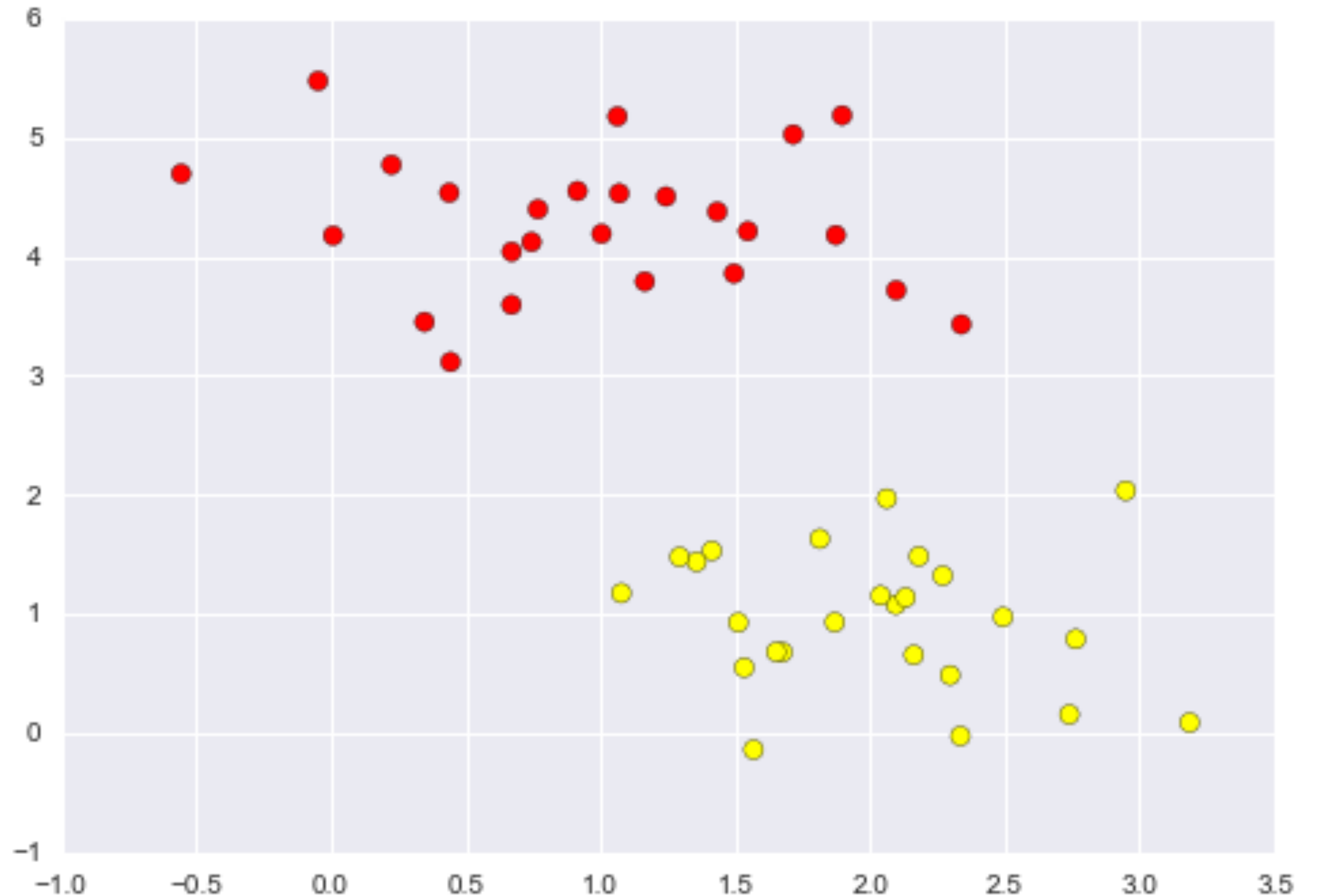


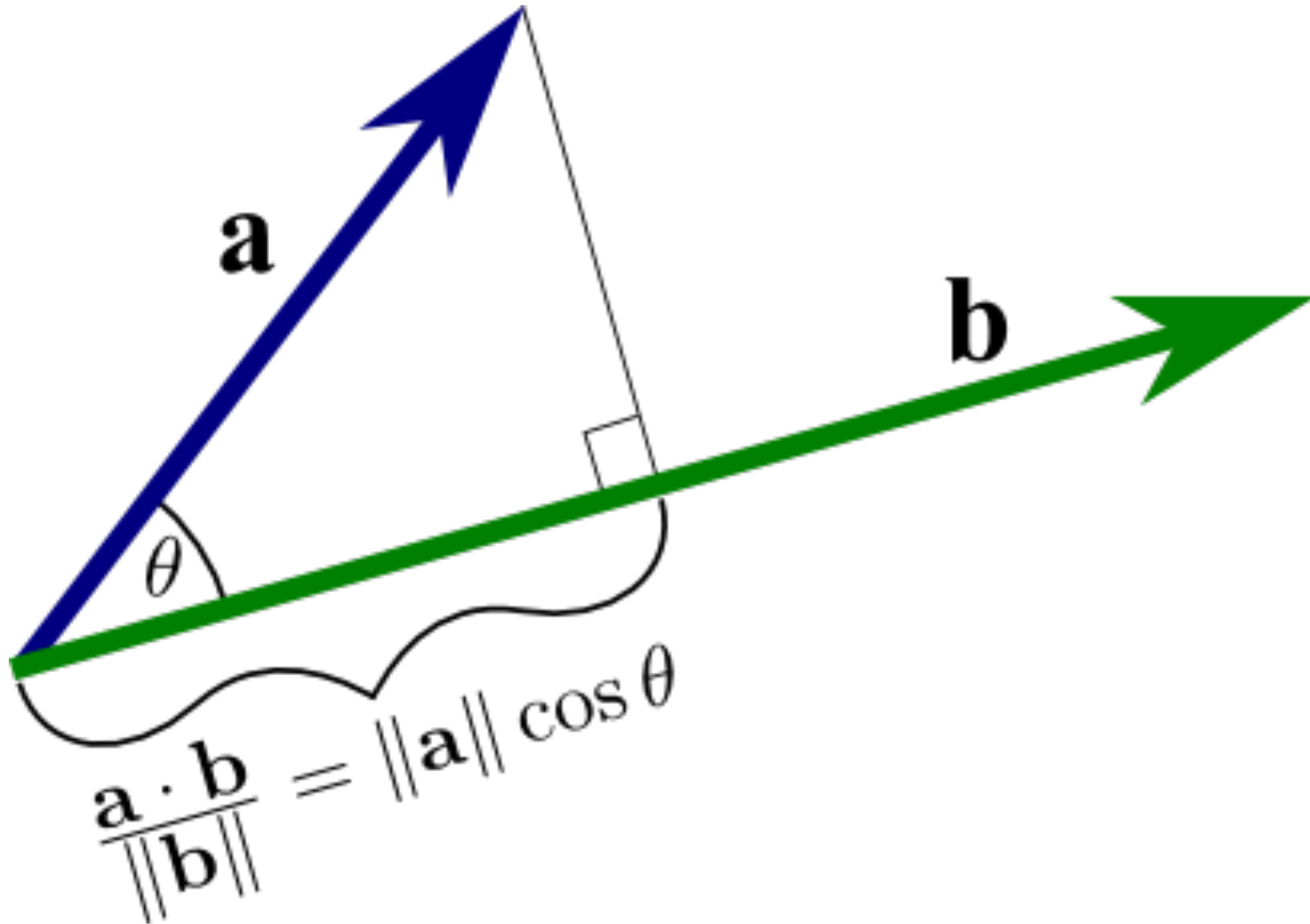
# Lecture 15 pre-video

**svm:**  
**Maximize**  
**Margins**

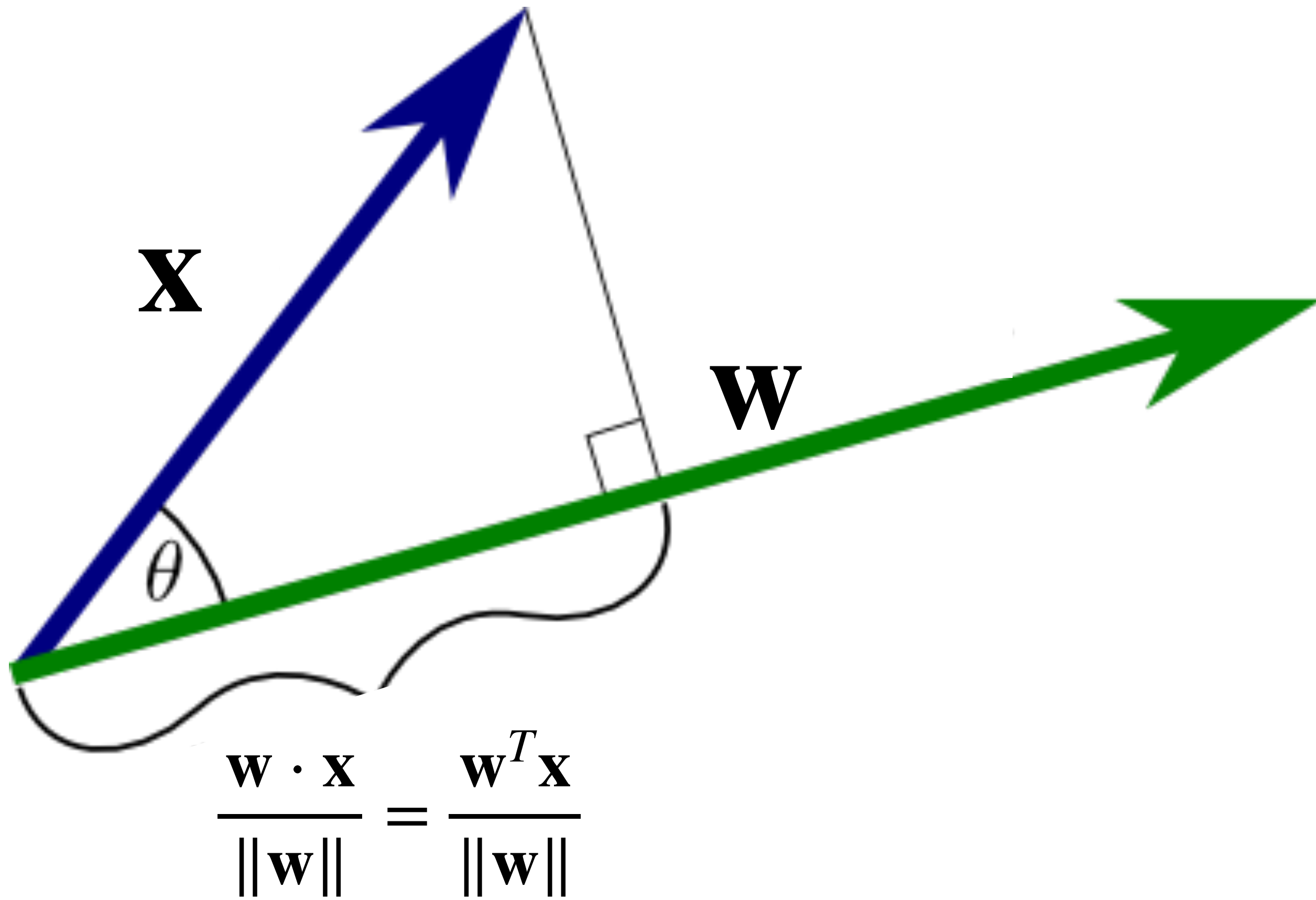


# Dot product - a scalar projection

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos \theta$$



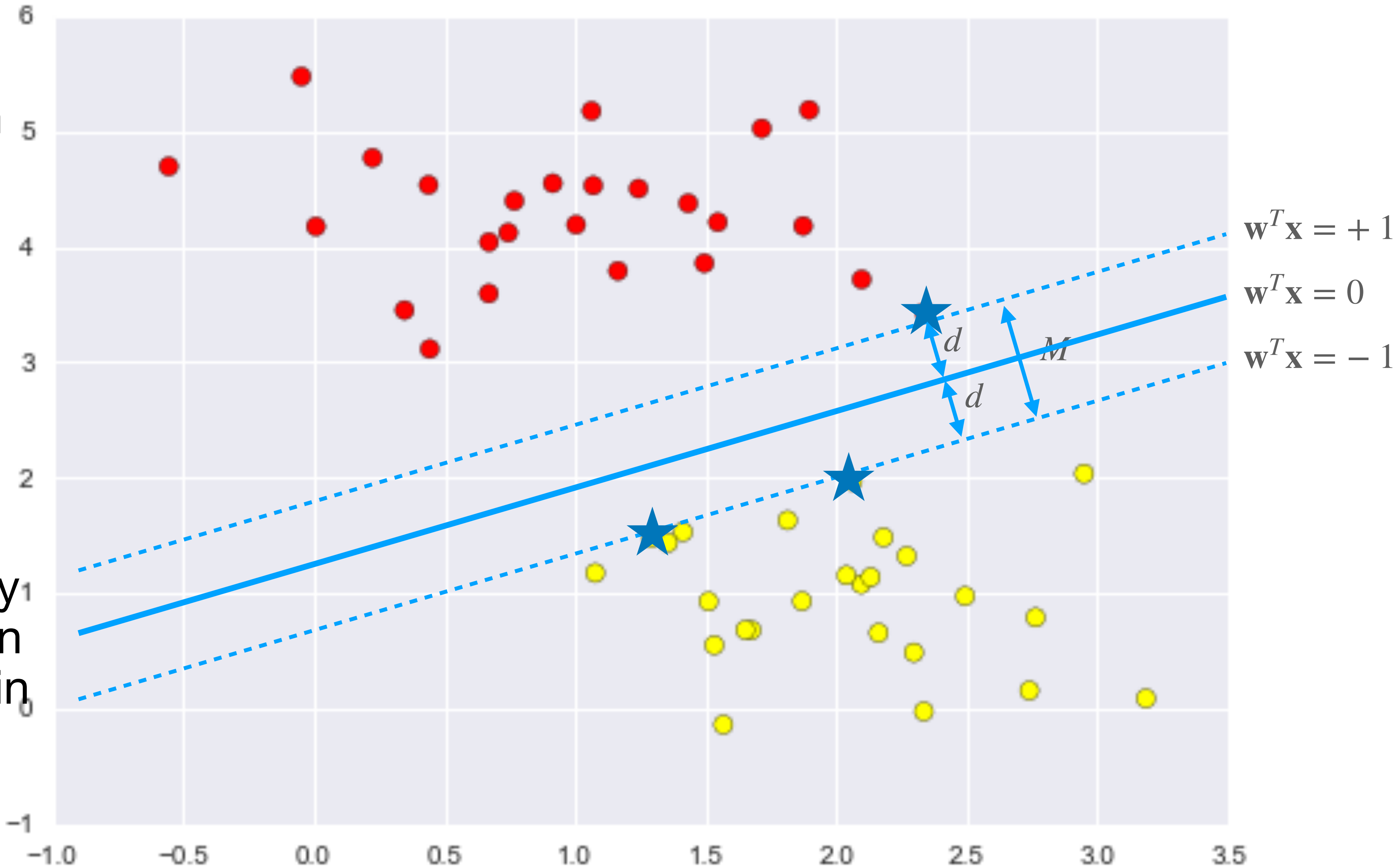
# Dot product - a scalar projection



From geometry:  
distance between  
a point and a  
hyperplane

$$d = \frac{|\mathbf{w}^T \mathbf{x}|}{\|\mathbf{w}\|}$$

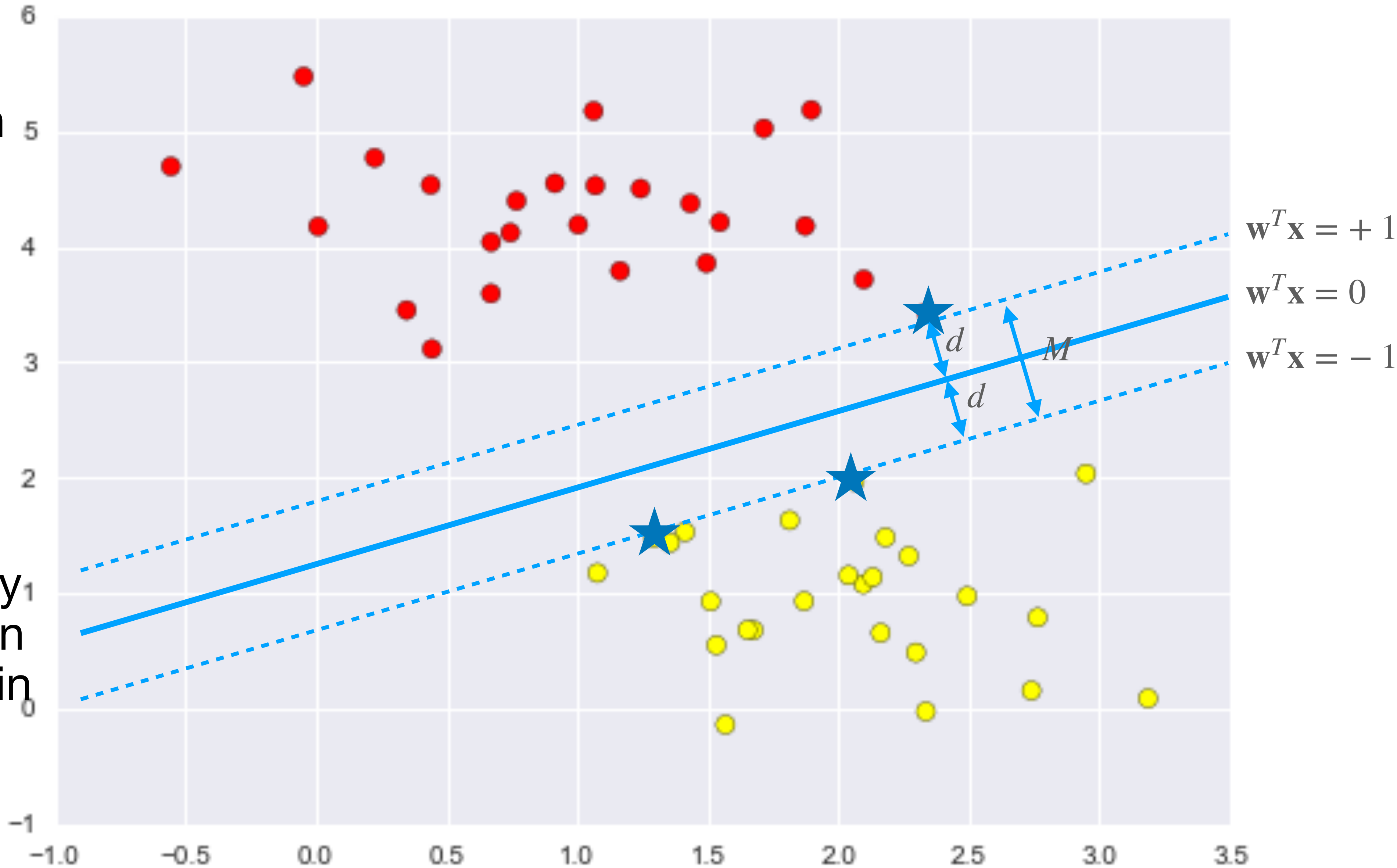
In our case, the  
hyperplane is the  
decision boundary  
and the point is on  
the positive margin



From geometry:  
distance between  
a point and a  
hyperplane

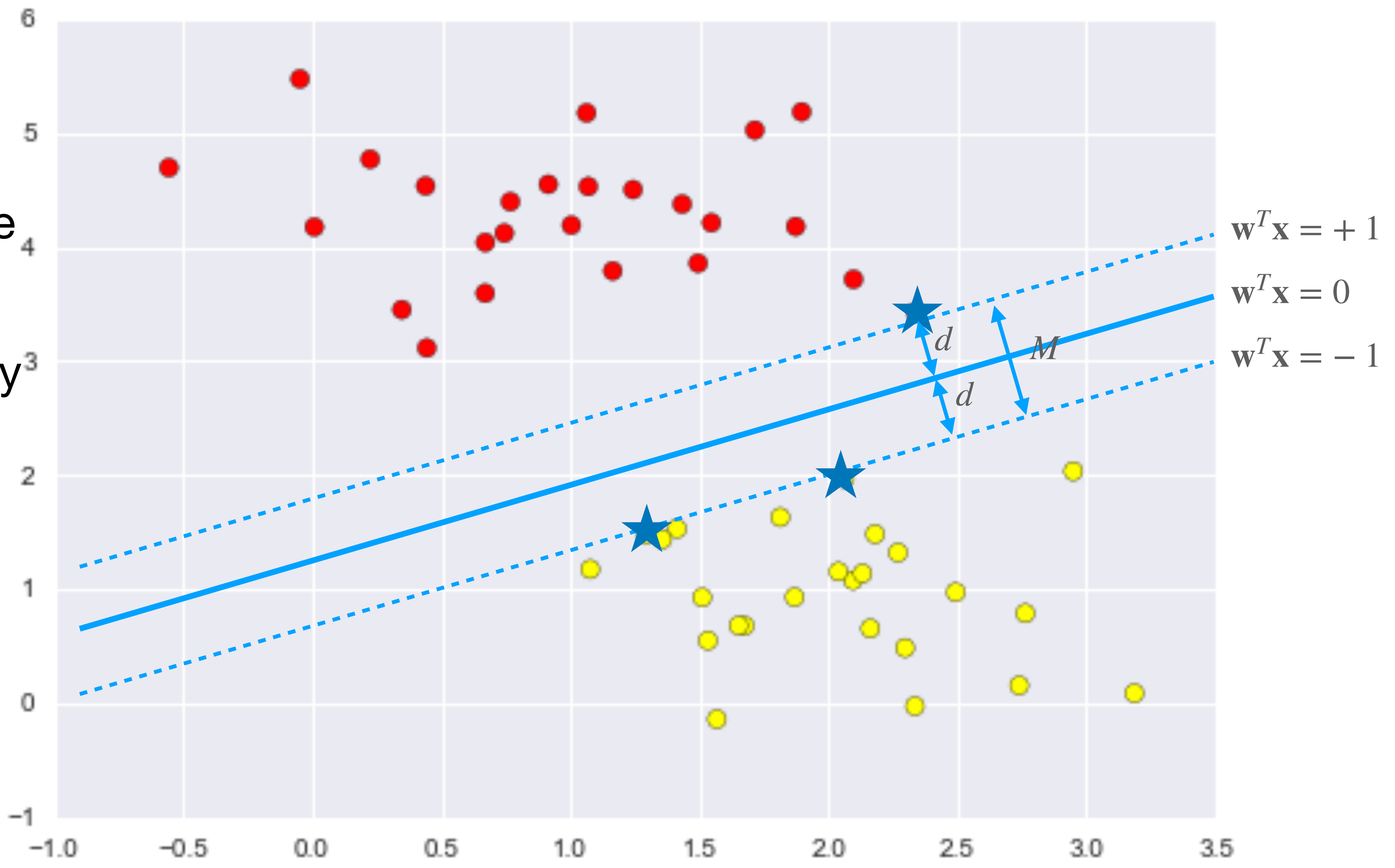
$$d = \frac{|1|}{\|\mathbf{w}\|}$$

In our case, the  
hyperplane is the  
decision boundary  
and the point is on  
the positive margin



So the overall margin must be twice the distance from the positive margin to the decision boundary

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

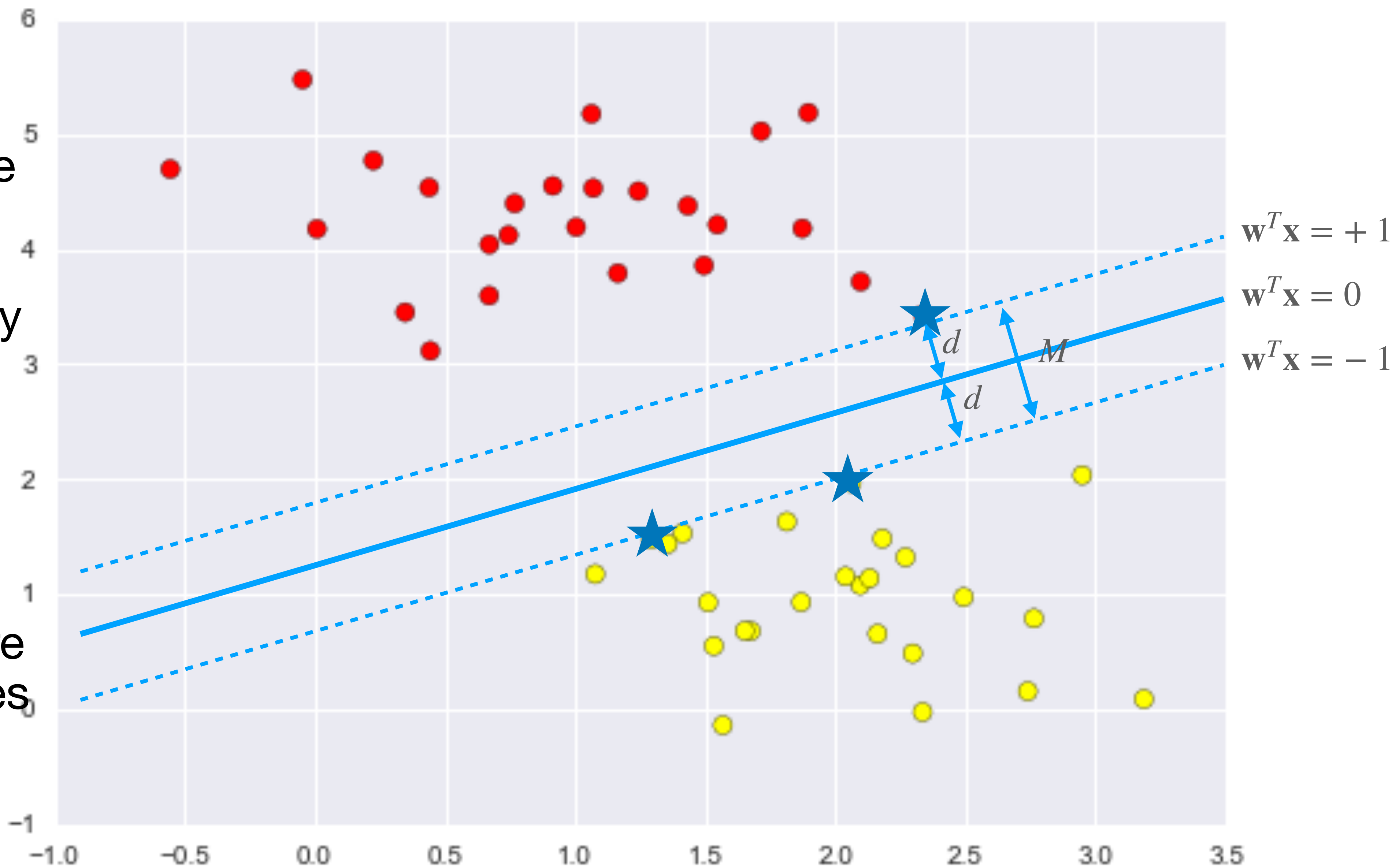




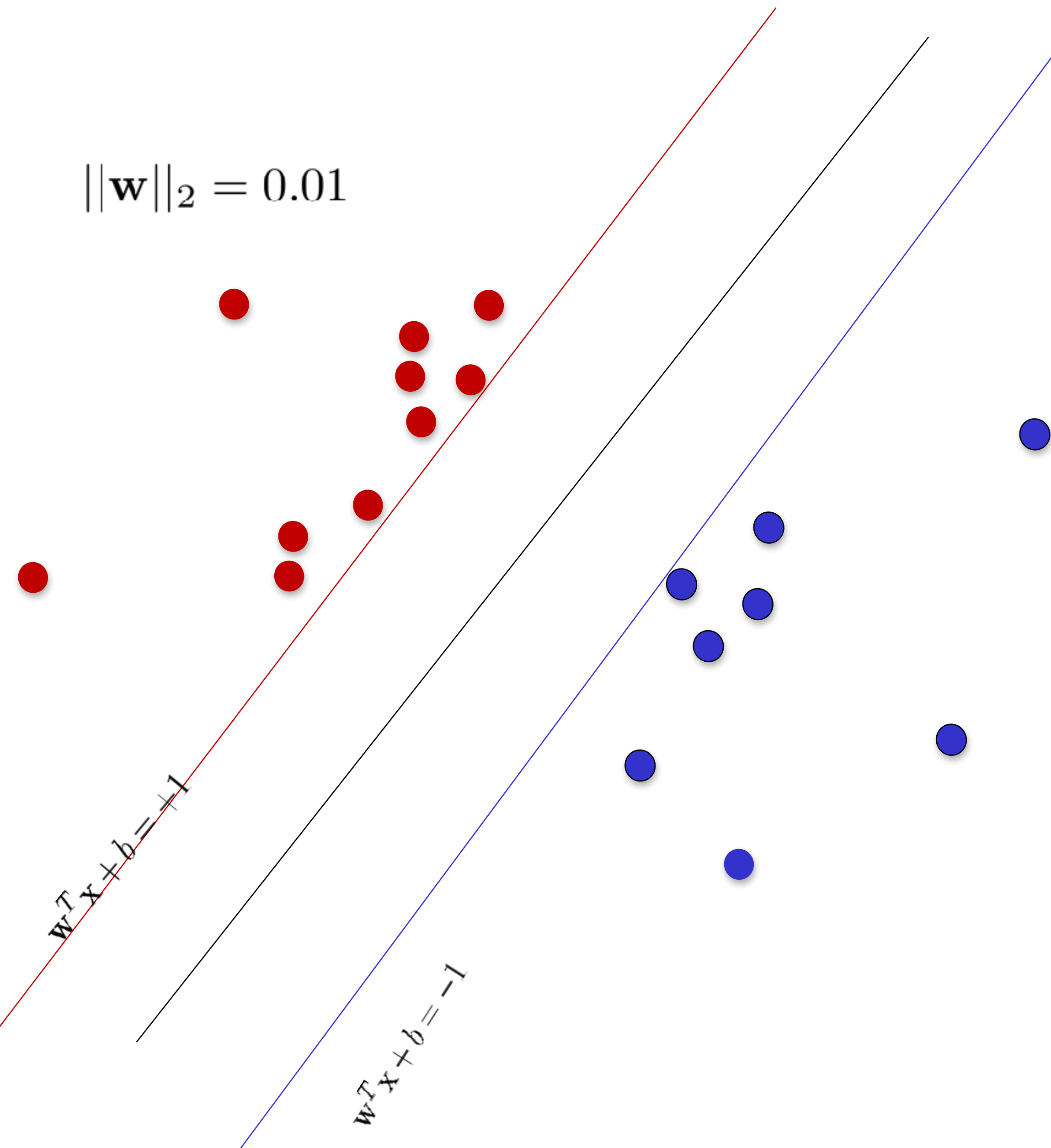
So the overall margin must be twice the distance from the positive margin to the decision boundary

