Support Vector Machine (SVM) - Introduction

🔹 What is SVM?

Support Vector Machine (SVM) is a **supervised machine learning algorithm** used for **classification** and **regression** tasks.
It is most commonly used for **binary classification** (e.g., yes/no, spam/ham).

💡 Basic Idea

SVM finds the **best boundary (hyperplane)** that separates data points of different classes.
It tries to **maximize the margin** — the distance between the hyperplane and the nearest data points from each class.

⚙️ How It Works

1. Plot data points in space (2D or 3D).
2. Draw possible lines (or planes) that separate the classes.
3. Choose the **line/plane with the maximum margin**.
4. For non-linear data, use a **kernel trick** to project data into a higher dimension where it becomes separable.

🧩 Key Terms

Term	Description
Hyperplane	The decision boundary separating classes.
Support Vectors	Data points closest to the hyperplane; they define the margin.
Margin	The distance between the hyperplane and the support vectors.
Kernel Trick	A mathematical method to handle non-linear data by transforming it into higher dimensions.

🧮 Types of SVM

- **Linear SVM** → Straight-line decision boundary

- **Non-linear SVM** → Uses kernel functions (Polynomial, RBF, Sigmoid) to handle complex data

---

💼 Applications

- Text & spam classification
- Face and object recognition
- Medical diagnosis (e.g., tumor classification)
- Fraud detection in finance

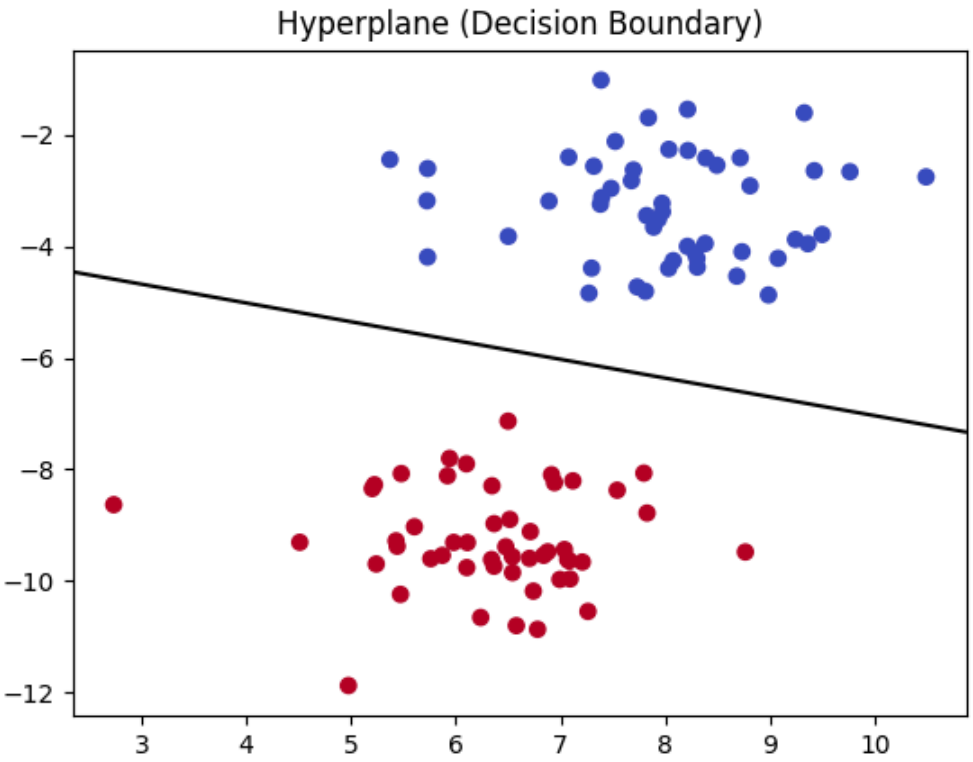
Support Vector Machine (SVM) – Key Terminologies

1. **Hyperplane**

- The **decision boundary** that separates data points of different classes.
- In **2D**, it's a **line**; in **3D**, it's a **plane**; in higher dimensions, it's a **hyperplane**.
- SVM tries to find the **best hyperplane** that divides the classes with the **maximum margin**.

In [19]:
```
#![hyperplane](./hyperplane.png)
from IPython.display import Image

# Display each image in the notebook
Image(filename='hyperplane.png')
```

Out[19]:

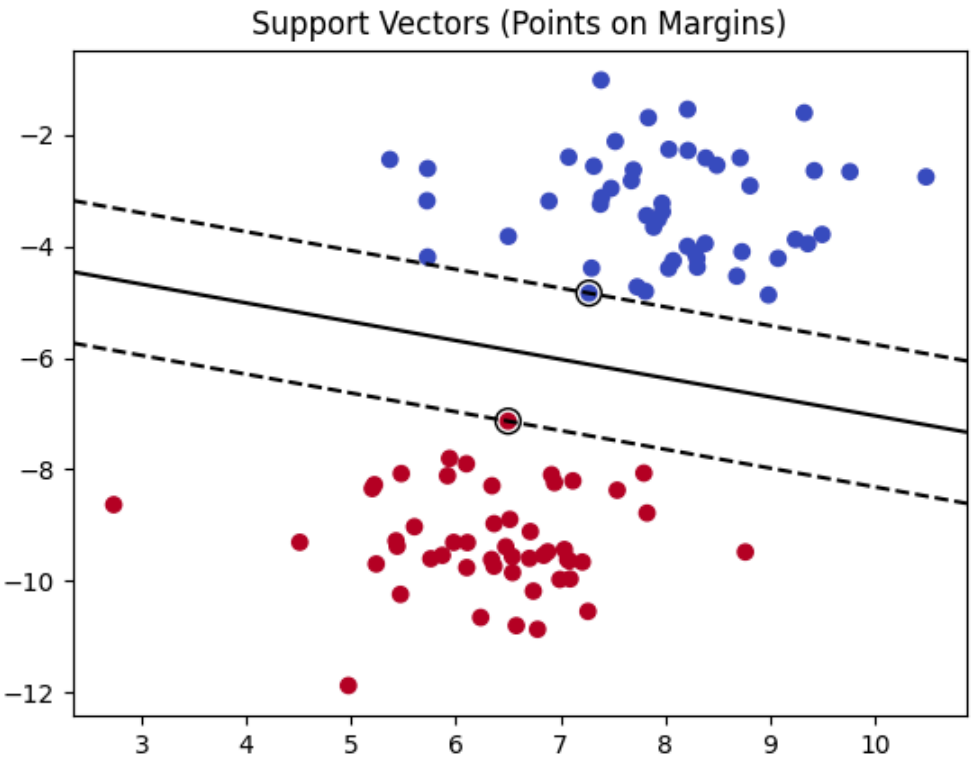


2. Support Vectors

- The **data points that are closest to the hyperplane**.
- These points directly influence the **position and orientation** of the hyperplane.
- If any of these points are moved or removed, the hyperplane will change.
- Hence, they are called **"support"** vectors — they support the boundary.

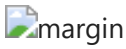
In [20]:
```
from IPython.display import Image

# Display each image in the notebook
Image(filename='supportvectors.png')
```

Out[20]:



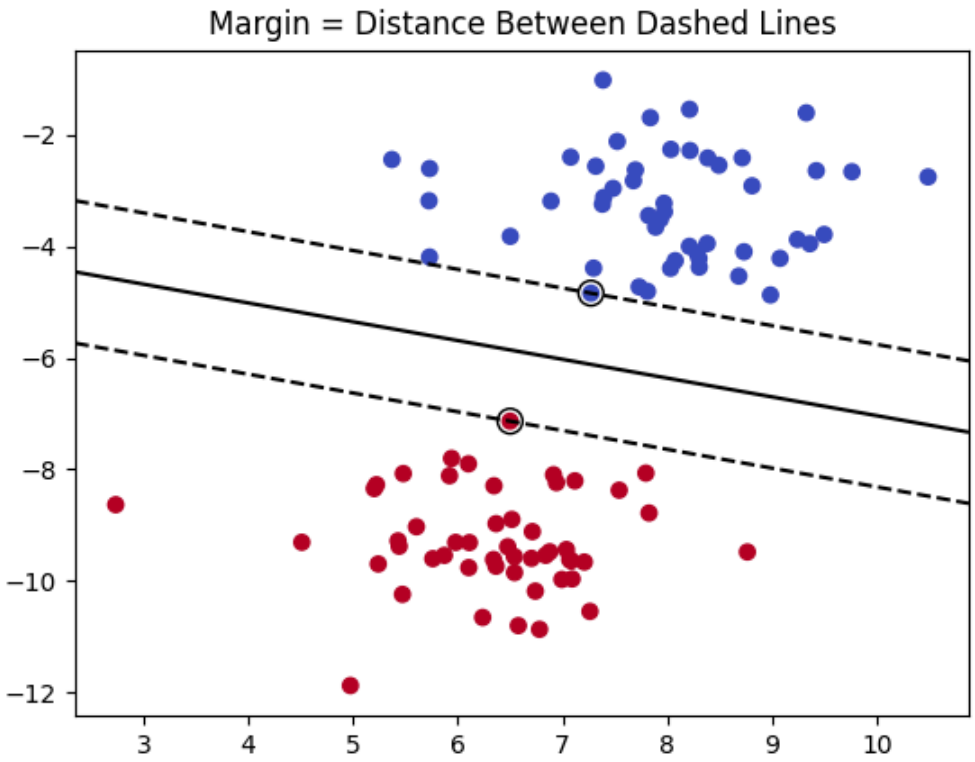
3. Margin

- The **distance between the hyperplane and the nearest support vectors** from each class.
- SVM aims to **maximize this margin** for better separation and generalization.
- Larger margin → better model performance and stability.



