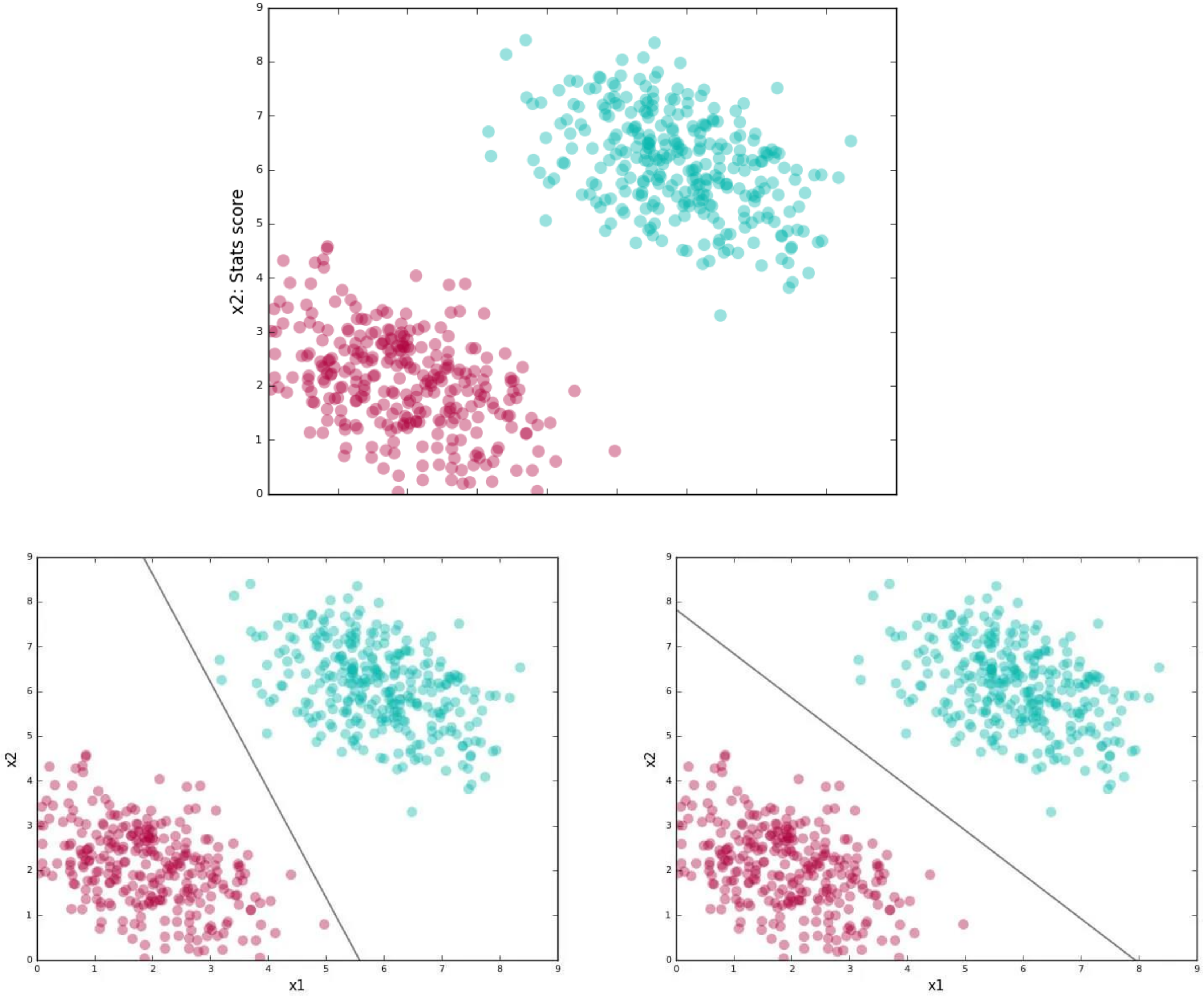
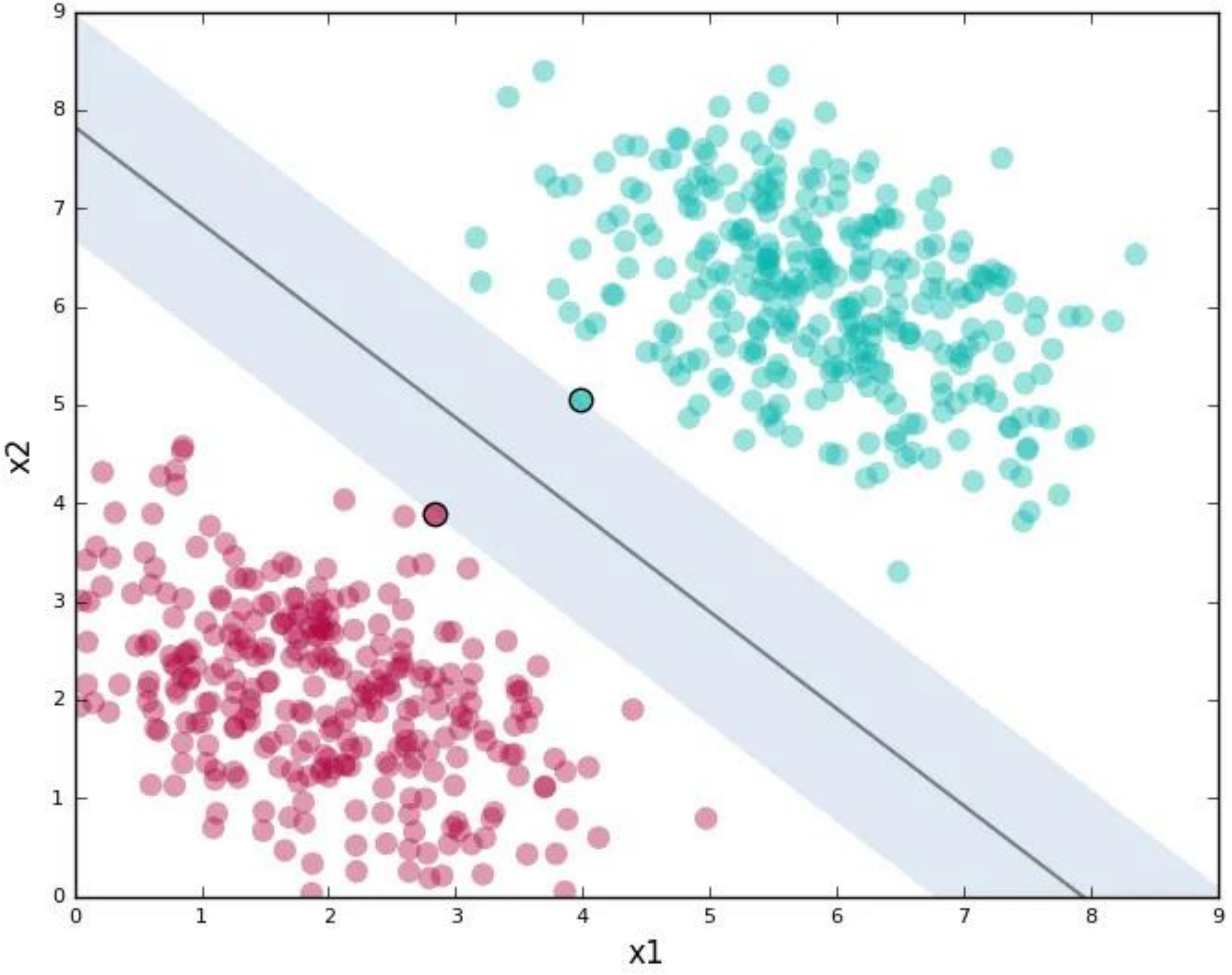
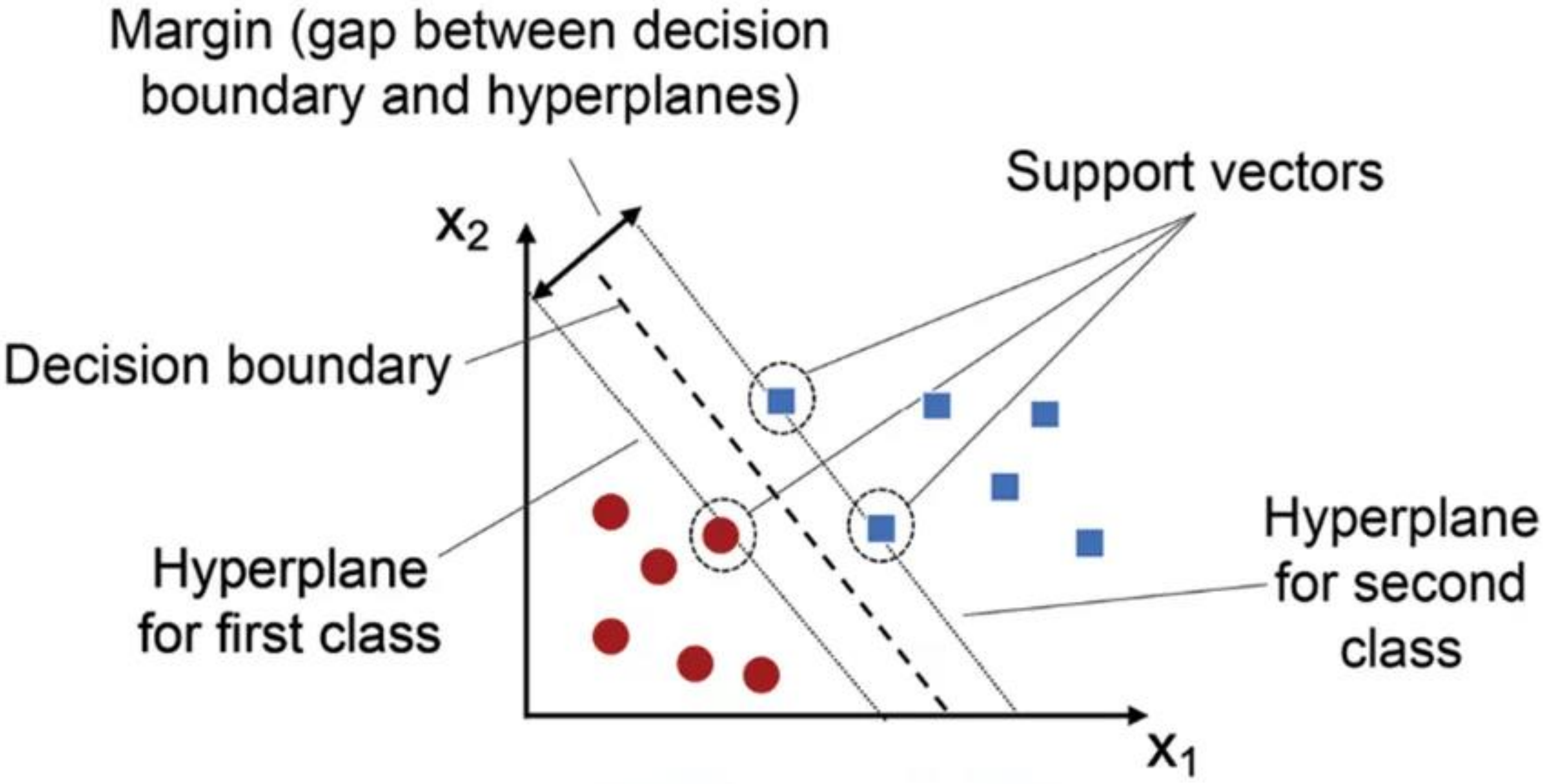
Support Vector Machine (SVM)


x2: Stats score
x2
x1
x2
x1

The underlying philosophy of SVM classifiers



- Find lines (hyperplanes) that correctly classify the training data
- Among all such lines (hyperplanes), pick the one that has the greatest distance to the points closest to it.



Margin (gap between decision boundary and hyperplanes)
Support vectors
x_2
Decision boundary
Hyperplane for first class
Hyperplane for second class
x_1

“Decision Boundary”

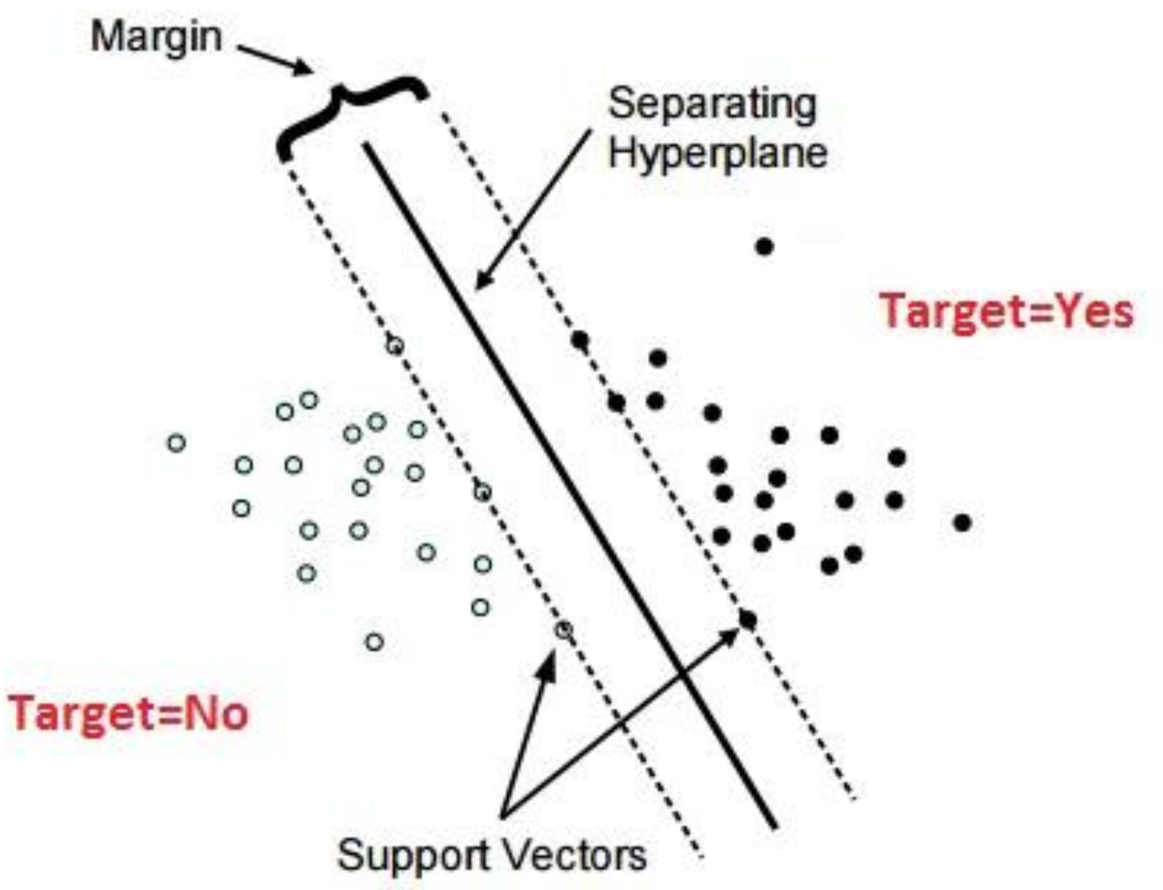
A linear classifier’s decision boundary is also calle
a “hyperplane”. Hyperplane is a (n-1)-dimensiona
subspace, for example, a line for observations in
2D space and a plane for observations in a 3D
space, separating different classes of data
observed in a n-dimensional space.

