Linear Classifiers

Qinliang Su （苏勤亮）

Sun Yat-sen University

suqliang@mail.sysu.edu.cn

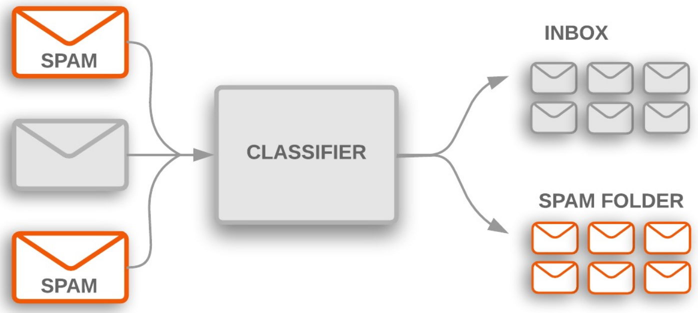
Outline

- Two-class Case
- Multi-class Case

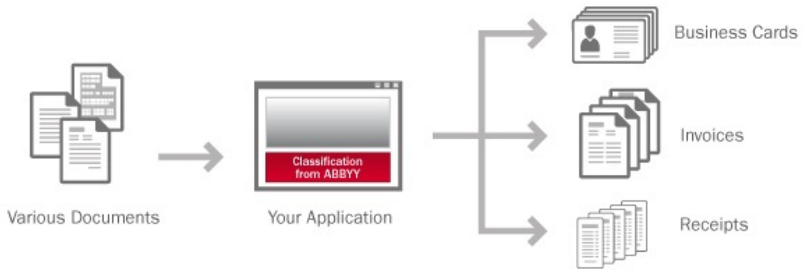
Examples

- Image category classification

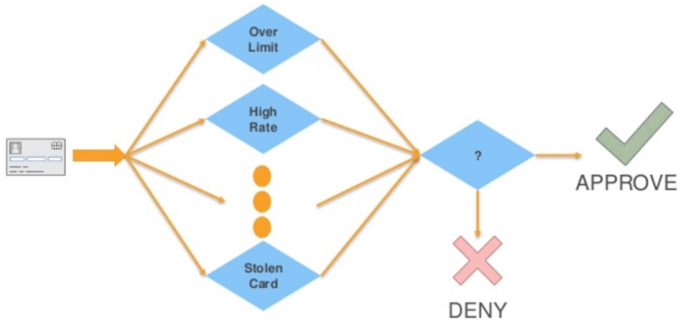
- Spam e-mails detection



- Document automatic categorization

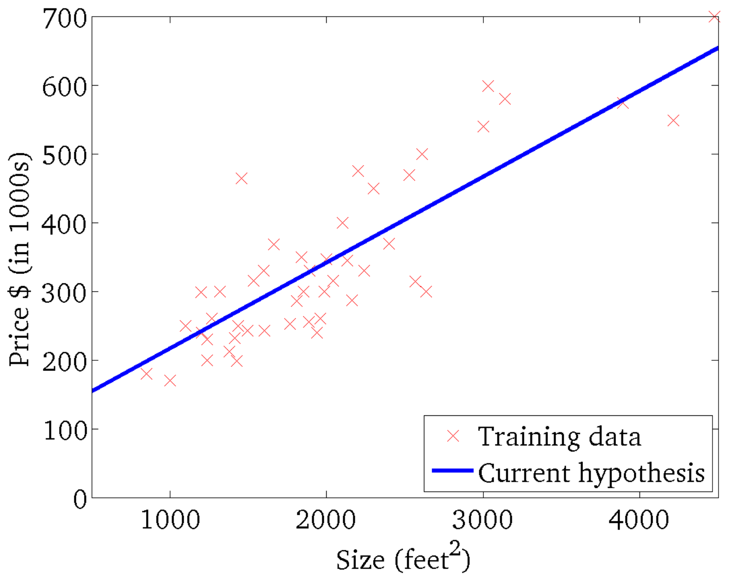


- Transaction fraud detection



Logistic Regression

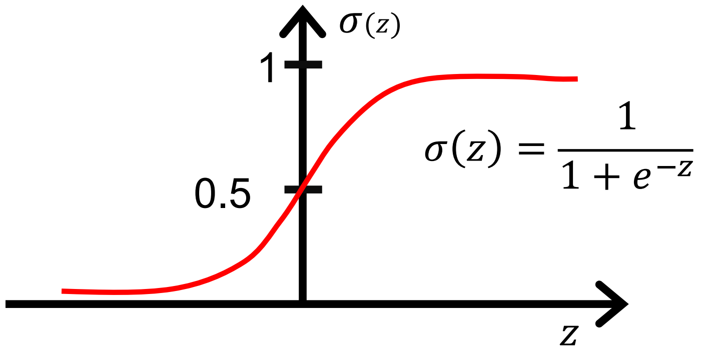
- In classification, the target variable $y \in \{0, 1\}$
- In linear regression, the output $f(x) = \boldsymbol{x}\boldsymbol{w}$ fall in the range $[-\infty, +\infty]$



- The output value of linear regression is not compatible with the target values in classification tasks

- Sigmoid/logistic function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



- Logistic regression

$$f(\boldsymbol{x}) = \sigma(\boldsymbol{xw})$$

- Linear regression

$$f(\boldsymbol{x}) = \boldsymbol{xw}$$

- The output range is transformed from $[-\infty, +\infty]$ to $[0, 1]$