In [21]:
```
from IPython.display import Image

# Display each image in the notebook
Image(filename='margin.png')
```

Out[21]:



4. Kernel Trick

- A **mathematical technique** used to handle **non-linear data**.
- It transforms the input data into a **higher-dimensional space**, where it becomes **linearly separable**.
- Common kernel types:
  - **Linear Kernel** → Straight-line separation
  - **Polynomial Kernel** → Curved separation
  - **RBF (Radial Basis Function)** → Circular or complex separation
  - **Sigmoid Kernel** → S-shaped decision boundary

---

5. Decision Boundary

- The **line or surface** that separates one class from another based on the SVM model.
- Points on one side belong to one class, and those on the other side belong to the second class.

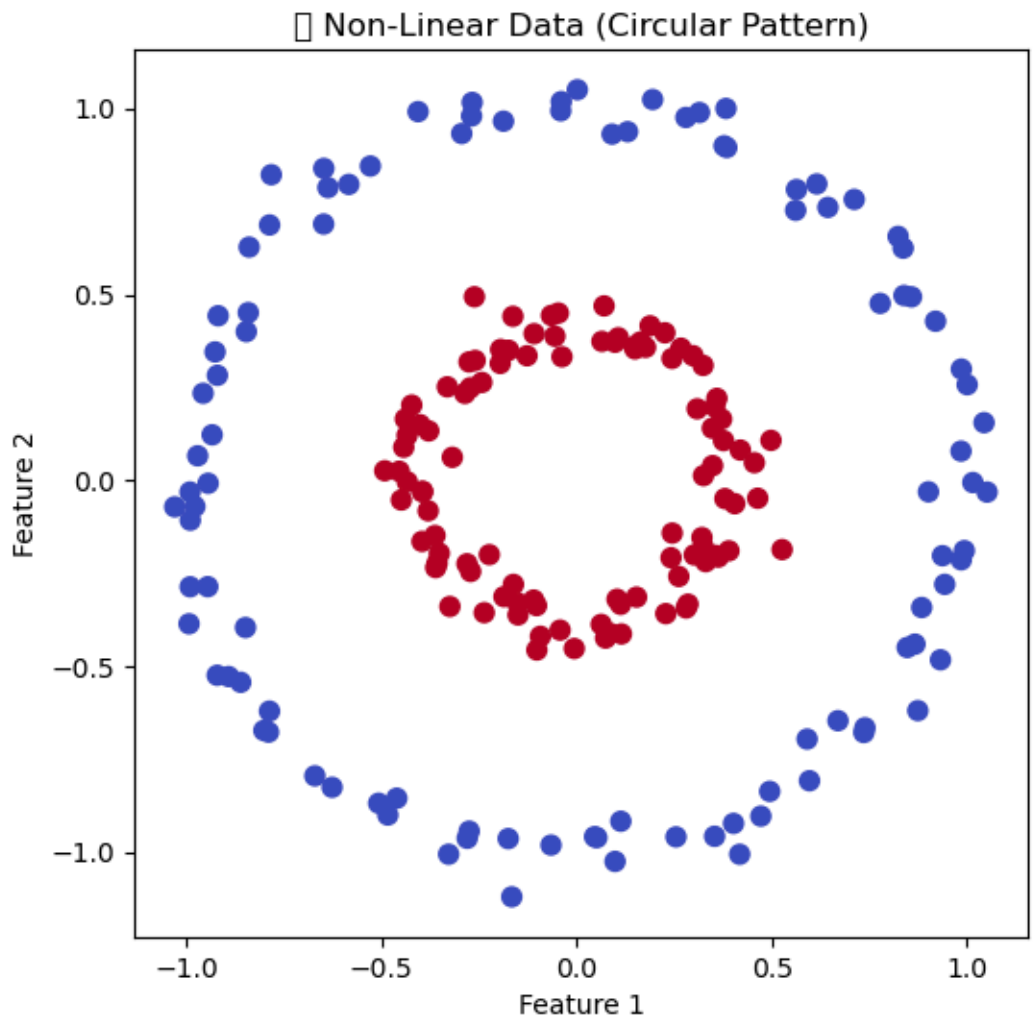
---

6. Margin Width (Distance Between Support Vectors)

- The **total width** between the two margin boundaries.

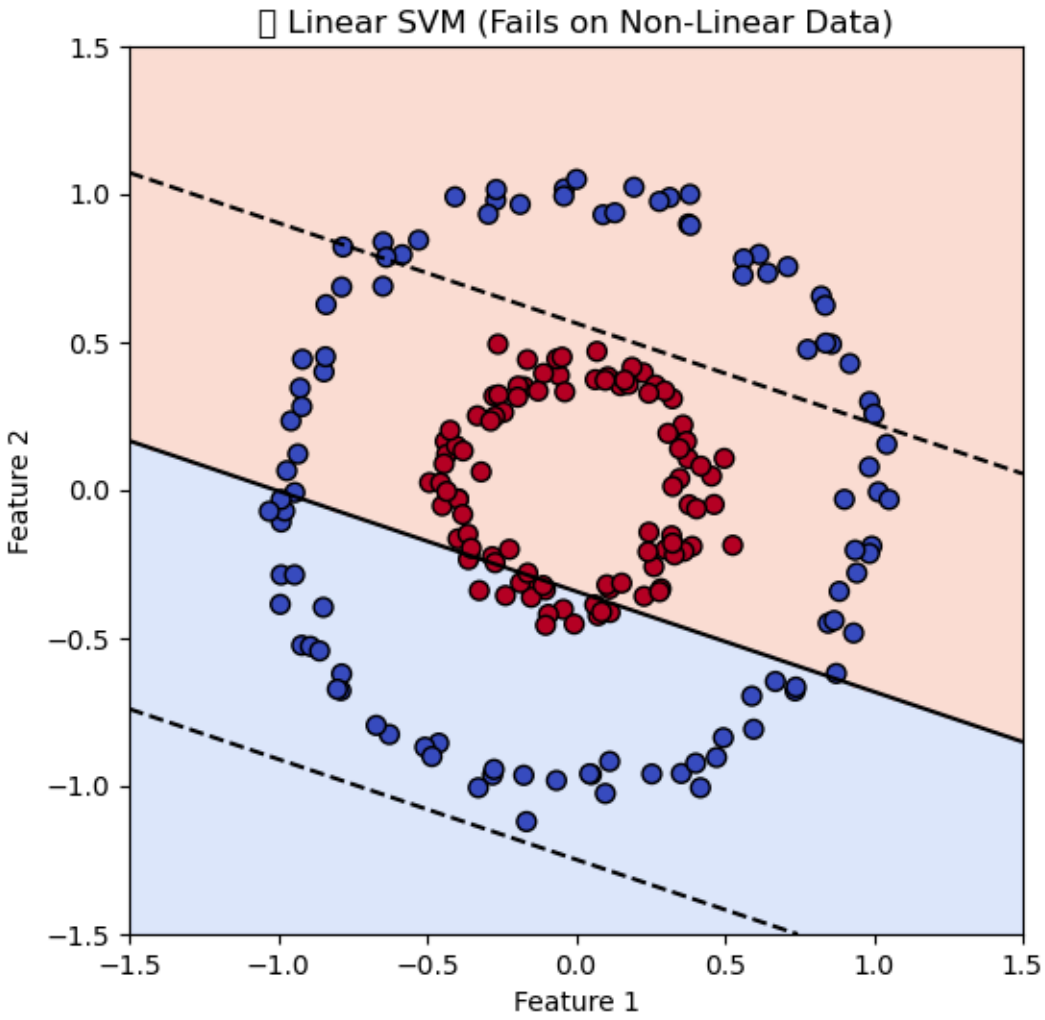
In [22]:
```
#![nonlinear](./non_linear_data.png)
from IPython.display import Image

# Display each image in the notebook
Image(filename='non_linear_data.png')
```

