Lecture 10. The Perceptron

COMP90051 Statistical Machine Learning

Lecturer: Jean Honorio

This lecture

- Perceptron model
  - Introduction to Artificial Neural Networks
  - The perceptron model
- Perceptron training rule
  - Stochastic gradient descent
- Kernel perceptron

The Perceptron Model

A building block for artificial neural networks, yet another linear classifier

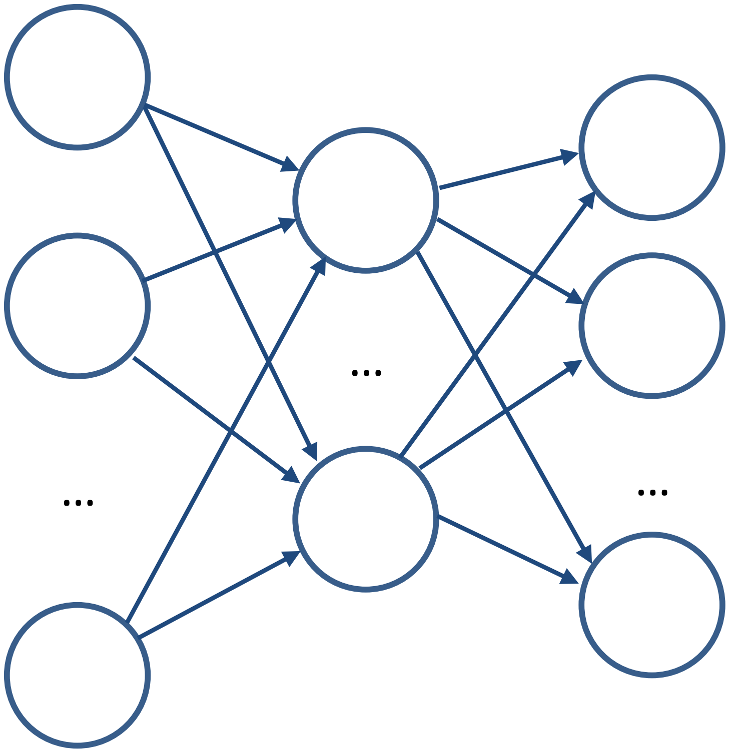
Biological inspiration

- Humans perform well at many tasks that matter
- Originally neural networks were an attempt to mimic the human brain

photo: Alvesgaspar,
Wikimedia Commons, CC3

Artificial neural network

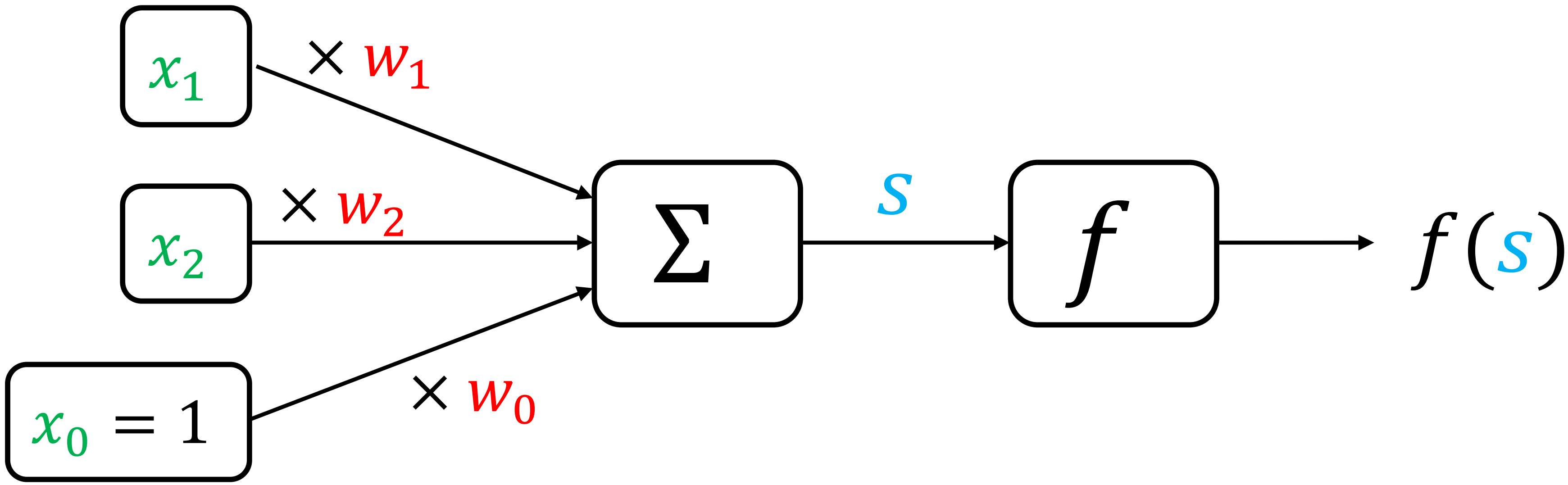
- As a *crude approximation*, the human brain can be thought as a mesh of interconnected processing nodes (neurons) that relay electrical signals
- Artificial neural network is a network of processing elements
- Each element converts inputs to output
- The output is a function (called *activation function*) of a weighted sum of inputs



Outline

- In order to use an ANN we need (a) to design network topology and (b) adjust weights to given data
  - In this subject, we will exclusively focus on task (b) for a particular class of networks called feed forward networks
- Training an ANN means adjusting weights for training data given a pre-defined network topology
- First we will turn our attention to an individual network element, before building deeper architectures

Perceptron model



Compare this model to logistic regression

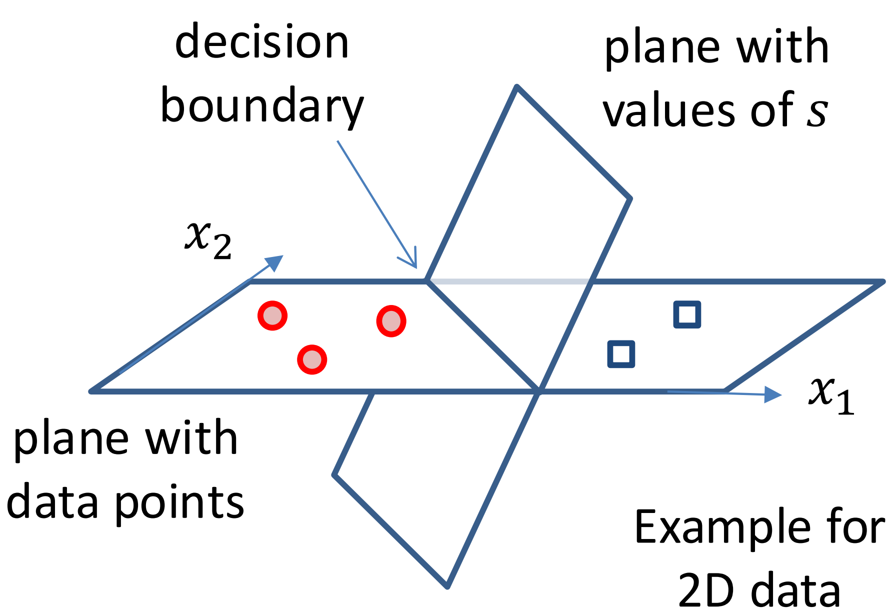
- x_1, x_2 – features/inputs
- w_1, w_2 – synaptic weights
- w_0 – bias weight
- f – activation function

Perceptron is a linear binary classifier

Perceptron is a binary classifier:

Predict class A if $s \geq 0$
Predict class B if $s < 0$
where $s = \boldsymbol{w}'\boldsymbol{x} = \sum_{j=0}^{m} w_j x_j$

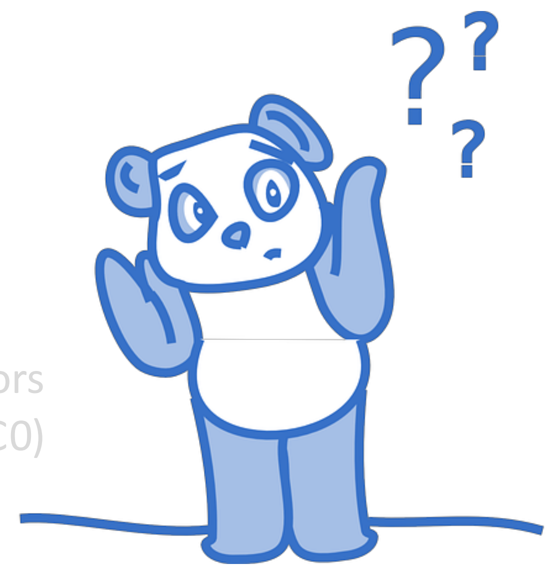
Perceptron is a <u>linear classifier</u>: s is a linear function of inputs, and the decision boundary is linear



Example for 2D data

<u>Exercise</u>: find weights of a perceptron capable of perfect classification of the following dataset

x_1	x_2	y
0	0	Class B
0	1	Class B
1	0	Class B
1	1	Class A

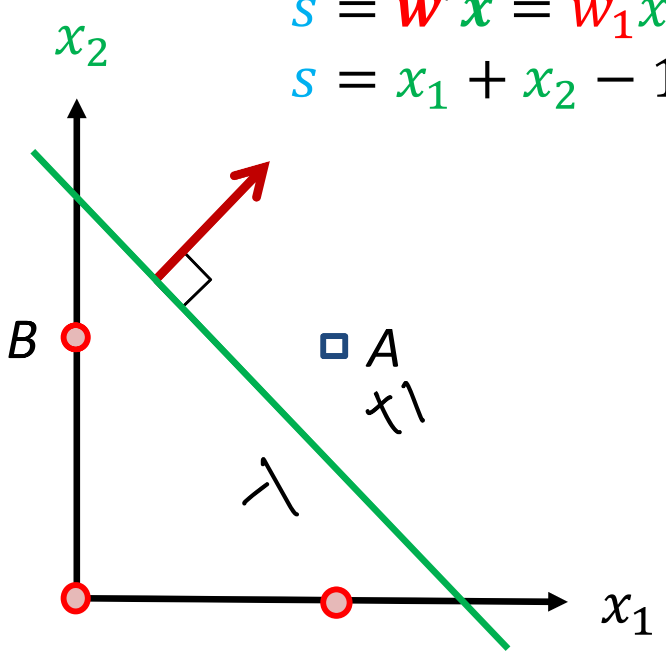


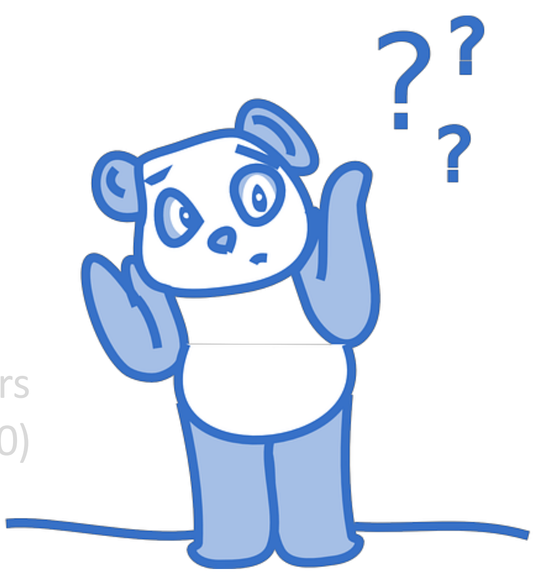
art: OpenClipartVectors at pixabay.com (CC0)

Exercise: find weights of a perceptron capable of perfect classification of the following dataset

x_1	x_2	y	s
0	0	Class B	-1.5
0	1	Class B	-0.5
1	0	Class B	-0.5
1	1	Class A	0.5

$$w_1 = 1, w_2 = 1, w_0 = -1.5$$
$$s = \boldsymbol{w}'\boldsymbol{x} = w_1 x_1 + w_2 x_2 + w_0$$
$$s = x_1 + x_2 - 1.5$$





Mini Summary

- Perceptron
  - Introduction to Artificial Neural Networks
  - The perceptron model

Next: Perceptron training

Perceptron Training Rule

Gateway to stochastic gradient descent.

Convergence guaranteed by convexity.

