Geometric Data Analysis

Support Vector Machines

MAT 6480W / STT 6705V

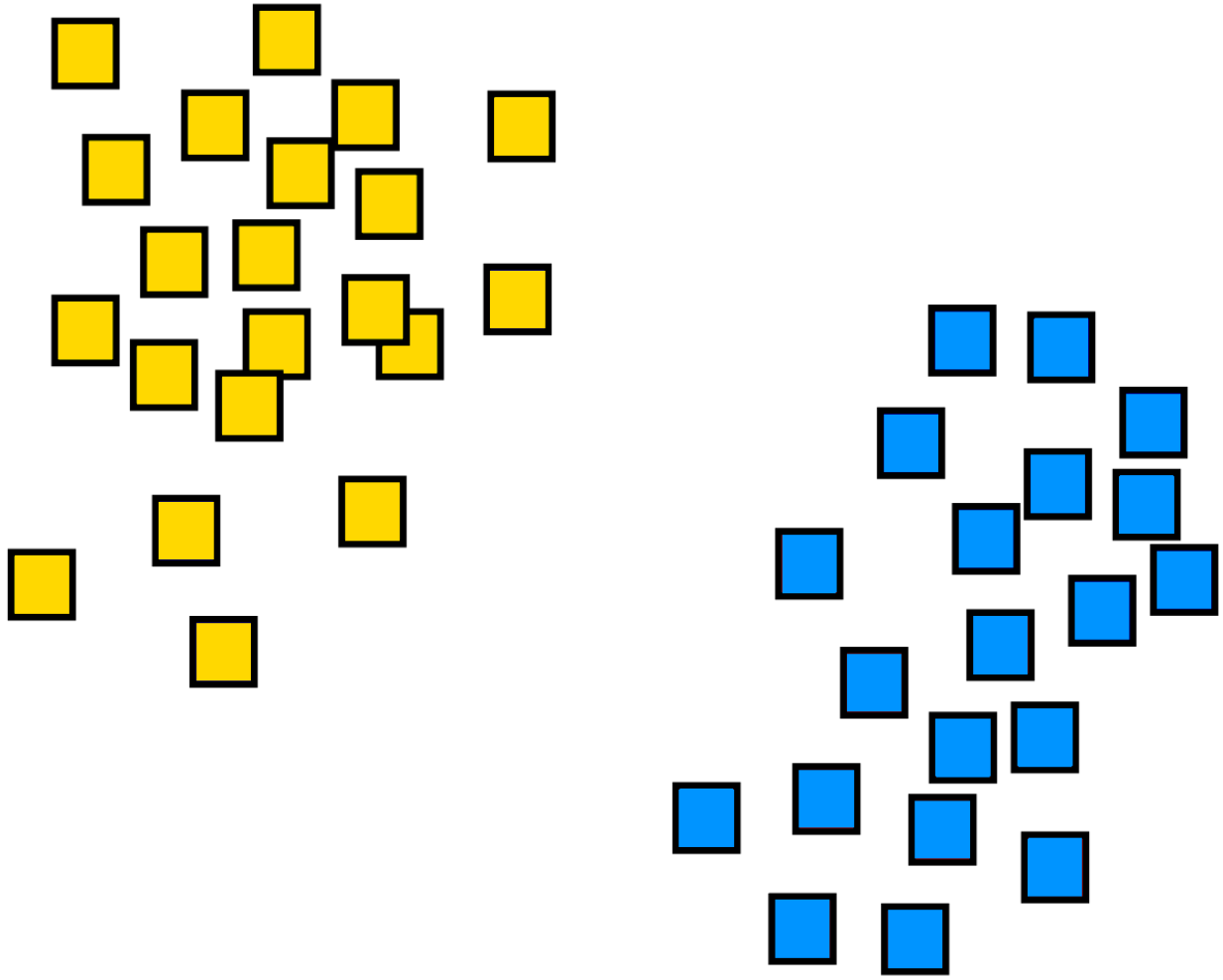
Guy Wolf
guy.wolf@umontreal.ca

Université de Montréal
Fall 2019

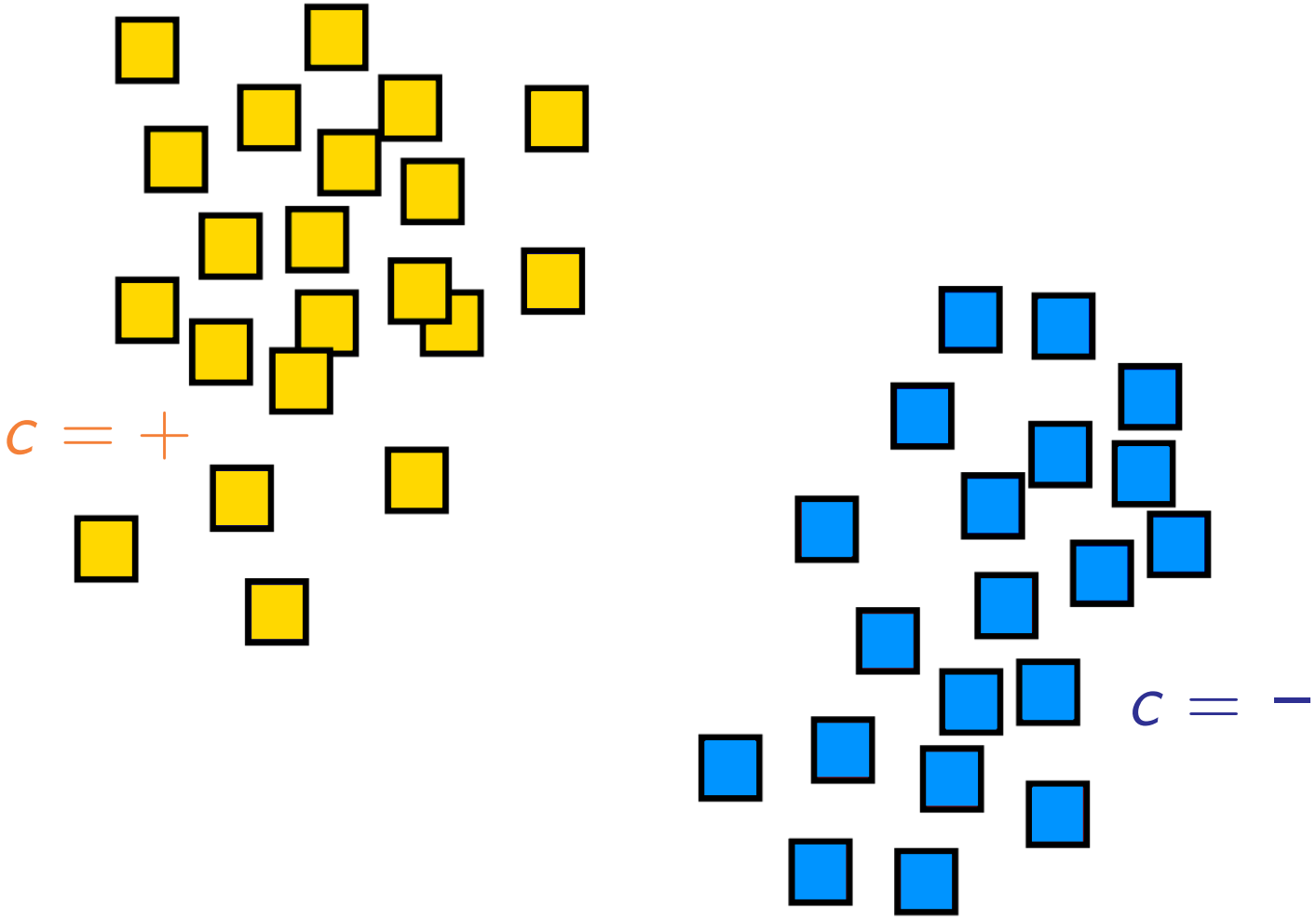
Outline

1. Support vector machines
   - Affine hyperplane
   - Hyperplane & margin formulation
   - Margin maximization
   - Quadratic programming
   - Optimization with Lagrange multipliers
   - Support vectors
   - Soft margin
2. Nonlinear SVM
   - Feature extraction
   - Kernel trick
   - Mercer kernels & RKHS
   - Polynomial & RBF kernels