“Margin”

Margin is defined as the perpendicular distance of the nearest data points to the hyperplane. There are “hard margin” and “soft margin”. Our objective is to find the hyperplane with the maximum margin.

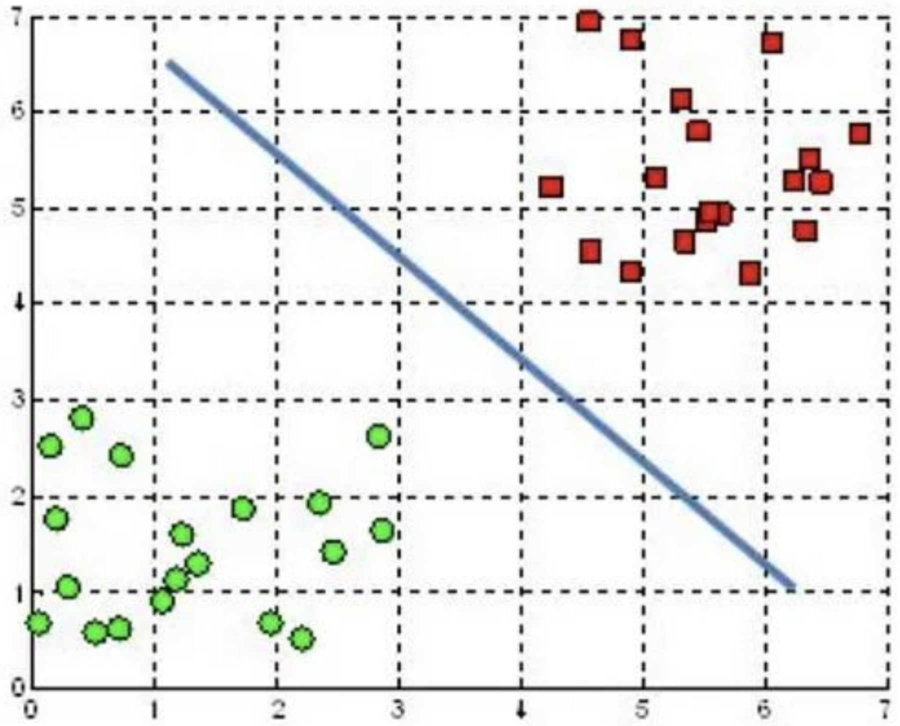
“Support Vector”

The nearest data points on the edge of margins are support vectors.

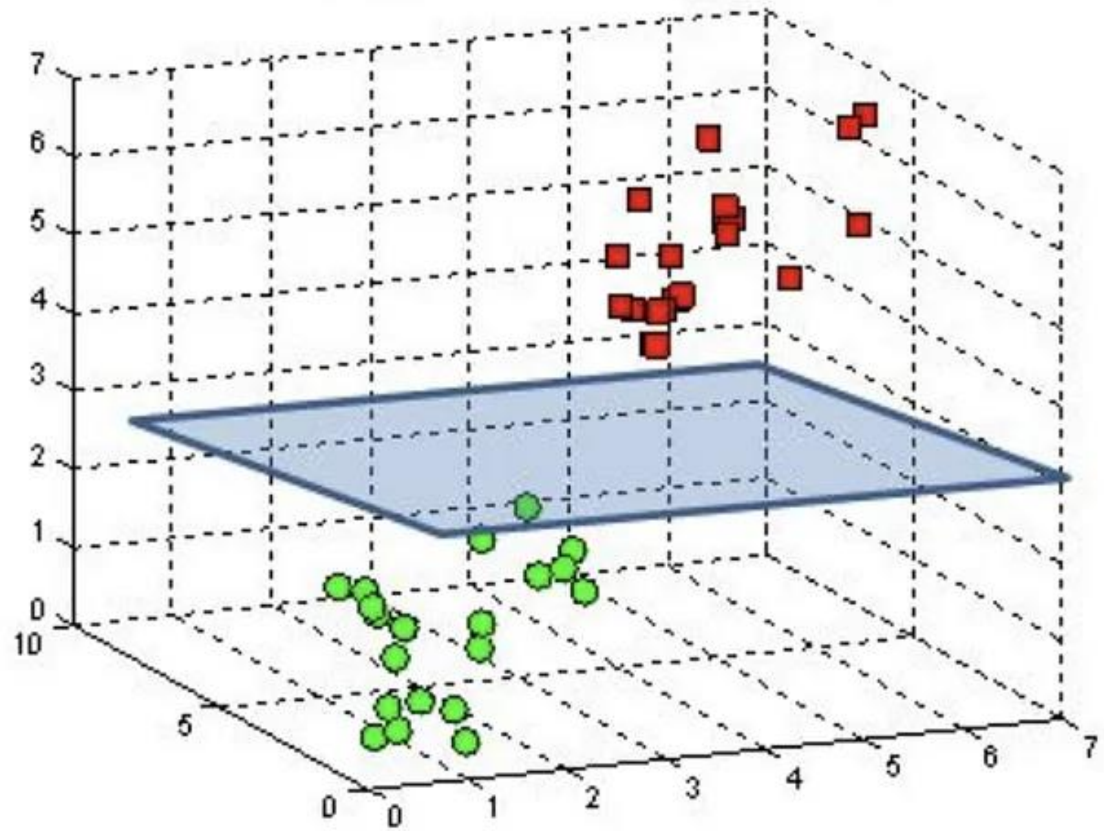


Support Vector Machine

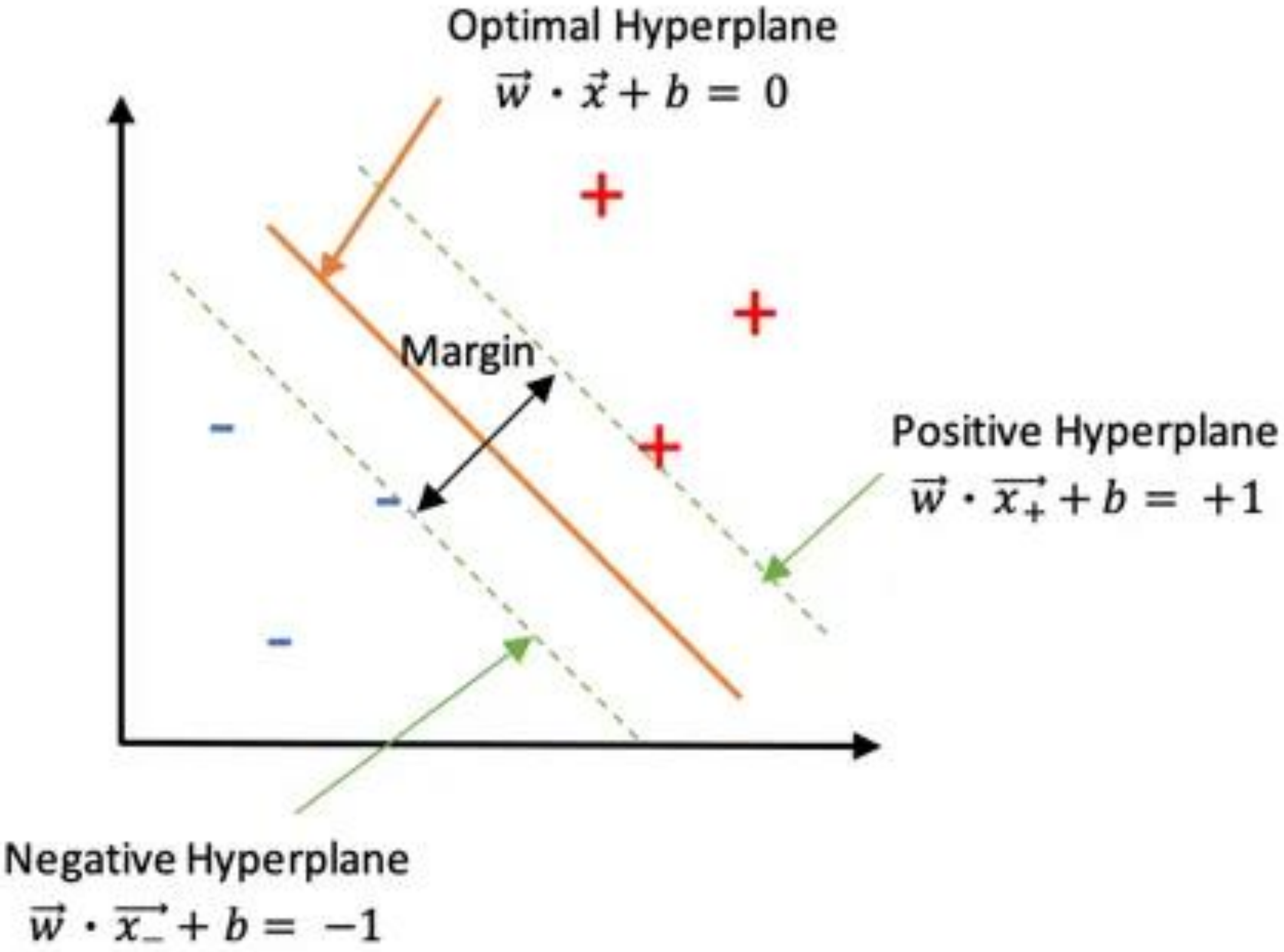
A hyperplane in $\mathbb{R}^2$ is a line



A hyperplane in $\mathbb{R}^3$ is a plane



Optimal hard margin classifier

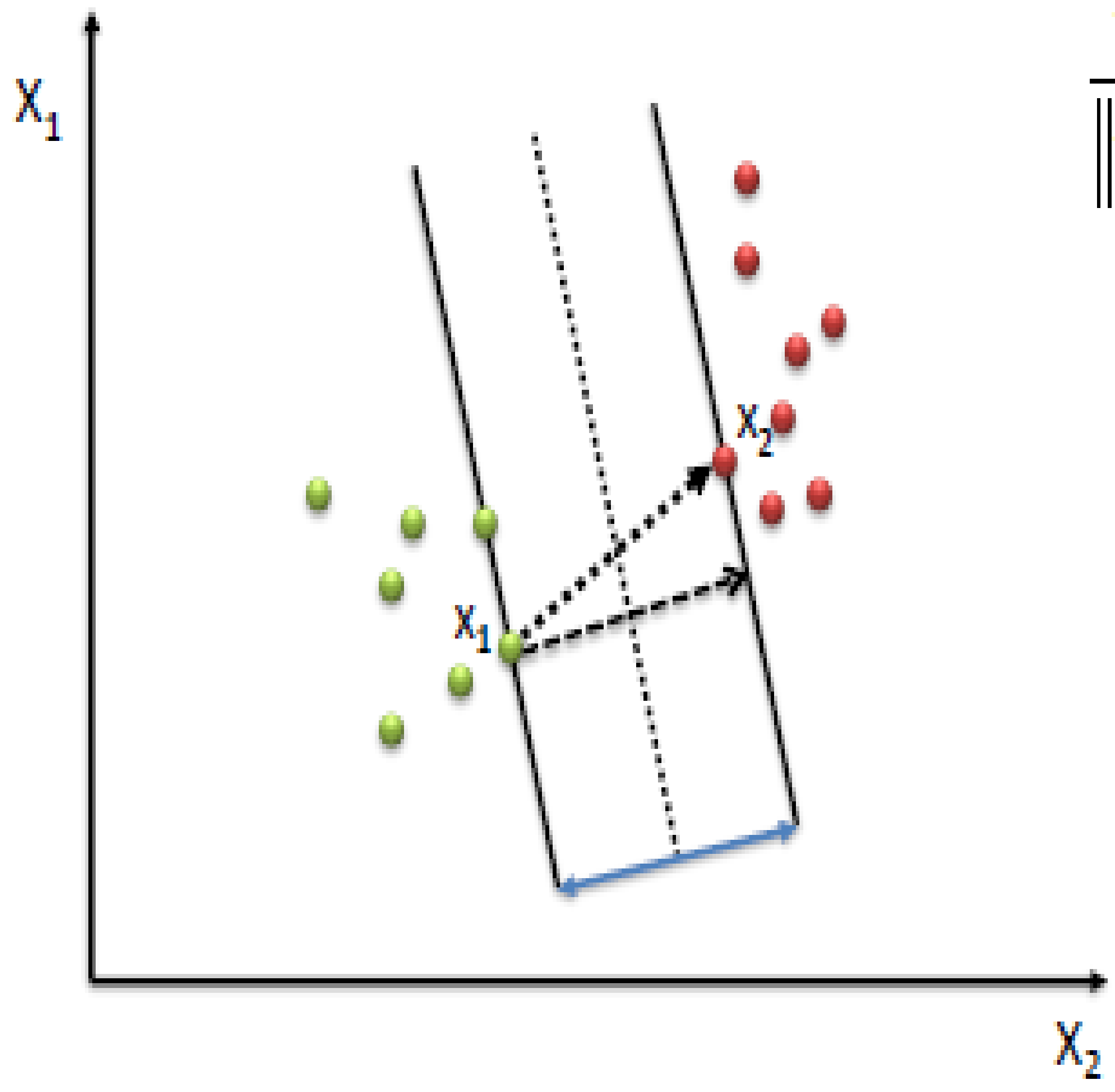


Geometric Intuition and basic understanding

- W (parameters of the model) is a vector (geometric) perpendicular to hyperplane $\mathbf{w^T X + b=0}$