Loss function for perceptron

- "Training": finds weights to minimise some loss. Which?
- Our task is binary classification. Encode one class as $+1$ and the other as -1. So each training example is now $(\boldsymbol{x}_i, y_i)$, where y_i is either $+1$ or -1
- Recall that, in a perceptron, $s_i = \boldsymbol{w}'\boldsymbol{x}_i = \sum_{j=0}^{m} w_j x_{ij}$, and the sign of s_i determines the predicted class: $+1$ if $s_i > 0$, and -1 if $s_i < 0$
- Consider a single training example.
  * If y_i and s_i have same sign then the example is classified correctly.
  * If y_i and s_i have different signs, the example is misclassified

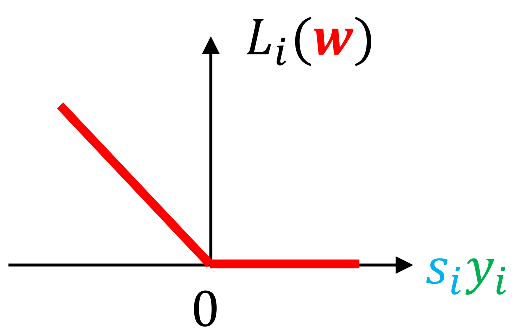
Loss function for perceptron

- The perceptron uses a loss function in which there is no penalty for correctly classified examples, while the penalty (loss) is equal to s_i for misclassified examples*
- Formally:
  - $L_i(\boldsymbol{w}) = 0$ if both s_i, y_i have the same sign
  - $L_i(\boldsymbol{w}) = |s_i|$ if both s_i, y_i have different signs

√预测的 多远值就扣分。 怕扣分

- This can be re-written as $L_i(\boldsymbol{w}) = \max(0, -s_i y_i)$

若 { 预测对了 0 / 预测错了 $|Wx_i|$



$s_i = \boldsymbol{w}'\boldsymbol{x}_i$

主要看loss的gradient ☆

* This is similar, but not identical to the SVM's ***hinge*** loss

Stochastic gradient descent

- Randomly shuffle/split all training examples in B batches
- Choose initial $\boldsymbol{\theta}^{(1)}$
- For t from 1 to T
- For b from 1 to B
- Do gradient descent update <u>using data from batch b</u>

Iterations over the entire dataset are called <u>*epochs*</u>

- Advantage of such an approach: computational feasibility for large datasets

true gradient: average over all training samples' gradients (太难)

Perceptron training algorithm

Choose initial guess $\boldsymbol{w}^{(0)}$, $k = 0$

Value of weight after k changes

For t from 1 to T (epochs)

内部 sample 一个个过

For i from 1 to N (training examples)

Consider example $(\boldsymbol{x}_i, y_i)$

<u>Update</u>*: $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \underbrace{\eta}_{\text{learning rate}} \underbrace{\boldsymbol{\nabla} L_i\left(\boldsymbol{w}^{(k)}\right)}_{\text{gradient of loss}}$

$k = k + 1$

$L_i(\boldsymbol{w}) = \max(0, -s_i y_i)$

$s_i = \boldsymbol{w}' \boldsymbol{x}_i$

η is learning rate

[illegible]假如一对就不更新了.

*There is no derivative when $s_i = 0$, but this case is handled explicitly in the algorithm, see next slides

Perceptron training rule

- We have $\nabla L_i(\mathbf{w}) = \mathbf{0}$ when $s_i y_i > 0$
  - We don't need to do update when sample i is correctly classified

预测对了 ⇒ gradient=0

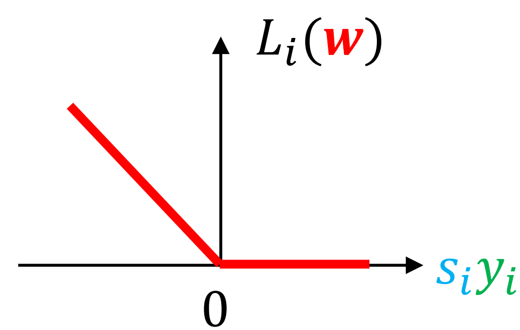
- What is $\nabla L_i(\mathbf{w})$ when $s_i y_i < 0$?

预测错了 ⇒ 要找梯度 $\nabla L_i(w) = -y_i x_i$

  - We need to update when sample i is misclassified
  - We have $\nabla L_i(\mathbf{w}) = \mathbf{x}_i$ when $y_i = -1$ and $s_i > 0$
  - We have $\nabla L_i(\mathbf{w}) = -\mathbf{x}_i$ when $y_i = 1$ and $s_i < 0$
  - Thus $\nabla L_i(\mathbf{w}) = -y_i \mathbf{x}_i$

$(-s_i y_i)$

- $L_i(\mathbf{w}) = \max(0, -s_i y_i)$



$s_i = \mathbf{w}'\mathbf{x}_i$

Perceptron training algorithm

Choose initial guess $\boldsymbol{w}^{(0)}$, $k = 0$

For t from 1 to T (epochs) go through data set several times.

For i from 1 to N (training examples) go through one data point at a time.

$-\eta \nabla L_i(w)$

$(y_i x_i)$

computed s_i

Compute $s_i = \left(\boldsymbol{w}^{(k)}\right)' \boldsymbol{x}_i$

If $s_i y_i \leq 0$: (sample i misclassified) 判断分类错没错. 异号 是否分错.

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \eta y_i \boldsymbol{x}_i$$

$-(\eta \nabla L_i(w))$

$k = k + 1$

$\nabla L_i(w) = -y_i x_i$

($\eta > 0$ is called *learning rate*)

Perceptron training algorithm

Choose initial guess $\mathbf{w}^{(0)}$, $k = 0$

For t from 1 to T (epochs)

For i from 1 to N (training examples)

> Compute $s_i = \left(\mathbf{w}^{(k)}\right)' \mathbf{x}_i$
>
> If $s_i y_i \leq 0$: (sample i misclassified)
>
> $$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \eta y_i \mathbf{x}_i$$
>
> $$k = k + 1$$

Strictly speaking it should be $s_i y_i < 0$ but $\leq$ allows handling the case $\mathbf{w}^{(k)} = \mathbf{0}$

$\mathbf{w}^{(k)}$ represents the value of $\mathbf{w}$ after k updates (useful for theory). If you implement this, just write: $\mathbf{w} = \mathbf{w} + \eta y_i \mathbf{x}_i$

此处是小于等于而非小于

☆ 在initialization阶段 w^0 可能从0开始 ⇒ 此时在第一次前向传播过程中 $s_iy_i=0$，⇒ 若假设 $s_iy_i<0$ ⇒ 永远 w不会被更新，不对

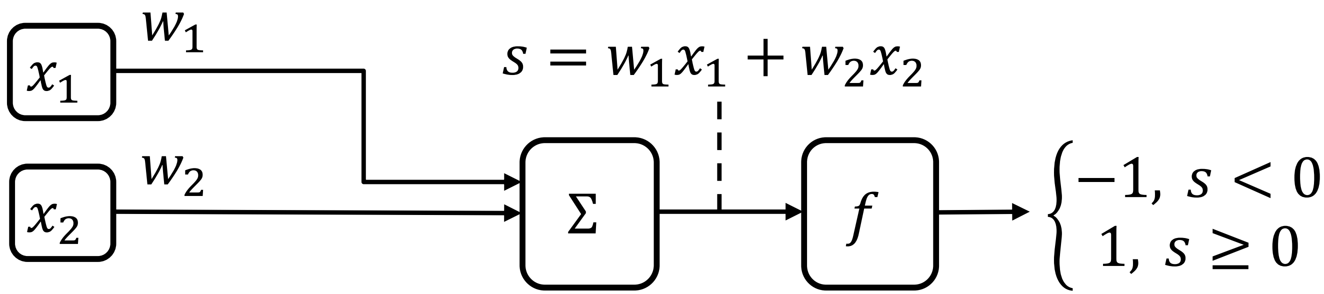
相应修改计算过程

Example 1: Perceptron learning

remove w_0, set $\eta=1$

<u>Basic setup</u>

$w = w + \eta y_i x_i$



(learning rate $\eta = 1$)

* We drop the sample index i to have a simpler notation.

Example 1: Perceptron learning

<u>Start with zero weights, Consider sample 1</u>

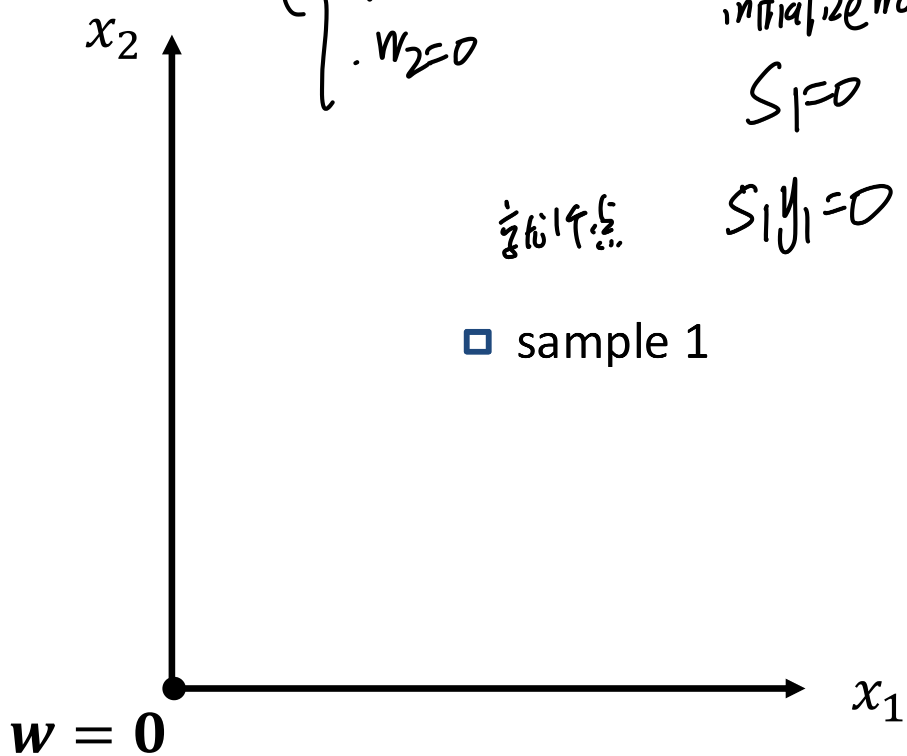
$w_1 = 0$

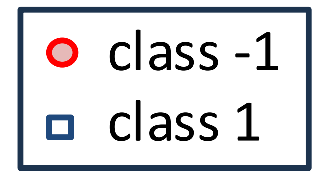
$w_2 = 0$

initialize W at zero:

$s_1 = 0$

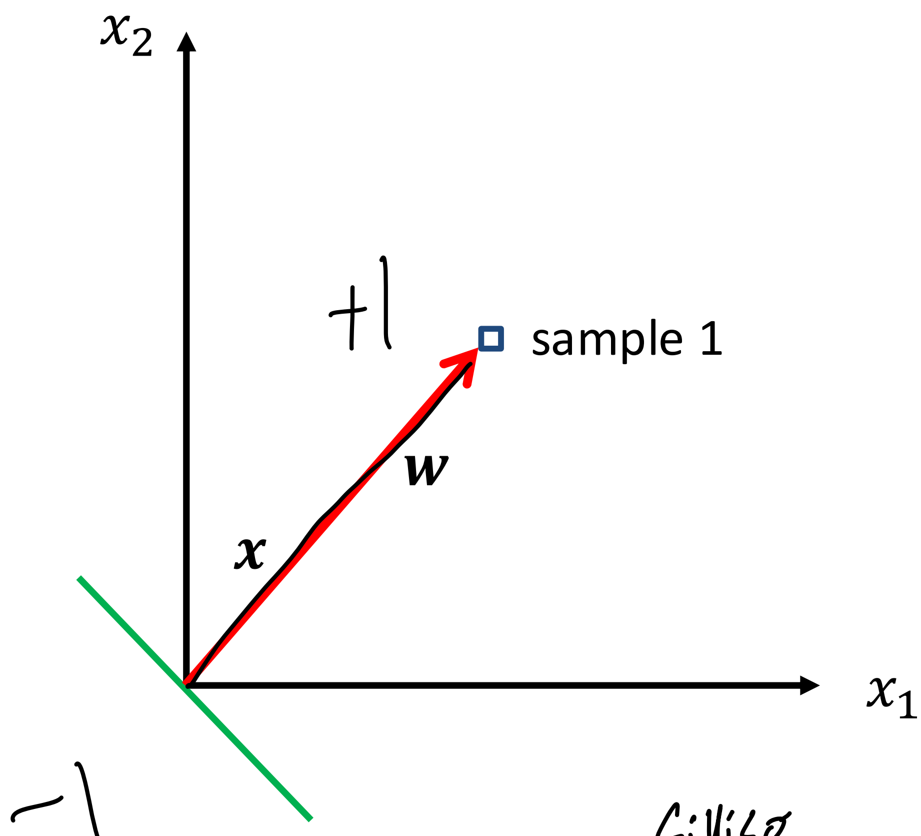
误分点 $s_1 y_1 = 0 \Rightarrow$ 符合 $\leq 0 \Rightarrow$ 更新

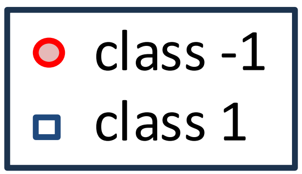




Example 1: Perceptron learning

Update





$s_i y_i \le 0$

Since $\boldsymbol{w} = \boldsymbol{0}$, then update (why?)