$$M = \frac{2}{\sqrt{\mathbf{w}^T \mathbf{w}}}$$

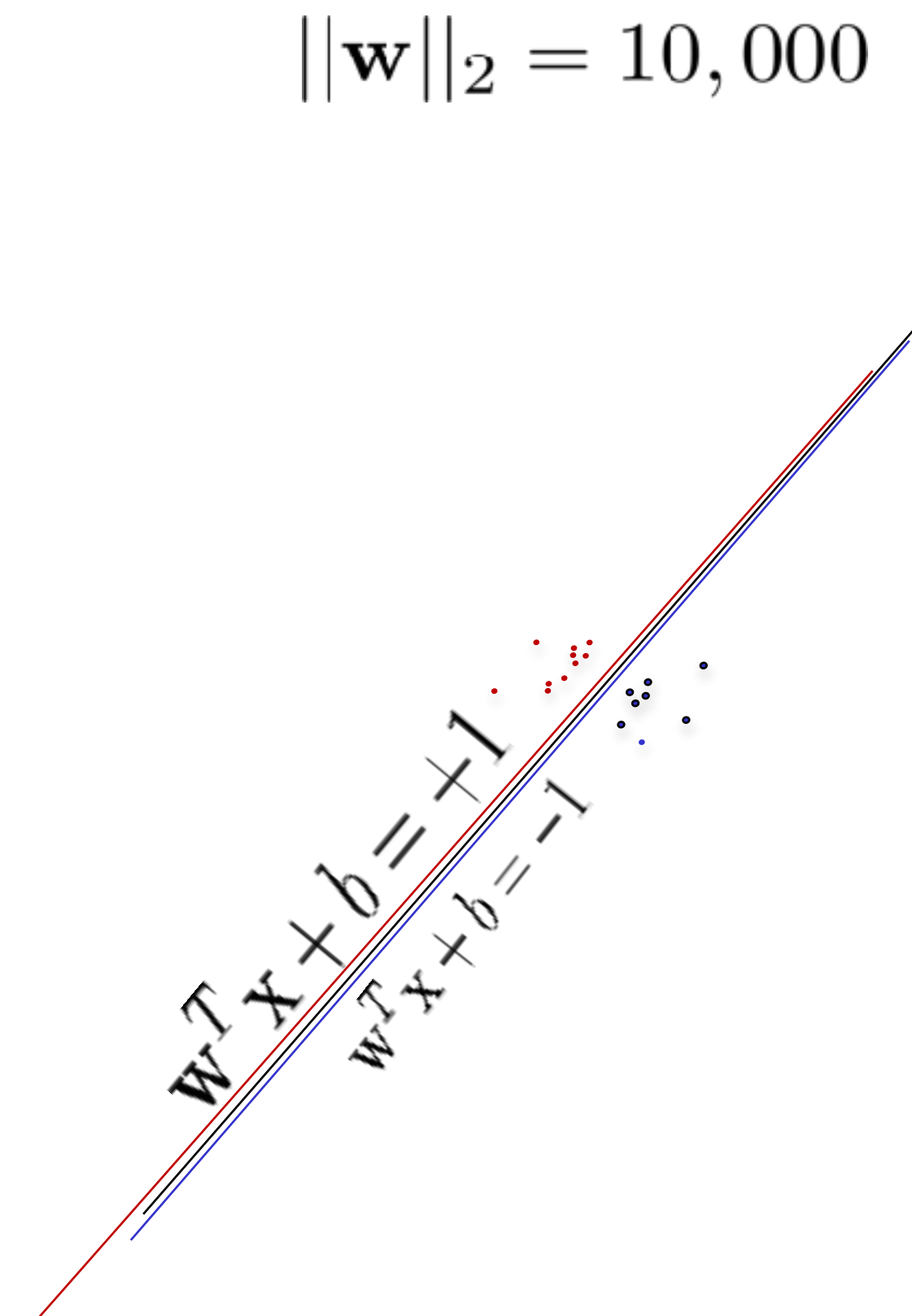
Because the L2 norm is the square root of the squares



# Large margin, small $w$



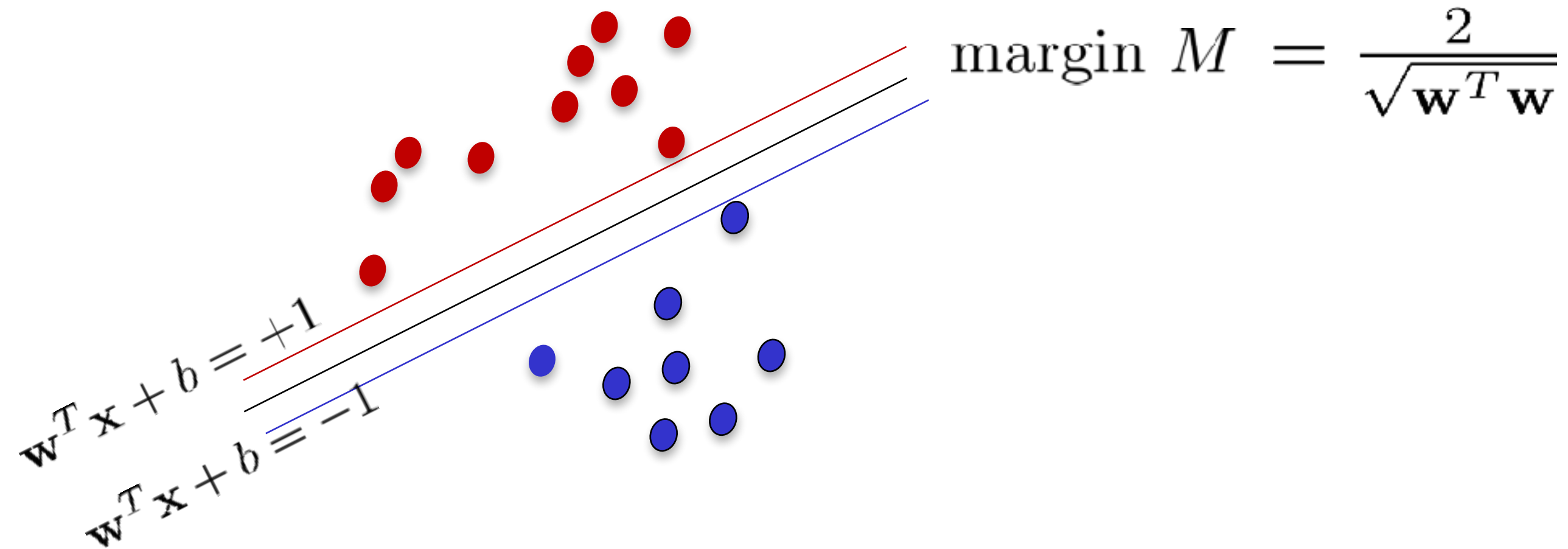
# Small margin, large $w$





# Training SVM using gradient descent

---



Separable case: all positive and negative points are perfectly separable.

Maximizing  $\frac{2}{\sqrt{\mathbf{w}^T \mathbf{w}}}$  is equivalent to minimizing  $\mathbf{w}^T \mathbf{w} = \|\mathbf{w}\|^2$

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$$

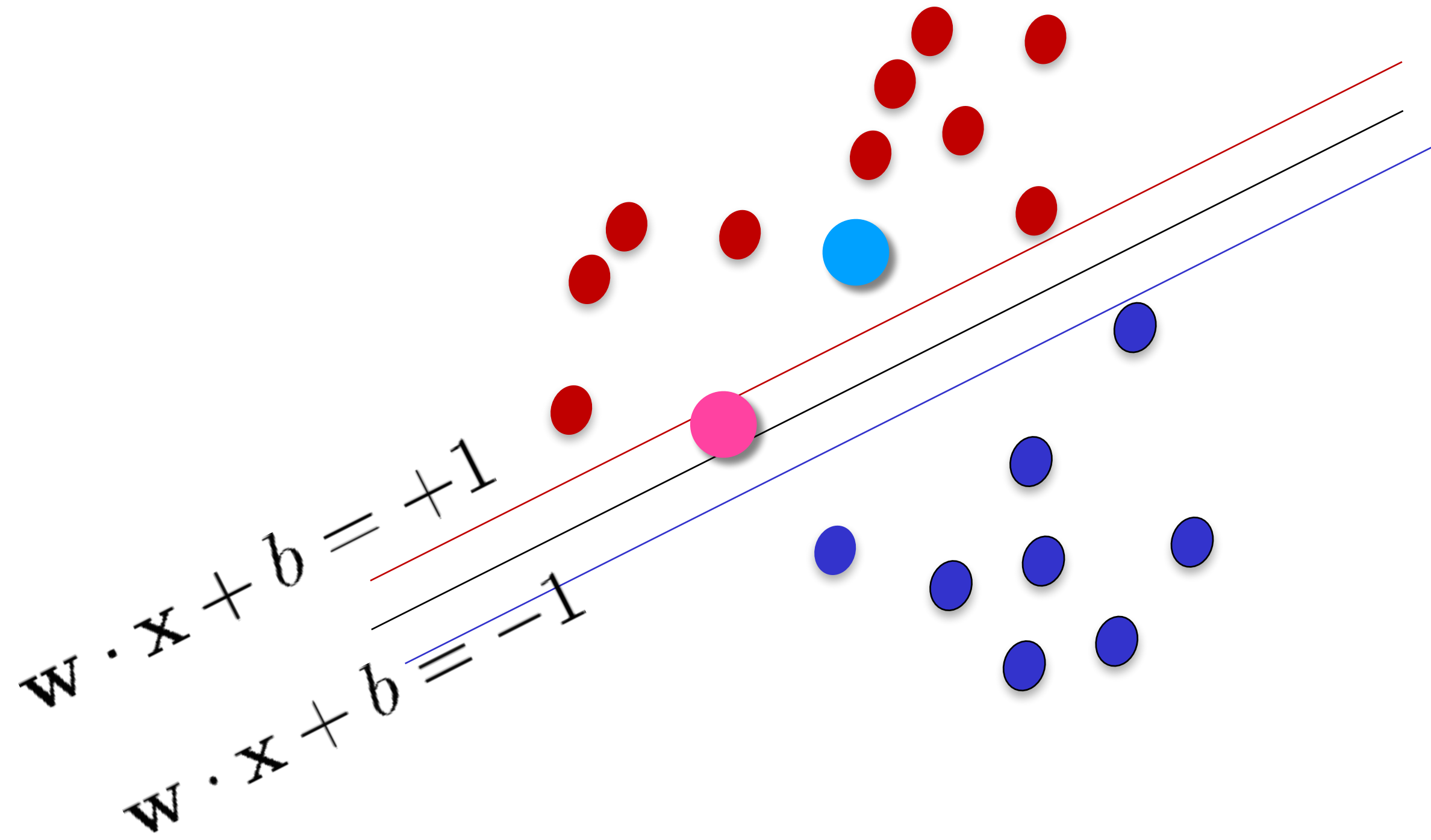
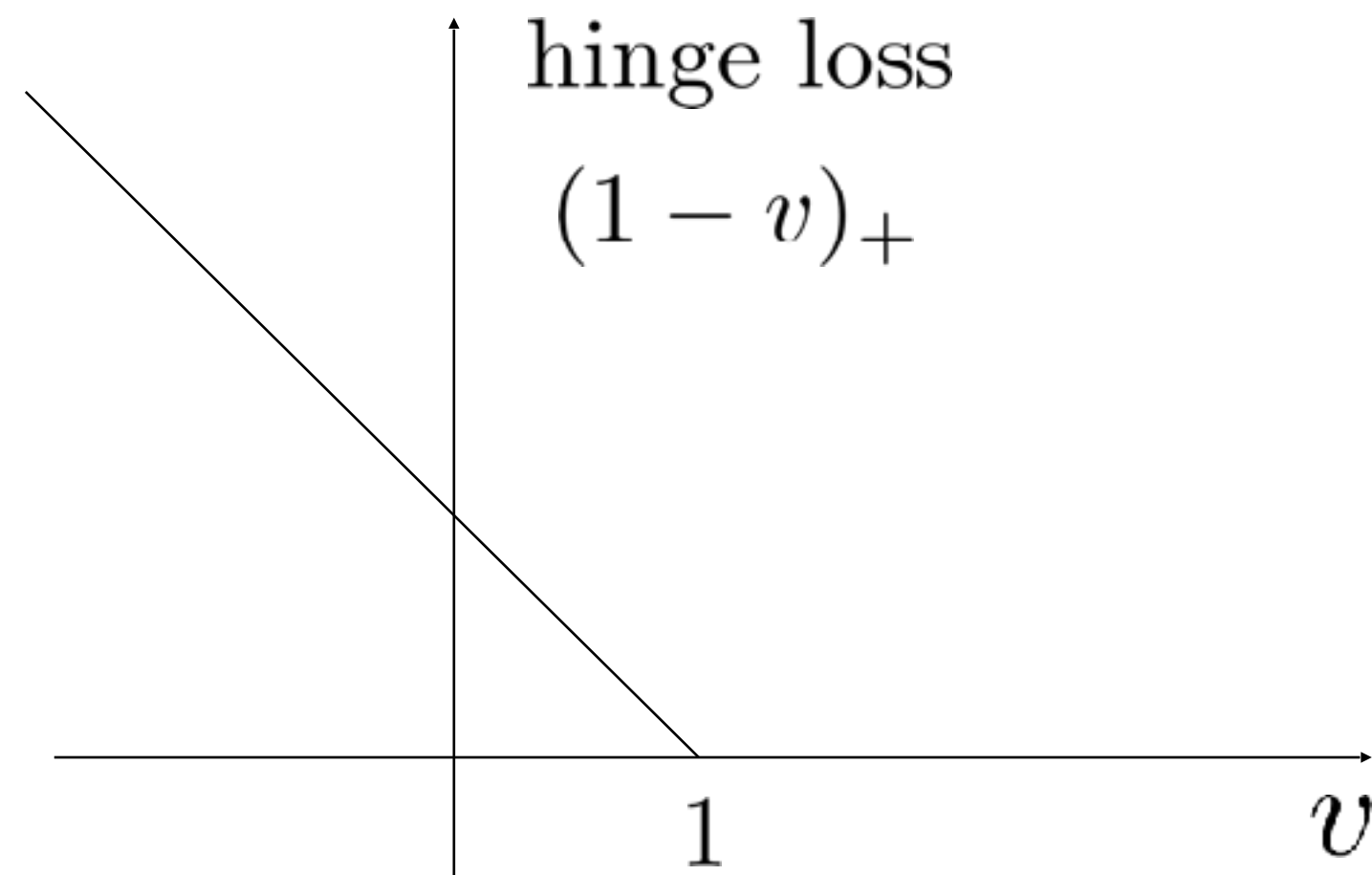
$$\text{Find: } \arg \min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{subject to } y_i(\mathbf{w}^T \mathbf{x} + b) - 1 \geq 0$$

# Hinge Loss

Find:  $\arg \min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2$

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

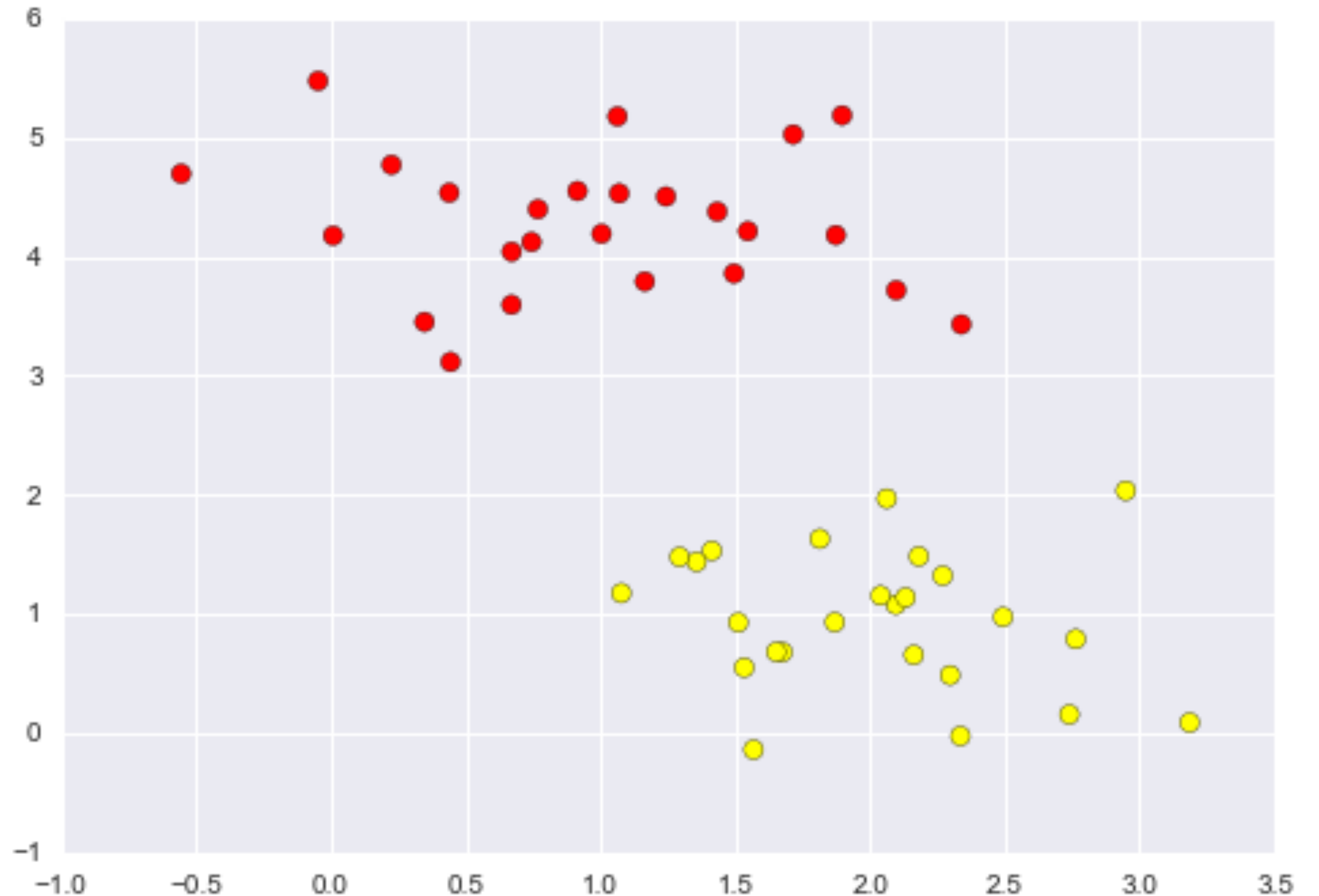
Hinge:  $(1 - v)_+ = \max(0, 1 - v)$



Find:  $\arg \min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 + C \times \sum_{i=1}^n (1 - y_i \times (\mathbf{w}^T \mathbf{x}_i + b))_+$

# Lecture 16 pre-video

**svm:  
Dual  
Form**



For the case of only one constraint and only two choice variables (as exemplified in Figure 1), consider the [optimization problem](#)

$$\begin{aligned} &\text{maximize } f(x, y) \\ &\text{subject to: } g(x, y) = 0 \end{aligned}$$

(Sometimes an additive constant is shown separately rather than being included in  $g$ , in which case the constraint is written  $g(x, y) = c$ , as in Figure 1.) We assume that both  $f$  and  $g$  have continuous first [partial derivatives](#). We introduce a new variable ( $\lambda$ ) called a **Lagrange multiplier** (or **Lagrange undetermined multiplier**) and study the **Lagrange function** (or **Lagrangian** or **Lagrangian expression**) defined by

$$\mathcal{L}(x, y, \lambda) = f(x, y) - \lambda g(x, y),$$

where the  $\lambda$  term may be either added or subtracted. If  $f(x_0, y_0)$  is a maximum of  $f(x, y)$  for the original constrained problem and  $\nabla g(x_0, y_0) \neq 0$ , then there exists  $\lambda_0$  such that  $(x_0, y_0, \lambda_0)$  is a [stationary point](#) for the Lagrange function (stationary points are those points where the first partial derivatives of  $\mathcal{L}$  are zero). The assumption  $\nabla g \neq 0$  is called constraint qualification. However, not all stationary points yield a solution of the original problem, as the method of Lagrange multipliers yields only a [necessary condition](#) for optimality in constrained problems.<sup>[9][10][11][12][13]</sup> Sufficient conditions for a minimum or maximum [also exist](#), but if a particular [candidate solution](#) satisfies the sufficient conditions, it is only guaranteed that that solution is the best one *locally* – that is, it is better than any permissible nearby points. The *global* optimum can be found by comparing the values of the original objective function at the points satisfying the necessary and locally sufficient conditions.

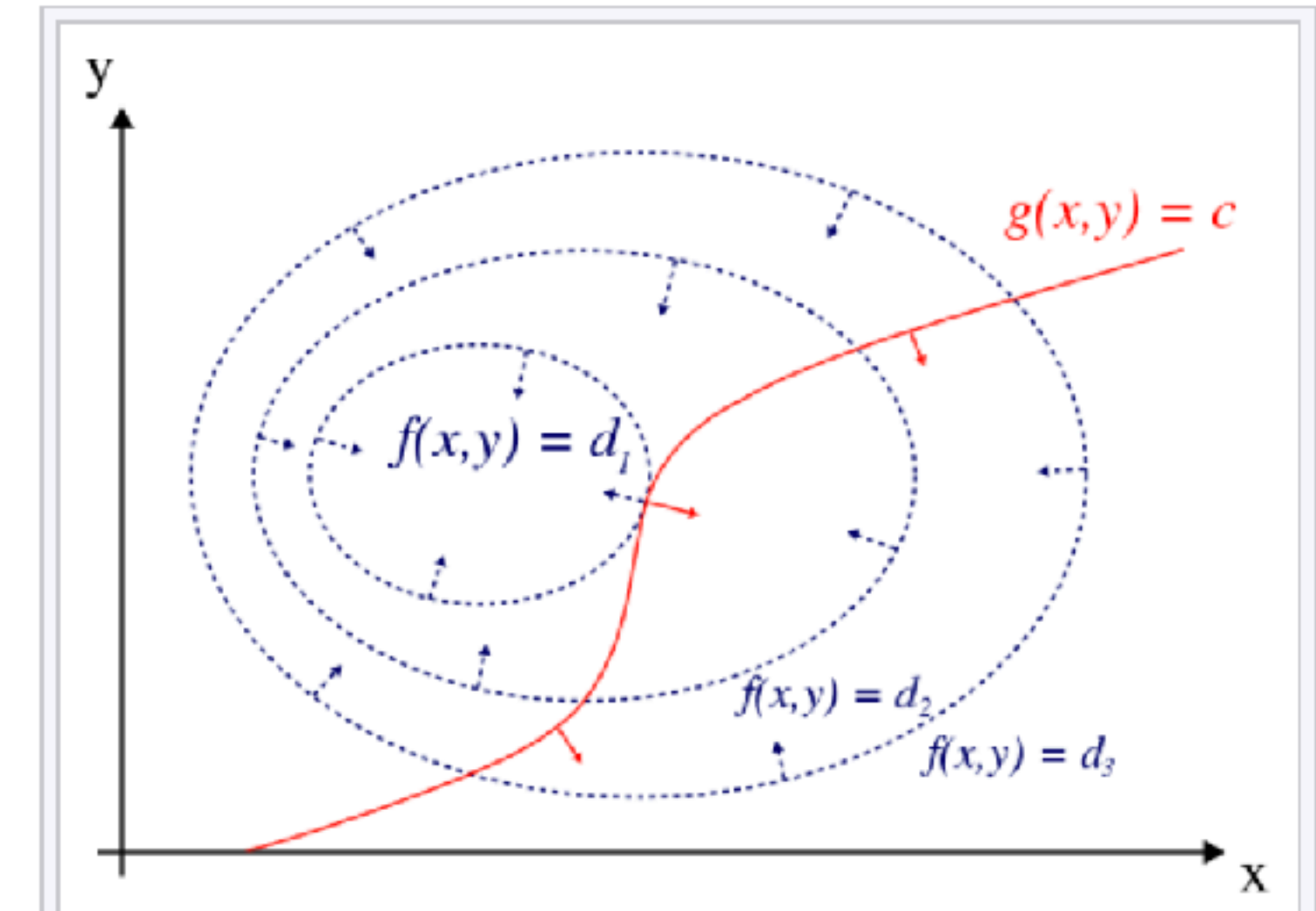


Figure 1: The red curve shows the constraint  $g(x, y) = c$ . The blue curves are contours of  $f(x, y)$ . The point where the red constraint tangentially touches a blue contour is the maximum of  $f(x, y)$  along the constraint, since  $d_1 > d_2$ .



# Sketch derivation of dual form

---

The **Representer Theorem** states that the solution  $\mathbf{w}$  can always be written as a linear combination of the training data:

$$\mathbf{w} = \sum_{j=1}^N \alpha_j y_j \mathbf{x}_j$$

Proof: **see example sheet** .

Now, substitute for  $\mathbf{w}$  in  $f(x) = \mathbf{w}^\top \mathbf{x} + b$

$$f(x) = \left( \sum_{j=1}^N \alpha_j y_j \mathbf{x}_j \right)^\top \mathbf{x} + b = \sum_{j=1}^N \alpha_j y_j (\mathbf{x}_j^\top \mathbf{x}) + b$$

and for  $\mathbf{w}$  in the cost function  $\min_{\mathbf{w}} \|\mathbf{w}\|^2$  subject to  $y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \forall i$

$$\|\mathbf{w}\|^2 = \left\{ \sum_j \alpha_j y_j \mathbf{x}_j \right\}^\top \left\{ \sum_k \alpha_k y_k \mathbf{x}_k \right\} = \sum_{jk} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j^\top \mathbf{x}_k)$$

Hence, an equivalent optimization problem is over  $\alpha_j$

$$\min_{\alpha_j} \sum_{jk} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j^\top \mathbf{x}_k) \quad \text{subject to} \quad y_i \left( \sum_{j=1}^N \alpha_j y_j (\mathbf{x}_j^\top \mathbf{x}_i) + b \right) \geq 1, \forall i$$

and a few more steps are required to complete the derivation.

# Primal and dual formulations

---

$N$  is number of training points, and  $d$  is dimension of feature vector  $\mathbf{x}$ .

