



Machine learning

Support Vector Machines

Mohammad-Reza A. Dehaqani

dehaqani@ut.ac.ir

History of SVM

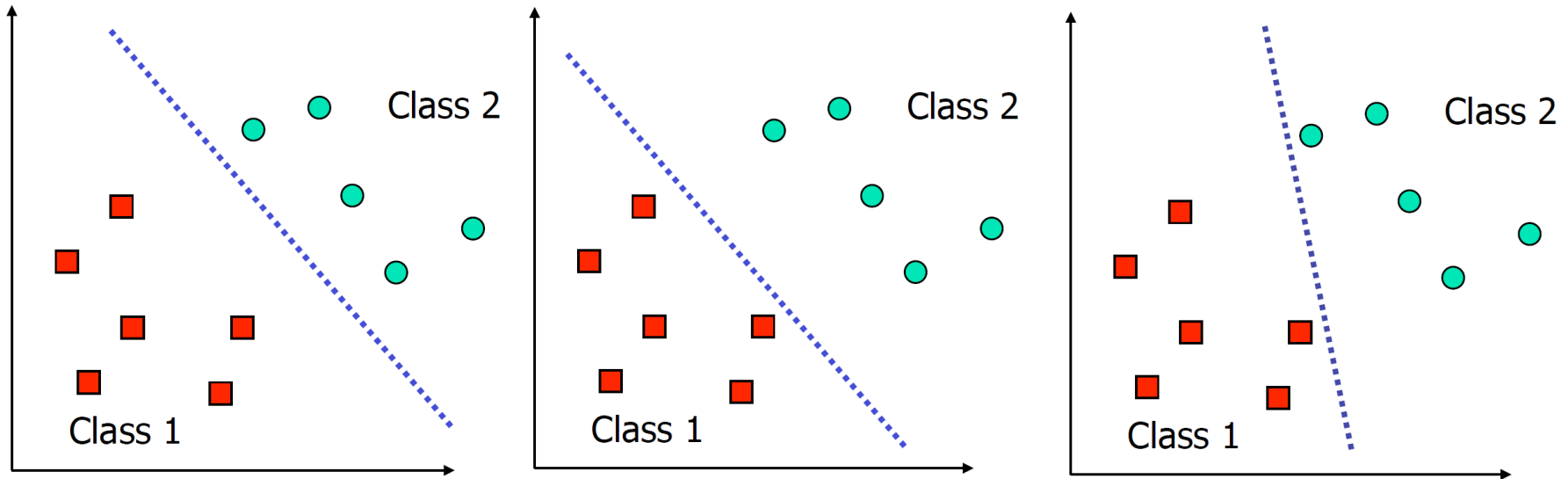


- SVM is related to **statistical learning theory** [3]
- SVM was first introduced in 1992 [1]
- SVM becomes popular because of its success in **handwritten** digit recognition
 - 1.1% test error rate for SVM. This is the same as the error rates of a carefully constructed neural network, LeNet 4.
- SVM is now regarded as an important example of “**kernel methods**”, one of the key area in machine learning
 - Note: the meaning of “kernel” is different from the “kernel” function for **Parzen windows**

What is a good Decision Boundary?



- Are all decision boundaries **equally good**?



- The decision boundary should be **as far away** from the data of **both classes** as possible
 - We should maximize the **margin, m**

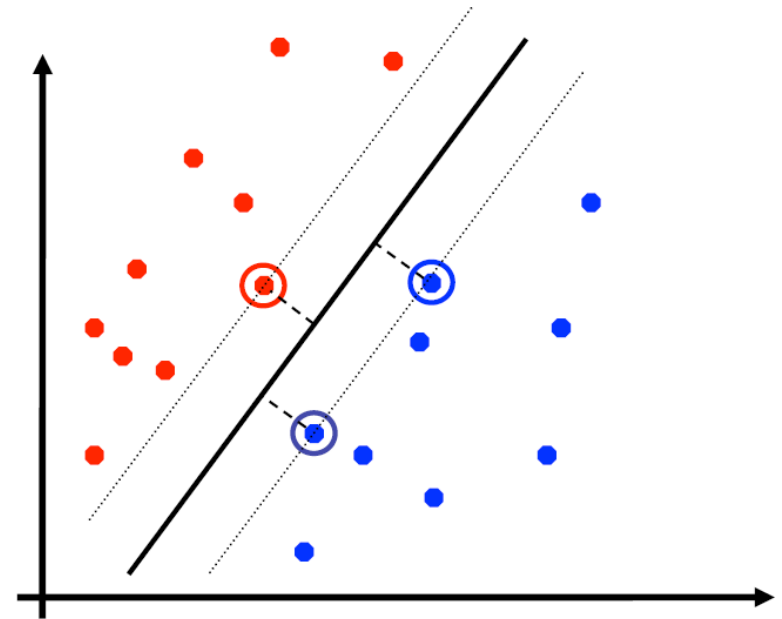
The maximum margin linear classifier is the



- SVMs (Vapnik, 1990's) choose the linear separator with the **largest margin**
- The simplest kind of SVM (Called an LSVM)
- **Support Vectors** are those data points that the **margin pushes** up against
 - These are **the most difficult samples** to classify.
- SVM is **good** according to **intuition, theory (VC dimension), practice**



V. Vapnik



Linear model for classification

bias-variance trade off



- Given a set of **training patterns** and class labels as $(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_n, \mathbf{y}_n) \in \mathbb{R}^d \times \{\pm 1\}$, the goal is to find a classifier function $f: \mathbb{R}^d \rightarrow \{\pm 1\}$ such that $f(\mathbf{x}) = \mathbf{y}$ will **correctly classify** new patterns.

- Support vector machines are based on the class of **hyperplanes**

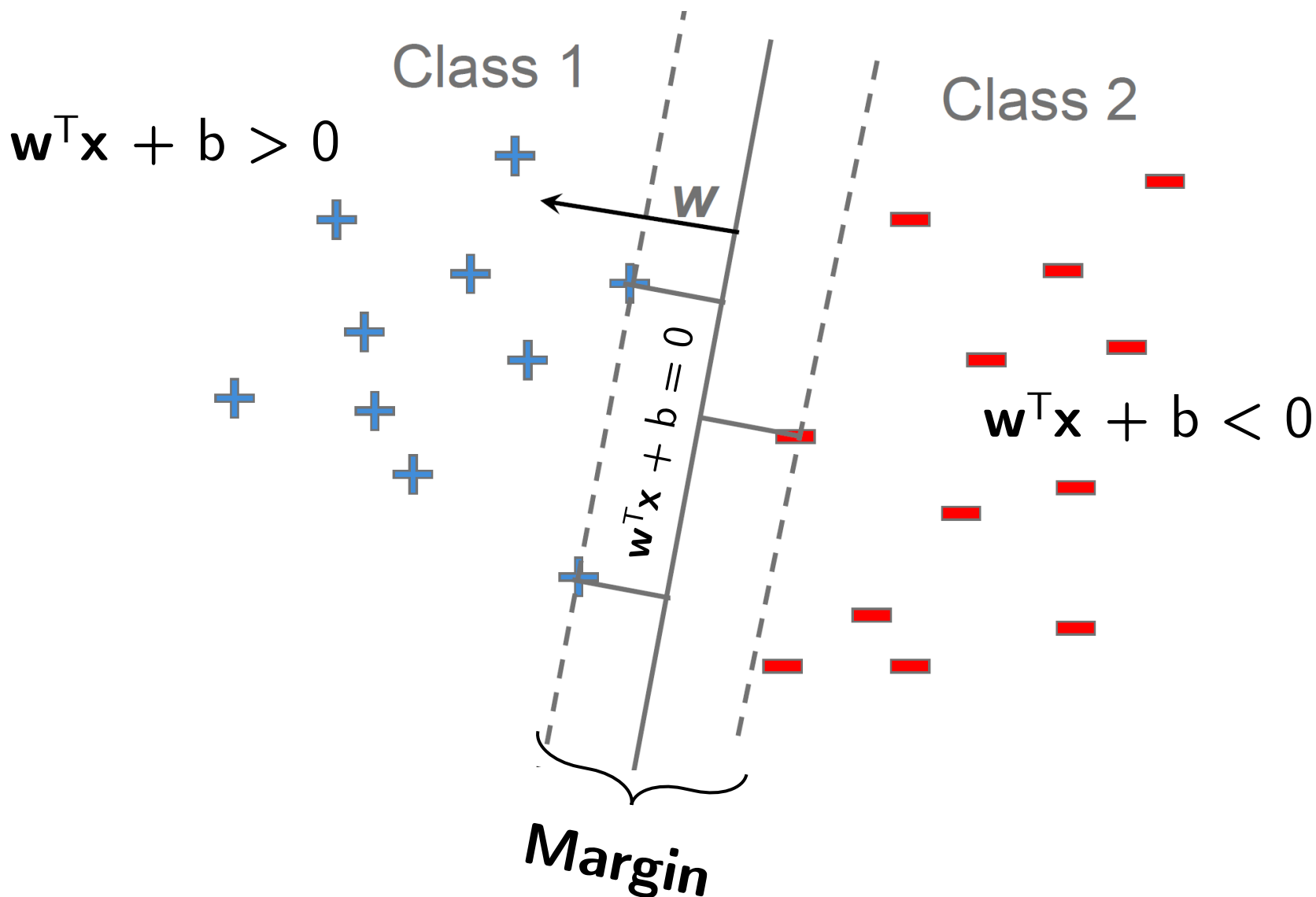
$$(\mathbf{w}^T \cdot \mathbf{x}) + b = 0, \quad \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$$