- b is the offset of the hyperplane in vector space

- Our objective to train SVM is to find w and b which maximize the margin.



$$\frac{w}{\|w\|}\cdot(x_2 - x_1) = \text{width} = \frac{2}{\|w\|}$$

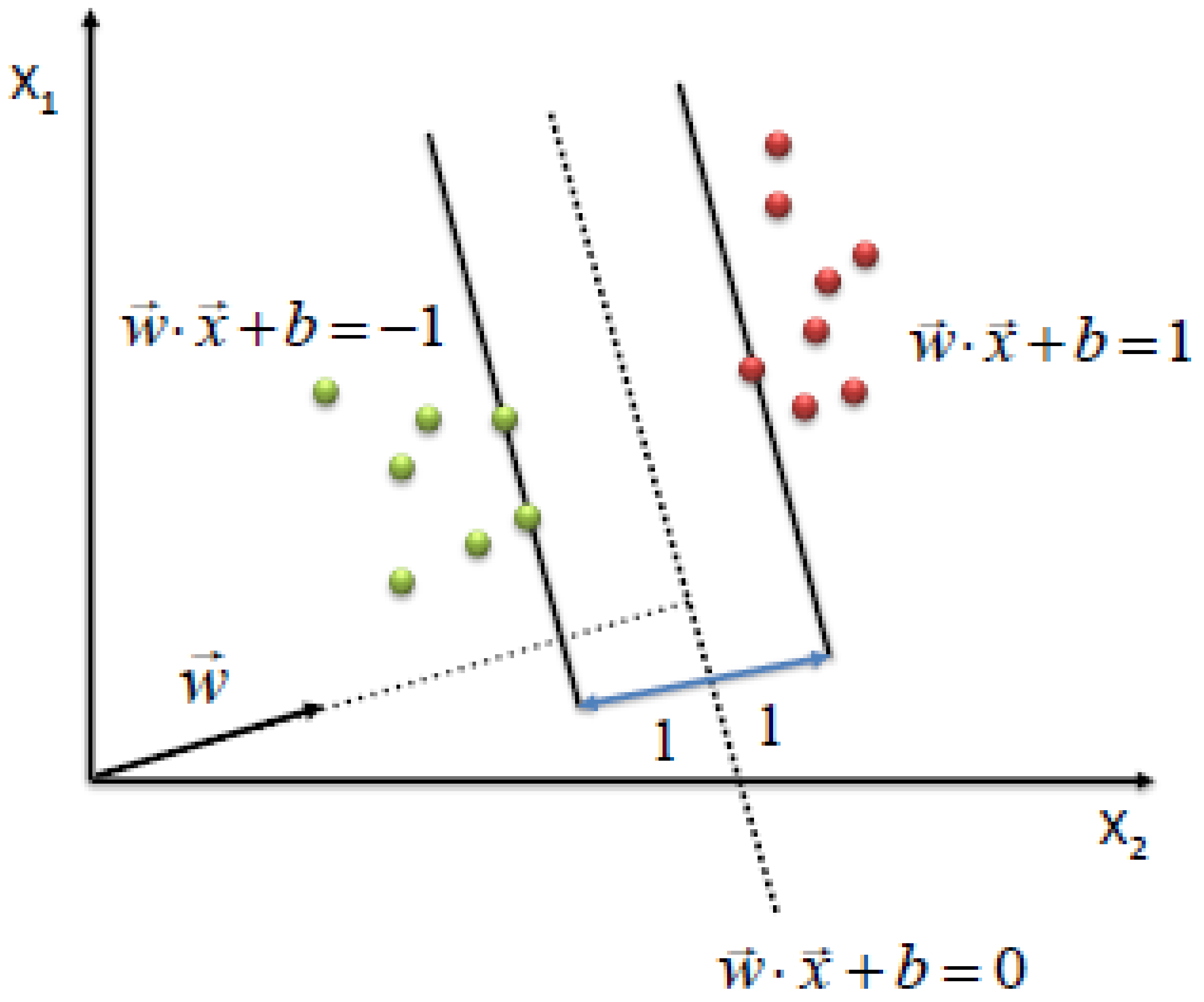
$$w\cdot x_2 + b = 1$$

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

$$w\cdot x_2 + b - w\cdot x_1 - b = 1-(-1)$$

$$w\cdot x_2 - w\cdot x_1 = 2$$

$$\frac{w}{\|w\|}(x_2 - x_1) = \frac{2}{\|w\|}$$



$$\boxed{\max \frac{2}{\|w\|}}$$

s.t.

$(w \cdot x + b) \geq 1, \forall x$ of class 1

$(w \cdot x + b) \leq -1, \forall x$ of class 2

Mathematical intuition of Support Vector Machine

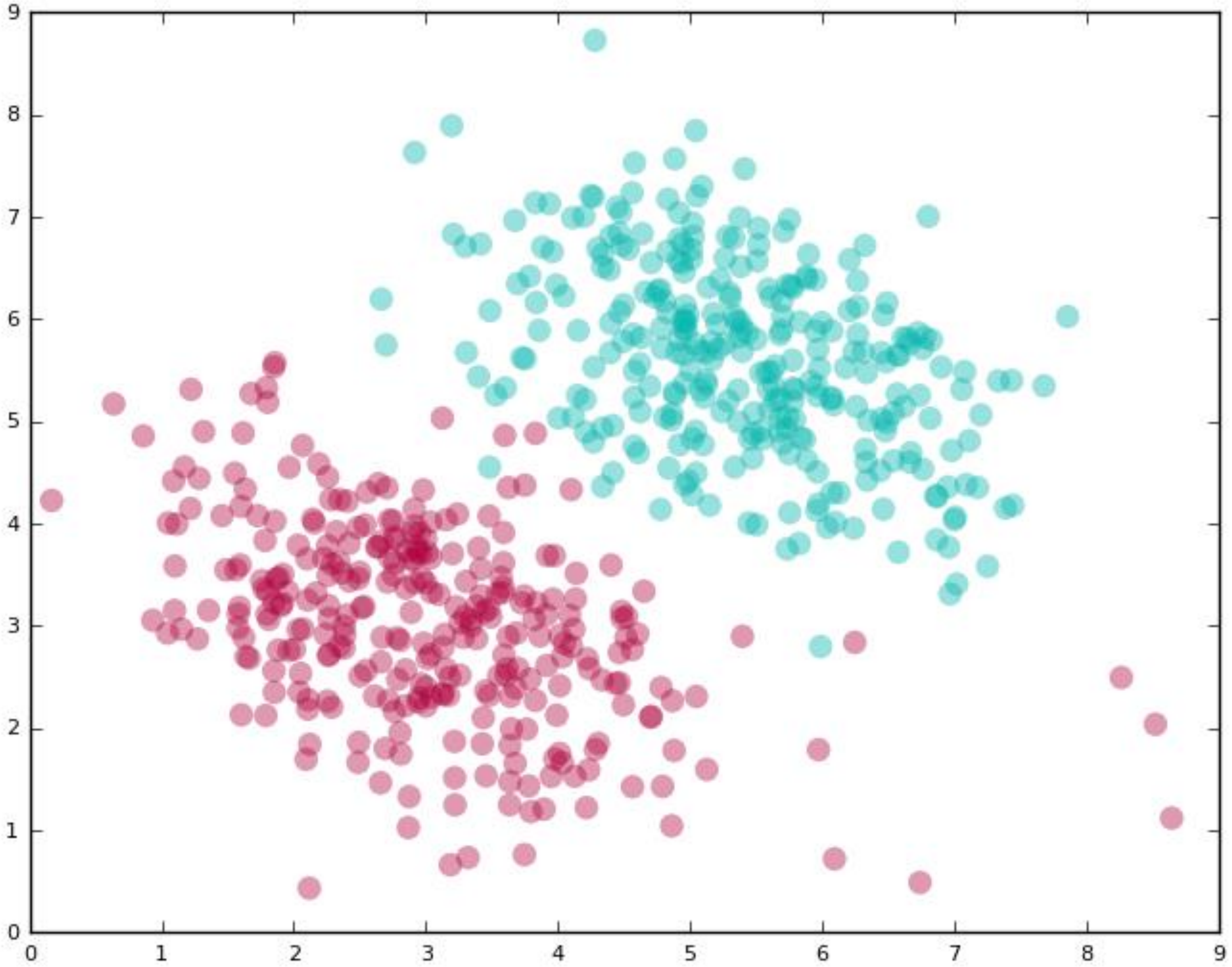
We find w and b by solving the following objective function using Quadratic Programming.

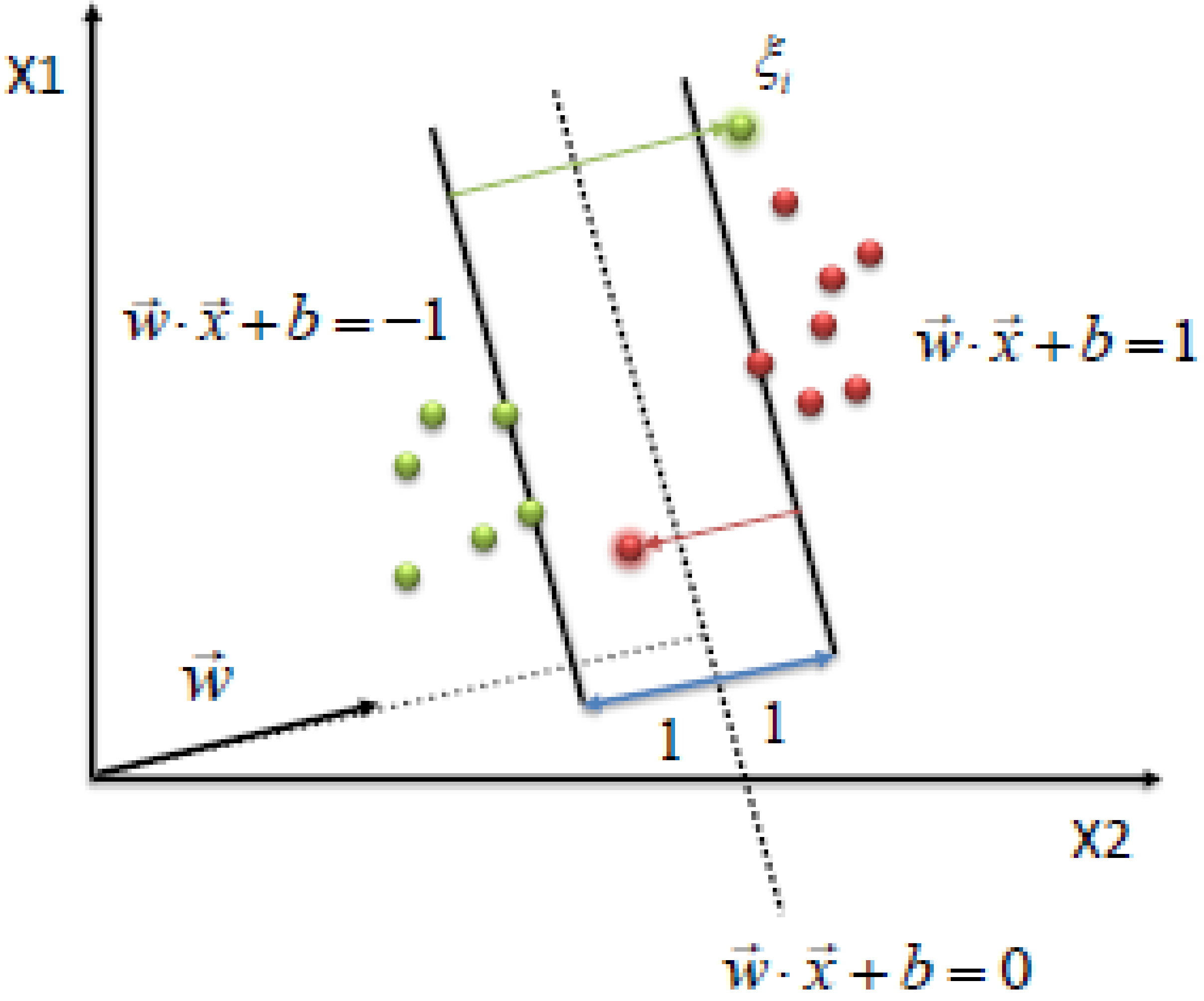
Constrained optimization problem

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

$$\textbf{subject to} \quad y_i \cdot (\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1, i = 1, \ldots, n$$

