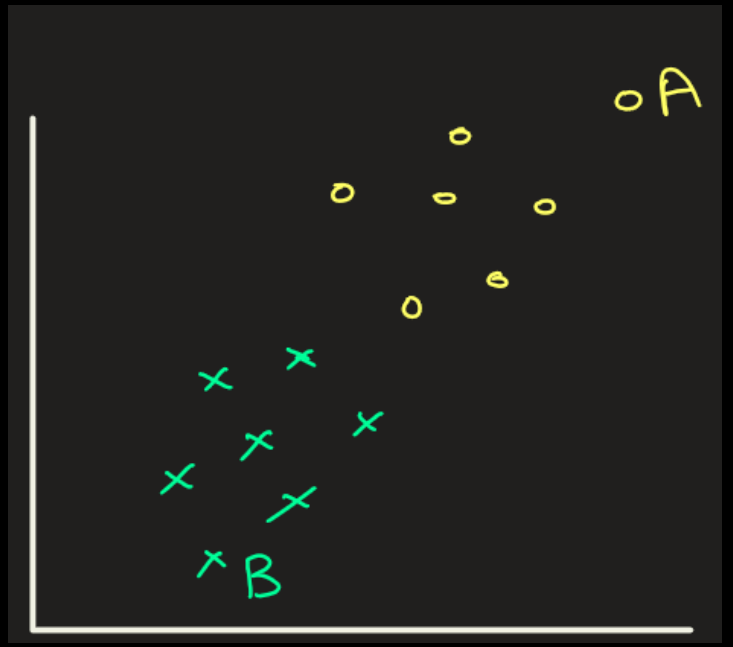
Support Vector Machine

Support Vector Machine (SVM) is a **supervised machine learning algorithm** used for:



- **Classification:** (most common). Also called SVC- Support Vector Classifier. SVM are used for both **linear or nonlinear classification**

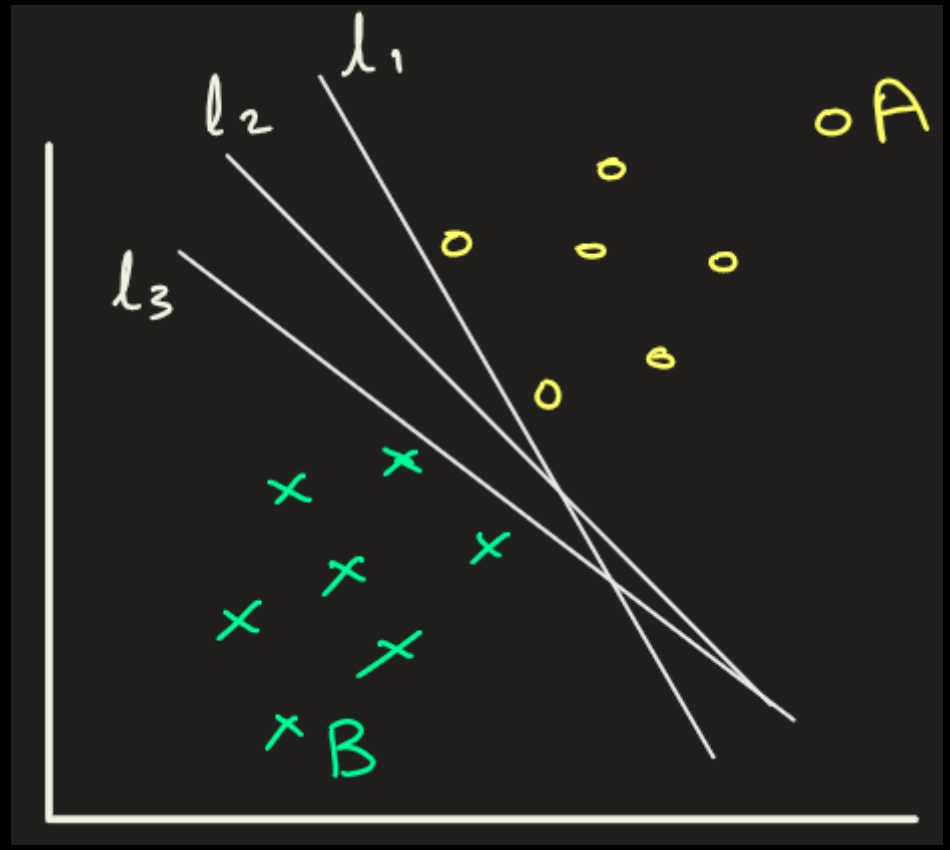
- **Regression:** Also called SVR – Support Vector Regression

Support Vector Machine

Imagine you are separating:

- Spam vs Not Spam emails
- Cancer vs No Cancer patients
- Loan Approved vs Rejected

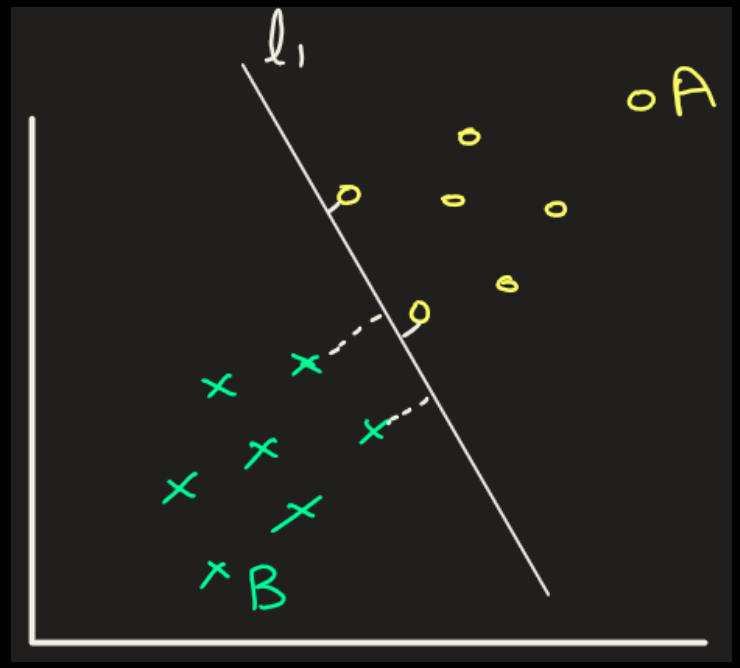
There are many possible lines that can separate these classes.

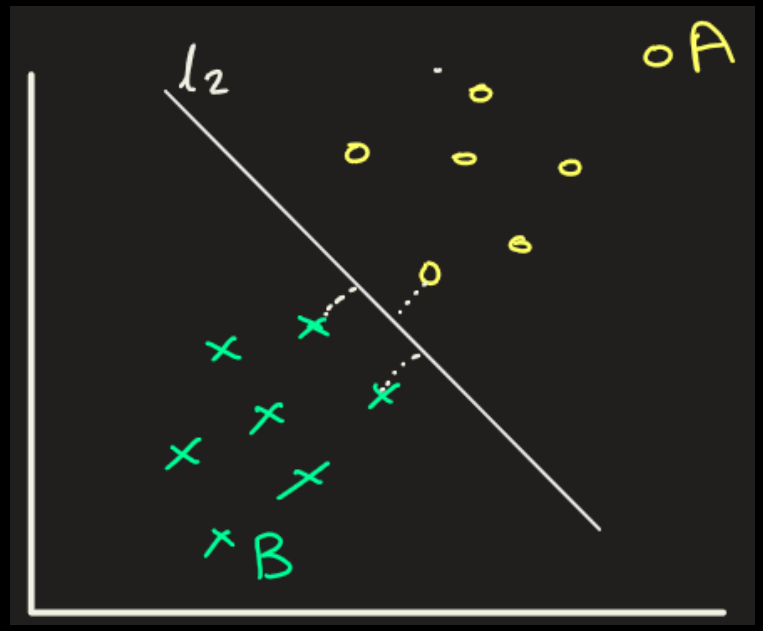


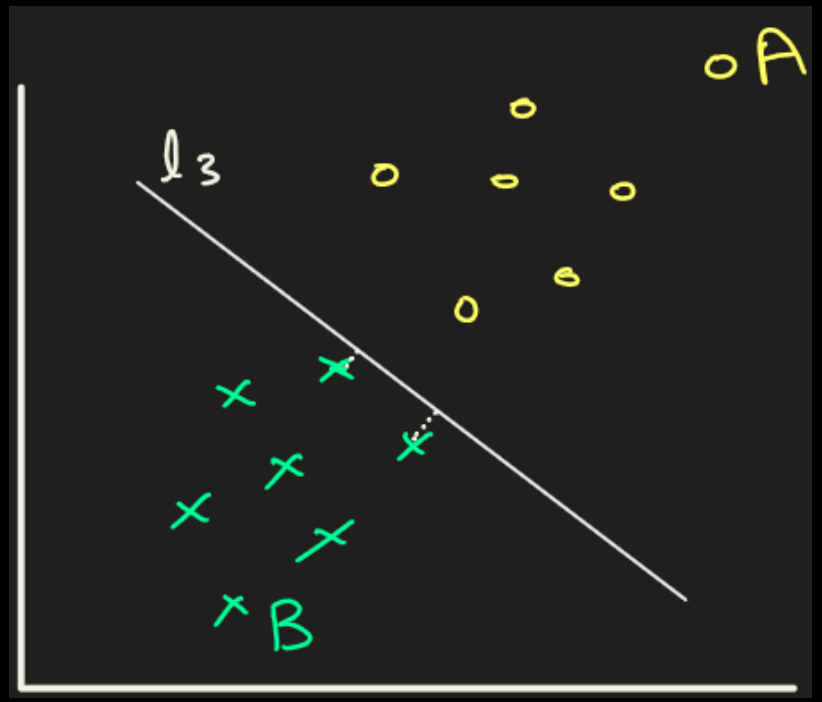
Support Vector Machine

SVM chooses the line that is farthest from both classes → This gives better generalization on new data.

Below line2 is farthest from class A and class B. This is a good line.