- Corresponding to **decision functions**

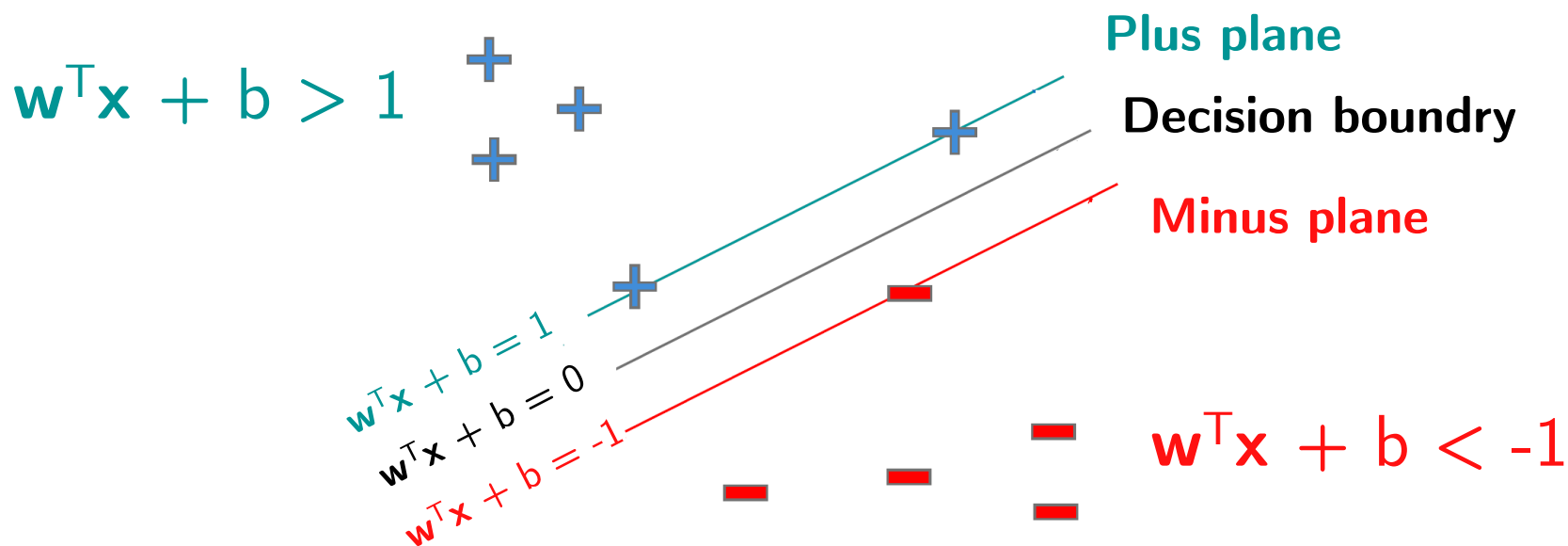
$$f(\mathbf{x}) = \text{sign}((\mathbf{w}^T \cdot \mathbf{x}) + b)$$

- Note that the decision boundary is **unaffected** by the **scaling** of the parameters $(\mathbf{w}, b) \rightarrow (\alpha \mathbf{w}, \alpha b)$.

Pick the one with the largest margin



Scaling



- Classification rule:
 - Classify as: $+1$ if $w^T x + b \geq 1$
 -1 if $w^T x + b \leq -1$
- **Goal:** Find the maximum margin classifier

The Primal Hard SVM



- A binary classification problem of separating **balls** from **diamonds**.
- The optimal hyperplane is orthogonal to the **shortest line** connecting the **convex hulls** of the two classes (dotted), and **intersects it half** way between the two classes.
- There is a weight vector \mathbf{w} and a threshold b such that the points closest to the **hyperplane** satisfy

$$|(\mathbf{w}^T \cdot \mathbf{x}_i) + b| = 1$$

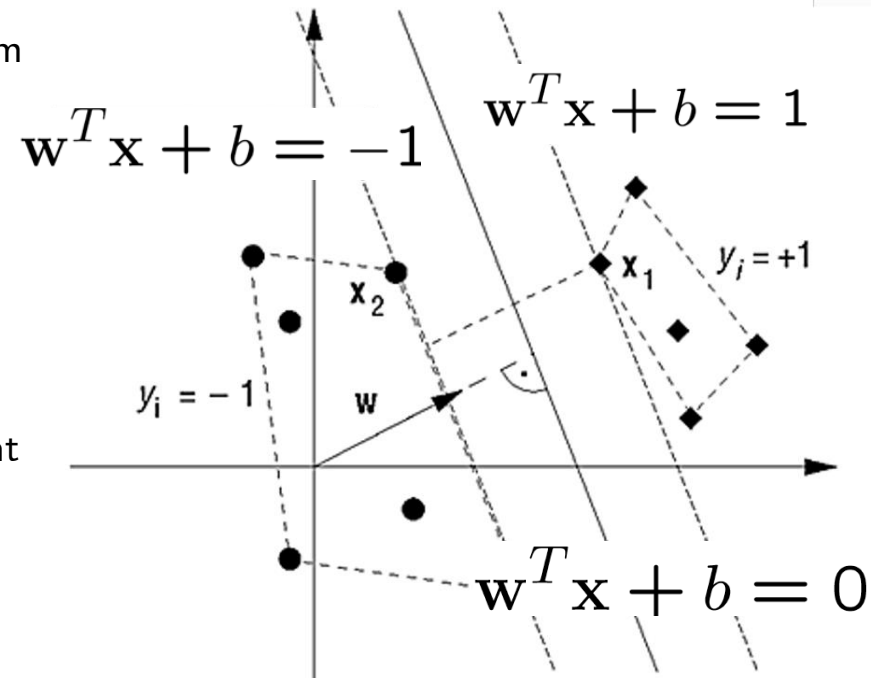
- corresponding to

$$y_i((\mathbf{w}^T \cdot \mathbf{x}_i) + b) \geq 1.$$

- The **margin**, measured perpendicularly to the hyperplane, equals

$$\frac{2}{\|\mathbf{w}\|}$$

- **Distance** between the **origin** and the line $\mathbf{w}^T \mathbf{x} = k$ is $k/\|\mathbf{w}\|$



Note:

$$\begin{aligned} (\mathbf{w}^T \cdot \mathbf{x}_1) + b &= +1 \\ (\mathbf{w}^T \cdot \mathbf{x}_2) + b &= -1 \end{aligned}$$

$$\Rightarrow (\mathbf{w}^T \cdot (\mathbf{x}_1 - \mathbf{x}_2)) = 2$$

$$\Rightarrow \left(\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot (\mathbf{x}_1 - \mathbf{x}_2) \right) = \frac{2}{\|\mathbf{w}\|}$$

Finding the Decision Boundary



- The decision boundary can be found by solving the following **constrained optimization problem**

$$\text{Minimize } \frac{1}{2} ||\mathbf{w}'||^2$$

subject to $y_i(\mathbf{w}'^T \mathbf{x}_i + b) \geq 1 \quad \forall i$

- This is a constrained optimization problem. Solving it requires **some new tools**
 - This is a **QP** problem (Quadratic cost function, linear constraints)
- The analytical solution of this equation provide great reduction in complexity (Vapnik works)