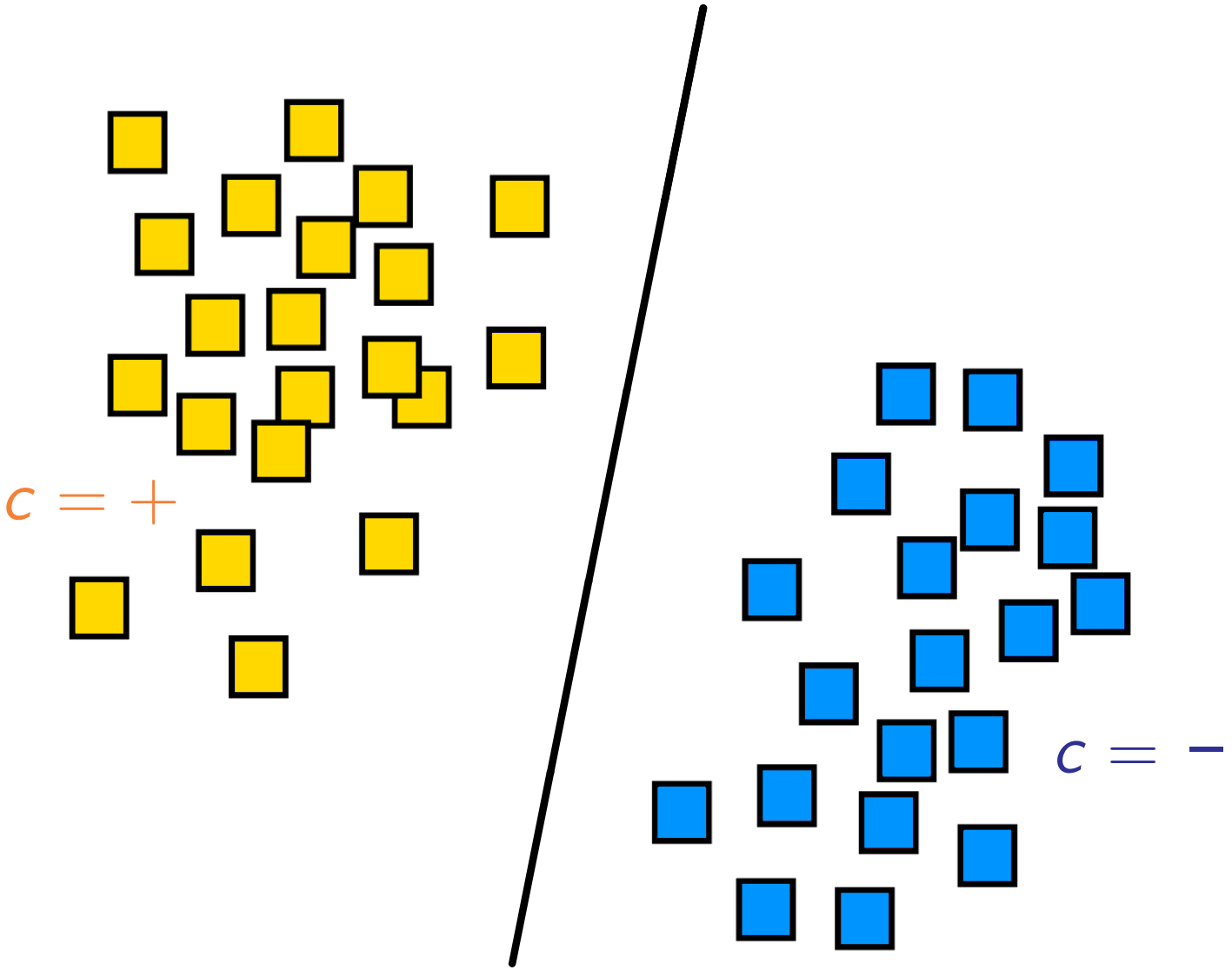
Support vector machines



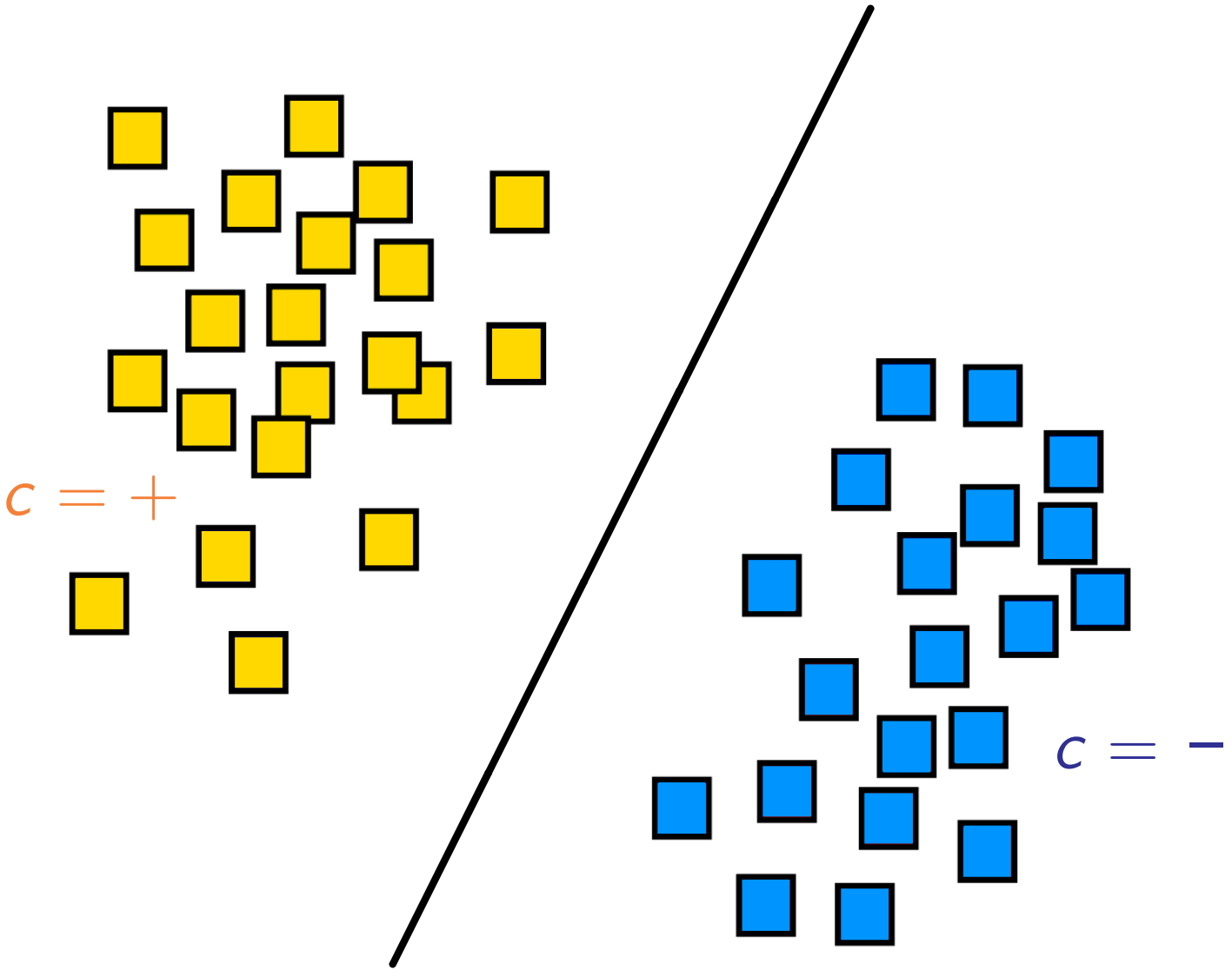
Support vector machines



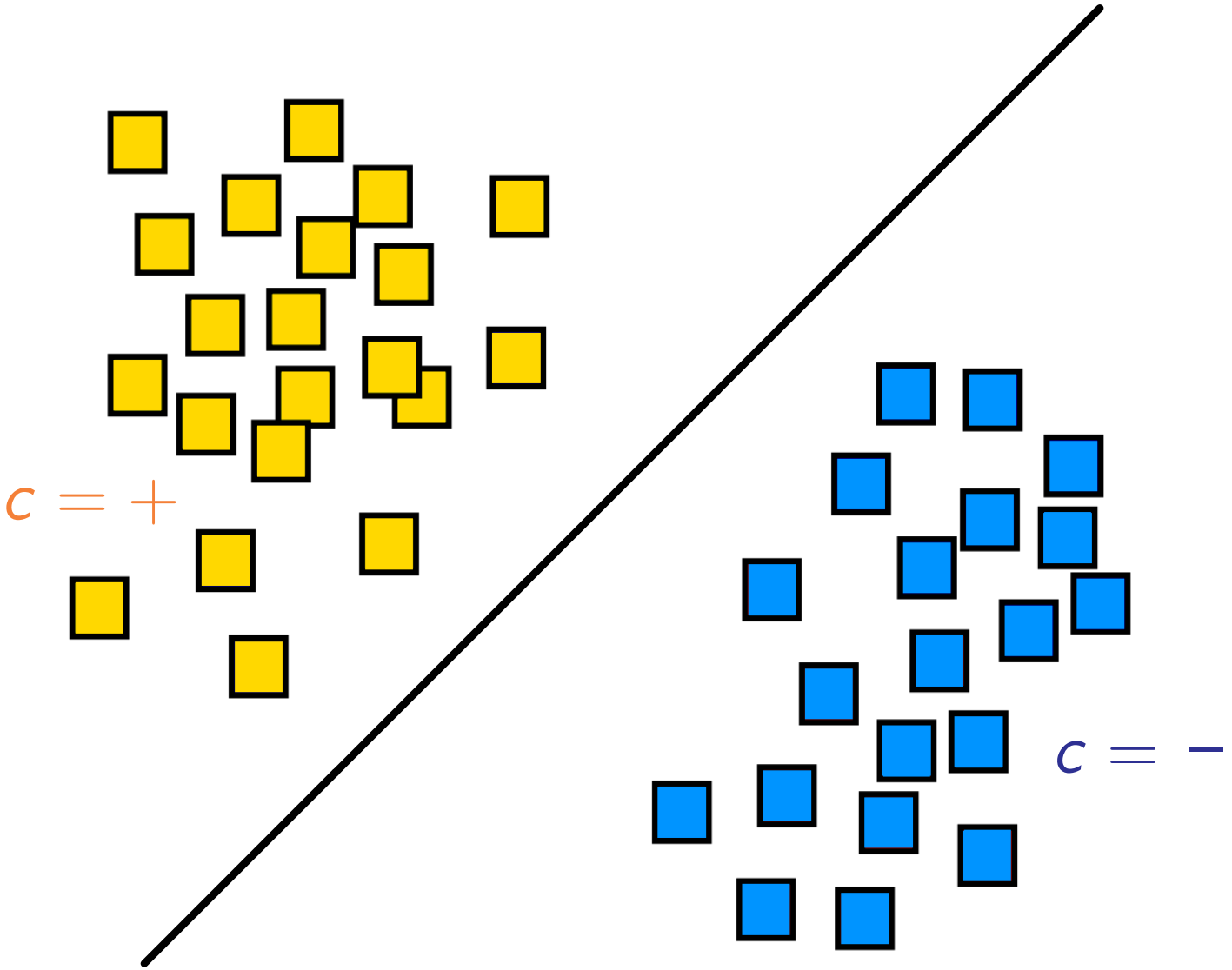
Support vector machines



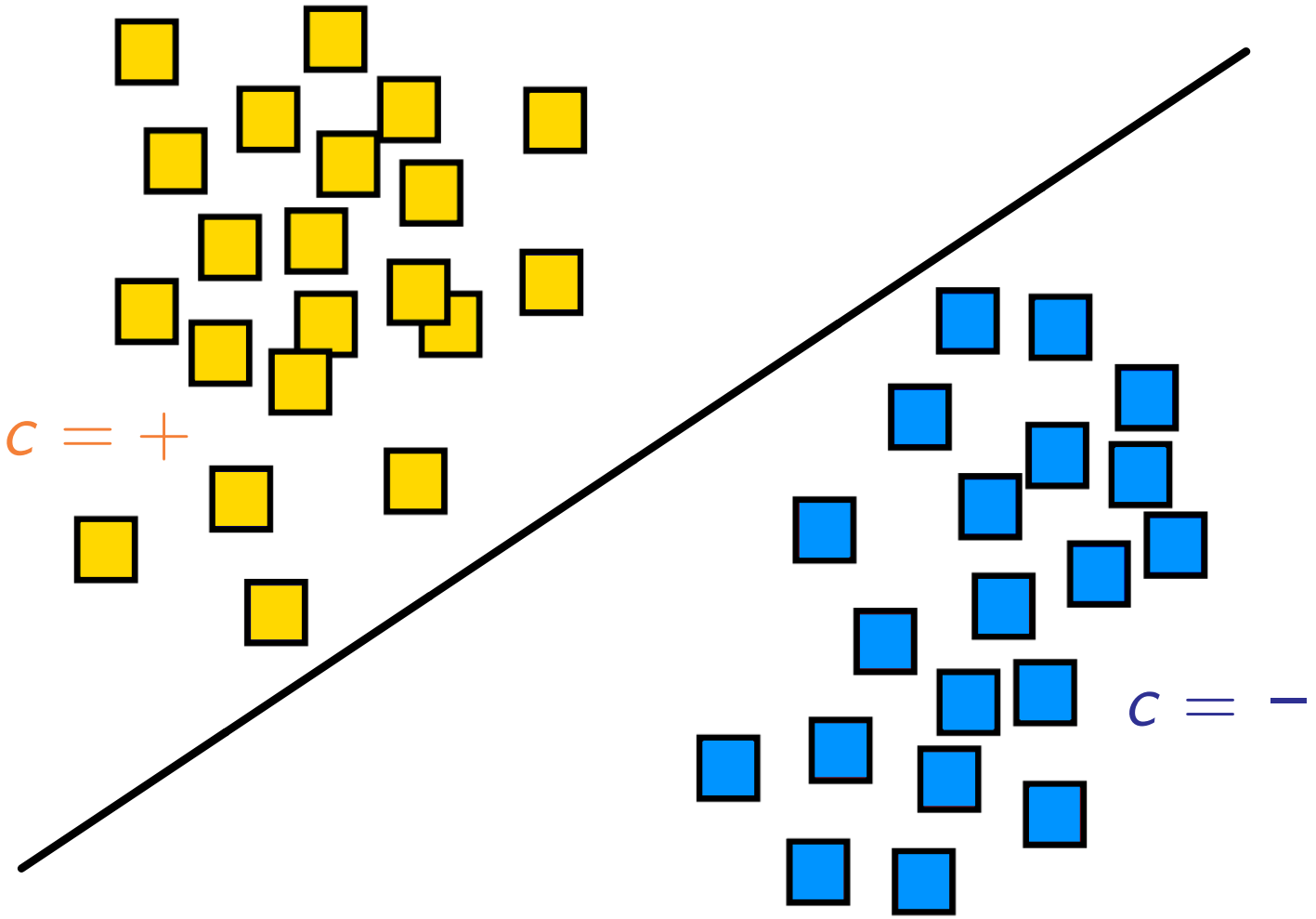
Support vector machines



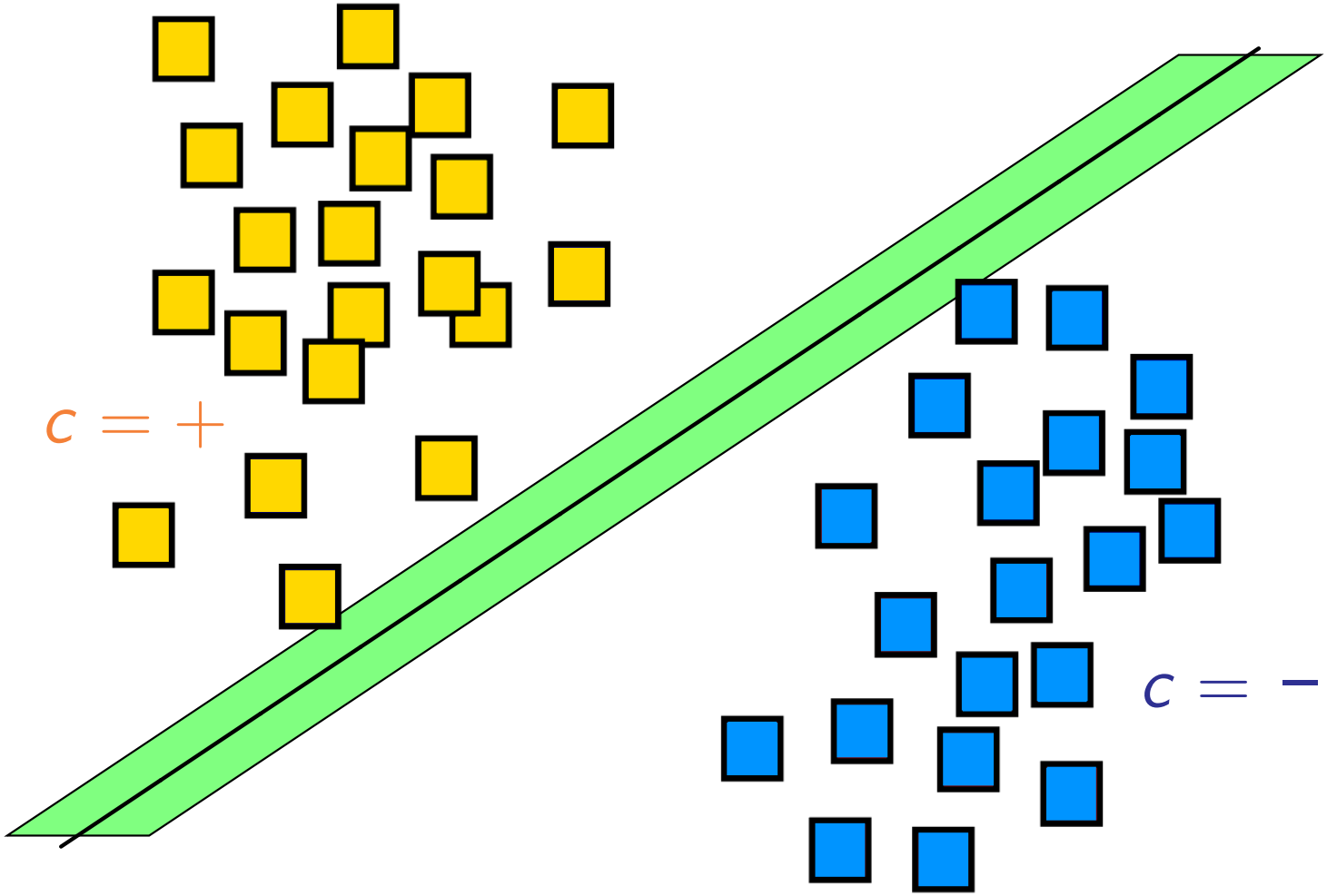
Support vector machines



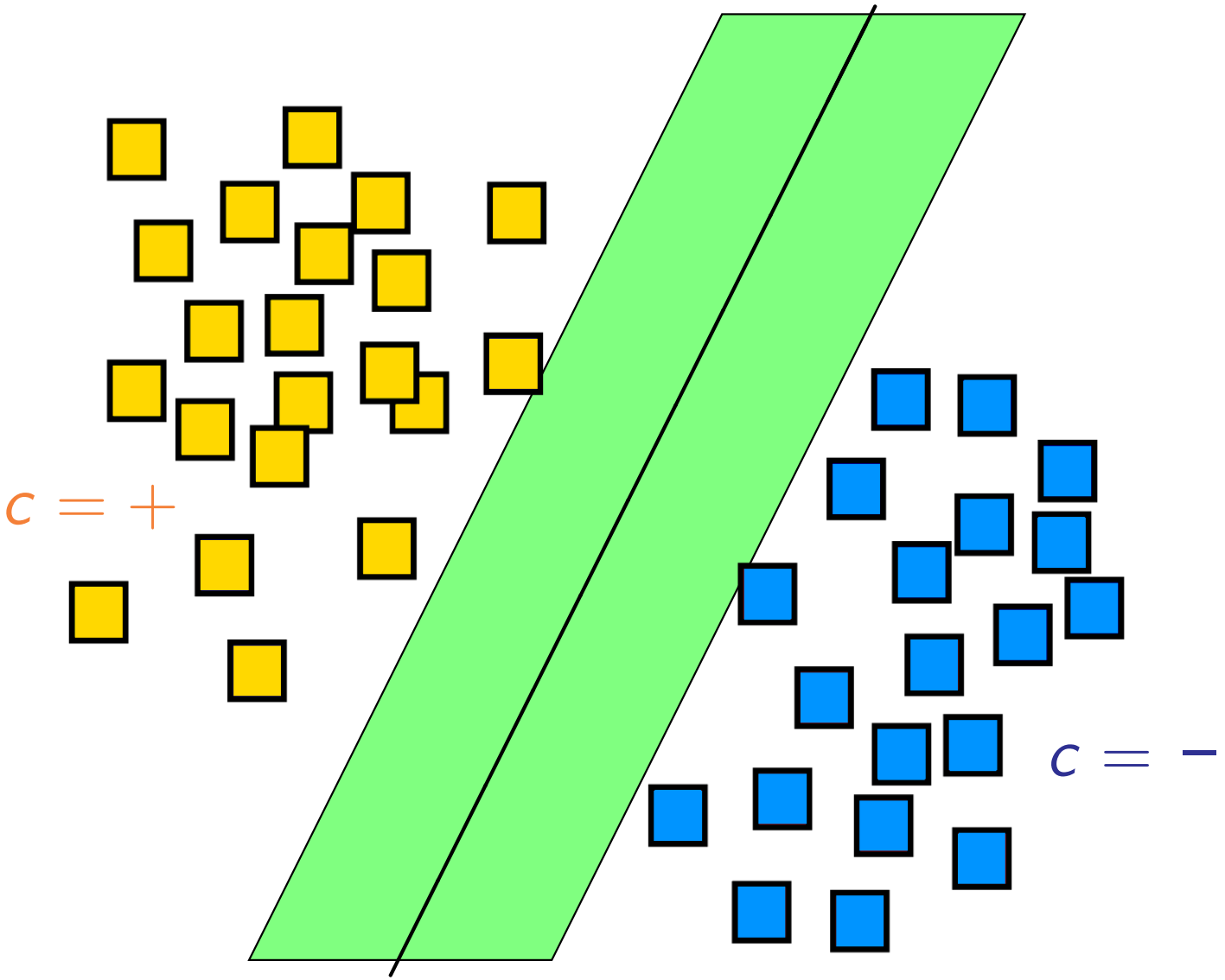
Support vector machines



Support vector machines



Support vector machines



Hyperplane

A hyperplane of $\mathbb{R}^n$ is a subspace of dimension $n-1$ of that space.