Cost Function

- Goal
  - If the true label $y = 1$, we want $f(\boldsymbol{x}) = \sigma(\boldsymbol{xw})$ to be close to 1
  - If the true label $y = 0$, we want $f(\boldsymbol{x}) = \sigma(\boldsymbol{xw})$ to be close to 0

- To achieve this goal, we can define a cost function similar to that in regression

$$L(\boldsymbol{w}) = (\sigma(\boldsymbol{xw}) - y)^2$$

- Alternatively, we can also seek to minimize

$$L(\boldsymbol{w}) = \begin{cases} -\log(\sigma(\boldsymbol{x}\boldsymbol{w})) & \text{if } y = 1 \\ -\log(1 - \sigma(\boldsymbol{x}\boldsymbol{w})) & \text{if } y = 0 \end{cases}$$

- The objective above can be equivalently written as

$$\boxed{L(\boldsymbol{w}) = -y\log(\sigma(\boldsymbol{x}\boldsymbol{w})) - (1-y)\log(1-\sigma(\boldsymbol{x}\boldsymbol{w}))}$$

If $y = 1$, $L(w)$ reduces to $L(w) = -log(\sigma(xw))$;
Otherwise, if $y = 0$, $L(w)$ reduces to $L(w) = -log(1 - \sigma(xw))$

- The loss above is called the *cross-entropy loss*

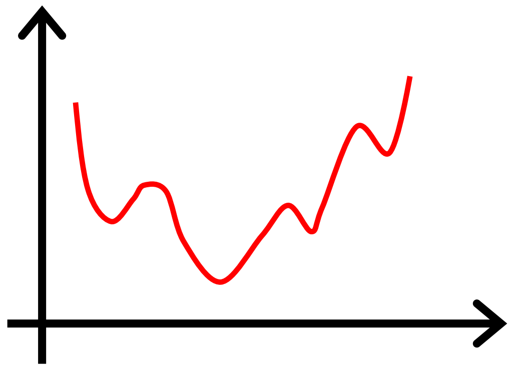
- Which cost function is better?

