CSE 575
Statistical Machine Learning

Lecture 9

YooJung Choi

Fall 2022

Recap: Generative vs discriminative

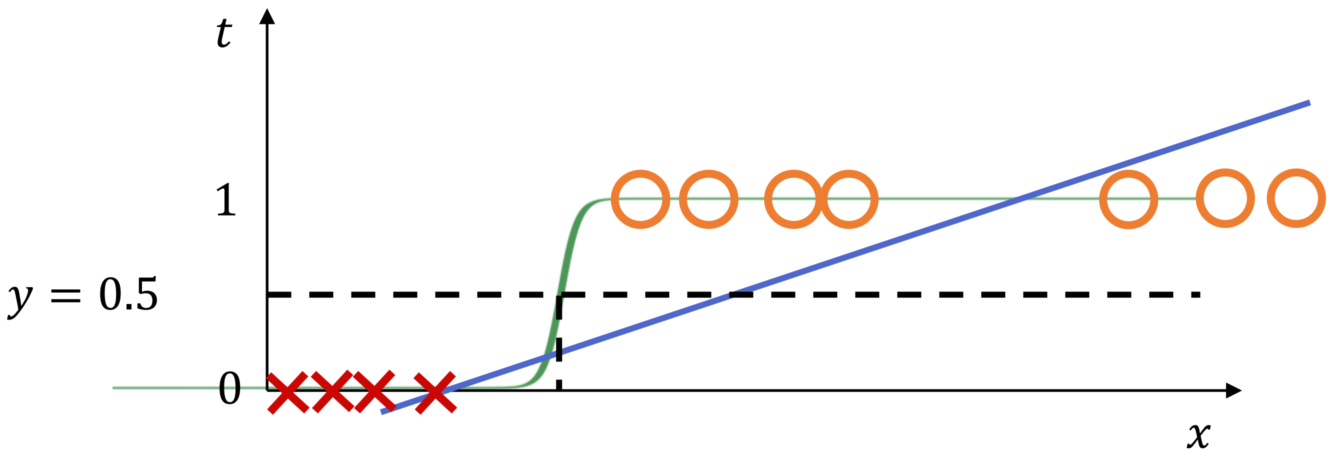
$p(\mathbf{x}, \mathcal{C}_k)$ given by (Linear) Gaussian discriminant analysis Naïve Bayes classifier	$p(\mathcal{C}_k \mid \mathbf{x})$ of the form $\sigma(\mathbf{w}^T\mathbf{x} + w_0)$ (binary class), or $s(\mathbf{w}^T\mathbf{x} + w_0)$ (multi-class)

- Can be used for various other tasks
- Performs very well *if* the modeling assumptions hold
- Closed-form solutions
- Tend to have more parameters

- Makes weaker assumptions—better performance in general
- Tend to have fewer parameters
- Limited to classification
- No closed-form solution

Logistic regression

- Model $p(t=1|\mathbf{x})$ via $\mathrm{y}(\mathbf{x}) = \frac{1}{1+\exp\{-\mathbf{w}^T\mathbf{x}\}} = \sigma(\mathbf{w}^T\mathbf{x})$
- Again, assume $x_0 = 1$, $t \in \{0,1\}$
- Recall: linear regression failed on this example



Logistic regression

- Note: $p(t=1|\mathbf{x}) = \mathrm{y}(\mathbf{x})$, i.e. $t \mid \mathbf{x} \sim \mathrm{Bernoulli}(\mathrm{y}(\mathbf{x}))$.
- Given *N* data points $\{(\mathbf{x}_n, t_n)\}$, the likelihood function is:

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n}(1-y_n)^{1-t_n} \qquad \text{where } y_n = y(\mathbf{x}_n) = \sigma(\mathbf{w}^T\mathbf{x}_n)$$

- Maximize log-likelihood, or equivalently, minimize the negative log-likelihood as the error function:

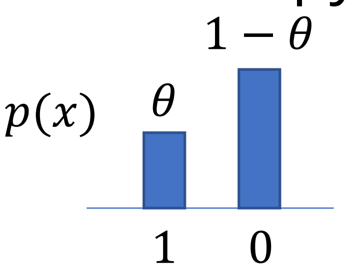
$$E(\mathbf{w}) = -\ln p(\mathbf{t}|\mathbf{w}) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\}$$