Affine hyperplane

An affine hyperplane is the set of vectors that satisfy $\sum w_j \vec{x}[j] = b$ where $\sum w_j^2 = 1$.

- The scalar b determines the distance from the origin.
- Using vector notations we have $\langle \vec{w}, \vec{x} \rangle = b$ where $\vec{w}$ is the normal unit vector that determines the orientation of the hyperplane.
- Affine hyperplanes split $\mathbb{R}^n$ into two parts: $> b$ and $< b$.

Support vector machines

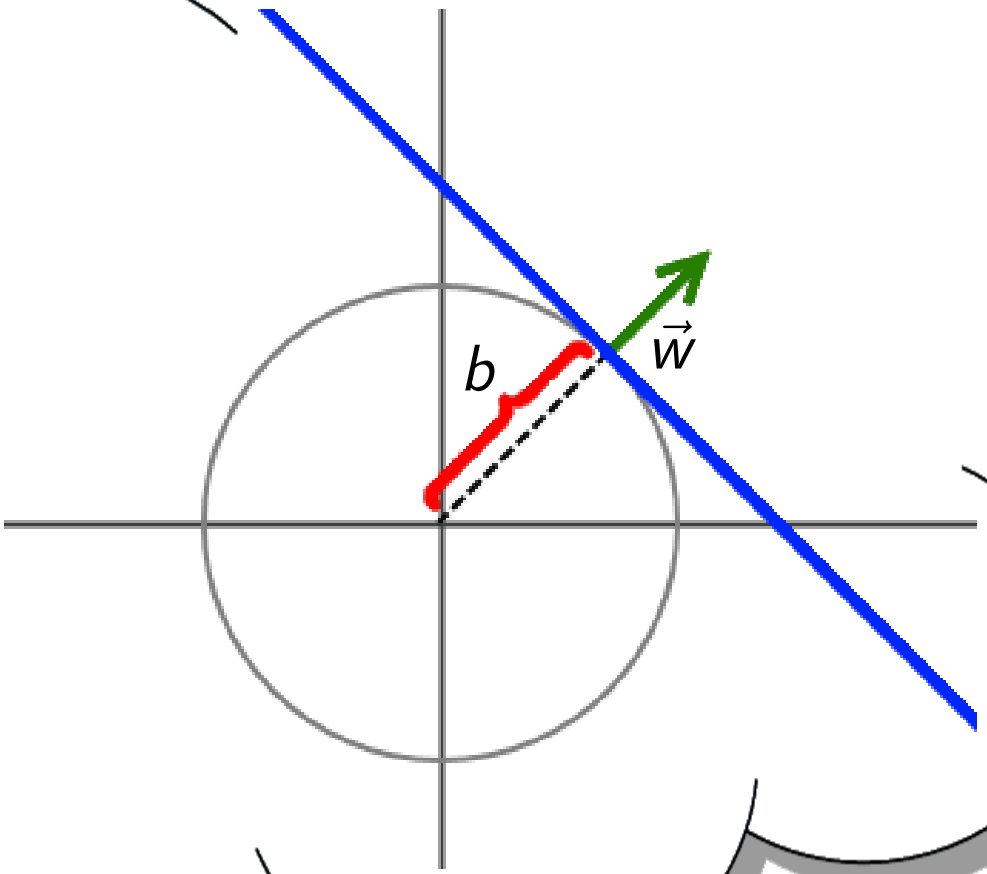
Affine hyperplane

Hyperplane

A hyperpla space.

Affine h

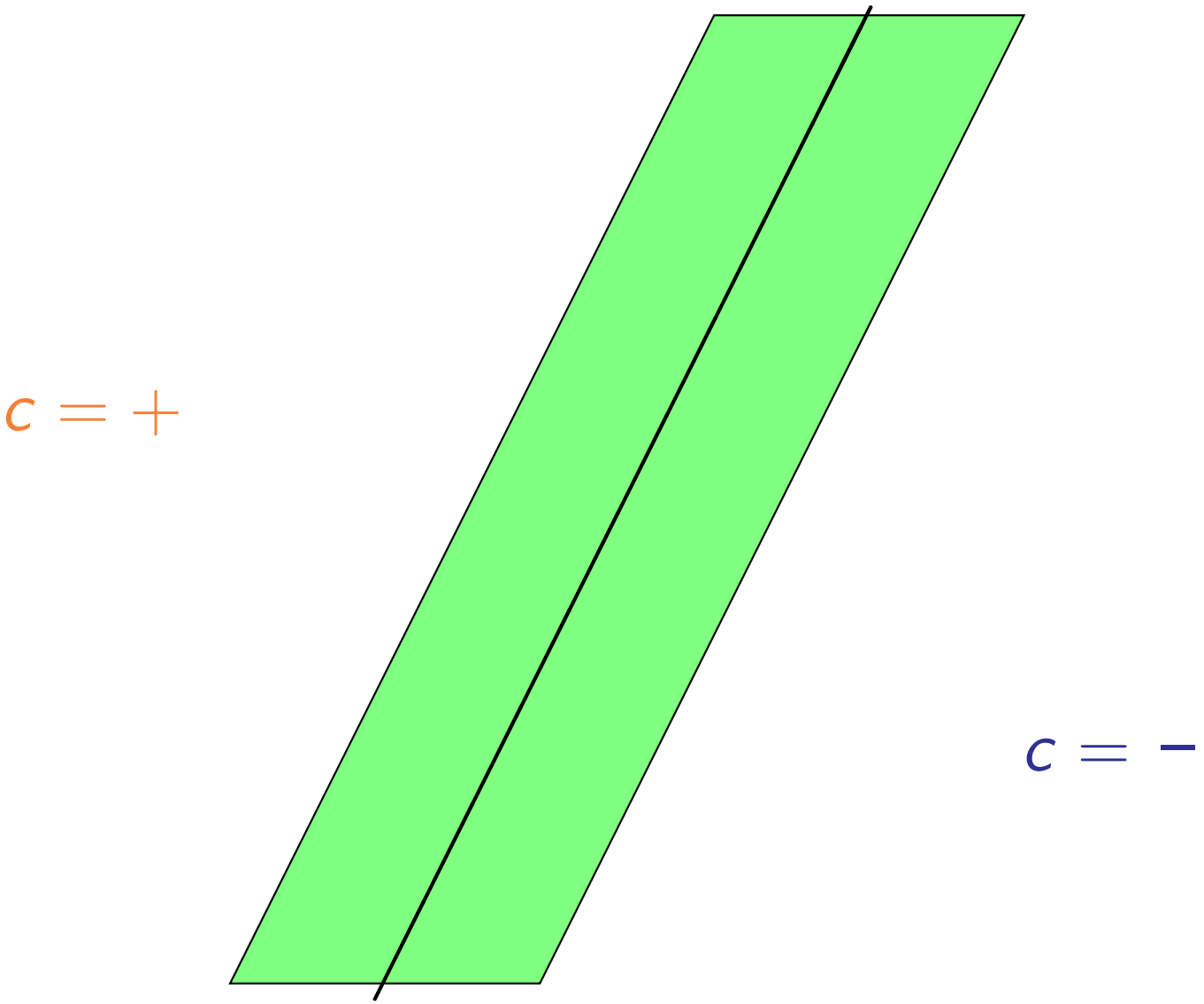
An a = b
wh



- is the
  no of the
  hyperpla
- Affine hyperplanes split parts: $> b$ and $< b$.

