CMPSC 448: Machine Learning

Lecture 11. Support Vector Machines

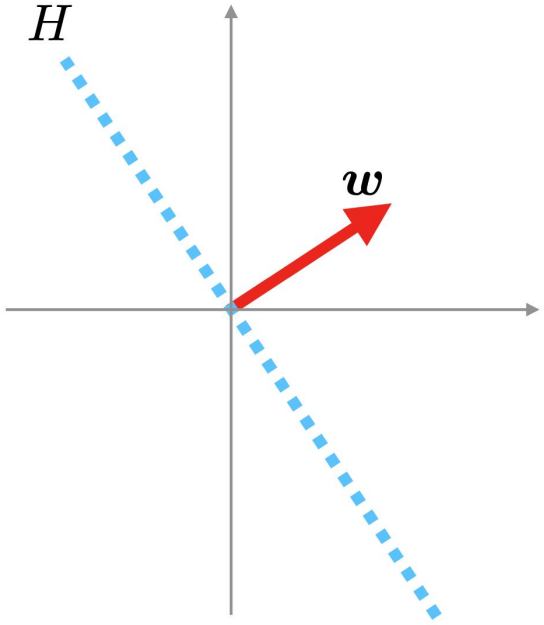
Rui Zhang

Fall 2021

Outline

- The concept of margin
- Distance of a point to a hyperplane
- The maximum margin classifier
- Hard-margin SVM for linearly separable data
- Soft-margin SVM for linearly inseparable data
- Kernelized SVM Prime-Dual and Kernel Trick

Geometry of linear models for classification



A hyperplane in $\mathbb{R}^d$ is a linear subspace:

- A $\mathbb{R}^2$-hyperplane is a line
- A $\mathbb{R}^3$-hyperplane is a plane

A hyperplane can be specified by a (nonzero) normal vector

$$\boldsymbol{w} \in \mathbb{R}^d, \|\boldsymbol{w}\|_2 = 1$$

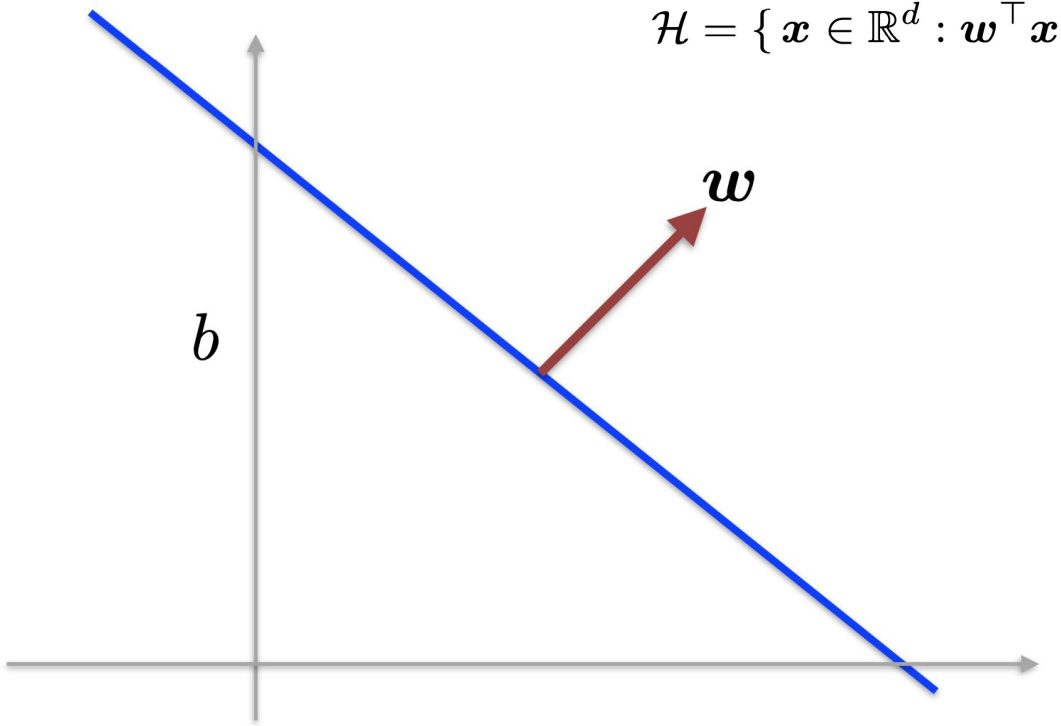
The hyperplane with normal vector can be represented by set of points orthogonal to its normal vector:

$$H = \{\boldsymbol{x} \in \mathbb{R}^d \ : \ \boldsymbol{w}^\top \boldsymbol{x} = 0\}$$

Linear models and hyperplanes

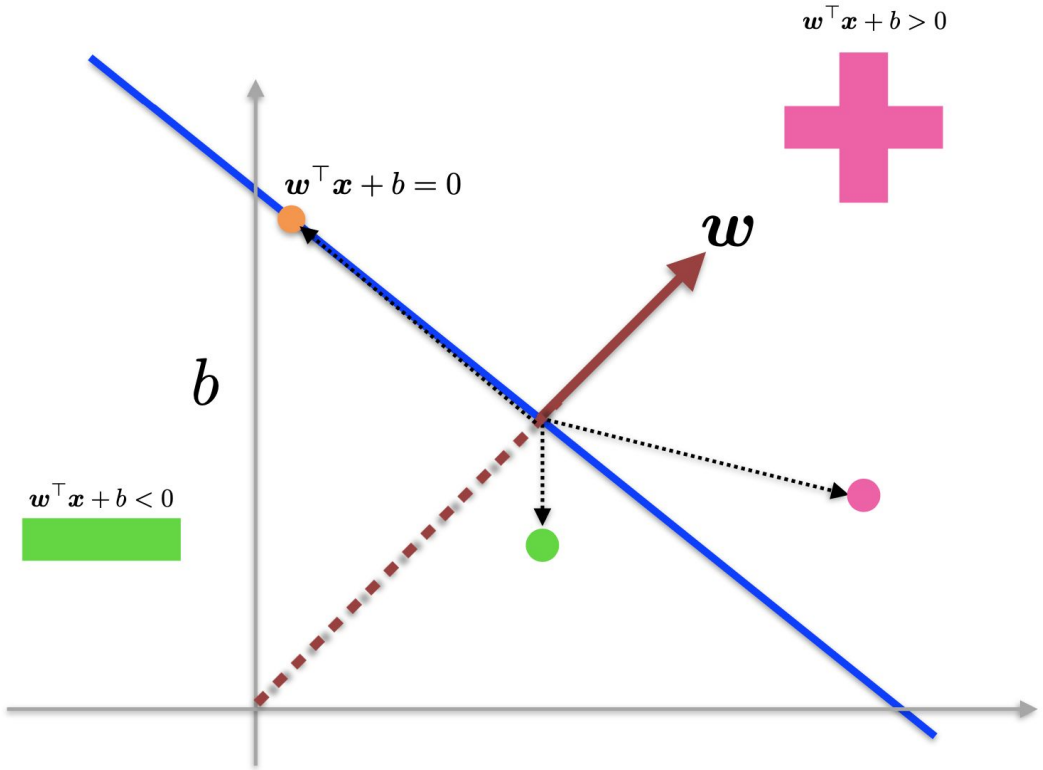
Linear classification models make decisions based on a linear combination of features:

$$\mathcal{H} = \{ \boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{w}^\top \boldsymbol{x} + b = 0 \}$$



Linear models and hyperplanes

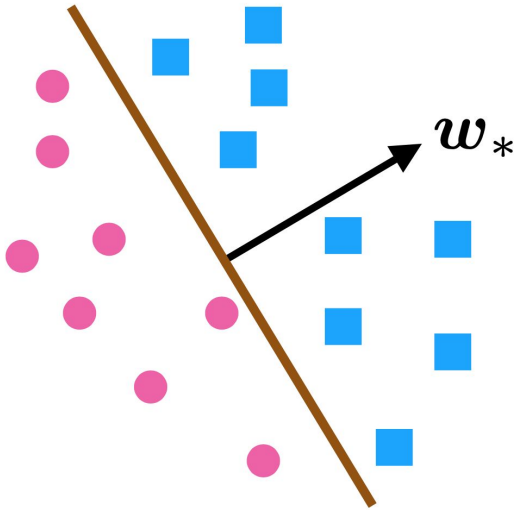
Every hyperplane is a decision boundary that splits the space into two subspaces:



Linear separability

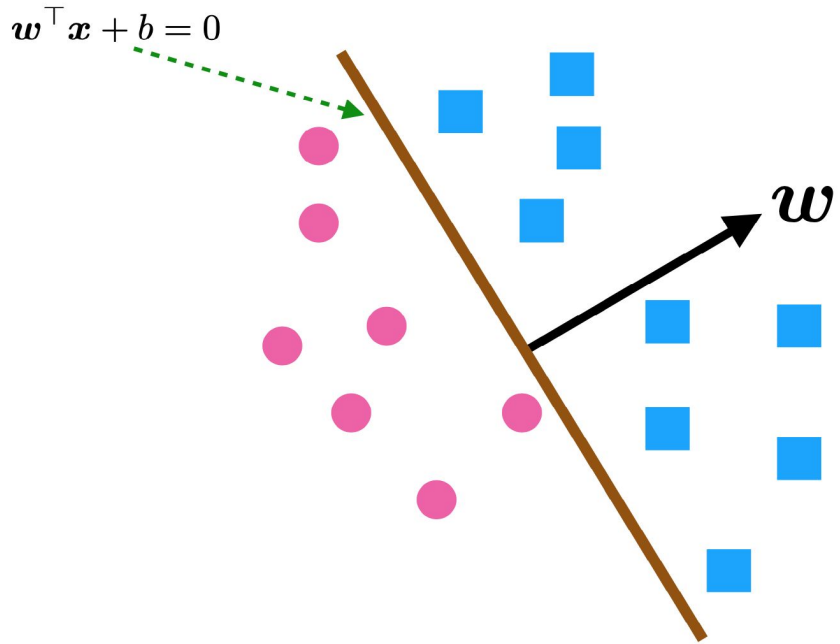
Assume there is a linear classifier that perfectly classifies the training data:

Given $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$



Perceptron as linear programming

The prediction function is: $f_{\boldsymbol{w}}(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x} + b)$



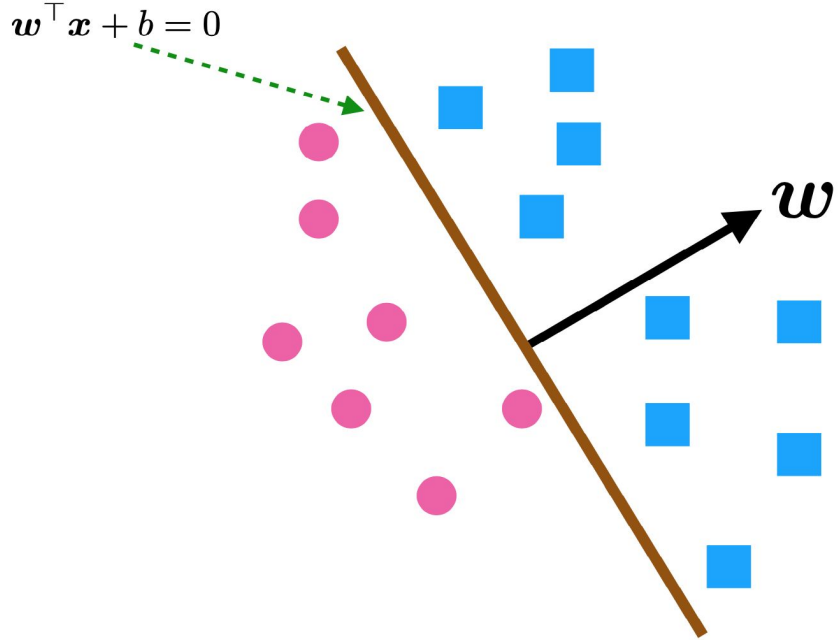
Perceptron as linear programming

The prediction function is: $f_{\boldsymbol{w}}(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x} + b)$

$\boldsymbol{w}^\top \boldsymbol{x} + b \geq 0$ **if** $y = +1$

$\boldsymbol{w}^\top \boldsymbol{x} + b < 0$ **if** $y = -1$

Or more compactly, $y\left(\boldsymbol{w}^\top \boldsymbol{x} + b\right) \geq 0$



Perceptron as linear programming

The prediction function is: $f_{\boldsymbol{w}}(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x} + b)$

$\boldsymbol{w}^\top \boldsymbol{x} + b \geq 0$ **if** $y = +1$

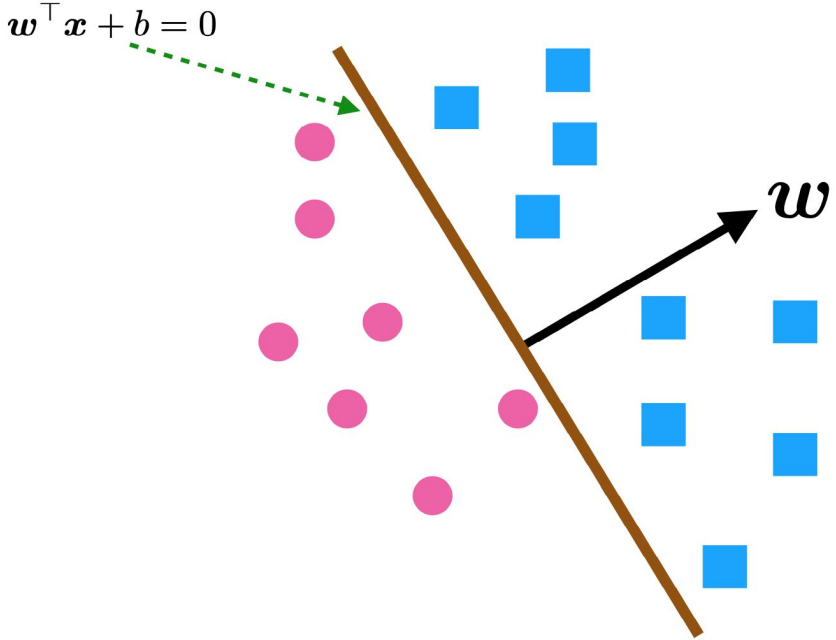
$\boldsymbol{w}^\top \boldsymbol{x} + b < 0$ **if** $y = -1$

Or more compactly, $y\left(\boldsymbol{w}^\top \boldsymbol{x} + b\right) \geq 0$

We can write the Perceptron algorithms as the following linear programming!

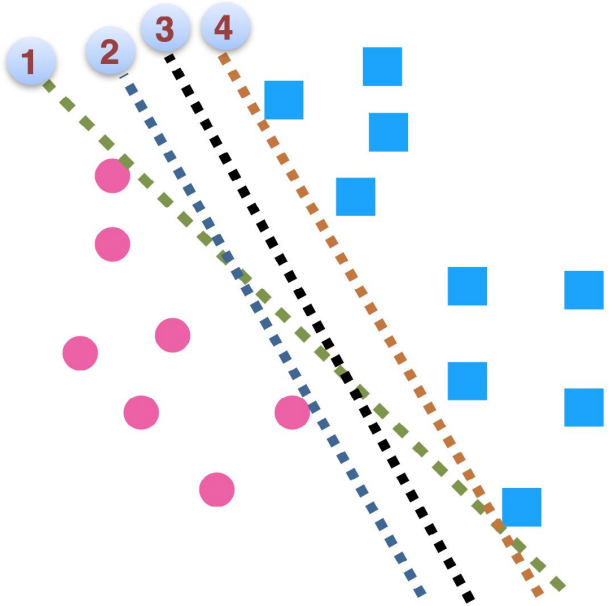
$$\max_{\boldsymbol{w}, b} \quad 0$$

$$\text{such that} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 0, \quad i = 1, 2, \ldots, n$$



Linear classifiers: which hyperplane?

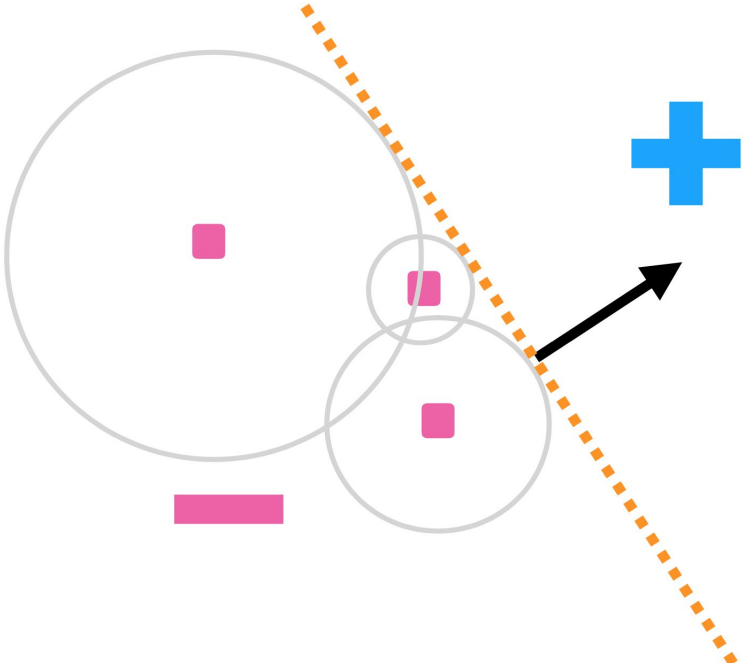
The Perceptron algorithm has infinite options to pick from:



Any of these solutions would be fine (zero training error)…

… but which one is the best (less test error)?