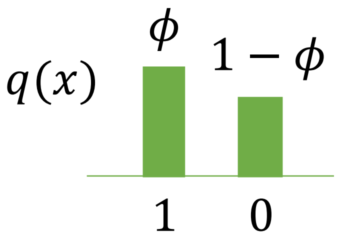
Logistic regression

- From last slide: *Cross-entropy loss*

$$E(\mathbf{w}) = -\ln p(\mathbf{t}|\mathbf{w}) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\}$$

- Cross entropy of two distributions p and q over Boolean variables:

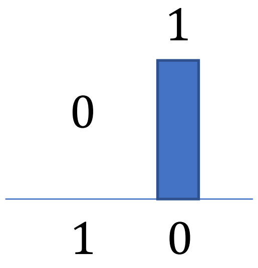


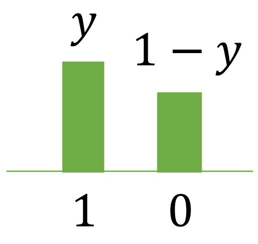


$$H(p,q) = -\theta \log \phi - (1-\theta)\log(1-\phi)$$

Quantifies how “close” the distributions are
Minimized when $p = q$

- $E(\mathbf{w})$: Minimize the cross entropy between the ground-truth distribution and the distribution given by logistic regression





Logistic regression

Exercise: derive $\nabla E(\mathbf{w})$.
Hint: first derive the derivative of logistic function $\frac{d\sigma(a)}{da} = \sigma(a) \cdot \big(1 - \sigma(a)\big)$

- Differentiate w.r.t. $\mathbf{w}$:

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n)\, \mathbf{x}_n$$

- Looks identical to the gradient of sum-of-squares error (linear regression)?
- $y_n = \sigma(\mathbf{w}^T \mathbf{x}_n)$ is *no longer linear*
- No known closed form solution
- Iteratively optimize via (stochastic) gradient descent!

$$\mathbf{w}^{(\text{new})} \leftarrow \mathbf{w}^{(\text{old})} - \eta \sum_{n=1}^{N} (y_n - t_n)\mathbf{x}_\text{n}$$

$$\mathbf{w}^{(\text{new})} \leftarrow \mathbf{w}^{(\text{old})} - \eta (y_n - t_n)\mathbf{x}_\text{n}$$

Newton's method (or Newton-Raphson method)

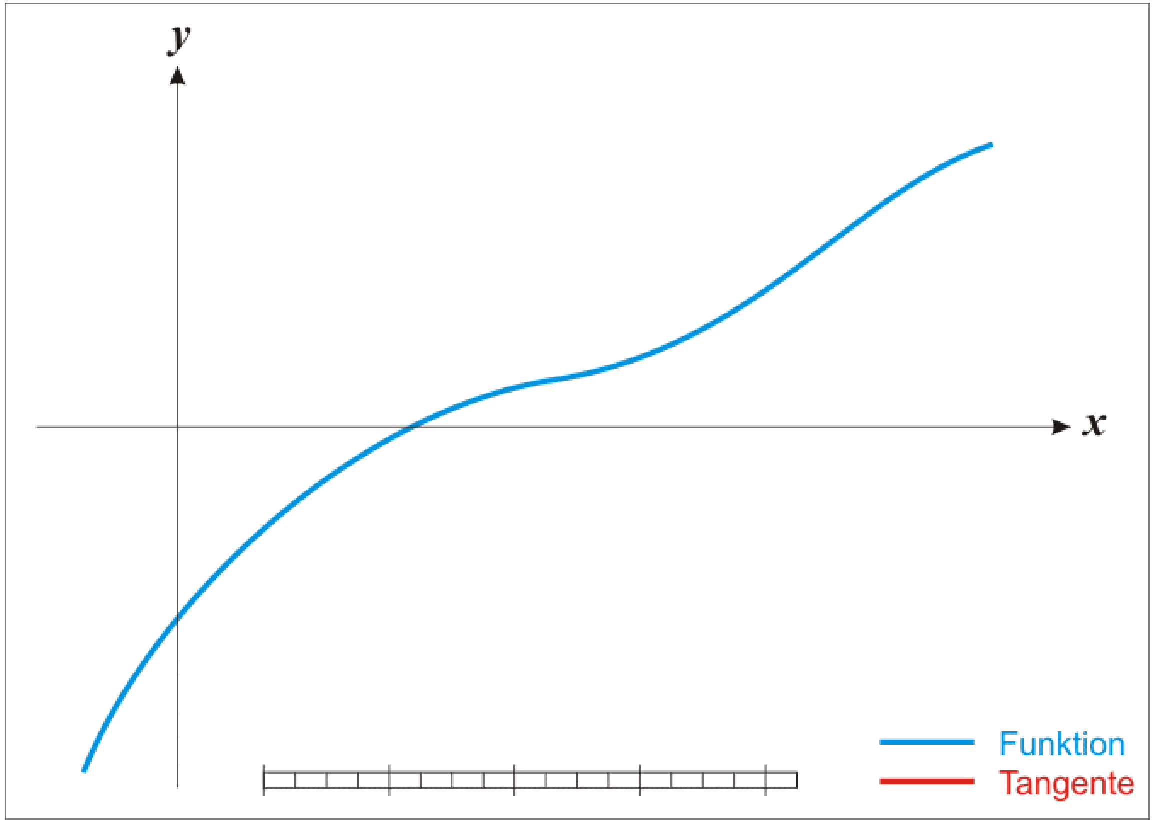
- Iteratively find the root of a function f: i.e. find x s.t. $f(x) = 0$
- Can be used to *minimize* $E(\mathbf{w})$ by finding the root of $\nabla E(\mathbf{w})$
- Compared to gradient descent, tends to converge in much *fewer iterations*.
- Each iteration is *more expensive*