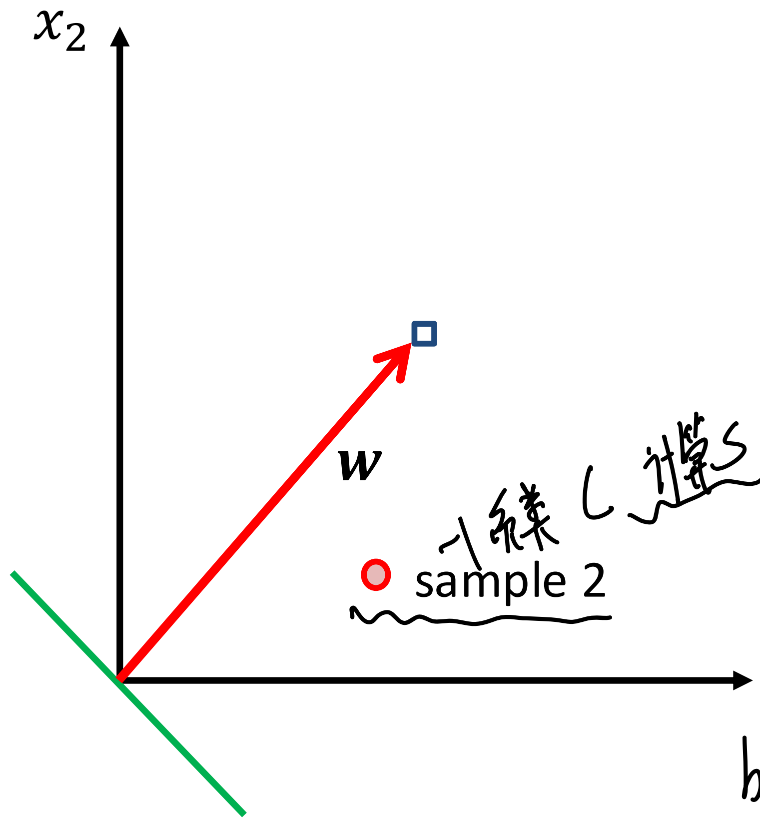
$$\boldsymbol{w} = \boldsymbol{w} + y\boldsymbol{x} = \boldsymbol{x}$$

$0 + 1 \cdot x = x$

* We drop the sample index i to have a simpler notation.

Example 1: Perceptron learning

Consider sample 2



一條（c 樣本S，算出來是正的，預測錯了）

class -1
class 1

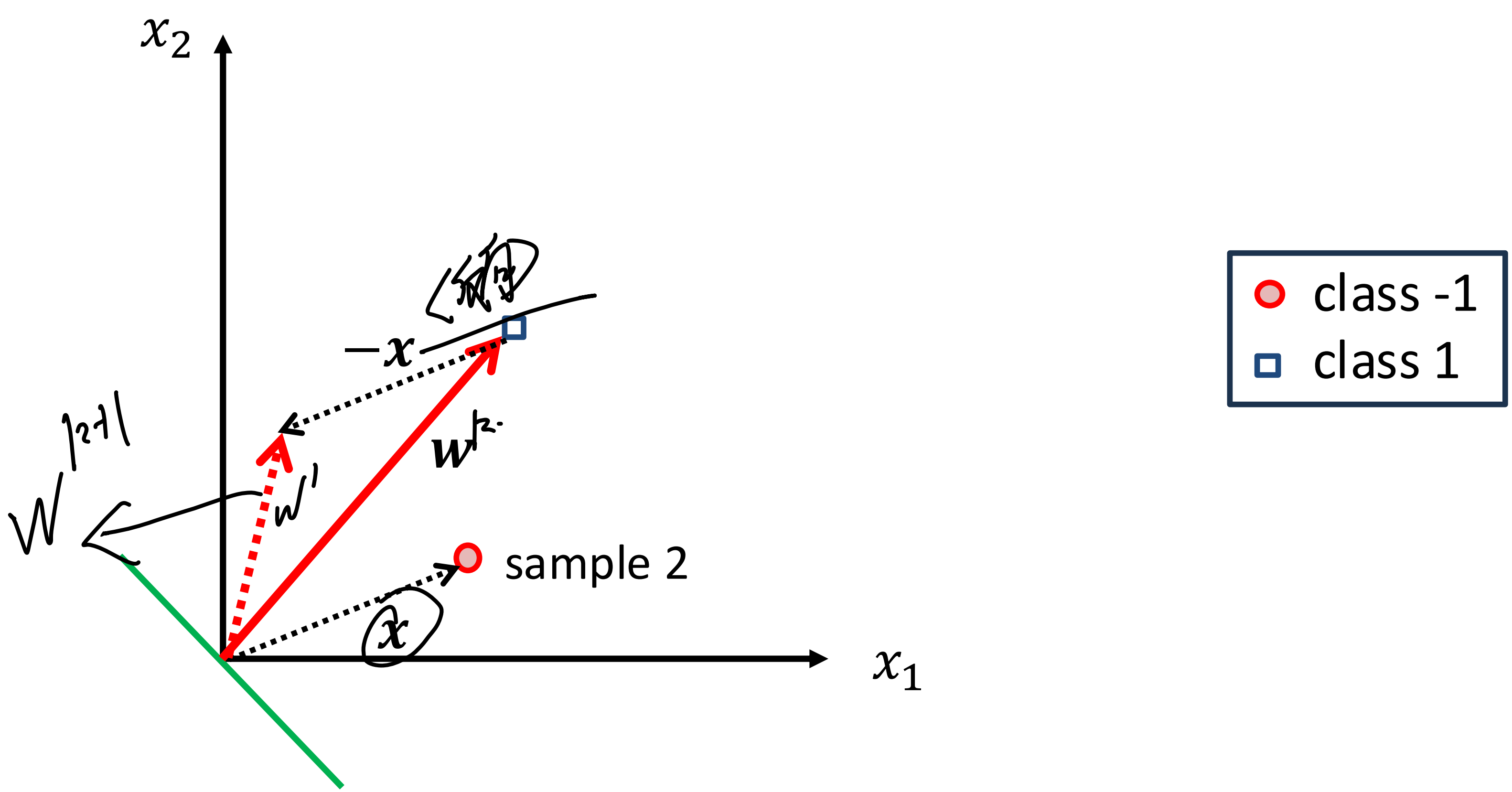
$S_i y_i \le 0$ ⇒ 要update

basically ⇒ 你在绿线上方的样本只要是负的都会预测错 ⇒ S=+1 ⇒ 都要更新 w.

Sample 2 is in the wrong side of the decision boundary

Example 1: Perceptron learning

Update

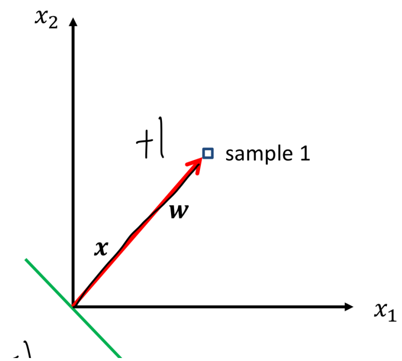


Sample 2 is in the wrong side of the decision boundary, then update

$$\boldsymbol{w} = \boldsymbol{w} + y\boldsymbol{x} = \boldsymbol{w} - \boldsymbol{x}$$

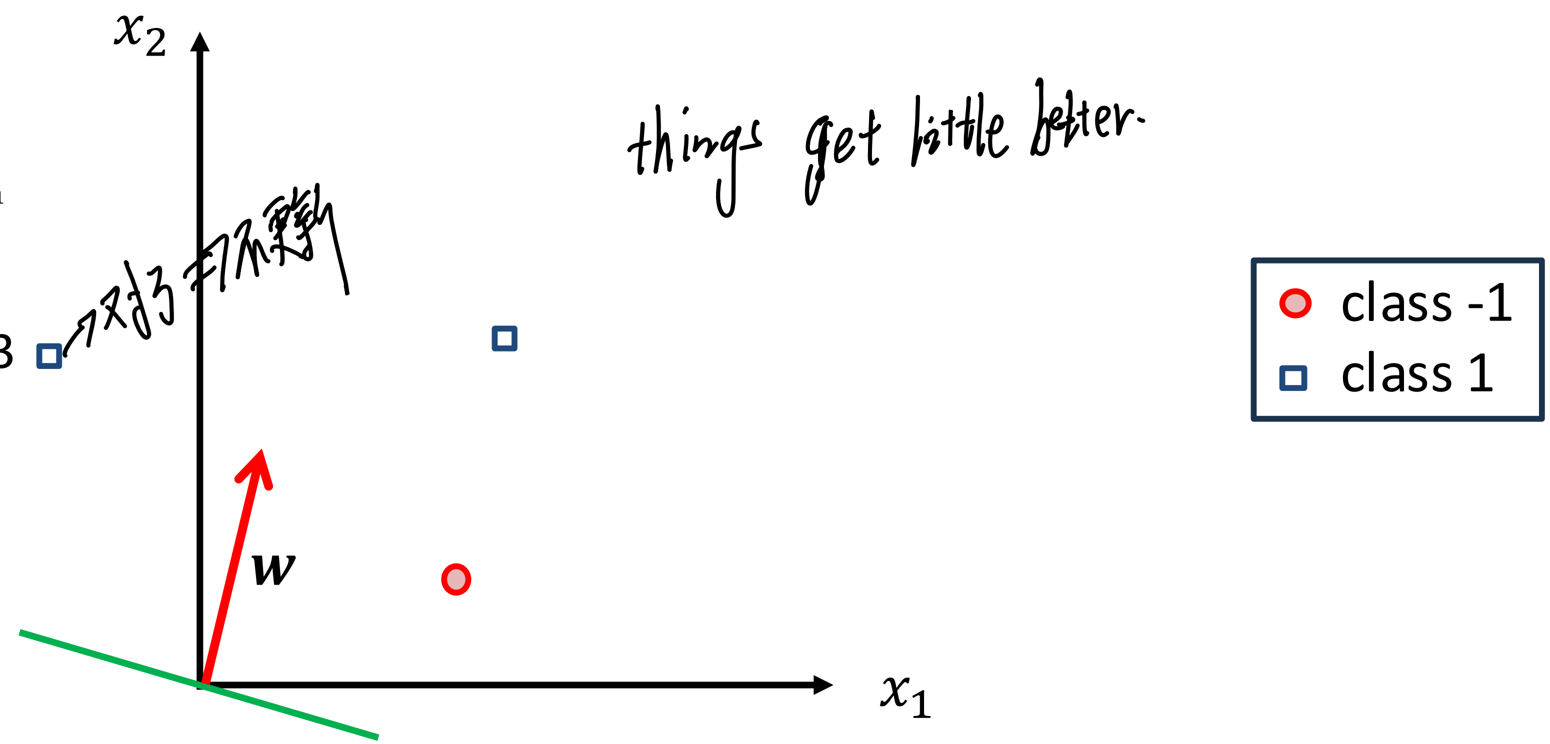
* We drop the sample index i to have a simpler notation.