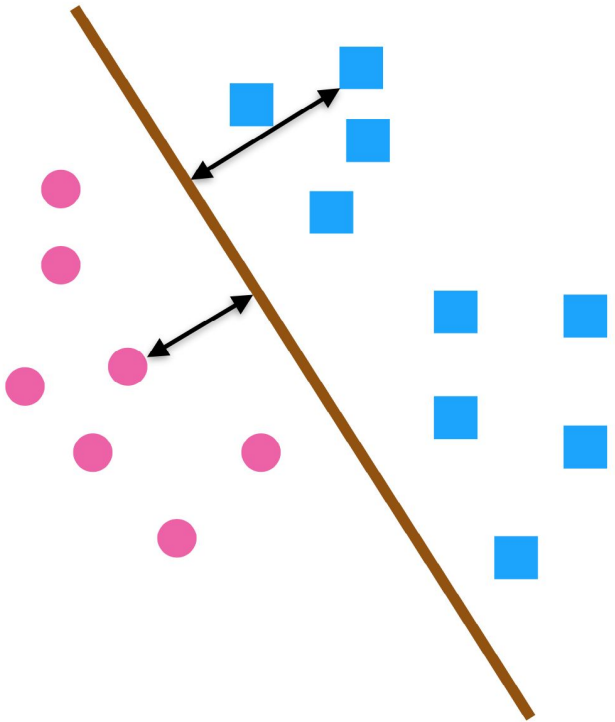
The measure of confidence



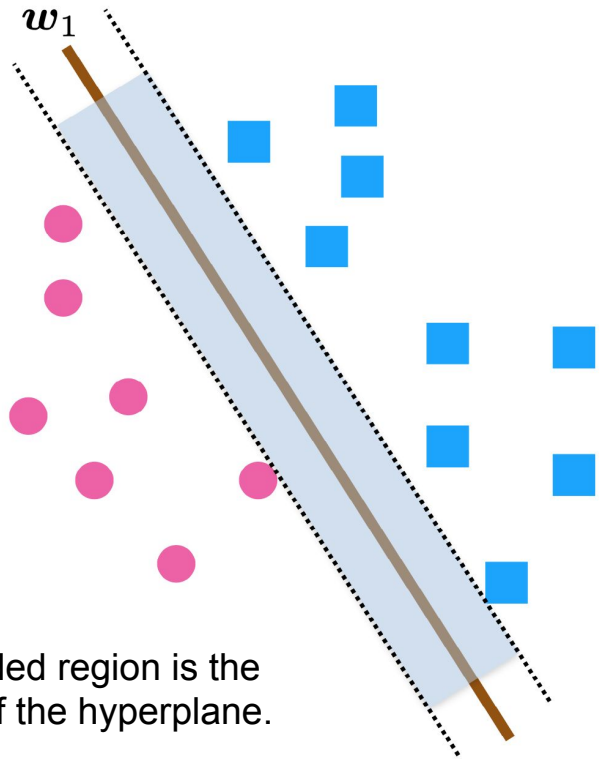
The larger the margin (distance from hyperplane), more confidence the algorithm will be in predicted label, which can be computed as

$$y(\boldsymbol{w}^\top \boldsymbol{x})$$

Classifier margin



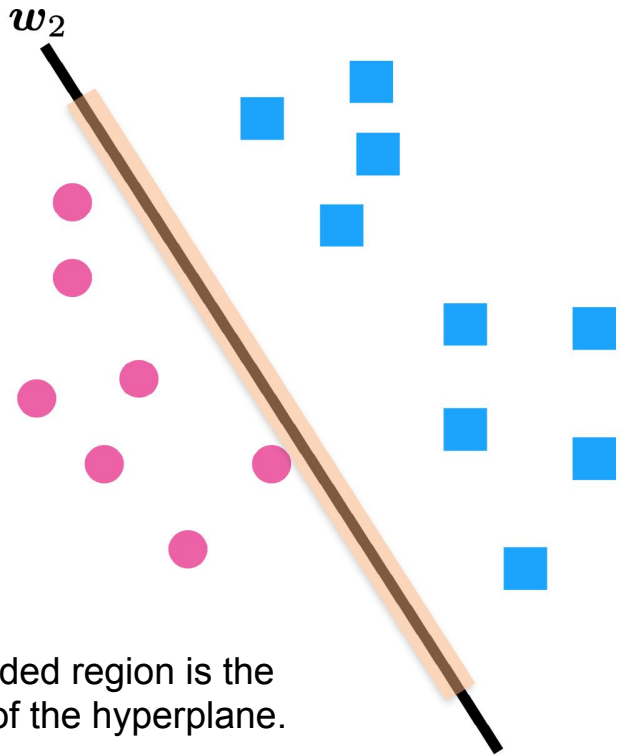
Classifier margin



The shaded region is the margin of the hyperplane.

The margin of the hyperplane for a linear classifier is the distance to the **closest** training sample.

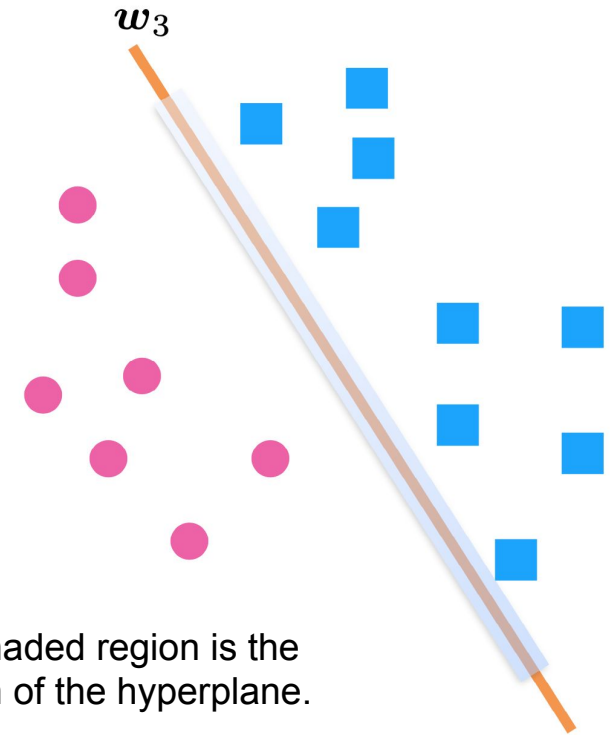
Classifier margin



The shaded region is the margin of the hyperplane.

The margin of the hyperplane for a linear classifier is the distance to the **closest** training sample.

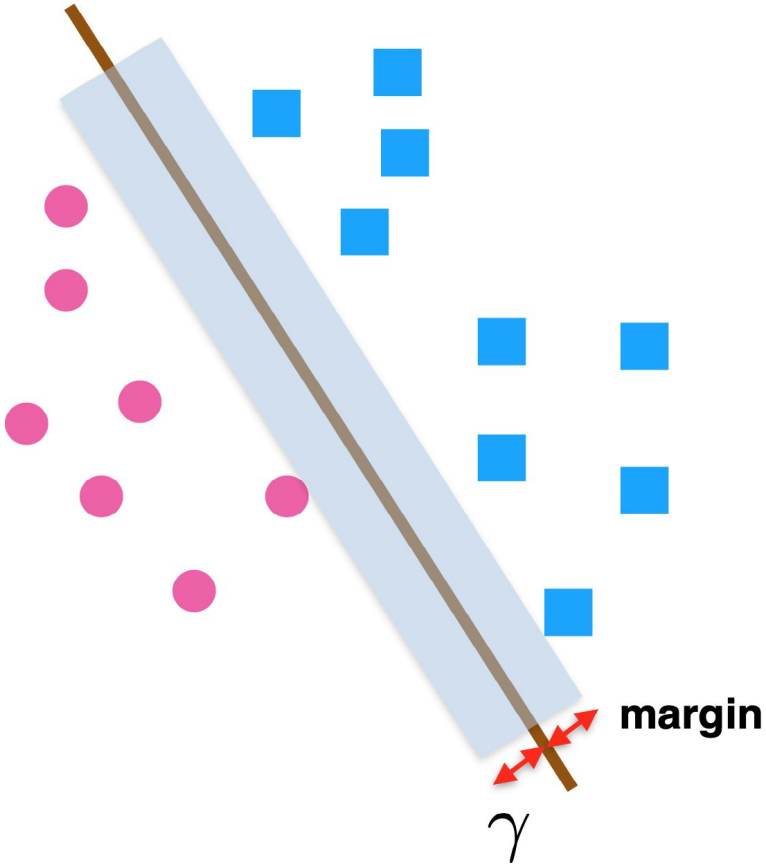
Classifier margin



The shaded region is the margin of the hyperplane.

The margin of the hyperplane for a linear classifier is the distance to the **closest** training sample.

Why maximum margin classifiers?



There is some theory (using VC dimension) that is related to (but not the same as) the proposition that this is a good thing.

Empirically it works very very well.

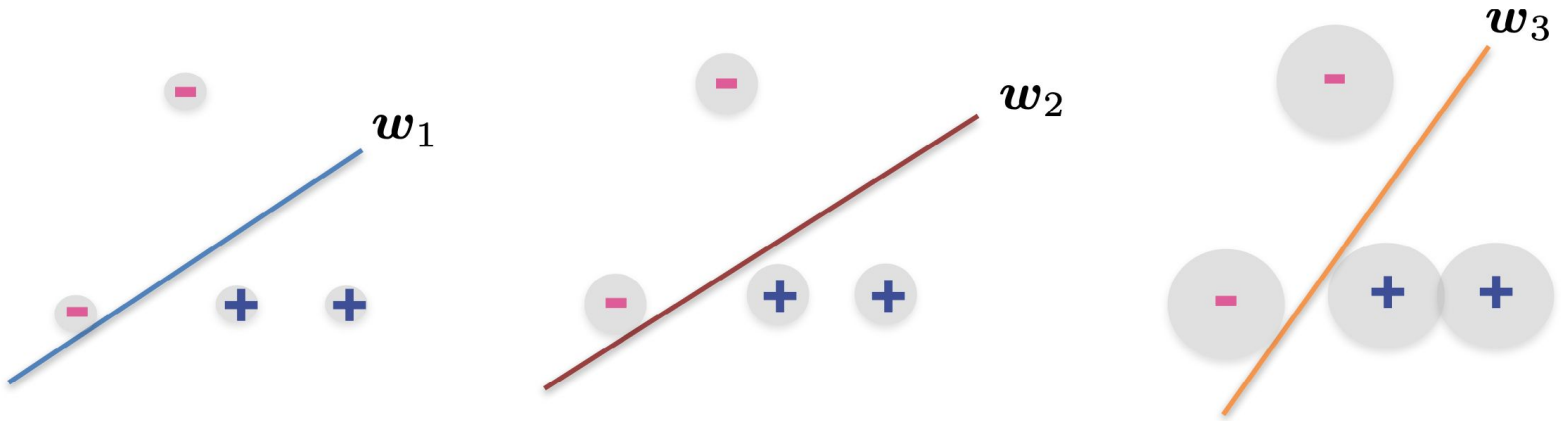
If there is noise in the location of the boundary, this gives us least chance of causing a misclassification.

Why maximum margin classifiers?

Consider the margin as the maximum amount of noise can be tolerated for all training data

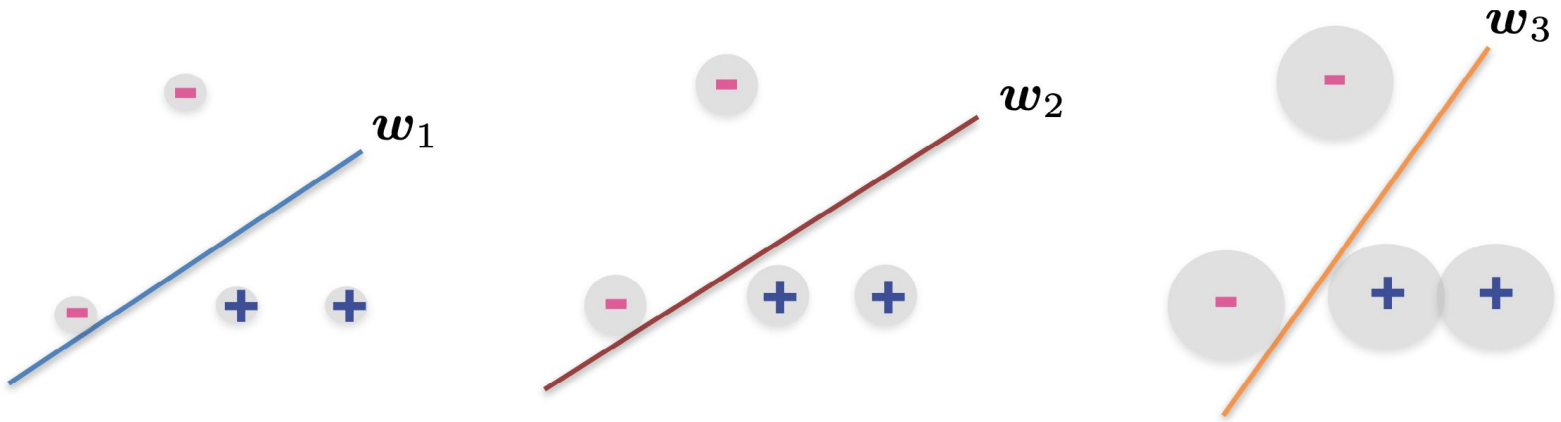
Why maximum margin classifiers?

Consider the margin as the maximum amount of noise can be tolerated for all training data



Why maximum margin classifiers?

Consider the margin as the maximum amount of noise can be tolerated for all training data

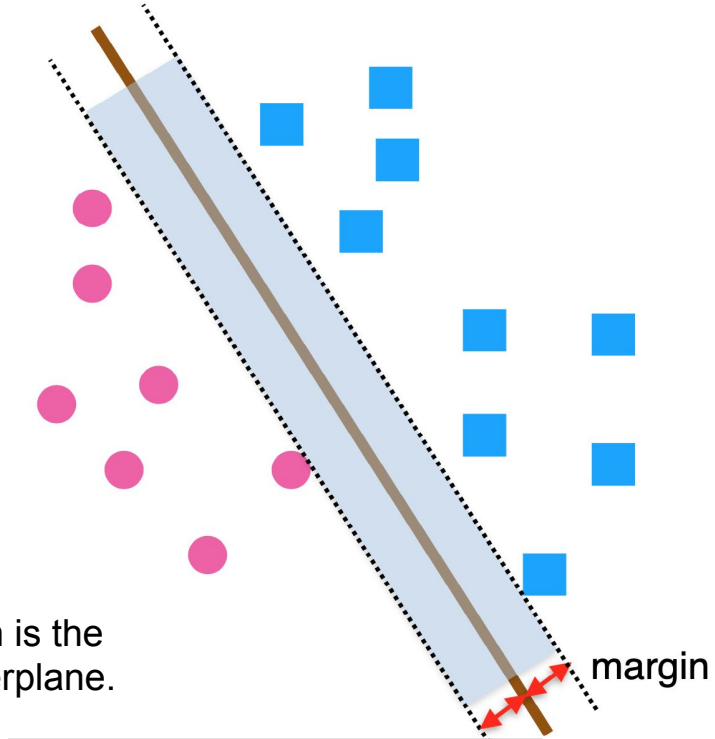


The ball around each training data shows the amount of noise that can be tolerated = the size of smallest ball for all points for a given hyperplane!

Intuition: A hyperplane that can tolerate more noise error is safer.

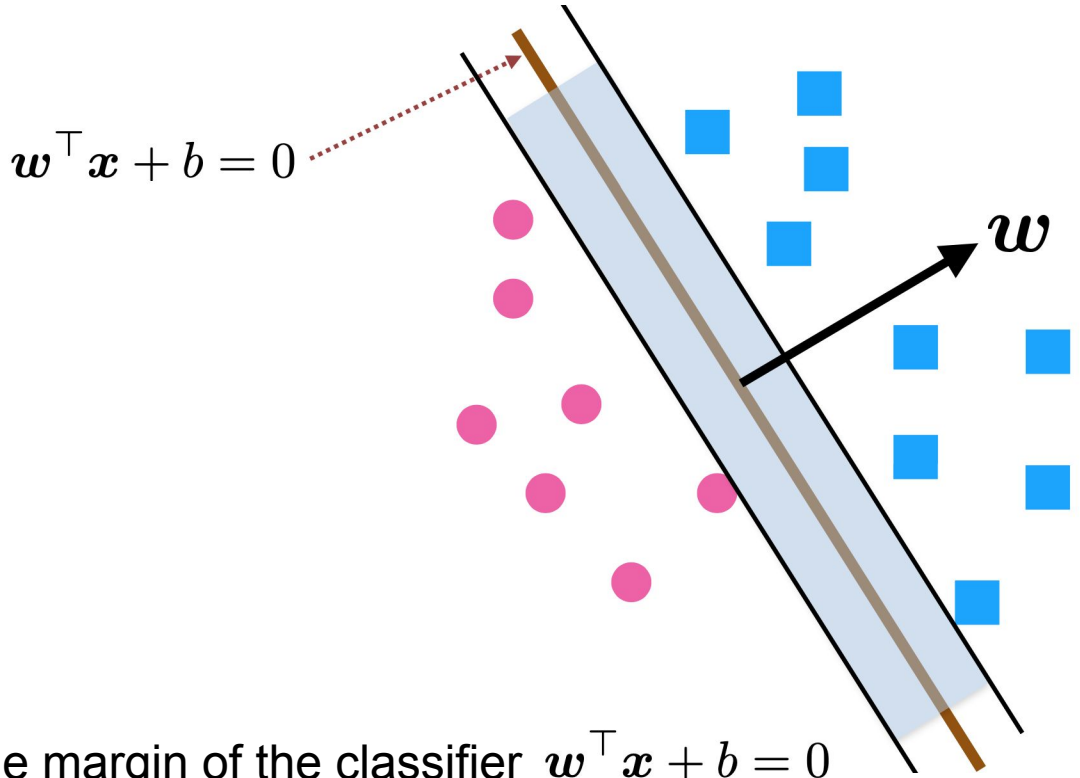
Maximum margin classifier

The **maximum margin linear classifier** is the linear classifier with the **maximum margin**
This is the simplest kind of SVM (called **linear SVM**)



The shaded region is the margin of the hyperplane.

Margin of an arbitrary hyperplane



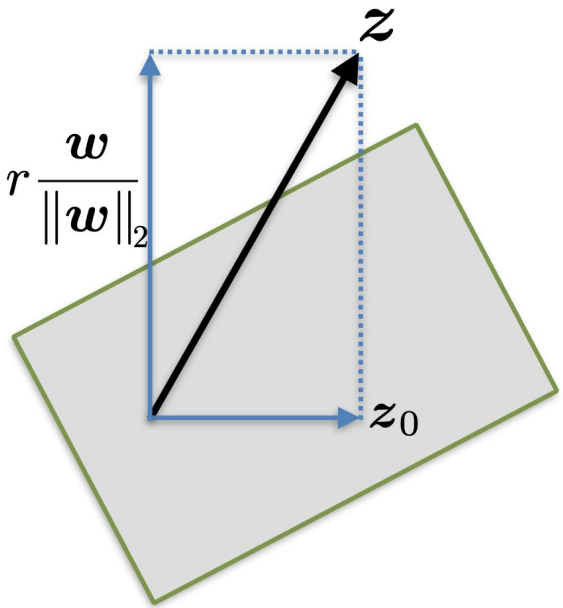
How to compute the margin of the classifier $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$

Point-plane distance

Question: how to compute the distance of a point from a hyperplane?

Consider the hyperplane $\mathcal{H} = \{\boldsymbol{x} : \boldsymbol{w}^\top \boldsymbol{x} + b = 0\}$

The distance of point $\boldsymbol{z} \in \mathbb{R}^d$ from hyperplane:



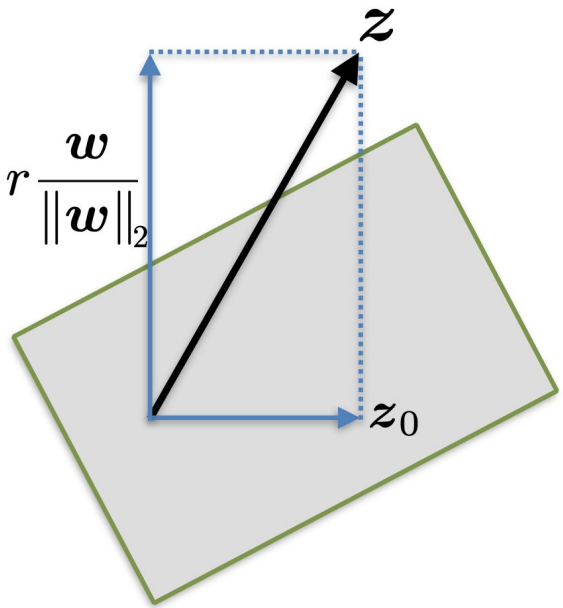
$$\boldsymbol{z} = \boldsymbol{z}_0 + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$$

Point-plane distance

Question: how to compute the distance of a point from a hyperplane?