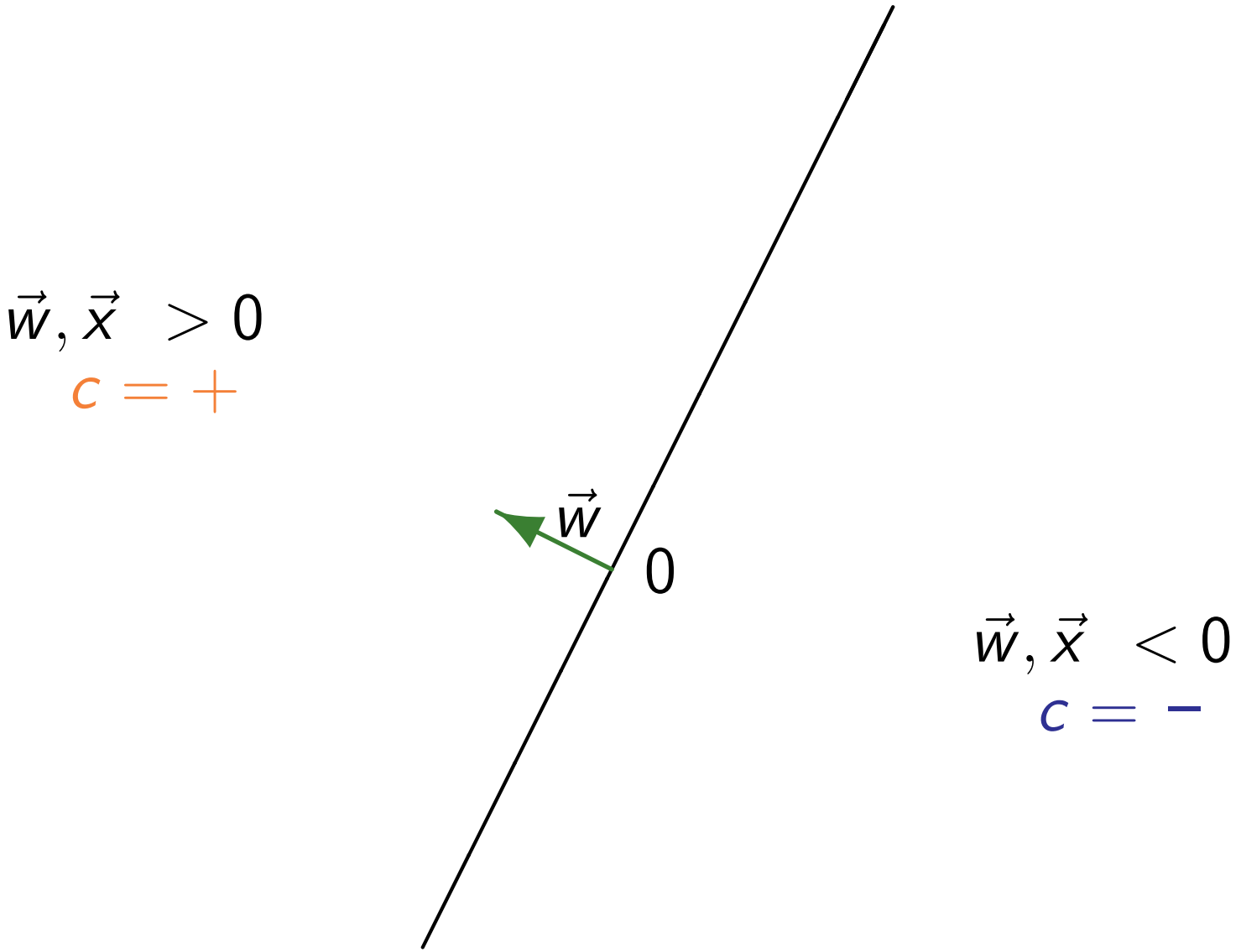
Support vector machines

Hyperplane & margin formulation



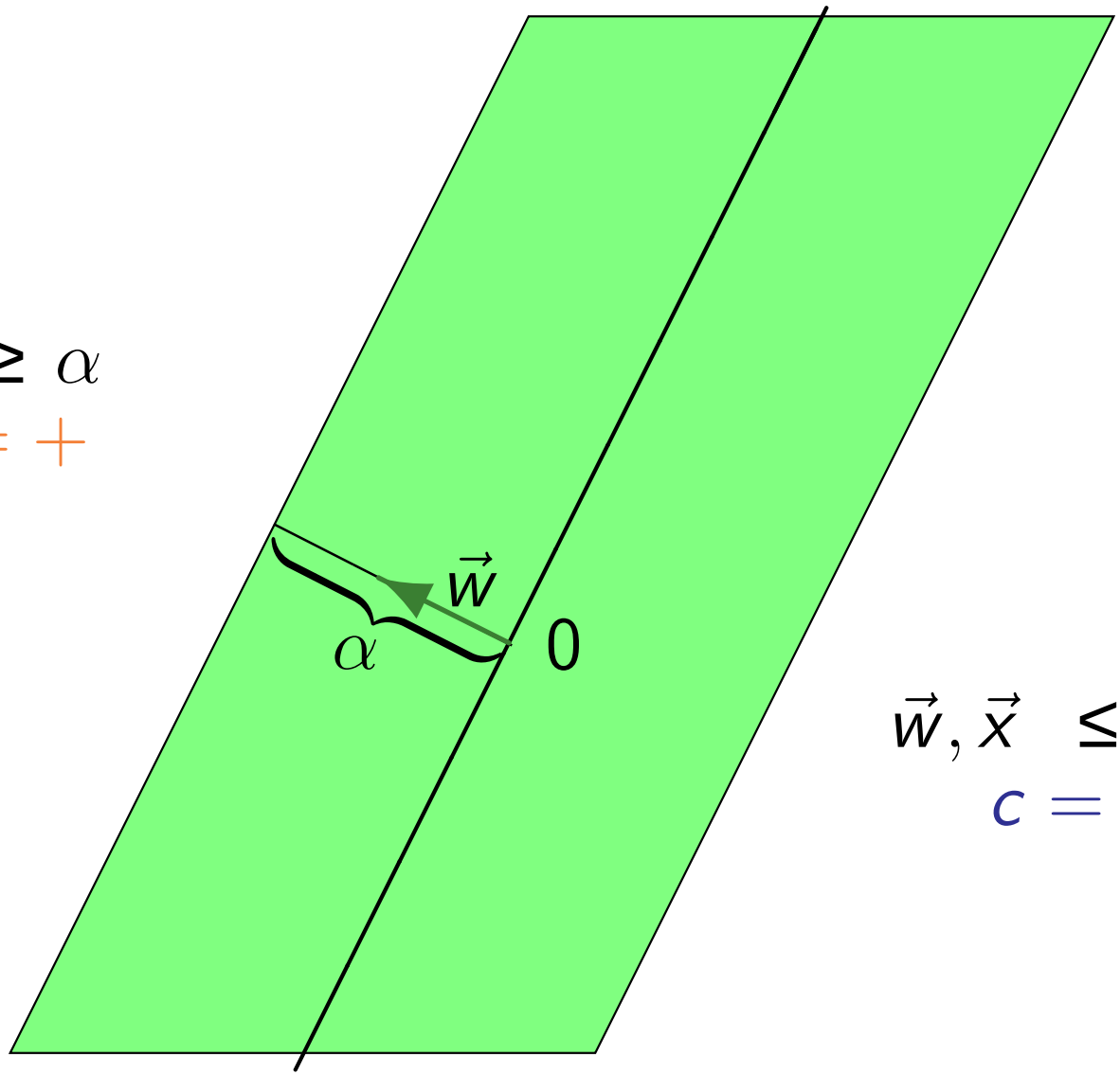
Support vector machines

Hyperplane & margin formulation



Support vector machines

Hyperplane & margin formulation



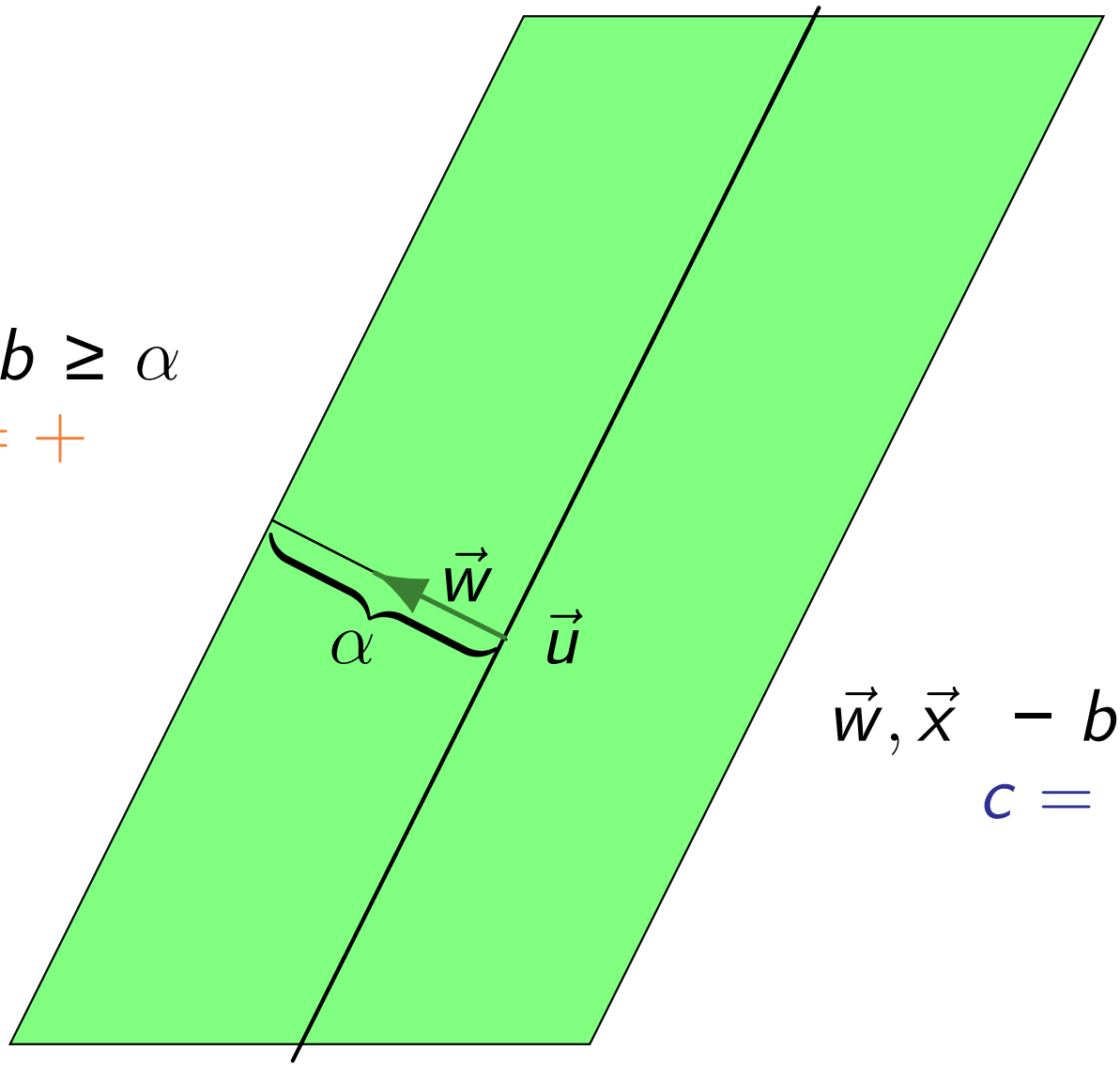
Support vector machines

Hyperplane & margin formulation

$$b = \vec{w}, \vec{u}$$

$$\vec{w}, \vec{x} - b \geq \alpha$$

$$c = +$$



$$\vec{w}, \vec{x} - b \leq -\alpha$$

$$c = -$$

SVM training

Input:

- Data points $\vec{x}_1, \ldots, \vec{x}_N \quad \mathbb{R}^n$
- Class labels $c_1, \ldots, c_N \quad \{-1, 1\}$

Solve the margin maximization problem:

$$\begin{array}{ll} \text{Find} & \max \alpha > 0 \\ \text{s.t.} & c_i(\ \vec{w}, \vec{x}_i\ - b) \geq \alpha \\ & \vec{w}\ = 1 \end{array}$$

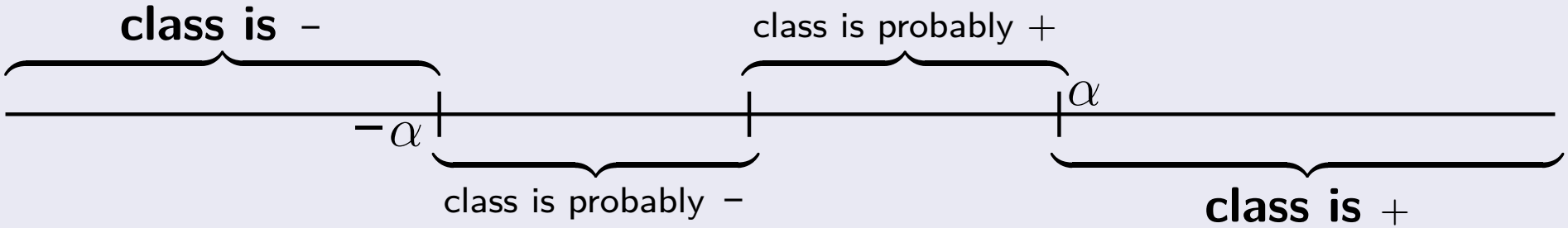
Output: the solution $(\vec{w}, b, \alpha)$

SVM classifier

Input:

- New data point $\vec{x}$
- The solution $(\vec{w}, b, \alpha)$ from SVM training

Classify by value of $\vec{w}, \vec{x} - b$:



Support vector machines

Quadratic program

Consider $\vec{w}_\alpha = \vec{w}/\alpha$ and $b_\alpha = \langle \vec{w}_\alpha, \vec{u} \rangle = b/\alpha$.

Then, the SVM constraint can be rephrased as

$$c_i(\langle \vec{w}, \vec{x}_i \rangle - b) \geq \alpha \Leftrightarrow c_i(\langle \vec{w}_\alpha, \vec{x}_i \rangle - b_\alpha) \geq 1$$

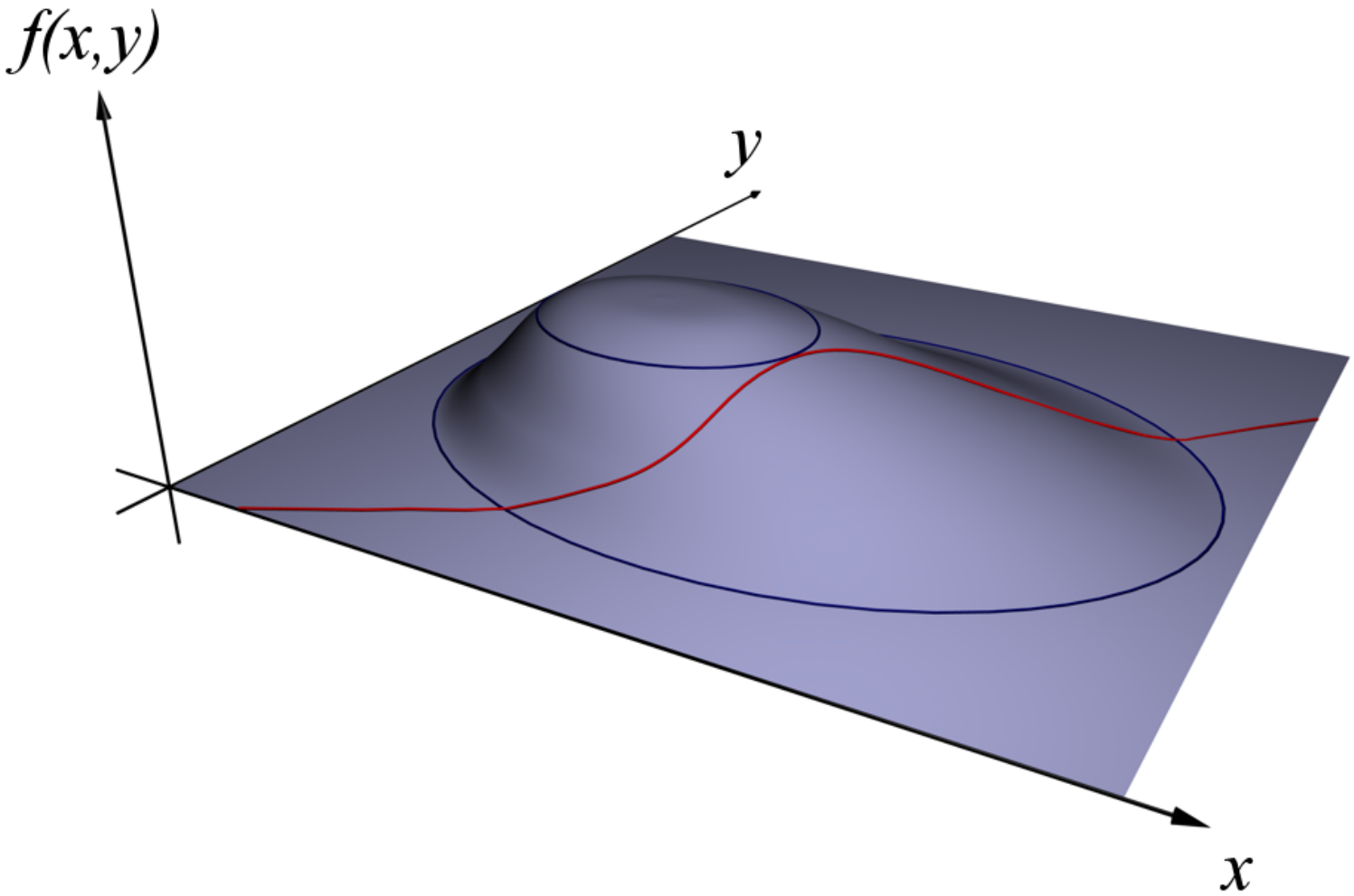
and the size of the margin becomes $2\alpha = 2/\|w_\alpha\|$. So we can rephrase the SVM optimization problem as:

$$\begin{array}{ll} \arg\min_{\vec{w}_\alpha, b_\alpha} & \|\vec{w}_\alpha\|^2/2 \\ \text{s.t.} \quad 1 \leq i \leq N & c_i(\langle \vec{w}_\alpha, \vec{x}_i \rangle - b_\alpha) \geq 1 \end{array}$$

Clearly, minimizing $\|\vec{w}_\alpha\|^2/2$ is equivalent to maximizing α, and with this formulation we have a quadratic target with linear constraints. Such optimizations are convex and yield a unique solution.