When a hard margin classifier is not applicable

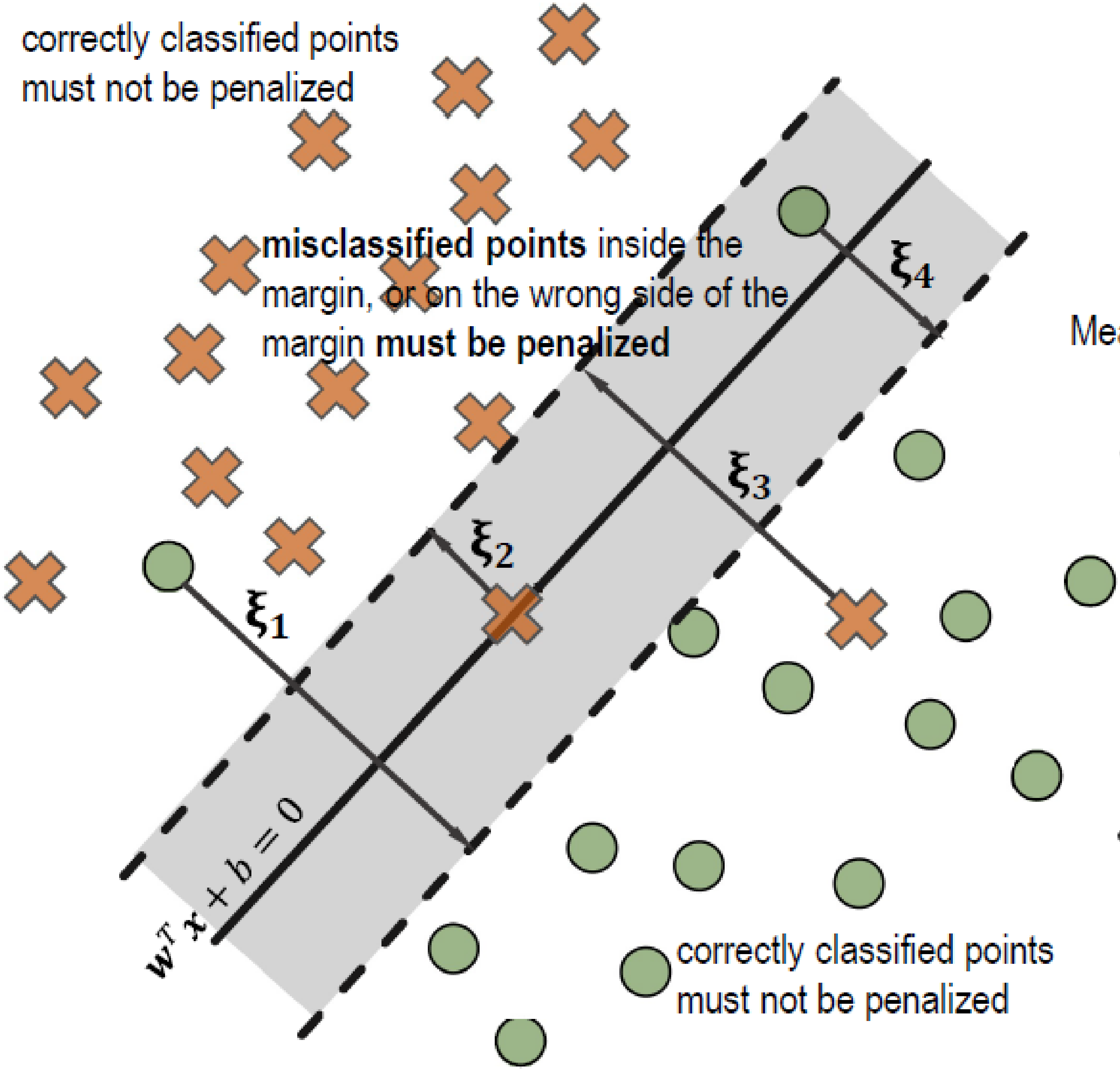




slack variable:

$$\xi_i$$

Allow some instances to fall off the margin, but penalize them



Measure the **misclassification error** for each training example

$$\xi_i = \begin{cases} 0, & y_i \cdot (\mathbf{w}^T \boldsymbol{x}_i + b) \geq 1 \\ y_i \cdot (\mathbf{w}^T \boldsymbol{x}_i + b) & y_i \cdot (\mathbf{w}^T \boldsymbol{x}_i + b) < 1 \end{cases}$$

*Penalize each **misclassification by the size of the violation**, using the **hinge loss** (contrast with the loss function of the Perceptron)*

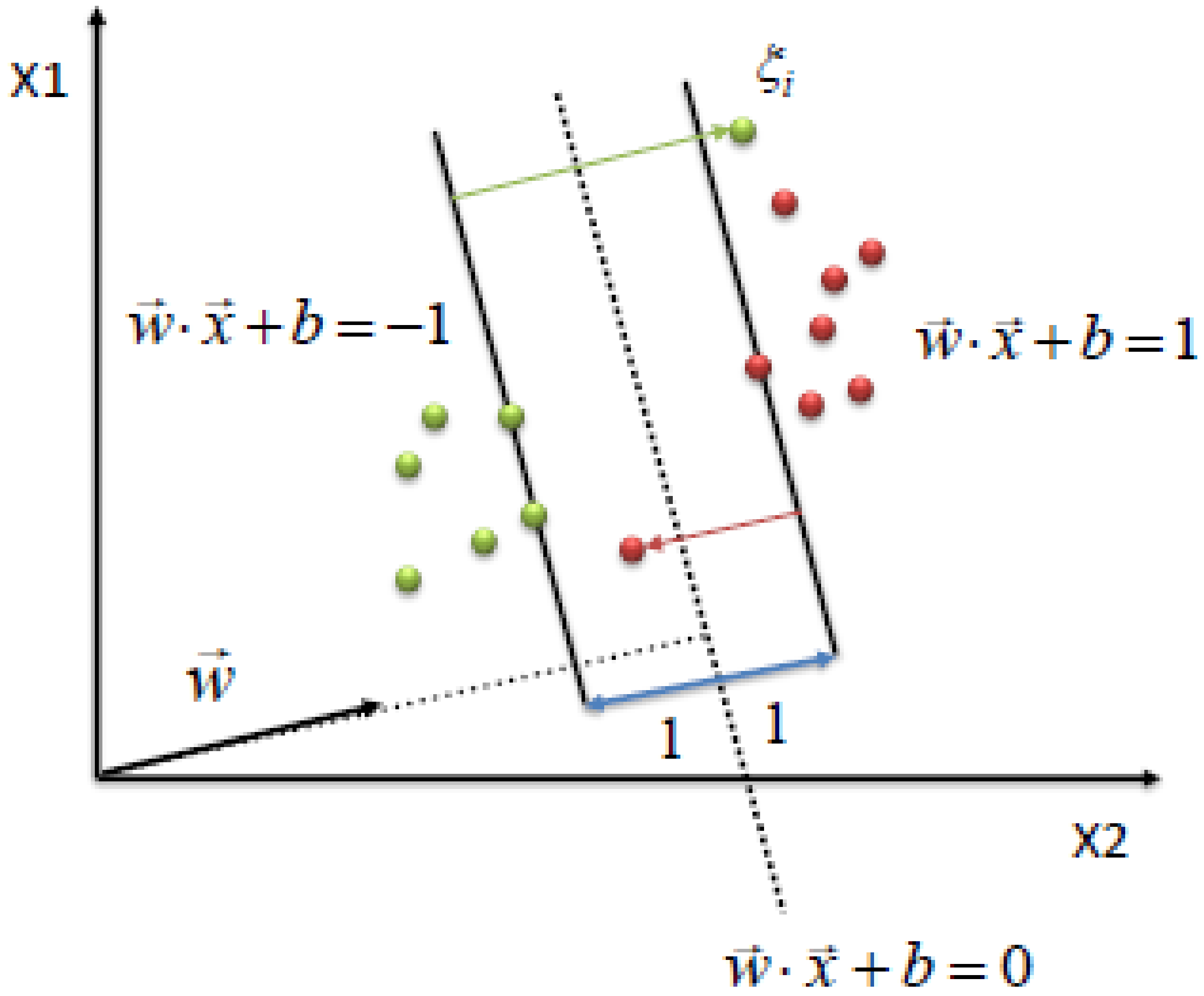
$$\xi_i = L(f(x_i), y_i) = \max\{0, 1 - y_i \cdot (\mathbf{w}^T \boldsymbol{x}_i + b)\}$$

Measure the **misclassification error** for each training example

$$\xi_i = \begin{cases} 0, & y_i \cdot (\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 \\ y_i \cdot (\boldsymbol{w}^T \boldsymbol{x}_i + b) & y_i \cdot (\boldsymbol{w}^T \boldsymbol{x}_i + b) < 1 \end{cases}$$

*Penalize each **misclassification by the size of the violation**, using the **hinge loss** (contrast with the loss function of the Perceptron)*

$$\xi_i = L(f(x_i), y_i) = \max\{0, 1 - y_i \cdot (\boldsymbol{w}^T \boldsymbol{x}_i + b)\}$$



Constraint becomes :

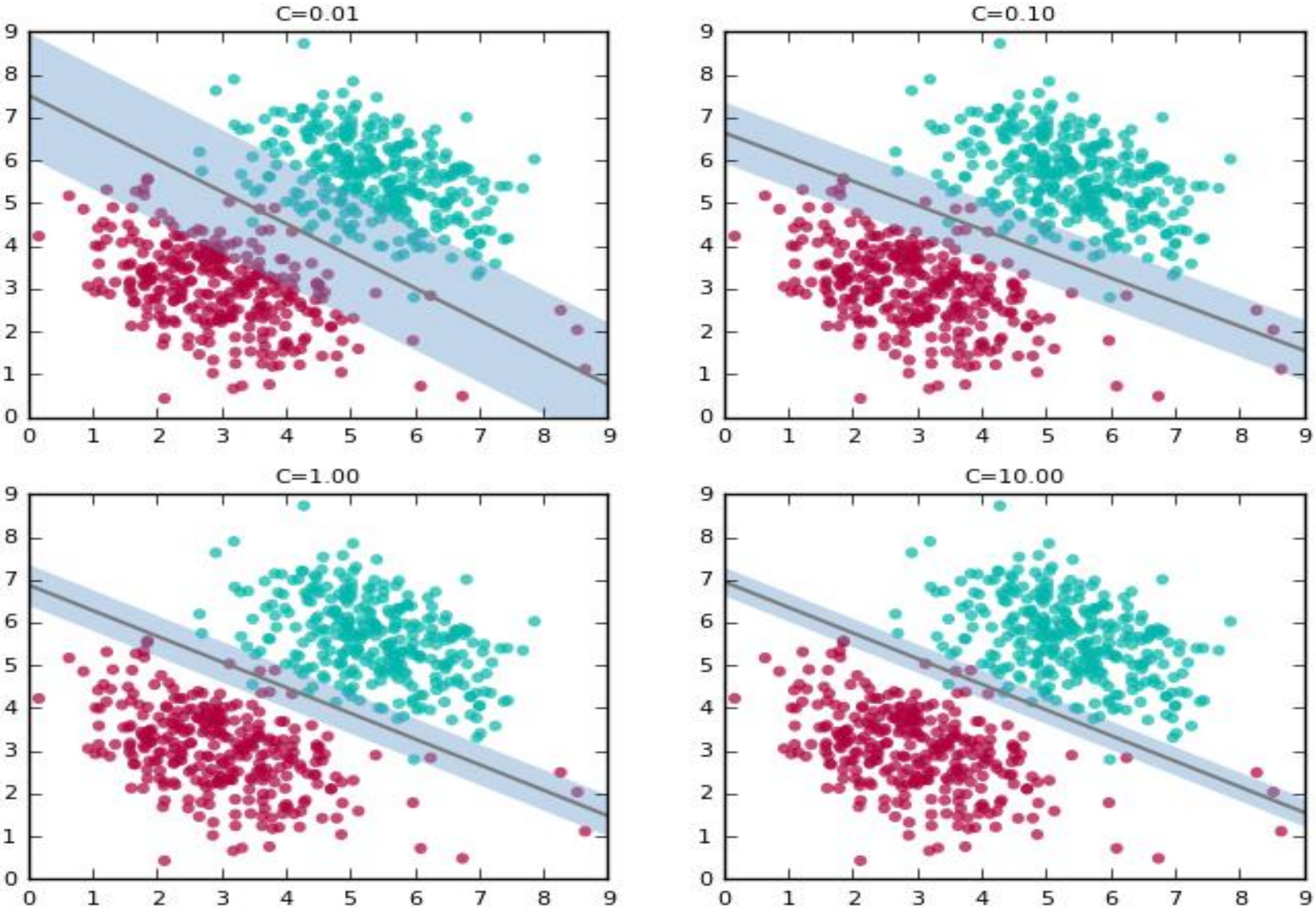
$$y_i(w\cdot x_i+b)\geq 1-\xi_i,\ \forall x_i$$

$$\xi_i \geq 0$$

Objective function penalizes for misclassified instances and those within the margin

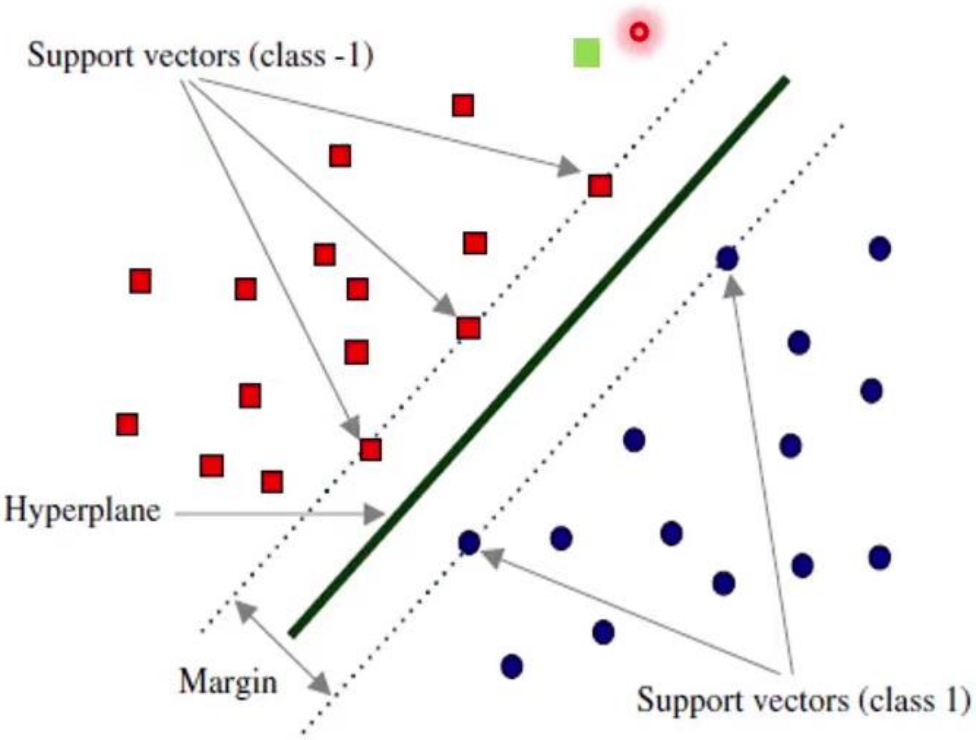
$$\min \frac{1}{2}\|w\|^2 + C\sum_i \xi_i$$