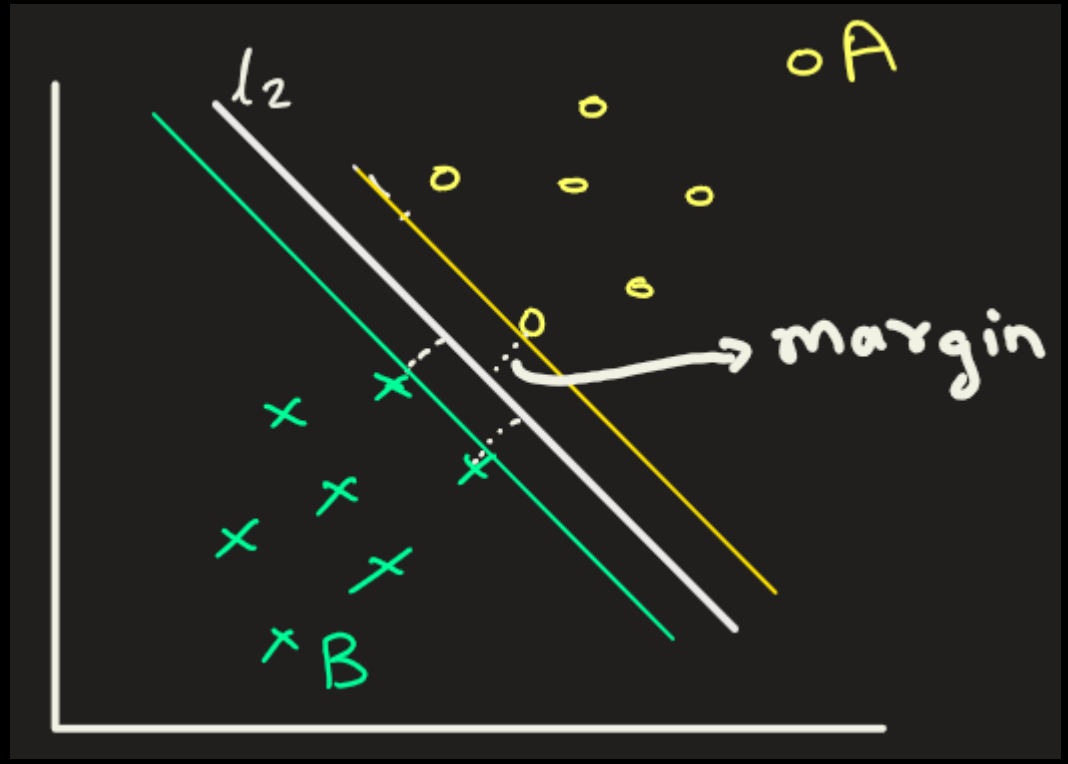






Support Vector Machine

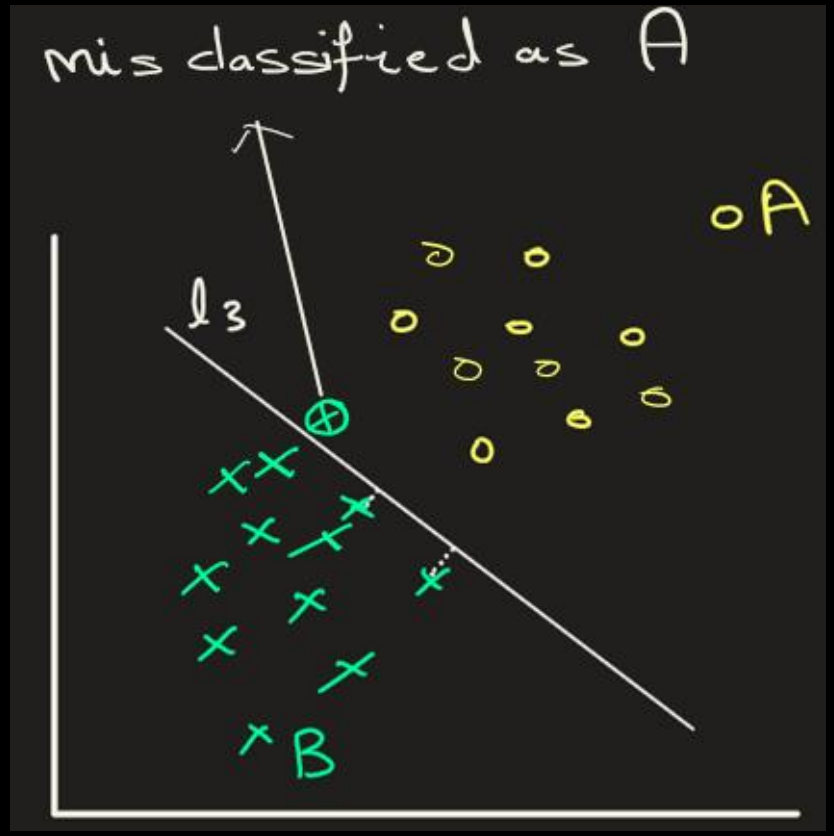
This line 2 is halfway between the 2 classes.



Support Vector Machine

Why does SVM chooses line that is farthest from both classes ?

Answer: Achieve better generalization to new, unseen data

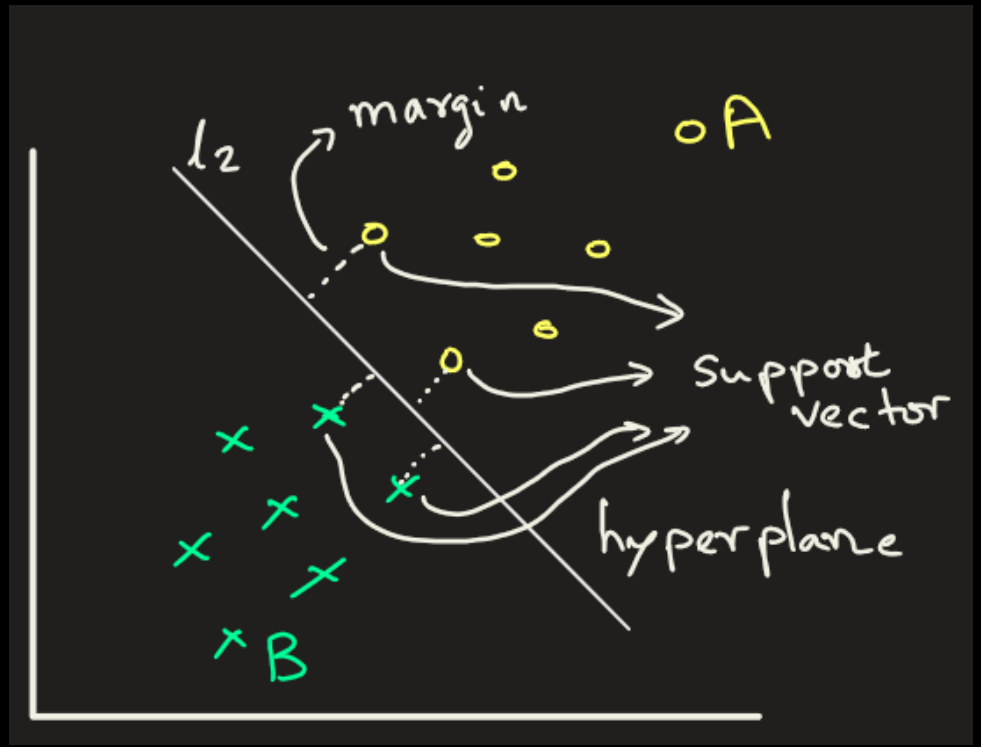


Support Vector Machine

The line/plane is called the hyperplane

The closest data points to the hyperplane are called support vectors

The distance between the boundary and support vectors = margin



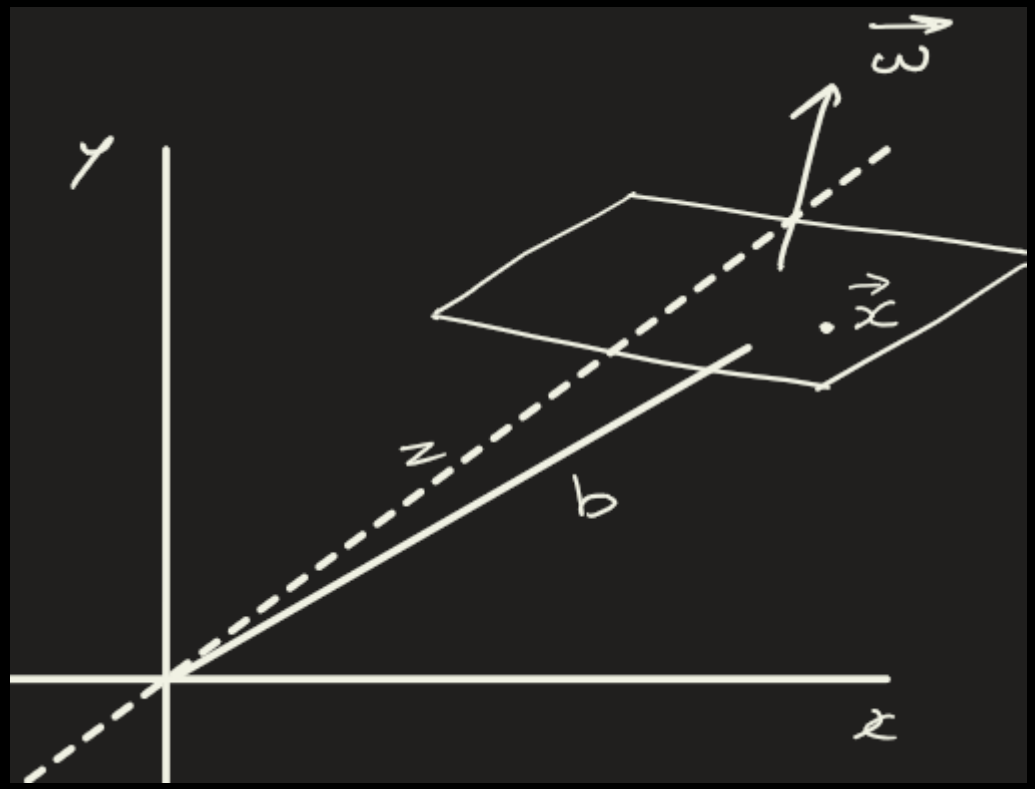
Hyperplane:

Any hyperplane can be written as the set of points **x** satisfying

w·**x** + b = 0,

w is a vector representing the direction of the hyperplane's normal (perpendicular).

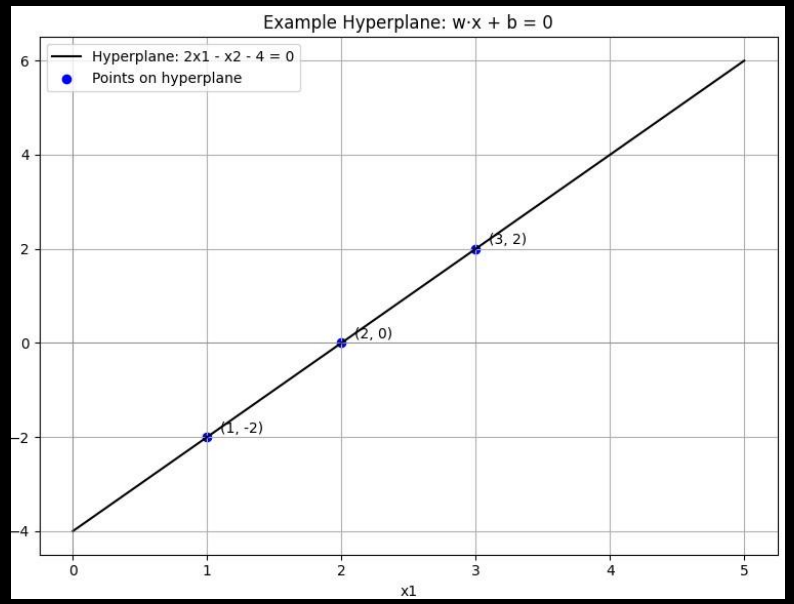
b is a scalar value that determines the offset of the hyperplane from the origin.



1) Example in 2D: Here **x** = $[x_1, x_2]$
If **w** = [2, −1] and b = -4
then equation of hyperplane is

w·x + b = 0

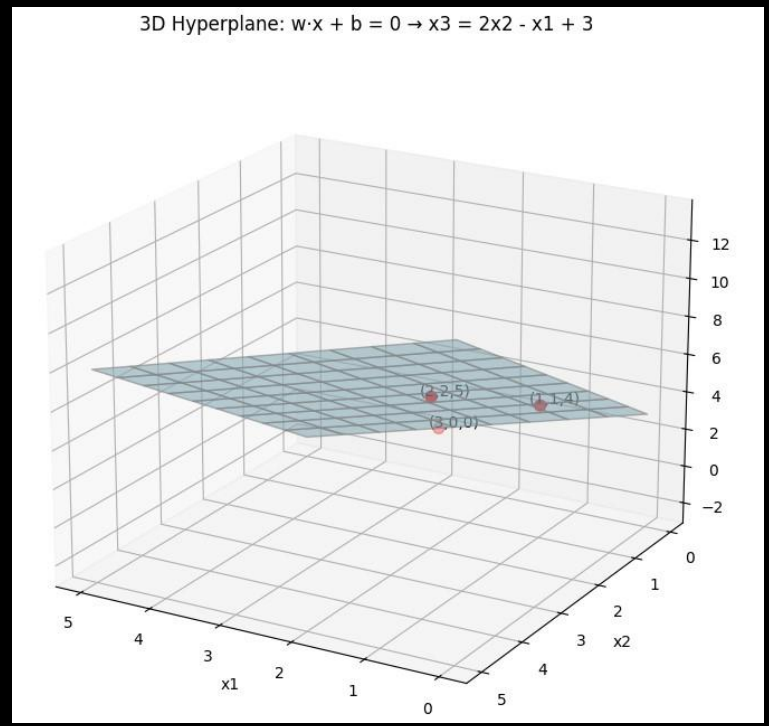
=> $2x_1 - x_2 - 4 = 0$



2) Example in 3D: Here **x** = $[x_1, x_2, x_3]$
If **w** = [1, -2, 1] and b = -3
then equation of hyperplane is

