

COMP 432 Machine Learning

# Support Vector Machines

Computer Science & Software Engineering  
Concordia University, Fall 2024



# Midterm & Final Exam

- The data for the midterm exam is confirmed:

Nov 4<sup>th</sup>, 2024, Monday  
5:45PM - 7:45PM  
Exam Room: H 435 SGW



# Summary of the last episode....

What we have seen **so far**:

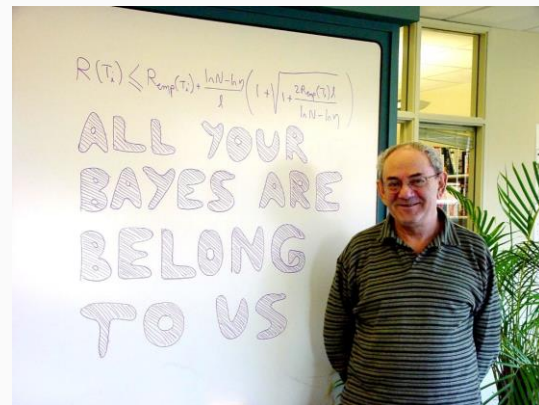
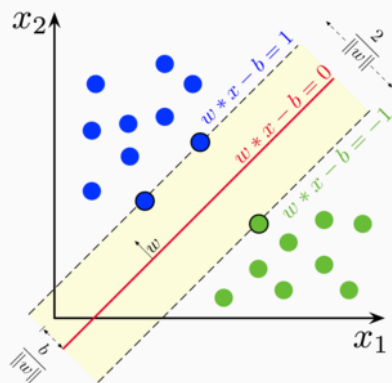
- Basic Machine Learning concepts.
- Optimization.
- Linear Models (linear regression, logistic regression, multi-class logistic regression).
- Neural Networks (MLPs, CNNs, RNNs).

What we are going to **learn today**:

- Linear Support Vector Machines (Maximum Margin Classifiers, Soft-Margin SVMs).

# Introduction

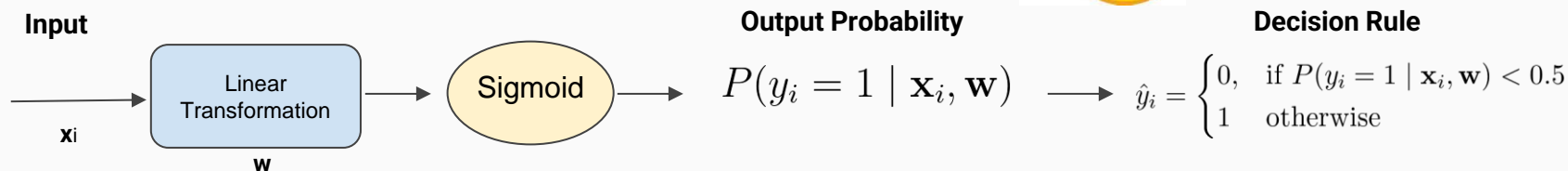
- **Support Vector Machine (SVM)** is an elegant and powerful machine learning model.
- Early ideas developed in 1960s and 1970s by Vladimir Vapnik and Alexey Chervonenkis.
- It was the dominant paradigm in machine learning before the advent of deep learning.
- Originally developed for **binary classification tasks**, then extended to regression and multi-class classification.



# Binary Classifiers

- **Support Vector Machines** are **linear models** for **binary classification**.
- Wait....we already have seen a linear model for binary classification! Do you remember its name?

**Logistic Regression:**  $P(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{w}^T \mathbf{x}_i)$

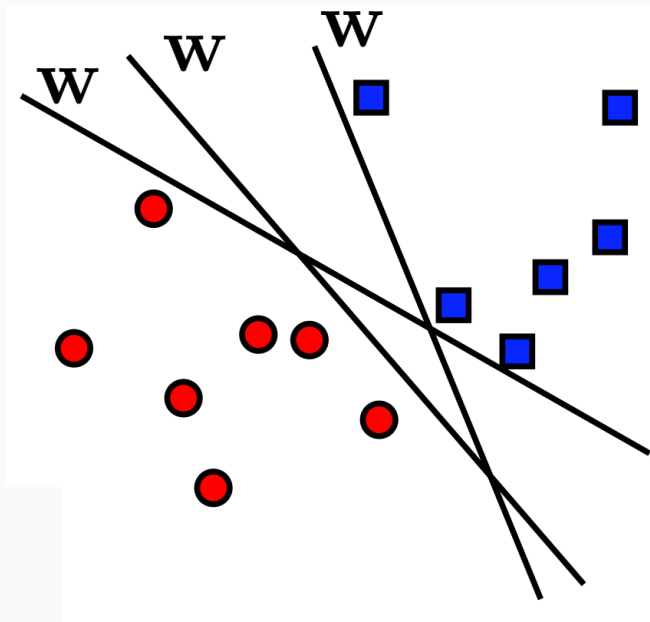


In logistic regression, we find the parameters  $\mathbf{w}$  that minimize the **Binary Cross-Entropy**.

- Logistic Regression is only one way to perform linear binary classification. There are many others!

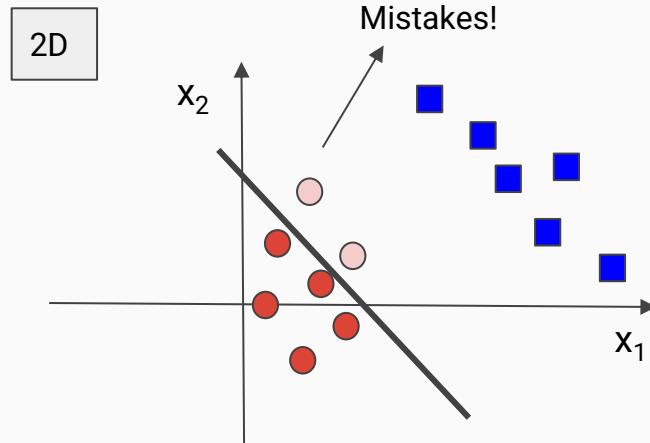
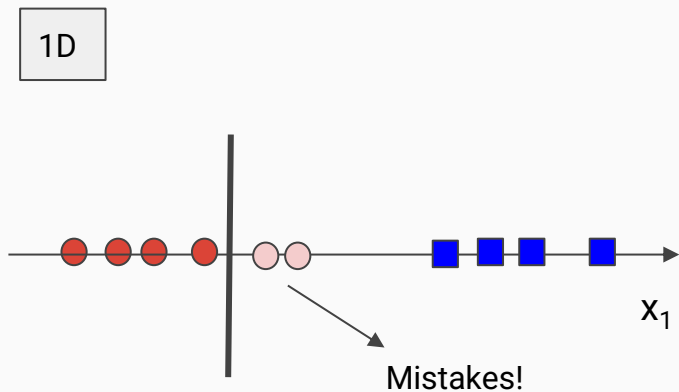
# Binary Classifiers

- If data is linearly separable, we can choose from among many possible separating **hyperplanes**.
- Different algorithms provide solutions.
- *Is any choice better than the others? In what sense?*



# Binary Classifiers

- For instance, let's consider the following training data points:



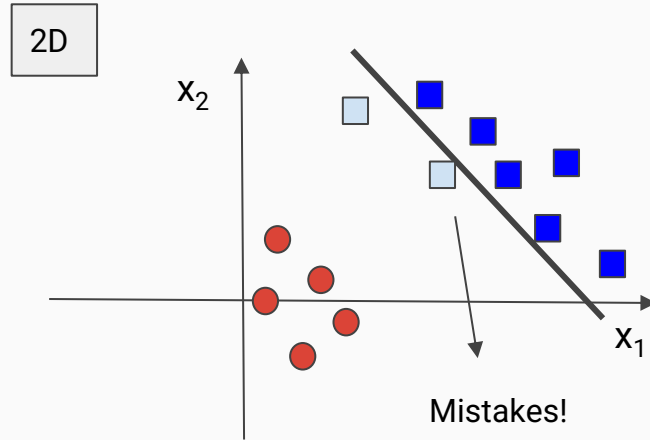
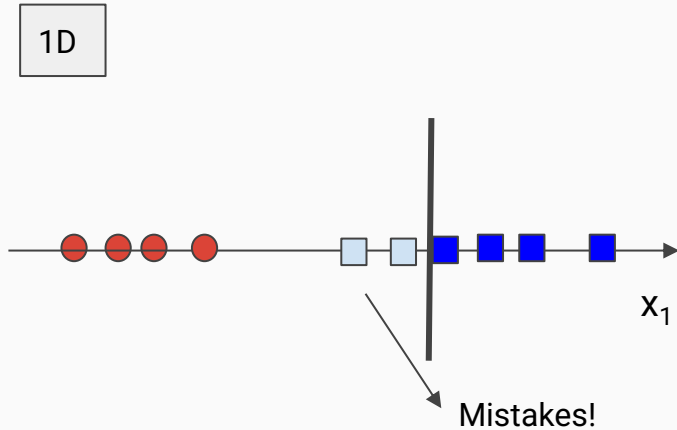
- Do you think this is a good choice for the decision boundary?*

**BAD IDEA**

If we place the boundary too close to the red cluster, we likely misclassify new "red" samples (generalization issue).

# Binary Classifiers

- For instance, let's consider the following training data points:



- Do you think this is a good choice for the decision boundary?*

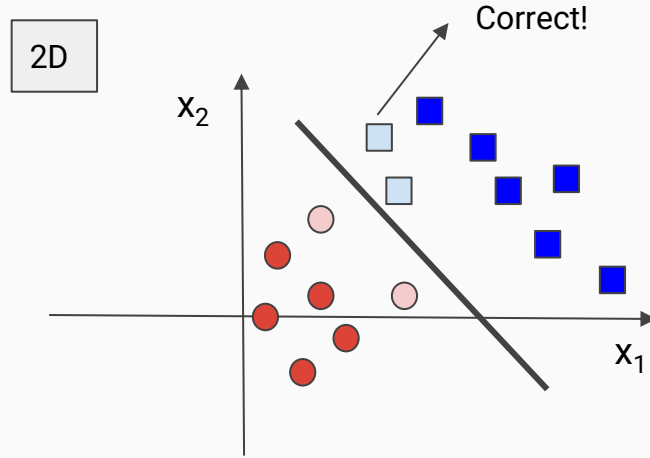
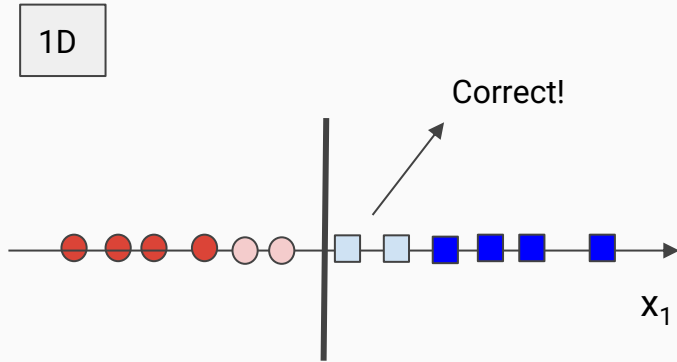
**BAD IDEA**

If we place the boundary too close to the blue cluster, we likely misclassify new "blue" samples (generalization issue).



# Binary Classifiers

- For instance, let's consider the following training data points:



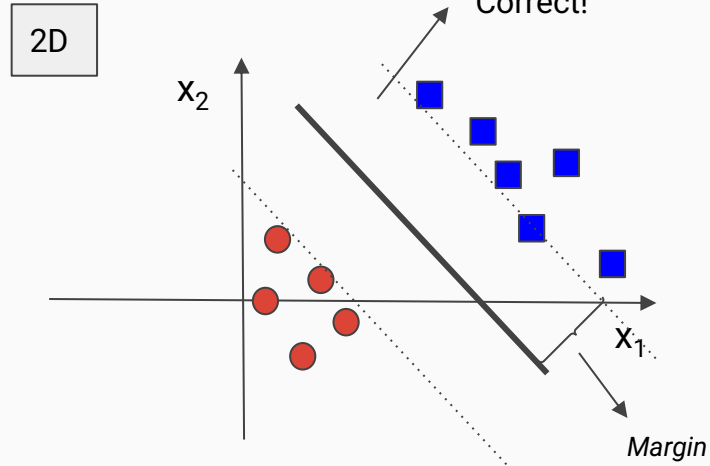
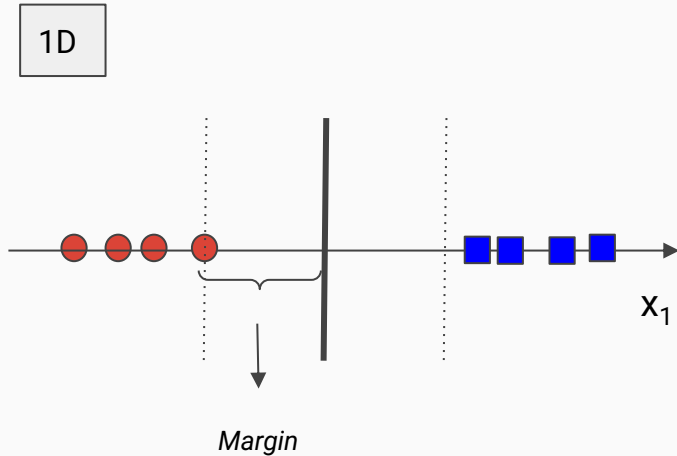
- Do you think this is a good choice for the decision boundary?*



If we place the boundary between the clusters, we are more likely to **generalize**.