Primal problem: for  $\mathbf{w} \in \mathbb{R}^d$

$$\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{w}\|^2 + C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i))$$

Dual problem: for  $\alpha \in \mathbb{R}^N$  (stated without proof):

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j^\top \mathbf{x}_k) \text{ subject to } 0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$

- Need to learn  $d$  parameters for primal, and  $N$  for dual
- If  $N \ll d$  then more efficient to solve for  $\alpha$  than  $\mathbf{w}$
- Dual form only involves  $(\mathbf{x}_j^\top \mathbf{x}_k)$ . We will return to why this is an advantage when we look at kernels.



## Primal and dual formulations

---

Primal version of classifier:

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

Dual version of classifier:

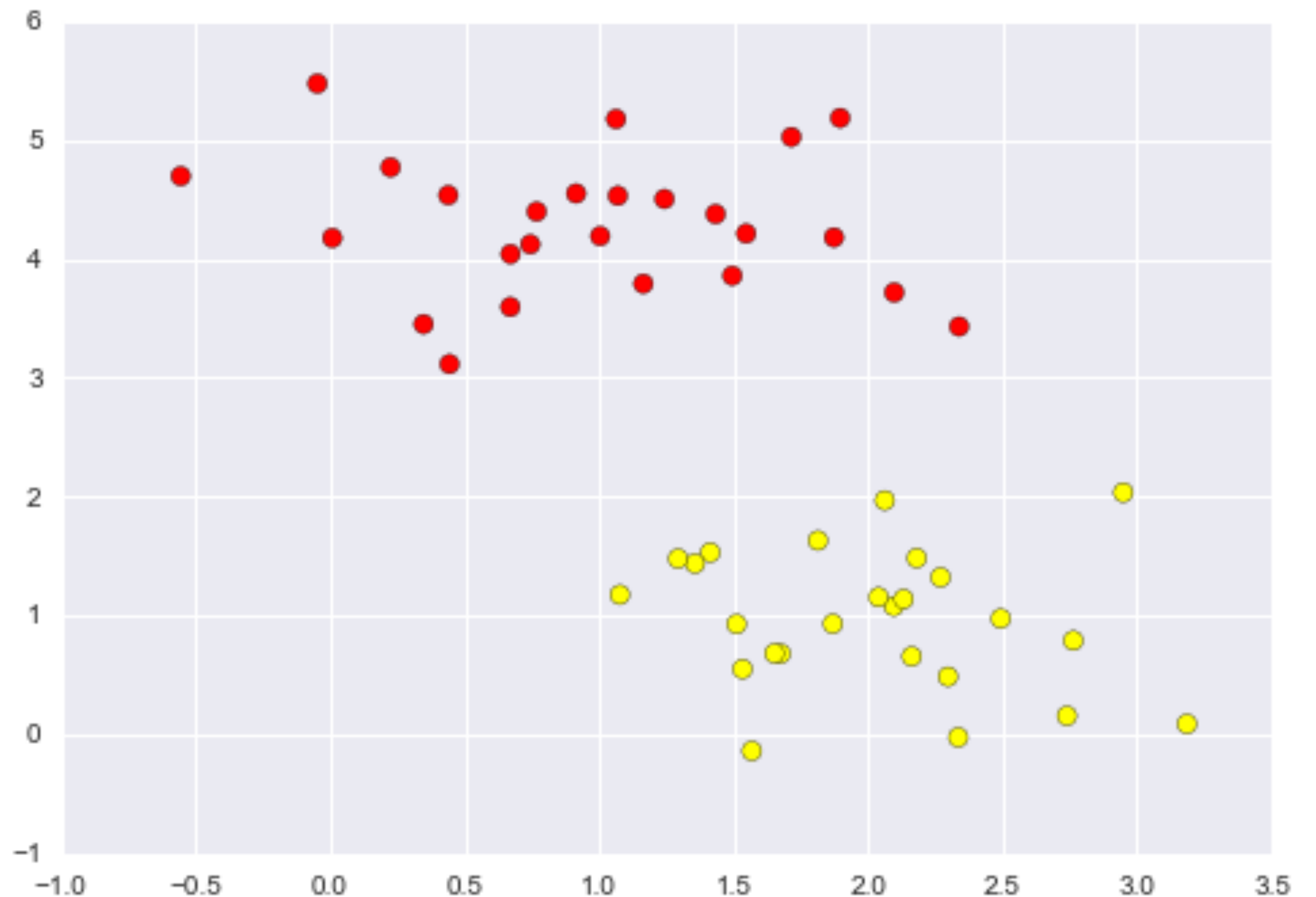
$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i (\mathbf{x}_i^\top \mathbf{x}) + b$$

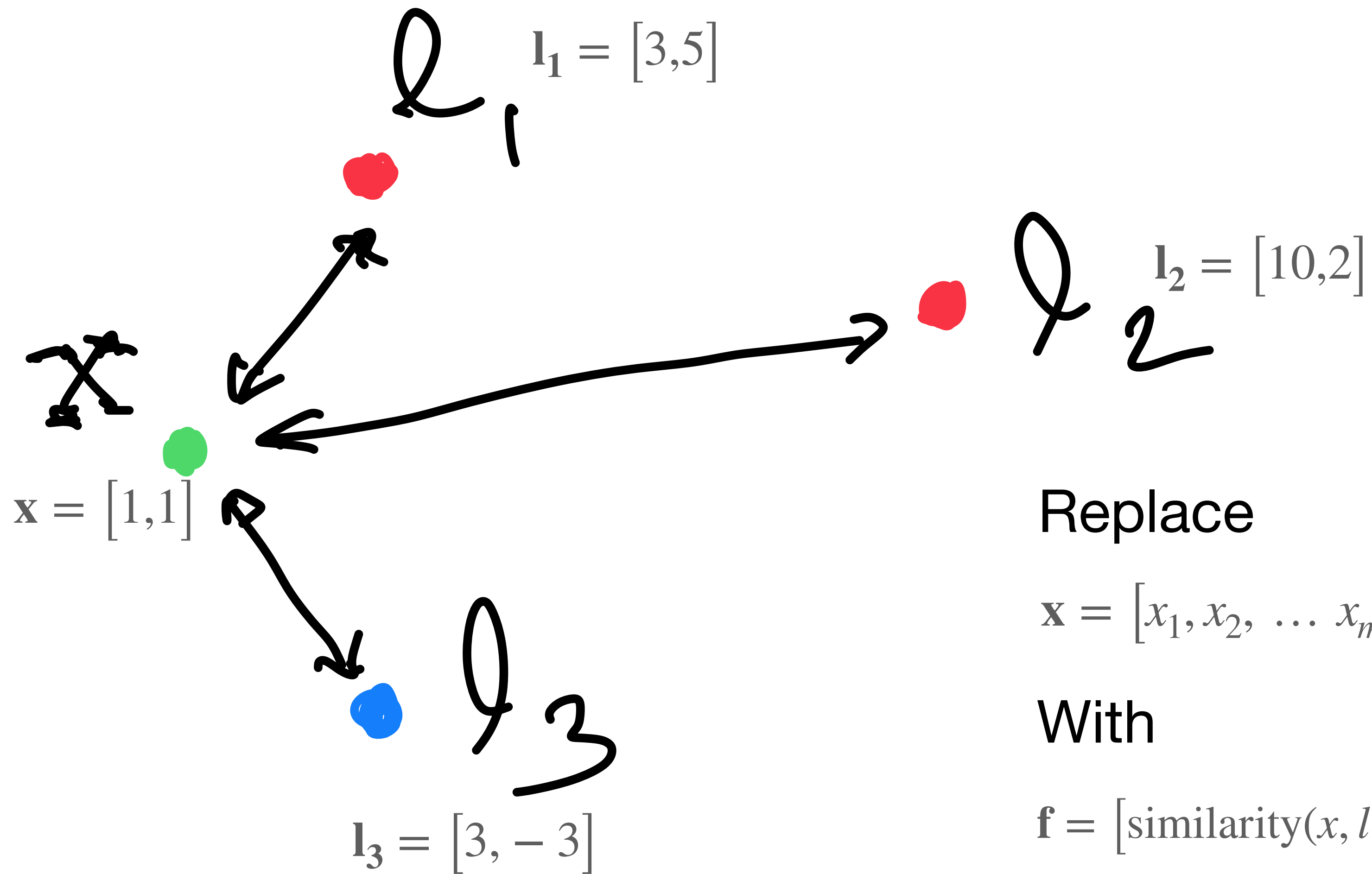
At first sight the dual form appears to have the disadvantage of a K-NN classifier – it requires the training data points  $\mathbf{x}_i$ . However, many of the  $\alpha_i$ 's are zero. The ones that are non-zero define the support vectors  $\mathbf{x}_i$ .



# Lecture 17 pre-video

**svm:**  
**The kernel**  
**trick**



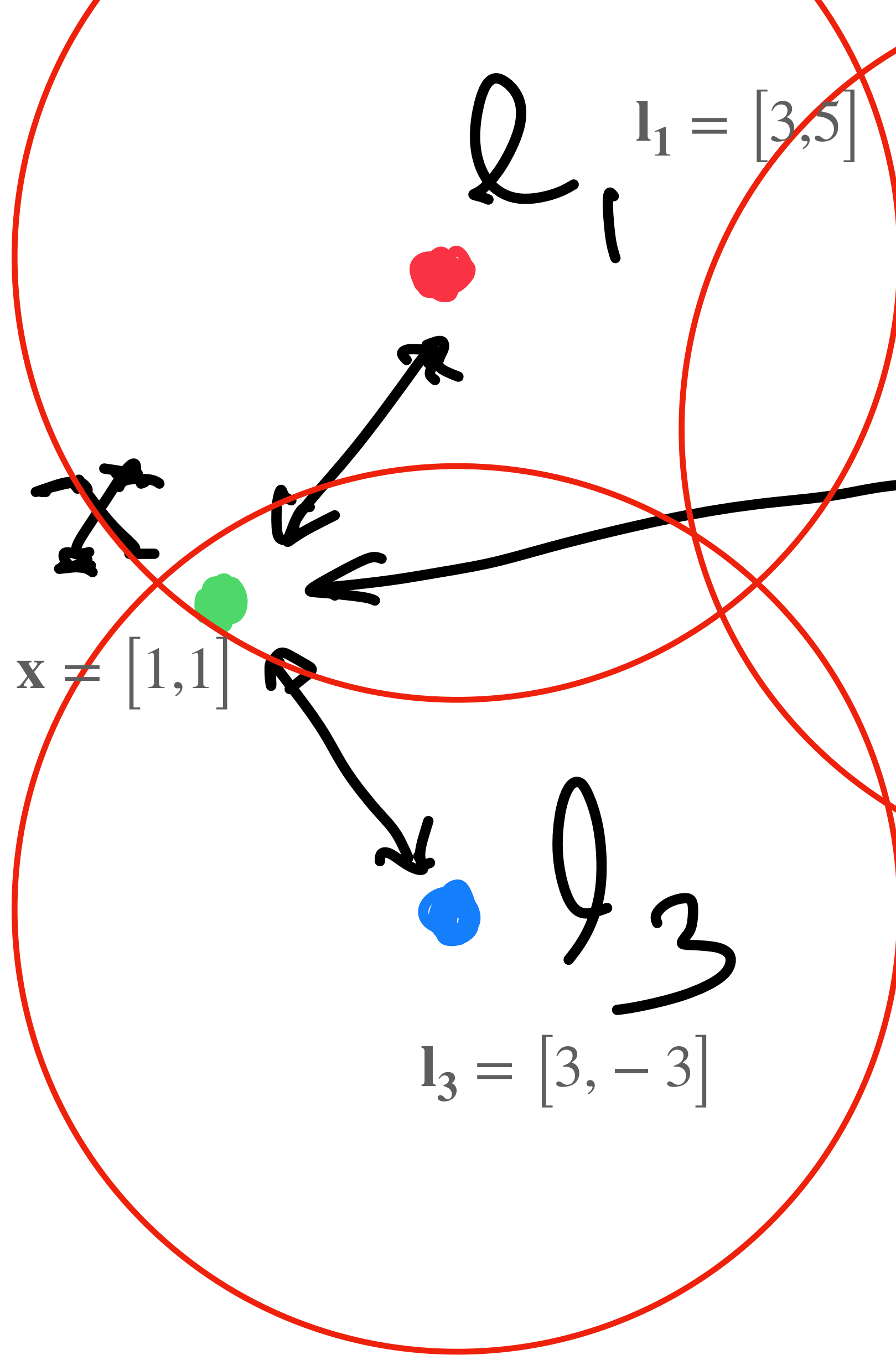


Replace

$$\mathbf{x} = [x_1, x_2, \dots, x_m]^T$$

With

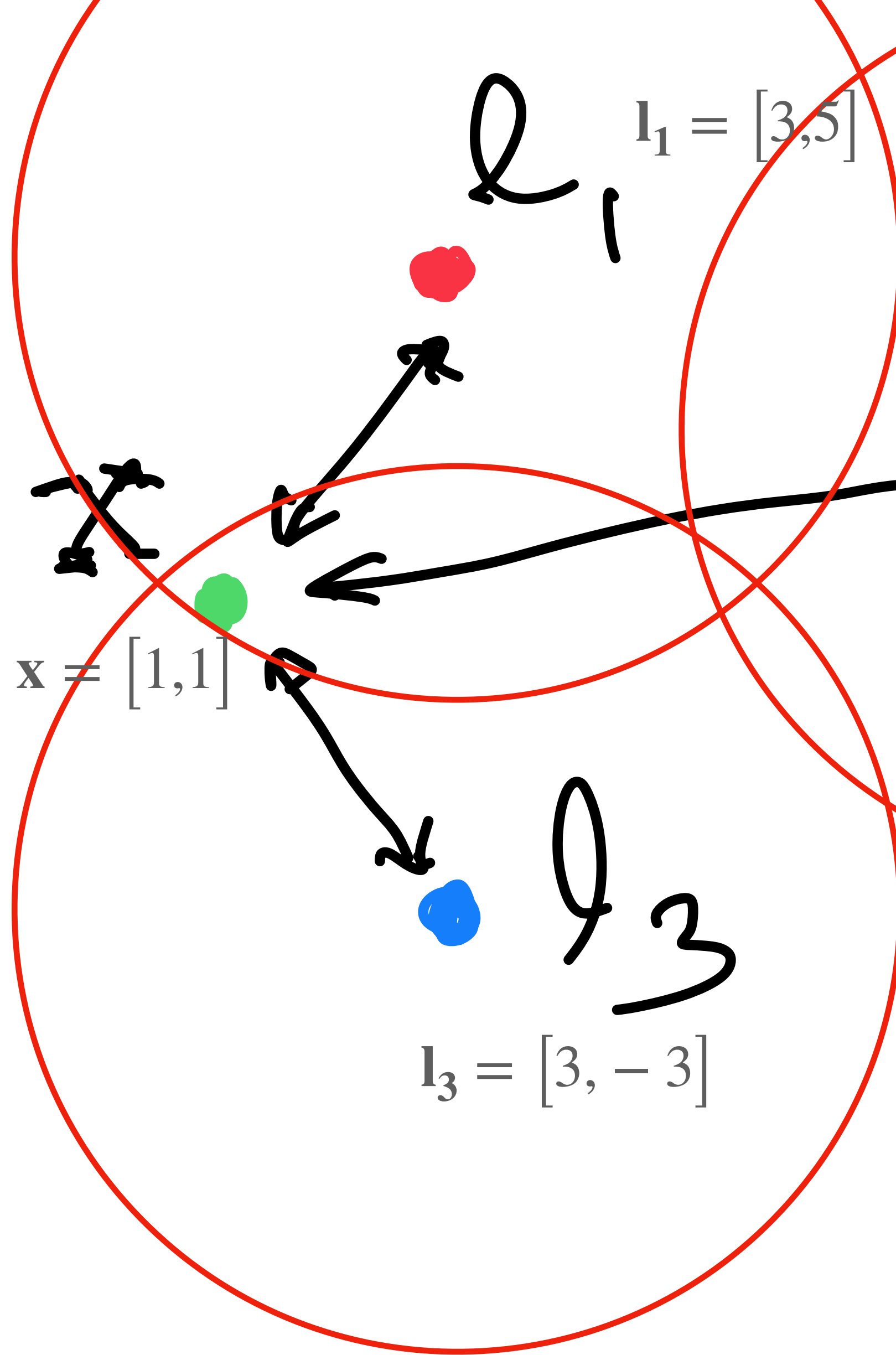
$$\mathbf{f} = [\text{similarity}(x, l_1), \text{similarity}(x, l_2), \dots, \text{similarity}(x, l_n)]$$



$$f(x, l, \sigma) = \exp \left( \frac{\|x - l\|}{\sigma^2} \right)$$

Replace  $\mathbf{x} = [1, 1]$  with

$$\begin{aligned} & [f(\mathbf{x}, l_1, \sigma = 10), f(\mathbf{x}, l_2, \sigma = 10), f(\mathbf{x}, l_3, \sigma = 10)] \\ & = [0.639, 0.404, 0.639] \end{aligned}$$



$$f_i = \exp \left( \frac{\|x - l_i\|}{\sigma^2} \right) \quad f(x) = [0.639, 0.404, 0.639]$$

Predict 1 when

$$\mathbf{w}_0 + \mathbf{w}_1 f_1 + \mathbf{w}_2 f_2 + \mathbf{w}_3 f_3 \geq 0$$

And let's say that

$$\mathbf{w}_0 = -0.1, \mathbf{w}_1 = \mathbf{w}_2 = 1, \mathbf{w}_3 = -1, \sigma = 10$$



# What is a kernel?

Like a metric plus a defined dot product

A **metric** on a set  $X$  is a **function** (called *distance function* or simply **distance**)

$$d : X \times X \rightarrow [0, \infty),$$

where  $[0, \infty)$  is the set of non-negative **real numbers** and for all  $x, y, z \in X$ , the following three axioms are satisfied:

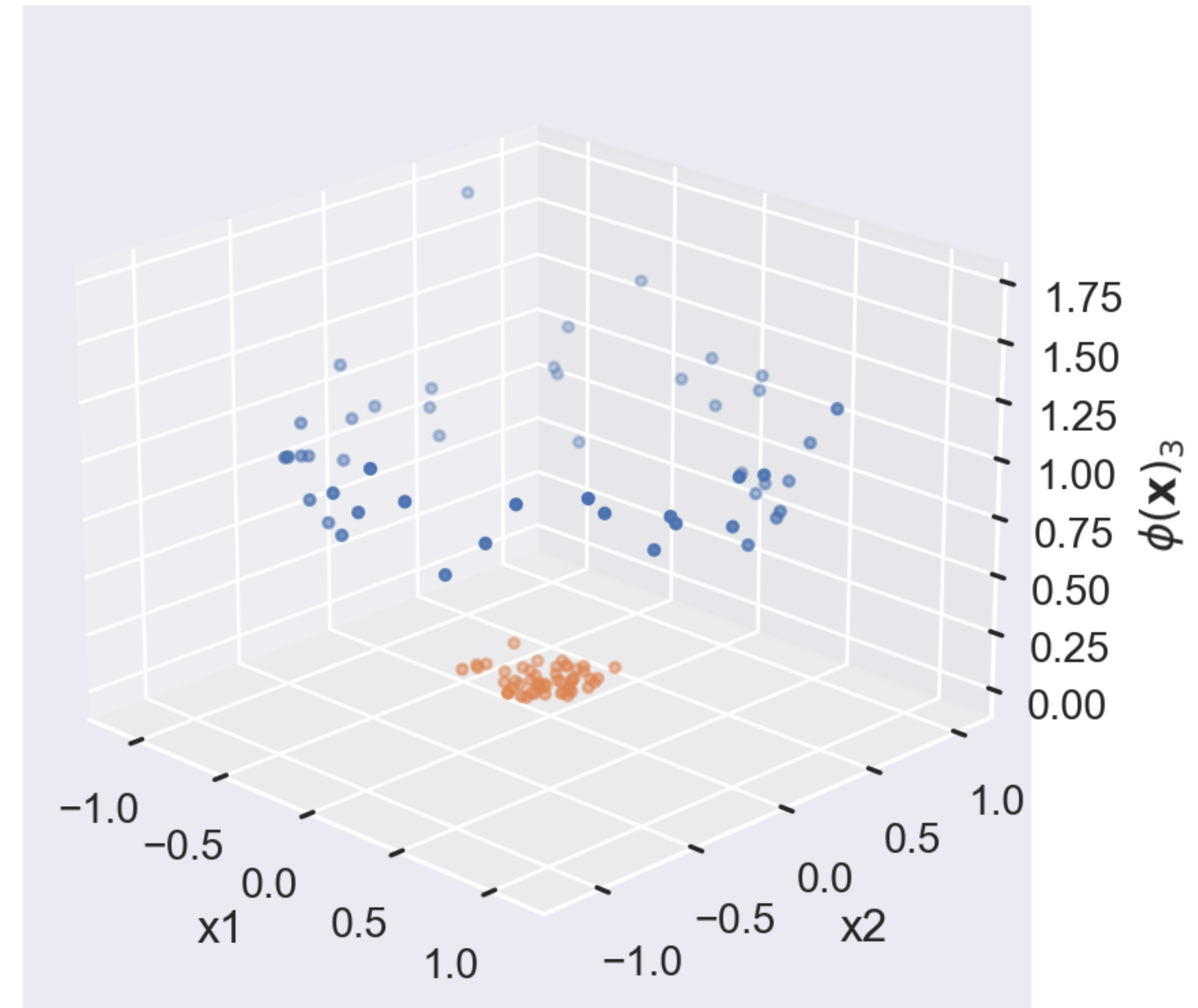
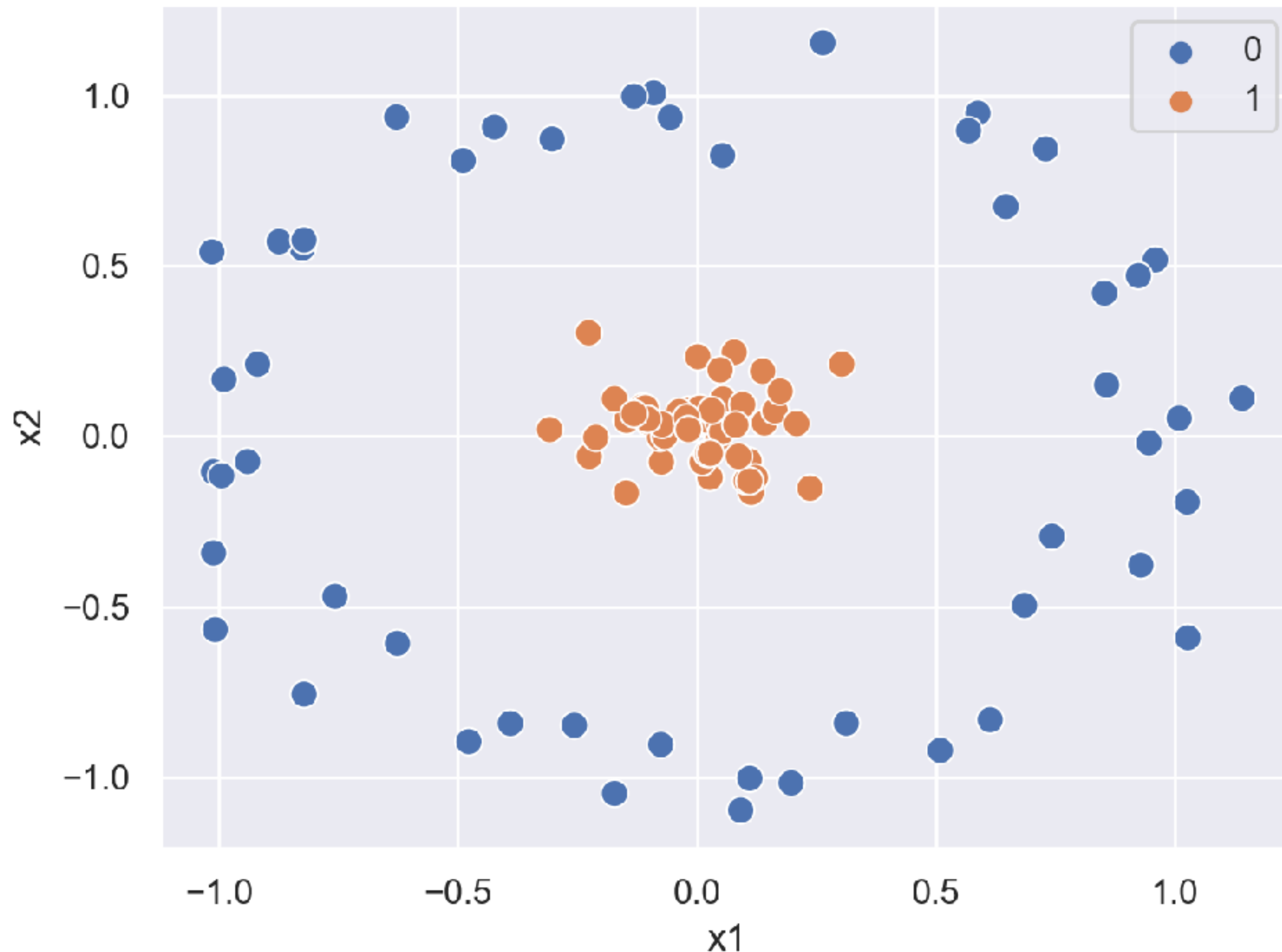
1.  $d(x, y) = 0 \Leftrightarrow x = y$  **identity of indiscernibles**
2.  $d(x, y) = d(y, x)$  **symmetry**
3.  $d(x, y) \leq d(x, z) + d(z, y)$  **subadditivity or triangle inequality**

# What is a kernel?

**Like a metric plus a defined dot product**

- For some vector spaces  $\mathcal{X} \in \mathbb{R}^n, \mathcal{V} \in \mathbb{R}^m$  a kernel is a function,  
 $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$
- The word "kernel" is used in mathematics to denote a weighting function for a weighted sum or integral
- The computation is made much simpler if the kernel can be written in the form of a "feature map"  $\phi : \mathcal{X} \mapsto \mathcal{V}$  that moves the problem from its original vector space to one where it is easier to solve the problem
- The feature map must satisfy  $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{V}}$  where the angle brackets mean this is a proper inner product on the space  $\mathcal{V}$

