Linear Classifiers

DCS310

Sun Yat-sen University

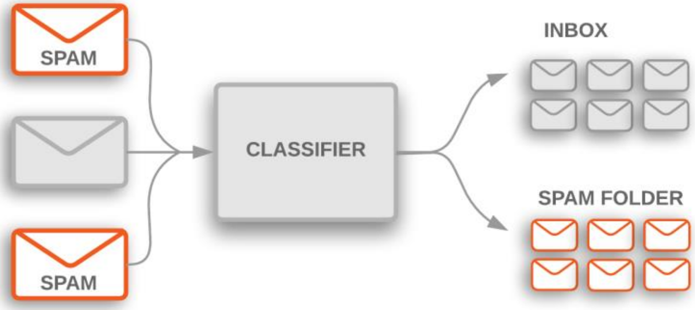
Outline

- Two-class Case
- Multi-class Case

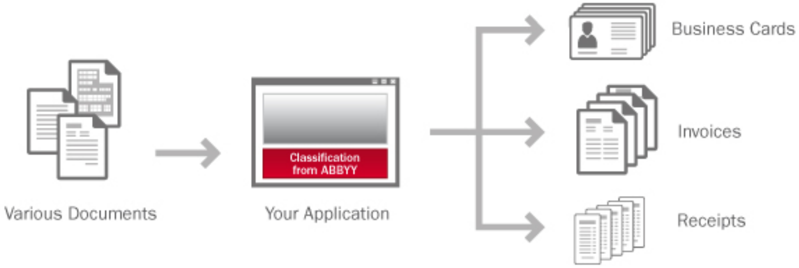
Examples

- Image category classification

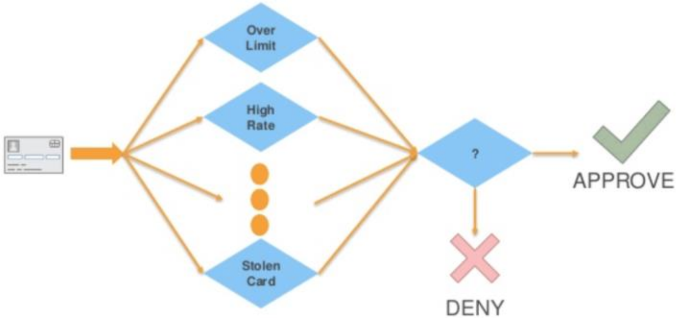
- Spam e-mails detection



- Document automatic categorization

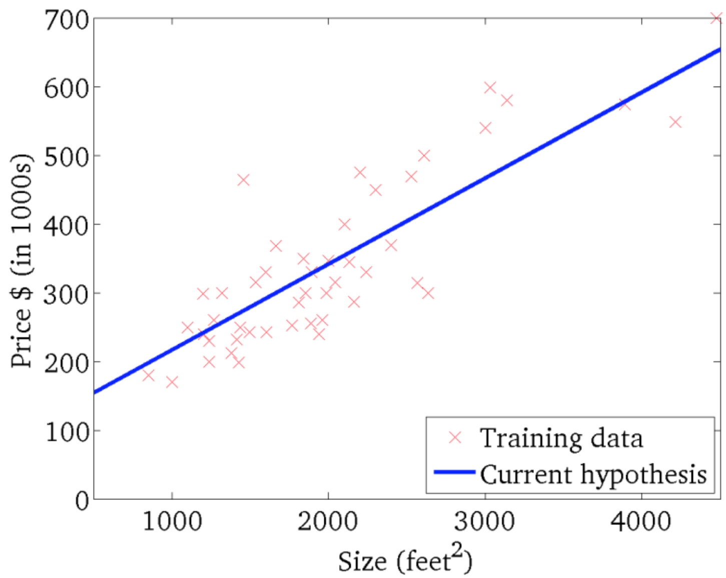


- Transaction fraud detection



Logistic Regression

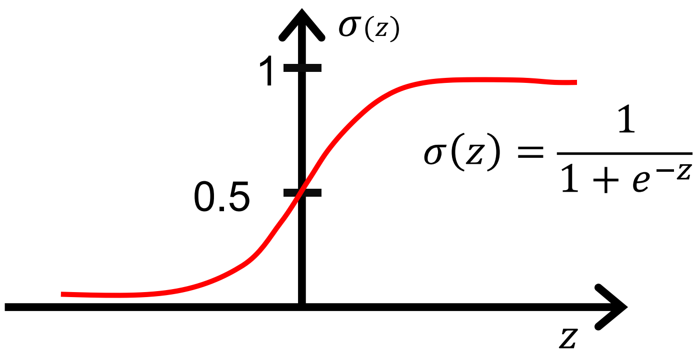
- In classification, the target variable $y \in \{0,1\}$
- In linear regression, the output $f(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{w}$ falls in the range $[-\infty, +\infty]$