Newton’s method

- Iteratively: find the root of a linear approximation of f

Newton’s method

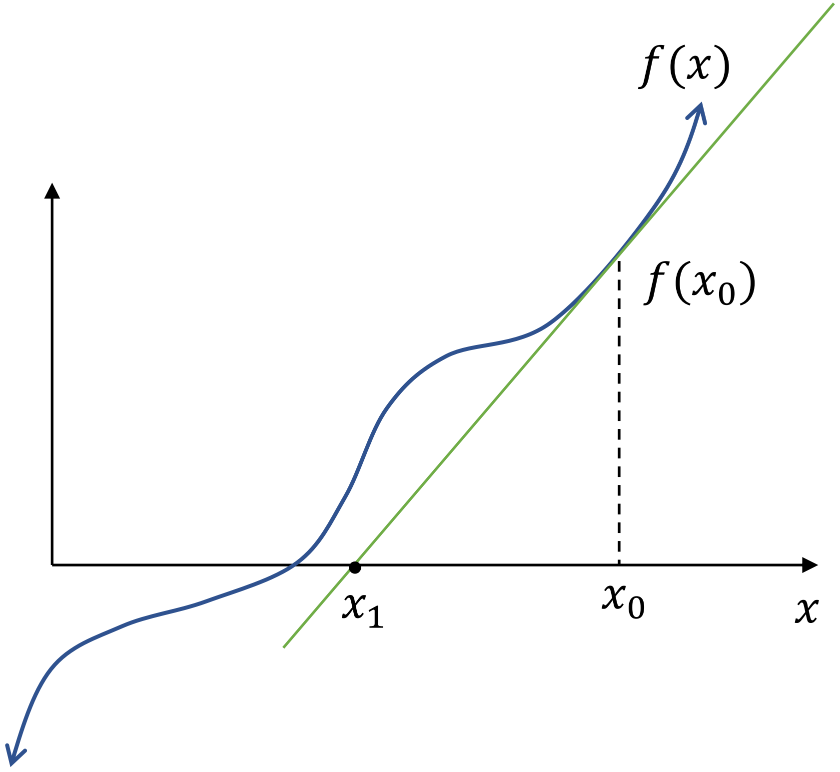
- Iteratively: find the root of a linear approximation of f



https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif

Newton's method

- Iteratively: find the root of a linear approximation of f



"Slope" = $f'(x_0) = \dfrac{f(x_0)}{x_0 - x_1} \quad \Rightarrow \quad x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$

To optimize f (i.e. find the root of f'):

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$$

For a *D* dimensional input $\mathbf{x}$:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \left(\nabla^2 f(\mathbf{x}_i)\right)^{-1} \nabla f(\mathbf{x}_i)$$