Out[22]:



In [23]:
```
#![linear_svm](./Linear_Linear_SVM_(Fails_on_Non-Linear_Data).png)
from IPython.display import Image

# Display each image in the notebook
Image(filename='Linear_Linear_SVM_(Fails_on_Non-Linear_Data).png')
```

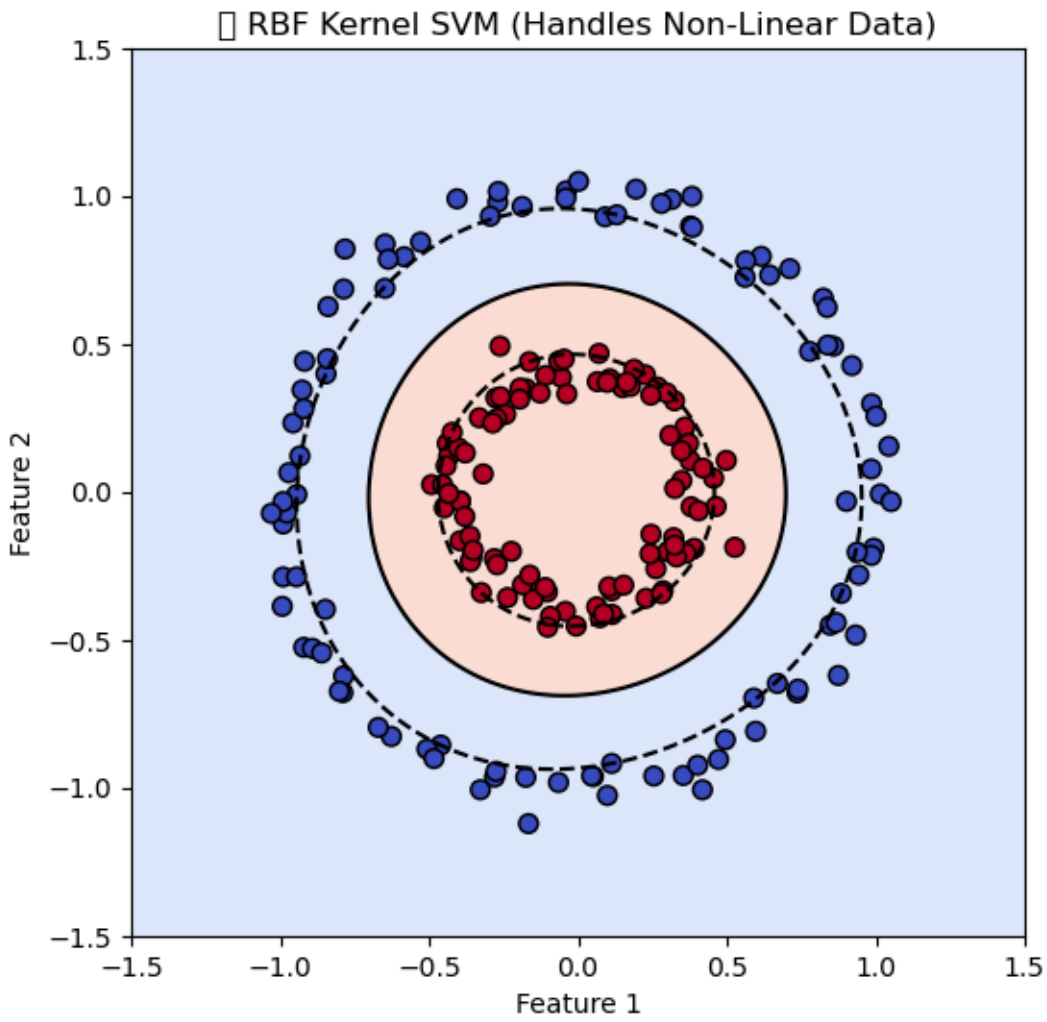
Out[23]:



In [24]:

```
#![rbf_svm](./RBF_RBF_Kernel_SVM_(Handles_Non-Linear_Data).png)
from IPython.display import Image

# Display each image in the notebook
Image(filename='RBF_RBF_Kernel_SVM_(Handles_Non-Linear_Data).png')
```

Out[24]:



Hyper Parameter

What are Hyperparameters?

👉 Hyperparameters are **settings or control**s that you choose before training your SVM model. They decide how the model learns and **how flexible or strict** it should be while separating data.

Think of them like tuning knobs 🔧 on a machine.

Important SVM Hyperparameters

There are 3 main hyperparameters you'll use often:

Hyperparameter	Symbol / Name	Works With	Simple Meaning
1️⃣ C	Regularization parameter	All kernels	Controls how strictly the model tries to classify training data

Hyperparameter	Symbol / Name	Works With	Simple Meaning
2️⃣ `gamma`	RBF kernel parameter	RBF / Polynomial kernels	Controls how far the influence of a single data point goes
3️⃣ `kernel`	Type of boundary	All	Decides the shape of the boundary (line, curve, etc.)

SVM Hyperparameters —

1️⃣ `C` — The Strictness Controller

Imagine: You're drawing a line to separate red and blue points.

- **If `C` is large (e.g. `C = 1000`) →**
  The SVM tries *very* hard to correctly classify every training point (even noise).
  👉 The boundary becomes **wiggly** and can **overfit**.
- **If `C` is small (e.g. `C = 0.1`) →**
  The SVM allows some mistakes to get a **smoother boundary**.
  👉 **Better generalization** (less overfitting).

In short:

C Value	Effect
High (strict)	Perfect training accuracy but may overfit 😬
Low (relaxed)	Allows errors but generalizes better 😎

2️⃣ `gamma` — The Influence Radius

(Used mostly with RBF or polynomial kernels.)

Imagine: Each data point creates a **"bubble of influence"** around it.

- **High `gamma` (γ large) →**
  Each point's bubble is **small** → model focuses on **nearby points** → **very complex** boundary.
- **Low `gamma` (γ small) →**
  Each point's bubble is **large** → model looks at **broader patterns** → **smoother** boundary.

In short:

gamma Value	Effect
High	Very detailed, may overfit

gamma Value	Effect
Low	Smoother, may underfit

3️⃣ `kernel` — The Shape of the Boundary

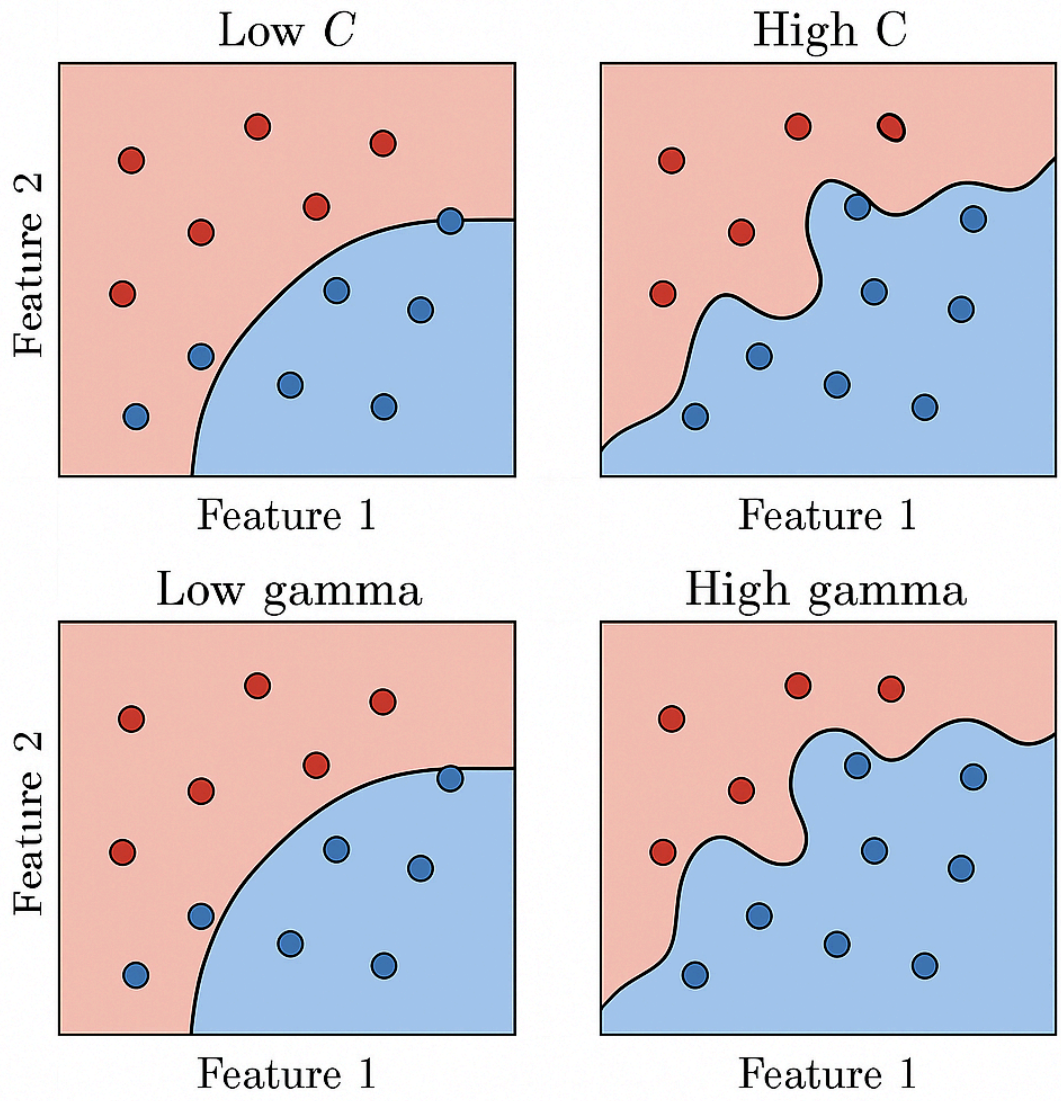
Kernel	Meaning	Shape of Boundary	Example Use
linear	Straight line	Linear	Simple, fast when data is linear
poly	Polynomial curve	Curved	When relationship is curved
rbf	Radial Basis Function	Circular / complex	Most common for non-linear data
sigmoid	S-shaped	Neural-net like	Rarely used

🧩 Putting It All Together — Example

```
from sklearn.svm import SVC
model = SVC(C=1, kernel='rbf', gamma=0.5)
```

In [25]:
```
#![c_gamma](./c_gamma.png)
from IPython.display import Image

# Display each image in the notebook
Image(filename='c_gamma.png')
```

Out[25]:



Cross Validation

💡 Definition

Cross-validation is a method used to check how well a machine learning model performs on **new or unseen data**.

It helps us find out if our model is:

- **Overfitting** (too focused on training data), or
- **Underfitting** (too simple to learn patterns).

🧩 How It Works — Step by Step (K-Fold Cross Validation)

Let's take **K = 5** folds.

Fold	Training Data	Testing Data
1	Parts 2,3,4,5	Part 1
2	Parts 1,3,4,5	Part 2
3	Parts 1,2,4,5	Part 3
4	Parts 1,2,3,5	Part 4
5	Parts 1,2,3,4	Part 5

So, the model is trained **5 times**, each time using a different test part.

🧠 After all 5 rounds → take the **average accuracy**
→ This gives a **more reliable performance score**.

🖼️ Simple Diagram Explanation

Imagine your dataset is split into 5 boxes (folds):

Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
✅	✅	✅	✅	🧪
✅	✅	✅	🧪	✅
✅	✅	🧪	✅	✅
✅	🧪	✅	✅	✅
🧪	✅	✅	✅	✅

✅ = used for **training**
🧪 = used for **testing**

Each fold gets one turn as the test set.

What is GridSearchCV?

Definition

GridSearchCV (Grid Search Cross-Validation) is a method used to **find the best hyperparameters** for a machine learning model automatically.

It tests **different combinations** of parameters (like `C`, `gamma`, `kernel` in SVM) and tells you which one gives the **best accuracy**.

How It Works

1. **You choose parameters to test** (a grid of values).
2. **GridSearchCV** trains and tests your model for every combination using **cross-validation**.
3. It returns:
    - Best parameters
    - Best model
    - Best accuracy score

RandomizedSearchCV for Random Forest

When we build a Random Forest model, it has many settings called hyperparameters, like:

- Number of trees (n_estimators)
- Maximum depth of each tree (max_depth)
- Number of features to consider when splitting nodes (max_features)
- And others...

Choosing the right hyperparameters is important to make the model accurate.

RandomizedSearchCV helps us find the best hyperparameters by:

- Randomly picking combinations of these settings.
- Testing each combination with parts of the training data (using cross-validation).
- Selecting the combination that gives the best model performance (like accuracy).

This way, instead of trying every possible combination (which can take a very long time), it tries a fixed number of random combinations, making the search faster but still effective.

Why use RandomizedSearchCV?

- It's faster than exhaustive search (GridSearchCV).
- It gives good hyperparameter tuning results.
- Helps avoid overfitting by using cross-validation.

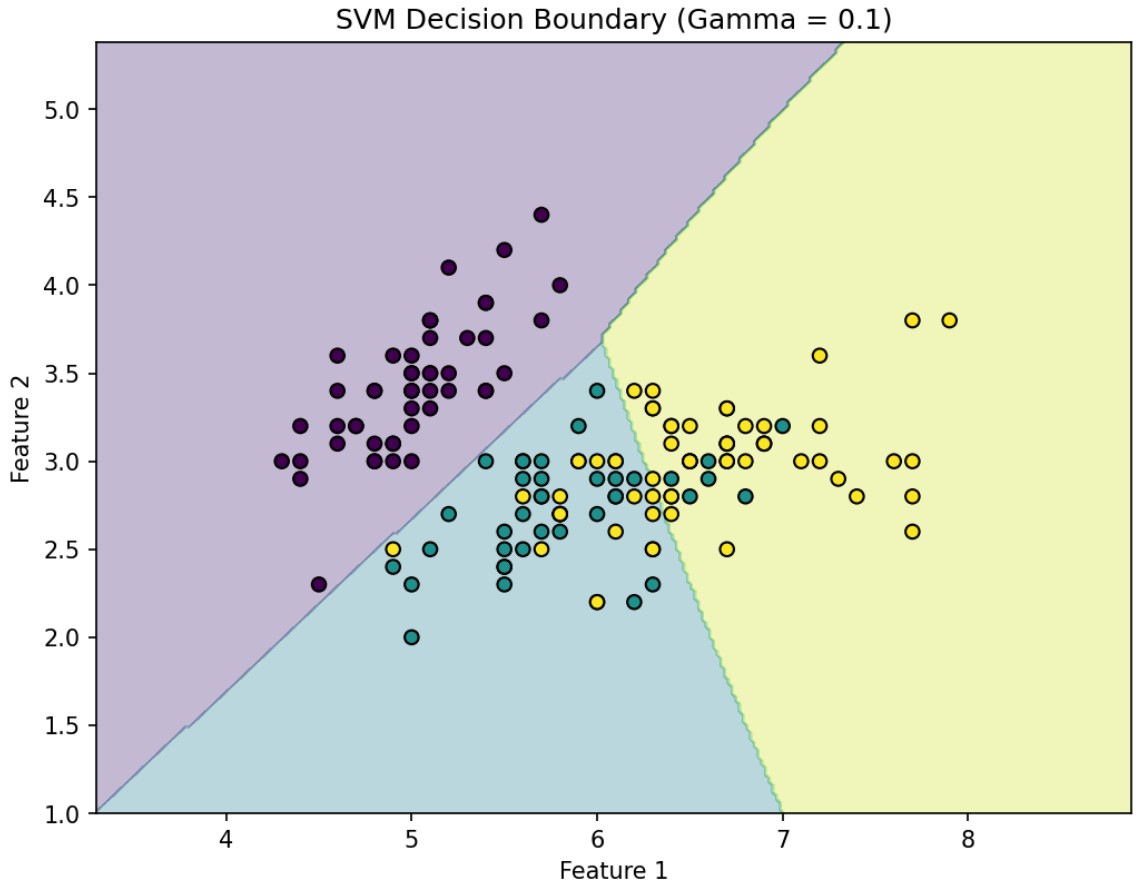
Basic Usage Steps:

1. Define a range for each hyperparameter.
2. Let RandomizedSearchCV randomly sample and test combinations.
3. Choose the best hyperparameters based on the evaluation.
4. Build the final model using these best settings.