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

$$-\log\big(\sigma(\boldsymbol{xw})\big) \text{ if } y = 1 \qquad \text{or} \qquad -\log\big(1-\sigma(\boldsymbol{xw})\big) \text{ if } y = 0$$

- The objective above can be equivalently written as

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

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

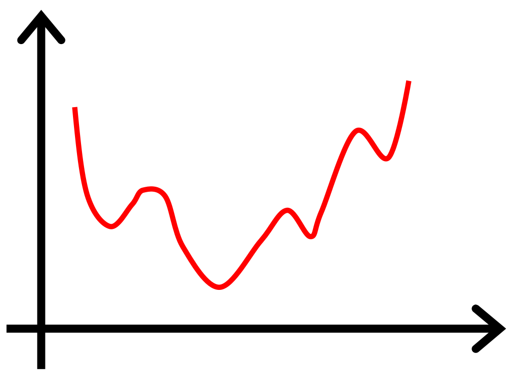
- The loss above is called the *cross-entropy loss*

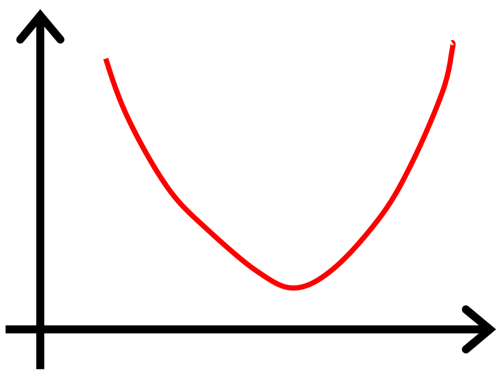
- Which cost function is better?

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

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

- 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 *more accurate modeling*

The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)

DeepCF: A Unified Framework of Representation Learning and Matching Function Learning in Recommender System

Zhi-Hong Deng,[1] Ling Huang,[1] Chang-Dong Wang,[1] Jian-Huang Lai,[1] Philip S. Yu[2,3]
[1]School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China
[2]Department of Computer Science, University of Illinois at Chicago, Chicago, USA
[3]Institute for Data Science, Tsinghua University, Beijing, China
dengzhh7@mail2.sysu.edu.cn, huanglinghl@hotmail.com, changdongwang@hotmail.com
stsljh@mail.sysu.edu.cn, psyu@uic.edu

mentioned in **Problem Statement**, we assume y_{ui} obeys a Bernoulli distribution, i.e., $y_{ui} \sim Bern(p_{ui})$. By replacing p_{ui} with $\hat{y}_{ui}$ in Equation 2, we can define the likelihood function as

$$\begin{aligned} L(\Theta) &= \prod_{(u,i)\in\mathcal{Y}^+\cup\mathcal{Y}^-} P(y_{ui}|\Theta) \\ &= \prod_{(u,i)\in\mathcal{Y}^+\cup\mathcal{Y}^-} \hat{y}_{ui}^{y_{ui}}(1-\hat{y}_{ui})^{1-y_{ui}}, \end{aligned} \tag{3}$$

where $\mathcal{Y}^+$ denotes all the observed interactions in $\mathbf{Y}$ and $\mathcal{Y}^-$ denotes the sampled unobserved interactions, i.e., the negative instances. Furthermore, taking the negative logarithm of the likelihood (NLL), we obtain

$$\begin{aligned} \ell_{BCE} = -&\sum_{(u,i)\in\mathcal{Y}^+\cup\mathcal{Y}^-} y_{ui}\log\hat{y}_{ui} \\ &+ (1-y_{ui})\log(1-\hat{y}_{ui}). \end{aligned} \tag{4}$$

Based on all the above assumptions and formulations, we finally obtain an objective function which is suitable for learning from implicit feedback data, i.e., the binary cross-entropy loss function.