Support vector machines

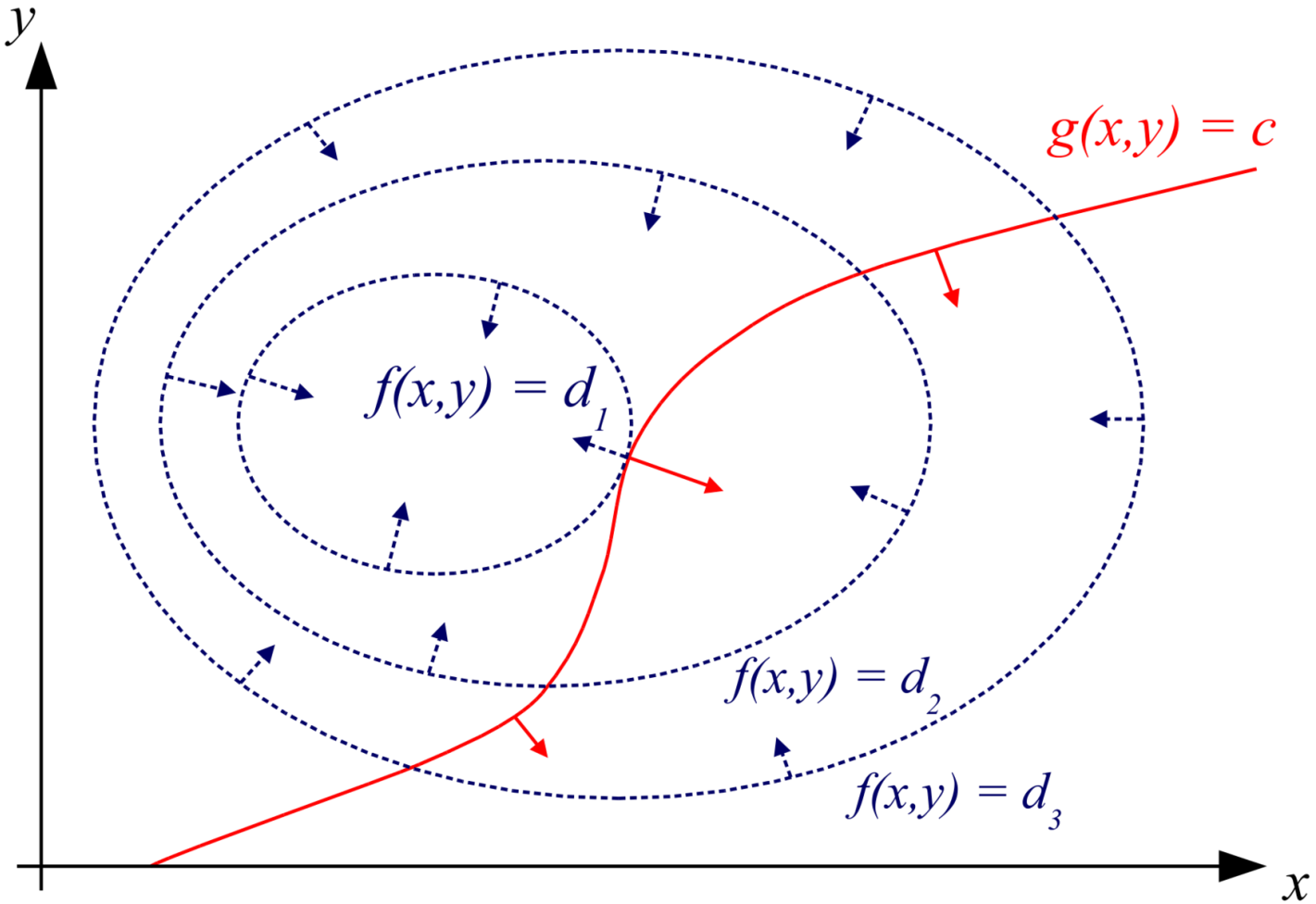
Lagrange multipliers



Taken from Wikipedia

Support vector machines

Lagrange multipliers



Taken from Wikipedia

Lagrangian

Given an optimization problem $\arg\min_{\vec{z}} f(\vec{z})$ s.t. $g_j(\vec{z}) = 0$, $j = 1, \ldots, k$, its optimal solution must be a stationary point of the **Lagrangian** function $f_P(\vec{z}, \lambda_1, \ldots, \lambda_k) = f(\vec{z}) - \sum_{j=1}^{k} \lambda_j g_j(\vec{z})$, i.e., a point where $f_P = 0$.

Notice that:

- Stationary points can be saddle points, not just extrema.
- Since $\frac{\partial f_P}{\partial \lambda_j} = g_j$, we only need to consider the original constraints and the additional $\frac{\partial f_P}{\partial \vec{z}} = 0$.
- This is a necessary condition, but not a sufficient one.

Applying the Lagrange multipliers method to:

$$\begin{array}{ll} \arg\min_{\vec{w},b} & \|\vec{w}\|^2/2 \\ \text{s.t.} & c_i(\langle \vec{w}, \vec{x}_i \rangle - b) \geq 1 \\ & 1 \leq i \leq N \end{array}$$

gets the (primal) Lagrangian

$$f_P(\vec{w}, b, \lambda_1, \ldots, \lambda_N) = \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{N} \lambda_i \left(c_i(\langle \vec{w}, \vec{x}_i \rangle - b) - 1\right)$$

and then by setting its gradient w.r.t. $\vec{w}$ and b to zero we get

$$\frac{\partial}{\partial \vec{w}} f_P = 0 \qquad \vec{w} = \sum_{i=1}^{N} \lambda_i c_i \vec{x}_i$$

$$\frac{\partial}{\partial b} f_P = 0 \qquad \sum_{i=1}^{N} \lambda_i c_i = 0$$

Support vector machines

Lagrange multipliers

We can now substitute $\vec{w} = \sum_{i=1}^{N} \lambda_i c_i \vec{x}_i$ in the linear constraints to get N inequalities:

$$\sum_{j=1}^{N} c_i c_j \ \vec{x}_i, \vec{x}_j \ \lambda_j - c_i b \geq 1 \qquad i = 1, \ldots, N$$

If these were equality constraints we could solve the set of $N+1$ equations (together with $\sum_{i=1}^{N} \lambda_i c_i = 0$) in $N+1$ variables $(b, \lambda_1, \ldots, \lambda_N)$. However, since these are inequalities this direct approach does not apply here.

To solve this problem we apply two advanced optimization techniques.

Primal-dual transformation: By substituting $\vec{w} = \sum_{i=1}^{N} \lambda_i c_i \vec{x}_i$ into the primal Lagrangian we get:

$$f_P = \frac{1}{2} \sum_{i,j=1}^{N} c_i c_j \lambda_i \lambda_j \ \vec{x}_i, \vec{x}_j \ - \sum_{i,j=1}^{N} c_i c_j \lambda_i \lambda_j \ \vec{x}_i, \vec{x}_j \ + b \sum_{i=1}^{N} c_i \lambda_i + \sum_{i=1}^{N} \lambda_i$$

Then, using $\sum_{i=1}^{N} \lambda_i c_i = 0$ and simplifying gets us the dual Lagrangian:

$$f_D(\lambda_1, \ldots, \lambda_N) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} c_i c_j \lambda_i \lambda_j \ \vec{x}_i, \vec{x}_j$$

Primal-dual transformation: Find the maximum of the dual:

$$f_D(\lambda_1, \ldots, \lambda_N) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} c_i c_j \lambda_i \lambda_j \ \vec{x}_i, \vec{x}_j$$

Notice that:

- Like the primal, the dual is also a quadratic form
- Unlike the primal, the dual does not depend on $\vec{w}, b$, but just on the Lagrange multipliers $\lambda_1, \ldots, \lambda_N$
- While the stationary point of the primal is a minimum, for the dual it's a maximum

Primal-dual transformation: Find the maximum of the dual:

$$f_D(\lambda_1, \ldots, \lambda_N) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} c_i c_j \lambda_i \lambda_j \ \vec{x}_i, \vec{x}_j$$

KKT conditions: It can be shown that solution of the original minimization satisfies for every $i = 1, \ldots, N$:

$$\lambda_i \geq 0 \qquad \text{and} \qquad \lambda_i \left(c_i (\ \vec{w}, x_i \ - b) - 1 \right) = 0$$

Therefore, both $\vec{w}$ and b can be computed from $\lambda_1, \ldots, \lambda_N \geq 0$ that maximize the dual, which in turn can be numerically estimated using Quadratic Programming.

The margin boundaries are given by $\langle \vec{w}, \vec{x}_i \rangle - b = c_i = \pm 1$, so any $\vec{x}_i$ that does not lie on them must have $\lambda_i = 0$, since $\lambda_i \left(c_i (\langle \vec{w}, \vec{x}_i \rangle - b) - 1 \right) = 0$. Furthermore, since $\sum_{i=1}^{N} \lambda_i c_i \vec{x}_i = \vec{w} = 0$ (otherwise the constraints become $\langle \vec{w}, \vec{x}_i \rangle - c_i b \geq 1$ and are violated by one of the classes), some λ_i must be nonzero.

Support vectors

Support vectors are the data points in $\{\vec{x}_i | \lambda_i = 0, i = 1, \ldots, N\} =$.
These vectors all lie exactly on the margin boundaries.

Both $\vec{w}$ and b only depend on the identified support vectors. However, due to numerical errors, these vectors may yield several estimations of b, so in practice the average estimation is used.

Support vector machines

Soft margin

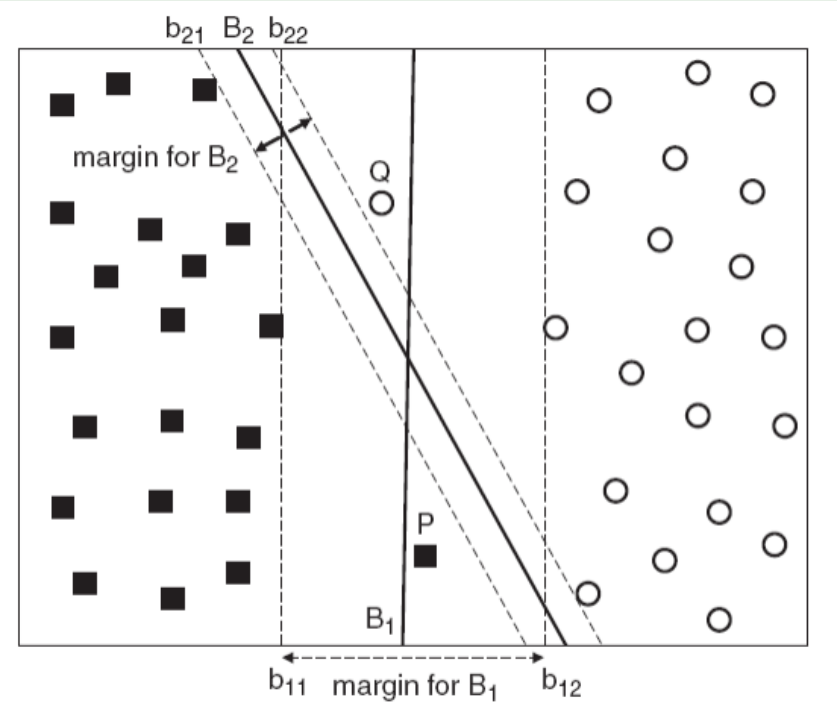
Question: does SVM always choose the best separation margin?

Support vector machines

Soft margin

Question: does SVM always choose the best separation margin?

Example

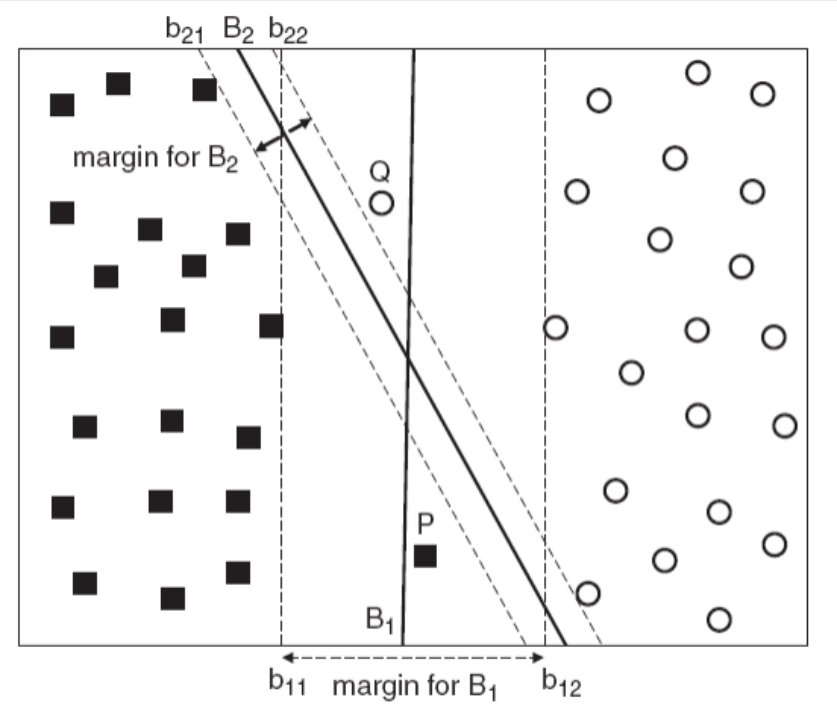


Support vector machines

Soft margin

Question: does SVM always choose the best separation margin?

Example



Conclusion: rigid constraints make the described SVM sensitive to noise and outliers!

Support vector machines

Soft margin

We can relax the SVM model by adding slack variables $\xi_1, \ldots, \xi_N \geq 0$ and reformulating the constraints as:

$$_{1 \leq i \leq N} c_i(\ \vec{w}, \vec{x}_i\ \ - b) \geq 1 - \xi_i$$

So in the positive class ($c_i = 1$) we get $\vec{w}, \vec{x}_i\ \ - b \geq 1 - \xi_i$ and in the negative class ($c_i = -1$) we get $\vec{w}, \vec{x}_i\ \ - b \leq -1 + \xi_i$.

The new constraints allow data points to lie within the margin, or even on the "wrong" side of the decision boundary, while the slack $\xi_i \geq 0$ quantifies how far x_i is from the satisfying the original constraints.

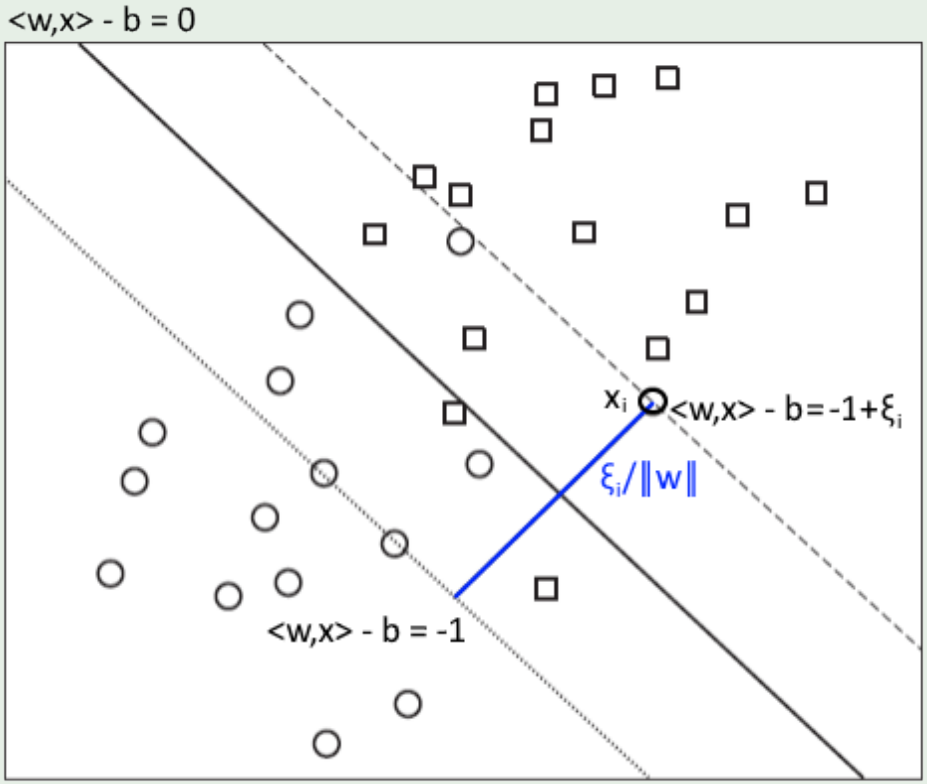
Similar to the Lagrange weight $\lambda_i \geq 0$, when $\xi_i = 0$ the data point x_i is well classified away from the margin boundary.