Unconstrained Minimization



- **Assume:**
 - Let $f : \Omega \rightarrow \mathbb{R}$ be a continuously differentiable function.
- Necessary and sufficient conditions for a local minimum: \mathbf{x}^* is a local minimum of $f(\mathbf{x})$ if and only if
 1. f has zero **gradient** at \mathbf{x}^* :

$$\nabla_{\mathbf{x}} f(\mathbf{x}^*) = \mathbf{0}$$

2. and the Hessian of f at \mathbf{x}^* is **positive semi-definite**

$$\mathbf{v}^t (\nabla^2 f(\mathbf{x}^*)) \mathbf{v} \geq \mathbf{0}, \quad \forall \mathbf{v} \in \mathbb{R}^n$$

where

$$\nabla^2 f(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n^2} \end{pmatrix}$$

Recall: Constrained Optimization: Equality Constraints



- The constrained **optimization** problem is

$$\min_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x}) \text{ subject to } h_i(\mathbf{x}) = 0 \text{ for } i = 1, \dots, l$$

- Construct the **Lagrangian** (introduce a multiplier for each constraint)

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\mu}) = f(\mathbf{x}) + \sum_{i=1}^l \mu_i h_i(\mathbf{x}) = f(\mathbf{x}) + \boldsymbol{\mu}^t \mathbf{h}(\mathbf{x})$$

- Then \mathbf{x}^* a local **minimum** \Leftrightarrow there exists a unique $\boldsymbol{\mu}^*$ s.t.

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\mu}^*) = \mathbf{0}$$

$$\nabla_{\boldsymbol{\mu}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\mu}^*) = \mathbf{0}$$

$$\mathbf{y}^t (\nabla_{\mathbf{xx}}^2 \mathcal{L}(\mathbf{x}^*, \boldsymbol{\mu}^*)) \mathbf{y} \geq 0 \quad \forall \mathbf{y} \text{ s.t. } \nabla_{\mathbf{x}} h(\mathbf{x}^*)^t \mathbf{y} = 0$$

Positive definite **Hessian** tells us we have a local minimum

Constrained Optimization

Inequality Constraints (Karush–Kuhn–Tucker (KKT) conditions)



- Given the **optimization** problem

$$\min_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x}) \text{ subject to } g_j(\mathbf{x}) \leq 0 \text{ for } j = 1, \dots, m$$

- Define the **Lagrangian** as

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{j=1}^m \lambda_j g_j(\mathbf{x}) = f(\mathbf{x}) + \boldsymbol{\lambda}^t \mathbf{g}(\mathbf{x})$$

- Then \mathbf{x}^* a local **minimum** \Leftrightarrow there exists a unique $\boldsymbol{\lambda}^*$ s.t.

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}$$

$$\lambda_j^* \geq 0 \text{ for } j = 1, \dots, m$$

$$\lambda_j^* g_j(\mathbf{x}^*) = 0 \text{ for } j = 1, \dots, m \quad \textbf{Complementary slackness}$$

$$g_j(\mathbf{x}^*) \leq 0 \text{ for } j = 1, \dots, m$$

Plus positive definite constraints on $\nabla_{\mathbf{xx}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*)$.

Back to the Original Problem



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

$$\text{subject to } 1 - y_i(\mathbf{w}^T \mathbf{x}_i + b) \leq 0 \quad \text{for } i = 1, \dots, n$$

- The **Lagrangian** is

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + \sum_{i=1}^n \alpha_i (1 - y_i(\mathbf{w}^T \mathbf{x}_i + b))$$

- Note that $\|\mathbf{w}\|^2 = \mathbf{w}^T \mathbf{w}$
- Setting the gradient of \mathcal{L} w.r.t. \mathbf{w} and b to zero, we have

$$\nabla_{\mathbf{w}} \mathcal{L}(w, b, \alpha) = \mathbf{w} + \sum_{i=1}^n \alpha_i (-y_i) \mathbf{x}_i = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial}{\partial b} \mathcal{L}(w, b, \alpha) = \sum_{i=1}^n \alpha_i y_i = 0$$

The Dual Problem



- If we substitute $\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$ to \mathcal{L} , we have

$$\begin{aligned}\mathcal{L} &= \frac{1}{2} \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i^T \sum_{j=1}^n \alpha_j y_j \mathbf{x}_j + \sum_{i=1}^n \alpha_i \left(1 - y_i \left(\sum_{j=1}^n \alpha_j y_j \mathbf{x}_j^T \mathbf{x}_i + b \right) \right) \\ &= \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \alpha_i - \sum_{i=1}^n \alpha_i y_i \sum_{j=1}^n \alpha_j y_j \mathbf{x}_j^T \mathbf{x}_i - b \sum_{i=1}^n \alpha_i y_i \\ &= -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^n \alpha_i\end{aligned}$$

- Note that $\sum_{i=1}^n \alpha_i y_i = 0$
- This is a function of α_i **only**

The Dual Problem



Dot product

$$\max. W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \underbrace{\alpha_i \alpha_j y_i y_j}_{\text{Scalar}} \underbrace{\mathbf{x}_i^T \mathbf{x}_j}_{\text{Dot product}}$$

- Subject to:

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

- **Matrix notation:**

$$\hat{\alpha} = \arg \max_{\alpha \in \mathbb{R}^n} \alpha^T \mathbf{1}_n - \frac{1}{2} \alpha^T \mathbf{Y} \mathbf{G} \mathbf{Y} \alpha$$

$$\mathbf{Y} \doteq \text{diag}(y_1, \dots, y_n), \quad y_i \in \{-1, 1\}^n$$

$$\mathbf{G} \in \mathbb{R}^{n \times n} \doteq \{G_{ij}\}_{i,j}^{n,n}, \quad \text{where } G_{ij} \doteq \langle \mathbf{x}_i, \mathbf{x}_j \rangle \text{ Gram matrix}$$

- This is a **quadratic programming** (QP) problem and a global maximum of α_i can always be found

Why is it called Support Vector Machine?

$$\mathcal{L} = \frac{1}{2} \mathbf{w}^T \mathbf{w} + \sum_{i=1}^n \alpha_i \left(1 - y_i (\mathbf{w}^T \mathbf{x}_i + b) \right)$$

- Based on **complementary slackness** in KKT conditions we have

$$\lambda_j^* g(\mathbf{x}^*) = 0 \text{ for } j = 1, \dots, m$$

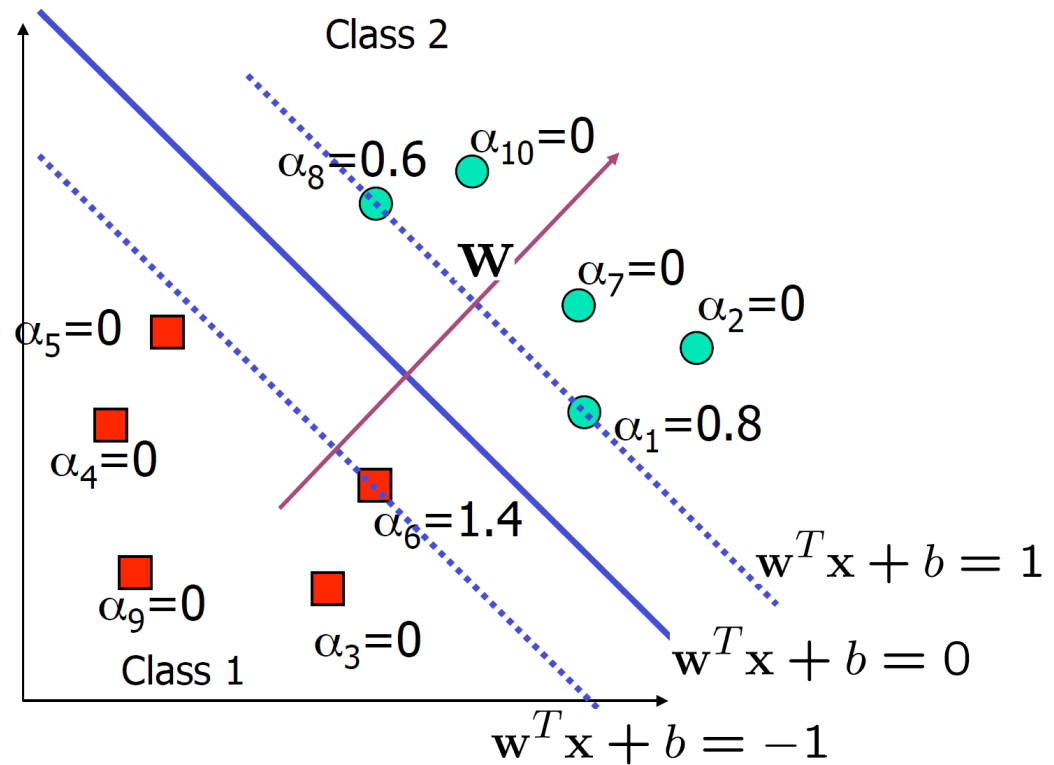
$$\alpha_i \left(1 - y_i (\mathbf{w}^T \mathbf{x}_i + b) \right) = 0$$