# Margin

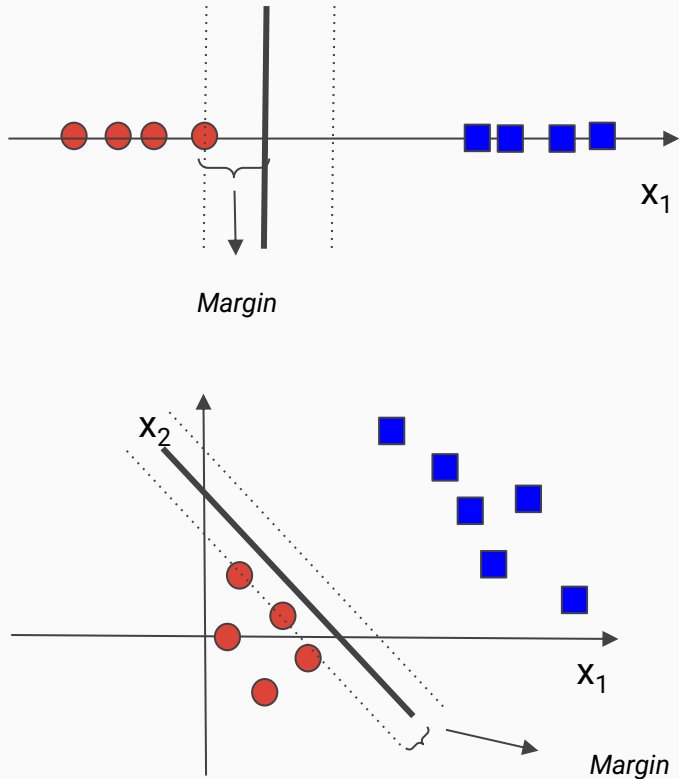
- To do that, we can introduce the concept of **margin**.



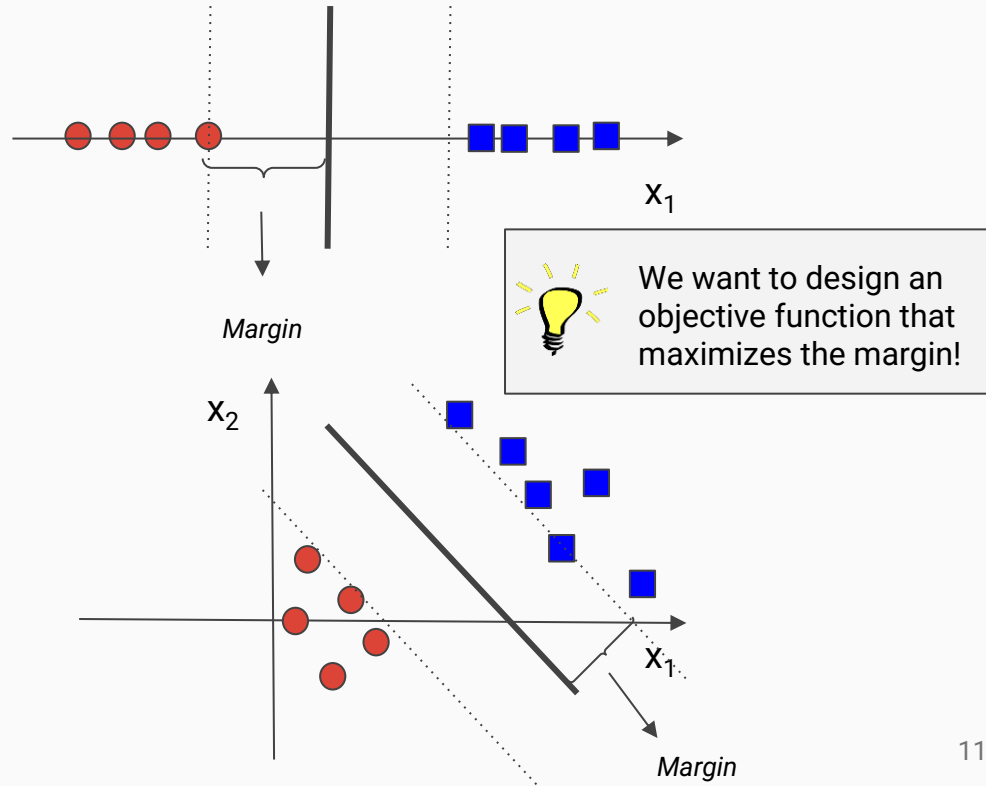
The margin is the **distance** of the **nearest point(s)** to the **hyperplane**.

# Margin

Small margin → Less likely to generalize

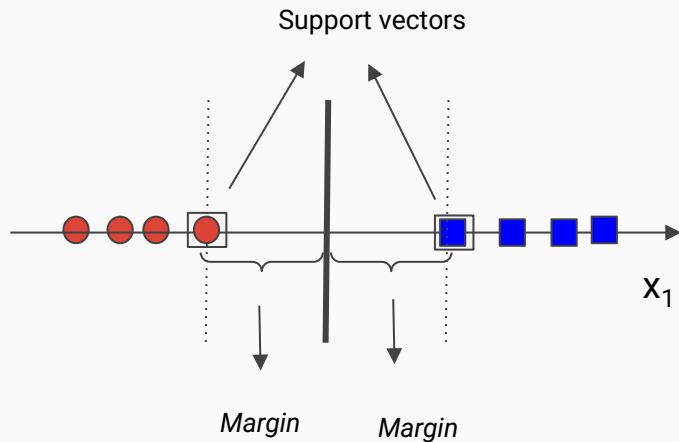


Large margin → More likely to generalize



# Maximum Margin Classifier

- The maximum margin classifier (used in the context of SVMs) seeks the separating hyperplane that has **maximum margin** from the training samples.

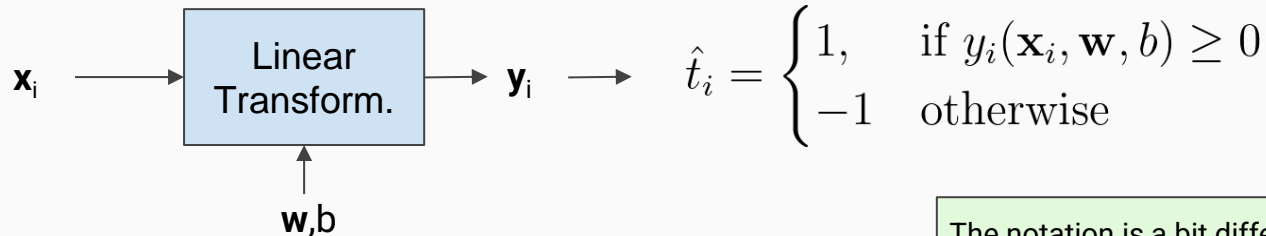


- The margin is maximized when the boundaries are placed halfway between the classes.
- The position of the boundary is determined by a subset of data points known as **support vectors**.

- A training data point is a “support vector” if no other data point has a strictly smaller distance to the hyperplane.

# Maximum Margin Classifier

## Decision Rule



$$y_i(\mathbf{x}_i, \mathbf{w}, b) = \mathbf{w}^T \mathbf{x}_i + b$$

$\mathbf{x}_i = [x_1, x_2, \dots, x_D]^T$  (**Input**, D-dimensional vector)

$\mathbf{w} = [w_1, w_2, \dots, w_D]^T$  (**Weights**, D-dimensional vector)

$b$  (**bias**, scalar)

$y_i(\mathbf{x}_i, \mathbf{w}, b)$  (**output**, scalar)

The notation is a bit different from that used so far



The two labels are -1 or +1 (and not 0 and 1 like assumed so far):

$$t_i = \{-1, +1\}$$

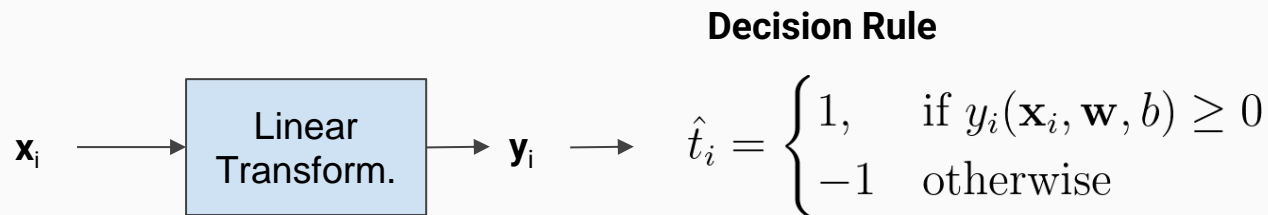
This is done for mathematical convenience (that will be clearer later).



The bias term is not included in  $\mathbf{w}$

This is done for mathematical convenience as well.

# Maximum Margin Classifier



$$y_i(\mathbf{x}_i, \mathbf{w}, b) = \mathbf{w}^T \mathbf{x}_i + b$$

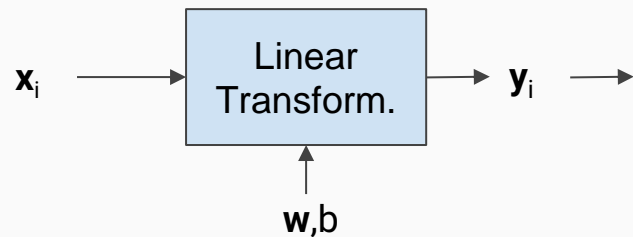


*Can you see an important difference with logistic regression?*

We are not turning the output into a **probability** (i.e., no **sigmoid** applied after the linear transformation)

In logistic regression, we train the binary classifier by minimizing the **binary cross-entropy** (which requires probabilities). If we don't use this training strategy, we don't necessarily need to turn the output into a probability.

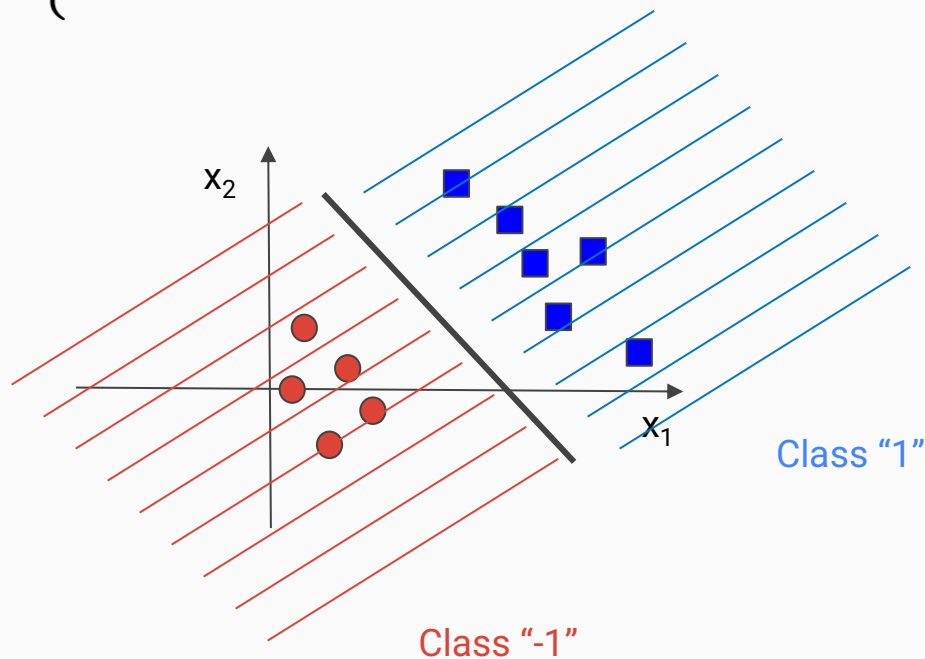
# Maximum Margin Classifier



$$y_i(\mathbf{x}_i, \mathbf{w}, b) = \mathbf{w}^T \mathbf{x}_i + b$$

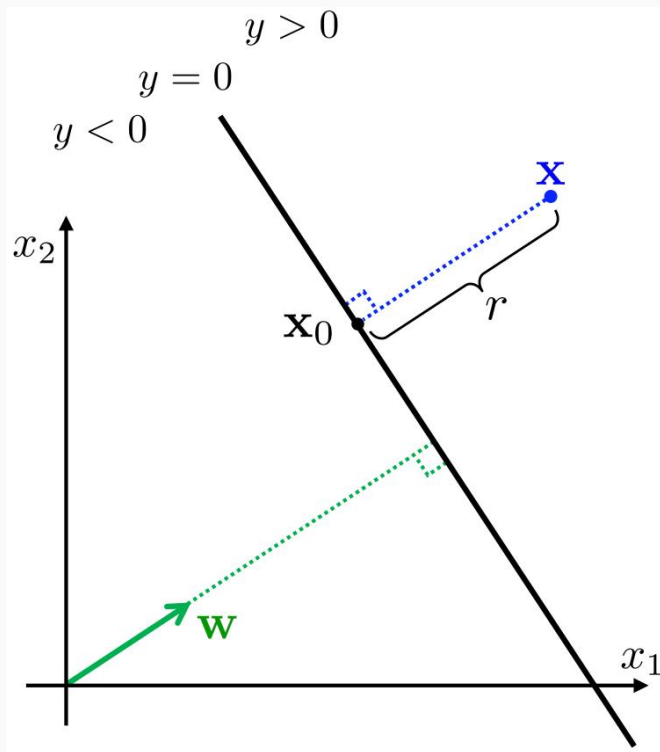
## Decision Rule

$$\hat{t}_i = \begin{cases} 1, & \text{if } y_i(\mathbf{x}_i, \mathbf{w}, b) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$



# Maximum Margin Classifier

- Let's now dive more in detail into the **geometry** of the input space.