Cross-Domain Explicit–Implicit-Mixed Collaborative Filtering Neural Network

Chang-Dong Wang, *Member, IEEE*, Yan-Hui Chen, Wu-Dong Xi, Ling Huang, *Member, IEEE*, and Guangqiang Xie, *Member, IEEE*

B. Loss Function

To predict the interaction probability $\hat{y}^d_{ui^d}$ $\forall d = 1, 2$, an appropriate loss function needs to be designed. The recommender system focusing on the explicit ratings usually formulates the recommendation problem as a rating prediction problem. Similarly, for the recommendation problem with the implicit interactions, it is often formulated as an interaction prediction problem. For any domain $\mathbb{D}^d$ $\forall d = 1, 2$, assume that the implicit interaction $y^d_{ui^d}$ obeys a Bernoulli distribution, the likelihood function in CEICFNet can be defined as follows:

$$\begin{aligned} L^1(\Theta) &= \prod_{(u,i^1)\in\mathcal{Y}^1_+\cup\mathcal{Y}^1_-} \left(\hat{y}^1_{ui^1}\right)^{y^1_{ui^1}}\left(1-\hat{y}^1_{ui^1}\right)^{1-y^1_{ui^1}} \\ L^2(\Theta) &= \prod_{(u,i^2)\in\mathcal{Y}^2_+\cup\mathcal{Y}^2_-} \left(\hat{y}^2_{ui^2}\right)^{y^2_{ui^2}}\left(1-\hat{y}^2_{ui^2}\right)^{1-y^2_{ui^2}} \end{aligned} \tag{2}$$

where $\mathcal{Y}^d_+$ and $\mathcal{Y}^d_-$, respectively, denote the observed interactions and the randomly selected negative samples in $\mathbb{D}^d$ $\forall d = 1, 2$.

Finally, by taking the negative logarithm of the likelihood (NLL), we obtain the loss function as follows:

$$\begin{aligned} l^1(\Theta) &= -\sum_{(u,i^1)\in\mathcal{Y}^1_+\cup\mathcal{Y}^1_-} y^1_{ui^1}\log\hat{y}^1_{ui^1} + \left(1-y^1_{ui^1}\right)\log\left(1-\hat{y}^1_{ui^1}\right) \\ l^2(\Theta) &= -\sum_{(u,i^2)\in\mathcal{Y}^2_+\cup\mathcal{Y}^2_-} y^1_{ui^2}\log\hat{y}^1_{ui^2} + \left(1-y^1_{ui^2}\right)\log\left(1-\hat{y}^1_{ui^2}\right) \\ l(\Theta) &= l^1(\Theta) + \lambda l^2(\Theta) \end{aligned} \tag{3}$$

where λ is the tradeoff parameter that is utilized to tune the importance of the two domains.

Gradient Descent

- The gradient of the cross-entropy loss

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

- 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
  - Coordinate 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

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

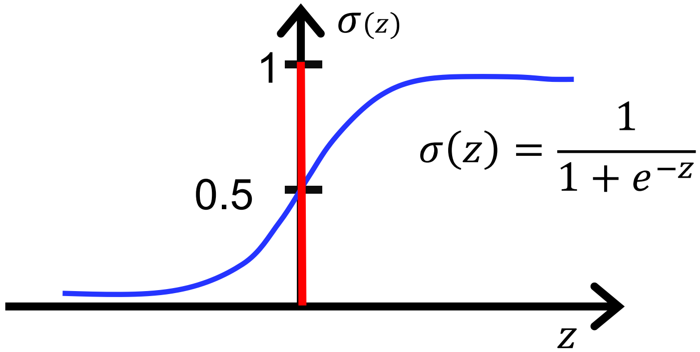
we see that the GD always seeks to reduce the prediction error