- Sinc $\alpha_i \geq 0$, either $\alpha_i = 0$ or $(1 - y_i (\mathbf{w}^T \mathbf{x}_i + b)) = 0$
- Only if \mathbf{x}_i is on the **margin line** (**support vectors**) the $(1 - y_i (\mathbf{w}^T \mathbf{x}_i + b))$ will be zero
- So, many of the α_i are **zero and** \mathbf{x}_i with non-zero α_i are called support vectors (SV)

A Geometrical Interpretation



- The decision boundary is **determined only** by the **SV**
- Let t_j ($j=1, \dots, s$) be the indices of the s support vectors.



$$\mathbf{w} = \sum_{j=1}^s \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$$

Finding w and b



- The optimal value of w states in terms of the optimal **value** of α_i , \mathbf{w} can be recovered by

$$\mathbf{w} = \sum_{j=1}^s \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j} \Rightarrow \mathbf{w} = \sum_{i=1}^s \alpha_i y_i \mathbf{x}_i$$

- The value of b can be computed as the solution of

$$\alpha_i (y_i ((\mathbf{w} \cdot \mathbf{x}_i) + b) - 1) = 0$$

- using any of the **support vectors** but it is numerically **safer** to take the **average value** of b resulting from all such equations.



- For SVM, **sequential minimal optimization** (SMO) seems to be the most popular:
 - A QP with **two variables** is trivial to solve
 - Each iteration of SMO picks a **pair of** (α_i, α_j) and solve the QP with these two variables; Repeat until convergence

$$\max. W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

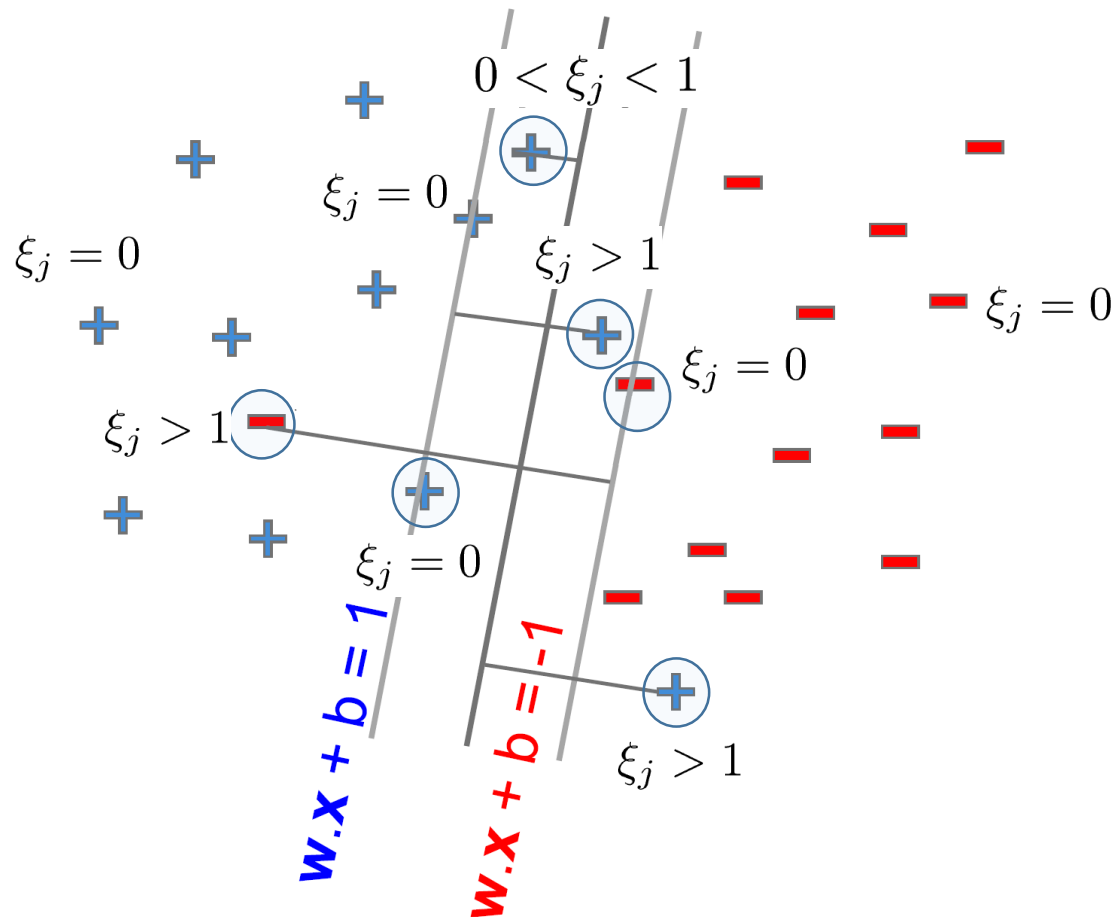
- Both the quadratic programming problem and the final decision function depend only **on the dot products between patterns**
- For **testing** with a new data \mathbf{z}

$$f(\mathbf{x}) = \text{sign}\left(\sum_{j=1}^s \alpha_{t_j} y_{t_j} (\mathbf{x}_{t_j}^T \mathbf{z}) + b\right)$$

Non-linearly Separable Problems (Soft SVM)



- We allow “**error**” ξ_i in classification; it is **based on the output** of the discriminant function $\mathbf{w}^T \mathbf{x} + b$



If we minimize $\sum_i \xi_i$, ξ_i can be computed by

$$\begin{cases} \mathbf{w}^T \mathbf{x}_i + b \geq 1 - \xi_i & y_i = 1 \\ \mathbf{w}^T \mathbf{x}_i + b \leq -1 + \xi_i & y_i = -1 \\ \xi_i \geq 0 & \forall i \end{cases}$$

- ξ_i are “**slack** variables” in optimization
- Note that $\xi_i = 0$ if there is no error for \mathbf{x}_i
- ξ_i is an **upper bound** of the number of errors

- ξ_i named **slack variable**

Soft Margin Hyperplane



- We want to **minimize**

$$\frac{1}{2} ||\mathbf{w}'||^2 + C \sum_{i=1}^n \xi_i$$

C : **tradeoff parameter** between **error** and **margin**

- The optimization problem becomes

$$\text{Minimize } \frac{1}{2} ||\mathbf{w}'||^2 + C \sum_{i=1}^n \xi_i$$

$$\text{subject to } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

- We can form the **Lagrangian** (α_i 's and r_i 's are our Lagrange multipliers):

$$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2} w^T w + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i [y_i (x^T w + b) - 1 + \xi_i] - \sum_{i=1}^n r_i \xi_i.$$

The Dual Soft SVM



$$\max. W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } C \geq \alpha_i \geq 0, \sum_{i=1}^n \alpha_i y_i = 0$$

- \mathbf{w} is recovered as:

$$\mathbf{w} = \sum_{j=1}^s \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$$

- This is very **similar** to the optimization problem in the linear separable case, except that there is an **upper bound C on α_i now**
- Once again, a QP solver can be used to find α_i

sequential minimal optimization (SMO)

coordinate ascent



- We've already seen two optimization algorithms, **gradient ascent** and **Newton's method**.