w·x + b = 0

=> $x_1 - 2x_2 + x_3 - 3 = 0$

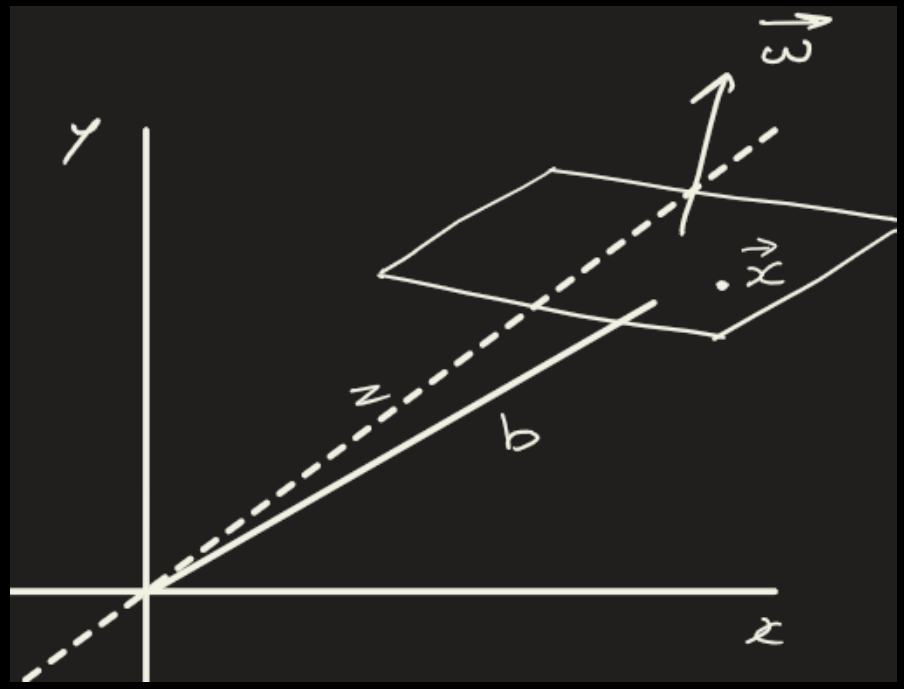


Support Vector Machine

The goal of SVM is to find the **maximum separating hyperplane** between the different classes, at the same time **minimizing the classification error**.
This maximum separation is also called **maximum margin**.

In other words, we find **w and b** of the maximum separating hyperplane from the data points.

SVM finds **w and b** from the **training data.**

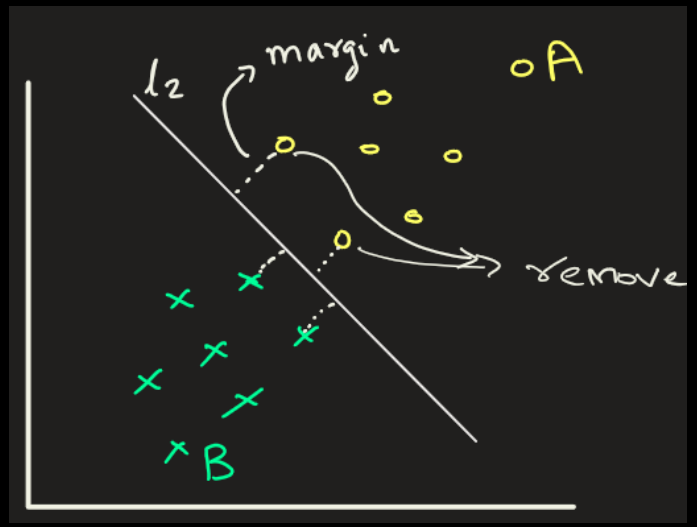


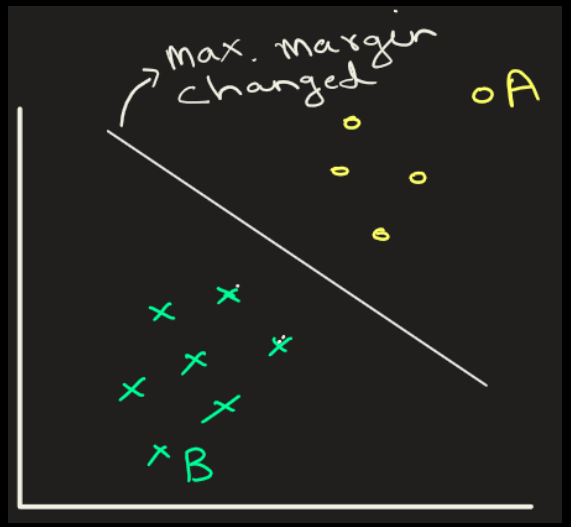
Support Vector Machine

SVM only uses points that lie closest to the boundary, i.e., **support vectors.**
These points determine the **w and b** of the plane.

If you remove support vectors then

- w changes
- b changes and
- Therefore, boundary changes

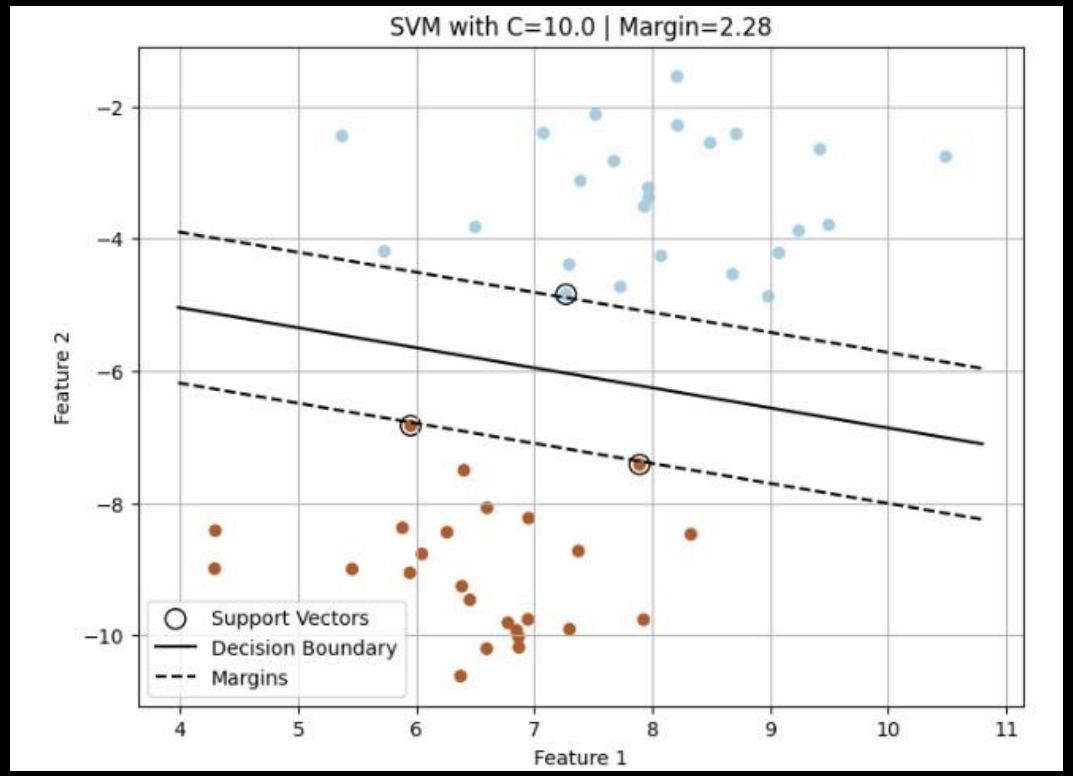




If you remove any **non-support vectors**:
w and b stay the **Same**

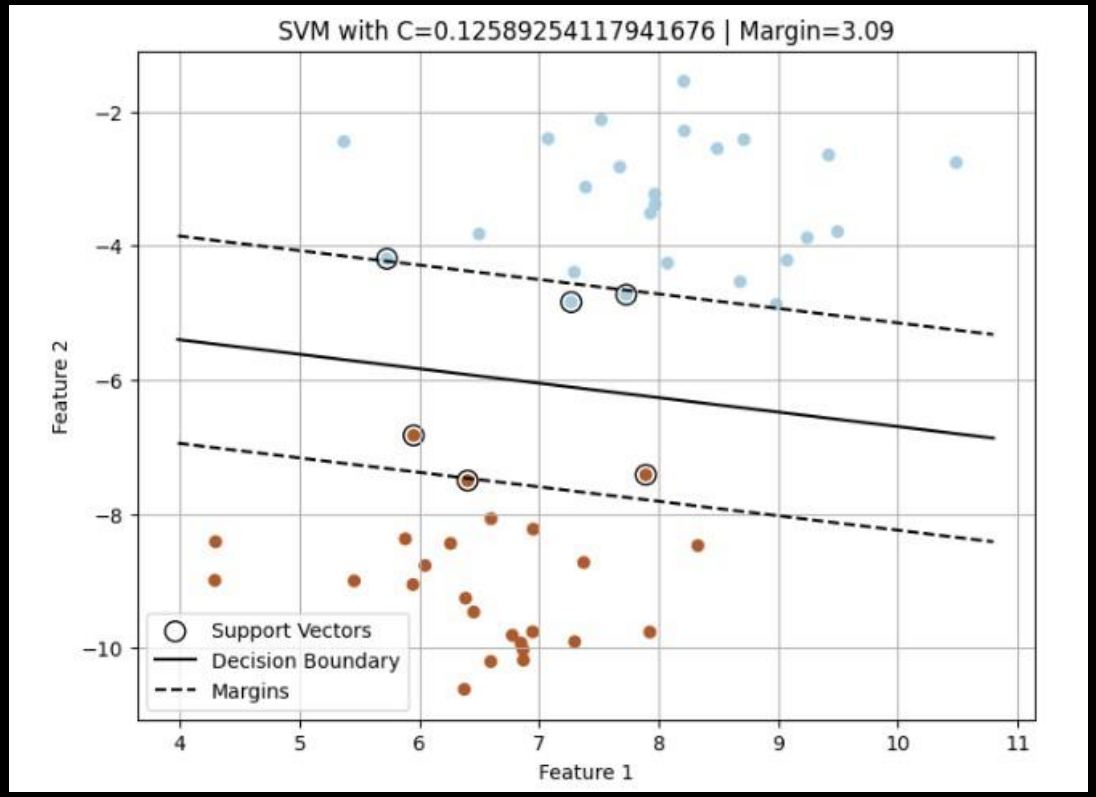
Hard margin: There will be **no data point inside the margin**.

- It tries to classify everything perfectly → **overfits**
- Here hyperparameter C is large.

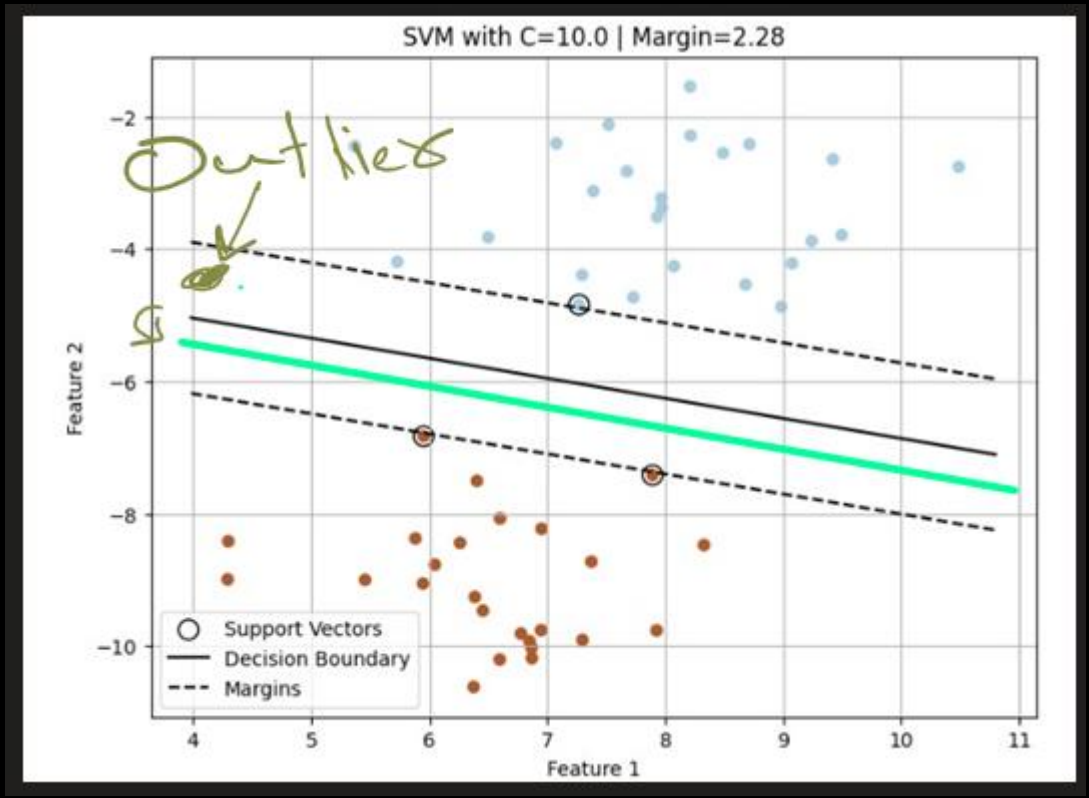


Soft margin: There can be **few data points inside the margin**.

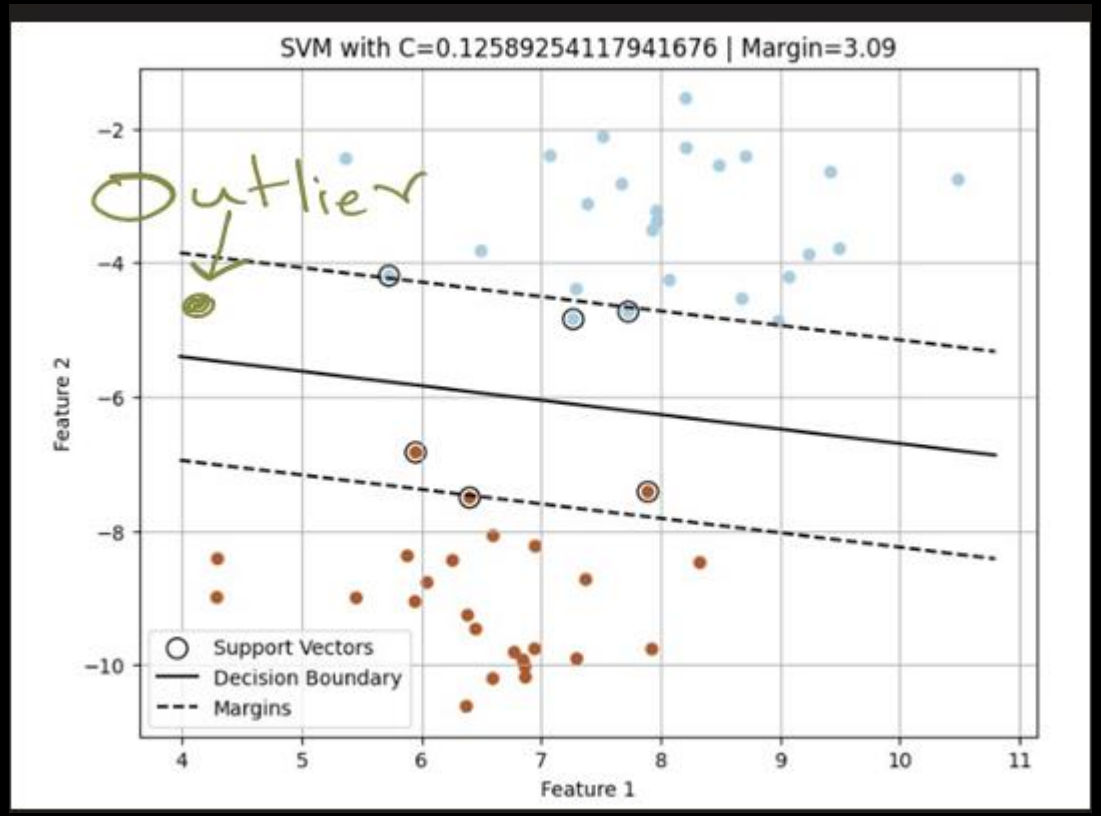
- It allows a few violations → **better generalization → less overfitting**
- Here hyperparameter C is small.
- More realistic for real-world ML tasks.



Hard margin:
It is less robust to outliers. An outlier can ruin its boundary.



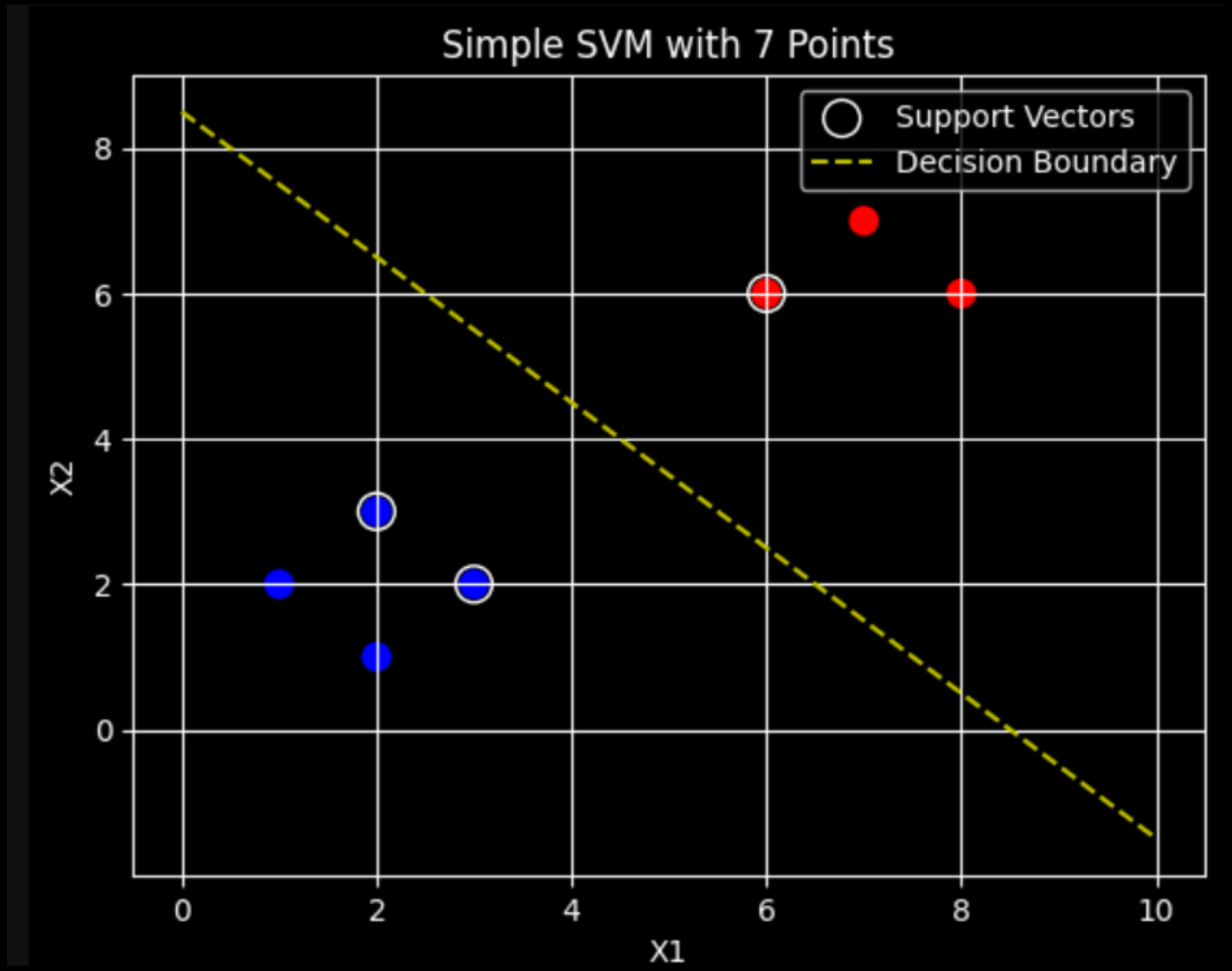
Soft margin:
It is more robust to outliers. An outlier would not ruin its boundary because it allows some errors.



Support Vector Machine: Scikit-Learn Code

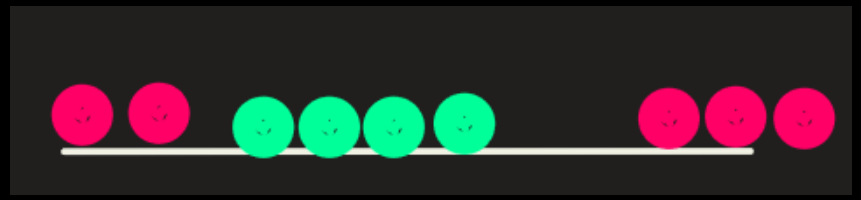
```python
import numpy as np
from sklearn.svm import SVC

X = np.array([
    [1, 2],
    [2, 1],
    [2, 3],
    [3, 2],
    [6, 6],
    [7, 7],
    [8, 6]
])

# Labels (0 or 1) for above data
y = np.array([0, 0, 0, 0, 1, 1, 1])

model = SVC(kernel='linear')
model.fit(X, y)

data = np.array([[2,2], [8,4]])
print("prediction:", model.predict(data))
```

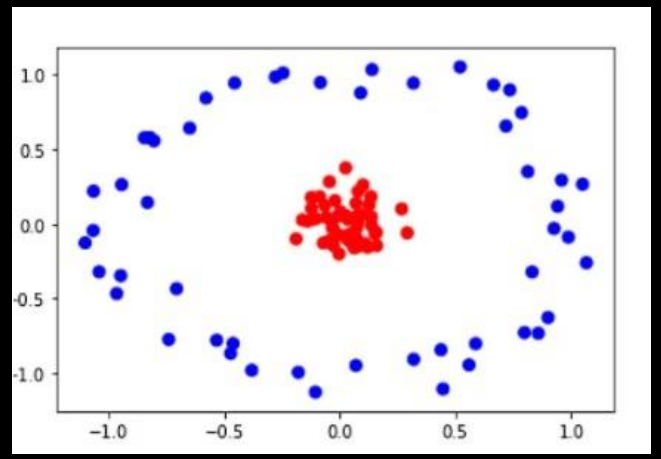
```
prediction: [0 1]
```



Support Vector Machine: Non-Linear

What if the points are not linearly separable, i.e. points are non linear?





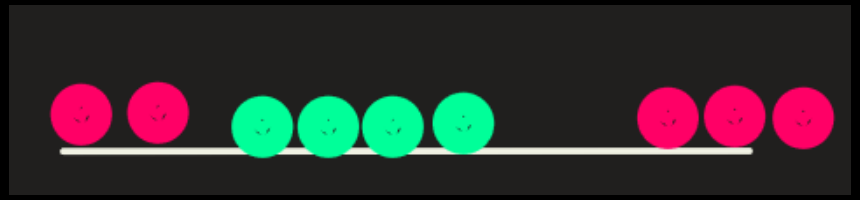
For example, 1D point could represent drug dosage:
Red points are dosage that did not work(either low/high dosage),
Green represents dosage that worked (right amount)

2 Solutions

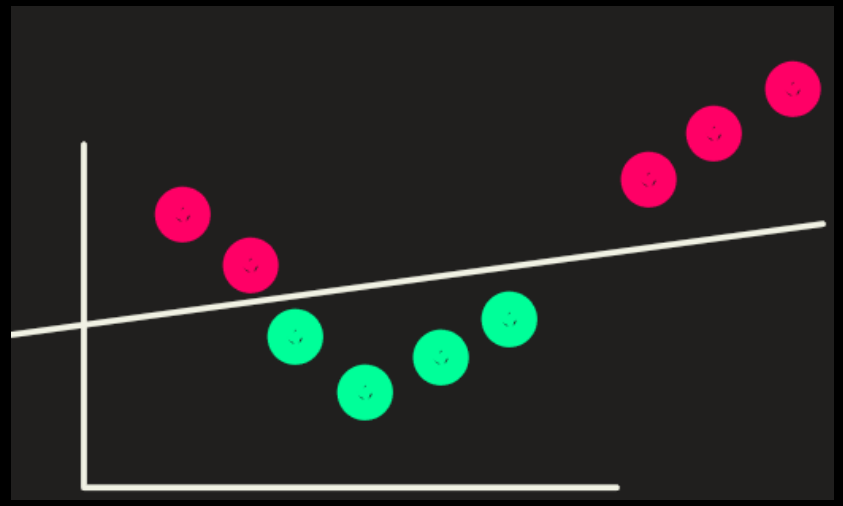
1) Soft-margin as explained previously

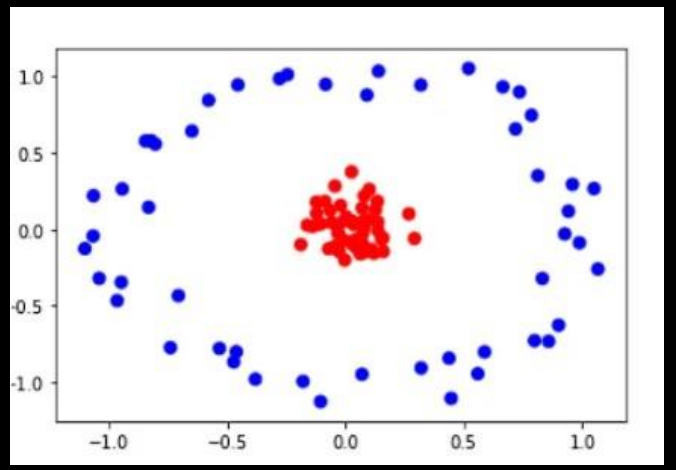
2) Kernel functions (AKA Kernel trick)

- **Project data points into higher dimension** where data becomes linearly separable.
- This is done by creating new variable by using function called kernel (K).

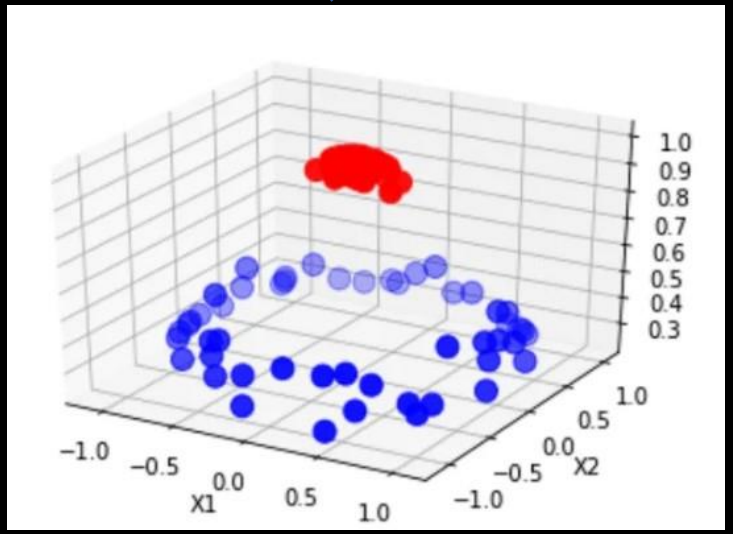


1D to 2D





2D to 3D



Example of kernel trick (1D->2D):

Suppose that data points are x = [-2, -1, 0, 1, 2]. The points in 1D are not linearly separable.

Then using kernel function ϕ(x) = (x, x^2)

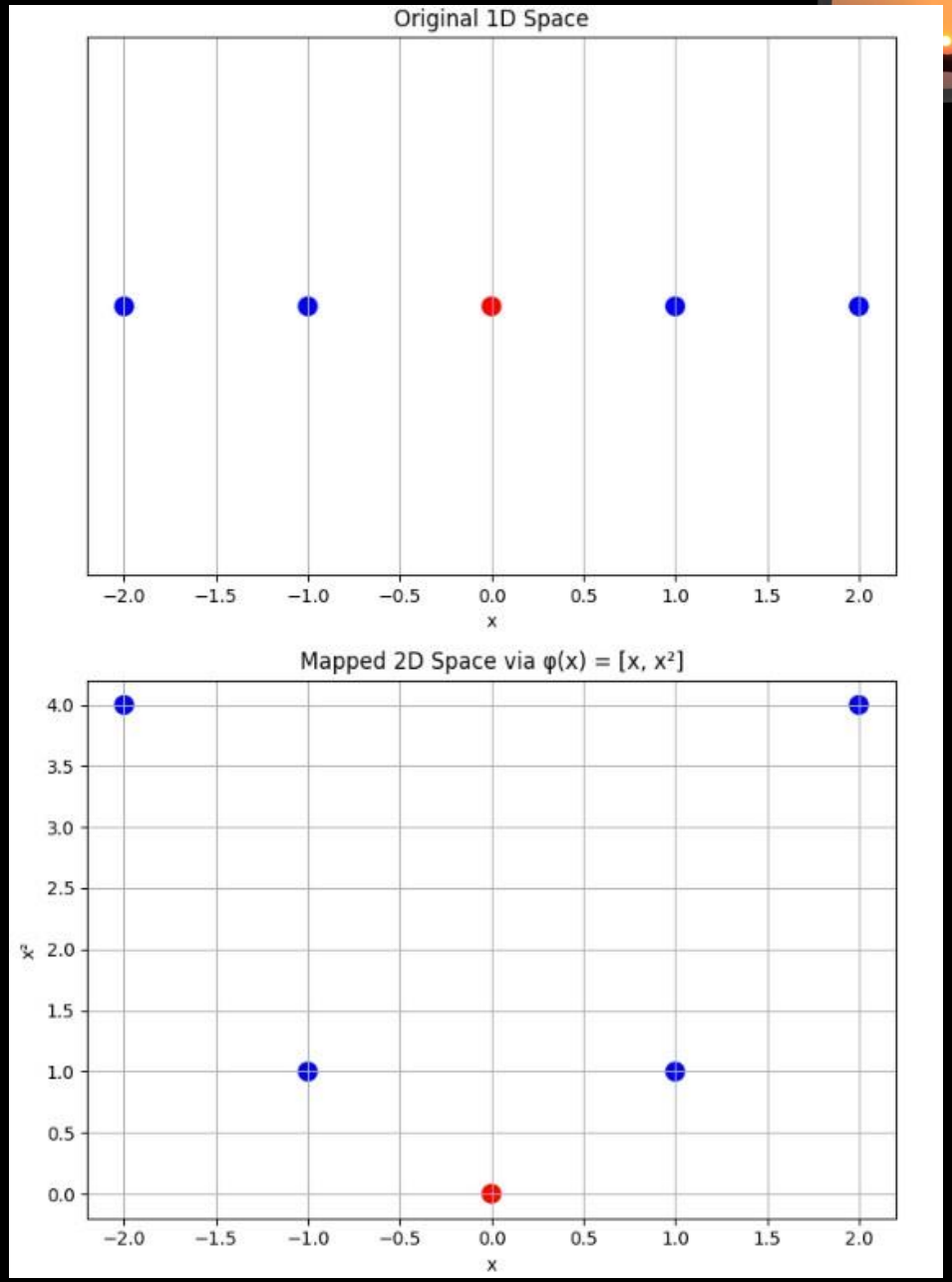
ϕ(-2)=(-2, 4)

ϕ(-1)=(-1, 1)

…and so on.

We see that points are now linearly separable in 2D.

Now we can use linear SVM method to find the hyperplane



Example of kernel trick(2D->3D):

We see that the points in 2D are not linearly separable.

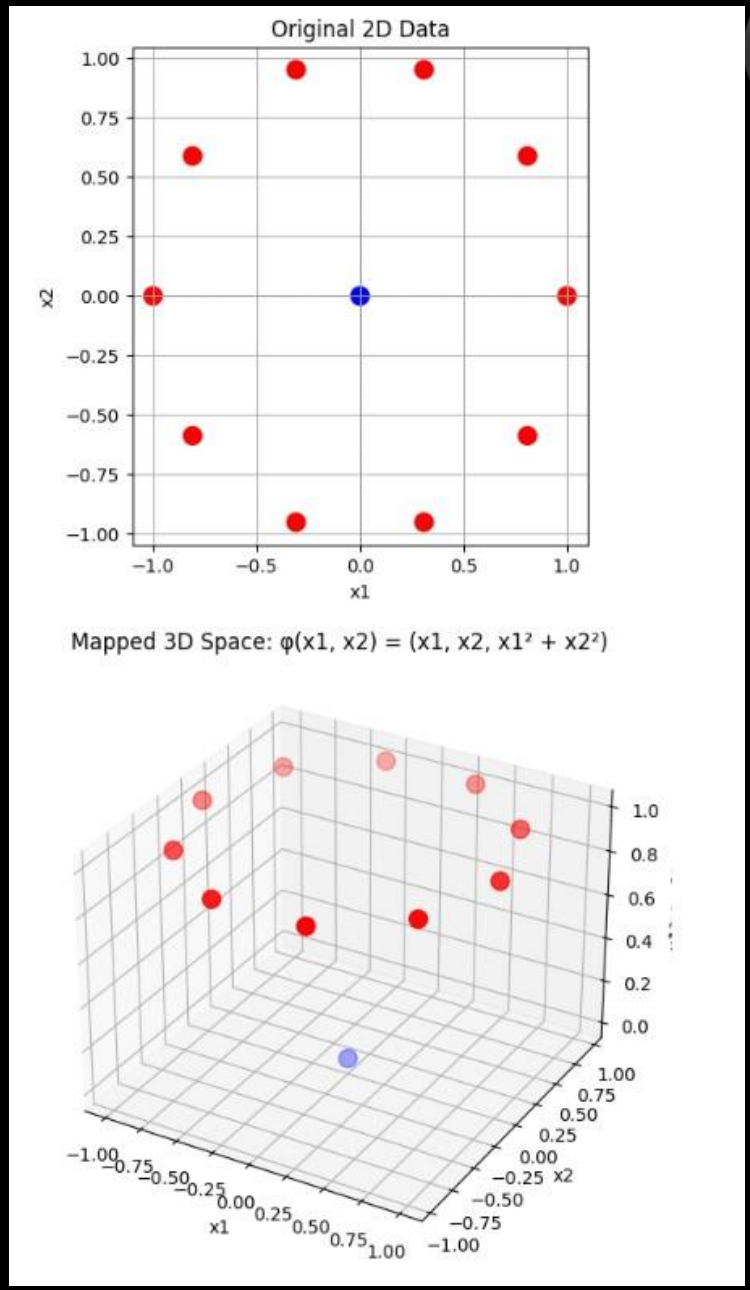
Then using kernel function
$\phi(x_1, x_2) = (x_1,\ x_2,\ x_1^2 + x_2^2)$

$\phi(-1, 0) = (-1, 0, 1)$

$\phi(1, 0) = (1, 0, 1)$

and so on.

We see that points in 3D are now linearly separable.
Now we can use linear SVM method in 3D to find the hyperplane



Support Vector Machine: Advantage

- **Works well with complex data:** SVM is great for datasets where the separation between categories is not clear. It can handle both linear and non-linear data effectively.

- **Effective in high-dimensional spaces:** SVM performs well even when there are more features (dimensions) than samples(i.e. **R^n > samples**), making it useful for tasks like text classification or image recognition.

- **Robust to outliers:** SVM is less affected by outliers because it focuses on the support vectors (data points closest to the margin), which helps in creating a more generalized model.

- **Avoids overfitting:** SVM focuses on finding the best decision boundary (margin) between classes, which helps in reducing the risk of overfitting, especially in high-dimensional data.

- **Versatile with kernels:** By using different kernel functions (like linear, polynomial, or radial basis function), SVM can adapt to various types of data and solve complex problems.
  K1*K2: By combining kernel functions, we can achieve more complex hyperplanes

Support Vector Machine: Disadvantage

- **Slow with large datasets:** SVM can be computationally expensive and slow to train, especially when the dataset is very large.
- **Difficult to tune:** Choosing the right kernel and parameters (like C and gamma) can be tricky and often requires a lot of trial and error.
- **Poor performance when num. of features > num. of samples**
- **Hard to interpret:** Unlike some other algorithms, SVM models are not easy to interpret or explain, especially when using non-linear kernels.
- **Hard to find the right kernels**

EXTRA

Examples of Kernel Functions

- Linear: $K(\mathbf{x}_i,\mathbf{x}_j)= \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j$
  - Mapping Φ: $\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$, where $\boldsymbol{\varphi}(\mathbf{x})$ is $\mathbf{x}$ itself

- Polynomial of power p: $K(\mathbf{x}_i,\mathbf{x}_j)= (1+ \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j)^p$
  - Mapping Φ: $\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$, where $\boldsymbol{\varphi}(\mathbf{x})$ has $\binom{d+p}{p}$ dimensions

- Gaussian (radial-basis function): $K(\mathbf{x}_i,\mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2\sigma^2}}$
  - Mapping Φ: $\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$, where $\boldsymbol{\varphi}(\mathbf{x})$ is *infinite-dimensional*: every point is mapped to *a function* (a Gaussian); combination of functions for support vectors is the separator.

How to Choose the Right Kernel ?

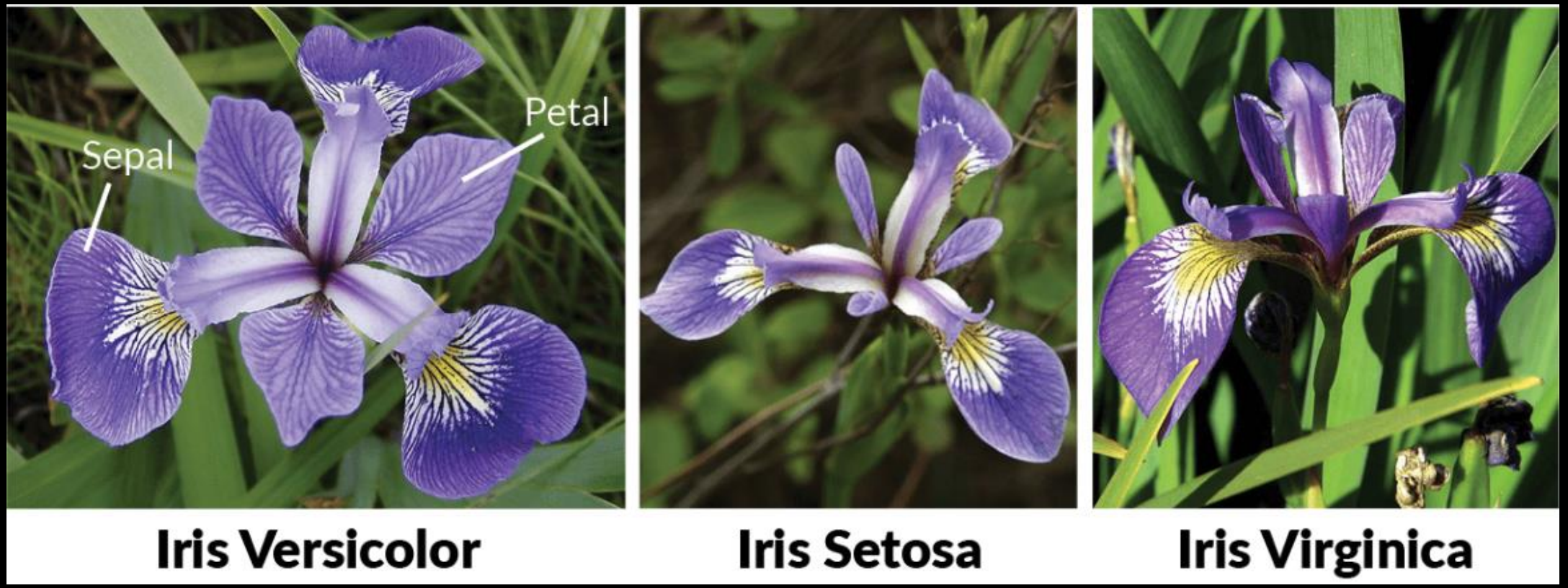
The various kernel functions are applied based on the linear and nonlinear relationships in the feature space.

The **linear kernel** is simple and fast, and it works well with linearly separable data but not with high-dimensional data.

The **polynomial kernel** is well-suited for data with non-linear or polynomial relationships, as well as low-dimensional data.

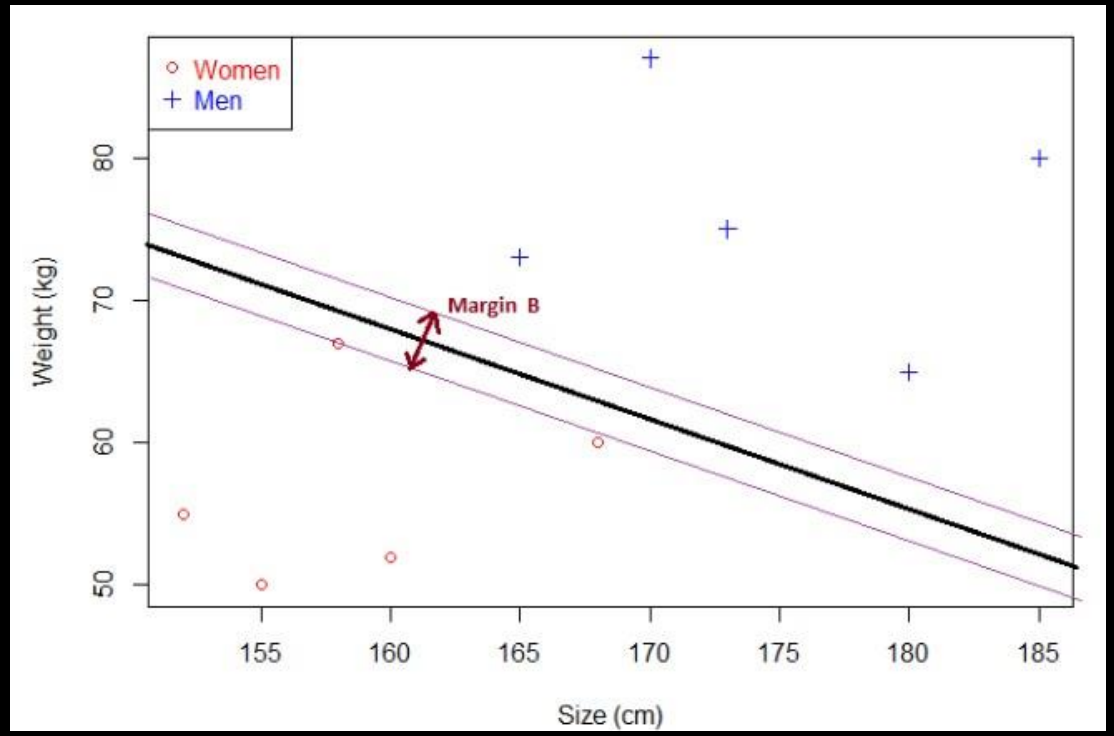
The **RBF kernel** is ideal for dense data that you have no prior knowledge of.

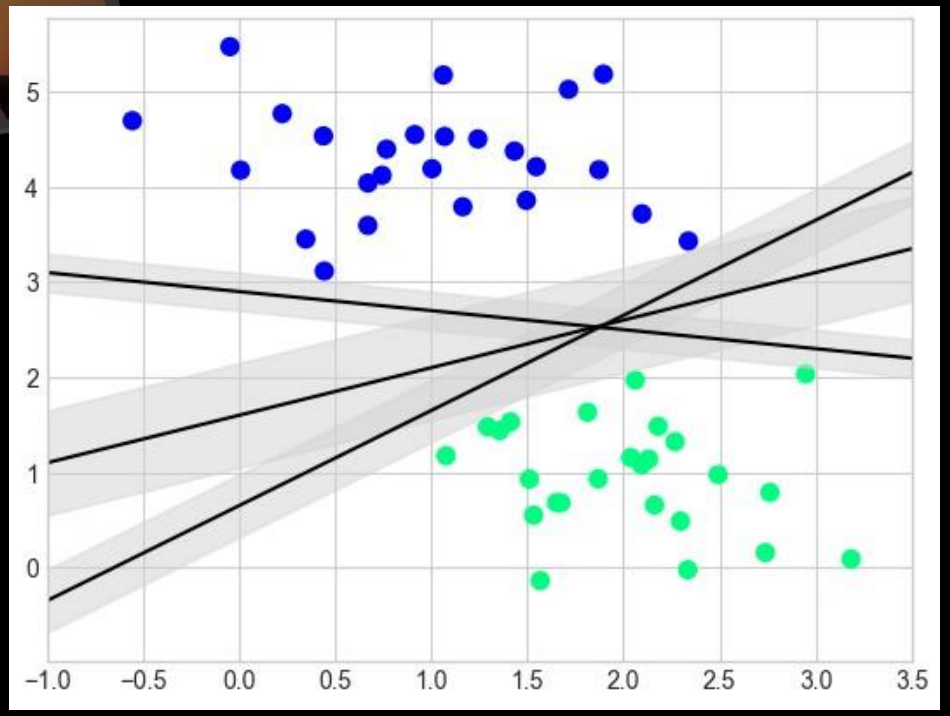
The **sigmoid kernel** works well for binary and categorical data points.


Sepal
Petal
Iris Versicolor
Iris Setosa
Iris Virginica

Key terms

- **Margin**: Given a particular hyperplane, we compute the distance between the hyperplane and the **closest** data point. Once we have this value, then we double it and we will get what is called the **margin**.





Choosing the right hyper plane:

Here we see that there are 3 hyperplanes with margin displayed. The HP in the center has **maximum margin or maximum separation.** This is the optimal hyperplane because it has the biggest margin.
The objective of the SVM **is to find the optimal separating hyperplane which maximizes the margin of the training data.**

How to find the margin? (p1)

In Support Vector Machines, the **margin** is the **distance between the separating hyperplane** and the **nearest data points** (called **support vectors**). SVM aims to maximize this margin.

The decision boundary is defined by:

$$w^T x + b = 0$$

The **margin** is calculated as:

$$\text{margin} = \frac{1}{\|w\|}$$

But in **your teaching example**, since we don't yet have the trained SVM, we'll:

- Manually pick a line (defined by `w` and `b`)
- Then manually compute **perpendicular distances** from that line to each data point
- The **smallest distance** from the decision boundary to any point is the **margin**

✅ Step-by-Step: Manual Margin Calculation

🎯 Let's assume:

You manually set the hyperplane as:

$$w = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad b = -1$$

So the decision boundary is:

$$x - y - 1 = 0 \quad (\text{equivalently: } y = x - 1)$$

📓 Step 1: Use the dataset

Your sample dataset:

```plaintext
Class +1: [1,7], [2,8], [3,8]
Class -1: [5,1], [6,-1], [7,3], [5,-2]
```

Note: Later we will see how to find **w and b** using data points.

How to find the margin? (p2)

Step 2: Plug into distance formula

For a point $x_i = (x, y)$, the **distance to the hyperplane** $w^T x + b = 0$ is:

$$\text{distance} = \frac{|w_0 \cdot x + w_1 \cdot y + b|}{\sqrt{w_0^2 + w_1^2}}$$

For example, let's do it for each point:

✔ **Point (1, 7):**

$$\text{distance} = \frac{|1 \cdot 1 + (-1) \cdot 7 - 1|}{\sqrt{1^2 + (-1)^2}} = \frac{|1 - 7 - 1|}{\sqrt{2}} = \frac{|-7|}{\sqrt{2}} = \frac{7}{1.414} \approx 4.95$$

✔ **Point (2, 8):**

$$\frac{|2 - 8 - 1|}{\sqrt{2}} = \frac{|-7|}{\sqrt{2}} = 4.95$$

✔ **Point (5,1):**

$$\frac{|5 - 1 - 1|}{\sqrt{2}} = \frac{3}{1.414} \approx 2.12$$

✔ **Point (7,3):**

$$\frac{|7 - 3 - 1|}{\sqrt{2}} = \frac{3}{1.414} \approx 2.12$$

✔ **Point (5,-2):**

$$\frac{|5 - (-2) - 1|}{\sqrt{2}} = \frac{6}{1.414} \approx 4.24$$

Step 3: Identify Support Vectors

Find the **smallest distance** — that's your **margin**.

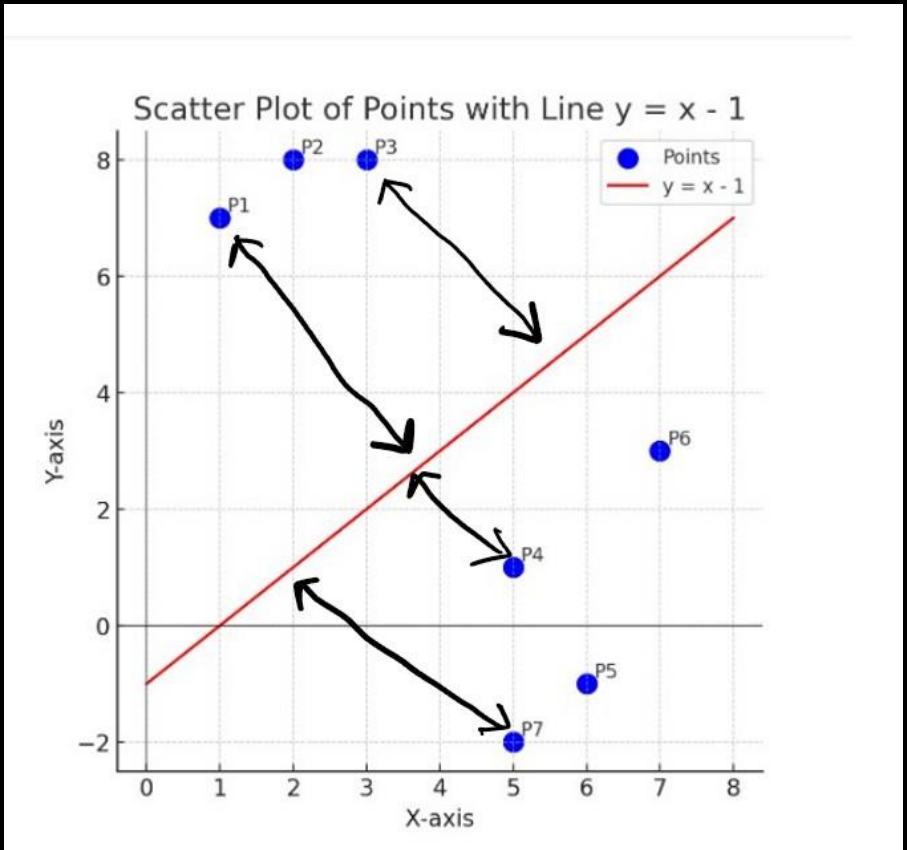
From above:

- Closest distance ≈ **2.12**
- Points at distance 2.12: **(5,1)** and **(7,3)** ⇒ **Support Vectors**

Final Result:

- **Hyperplane:** $x - y - 1 = 0$
- **Support Vectors:** (5,1), (7,3)
- **Margin:** ≈ **2.12**

How to find the margin? (p3)



Summary Table:

Point	Class	Distance to Hyperplane
(1,7)	+1	4.95
(2,8)	+1	4.95
(3,8)	+1	4.24
(5,1)	-1	**2.12**
(6,-1)	-1	4.95
(7,3)	-1	**2.12**
(5,-2)	-1	4.24

What is Goal of SVM? (p1)

🎯 Goal of SVM

We are given data points (x_i, y_i) where $x_i \in \mathbb{R}^n$, and labels $y_i \in \{-1, +1\}$. Our goal is to **find a hyperplane that separates the two classes**.

A hyperplane in n-dimensions is:

$$w^T x + b = 0$$

This line (or plane) splits the space into two sides:

- One side is where $w^T x + b > 0$
- The other side is where $w^T x + b < 0$

We want:

$$y_i(w^T x_i + b) \geq 1 \quad \text{for all } i$$

💡 So what's the problem?

There are **infinitely many hyperplanes** that can separate two classes. But SVM wants the **best** one. What does "best" mean?

🎯 SVM's Big Idea: **Maximum Margin**

We define the **margin** as the distance from the hyperplane to the **closest points** in either class (the support vectors).

The SVM chooses the hyperplane that **maximizes the margin**, i.e., it **stays as far away as possible from both classes**, giving better generalization on unseen data.

📏 Geometry: Margin and the Norm of `w`

Let's say we have a hyperplane:

$$w^T x + b = 0$$

Then the **distance** from a point x_i to the hyperplane is:

$$\text{Distance} = \frac{|w^T x_i + b|}{\|w\|}$$

For support vectors (closest points), we **enforce**:

$$y_i(w^T x_i + b) = 1$$

So their distance is:

$$\frac{1}{\|w\|}$$

Since there are two margin boundaries (for +1 and -1 classes), the **full margin** becomes:

$$\text{Margin} = \frac{2}{\|w\|}$$

What is Goal of SVM? (p2)

✅ Now comes the key point:

Maximize the margin ⇒ Maximize $\frac{2}{\|w\|}$

But maximizing $\frac{2}{\|w\|}$ is the same as minimizing $\|w\|$, or more commonly $\frac{1}{2}\|w\|^2$ (for math convenience in optimization).

🧠 So the optimization problem becomes:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

subject to:

$$y_i(w^T x_i + b) \geq 1 \quad \forall i$$

🎓 Why do we minimize the norm of `w`?

- It directly corresponds to **maximizing the margin**, giving the best separation.
- Leads to **better generalization** (less overfitting).
- Ensures a **unique, well-defined solution.**

How to find the w, b? (p1)

🧠 Background: What is the dual form?

For a **hard-margin** SVM (perfectly separable data), the **primal problem** is:

🔷 Primal Form:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

subject to:

$$y_i(w^T x_i + b) \geq 1 \quad \forall i$$

We convert this to its **dual form** to make computation easier and to rely only on dot products.

🔷 Step-by-Step Dual Form Derivation:

Let's denote:

- α_i are the **Lagrange multipliers**

The **dual optimization problem** is:

🔷 Dual Form:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

subject to:

$$\sum_{i=1}^{n} \alpha_i y_i = 0, \quad \alpha_i \geq 0$$

Once we find optimal α_i, we compute:

🔸 Weight vector:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

🔸 Bias term:

Pick any **support vector** x_k (where $\alpha_k > 0$) and use:

$$b = y_k - w^T x_k$$

How to find the w, b? (p2)

Now we can use programming language to

1. Set up the quadratic programming (QP) problem.
2. Solve for α using data points (x,y).
3. Recover w and b.
4. Verify with a small printout.

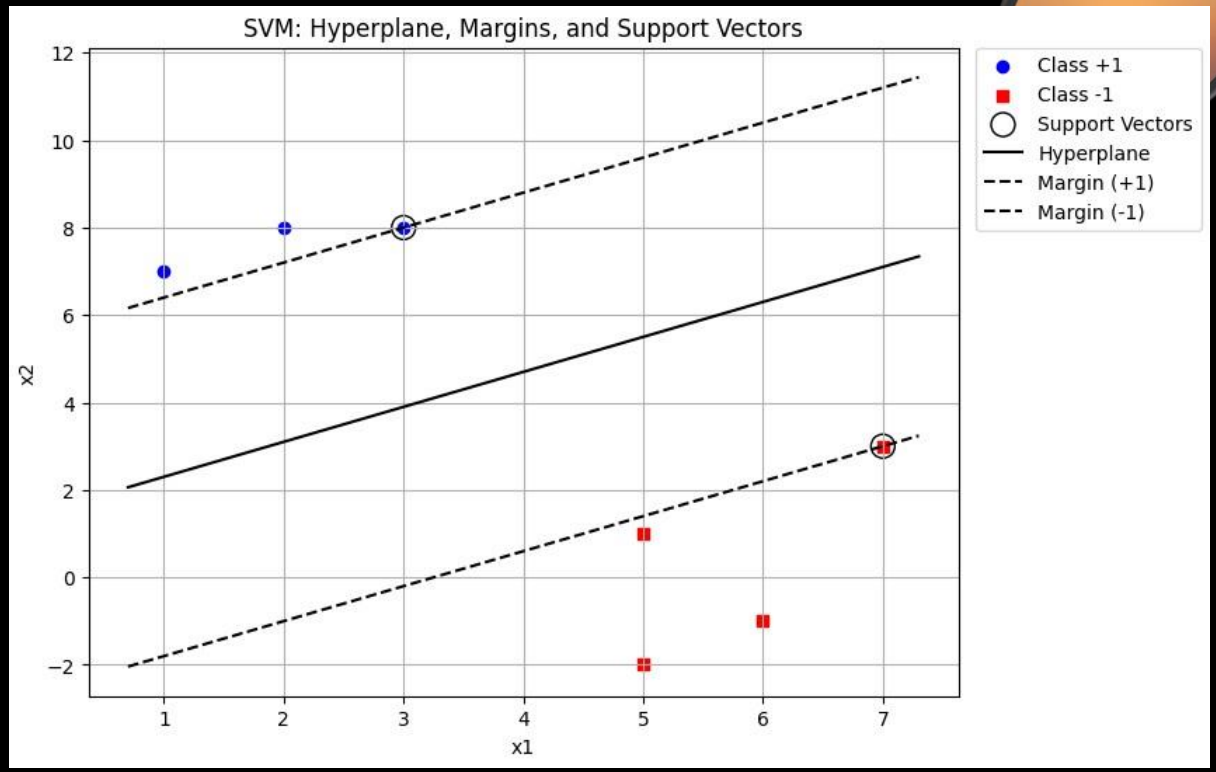
(See jupyter notebook on how the above is done using python package cvxopt)

For above example:
Alphas α :
[1.95065919e-10 3.33152447e-10 4.87804874e-02 8.40968653e-10 1.75231202e-10
4.87804867e-02 1.91523417e-10]

Support Vectors: [[3. 8.] [7. 3.]]

Weights (w): [-0.19512194 0.24390244]
Bias (b): -0.36585369063492074

Predictions:
Point [1. 7.] => Predicted: 1.0, Actual: 1.0
Point [2. 8.] => Predicted: 1.0, Actual: 1.0
Point [3. 8.] => Predicted: 1.0, Actual: 1.0
Point [5. 1.] => Predicted: -1.0, Actual: -1.0
Point [6. -1.] => Predicted: -1.0, Actual: -1.0
Point [7. 3.] => Predicted: -1.0, Actual: -1.0
Point [5. -2.] => Predicted: -1.0, Actual: -1.0



Hard and soft margin (p1)

Hard Margin SVM

The Hard Margin SVM ensures all the data points are all properly classified without error, ensuring that the data points don't find themselves in the other part of the hyperplane, and also maximizing the margin. It's an effective method for a "noise-free" dataset. This is achieved by minimizing an objective function given below:

Hard Margin SVM Objective Function: $\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$

Subject to:

$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \ \forall i$

Where:

y_i : Class label of the i-th sample ($+1$ or -1)

$\mathbf{x}_i$: Feature vector of the i-th sample

This constraint given above in the objective function ensures that all the data points are not misclassified and the stay outside the margin.

Soft Margin SVM

The Soft Margin SVM is lenient, as it allows some misclassifications. It's suitable for real-world datasets, which are noisy, and it handles non-linearly separable data. It introduces a slack variable that penalizes incorrect predictions.

Objective Function: $\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$

Subject to:

$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \ \forall i$

$\xi_i \geq 0, \ \forall i$

Where:

ξ_i : Slack variables representing the degree of misclassification or margin violation.

C : Regularization parameter controlling the trade-off between margin maximization and error minimization.

See next slide for plot of both

Hard and soft margin (p2): see jupyter n/b

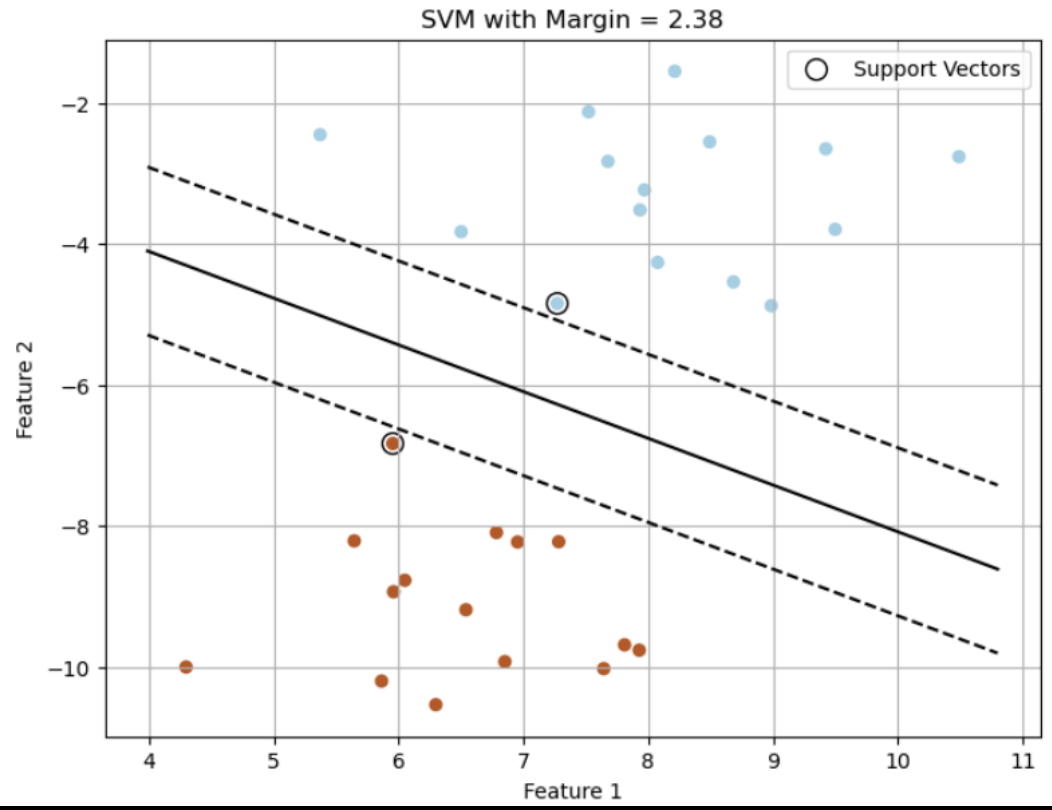
yter tut_ml_SVM_Example_Hard-Soft-Margin Last Checkpoint: 2 minutes ago

View Run Kernel Settings Help

Code

```
plt.grid(True)
plt.show()
```

```
w: [-0.4635122  -0.69914599]
b: -1.0154377593826265
```



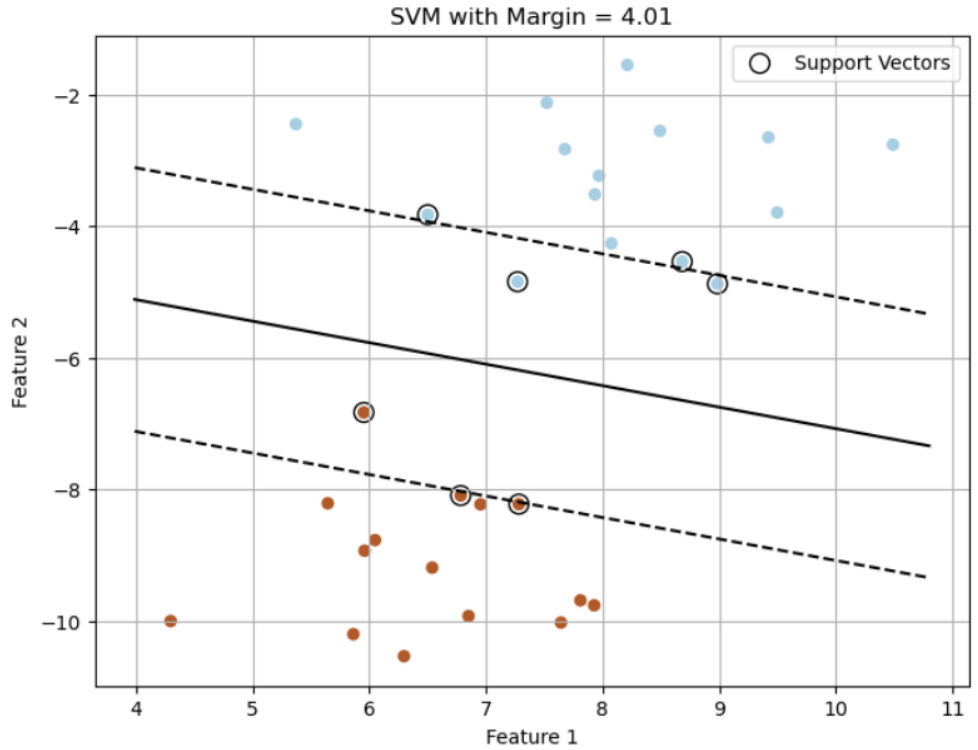
jupyter tut_ml_SVM_Example_Hard-Soft-Margin Last Checkpoint: 3 minutes ago

Edit View Run Kernel Settings Help

Code

```
plt.grid(True)
plt.show()
```

```
w: [-0.15499481 -0.47427575]
b: -1.8060684271730607
```



Non Linear SVM(TODO):

Resources

- https://www.freecodecamp.org/news/svm-kernels-how-to-tackle-nonlinear-data-in-machine-learning/

Non linear SVM(TODO):

- **But there is a little big problem in finding the linear hyperplane in higher dimension**: If you recall, to determine hyperplane we have to solve the dual problem:

💡 The dual problem depends entirely on dot products:

The optimization problem becomes:

$$\max_{\alpha} \sum_{i} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \underbrace{\phi(x_i)^\top \phi(x_j)}_{\text{dot product in high-dim space}}$$

So:

✅ **We only ever need the dot product** between pairs of data points in the feature space.

You can map **n-dimensional data into any d≤n−1 dimensions** *to guarantee linear separability*. Since the number of dimensions/features went up, so it turned the mapping and dot-product process into **computationally expensive** process.

Enter the Kernel Trick

A **kernel function** lets you compute the **dot product in the higher-dimensional space without explicitly mapping** the data. The KF calculates the relationship between pairs of points in higher dimensions. The kernel function is defined as:

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

The kernel trick is the computation of the inner or dot product between the data points in the **original lower dimensional space** instead of transforming the data into a higher-dimensional space. In essence, the data points are not actually transformed into higher dimensions, but it performs computation as if in higher dimension.

Example1: **quadratic polynomial kernel**:
A quadratic polynomial kernel (degree-2) transforms input vectors into a higher-dimensional space and calculates their dot product. For vectors x and y, it's defined as **(1 + x · y)^2, which expands to 1 + 2(x · y) + (x · y)^2**. This kernel can be used in algorithms like Support Vector Machines (SVMs) to find non-linear decision boundaries.
Here's a breakdown:
1.Input Vectors: Consider two vectors, **$x = (x_1, x_2)$** and **$y = (y_1, y_2)$.** These are your original data points in a 2-dimensional space.

2.Dot Product: The dot product of x and y is calculated as **$x_1y_1 + x_2y_2$.**

3.Kernel Function: The quadratic polynomial kernel (degree 2) is defined as:
$K(x, y) = (1 + x \cdot y)^2 = (1 + x_1y_1 + x_2y_2)^2$.

Expansion: Expanding this expression, you get:
$K(x, y) = 1 + 2(x_1y_1 + x_2y_2) + (x_1y_1 + x_2y_2)^2$
$= 1 + 2x_1y_1 + 2x_2y_2 + (x_1^2y_1^2 + 2x_1x_2y_1y_2 + x_2^2y_2^2)$
$= 1 + 2x_1y_1 + 2x_2y_2 + x_1^2y_1^2 + 2x_1x_2y_1y_2 + x_2^2y_2^2$.

Higher Dimensional Space: This kernel implicitly maps the original vectors to a higher-dimensional space. The transformed features would be:
$(x_1 , x_2) \rightarrow (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2)$ and
$(y_1 , y_2) \rightarrow (1, \sqrt{2}y_1, \sqrt{2}y_2, y_1^2, \sqrt{2}y_1y_2, y_2^2)$.
The kernel function then calculates the dot product in this higher-dimensional space.

Kernel Trick: The key advantage is that we don't need to explicitly calculate the transformation φ(x) and φ(y) and their dot product. Instead, we can directly **compute K(x, y) using the original data and the kernel function**. This is known as the kernel trick and it allows us to work with complex non-linear relationships without the computational burden of explicitly transforming the data to a higher dimension,
In essence, the quadratic polynomial kernel allows you to find non-linear decision boundaries in your data by effectively operating in a higher-dimensional space without explicitly calculating the coordinates of the transformed data points.

Non linear SVM(TODO):

Example2: **Polynomial kernel**

Mapping Without Kernel Trick:

The 2D data is given as:

$$\mathbf{x}_1 = (1,1), \quad y_1 = +1$$
$$\mathbf{x}_2 = (-1,-1), \quad y_2 = -1$$

Let's use a mapping function:

$$\phi(x,y) = (x^2, \sqrt{2}xy, y^2)$$

Mapping $\mathbf{x}_1$ *and* $\mathbf{x}_2$:

$$-\ \phi(\mathbf{x}_1) = (1^2, \sqrt{2}(1)(1), 1^2) = (1, \sqrt{2}, 1)$$

$$-\ \phi(\mathbf{x}_2) = ((-1)^2, \sqrt{2}(-1)(-1), (-1)^2) = (1, \sqrt{2}, 1)$$

Dot Product in Higher-Dimensional Space:

$$\phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2) = (1)(1) + (\sqrt{2})(\sqrt{2}) + (1)(1) = 1 + 2 + 1 = 4$$

This is the dot product of $\mathbf{x}_1$ and $\mathbf{x}_2$ after explicitly mapping them to the higher-dimensional space.

Using the Kernel Trick:

Polynomial Kernel Definition:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j + c)^d$$

For this example:

$d = 2$ (degree of the polynomial), $\quad c = 0$ (no bias term)

Given:

$$\mathbf{x}_1 = (1,-1), \quad \mathbf{x}_2 = (-1,-1)$$

Compute $K(\mathbf{x}_1, \mathbf{x}_2)$:

$$\begin{aligned} K(\mathbf{x}_1, \mathbf{x}_2) &= ((1)(-1) + (1)(-1))^2 \\ &= (-1-1)^2 \\ &= (-2)^2 \\ &= 4 \end{aligned}$$

Using the kernel trick, we are able to compute dot product of 2 points in higher dimension without mapping the points to high dimension

Heading Goes Here

Fhdsklf
Fjdsklf
Fjskldf