Newton's method for logistic regression

- Applying Newton's method to minimize $E(\mathbf{w})$:

$$\mathbf{w}^{(\text{new})} \leftarrow \mathbf{w}^{(\text{old})} - \mathbf{H}^{-1}\nabla E(\mathbf{w})$$

where $\nabla E(\mathbf{w}) = \sum_{n=1}^{N}(y_n - t_n)\,\mathbf{x}_n = \mathbf{X}^T(\mathbf{y} - \mathbf{t})$ $\quad O(ND)$ time

$$\mathbf{H} = \nabla^2 E(\mathbf{w}) = \sum_{n=1}^{N} y_n(1 - y_n)\mathbf{x}_n\mathbf{x}_n^T = \mathbf{X}^T\mathbf{R}\mathbf{X}$$

$O(ND^2)$ time

Inverse: $O(D^3)$ time

$$R_{nn} = y_n(1 - y_n)$$

$$\mathbf{X} = \begin{bmatrix} - & \mathbf{x}_1^T & - \\ & \vdots & \\ - & \mathbf{x}_N^T & - \end{bmatrix}, \qquad \mathbf{y} - \mathbf{t} = \begin{bmatrix} y(\mathbf{x}_1) - \mathbf{t}_1 \\ \vdots \\ y(\mathbf{x}_\mathrm{N}) - \mathbf{t}_\mathrm{N} \end{bmatrix}, \qquad \mathbf{R} = \begin{bmatrix} y(\mathbf{x}_1)(1 - y(\mathbf{x}_1)) & & \\ & \ddots & \\ & & y(\mathbf{x}_\mathrm{N})(1 - y(\mathbf{x}_\mathrm{N})) \end{bmatrix}$$

Softmax regression

- i.e. multiclass logistic regression
- For each class k: $p(\mathcal{C}_k|\mathbf{x}) = y_k(\mathbf{x}) = s(\mathbf{w}_k^T\mathbf{x})$

$$s(a_k) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

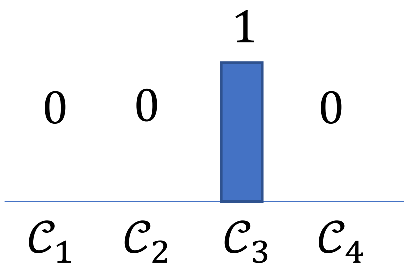
- Given N data points $\{(\mathbf{x}_n, \mathbf{t}_n)\}$, the likelihood function is:

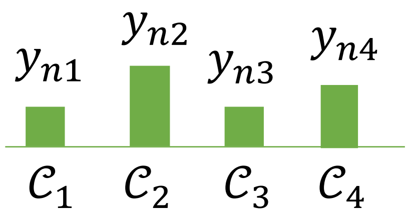
$\mathbf{t}_n$: 1-of-K ("one-hot") vector

$$p(\mathbf{T}|\mathbf{w}_1, \dots, \mathbf{w}_K) = \prod_{n=1}^N \prod_{k=1}^K p(\mathcal{C}_k|\mathbf{x}_n)^{t_{nk}} = \prod_{n=1}^N \prod_{k=1}^K y_{nk}^{t_{nk}} \quad \text{where } y_{nk} = y_k(\mathbf{x}_n)$$

- Maximum likelihood = minimum log-likelihood = minimum cross-entropy loss:

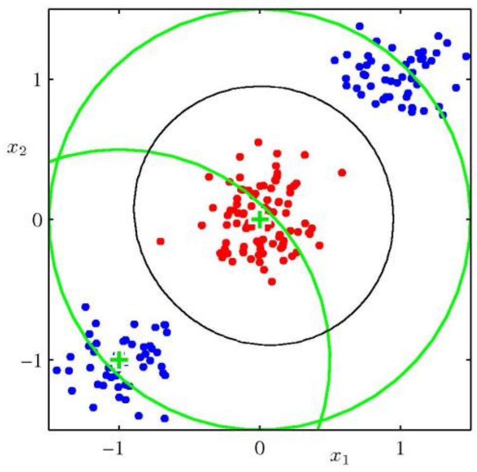
$$E(\mathbf{w}_1, \dots, \mathbf{w}_K) = -\ln p(\mathbf{T}|\mathbf{w}_1, \dots, \mathbf{w}_K) = -\sum_{n=1}^N \sum_{k=1}^K t_{nk} \ln y_{nk}$$

