



Machine Learning

Chapter 6: Logistic Regression

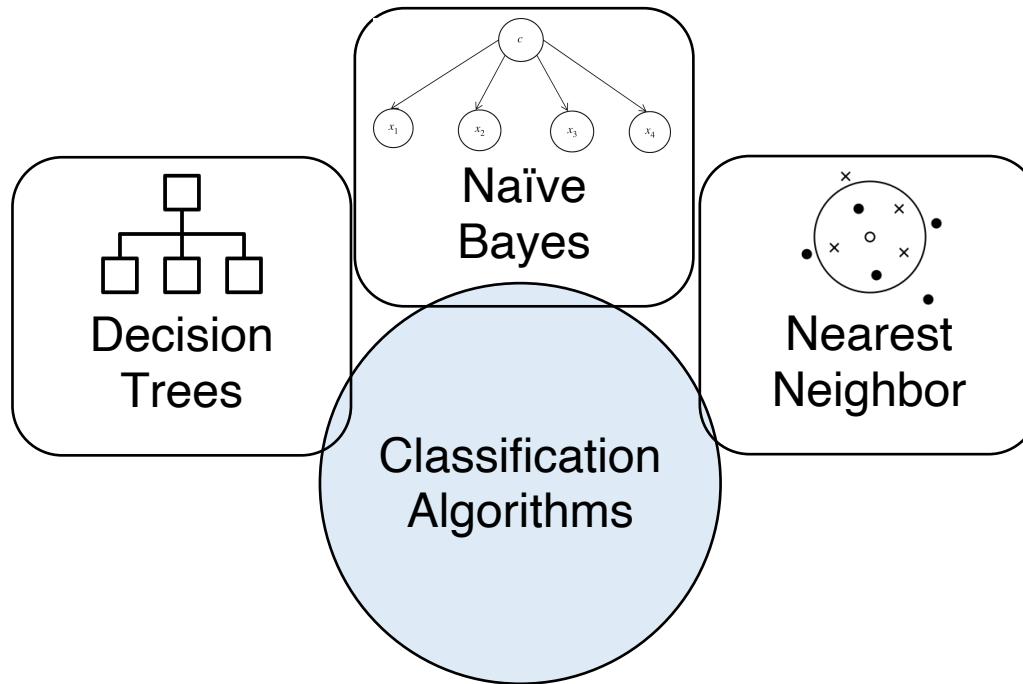
Fall 2021

Instructor: Xiaodong Gu





The family of classification



Recall: Bayes Classification

The **likelihood** (class-conditional density) of x in class C_i

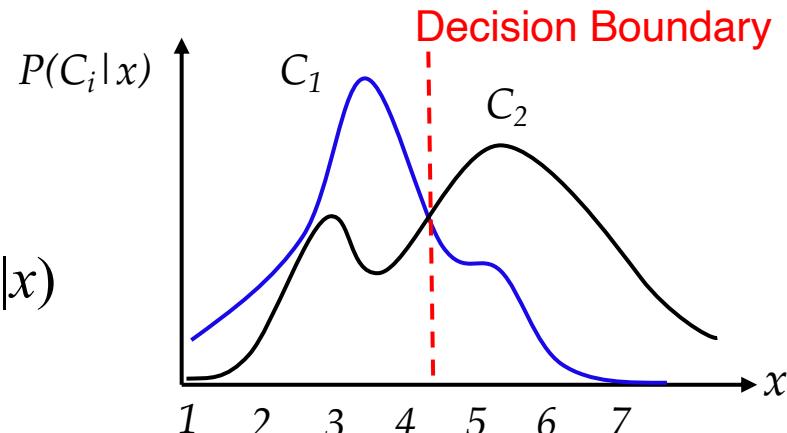
The **prior probability** of class C_i

- Can be simply estimated by frequencies on the training set

$$P(C_i|x) = \frac{p(x|C_i)P(C_i)}{p(x)}, \quad i=1, 2$$

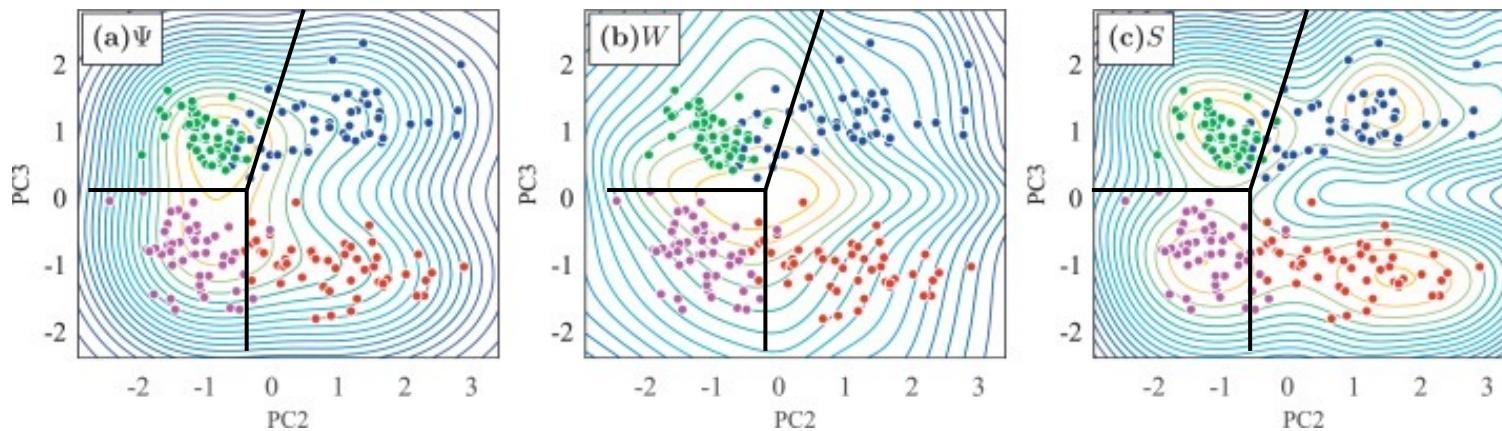
the **posterior probability** of class C_i

- By **MAP**, $\hat{C} = \arg \max_i p(C_i|x)$
- The **decision rule**:



Classify x into C_i if $p(C_i|x) > p(C_j|x)$ else C_j .

Do we really need the probability density?



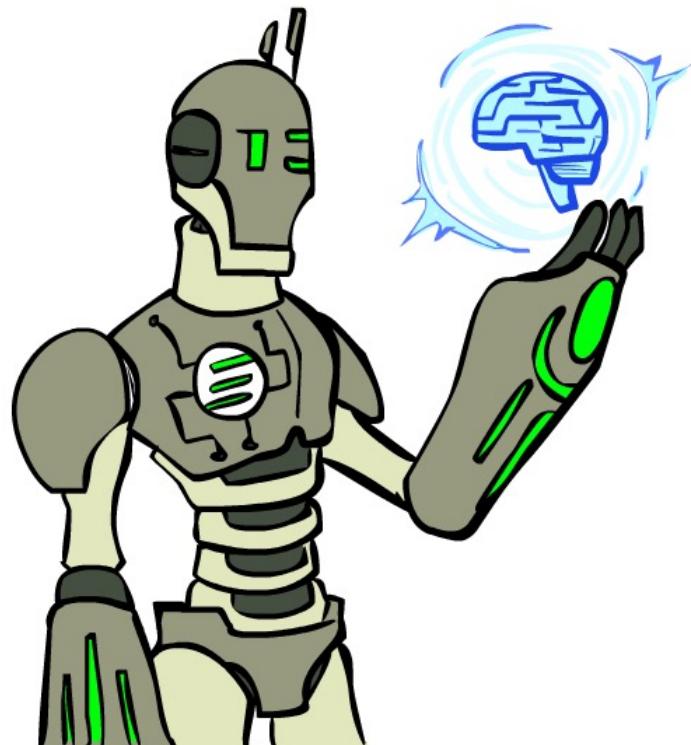
Why not estimate the decision boundary directly?