Consider the hyperplane $\mathcal{H} = \{\boldsymbol{x} : \boldsymbol{w}^\top \boldsymbol{x} + b = 0\}$

The distance of point $\boldsymbol{z} \in \mathbb{R}^d$ from hyperplane:



$$\boldsymbol{z} = \boldsymbol{z}_0 + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$$

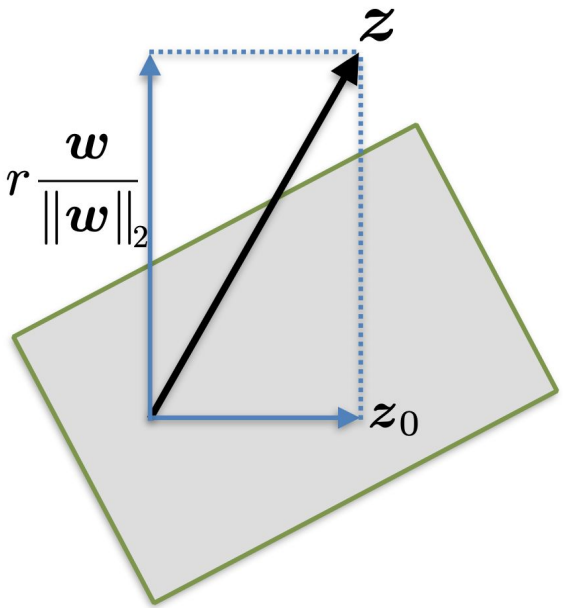
$$\boldsymbol{w}^\top \boldsymbol{z} + b = \boldsymbol{w}^\top \left(\boldsymbol{z}_0 + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}\right) + b$$

Point-plane distance

Question: how to compute the distance of a point from a hyperplane?

Consider the hyperplane $\mathcal{H} = \{\boldsymbol{x} : \boldsymbol{w}^\top \boldsymbol{x} + b = 0\}$

The distance of point $\boldsymbol{z} \in \mathbb{R}^d$ from hyperplane:



$$\boldsymbol{z} = \boldsymbol{z}_0 + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$$

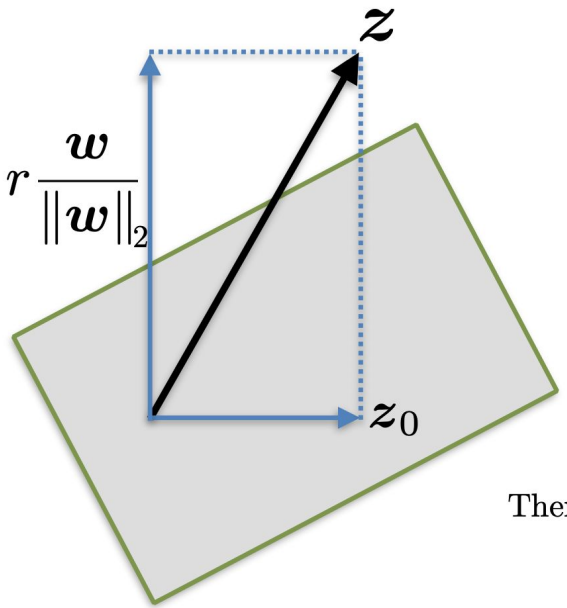
$$\begin{aligned}
\boldsymbol{w}^\top \boldsymbol{z} + b &= \boldsymbol{w}^\top \left(\boldsymbol{z}_0 + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}\right) + b \\
&= \underbrace{\boldsymbol{w}^\top \boldsymbol{z}_0 + b}_{=0} + \boldsymbol{w}^\top \left(r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}\right) \\
&= r\frac{\boldsymbol{w}^\top \boldsymbol{w}}{\|\boldsymbol{w}\|_2} \\
&= \underbrace{r\|\boldsymbol{w}\|_2}_{\text{since } \boldsymbol{w}^\top \boldsymbol{w} = \|\boldsymbol{w}\|_2^2}
\end{aligned}$$

Point-plane distance

Question: how to compute the distance of a point from a hyperplane?

Consider the hyperplane $\mathcal{H} = \{\boldsymbol{x} : \boldsymbol{w}^\top \boldsymbol{x} + b = 0\}$

The distance of point $\boldsymbol{z} \in \mathbb{R}^d$ from hyperplane:



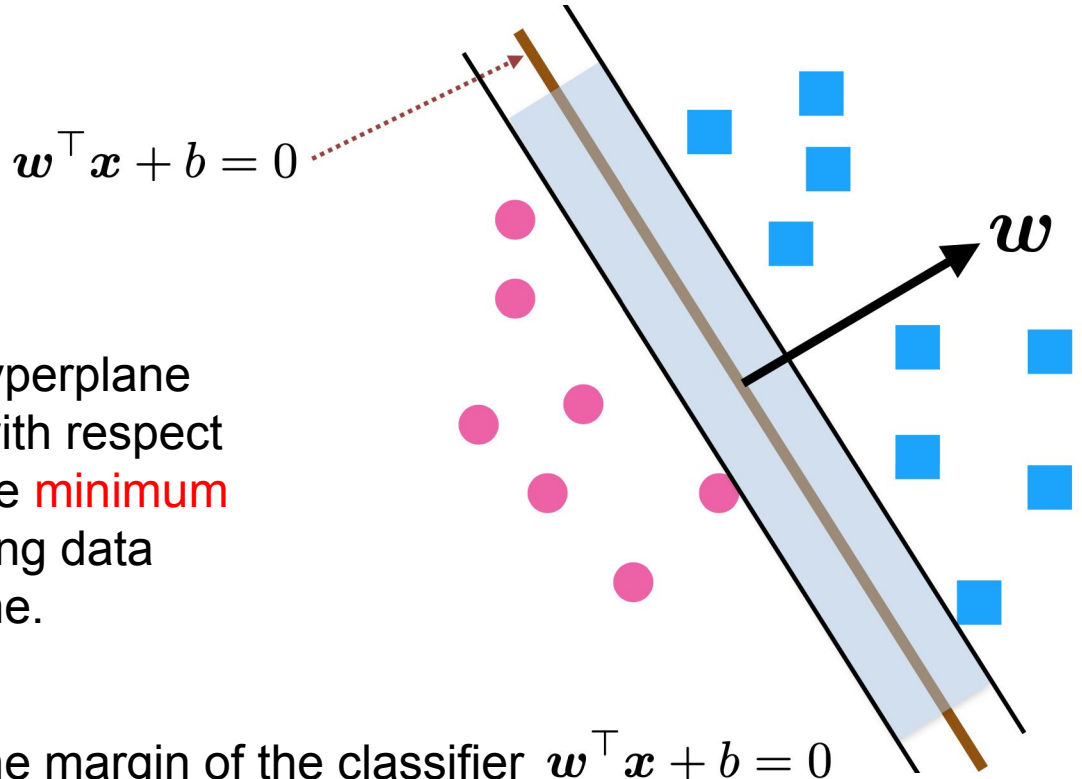
$$\boldsymbol{z} = \boldsymbol{z}_0 + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$$

$$\begin{aligned}
\boldsymbol{w}^\top \boldsymbol{z} + b &= \boldsymbol{w}^\top \left(\boldsymbol{z}_0 + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}\right) + b \\
&= \underbrace{\boldsymbol{w}^\top \boldsymbol{z}_0 + b}_{=0} + \boldsymbol{w}^\top \left(r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}\right) \\
&= r\frac{\boldsymbol{w}^\top \boldsymbol{w}}{\|\boldsymbol{w}\|_2} \\
&= \underbrace{r\|\boldsymbol{w}\|_2}_{\text{since } \boldsymbol{w}^\top \boldsymbol{w} = \|\boldsymbol{w}\|_2^2}
\end{aligned}$$

Therefore, the distance of point $\boldsymbol{z}$ to the plane $\{\boldsymbol{x} : \boldsymbol{w}^\top \boldsymbol{x} + b = 0\}$ is

$$|r| = \frac{|\boldsymbol{w}^\top \boldsymbol{z} + b|}{\|\boldsymbol{w}\|_2}$$

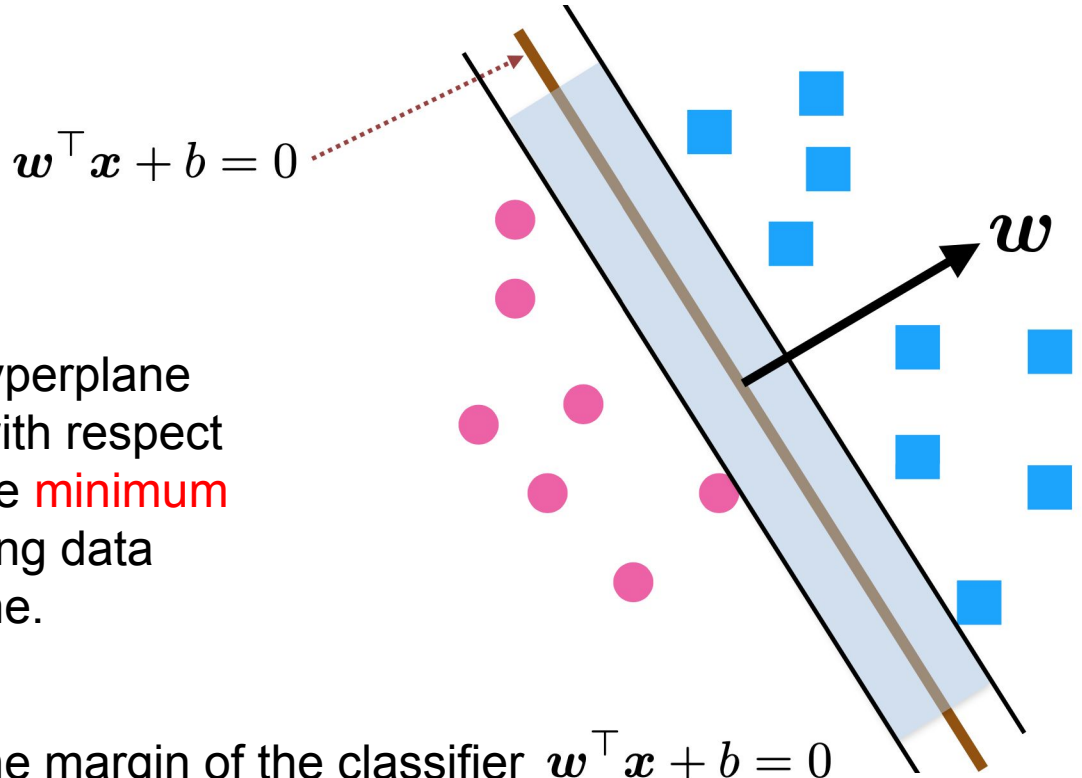
Margin of an arbitrary hyperplane



The margin of a hyperplane (linear classifier) with respect to training set is the minimum distance of a training data from the hyperplane.

How to compute the margin of the classifier $w^\top x + b = 0$

Margin of an arbitrary hyperplane



The margin of a hyperplane (linear classifier) with respect to training set is the minimum distance of a training data from the hyperplane.

How to compute the margin of the classifier $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$

$$\text{margin}(\boldsymbol{w}, b, \mathcal{S}) = \min_{\boldsymbol{x}_i \in \mathcal{S}} d(\boldsymbol{x}_i, \boldsymbol{w}, b)$$

Maximum margin classifier

We are interested in finding a linear classifier with normal vector $\boldsymbol{w}$ and intercept (bias) b that maximizes the margin!

$$\max_{\boldsymbol{w},b} \left[\text{margin}(\boldsymbol{w}, b, \mathcal{S})\right]$$

Maximum margin classifier

We are interested in finding a linear classifier with normal vector $\boldsymbol{w}$ and intercept (bias) b that maximizes the margin!

$$\max_{\boldsymbol{w},b} [\text{margin}(\boldsymbol{w}, b, \mathcal{S})]$$

$$= \max_{\boldsymbol{w},b} \min_{\boldsymbol{x}_i \in \mathcal{S}} d(\boldsymbol{x}_i, \boldsymbol{w}, b)$$

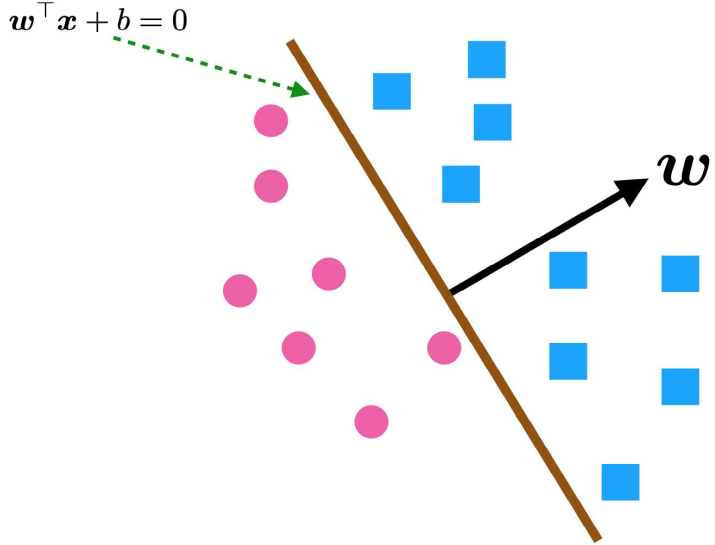
$$= \max_{\boldsymbol{w},b} \min_{\boldsymbol{x}_i \in \mathcal{S}} \frac{|\boldsymbol{w}^\top \boldsymbol{x}_i + b|}{\|\boldsymbol{w}\|_2}$$

Maximize the minimum distance of points to a given hyperplane.

Maximum margin classifier

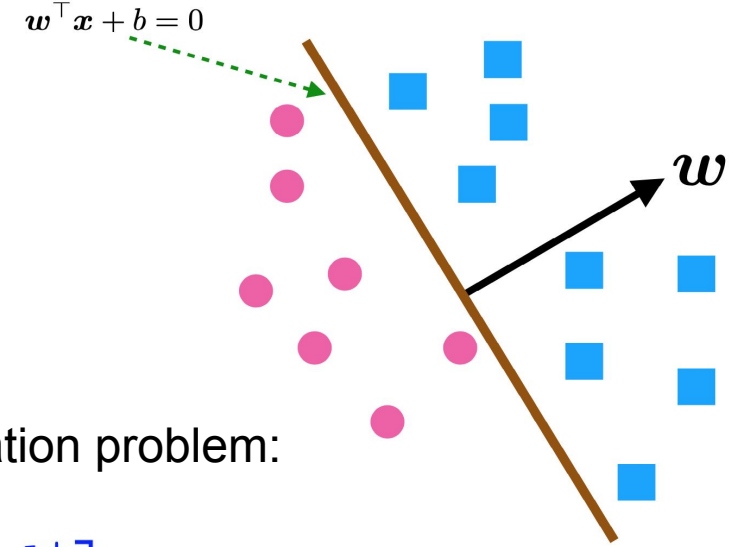
Find a hyperplane that

1. perfectly classifies training data



$$y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 0,\ i = 1, 2, \ldots, n$$

Maximum margin classifier



Find a hyperplane that

1. perfectly classifies training data
2. maximizes the margin.

This can be found by solving the following optimization problem:

$$\max_{\boldsymbol{w},b} \underbrace{\left[\min_{\boldsymbol{x}_i \in \mathcal{S}} \frac{|\boldsymbol{w}^\top \boldsymbol{x}_i + b|}{\|\boldsymbol{w}\|_2}\right]}_{\text{minimum distance from points to } \boldsymbol{w}}$$

$$\text{s.t} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 0, \; i = 1, 2, \ldots, n$$

Let's try to simplify this optimization...

If $(\boldsymbol{w}, b)$is a solution, then if we rescale it to $(\alpha\boldsymbol{w}, \alpha b)$, then the distance from any point to the decision surface (hyperplane) is unchanged:

$$\frac{|\alpha\boldsymbol{w}^\top \boldsymbol{x}_i + \alpha b|}{\|\alpha\boldsymbol{w}\|_2} = \frac{\alpha}{\alpha}\frac{|\boldsymbol{w}^\top \boldsymbol{x}_i + b|}{\|\boldsymbol{w}\|_2} = \frac{|\boldsymbol{w}^\top \boldsymbol{x}_i + b|}{\|\boldsymbol{w}\|_2}$$

Let's try to simplify this optimization...

If $(\boldsymbol{w}, b)$is a solution, then if we rescale it to $(\alpha\boldsymbol{w}, \alpha b)$, then the distance from any point to the decision surface (hyperplane) is unchanged:

$$\frac{|\alpha\boldsymbol{w}^\top \boldsymbol{x}_i + \alpha b|}{\|\alpha\boldsymbol{w}\|_2} = \frac{\alpha}{\alpha}\frac{|\boldsymbol{w}^\top \boldsymbol{x}_i + b|}{\|\boldsymbol{w}\|_2} = \frac{|\boldsymbol{w}^\top \boldsymbol{x}_i + b|}{\|\boldsymbol{w}\|_2}$$

Implication: We can use this freedom to make $\min_{\boldsymbol{x}_i \in \mathcal{S}} |\boldsymbol{w}^\top \boldsymbol{x}_i + b| = 1$
i.e., for the point that is closest to the surface

$$y_i \left(\boldsymbol{w}^\top \boldsymbol{x}_i + b\right) = 1$$

Let's try to simplify this optimization...

If $(\boldsymbol{w}, b)$is a solution, then if we rescale it to $(\alpha\boldsymbol{w}, \alpha b)$, then the distance from any point to the decision surface (hyperplane) is unchanged:

$$\frac{|\alpha\boldsymbol{w}^\top \boldsymbol{x}_i + \alpha b|}{\|\alpha\boldsymbol{w}\|_2} = \frac{\alpha}{\alpha}\frac{|\boldsymbol{w}^\top \boldsymbol{x}_i + b|}{\|\boldsymbol{w}\|_2} = \frac{|\boldsymbol{w}^\top \boldsymbol{x}_i + b|}{\|\boldsymbol{w}\|_2}$$

Implication: We can use this freedom to make $\min_{\boldsymbol{x}_i \in \mathcal{S}} |\boldsymbol{w}^\top \boldsymbol{x}_i + b| = 1$
i.e., for the point that is closest to the surface

$$y_i \left(\boldsymbol{w}^\top \boldsymbol{x}_i + b\right) = 1$$

In this case, all training data points will satisfy the constraints

$$y_i \left(\boldsymbol{w}^\top \boldsymbol{x}_i + b\right) \geq 1$$

Let's try to simplify this optimization...