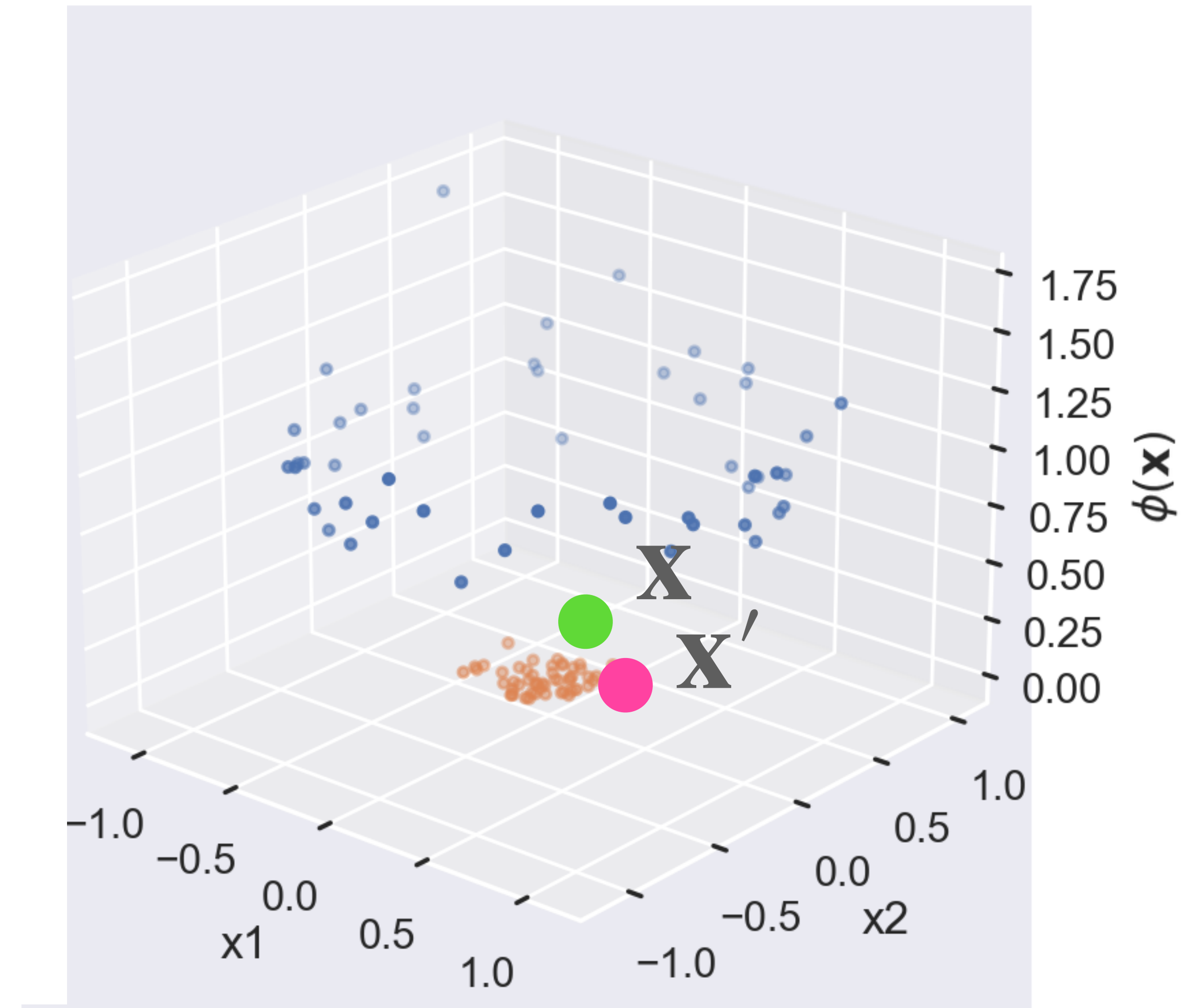
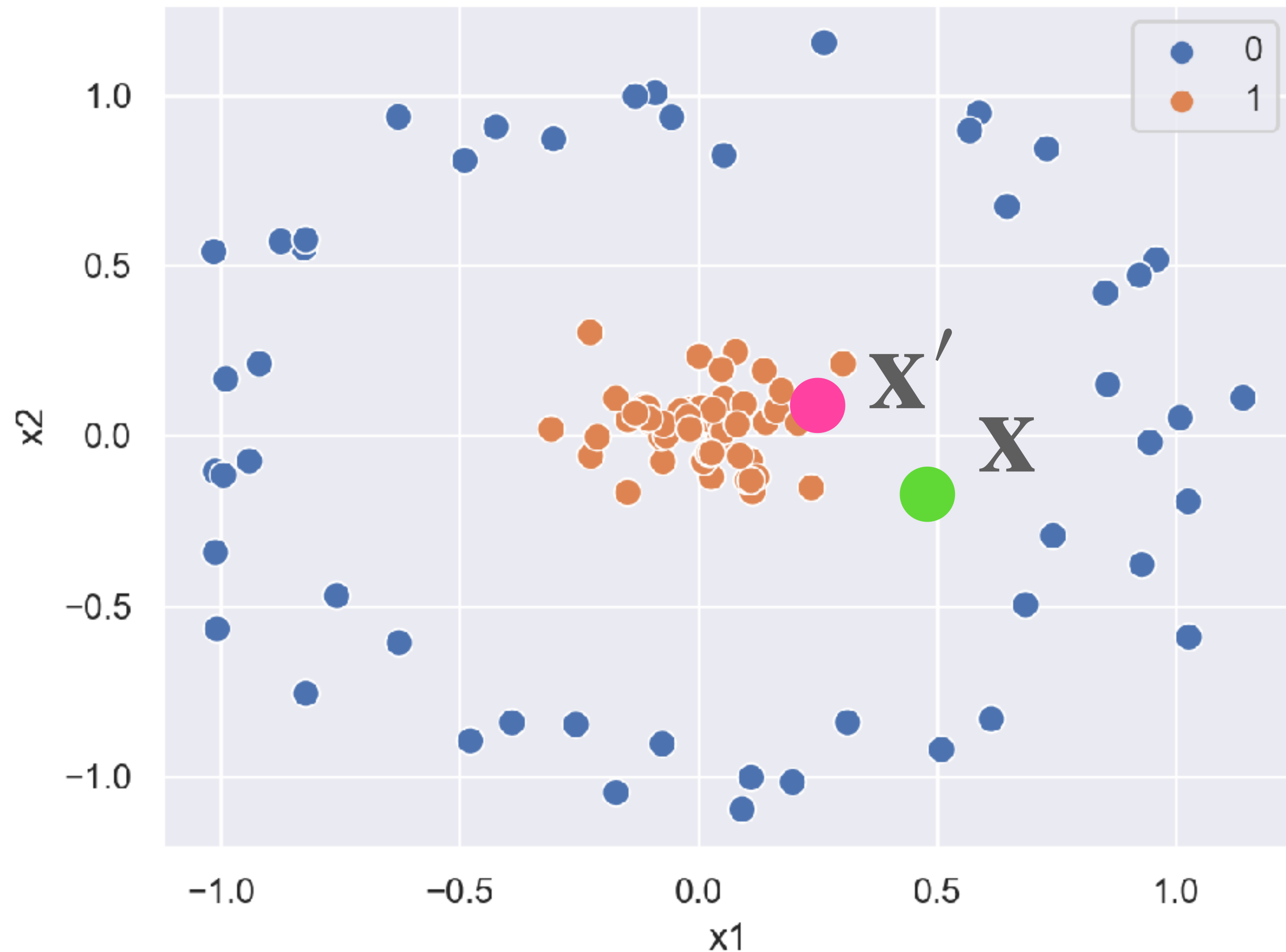
$$\phi(\mathbf{x}) = [x_1, x_2, x_1 \cdot x_2 + x_1^2 \cdot x_2^2]$$



More generally this is the polynomial kernel

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^d = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$$

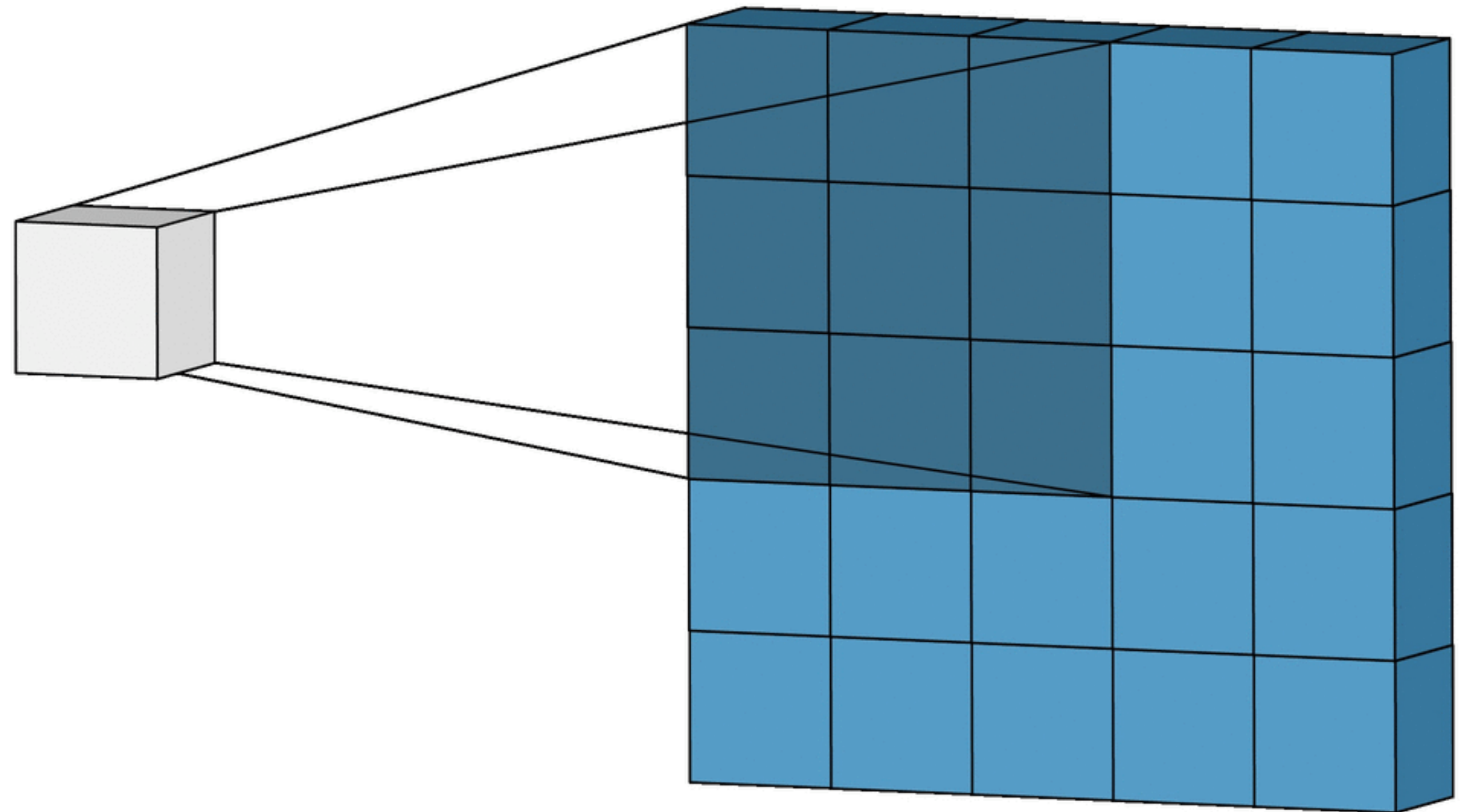
$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$  means you never have to calculate  
the decision boundary in either  $\mathcal{X}$  or  $\mathcal{V}$  ...  
its in terms of a scalar dot product of vectors in  $\mathcal{V}$ !





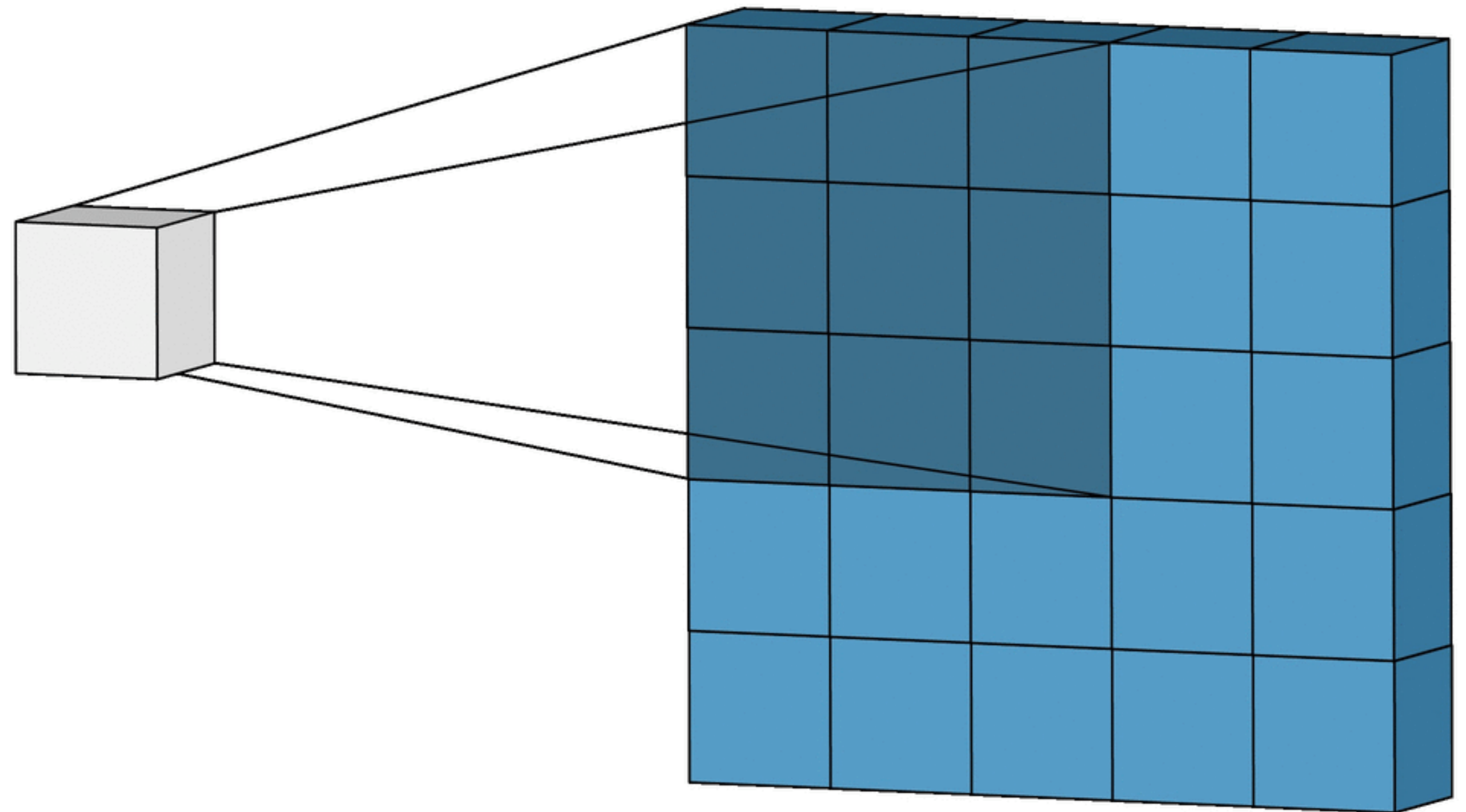
# Common kinds of kernels

- Moving average window
- Polynomial of order  $d$
- Radial basis / Gaussian
- Sigmoid / tanh



# Build your own kernels!

- There's a linear algebra def of what makes a kernel...
- But also anything that adds two kernels is a kernel!
- Multiplying two kernels (convolution) is a kernel!
- sklearn implements a lookup function for new kernels







**The world is not black and white:  
multi-class decision making**

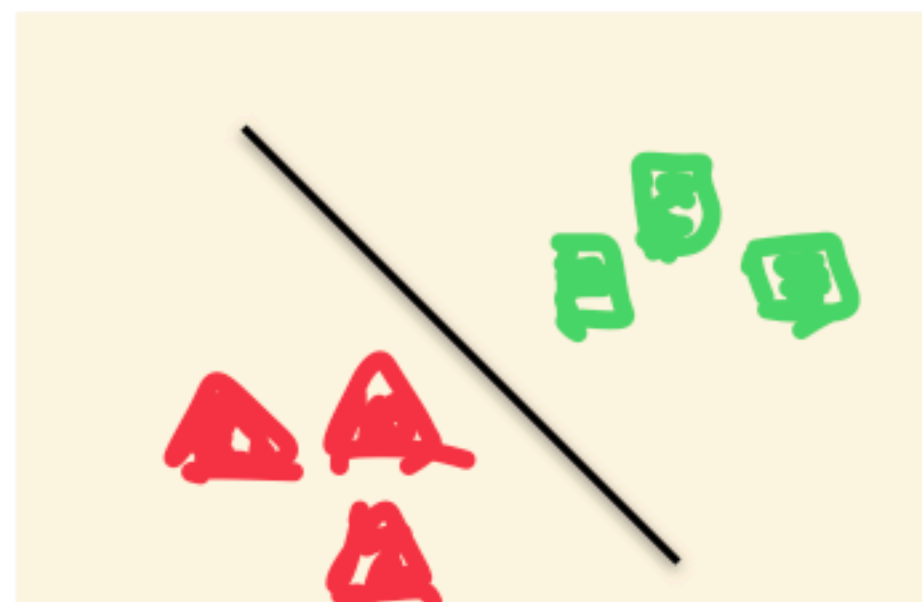
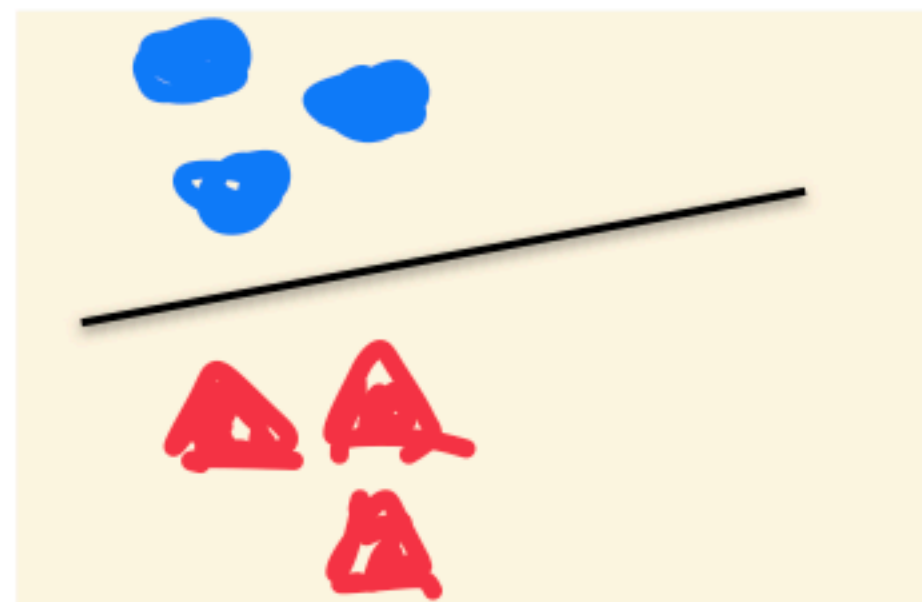
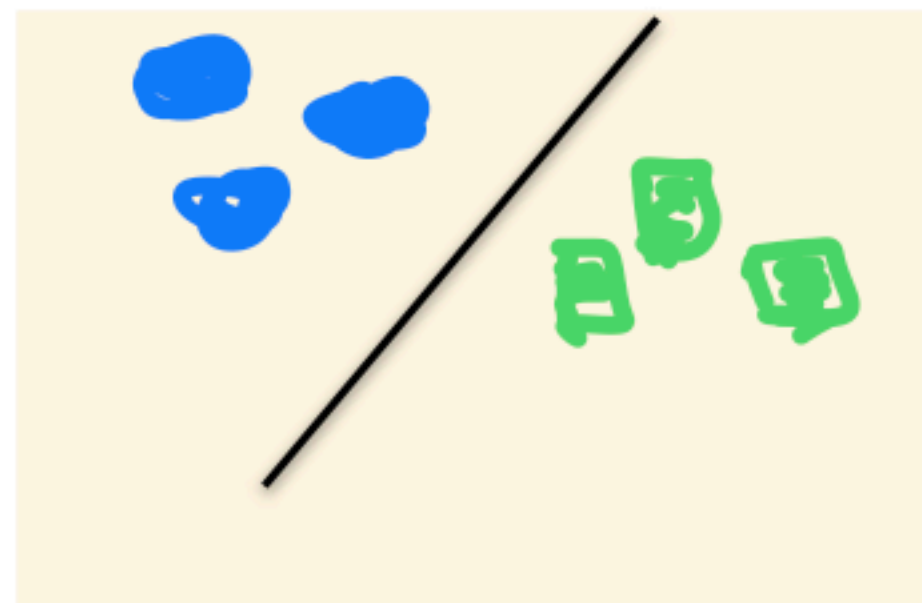


$k=3$   
classes

One v One uses class with highest confidence score

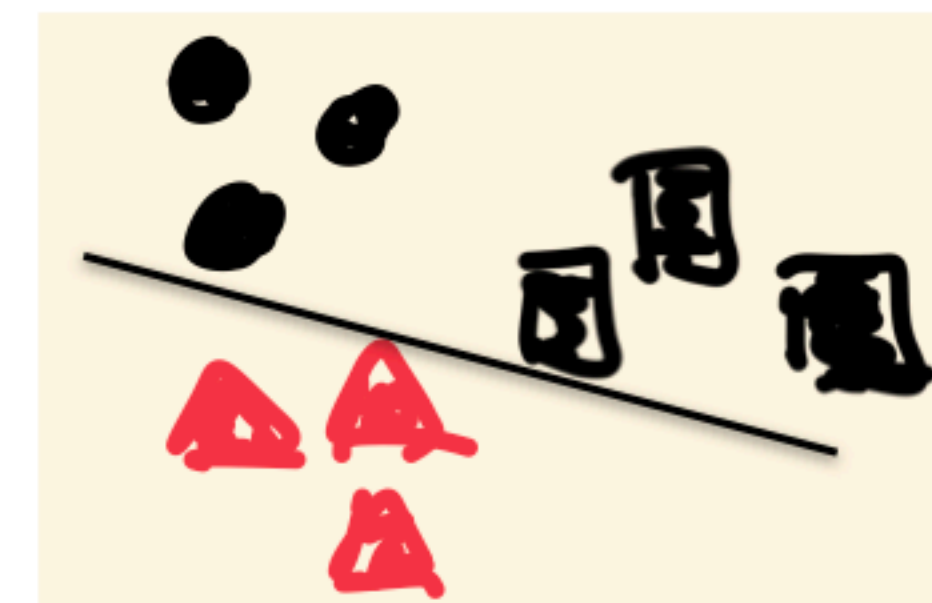
One v Rest uses plurality (mode) vote, tie breaker with confidence score

One v. One



$\frac{k(k-1)}{2}$  classifiers

One v. Rest



$k$  classifiers

# Support Vector Machines

**Jason G. Fleischer, Ph.D.**

**Asst. Teaching Professor**

**Department of Cognitive Science, UC San Diego**

**[jfleischer@ucsd.edu](mailto:jfleischer@ucsd.edu)**

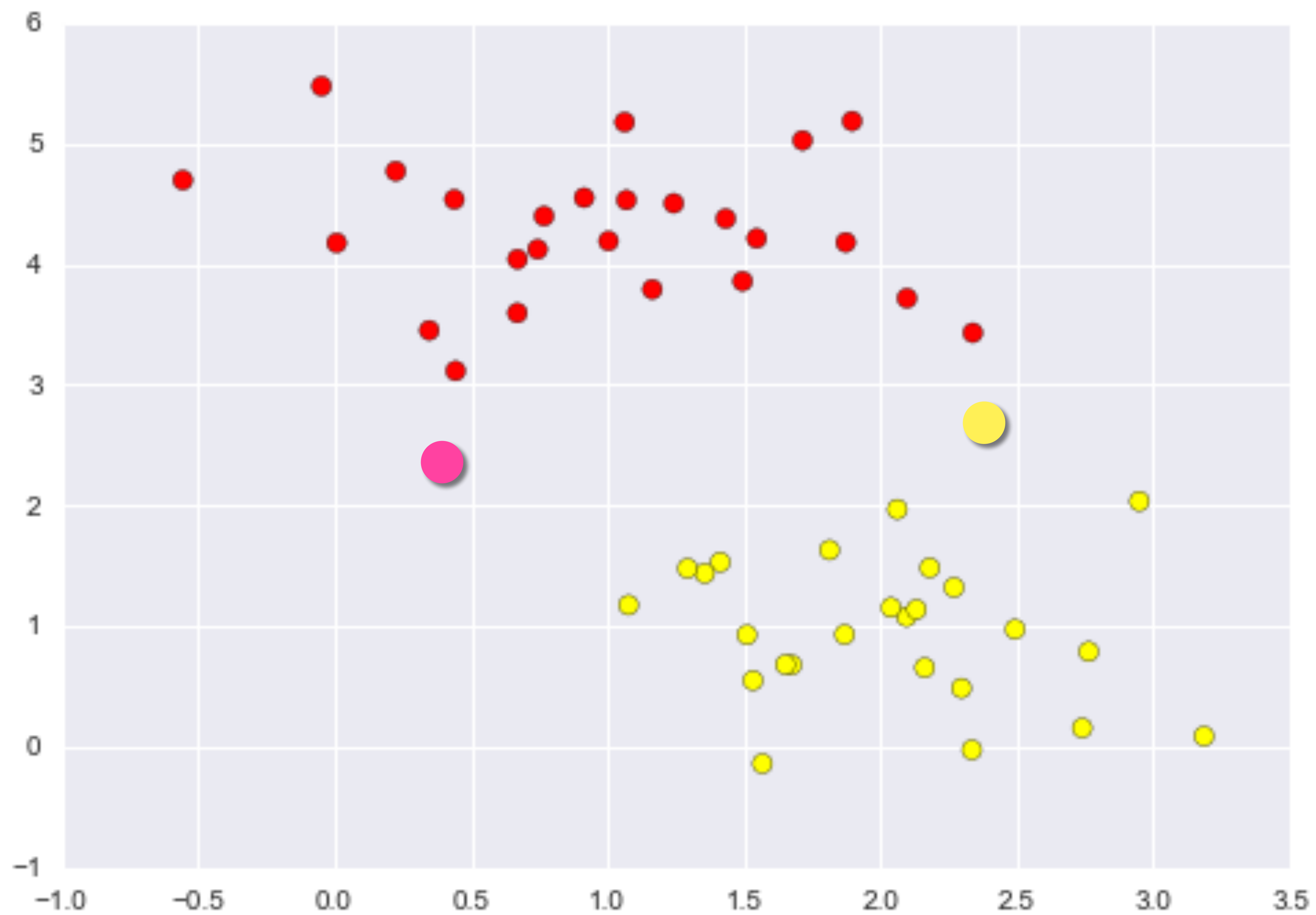
 **@jasongfleischer**

**<https://jgfleischer.com>**

Slides in this presentation are from material kindly provided by  
Zhuowen Tu and others credited at those slides

**Mid course survey, please respond!!**

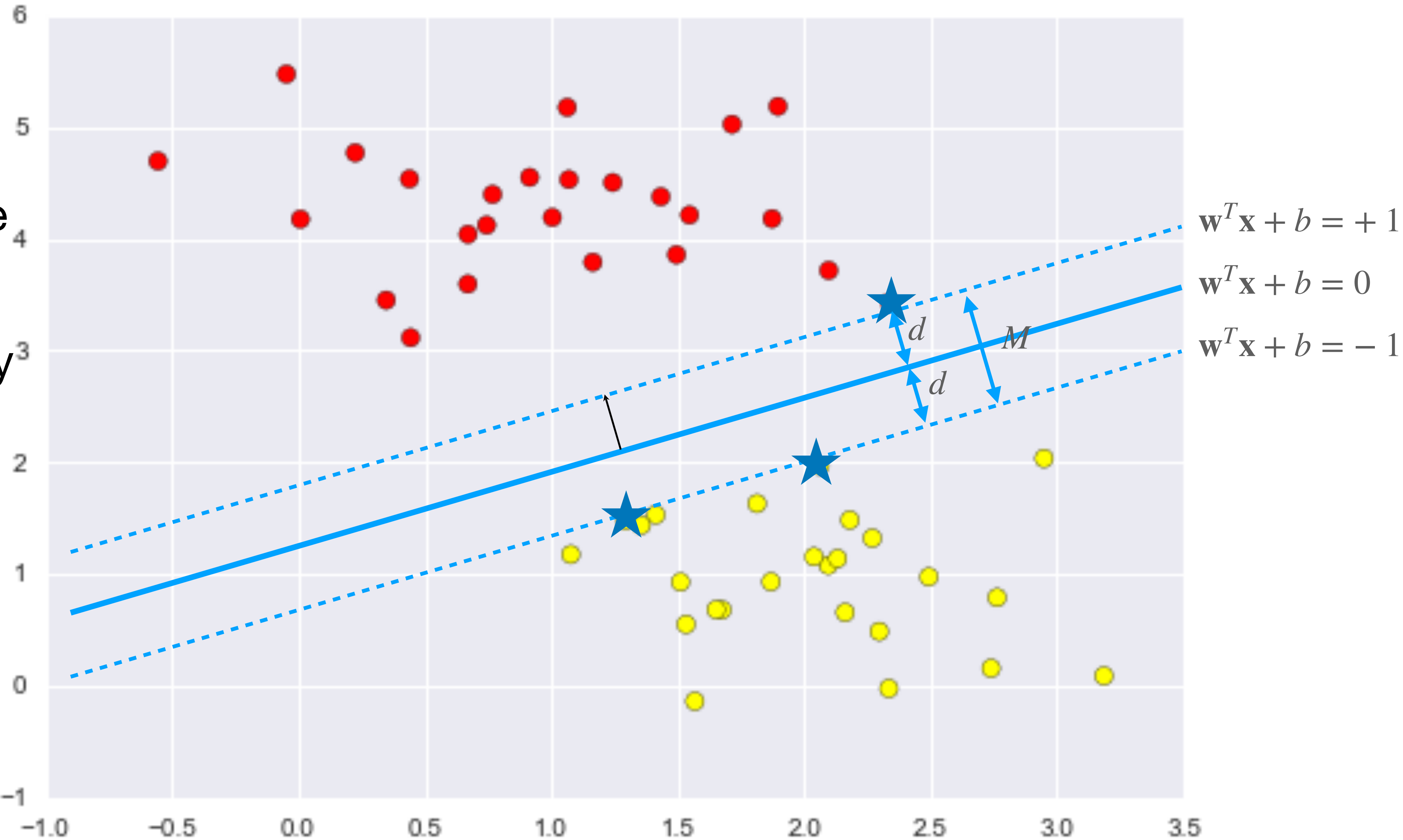
**<https://forms.gle/FyoiqAdpozczjxeTA>**