- The new algorithm is called **coordinate ascent**:

Loop until convergence: {

For $i = 1, \dots, n$, {

$$\alpha_i := \arg \max_{\hat{\alpha}_i} W(\alpha_1, \dots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \dots, \alpha_n).$$

}

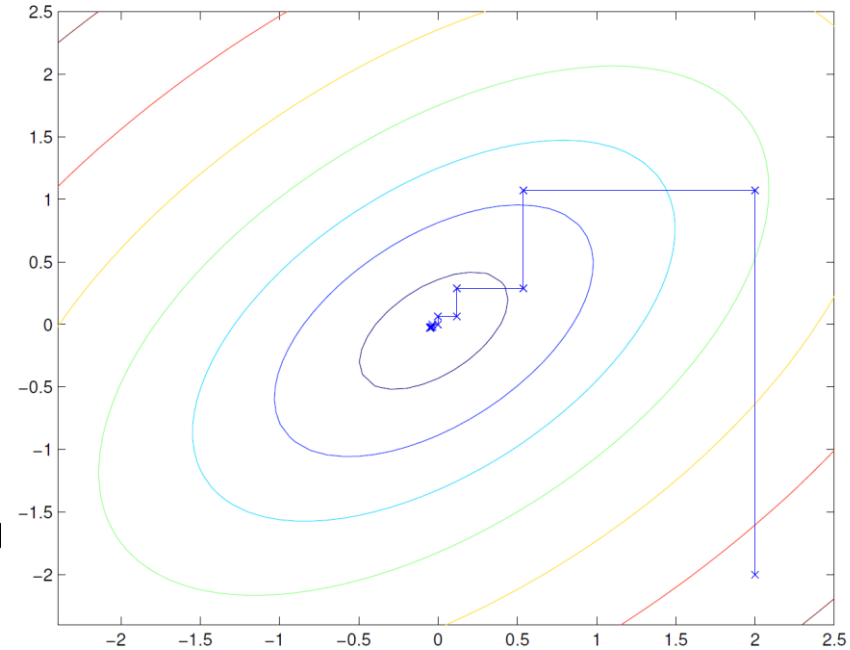
}

- We will **hold** all the **variables except** for some α_i fixed, and **reoptimize** W with respect to just the parameter α_i .
- The inner-loop **reoptimizes** the variables in order $\alpha_1, \alpha_2, \dots, \alpha_n, \alpha_1, \alpha_2, \dots$
- More **sophisticated** version might choose other orderings; (choose the next variable to update according to which one we expect to allow us to make the **largest increase** in $W(\alpha)$.)

Coordinate ascent



- The **ellipses** in the figure are the contours of a **quadratic function**
- Coordinate ascent was initialized at $(2, -2)$,
- Notice that on each step, coordinate ascent takes a step that's **parallel to one** of the axes, since **only one variable** is being optimized at a time.



SMO algorithm for SVM



- Dual optimization problem that we want to solve:

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle. \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \dots, n \\ & \sum_{i=1}^n \alpha_i y_i = 0. \end{aligned}$$

- If we want to update some subject of the α_i 's, we must update **at least two of them** simultaneously
 - If **we fixed** α_1 , we **can't make** any change to α_1

$$\sum_{i=1}^n \alpha_i y_i = 0. \Rightarrow \alpha_1 y_1 = - \sum_{i=2}^n \alpha_i y_i \Rightarrow \alpha_1 = -y_1 \sum_{i=2}^n \alpha_i y_i$$

SMO algorithm for SVM



- Repeat till convergence
 1. Select some pair α_i and α_j to update next
 2. Reoptimize $W(\alpha)$ with respect to α_i and α_j , while holding all the other α_k 's ($k \neq i, j$) fixed.

- We've decided to hold $\alpha_3, \dots, \alpha_n$ **fixed**:

$$\alpha_1 y_1 + \alpha_2 y_2 = - \sum_{i=3}^n \alpha_i y_i$$

- right hand side is fixed, we can just let it be denoted by some **constant** ζ :

$$\alpha_1 y_1 + \alpha_2 y_2 = \zeta.$$

$$\alpha_1 = (\zeta - \alpha_2 y_2) y_1$$

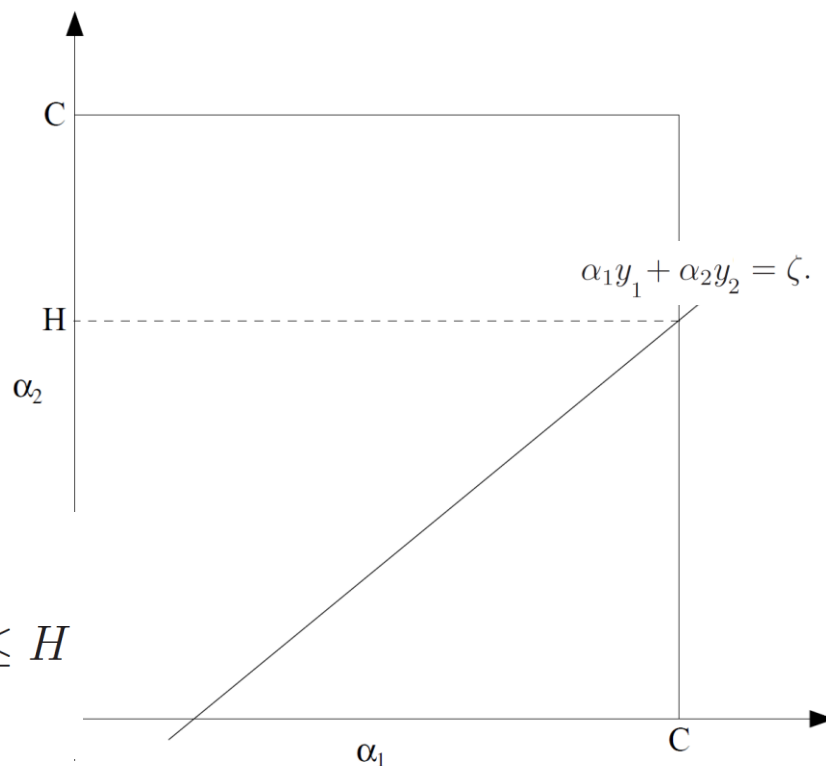
$$W(\alpha_1, \alpha_2, \dots, \alpha_n) = W((\zeta - \alpha_2 y_2) y_1, \alpha_2, \dots, \alpha_n).$$

This can also be **expressed** in the form

$$a\alpha_2^2 + b\alpha_2 + c$$

We can **find** the resulting value optimal simply by taking α_2^{new} , and “**clipping**” it to lie in the $[L, H]$ interval

$$\alpha_2^{\text{new}} = \begin{cases} H & \text{if } \alpha_2^{\text{new,unclipped}} > H \\ \alpha_2^{\text{new,unclipped}} & \text{if } L \leq \alpha_2^{\text{new,unclipped}} \leq H \\ L & \text{if } \alpha_2^{\text{new,unclipped}} < L \end{cases}$$