It is a practical and smart way to improve Random Forest model performance with less time.

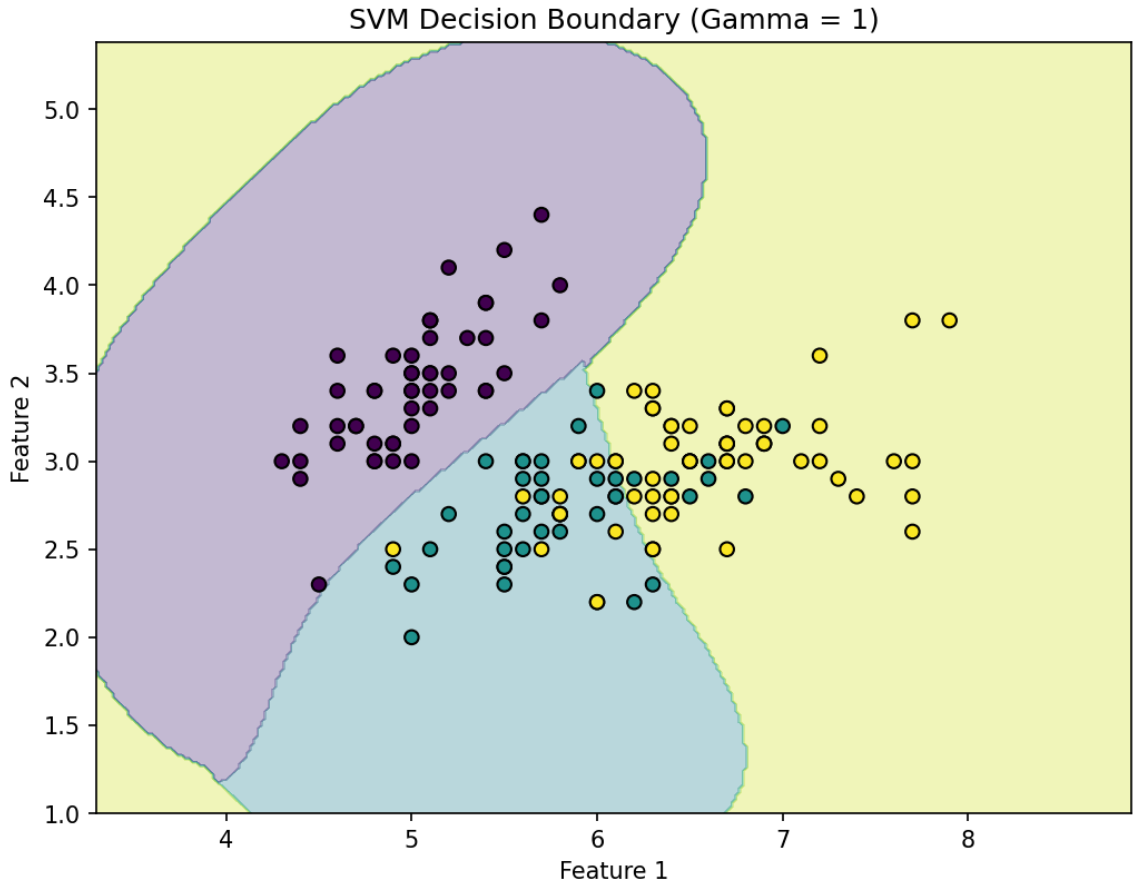
In [26]:
```
from IPython.display import Image

# Display each image in the notebook
Image(filename='svm_gamma_0.1.png')
```

Out[26]:



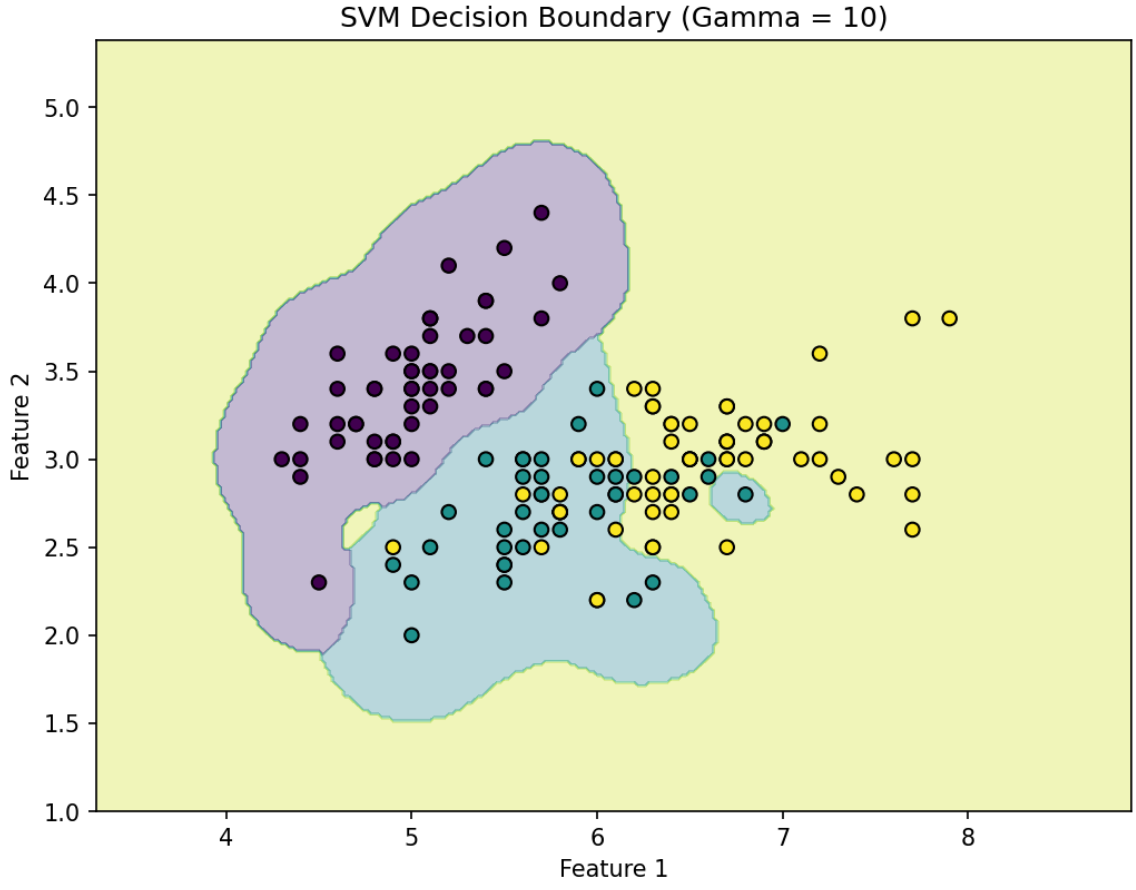
In [27]:
```
Image(filename='svm_gamma_1.png')
```

Out[27]:



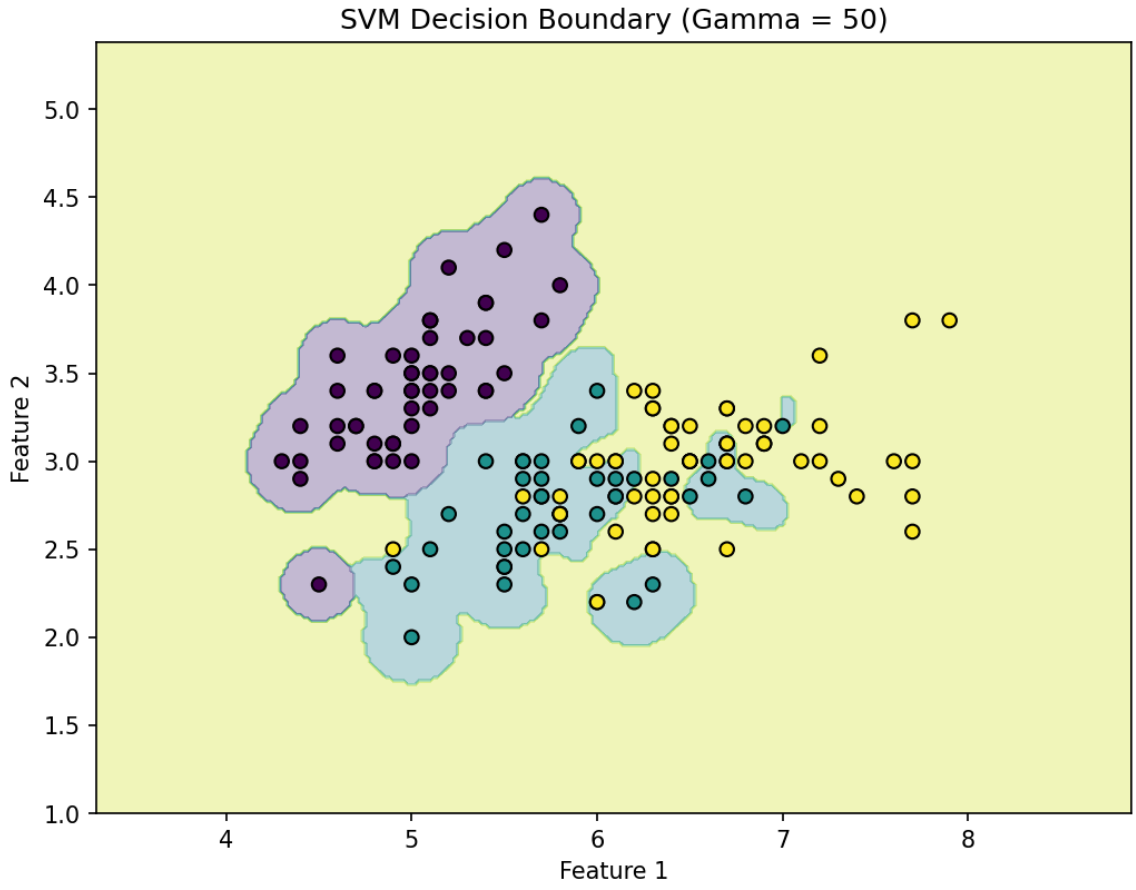
In [28]:
```
Image(filename='svm_gamma_10.png')
```

Out[28]:



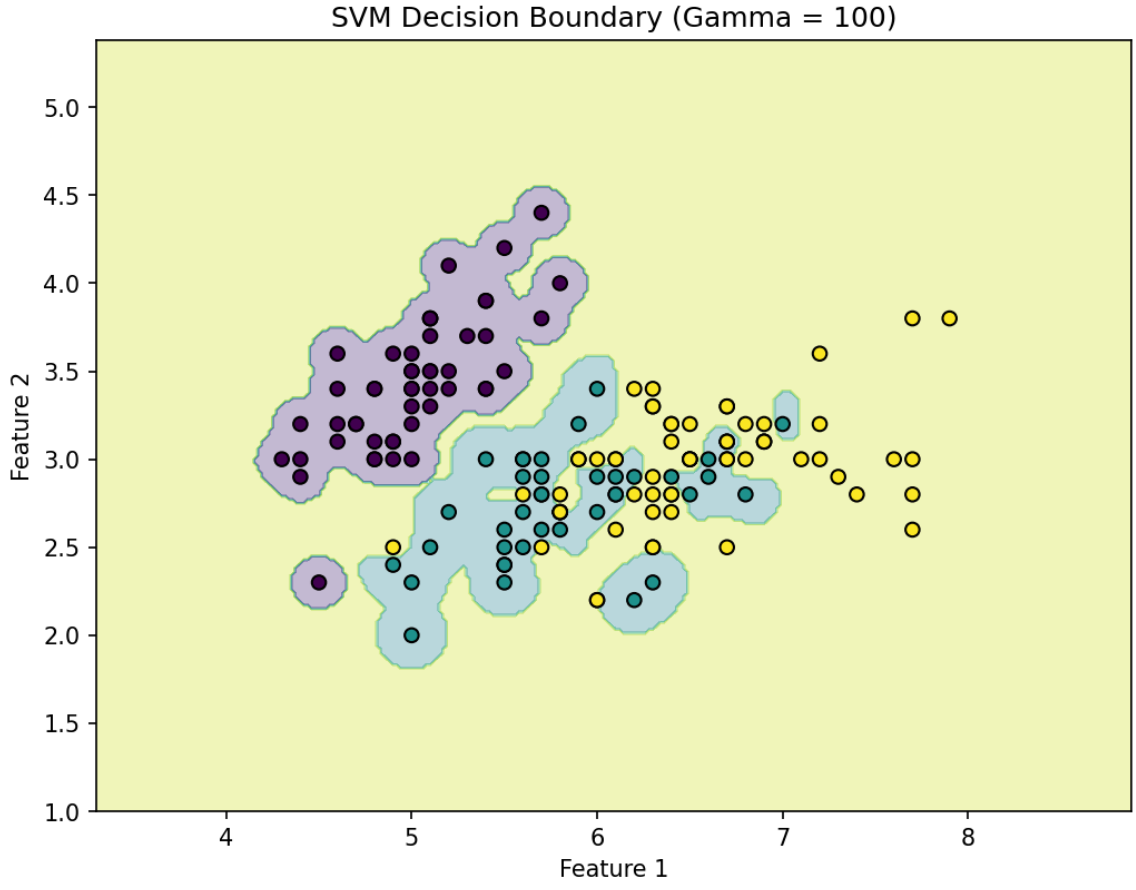
In [29]:
```
Image(filename='svm_gamma_50.png')
```

Out[29]:



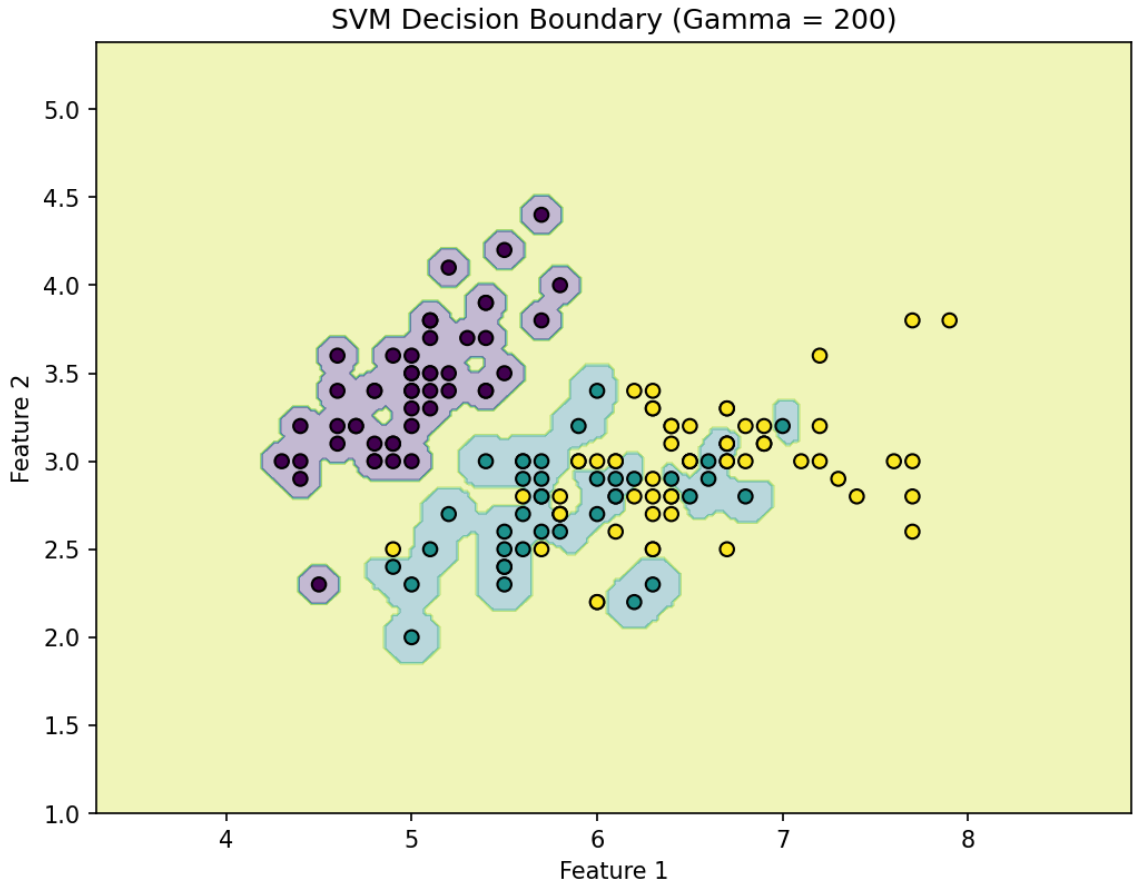
In [30]:
```
Image(filename='svm_gamma_100.png')
```

