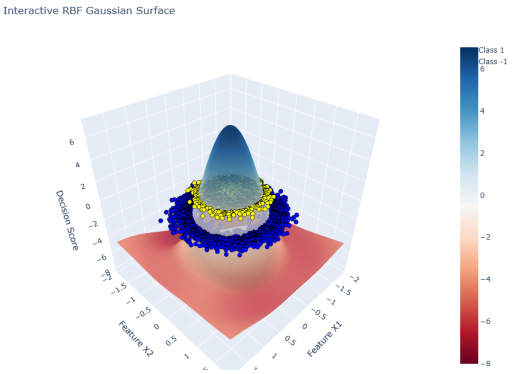


Figure 18: Smooth RBF Hill

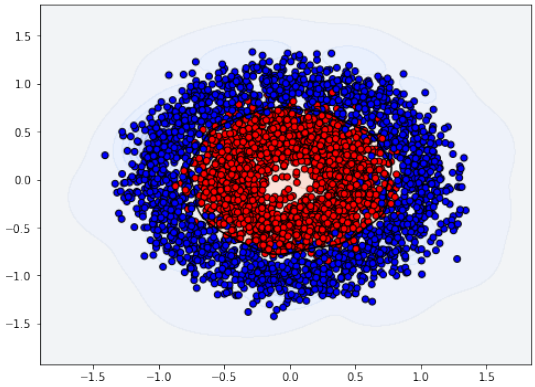


Figure 19:
$\gamma = 50$ and C= 10

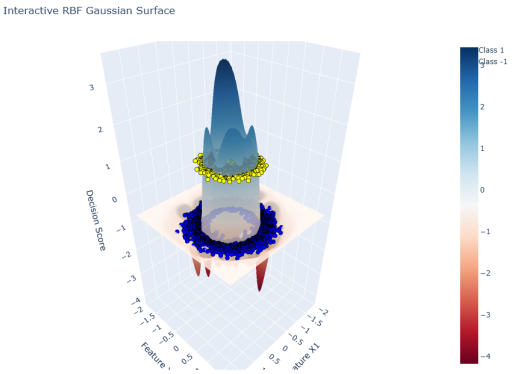


Figure 20: Pointed RBF Peaks

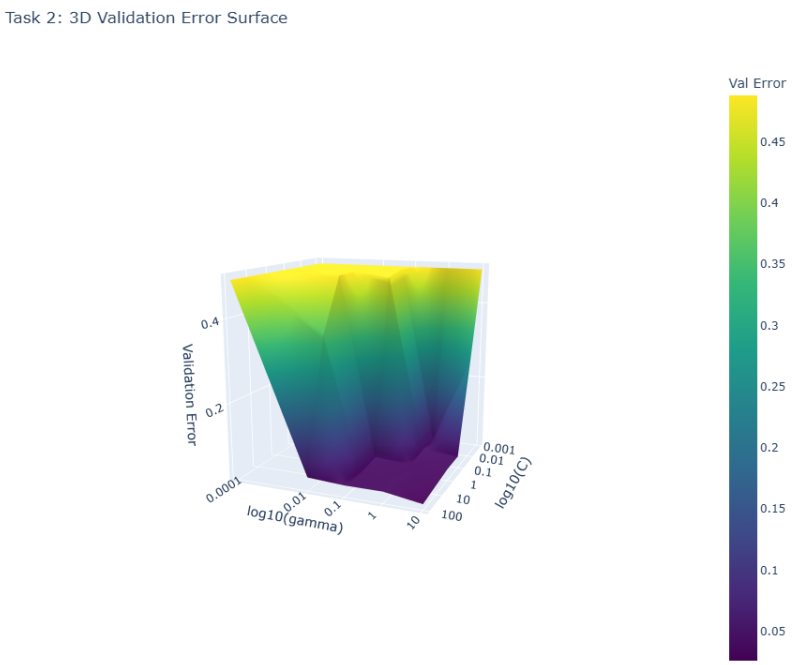


Figure 21: Validation Error v/s γ and C

In the `two_moons` dataset, we observed:

- $\gamma = 0.01$**:** Decision boundary was nearly linear (Underfitting).
- $\gamma = 1$**:** A smooth curve separating the two moons.
- $\gamma = 50$**:** The model created tight circles around each point (Overfitting).
- Lowest Validation Error: **0.0260** for **C=100** and **gamma=10**

6 Discussion

The 3D Grid Search revealed a "valley" of low validation error. For the RBF kernel, C and γ are interdependent. If γ is too high, the model overfits, creating complex Gaussian "islands." If C is too high, the model penalizes every overlap, leading to a jagged boundary. The optimal balance for our dataset was found at $log(C) = 2$ and $log(\gamma) = 0$.

6.1 Methodological Comparison: Logistic Regression vs. SVM

The transition from Logistic Regression to Support Vector Machines represents a shift from likelihood maximization to geometric margin maximization.

Table 2: Conceptual Differences: Probabilistic vs. Geometric Classification

Feature	**Logistic Regression**	**Support Vector Machine**
Objective	Maximize Likelihood	Maximize Geometric Margin
Decision Rule	Probabilistic ($P > 0.5$)	Closest point distance (Widest Street)
Sensitivity	Sensitive to all data points	Sensitive only to Support Vectors
Non-linearity	Requires manual feature engineering	Handled via the "Kernel Trick"
Boundary Goal	Separate classes with a curve	Find the most robust "buffer" zone

6.2 Kernel Suitability and Manifold Complexity

The choice of kernel is dictated by the distribution of the dataset in the feature space.

Table 3: Kernel Selection Guide based on Dataset Geometry

Kernel	Ideal Dataset Shape	Key Limitation
Linear	Linearly separable clusters	Fails on interlocking/curved data
Polynomial	Spherical or concentric shapes	Computationally expensive at high degrees
RBF (Gaussian)	Complex, interlocking manifolds	High risk of overfitting with small γ
Sigmoid	Neural network-like structures	May not represent a valid Mercer kernel

7 Conclusion

Support Vector Machines provide a robust framework for classification by maximizing geometric margins. While linear SVMs are effective for separable data, the RBF kernel extends this capability to non-linear manifolds. Through hyperparameter tuning of C and γ, we can control the model's complexity to achieve optimal generalization on complex datasets.