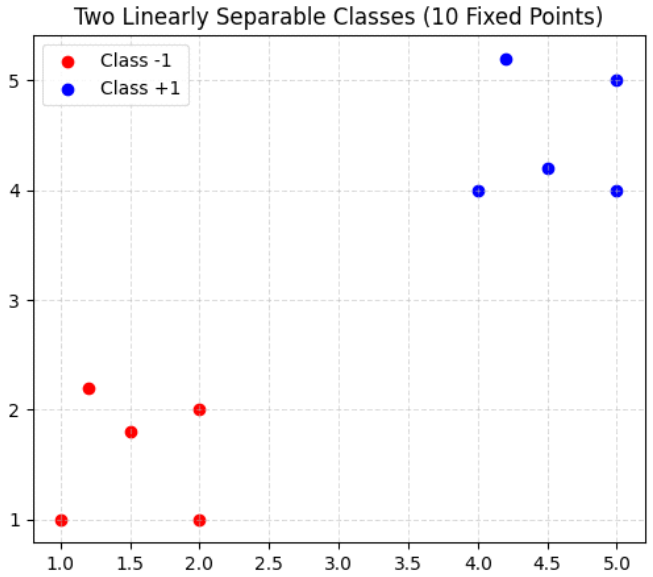
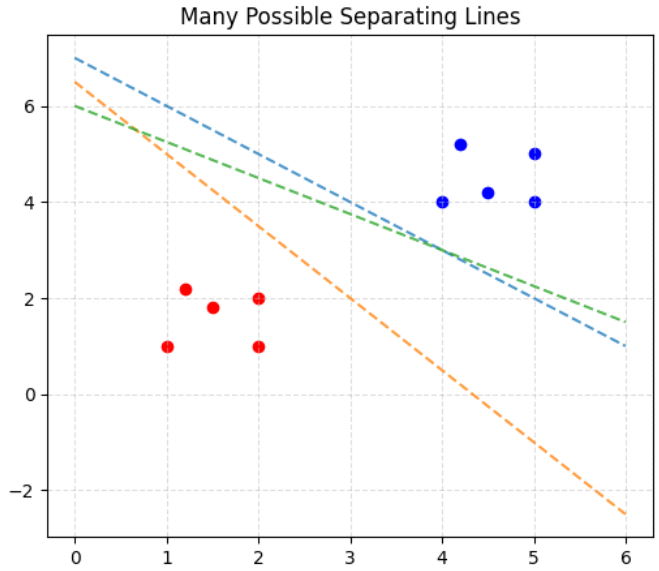
🧠 1. What is a Support Vector Machine (SVM)?

Support Vector Machine (SVM) is a **supervised machine learning algorithm** used for **classification** (and sometimes regression).
It aims to find the **best decision boundary** (called a **hyperplane**) that separates different classes with the **maximum margin**.

✨ 2. Intuitive Idea

Let's say we have two classes of data (red and blue points). There can be many possible lines (in 2D) that separate them, but SVM finds the **one with the largest distance (margin)** between both classes.