If $(\boldsymbol{w}, b)$is a solution, then if we rescale it to $(\alpha\boldsymbol{w}, \alpha b)$, then the distance from any point to the decision surface (hyperplane) is unchanged:

$$\frac{|\alpha\boldsymbol{w}^\top \boldsymbol{x}_i + \alpha b|}{\|\alpha\boldsymbol{w}\|_2} = \frac{\alpha}{\alpha}\frac{|\boldsymbol{w}^\top \boldsymbol{x}_i + b|}{\|\boldsymbol{w}\|_2} = \frac{|\boldsymbol{w}^\top \boldsymbol{x}_i + b|}{\|\boldsymbol{w}\|_2}$$

Implication: We can use this freedom to make $\min_{\boldsymbol{x}_i \in \mathcal{S}} |\boldsymbol{w}^\top \boldsymbol{x}_i + b| = 1$
i.e., for the point that is closest to the surface

$$y_i \left(\boldsymbol{w}^\top \boldsymbol{x}_i + b\right) = 1$$

In this case, all training data points will satisfy the constraints

$$y_i \left(\boldsymbol{w}^\top \boldsymbol{x}_i + b\right) \geq 1$$

We can exploit this observation to simplify the maximum margin classifier optimization problem (we will show two different ways to accomplish this)

$$\max_{\boldsymbol{w},b} \quad \underbrace{\left[\min_{\boldsymbol{x}_i \in \mathcal{S}} \frac{|\boldsymbol{w}^\top \boldsymbol{x}_i + b|}{\|\boldsymbol{w}\|_2}\right]}_{\text{minimum distance from points to } \boldsymbol{w}}$$

$$\text{s.t} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 0, \; i = 1, 2, \ldots, n$$

$$\max_{\boldsymbol{w},b} \underbrace{\left[\min_{\boldsymbol{x}_i \in \mathcal{S}} \frac{|\boldsymbol{w}^\top \boldsymbol{x}_i + b|}{\|\boldsymbol{w}\|_2}\right]}_{\text{minimum distance from points to } \boldsymbol{w}}$$

$$\text{s.t} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 0, \ i = 1, 2, \ldots, n$$

$$\min_{\boldsymbol{x}_i \in \mathcal{S}} |\boldsymbol{w}^\top \boldsymbol{x}_i + b| = 1$$

$$\max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|_2}$$

$$\text{s.t} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1, \ i = 1, 2, \ldots, n$$

$$\max_{\boldsymbol{w},b} \underbrace{\left[\min_{\boldsymbol{x}_i \in \mathcal{S}} \frac{|\boldsymbol{w}^\top \boldsymbol{x}_i + b|}{\|\boldsymbol{w}\|_2}\right]}_{\text{minimum distance from points to } \boldsymbol{w}}$$

$$\text{s.t} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 0, \ i = 1, 2, \ldots, n$$

$$\min_{\boldsymbol{x}_i \in \mathcal{S}} |\boldsymbol{w}^\top \boldsymbol{x}_i + b| = 1$$

$$\max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|_2}$$

$$\max_{\boldsymbol{w}} \frac{1}{\|\boldsymbol{w}\|_2} = \min_{\boldsymbol{w}} \|\boldsymbol{w}\|_2$$

$$\text{s.t} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1, \ i = 1, 2, \ldots, n$$

Added for optimization convenience!

$$\min_{\boldsymbol{w},b} \|\boldsymbol{w}\|_2^2$$

$$\text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1, \ i = 1, 2, \ldots, n$$

Hard-margin SVM for separable training data

$$\min_{\boldsymbol{w},b} \quad \|\boldsymbol{w}\|_2^2$$

$$\text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1, \; i = 1, 2, \ldots, n$$

- This is called **Hard-margin SVM** (separable data, or no constraint violation)

Hard-margin SVM for separable training data

$$\gamma = \frac{1}{\|\boldsymbol{w}\|_2}$$

Final result: can maximize margin by minimizing $\|\boldsymbol{w}\|_2$

Find a separating hyperplane such that maximizes margin γ or minimizes $\|\boldsymbol{w}\|_2$

Hard-margin SVM versus Perceptron

Perceptron tries to find a feasible solution (perfectly classifies the training data):

$$\min_{\boldsymbol{w},b} \quad 0$$

$$\text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 0, \; i = 1, 2, \ldots, n$$

SVM tries to find a feasible solution that maximizes the margin:

$$\min_{\boldsymbol{w},b} \quad \|\boldsymbol{w}\|_2^2$$

$$\text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1, \; i = 1, 2, \ldots, n$$

Hard-margin SVM for separable training data

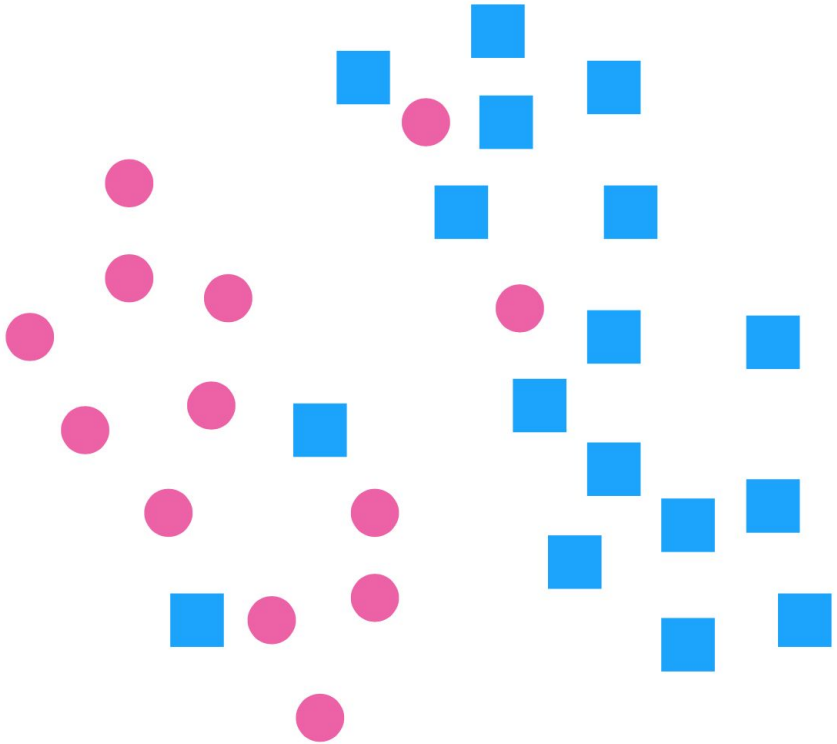
$$\min_{\boldsymbol{w},b} \quad \|\boldsymbol{w}\|_2^2$$

$$\text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1,\ i = 1, 2, \ldots, n$$

- This is called **Hard-margin SVM** (separable data, or no constraint violation)
- This is a convex optimization problem (convex quadratic program)
  - Quadratic objective function
  - Linear equality and inequality constraints
  - can be solved in polynomial time
- If there is a solution (i.e., data is linearly separable), then the solution is unique (Compare to Linear Programming and Perceptron lack of uniqueness)

Next lecture: soft-margin SVM

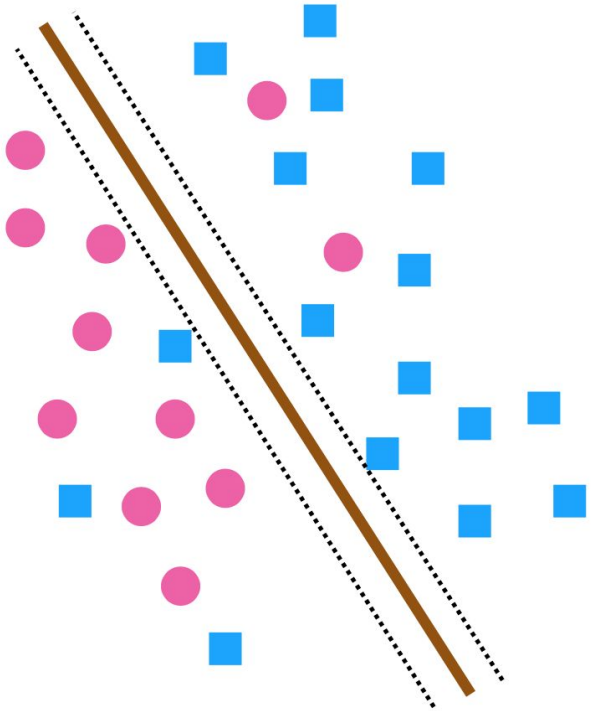
Can we generalize hard-margin SVM to cases where data is not separable?



Outline

- The concept of margin
- Distance of a point to a hyperplane
- The maximum margin classifier
- Hard-margin SVM for linearly separable data
- Soft-margin SVM for linearly inseparable data
  - Slack Variables
  - Constrained Optimization with Slack Variables
  - Unconstrained Optimization with Hinge loss
    - Subgradient Descent
- Kernelized SVM Prime-Dual and Kernel Trick

Linearly inseparable data



$$\min_{\boldsymbol{w},b} \|\boldsymbol{w}\|_2^2$$

$$\text{s.t.} \quad y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) \geq 1$$

$$y_2(\boldsymbol{w}^\top \boldsymbol{x}_2 + b) \geq 1$$

$$\vdots$$

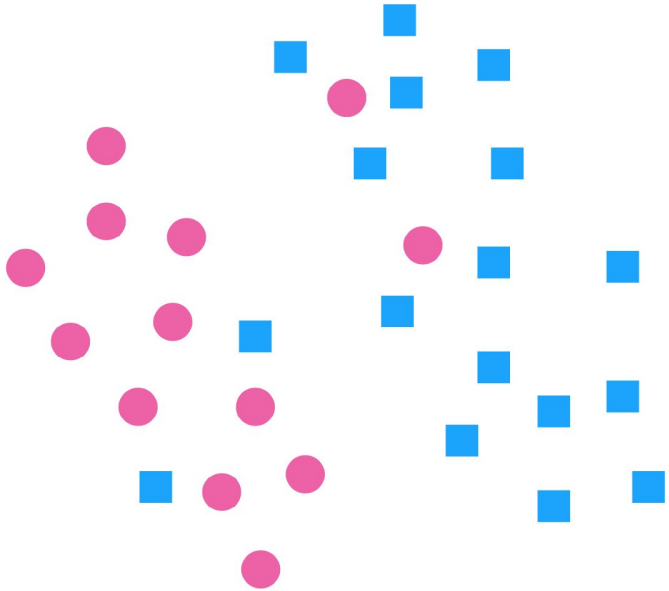
$$y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1$$

This is going to be a problem!

What should we do?

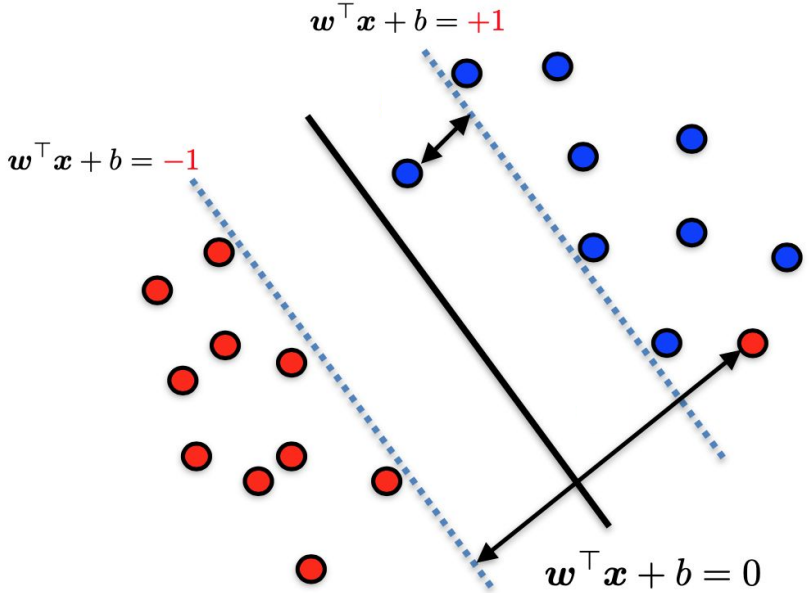
Linearly inseparable data

No linear classifier has zero training error on training data!



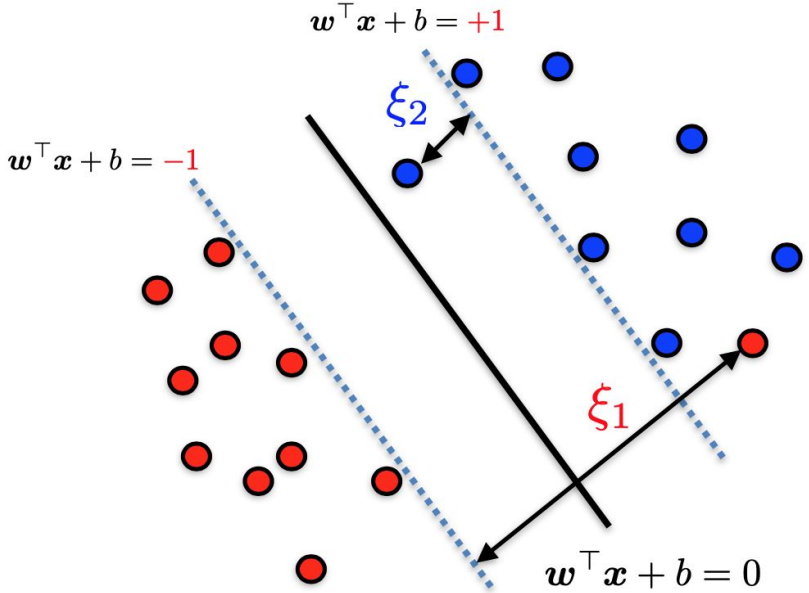
But if non-separability is only due to a handful of points, can we still find a good linear classier?

The amount of violations



How much a point fails to be separated with right margin?

The amount of violations: Slack Variables ξ



How much a point fails to be separated with right margin?

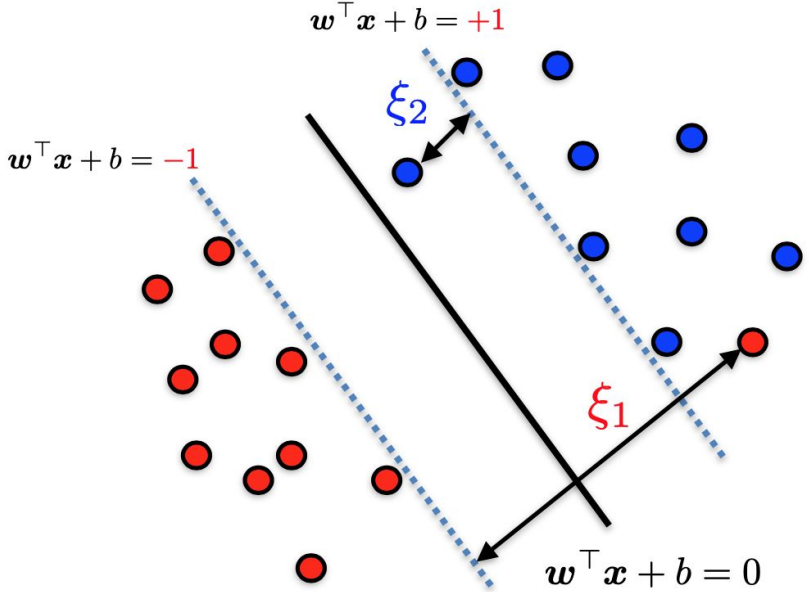
Introduce a **non-negative slack variable** for each training data point:

$$\xi$$

The amount of violations: Slack Variables ξ

The key idea is to relax the constraints by using **slack variables**

- This specifies the amount that you need to move the point to the right side of the correct margin.
- This allows tiny violations of constraints.



How much a point fails to be separated with right margin?

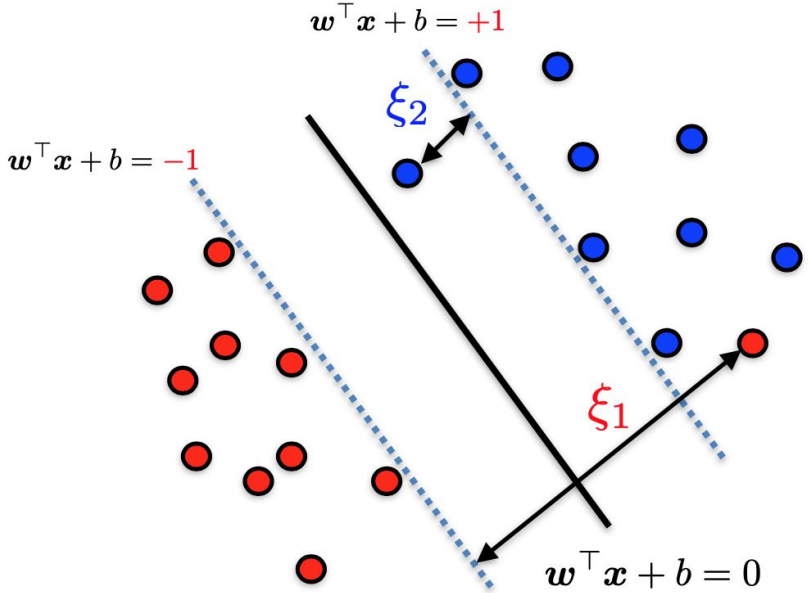
Introduce a **non-negative slack variable** for each training data point:

ξ

The amount of violations: Slack Variables ξ

The key idea is to relax the constraints by using **slack variables**

- This specifies the amount that you need to move the point to the right side of the correct margin.
- This allows tiny violations of constraints.



For a fixed hyperplane, define a non-negative slack variable for each training data as:

$\xi_1 \geq 0$ The amount $(\boldsymbol{x}_1, y_1)$ violates the margin constraints!

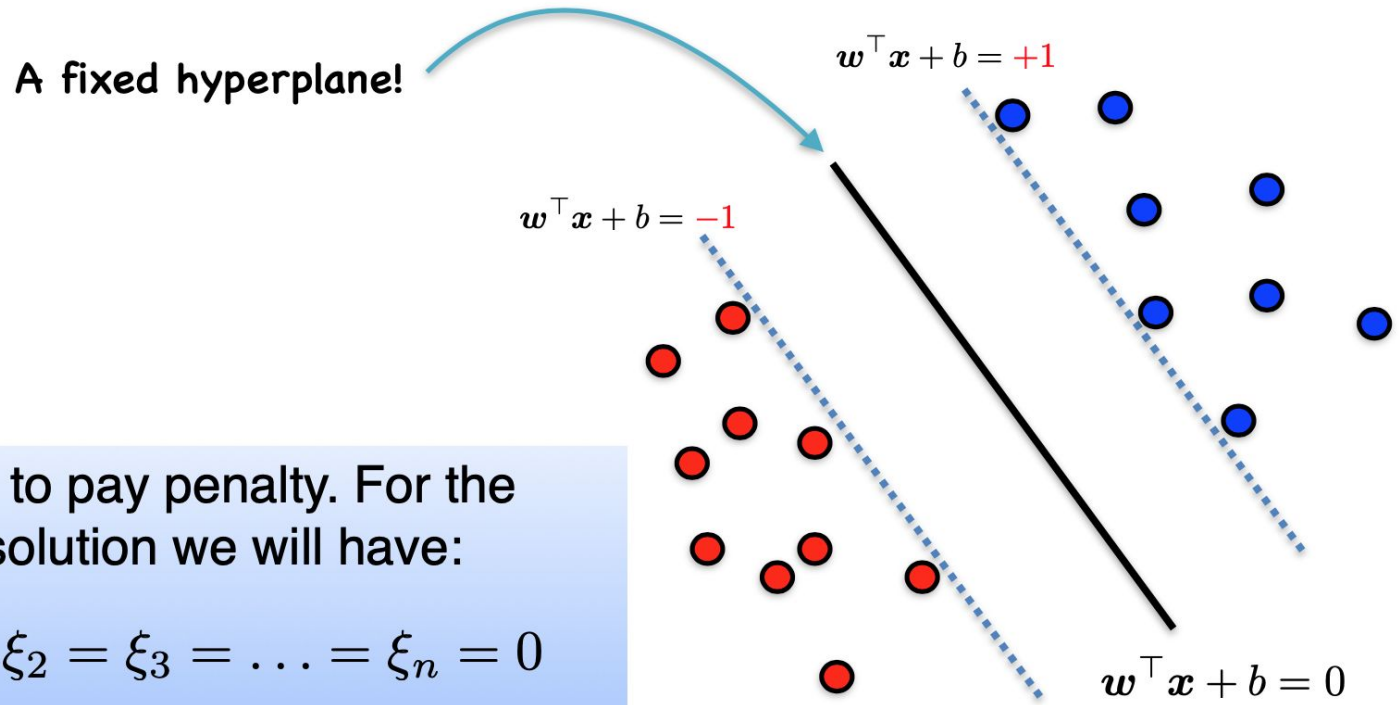
$\xi_2 \geq 0$ The amount $(\boldsymbol{x}_2, y_2)$ violates the margin constraints!

$\xi_3 \geq 0$ The amount $(\boldsymbol{x}_3, y_3)$ violates the margin constraints!

$\vdots$ $\vdots$

$\xi_n \geq 0$ The amount $(\boldsymbol{x}_n, y_n)$ violates the margin constraints!