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

Decision Boundary

- The sample is classified into 1 and 0 as

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

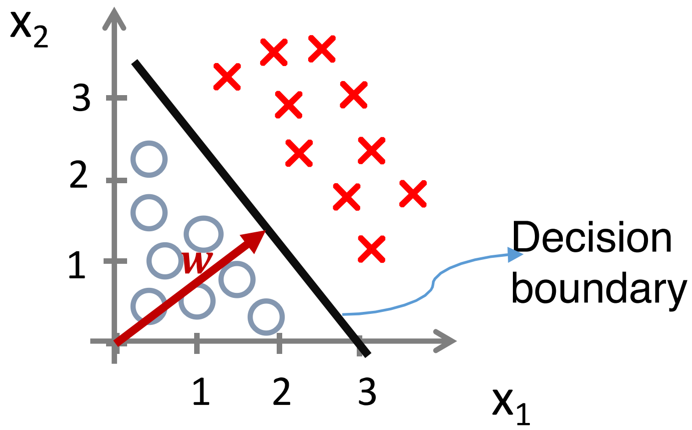


- This is equivalent to classify the samples as

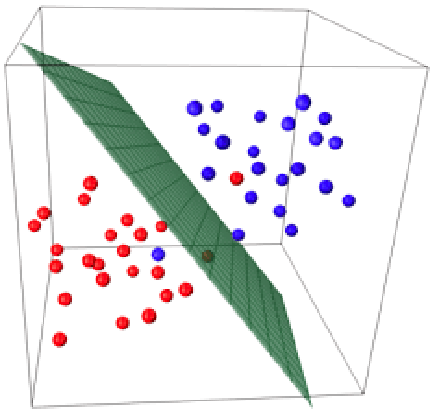
$$\hat{y} = \begin{cases} 1, & \text{if } \boldsymbol{xw} \geq 0 \\ 0, & \text{if } \boldsymbol{xw} < 0 \end{cases}$$

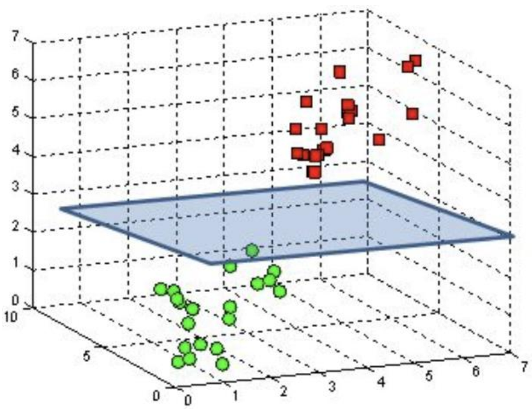
- The decision boundary consists of $\boldsymbol{x}$ *that satisfy* $\boldsymbol{xw} = 0$

- Since $\boldsymbol{w}$ is a vector, all $\boldsymbol{x}$ that satisfies $\boldsymbol{xw} = 0$ 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



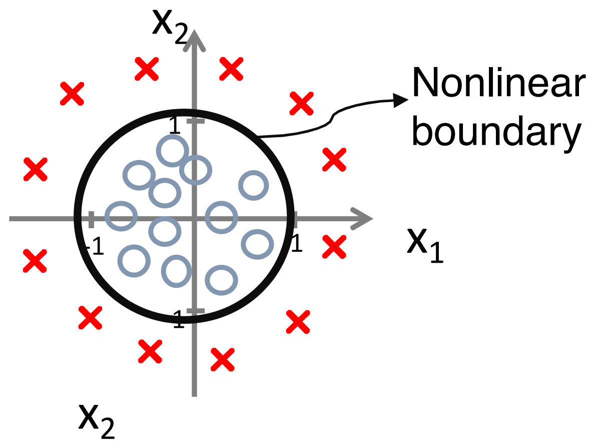


- For a fixed vector $\boldsymbol{w} \in \mathbb{R}^{d\times 1}$, the set of points

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

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

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

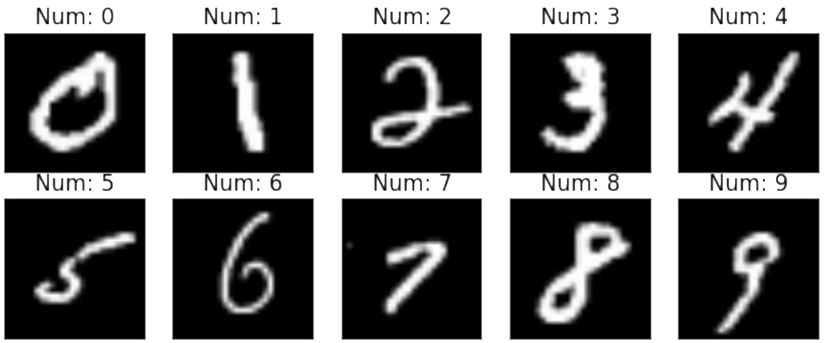


- That's why we call the logistic regression a linear classifier

Outline

- Two-class Case
- Multi-class Case

- Many applications have more than 2 classes



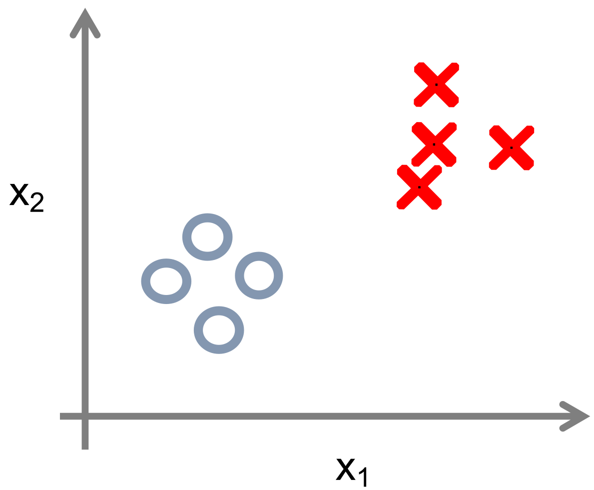
- Two methods to deal with multiclass classification
  - One-vs-All