Example 1: Perceptron learning



Consider sample 3

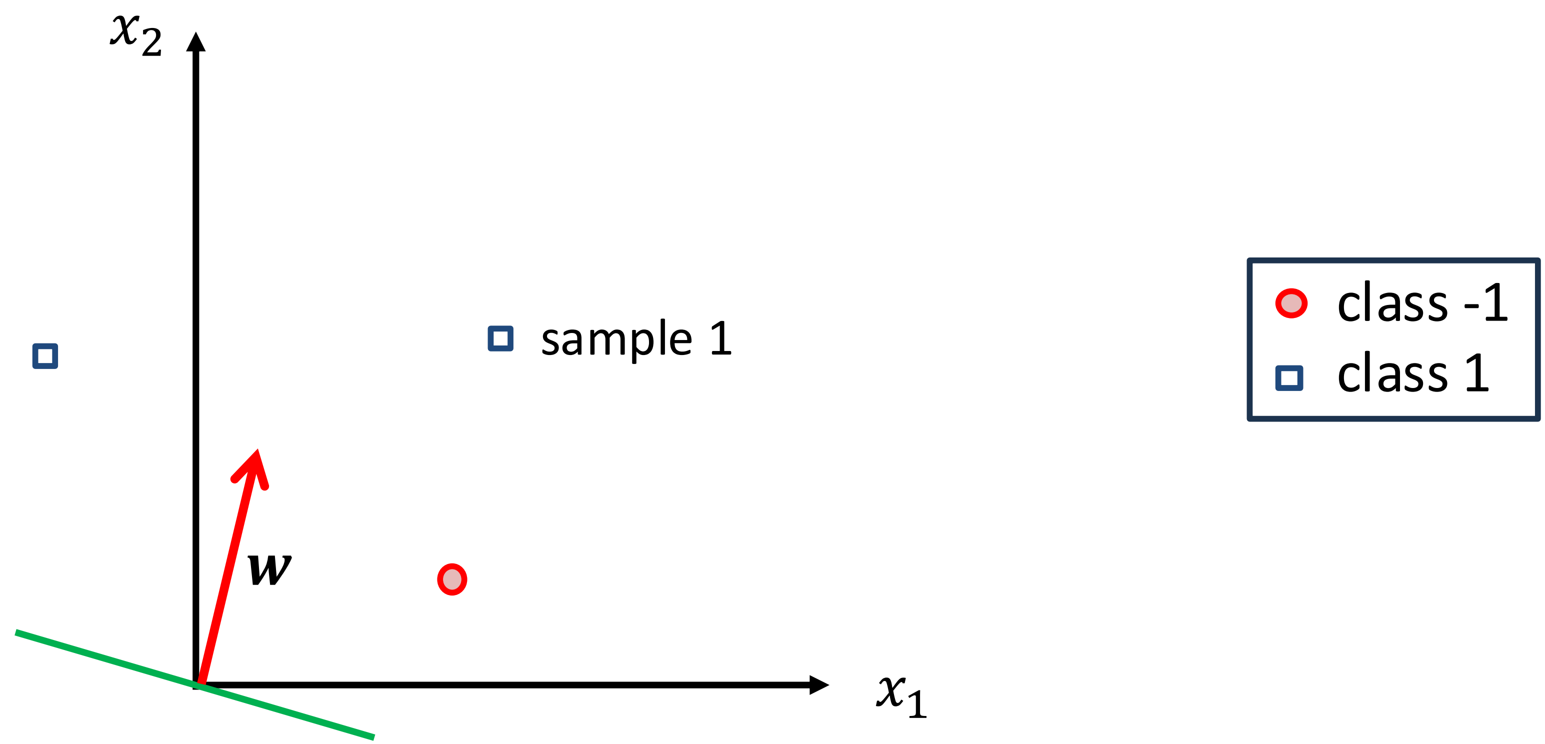
things get little better.



Sample 3 is in the correct side of the decision boundary, no update

Example 1: Perceptron learning

第2batch Consider sample 1 again



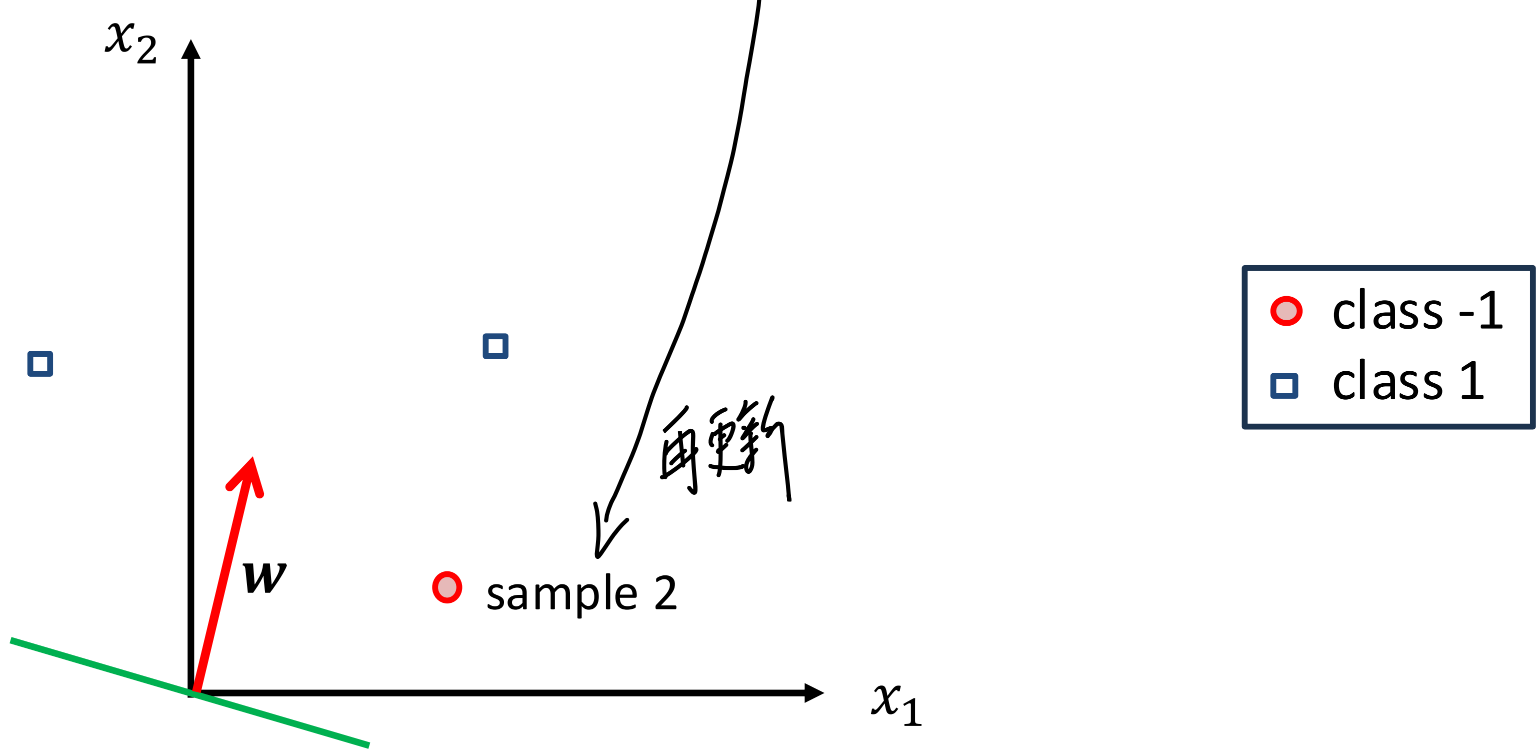
Sample 1 is in the correct side of the decision boundary, no update

* We drop the sample index i to have a simpler notation.

Example 1: Perceptron learning

<u>Consider sample 2 again</u>

再更新

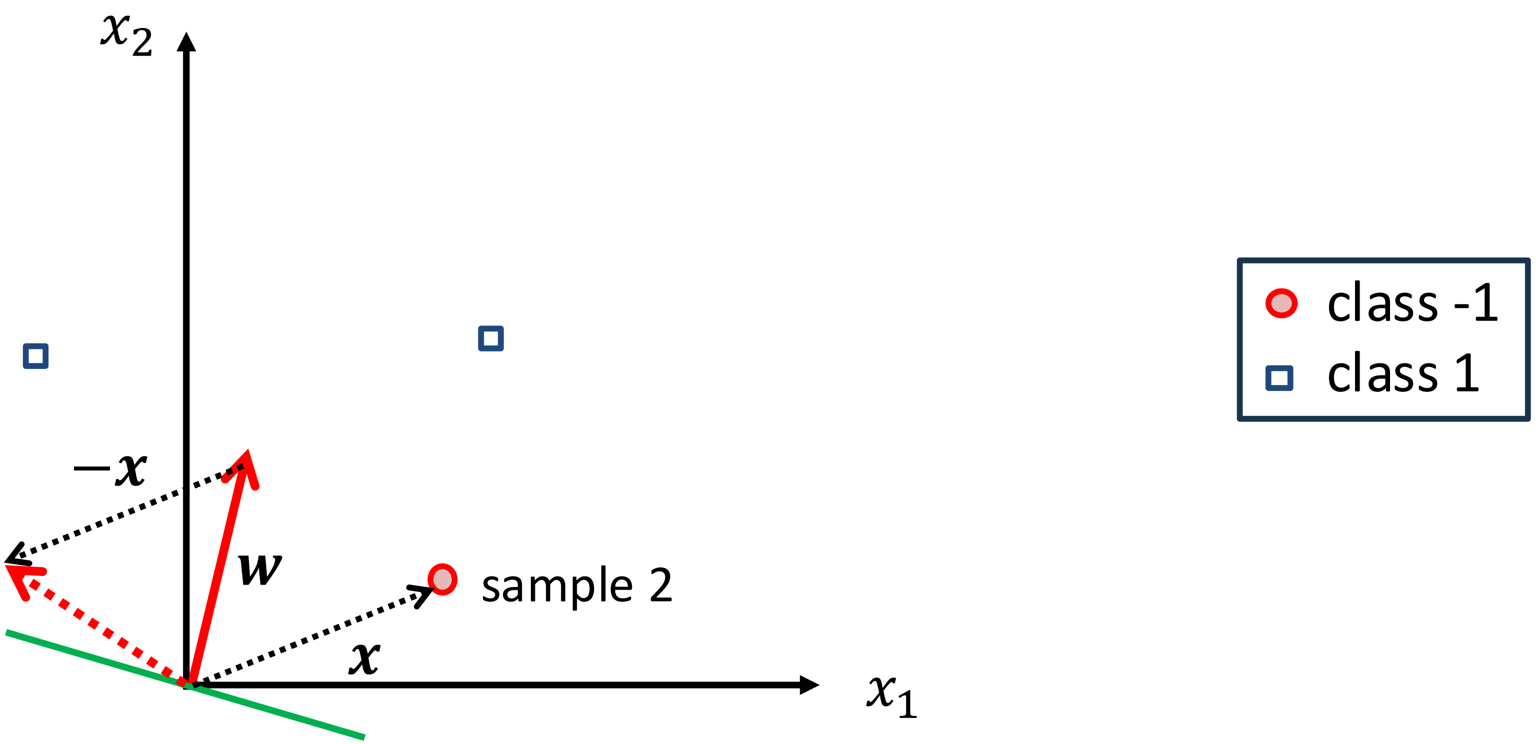


Sample 2 is in the wrong side of the decision boundary, then update

* We drop the sample index i to have a simpler notation.