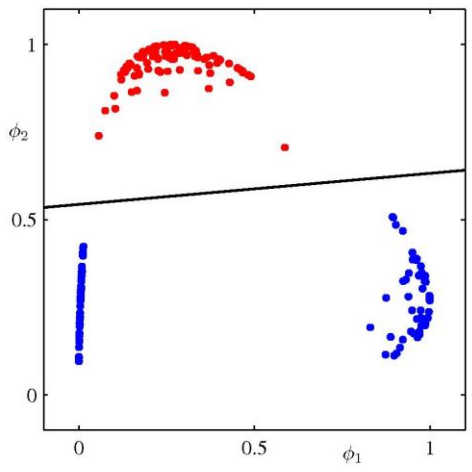




Non-linear decision boundaries

- Our discussion of linear regression $\sigma(\mathbf{w}^T\mathbf{x})$ and softmax regression $s(\mathbf{w}_k^T\mathbf{x})$ so far is limited to *linear decision boundaries*
- We can learn *non-linear decision boundaries* using fixed *non-linear basis functions* $\boldsymbol{\phi}(\mathbf{x})$ (c.f. linear basis function models for regression)

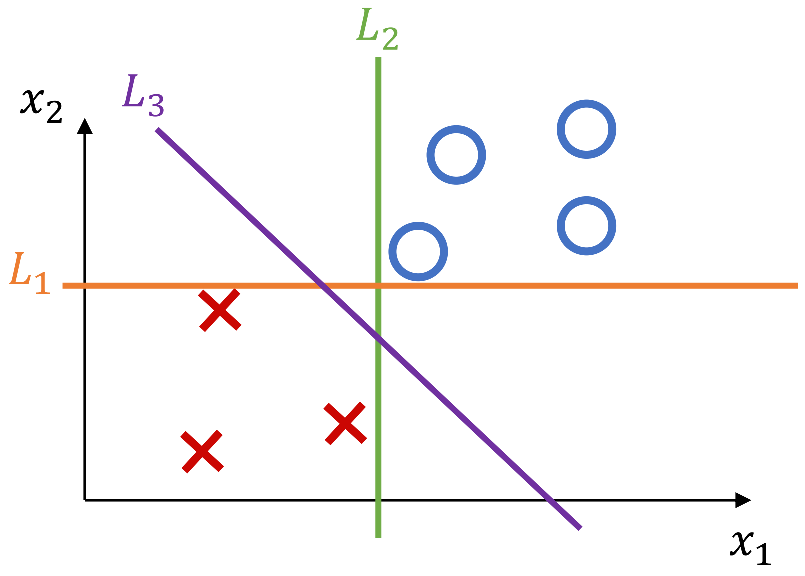




May require a very high dimensional feature space…

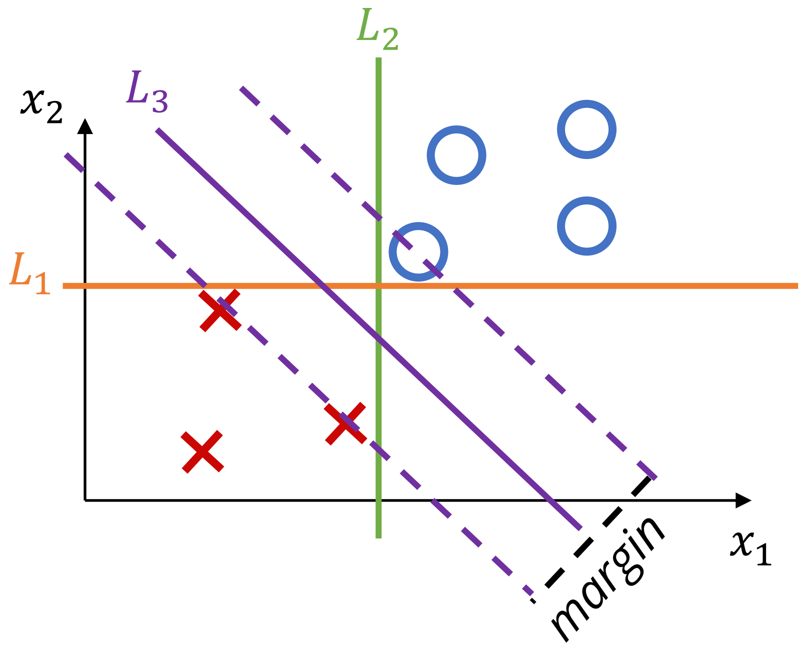
Support vector machines handle this with the "kernel trick"