Geometric consideration 1:

- A point  $\mathbf{x}$  lies on the **decision boundary** if:

$$y(\mathbf{x}, \mathbf{w}, b) = \mathbf{w}^T \mathbf{x} + b = 0$$

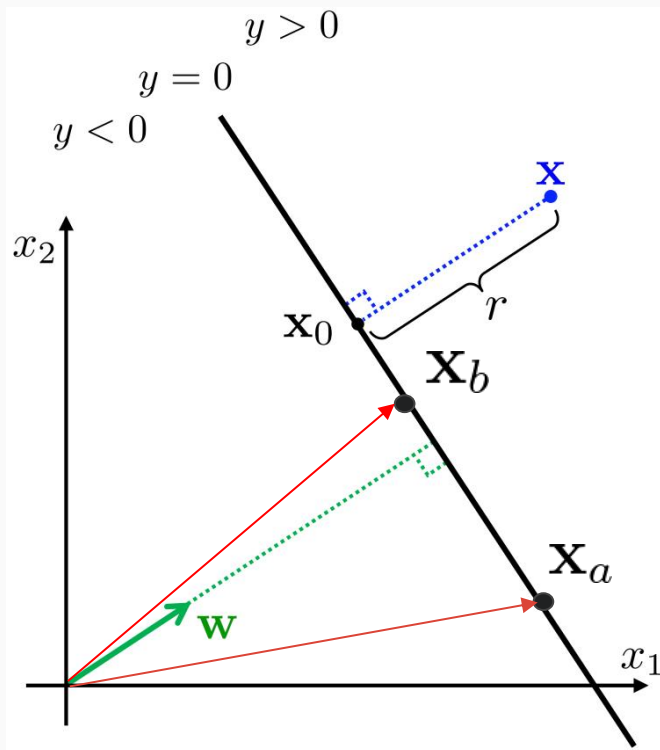
Geometric consideration 2:

- The weight vector  $\mathbf{w}$  is perpendicular to the decision boundary.

Why?



# Maximum Margin Classifier



Geometric consideration 2:

The weight vector  $\mathbf{w}$  is perpendicular to the decision boundary.

Let's take two generic points  $\mathbf{X}_a$  and  $\mathbf{X}_b$  lying on the decision boundary:

$$\mathbf{w}^T \mathbf{x}_a + b = 0$$

$$\mathbf{w}^T \mathbf{x}_b + b = 0$$



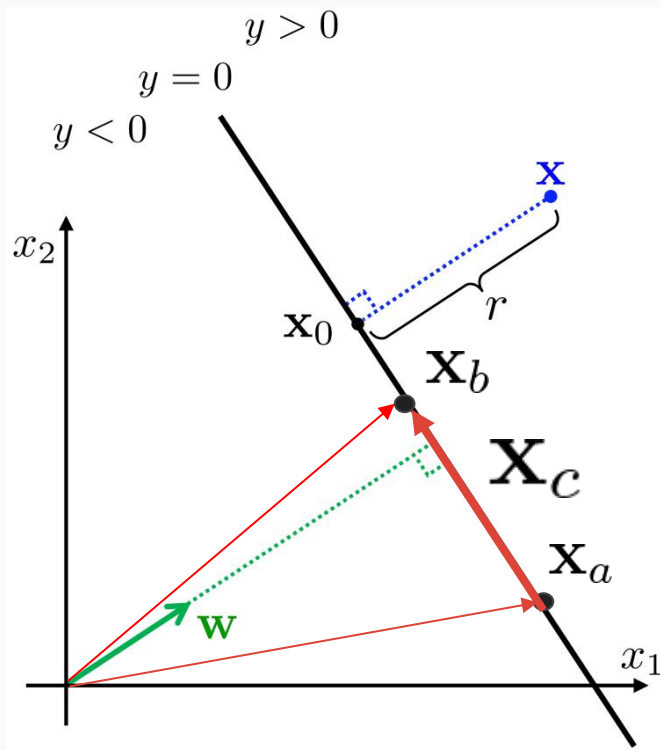
subtract

$$\mathbf{w}^T (\mathbf{x}_a - \mathbf{x}_b) = 0 \quad \longrightarrow \quad \mathbf{w} \text{ and } \mathbf{x}_c \text{ are perpendicular}$$

$\underbrace{\hspace{1.5cm}}$   
 $\mathbf{x}_c = \mathbf{x}_a - \mathbf{x}_b$

Remember: Two vectors are **perpendicular** (orthogonal) if their **dot product** is 0.

# Maximum Margin Classifier



Geometric consideration 2:

The weight vector  $\mathbf{w}$  is perpendicular to the decision boundary.

$$\mathbf{w}^T (\mathbf{x}_a - \mathbf{x}_b) = 0 \quad \longrightarrow \quad \mathbf{w} \text{ and } \mathbf{x}_c \text{ are perpendicular}$$
$$\mathbf{x}_c = \mathbf{x}_a - \mathbf{x}_b$$

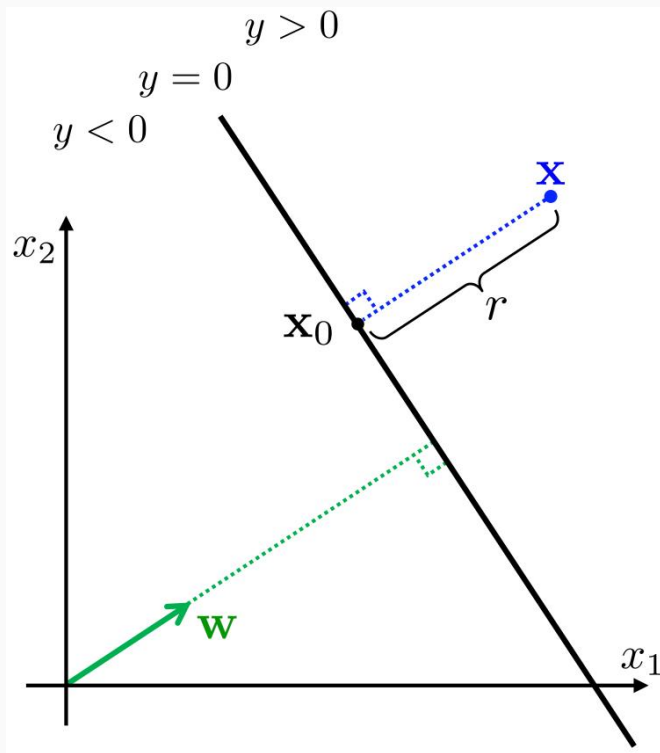
The point  $\mathbf{x}_c$  is parallel to the boundary!

If  $\mathbf{w}$  is orthogonal to  $\mathbf{x}_c$  (which is parallel with the decision boundary)



$\mathbf{w}$  is orthogonal to the decision boundary!

# Maximum Margin Classifier



- Let's now introduce the quantity  $r$ , which is the distance of the point  $\mathbf{x}$  from the decision boundary.

- The point  $\mathbf{x}$  can be written as:

$$\mathbf{X} = \mathbf{X}_0 + r \frac{\mathbf{W}}{\|\mathbf{W}\|}$$

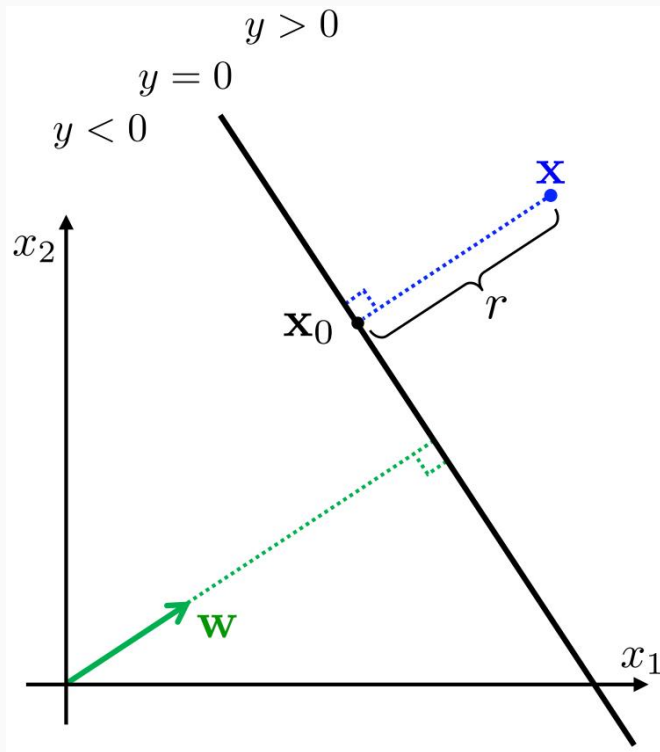
Unit vector  
(needed to point  
in the "right"  
direction)

Where:

$\mathbf{X}_0 \longrightarrow$  Projection of  $\mathbf{x}$  onto the decision boundary

$$\|\mathbf{w}\| = \sqrt{\mathbf{w}^T \mathbf{w}} \longrightarrow \text{Length of } \mathbf{w}$$

# Maximum Margin Classifier



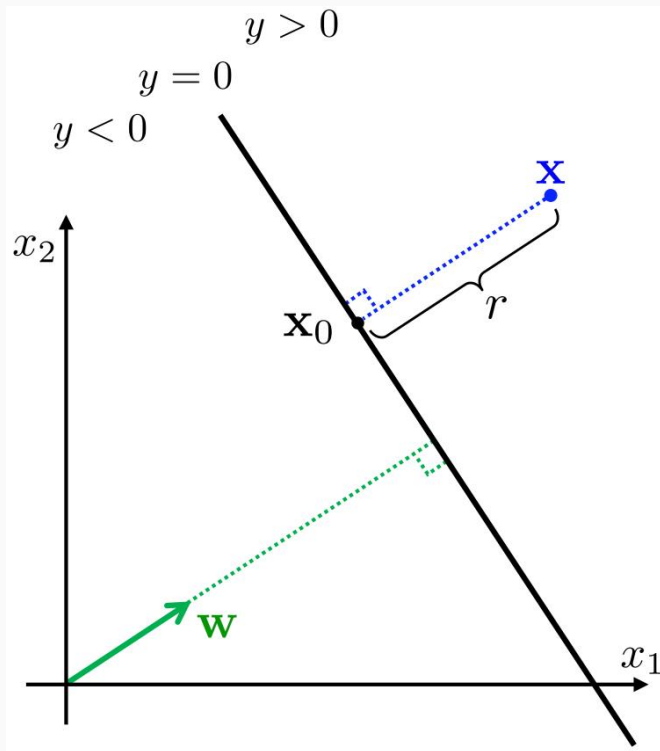
- Let's try to find an expression of  $r$  that only depends on  $\mathbf{x}$ ,  $\mathbf{w}$ , and  $\mathbf{b}$ .
- A point  $\mathbf{x}_0$  in the boundary can be written as:

$$\mathbf{x}_0 = \mathbf{x} - r \frac{\mathbf{w}}{\|\mathbf{w}\|} \quad (1) \text{ Boundary point}$$

$$\mathbf{w}^T \mathbf{x}_0 + b = 0 \quad (2) \text{ Boundary condition}$$

$$\mathbf{w}^T \left( \mathbf{x} - r \frac{\mathbf{w}}{\|\mathbf{w}\|} \right) + b = 0 \quad (3) \text{ Replace (1) in (2)}$$

# Maximum Margin Classifier



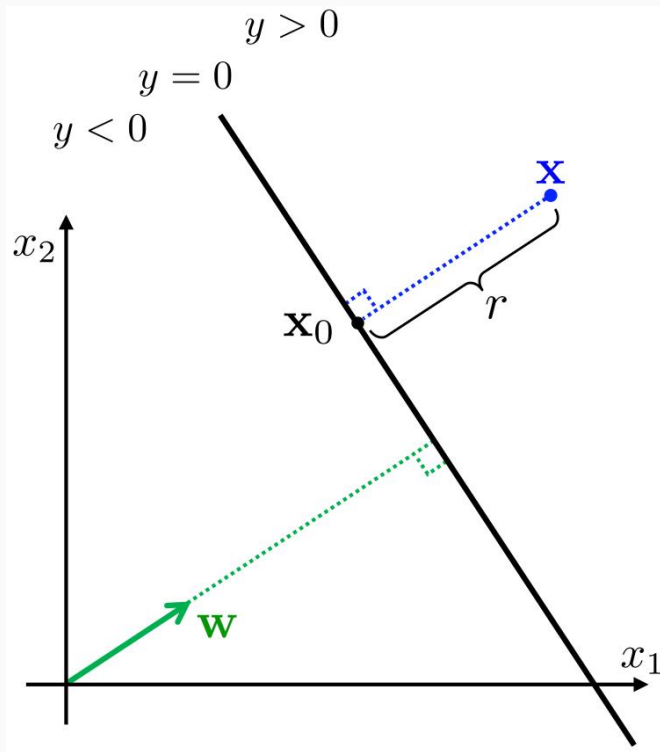
- Let's try to find an expression of  $r$  that only depends on  $\mathbf{x}$ ,  $\mathbf{w}$ , and  $\mathbf{b}$ .

$$\mathbf{w}^T \left( \mathbf{x} - r \frac{\mathbf{w}}{\|\mathbf{w}\|} \right) + b = 0 \quad (3) \text{ Replace (1) in (2)}$$

$$\underbrace{\mathbf{w}^T \mathbf{x} + b}_{y(\mathbf{x}, \mathbf{w}, b)} - r \underbrace{\frac{\mathbf{w}^T \mathbf{w}}{\|\mathbf{w}\|}}_{\|\mathbf{w}\|} = 0 \quad (4) \text{ Expansion}$$

$$y(\mathbf{x}, \mathbf{w}, b) - r \|\mathbf{w}\| = 0 \quad (5) \text{ Substitution}$$

# Maximum Margin Classifier



- Let's try to find an expression of  $r$  that only depends on  $\mathbf{x}$ ,  $\mathbf{w}$ , and  $\mathbf{b}$ .