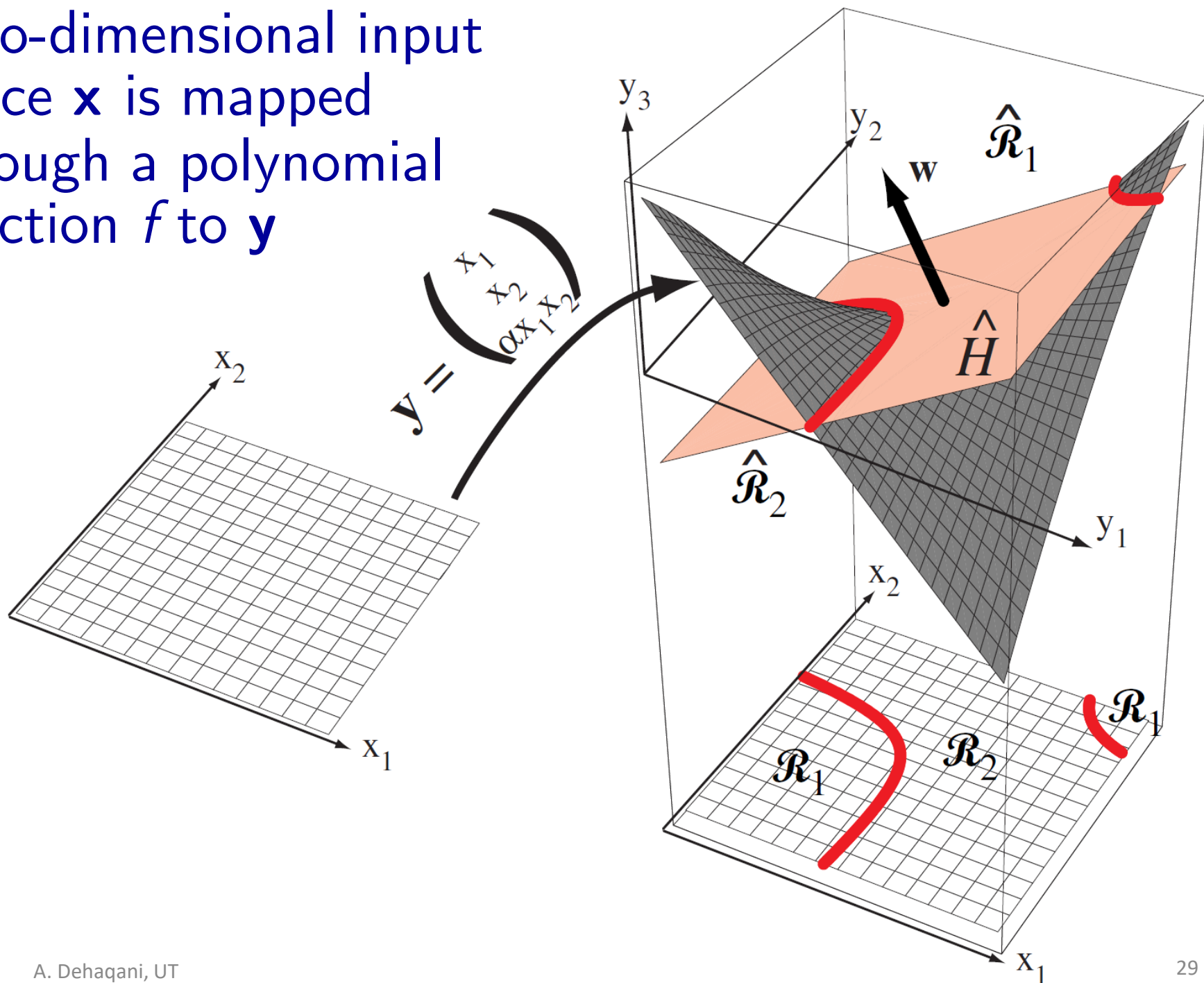
We can compute α_1^{new} by first equation

Extension to Non-linear Decision Boundary

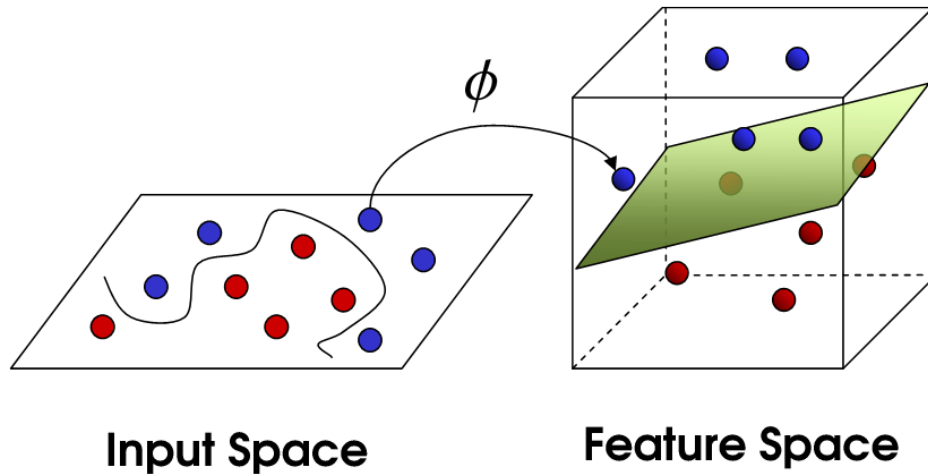


- So far, we have only considered large-margin classifier with a **linear decision boundary (correspond with equal variance Gaussian)**
- How to **generalize** it to become **nonlinear**?
- Key idea: transform \mathbf{x}_i to a **higher dimensional** space to “**make life easier**”
 - **Input** space: the space the point \mathbf{x}_i are located
 - **Feature** space: the space of $\varphi(\mathbf{x}_i)$ after transformation
- **Linear operation** in the feature space is **equivalent** to **nonlinear** operation in input space

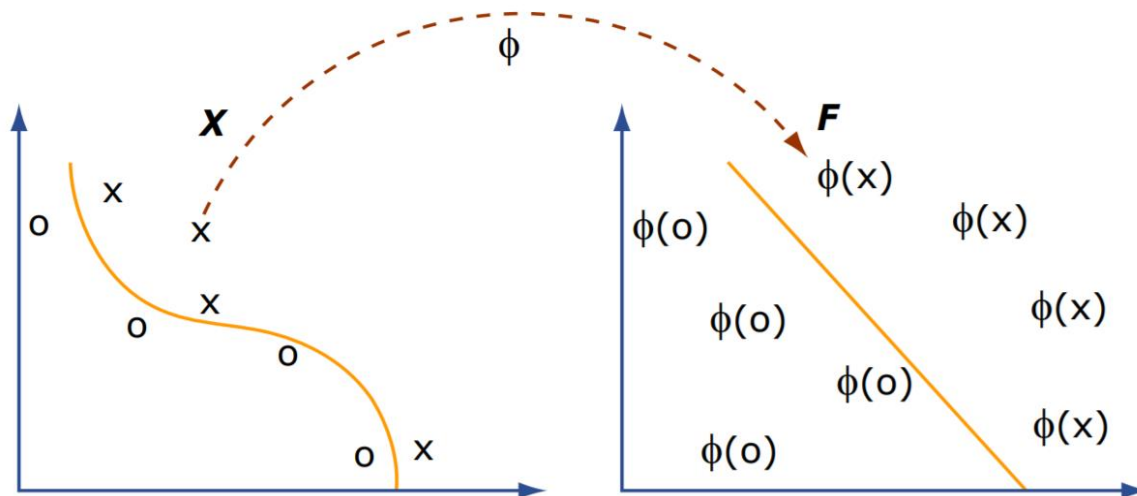
Two-dimensional input space \mathbf{x} is mapped through a polynomial function f to \mathbf{y}



Transforming the Data



- Feature space is of **higher dimension** than the **input space** in practice
- Computation in the feature space can be **costly** because it is high dimensional



- The **kernel trick** comes to rescue

The Kernel Trick



- Recall the SVM **optimization problem**

$$\max. W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

subject to $C \geq \alpha_i \geq 0, \sum_{i=1}^n \alpha_i y_i = 0$

- The data points only appear as **inner product**
- As long as we can calculate the **inner product in the feature space**, we **do not** need the **mapping explicitly**
- Many common **geometric operations** (angles, distances) can be expressed by **inner products**
- Define the **kernel function** K by $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$

$$\max. W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

Testing the new data



- For testing, the new data \mathbf{z} is classified as class 1 if $f \geq 0$, and as class 2 if $f < 0$

Original

$$\mathbf{w} = \sum_{j=1}^s \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$$
$$f = \mathbf{w}^T \mathbf{z} + b = \sum_{j=1}^s \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}^T \mathbf{z} + b$$

- With kernel function (change all **inner products** to **kernel functions**)

$$\mathbf{w} = \sum_{j=1}^s \alpha_{t_j} y_{t_j} \phi(\mathbf{x}_{t_j})$$
$$f = \langle \mathbf{w}, \phi(\mathbf{z}) \rangle + b = \sum_{j=1}^s \alpha_{t_j} y_{t_j} K(\mathbf{x}_{t_j}, \mathbf{z}) + b$$

kernel trick



- Suppose $\phi(\cdot)$ is given as follows

$$\phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

- An **inner product** in the feature space is

$$\langle \phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right), \phi\left(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}\right) \rangle = (1 + x_1y_1 + x_2y_2)^2$$

- So, if we define the kernel function as follows, there is no need to **carry out $\phi(\cdot)$ explicitly**

$$K(\mathbf{x}, \mathbf{y}) = (1 + x_1y_1 + x_2y_2)^2$$

- This use of kernel function to **avoid carrying out $\phi(\cdot)$ explicitly** is known as the kernel trick