Example 1: Perceptron learning

Update



Sample 2 is in the wrong side of the decision boundary, then update

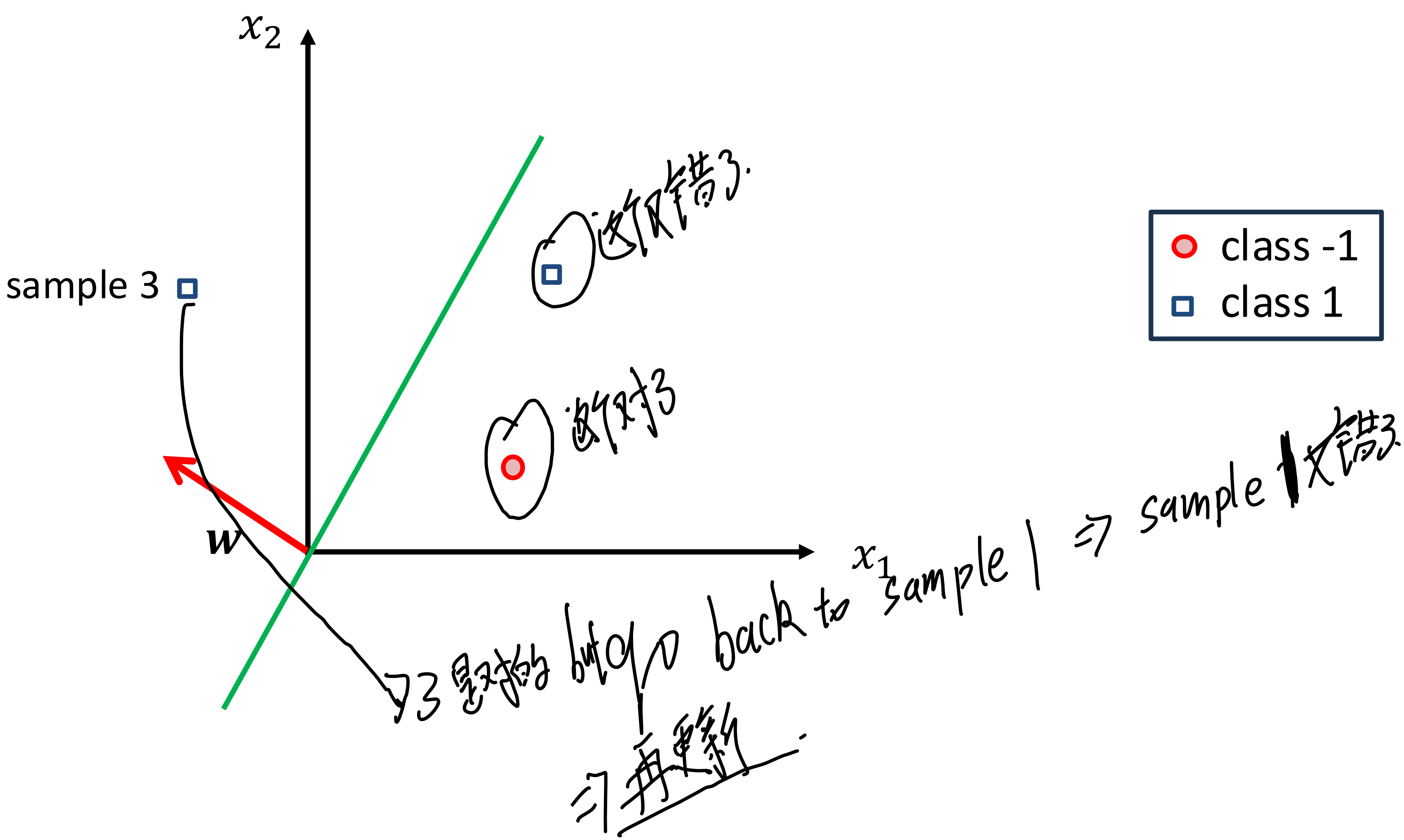
$$\boldsymbol{w} = \boldsymbol{w} + y\boldsymbol{x} = \boldsymbol{w} - \boldsymbol{x}$$

y=-1

* We drop the sample index i to have a simpler notation.

Example 1: Perceptron learning

And we continue…



Perceptron training algorithm

Choose initial guess $\boldsymbol{w}^{(0)}$, $k = 0$

For t from 1 to T (epochs)

把所有点分好的时候最终会停 ⇒ 更新法 好像

For i from 1 to N (training examples)

> Compute $s_i = \left(\boldsymbol{w}^{(k)}\right)' \boldsymbol{x}_i$
>
> If $s_i y_i \leq 0$: (sample i misclassified)
>
> $$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \eta y_i \boldsymbol{x}_i$$
>
> $$k = k + 1$$

Convergence Theorem: if the training data is linearly separable, the algorithm is guaranteed to converge to a solution. That is, there exist a finite k such that $L\left(\boldsymbol{w}^{(k)}\right) = 0$

Perceptron convergence theorem

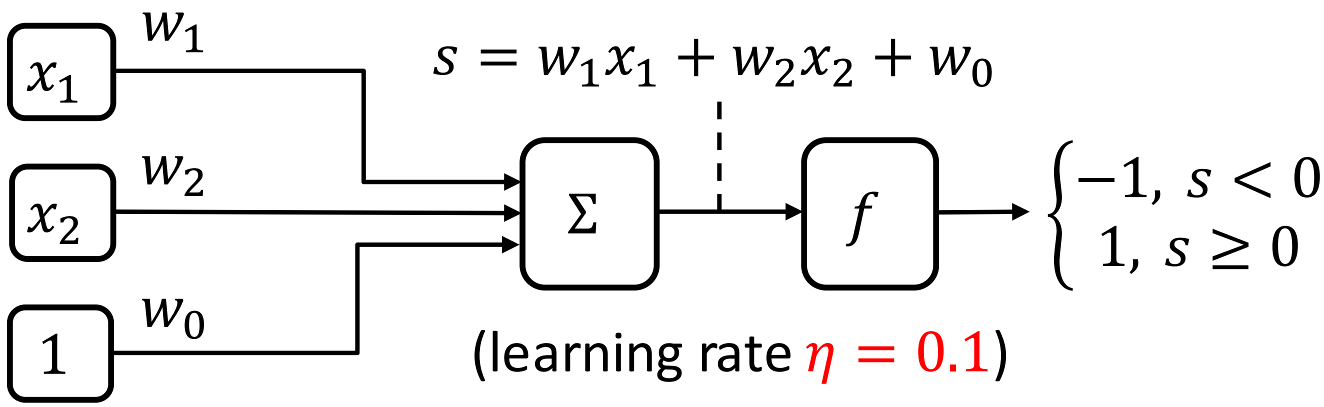
- Assumptions
  - Linear separability: There exists $\boldsymbol{w}^*$ so that $\frac{y_i(\boldsymbol{w}^*)'\boldsymbol{x}_i}{\|\boldsymbol{w}^*\|} \geq \gamma$ for all training data $i = 1, \dots, N$ and some positive γ.
  - Bounded data: $\|\boldsymbol{x}_i\| \leq R$ for $i = 1, \dots, N$ and some finite R.
- Proof sketch (for $\eta = 1$)
  - Assumes $\boldsymbol{w}^{(0)} = \boldsymbol{0}$ to
    - Establish that $(\boldsymbol{w}^*)'\boldsymbol{w}^{(k)} \geq k\gamma\|\boldsymbol{w}^*\|$
    - Establish that $\left\|\boldsymbol{w}^{(k)}\right\|^2 \leq kR^2$
  - Note that $1 \geq \cos\left(\boldsymbol{w}^*, \boldsymbol{w}^{(k)}\right) = \frac{(\boldsymbol{w}^*)'\boldsymbol{w}^{(k)}}{\|\boldsymbol{w}^*\|\left\|\boldsymbol{w}^{(k)}\right\|} \geq \frac{k\gamma\|\boldsymbol{w}^*\|}{\|\boldsymbol{w}^*\|\sqrt{k}R}$
  - Take the left-most and right-most equations $1 \geq \frac{k\gamma\|\boldsymbol{w}^*\|}{\|\boldsymbol{w}^*\|\sqrt{k}R}$
  - Rearranging we get $k \leq \frac{R^2}{\gamma^2}$

Pros and cons of perceptron learning

- If the data is linearly separable, the perceptron training algorithm will converge to a correct solution
  - There is a formal proof ← good!
  - It will converge to some solution (separating boundary), one of infinitely many possible ← bad!
- However, if the data is not linearly separable, the training will fail completely rather than give some approximate solution
  - Ugly ☹

Example 2: Perceptron learning

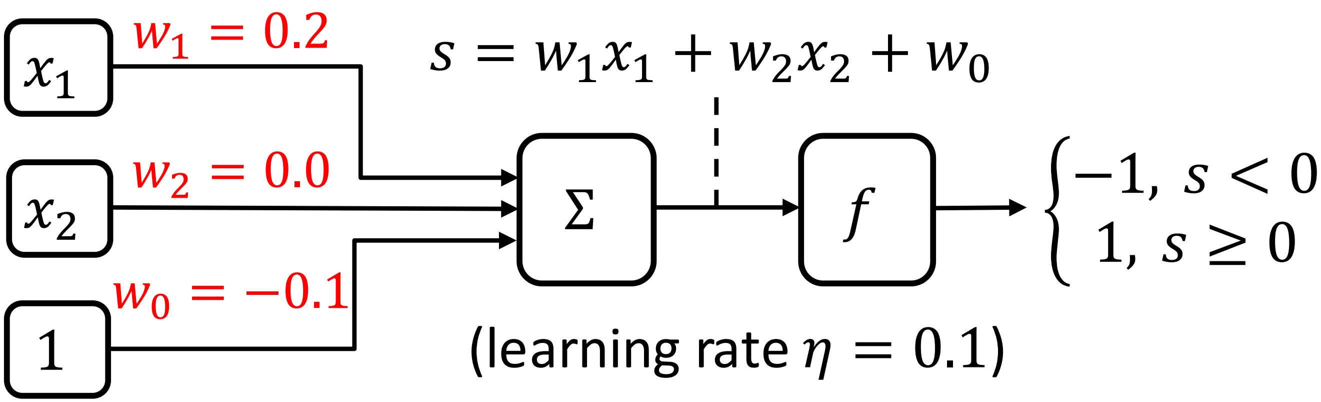
Basic setup



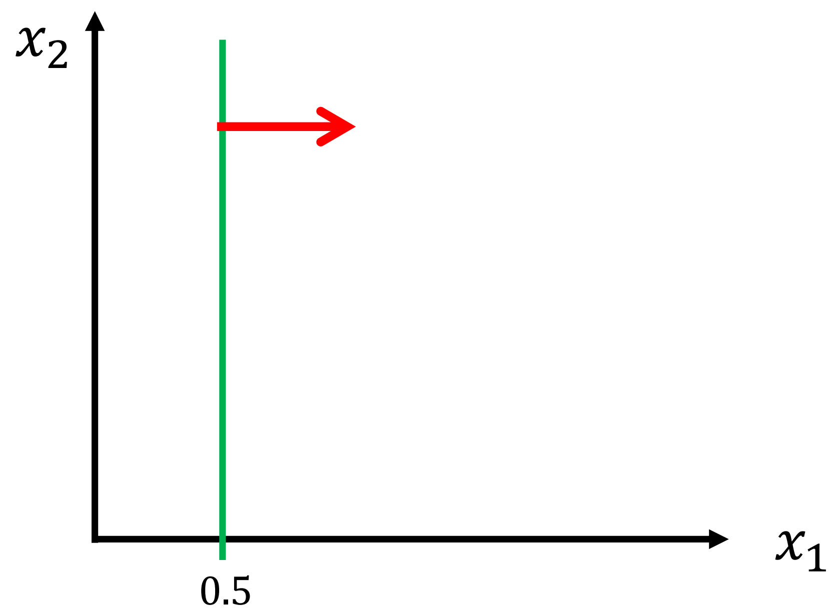
* We drop the sample index i to have a simpler notation.

Example 2: Perceptron learning

Start with random weights



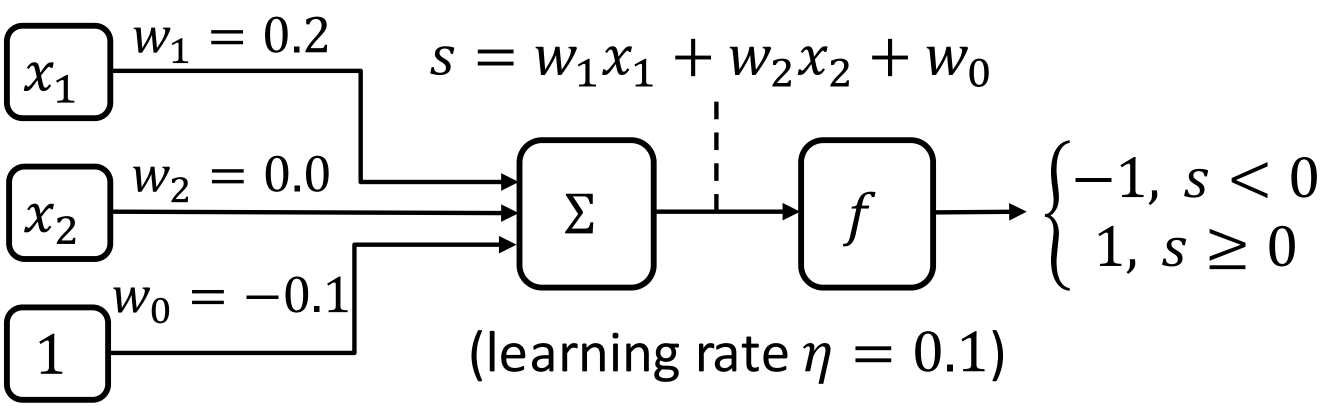
(learning rate $\eta = 0.1$)