Support vector machines

Soft margin

The slack variables not only make SVM more robust, but they also enable classification for nonseparable (or nonlinearly separable) data:

Example

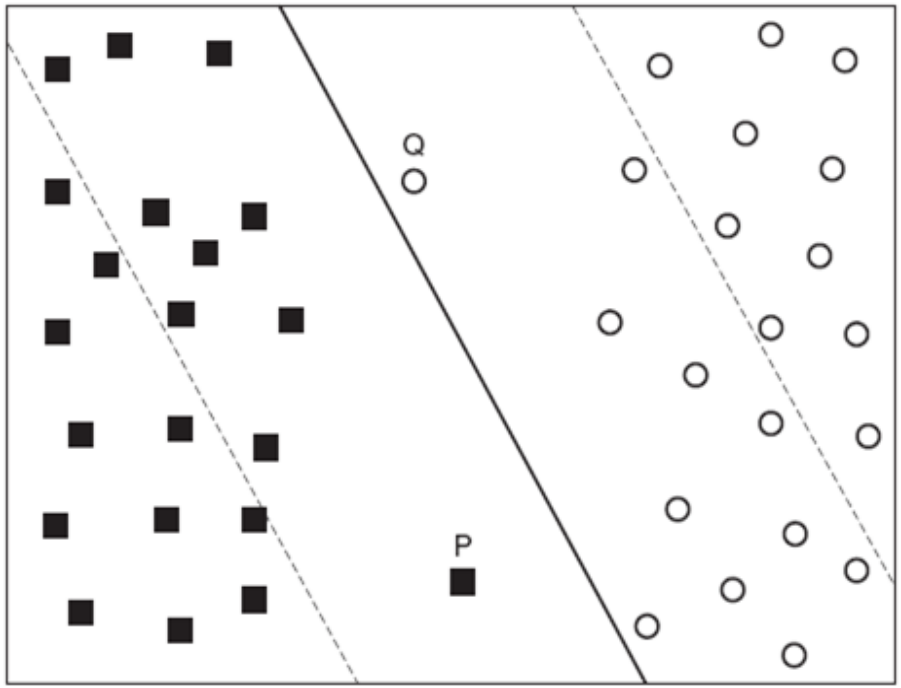


Support vector machines

Soft margin

With the new constraints, many choices of hyperplane and margin can be fit to satisfy them with appropriate slack values.

Problem: the original maximal margin optimization target is insufficient with these constraints, since it does not control the amount or magnitude of classification errors in the model. We can easily get a wide margin where most of the data is misclassified or lies in the margin.



Solution: adapt the target function to also minimize the slack values.

Support vector machines

Soft margin

The soft-margin SVM training uses the following optimization problem, where β is some user-configurable constant:

$$\begin{array}{ll} \arg\min_{\vec{w},b,\xi_1,\ldots,\xi_N} & \|\vec{w}\|^2/2 + \beta\sum_{i=1}^{N}\xi_i \\ \text{s.t.} \quad {\scriptstyle 1\le i\le N} & c_i(\langle\vec{w},\vec{x}_i\rangle - b) \ge 1-\xi_i \\ & \xi_i \ge 0 \end{array}$$

Applying Lagrange multipliers gives the primal Lagrangian:

$$\begin{aligned} L_P = {} & \|\vec{w}\|^2/2 + \beta\sum_{i=1}^{N}\xi_i \\ & - \sum_{i=1}^{N}\lambda_i(c_i(\langle\vec{w},\vec{x}_i\rangle - b) - 1 + \xi_i) - \sum_{i=1}^{N}\mu_i\xi_i \end{aligned}$$

Applying Lagrange multipliers gives the primal Lagrangian:

$$L_P = \|\vec{w}\|^2/2 + \beta \sum_{i=1}^{N} \xi_i$$

$$- \sum_{i=1}^{N} \lambda_i (c_i(\langle \vec{w}, \vec{x}_i \rangle - b) - 1 + \xi_i) - \sum_{i=1}^{N} \mu_i \xi_i$$

Setting $L_P = 0$ gives:

$$\frac{\partial L_P}{\partial \vec{w}[j]} = 0 \qquad \vec{w}[j] - \sum_{i=1}^{N} \lambda_i c_i x_i[j] = 0 \qquad \vec{w} = \sum_{i=1}^{N} \lambda_i c_i \vec{x}_i$$

$$\frac{\partial L_P}{\partial b} = 0 \qquad - \sum_{i=1}^{N} \lambda_i c_i \qquad \sum_{i=1}^{N} \lambda_i c_i = 0$$

$$\frac{\partial L_P}{\partial \xi_i} = 0 \qquad \beta - \lambda_i - \mu_i = 0 \qquad \lambda_i + \mu_i = \beta$$

Thus, we get the dual:

$$
\begin{aligned}
L_D &= \frac{1}{2}\sum_{i,j=1}^{N} \lambda_i \lambda_j c_i c_j \ \langle x_i, x_j \rangle + \beta \sum_{i=1}^{N} \xi_i - \sum_{i,j=1}^{N} \lambda_i \lambda_j c_i c_j \ \langle x_i, x_j \rangle \\
&\quad + b\sum_{i=1}^{N} c_i \lambda_i + \sum_{i=1}^{N} \lambda_i - \sum_{i=1}^{N} \lambda_i \xi_i - \sum_{i=1}^{N} (\beta - \lambda_i)\xi_i \\
&= \sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i,j=1}^{N} \lambda_i \lambda_j c_i c_j \ \langle x_i, x_j \rangle
\end{aligned}
$$

This dual is the same as the linearly separable one. However, in this case the KKT conditions require $\lambda_i \geq 0$ and $\mu_i \geq 0$, $i = 1, \ldots, N$, so together with $\lambda_i + \mu_i = \beta$ we get a constraint $0 \leq \lambda_i \leq \beta$.

Support vector machines

Soft margin

Finally, soft-margin SVM applies quadratic programming to solve:

$$\begin{aligned} \arg\max_{\lambda_1,\ldots,\lambda_N} \quad & \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j c_i c_j \ \langle x_i, x_j \rangle \\ \text{s.t.} \quad _{1 \le i \le N} \quad & 0 \le \lambda_i \le \beta \\ & \sum_{i=1}^{N} c_i \lambda_i = 0 \end{aligned}$$

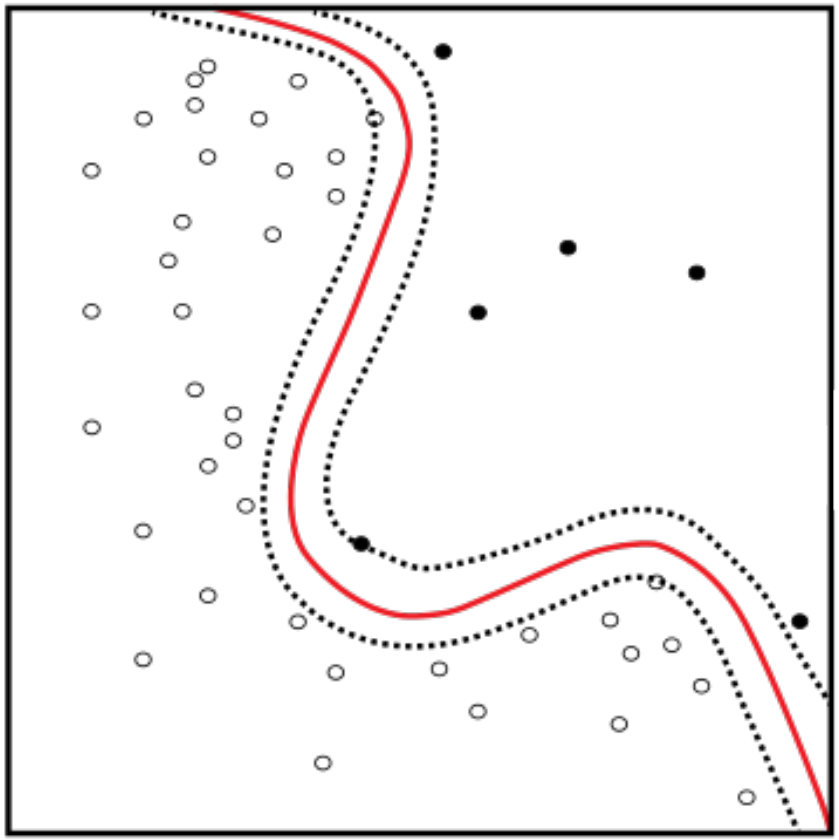
Then set $\mu_i = \beta - \lambda_i$, $\vec{w} = \sum_{i=1}^{N} c_i \lambda_i x_i$, and recover $b, \xi_1, \ldots, \xi_n$ from the KKT conditions, which provide a sufficient set of linear equations:

$$\mu_i \xi_i = 0 \quad \text{and} \quad \lambda_i \left(c_i (\langle \vec{w}, x_i \rangle - b) - 1 + \xi_i \right) = 0 \quad \text{and} \quad \xi_i \ge 0$$

Support vectors in this case consist of data points on the margin boundaries or with $\xi_i > 0$ (i.e., misclassified and inside the margin). Notice that the resulting classifier is still a linear SVM, so it does not require the slack variables (i.e., it only uses $\vec{w}$ and b as shown before).

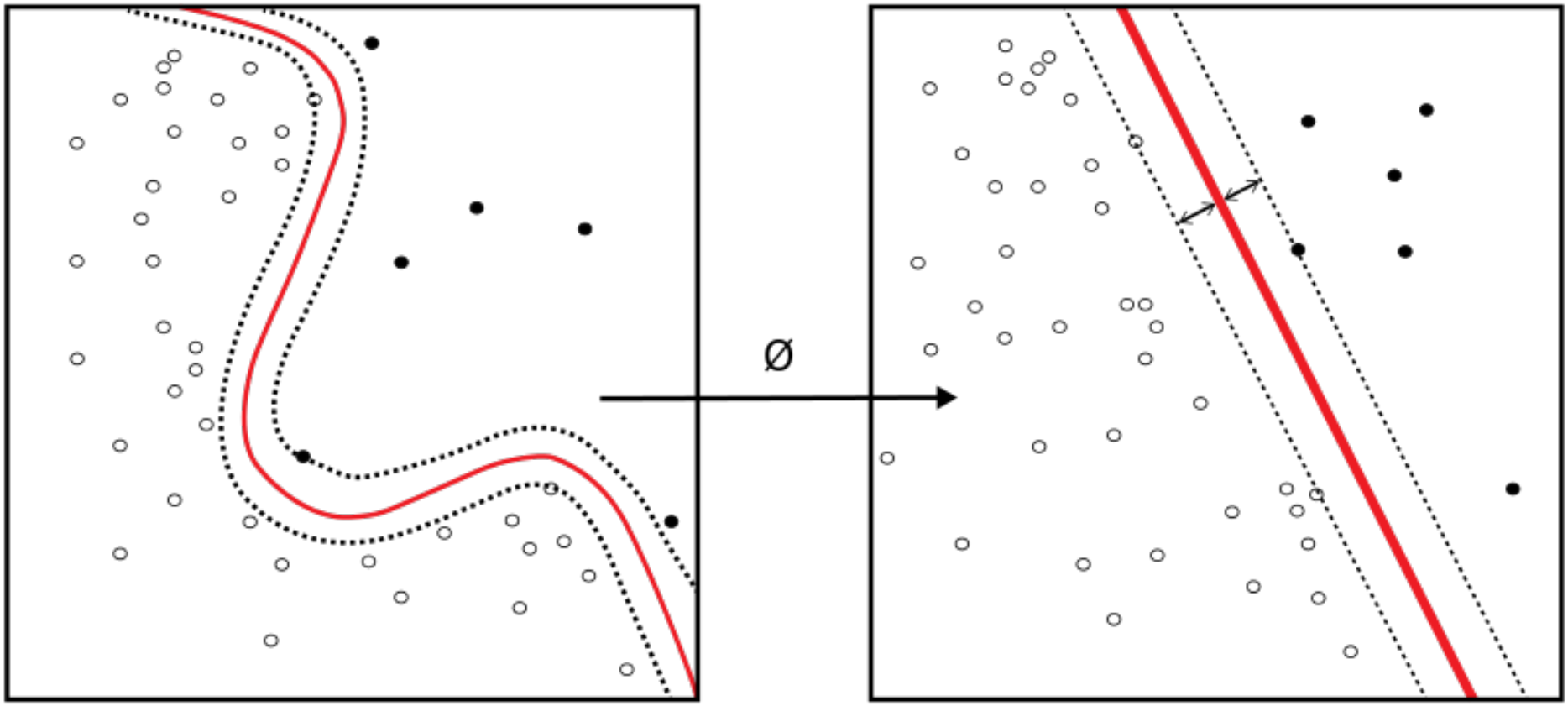
Nonlinear SVM

What if the data is not linearly separable? **Question:** Can we find a nonlinear decision boundary?



Nonlinear SVM

What if the data is not linearly separable? **Question:** Can we find a nonlinear decision boundary?

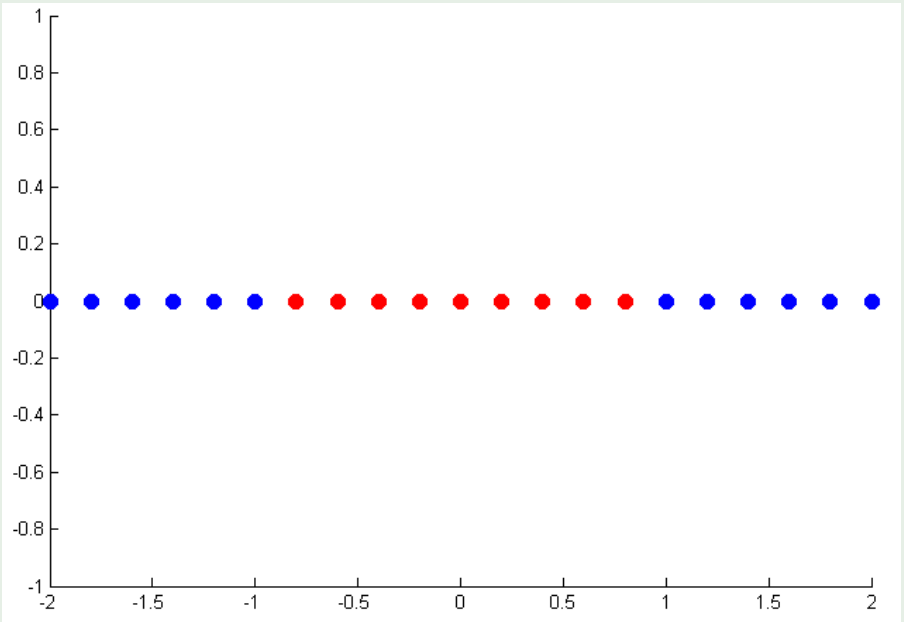


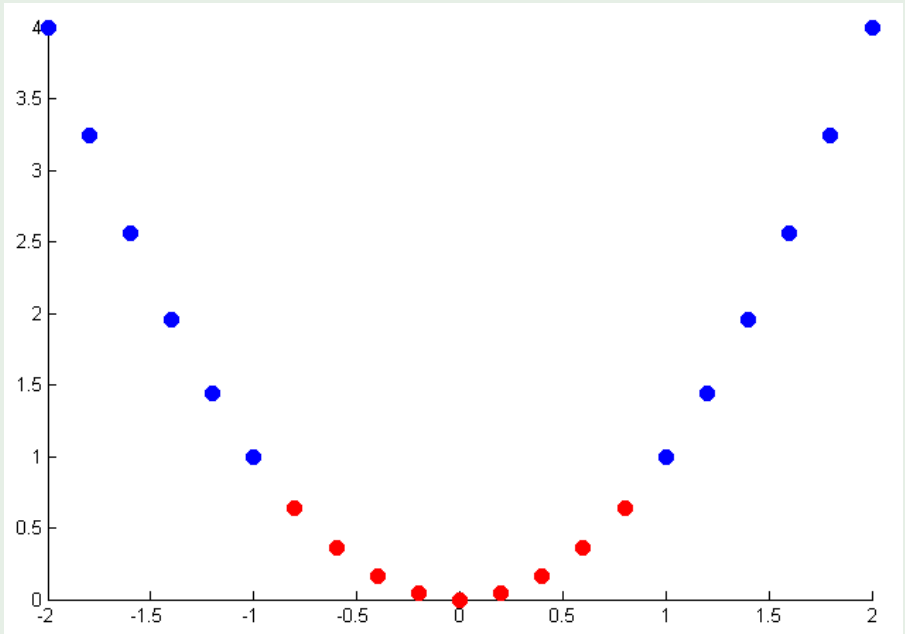
Solution: use a feature map $x \rightarrow \Phi(x)$ to transform the data to a new feature space where it is linearly separable.