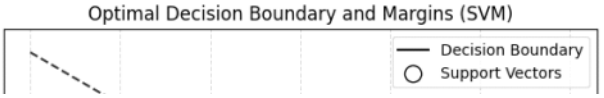


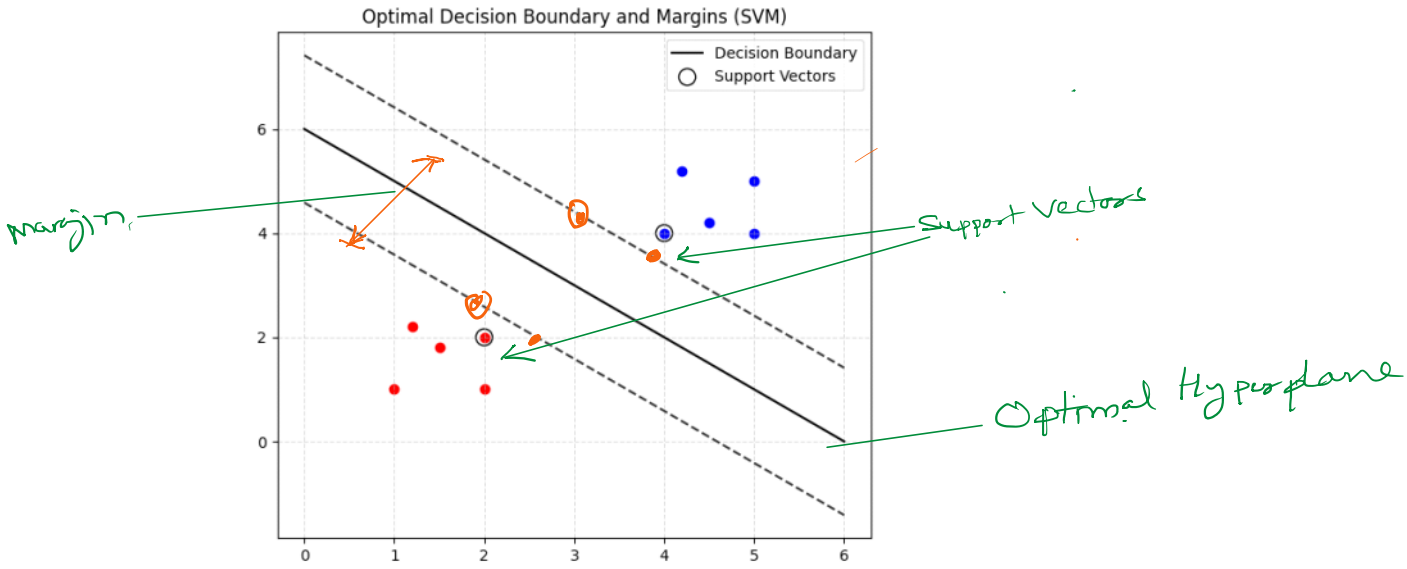


0.5 1 ?, 0.5

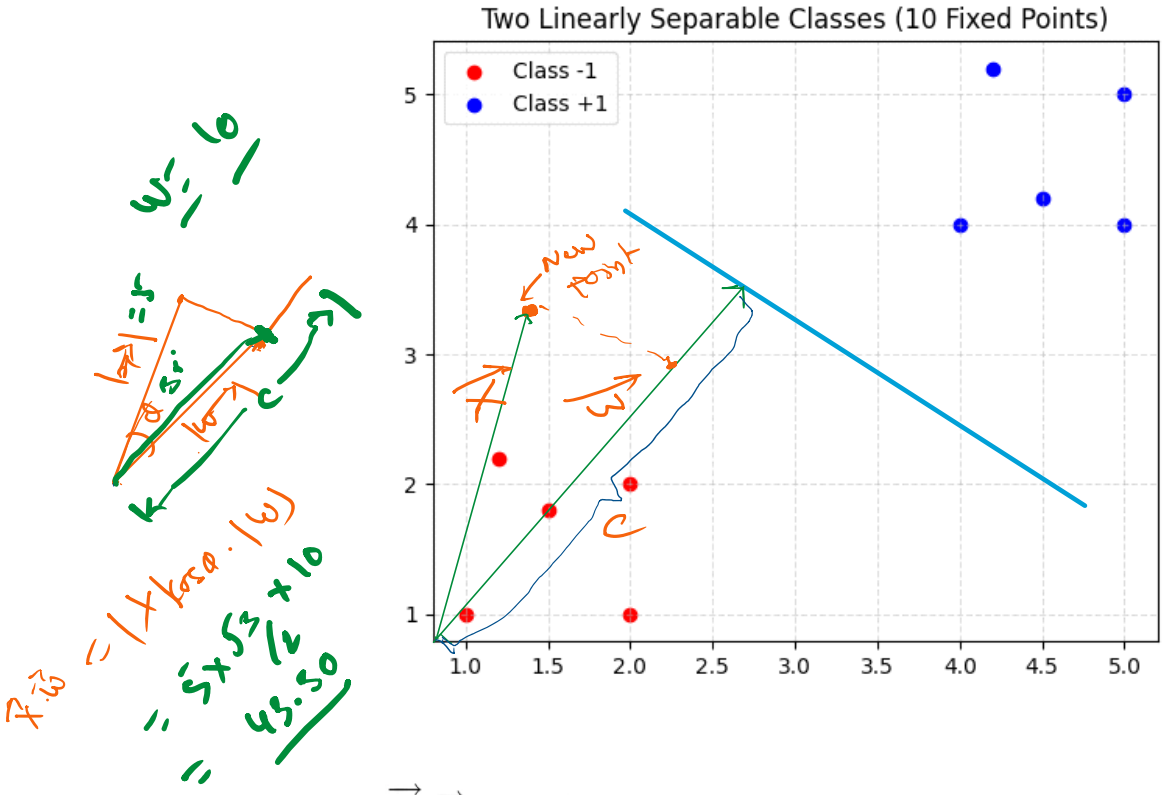
👉 Key Idea:

Maximize the margin between the classes while minimizing classification errors.