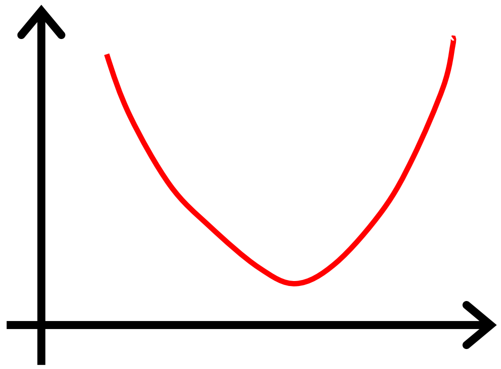
What's the shape of the function $f(z) = \log(\sigma(z))$?

Squared error: $L(\boldsymbol{w}) = (\sigma(\boldsymbol{xw}) - y)^2$

Cross entropy: $L(\boldsymbol{w}) = -\left[y \log\left(\sigma(\boldsymbol{xw})\right) + (1-y)\log\left(1-\sigma(\boldsymbol{xw})\right)\right]$

- ➢ Squared loss is non-convex
- ➢ Cross entropy is convex





Convex function is easier to optimize

- In next lecture, another advantage of using cross-entropy loss will be manifested from the perspective of *accurate modeling*

Gradient Descent

- The gradient of the cross-entropy loss

$$\boxed{\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{1}{N}\sum_{\ell=1}^{N}\left[\sigma\left(\boldsymbol{x}^{(\ell)}\boldsymbol{w}\right) - y^{(\ell)}\right]\boldsymbol{x}^{(\ell)T}}$$

- The optimal $\boldsymbol{w}^*$ can be obtained by solving $\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} = 0$. But here, *the analytical solution does not exist*
- Thus, we can only resort to the numerical methods
  - Gradient descent
  - Newton methods
  - Coordinated descent
  - ……

- Since the cross-entropy loss is convex, the gradient descent

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - r \cdot \frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}}$$

is guaranteed to converge to the optimal value $\boldsymbol{w}^*$

- By examining the gradient

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{1}{N} \sum_{\ell=1}^{N} \Bigg[\underbrace{\sigma\left(\boldsymbol{x}^{(\ell)} \boldsymbol{w}\right) - y^{(\ell)}}_{\textit{prediction error}} \Bigg] \boldsymbol{x}^{(\ell)T}$$

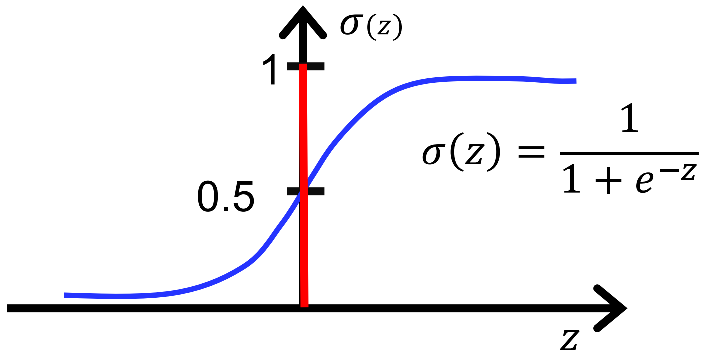
we see that the GD always seeks to reduce the prediction error

- If $y^{(\ell)} = 1$, the algorithm drives $\sigma\left(\boldsymbol{x}^{(\ell)} \boldsymbol{w}\right)$ towards 1
- If $y^{(\ell)} = 0$, the algorithm drives $\sigma\left(\boldsymbol{x}^{(\ell)} \boldsymbol{w}\right)$ towards 0

Decision Boundary

- The sample is classified into 1 and 0 as

$$\hat{y} = \begin{cases} 1, & \text{if } \sigma(\boldsymbol{xw}) \geq 0.5 \\ 0, & \text{if } \sigma(\boldsymbol{xw}) < 0.5 \end{cases}$$