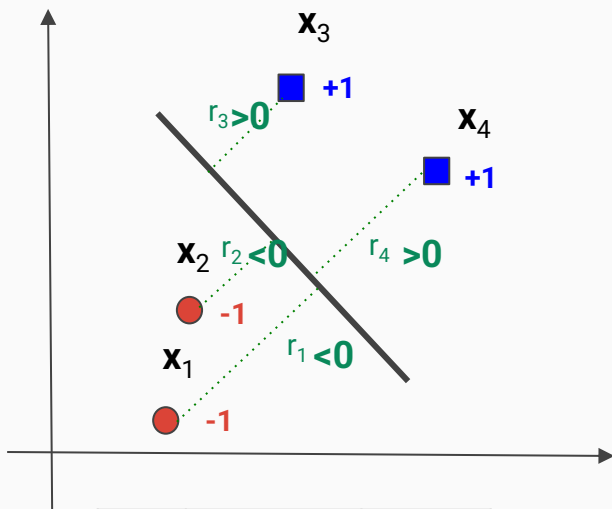
$$y(\mathbf{x}, \mathbf{w}, b) - r \|\mathbf{w}\| = 0 \quad (5) \text{ Substitution}$$



$$r = \frac{y(\mathbf{x}, \mathbf{w}, b)}{\|\mathbf{w}\|}$$

- $r$  is actually a **signed distance** to hyperplane:  
 $r < 0$  for **negative** samples (class “-1”)  
 $r > 0$  for **positive** samples (class “1”)

# Maximum Margin Classifier



	Label (t)	r
$\mathbf{x}_1$	-1	-1.5
$\mathbf{x}_2$	-1	-0.7
$\mathbf{x}_3$	+1	+0.8
$\mathbf{x}_4$	+1	+0.9

- We have a training dataset of points  $\mathbf{X}=[\mathbf{x}_1^\top, \mathbf{x}_2^\top, \dots, \mathbf{x}_N^\top]$  with labels  $\mathbf{t}=[t_1, t_2, \dots, t_N]^\top$  (where each label can be either -1 or +1)

How can I compute the **margin**?



Remember:

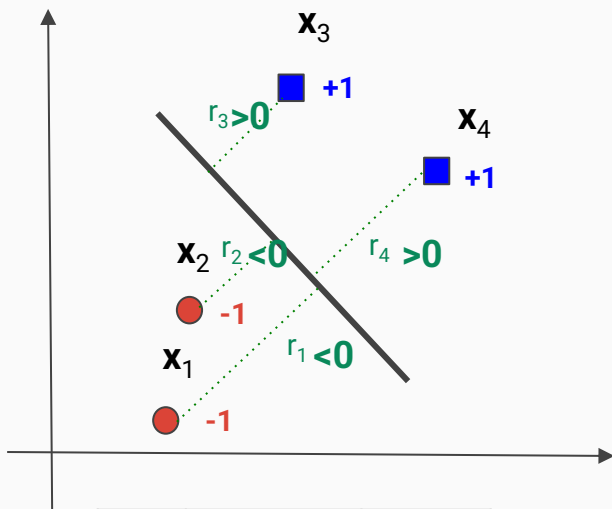
The margin is the **distance** of the **nearest point(s)** to the **hyperplane**.

What about computing the margin as follows?

$$\text{margin} = \min_{i=1, \dots, N} r_i = \min_{i=1, \dots, N} \frac{y(\mathbf{x}_i, \mathbf{w}, b)}{\|\mathbf{w}\|}$$

**BAD IDEA Why?**

# Maximum Margin Classifier



	Label (t)	r
$x_1$	-1	-1.5
$x_2$	-1	-0.7
$x_3$	+1	+0.8
$x_4$	+1	+0.9

Remember:

The margin is the **distance** of the **nearest point(s)** to the **hyperplane**.

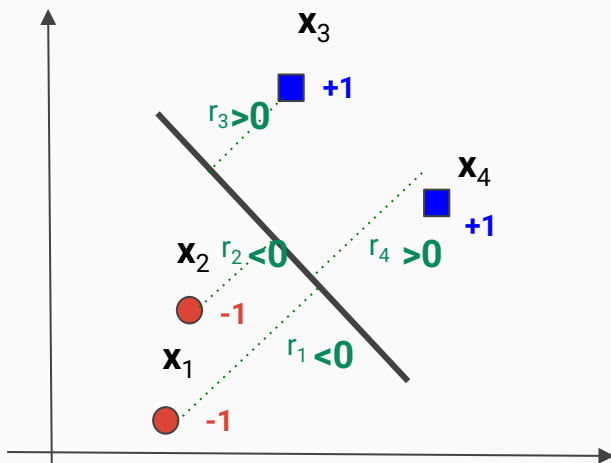
$$\text{margin} = \min_{i=1,\dots,N} r_i = \min_{i=1,\dots,N} \frac{y(\mathbf{x}_i, \mathbf{w}, b)}{\|\mathbf{w}\|}$$

**BAD IDEA**

- $r$  is a **signed distance** that is negative for values belonging to the class “-1”.
- If we do the min, it will return the  $r$  corresponding to the farthest point from the hyperplane belonging to the class “-1”.
- This is not the margin!



# Maximum Margin Classifier



- What about this way?

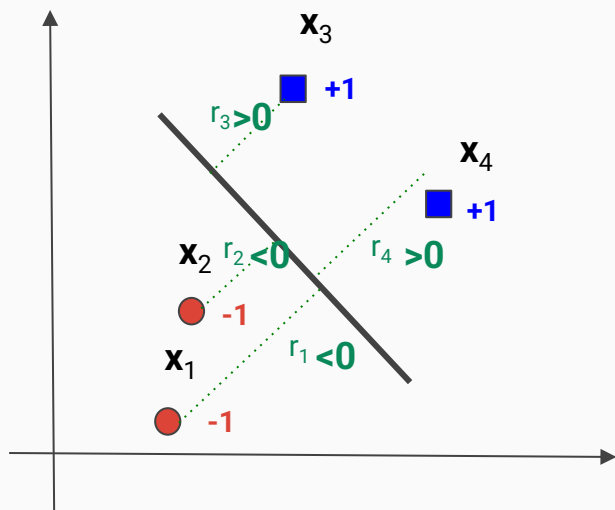
$$margin = \min_{i=1,\dots,N} |r_i|$$

- This seems more meaningful, as it returns the distance between the points closest to the hyperplane.
- However, this is still a **BAD IDEA**

**Why?**

	Label (t)	r	r
$x_1$	-1	-1.5	+1.5
$x_2$	-1	-0.7	+0.7
$x_3$	+1	+0.8	+0.8
$x_4$	+1	+0.9	+0.9

# Maximum Margin Classifier

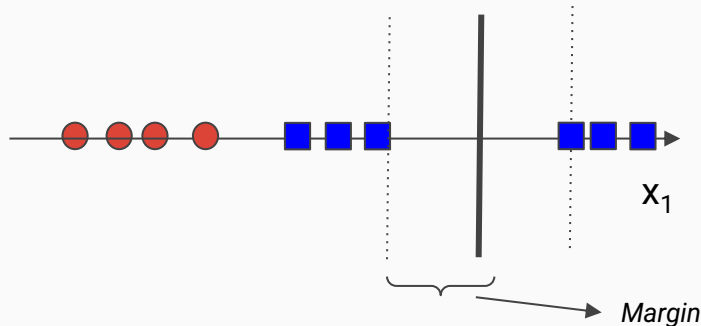


- What about this way?

$$\text{margin} = \min_{i=1,\dots,N} |r_i|$$

**BAD IDEA**

- We want to use the **labels** as well in the **margin** computation.
- This is because we will use the **maximum margin** as a training objective.



- If we do not involve the labels, we risk to converge to “**dummy**” solutions (see figure on the left).

# Maximum Margin Classifier

- What about this way?

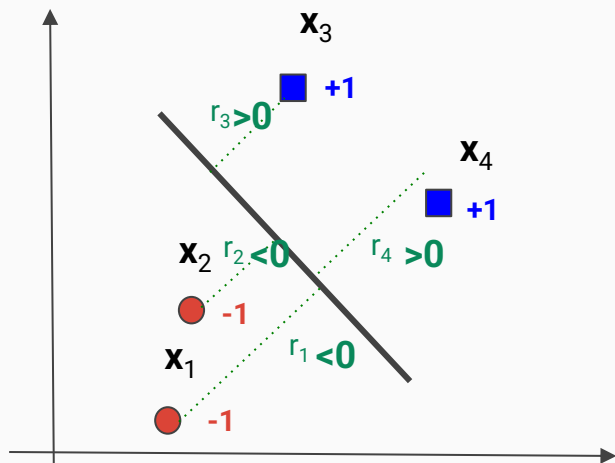
$$\text{margin} = \min_{i=1,\dots,N} t_i r_i$$



- This is more meaningful because we have the **labels** involved in the margin definition.

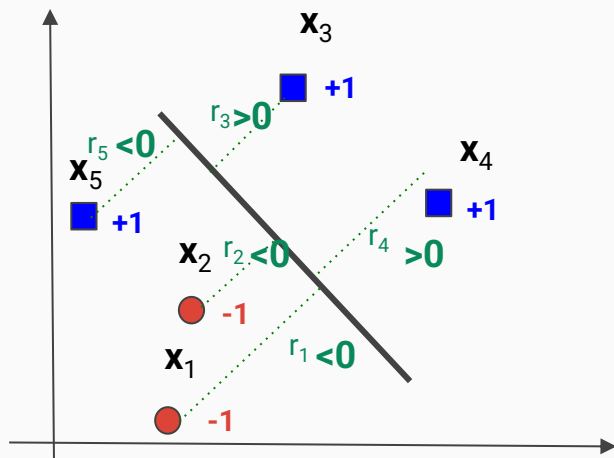
If all points are **correctly classified** (as in the figure), the margin is a **positive quantity**:

- For **negative samples**, we multiply the negative  $r$  with the negative label  $-1$  (thus obtaining a positive quantity).
- For **positive samples**, we multiply the positive  $r$  with the positive label  $+1$  (thus obtaining a positive quantity).



	Label (t)	r	tr
$x_1$	-1	-1.5	+1.5
$x_2$	-1	-0.7	+0.7
$x_3$	+1	+0.8	+0.8
$x_4$	+1	+0.9	+0.9

# Maximum Margin Classifier



- What about this way?

$$\text{margin} = \min_{i=1, \dots, N} t_i r_i$$



If there is at least one point misclassified, the margin is a **negative quantity**:

- If a **positive sample** is misclassified, we multiply the negative  $r$  with the positive label  $+1$  (thus obtaining a negative quantity).
- If a **negative sample** is misclassified, we multiply the positive  $r$  with the negative label  $-1$  (thus obtaining a negative quantity).

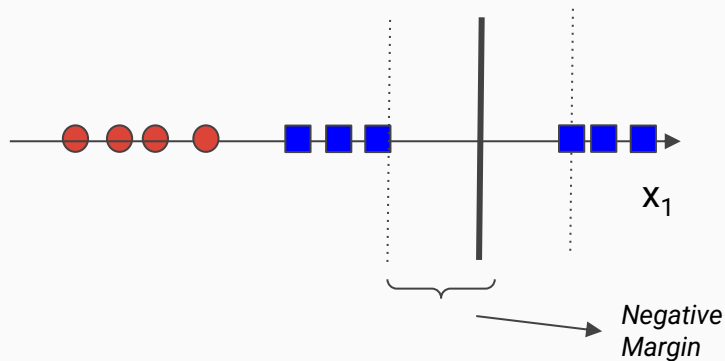
	Label (t)	r	tr
$x_1$	-1	-1.5	+1.5
$x_2$	-1	-0.7	+0.7
$x_3$	+1	+0.8	+0.8
$x_4$	+1	+0.9	+0.9
$x_5$	+1	-0.8	-0.8

# Maximum Margin Classifier

$$\text{margin} = \min_{i=1,\dots,N} t_i r_i$$

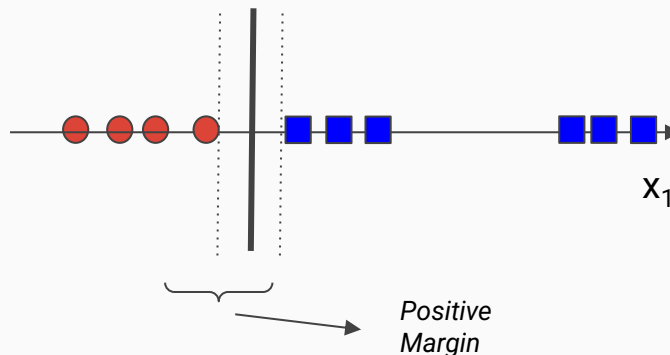
- This definition avoids the risk of learning “dummy” solutions when looking for maximum margin:

**Bad solution**



This is a **bad** solution now because it has a **negative margin**.

**Good solution**



This is a **good** solution now because it has a **positive margin** (the largest one)

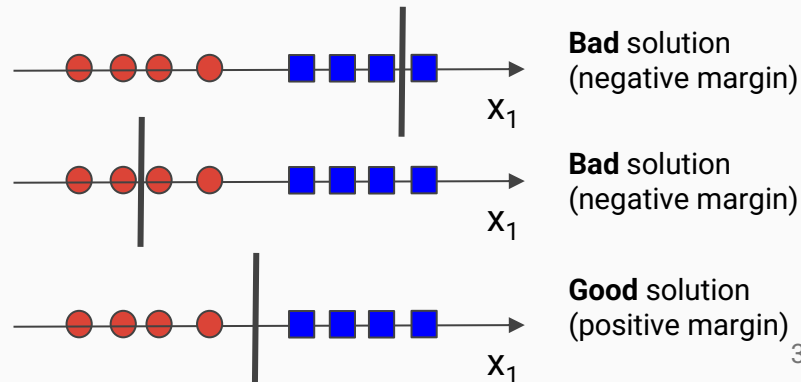
# Maximum Margin Classifier

- We now have a way to compute the **margin** given a particular parameter configuration  $\mathbf{w}, b$ .
- We can train the model by solving the following max-min **optimization problem**:

$$\mathbf{w}^*, b^* = \underset{\mathbf{w}, b}{\operatorname{argmax}} \underbrace{\min_{i=1, \dots, N} t_i r_i}_{\text{margin}}$$

Among all the possible combinations of  $\mathbf{w}$  and  $b$ , we want the one that maximizes the margin.

$$r_i = \frac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$$



# Maximum Margin Classifier

$$\mathbf{w}^*, b^* = \operatorname{argmax}_{\mathbf{w}, b} \min_{i=1, \dots, N} t_i r_i$$

$$r_i = \frac{\mathbf{w}^T \mathbf{x}_i + b}{||\mathbf{w}||}$$

- *How can we solve this problem?*



- Potentially we can solve this problem with **gradient descent**.
- In this case we have to use the  $J(\mathbf{w}, b) = - \min_{i=1, \dots, N} t_i r_i$  as the **objective function to minimize**.