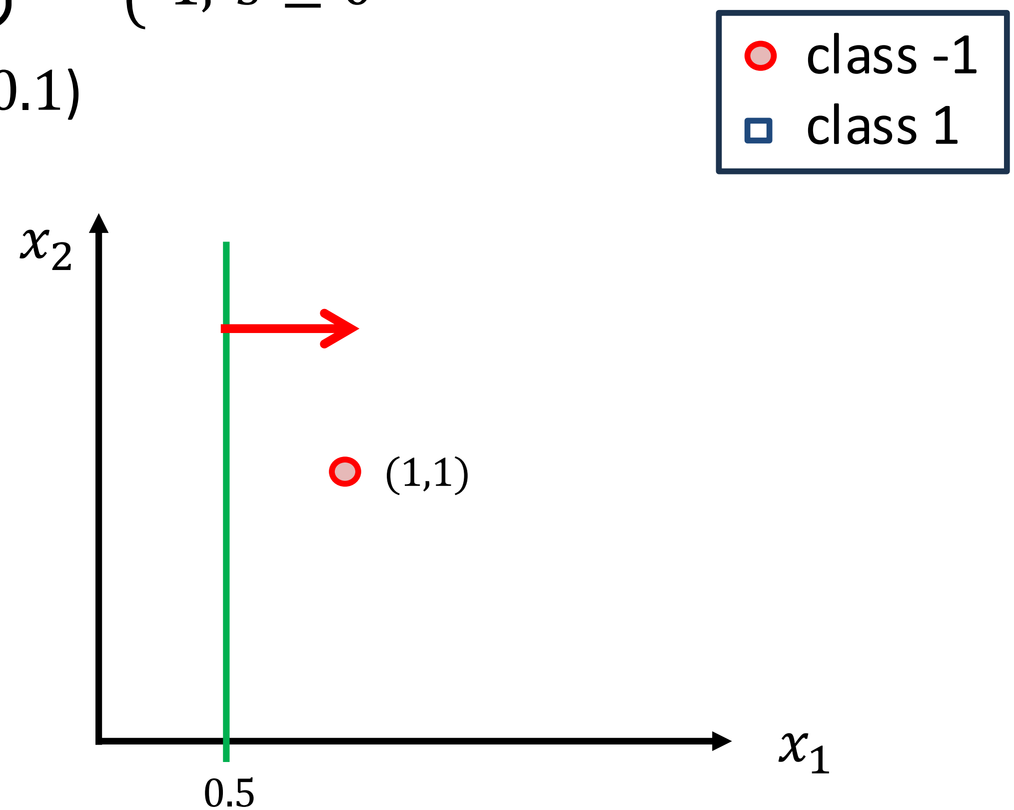


* We drop the sample index i to have a simpler notation.

Example 2: Perceptron learning

Consider training example 1





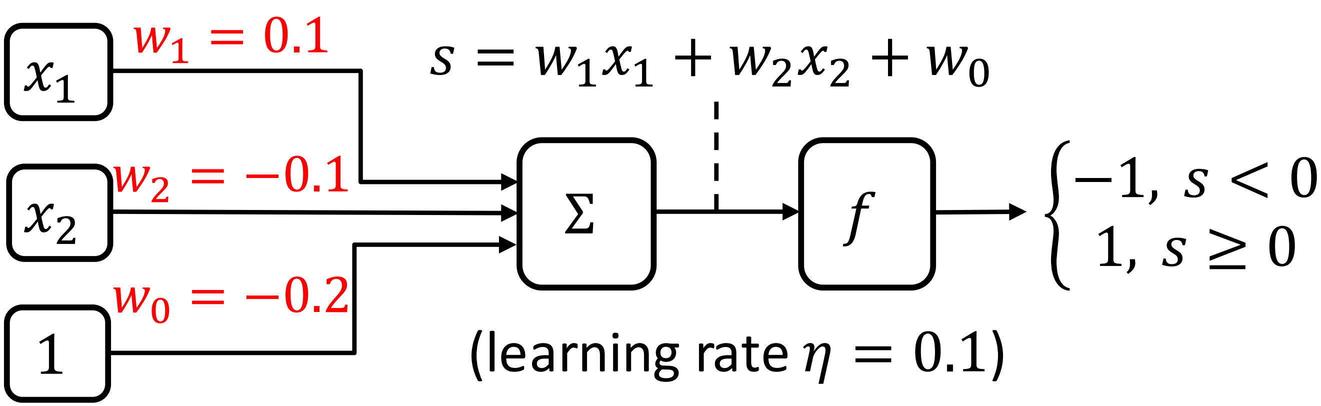
$$y(0.2x_1 + 0.0x_2 - 0.1) = -0.1 \leq 0$$

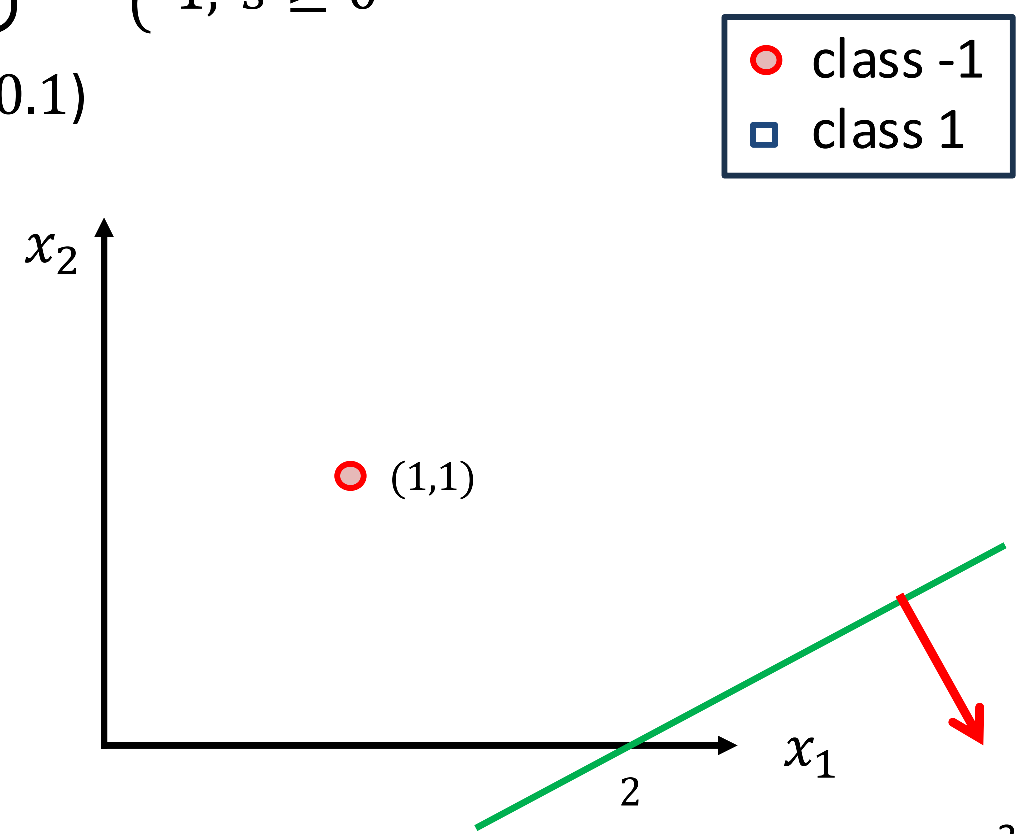
$$\begin{aligned} w_1 &\leftarrow w_1 - \eta x_1 = 0.1 \\ w_2 &\leftarrow w_2 - \eta x_2 = -0.1 \\ w_0 &\leftarrow w_0 - \eta = -0.2 \end{aligned}$$

* We drop the sample index i to have a simpler notation.

Example 2: Perceptron learning

Update weights

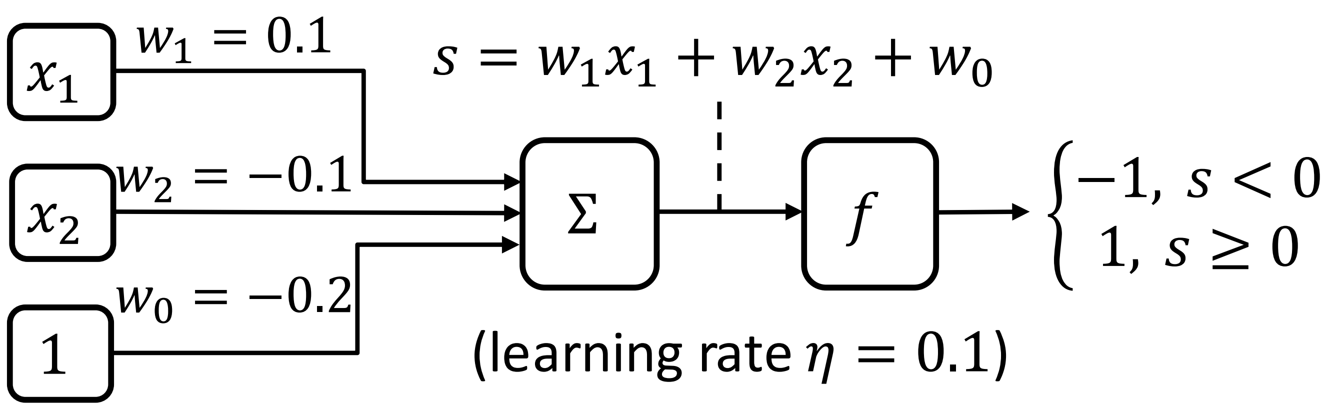




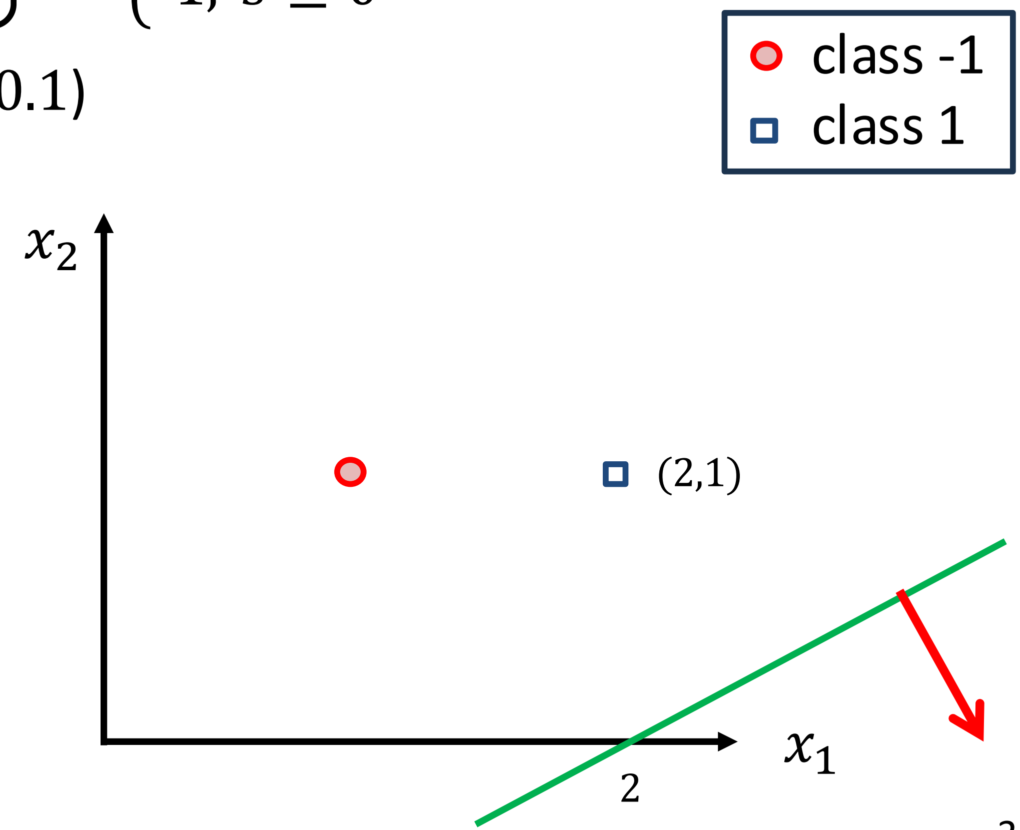
* We drop the sample index i to have a simpler notation.