C trades-off margin width and misclassifications



C=0.01
C=0.10
C=1.00
C=10.00

Support Vector Machine Classifier



- Hyperplane
- Support Vectors
- Margin
- Linearly separable data

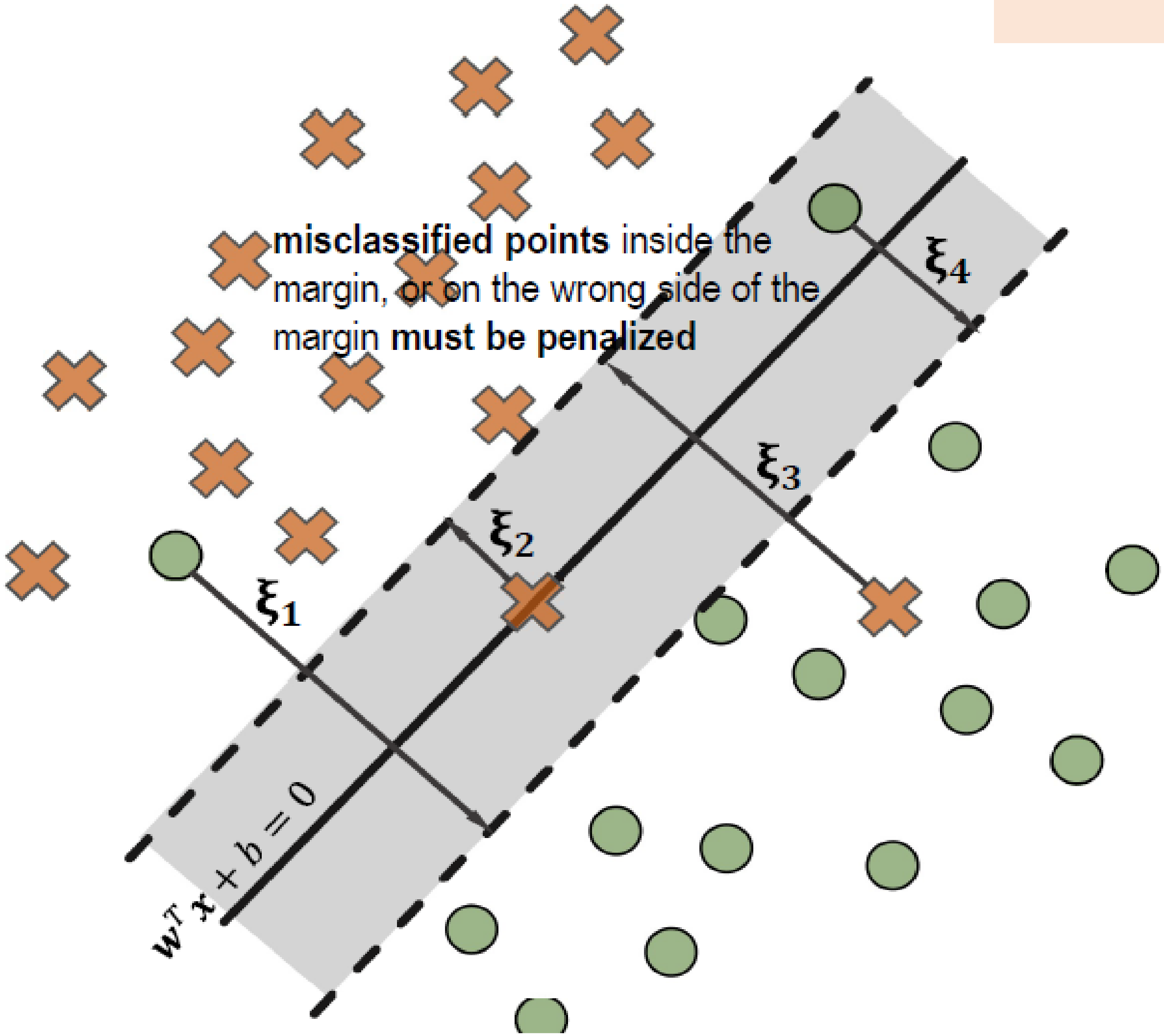
Sidd

Soft-Margin SVM: Primal Problem

Problem: Find a **linear classifier** $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$ with the largest margin such that

$$\mathbf{sign}(f(\boldsymbol{x})) = +1, \text{ when positive example}$$
$$\mathbf{sign}(f(\boldsymbol{x})) = -1, \text{ when negative example}$$

and **misclassifications are minimized**.



soft-margin support vector machine

$$\min \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$
$$\text{s.t.} \quad y_i(\mathbf{w}'\mathbf{x}_i - b) \geq 1 - \xi_i \quad \forall i = 1 \ldots n$$
$$\xi_i \geq 0$$

This model is called the soft-margin SVM as it is softens the classification constraints with slack variables (ξ_i) and **allows flexibility for misclassifications** by the model

hard-margin support vector machine

$$\min \quad \frac{1}{2}\|\mathbf{w}\|_2^2$$
$$\text{s.t.} \quad y_i(\mathbf{w}'\mathbf{x}_i - b) \geq 1 \quad \forall i = 1 \ldots n$$

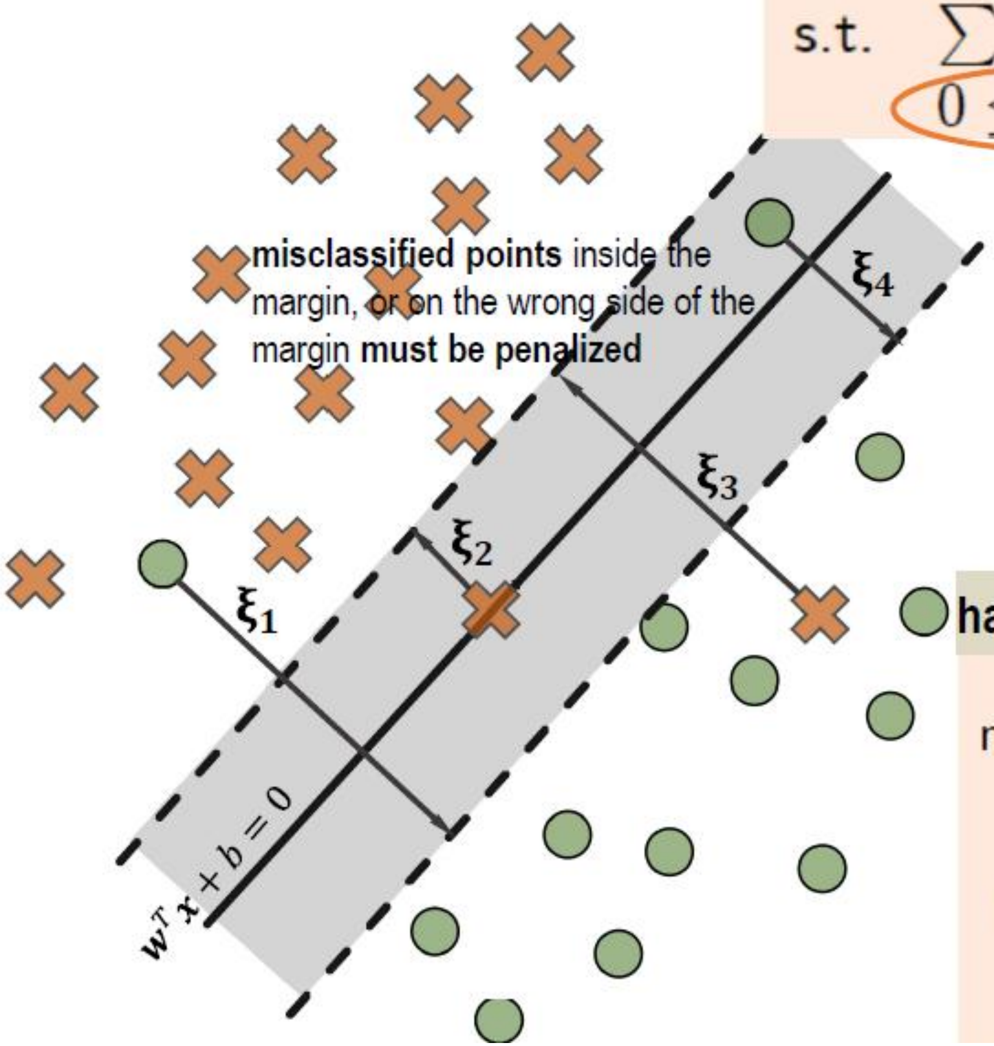
Soft-Margin SVM: Dual Problem

soft-margin svm dual

$$\max \quad -\tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i'\mathbf{x}_j + \sum_{i=1}^{n}\alpha_i$$

$$\text{s.t.} \quad \sum_{i=1}^{n}\alpha_i y_i = 0$$

$$0 \le \alpha_i \le C, \quad \forall i = 1\dots n$$



the only difference between the soft-margin and hard-margin SVM dual problems is that the Lagrange multipliers (α_i training example weights) are upper-bounded by the **regularization parameter** $(0 \le \alpha_i \le C)$

hard-margin svm dual

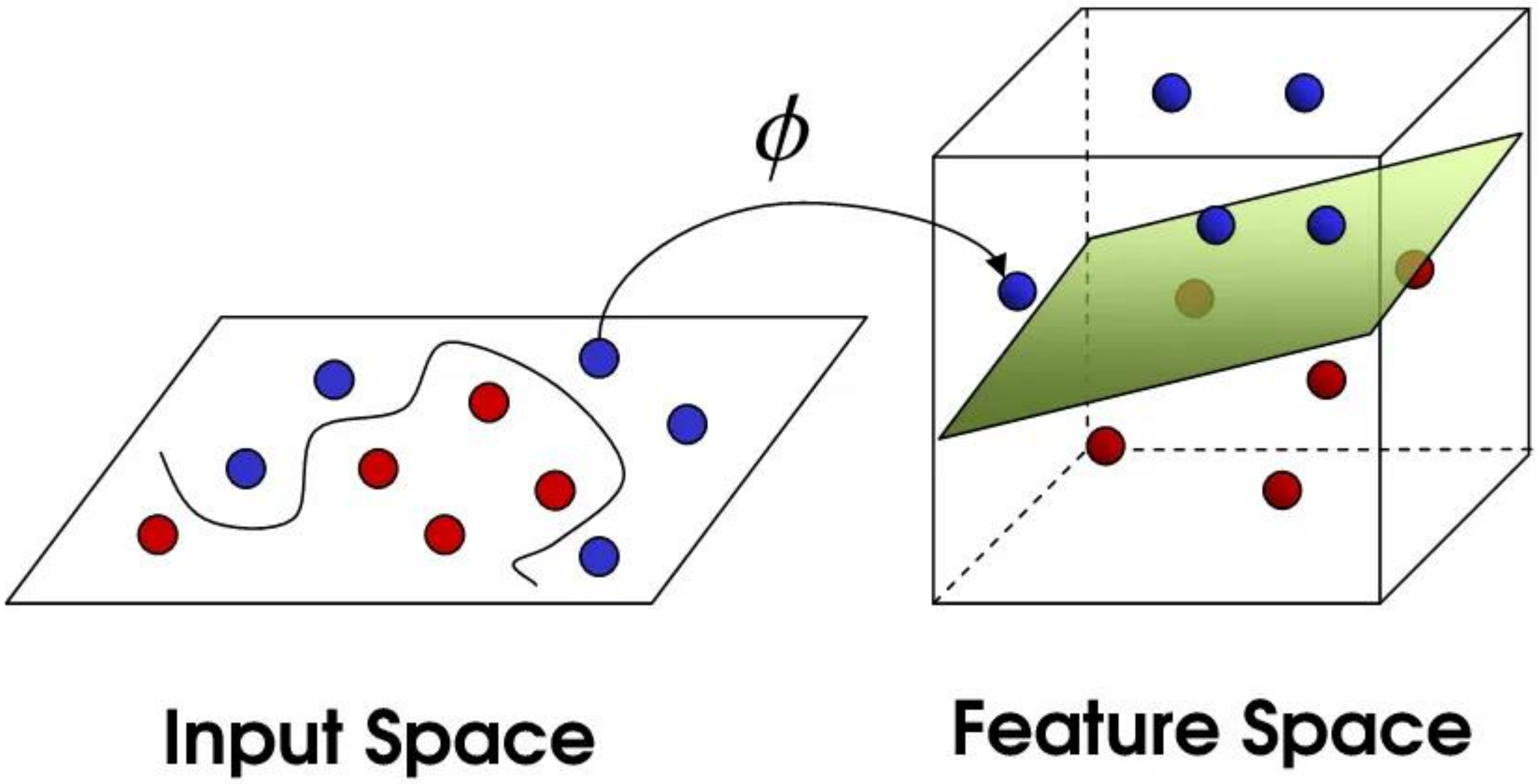
$$\max \quad -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i'\mathbf{x}_j + \sum_{i=1}^{n}\alpha_i$$