# Maximum Margin Classifier

$$\mathbf{w}^*, b^* = \operatorname{argmax}_{\mathbf{w}, b} \min_{i=1, \dots, N} t_i r_i$$

$$r_i = \frac{\mathbf{w}^T \mathbf{x}_i + b}{||\mathbf{w}||}$$

- However, gradient descend is not the best way to go here.
- The min functions is quite critical. It is **differentiable** in many points (but not in all of them).
- This objective is **non-convex**, making the gradient descend sensitive to possible **local minima** and **initialization points**.
- Moreover, gradient descent will introduce the **learning rate** that has to be tuned on validation data.
- Fortunately, there is a way to turn this problem into an **equivalent convex (constrained) optimization** problem with a **direct solution**.



# Maximum Margin Classifier

$$\mathbf{w}^*, b^* = \operatorname{argmax}_{\mathbf{w}, b} \min_{i=1, \dots, N} t_i r_i$$

$$r_i = \frac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$$

- The direct solution to this problem would be very complex. Let's try to convert it into an **equivalent problem** much **easier to solve**.

$$\mathbf{w}^*, b^* = \operatorname{argmax}_{\mathbf{w}, b} \min_{i=1, \dots, N} \frac{t_i(\mathbf{w}^T \mathbf{x}_i + b)}{\|\mathbf{w}\|}$$

such that  $t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 0$  for  $i = 1, \dots, N$

- Since we assume that the classes are **linearly separable**, we can restrict the search to those parameters that **classify correctly all the points**.

Still hard to solve...

# Maximum Margin Classifier

$$\mathbf{w}^*, b^* = \operatorname{argmax}_{\mathbf{w}, b} \min_{i=1, \dots, N} \frac{t_i(\mathbf{w}^T \mathbf{x}_i + b)}{\|\mathbf{w}\|}$$

such that  $t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 0$  for  $i = 1, \dots, N$

$$r_i = \frac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$$

- We can note that the problem is **scale-invariant**:

$$\frac{\alpha \mathbf{w}^T \mathbf{x}_i + \alpha b}{\|\alpha \mathbf{w}\|} = \alpha \frac{\mathbf{w}^T \mathbf{x}_i + b}{\alpha \|\mathbf{w}\|} = r_i$$

If we multiply  $\mathbf{w}$  and  $b$  by a positive constant  $\alpha$ , the margin is the same. Basically, the scale of  $\mathbf{w}$  (and  $b$ ) does not matter and we can choose a convenient one...

# Maximum Margin Classifier

$$\mathbf{w}^*, b^* = \operatorname{argmax}_{\mathbf{w}, b} \min_{i=1, \dots, N} \frac{t_i(\mathbf{w}^T \mathbf{x}_i + b)}{\|\mathbf{w}\|}$$

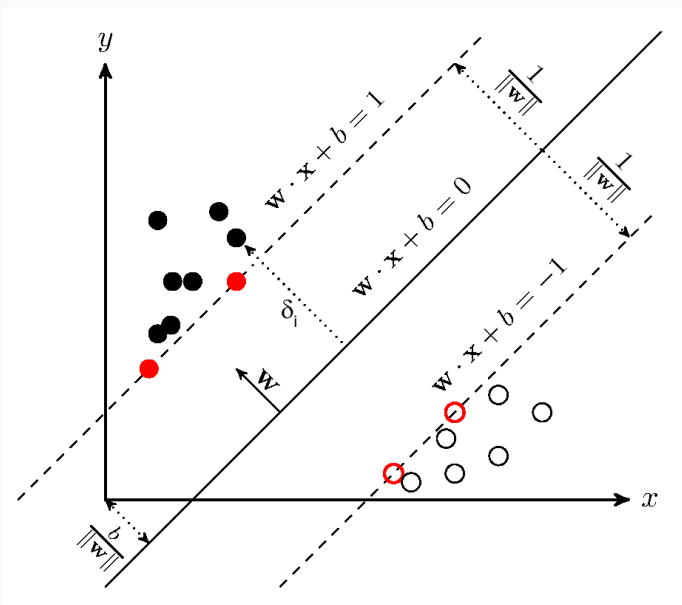
such that  $t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 0$  for  $i = 1, \dots, N$

Scale  $\mathbf{w}$  and  $b$  such that:

$$t_n(\mathbf{w}^T \mathbf{x}_n + b) = 1$$

$$t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \text{ for } i = 1, \dots, N$$

Points lying on the margin



This means that, for a given set of parameters  $\mathbf{w}$  and  $b$ , the margin will be:

$$\text{margin} = \min_{i=1, \dots, N} \frac{t_i(\mathbf{w}^T \mathbf{x}_i + b)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$$

# Maximum Margin Classifier

$$\mathbf{w}^*, b^* = \operatorname{argmax}_{\mathbf{w}, b} \min_{i=1, \dots, N} \frac{t_i(\mathbf{w}^T \mathbf{x}_i + b)}{\|\mathbf{w}\|}$$

such that  $t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 0$  for  $i = 1, \dots, N$



$$\mathbf{w}^*, b^* = \operatorname{argmax}_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|}$$

such that  $t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$  for  $i = 1, \dots, N$

Scale  $\mathbf{w}$  and  $b$  such that:

$$t_n(\mathbf{w}^T \mathbf{x}_n + b) = 1$$

$$t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \text{ for } i = 1, \dots, N$$

# Maximum Margin Classifier

$$\mathbf{w}^*, b^* = \operatorname{argmax}_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|}$$

such that  $t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$  for  $i = 1, \dots, N$



$$\mathbf{w}^*, b^* = \operatorname{argmax}_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|^2}$$

such that  $t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$  for  $i = 1, \dots, N$

Maximizing  $1/\|\mathbf{w}\|$  is equivalent to Maximizing  $1/\|\mathbf{w}\|^2$



$$\mathbf{w}^*, b^* = \operatorname{argmax}_{\mathbf{w}, b} \min_{i=1, \dots, N} \frac{t_i(\mathbf{w}^T \mathbf{x}_i + b)}{\|\mathbf{w}\|}$$

such that  $t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 0$  for  $i = 1, \dots, N$

This is introduced for mathematical convenience (useful when solving the problem)

$$\mathbf{w}^*, b^* = \operatorname{argmin}_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$$

such that  $t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$  for  $i = 1, \dots, N$

Maximizing  $1/\|\mathbf{w}\|^2$  is equivalent to minimizing  $\|\mathbf{w}\|^2$

# Maximum Margin Classifier

$$\mathbf{w}^*, b^* = \operatorname{argmin}_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$$

such that  $t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$  for  $i = 1, \dots, N$

→ This part finds the parameters with the maximum margin.

→ This part restricts the optimization space to only those parameters for which we have a perfect classifier.

Among all the possible solutions that lead to a perfect classifier, take the one with the smallest  $\|\mathbf{w}\|^2$



*Does that remind something to you?*

This is **L2 regularization!**

The effect of L2 regularization is to increase the margin between the classes.

# Maximum Margin Classifier

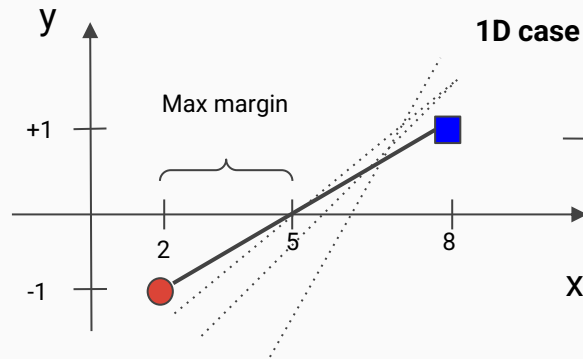
$$\mathbf{w}^*, b^* = \operatorname{argmin}_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$$

such that  $t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$  for  $i = 1, \dots, N$

→ This part finds the parameters with the minimum margin.

→ This part restricts the optimization space to only those parameters for which we have a perfect classifier.

Among all the possible solutions that lead to a perfect classifier, take the one with the smallest  $\|\mathbf{w}\|^2$



→ The line corresponding to the highest margin (on the x axis) is the one with the smallest slope that classifies correctly the two points.

# Maximum Margin Classifier

$$\mathbf{w}^*, b^* = \operatorname{argmin}_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$$

such that  $t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$  for  $i = 1, \dots, N$

→ This part finds the parameters with the minimum margin.

→ This part restricts the optimization space to only those parameters for which we have a perfect classifier.

Let's consider the following dataset:  $D = \{(x_1, t_1), (x_2, t_2)\} = \{(2, -1), (8, 1)\}$

$$y = wx + b \quad (1D \text{ Case})$$

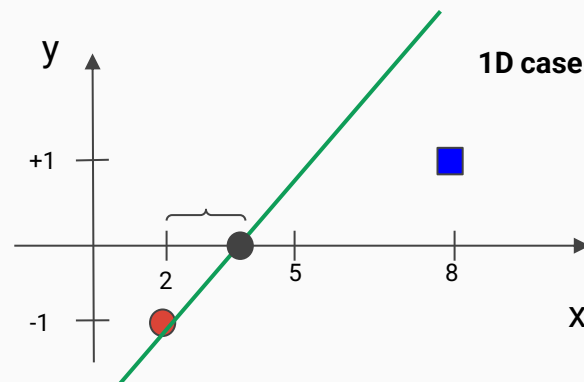
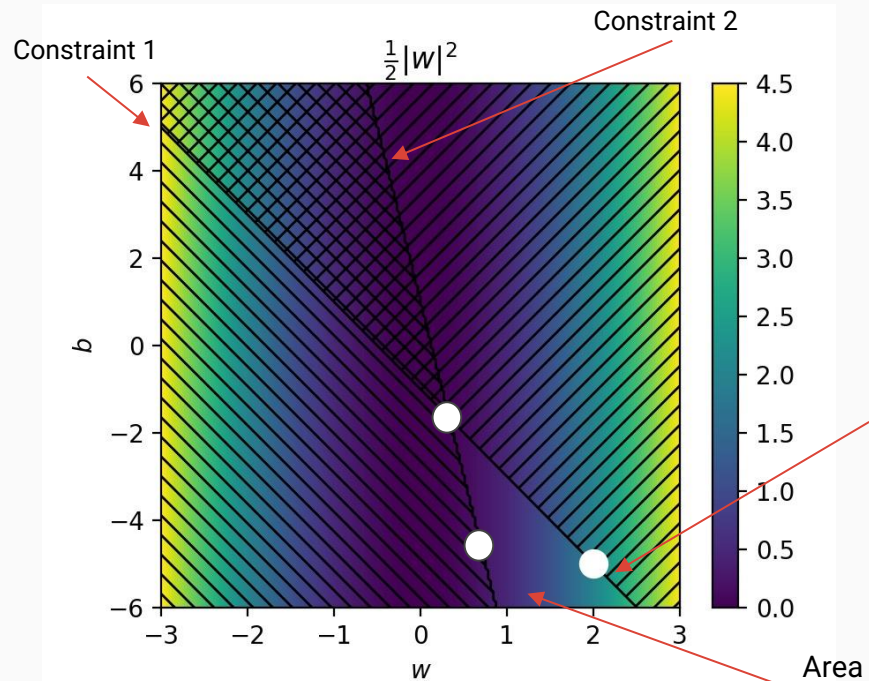
- In this case, the optimization problem becomes the following:

$$w^*, b^* = \operatorname{argmin}_{w, b} \frac{1}{2} w^2 \quad \text{such that: } \begin{cases} -2w - b \geq 1 \\ 8w + b \geq 1 \end{cases}$$



# Maximum Margin Classifier

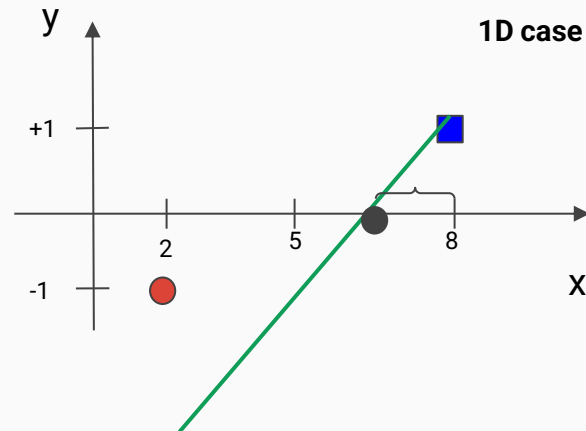
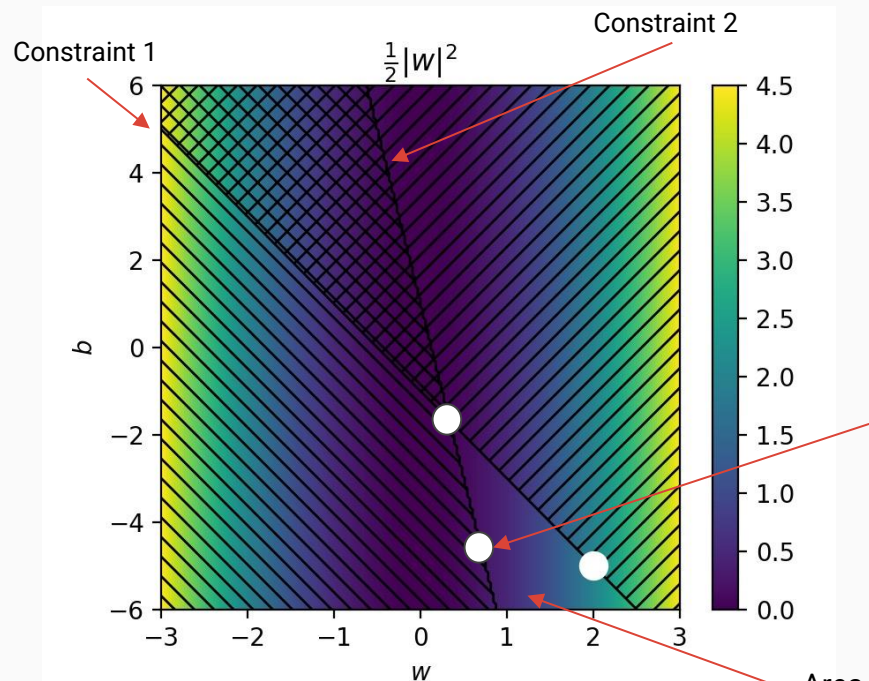
$$w^*, b^* = \operatorname{argmin}_{w, b} \frac{1}{2} w^2 \quad \text{such that:} \quad \begin{cases} -2w + b \geq 1 & \text{Constraint 1} \\ 8w + b \geq 1 & \text{Constraint 2} \end{cases}$$