Example 2: Perceptron learning

Consider training example 2



(learning rate $\eta = 0.1$)



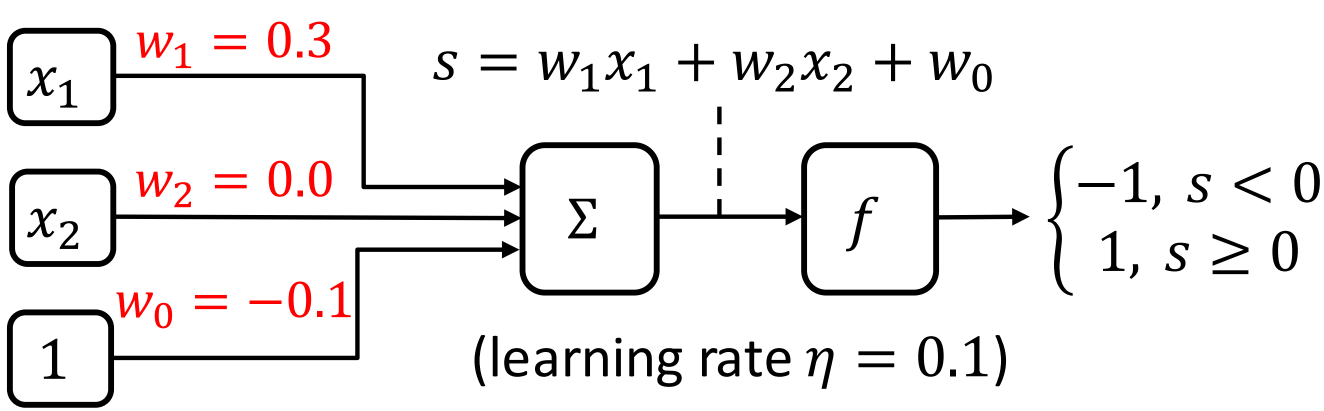
$$y(0.1x_1 - 0.1x_2 - 0.2) = -0.1 \leq 0$$

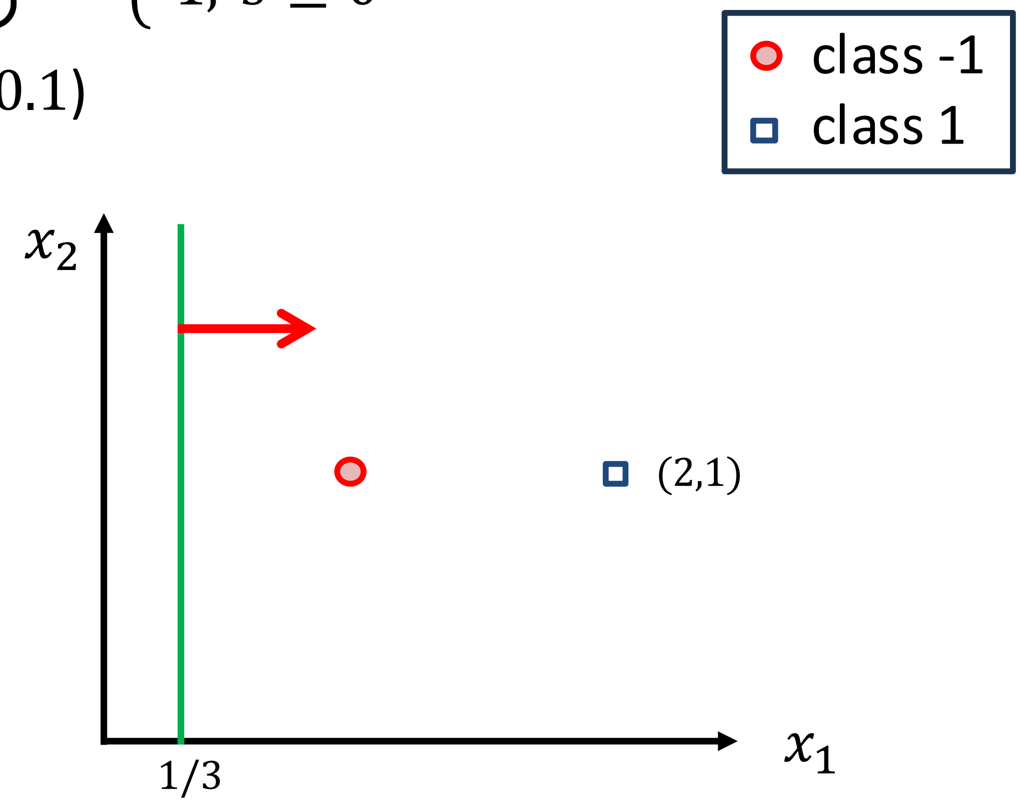
$$\begin{aligned} w_1 &\leftarrow w_1 + \eta x_1 = 0.3 \\ w_2 &\leftarrow w_2 + \eta x_2 = 0.0 \\ w_0 &\leftarrow w_0 + \eta = -0.1 \end{aligned}$$

* We drop the sample index i to have a simpler notation.

Example 2: Perceptron learning

Update weights

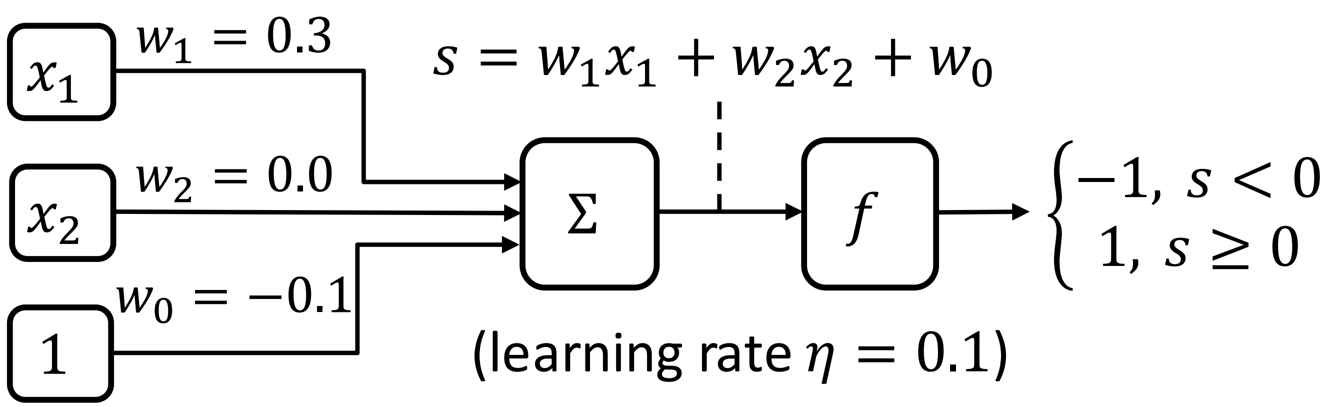




* We drop the sample index i to have a simpler notation.

Example 2: Perceptron learning

Further examples

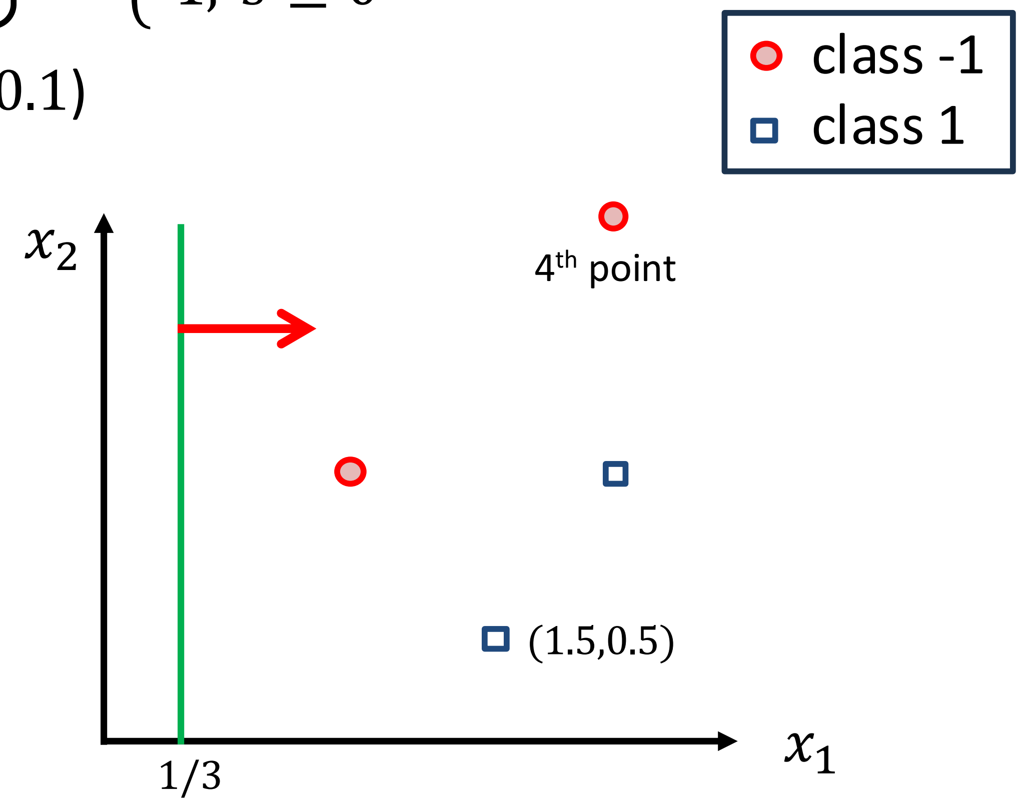


$y(0.3x_1 - 0.0x_2 - 0.1) = 0.35 > 0$

3rd point: correctly classified

4th point: incorrect, update

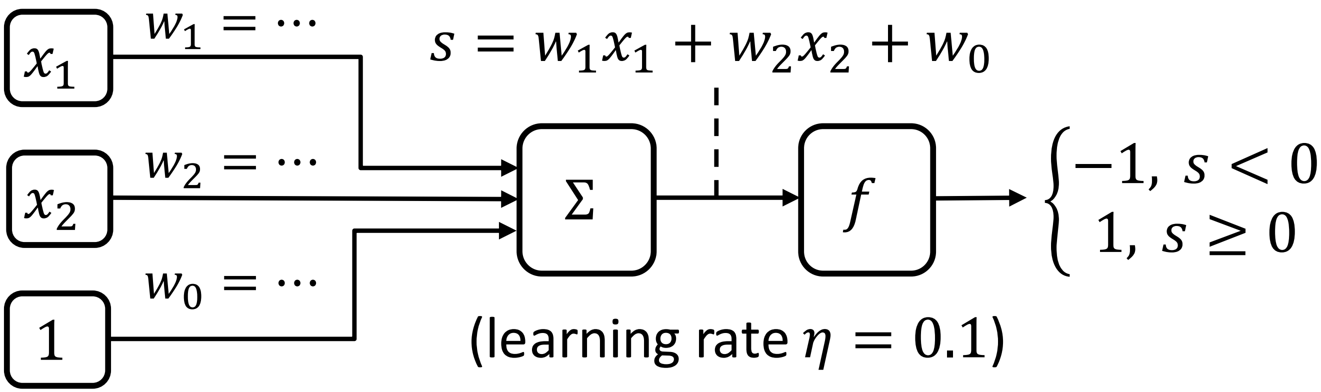
etc.



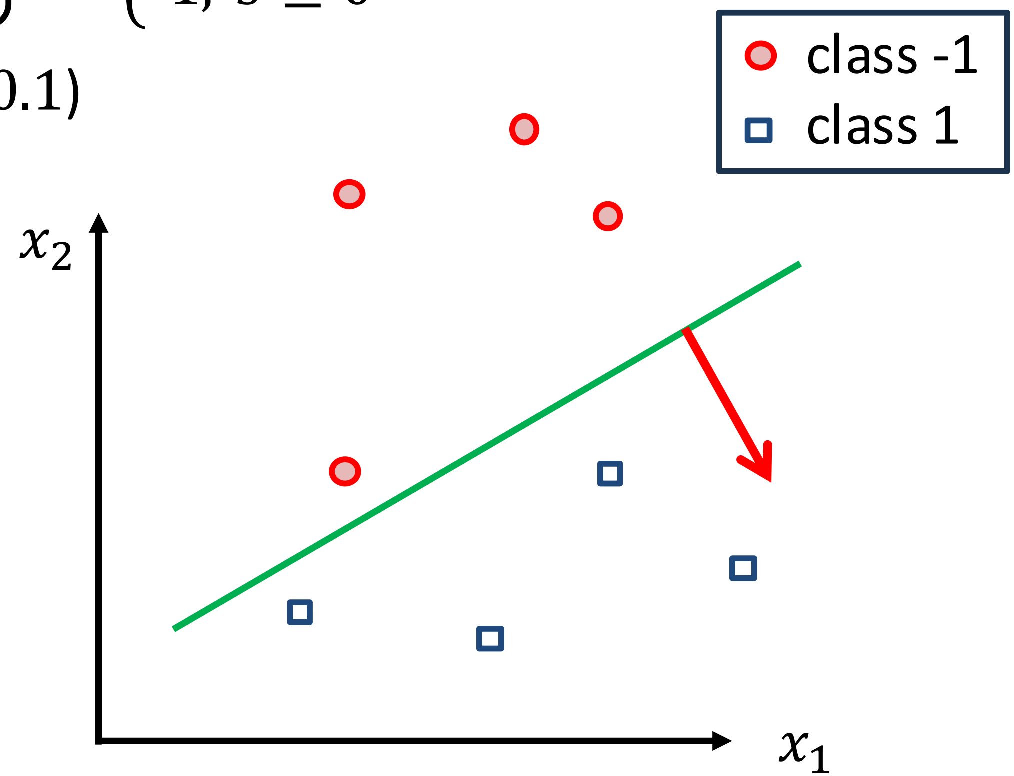
* We drop the sample index i to have a simpler notation.