Today



Linear Classifiers

- Likelihood vs. Discriminative
- (Linear) Discriminant Functions
- Perceptron
- Logistic Regression
- Softmax Regression



Likelihood vs. Discriminant-based Classification



Likelihood-based Classification

- Assume a model for the class-conditional **prob densities** $p(x|C_i)$
- Use Bayes' rule to compute the posterior probabilities $p(C_i|x)$
- Perform classification based on $g_i(x) = \log P(C_i|x)$

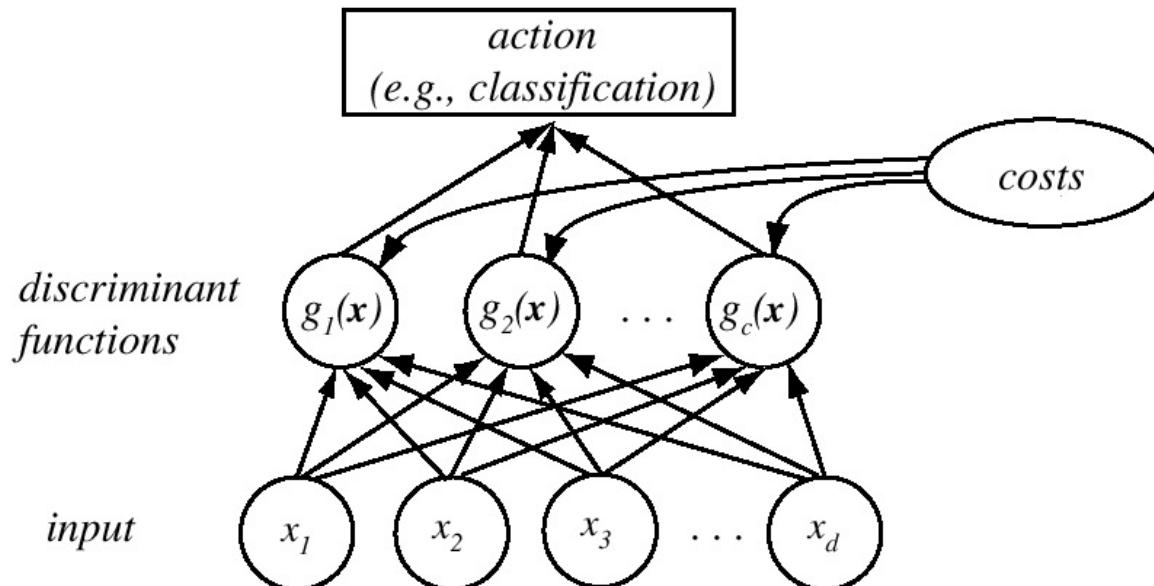
Discriminant-based classification

- Assume a model (discriminant functions $g_i(x|\theta_i)$) directly for the decision boundary;
- no density estimation.

Estimating the boundaries is enough; no need to accurately estimate the densities inside the boundaries.

Classification as Discriminants

- consider a classifier with c classes
- define c discriminant functions $g_i(x)$
- decision rule: assign x to C_j if $g_j(x) > g_i(x), \forall i \neq j$





Discriminant Functions

Discriminant functions:

$$g_i(x \mid \theta_i)$$

- functions which are explicitly parameterized with a set of parameters θ_i .
- model the **decision boundaries** between classes **directly** and **simultaneously**.

Estimating the class boundaries (discriminants) is usually an easier problem to solve than estimating the class densities.

Example – Bayes Decision

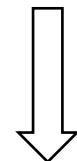
$$g_i(x) = P(C_i | x) = \frac{p(x|C_i)P(C_i)}{\sum_{j=1}^c p(x|C_j)P(C_j)}, \text{ or}$$

$$g_i(x) = p(x | C_i) P(C_i), \text{ or}$$

$$g_i(x) = \ln p(x | C_i) + \ln P(C_i)$$

Decision Rule (2-category case):

Decide C_1 if $g_1(x) > g_2(x)$ else C_2 .

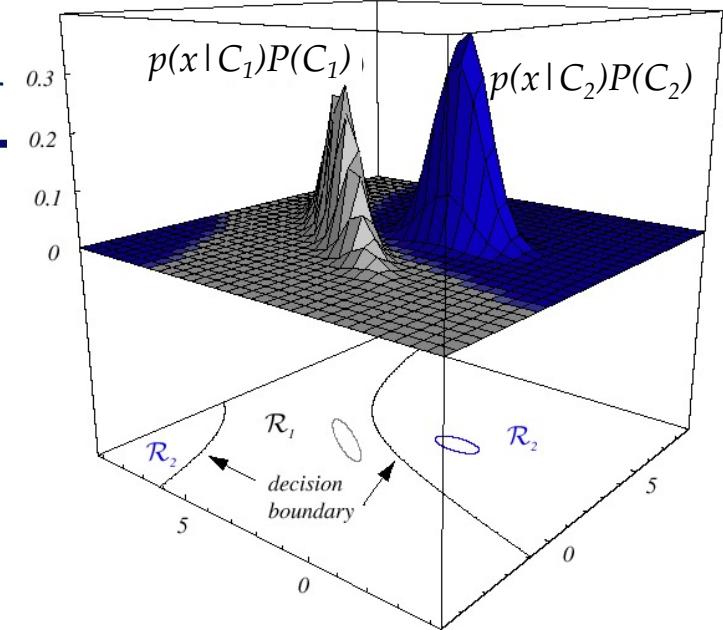


We can define a **single** discriminant function:

$$g(x) = g_1(x) - g_2(x)$$

Decide C_1 if $g(x) > 0$ else C_2 .

decision boundary: $g(x) = 0$



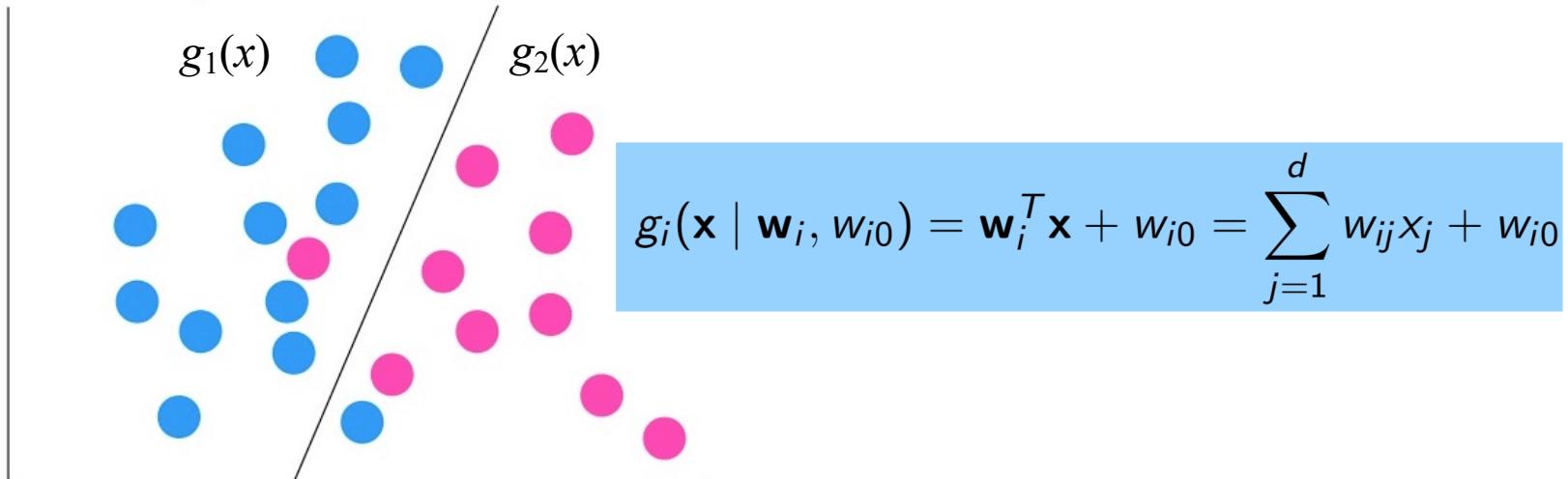
decision boundary: $g_1(x) = g_2(x)$



Linear Discriminant Functions

linear discriminants

"draw a line through it"