Out[30]:



In [31]:
```
Image(filename='svm_gamma_200.png')
```

Out[31]:

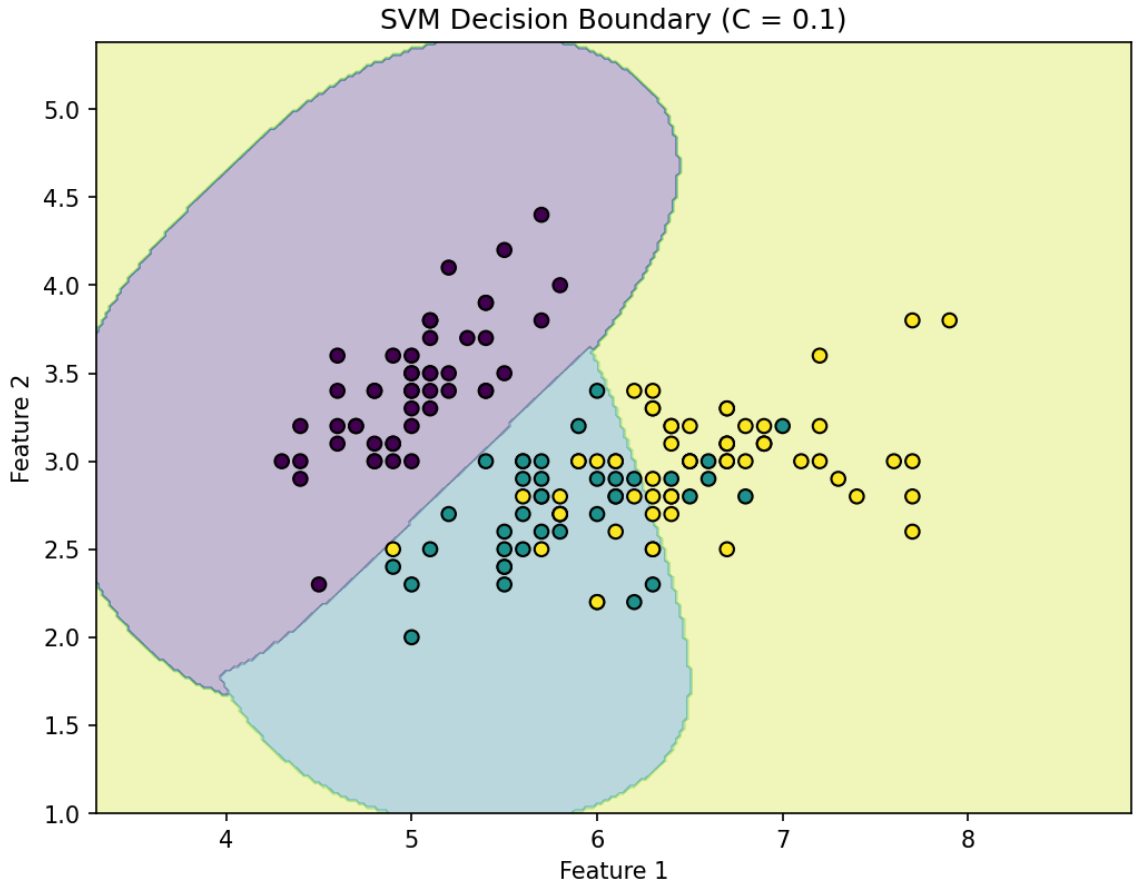


C parameter

In [32]:

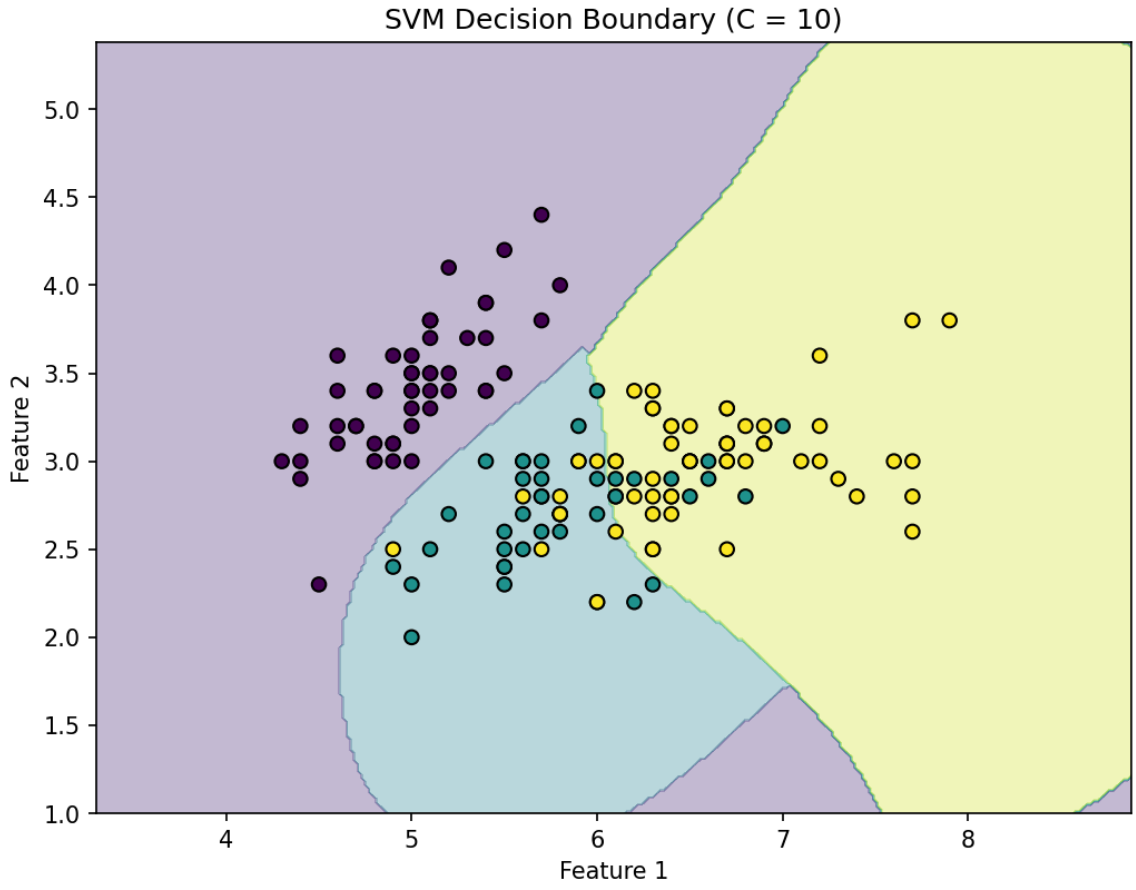
```
from IPython.display import Image
Image(filename='svm_C_0.1.png')
```

Out[32]:



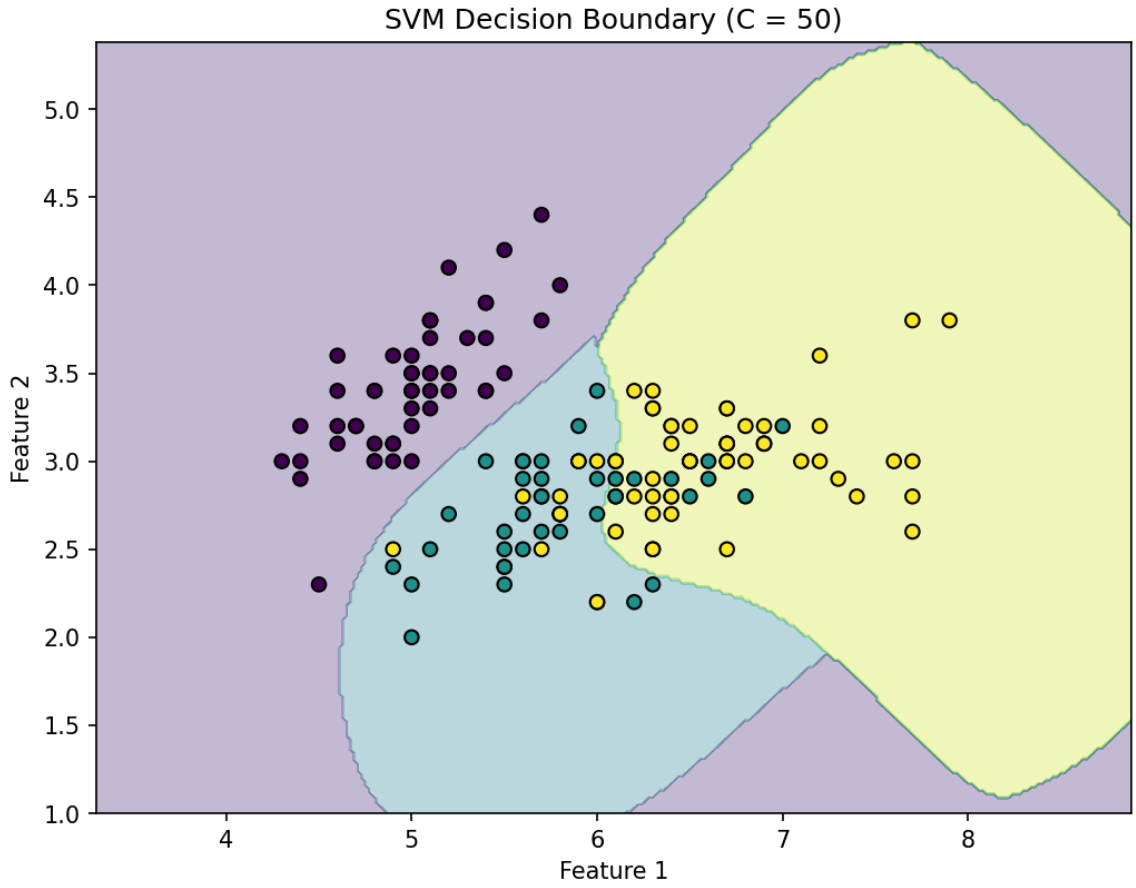
In [33]:
```
from IPython.display import Image
Image(filename='svm_C_10.png')
```