Decision rule:

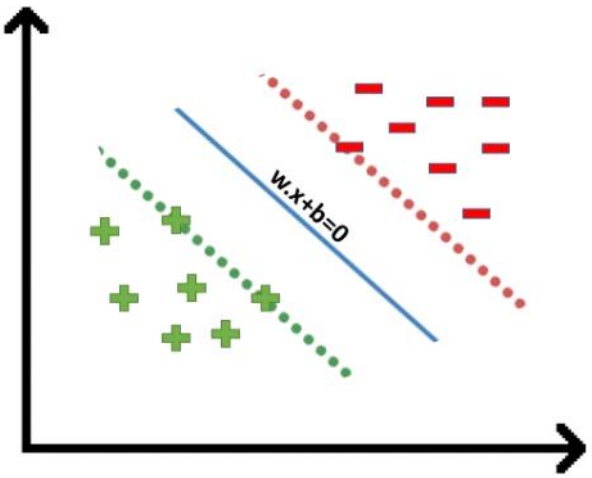


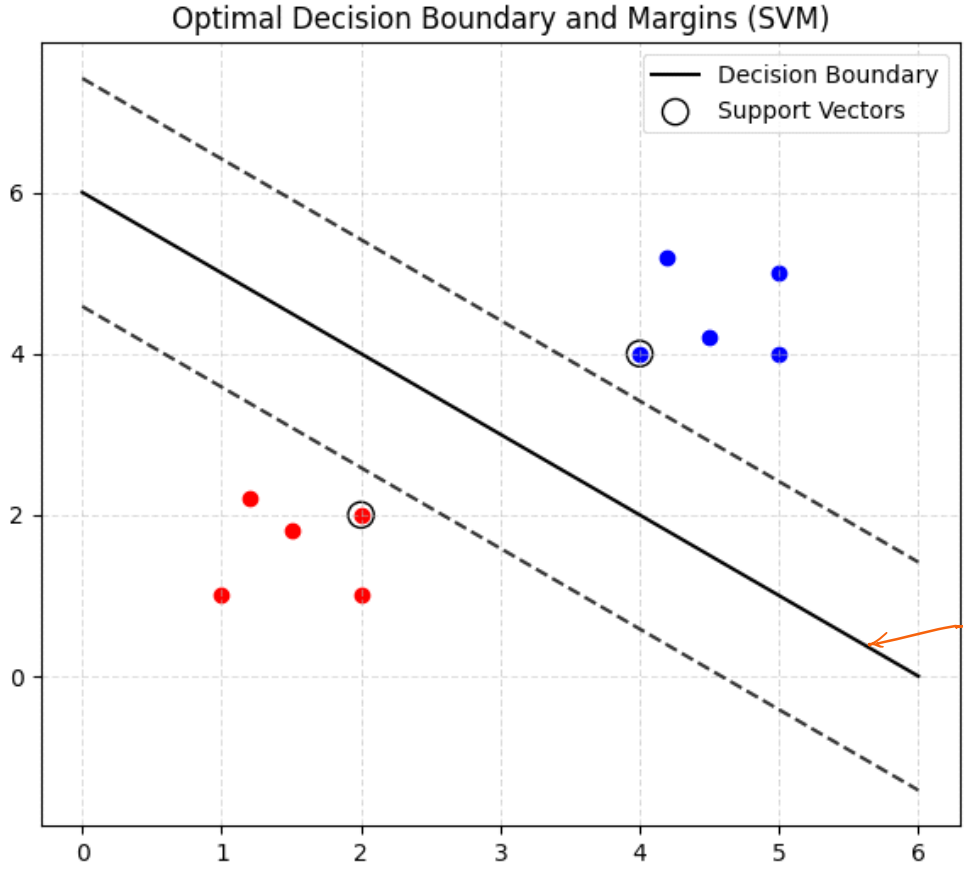
$$\vec{X}.\vec{w} = c \quad (the\ point\ lies\ on\ the\ decision\ boundary)$$

$$\vec{X}.\vec{w} > c \ (positive\ samples)$$

$$\vec{X}.\vec{w} < c \ (negative\ samples)$$

Margin in Support Vector Machine:


w.x+b=0


Optimal Decision Boundary and Margins (SVM)
Decision Boundary
Support Vectors
0
2
4
6
0
1
2
3
4
5
6