Kernel Functions



- In practical use of SVM, the user **specifies** the kernel function; the **transformation** $\varphi(\cdot)$ is **not explicitly stated**
 - Given a kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$, the **transformation** $\varphi(\cdot)$ is given by its **eigenfunctions** (a concept in functional analysis)
 - Eigenfunctions can be difficult to construct explicitly
- This is why people **only** specify the **kernel** function **without** worrying about the **exact transformation**
- Another view: kernel function, being an inner product, is really a **similarity measure** between the objects

Examples of Kernel Functions



- **Polynomial** kernel with **degree** d

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + 1)^d$$

- **Gaussian kernel** with width σ

$$K(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2 / (2\sigma^2))$$

- Closely related to **radial basis function neural networks**
- The feature space is **infinite-dimensional** (it still be written as a dot product in a new feature space $k(\mathbf{x}, \mathbf{x}_0) = \Phi(\mathbf{x}) * \Phi(\mathbf{x}_0)$, only with an **infinite number of dimensions**)

- **Sigmoid** with parameter κ and θ

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa \mathbf{x}^T \mathbf{y} + \theta)$$

- It does not satisfy the **Mercer condition** on all κ and θ

More on Kernel Functions



- Since the training of SVM **only** requires the value of $K(\mathbf{x}_i, \mathbf{x}_j)$, there is **no restriction** of the **form** of \mathbf{x}_i and \mathbf{x}_j
 - \mathbf{x}_i can be a **sequence or a tree**, instead of a feature vector
- $K(\mathbf{x}_i, \mathbf{x}_j)$ is just a **similarity measure** comparing \mathbf{x}_i and \mathbf{x}_j
- For a test object \mathbf{z} , the **discriminant function** essentially is a weighted sum of the **similarity between \mathbf{z} and \mathbf{a}** preselected set of objects (the support vectors)

$$f(\mathbf{z}) = \sum_{\mathbf{x}_i \in \mathcal{S}} \alpha_i y_i K(\mathbf{z}, \mathbf{x}_i) + b$$

\mathcal{S} : the set of support vectors

Necessary conditions for valid kernels.



- **Kernel matrix:**
 - A square, n -by- n matrix K be defined so that its (i,j) -entry is given by $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
- K must be **symmetric**.
- Theorem (**Mercer**). Let $K : \mathcal{R}^d \times \mathcal{R}^d \rightarrow \mathcal{R}$ be given. Then for K to be a valid (Mercer) kernel, it is **necessary** and **sufficient** that for any $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, ($n < \infty$), the corresponding **kernel matrix** is **symmetric** positive semi-definite
- Theorem (Mercer). If we have a kernel $K(\mathbf{x}, \mathbf{z})$ that is positive, we can expand K in terms of **eigenfunctions** and **eigenvalues**
 - K is **positive**; which here means

$$\forall f \in L_2(\mathcal{X}), \quad \int_{\mathcal{X} \times \mathcal{X}} k(\mathbf{x}, \mathbf{z}) f(\mathbf{x}) f(\mathbf{z}) d\mathbf{x} d\mathbf{z} \geq 0,$$

- we can expand $K(\mathbf{x}, \mathbf{z})$

$$k(\mathbf{x}, \mathbf{z}) = \sum_{j=1}^{\infty} \lambda_j \psi_j(\mathbf{x}) \psi_j(\mathbf{z}).$$

- $\psi_j \in L_2(\mathcal{X})$ **eigenfunctions** and $\lambda_j > 0$ **eigenvalues**

K in the finite states (finding feature map)



- **Eigendecomposition** takes this form

$$\mathbf{K} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^T$$

$$\mathbf{K} = \begin{matrix} & \begin{matrix} 1 & j & m \end{matrix} \\ \begin{matrix} m \\ i \\ 1 \end{matrix} & \begin{bmatrix} & & \\ & k(x_i, x_j) & \\ & & \end{bmatrix} \end{matrix}$$

- Consider this **feature map**:

$$\Phi(x_i) = [\sqrt{\lambda_1} v_1^{(i)}, \dots, \sqrt{\lambda_t} v_t^{(i)}, \dots, \sqrt{\lambda_m} v_m^{(i)}].$$

- writing it for \mathbf{x}_j too

$$\Phi(x_j) = [\sqrt{\lambda_1} v_1^{(j)}, \dots, \sqrt{\lambda_t} v_t^{(j)}, \dots, \sqrt{\lambda_m} v_m^{(j)}].$$

- With this choice, k is just a **dot product** in \mathbf{R}^m

$$\langle \Phi(x_i), \Phi(x_j) \rangle_{\mathbf{R}^m} = \sum_{t=1}^m \lambda_t v_t^{(i)} v_t^{(j)} = (\mathbf{V} \mathbf{\Lambda} \mathbf{V}^T)_{ij} = K_{ij} = k(x_i, x_j).$$

More on Kernel Functions

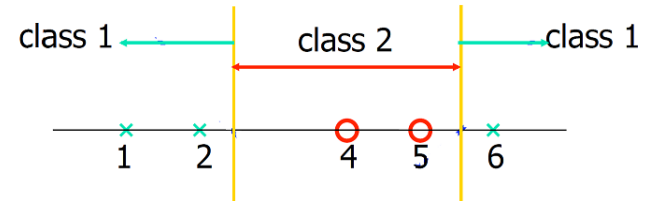


- **Not** all similarity measure can be used as kernel function, however
 - The kernel function needs to satisfy the **Mercer** function,
 - i.e., the function is “**positive-definite**”
 - This implies that the n by n **kernel matrix**, in which the (i,j) -th entry is the $K(\mathbf{x}_i, \mathbf{x}_j)$, is always **positive definite**
- This also means that the **QP is convex** and can be solved in **polynomial** time
- The learning algorithm that you can write in terms of **only inner products** between **input feature vectors**, then by replacing this with $K(\mathbf{x}, \mathbf{y})$ where K is a kernel, you can “magically” work efficiently in the **high dimensional feature space** corresponding to K .
- **linear regression** and SVMs are two well known examples

Example



- Suppose we have 5 1D data points
 - $x_1=1, x_2=2, x_3=4, x_4=5, x_5=6$, with 1, 2, 6 as class 1 and 4, 5 as class 2
 - $\Rightarrow y_1=1, y_2=1, y_3=-1, y_4=-1, y_5=1$



- We use the polynomial **kernel** of degree 2
 - $K(x,y) = (xy+1)^2$
 - C is set to 100
- We first **find** α_i ($i=1, \dots, 5$) by

$$\max. \quad \sum_{i=1}^5 \alpha_i - \frac{1}{2} \sum_{i=1}^5 \sum_{j=1}^5 \alpha_i \alpha_j y_i y_j (x_i x_j + 1)^2$$