$$\text{s.t.} \quad \sum_{i=1}^{n}\alpha_i y_i = 0,$$

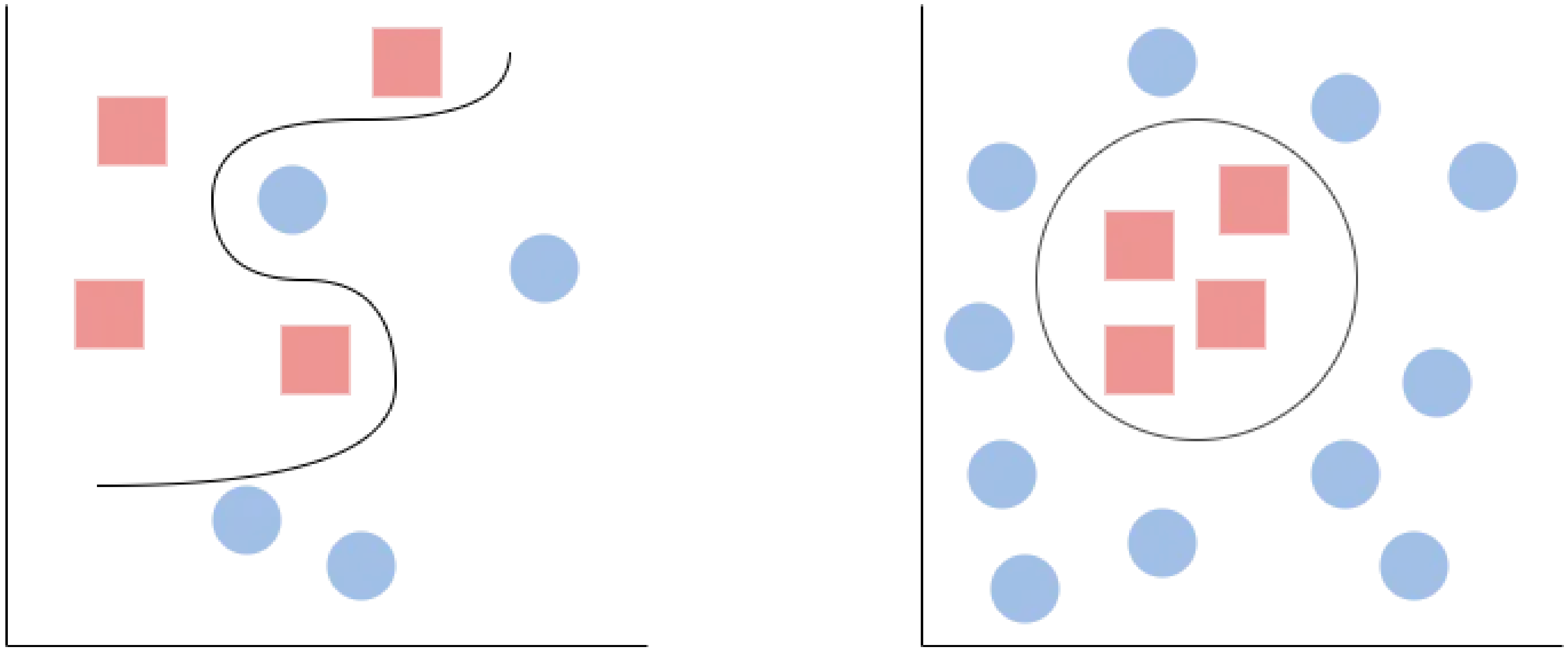
$$\alpha_i \ge 0, \quad \forall i = 1\dots n$$

$$0 \le \alpha_i \le \infty$$

Kernel Trick in Support Vector Classification

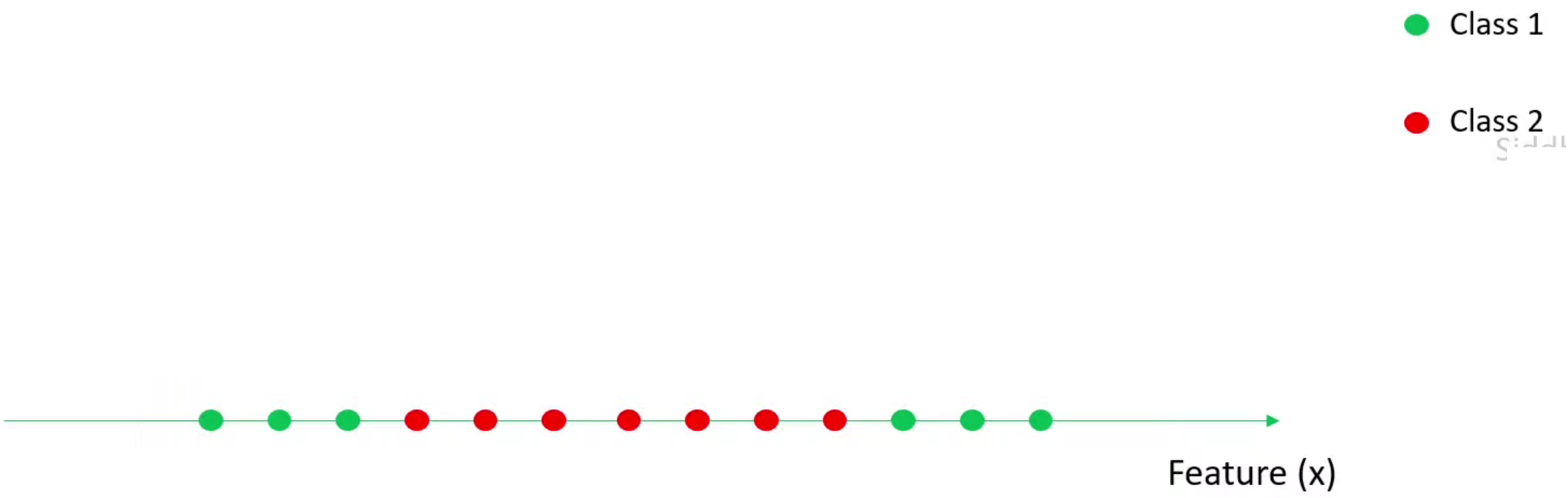


Motivation For Kernels



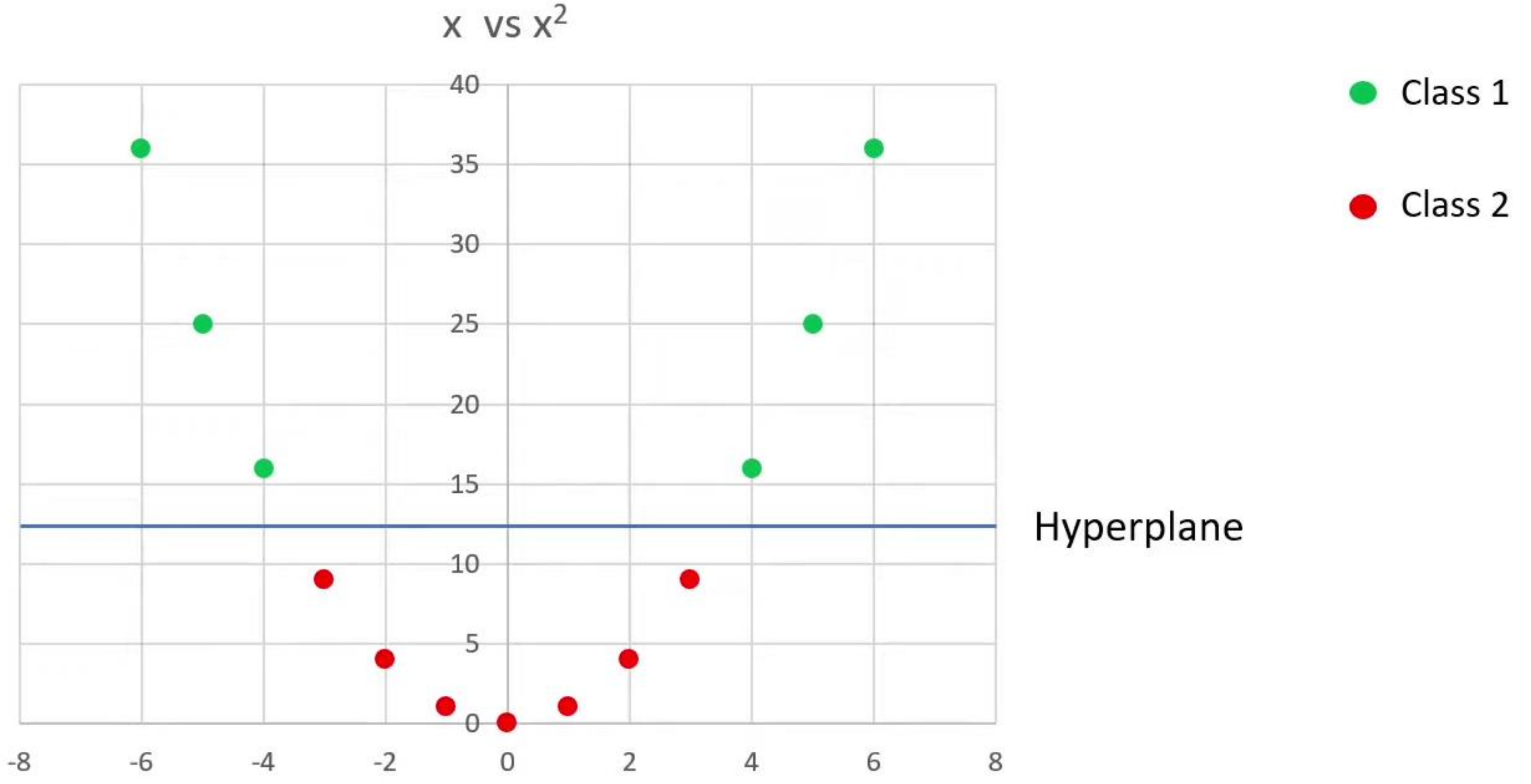
SVM Kernels

Feature (x)	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6

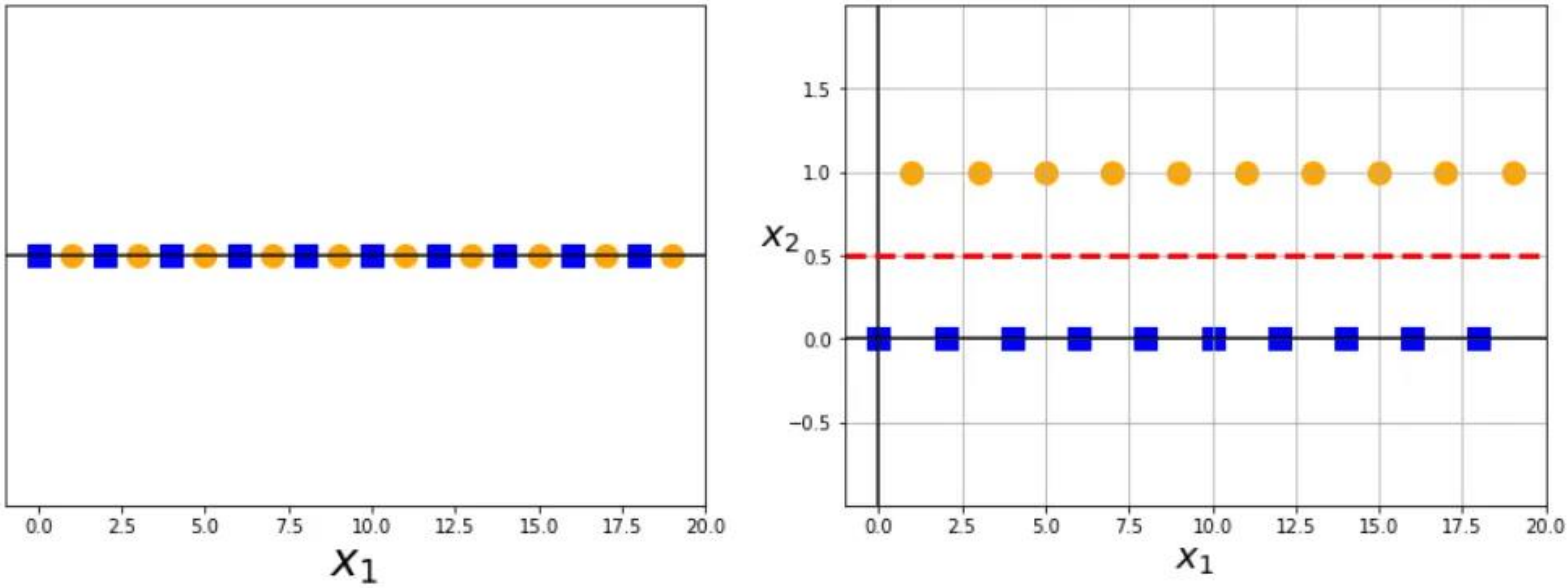


SVM Kernels

Feature (x)	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
x^2	36	25	16	9	4	1	0	1	4	9	16	25	36