$wx+b=0$

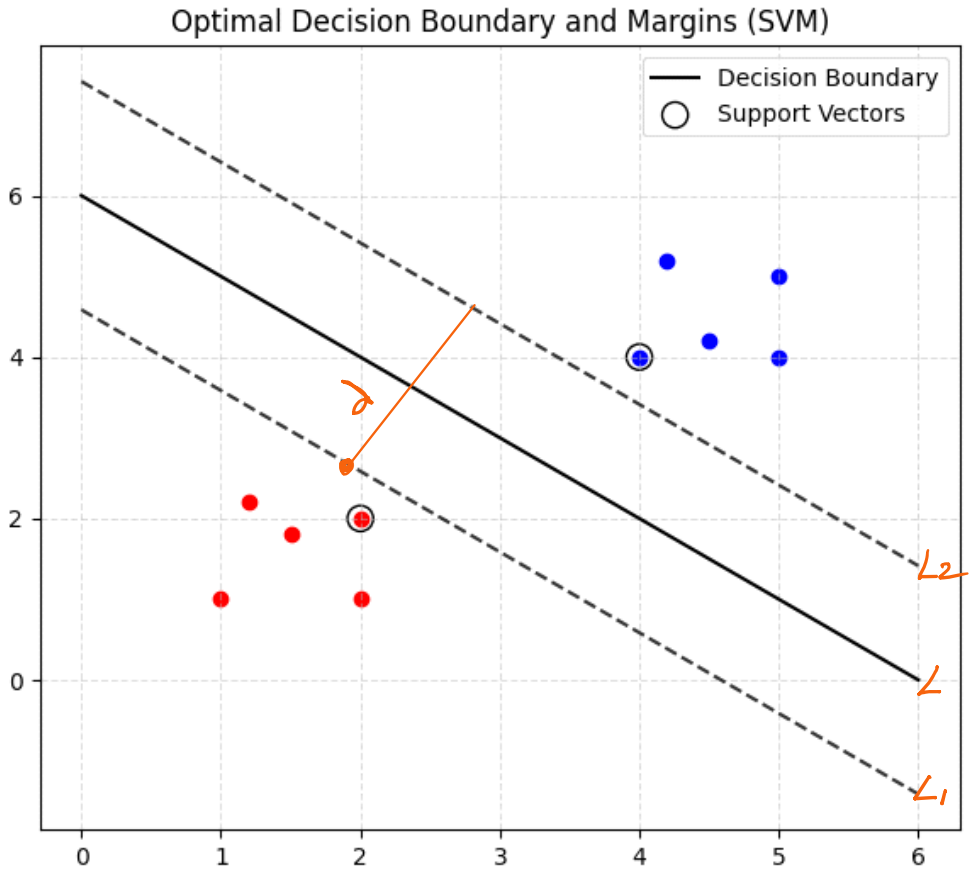
$$\overrightarrow{X}.\overrightarrow{w} - c \geq 0$$

putting $-c$ as b, we get

$$\overrightarrow{X}.\overrightarrow{w} + b \geq 0$$

hence

$$y = \begin{cases} +1 & \text{if } \overrightarrow{X}.\overrightarrow{w} + b \geq 0 \\ -1 & \text{if } \overrightarrow{X}.\overrightarrow{w} + b < 0 \end{cases}$$



Let us assume that the equation of L1 is **w.x+b=1** and for L2 it is **w.x+b=-1**.

Optimization Function and its Constraints:

Constraints:

$$For\ all\ the\ Red\ points\ \overrightarrow{w}.\overrightarrow{X} + b \leq -1$$

