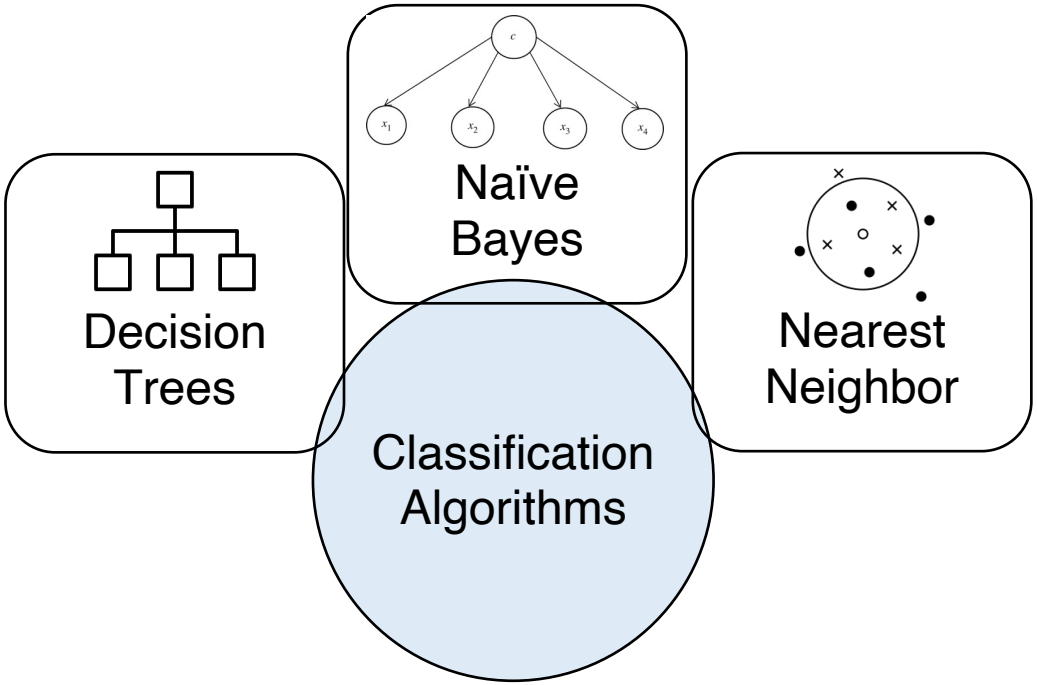
Machine Learning

Lecture 6: Logistic Regression

Fall 2022

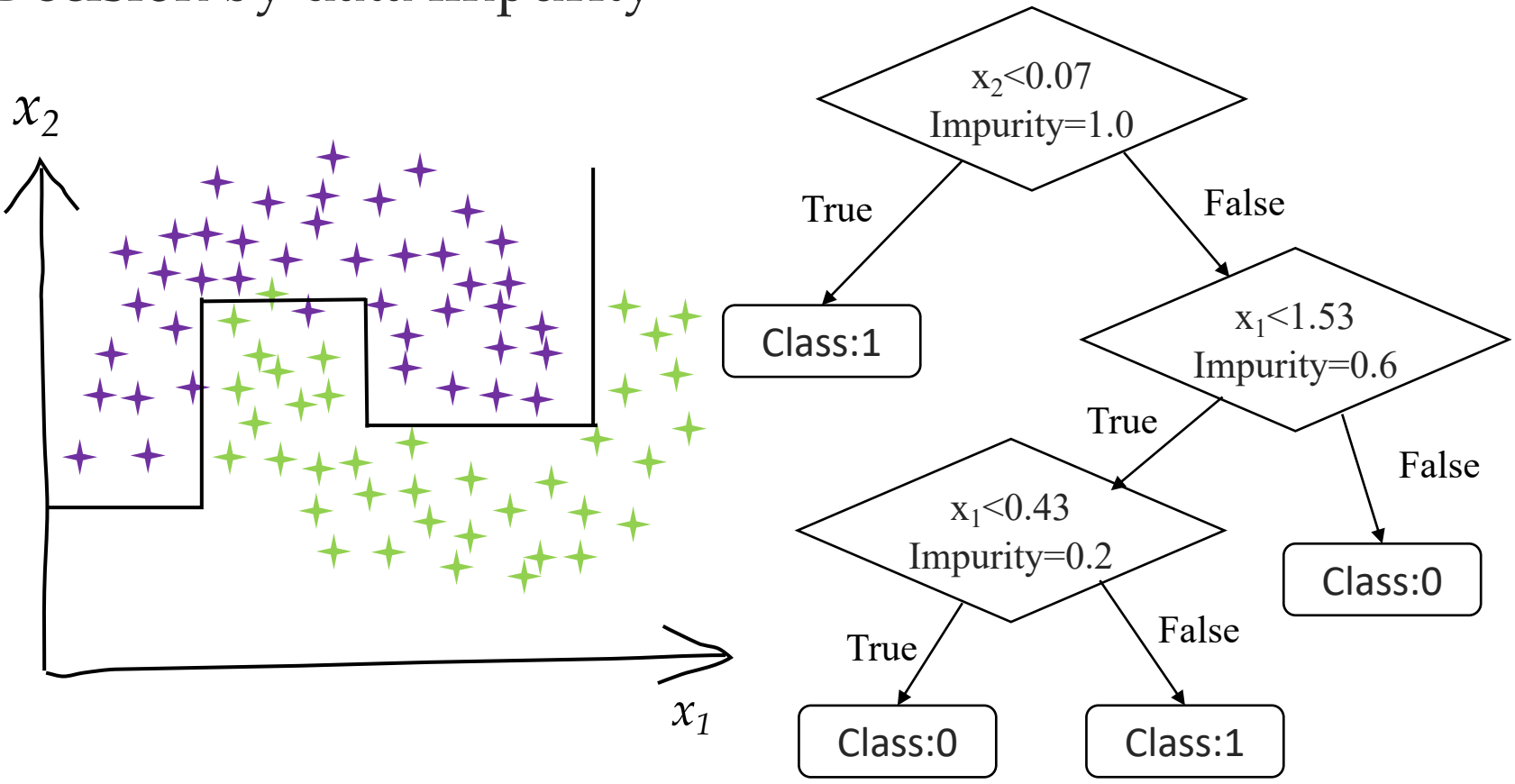
Instructor: Xiaodong Gu

The family of classification



Recall: Decision Trees

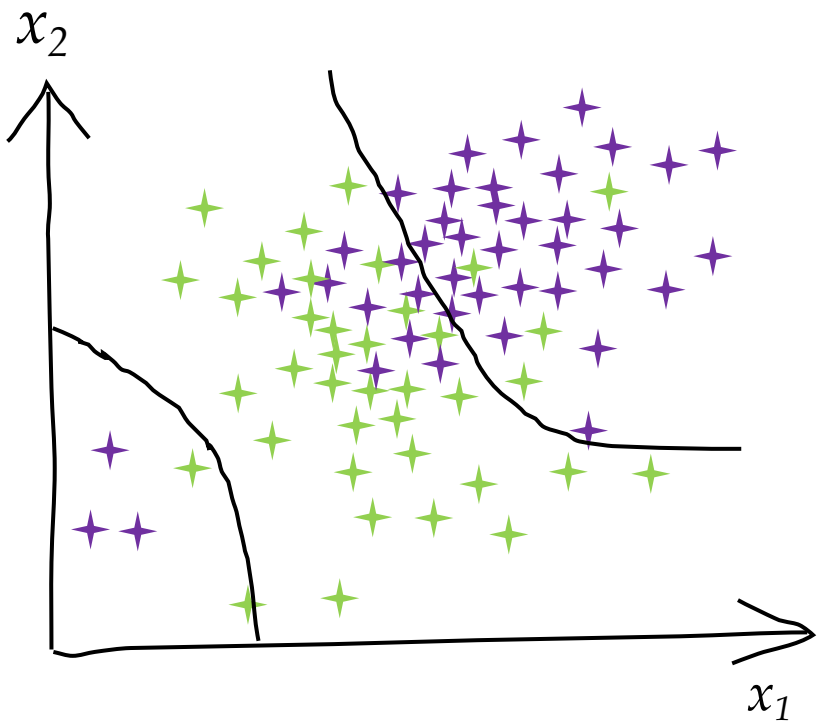
Decision by data impurity

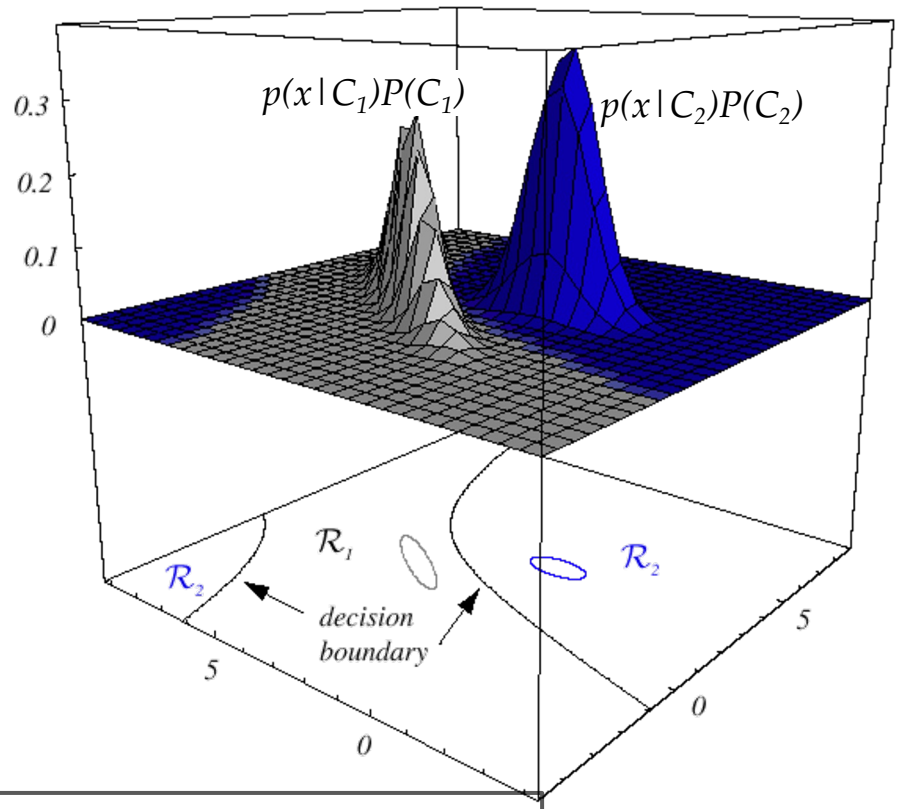


Recall: Bayes Classification

Decision by probability density

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

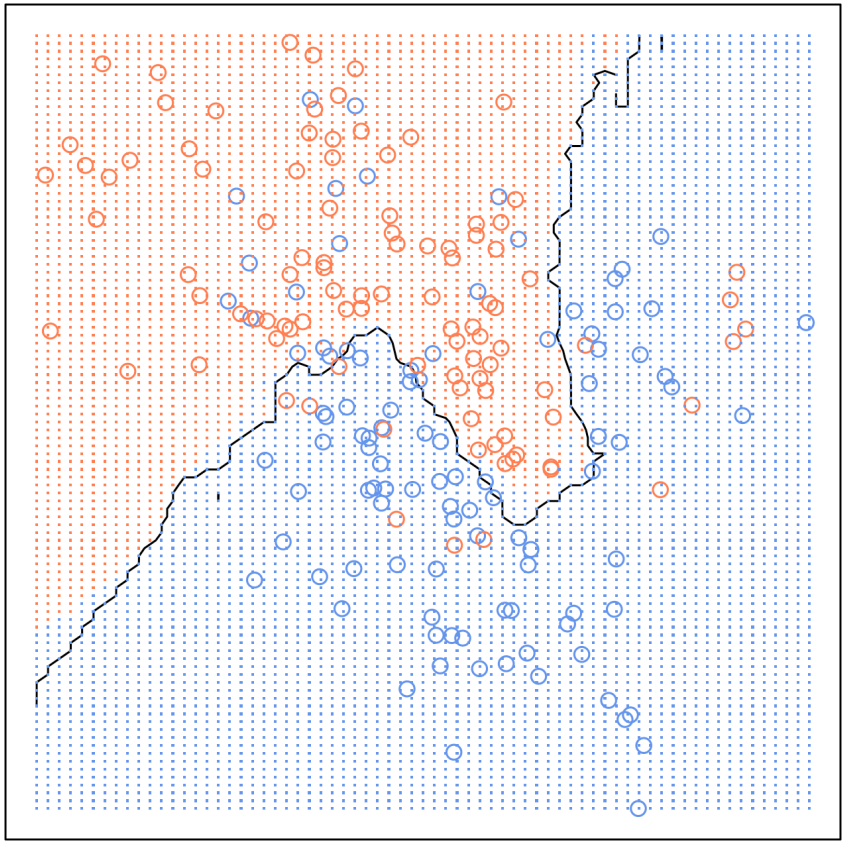




Decision rule: Classify x into C_i if $p(C_i|x) > p(C_j|x)$ else C_j.

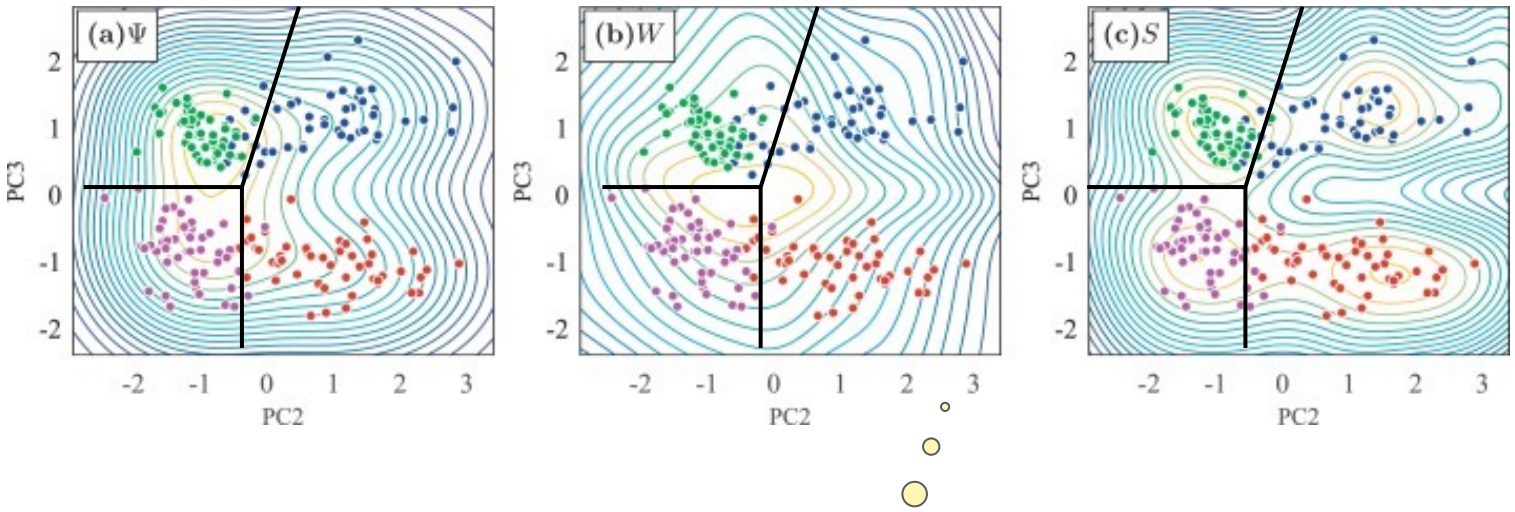
Recall: K-Nearest Neighbor

Decision by data distances



Decision Boundaries Are All You Need

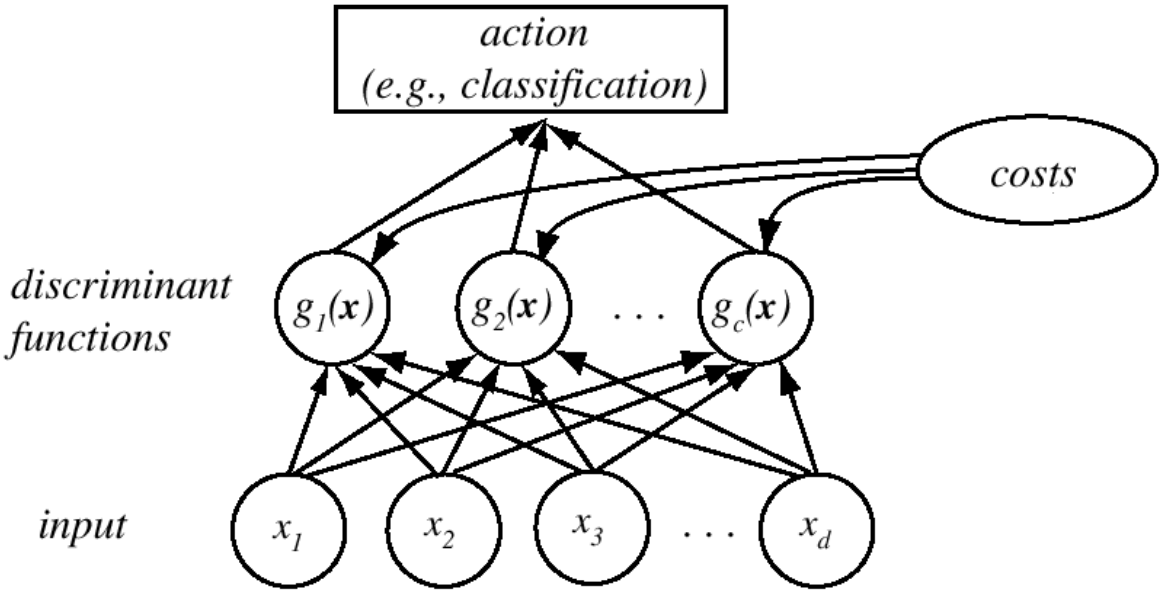
Do we really need to know the structure (impurity/density /distance/…) of data ?



Why not estimate the decision boundary directly?

Classification as Discriminants

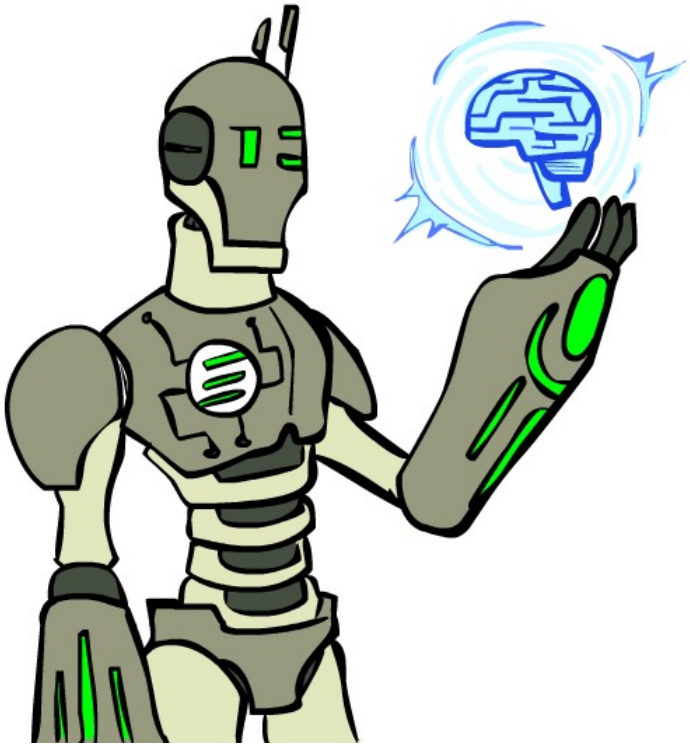
- consider a classifier with c classes
- define c discriminant functions $g_i(x)$
- **decision rule**: assign x to C_j if $g_j(x) > g_i(x)$, $\forall i \neq j$



Today

Linear Classifiers

- (Linear) Discriminant Functions
- Perceptron
- Logistic Regression
- Softmax Regression



Discriminant Functions

- Function that characterize the degree of data belonging to a class, parameterized with a set of parameters θ_i.

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

- Model the decision boundaries between classes **directly** and **simultaneously**.

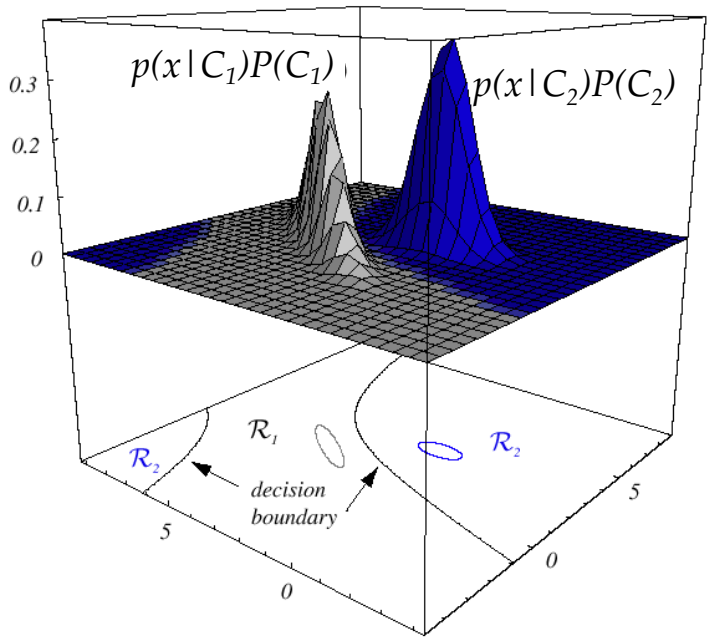
> Determinizing class boundaries (discriminants) is usually easier than estimating the class densities.

Example – Bayes Decision

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

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

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



Decision Rule (2-category case):

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

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

For simplicity, we can define a single discriminant function:

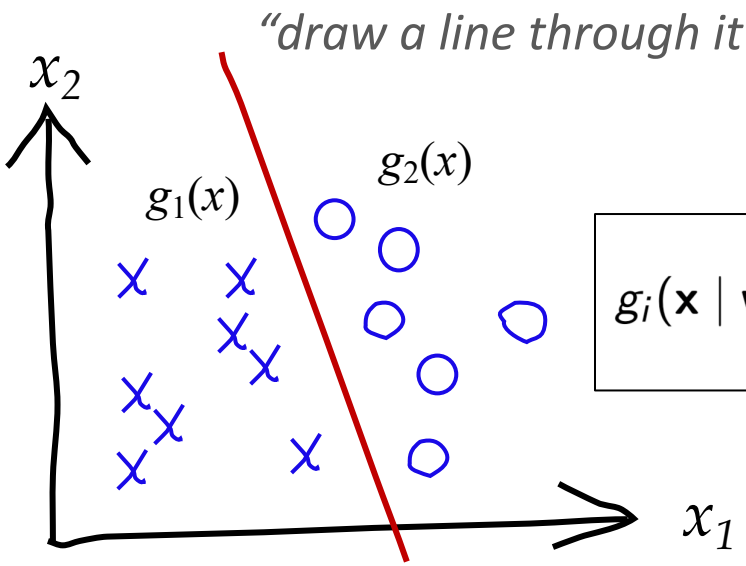
$$g(x) = g_1(x) - g_2(x)$$

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

decision boundary: $g(x) = 0$

Linear Discriminant Functions

- A linear function to model the class affiliation of data.



$$g_i(\mathbf{x} \mid \mathbf{w}_i, w_{i0}) = \mathbf{w}_i^T \mathbf{x} + w_{i0} = \sum_{j=1}^{d} w_{ij} x_j + w_{i0}$$

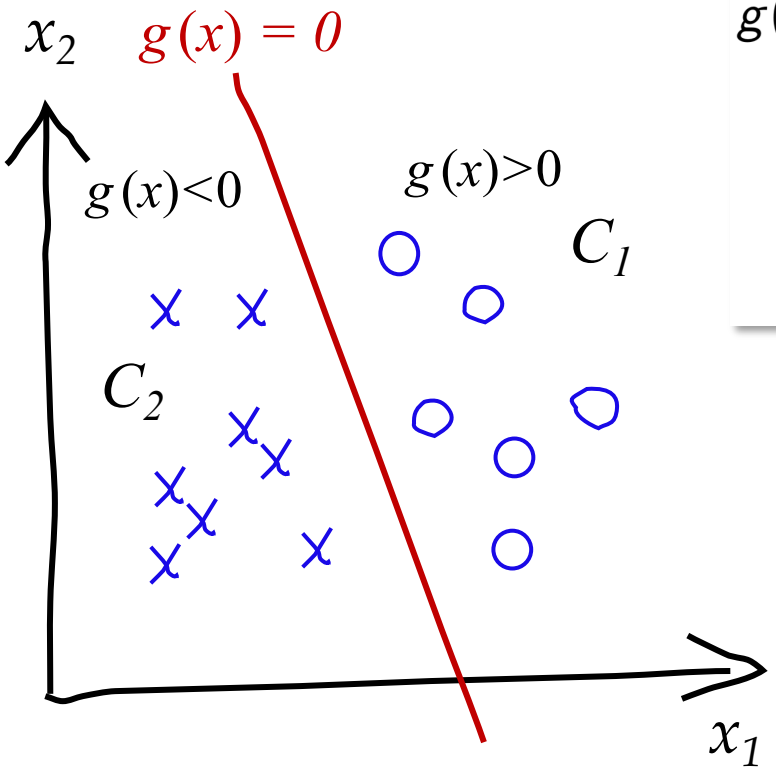
Decide C_1 if $g_1(x) > g_2(x)$ else C_2.

Advantages:

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

Two Classes

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



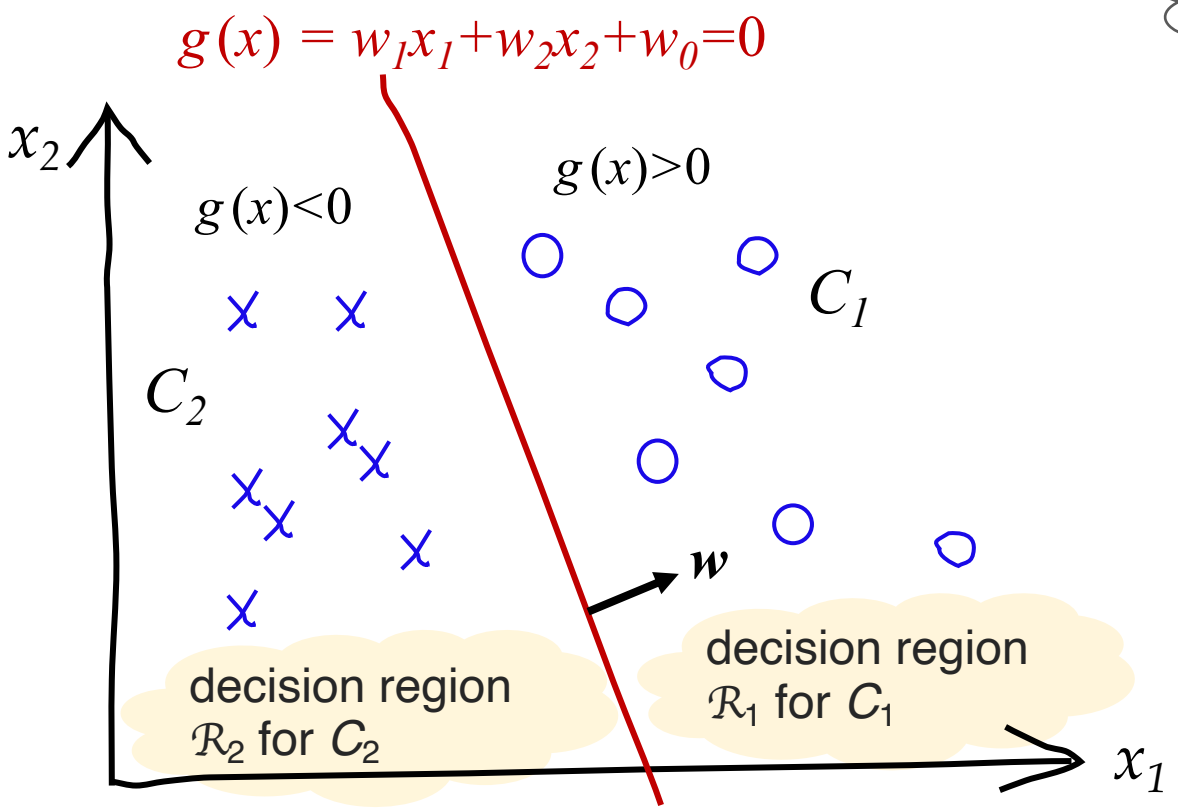
$$
\begin{aligned}
g(\mathbf{x}) &= g_1(\mathbf{x}) - g_2(\mathbf{x}) \\
&= (\mathbf{w}_1^T\mathbf{x} + w_{10}) - (\mathbf{w}_2^T\mathbf{x} + w_{20}) \\
&= (\mathbf{w}_1 - \mathbf{w}_2)^T\mathbf{x} + (w_{10} - w_{20}) \\
&= \mathbf{w}^T\mathbf{x} + w_0
\end{aligned}
$$

- Decision rule:

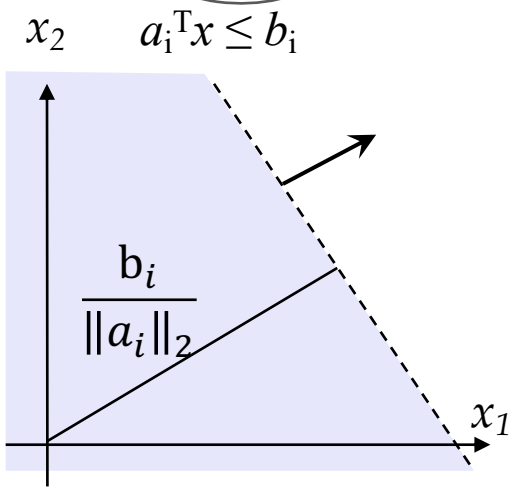
$$
\text{Choose} \begin{cases} C_1 & \text{if } g(\mathbf{x}) > 0 \\ C_2 & \text{otherwise} \end{cases}
$$

Geometry Interpretation: Hyperplane

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



Recall Linear Programming



Geometric Interpretation (扩展内容)

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

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

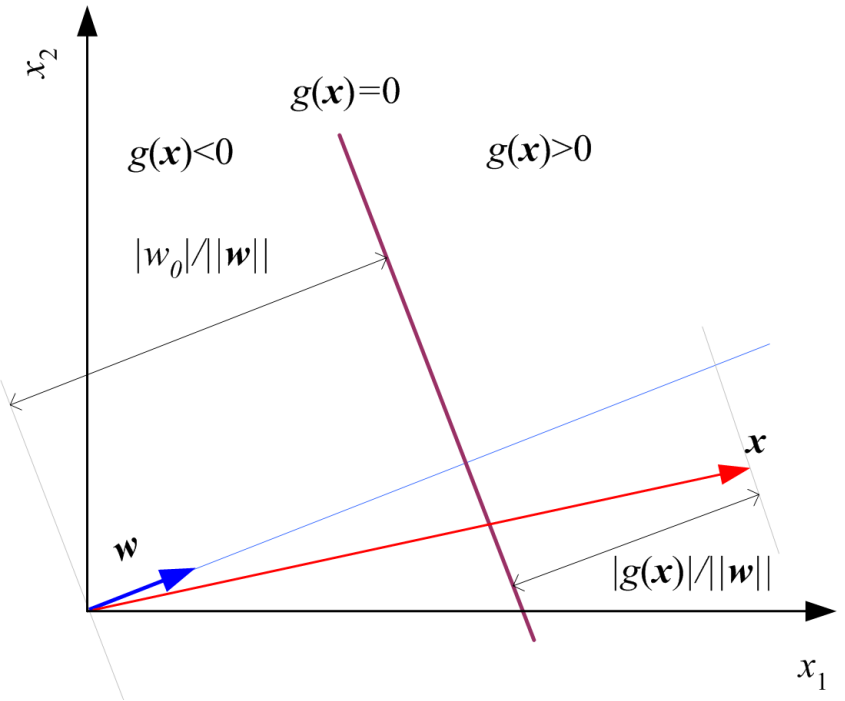
  r: distance from $\boldsymbol{x}$ to the hyperplane

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

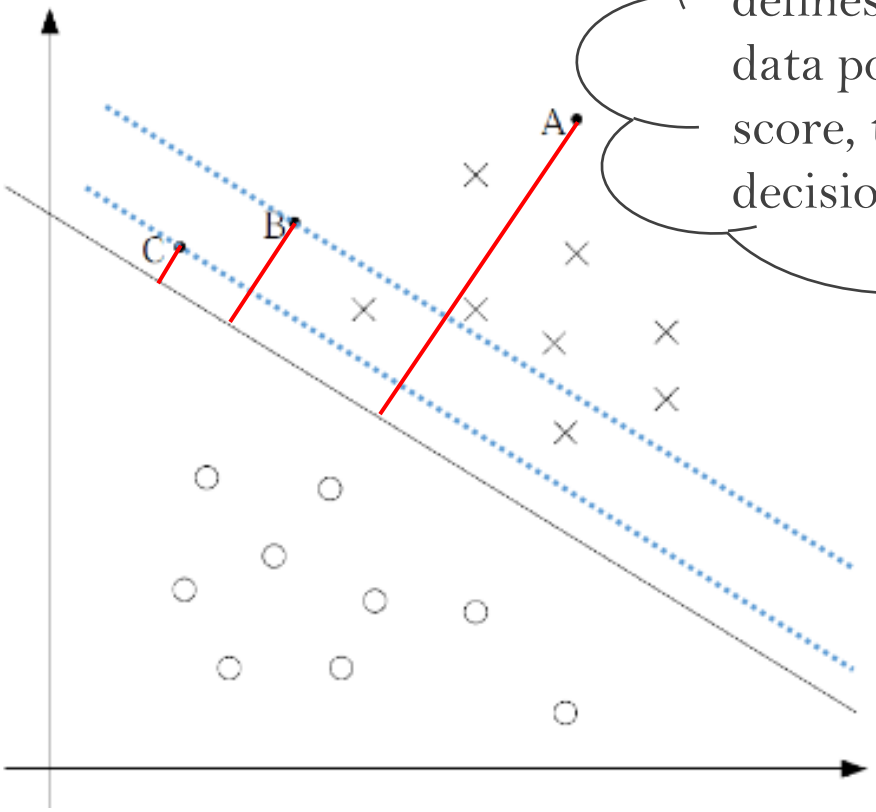
$$\Downarrow$$

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

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



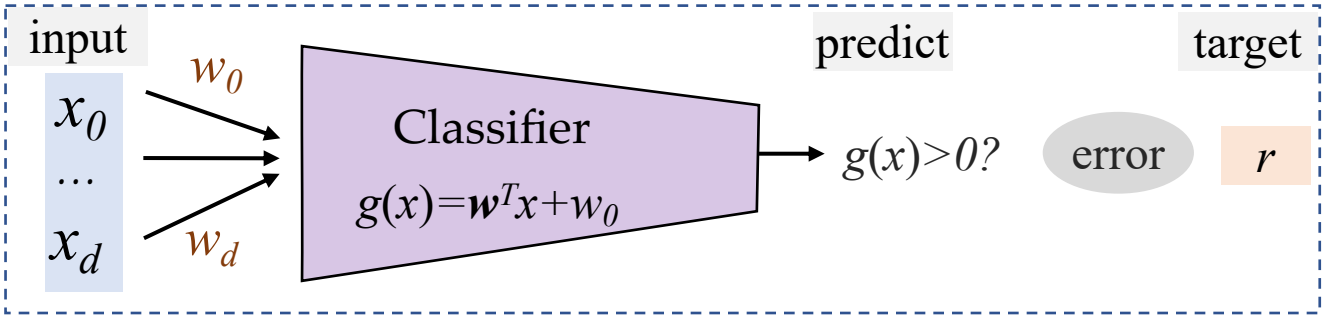
Geometric Interpretation (扩展内容)



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

- The first, naïve linear classifier. (to introduce in future lectures)



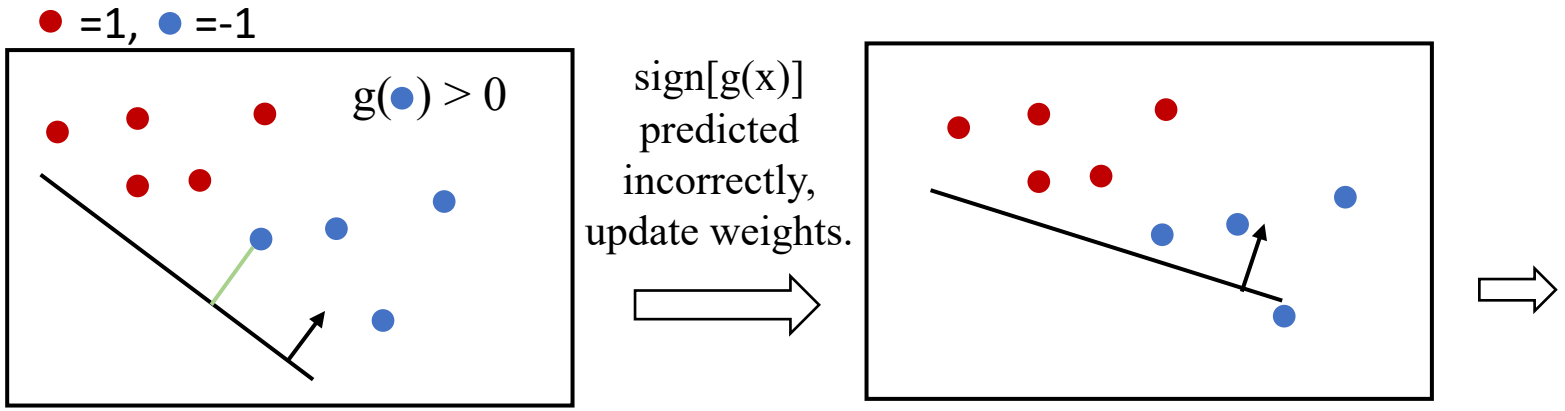
- Train:
  - estimate the parameters $\mathbf{w}$ and w_0 from data
- Predict:
  - calculate $g(x) = \mathbf{w}^T x+w_0$ and choose C_1 if $g(x) > 0$ or choose C_2 if $g(x)<0$.

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

Training

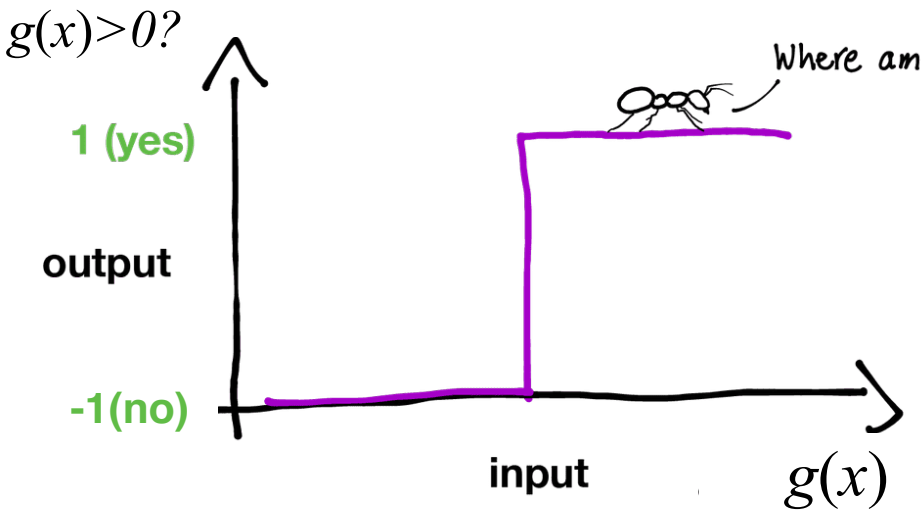
Let $\boldsymbol{x}$ denote the data input, and $r \in \{1,-1\}$ (means the sign of $g(x)$) denotes the **label** of target classes.

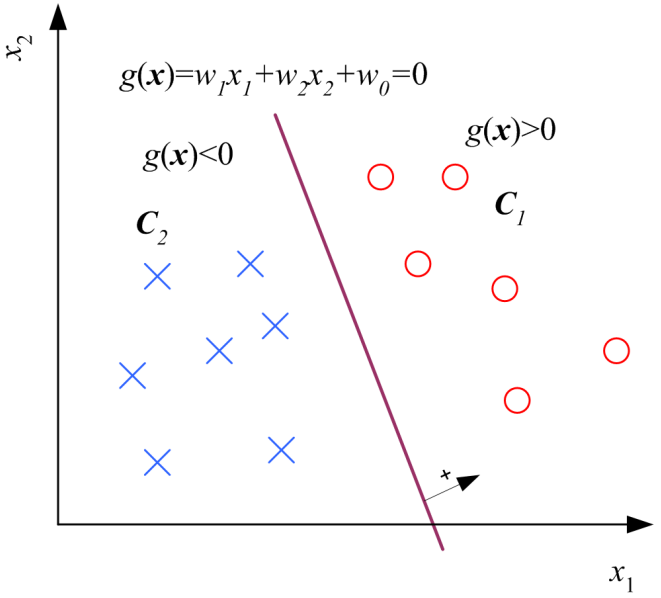
Input: dataset $D = \{(x^{(1)}, r^{(1)}), (x^{(2)}, r^{(2)}), \ldots (x^{(N)}, r^{(N)})\}$
for each training instance $(x^{(\ell)}, r^{(\ell)}) \in D$:
if $r^{(\ell)}\, g_i(x^{(\ell)}) \leq 0$: // a misclassification occurs
$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta\, r^{(\ell)}\, x^{(\ell)}$ // move the hyperplane (defined by $\boldsymbol{w}$) 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 to what extent?
(0-1 decision is too hard to optimize)

Linear Classification with Uncertainty

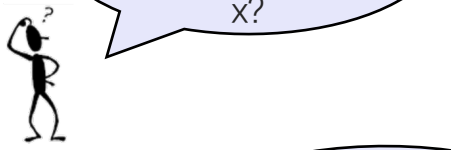
- What is the posterior probability of choosing C1 (or C2)?

Let

$$P(C_1 \mid \boldsymbol{x}) = y \qquad P(C_2 \mid \boldsymbol{x}) = 1 - y$$

Classification rule:

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

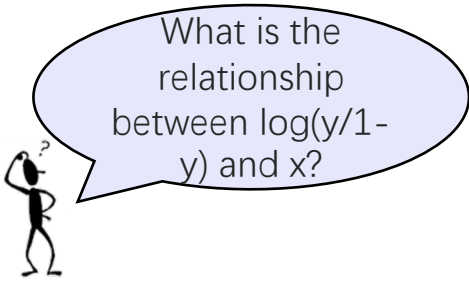


Equivalent Rule:

$$\frac{y}{1-y} > 1 \qquad \textit{or} \qquad \log \frac{y}{1-y} > 0$$

odds of y

Log odds (logit function) of y



Logit Function (扩展内容)

Assume that x follows the Gaussian distribution in each class.

- Logit function:

$$\text{logit}(P(C_1 \mid \mathbf{x})) \quad \log \frac{y}{1-y}$$

$$= \log \frac{P(C_1 \mid \mathbf{x})}{1 - P(C_1 \mid \mathbf{x})} = \log \frac{P(C_1 \mid \mathbf{x})}{P(C_2 \mid \mathbf{x})}$$

$$= \log \frac{p(\mathbf{x} \mid C_1)}{p(\mathbf{x} \mid C_2)} + \log \frac{P(C_1)}{P(C_2)}$$

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

$$= \log \frac{(2\pi)^{-d/2}|\mathbf{\Sigma}|^{-1/2} \exp[-(1/2)(\mathbf{x}-\boldsymbol{\mu}_1)^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_1)]}{(2\pi)^{-d/2}|\mathbf{\Sigma}|^{-1/2} \exp[-(1/2)(\mathbf{x}-\boldsymbol{\mu}_2)^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_2)]} + \log \frac{P(C_1)}{P(C_2)}$$

$$= \mathbf{w}^T \mathbf{x} + w_0$$

where

$$\mathbf{w} = \mathbf{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

$$w_0 = -\frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2)\mathbf{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) + \log \frac{P(C_1)}{P(C_2)}$$

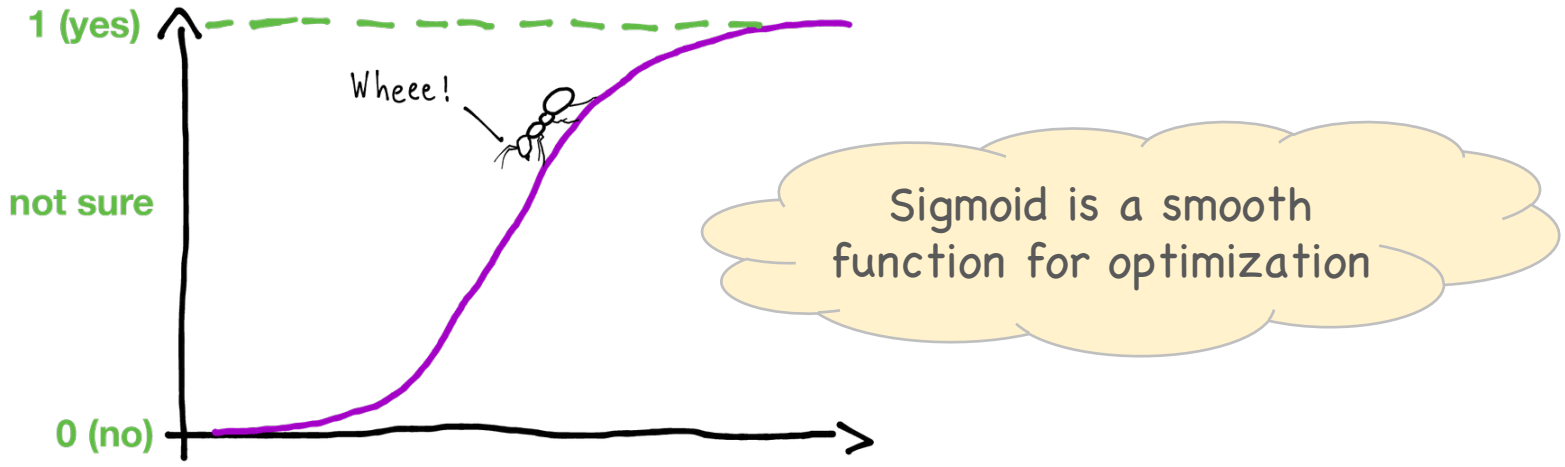
The logit (log odds) of the posterior probability is a linear combination (regression) of the input x~!!

The Sigmoid Function

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

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

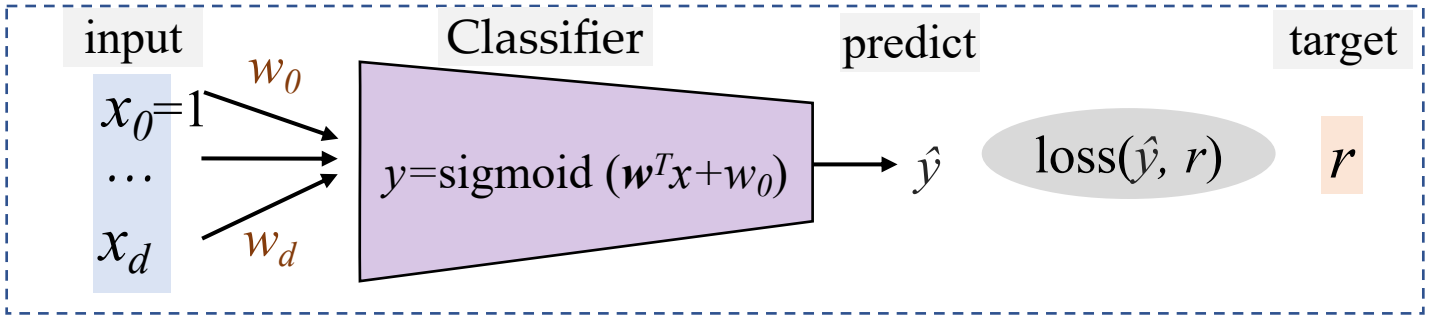
- The sigmoid function (or logistic function) is the **inverse function** of logit, which directly computes the **posterior** class probability $P(C_1|x)$.



Logistic Regression

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

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



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

Loss Function

$y \equiv \text{p}(C_1|x) = \text{sigmoid}(w^{\text{T}}x+w_0)$ 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 writhe this succinctly as a cross-entropy loss:

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

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

- Equivalent to maximize the likelihood:

$$r \,|\, x \sim \text{Bernoulli}(y)$$

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

Training

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

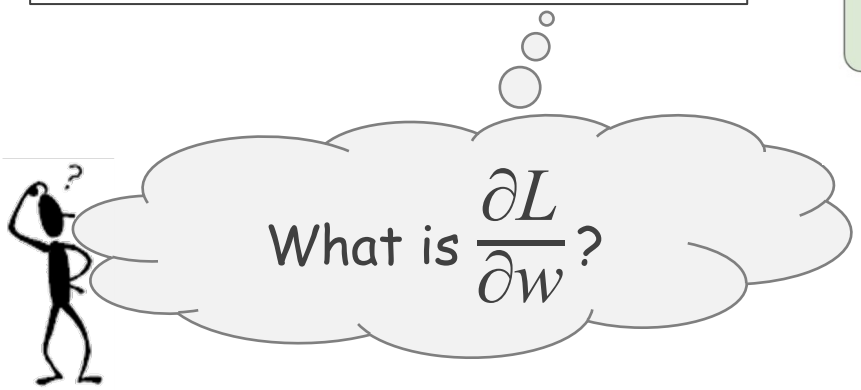
minimize the loss function using gradient descend:

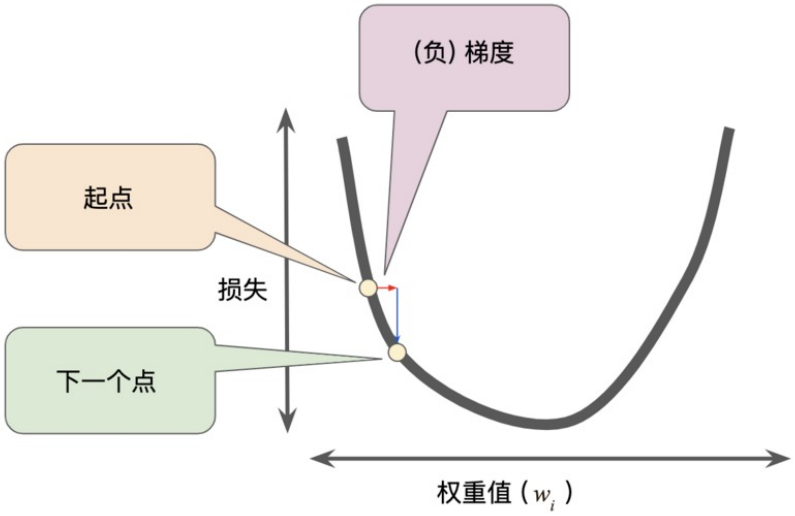
- Goal:

$$\min_w L(\boldsymbol{w})$$

- Iteration:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \frac{\partial L}{\partial w}$$





Optimization – Gradient Descend

$$\ell(\boldsymbol{w}, w_0 \mid x, r) = -\, r \log y - (1-r) \log (1-y)$$

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

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

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

For each $\boldsymbol{w}_j$ (j =1,…,d):

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

Chain rule

Gradient-Descend Learning (2)

For each $\boldsymbol{w}_j$ (j =1,…,d):

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

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

So,

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

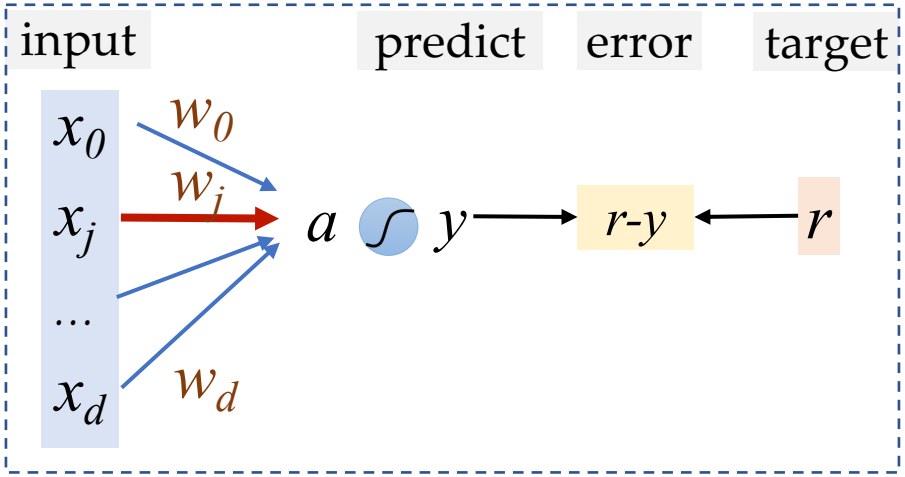
Gradient-Descend Learning (3)

- An interesting point

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

error input

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



The Algorithm

Gradient Descend for LR

Input: $D = \{(\mathrm{x}^{(l)}, \mathrm{r}^{(l)})\}$ $(l = 1{:}N)$
for $j = 0,\ldots,d$
    $\boldsymbol{w}_j \leftarrow rand\ (-0.01, 0.01)$
repeat
    for $j = 0,\ldots,d$
        $\Delta \boldsymbol{w}_j \leftarrow 0$
    for $l = 1,\ldots,N$
        $a \leftarrow 0$
        for $j = 0, \ldots, d$
            $a \leftarrow a + \boldsymbol{w}_j x_j^{(l)}$
        $y \leftarrow \text{sigmoid}\ (a)$
        $\Delta \boldsymbol{w}_j \leftarrow \Delta \mathbf{w}_j + (r^{(l)} - y) x_j^{(l)}$
    for $j = 0,\ldots,d$
        $w_j \leftarrow w_j + \eta \Delta w_j$
until convergence

Multiple Classes

- How about classification with more than 2 categories?

Binary Classification

- Spam
- Not spam

Targets from a binary set {0, 1}.

Multiclass Classification

- Dog
- Cat
- Horse
- Fish

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

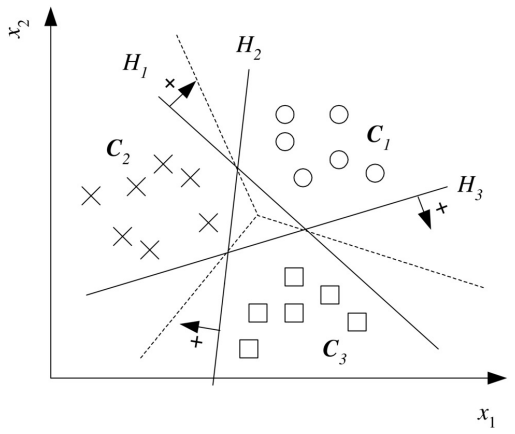
Linear Classifier for Multiclass

- K discriminant functions:

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

- Linearly separable classes:

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



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 $\mathcal{R}_i$.

Multiclass Classification with Uncertainty

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

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

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

So we have

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

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

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:

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

Softmax Function

- In general, the **softmax function** is defined as

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

where the inputs $a_i = \boldsymbol{W}_i x + w_{i0}$ are called the **logits**. $\boldsymbol{W}_i$ and w_{i0} are the trainable parameters.

$$\text{if } a_k \gg a_j,\ \forall j \neq k, \text{ then } p(\mathcal{C}_k|\mathbf{x}) \approx 1,\ p(\mathcal{C}_j|\mathbf{x}) \approx 0.$$

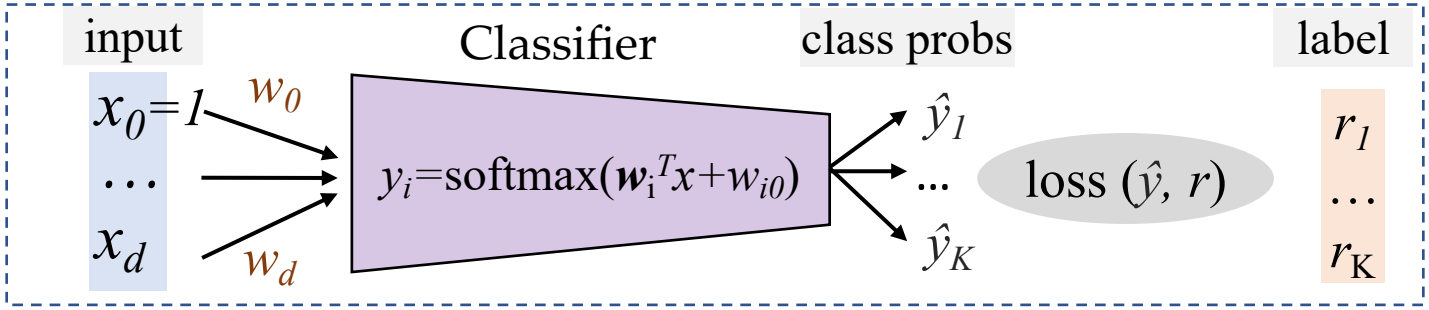
- If one of the a_k's is much larger than the others, softmax $(a_1, \ldots a_K)$ is a smoothed approximation of argmax. (so really it's more like "soft-argmax".)
  - "max" because it amplifies probability of the largest logit a_i.
  - "soft" because still assigns some probability to smaller a_j.
- The 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 sigmoid?

Softmax Regression

- A classifier that estimates the decision boundaries as **Softmax functions**:

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



- Train:
  - estimate the parameters $\boldsymbol{w}_i$ and w_{i0} $(i=1{:}K)$ from data
- Test:
  - calculate $y_i = \text{softmax}\left(\boldsymbol{w}_i^{\mathrm{T}} x + w_0\right)$ and choose C_i if $y_i = \max\{y_{1:K}\}$ (y can be interpreted as a posterior probability).

Loss Function

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

- For a given input $\boldsymbol{x}$, the model outputs a vector of class probabilities $\boldsymbol{y} = (y_1, \ldots y_K)$, and the label of target class is a one-hot vector $\boldsymbol{r} = (r_1, \ldots, r_K)$ (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:

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

where the *log* is applied elementwise.

Training

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

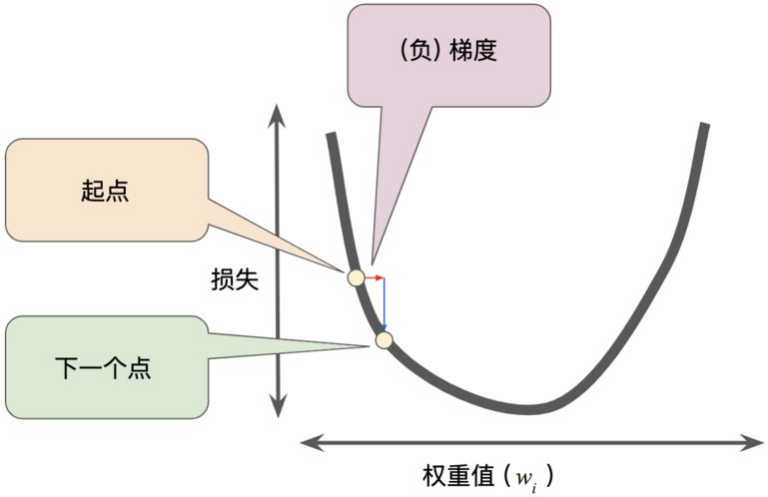
- Goal:

$$\min_w L(\boldsymbol{w})$$

- Iteration:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \frac{\partial L}{\partial w}$$

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



Optimization – Gradient Descend*

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

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

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

Hint: if $y_i = \exp(a_i)/\Sigma_j \exp(a_j)$, 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,…,K), given $\Sigma_i r_i^{(\ell)} = 1$:

$$-\frac{\partial L}{\partial w_j} = \sum_l \sum_i \frac{\partial L}{\partial y_i^{(\ell)}} \frac{\partial y_i^{(\ell)}}{\partial a^{(\ell)}} \frac{\partial a^{(\ell)}}{\partial w_j} = \sum_l \sum_i r_i^{(\ell)} (\delta_{ij} - y_j^{(l)}) \boldsymbol{x}^{(\ell)}$$

$$= \sum_l \left[\sum_i r_i^{(\ell)} \delta_{ij} - y_j^{(\ell)} \sum_i r_i^{(\ell)} \right] x^{(\ell)} = \sum_l (r_j^{(\ell)} - y_j^{(\ell)}) \boldsymbol{x}^{(\ell)}$$

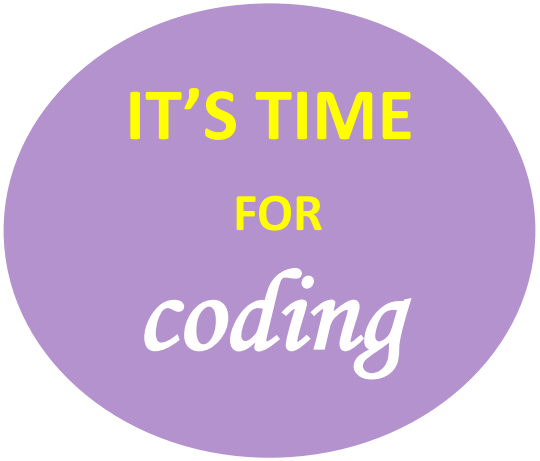
The Algorithm

Gradient Descend for Softmax Regression

for $i = 1, \ldots, K$, **for** $j = 0,\ldots,d$,
  $w_{ij} \leftarrow$ rand (-0.01, 0.01) // initialization
repeat
  for $i = 1,\ldots,K$, **for** $j = 0,\ldots, d$, $\Delta w_{ij} \leftarrow 0$
  for $l = 1,\ldots,N$
    for $i = 1,\ldots,K$
      $a_i \leftarrow 0$
      for $j = 0,\ldots, d$
        $a_i \leftarrow a_i + w_{ij}x_j^{(l)}$
    for i $= 1,\ldots, K$
      $y_i \leftarrow \exp(a_i)/\boldsymbol{\Sigma}_j\exp(a_j)$
    for $i = 1,\ldots,K$
      for $j = 0,\ldots, d$
        $\Delta w_{ij} \leftarrow \Delta w_{ij}+(r_i^{(l)}-y_i)x_j^{(l)}$
  for $i = 1,\ldots,K$
    for $j = 0,\ldots, d$
      $w_{ij} \leftarrow w_{ij}+ \boldsymbol{\eta}\Delta w_{ij}$
until convergence

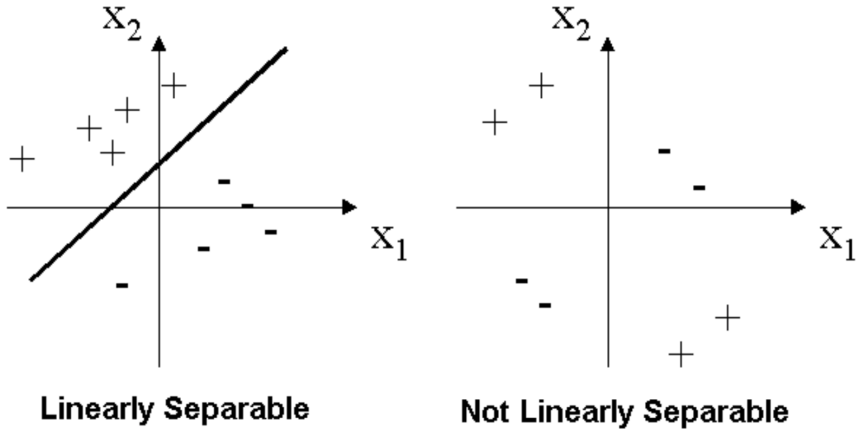
Tutorial: Logistic regression from scratch with Python

https://www.kaggle.com/code/ujjalkumarmaity/logistic-regression-from-scratch

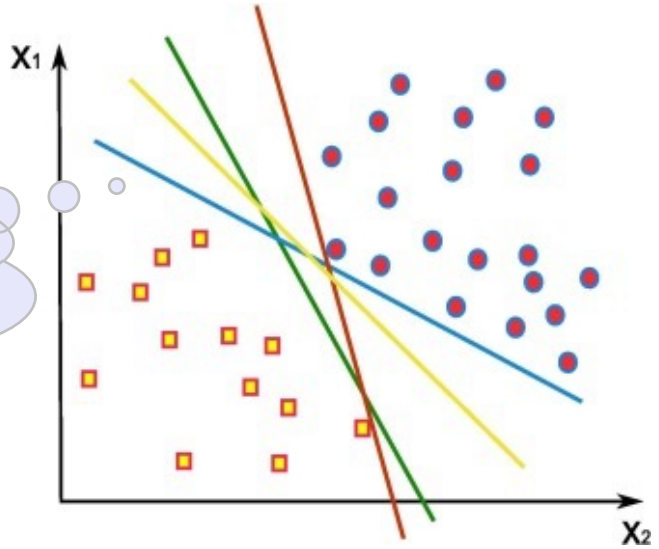


Limitations of Linear Classifiers

Requires data to be linearly separable



Even if the data is linearly separable, there can be many separations.



What's Next?

Support Vector Machine

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