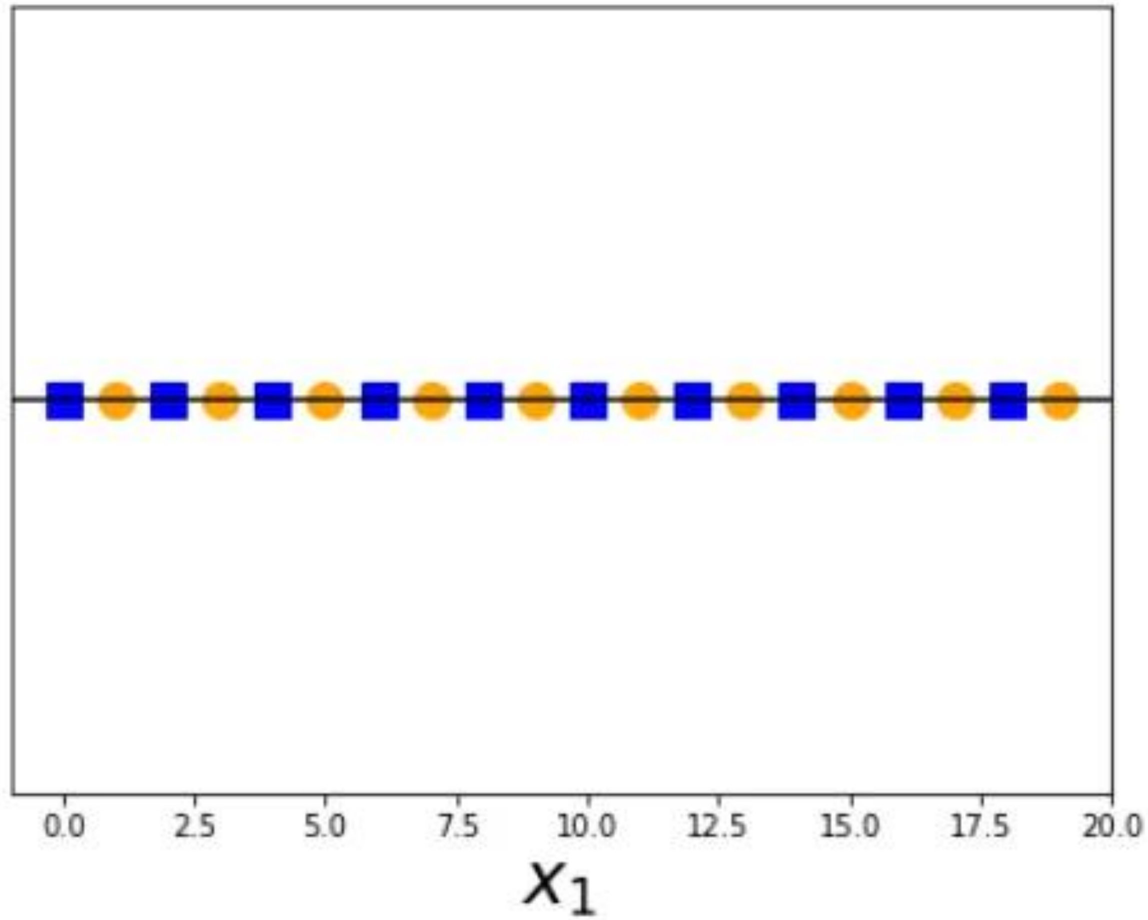


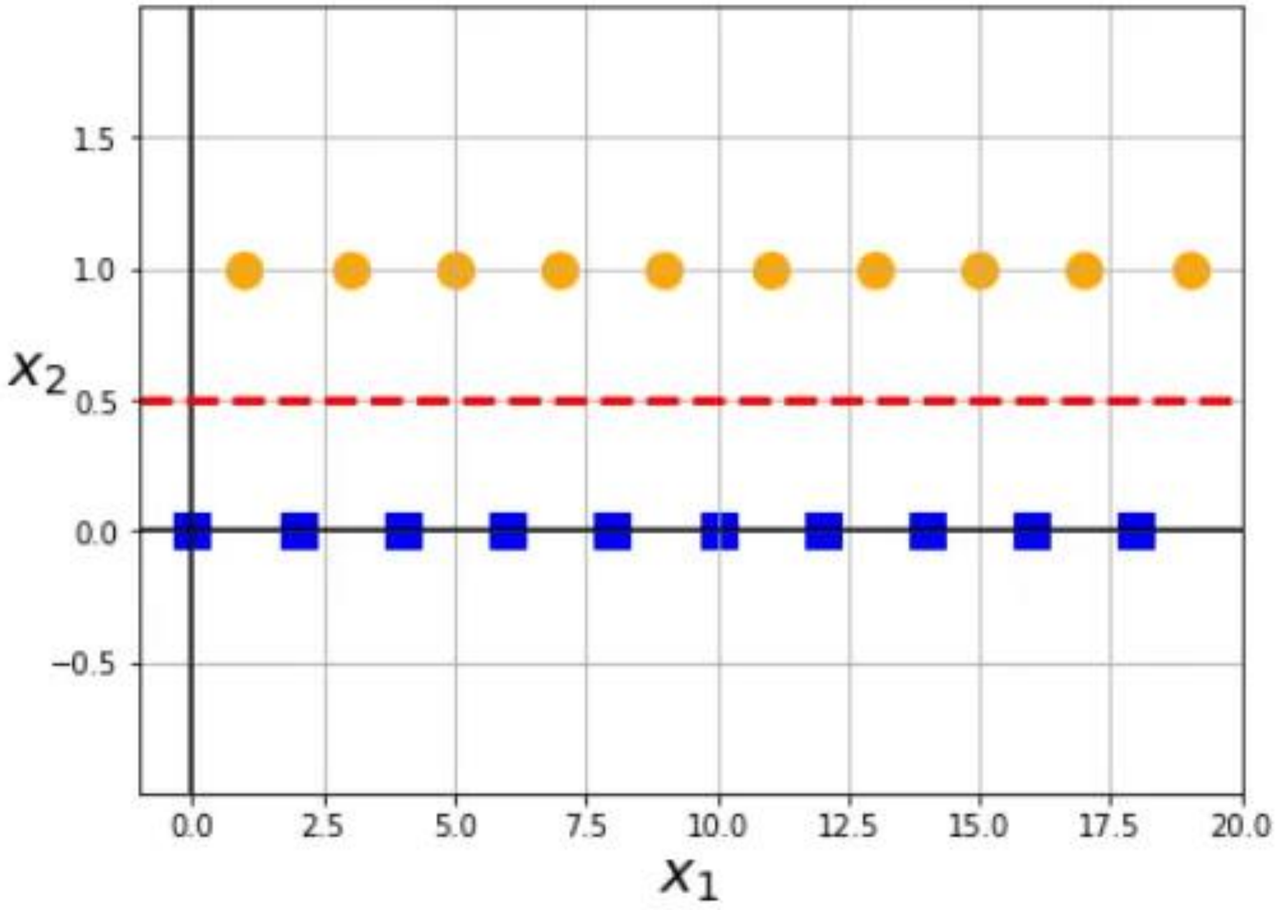
Guess the feature transformation function

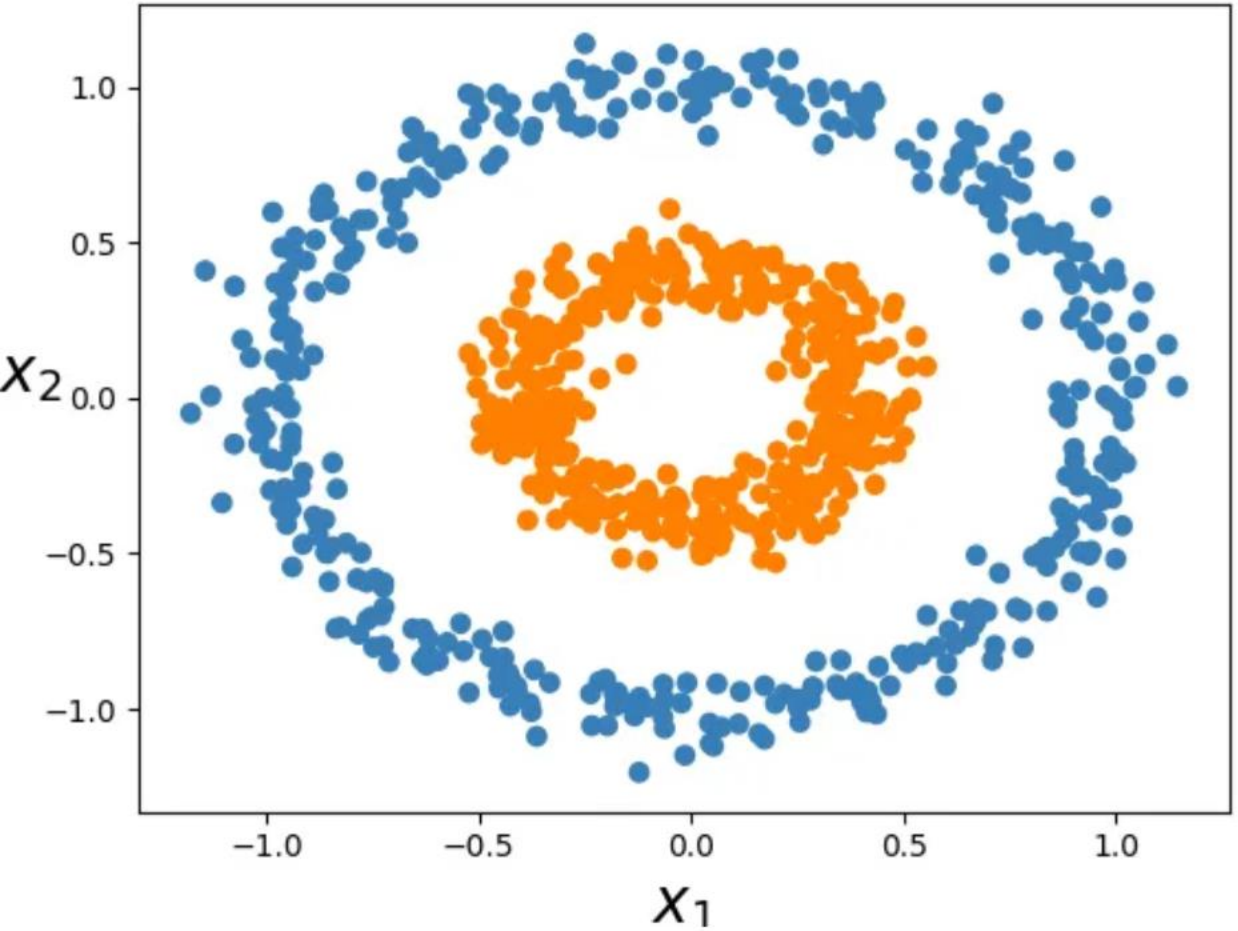


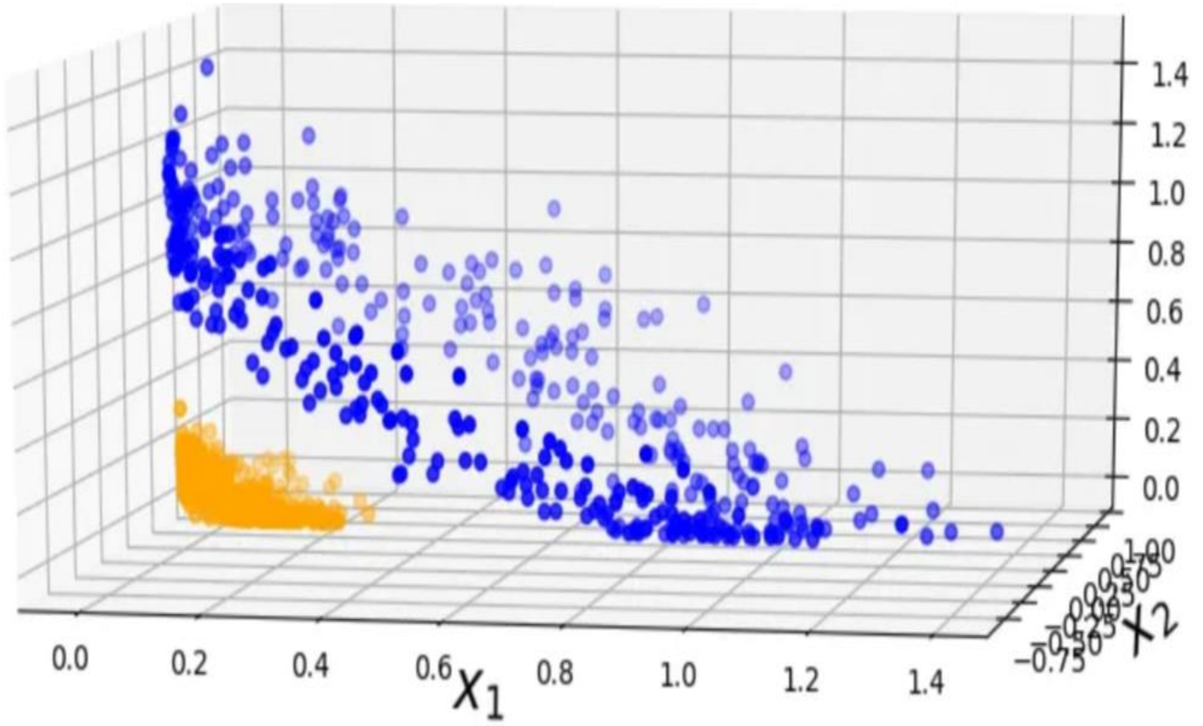
Guess the feature transformation function

Here we apply the transformation $\phi(x) = x \bmod 2$









$$\phi : \mathbb{R}^2 \to \mathbb{R}^3$$

$\phi : \chi \to F$ *Second-degree polynomial mapping*

$$\phi(\mathbf{x}) = \phi\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{pmatrix} {x_1}^2 \\ \sqrt{2}\, x_1 x_2 \\ {x_2}^2 \end{pmatrix}$$

Example

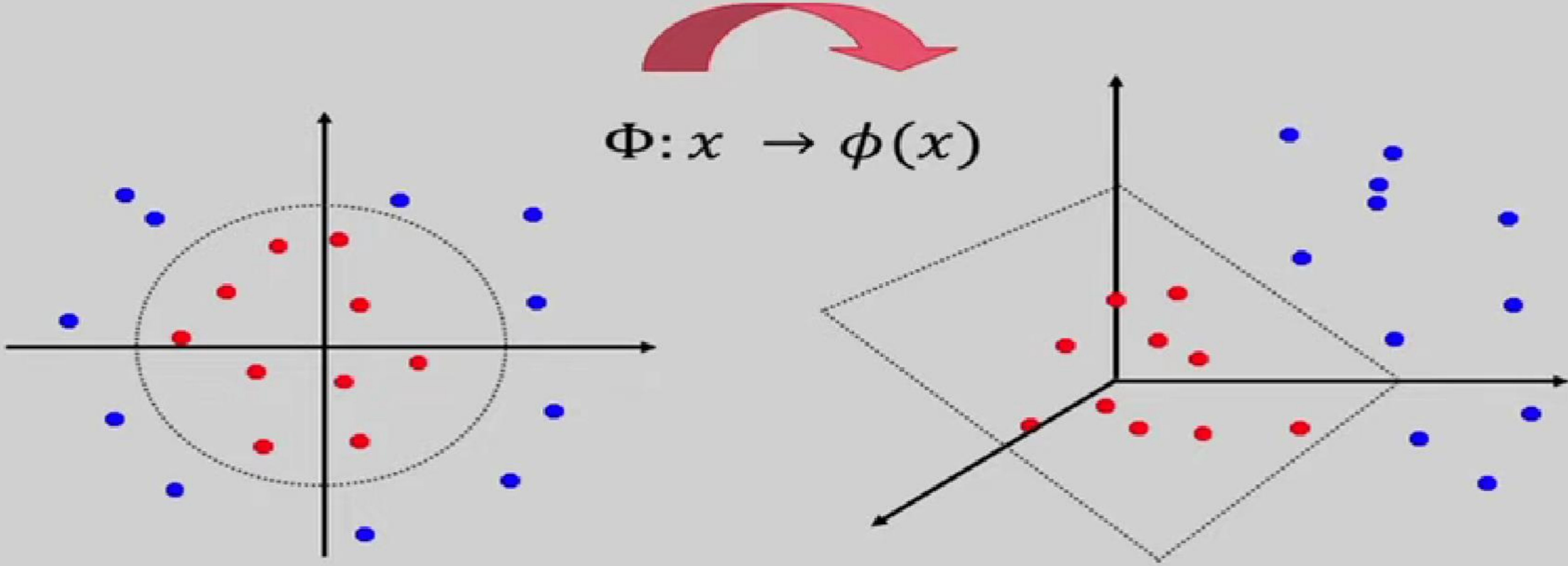
$$x = \begin{array}{c} \begin{matrix} x_1 & x_2 \end{matrix} \\ \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 2 & 1 \end{bmatrix} \end{array} \quad \phi(x) = \begin{array}{c} \begin{matrix} x_1 & x_2 & x_3 \end{matrix} \\ \begin{bmatrix} 1 & 1 & \sqrt{2} \\ 1 & 4 & \sqrt{2}\cdot 2 \\ 1 & 9 & \sqrt{2}\cdot 3 \\ 4 & 1 & 2 \end{bmatrix} \end{array}$$

Take away points.................