Nonlinear SVM

Typically, increasing the dimensionality of nonlinearly separable data can transforms them to a linearly separable representation.

Example

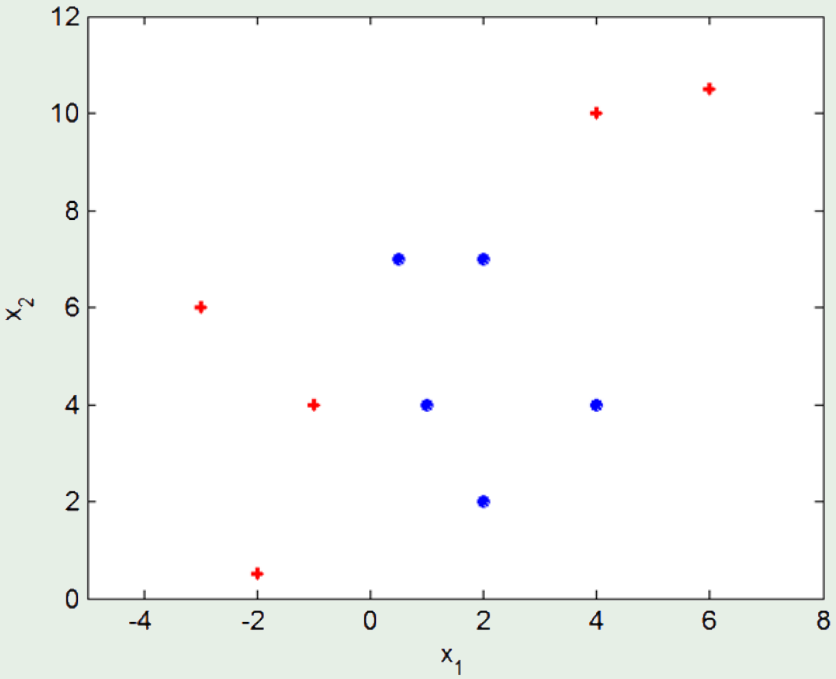




$$x \quad \mathbb{R} \rightarrow (x, x^2) \quad \mathbb{R}^2$$

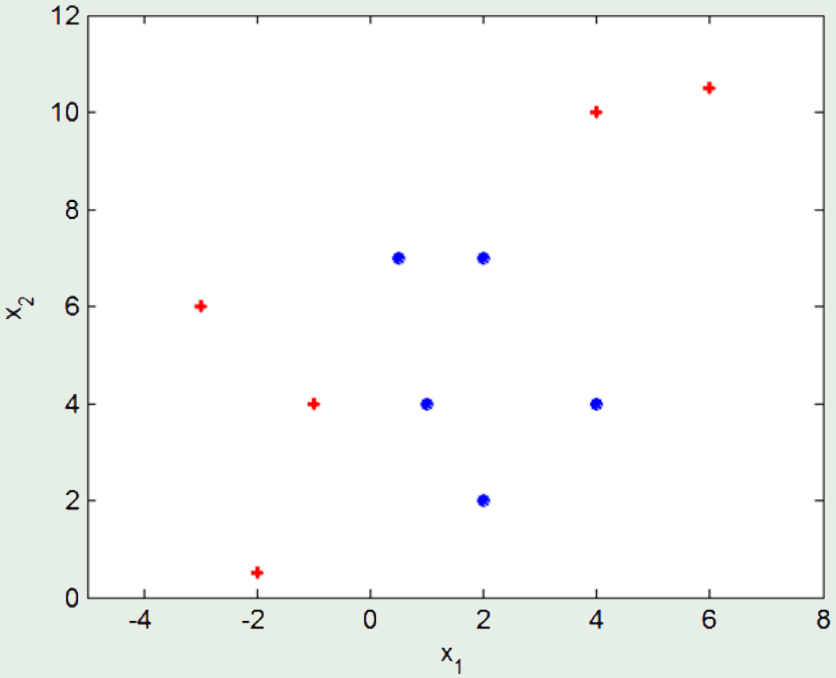
Nonlinear SVM

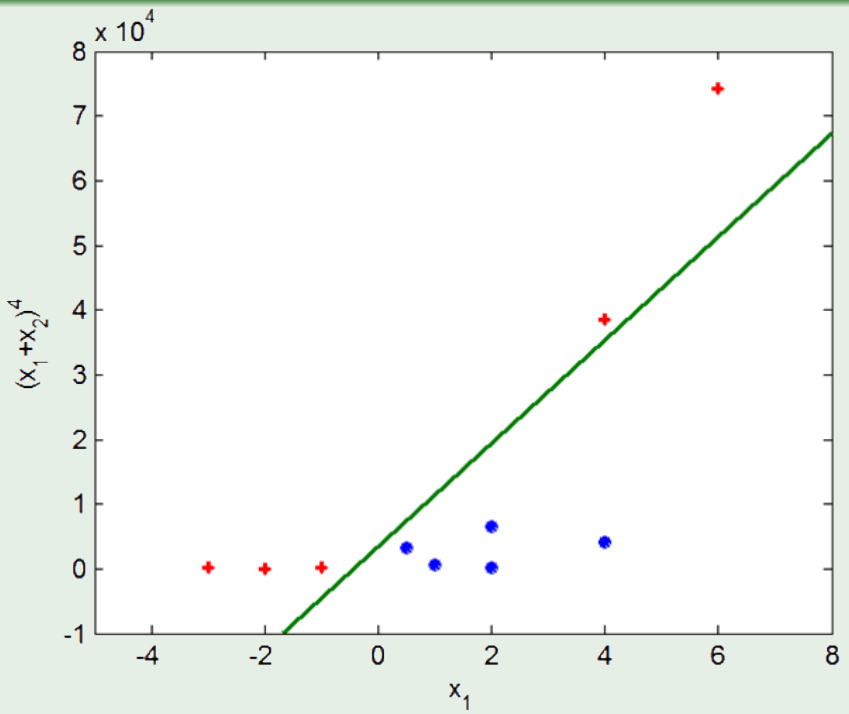
Example



Nonlinear SVM

Example

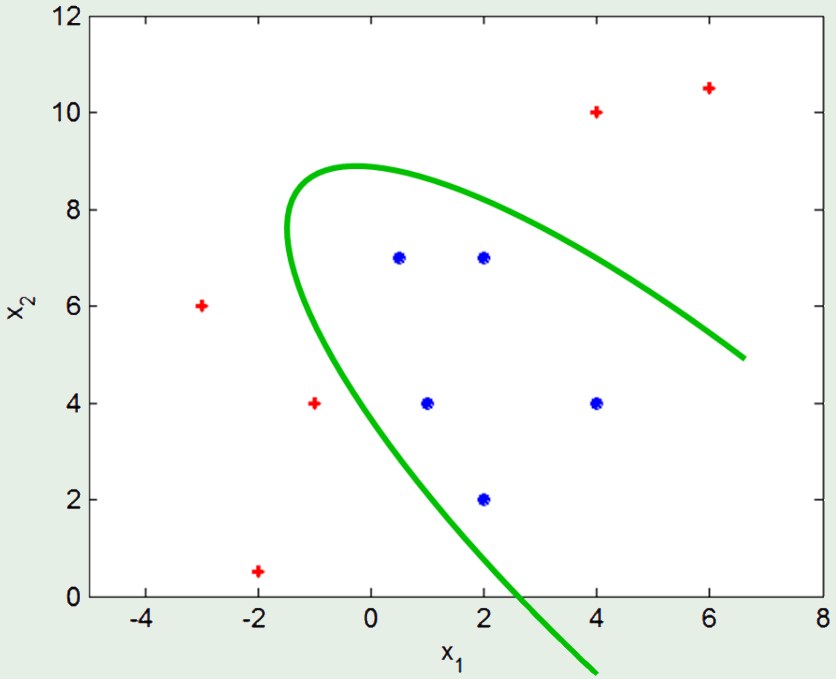


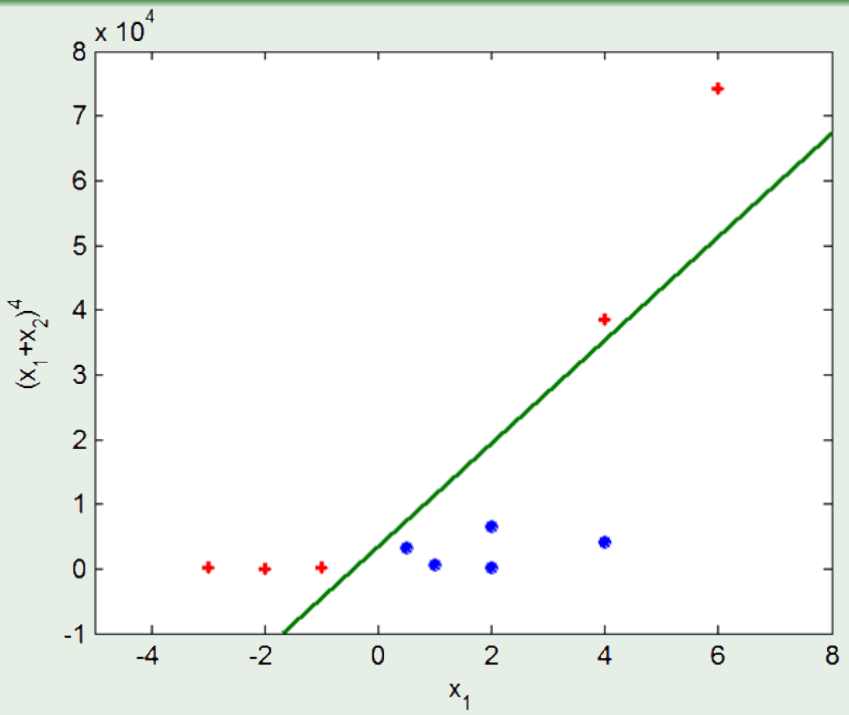


$$\Phi(x) = (x[1], (x[1] + x[2])^4)$$

Nonlinear SVM

Example

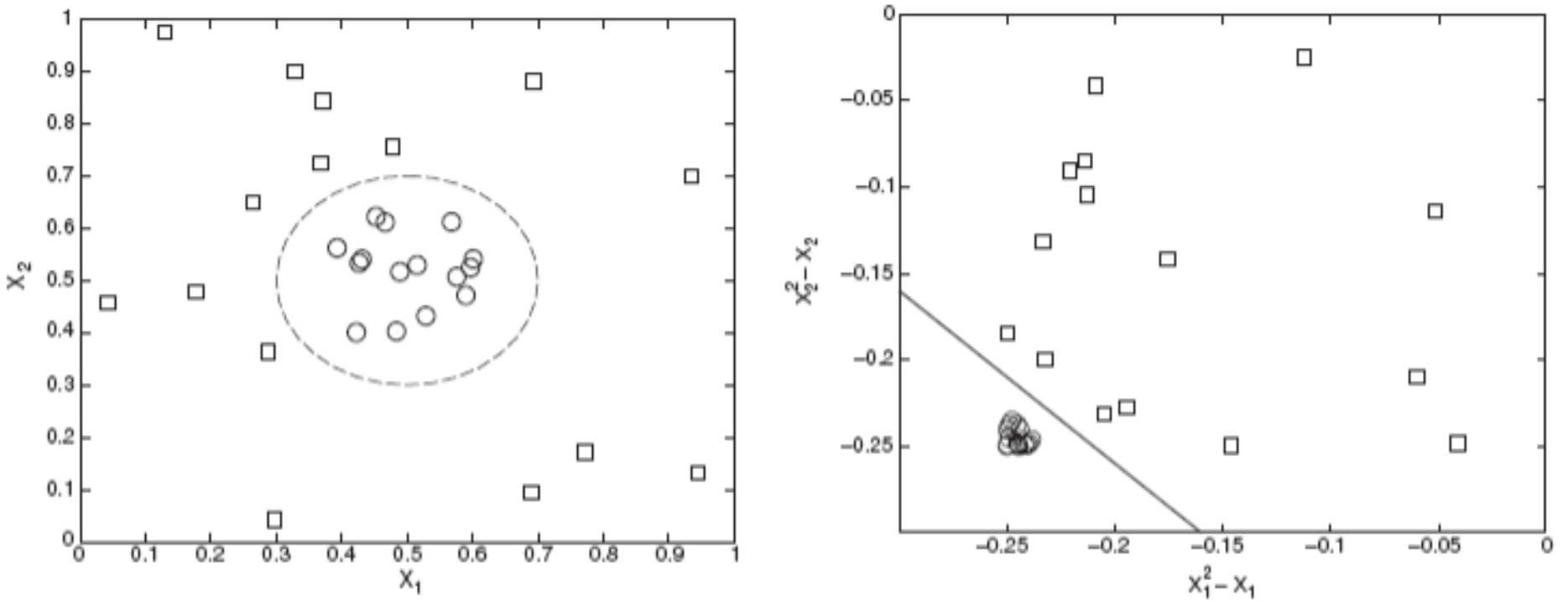




$$\Phi(x) = (x[1], (x[1] + x[2])^4)$$

Nonlinear SVM

Example



$$\Phi(x) = (x[1]^2 - x[1], x[2]^2 - x[2])$$

Nonlinear SVM

Feature extraction

Using Φ SVM can be reformulated to use these features instead of the original data, which doesn't have to be in $\mathbb{R}^n$ anymore:

$$\begin{aligned} &\arg\min_{\vec{w},b} && \|\vec{w}\|^2/2 \\ &\text{s.t.} \quad _{1\le i\le N} && c_i(\langle \vec{w}, \Phi(x_i)\rangle - b) \ge 1 \end{aligned}$$

Nonlinear SVM

Feature extraction

Using Φ SVM can be reformulated to use these features instead of the original data, which doesn't have to be in $\mathbb{R}^n$ anymore:

$$\begin{aligned} &\arg\min_{\vec{w},b} && \|\vec{w}\|^2/2 \\ &\text{s.t.} \quad _{1\le i\le N} && c_i(\langle \vec{w}, \Phi(x_i)\rangle - b) \ge 1 \end{aligned}$$

The optimization follows the same steps as before, and classification considers $\langle \vec{w}, \Phi(y)\rangle - b$ for a new data point y.