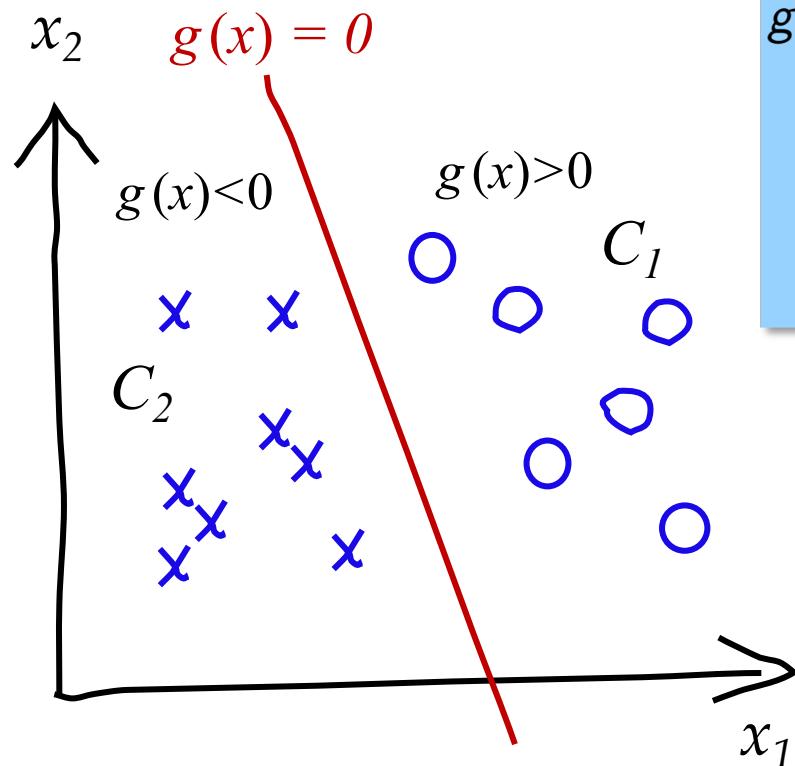


Advantages:

- **simplicity**: $O(d)$ time and space complexity
- **understandability**: final output is a weighted sum of attributes;
- **accuracy**: model is quite accurate if some assumptions are satisfied.

Two Classes

- A **linear** discriminant function models a **linear decision boundary** of two classes.



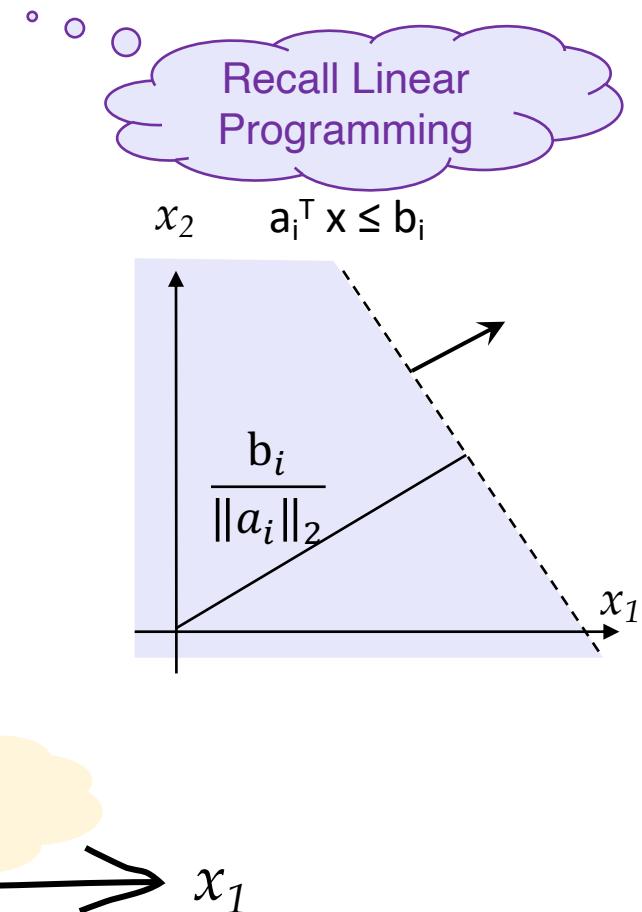
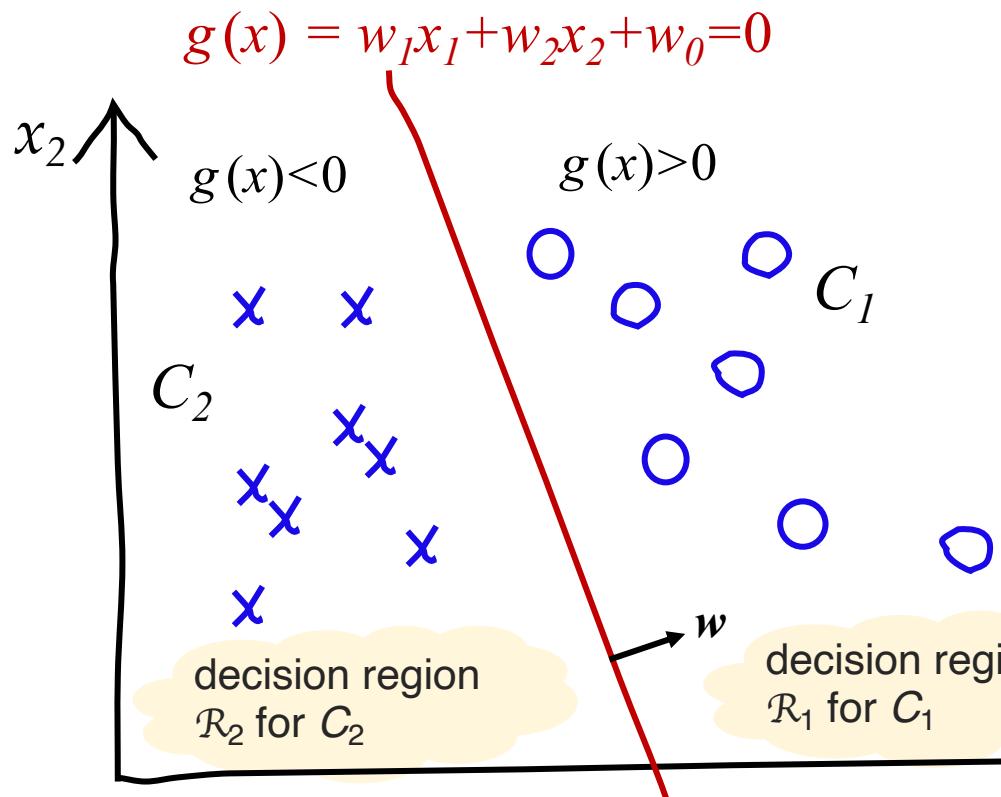
$$\begin{aligned}
 g(\mathbf{x}) &= g_1(\mathbf{x}) - g_2(\mathbf{x}) \\
 &= (\mathbf{w}_1^T \mathbf{x} + w_{10}) - (\mathbf{w}_2^T \mathbf{x} + w_{20}) \\
 &= (\mathbf{w}_1 - \mathbf{w}_2)^T \mathbf{x} + (w_{10} - w_{20}) \\
 &= \mathbf{w}^T \mathbf{x} + w_0
 \end{aligned}$$

- Decision rule:

Choose $\begin{cases} C_1 & \text{if } g(\mathbf{x}) > 0 \\ C_2 & \text{otherwise} \end{cases}$

Geometry: Hyperplane

The linear function $g(x)$ defines a **hyperplane** that divides the input space into 2 half-spaces.



Geometric Interpretation

- Let us express any point x as $\mathbf{x} = \mathbf{x}_p + r \frac{\mathbf{w}}{\|\mathbf{w}\|}$

where

\mathbf{x}_p : normal projection of \mathbf{x} onto the hyperplane

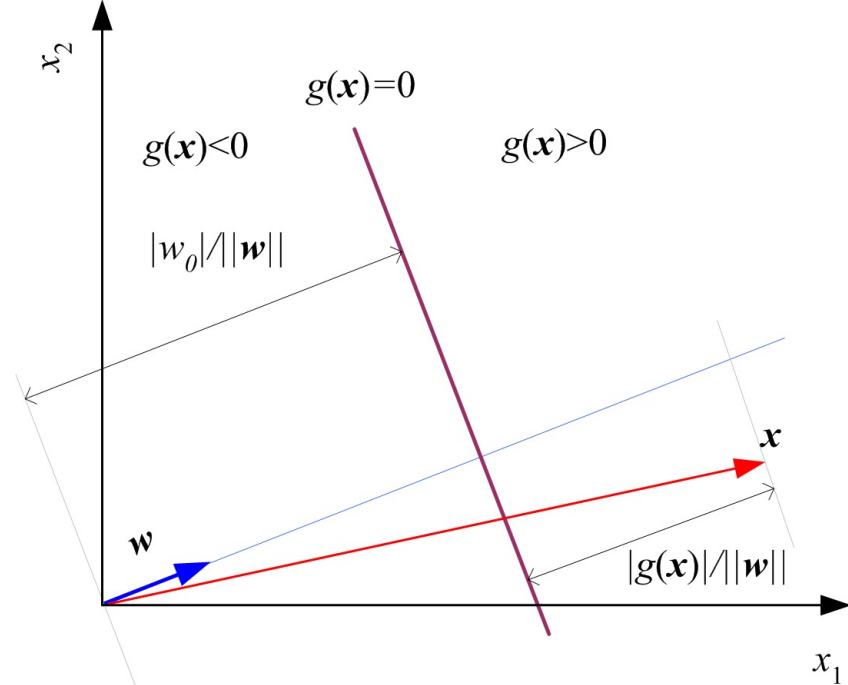
r : distance from \mathbf{x} to the hyperplane

$$\begin{aligned} g(\mathbf{x}) &= \mathbf{w}^T \mathbf{x} + w_0 \\ &= \mathbf{w}^T \mathbf{x}_p + r \frac{\mathbf{w}^T \mathbf{w}}{\|\mathbf{w}\|} + w_0 \\ &= g(\mathbf{x}_p) + r \|\mathbf{w}\| = r \|\mathbf{w}\| \end{aligned}$$