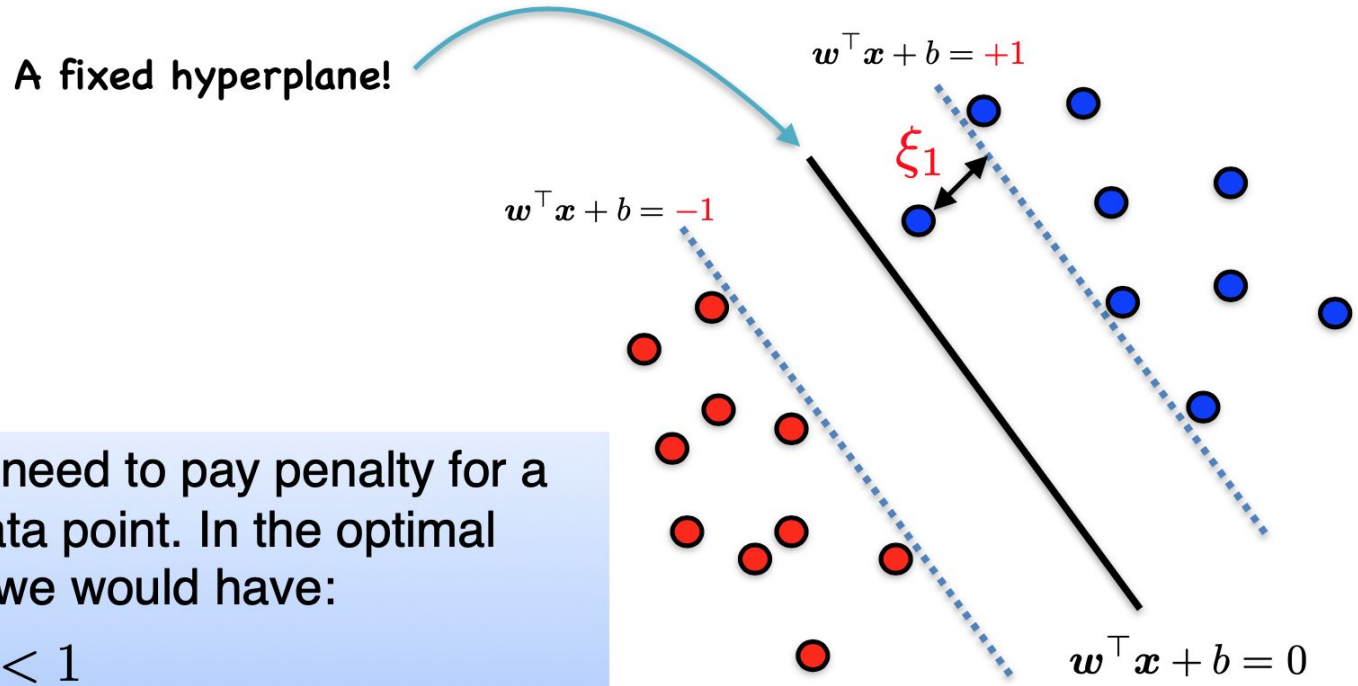
Example 1



No need to pay penalty. For the optimal solution we will have:

$$\xi_1 = \xi_2 = \xi_3 = \ldots = \xi_n = 0$$

Example 2

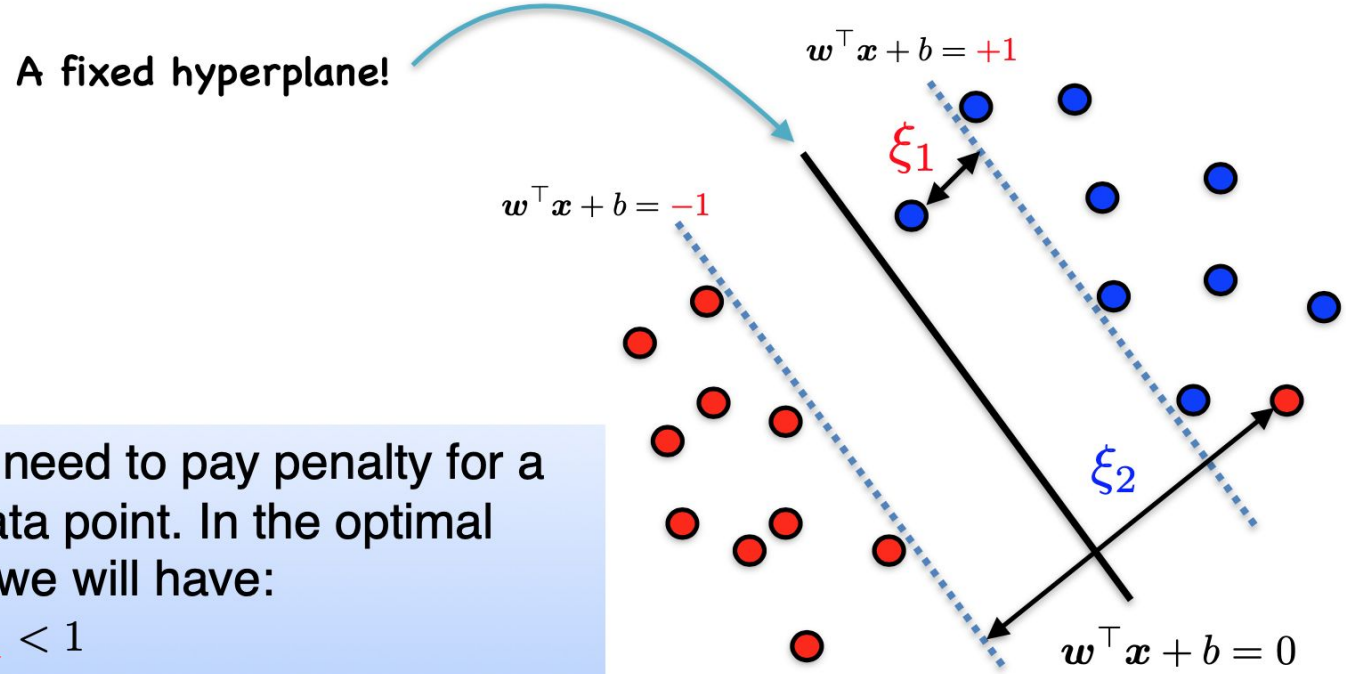


We only need to pay penalty for a single data point. In the optimal solution we would have:

$$0 < \xi_1 < 1$$

$$\xi_2 = \xi_3 = \xi_4 = \ldots = \xi_n = 0$$

Example 3



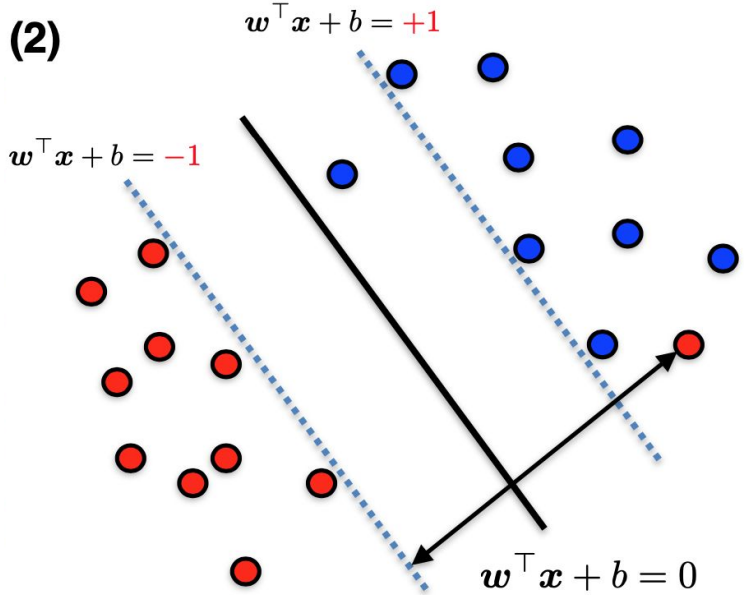
We only need to pay penalty for a single data point. In the optimal solution we will have:

$$0 < \xi_1 < 1$$

$$\xi_2 > 1$$

$$\xi_3 = \xi_4 = \xi_5 = \ldots = \xi_n = 0$$

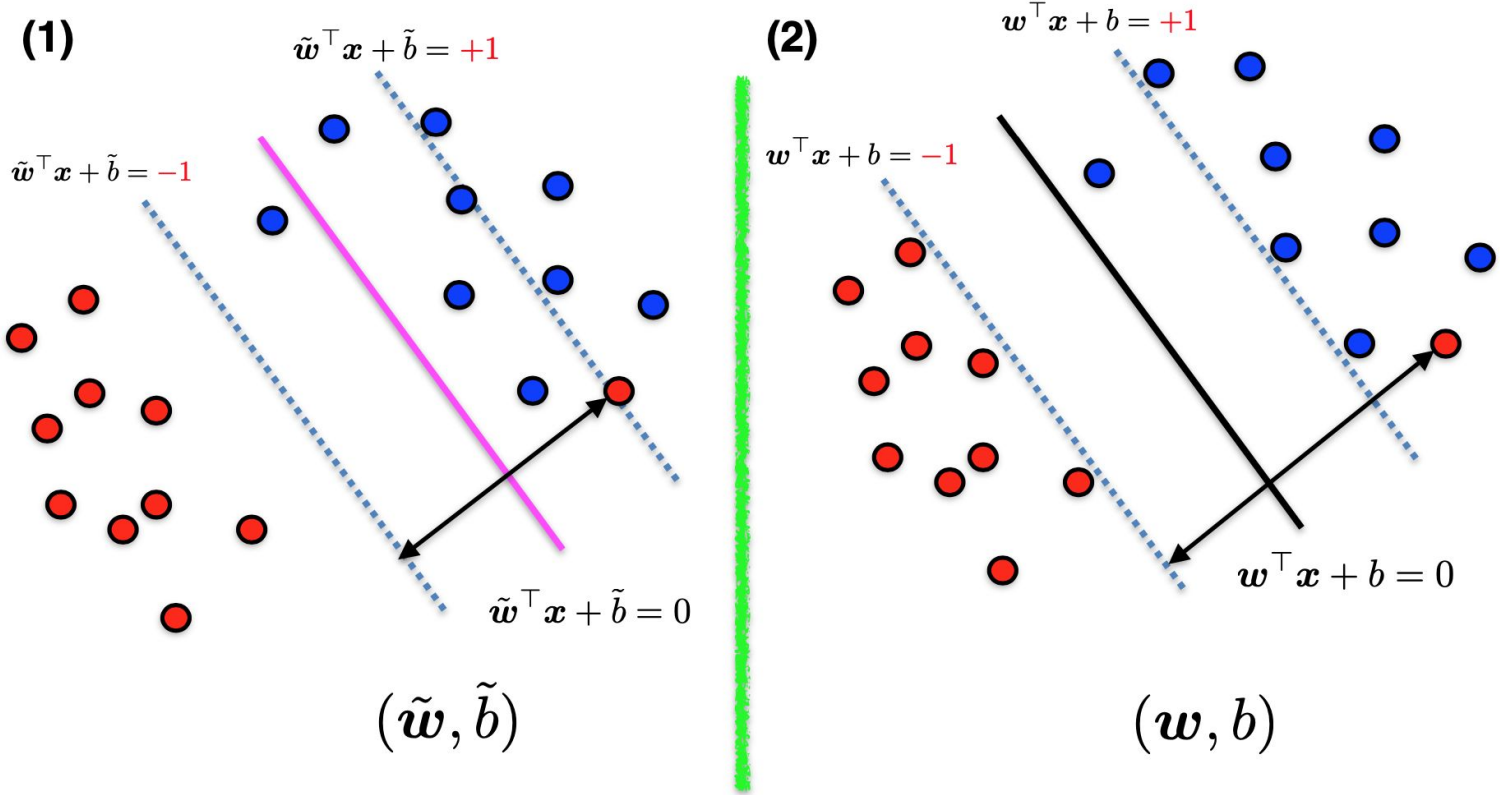
One Hyperplane



(w, b)

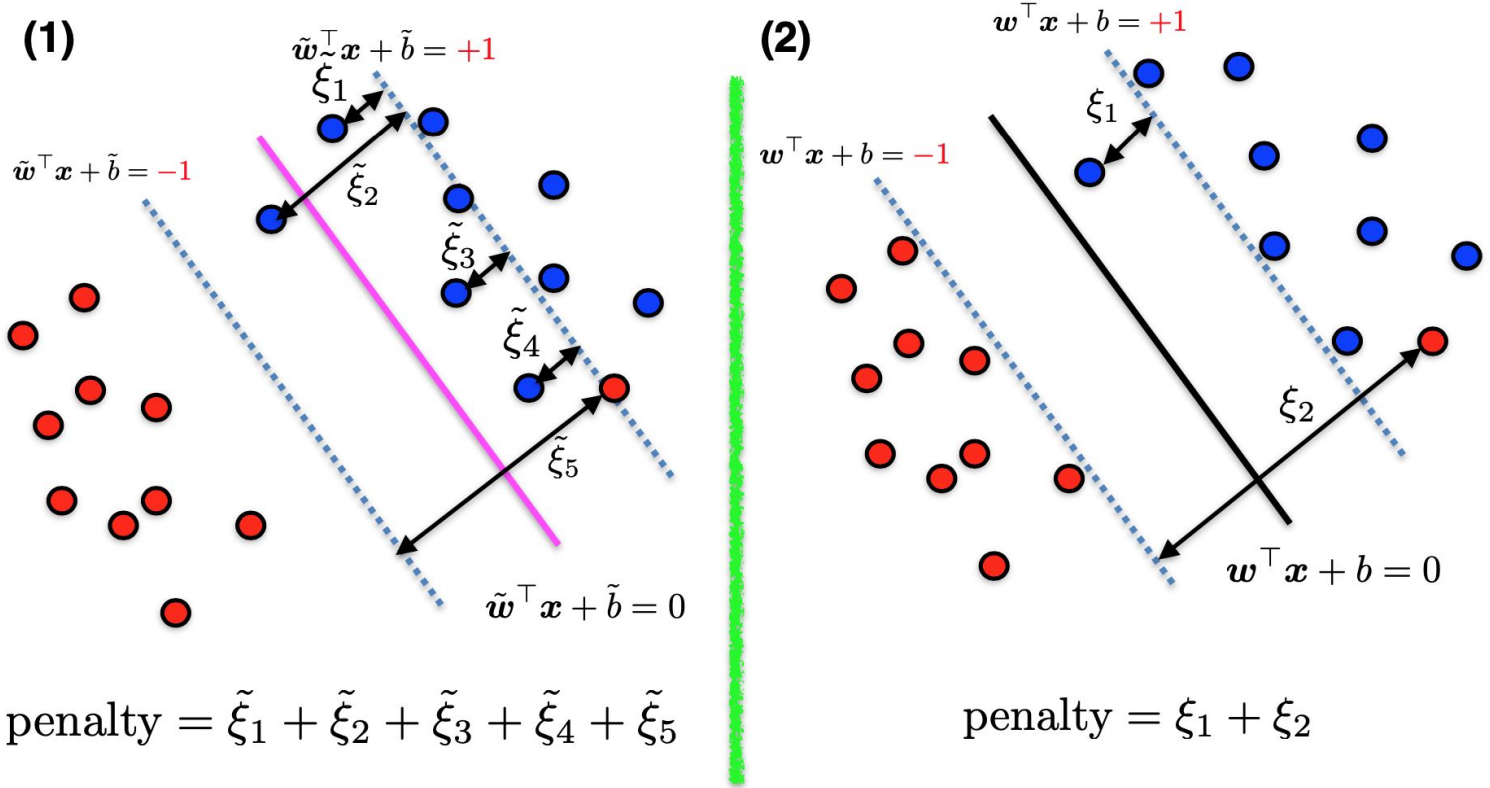
Another Hyperplane

(1) tries to reduce the penalty for  in (2)!



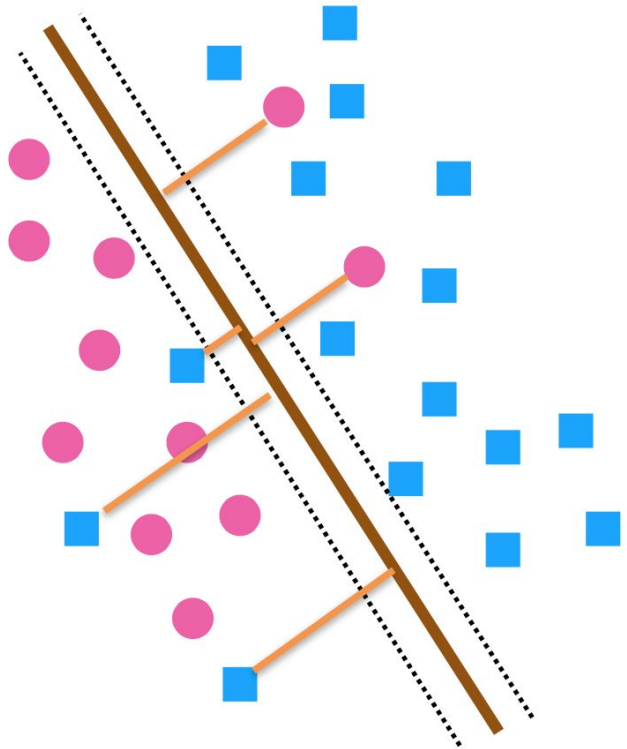
Which hyperplane?

By reducing penalty for ● in (2), we suffer penalty for other data points in (1)!



From Hard-margin SVM to Soft-margin SVM

Find a hyperplanes that (1) **maximizes margin**

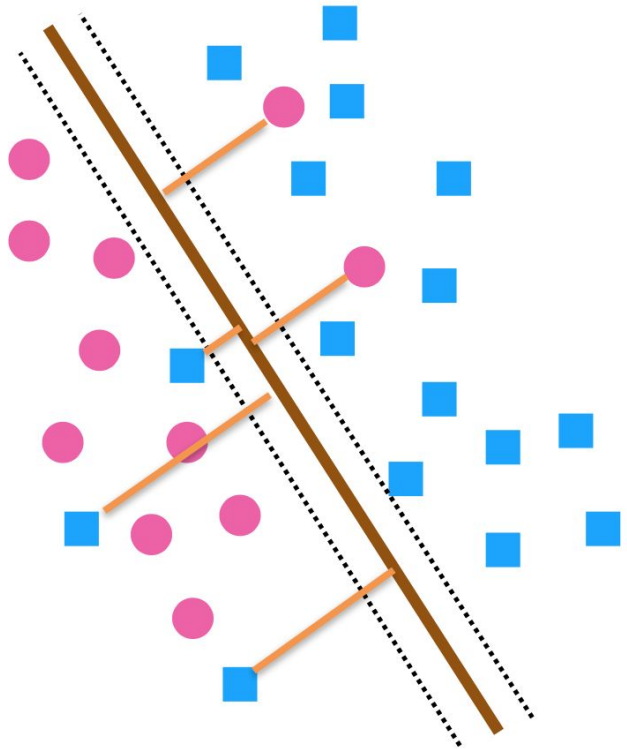


$$\min_{\boldsymbol{w},b,} \quad \|\boldsymbol{w}\|_2^2$$

$$\begin{aligned} \text{s.t.} \quad & y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) \geq 1 \\ & y_2(\boldsymbol{w}^\top \boldsymbol{x}_2 + b) \geq 1 \\ & \vdots \\ & y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 \end{aligned}$$

Adding Slack Variable in the constraints...