Area that satisfies the constraints (perfect classification)

# Maximum Margin Classifier

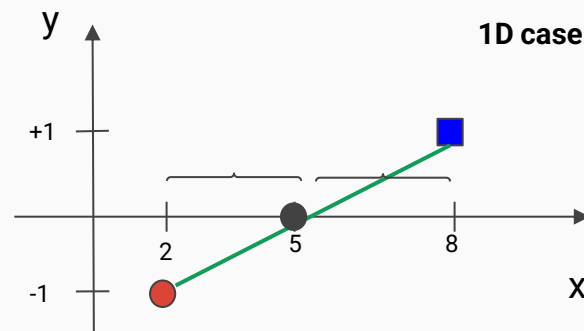
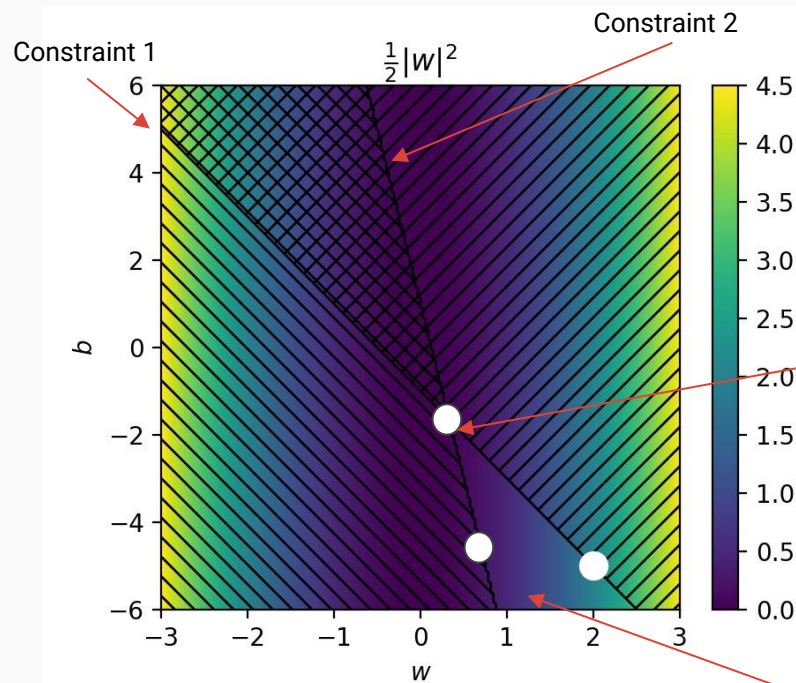
$$w^*, b^* = \operatorname{argmin}_{w, b} \frac{1}{2} w^2 \quad \text{such that:} \quad \begin{cases} -2w + b \geq 1 & \text{Constraint 1} \\ 8w + b \geq 1 & \text{Constraint 2} \end{cases}$$



Area that satisfies the constraints (perfect classification)

# Maximum Margin Classifier

$$w^*, b^* = \operatorname{argmin}_{w, b} \frac{1}{2} w^2 \quad \text{such that:} \quad \begin{cases} -2w + b \geq 1 & \text{Constraint 1} \\ 8w + b \geq 1 & \text{Constraint 2} \end{cases}$$



**Optimal solution**

Area that satisfies the constraints (perfect classification)

# Maximum Margin Classifier

$$\mathbf{w}^*, b^* = \operatorname{argmin}_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$$

such that  $t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$  for  $i = 1, \dots, N$



- The objective function is now **convex** and easy to optimize!
- This objective function is **quadratic** in the unknowns, and all of the **constraints** are **linear** in the unknowns.
- The single **global minima** can be found **efficiently** with quadratic optimization packages.

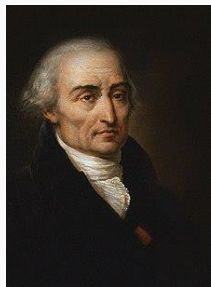
# Constrained Optimization

# Constrained Optimization

$$\mathbf{w}^*, b^* = \underset{\mathbf{w}, b}{\operatorname{argmin}} \frac{1}{2} \|\mathbf{w}\|^2$$

such that  $t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$  for  $i = 1, \dots, N$

How can we solve it?



We can use **Lagrangian multipliers!** This method can be used to find the maximum of a function  $f(\mathbf{x})$  given a constraint  $g(\mathbf{x}) = 0$ . It works in this way:

1. Define the *Lagrangian function*:

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x}) \quad \lambda \neq 0$$

2. Find the stationary points that solve this system of equations:

$$\begin{cases} \nabla_x L(\mathbf{x}, \lambda) = 0 \\ \frac{\partial L(\mathbf{x}, \lambda)}{\partial \lambda} = 0 \end{cases}$$

3. Evaluate the objective function  $f$  at these points and take the one that maximizes (or minimizes) the function (\*).

(\*) If you want to classify each point, you have to consider the second-order derivatives (in particular the determinant of the bordered Hessian Matrix). However, if the goal is to only find the global minimum/maximum this is not needed (you just choose the stationary point with the largest/smallest value of  $f$ ).

# Constrained Optimization

- For example, let's assume we want to find the **maximum of this function**.

$$f(x_1, x_2) = 1 - x_1^2 - x_2^2 \quad \text{With this constraint: } g(x_1, x_2) = x_1 + x_2 - 1 = 0$$

1. Define the **Lagrangian function**:

$$L(x_1, x_2, \lambda) = 1 - x_1^2 - x_2^2 + \lambda(x_1 + x_2 - 1)$$

2. Find the stationary point(s):

$$\begin{cases} \frac{\partial L}{\partial x_1} = -2x_1 + \lambda = 0 \\ \frac{\partial L}{\partial x_2} = -2x_2 + \lambda = 0 \\ \frac{\partial L}{\partial \lambda} = x_1 + x_2 - 1 = 0 \end{cases}$$

$$x_1 = \frac{1}{2}, x_2 = \frac{1}{2}, \lambda = 1$$

3. Evaluate the function in the stationary point(s):

$$f\left(\frac{1}{2}, \frac{1}{2}\right) = \frac{1}{2} \longrightarrow \text{Maximum}$$

# Constrained Optimization

- In our Maximum Margin Classifier, however, the constraint is not an equality but an **inequality**!
- We want to find the **maximum** of a function  $f(\mathbf{x})$  given a constraint  $g(\mathbf{x}) \geq 0$
- Similarly to the equality case, we can define a **Lagrangian function**:



$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$$

And we want to solve the following problem:

$$\max_{\mathbf{x}} \min_{\lambda \geq 0} f(\mathbf{x}) + \lambda g(\mathbf{x})$$

The Lagrange function is built smartly:

- If the **constraint holds** ( $g(\mathbf{x}) \geq 0$ ):

$$\min_{\lambda \geq 0} f(\mathbf{x}) + \lambda g(\mathbf{x}) = f(\mathbf{x})$$



The value of  $\lambda$  that minimizes the expression is  $\lambda=0$ .

If the constraint holds, we just look for the maximum of  $f(\mathbf{x})$ .



# Constrained Optimization

- In our Maximum Margin Classifier, however, the constraint is not an equality but an **inequality**!
- We want to find the **maximum** of a function  $f(\mathbf{x})$  given a constraint  $g(\mathbf{x}) \geq 0$
- Similarly to the equality case, we can define a **Lagrangian function**:



$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$$

And we want to solve the following problem:

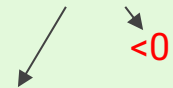
$$\max_{\mathbf{x}} \min_{\lambda \geq 0} f(\mathbf{x}) + \lambda g(\mathbf{x})$$

This way, such a solution is automatically excluded

The Lagrange function is built smartly:

- If the **constraint does not hold** ( $g(\mathbf{x}) < 0$ ):

$$\min_{\lambda \geq 0} f(\mathbf{x}) + \lambda g(\mathbf{x}) = -\infty$$



The value of  $\lambda$  that minimizes the expression is  $\lambda = +\infty$ .

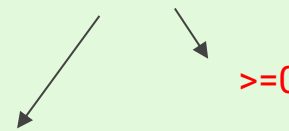
If the constraint does not hold, we replace the value of  $f(\mathbf{x})$  with the minimum possible value ( $-\infty$ ).

# Constrained Optimization

- What about if we want to minimize  $f(\mathbf{x})$  such that  $g(\mathbf{x}) \geq 0$ ?
- We have to solve the following problem:

$$\min_{\mathbf{x}} \max_{\lambda \geq 0} f(\mathbf{x}) - \lambda g(\mathbf{x})$$

- If the **constraint holds** ( $g(\mathbf{x}) \geq 0$ ):

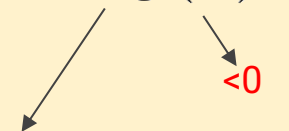
$$\max_{\lambda \geq 0} f(\mathbf{x}) - \lambda g(\mathbf{x}) = f(\mathbf{x})$$


$\geq 0$

The value of  $\lambda$  that maximizes the expression is  $\lambda=0$ .

If the constraint holds, we just look for the minimum of  $f(\mathbf{x})$ .

- If the **constraint does not hold** ( $g(\mathbf{x}) < 0$ ):

$$\max_{\lambda \geq 0} f(\mathbf{x}) - \lambda g(\mathbf{x}) = +\infty$$


$< 0$

The value of  $\lambda$  that maximizes the expression is  $\lambda=\infty$ .

If the constraint does not hold, we replace the value of  $f(\mathbf{x})$  with the max possible value ( $\infty$ ).

# Constrained Optimization

- We can solve this problem with the **Karush-Kuhn-Tucker (KKT)** constraints:

1. Define the *Lagrangian function*:

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

2. Find the stationary point(s):

$$\begin{cases} \nabla_x L(\mathbf{x}, \lambda) = 0 \\ \lambda g(\mathbf{x}) = 0 \\ g(\mathbf{x}) \geq 0 \\ \lambda \geq 0 \end{cases} \longrightarrow \text{KKT Conditions}$$

3. Evaluate the objective function  $f$  at these points and take the one that maximizes (or minimizes) the function (\*).

# Constrained Optimization

- What about if we want to minimize  $f$  with **many constraints**?

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) - \sum_{j=1}^J \lambda_j g_j(\mathbf{x})$$

We have a Lagrange multiplier for each constraint

KKT conditions:

$$\begin{cases} \nabla_x L(\mathbf{x}, \lambda) = 0 \\ \lambda_j g_j(\mathbf{x}) = 0 \text{ for } j=1, \dots, J \\ g_j(\mathbf{x}) \geq 0 \text{ for } j=1, \dots, J \\ \lambda_j \geq 0 \text{ for } j=1, \dots, J \end{cases}$$

Conditions must be valid for all the constraints

# Constrained Optimization

$$\text{maximize } 1 - x_1^2 - x_2^2 \quad \text{Such that } x_1 + x_2 - 2 > 0$$

1. Define the **Lagrangian function**:

$$L(x_1, x_2, \lambda) = 1 - x_1^2 - x_2^2 + \lambda(x_1 + x_2 - 2)$$

2. Find the stationary point(s) with KKT conditions

$$\left\{ \begin{array}{l} \frac{\partial L}{\partial x_1} = -2x_1 + \lambda = 0 \\ \frac{\partial L}{\partial x_2} = -2x_2 + \lambda = 0 \\ \lambda(x_1 + x_2 - 2) = 0 \\ x_1 + x_2 - 2 \geq 0 \\ \lambda \geq 0 \end{array} \right. \rightarrow \left\{ \begin{array}{l} x_1 = x_2 \\ \lambda = 2x_1 \\ \left\{ \begin{array}{l} x_1 = 0, x_2 = 0, \lambda = 0 \\ x_1 = 1, x_2 = 1, \lambda = 2 \end{array} \right. \end{array} \right. \Rightarrow f(1, 1) = -1$$

# Constrained Optimization

1. Define the **Lagrangian function**:

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N a_i [t_i (\mathbf{w}^T \mathbf{x}_i + b) - 1]$$

How can we solve it?