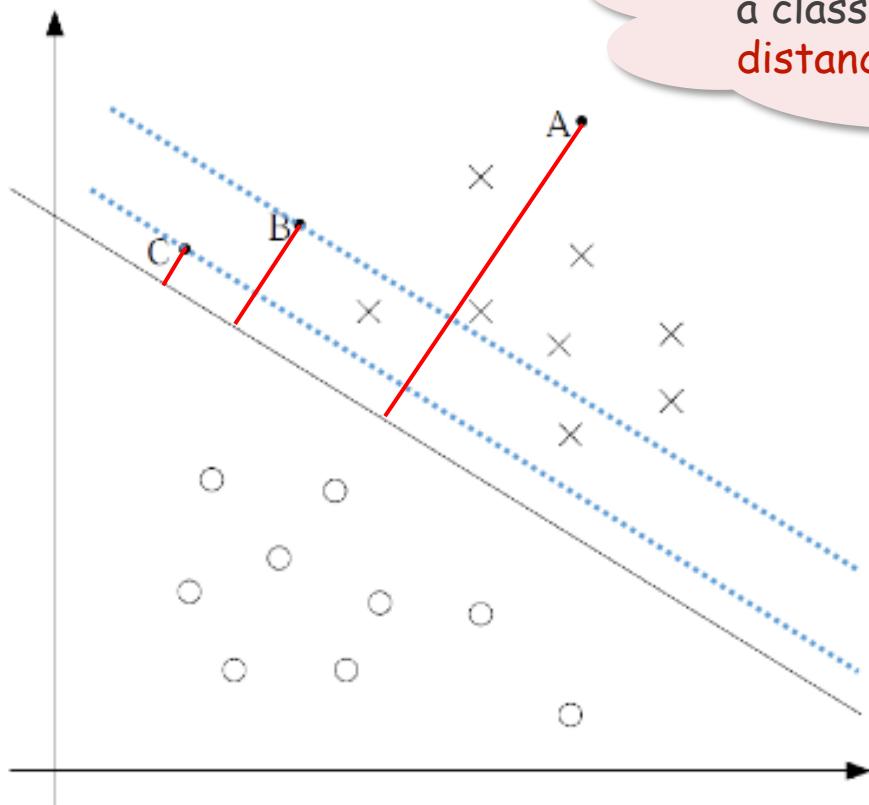
$$r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|}$$

(sign of r = sign of $g(\mathbf{x})$)



Geometric Interpretation

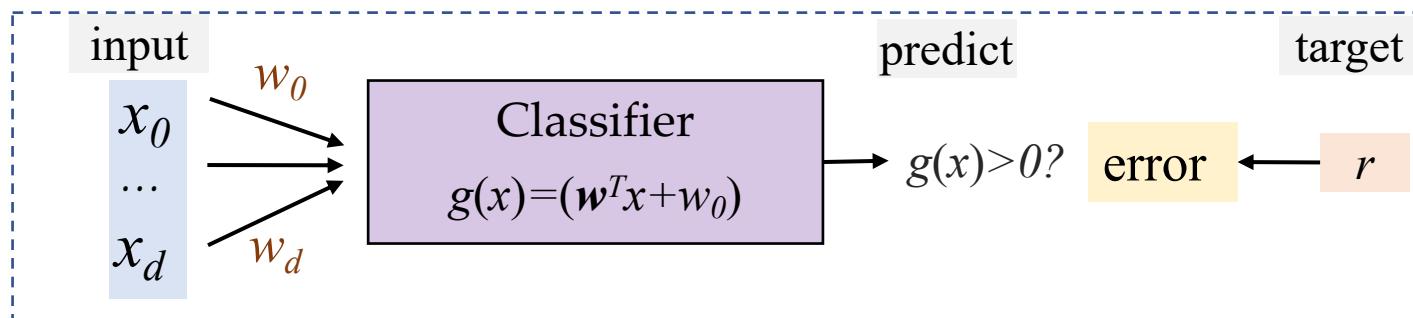
The discriminant function defines a **confidence score** of an data point to a class. The higher score, the larger **distance** to the decision boundary.





Perceptron

A simple, naïve linear classifier. (to introduce in future lectures)



- **Train:**
 - estimate the parameters \mathbf{w} and w_0 from data
- **Test:**
 - calculate $g(x) = (\mathbf{w}^T x + w_0)$ and choose C_1 if $g(x) > 0$ or choose C_2 if $g(x) < 0$.

perceptron classifies data based on which **side** of the plane the new point lies on.

Training



Let \mathbf{x} denote the data input, and $r \in \{1, -1\}$ (means the sign of $g(\mathbf{x})$) denotes the **label** of target classes.

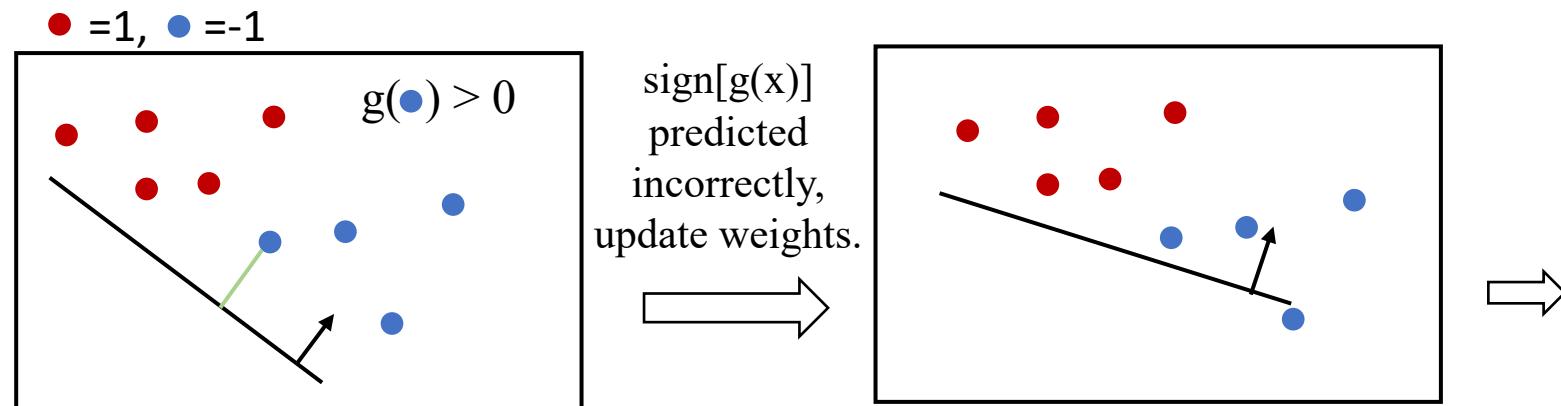
Input: dataset $D = \{(x^{(1)}, r^{(1)}), (x^{(2)}, r^{(2)}), \dots (x^{(N)}, r^{(N)})\}$