Find a hyperplanes that (1) **maximizes margin**

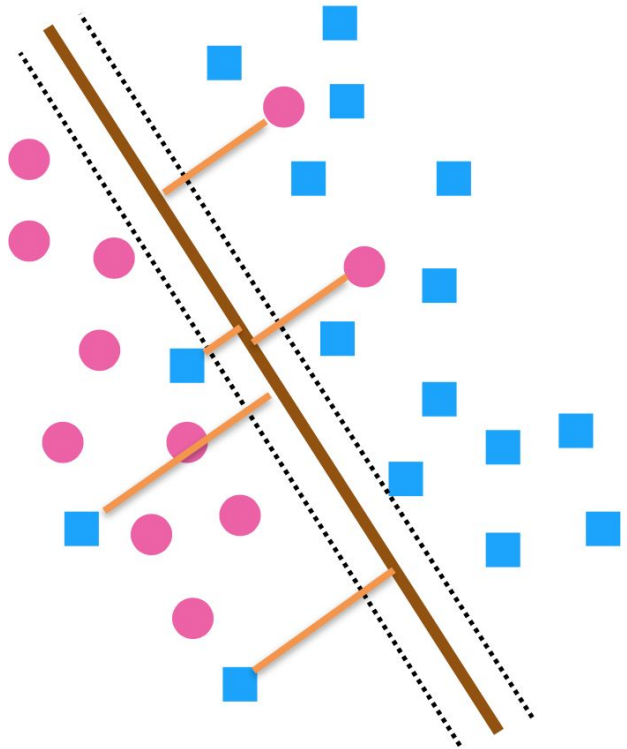


$$\min_{\boldsymbol{w},b,\xi_1,\ldots,\xi_n} \|\boldsymbol{w}\|_2^2$$

$$\begin{aligned}
\text{s.t.} \quad & y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) \geq 1 - \xi_1 \\
& y_2(\boldsymbol{w}^\top \boldsymbol{x}_2 + b) \geq 1 - \xi_2 \\
& \vdots \\
& y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 - \xi_n \\
& \xi_i \geq 0, \; i = 1, 2, \ldots, n
\end{aligned}$$

Also we need to minimize the violations

Find a hyperplanes that (1) **maximizes margin** while (2) **minimizing the amount of violations**

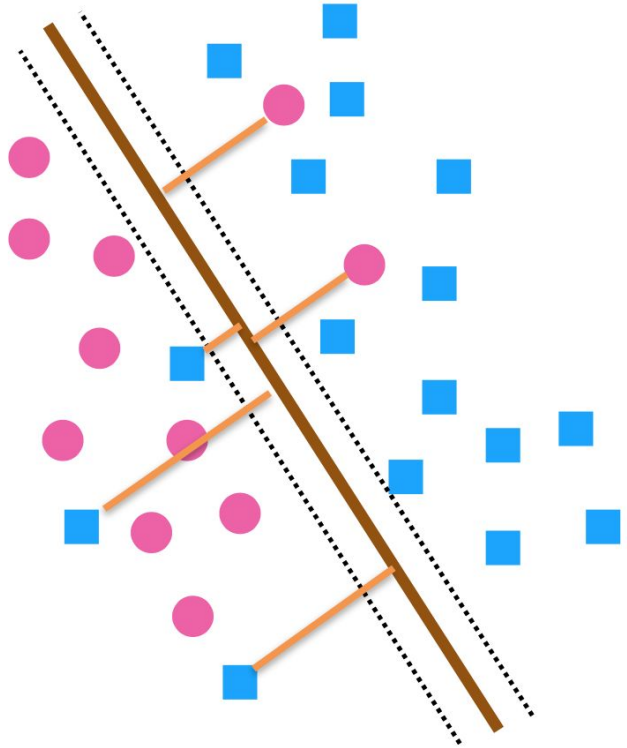


$$\min_{\boldsymbol{w},b,\xi_1,\ldots,\xi_n} \|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i$$

$$\begin{aligned}
\text{s.t.} \quad & y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) \geq 1 - \xi_1 \\
& y_2(\boldsymbol{w}^\top \boldsymbol{x}_2 + b) \geq 1 - \xi_2 \\
& \vdots \\
& y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 - \xi_n \\
& \xi_i \geq 0, \ i = 1, 2, \ldots, n
\end{aligned}$$

Soft-margin SVM for linearly inseparable data

Find a hyperplanes that (1) **maximizes margin** while (2) **minimizing the amount of violations**



Parameter to trade-off between sum of violations and margin!

$$\min_{\boldsymbol{w},b,\xi_1,\ldots,\xi_n} \|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \quad y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) \geq 1 - \xi_1$$

$$y_2(\boldsymbol{w}^\top \boldsymbol{x}_2 + b) \geq 1 - \xi_2$$

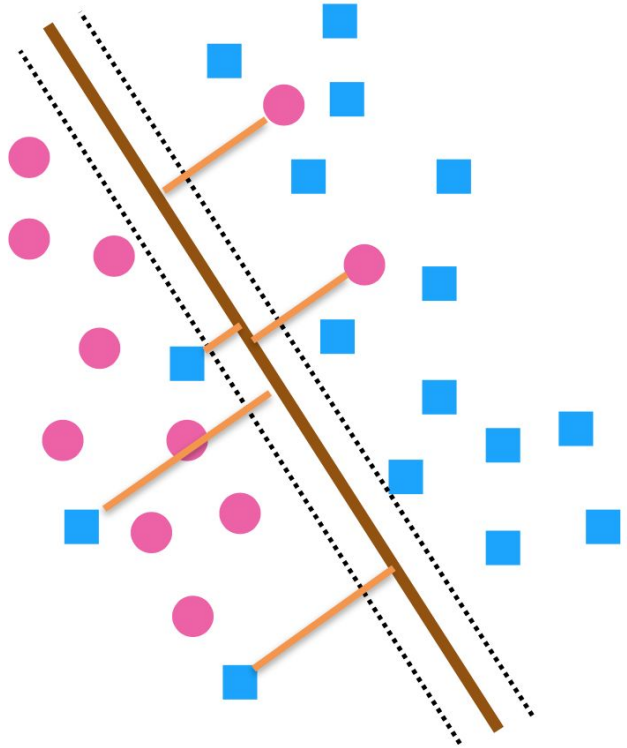
$$\vdots$$

$$y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 - \xi_n$$

$$\xi_i \geq 0, \; i = 1, 2, \ldots, n$$

Soft-margin SVM for linearly inseparable data

Find a hyperplanes that (1) **maximizes margin** while (2) **minimizing the amount of violations**



Parameter to trade-off between sum of violations and margin!

$$\min_{\boldsymbol{w},b,\xi_1,\ldots,\xi_n} \|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \quad y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) \geq 1 - \xi_1$$

$$y_2(\boldsymbol{w}^\top \boldsymbol{x}_2 + b) \geq 1 - \xi_2$$

$$\vdots$$

$$y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 - \xi_n$$

$$\xi_i \geq 0, \; i = 1, 2, \ldots, n$$

When data is linearly separable, all the slack variables will be zero in optimal solution.

Tradeoff parameter C

$$\min_{\boldsymbol{w},b,\xi_1,\ldots,\xi_n} \|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i$$

$$\begin{aligned}
\text{s.t.} \quad & y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) \geq 1 - \xi_1 \\
& y_2(\boldsymbol{w}^\top \boldsymbol{x}_2 + b) \geq 1 - \xi_2 \\
& \vdots \\
& y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 - \xi_n \\
& \xi_i \geq 0, \; i = 1, 2, \ldots, n
\end{aligned}$$

Parameter C is the trade-off for margin and amount of violations!

Tradeoff parameter C

$$\min_{\boldsymbol{w},b,\xi_1,\ldots,\xi_n} \|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \quad y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) \geq 1 - \xi_1$$

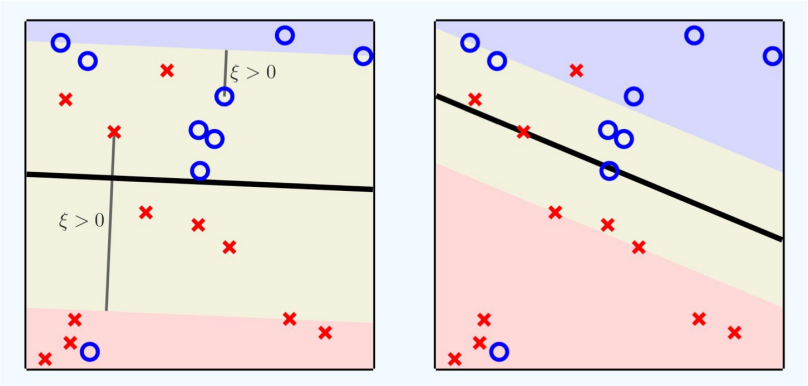
$$y_2(\boldsymbol{w}^\top \boldsymbol{x}_2 + b) \geq 1 - \xi_2$$

$$\vdots$$

$$y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 - \xi_n$$

$$\xi_i \geq 0, \; i = 1, 2, \ldots, n$$

Parameter C is the trade-off for margin and amount of violations!



Tradeoff parameter C

$$\min_{\boldsymbol{w},b,\xi_1,\ldots,\xi_n} \|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \quad y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) \geq 1 - \xi_1$$

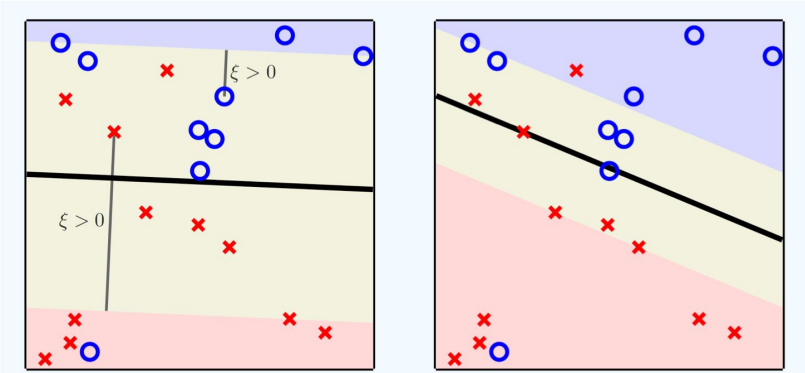
$$y_2(\boldsymbol{w}^\top \boldsymbol{x}_2 + b) \geq 1 - \xi_2$$

$$\vdots$$

$$y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 - \xi_n$$

$$\xi_i \geq 0, \; i = 1, 2, \ldots, n$$

Parameter C is the trade-off for margin and amount of violations!



(a) $C = 1$ (b) $C = 500$

When C is **small**, we obtain a classifier with very **large margin** but with many cases of margin violations.

When C is **large**, we obtain a classifier with **smaller margin** but with less margin violation.

Let's get rid of Slack Variables...

$$\min_{\boldsymbol{w},b,\xi_1,\ldots,\xi_n} \|\boldsymbol{w}\|_2^2 + C\xi_1 + C\sum_{i=2}^{n} \xi_i$$

$$\text{s.t.} \quad y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) > 1 - \xi_1,\ \xi_1 \geq 0,$$

$$y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) > 1 - \xi_i,\ \xi_i \geq 0,\ i = 2, 3, \ldots, n$$

Let's get rid of Slack Variables...

$$\min_{\boldsymbol{w},b,\xi_1,\ldots,\xi_n} \|\boldsymbol{w}\|_2^2 + C\xi_1 + C\sum_{i=2}^{n}\xi_i$$

$$\text{s.t.} \quad y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) > 1 - \xi_1,\ \xi_1 \geq 0,$$

$$y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) > 1 - \xi_i,\ \xi_i \geq 0,\ i = 2, 3, \ldots, n$$

Case 1: If $\boldsymbol{x}_1$ does not violate the constraint, $y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) \geq 1$, then its contribution to overall loss would be zero $\xi_1 = 0$

Let's get rid of Slack Variables...

$$\min_{\boldsymbol{w},b,\xi_1,\ldots,\xi_n} \|\boldsymbol{w}\|_2^2 + C\xi_1 + C\sum_{i=2}^{n}\xi_i$$

$$\text{s.t.} \quad y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) > 1 - \xi_1,\ \xi_1 \geq 0,$$

$$y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) > 1 - \xi_i,\ \xi_i \geq 0,\ i = 2, 3, \ldots, n$$

Case 1: If $\boldsymbol{x}_1$ does not violate the constraint, $y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) \geq 1$, then its contribution to overall loss would be zero $\xi_1 = 0$

Case 2: If $\boldsymbol{x}_1$ does violate the constraint, $y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) \ngeq 1$, then its minimum contribution to overall loss is $\xi_1 = 1 - y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b)$

Let's get rid of Slack Variables...

$$\min_{\boldsymbol{w},b,\xi_1,\ldots,\xi_n} \|\boldsymbol{w}\|_2^2 + C\xi_1 + C\sum_{i=2}^{n}\xi_i$$

$$\text{s.t.} \quad y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) > 1 - \xi_1,\ \xi_1 \geq 0,$$

$$y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) > 1 - \xi_i,\ \xi_i \geq 0,\ i = 2, 3, \ldots, n$$

Case 1: If $\boldsymbol{x}_1$ does not violate the constraint, $y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) \geq 1$, then its contribution to overall loss would be zero $\xi_1 = 0$

Case 2: If $\boldsymbol{x}_1$ does violate the constraint, $y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) \ngeq 1$, then its minimum contribution to overall loss is $\xi_1 = 1 - y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b)$

By combining these two cases, we can rewrite the violation of $\boldsymbol{x}_1$ as:

$$\xi_1 = \max\left(0, 1 - y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b)\right)$$

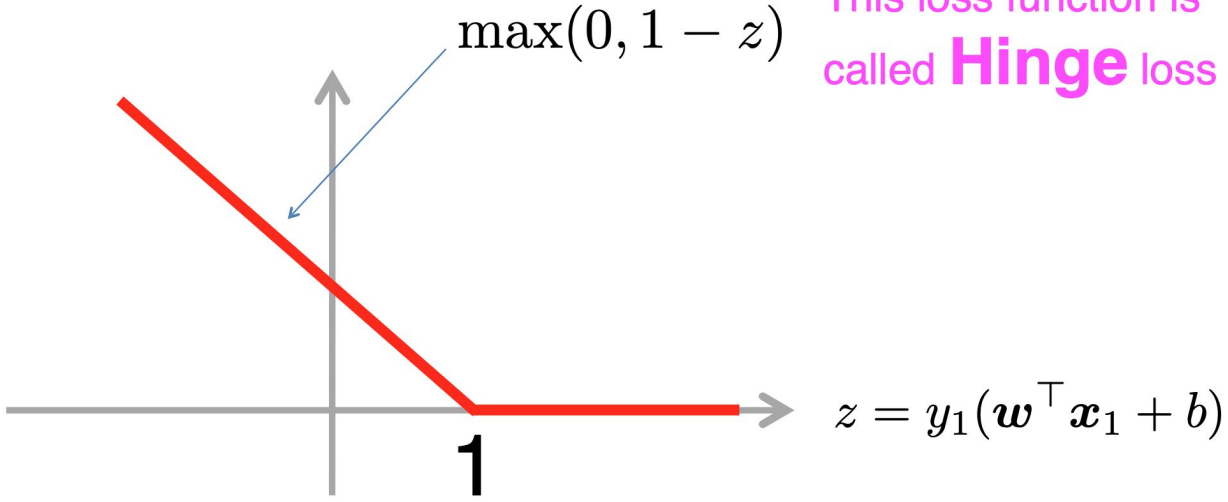
Support Vectors

In SVM, only training data points with margin of 1 or less actually affect the final solutions.

These are called **support vectors**.

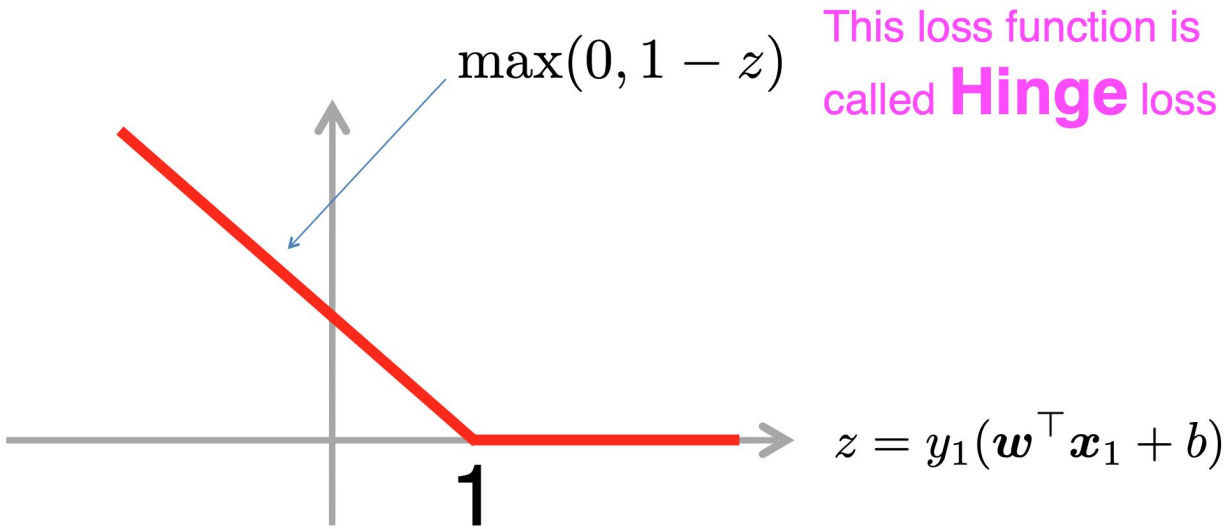
The loss minimization viewpoint

$$\xi_1 = \max\left(0, 1 - y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b)\right)$$



The loss minimization viewpoint

$$\xi_1 = \max\left(0, 1 - y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b)\right)$$



The optimal value for a slack variable is exactly the Hinge loss on the corresponding example!