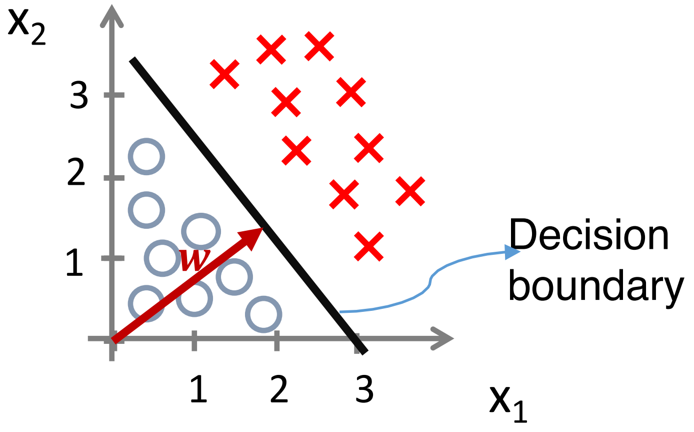


- This is equivalent to classify the samples as

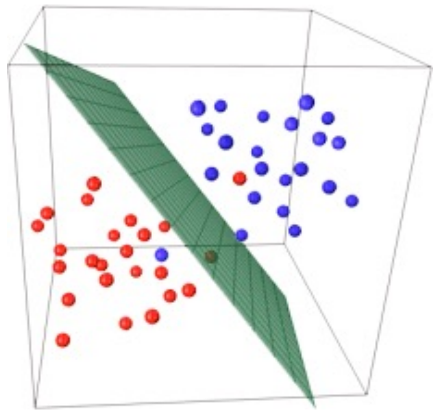
$$\hat{y} = \begin{cases} 1, & \text{if } \boldsymbol{xw} \geq 0 \\ 0, & \text{if } \boldsymbol{xw} < 0 \end{cases}$$

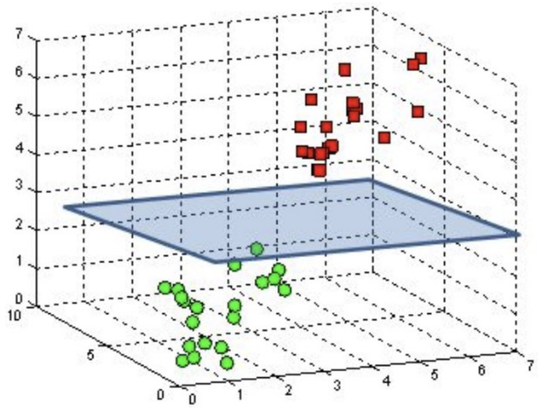
- The decision boundary consists of $\boldsymbol{x}$ *that satisfies* $\boldsymbol{xw} = 0$

- Since $\boldsymbol{w}$ is a vector, all $\boldsymbol{x}$ that satisfies $\boldsymbol{x}\boldsymbol{w} = 0$ constitute *a space that is orthogonal to* $\boldsymbol{w}$



- In the two-dimensional case, the space is a straight line
- In the three-dimensional case, the space is a plane



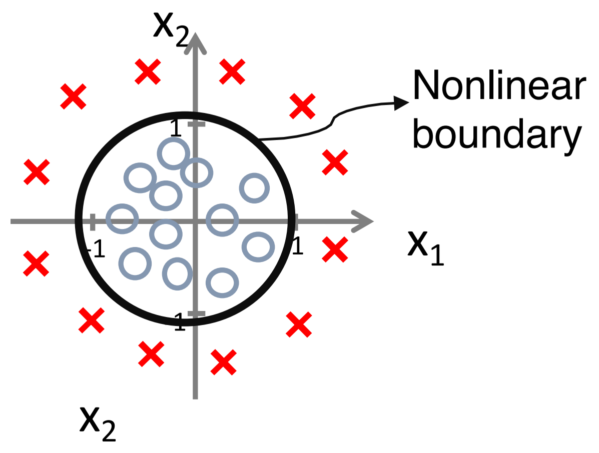


- For a fixed vector $\boldsymbol{w} \in \mathbb{R}^K$, the set of points

$$\boldsymbol{x} \in \{\boldsymbol{x} | \boldsymbol{x}\boldsymbol{w} = 0\}$$

constitute a $(K-1)$-dimensional *hyper-plane*

- The hyper-planes can *never represent a nonlinear decision boundary, e.g.*,

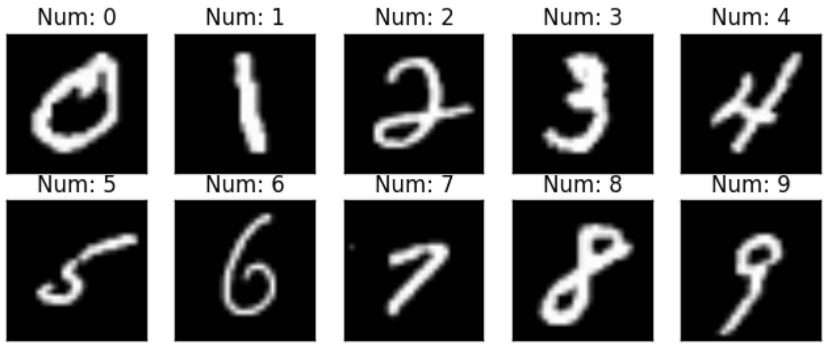


That's the reason why logistic regression is called as a linear classifier

Outline

- Two-class Case
- Multi-class Case

- Many applications contain more than 2 classes

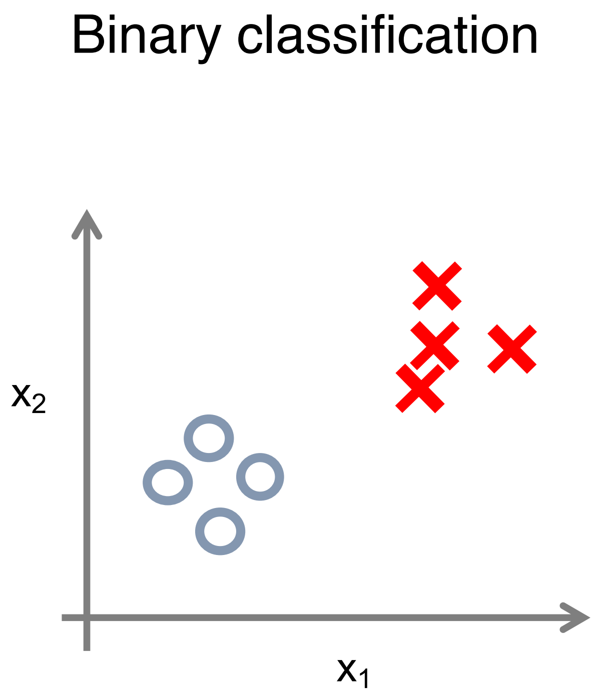


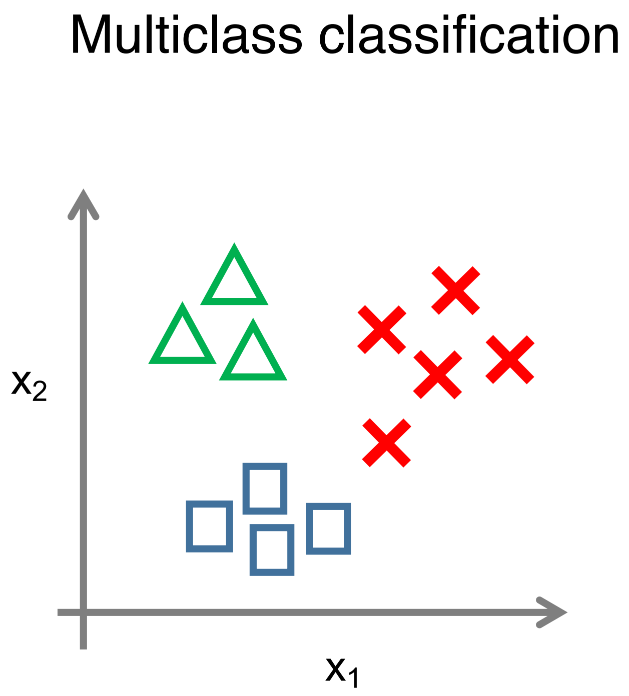
- Two methods to deal with multi-class classification
  - ➢ One-vs-All

    Transform the multi-class problem into multiple binary problems
  - ➢ Softmax function

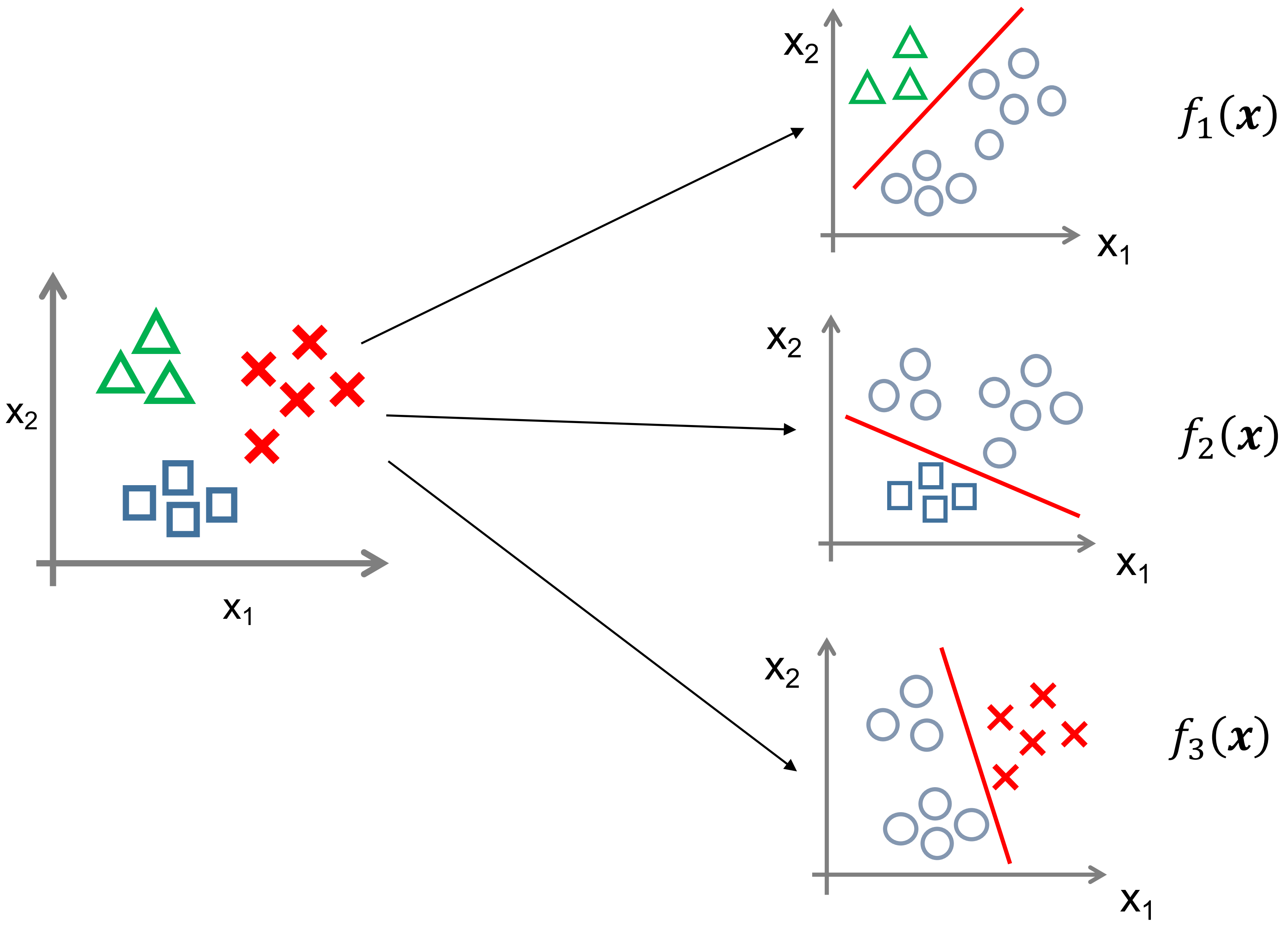
    Classifying the sample into one of the classes directly

One-vs-All


Binary classification
x_2
x_1

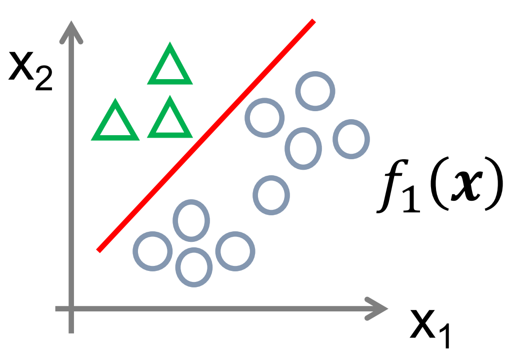

Multiclass classification
x_2
x_1

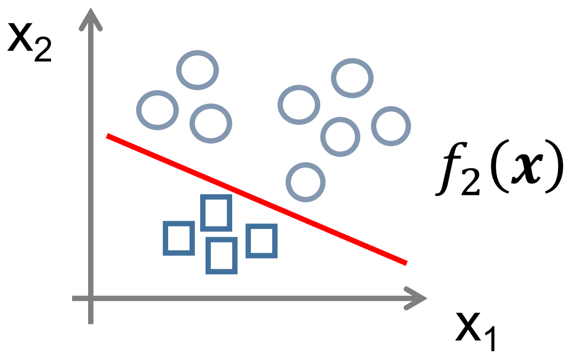
- Train a classifier for each class

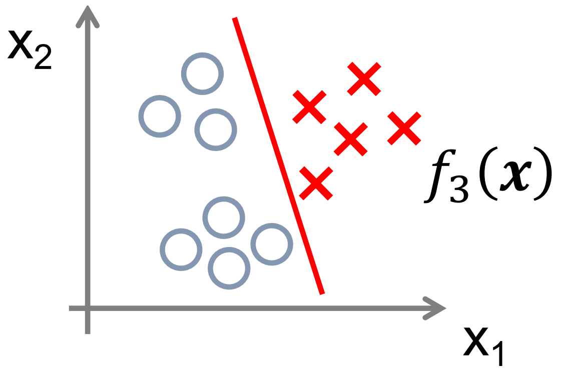


- To predict the class for a new sample $\boldsymbol{x}$, pick the class such that

$$k = \arg\max_i f_i(\boldsymbol{x})$$







Softmax Function

- Softmax function

$$\boxed{softmax_i(\boldsymbol{z}) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}}$$

It can be seen that $\sum_{i=1}^{K} softmax_i(\boldsymbol{z}) = 1$

- The probability that a data $\boldsymbol{x}$ is classified to the i-th class is

$$f_i(\boldsymbol{x}) = softmax_i(\boldsymbol{x}\boldsymbol{W}) = \frac{e^{\boldsymbol{x}\boldsymbol{w}_i}}{\sum_{k=1}^{K} e^{\boldsymbol{x}\boldsymbol{w}_k}}$$

where $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_K]$

- If $\boldsymbol{x}$ belongs to the i-th class, the model should encourage $f_i(\boldsymbol{x})$ to be as large as possible

- The softmax function with $K = 2$ is equivalent to logistic function
  - Under the two-class case, we have

$$softmax_1(\boldsymbol{xW}) = \frac{e^{\boldsymbol{xw}_1}}{e^{\boldsymbol{xw}_1} + e^{\boldsymbol{xw}_2}} \qquad softmax_2(\boldsymbol{xW}) = \frac{e^{\boldsymbol{xw}_2}}{e^{\boldsymbol{xw}_1} + e^{\boldsymbol{xw}_2}}$$

$$= \frac{1}{1 + e^{-\boldsymbol{x}(\boldsymbol{w}_1 - \boldsymbol{w}_2)}} \qquad = \frac{e^{-\boldsymbol{x}(\boldsymbol{w}_1 - \boldsymbol{w}_2)}}{1 + e^{-\boldsymbol{x}(\boldsymbol{w}_1 - \boldsymbol{w}_2)}}$$

  - It can be seen that

$$softmax_1(\boldsymbol{xW}) = \sigma\big(\boldsymbol{x}(\boldsymbol{w}_1 - \boldsymbol{w}_2)\big)$$

$$softmax_2(\boldsymbol{xW}) = 1 - \sigma\big(\boldsymbol{x}(\boldsymbol{w}_1 - \boldsymbol{w}_2)\big)$$

The two-class softmax classification is equivalent to the logistic regression, with the model parameter to be $\boldsymbol{w}_1 - \boldsymbol{w}_2$

Cost Function

- For a training dataset with K classes, its label $\boldsymbol{y}$ is represented by a one-hot vector, which is illustrated as below

$$\begin{gathered}[1,0,0,\cdots,0],\\ [0,1,0,\cdots,0],\\ \vdots\\ [0,0,0,\cdots,1]\end{gathered}$$

- The objective is to maximize the corresponding probability $f_i(\boldsymbol{x})$. Thus, the cost function can be written as

$$L(\boldsymbol{w}_1,\boldsymbol{w}_2,\cdots,\boldsymbol{w}_K) = -\frac{1}{N}\sum_{\ell=1}^{N}\sum_{k=1}^{K} y_k^{(\ell)} \log\big[\underbrace{softmax_k(\boldsymbol{x}^{(\ell)}\boldsymbol{W})}_{\text{Cross-entropy loss}}\big]$$

  - $y_k^{(\ell)}$ is the k-th element of $\boldsymbol{y}^{(\ell)}$

Gradient Descent

- The gradient *w.r.t.* $\boldsymbol{w}_j$ is

$$\frac{\partial L(\boldsymbol{w}_1,\boldsymbol{w}_2,\cdots,\boldsymbol{w}_K)}{\partial \boldsymbol{w}_j} = \frac{1}{N}\sum_{\ell=1}^{N}\left(\underbrace{softmax_j\left(\boldsymbol{x}^{(\ell)}\boldsymbol{W}\right) - y_j^{(\ell)}}_{\boldsymbol{prediction\ error}}\right)\boldsymbol{x}^{(\ell)T}$$

Note that all $\boldsymbol{w}_j$ for $j = 1, \cdots, K$ should be updated simultaneously

- By representing $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_K]$, we have

$$\frac{\partial L(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{N}\sum_{\ell=1}^{N}\boldsymbol{x}^{(\ell)T}\left(softmax\left(\boldsymbol{x}^{(\ell)}\boldsymbol{W}\right) - \boldsymbol{y}^{(\ell)}\right)$$

  - $softmax\left(\boldsymbol{x}^{(\ell)}\boldsymbol{W}\right) = \left[softmax_1\left(\boldsymbol{x}^{(\ell)}\boldsymbol{W}\right), \cdots, softmax_K\left(\boldsymbol{x}^{(\ell)}\boldsymbol{W}\right)\right]$ is a row vector

- Updating: $\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - r \cdot \left.\frac{\partial L(\boldsymbol{W})}{\partial \boldsymbol{W}}\right|_{\boldsymbol{W}=\boldsymbol{W}_t}$