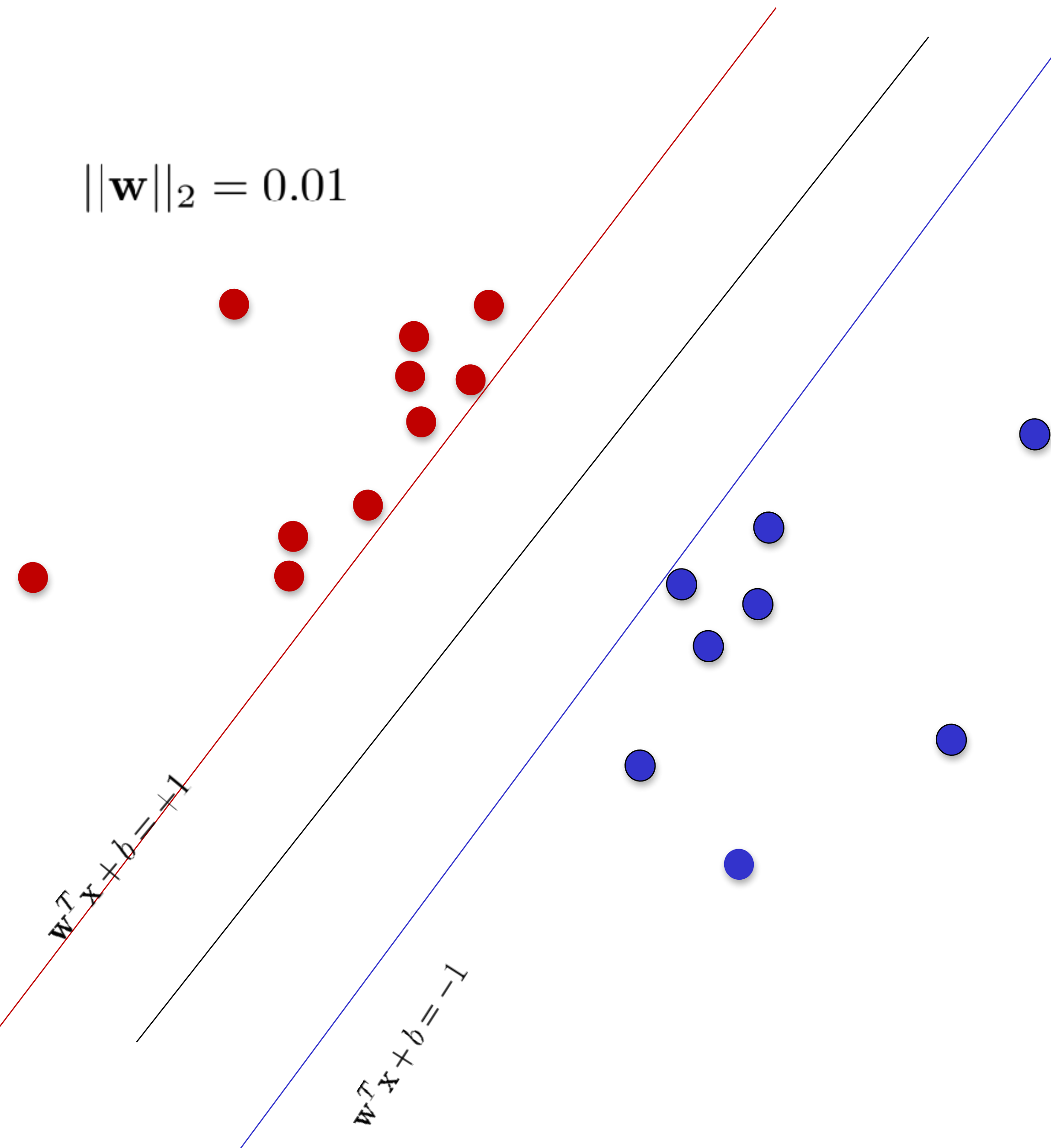


So the overall margin must be twice the distance from the positive margin to the decision boundary

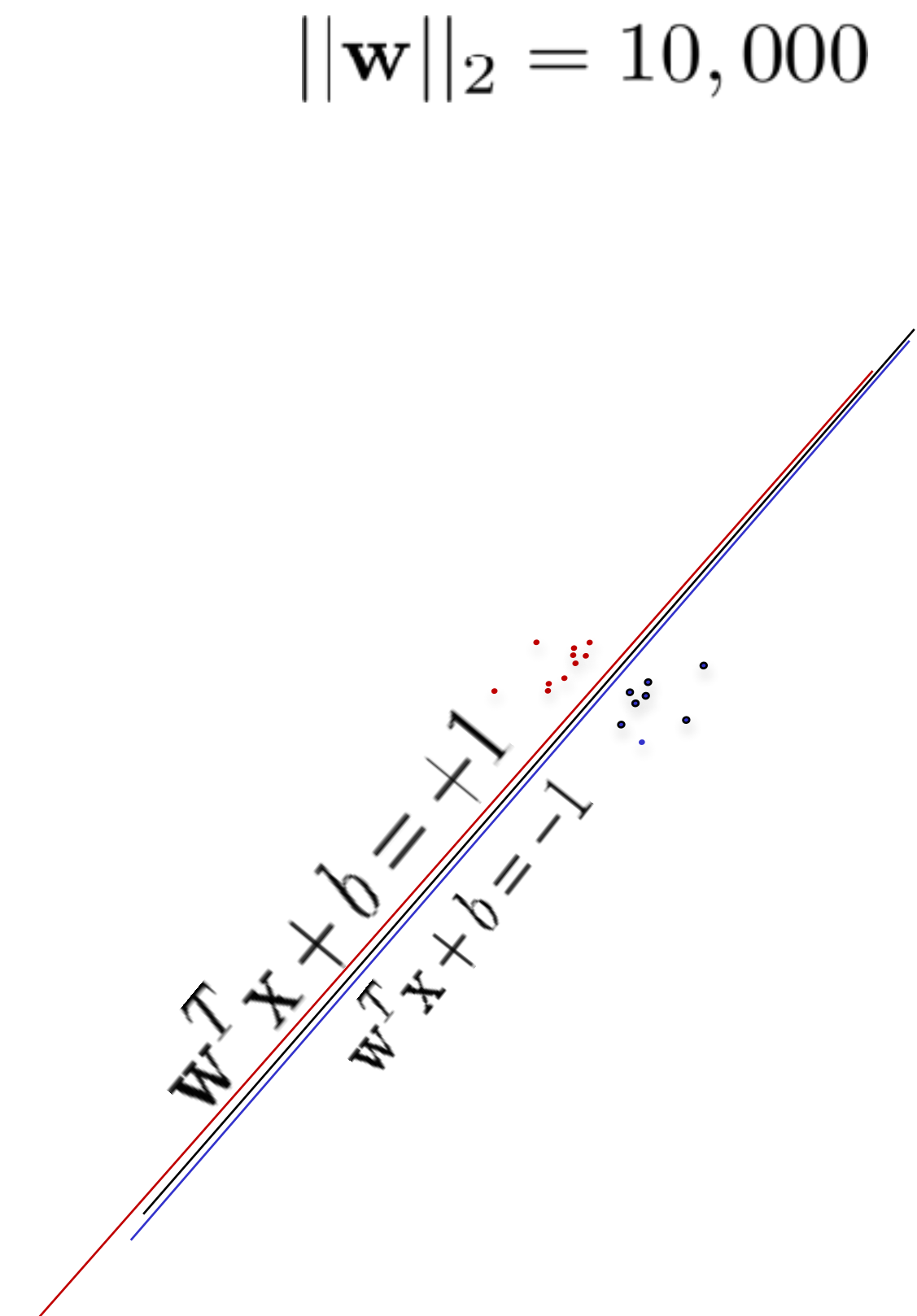
$$M = \frac{2}{\|\mathbf{w}\|}$$



# Large margin, small $w$



# Small margin, large $w$



SVM are linear models

A linear model's decision function:

$$\begin{aligned} f(\mathbf{x}; \mathbf{w}, b) &= \langle \mathbf{w}, \mathbf{x} \rangle + b \\ &= \mathbf{w} \cdot \mathbf{x} + b \\ &= \mathbf{w}^T \mathbf{x} + b \end{aligned}$$

$$\mathbf{x} \in \mathbb{R}^m \qquad \mathbf{w} \in \mathbb{R}^m \qquad b \in \mathbb{R}$$

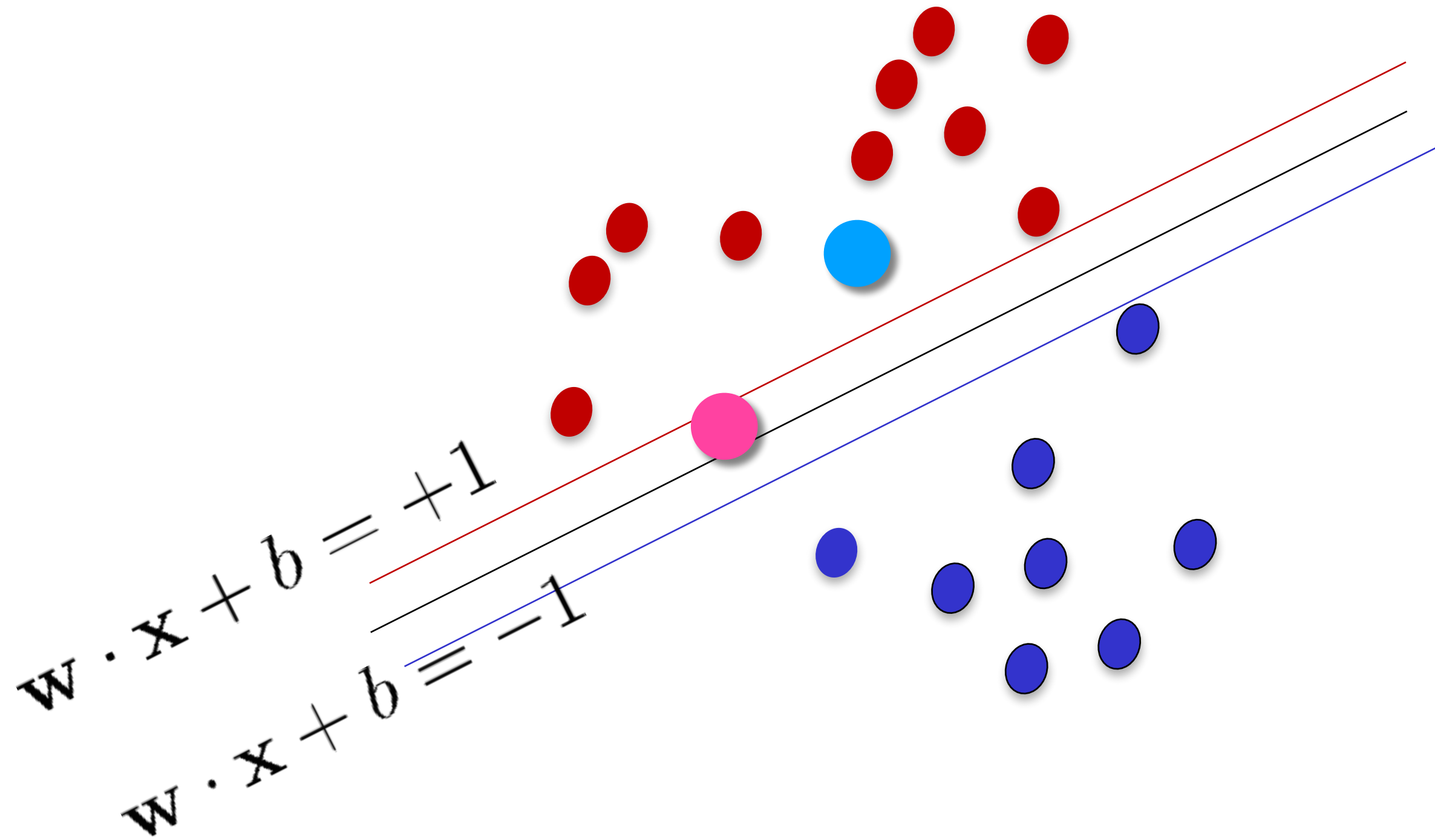
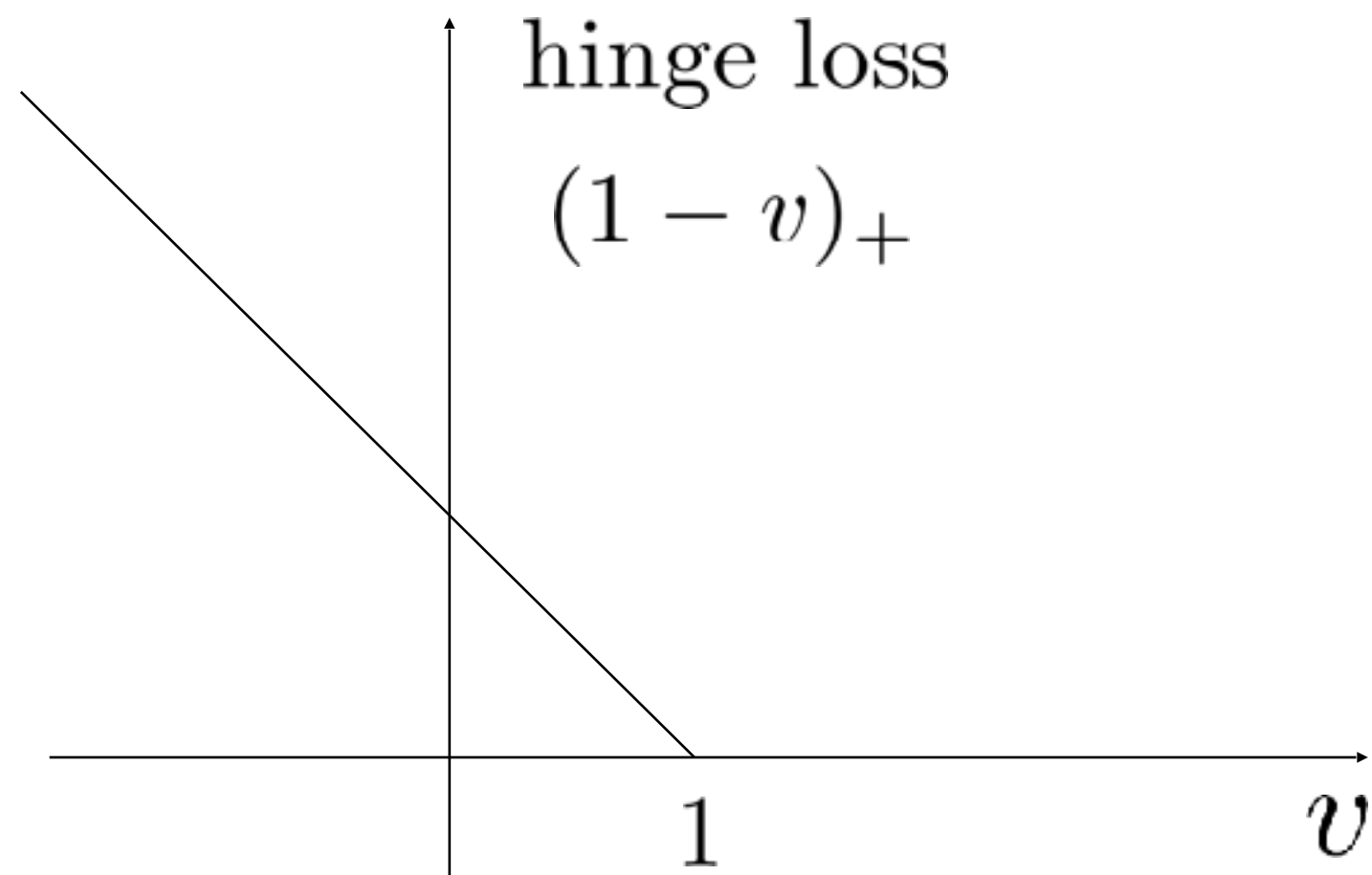
This is a linear function and our job is find the optimal  $\mathbf{w}$  and  $b$  to best fit the prediction in learning.

# Hinge Loss

Find:  $\arg \min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2$

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

Hinge:  $(1 - v)_+ = \max(0, 1 - v)$



Find:  $\arg \min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 + C \times \sum_{i=1}^n (1 - y_i \times (\mathbf{w}^T \mathbf{x}_i + b))_+$

Why

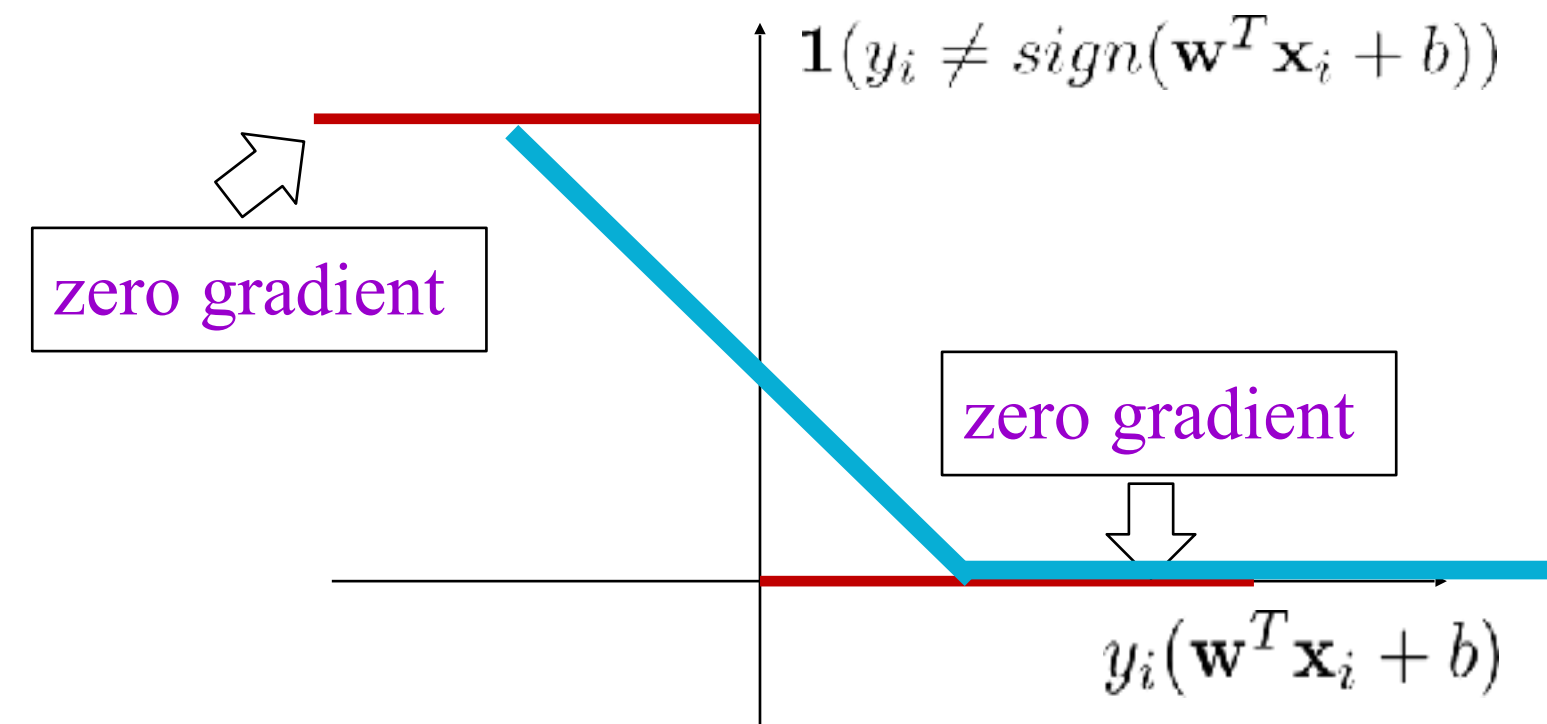
subject to  $y_i(\mathbf{w}^T \mathbf{x} + b) - 1 \geq 0$

Why not 0/1 loss?

## Standard loss (error) function

Standard 0/1 loss (gradient 0 nearly everywhere,  
**no gradient feedback**):

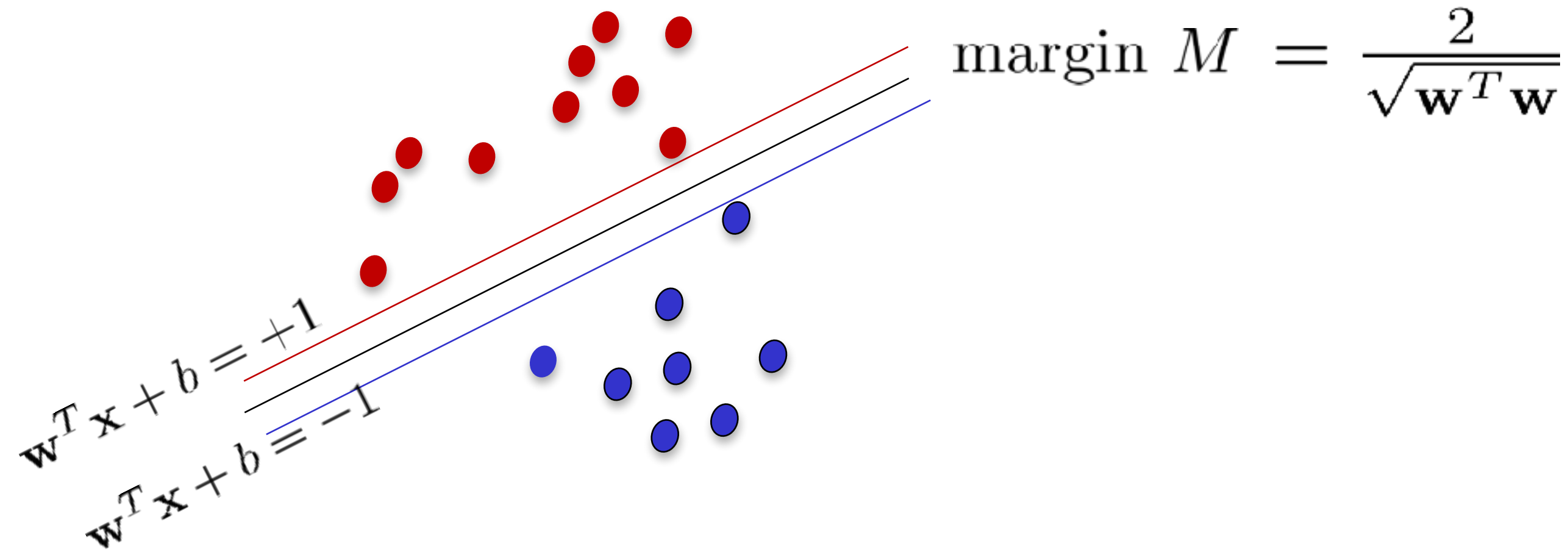
**Training:** Minimize  $\mathcal{L}(\mathbf{w}, b) = \sum_i \mathbf{1}(y_i \neq \text{sign}(\mathbf{w}^T \mathbf{x}_i + b))$



Hinge loss approximates 0/1

# Training SVM using gradient descent

---



Maximizing  $\frac{2}{\sqrt{\mathbf{w}^T \mathbf{w}}}$  is equivalent to minimizing  $\mathbf{w}^T \mathbf{w} = \|\mathbf{w}\|^2$

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$$

Find:  $\arg \min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2$  subject to  $y_i(\mathbf{w}^T \mathbf{x} + b) - 1 \geq 0$

$$\text{Minimize } \mathcal{L}(\mathbf{w}, b) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n (1 - y_i(\mathbf{w}^T \mathbf{x}_i + b))_+$$

# SVM gradient descent

---

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

$$\frac{\mathcal{L}(\mathbf{w}, b)}{\partial \mathbf{w}} = \mathbf{w} + C \sum_{i=1}^n \begin{cases} 0 & \text{if } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \\ -y_i \mathbf{x}_i & \text{otherwise} \end{cases}$$

$$\frac{\mathcal{L}(\mathbf{w}, b)}{\partial b} = C \sum_{i=1}^n \begin{cases} 0 & \text{if } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \\ -y_i & \text{otherwise} \end{cases}$$

# Dual form

Find:  $\arg \min_{\mathbf{w}} \frac{1}{2} ||\mathbf{w}'||^2$     subject to  $y_i(\mathbf{w}^T \mathbf{x} + b) - 1 \geq 0$



For the case of only one constraint and only two choice variables (as exemplified in Figure 1), consider the [optimization problem](#)

$$\begin{aligned} &\text{maximize } f(x, y) \\ &\text{subject to: } g(x, y) = 0 \end{aligned}$$

(Sometimes an additive constant is shown separately rather than being included in  $g$ , in which case the constraint is written  $g(x, y) = c$ , as in Figure 1.) We assume that both  $f$  and  $g$  have continuous first [partial derivatives](#). We introduce a new variable ( $\lambda$ ) called a **Lagrange multiplier** (or **Lagrange undetermined multiplier**) and study the **Lagrange function** (or **Lagrangian** or **Lagrangian expression**) defined by

$$\mathcal{L}(x, y, \lambda) = f(x, y) - \lambda g(x, y),$$

where the  $\lambda$  term may be either added or subtracted. If  $f(x_0, y_0)$  is a maximum of  $f(x, y)$  for the original constrained problem and  $\nabla g(x_0, y_0) \neq 0$ , then there exists  $\lambda_0$  such that  $(x_0, y_0, \lambda_0)$  is a [stationary point](#) for the Lagrange function (stationary points are those points where the first partial derivatives of  $\mathcal{L}$  are zero). The assumption  $\nabla g \neq 0$  is called constraint qualification. However, not all stationary points yield a solution of the original problem, as the method of Lagrange multipliers yields only a [necessary condition](#) for optimality in constrained problems.<sup>[9][10][11][12][13]</sup> Sufficient conditions for a minimum or maximum [also exist](#), but if a particular [candidate solution](#) satisfies the sufficient conditions, it is only guaranteed that that solution is the best one *locally* – that is, it is better than any permissible nearby points. The *global* optimum can be found by comparing the values of the original objective function at the points satisfying the necessary and locally sufficient conditions.

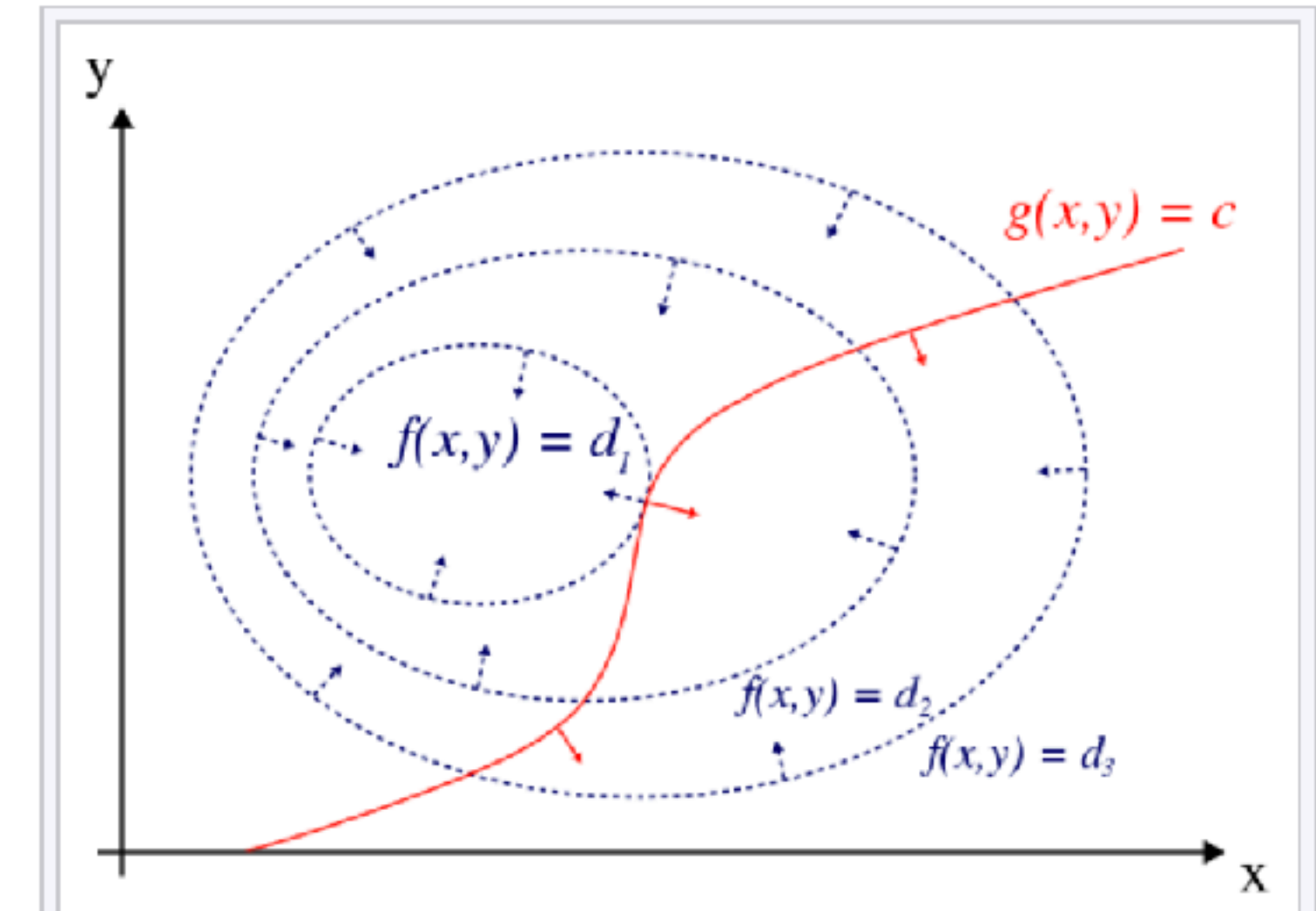


Figure 1: The red curve shows the constraint  $g(x, y) = c$ . The blue curves are contours of  $f(x, y)$ . The point where the red constraint tangentially touches a blue contour is the maximum of  $f(x, y)$  along the constraint, since  $d_1 > d_2$ .