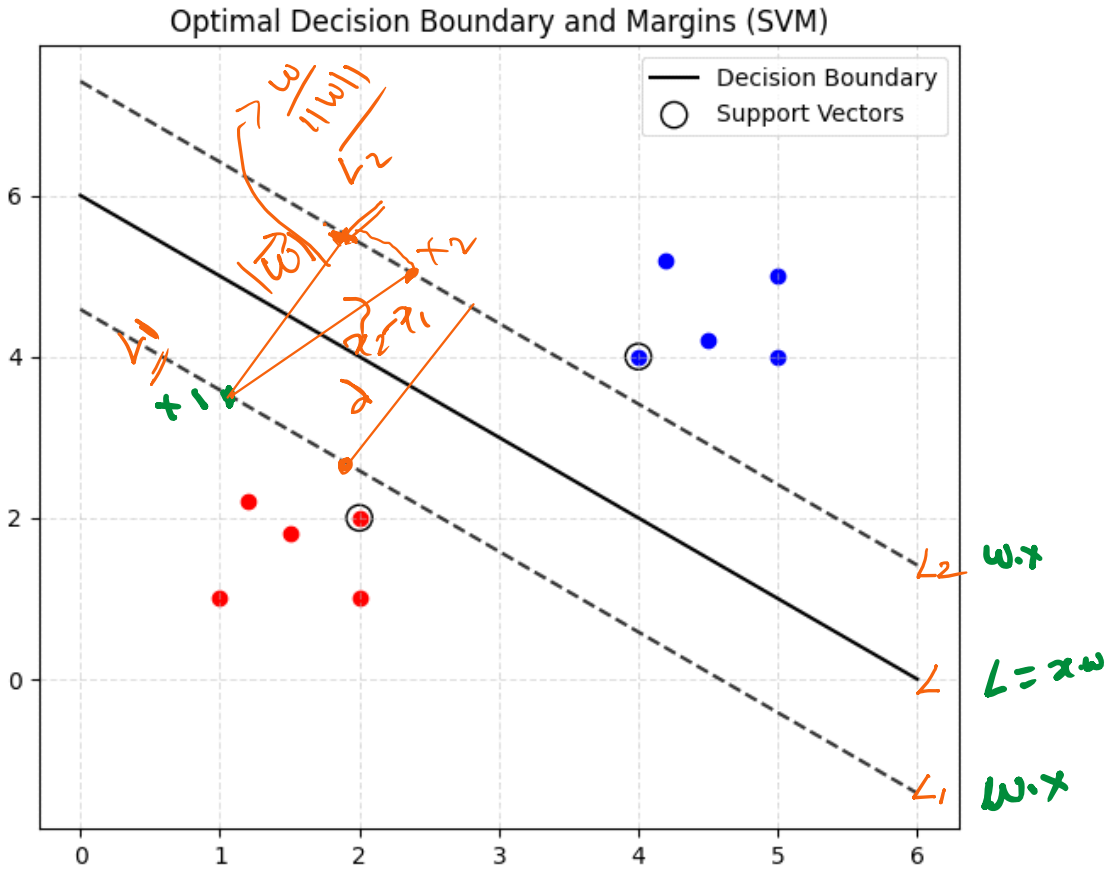
$$For\ all\ the\ Green\ points\ \overrightarrow{w}.\overrightarrow{X} + b \geq 1$$

$$For\ all\ the\ Red\ points\ \vec{w}.X + b \leq -1$$

$$For\ all\ the\ Green\ points\ \vec{w}.\vec{X} + b \geq 1$$

- Rather than taking 2 constraints forward, we'll now try to simplify these two constraints into 1.
- We assume that negative classes have ***y=-1*** and positive classes have ***y=1.***
- We can say that for every point to be correctly classified this condition should always be true:

$$y_i(\vec{w}.\vec{X} + b) \geq 1$$



$$d = (\vec{x}_2 - \vec{x}_1) \cdot \frac{\vec{w}}{\|w\|}$$

$$d = \frac{\vec{w}\cdot\vec{x}_2 - \vec{w}\vec{x}_1}{\|w\|} \quad \longleftarrow (1)$$

$$y(\vec{w}\cdot\vec{x} + b) \geq 1$$

+ve $(L_2$ Line)

$y = 1$

$$1(\vec{w}\cdot\vec{x}_2 + b) \geq 1$$

$$w\cdot\vec{x}_2 + b \geq 1 \Rightarrow \quad w\cdot x_2 + b - 1 = 0$$

$$w\cdot x_2 = 1 - b$$

-ve $\quad y = -1$

$$w\cdot x_1 + b \leq -1$$

$$w\cdot x_1 = -1 - b$$

put value into eq (1)

put value into eq (1)