2. Find the stationary point(s) with KKT conditions

$$\left\{ \begin{array}{ll} \nabla_{\mathbf{w}} L = \mathbf{w} - \sum_{i=1}^N a_i t_i \mathbf{x}_i = 0 & \longrightarrow \text{Linear equation} \\ \frac{\partial L}{\partial b} = - \sum_{i=1}^N a_i t_i = 0 & \longrightarrow \text{Linear equation} \\ a_i [t_i (\mathbf{w}^T \mathbf{x}_i + b) - 1] = 0 \text{ for } i = 1, \dots, N & \longrightarrow \text{Non-Linear equation (we multiply the unknown } \mathbf{a} \text{ and } \mathbf{w}) \\ a_i \geq 0 \text{ for } i = 1, \dots, N & \longrightarrow \text{Linear inequality} \\ t_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 \geq 0 \text{ for } i = 1, \dots, N & \longrightarrow \text{Linear inequality} \end{array} \right.$$

↘ If  $\mathbf{x}_i$  is not a support vector ( $t_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 > 0$ )  $a_i$  must be zero to satisfy the constraint

# Constrained Optimization

1. Define the **Lagrangian function**:

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N a_i [t_i (\mathbf{w}^T \mathbf{x}_i + b) - 1]$$

2. Find the stationary point(s) with KKT conditions

$$\begin{cases} \nabla_{\mathbf{w}} L = \mathbf{w} - \sum_{i=1}^N a_i t_i \mathbf{x}_i = 0 \\ \frac{\partial L}{\partial b} = - \sum_{i=1}^N a_i t_i = 0 \\ a_i [t_i (\mathbf{w}^T \mathbf{x}_i + b) - 1] = 0 \text{ for } i = 1, \dots, N \\ a_i \geq 0 \text{ for } i = 1, \dots, N \\ t_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 \geq 0 \text{ for } i = 1, \dots, N \end{cases}$$

- We **efficiently** solve this **system** of “mostly linear” **equations** with **quadratic programming**.
- Moreover, due to the quadratic nature of the problem, we only have a single stationary point which is a **global minimum**.

# Constrained Optimization

- The direct solution of the problem is often infeasible due to the amount of computing and **memory requirement**.
- There are many efficient approaches for solving this problem. Most of them are based on breaking down the “*big*” problem into many “*smaller*” ones.
- Examples are *Chunking* (Vapnik, 1982), *Decomposition Methods* (Osuna, 1996), *Sequential Minimal Optimization* (Platt, 1999)

## Sequential Minimal Optimization:

### A Fast Algorithm for Training Support Vector Machines

John C. Platt  
Microsoft Research  
jplatt@microsoft.com  
Technical Report MSR-TR-98-14  
April 21, 1998  
© 1998 John Platt

#### ABSTRACT

This paper proposes a new algorithm for training support vector machines: *Sequential Minimal Optimization*, or *SMO*. Training a support vector machine requires the solution of a very large quadratic programming (QP) optimization problem. SMO breaks this large QP problem into a series of smallest possible QP problems. These small QP problems are solved analytically, which avoids using a time-consuming numerical QP optimization as an inner loop. The amount of memory required for SMO is linear in the training set size, which allows SMO to handle very large training sets. Because matrix computation is avoided, SMO scales somewhere between linear and quadratic in the training set size for various test problems, while the standard chunking SVM algorithm scales somewhere between linear and cubic in the training set size. SMO's computation time is dominated by SVM evaluation, hence SMO is fastest for linear SVMs and sparse data sets. On real-world sparse data sets, SMO can be more than 1000 times faster than the chunking algorithm.



# Constrained Optimization

- There are many efficient toolkits with different programming languages to solve this problem:

Solvers and scripting (programming) languages [\[ edit \]](#)

Name	Brief info
AIMMS	A software system for modeling and solving optimization and scheduling-type problems
ALGLIB	Dual licensed (GPL/proprietary) numerical library (C++, .NET).
AMPL	A popular modeling language for large-scale mathematical optimization.
APMonitor	Modeling and optimization suite for LP, QP, NLP, MILP, MINLP, and DAE systems in MATLAB and Python.
Artelys Knitro	An Integrated Package for Nonlinear Optimization
CGAL	An open source computational geometry package which includes a quadratic programming solver.
CPLEX	Popular solver with an API (C, C++, Java, .Net, Python, Matlab and R). Free for academics.
Excel Solver Function	A nonlinear solver adjusted to spreadsheets in which function evaluations are based on the recalculating cells. Basic version available as a standard add-on for Excel.
GAMS	A high-level modeling system for mathematical optimization
GNU Octave	A free (its licence is GPLv3) general-purpose and matrix-oriented programming-language for numerical computing, similar to MATLAB. Quadratic programming in GNU Octave is available via its <code>qp</code> command
IMSL	A set of mathematical and statistical functions that programmers can embed into their software applications.
IPOPT	IPOPT (Interior Point OPTimizer) is a software package for large-scale nonlinear optimization.
Maple	General-purpose programming language for mathematics. Solving a quadratic problem in Maple is accomplished via its <code>QPSolve</code> command.
MATLAB	A general-purpose and matrix-oriented programming-language for numerical computing. Quadratic programming in MATLAB requires the Optimization Toolbox in addition to the base MATLAB product
Mathematica	A general-purpose programming-language for mathematics, including symbolic and numerical capabilities.
MOSEK	A solver for large scale optimization with API for several languages (C++, Java, .Net, Matlab and Python).
NAG Numerical Library	A collection of mathematical and statistical routines developed by the <a href="#">Numerical Algorithms Group</a> for multiple programming languages (C, C++, Fortran, Visual Basic, Java and C#) and packages (MATLAB, Excel, R, LabVIEW). The Optimization chapter of the NAG Library includes routines for quadratic programming problems with both sparse and non-sparse linear constraint matrices, together with routines for the optimization of linear, nonlinear, sums of squares of linear or nonlinear functions with nonlinear, bounded or no constraints. The NAG Library has routines for both local and global optimization, and for continuous or integer problems.
Python	High-level programming language with bindings for most available solvers. Quadratic programming is available via the <code>solve_qp</code> function or by calling a specific solver directly.
R (Fortran)	GPL licensed universal cross-platform statistical computation framework.
SAS/OR	A suite of solvers for Linear, Integer, Nonlinear, Derivative-Free, Network, Combinatorial and Constraint Optimization; the <a href="#">Algebraic modeling language</a> OPTMODEL; and a variety of vertical solutions aimed at specific problems/markets, all of which are fully integrated with the <a href="#">SAS System</a> .
SuanShu	an open-source suite of optimization algorithms to solve LP, QP, SOCP, SDP, SQP in Java
TK Solver	Mathematical modeling and problem solving software system based on a declarative, rule-based language, commercialized by Universal Technical Systems, Inc..
TOMLAB	Supports global optimization, integer programming, all types of least squares, linear, quadratic and unconstrained programming for MATLAB. TOMLAB supports solvers like CPLEX, SNOPT and KNITRO.
XPRESS	Solver for large-scale linear programs, quadratic programs, general nonlinear and mixed-integer programs. Has API for several programming languages, also has a modelling language Mosel and works with AMPL, GAMS. Free for academic use.

Scikit-learn relies on

**LIBSVM**

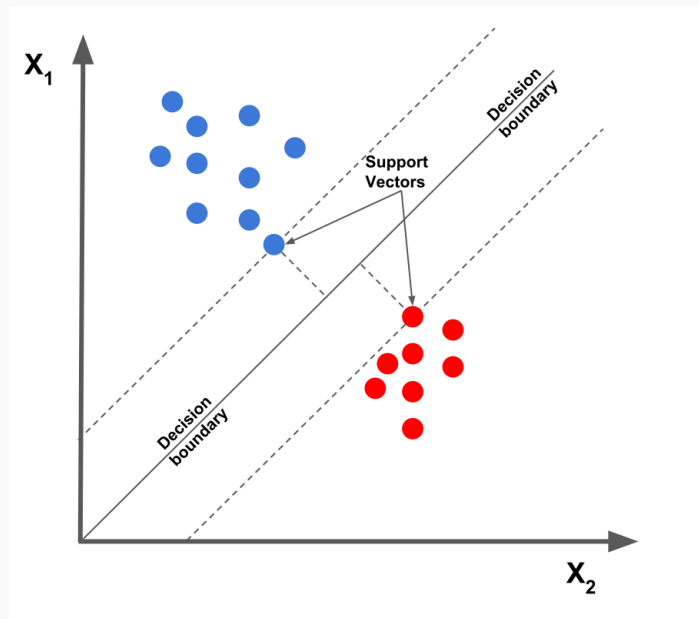


Efficient C++ Library  
for SVMs optimized  
with Sequential  
Minimal Optimization

In our tutorial, we  
will use **CVXOPT**

# Constrained Optimization

- The position of the hyperplane found by the optimizer only depends on the data points closer to the margin.



These point(s) are called **support vectors**.

Small movements of the other data points have no effect on the decision boundary.

This is in contrast to the logistic regression, in which the positions of all data points affect the location of the decision boundary,

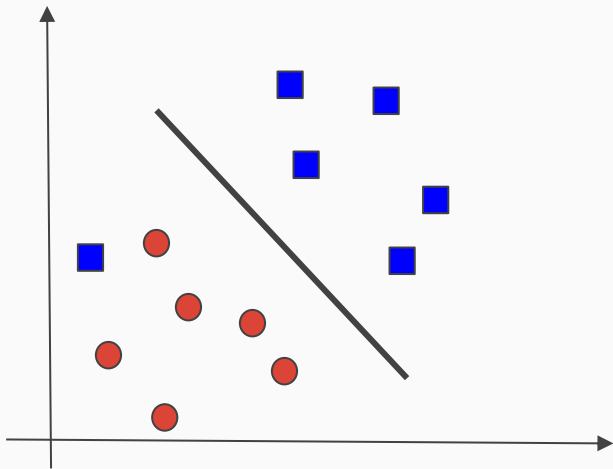
The Maximum Margin Classifier is also called **Hard Margin Linear Support Vector Machine (SVM)**.

# Soft Margin Linear SVMs

# Soft Margin SVM



Can you identify an important limitation of hard-margin linear SVM?



- The *hard-margin linear SVM* can only work with classes that are **perfectly linearly separable**.
- Even if the classes are “almost” linearly separable like in the image, the optimization fails.

**Why?**

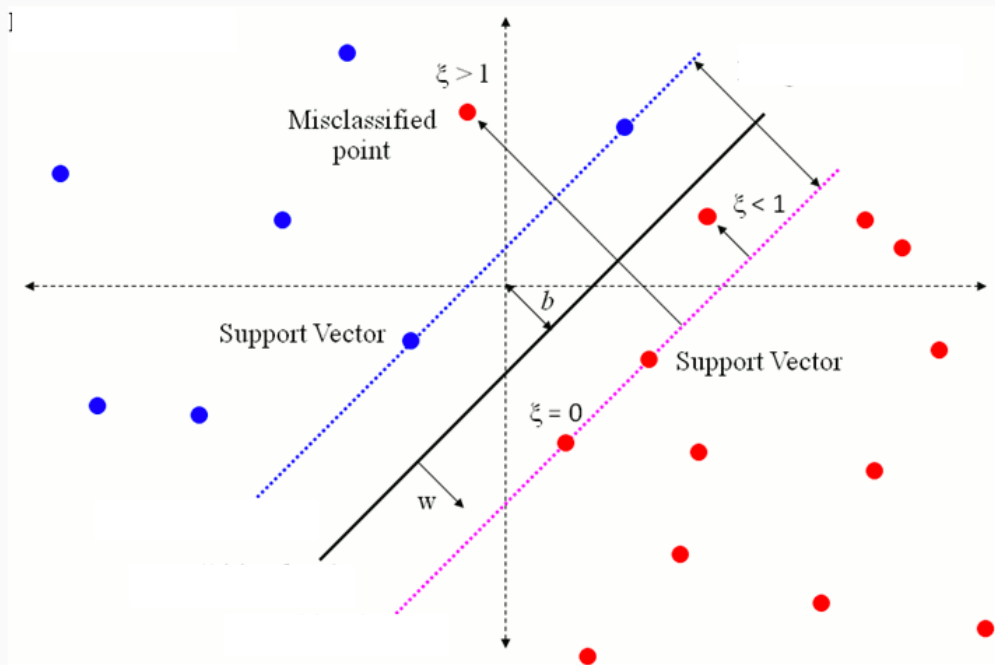
$$\mathbf{w}^*, b^* = \underset{\mathbf{w}, b}{\operatorname{argmin}} \frac{1}{2} \|\mathbf{w}\|^2$$

such that  $t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$  for  $i = 1, \dots, N$

- One misclassified point is enough to **violate this constrain!**
- In that case, a quadratic programming solver will report the problem instance as being “**infeasible**”.

# Soft Margin SVM

- We want to **relax** this hard constraint.
- We can allow data to be on the “*wrong*” side of the margin, but with a (linear) penalty that increases with the distance from the boundary.



- To do it, we introduce a **slack variable** for each sample:

$\xi_i = 0$  For data points on or inside the correct margin.

$\xi_i = |t_i - y(\mathbf{x}_i)|$  For all other points.

$$0 < \xi_i \leq 1$$

$$\xi_i \geq 1$$

Points inside the margin but on the correct side on the decision boundary.

Misclassified points

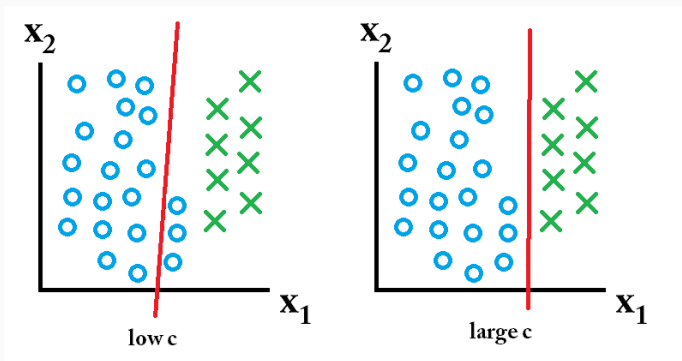
# Soft Margin SVM

- If we relax the hard constraint using the slack variable, the optimization problem becomes:

$$\mathbf{w}^*, b^*, \boldsymbol{\xi}^* = \operatorname{argmin}_{\mathbf{w}, b, \boldsymbol{\xi}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i$$

$$\text{such that } \begin{cases} t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i & \text{for } i = 1, \dots, N \\ \xi_i \geq 0 & \text{for } i = 1, \dots, N \end{cases}$$

- This term penalizes solutions containing many errors.
- It is actually an upper bound of the classification error ( $\xi_i \geq 1$  for all misclassified points).



- $C$  is a **hyperparameter** that controls the amount of penalization introduced.

Large values of  $C$   $\Rightarrow$  Little tolerance against classification errors  
 $\Rightarrow$  Small margin

Small values of  $C$   $\Rightarrow$  More tolerant against classification errors  
 $\Rightarrow$  Larger margin

# Soft Margin SVM

$$\mathbf{w}^*, b^*, \boldsymbol{\xi}^* = \operatorname{argmin}_{\mathbf{w}, b, \boldsymbol{\xi}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i$$

$$\text{such that } \begin{cases} t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i & \text{for } i = 1, \dots, N \\ \xi_i \geq 0 & \text{for } i = 1, \dots, N \end{cases}$$



We can still solve it with **quadratic programming**!

The Lagrange function is:

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \mathbf{a}, \boldsymbol{\mu}) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i - \sum_{i=1}^N a_i [t_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i] - \sum_{i=1}^N \mu_i \xi_i$$

Constraint 1

Constraint 2

KKT conditions

$$\begin{cases} \nabla_{\mathbf{w}} L = \mathbf{w} - \sum_{i=1}^N a_i t_i \mathbf{x}_i = 0 \\ \frac{\partial L}{\partial b} = - \sum_{i=1}^N a_i t_i = 0 \end{cases}$$

Constraint 1

Constraint 2

$$\begin{cases} a_i [t_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i] = 0 & \text{for } i = 1, \dots, N \\ t_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i \geq 0 & \text{for } i = 1, \dots, N \\ a_i \geq 0 & \text{for } i = 1, \dots, N \end{cases}$$

$$\begin{cases} \mu_i \xi_i = 0 & \text{for } i = 1, \dots, N \\ \xi_i \geq 0 & \text{for } i = 1, \dots, N \\ \mu_i \geq 0 & \text{for } i = 1, \dots, N \end{cases}$$

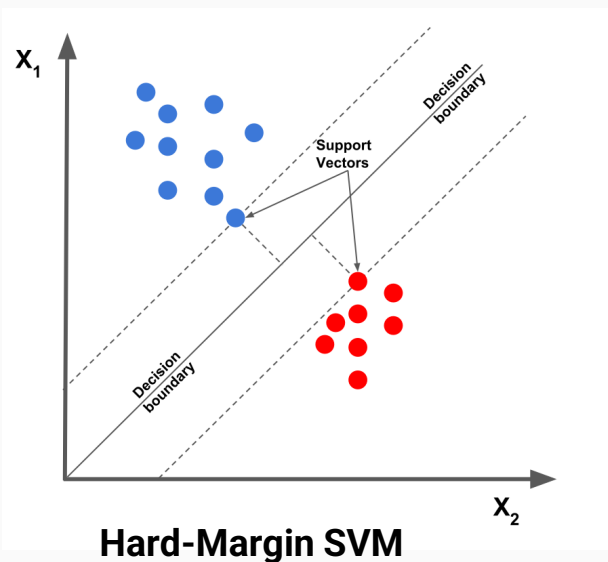
The problem is a bit **more complex** than that of hard-margin linear SVM (because we have 2 constraints here). It can be solved efficiently anyway!

# Soft Margin SVM

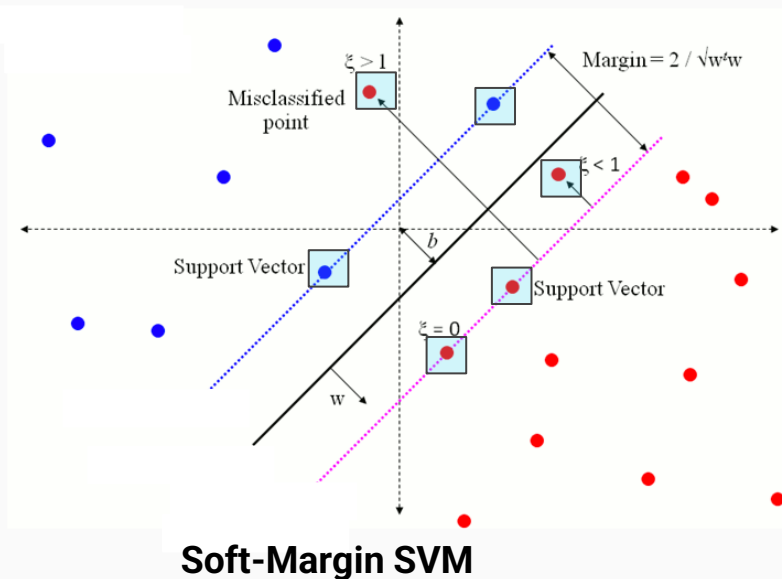


*What is a support vector in a soft margin SVM?*

- In general, **support vectors** are those points that influence the position of the **decision boundary**.



Support vectors are those points lying on the margin.



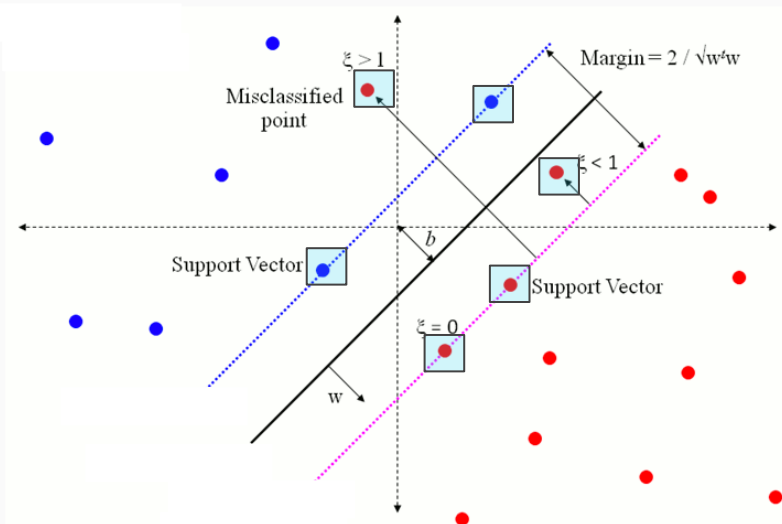
Support vectors are those points lying on the **margin** or on the **wrong side** of the margin.



# Soft Margin SVM



*What is a support vector in a soft margin SVM?*



**Soft-Margin SVM**

- The number of support vectors depends on  $C$ :

Small  $C$   $\Rightarrow$  More errors  $\Rightarrow$  More support vectors

Large  $C$   $\Rightarrow$  Fewer errors  $\Rightarrow$  Fewer support vectors

Support vectors are those points lying on the **margin** or on the **wrong side** of the margin.

# Soft Margin SVM

$$\mathbf{w}^*, b^*, \boldsymbol{\xi}^* = \underset{\mathbf{w}, b, \boldsymbol{\xi}}{\operatorname{argmin}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i$$

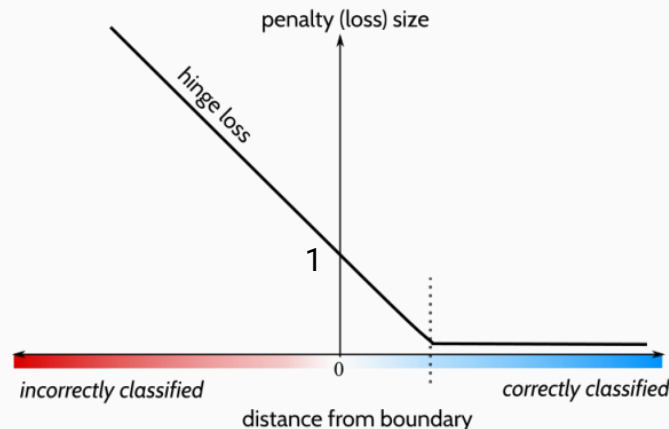
such that  $\begin{cases} t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i & \text{for } i = 1, \dots, N \\ \xi_i \geq 0 & \text{for } i = 1, \dots, N \end{cases}$

Note that the slack variable can be written as:

$$\xi_i = \max(0, 1 - t_i(\mathbf{w}^T \mathbf{x} + b))$$



This term is called **hinge loss**



# Relation to Logistic Regression

- What is the connection between a linear soft margin SVM and logistic regression?

The training loss for the **logistic regression** is the following:

$$BCE = - \sum_{i=1}^N y_i \ln \left( \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x}_i + b)}} \right) + (1 - y_i) \ln \left( 1 - \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x}_i + b)}} \right)$$

We here separated weight vector and bias as done for SVMs

This loss assumes labels **y** which are 0 or 1

- If we assume labels **t** which are -1 or +1 (as for SVMs) the binary cross-entropy becomes:

$$BCE = \sum_{i=1}^N \ln \left( 1 + e^{-t_i(\mathbf{w}^T \mathbf{x}_i + b)} \right)$$

(Feel free to double-check as an exercise)

# Relation to Logistic Regression

- If we add L2 regularization, the objective function becomes:

$$L = \sum_{i=1}^N \ln \left( 1 + e^{-t_i(\mathbf{w}^T \mathbf{x}_i + b)} \right) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

- If we multiply by a positive constant  $C=1/\lambda$  (no influence on the outcome of the optimization problem):

$$L = C \sum_{i=1}^N \ln \left( 1 + e^{-t_i(\mathbf{w}^T \mathbf{x}_i + b)} \right) + \frac{1}{2} \|\mathbf{w}\|^2$$

Logistic Regression

$$L = C \sum_{i=1}^N \max(0, 1 - t_i(\mathbf{w}^T \mathbf{x} + b)) + \frac{1}{2} \|\mathbf{w}\|^2$$

Soft-margin Linear SVM

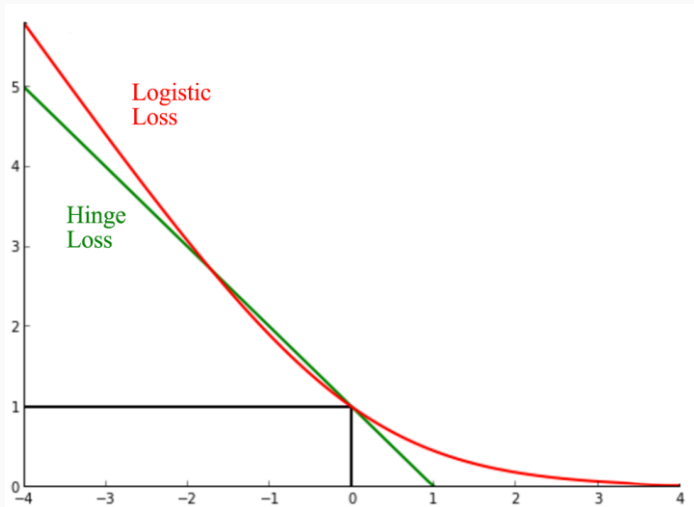
# Relation to Logistic Regression

$$L = C \sum_{i=1}^N \ln \left( 1 + e^{-t_i(\mathbf{w}^T \mathbf{x}_i + b)} \right) + \frac{1}{2} \|\mathbf{w}\|^2$$

Logistic Regression

$$L = C \sum_{i=1}^N \max(0, 1 - t_i(\mathbf{w}^T \mathbf{x} + b)) + \frac{1}{2} \|\mathbf{w}\|^2$$

Soft-margin Linear SVM



The logistic loss is a “soft” approximation of the hinge loss.

This “soft” approximation makes it easier to use gradient descent (it is a smooth and continuous loss function).

Note, however, that we can use gradient descent with the hinge loss as well (as everything is still differentiable almost everywhere).

# Relation to Logistic Regression

## Soft-margin Linear SVM

- Based on the hinge loss.
- Maximum Margin training
- Optimized with quadratic programming.
- The decision boundary only depends on the support vectors.
- No probabilities in output.
- (Slightly) less vulnerable to overfitting (due to max-margin classification that helps generalization)

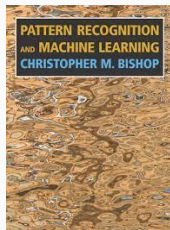
## Logistic Regression

- Based on the binary cross-entropy loss (soft approximation of the hinge loss).
- Minimum binary cross-entropy
- Optimized with gradient descent.
- The decision boundary depends on all the samples.
- Probabilities in output.
- (Slightly) more vulnerable to overfitting



Soft-margin Linear SVM and logistic regression perform similarly in practice. They work well when the classes are “almost” linearly separable.

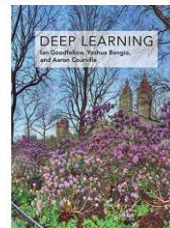
# Additional Material



## 7.1 Maximum Margin Classifiers

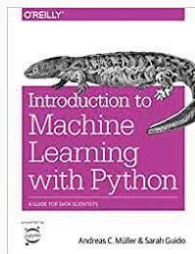
### 4.1.0 Discriminant functions

#### 4.1.1 Two classes



## 4.4 Constrained Optimization

## 5.7.2 Support Vector Machines



## Kernelized Support Vector Machines (94-195)

### Short notes from UoT:

<http://www.cs.toronto.edu/~mbrubake/teaching/C11/Handouts/SupportVectorMachines.pdf>



### ***StatQuest with Josh Starmer***

<https://youtu.be/efR1C6CvhmE>

# Lab Session

- During the weekly lab session, we will do:



Tutorial on hyperparameter tuning



Tutorial on Linear SVMs



Hyperparameter search