Nonlinear SVM

Feature extraction

Using Φ SVM can be reformulated to use these features instead of the original data, which doesn't have to be in $\mathbb{R}^n$ anymore:

$$\begin{aligned} &\arg\min_{\vec{w},b} && \|\vec{w}\|^2/2 \\ &\text{s.t.} \quad _{1\le i\le N} && c_i(\langle \vec{w}, \Phi(x_i)\rangle - b) \ge 1 \end{aligned}$$

The optimization follows the same steps as before, and classification considers $\langle \vec{w}, \Phi(y)\rangle - b$ for a new data point y.

Feature maps can either be **designed** (e.g., filter banks and the scattering transform) or **learned** from data (e.g., deep learning). However, extracting suitable features may not always clear or convenient.

Nonlinear SVM

Feature extraction

Using Φ SVM can be reformulated to use these features instead of the original data, which doesn't have to be in $\mathbb{R}^n$ anymore:

$$\begin{aligned} &\arg\min_{\vec{w},b} && \|\vec{w}\|^2/2 \\ &\text{s.t.} \quad _{1\le i\le N} && c_i(\langle \vec{w}, \Phi(x_i)\rangle - b) \ge 1 \end{aligned}$$

The optimization follows the same steps as before, and classification considers $\langle \vec{w}, \Phi(y)\rangle - b$ for a new data point y.

Feature maps can either be **designed** (e.g., filter banks and the scattering transform) or **learned** from data (e.g., deep learning). However, extracting suitable features may not always clear or convenient.

Question: can we use relations between data points (e.g., distances, similarities, or dissimilarities) instead of directly using input or extracted features?

We begin by treating inner products as relations between data points, and consider the dual optimization of SVM:

$$\begin{array}{ll} \arg\max_{\lambda_1,\ldots,\lambda_N} & \sum_{i=1}^N \lambda_i - \frac{1}{2}\sum_{i,j=1}^N c_i c_j \lambda_i \lambda_j \ \Phi(x_i), \Phi(x_j) \\ \text{s.t.} \quad 1 \le i \le N & \lambda_i \ge 0 \\ & \sum_{i=1}^N c_i \lambda_i = 0 \end{array}$$

Notice that this optimization does not require access to explicit features, but just inner products $\Phi(x_i), \Phi(x_j)$

Therefore, the Lagrange multipliers $\lambda_1, \ldots, \lambda_N$ can be recovered by only using inner products of features.

Nonlinear SVM

Kernel trick

Recall that $\vec{w} = \sum_{i=1}^{N} c_i \lambda_i \Phi(x_i)$, and thus we can reformulate the extraction of b as:

$$\lambda_j \left(c_j \left(\vec{w}, \Phi(x_j) \right) - b \right) - 1 = 0 \qquad \lambda_j = 0$$

$$c_j \left(\vec{w}, \Phi(x_j) \right) - b = 1 \overset{c_j = \pm 1}{=\!=} \vec{w}, \Phi(x_j) - b = c_j$$

$$b = \vec{w}, \Phi(x_j) - c_j = \sum_{i=1}^{N} c_i \lambda_i \Phi(x_i), \Phi(x_j) - c_j$$

$$b = \text{mean} \left\{ \sum_{i=1}^{N} c_i \lambda_i \Phi(x_i), \Phi(x_j) - c_j \middle| \lambda_j > 0, j = 1, \ldots, N \right\}$$

Therefore, b can also be recovered by only using inner products, without access to the features themselves.

Nonlinear SVM

Kernel trick

The last step of the SVM algorithm is classifying a new data point y by testing $\langle \vec{w}, \Phi(y) \rangle - b$, but again using $\vec{w} = \sum_{i=1}^{N} c_i \lambda_i \Phi(x_i)$ we get the following classification rule:

$$\sum_{i=1}^{N} c_i \lambda_i \langle \Phi(x_i), \Phi(y) \rangle - b \begin{cases} (-\infty, -1] & \text{class is } - \\ (-1, 0) & \text{class is probably } - \\ [0, 1) & \text{class is probably } + \\ [1, \infty) & \text{class is } + \end{cases}$$

This rule does not rely on recovering $\vec{w}$, so once we have b and $\lambda_1, \ldots, \lambda_N$ the classification can be done based on inner products, without requiring explicit features.

Nonlinear SVM

Kernel trick

Since SVM only relies on inner products, it can be reformulated using a **kernel function** $k : X \times X \to \mathbb{R}$, s.t. $k(x, y) = \langle \Phi(x), \Phi(y) \rangle$ instead of explicitly using Φ.

It is often **simpler to formulate kernels** than feature maps, and they **do not require** knowledge of the feature space **dimensionality**.

Example

For a $\Phi : \mathbb{R}^2 \to \mathbb{R}^6$ feature map $\Phi(u) = (u[1]^2, u[2]^2, \sqrt{2}u[1], \sqrt{2}u[2], \sqrt{2}u[1]u[2], 1)$, we can use the quadratic kernel $k(x, y) = \langle x, y \rangle + 1)^2 = x[1]^2 y[1]^2 + x[2]^2 y[2]^2 + 2x[1]y[1] + 2x[2]y[2] + 2x[1]x[2]y[1]y[2] + 1 = \langle \Phi(x), \Phi(y) \rangle$ without directly computing Φ.

Nonlinear SVM

Mercer theorem and reproducing kernel Hilbert space

Question: how do we know which functions can be used as a kernel without considering feature maps?

Nonlinear SVM

Mercer theorem and reproducing kernel Hilbert space

Question: how do we know which functions can be used as a kernel without considering feature maps?

Mercer theorem

If $k(x, y)$ is symmetric, continuous, and positive semidefinite (i.e., it satisfies $\int_S \int_S k(x, y)g(x)g(y)dxdy \geq 0$, where L^2 is the space of square integrable functions over the data) then there exist a function Φ such that $k(x, y) = \langle \Phi(x), \Phi(y) \rangle$.

Question: how do we know which functions can be used as a kernel without considering feature maps?

Mercer theorem

If $k(x, y)$ is symmetric, continuous, and positive semidefinite (i.e., it satisfies $\int_S \int_S k(x, y)g(x)g(y)dxdy \geq 0$, where L^2 is the space of square integrable functions over the data) then there exist a function Φ such that $k(x, y) = \langle \Phi(x), \Phi(y) \rangle$.

A **mercer kernel**, together with the ambient space of the data, define an inner product space. In continuous settings, such spaces are known as **reproducing kernel Hilbert spaces (RKHS)** and are extensively studied in functional analysis.

Kernel SVM training

Input:

- Mercer kernel matrix $K \quad \mathbb{R}^{N\times N}$ s.t. $K_{ij} = k(x_i, x_j)$
- Class labels $c_1, \ldots, c_N \quad \{-1, 1\}$

Solve the margin maximization problem:

$$\begin{array}{ll} \arg\max_{\lambda_1,\ldots,\lambda_N} & \sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i,j=1}^{N} c_i c_j \lambda_i \lambda_j K_{ij} \\ \text{s.t.} \quad {\scriptstyle 1\le i\le N} & \lambda_i \geq 0 \\ & \sum_{i=1}^{N} c_i \lambda_i = 0 \end{array}$$

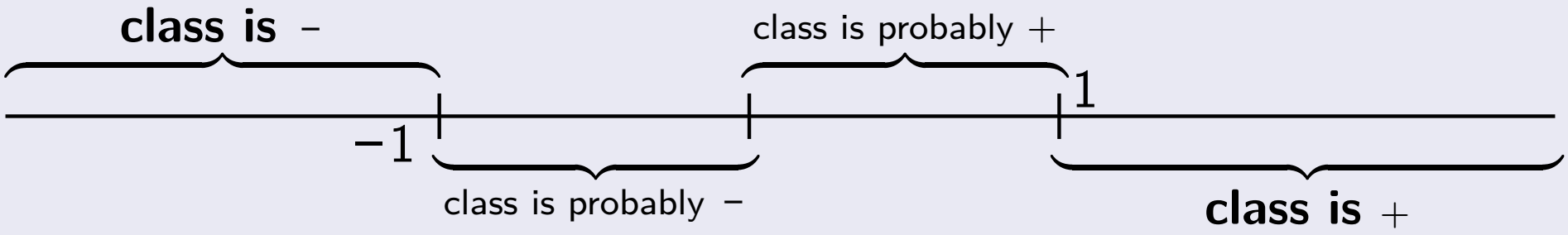
Compute: $b = \text{mean}\{\sum_{i=1}^{N} c_i \lambda_i K_{ij} - c_j | \lambda_j > 0, j = 1, \ldots, N\}$
Output: bias b and weight vector $\vec{v} \quad \mathbb{R}^N$ s.t. $\vec{v}[i] = c_i \lambda_i$
$i = 1, \ldots, N$

Kernel SVM classifier

Input:

- New kernel row $\vec{k}_y \quad \mathbb{R}^N$ s.t. $\vec{k}_y[i] = k(y, x_i)$
- The model parameters $b, \vec{v}$ from training

Classify by value of $\sum_{i=1}^{N} \vec{v}[i]\vec{k}_y[i] - b$:



Notice that this can be optimized to only consider support vectors by discarding all $\{i|\lambda_i = 0\}$, so $\vec{v}$ and $\vec{k}_y$ can be very sparse.

Nonlinear SVM

Popular kernels

Many task- or data-dependent kernels can be defined, but many applications use popular standard kernels, such as:

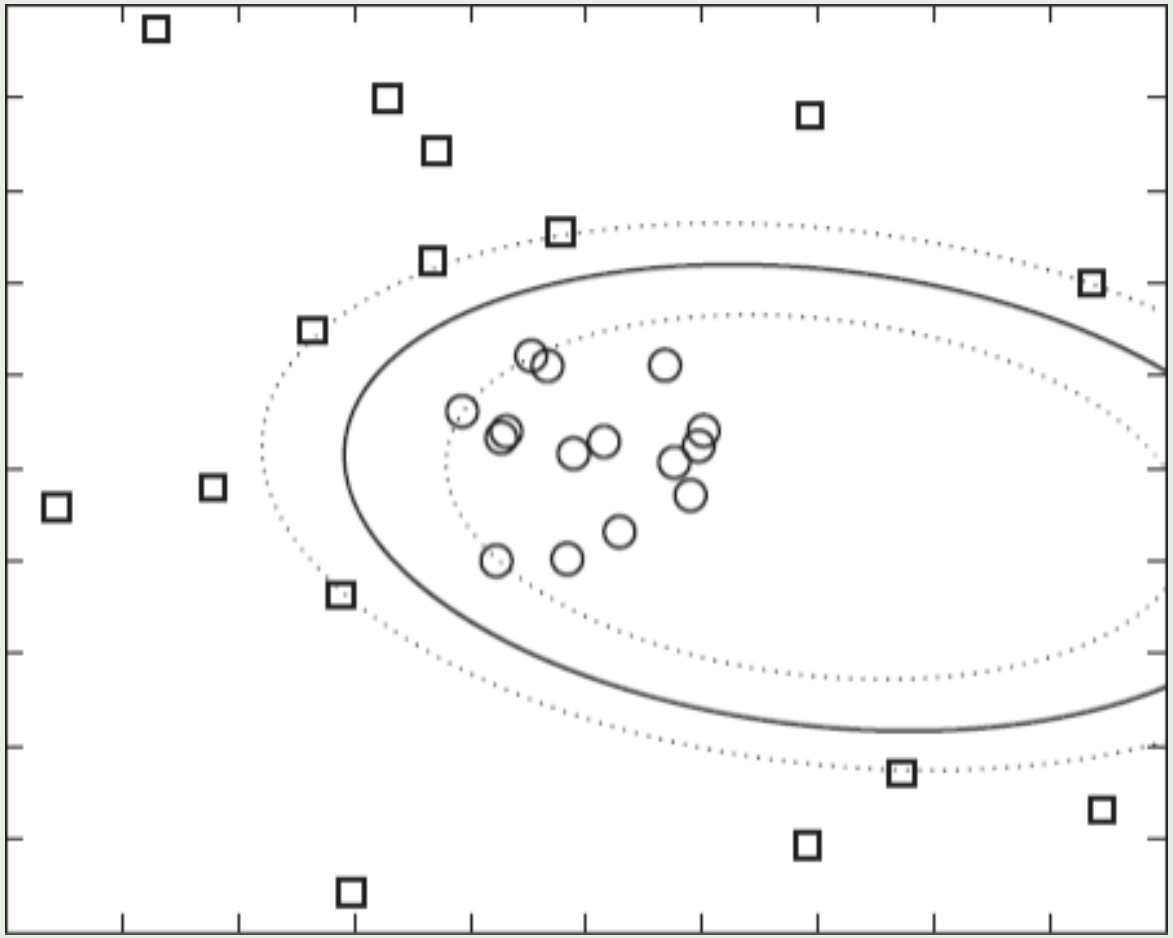
Polynomial kernel

A polynomial kernel is defined as $k(x, y) = (\ x, y\ \ + 1)^p$, for some degree p. Particular degrees include $p = 1$, which yields the classic linear SVM, and $p = 2$, which gives the quadratic kernel.

Radial basis function

An RBF kernel is defined as $k(x, y) = \exp\left(-\frac{\|x - y\|^2}{\sigma^2}\right)$, for some $\sigma > 0$.

Example (Polynomial kernel)



Summary

Linear SVM defines a classification model based on a hyperplane and margin surrounding it.

- Training is done via quadratic programming to maximize the margin width.
- The trained classifier only depends on a small number of support vectors.

Kernel SVM extends this model to nonlinear decision boundaries.

- SVM training only relies on inner products in the data, which are replaced by a Mercer kernel.
- Alternatively, nonlinear featured maps can be designed and learned (e.g., via Deep Learning).
- Kernels are chosen and tuned by cross validation over of several alternatives.

Finally, soft-margin can be applied in both the linear and nonlinear settings to allow robustness to noise and outliers.