$$d = \frac{wx_2 - wx_1}{\|w\|}$$

$$= [1 - b + (-1 - b)] / \|w\|$$

$$= (1 - b + 1 + b) / \|w\|$$

$$\boxed{d = \frac{2}{\|w\|}}$$

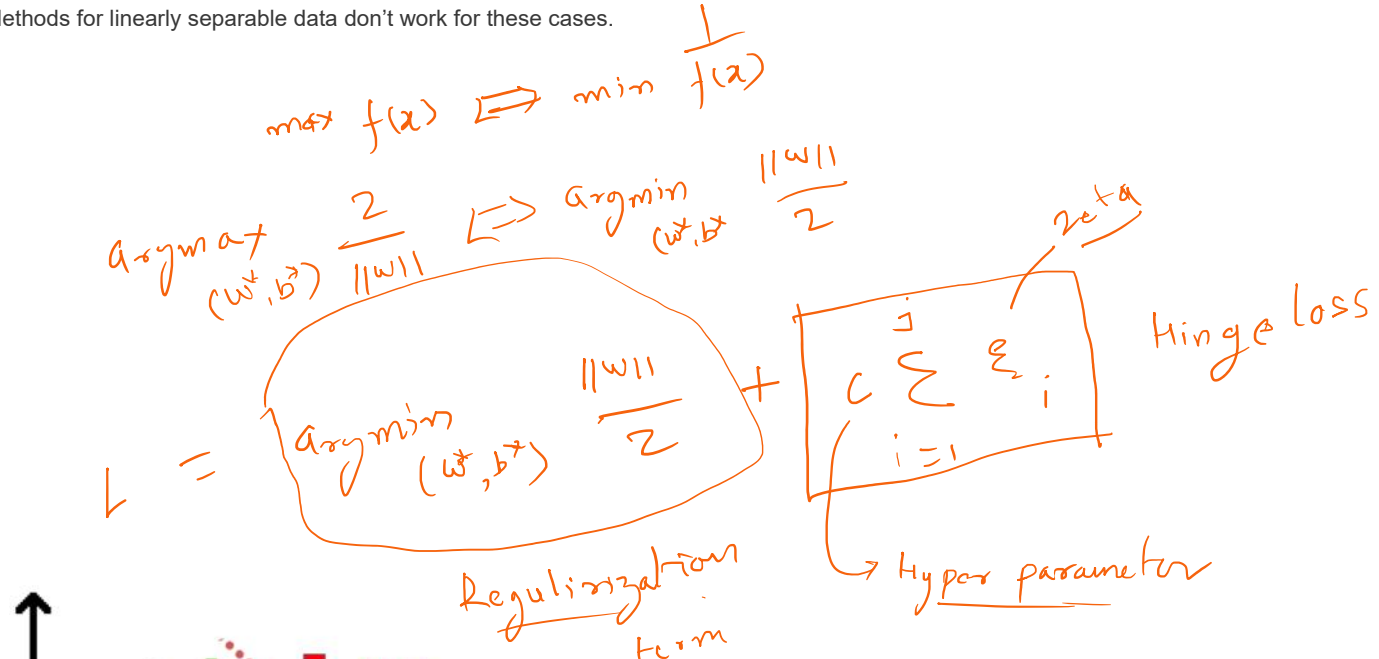
Objective function

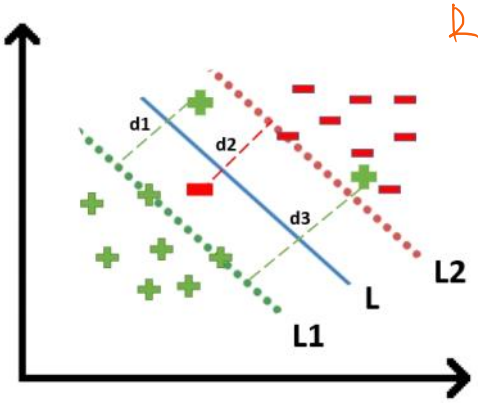
$$\underset{(w,b)}{\text{argmax}} \frac{2}{\|w\|}$$

Soft Margin SVM

13 August 2025 17:59

- Real-world datasets are rarely perfectly linearly separable; they are often nearly or completely non-linearly separable.
- Methods for linearly separable data don't work for these cases.





ξi will be zero for all correctly classified data points.

SUM error = margin error + classification error

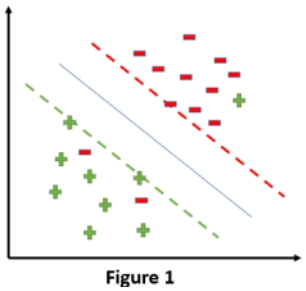


Figure 1

Train → Acc = 90 %

Test = 90 %

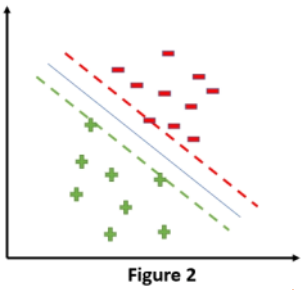


Figure 2

Acc = 100 %

Test = 85 %

Test - 90 +,

SVM Kernals

13 August 2025 18:00

Why Do We Need Kernels in SVM?