Optimal separating hyperplane



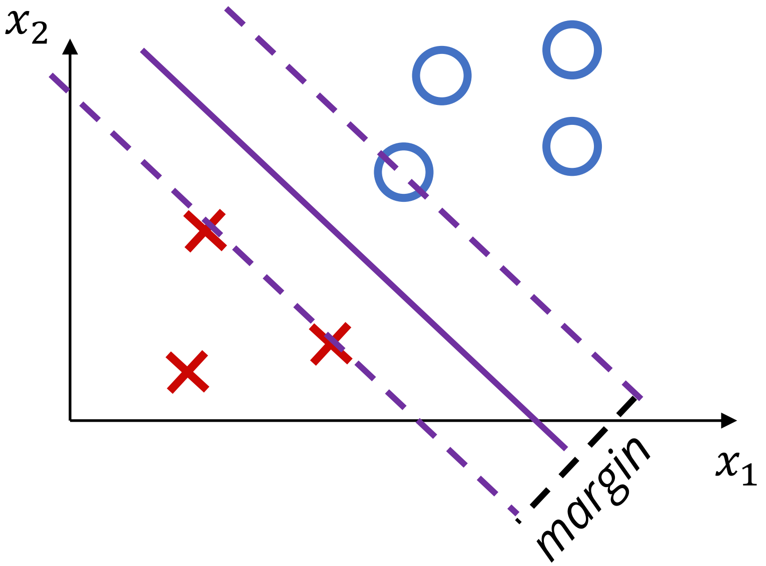
- Which decision boundary to choose? L_3
  - L_1 and L_2 just barely classify examples correctly. We would not expect them to generalize well
- How to design a learning algorithm that chooses L_3 over L_1 and L_2?
  - E.g. you could end up with any of these boundaries using the perceptron algorithm

Maximum margin classifier



- Idea: find the hyperplane with the most "cushion" between itself and the training examples
- Maximize the *margin*

Maximum margin classifier

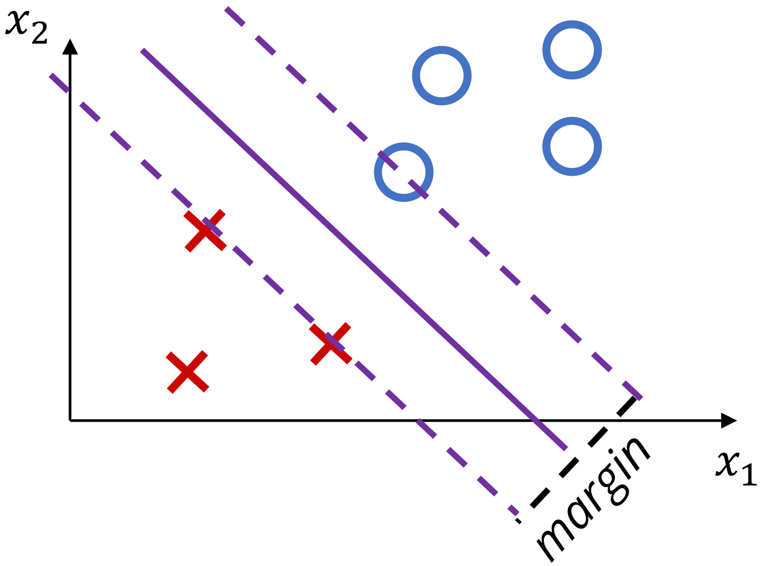


- Assumptions & notations:
  - Binary class: $t \in \{-1, +1\}$
  - Training data is linearly separable
  - Classification function:

$$\begin{cases} +1 & \text{if } \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1 & \text{if } \mathbf{w}^T\mathbf{x} + b < 0 \end{cases}$$