- We have seen how higher dimensional transformations can allow us to separate data in order to make classification predictions. It seems that in order to train a support vector classifier and optimize our objective function, we would have to perform operations with the higher dimensional vectors in the transformed feature space.

- In real applications, there might be many features in the data and applying transformations that involve many polynomial combinations of these features will lead to extremely high and impractical computational costs.

Non-linear SVMs: Feature Space



What are the steps for nonlinear SVM classification?

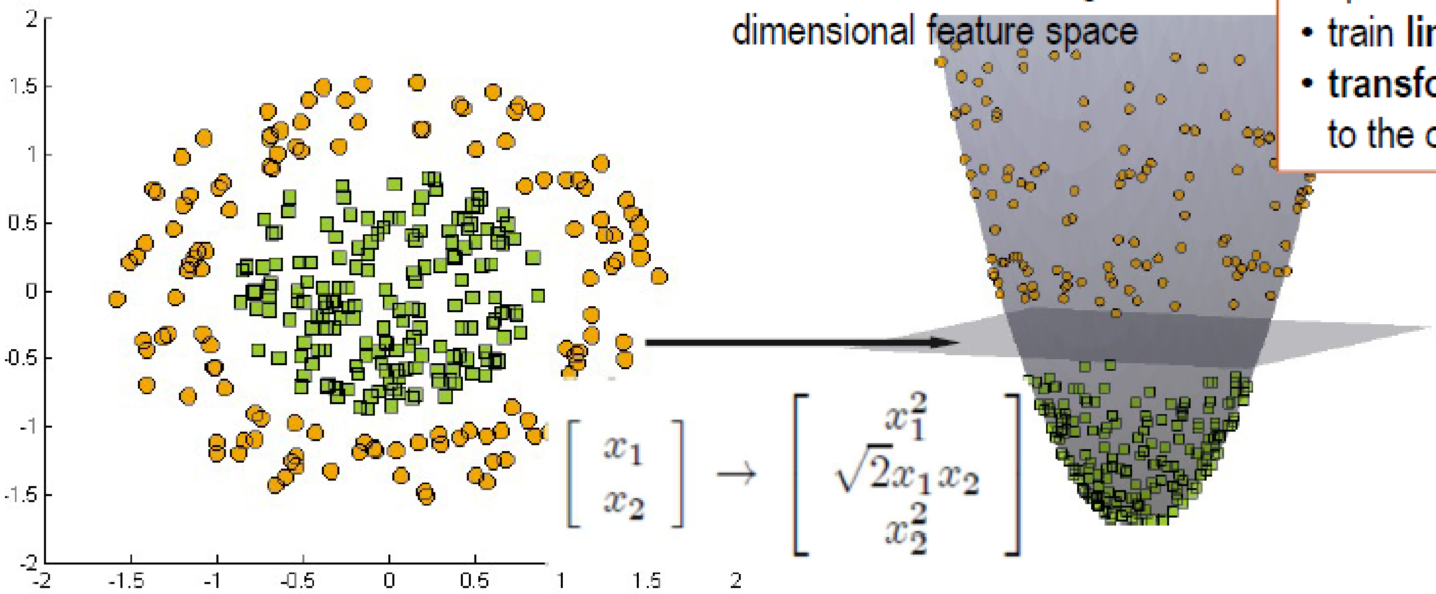
- Feature transformation in higher dimensional space
- Train linear SVM classifier and find out the hyperplane
- During test time also use the transformation function on test input features and do the inference.

Nonlinear SVM Classifiers

Solution Approach 1 (Explicit Transformation)

- **transform data** to a higher dimensional feature space
- train **linear SVM classifier in the high-dim. space**
- **transform high-dimensional linear classifier** back to the original space to obtain a nonlinear classifier

transform data to a higher dimensional feature space

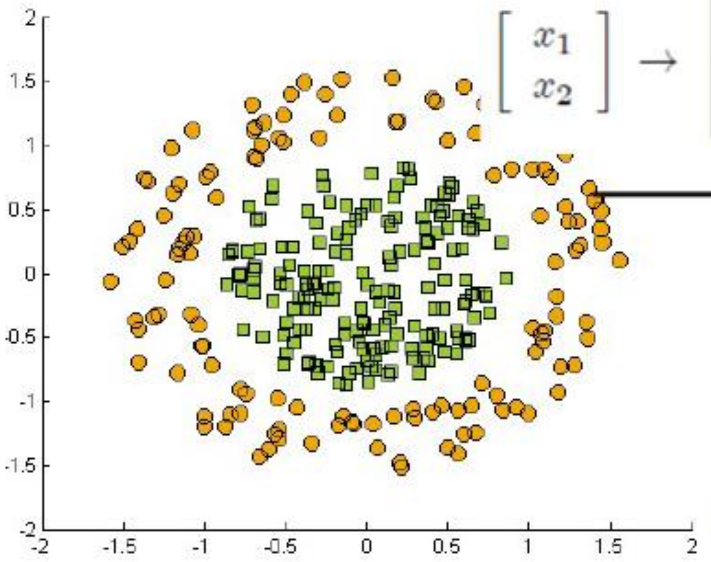


$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

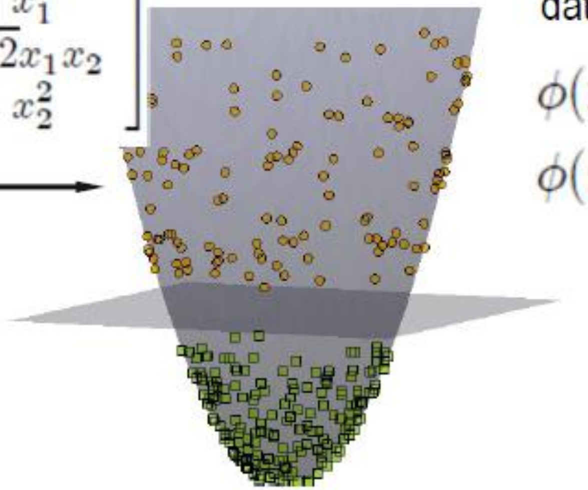
$$\Phi(\mathbf{x}) = \left(\begin{array}{cl} 1 & \}\ \text{Constant Term} \\ \left.\begin{matrix} \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \vdots \\ \sqrt{2}x_m \end{matrix}\right\} & \text{Linear Terms} \\ \left.\begin{matrix} x_1^2 \\ x_2^2 \\ \vdots \\ x_m^2 \end{matrix}\right\} & \text{Pure Quadratic Terms} \\ \left.\begin{matrix} \sqrt{2}x_1x_2 \\ \sqrt{2}x_1x_3 \\ \vdots \\ \sqrt{2}x_1x_m \\ \sqrt{2}x_2x_3 \\ \vdots \\ \sqrt{2}x_1x_m \\ \vdots \\ \sqrt{2}x_{m-1}x_m \end{matrix}\right\} & \text{Quadratic Cross-Terms} \end{array}\right)$$

if we have m training features, the size of the transformation grows very fast; **explicit transformations** can become **very expensive**

The Kernel Trick

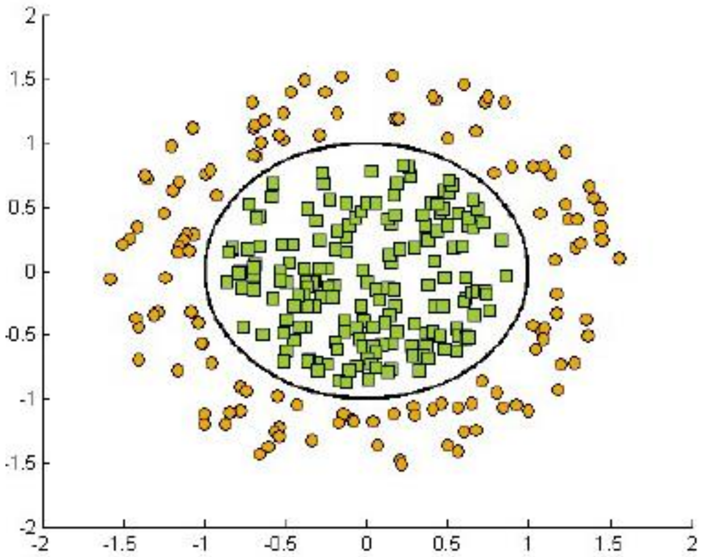


$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$



data in higher-dimensional space

$$\phi(\mathbf{x}) = (x_1^2,\, \sqrt{2}x_1x_2,\, x_2^2)$$
$$\phi(\mathbf{z}) = (z_1^2,\, \sqrt{2}z_1z_2,\, z_2^2)$$



when learning a linear SVM, recall that the dual solution depends **only on the inner products** of the training data, so we only need to compute inner products in the higher-dimensional space:

$$\begin{aligned} \phi(\mathbf{x})^T\phi(\mathbf{z}) &= x_1^2z_1^2 + 2x_1x_2z_1z_2 + x_2^2z_2^2 \\ &= (x_1z_1 + x_2z_2)^2 = (\mathbf{x}^T\mathbf{z})^2 \\ &= (\mathbf{x}^T\mathbf{z})^2 \end{aligned}$$

the function $\kappa(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^T\boldsymbol{z})^2$ is an example of a **kernel function**

the kernel function relates the inner-products in the original and transformed spaces; with a kernel, ***we can avoid explicit transformation***

Soft margin linear SVM classifier objective

$$\underset{\mathbf{w}, b, \zeta}{\text{minimize}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{m}\zeta^{(i)}$$

→ **Primal form**

$$\text{subject to } y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \geq 1 - \zeta^{(i)} \quad \text{and} \quad \zeta^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m$$

Dual form of the linear SVM objective

$$\underset{\alpha}{\text{minimize}} \; \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\mathbf{x}^{(i)T}\mathbf{x}^{(j)} \quad - \quad \sum_{i=1}^{m}\alpha^{(i)}$$

$$\text{subject to} \quad \alpha^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m$$

From the dual solution to the primal solution

$$\widehat{\mathbf{w}} = \sum_{i=1}^{m} \hat{\alpha}^{(i)} y^{(i)} \mathbf{x}^{(i)}$$

$$\hat{b} = \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \left(y^{(i)} - \widehat{\mathbf{w}}^T \mathbf{x}^{(i)} \right)$$

Kernel Trick

In Machine Learning, a kernel is a function capable of computing the dot product ϕ(a)T ϕ(b) based only on the original vectors a and b, without having to compute (or even to know about) the transformation ϕ.

$$K(\mathbf{a}, \mathbf{b}) = \phi(\mathbf{a})^T \ \phi(\mathbf{b})$$

Example of kernelized SVM

Second-degree polynomial mapping

$$\phi(\mathbf{x}) = \phi\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{pmatrix} {x_1}^2 \\ \sqrt{2}\, x_1 x_2 \\ {x_2}^2 \end{pmatrix}$$

Kernel trick for a 2nd-degree polynomial mapping

$$\phi(\mathbf{a})^T \phi(\mathbf{b}) = \begin{pmatrix} {a_1}^2 \\ \sqrt{2}\, a_1 a_2 \\ {a_2}^2 \end{pmatrix}^T \begin{pmatrix} {b_1}^2 \\ \sqrt{2}\, b_1 b_2 \\ {b_2}^2 \end{pmatrix} = {a_1}^2 {b_1}^2 + 2 a_1 b_1 a_2 b_2 + {a_2}^2 {b_2}^2$$

$$= \left(a_1 b_1 + a_2 b_2\right)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^T \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)^2 = \left(\mathbf{a}^T \mathbf{b}\right)^2$$

Most frequently used popular Kernels

Linear: $K(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T \mathbf{b}$

Polynomial: $K(\mathbf{a}, \mathbf{b}) = \left(\gamma \mathbf{a}^T \mathbf{b} + r\right)^d$

Gaussian RBF: $K(\mathbf{a}, \mathbf{b}) = \exp\left(-\gamma \| \mathbf{a} - \mathbf{b} \|^2\right)$

Sigmoid: $K(\mathbf{a}, \mathbf{b}) = \tanh\left(\gamma \mathbf{a}^T \mathbf{b} + r\right)$

Mercer's Theorem

According to *Mercer's theorem*, if a function $K(\mathbf{a}, \mathbf{b})$ respects a few mathematical conditions called *Mercer's conditions* (K must be continuous, symmetric in its arguments so $K(\mathbf{a}, \mathbf{b}) = K(\mathbf{b}, \mathbf{a})$, etc.), then there exists a function ϕ that maps $\mathbf{a}$ and $\mathbf{b}$ into another space (possibly with much higher dimensions) such that $K(\mathbf{a}, \mathbf{b}) = \phi(\mathbf{a})^T \phi(\mathbf{b})$. So you can use K as a kernel since you know ϕ exists, even if you don't know what ϕ is. In the case of the Gaussian RBF kernel, it can be shown that ϕ actually maps each training instance to an infinite-dimensional space, so it's a good thing you don't need to actually perform the mapping!

Note that some frequently used kernels (such as the Sigmoid kernel) don't respect all of Mercer's conditions, yet they generally work well in practice.

Why Support Vector Machines (SVM) is good?

Clear Margin Separation: Input of the SVM is a set of datasets and output is an optimal hyperplane. In one dimensional space, the hyperplane is a point. In two-dimensional space, the hyperplane is a line. In three-dimensional space, the hyperplane is a surface. [7]

Effective in high-dimensional Spaces: When the number of dimensions is greater than the number of samples, SVM is automatically regularized. That avoids the overfitting of the high-dimensional data.[7]

Memory Efficient: Instead of using the entire training set, SVM only uses a subset of the training dataset to train the models. The subset is called support vectors.

Effective in non-linear classification: SVM uses the kernel function to map the data into high-dimensional spaces, then separate them linearly. It can solve complex problems with an appropriate kernel function.

Straightforward classification concept: SVM maximizes the margin between the support vectors and the target hyperplane.