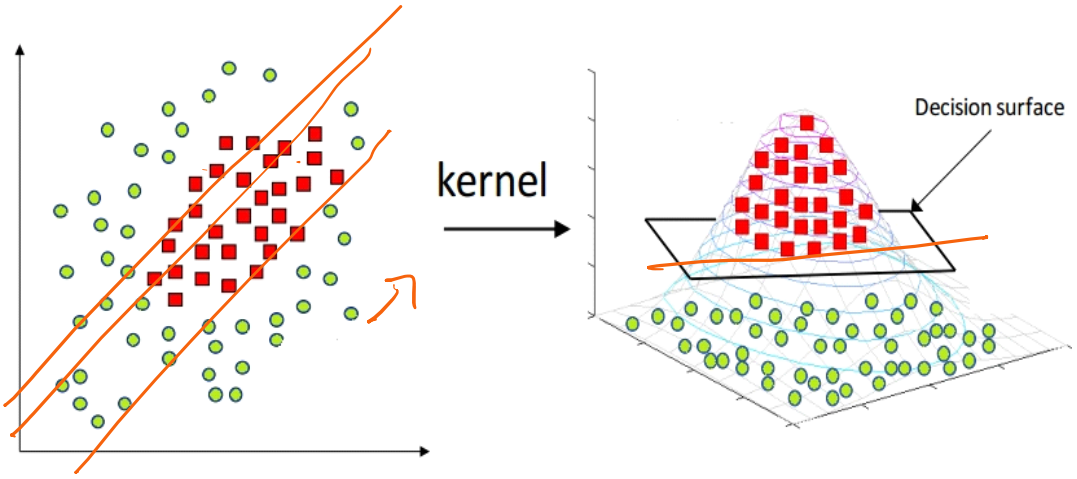
Support Vector Machines (SVMs) are designed to find a **hyperplane** that separates classes with maximum margin.

In the simplest case:

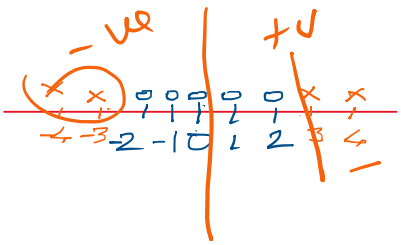
- If the data is **linearly separable**, we can find a straight line (2D), plane (3D), or hyperplane (nD).
- But in reality, data is often **non-linear** — no straight line can separate classes well.

Idea:

Map the data to a **higher-dimensional space** where it becomes linearly separable, then find the hyperplane there.

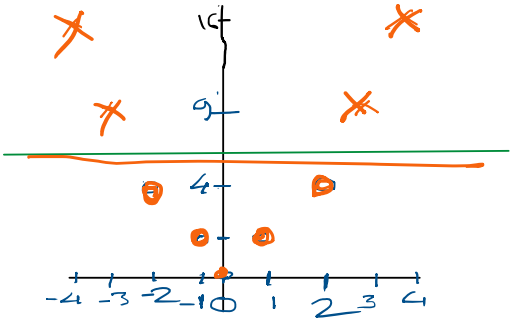


Ex 1D → 2D



Kernal transformations

$Y = x^2$



The Kernel Trick

- Instead of **explicitly computing** the transformation into higher dimensions (which may be computationally expensive or infinite-dimensional),
  we use a **kernel function** $K(x_i, x_j)$ that directly computes the **dot product** in that higher-dimensional space.

Mathematically:

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$$

where:

- $\phi(\cdot)$ = mapping function from original space → higher-dimensional feature space.
- $K(\cdot, \cdot)$ = kernel function.

This is why it's called the **kernel trick** — we avoid explicitly mapping ϕ, but still reap the benefits.

2. Main SVM Kernel Types

Let's go one by one.

(A) Linear Kernel

$$K(x_i, x_j) = x_i \cdot x_j$$

- No transformation — works directly in the original space.
- Decision boundary is a **straight hyperplane**.

✅ **When to use:**

- Data is linearly separable or high-dimensional (e.g., text classification).
- Very fast training and prediction.

❌ **Not good for:**

- Complex, non-linear boundaries.

📌 **Example:** Spam vs. Ham email classification.

(B) Polynomial Kernel

$$K(x_i, x_j) = (x_i \cdot x_j + c)^d$$

where:

- d = degree of the polynomial.
- c = constant term controlling influence of higher-order terms.