The loss minimization viewpoint

Repeating the same (re-writing of constraints) for all training data gives:

$$\min_{\boldsymbol{w},b} \ \|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n} \max\left(0, 1 - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)\right)$$

So, SVM uses the Hinge loss as the error (loss) function for training

Regularization is automatically introduced by margin intuition

SVM vs Regularized Logistic Regression

$$\min_{\boldsymbol{w},b} \ \|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n} \max\left(0, 1 - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)\right)$$

$$\min_{\boldsymbol{w},b} \ \|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n} \ln\left(1 + \exp(-y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b))\right)$$

SVM vs Regularized Logistic Regression

$$\min_{\boldsymbol{w},b} \|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n} \max\left(0, 1 - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)\right)$$

SVM: Hinge loss

$$\min_{\boldsymbol{w},b} \|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n} \ln\left(1 + \exp(-y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b))\right)$$

Regularized Logistic Regression: Cross-Entropy Loss

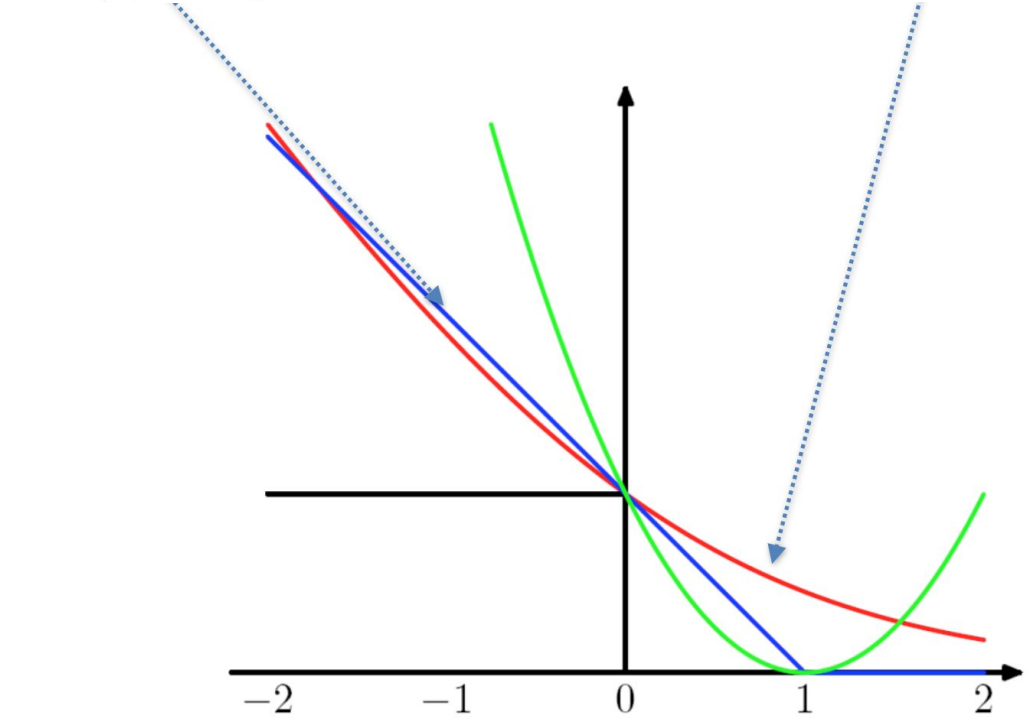
SVM vs Regularized Logistic Regression

SVM uses the Hinge loss:

$\max(0, 1 - z)$

LR uses the cross-entropy loss:

$\ln(1 + \exp(-z))$



Optimization methods

$$\min_{\boldsymbol{w},b} \|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n} \max\left(0, 1 - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)\right)$$

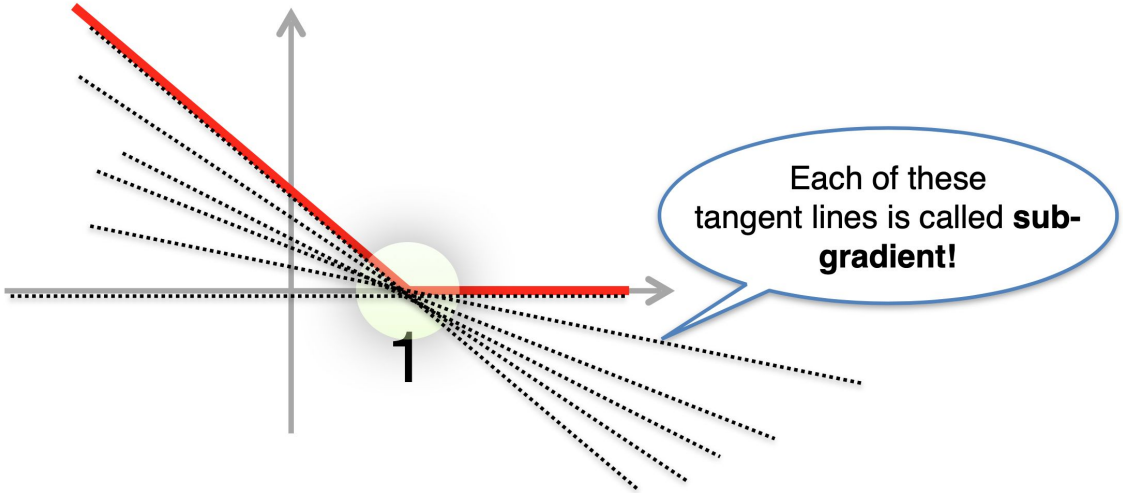
Quadratic programming problem with (1) Quadratic objective function, and (2) Linear equality and inequality constraints

Gradient Descent (GD) and Stochastic Gradient Descent (SGD)

Derivative of hinge loss

Hinge loss is not smooth (derivative does not exist at point (1, 0))

Infinite tangent lines (also called sub-gradients)

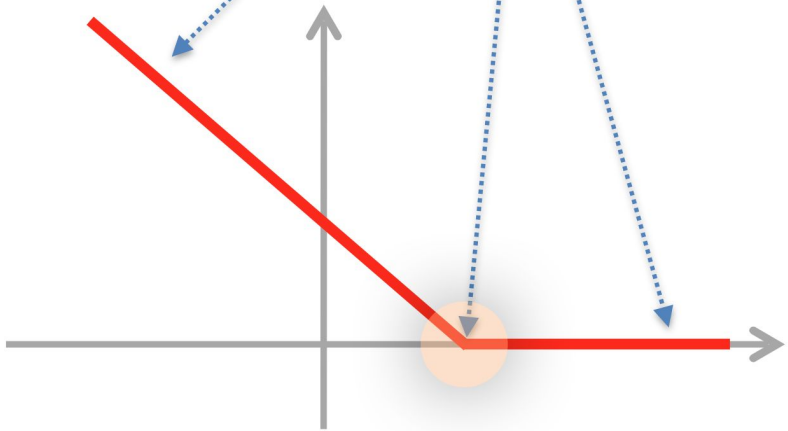


Sub-gradient Descent: when applying GD or SGD, at non-differentiable points, pick an arbitrary sub-gradient!

Derivative of hinge loss

Hinge loss is not smooth (derivative does not exist at point (1, 0))

$$\frac{\partial \max\left(0, 1 - y(\boldsymbol{w}^\top \boldsymbol{x} + b)\right)}{\partial \boldsymbol{w}} = \begin{cases} -y\boldsymbol{x} & y(\boldsymbol{w}^\top \boldsymbol{x} + b) < 1 \\ \mathbf{0} & y(\boldsymbol{w}^\top \boldsymbol{x} + b) > 1 \\ \mathbf{0} & y(\boldsymbol{w}^\top \boldsymbol{x} + b) = 1 \end{cases}$$



Stochastic subgradient Descent for SVM learning

- An application of stochastic subgradient descent for SVM learning
- Note: we can ignore parameter b by adding a column of all ones to data and learning a single parameter vector $\boldsymbol{w} = (w_1, \ldots, w_d, b) \in \mathbb{R}^{d+1}$

initialize $\boldsymbol{w}_1 = \mathbf{0}$

for t = 1, 2, …, T

- **choose** $i_t \in \{1, 2, 3, \ldots, n\}$ uniformly at random
- **set** $\eta_t \approx \frac{1}{t}$
- **if** $y_{i_t} \boldsymbol{w}_t^\top \boldsymbol{x}_{i_t} < 1$ then: $\Delta_t = -y_{i_t} \boldsymbol{x}_{i_t}$
  else if $y_{i_t} \boldsymbol{w}_t^\top \boldsymbol{x}_{i_t} \geq 1$ then: $\Delta_t = \mathbf{0}$

subgradient!

- **update** solution as:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \left(2\boldsymbol{w}_t + C\Delta_t\right)$$

Return $\boldsymbol{w}_{T+1}$

Potential issues with SVM

- Robustness: sensitive to **over-fitting**
- Not straightforward to generalize to **multi-class classification**
- Hard to tweak for **imbalanced data**

Tools

LibSVM

https://www.csie.ntu.edu.tw/~cjlin/libsvm/

scikit-learn (a Python wrapper on LibSVM)

http://scikit-learn.org/stable/modules/svm.html

SVMSGD (Pegasos)

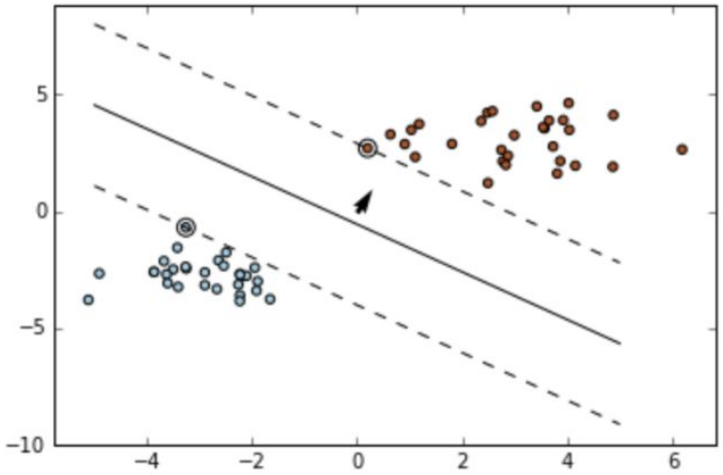
http://leon.bottou.org/projects/sgd

Vowpal Wabbit

http://hunch.net/~vw/

```
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")

import numpy as np
from sklearn import svm

def max_margin_classifier(ax, X, Y):
    clf = svm.SVC(kernel='linear')
    clf.fit(X, Y)

    # get the separating hyperplane
    w = clf.coef_[0]
    m = -w[0] / w[1]
    xx = np.linspace(-5, 5)
    yy = m * xx - (clf.intercept_[0]) / w[1]

    # plot the parallel lines to the separating hyperplane that pass through the
    # support vectors
    t = clf.support_vectors_[0]
    yy_down = m * xx + (t[1] - m * t[0])
    t = clf.support_vectors_[-1]
    yy_up = m * xx + (t[1] - m * t[0])

    ax.plot(xx, yy, 'k-')
    ax.plot(xx, yy_down, 'k--')
    ax.plot(xx, yy_up, 'k--')

    ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1],
            s=80, facecolors='none')
    ax.scatter(X[:, 0], X[:, 1], c=Y, cmap=plt.cm.Paired)

    ax.axis('tight')

    return w
```

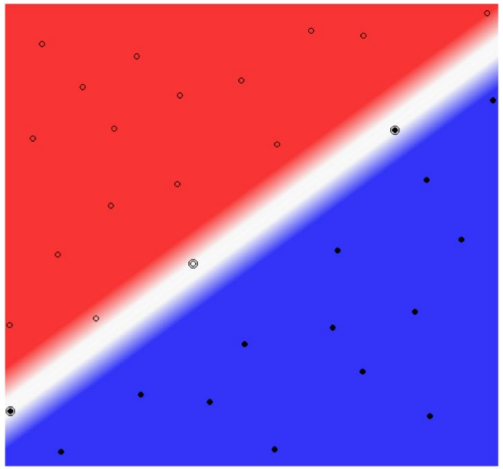
```
import matplotlib.pyplot as plt

# Create 30 random points
np.random.seed()
X = np.r_[np.random.randn(30, 2) - [3, 3], np.random.randn(30, 2) + [3, 3]]
Y = [0] * 30 + [1] * 30

fig = plt.figure()
ax = fig.add_subplot(111)
w = max_margin_classifier(ax, X, Y)
plt.quiver(0, 0, 2 * w[0], 2 * w[0]/w[1],
           angles='xy', scale_units='xy', scale=2)
plt.show()

# Note: The separating hyperplane is the line between the two dotted lines.
# The two dotted lines specify the margin and points on the margin are circled.
```

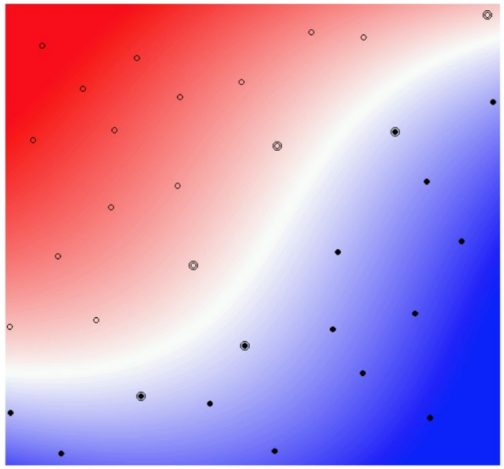


Outline

- The concept of margin
- Distance of a point to a hyperplane
- The maximum margin classifier
- Hard-margin SVM for linearly separable data
- Soft-margin SVM for linearly inseparable data
- Kernelized SVM Prime-Dual and Kernel Trick
    - Non-linear decision boundaries
    - Kernel Trick
    - Prime and Dual

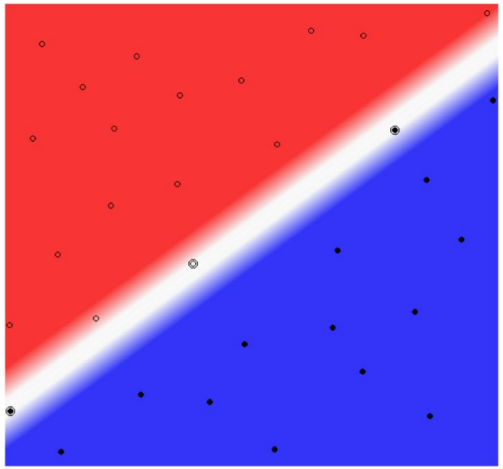
Non-linear separation

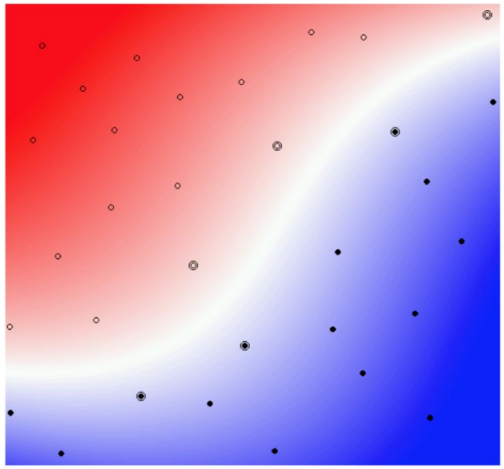




Linear separability is impossible in most real datasets

Non-linear separation





Linear separability is impossible in most real datasets

Idea: map input data to a high-dimensional feature space (hopefully data becomes separable in mapped spaces) and then run linear SVM!

- You already saw such a trick for perceptron in HW2

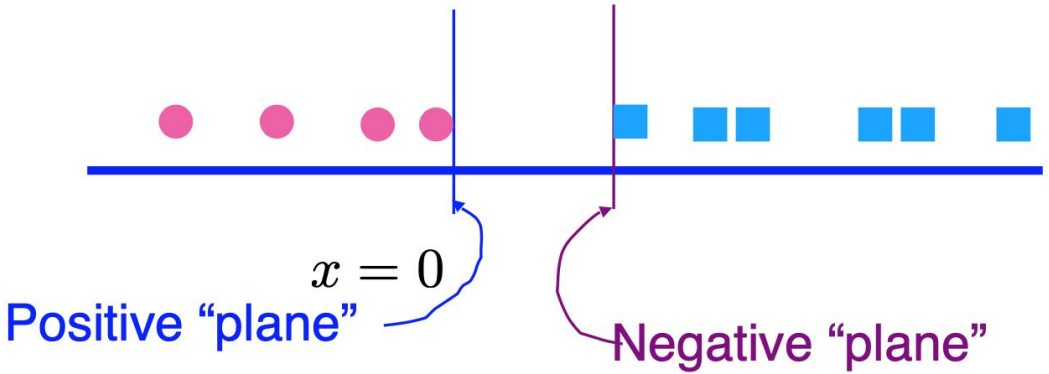
Suppose we are in 1-dimension

What would SVM do with this data?

$x = 0$

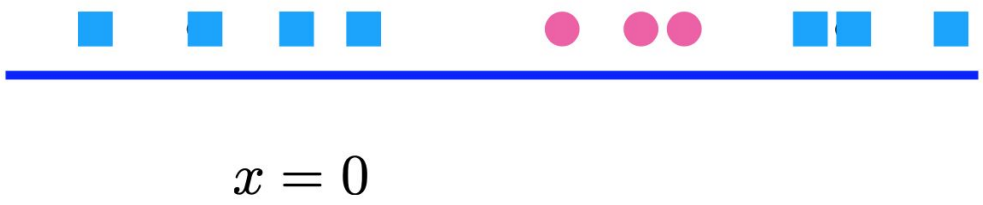
Suppose we are in 1-dimension

Not a big surprise



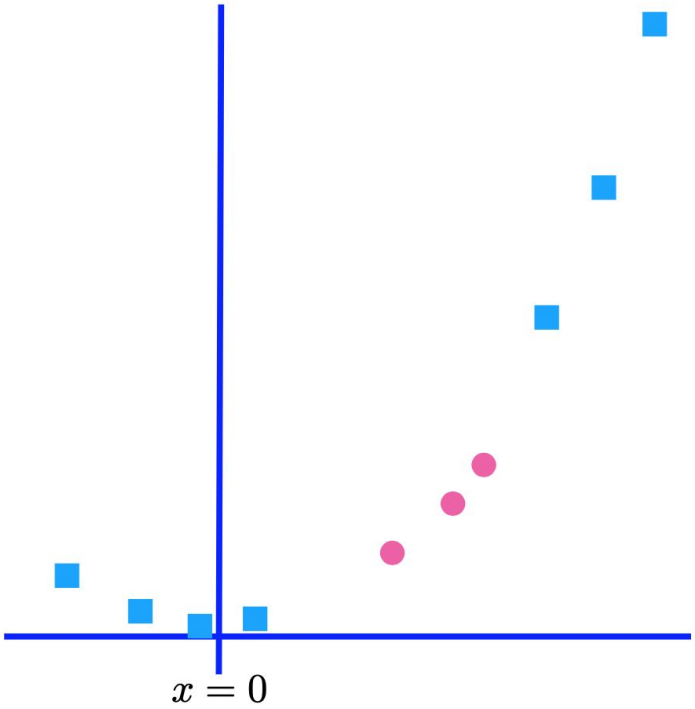
Harder 1-dimensional dataset

How about this?



Harder 1-dimensional dataset

Let's map data to two dimensions!