Out[33]:



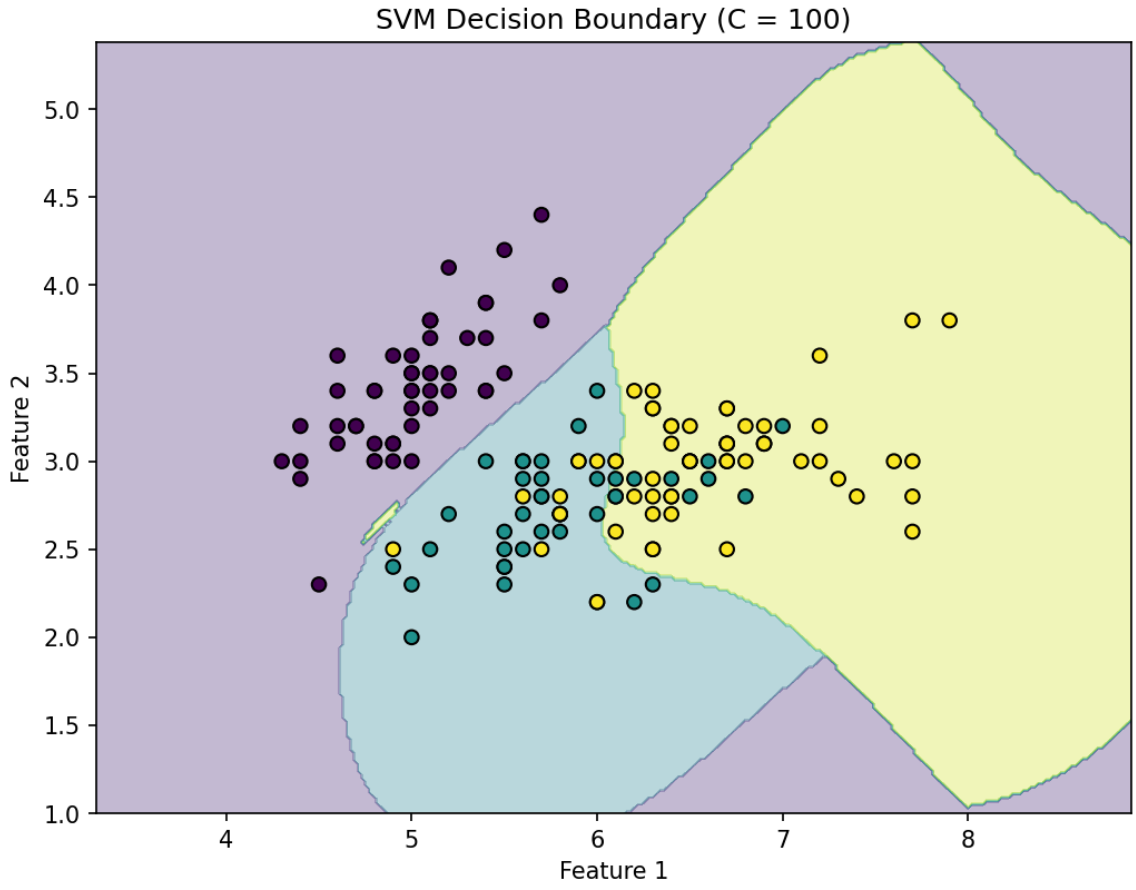
In [34]:
```
from IPython.display import Image
Image(filename='svm_C_50.png')
```

Out[34]:



In [35]:
```
from IPython.display import Image
Image(filename='svm_C_100.png')
```

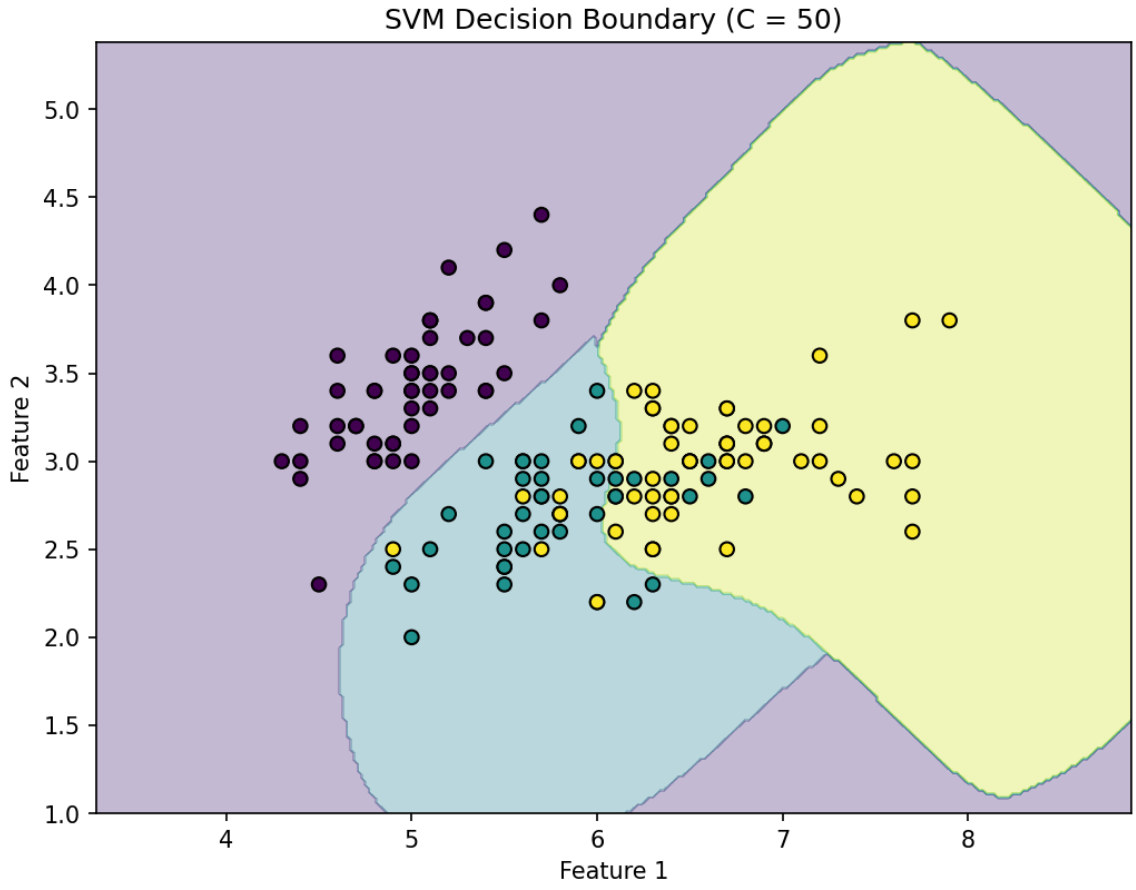
Out[35]:



In [36]:

```
from IPython.display import Image
Image(filename='svm_C_50.png')
```

Out[36]:



In [ ]: