مقرر التعلم التلقائي | السنة الخامسة – هندسة المعلوماتية / الذكاء الصنعي | Week 5

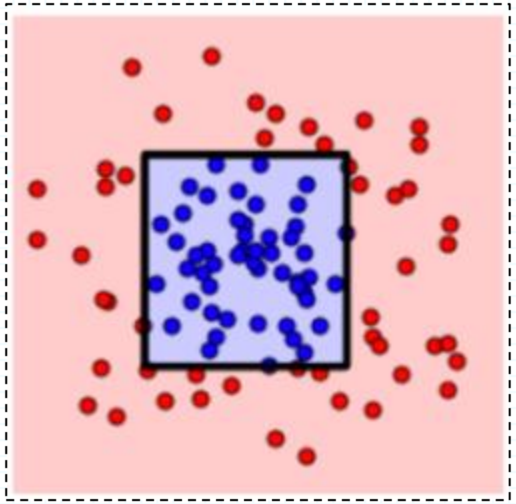
SVM

د. رياض سنبل

Access Course Materials

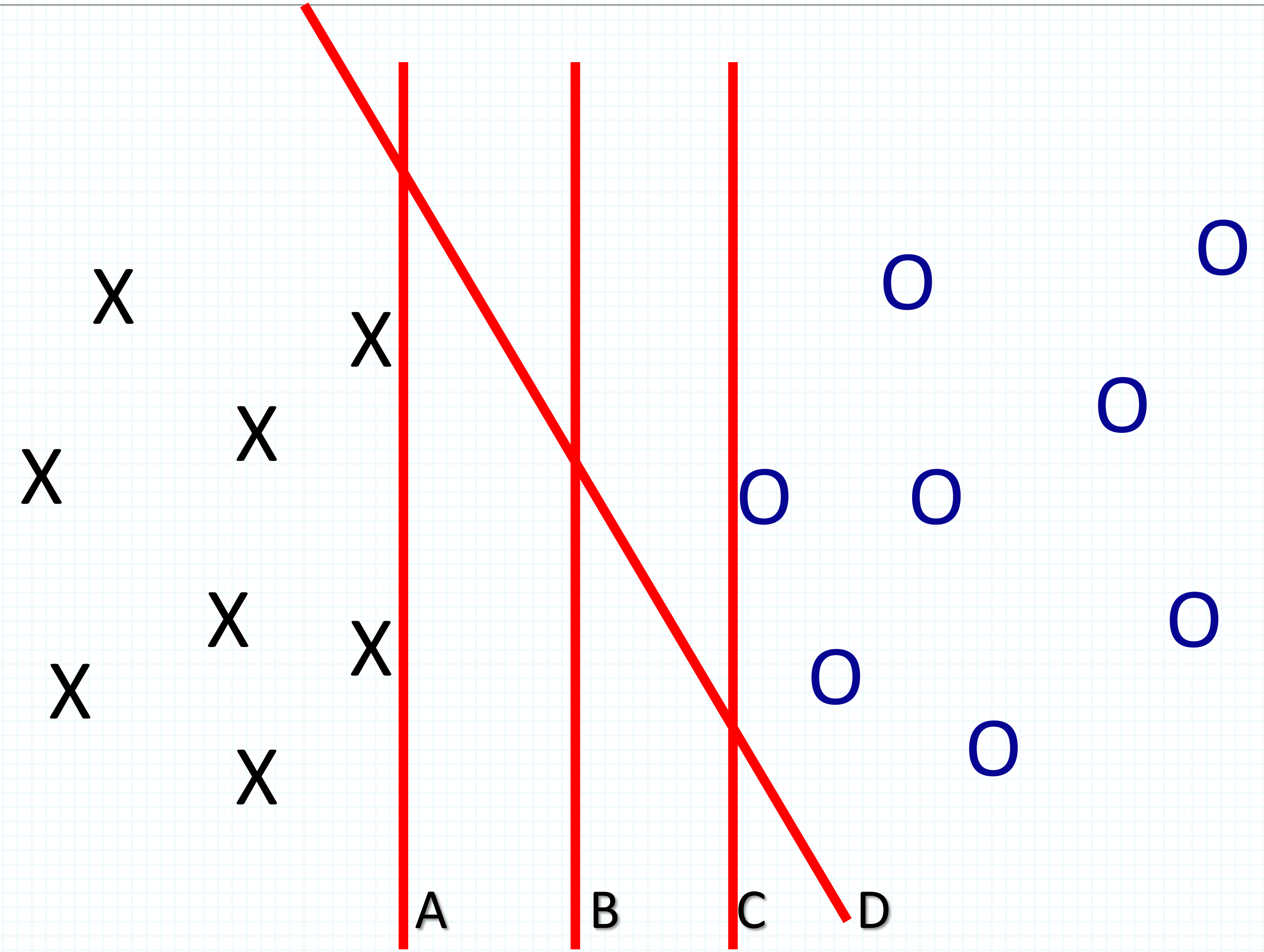
Why SVM? … Why not decision trees?

- **Can Decision Trees detect non-linear models?**
  - Yes, Decision trees can detect non-linear relationships
- **What type of boundaries can be detected using decision trees in each step?**
  - The decision boundary in a Decision Tree is linear and perpendicular to one of the input dimensions, which means that it is limited to finding only axis-parallel splits.

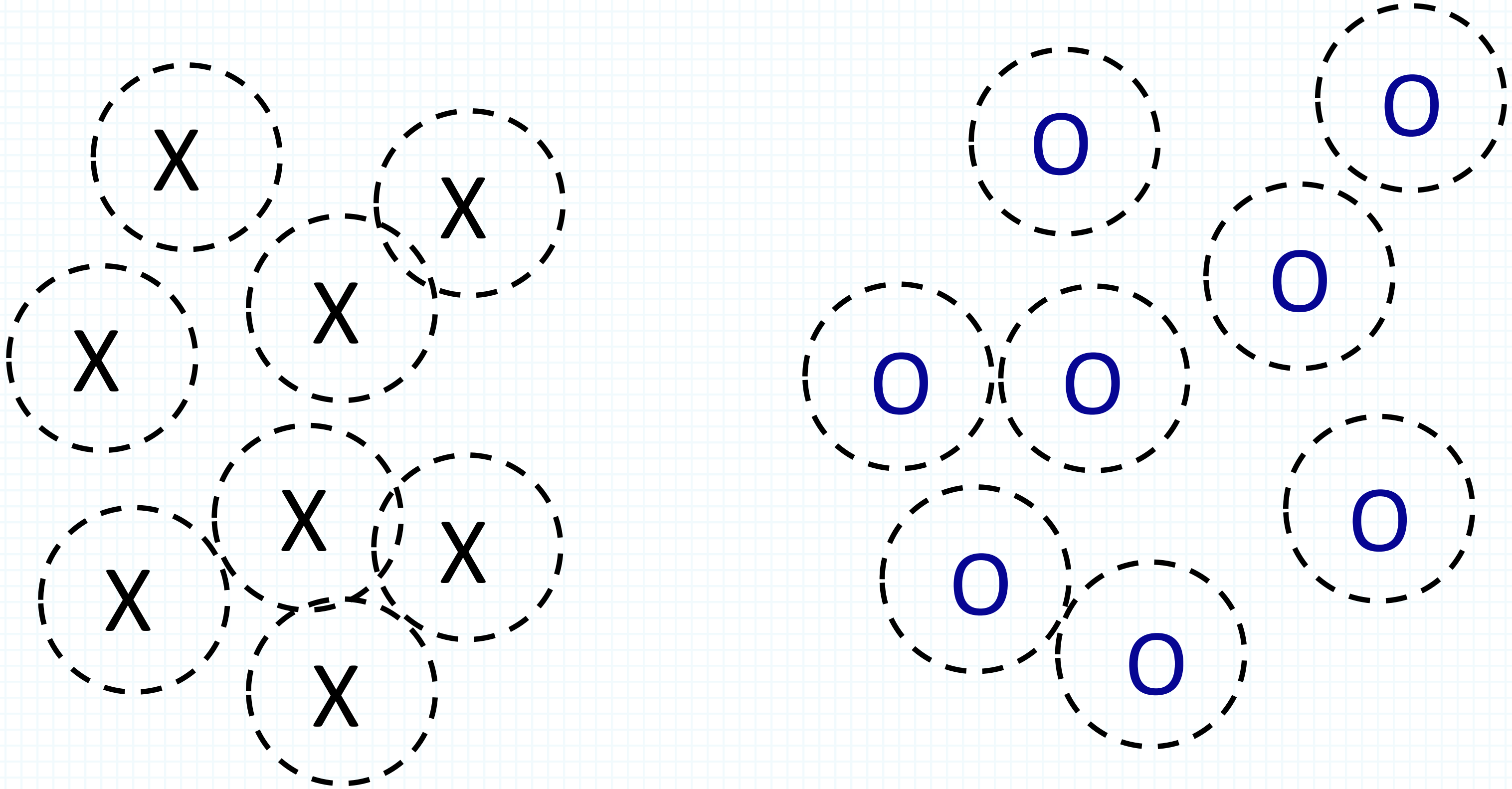


- **What if we have higher-dimensional feature space, more complex relationships between input features and target class?**
  - In the higher-dimensional feature space, the decision boundary can take on a more complex shape, such as a curved or nonlinear boundary.
  - More problems when the relationship between the input features and the target variable is complex (ex: image classification, sentiment analysis, etc)

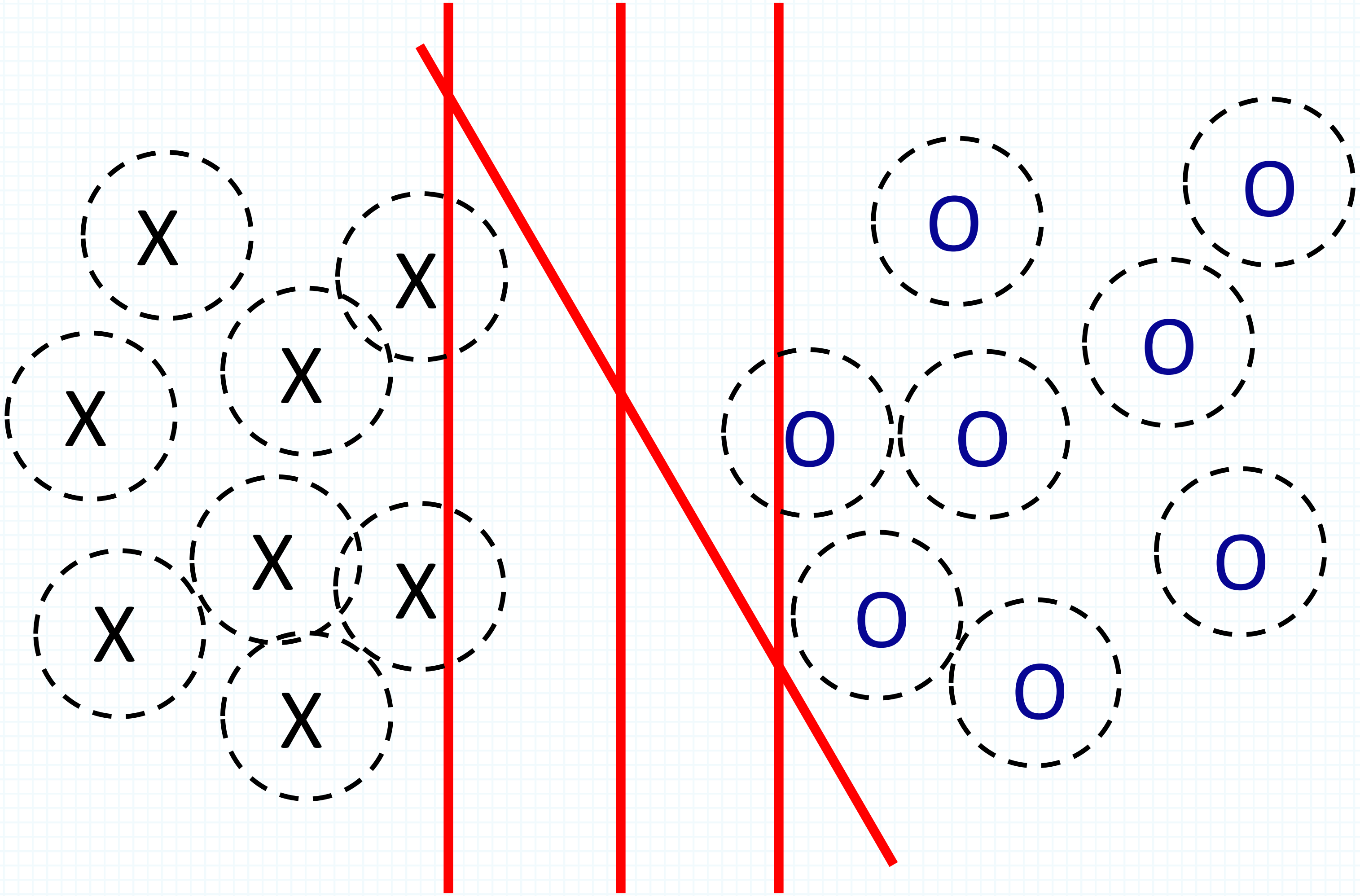
Intuitions



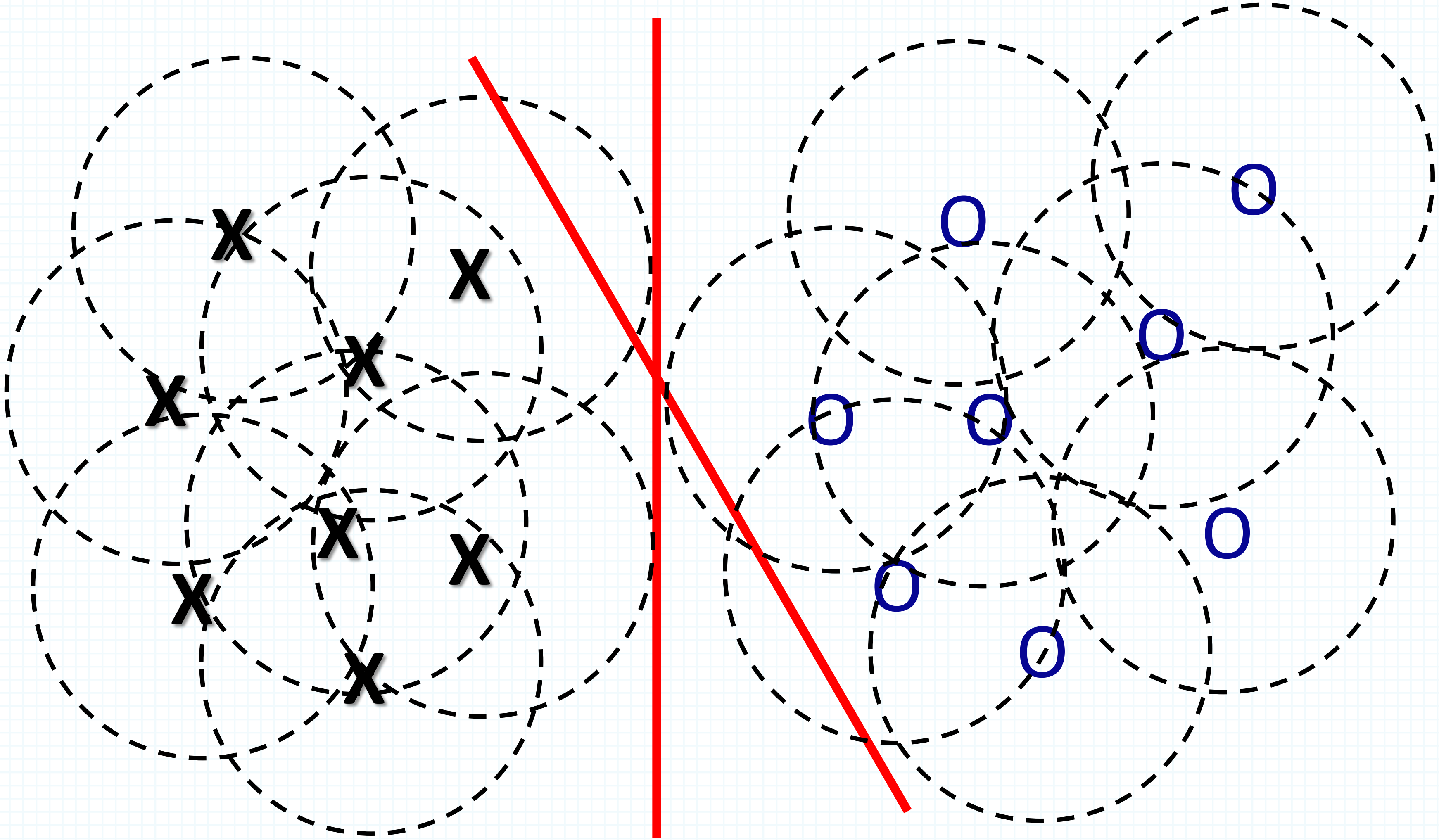
Noise in the Observations



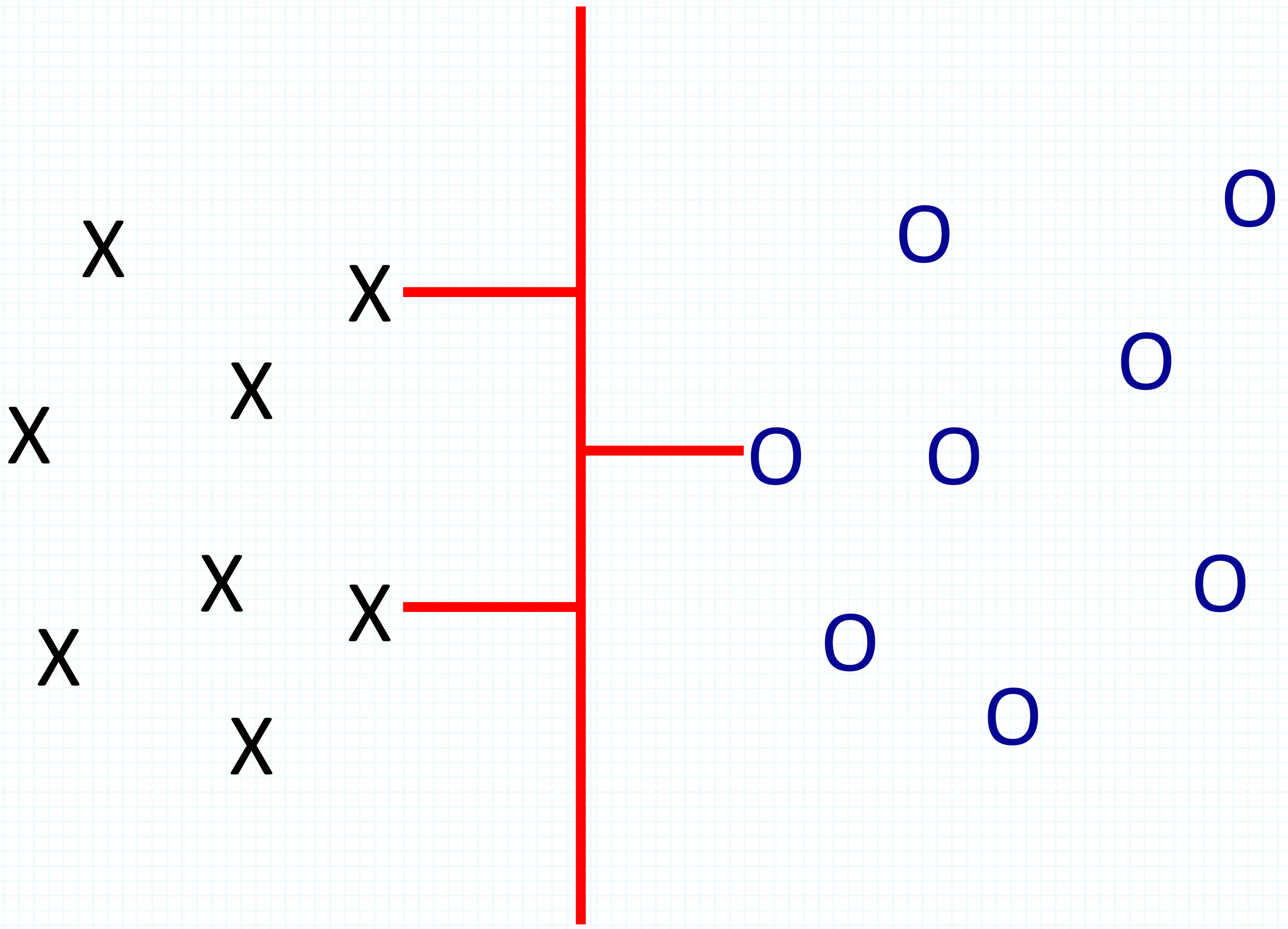
Ruling Out Some Separators



Lots of Noise

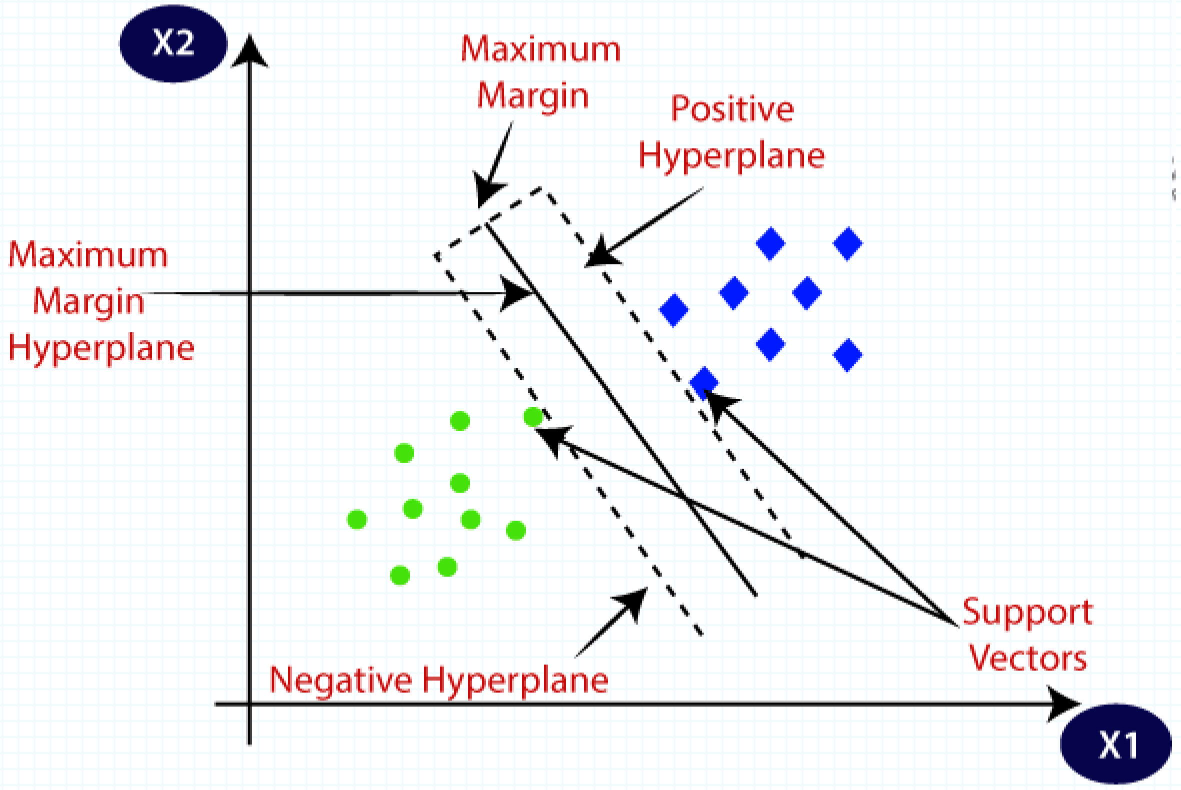


Maximizing the Margin



Terms

- **Support Vectors:**
  - These are the points that are closest to the hyperplane.
  - A separating line will be defined with the help of these data points.
- **Margin:**
  - It is the distance between the hyperplane and the observations closest to the hyperplane (support vectors).
  - In SVM large margin is considered a good margin.
  - There are two types of margins **hard margin** and **soft margin.**

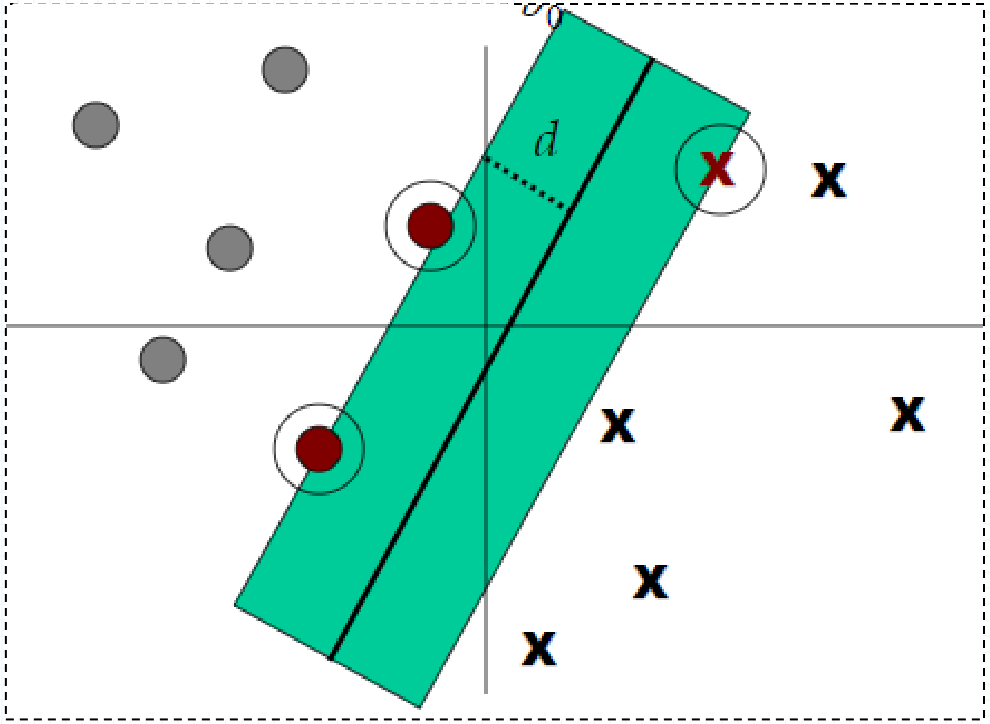


Intuition

- **Our problem:**

Maximizing the shortest distance to the closest positive or negative point.

$$w^* = arg_w max\,[min_n\, d_H(\phi(x_n))]$$



Note that W represents all parameters
i.e. w and b

So.. What is our optimization problem?

- **Our problem:**

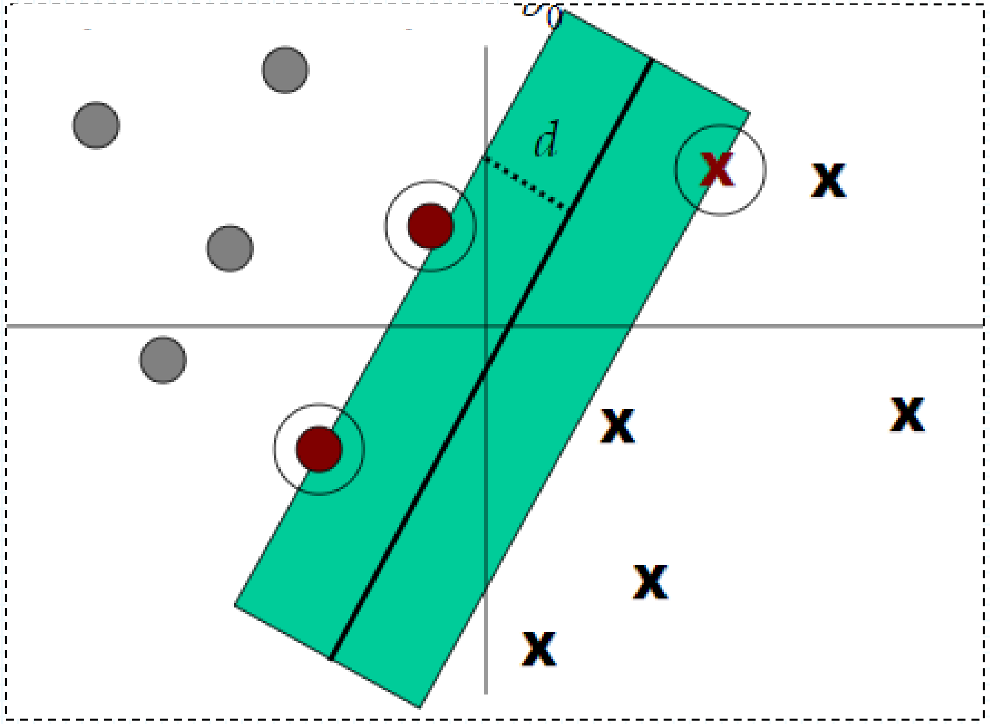
Maximizing the shortest distance to the closest positive or negative point.

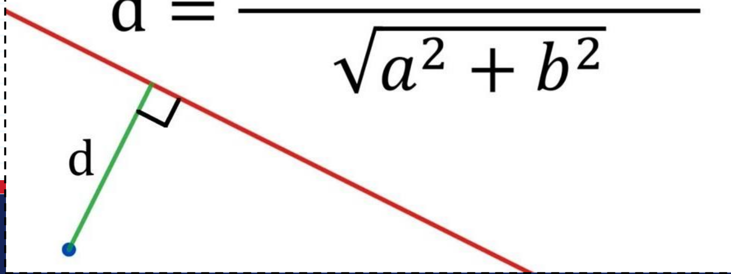
In our case

the distance of a hyperplane equation: $w^T\Phi(x) + b = 0$ from a given point vector $\Phi(x_0)$ can be easily written as :

$$d_H(\phi(x_0)) = \frac{|w^T(\phi(x_0)) + b|}{||w||_2}$$

$$||w||_2 =: \sqrt{w_1^2 + w_2^2 + w_3^2 + \dots w_n^2}$$





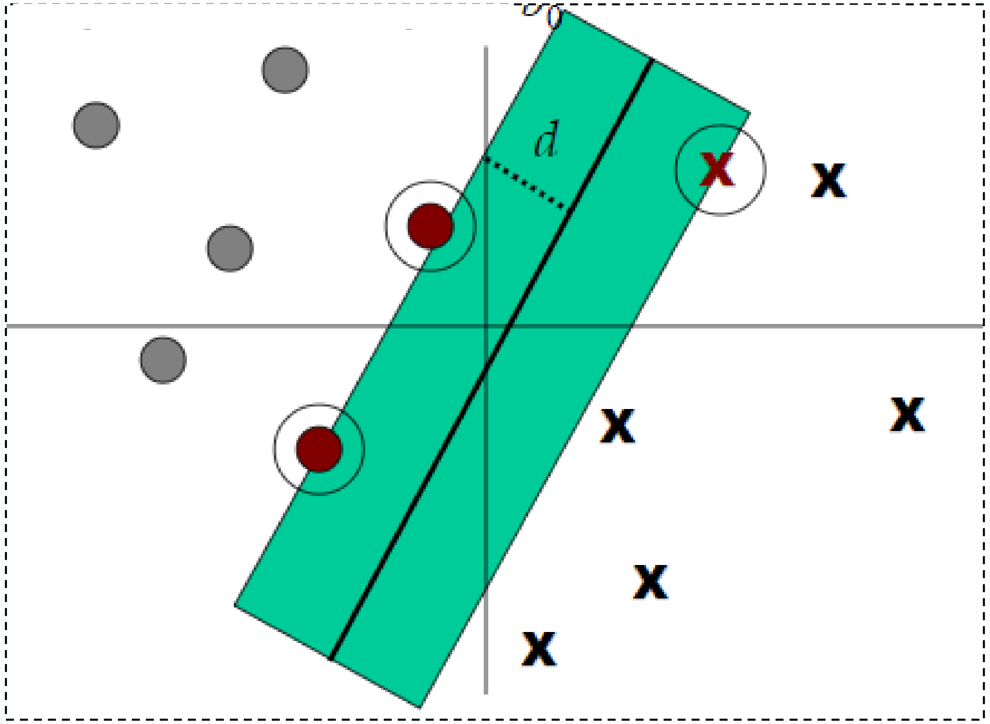
$$d = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

So.. What is our optimization problem?

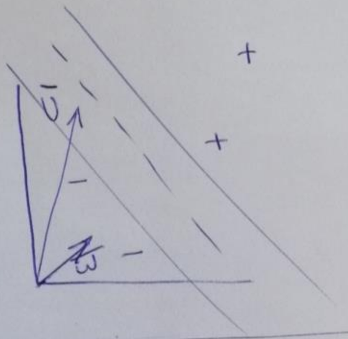
- **Our problem:**

Maximizing the shortest distance to the closest positive or negative point.

$$w^* = arg_w max\ [min_n\ d_H(\phi(x_n))]$$



Note that W represents all parameters i.e. w and b



$\bar{w}\cdot\bar{u} \geq c$ the N^+

$\bar{w}\cdot\bar{u} + b \geq \phi$

$\bar{w}\, x_+ + b \geq 1$

$w\, x_- + b \leq -1$

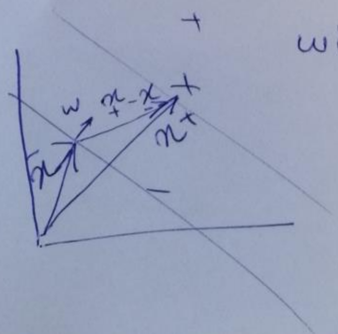
$y_i = +1$ for + samples

-1 for − samples

constrait

$\boxed{y_i(\bar{w}\cdot\bar{x}_i + b) \geq 1}$ for all points

(*) $y_i(\bar{w}\bar{x}_i + b) = 1$ for support vector



width $= (\bar{x}_+ - \bar{x}_-)\cdot\frac{\bar{w}}{\|w\|}$

support vector

(*)

$= (\bar{x}_+\bar{w} - \bar{x}_-\bar{w})\frac{1}{\|w\|}$

from (*)

$x_+ \rightarrow \bar{w}x_+ + b = 1 \rightarrow \bar{w}x_+ = 1 - b$

$x_- \rightarrow \bar{w}x_- + b = -1 \rightarrow -\bar{w}x_- = 1 + b$

$\Downarrow$

width $= \frac{2}{\|w\|}$

Goal Maximize width $\frac{2}{\|w\|}$

$\Downarrow$

Minimize $\|w\|$

$\Downarrow$

Goal $\boxed{\text{Minimize } \frac{1}{2}\|w\|^2}$

true only if the constrait is satisfied

$\Downarrow$

use lagrange Multiplier

$L = \frac{1}{2}\|\bar{w}\|^2 - \sum \alpha_i\left[y_i(\bar{w}\bar{x}_i + b) - 1\right]$

SVM Optimization

$$w^*, b^* = \arg\underset{w,b}{Min}\frac{1}{2}\|w\|^2\ ,\quad s.t.\ \ y_n\,(w^T\big(\emptyset(x_n)\big) + b) \geq 1 \quad \forall n$$

Solved by Lagrange multiplier method:

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 \ - \sum_n \alpha_n[y_n\big(w^T\big(\emptyset(x_n)\big) + b\big) - 1]$$

where $\boldsymbol{\alpha}$ is the Lagrange multiplier

The optimization problem can be solved by setting **derivatives** of *Lagrangian* to 0

SVM Optimization

$$L(w,b,\alpha) = \frac{1}{2}\|w\|^2 \; - \sum_n \alpha_n [y_n(w^T(\emptyset(x_n)) + b) - 1]$$

$$\begin{cases} \dfrac{\partial L}{\partial w} = w \; - \sum_n \alpha_n y_n \emptyset(x_n) \quad = 0 \;\; \Rightarrow \;\; w = \sum_n \alpha_n y_n \emptyset(x_n) \\ \dfrac{\partial L}{\partial b} = \sum_n \alpha_n y_n = 0 \;\; \Rightarrow \sum_n \alpha_n y_n \; = 0 \end{cases}$$

SVM Optimization

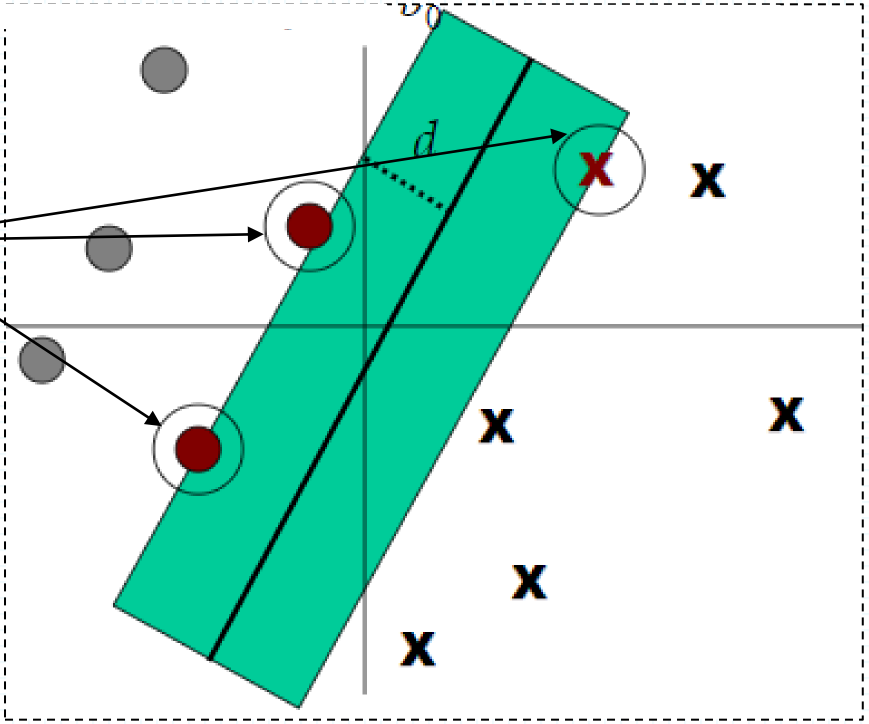
$$w^*, b^* = \arg\underset{w,b}{Min} \frac{1}{2}\|w\|^2, \quad s.t. \;\; y_n(w^T(\emptyset(x_n)) + b) \geq 1 \quad \forall n$$

$$Y = w^T(\emptyset(x)) + b = \sum_n \alpha_n y_n \emptyset^T(x_n)\, \emptyset(x)$$

The decision rule in SVMs only depends on the dot product with support vectors

several important implications

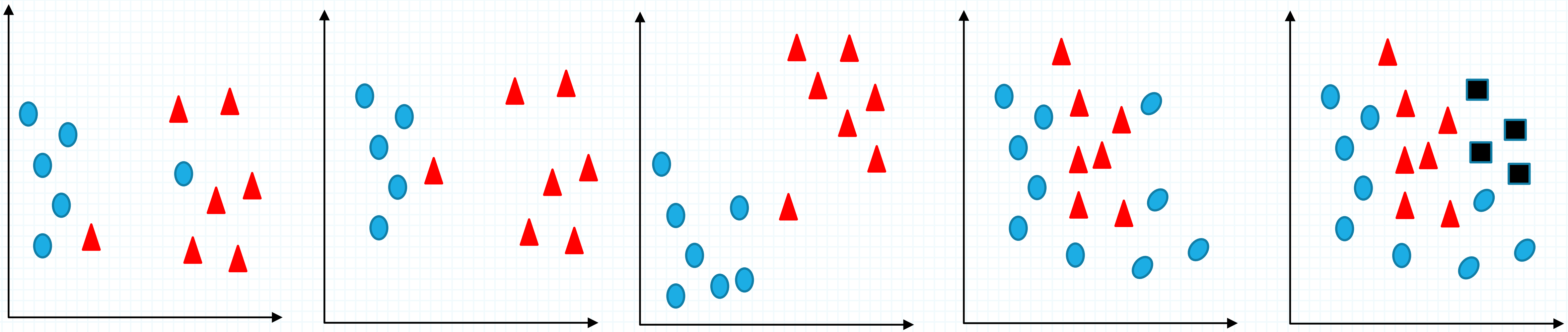
Computational efficiency
Memory efficiency
Robustness to noise and outliers



What if?

What are the problems of the current version for SVM?

(A) (B) (C) (D) (E)



1st Improvement
Soft Margin SVM
(allows few misclassifications)

Soft Margin SVM

- In real-life applications we don't find any dataset which is linearly separable, what we'll find is either an almost linearly separable dataset or a non-linearly separable dataset
- To tackle this problem what we do is modify that equation in such a way that it allows few misclassifications that means it allows few points to be wrongly classified.

$$\text{argmin}\left(\mathrm{w}^*, \mathrm{b}^*\right) \frac{\|\mathrm{w}\|}{2} + c\sum_{i=1}^{n} \zeta_i$$

- For all the ***correctly classified*** points our **zeta** will be equal to 0 and for all the ***incorrectly classified*** points the **zeta** is simply one or the distance of that particular point (misclassification error) from its correct hyperplane

Soft Margin SVM

$$\operatorname{argmin}\left(w^*, b^*\right) \frac{\|w\|}{2} + c\sum_{i=1}^{n} \zeta_i$$

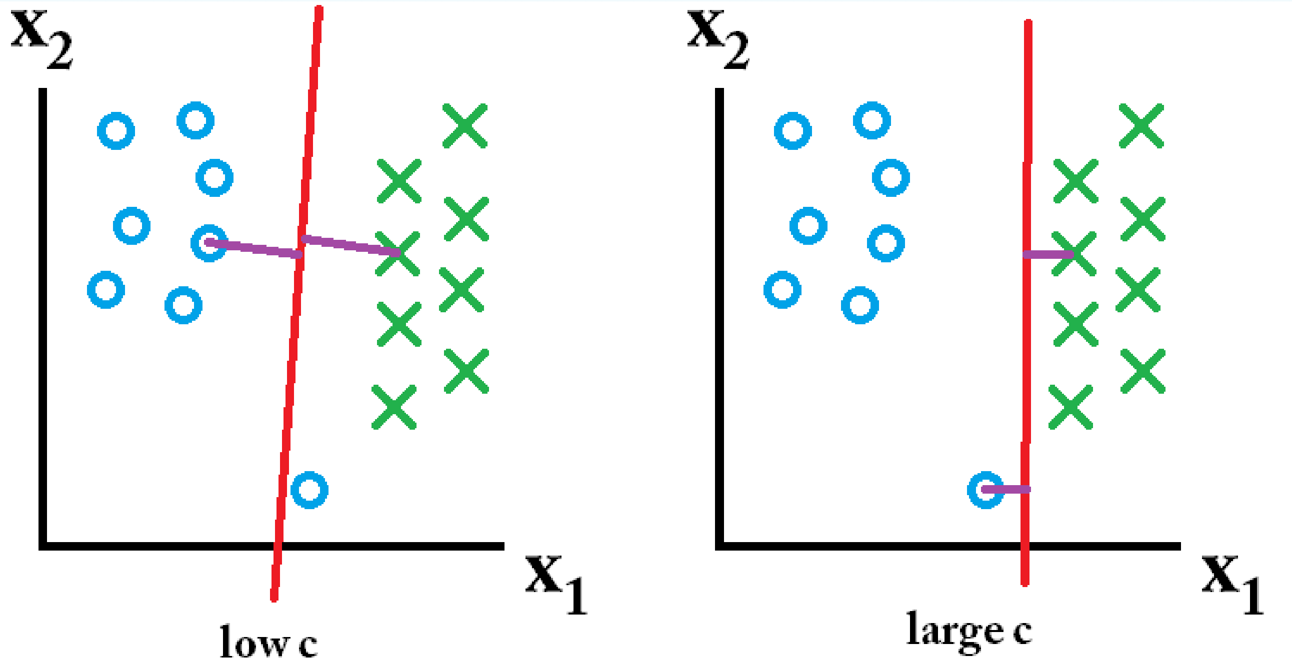
- So now, we can say: ***SVM Error = Margin Error + Classification Error***. The higher the margin, the lower would-be margin error, and vice versa.
- Let's say you take a **high value of 'c'** =1000, this would mean that you <u>don't want to focus on margin error</u> and just want a model which <u>doesn't misclassify any data point</u>

C Hyper-parameter

- When **C** is high it will classify all the data points correctly, also there is a chance to overfit.

$$\operatorname{argmin}\left(w^*, b^*\right) \frac{\|w\|}{2} + c\sum_{i=1}^{n} \zeta_i$$

- ***SVM Error = Margin Error + Classification Error***

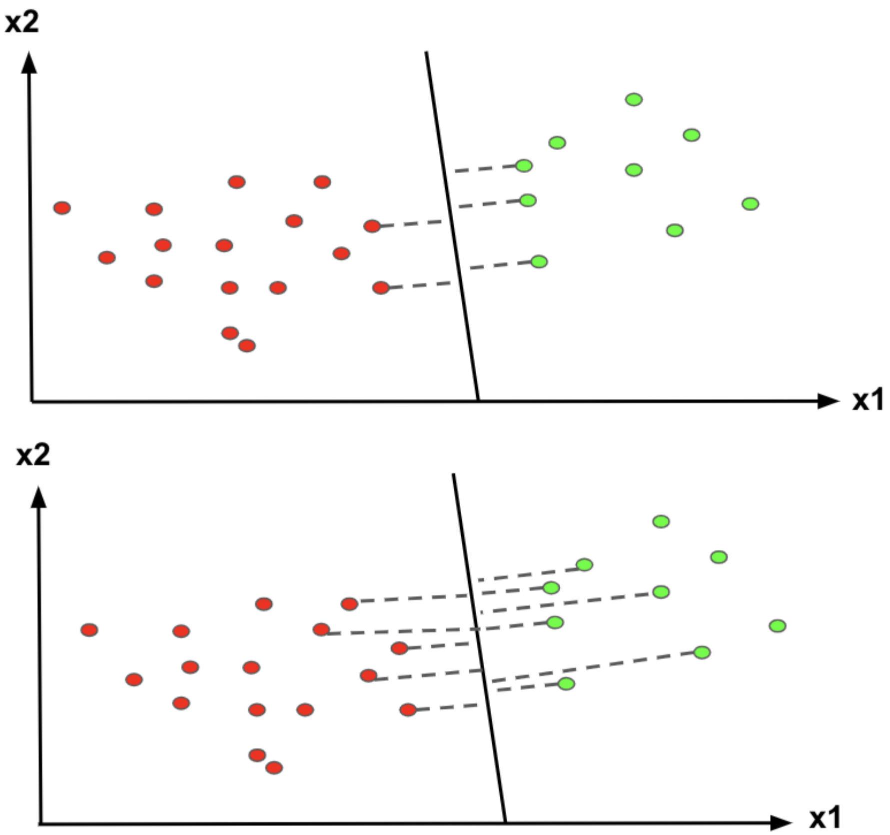


2nd Improvement

Consider more points to get the decision boundary.

Gamma Hyper-Parameters

- It defines how far influences the calculation of plausible line of separation.
- when gamma is higher, nearby points will have high influence; low gamma means far away points also be considered to get the decision boundary.



High Gamma

- only near points are considered.

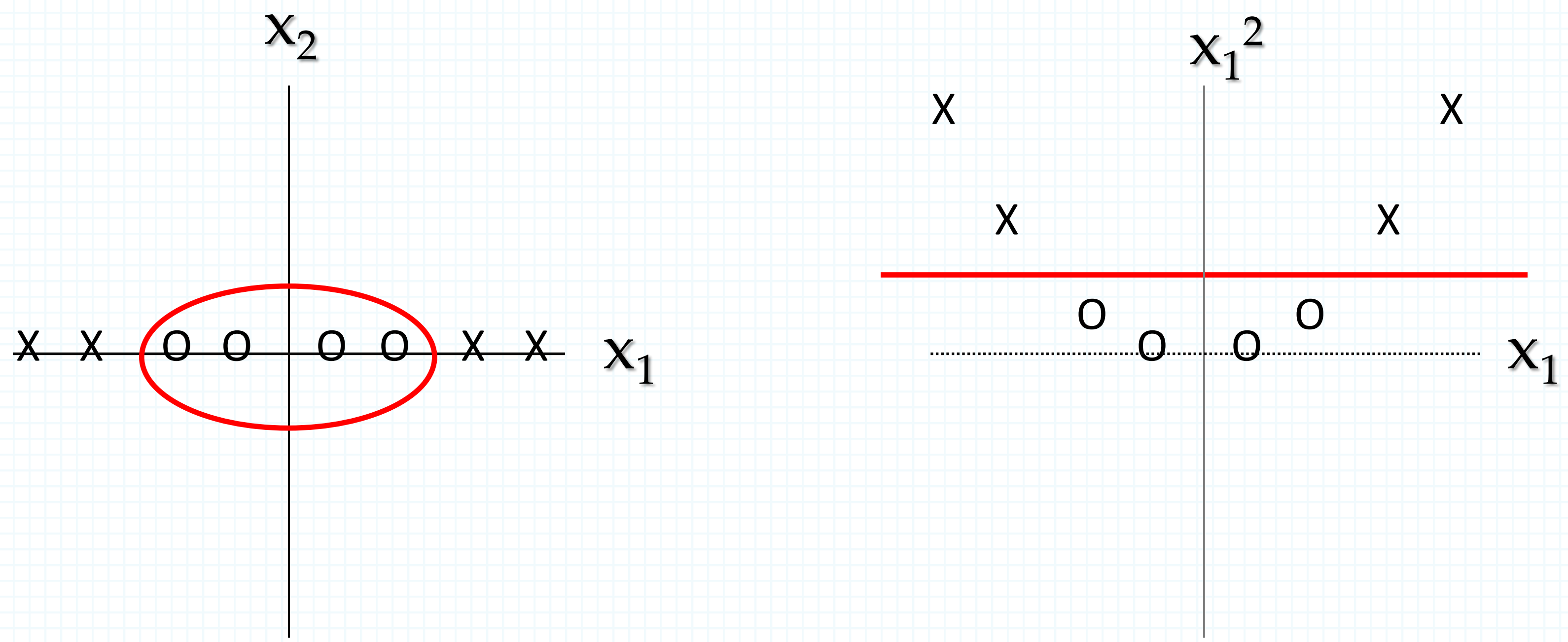
Low Gamma

- far away points are also considered

3rd Improvement
How to make a plane curved?

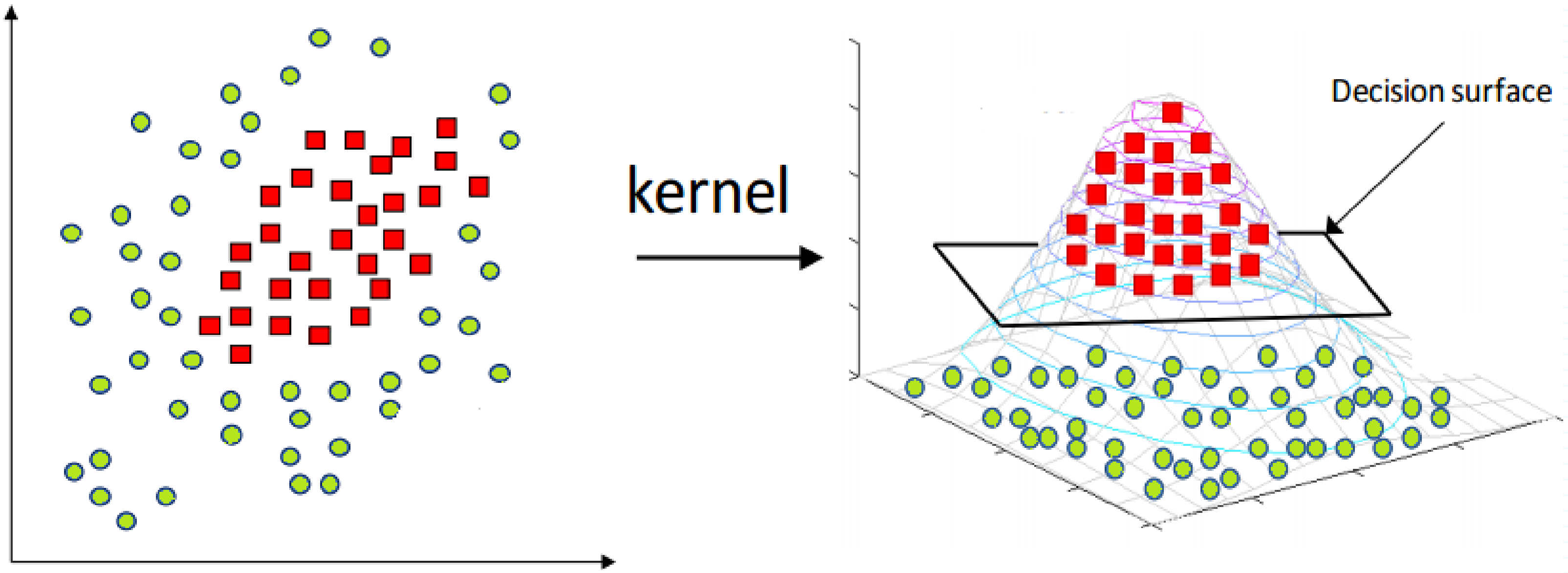
When Linear Separators Fail

- The most interesting feature of SVM is that it can even work with a non-linear dataset.
- We use “Kernel Trick” which makes it easier to classifies the points.



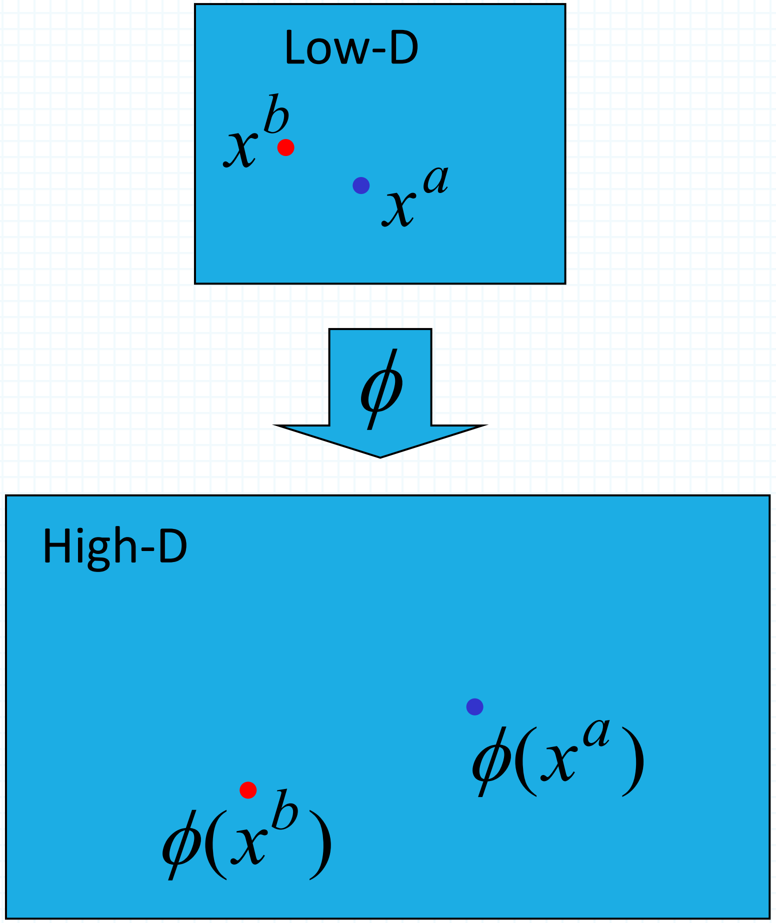
The Kernel Trick

- try converting this lower dimension space to a higher dimension space using some quadratic functions which will allow us to find a decision boundary that clearly divides the data points.
- These functions which help us do this are called Kernels and which kernel to use is purely determined by hyperparameter tuning.



The Kernel Trick

- For many mappings from a low-D space to a high-D space, there is a simple operation on two vectors in the low-D space that can be used to compute the scalar product of their two images in the high-D space.



$$K(x^a, x^b) = \phi(x^a) . \phi(x^b)$$

Letting the kernel do the work

doing the scalar product in the obvious way

Kernel functions: Polynomial kernel

- Following is the formula for the polynomial kernel:

$$f(X1, X2) = (X1^T . X2 + 1)^d$$

- Here d is the degree of the polynomial, which we need to specify manually.
- Suppose we have two features X1 and X2 and output variable as Y, so using polynomial kernel we can write it as:

$$X1^T . X2 = \begin{bmatrix} X1 \\ X2 \end{bmatrix} . \begin{bmatrix} X1 & X2 \end{bmatrix}$$

$$= \begin{bmatrix} X1^2 & X1 . X2 \\ X1 . X2 & X2^2 \end{bmatrix}$$

- So we basically need to find $X1^2$, $X2^2$ and X1.X2, and now we can see that 2 dimensions got converted into 5 dimensions.

Other commonly used kernels

- linear: $\langle x, x' \rangle$.
- polynomial: $(\gamma \langle x, x' \rangle + r)^d$, where d is specified by parameter `degree`, r by `coef0`.
- rbf: $\exp(-\gamma \|x - x'\|^2)$, where γ is specified by parameter `gamma`, must be greater than 0.
- sigmoid $\tanh(\gamma \langle x, x' \rangle + r)$, where r is specified by `coef0`.

How to choose the right Kernel?

- Choosing a kernel totally depends on what kind of dataset are you working on.
- You can start with a hypothesis that your data is linearly separable and choose a linear kernel function.
- Once you have established it is a problem requiring a non-linear model, the **Radial Basis Function kernel** makes a good default kernel

4th Improvement

Dealing with multiclass (more than 2 possible classes)

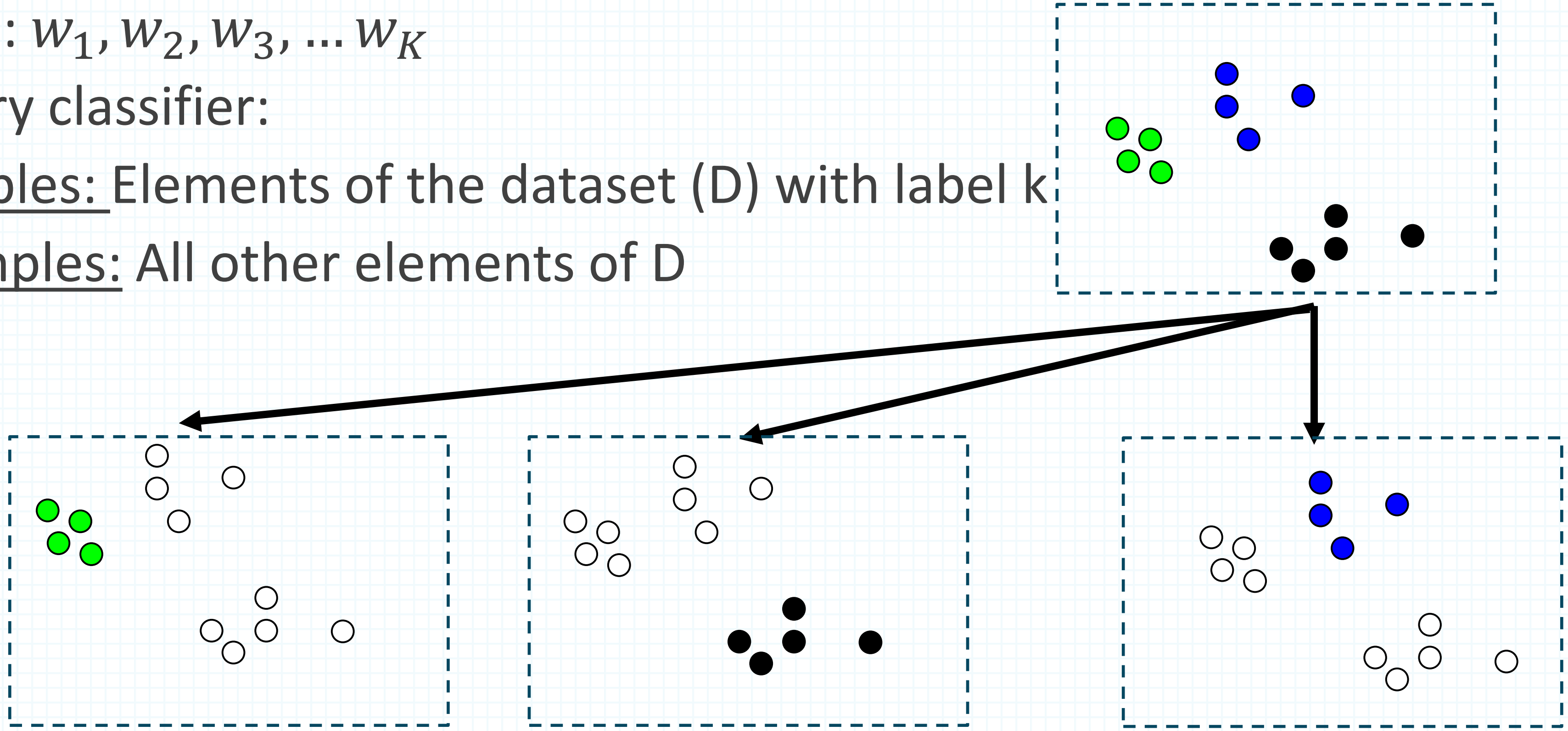
Multiclass classification

- Any suggested solutions?

One-against-all & One-vs-one

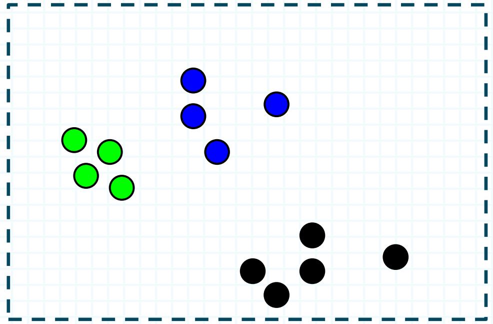
One against All learning

- Decompose into binary problems
- IF $y_i \in \{1,2,3,\ldots K\}$, Decompose into K binary classification tasks
  - Learn K models: $w_1, w_2, w_3, \ldots w_K$
  - For the K[th] binary classifier:
    - Positive examples: Elements of the dataset (D) with label k
    - Negative examples: All other elements of D



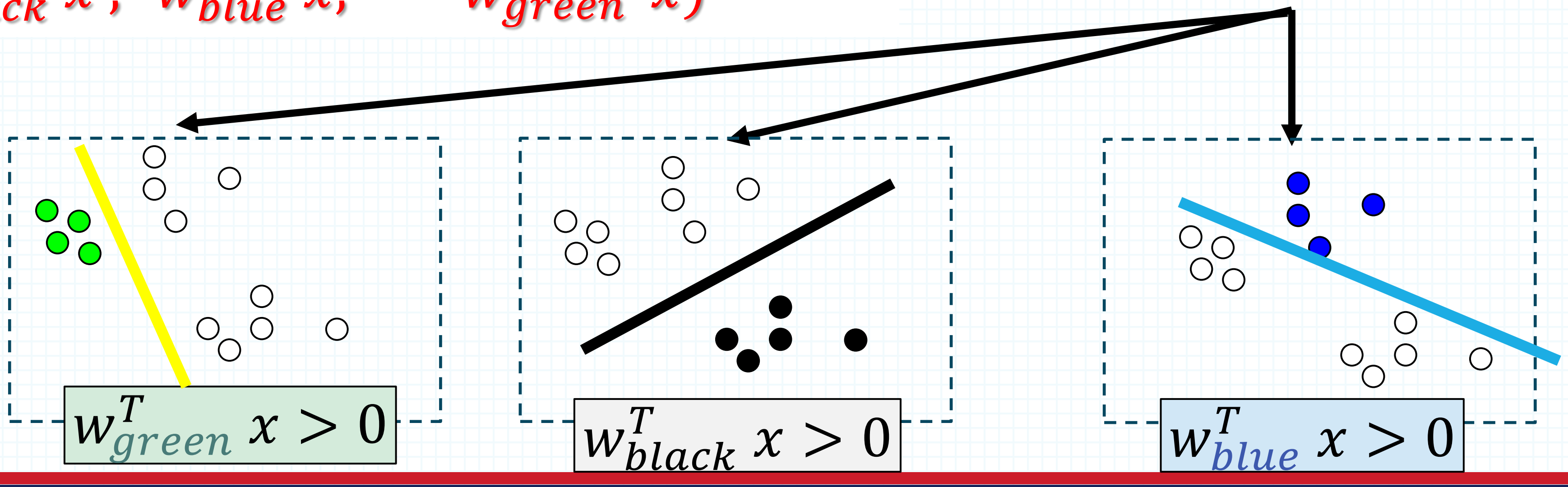
One against All learning

- Decompose into binary problems
- IF $y_i \in \{1,2,3,\ldots K\}$, Decompose into K binary classification tasks
  - Learn K models: $w_1, w_2, w_3, \ldots w_K$



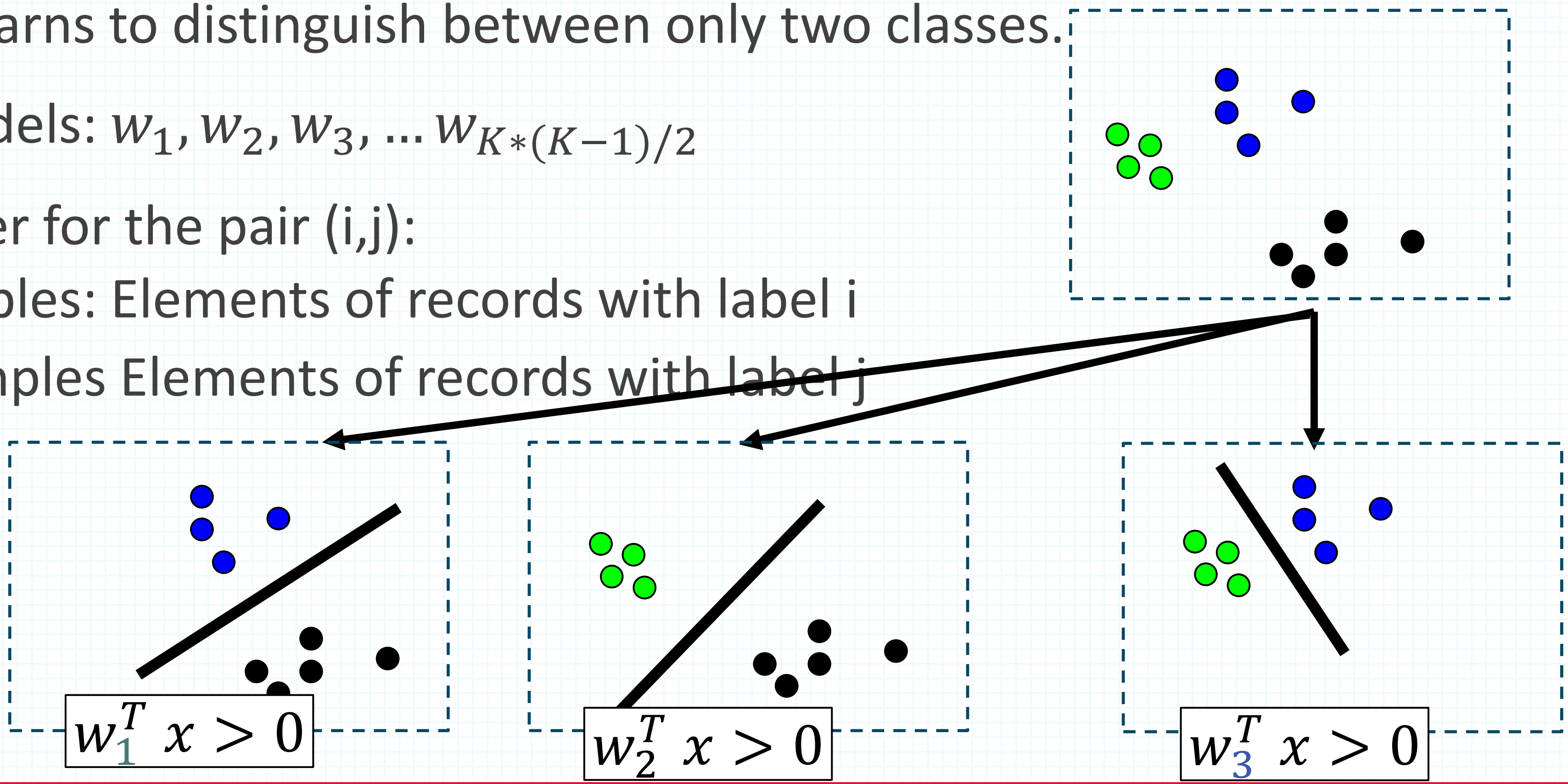
But what if not only the correct label will have a positive score?

$$y = \operatorname{argmax}(w_{black}^T\, x\,,\ \ w_{blue}^T\, x, \qquad w_{green}^T\ \ x)$$



One v.s. One learning

- IF $y_i \in \{1,2,3, \dots K\}$, Decompose into C(K,2) binary classifier i.e. $K * (K-1)/2$ classifier
- Each classifier learns to distinguish between only two classes.
- Learn C(K,2) models: $w_1, w_2, w_3, \dots w_{K*(K-1)/2}$
- For each classifier for the pair (i,j):
  - Positive examples: Elements of records with label i
  - Negative examples Elements of records with label j



One v.s. One learning

For example, if k=4 (say classes A, B, C, D), we train classifiers for:

- A vs B
- A vs C
- A vs D
- B vs C
- B vs D
- C vs D

But how do we decide the final class for a new input?

<u>(Solution 1) Majority Voting:</u>

✓ For a test example x, each binary classifier casts **one vote** for the class it predicted.

✓ Then, **count how many votes each class got**, and assign x to the class with the most votes. (note that Label i gets k-1 votes)

Example:

A vs B → A wins | A vs C → C wins | A vs D → D wins
B vs C → B wins | B vs D → D wins | C vs D → D wins

Majority: D win

One v.s. One learning

For example, if k=4 (say classes A, B, C, D), we train classifiers for:

- A vs B
- A vs C
- A vs D
- B vs C
- B vs D
- C vs D

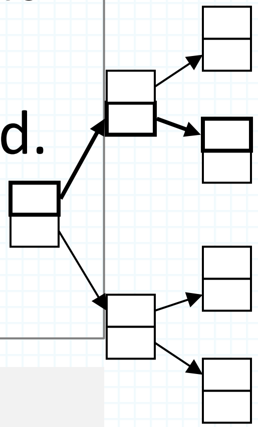
But how do we decide the final class for a new input?

(Solution 2) Tournament Style Decision:

✓ You organize all classes into **pairs**, have each pair "fight" using the corresponding classifier.

✓ Winners move to the next round, losers are eliminated.

✓ Repeat until **one winner remains** — that's your predicted class.



Example:

Round 1: A vs B → A wins | C vs D → D wins
Round 2: A vs D → D wins

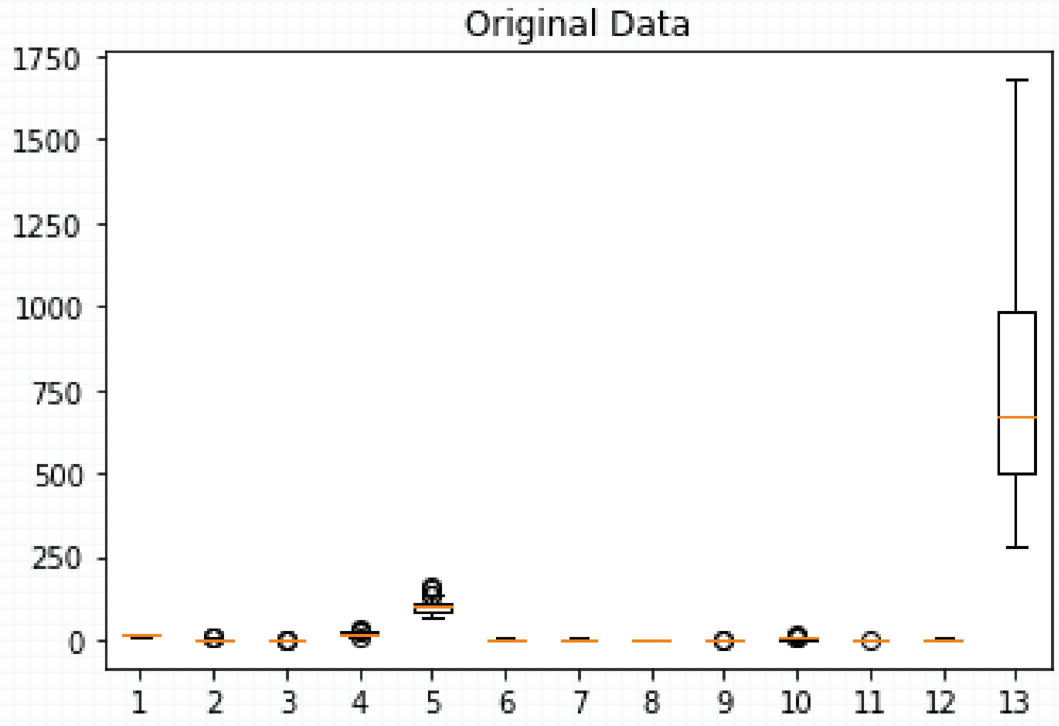
Final prediction = D

5^{th} Improvement

Feature Scaling

Why feature scaling?

- Why?
- Any suggestion?



Feature scaling

- **Feature scaling** is mapping the feature values of a dataset into the same range.
- The two most widely adopted approaches for feature scaling are normalization and standardization.
- Normalization maps the values into the [0, 1] interval:

$$z = \frac{x - min(x)}{max(x) - min(x)}$$

- Standardization shifts the feature values to have a mean of zero, then maps them into a range such that they have a standard deviation of 1:

$$z = \frac{x - \mu}{\sigma}$$

  - It centers the data, and it’s more flexible to new values that are not yet seen in the dataset. That’s why we prefer standardization in general.

Feature scaling