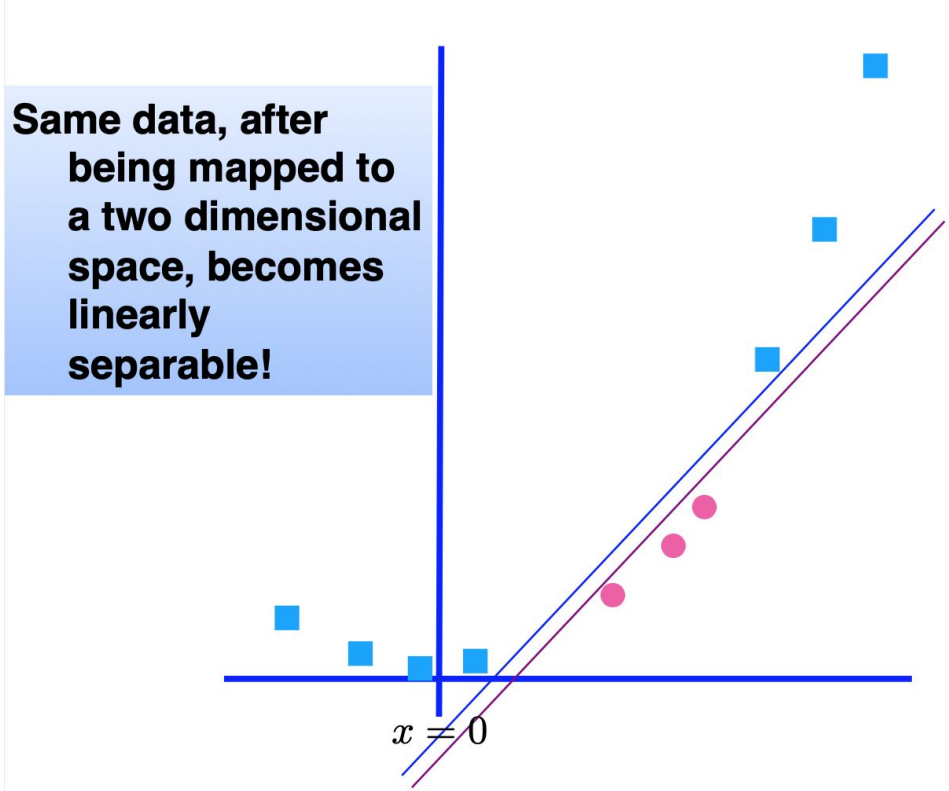
Mapping function

$$\boldsymbol{x} \longmapsto \Phi(\boldsymbol{x})$$

$$x \longmapsto (x, x^2)$$

Harder 1-dimensional dataset

Let's map data to two dimensions!



$$\boldsymbol{x} \longmapsto \Phi(\boldsymbol{x})$$

$$x \longmapsto (x, x^2)$$

Examples of high-dimensional mappings

Polynomial (with different degrees)

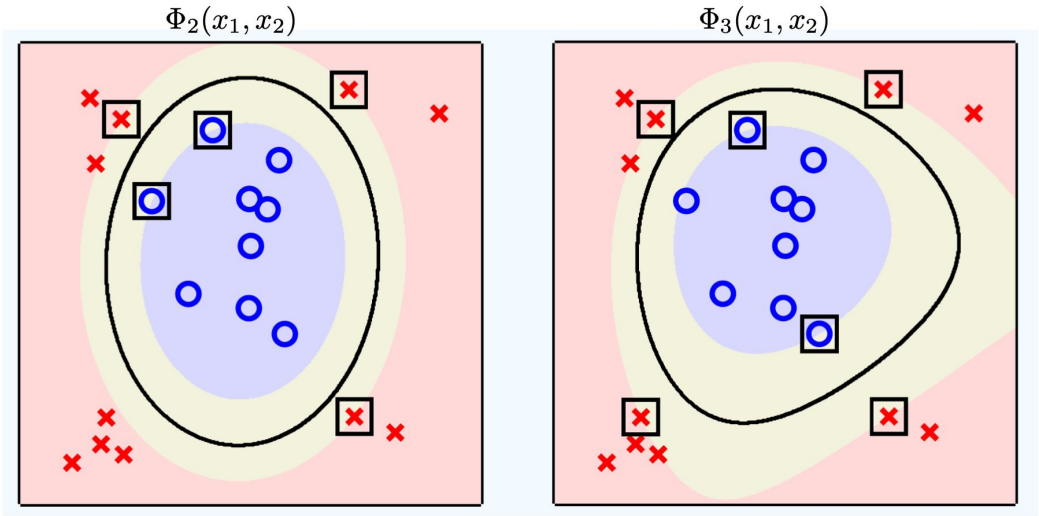
e.g., with 2nd order

$$\Phi : \mathbb{R}^d \mapsto \mathbb{R}^{1+d+d^2}$$

$$\Phi_2(\boldsymbol{x}) = (1, x_1, x_2, \ldots, x_1x_1, x_1x_2, x_1x_3, \ldots, x_dx_d)$$

Mappings and non-linear boundaries

Nonlinear separation using the SVM with 2nd and 3rd order polynomial transforms. The margin is shaded in yellow



$$\Phi_2(x_1, x_2) = (x_1, x_2, x_1^2, x_1 x_2, x_2^2)$$

$$\Phi_3(x_1, x_2) = (x_1, x_2, x_1^2, x_1 x_2, x_2^2, x_1^3, x_1^2 x_2, x_1 x_2^2, x_2^3)$$

Examples of high-dimensional mappings

Polynomial (with different degrees)

$$\Phi : \mathbb{R}^d \mapsto \mathbb{R}^{1+d+d^2}$$

e.g., with 2nd order

$$\Phi_2(\boldsymbol{x}) = (1, x_1, x_2, \ldots, x_1x_1, x_1x_2, x_1x_3, \ldots, x_dx_d)$$

Gaussian

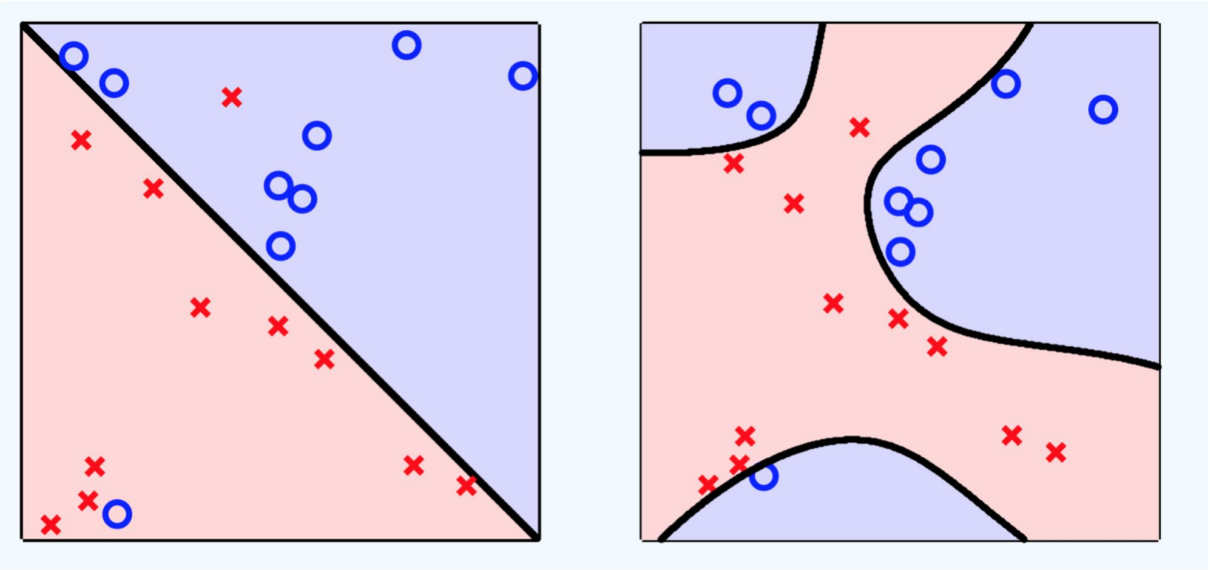
For simplicity, just consider scalar variables ($d = 1$)

$$\Phi : \mathbb{R}^1 \mapsto \mathbb{R}^{\infty}$$

$$\Phi(x) = \exp(-x^2)\left(1, \sqrt{\frac{2^1}{1!}}x, \sqrt{\frac{2^2}{2!}}x^2, \sqrt{\frac{2^3}{3!}}x^3, \ldots\right)$$

Mappings and non-linear boundaries

Nonlinear separation using Gaussian mapping!



Non-linear SVM

Map the data to a high-dimensional space (hopefully the data will be separable)

$$\boldsymbol{x}_i \mapsto \Phi\left(\boldsymbol{x}_i\right)$$

Solve the optimization problem in new space

$$\min_{\boldsymbol{w},b} \ \|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n} \max\left(0, 1 - y_i(\boldsymbol{w}^\top \Phi(\boldsymbol{x}_i) + b)\right)$$

Non-linear SVM

Map the data to a high-dimensional space (hopefully the data will be separable)

$$\boldsymbol{x}_i \mapsto \Phi\big(\boldsymbol{x}_i\big)$$

Solve the optimization problem in new space

$$\min_{\boldsymbol{w},b} \ \|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n} \max\left(0, 1 - y_i(\boldsymbol{w}^\top \Phi(\boldsymbol{x}_i) + b)\right)$$

Do we really need to do the explicit mapping?

We are really blowing up the number of features and the learning algorithm will take approximately forever to run!

Non-linearity and kernel trick

Map the data to a high-dimensional space (hopefully the data will be separable)

$$\boldsymbol{x}_i \mapsto \Phi\left(\boldsymbol{x}_i\right)$$

One issue is that if the dimension is high, we cannot compute $\Phi(\boldsymbol{x}_i)$ explicitly.

Non-linearity and kernel trick

Map the data to a high-dimensional space (hopefully the data will be separable)

$$\boldsymbol{x}_i \mapsto \Phi\left(\boldsymbol{x}_i\right)$$

One issue is that if the dimension is high, we cannot compute $\Phi(\boldsymbol{x}_i)$ explicitly.

Fortunately, due to the following two facts, we do not need to compute $\Phi(\boldsymbol{x}_i)$ explicitly:

Non-linearity and kernel trick

Map the data to a high-dimensional space (hopefully the data will be separable)

$$\boldsymbol{x}_i \mapsto \Phi\big(\boldsymbol{x}_i\big)$$

One issue is that if the dimension is high, we cannot compute $\Phi(\boldsymbol{x}_i)$ explicitly.

Fortunately, due to the following two facts, we do not need to compute $\Phi(\boldsymbol{x}_i)$ explicitly:

- Many ML algorithms depend on $\Phi(\boldsymbol{x})$ only via inner product $\langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j) \rangle$
- For certain Φ, the inner product can be computed efficiently using kernel function:

$$k\big(\boldsymbol{x}_i, \boldsymbol{x}_j\big) = \big\langle \Phi\big(\boldsymbol{x}_i\big), \Phi\big(\boldsymbol{x}_j\big) \big\rangle$$

  even if the dimension of $\Phi(\boldsymbol{x})$ is huge or even infinite.

Non-linearity and kernel trick

Map the data to a high-dimensional space (hopefully the data will be separable)

$$\boldsymbol{x}_i \mapsto \Phi\big(\boldsymbol{x}_i\big)$$

One issue is that if the dimension is high, we cannot compute $\Phi(\boldsymbol{x}_i)$ explicitly.

Fortunately, due to the following two facts, we do not need to compute $\Phi(\boldsymbol{x}_i)$ explicitly:

- Many ML algorithms depend on $\Phi(\boldsymbol{x})$ only via inner product $\langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j) \rangle$
- For certain Φ, the inner product can be computed efficiently using kernel function:

$$k\big(\boldsymbol{x}_i, \boldsymbol{x}_j\big) = \big\langle \Phi\big(\boldsymbol{x}_i\big), \Phi\big(\boldsymbol{x}_j\big) \big\rangle$$

  even if the dimension of $\Phi(\boldsymbol{x})$ is huge or even infinite.

Thanks to this kernel trick, we do NOT need explicit mapping!

Examples of Kernel Function

Suppose $\boldsymbol{x} \in \mathbb{R}^2$, consider the following function

$$k(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^\top \boldsymbol{v})^2$$

To show this is a kernel function, we need to show it can be written as

$$k(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^\top \boldsymbol{v})^2 = \langle \Phi(\boldsymbol{u}), \Phi(\boldsymbol{v}) \rangle \text{ for some } \Phi$$

Examples of Kernel Function

Suppose $\boldsymbol{x} \in \mathbb{R}^2$, consider the following function

$$k(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^\top \boldsymbol{v})^2$$

To show this is a kernel function, we need to show it can be written as

$$k(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^\top \boldsymbol{v})^2 = \langle \Phi(\boldsymbol{u}), \Phi(\boldsymbol{v}) \rangle \text{ for some } \Phi$$

In fact, we have

$$\begin{aligned} k(\boldsymbol{u}, \boldsymbol{v}) &= (\boldsymbol{u}^\top \boldsymbol{v})^2 \\ &= (u_1 v_1 + u_2 v_2)^2 \\ &= (u_1 v_1)^2 + 2 u_1 v_1 u_2 v_2 + (u_2 v_2)^2 \\ &= \langle \Phi(\boldsymbol{u}), \Phi(\boldsymbol{v}) \rangle \end{aligned}$$

by Choosing:

$$\Phi(\boldsymbol{u}) = (u_1^2, \sqrt{2} u_1 u_2, u_2^2)$$

$$\Phi(\boldsymbol{v}) = (v_1^2, \sqrt{2} v_1 v_2, v_2^2)$$

Examples of Kernel Function

For $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d$

Homogeneous Polynomial Kernel

$$k(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^\top \boldsymbol{v})^p$$

Examples of Kernel Function

For $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d$

Homogeneous Polynomial Kernel

$$k(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^\top \boldsymbol{v})^p$$

Inhomogeneous Polynomial Kernel

$$k(\boldsymbol{u}, \boldsymbol{v}) = (c + \boldsymbol{u}^\top \boldsymbol{v})^p, c > 0$$

Examples of Kernel Function

For $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d$

Homogeneous Polynomial Kernel

$$k(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^\top \boldsymbol{v})^p$$

Inhomogeneous Polynomial Kernel

$$k(\boldsymbol{u}, \boldsymbol{v}) = (c + \boldsymbol{u}^\top \boldsymbol{v})^p, c > 0$$

Gaussian Kernel (RBF Kernel)

$$k(\boldsymbol{u}, \boldsymbol{v}) = c \exp\left(-\frac{||\boldsymbol{u} - \boldsymbol{v}||_2^2}{2\sigma^2}\right), c > 0, \sigma > 0$$

Big Picture - The Kernel Trick

Using Kernel, we can obtain non-linear methods from linear methods as follows:

1. Select an inner product kernel k

2. Formulate your linear method such that feature vector only appear via inner product form $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$

3. Replace $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ with $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ through your algorithms

The Primal of Soft-margin SVM

Recall Soft-margin SVM is the following constrained optimization problem

$$\min_{\boldsymbol{w},b,\xi_i} \frac{1}{2}||\boldsymbol{w}||_2^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x} + b) \geq 1 - \xi_i, \forall i = 1, \ldots, n$$

$$\xi_i \geq 0, \forall i = 1, \ldots, n$$

The Primal of Soft-margin SVM

Recall Soft-margin SVM is the following constrained optimization problem

$$\min_{\boldsymbol{w},b,\xi_i} \frac{1}{2}||\boldsymbol{w}||_2^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x} + b) \geq 1 - \xi_i, \forall i = 1, \ldots, n$$

$$\xi_i \geq 0, \forall i = 1, \ldots, n$$

This is called the **primal** problem.

We denote the solution of this primal problem as

$$\boldsymbol{w}^*, b^*$$

Kernelized SVM - Make nonlinear SVM

Let's use the kernel trick on SVM to make non-linear SVM (i.e., Kernelized SVM)

The key step is step 2

2. Formulate your linear method such that feature vector only appear via inner product form

Kernelized SVM - Make nonlinear SVM

Let's use the kernel trick on SVM to make non-linear SVM (i.e., Kernelized SVM)

The key step is step 2

2. Formulate your linear method such that feature vector only appear via inner product form

How? We formulate the previous optimization using the **Dual** problem of SVM

- We will show what is the dual problem of SVM
- How dual solution is related to primal solution (without proof)
- We will see that the feature vector only appear via inner product in Dual, so that we can kernelize it

The Dual of Soft-margin SVM

In fact, the primal problem is equivalent to the following **Dual** problem

$$\max_{\alpha_i} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + \sum_{i=1}^{n}\alpha_i$$

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

$$0 \le \alpha_i \le \frac{C}{n}, \forall i = 1, \ldots, n$$

where α_i are called **lagrange multipliers** or **dual variables**

The Dual of Soft-margin SVM

In fact, the primal problem is equivalent to the following **Dual** problem

$$\max_{\alpha_i} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + \sum_{i=1}^{n}\alpha_i$$

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

$$0 \le \alpha_i \le \frac{C}{n}, \forall i = 1, \ldots, n$$

where α_i are called **lagrange multipliers** or **dual variables**

We denote the solution of this dual optimization problem is

$$\alpha_i^*$$

The relationship between primal and dual

primal

$$\min_{\boldsymbol{w},b,\xi_i} \frac{1}{2}||\boldsymbol{w}||_2^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x} + b) \geq 1 - \xi_i, \forall i = 1, \ldots, n$$

$$\xi_i \geq 0, \forall i = 1, \ldots, n$$

dual

$$\max_{\alpha_i} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + \sum_{i=1}^{n}\alpha_i$$

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

$$0 \leq \alpha_i \leq \frac{C}{n}, \forall i = 1, \ldots, n$$

The relationship between primal and dual

primal

$$\min_{\boldsymbol{w},b,\xi_i} \frac{1}{2}||\boldsymbol{w}||_2^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x} + b) \geq 1 - \xi_i, \forall i = 1, \ldots, n$$

$$\xi_i \geq 0, \forall i = 1, \ldots, n$$

dual

$$\max_{\alpha_i} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + \sum_{i=1}^{n}\alpha_i$$

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

$$0 \leq \alpha_i \leq \frac{C}{n}, \forall i = 1, \ldots, n$$

The solutions of primal and dual are related in the following manner:

$$\boldsymbol{w}^* = \sum_{i=1}^{n}\alpha_i^* y_i \boldsymbol{x}_i$$

The relationship between primal and dual

primal

$$\min_{\boldsymbol{w},b,\xi_i} \frac{1}{2}||\boldsymbol{w}||_2^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x} + b) \geq 1 - \xi_i, \forall i = 1, \ldots, n$$

$$\xi_i \geq 0, \forall i = 1, \ldots, n$$

dual

$$\max_{\alpha_i} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + \sum_{i=1}^{n}\alpha_i$$

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

$$0 \leq \alpha_i \leq \frac{C}{n}, \forall i = 1, \ldots, n$$

The solutions of primal and dual are related in the following manner:

$$\boldsymbol{w}^* = \sum_{i=1}^{n}\alpha_i^* y_i \boldsymbol{x}_i$$

$$b^* = y_i - \boldsymbol{w}^{*\top}\boldsymbol{x}_i = y_i - \sum_{j=1}^{n}\alpha_j^* y_j \boldsymbol{x}_j^\top \boldsymbol{x}_i$$

$$\text{for any } i, 0 < \alpha_i^* < \frac{C}{n}$$

Kernelized SVM using the Dual

In fact, the primal problem is equivalent to the following **Dual** problem

$$\max_{\alpha_i} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + \sum_{i=1}^{n}\alpha_i$$

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

$$0 \le \alpha_i \le \frac{C}{n}, \forall i = 1, \ldots, n$$

Note that here, the feature vectors only appear in the inner product! So we can choose a kernel to kernelize this.

Kernelized SVM using the Dual

In fact, the primal problem is equivalent to the following **Dual** problem

$$\max_{\alpha_i} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + \sum_{i=1}^{n}\alpha_i$$

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

$$0 \le \alpha_i \le \frac{C}{n}, \forall i = 1, \ldots, n$$

$$\max_{\alpha_i} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) + \sum_{i=1}^{n}\alpha_i$$

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

$$0 \le \alpha_i \le \frac{C}{n}, \forall i = 1, \ldots, n$$

Note that here, the feature vectors only appear in the inner product! So we can choose a kernel to kernelize this.

The relationship between primal and dual

The solutions of primal and dual are related in the following manner:

$$\boldsymbol{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \boldsymbol{x}_i$$

$$b^* = y_i - \boldsymbol{w}^{*\top} \boldsymbol{x}_i = y_i - \sum_{j=1}^{n} \alpha_j^* y_j \boldsymbol{x}_j^\top \boldsymbol{x}_i$$

$$\text{for any } i, 0 < \alpha_i^* < \frac{C}{n}$$

$$f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^{*\top} \boldsymbol{x} + b^*) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i^* y_i \boldsymbol{x}_i^\top \boldsymbol{x} + b^* \right)$$

$\alpha_i^* \neq 0$ ⇒ $\boldsymbol{x}_i$ is a support vector!

Kernelized SVM using the Dual

The solutions of primal and dual are related in the following manner:

$$\boldsymbol{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \boldsymbol{x}_i$$

$$b^* = y_i - \boldsymbol{w}^{*\top} \boldsymbol{x}_i = y_i - \sum_{j=1}^{n} \alpha_j^* y_j k(\boldsymbol{x}_j, \boldsymbol{x}_i)$$

$$\text{for any } i, 0 < \alpha_i^* < \frac{C}{n}$$

$$f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^{*\top} \boldsymbol{x} + b^*) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i^* y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b^* \right)$$

Big Picture - The Kernel Trick

Using Kernel, we can obtain non-linear methods from linear methods as follows:

1. Select an inner product kernel k

2. Formulate your linear method such that feature vector only appear via inner product form $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$

3. Replace $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ with $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ through your algorithms

We just demonstrate this process on Kernelized SVM, but this can be done on many ML algorithms, e.g., **kernelized ridge regression**.

Summary

SVM: Support Vector Machines

SVMs are motivated by finding linear classifiers with **maximum margins**

SVM is one of the most theoretically well motivated algorithms

SVM is also shown to be practically very effective

It also can utilize **kernel** trick to learn non-linear decision boundaries