  - No longer assume a dummy feature $x_0 = 1$

Maximum margin classifier



- We want:
  - For every positive training example $\mathbf{x}_+$,

$$\mathbf{w}^T\mathbf{x}_+ + b \geq 1$$

  - For every negative training example $\mathbf{x}_-$,

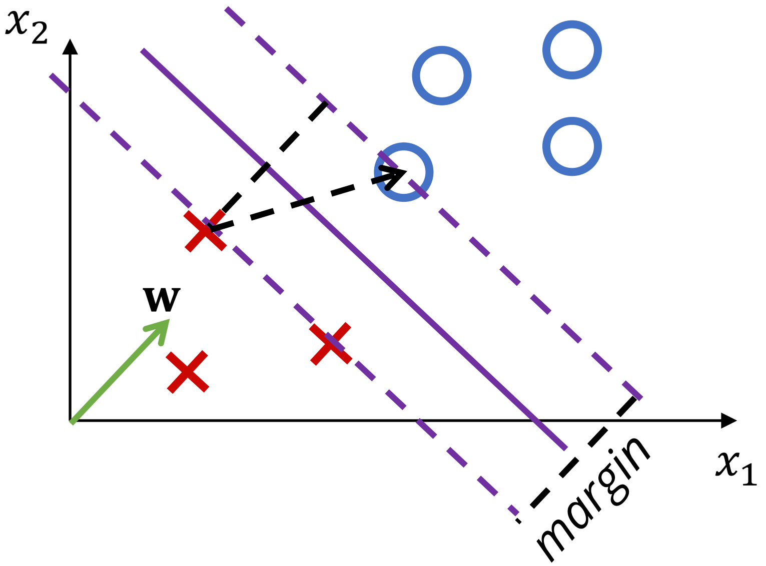
$$\mathbf{w}^T\mathbf{x}_- + b \leq -1$$

- Equivalently, for every training example $(\mathbf{x}_n, t_n)$,

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

$$t_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \quad \forall n$$

Maximum margin classifier



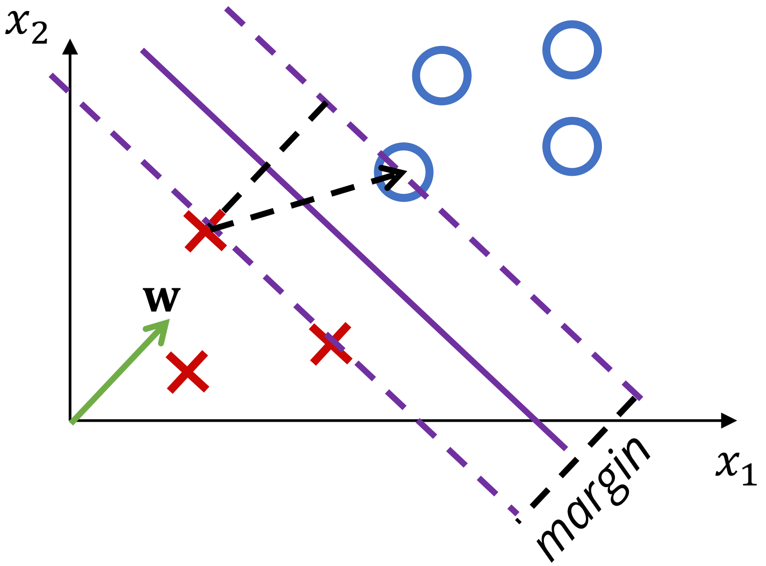
- If $\mathbf{x}_+$ is the positive training example *closest to the hyperplane*, and $\mathbf{x}_-$ the closest negative example,

$$\text{margin} = \frac{\mathbf{w}^T(\mathbf{x}_+ - \mathbf{x}_-)}{\|\mathbf{w}\|}$$

- Recall: $\mathbf{w}^T\mathbf{x}_+ + b = 1$ and $\mathbf{w}^T\mathbf{x}_- + b = -1$
- Then the margin is: $\frac{2}{\|\mathbf{w}\|}$
- Maximum margin classifier:

$$\text{argmax}_{\mathbf{w},b} \frac{2}{\|\mathbf{w}\|} \quad \text{s.t. } t_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \quad \forall n$$

Maximum margin classifier



$$\text{argmax}_{\mathbf{w},b} \frac{2}{\|\mathbf{w}\|} \quad \text{s.t. } t_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \quad \forall n$$

$$\text{argmin}_{\mathbf{w},b} \|\mathbf{w}\| \quad \text{s.t. } t_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \quad \forall n$$

$$\text{argmin}_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t. } t_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \quad \forall n$$

$$\boxed{\text{argmin}_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t. } t_n(\mathbf{w}^T\mathbf{x}_n + b) - 1 \geq 0 \quad \forall n}$$