# Sketch derivation of dual form

---

The **Representer Theorem** states that the solution  $\mathbf{w}$  can always be written as a linear combination of the training data:

$$\mathbf{w} = \sum_{j=1}^N \alpha_j y_j \mathbf{x}_j$$

Proof: **see example sheet** .

Now, substitute for  $\mathbf{w}$  in  $f(x) = \mathbf{w}^\top \mathbf{x} + b$

$$f(x) = \left( \sum_{j=1}^N \alpha_j y_j \mathbf{x}_j \right)^\top \mathbf{x} + b = \sum_{j=1}^N \alpha_j y_j (\mathbf{x}_j^\top \mathbf{x}) + b$$

and for  $\mathbf{w}$  in the cost function  $\min_{\mathbf{w}} \|\mathbf{w}\|^2$  subject to  $y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \forall i$

$$\|\mathbf{w}\|^2 = \left\{ \sum_j \alpha_j y_j \mathbf{x}_j \right\}^\top \left\{ \sum_k \alpha_k y_k \mathbf{x}_k \right\} = \sum_{jk} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j^\top \mathbf{x}_k)$$

Hence, an equivalent optimization problem is over  $\alpha_j$

$$\min_{\alpha_j} \sum_{jk} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j^\top \mathbf{x}_k) \quad \text{subject to} \quad y_i \left( \sum_{j=1}^N \alpha_j y_j (\mathbf{x}_j^\top \mathbf{x}_i) + b \right) \geq 1, \forall i$$

and a few more steps are required to complete the derivation.

# Primal and dual formulations

---

$N$  is number of training points, and  $d$  is dimension of feature vector  $\mathbf{x}$ .

Primal problem: for  $\mathbf{w} \in \mathbb{R}^d$

$$\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{w}\|^2 + C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i))$$

Dual problem: for  $\alpha \in \mathbb{R}^N$  (stated without proof):

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j^\top \mathbf{x}_k) \text{ subject to } 0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$

- Need to learn  $d$  parameters for primal, and  $N$  for dual
- If  $N \ll d$  then more efficient to solve for  $\alpha$  than  $\mathbf{w}$
- Dual form only involves  $(\mathbf{x}_j^\top \mathbf{x}_k)$ . We will return to why this is an advantage when we look at kernels.



## Primal and dual formulations

---

Primal version of classifier:

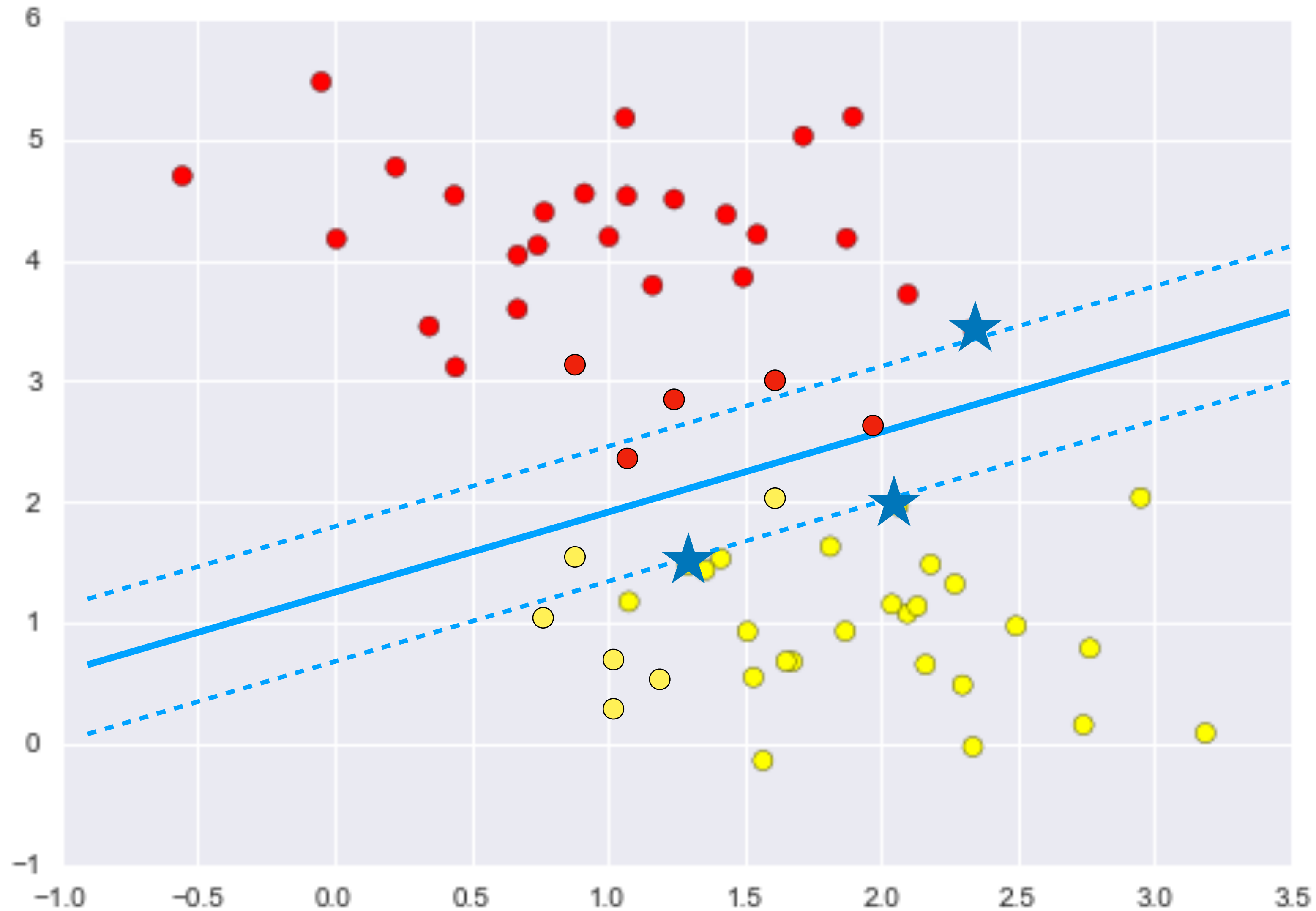
$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

Dual version of classifier:

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i (\mathbf{x}_i^\top \mathbf{x}) + b$$

At first sight the dual form appears to have the disadvantage of a K-NN classifier – it requires the training data points  $\mathbf{x}_i$ . However, many of the  $\alpha_i$ 's are zero. The ones that are non-zero define the support vectors  $\mathbf{x}_i$ .

Robust to dataset noise, small number of sv to optimize



$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j^\top \mathbf{x}_k) \quad \text{subject to } 0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$

**Want to learn another perspective on the derivation of SVM,  
especially the dual formulation using Lagrangian multipliers?**

**Listen to a lecture by the late great Patrick Winston  
[https://youtu.be/\\_PwhiWxHK8o](https://youtu.be/_PwhiWxHK8o)**

## Nearest Neighbor Algorithm

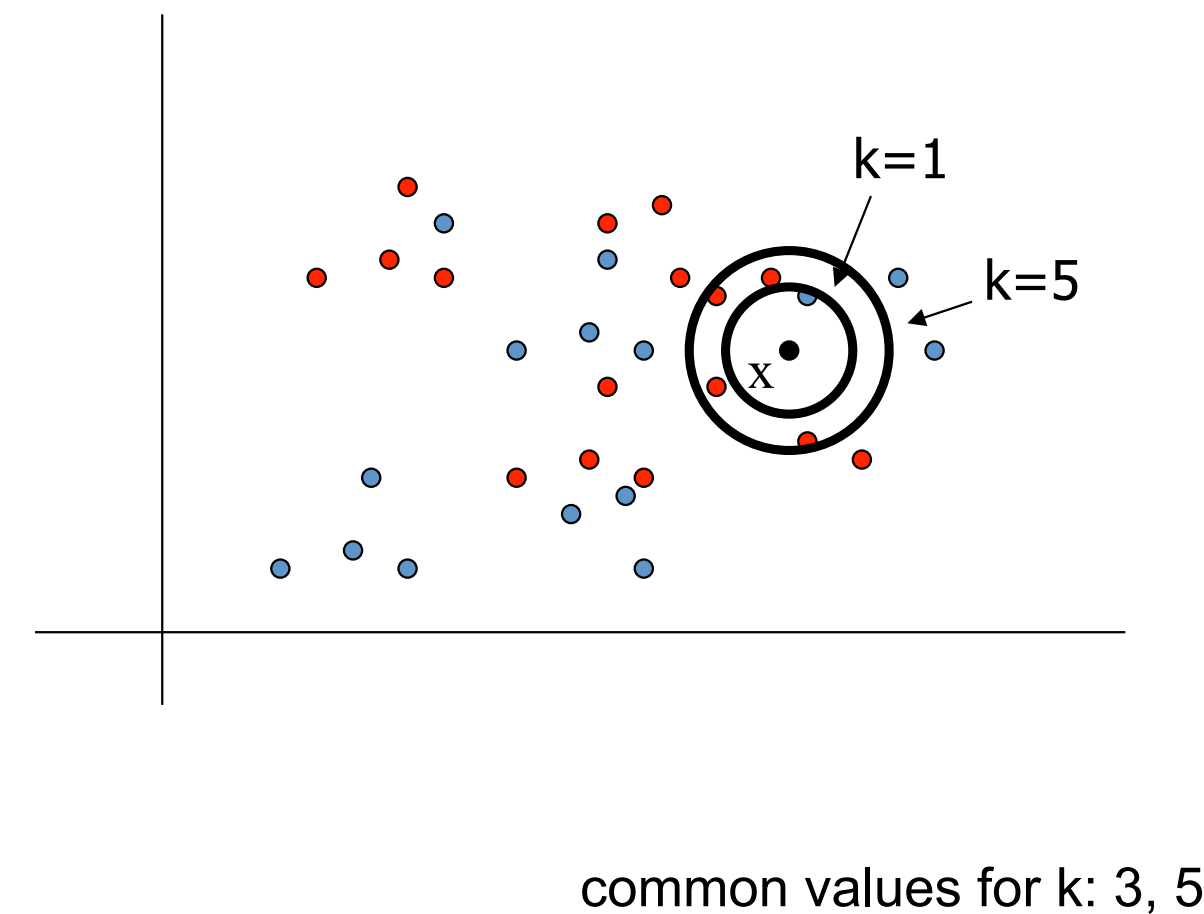
- Learning Algorithm:
  - Store training examples
- Prediction Algorithm:
  - To classify a new example  $\mathbf{x}$  by finding the training example  $(\mathbf{x}^i, y^i)$  that is *nearest* to  $\mathbf{x}$
  - Guess the class  $y = y^i$

Non parametric  
Data based!  
More prone to overfitting

Vs parametric  
Model - based!  
Prone to underfitting

## K-Nearest Neighbor Methods

- To classify a new input vector  $\mathbf{x}$ , examine the  $k$ -closest training data points to  $\mathbf{x}$  and assign the object to the most frequently occurring class

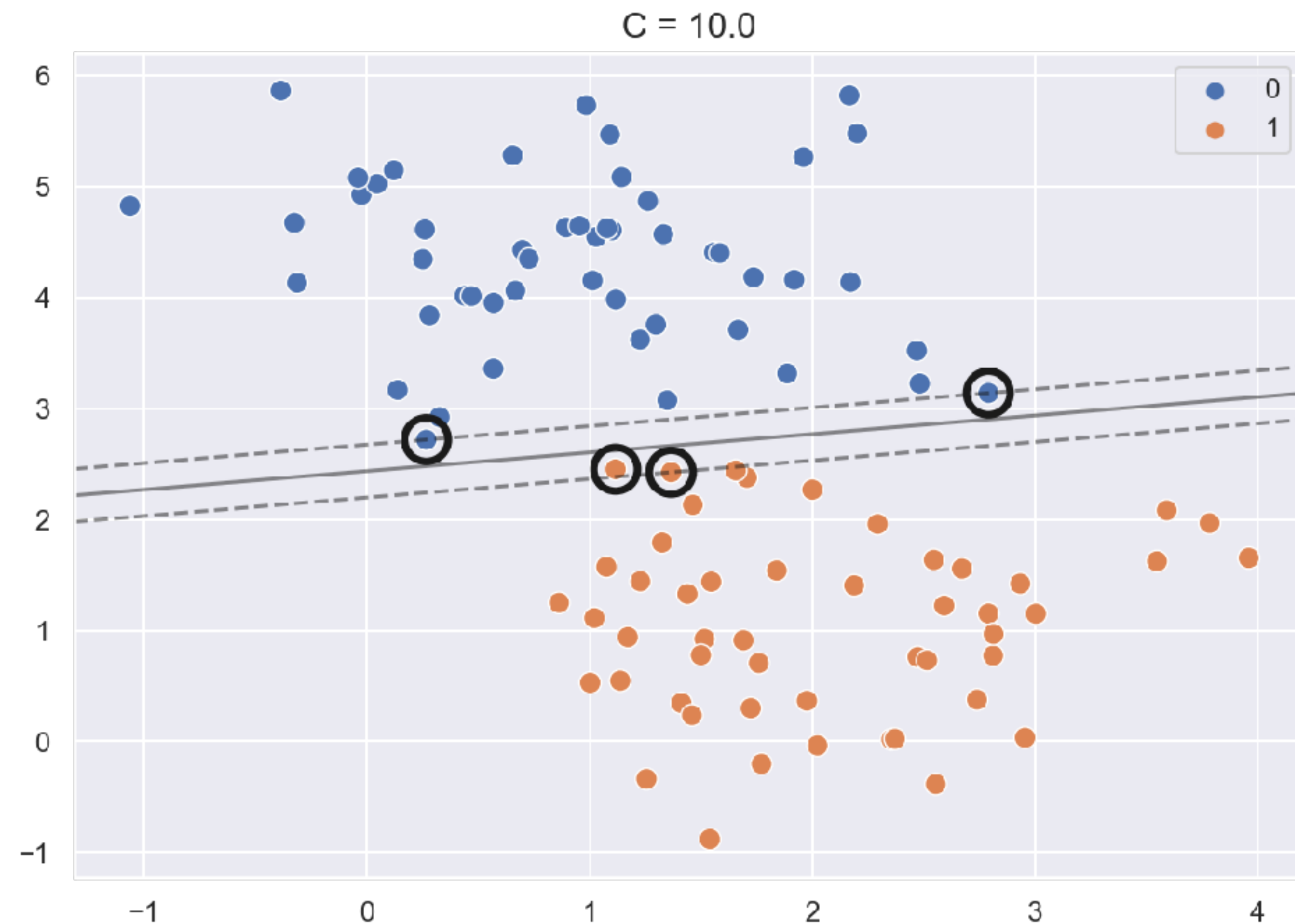


**Need to pick  $C$  to make the  
tradeoff of margin and hinge loss**



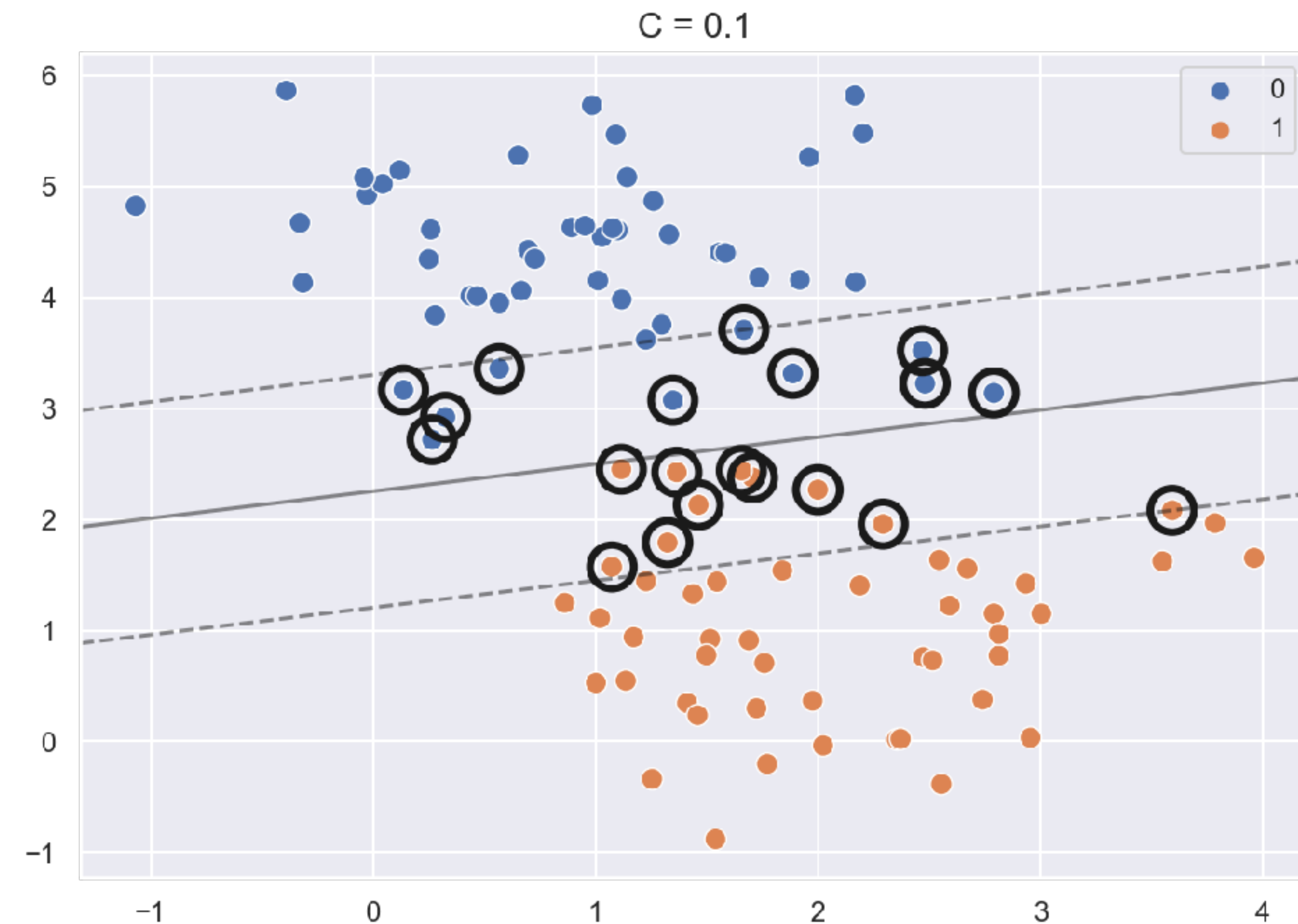
## Hard margin

e.g.,  $C = 10$  resulting in  $\|\mathbf{w}\| = 100$



## Soft margin

e.g.,  $C = 0.1$  resulting in  $\|\mathbf{w}\| = 0.01$



Find:  $\arg \min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 + C \times \sum_{i=1}^n (1 - y_i \times (\mathbf{w}^T \mathbf{x}_i + b))_+$

# SVM components

- Place a decision boundary (ie, optimize  $\mathbf{w}$ ) s.t.
  - the margin is as wide as possible
  - the fewest errors are made
  - $\mathbf{w}$  is a linear combination of the support vectors
    - support vectors are a few examples of each class from the training set
- But at decision time we are just doing the usual linear model thing

## What's the difference between logistic regression and linear SVM?

---

Logistic regression:

SVM:

- A. They are different in both training and testing.
- B. They differ in training but are the same in testing.
- C. They differ in testing but are the same in training.
- D. They are completely different.

# What's the difference between logistic regression and linear SVM?

Logistic regression:

$$p(y|\mathbf{x}) = \frac{1}{1+e^{-y(\mathbf{w}^T \mathbf{x} + b)}}$$

Training:  $\arg \min_{\mathbf{w}, b} \sum_{i=1}^n \ln(1 + e^{-y_i(\mathbf{w}^T \mathbf{x}_i + b)})$


Test:  $f(\mathbf{x}; \mathbf{w}, b) = \begin{cases} +1 & \text{if } \frac{1}{1+e^{-(\mathbf{w}^T \mathbf{x} + b)}} \geq 0.5 \\ -1 & \text{otherwise} \end{cases}$

Equivalent to:  $f(\mathbf{x}; \mathbf{w}, b) = \begin{cases} +1 & \text{if } \mathbf{w}^T \mathbf{x} + b \geq 0 \\ -1 & \text{otherwise} \end{cases}$

SVM:

Training:  $\arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n (1 - y_i(\mathbf{w}^T \mathbf{x}_i + b))_+$

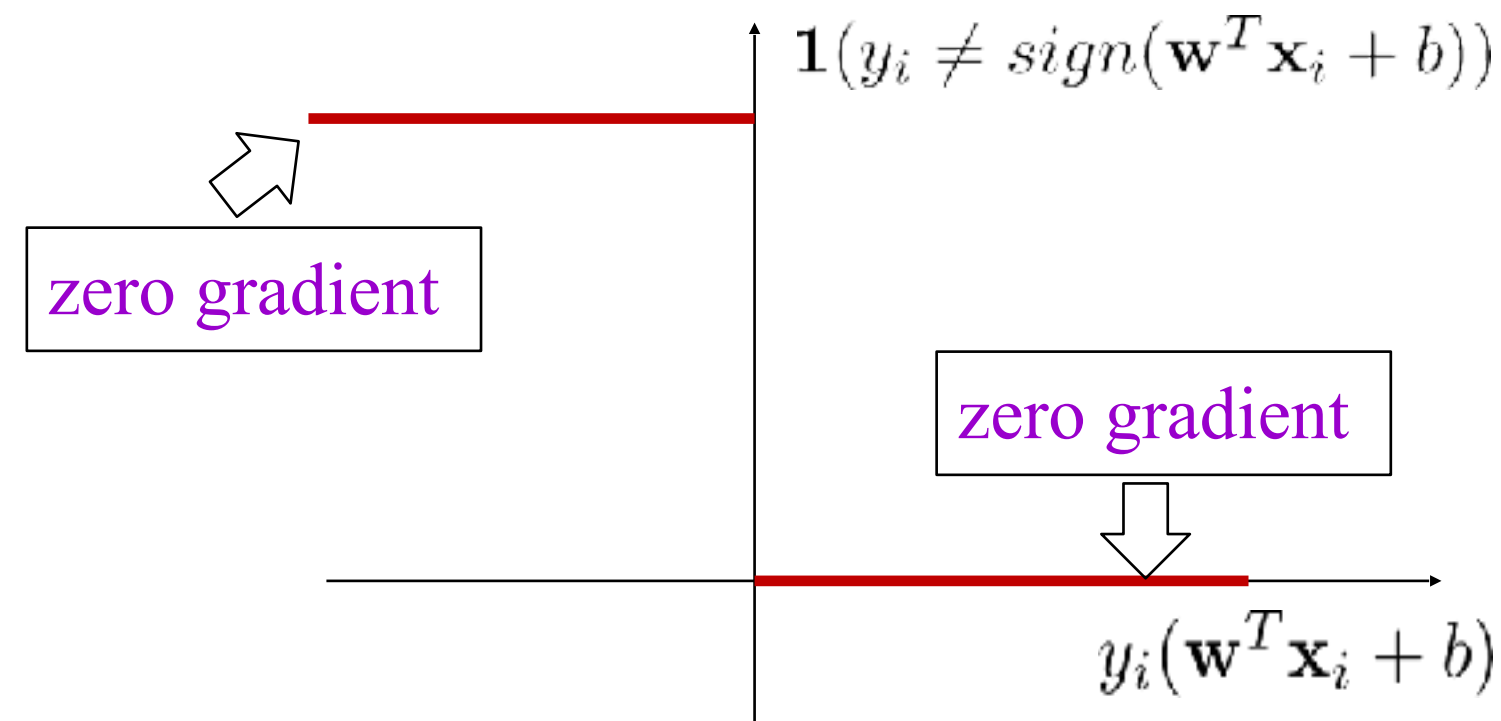
Test:  $f(\mathbf{x}; \mathbf{w}, b) = \begin{cases} +1 & \text{if } \mathbf{w}^T \mathbf{x} + b \geq 0 \\ -1 & \text{otherwise} \end{cases}$

- A. They are different in both training and testing.
-  B. They differ in training but are the same in testing.
- C. They differ in testing but are the same in training.
- D. They are completely different.

# Standard loss (error) function

Standard 0/1 loss (gradient 0 nearly everywhere,  
**no gradient feedback**):

**Training:** Minimize  $\mathcal{L}(\mathbf{w}, b) = \sum_i \mathbf{1}(y_i \neq \text{sign}(\mathbf{w}^T \mathbf{x}_i + b))$



Main motivation

**Hard**->Half-hard->Soft Error

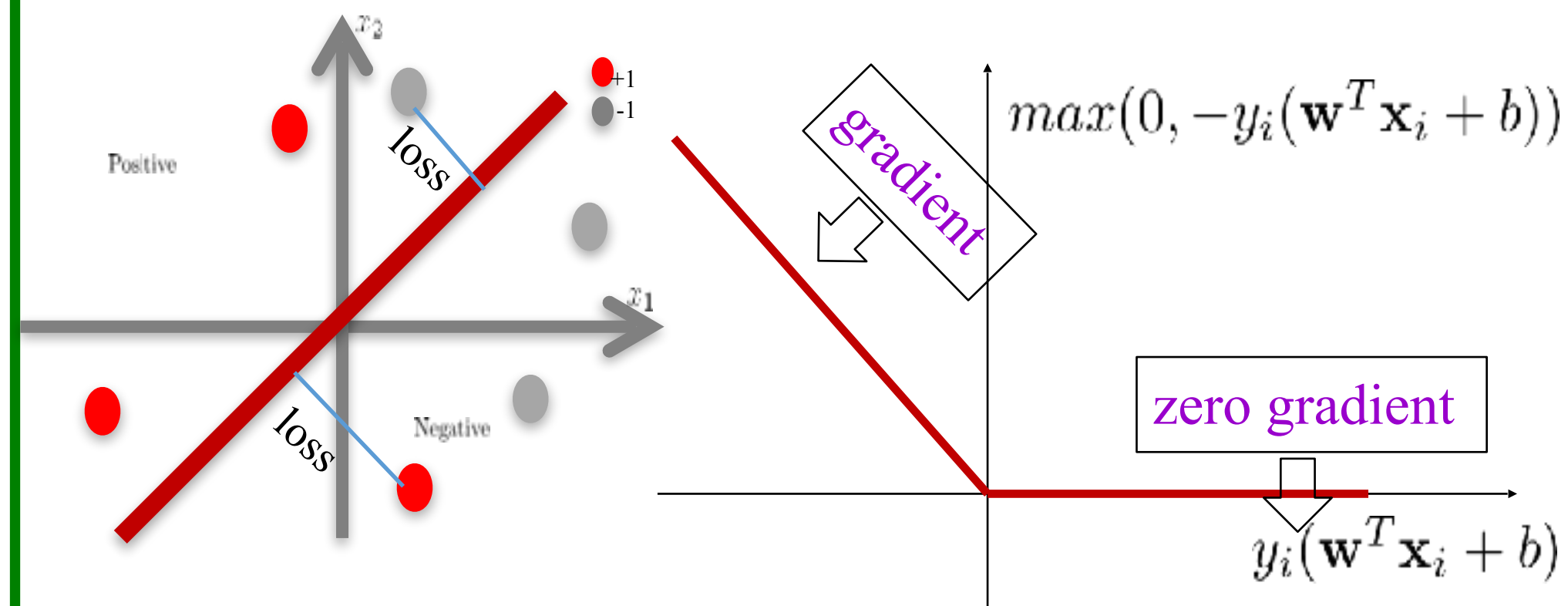
It is the most **direct** loss, but is also  
the **hardest** to minimize.

Zero gradient everywhere!

# Half-hard loss (error) function

Loss implicitly used in the perceptron algorithm: with **gradient feedback** when the target (ground-truth label) and the output (classification) are different).

**Training:** Minimize  $\mathcal{L}(\mathbf{w}, b) = \sum_i \max(0, -y_i(\mathbf{w}^T \mathbf{x}_i + b))$



Main motivation

Hard->**Half-hard**->Soft Error

**Zero loss** for correct classification (**no gradient**).

A loss based on the **distance** to the decision boundary for **misclassification** (**with gradient**).

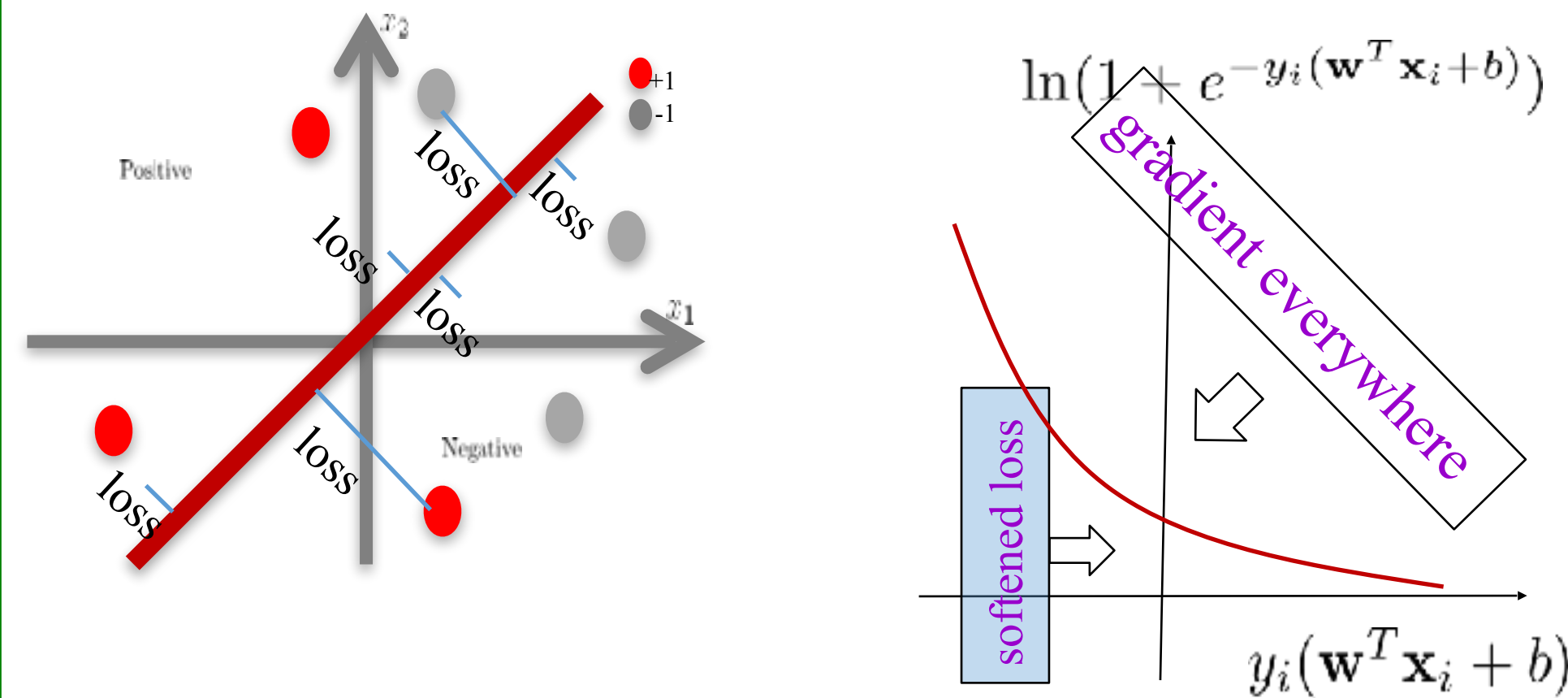
Used in the **perceptron** training.



# Soft loss (error) function

Loss used in logistic regression.

**Training:** minimize  $\mathcal{L}(\mathbf{w}, b) = \sum_{i=1}^n \ln(1 + e^{-y_i(\mathbf{w}^T \mathbf{x}_i + b)})$



Main motivation

Hard->Half-hard->**Soft** Error

**Every data point** receives a loss (gradient everywhere).

A loss based on the **distance to the decision boundary** for wrong classification (has a gradient).

Used in **logistic regression** classifier.

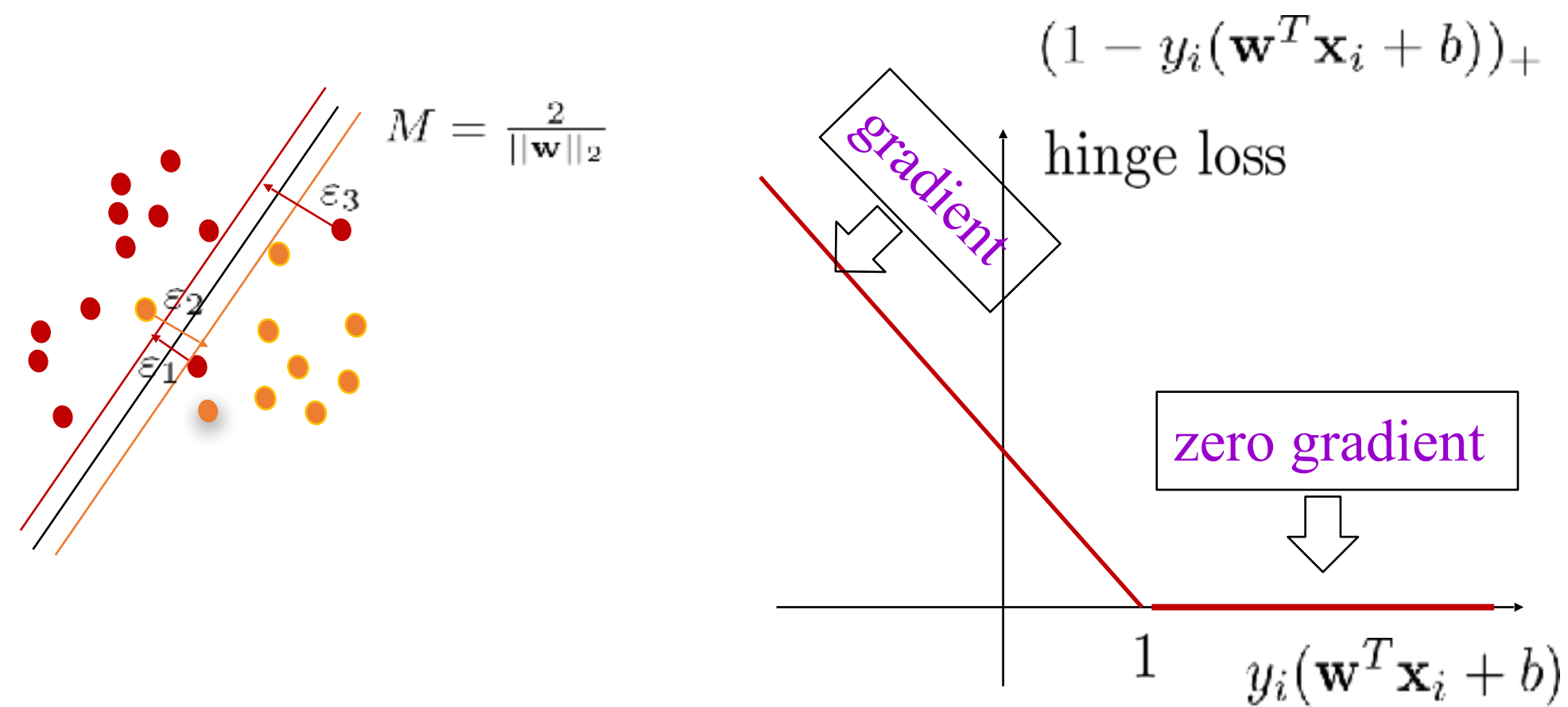


# Loss in SVM

Main motivation

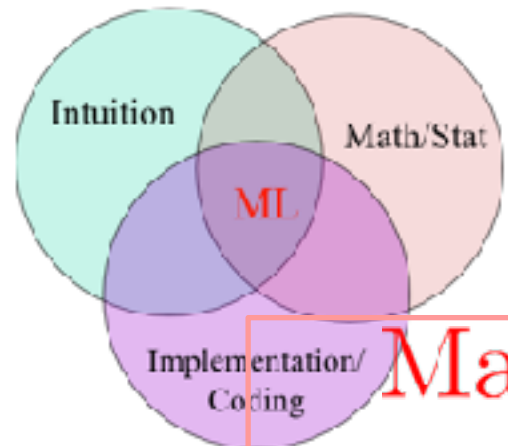
Hard->Half-hard->Soft Error

$$\text{Minimize } \mathcal{L}(\mathbf{w}, b) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n (1 - y_i(\mathbf{w}^T \mathbf{x}_i + b))_+$$



**Zero loss** for correct classification beyond the margin (**no gradient**).

A loss based on the **distance** to the decision boundary for **misclassification** or **within the margin** (**with gradient**).



# Recap: Support Vector Machine

**Math:**

*Training :*

$$\text{Minimize } \mathcal{L}(\mathbf{w}, b) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n (1 - y_i(\mathbf{w}^T \mathbf{x}_i + b))_+$$

$$\frac{\mathcal{L}(\mathbf{w}, b)}{\partial \mathbf{w}} = \mathbf{w} + C \sum_{i=1}^n \begin{cases} 0 & \text{if } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \\ -y_i \mathbf{x}_i & \text{otherwise} \end{cases}$$

$$\frac{\mathcal{L}(\mathbf{w}, b)}{\partial b} = C \sum_{i=1}^n \begin{cases} 0 & \text{if } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \\ -y_i & \text{otherwise} \end{cases}$$

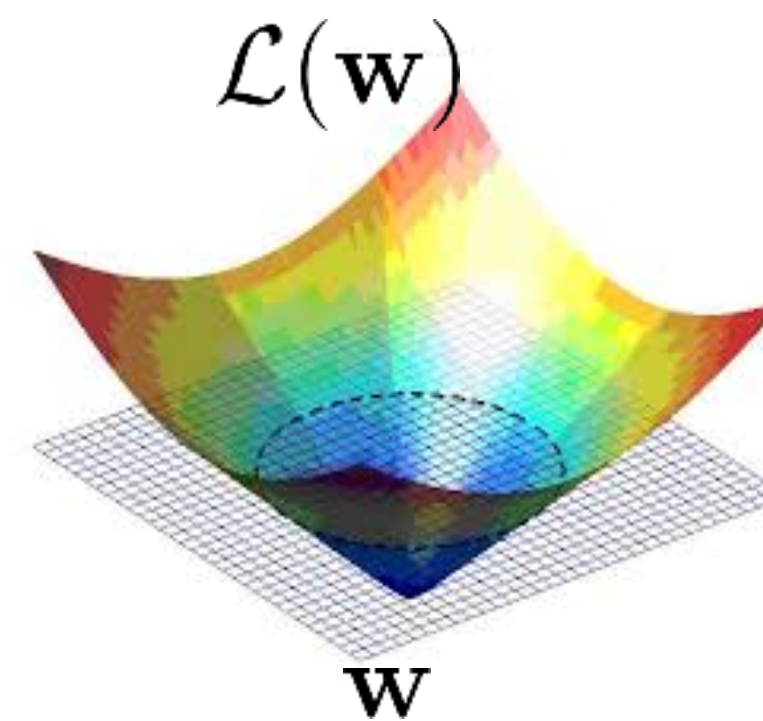
*Testing :*

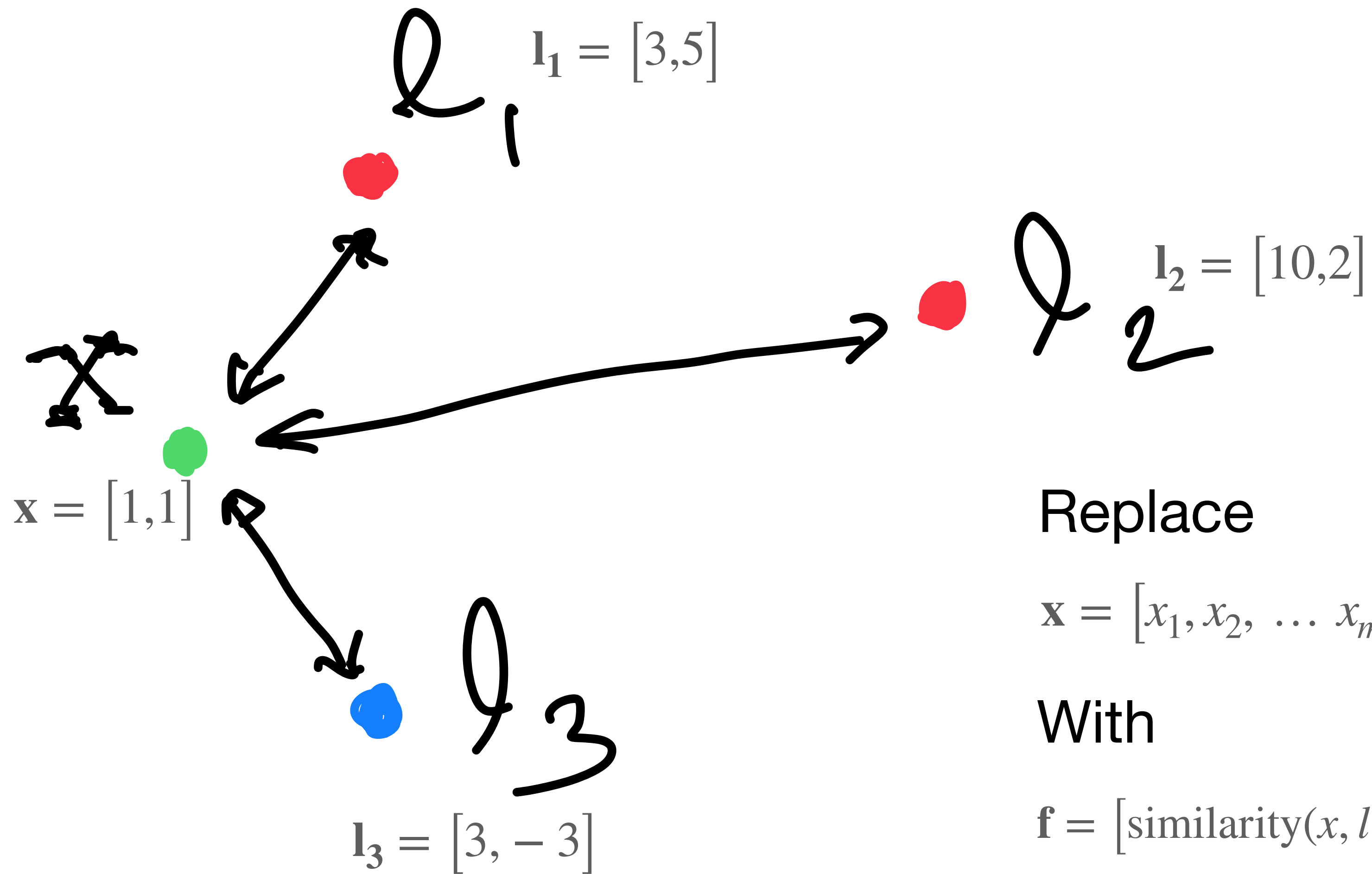
$$f(\mathbf{x}; \mathbf{w}, b) = \begin{cases} +1 & \text{if } \mathbf{w}^T \mathbf{x} + b \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

**Implementation:**

Gradient Descent Direction

- Pick a direction  $\nabla \mathcal{L}(\mathbf{w}_t, b_t)$
- Pick a step size  $\lambda_t$
- $\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda_t \times \nabla \mathcal{L}_{\mathbf{w}_t}(\mathbf{w}_t, b_t)$  such that function decreases;  
 $b_{t+1} = b_t - \lambda_t \times \nabla \mathcal{L}_{b_t}(\mathbf{w}_t, b_t)$
- Repeat



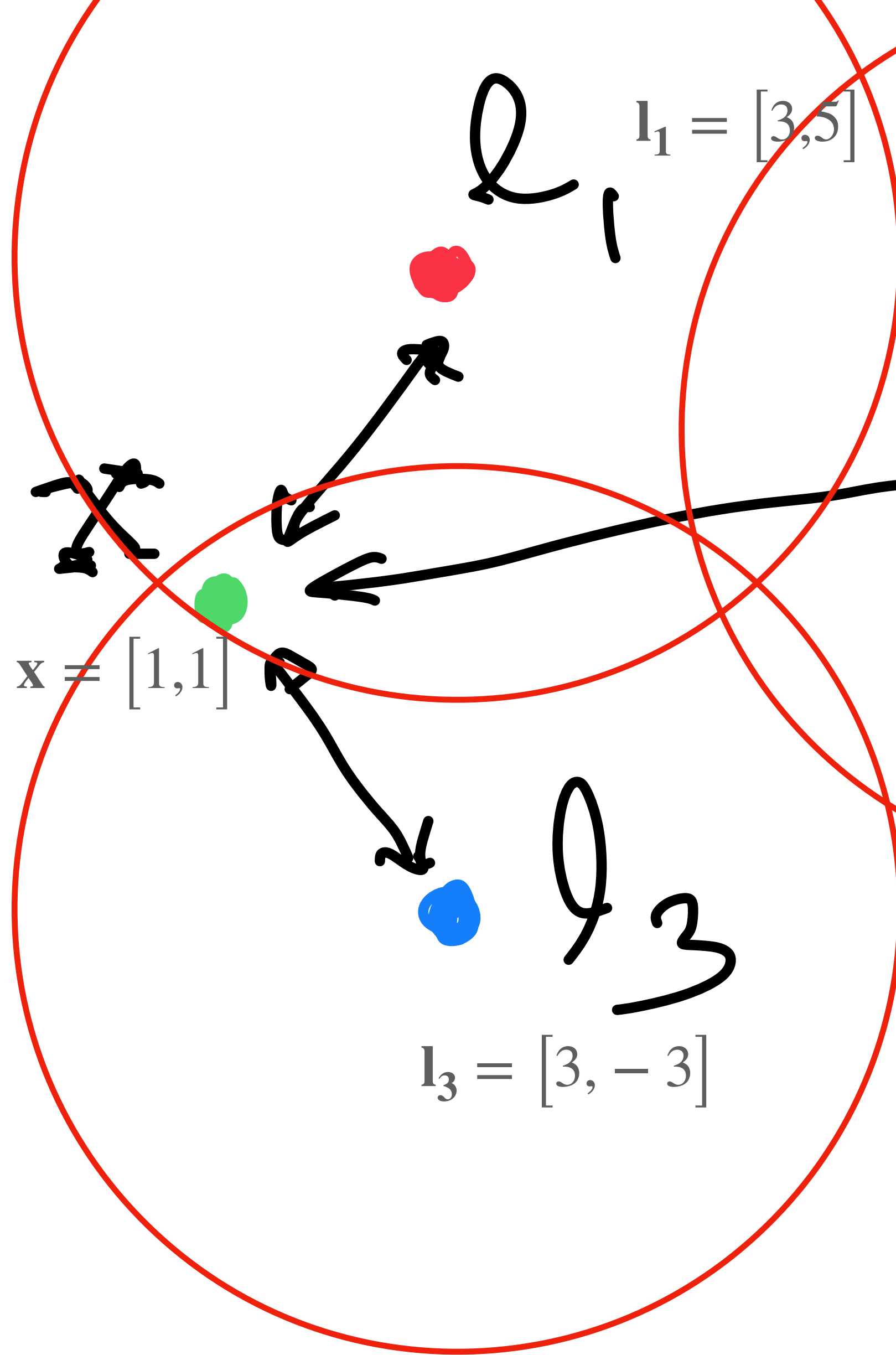


Replace

$$\mathbf{x} = [x_1, x_2, \dots, x_m]^T$$

With

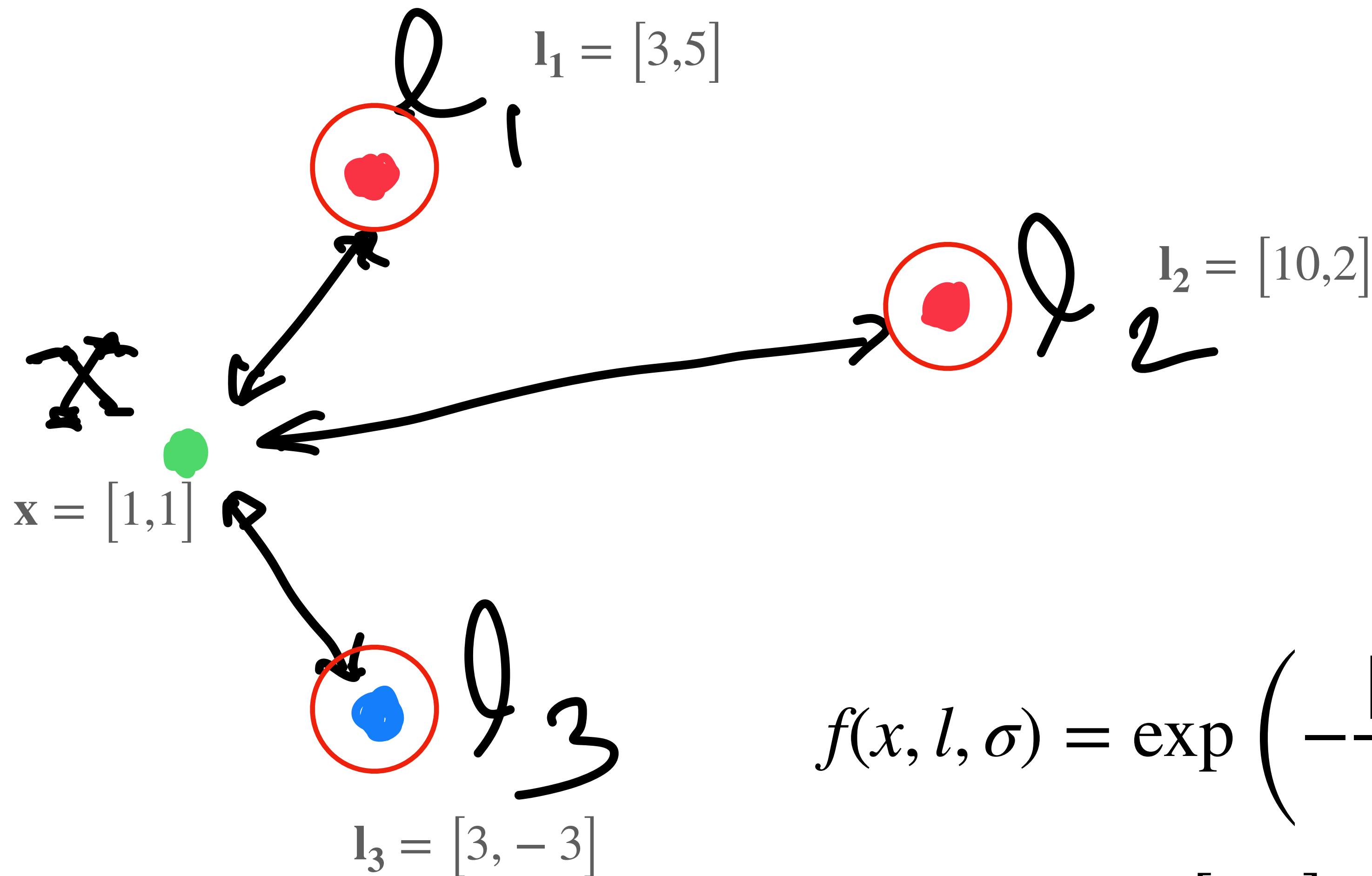
$$\mathbf{f} = [\text{similarity}(x, l_1), \text{similarity}(x, l_2), \dots, \text{similarity}(x, l_n)]$$



$$f(x, l, \sigma) = \exp \left( -\frac{\|x - l\|}{\sigma^2} \right)$$

Replace  $\mathbf{x} = [1, 1]$  with

$$\begin{aligned} & [f(\mathbf{x}, l_1, \sigma = 10), f(\mathbf{x}, l_2, \sigma = 10), f(\mathbf{x}, l_3, \sigma = 10)] \\ & = [0.639, 0.404, 0.639] \end{aligned}$$

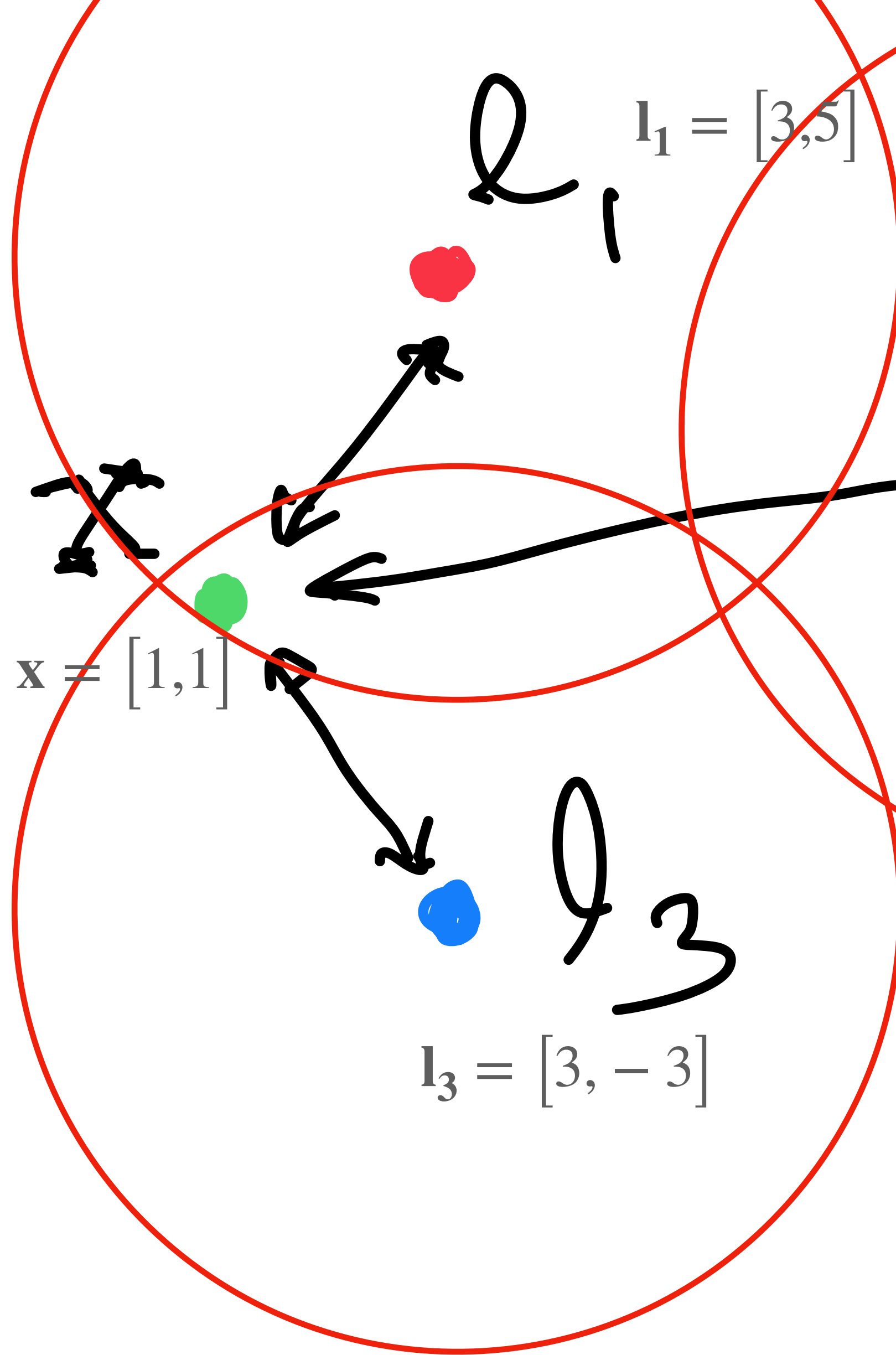


$$f(x, l, \sigma) = \exp \left( -\frac{\|x - l\|}{\sigma^2} \right)$$

Replace  $\mathbf{x} = [1, 1]$  with

$$[f(\mathbf{x}, l_1, \sigma = 1), f(\mathbf{x}, l_2, \sigma = 1), f(\mathbf{x}, l_3, \sigma = 1)]$$

$$= [0.011, 1e^{-4}, 0.011]$$



$$f_i = \exp \left( -\frac{\|x - l_i\|}{\sigma^2} \right) f(x) = [0.639, 0.404, 0.639]$$

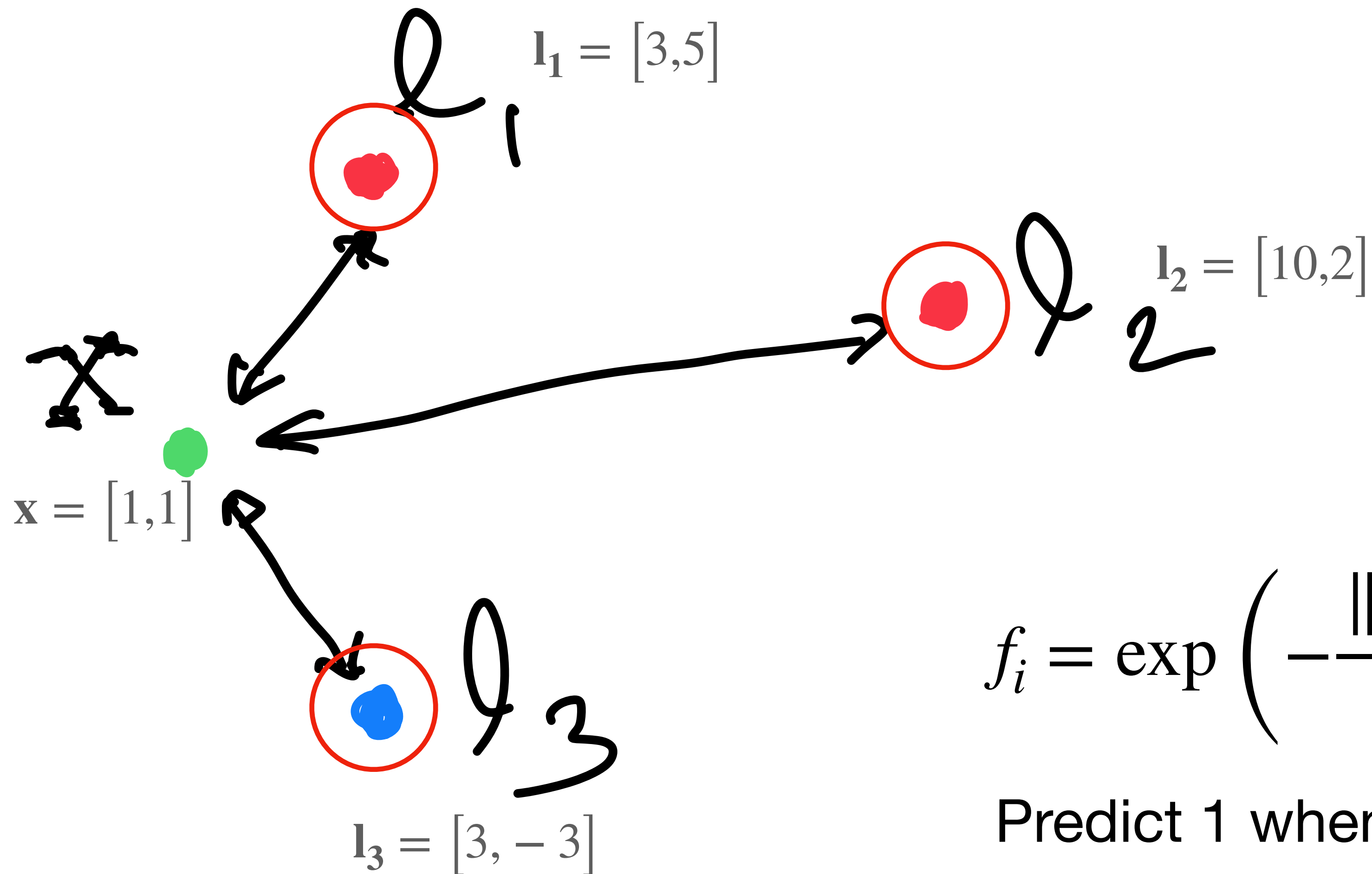
Predict 1 when

$$\mathbf{w}_0 + \mathbf{w}_1 f_1 + \mathbf{w}_2 f_2 + \mathbf{w}_3 f_3 \geq 0$$

And let's say that

$$\mathbf{w}_0 = -0.1, \mathbf{w}_1 = \mathbf{w}_2 = 1, \mathbf{w}_3 = -1, \sigma = 10$$





$$f_i = \exp \left( -\frac{\|\mathbf{x} - l_i\|}{\sigma^2} \right) \quad f(\mathbf{x}) = [0.011, 1e^{-4}, 0.011]$$

Predict 1 when

$$\mathbf{w}_0 + \mathbf{w}_1 f_1 + \mathbf{w}_2 f_2 + \mathbf{w}_3 f_3 \geq 0$$

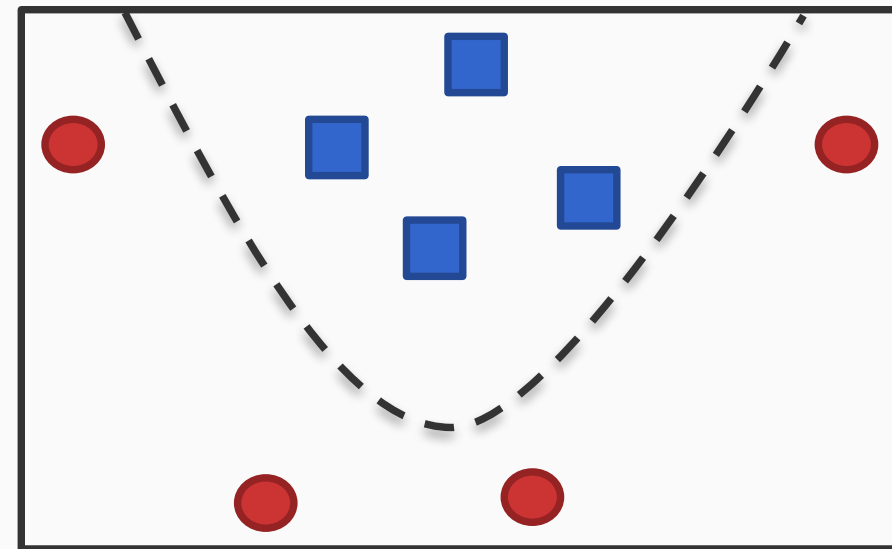
And let's say that

$$\mathbf{w}_0 = -0.1, \mathbf{w}_1 = \mathbf{w}_2 = 1, \mathbf{w}_3 = -1, \sigma = 10$$

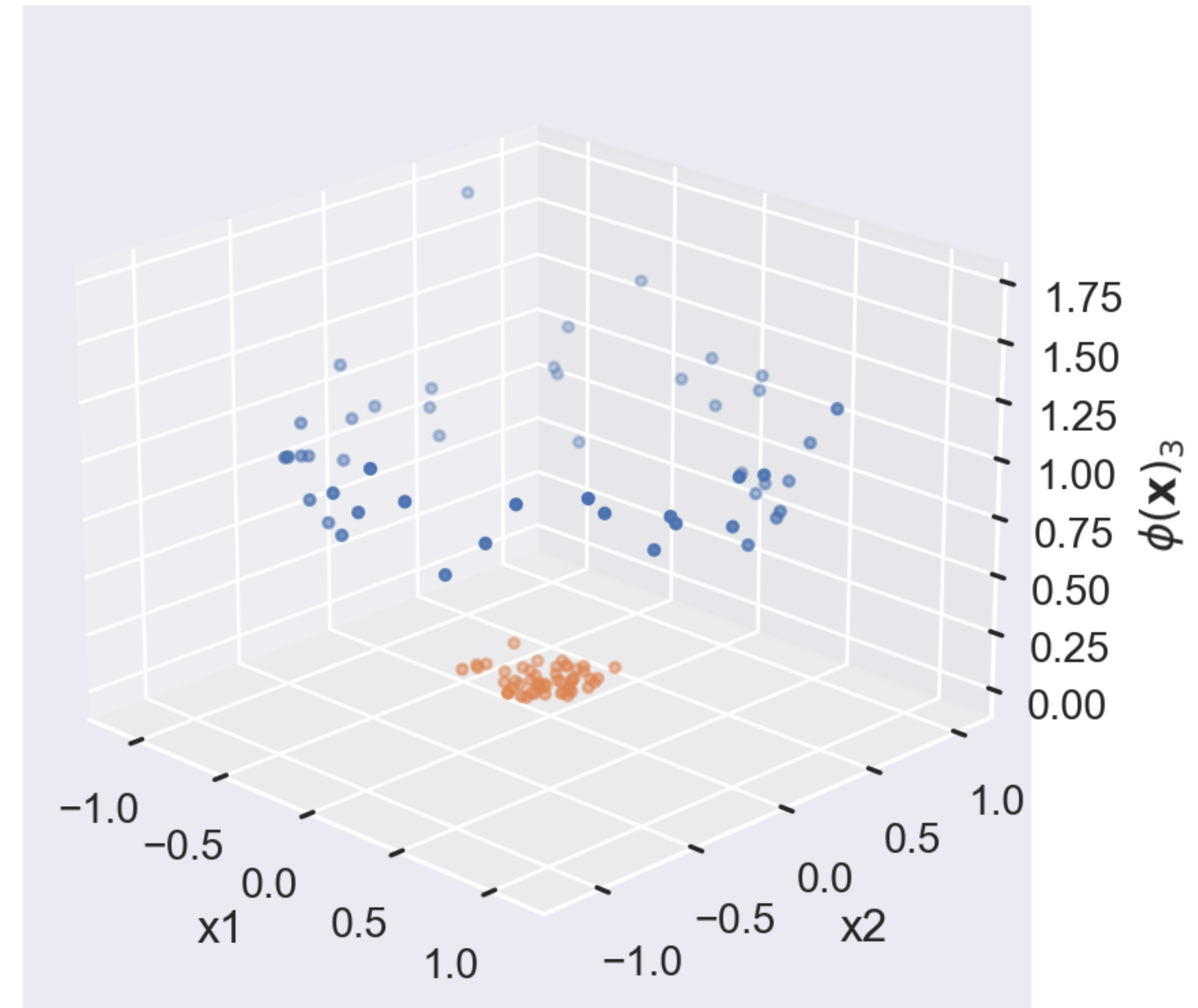
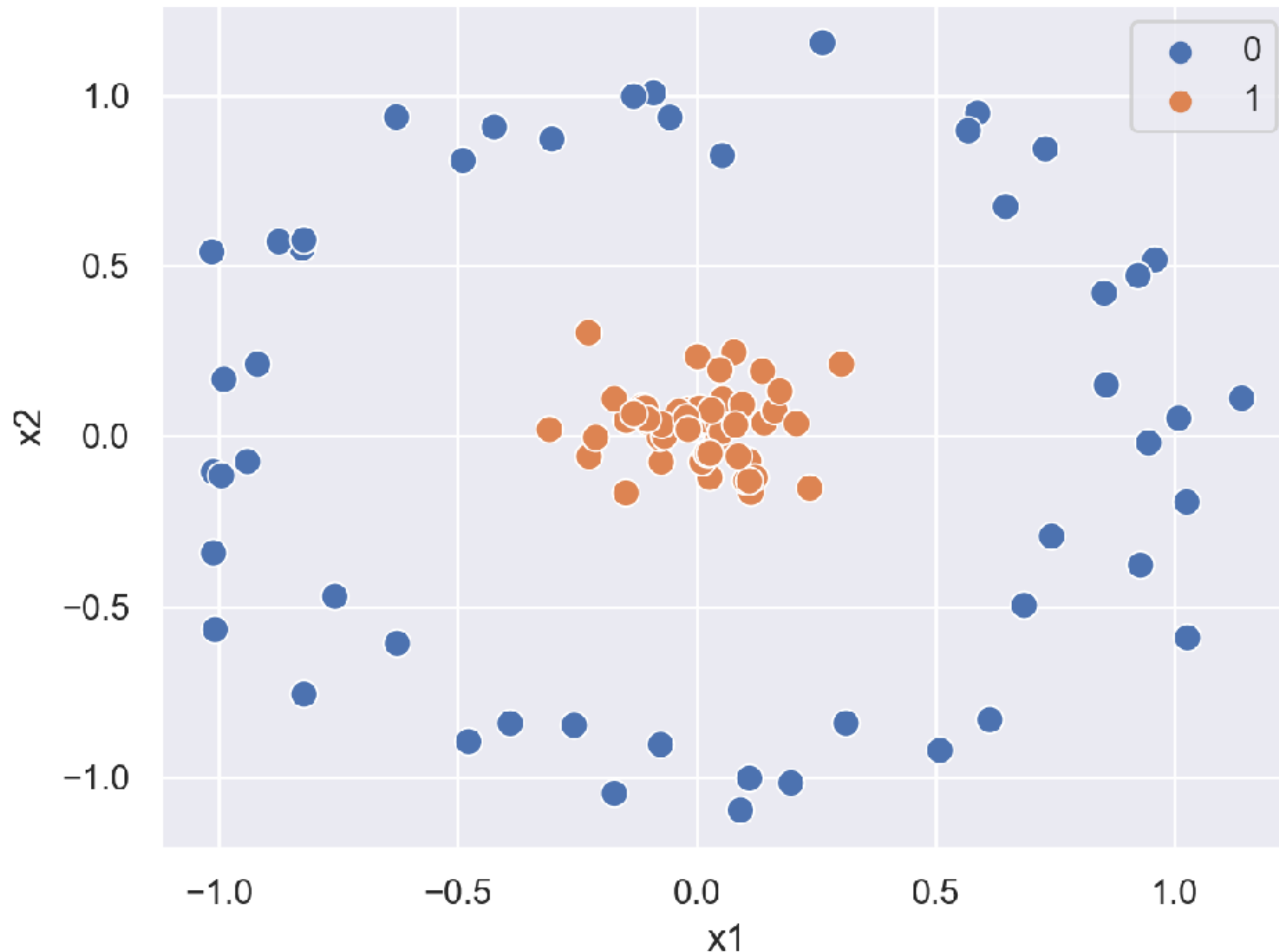
# One way to learn non-linear models

Explicitly introduce non-linearity into the feature space

If the true separator is quadratic



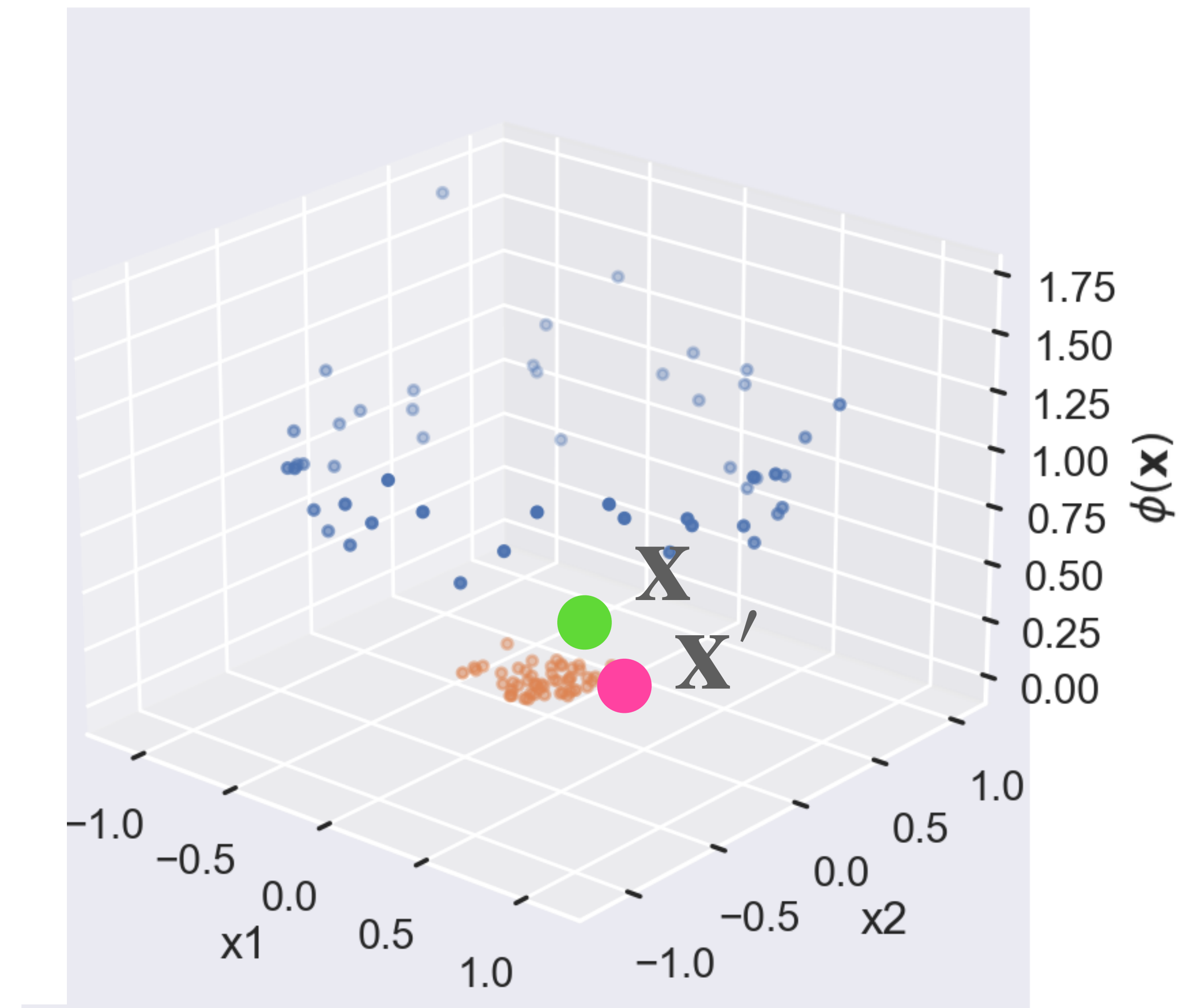
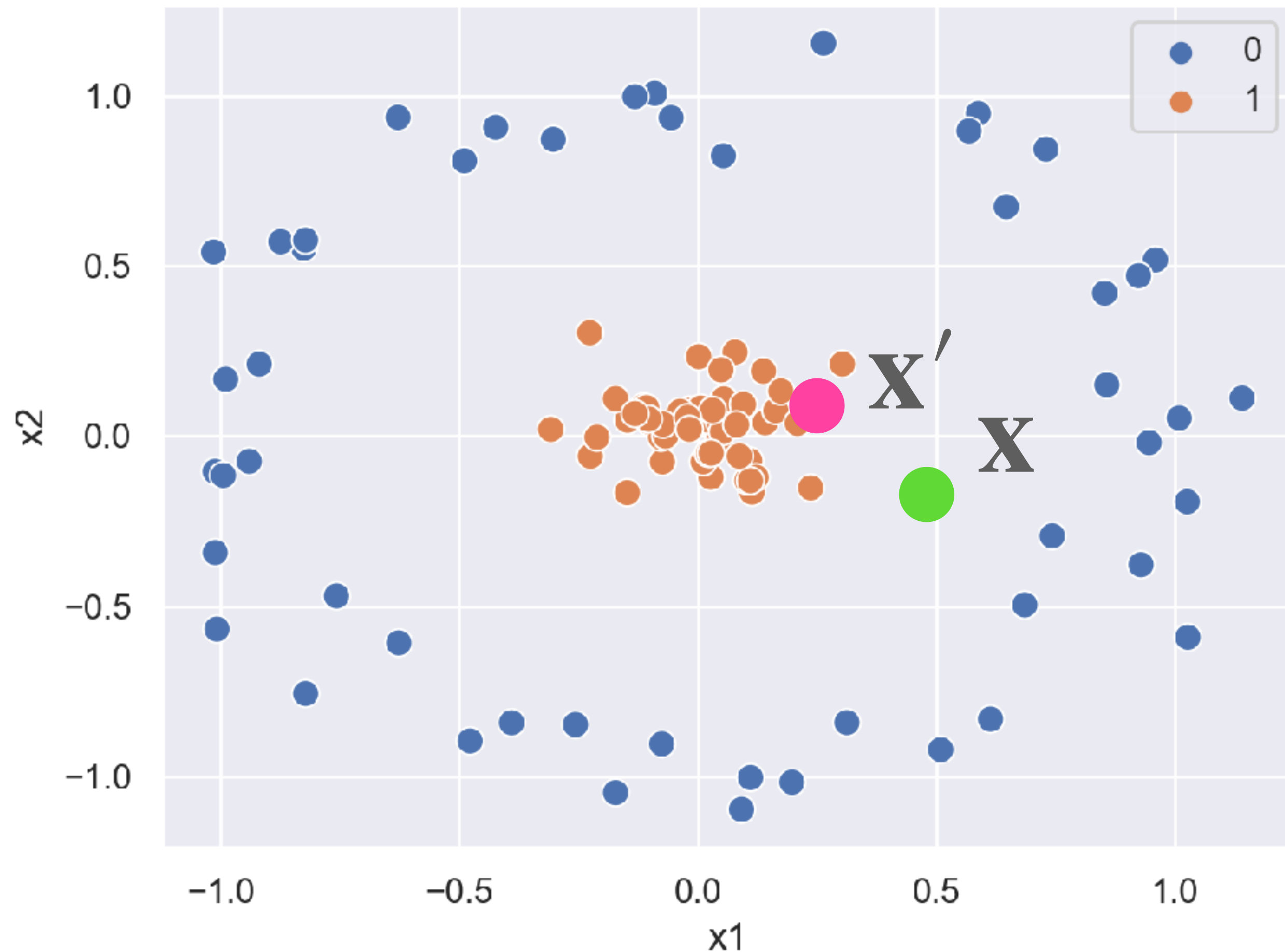
$$\phi(\mathbf{x}) = [x_1, x_2, x_1 \cdot x_2 + x_1^2 \cdot x_2^2]$$



More generally this is the polynomial kernel

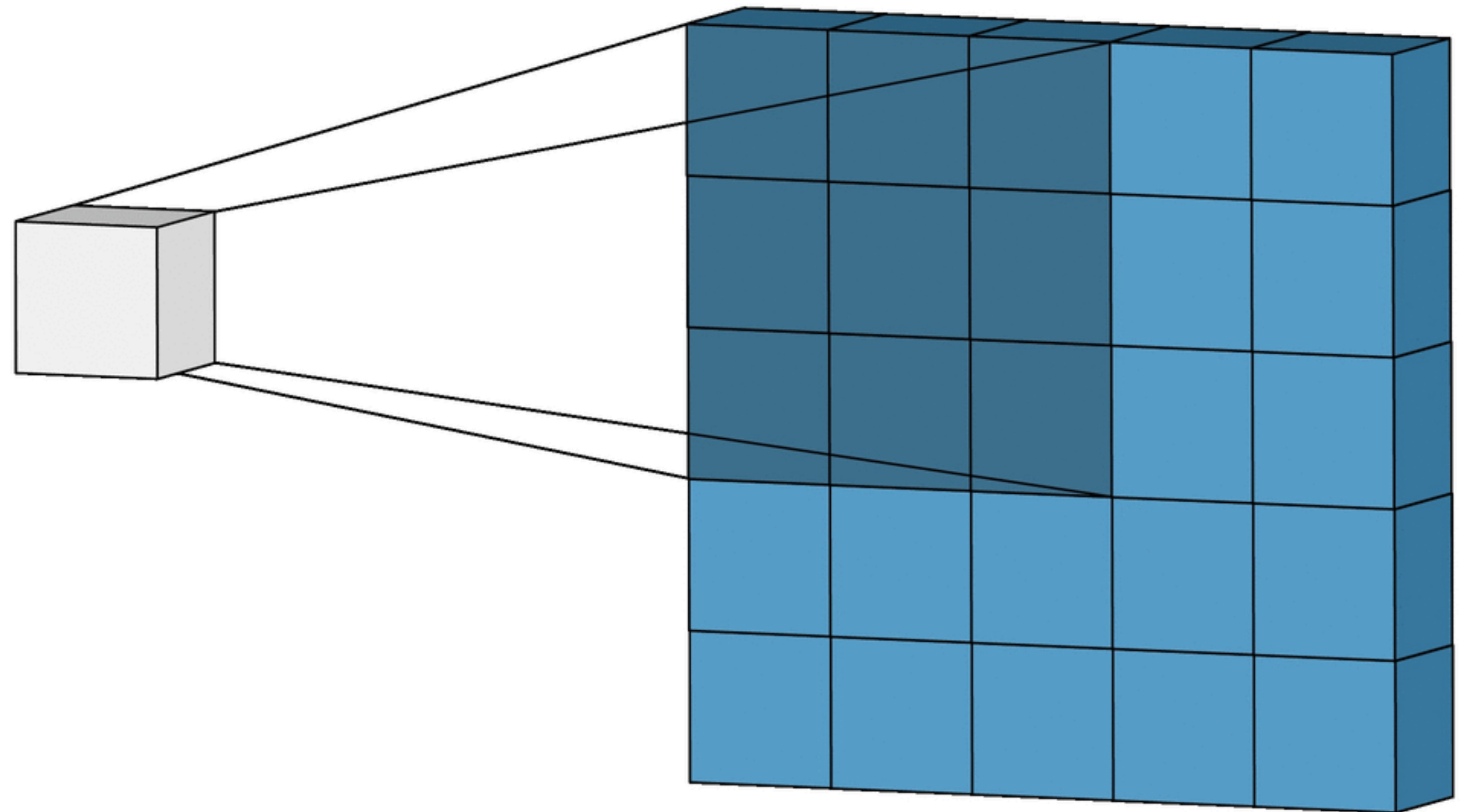
$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^d = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$$

$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$  means you never have to calculate  
the decision boundary in either  $\mathcal{X}$  or  $\mathcal{V}$  ...  
its in terms of a scalar dot product of vectors in  $\mathcal{V}$ !



# Common kinds of kernels

- Moving average window
- Polynomial of order  $d$
- Radial basis / Gaussian
- Sigmoid / tanh





# Totally skipping all the math: the dual formulation of the Lagrangian

## Support vector machines

$$\begin{aligned} \min_{\mathbf{w}, \xi} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_i \xi_i \\ \text{s.t.} \quad & \forall i, \quad y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i \\ & \forall i, \quad \xi_i \geq 0. \end{aligned}$$

Let  $\mathbf{w}$  be the minimizer of the SVM problem for some dataset with  $m$  examples:  $\{(\mathbf{x}_i, y_i)\}$

Then, for  $i = 1 \dots m$ , there exist  $\alpha_i \geq 0$  such that the optimum  $\mathbf{w}$  can be written as

$$\mathbf{w} = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i$$

Weight vector is completely defined by data samples with non-zero alpha... these are the support vectors!



## Dot products in high dimensional spaces

Let us define a dot product in the high dimensional space

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z})$$

So prediction with this *high dimensional lifting map* is

$$\text{sgn}(\mathbf{w}^T \phi(\mathbf{x})) = \text{sgn} \left( \sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) \right)$$

$$\text{because } \mathbf{w}^T \phi(\mathbf{x}) = \sum_i \alpha_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x})$$

26

Inner product: output is a scalar!

This is the kernel trick; we are doing the math in 1-D space

- not the input space

- not in the higher dimensional space the kernel transforms into

## Example: Polynomial Kernel

- Given two examples  $\mathbf{x}$  and  $\mathbf{z}$  we want to map them to a [high dimensional space](#) [for example, quadratic]

$$\phi(x_1, x_2, \dots, x_n) = [1, x_1, x_2, \dots, x_n, x_1^2, x_2^2, \dots, x_n^2, x_1x_2, \dots, x_{n-1}x_n]^T$$

and compute the dot product  $A = \phi(\mathbf{x})^T \phi(\mathbf{z})$  [takes time ]

- Instead, in the original space, compute

$$B = K(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x}^T \mathbf{z})^2$$

# The Kernel Trick

Suppose we wish to compute  $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^\top \phi(\mathbf{z})$

Here  $\phi$  maps  $\mathbf{x}$  and  $\mathbf{z}$  to a high dimensional space

***The Kernel Trick***: Save time/space by computing the value of  $K(\mathbf{x}, \mathbf{z})$  by performing operations in the original space (without a feature transformation!)

**The trick:**

**Replace a high dimensional projection with pairwise similarities between data samples.**

**Still roughly  $n^2$  but that's better than the high-D case!**

# Jupyter demo