Example 2: Perceptron learning

Further examples



Eventually, all the data will be correctly classified (provided it is linearly separable)



* We drop the sample index i to have a simpler notation.

Mini Summary

- Perceptron loss function
- Stochastic gradient descent
- Perceptron training rule
  - Perceptron convergence theorem

Next: Kernel perceptron

Kernel Perceptron

Another example of a kernelizable learning algorithm (like the SVM).

Perceptron training rule: Recap

> Compute $s_i = \left(\boldsymbol{w}^{(k)}\right)' \boldsymbol{x}_i$
>
> If $s_i y_i \leq 0$: (sample i misclassified)
>
> $$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \eta y_i \boldsymbol{x}_i$$
>
> $$k = k + 1$$

Suppose weights are initially set to $\boldsymbol{w}^{(0)} = \boldsymbol{0}$

Suppose the algorithm misclassifies sample 1, 7, 29, and 1 again

$W^k - \eta \Delta L_i(w) = W^k + \eta x_i y_i$

$\Delta L_i(w) = \frac{\partial(-s_i y_i)}{\partial w_i}$

哪个点错了加哪个 $= -x_i y_i$

First update: $\boldsymbol{w}^{(1)} = \eta y_1 \boldsymbol{x}_1$

Second update: $\boldsymbol{w}^{(2)} = \eta y_1 \boldsymbol{x}_1 + \eta y_7 \boldsymbol{x}_7$

Third update: $\boldsymbol{w}^{(3)} = \eta y_1 \boldsymbol{x}_1 + \eta y_7 \boldsymbol{x}_7 + \eta y_{29} \boldsymbol{x}_{29}$

Third update: $\boldsymbol{w}^{(4)} = 2\eta y_1 \boldsymbol{x}_1 + \eta y_7 \boldsymbol{x}_7 + \eta y_{29} \boldsymbol{x}_{29}$ etc.

☆ 哪个点错了加哪个 ☆

Accumulating updates: Data enters via dot products

- Weights always take the form $\boldsymbol{w} = \sum_{j=1}^{N} \alpha_j y_j \boldsymbol{x}_j$, where $\boldsymbol{\alpha}$ some coefficients

  α: summation of how many η I have to assign to that sample when it was misclassified

- Perceptron weights always linear comb. of data!

  how many learning rate I was adding to that.

- Recall that prediction for a new point $\boldsymbol{x}$ is based on sign of $\boldsymbol{w}'\boldsymbol{x}$

- Substituting $\boldsymbol{w}$ we get $\boldsymbol{w}'\boldsymbol{x} = \sum_{j=1}^{N} \alpha_j y_j \boldsymbol{x}_j'\boldsymbol{x}$

- The dot product $\boldsymbol{x}_j'\boldsymbol{x}$ can be replaced with a kernel ☆

keeping $\alpha \cdot y$

$\alpha_j y_j x_j' x$

Kernelised perceptron training rule

Set $\boldsymbol{\alpha} = \mathbf{0}$

For t from 1 to T (epochs)

if sample 被分类正确, 对应的 α 是 0

分类错 one time → 加 η

错两次 two times → 2η

For i from 1 to N (training examples)

Compute $s_i = \sum_{j=1}^{N} \alpha_j y_j \mathbf{x}_j' \mathbf{x}_i$ kernel

If $s_i y_i \le 0$: (sample i misclassified)

对 i 分错一次 就对 α_i 加一次 η

$\alpha_i \leftarrow \alpha_i + \eta$

($\eta > 0$ is called *learning rate)*

Kernelised perceptron training rule

Set $\boldsymbol{\alpha} = \mathbf{0}$

For t from 1 to T (epochs)

For i from 1 to N (training examples)

帶標籤 kernel -> inner product

Compute $s_i = \sum_{j=1}^{N} \alpha_j y_j K(\boldsymbol{x}_j, \boldsymbol{x}_i)$

預測錯了

If $s_i y_i \leq 0$: (sample i misclassified)

$\alpha_i \leftarrow \alpha_i + \eta$

加 learning rate

α_i 时调在数发生错误的次数

部份是

Mini Summary

- Accumulating weight updates leads to linear combinations of data
- Predictions are dot products with data
- Can replace these with kernel evaluations
- Leads to kernel perceptron with kernel training rule

Next time: Deep learning

$$\Leftarrow \sup_{g\in G}\left(Eg(z)-\frac{1}{n}\sum_{i=1}^{n}g(z_i)\right)\le 2R_n(G)+\sqrt{\frac{\log\frac{1}{\delta}}{2n}}$$

$$\Leftarrow \Phi(z_1,\dots,z_n)=\sup_{g\in G}\left(Eg(z)-\frac{1}{n}\sum_{i=1}^{n}g(z_i)\right)$$

$$\le E\Phi(z_1,\dots,z_n)+\sqrt{\frac{\log\frac{1}{\delta}}{2n}}$$

$$\le 2R_n(G)$$

$$\Leftarrow \Phi(z_1,\dots,z_n)-E_{z_1,\dots,z_n}\Phi(z_1,\dots,z_n)$$

$$\le \cancel{\varepsilon} \qquad \varepsilon=\sqrt{\frac{\log\frac{1}{\delta}}{2n}} \qquad \delta=\exp(-2n\varepsilon^2)$$

$$\Leftarrow Eg(z)\le\frac{1}{n}\sum_{i=1}^{n}g(z_i)+2\hat{R}_S(G)+\sqrt{\frac{\log\frac{2}{\delta}}{2n}}$$

Apply McDiamid to $\hat{R}_S(G)$ as function of $S=(z_1,\dots,z_n)$

Bounded difference property:

$$\hat{R}_{z_1,\dots,z_n}(G) - \hat{R}_{z_1,\dots,z_{k-1},z_k',\dots}(G)$$

$$= E_\sigma\left[\sup_{g\in G} \frac{1}{n}\sum \sigma_i g(z_i) - \sup_{g\in G}\left(\frac{1}{n}\sum \sigma_i g(z_i) + \frac{1}{n}\sigma_k g(z_k')\right)\right]$$

$$\leq E_\sigma \sup_{g\in G}\left(\frac{1}{n}\sigma_k g(z_k) - \frac{1}{n}\sigma_k g(z_k')\right) \leq \frac{1}{n}$$

---

how to apply to binary classification?

Lemma: [M+ Lemma 3.4]

Let H: ~~f~~ family of functions $h: \mathcal{X} \to \{-1,+1\} =: \mathcal{Y}$

G: loss function

$\mathcal{G} := \{ g: \underline{\mathcal{Z}} \to \{0,1\}, (x,y) \mapsto 1_{h(x)\neq y} : h \in H \}$

$: \mathcal{X}\times\mathcal{Y}$

$z_1 = (x_1, y_1), \dots, z_n = (x_n, y_n)$

Then $\underbrace{R_{z_1,\dots,z_n}(\mathcal{G}) = \frac{1}{2}\hat{R}_{x_1,\dots,x_n}(H)}$ ☆

Proof see [M+]

Simple corollary $R_n(G) = \frac{1}{2} R_n(H)$

Theorem: H family of functions

P_X dist on X

$\delta \in (0,1)$

with p $\geq 1-\delta$ over P_X^n, for all $h \in H$ simultaneously

$$\begin{cases} R(h) \leq \hat{R}(h) + R_n(H) + \sqrt{\frac{\log \frac{1}{\delta}}{2n}} \\ R(h) \leq \hat{R}(h) + \hat{R}_{x_1,\dots,x_n}(H) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2n}} \end{cases}$$

Proof:

Let $g \in G$ be associated to h i.e. $g(x,y) \to \mathbb{1}_{h(x) \neq y}$

Then $R(h) = E g(z)$ $\quad \hat{R}(h) = \frac{1}{n}\sum_{i=1}^{n} g(z_i)$

Lemma: $2\hat{R}_{z_1,\dots,z_n}(G) = \hat{R}_{x_1,\dots,x_n}(H)$

$2R_n(G) = R_n(H)$

$\Pi_H(m) = 2^m$

maximim ~~set~~

of this set of dichotomy.

If $\exists S$ with $|S| = m$ shattered by H, i.e. $|\Pi_H(S)| = 2^m$ then $\Pi_H(m) = \max_{S \subset X^m} |\Pi_H(S)| = 2^m$

$d_{VC}(H) := \max\{m : \Pi_H(m) = 2^m\}$

Note: ① does not imply all sets of size $\le d$ are shattered by H.

② often easier to compute the growth function GF or RC

③ connect to VC dimension & growth function. Via Sauer's Lemma.

Sauer's Lemma: Let H be a class of function $h: X \to \{-1, 1\}$ and let $d := d_{VC}(H)$. Then $\Pi_H(m) \le \sum \binom{m}{i}$

upper bound for growth function

Proof: (by induction on the sum $m+d$) Define $\Phi := \sum_{i=0}^{d} \binom{m}{i}$

Basic case: $m=0$ (labelling on an empty set)

$\Pi_H(0) = 1$

$\Phi_d(0) = \binom{m}{0} = 1$

if $d=0$, no data set can be shattered

$\Pi_H(m) \le 1$

$\Phi_0(m) = \binom{m}{0} = 1$

if $m=1$

(24): if $m=1$ (1-elt set) $d=0$

$\Pi_H(1) \le$

$\Phi_d(m) = \sum_{i=0}^{d} \binom{m}{i}$

$\to \Pi_{(H)}(1) \le 2$.

inductive step:

assume for some $m, d \ge 1$, we have $m'+d' < m+d$.

and $\Pi_{H'}(m') \le \Phi_{d'}(m') = \sum_{i=0}^{d'} \binom{m'}{i}$,

where $H' = H$ restricted to $m' = m-1$ (H_1是H的subset).

$d' = d_{vc}(H') = d$ or $d-1$

Consider: labellings induced by H on any set $S = \{x_1, \cdots, x_m\}$

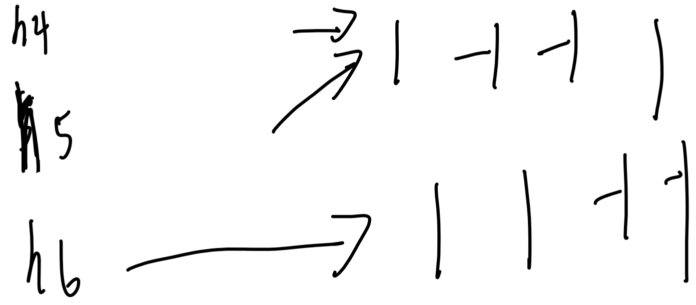
wlog Let $S_1 = \{x_1, \cdots, x_{m-1}\}$

$= S \setminus \{x_m\}$

Let H_1 be the set of hypothesis restricted to S_1

$\Pi_H(S)$ $\Pi_{H_1}(S_1)$ 少 -3

	x_1	x_2	x_3	x_4	x_5					
h_1	-1	1	1	+1	-1	$\to$	-1	1	1	-1
h_2	-1	1	1	+1		$\to$				
$h3$						$\to$	-1	1	1	1



H 同 S 不同

H shatters set S, H_1 shatter a set S

H_1 shatter a set $\Rightarrow$ then so does H.

$$d_{VC}(H_1) \le d_{VC}(H) = d$$