for each training instance $(x^{(\ell)}, r^{(\ell)}) \in D$,

if $r^{(\ell)} g_i(x^{(\ell)}) \leq 0$: // a misclassification occurs

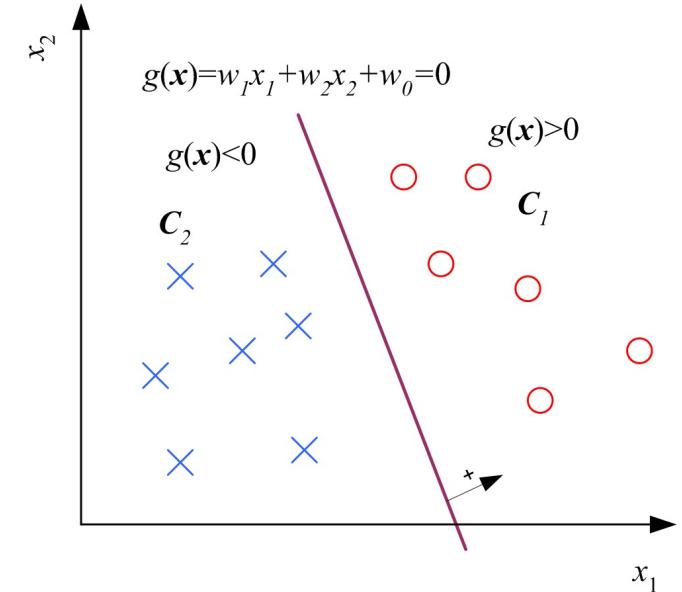
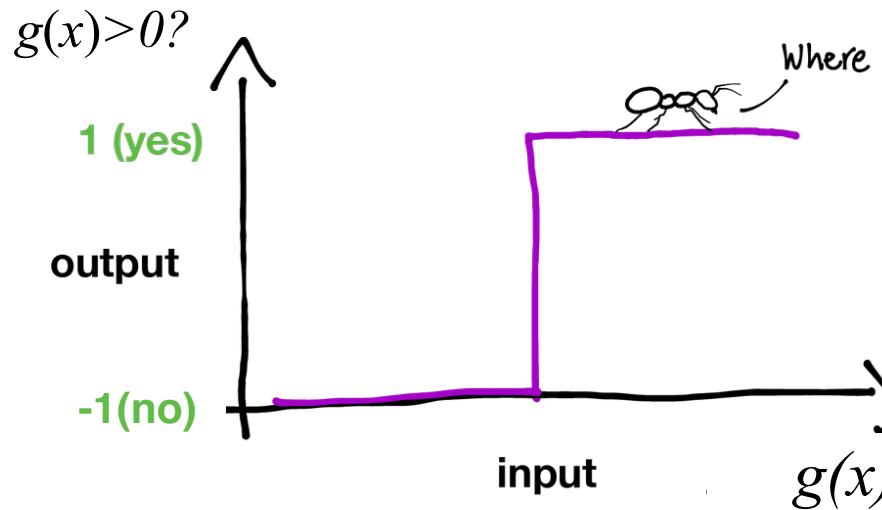
$w \leftarrow w + \eta r^{(\ell)} x^{(\ell)}$ // move the hyperplane (defined by w) towards the misclassified data point

repeat until the entire training set is classified correctly



Limitations of Perceptron

Hard Decision and Optimization



I know it belongs to C_1 but what is the extent?
 (0-1 decision is too hard to optimize)



Linear Classification with Uncertainty

What is the posterior probability of choosing C_1/C_2 ?

Let

$$P(C_1 | \mathbf{x}) = y \quad P(C_2 | \mathbf{x}) = 1 - y$$

Classification rule:

Choose $\begin{cases} C_1 & \text{if } y > 0.5 \\ C_2 & \text{otherwise} \end{cases}$

Equivalent Rule:

$$\frac{y}{1-y} > 1 \quad \text{or}$$

odds of y

$$\log \frac{y}{1-y} > 0$$

Log odds (logit function) of y



Logit Function*

Assume that x follows the Gaussian distribution in each class.

- Logit function:

$$\begin{aligned} \text{logit}(P(C_1 | x)) &= \log \frac{y}{1-y} \\ &= \log \frac{P(C_1 | x)}{1 - P(C_1 | x)} = \log \frac{P(C_1 | x)}{P(C_2 | x)} \\ &= \log \frac{p(x | C_1)}{p(x | C_2)} + \log \frac{P(C_1)}{P(C_2)} \\ &= \log \frac{(2\pi)^{-d/2} |\Sigma|^{-1/2} \exp[-(1/2)(x - \mu_1)^T \Sigma^{-1} (x - \mu_1)]}{(2\pi)^{-d/2} |\Sigma|^{-1/2} \exp[-(1/2)(x - \mu_2)^T \Sigma^{-1} (x - \mu_2)]} + \log \frac{P(C_1)}{P(C_2)} \\ &= \mathbf{w}^T \mathbf{x} + w_0 \end{aligned}$$

where

$$\mathbf{w} = \Sigma^{-1}(\mu_1 - \mu_2)$$

$$w_0 = -\frac{1}{2}(\mu_1 + \mu_2) \Sigma^{-1}(\mu_1 - \mu_2) + \log \frac{P(C_1)}{P(C_2)}$$

Logit (log odds) of the posterior probability is a linear function!

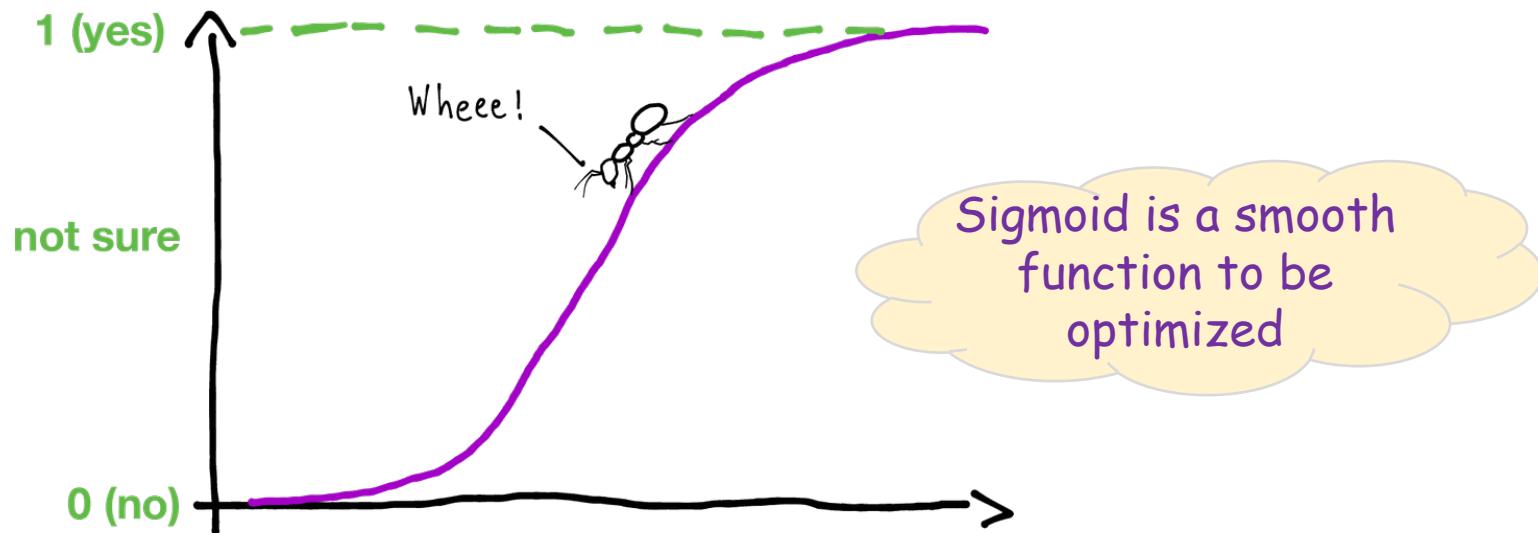
The Sigmoid Function

$$\log \frac{y}{1-y} = \mathbf{w}^T \mathbf{x} + w_0$$



$$y = \text{sigmoid}(\mathbf{w}^T \mathbf{x} + w_0) = \frac{1}{1 + \exp[-(\mathbf{w}^T \mathbf{x} + w_0)]}$$

- The sigmoid function (or logistic function) is the **inverse function** of logit, which directly computes the **posterior class probability** $P(C_1|x)$.

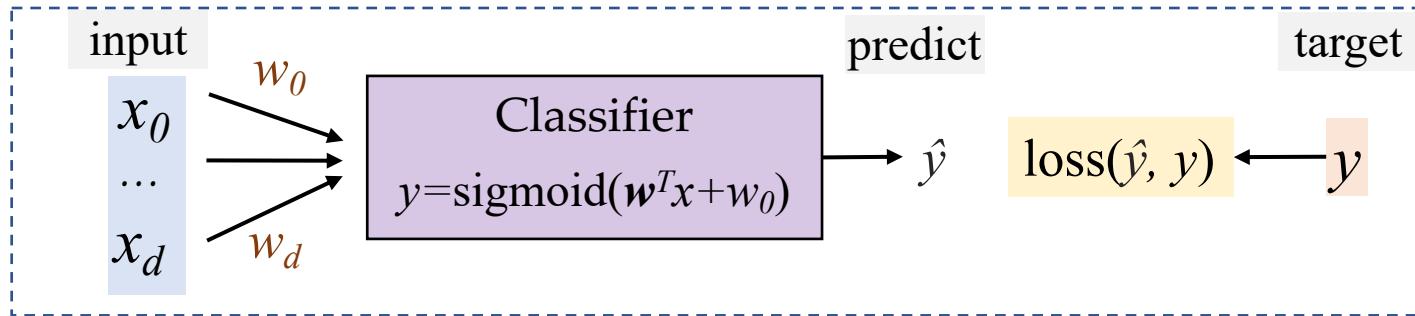




Logistic Regression

A classifier that estimates the **decision boundary** as a **logistic function**:

$$y = \text{sigmoid}(\mathbf{w}^T \mathbf{x} + w_0) = \frac{1}{1 + \exp[-(\mathbf{w}^T \mathbf{x} + w_0)]}$$



- **Train:**
 - estimate the parameters \mathbf{w} and w_0 from data
- **Test:**
 - calculate $y = \text{sigmoid}(\mathbf{w}^T \mathbf{x} + w_0)$ and choose C_1 if $y > 0.5$ (y can be interpreted as a posterior probability).

Loss Function

$y \equiv p(C_1|x) = \text{sigmoid}(w^\top x + w_0)$ is the estimated posterior probability of $x \in C_1$.



- For a given input x , the model outputs a probability y of $x \in C_1$. Let $r \in \{0, 1\}$ be the label of the real class ($r = 1: x \in C_1$, $r = 0: x \in C_2$):
 - if $r = 1$: we aim to maximize $\log p(C_1|x) = \log y$, cost is $-\log y$
 - if $r = 0$: we aim to maximize $\log p(C_2|x) = \log(1-y)$, cost is $-\log(1-y)$
- Can write this succinctly as a **cross-entropy loss**:

$$\ell(w, w_0 | x, r) = -r \underbrace{\log y}_{\text{nonzero only if } r=1} - (1-r) \underbrace{\log(1-y)}_{\text{nonzero only if } r=0}$$

$$\begin{aligned}\text{CE}(p, q) &= \mathbb{E}_{x \sim p(x)} \left\{ \log \frac{1}{q(x)} \right\} \\ &= - \sum_{x \in X} p(x) \log_2 q(x)\end{aligned}$$

- Equivalent to **maximize the likelihood**:

$$r | x \sim \text{Bernoulli}(y)$$
$$p(r|x) = y^r (1-y)^{1-r} = \begin{cases} y & \text{if } r=1 \\ 1-y & \text{if } r=0 \end{cases}$$

Training

Given: $D = \{(x^{(1)}, r^{(1)}), \dots, (x^{(N)}, r^{(N)})\}$

minimize the loss function using **gradient descend**:

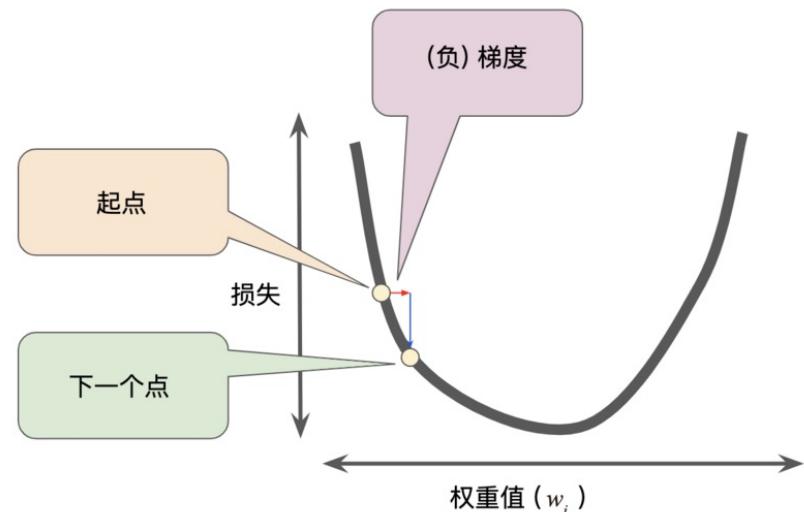
- Goal:

$$\min_w L(w)$$

- Iteration:

$$w_{t+1} = w_t - \eta_t \frac{\partial L}{\partial w}$$

What is $\frac{\partial L}{\partial w}$?





Optimization – Gradient Descend

$$\ell(w, w_0 | x, r) = -r \log y - (1-r) \log (1-y)$$

$$L(w, w_0 | D) = -\sum_{l=1}^N r^{(l)} \log y^{(l)} + (1-r^{(l)}) \log (1-y^{(l)})$$

What is $\frac{\partial L}{\partial w}$?

Hint: if $y = \text{sigmoid}(a) = 1/[1+\exp(-a)]$,
its derivative is $\frac{dy}{da} = y(1-y)$

For each w_j ($j = 1, \dots, d$):

$$\frac{\partial L}{\partial w_j} = -\sum_l \underbrace{\left(\frac{\partial L}{\partial y^{(l)}} \frac{\partial y^{(l)}}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial w_j} \right)}_{\text{Chain rule}} = -\sum_l \left(\frac{r^{(l)}}{y^{(l)}} - \frac{1-r^{(l)}}{1-y^{(l)}} \right) \frac{\partial y^{(l)}}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial w_j}$$



Gradient-Descend Learning (2)

For each w_j ($j = 1, \dots, d$):

$$\frac{\partial L}{\partial w_j} = - \sum_l \left(\frac{r^{(l)}}{y^{(l)}} - \frac{1 - r^{(l)}}{1 - y^{(l)}} \right) \frac{\partial y^{(l)}}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial w_j}$$

Since $a^{(l)} = \mathbf{w}^T \mathbf{x}^{(l)} + w_0$, we have $\frac{\partial a^{(l)}}{\partial w_j} = x_j^{(l)}$

So,

$$\frac{\partial L}{\partial w_j} = - \sum_l \left(\frac{r^{(l)}}{y^{(l)}} - \frac{1 - r^{(l)}}{1 - y^{(l)}} \right) y^{(l)} (1 - y^{(l)}) x_j^{(l)} = - \sum_l (r^{(l)} - y^{(l)}) x_j^{(l)}$$

$$\frac{\partial L}{\partial w_0} = - \sum_l (r^{(l)} - y^{(l)})$$

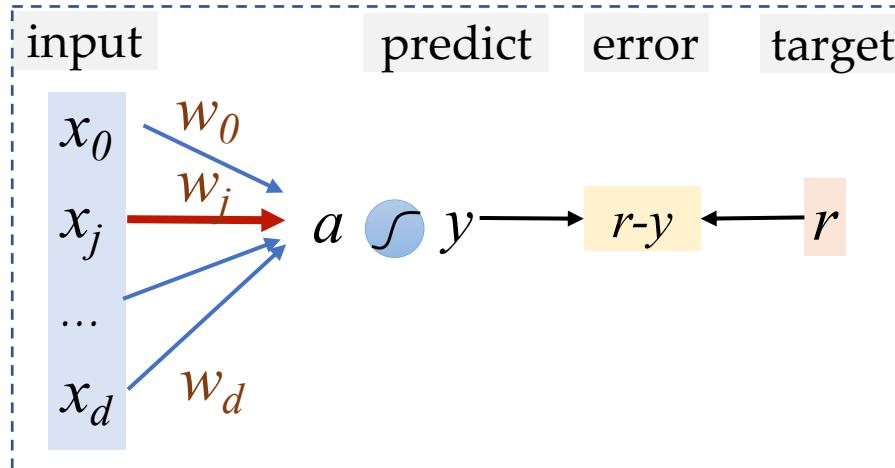
Gradient-Descend Learning (3)

An interesting point

$$\frac{\partial L}{\partial w_j} = - \sum_l \left(\frac{r^{(l)}}{y^{(l)}} - \frac{1 - r^{(l)}}{1 - y^{(l)}} \right) y^{(l)} (1 - y^{(l)}) x_j^{(l)} = - \sum_l (r^{(l)} - y^{(l)}) x_j^{(l)}$$

error
 input

⋮



The update to each weight is the product of **error** and **input** (signal)

The Algorithm

Gradient Descend for LR

Input: $D = \{(x^{(l)}, r^{(l)})\}$ ($l=1:N$)

for $j = 0, \dots, d$

$w_j \leftarrow \text{rand} (-0.01, 0.01)$

repeat

for $j = 0, \dots, d$

$\Delta w_j \leftarrow 0$

for $l = 1, \dots, N$

$a \leftarrow 0$

for $j = 0, \dots, d$

$a \leftarrow a + w_j x_j^{(l)}$

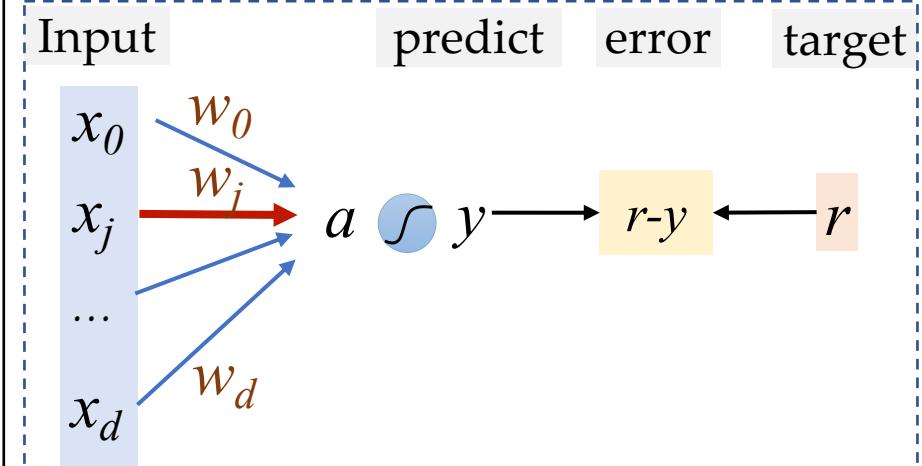
$y \leftarrow \text{sigmoid}(a)$

$\Delta w_j \leftarrow \Delta w_j + (r^{(l)} - y) x_j^{(l)}$

for $j = 0, \dots, d$

$w_j \leftarrow w_j + \eta \Delta w_j$

until convergence





Multiple Classes

How about classification with more than 2 categories?

Binary
Classification



Multiclass
Classification



- Spam
- Not spam

Targets from a
binary set {0, 1}.

- Dog
- Cat
- Horse
- Fish

Targets form a discrete set
of K classes {1, . . . , K}.

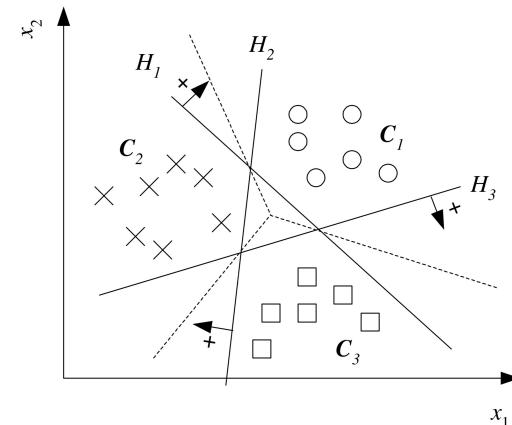
Linear Classifier for Multiclass

- K discriminant functions:

$$g_i(\mathbf{x} | \mathbf{w}_i, w_{i0}) = \mathbf{w}_i^T \mathbf{x} + w_{i0}$$

- Linearly separable classes:

$$g_i(\mathbf{x} | \mathbf{w}_i, \mathbf{w}_{i0}) = \begin{cases} > 0 & \text{if } \mathbf{x} \in C_i \\ \leq 0 & \text{otherwise} \end{cases}$$



For each class C_i , there exists a hyperplane H_i such that all $\mathbf{x} \in C_i$ lie on the positive side and all other $\mathbf{x} \in C_j, j \neq i$ lie on the negative side.

- decision rule for any test case x :

Choose C_i if $g_i(\mathbf{x}) = \max_{j=1}^K g_j(\mathbf{x})$

- geometrically a linear classifier partitions the feature space into K convex decision regions R_i .



Multiclass Classification with Uncertainty

What is the posterior probability of choosing C_i ($i=1,\dots,K$)?

- One of the K classes, e.g., C_K , is taken as the **reference class**.
- Assume that

$$\log \frac{p(\mathbf{x} | C_i)}{p(\mathbf{x} | C_K)} = \mathbf{w}_i^T \mathbf{x} + w_{i0}^0, \quad i = 1, \dots, K - 1$$

So we have

$$\begin{aligned} \frac{P(C_i | \mathbf{x})}{P(C_K | \mathbf{x})} &= \frac{p(\mathbf{x} | C_i)P(C_i)}{p(\mathbf{x} | C_K)P(C_K)} \\ &= \exp(\mathbf{w}_i^T \mathbf{x} + w_{i0}^0) \cdot \exp\left(\log \frac{P(C_i)}{P(C_K)}\right) \\ &= \exp(\mathbf{w}_i^T \mathbf{x} + w_{i0}) \end{aligned} \tag{1}$$

where $w_{i0} = w_{i0}^0 + \log[p(C_i)/P(C_K)]$.



Generalization of Logistic Function*

- Summing (1) over $i = 1, \dots, K - 1$:

$$\sum_{i=1}^{K-1} \frac{P(C_i | \mathbf{x})}{P(C_K | \mathbf{x})} = \frac{1 - P(C_K | \mathbf{x})}{P(C_K | \mathbf{x})} = \sum_{i=1}^{K-1} \exp(\mathbf{w}_i^T \mathbf{x} + w_{i0})$$

we get

$$P(C_K | \mathbf{x}) = \frac{1}{1 + \sum_{i=1}^{K-1} \exp(\mathbf{w}_i^T \mathbf{x} + w_{i0})} \quad (2)$$

- From (1) and (2), we get

$$P(C_i | \mathbf{x}) = \frac{\exp(\mathbf{w}_i^T \mathbf{x} + w_{i0})}{1 + \sum_{j=1}^{K-1} \exp(\mathbf{w}_j^T \mathbf{x} + w_{j0})}, \quad i = 1, \dots, K - 1$$



Softmax Function

If we want to treat all classes uniformly without having to choose a reference class, we can use the softmax function instead for the posterior class probabilities:

$$y_i = \hat{P}(C_i | \mathbf{x}) = \frac{\exp(\mathbf{w}_i^T \mathbf{x} + w_{i0})}{\sum_{j=1}^K \exp(\mathbf{w}_j^T \mathbf{x} + w_{j0})}, \quad i = 1, \dots, K$$



Softmax Function

In general, the softmax function is defined as

$$y_i = \text{softmax}(a_1, \dots, a_K)_i = \frac{e^{a_i}}{\sum_{j=1}^K e^{a_j}}$$

where the inputs $a_i = W_i x + w_{i0}$ are called the logits. W_i and w_{i0} are the trainable parameters.

- If one of the a_k 's is much larger than the others, softmax (a_1, \dots, a_K) is approximately the argmax. (so really it's more like "soft-argmax").
 - "max" because it amplifies probability of the largest a_i .
 - "soft" because still assigns some probability to smaller a_j .
- The softmax function behaves like taking a maximum, but it has the advantage of being differentiable.

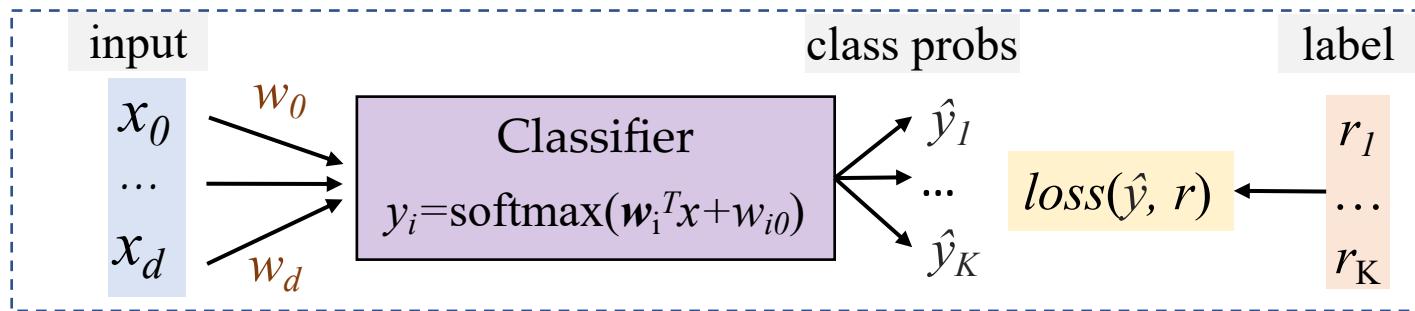
Exercise: how does the case of K=2 relate to the logistic function?



Softmax Regression

A classifier that estimates the decision boundaries as Softmax functions:

$$y_i = \text{softmax}(\mathbf{w}_i^T x + w_{i0}) = \frac{\exp[\mathbf{w}_i^T x + w_{i0}]}{\sum_{j=1}^K \exp[\mathbf{w}_j^T x + w_{j0}]} \quad (i=1,\dots,K)$$



- **Train:**
 - estimate the parameters \mathbf{w}_i and w_{i0} ($i=1:K$) from data
- **Test:**
 - calculate $y_i = \text{softmax}(\mathbf{w}_i^T x + w_{i0})$ and choose C_i if $y_i = \max\{y_{1:K}\}$ (y can be interpreted as a posterior probability).

Loss Function

$$\mathbf{r} = \underbrace{(0, \dots, 0, 1, 0, \dots, 0)}_{\text{entry } k \text{ is 1}}$$



- For a given **input** x , the model **outputs** a vector of class probabilities $\mathbf{y} = (y_1, \dots, y_K)$, and the **label** of target class is a one-hot vector $\mathbf{r} = (r_1, \dots, r_K)$ ($r_i=1:x \in C_i, r_i=0:x \notin C_i$)
 - if $r_1 = 1$: we aim to maximize $\log p(C_1|x) = \log y_1$, cost is $-\log y_1$
 - if $r_2 = 1$: we aim to maximize $\log p(C_2|x) = \log y_2$, cost is $-\log y_2$
 -
- We can write this succinctly as a **cross-entropy** loss function:

$$L_{\text{CE}}(\mathbf{y}, \mathbf{r}) = - \sum_{i=1}^K r_i \log y_i = -\mathbf{r}^T(\log \mathbf{y})$$

where the *log* is applied elementwise.

Training

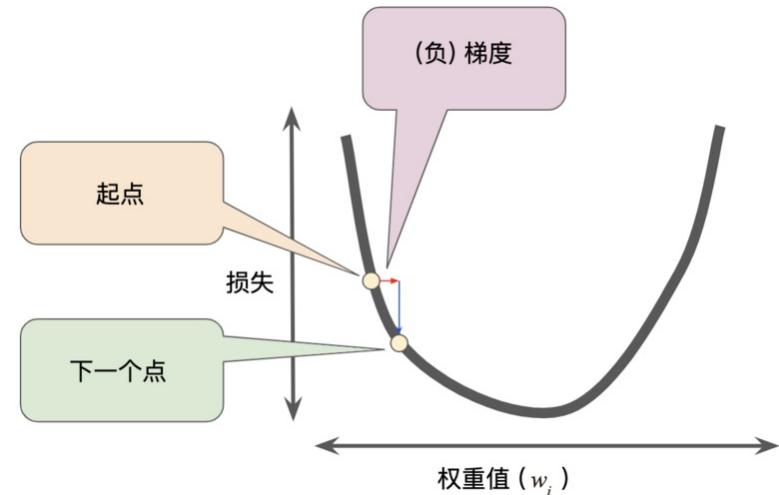
- Given: $D = \{(x^{(1)}, r^{(1)}), \dots, (x^{(N)}, r^{(N)})\}$
- minimize the loss function using **gradient descend**:

- Goal:

$$\min_w L(w)$$

- Iteration:

$$w_{t+1} = w_t - \eta_t \frac{\partial L}{\partial w}$$



What is $\frac{\partial L}{\partial w}$?



Optimization – Gradient Descend*

$$L(\mathbf{w} | \mathbf{x}, \mathbf{r}) = -\sum_{i=1}^K r_i \log y_i$$

$$L(\mathbf{w} | D) = -\sum_{l=1}^N [\sum_{i=1}^K r_i^{(l)} \log y_i^{(l)}]$$

What is $\frac{\partial L}{\partial w}$?

Hint: if $y_i = \exp(a_i)/\sum_j \exp(a_j)$, its derivative is $\frac{\partial y}{\partial a} = y_i(\delta_{ij} - y_j)$ where $\delta_{ij} = \begin{cases} 0 & \text{if } i \neq j, \\ 1 & \text{if } i = j. \end{cases}$

For each w_j and w_{j0} ($j=1,\dots,K$), given $\sum_i r_i^{(l)} = 1$:

$$\begin{aligned}\frac{\partial L}{\partial w_j} &= \sum_l \sum_i \frac{\partial L}{\partial y_i^{(l)}} \frac{\partial y_i^{(l)}}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial w_j} = \sum_l \sum_i r_i^{(l)} (\delta_{ij} - y_j^{(l)}) x^{(l)} \\ &= \sum_l \left[\sum_i r_i^{(l)} \delta_{ij} - y_j^{(l)} \sum_i r_i^{(l)} \right] x^{(l)} = \sum_l (r_j^{(l)} - y_j^{(l)}) x^{(l)} \\ \frac{\partial L}{\partial w_{j0}} &= \sum_l (r_j^{(l)} - y_j^{(l)})\end{aligned}$$



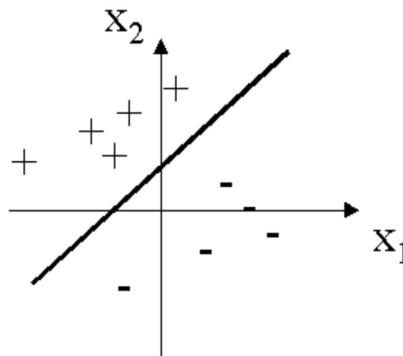
The Algorithm

Gradient Descend for Softmax Regression

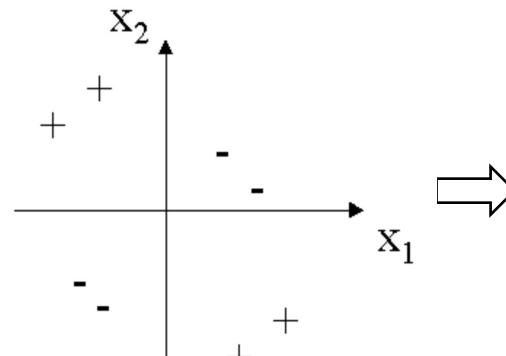
```
for i = 1, ... , K, for j = 0,...,d,  
    wij ← rand (-0.01, 0.01) // initialization  
repeat  
    for i = 1,...,K, for j = 0,..., d, Δwij ← 0  
        for l = 1,...,N  
            for i = 1,...,K  
                ai ← 0  
                for j = 0,..., d  
                    ai ← ai + wijxj(l)  
                for i = 1,..., K  
                    yi ← exp(ai)/Σjexp(aj)  
                for i = 1,...,K  
                    for j = 0,..., d  
                        Δwij ← Δwij+(ri(l)-yi)xj(l)  
                for i = 1,...,K  
                    for j = 0,..., d  
                        wij ← wij+ ηΔwij  
until convergence
```

Limitations of Linear Classifiers

Requires data to be linearly separable



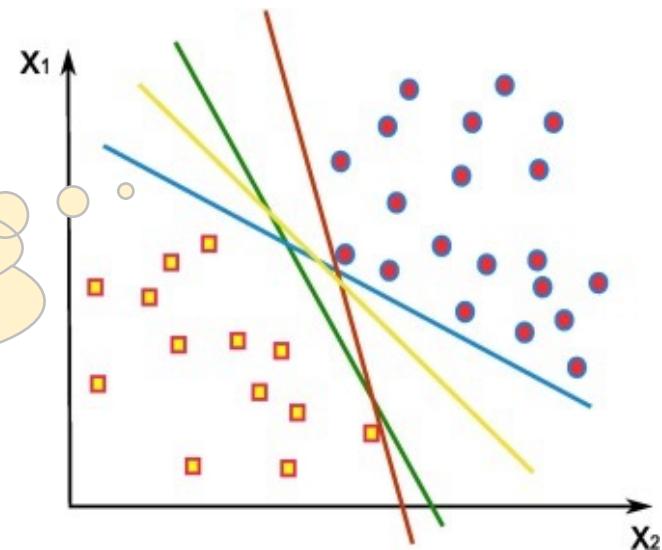
Linearly Separable



Not Linearly Separable

- ✓ Kernel Technique
- ✓ Non-Linear Models
(e.g., Neural Networks)

Even if data is linearly separable, we can have many numbers of separations.



What's Next?

WHAT'S
NEXT?



Support Vector Machine

Find a decision boundary that **maximizes the margin** between two classes.