$$\text{subject to } 100 \geq \alpha_i \geq 0, \sum_{i=1}^5 \alpha_i y_i = 0$$

Example



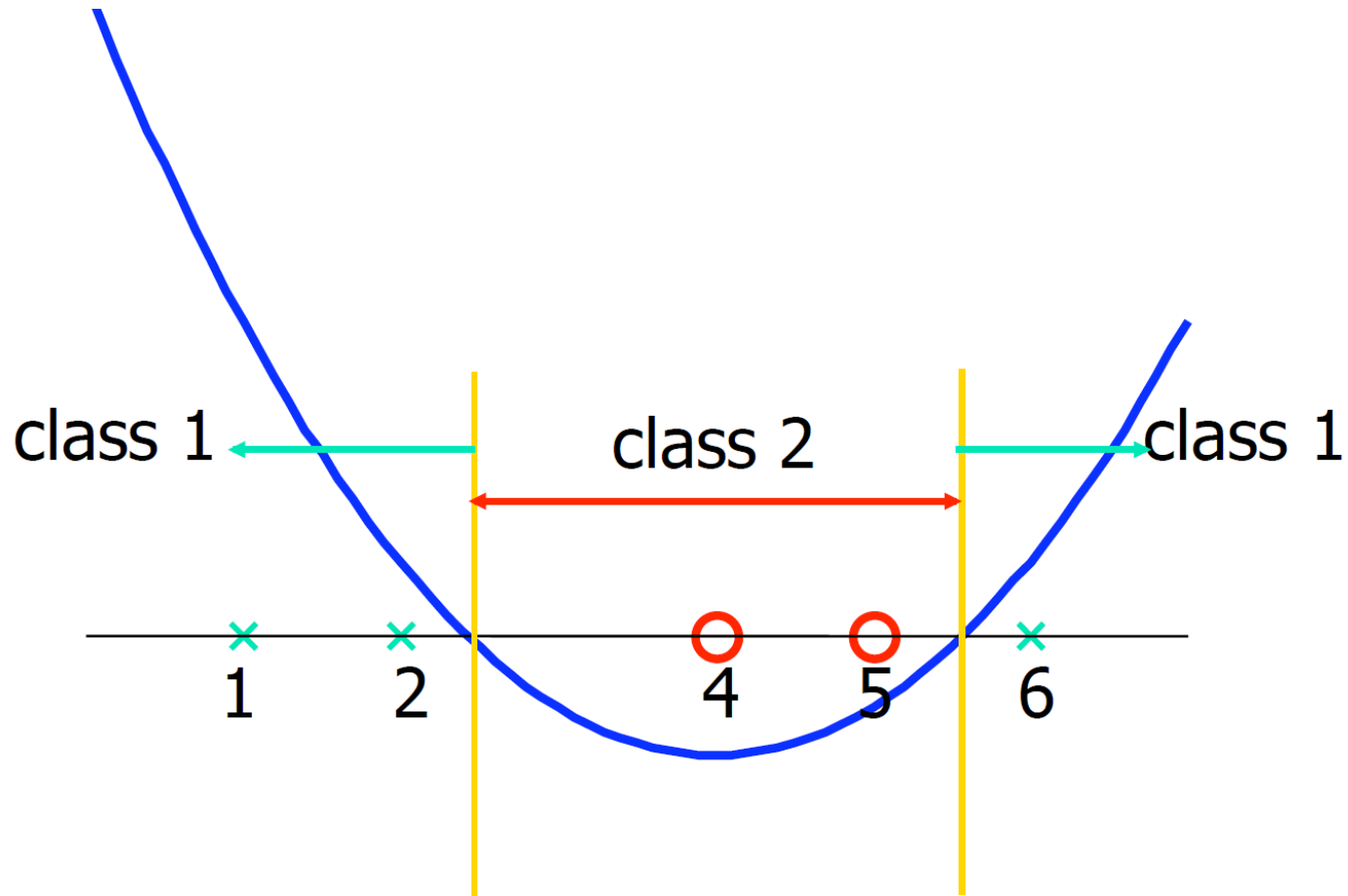
- By using a QP solver, we get
 - $\alpha_1=0, \alpha_2=2.5, \alpha_3=0, \alpha_4=7.333, \alpha_5=4.833$
 - Note that the **constraints** are indeed **satisfied**
- The **support vectors** are $\{x_2=2, x_4=5, x_5=6\}$

- The **discriminant function** is
$$w = \sum_{j=1}^s \alpha_{t_j} y_{t_j} \phi(x_{t_j})$$
$$f = \langle w, \phi(z) \rangle + b = \sum_{j=1}^s \alpha_{t_j} y_{t_j} K(x_{t_j}, z) + b$$
$$f(z) = 2.5(1)(2z + 1)^2 + 7.333(-1)(5z + 1)^2 + \underbrace{4.833}_{\alpha_5} \underbrace{(1)}_{y_5} \underbrace{(6z + 1)^2}_{K(z, x_5)} + b$$
$$= 0.6667z^2 - 5.333z + b$$

- b is recovered by solving $f(2)=1$ or by $f(5)=-1$ or by $f(6)=1$,

$$f(z) = 0.6667z^2 - 5.333z + 9$$

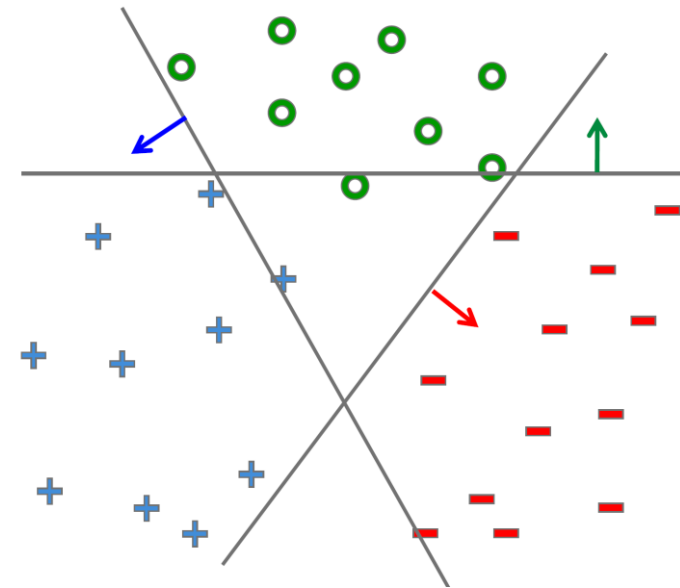
Value of discriminant function



What about multiple classes?



- One against all
 - Learn 3 classifiers separately:
 - Class k vs. rest
 - $(\mathbf{w}_k, b_k)_{k=1,2,3}$
 - $y = \arg \max_k \mathbf{w}_k x + b_k$
 - Disadvantages: ambiguous area
- In each step, **remove one class**:
 - Problem: sensitive to order
- One against one:
- Majority voting



Multi-class SVM



- Simultaneously learn **3 sets of weights**
- **Need new constraints** : (The “score” of the correct class **must be better** than the “score” of wrong classes)

$$\text{minimize}_{\mathbf{w}, b} \quad \sum_y \mathbf{w}^{(y)} \cdot \mathbf{w}^{(y)} \quad \text{Margin: gap between } \mathbf{correct \ class} \text{ and } \mathbf{nearest \ other \ class}$$

$$\mathbf{w}^{(y_j)} \cdot \mathbf{x}_j + b^{(y_j)} \geq \mathbf{w}^{(y')} \cdot \mathbf{x}_j + b^{(y')} + \textcircled{1}, \quad \forall y' \neq y_j, \quad \forall j$$

- Introducing **slack variables** and **maximize margin** (joint optimization):

$$\text{minimize}_{\mathbf{w}, b} \quad \sum_y \mathbf{w}^{(y)} \cdot \mathbf{w}^{(y)} + C \sum_j \sum_{y \neq y_j} \xi_j^{(y)}$$

$$\begin{aligned} \mathbf{w}^{(y_j)} \cdot \mathbf{x}_j + b^{(y_j)} &\geq \mathbf{w}^{(y)} \cdot \mathbf{x}_j + b^{(y)} + 1 - \xi_j^{(y)}, \quad \forall y \neq y_j, \quad \forall j \\ \xi_j^{(y)} &\geq 0 \end{aligned}$$

- **Finally** we select class:

$$y = \arg \max_k \mathbf{w}^{(k)} \cdot \mathbf{x} + b^{(k)}$$

- Usually the **binary classification** works **better**

Epsilon Support Vector Regression (ε -SVR)



- Given: a data set $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ with target values $\{u_1, \dots, u_n\}$, we want to do ε -SVR
- The **optimization** problem is

$$\text{MIN } \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n |\xi_i|$$

Subject to

$$|y_i - w_i x_i| \leq \varepsilon + |\xi_i|$$

Ordinary Least Squares (OLS)

$$\text{MIN } \sum_{i=1}^n (y_i - w_i x_i)^2$$

- The error term is instead **handled** in the constraints
- SVR gives us the **flexibility** to define how much error is acceptable in our model



- C is a parameter to **control** the amount of **influence of** the error
- The $\frac{1}{2}||\mathbf{w}'||^2$ term serves as **controlling** the **complexity** of the **regression** function
- This is similar to **ridge regression** (a technique for **regression** data that suffer from multicollinearity.)

$$\hat{\beta}^{ridge} = \underset{\beta \in \mathbb{R}}{\operatorname{argmin}} ||y - XB||_2^2 + \lambda ||B||_2^2$$

- After training (solving the QP), we get values of α_i and α_i^* , which are both zero if \mathbf{x}_i does not contribute to the **error function**
- For a new data \mathbf{z} ,

$$f(\mathbf{z}) = \sum_{j=1}^s (\alpha_{t_j} - \alpha_{t_j}^*) K(\mathbf{x}_{t_j}, \mathbf{z}) + b$$

ϵ -insensitive loss function