Intuition:

Allows curved decision boundaries by adding **interaction terms** (e.g., $x_1 x_2, x_1^2$) without explicitly computing them.

✅ **When to use:**

- Data where interactions between features matter.
- Moderate-sized datasets.

❌ **Not good for:**

- Very high-degree polynomials → overfitting.
- Very large datasets → slow.

📌 **Example:** Handwritten digit classification (when RBF is not desired).

(C) Radial Basis Function (RBF) / Gaussian Kernel

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

or equivalently:

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$$

where:

- $\gamma = \frac{1}{2\sigma^2}$ controls influence range.

Intuition:

- Measures similarity — points close in input space have high kernel value, far points have near-zero similarity.
- Maps data into **infinite-dimensional space**.

✅ **When to use:**

- Non-linear data with complex boundaries.
- Most general-purpose kernel.

❌ **Not good for:**

- Very large datasets (training slow).
- Sensitive to γ and C parameter tuning.

📌 **Example:** Face recognition, medical diagnosis classification.

(D) Sigmoid Kernel (MLP Kernel)

$$K(x_i, x_j) = \tanh(\kappa(x_i \cdot x_j) + \theta)$$

where:

- κ = slope parameter.
- θ = intercept.

Intuition:

- Inspired by **neural network activation functions**.
- Can behave like RBF for certain parameters.

✅ **When to use:**

- Sometimes works well in text and neural net–inspired setups.

❌ **Not widely used** because:

- Not always positive semi-definite → may not satisfy Mercer's theorem (theoretical guarantee for SVM convergence).
- Sensitive to parameters.

3. Comparison Table

Kernel	Non-linearity	Parameters to Tune	Dimensionality	Pros	Cons
Linear	No	C	Original space	Fast, simple	Can't handle non-linear patterns
Polynomial	Yes	C, degree, coef0	Higher	Captures feature interactions	Overfitting for high degree
RBF	Yes	C, γ	Infinite	Flexible, powerful	Slower, parameter-sensitive
Sigmoid	Yes	C, κ, θ	Higher	NN-like mapping	Less stable

4. Practical Notes

- **Scaling** is critical — SVM kernels assume features are normalized.
- **Parameter tuning**:
  - Use **Grid Search + Cross-Validation** for `C`, `gamma`, `degree`.
  - Small γ → smoother boundaries, large γ → more complex boundaries.
- **Kernel choice**:
  - Try **Linear** first for high-dimensional, sparse data.
  - Try **RBF** for most other cases.
  - Polynomial only if you have good reason (feature interactions matter).

https://github.com/MAHESHGOUR/MLDL-2025

Support Vector Regression (SVR)

19 August 2025 18:59

Tutorial: Support Vector Regression (SVR)

1. Introduction

Support Vector Machines (SVMs) are widely used for **classification**, but they can also be adapted for **regression problems**, called **Support Vector Regression (SVR)**.
Unlike traditional regression, which tries to minimize the error between predicted and actual values, SVR tries to fit the data **within a certain margin of tolerance** (called **ε-insensitive tube**).

2. Intuition Behind SVR

- Imagine you have a function (line, curve, hyperplane) that approximates your data.
- Instead of minimizing **MSE (Mean Squared Error)** like in linear regression, SVR tries to ensure:
  - Predictions within **ε distance** from the true value are considered "correct".
  - Only data points **outside ε** margin are penalized (called support vectors).
- The model tries to be **as flat (simple)** as possible while still capturing important trends.

3. Mathematical Formulation

SVR tries to solve the following optimization problem:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

subject to:

$$y_i - (w \cdot x_i + b) \leq \epsilon + \xi_i$$

$$(w \cdot x_i + b) - y_i \leq \epsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \geq 0$$

Where:

- w, b: weights and bias of the regression function
- ϵ: margin of tolerance (ε-insensitive zone)
- C: regularization parameter (trade-off between flatness and tolerance violations)
- ξ_i, ξ_i^*: slack variables for points outside ε

4. Important Hyperparameters

- **C**: Regularization parameter
  - Large C → tries to minimize violations (fit training data closely, risk overfitting)
  - Small C → more tolerance for errors (better generalization)
- **ε (epsilon)**: Tube size
  - Larger ε → ignores small errors, simpler model
  - Smaller ε → stricter fitting
- **Kernel**: Decides shape of regression function
  - `"linear"`: Straight line (good for linear data)
  - `"poly"`: Polynomial
  - `"rbf"`: Non-linear, flexible (default and most used)