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

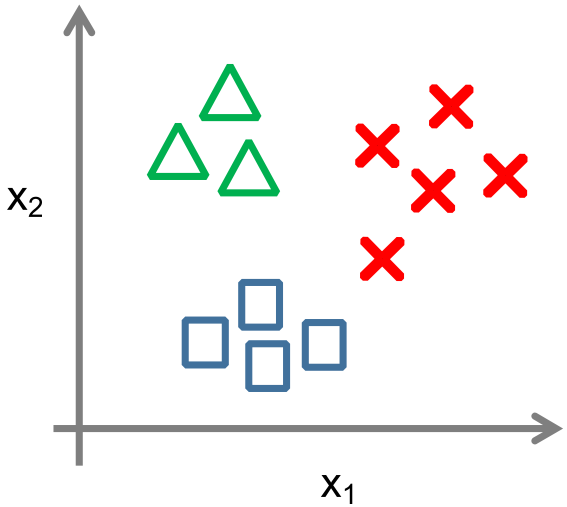
    Classifying the sample into one of the classes directly

One-vs-All

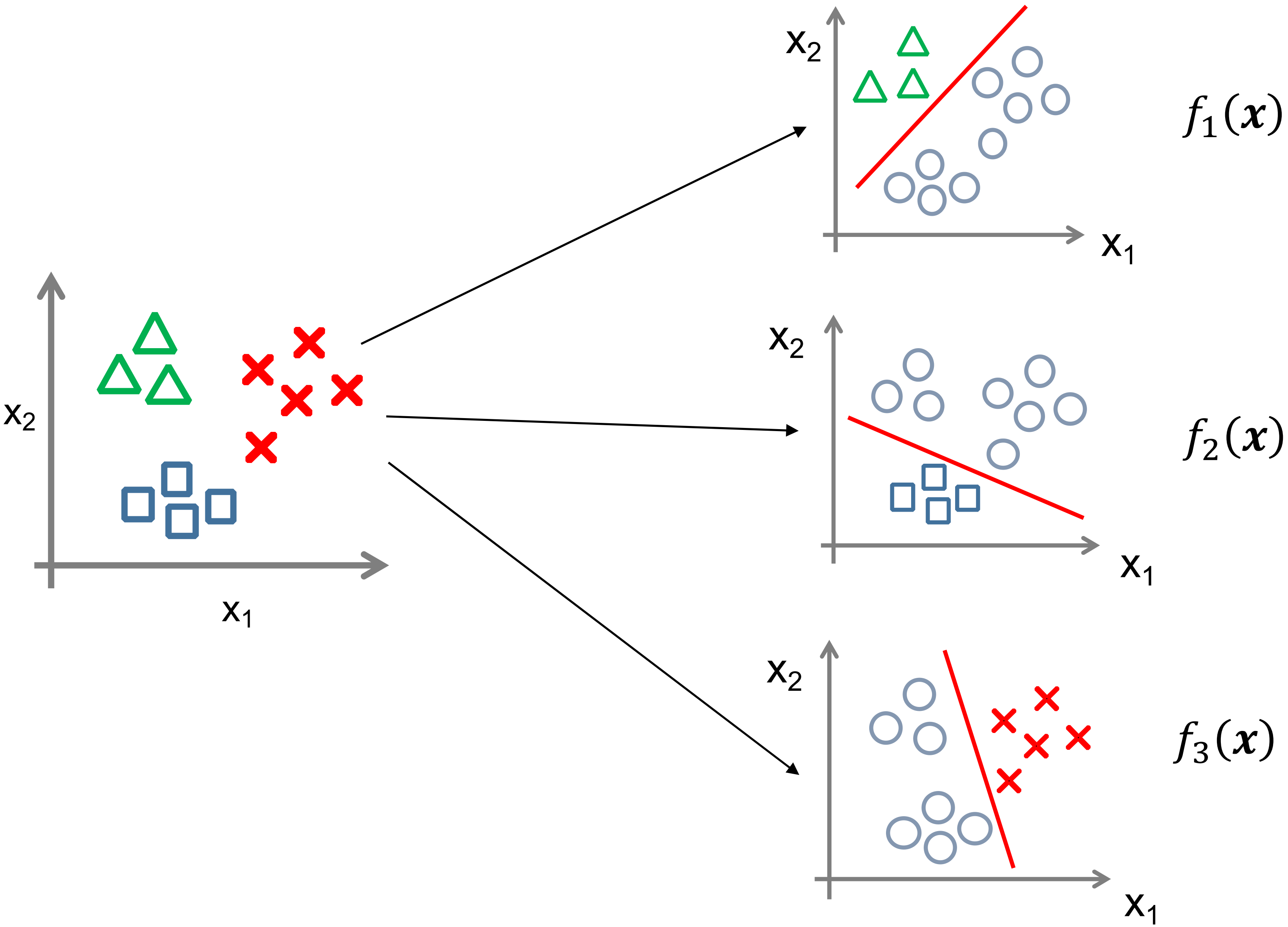
Binary classification



Multiclass classification

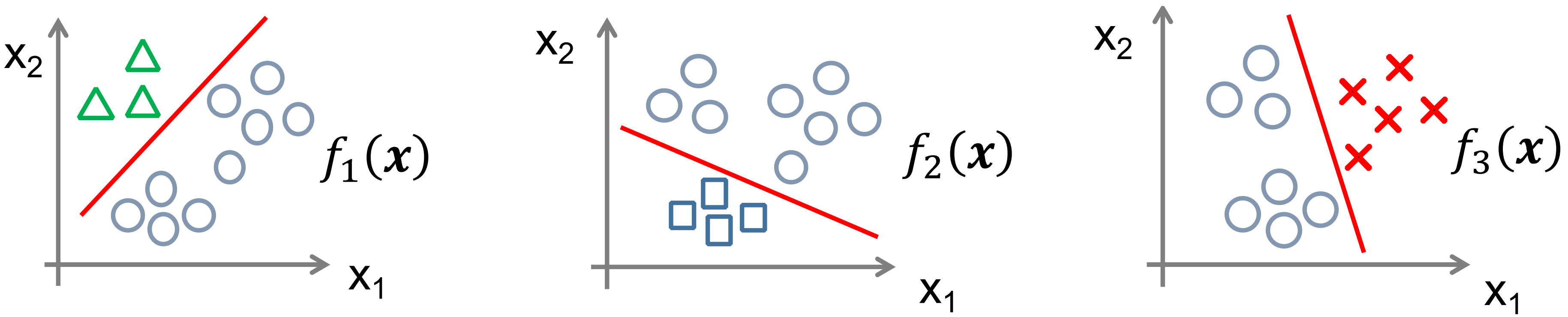


- Train a classifier for each class



- 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 $\boldsymbol{x} \in \mathbb{R}^{1\times d}$ 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 $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_K] \in \mathbb{R}^{d\times K}$

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

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

$$softmax_1(\boldsymbol{x}\boldsymbol{W}) = \frac{e^{\boldsymbol{x}\boldsymbol{w}_1}}{e^{\boldsymbol{x}\boldsymbol{w}_1}+e^{\boldsymbol{x}\boldsymbol{w}_2}} \qquad softmax_2(\boldsymbol{x}\boldsymbol{W}) = \frac{e^{\boldsymbol{x}\boldsymbol{w}_2}}{e^{\boldsymbol{x}\boldsymbol{w}_1}+e^{\boldsymbol{x}\boldsymbol{w}_2}}$$

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

  - It can be seen that

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

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

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

Cost Function

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

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

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

$$
L(\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_K) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} y_k^{(i)} \log softmax_k\left(\boldsymbol{x}^{(i)} \boldsymbol{W}\right)
$$

Cross-entropy loss

  - $y_k^{(i)}$ is the *k*-th element of $\boldsymbol{y}^{(i)} \in \mathbb{R}^{1 \times K}$

Gradient Descent

- The gradient *w.r.t.* $\boldsymbol{w}_j \in \mathbb{R}^{d\times 1}$ is

$$\frac{\partial L(\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_K)}{\partial \boldsymbol{w}_j} = \frac{1}{n}\sum_{i=1}^{n}\left(softmax_j\left(\boldsymbol{x}^{(i)}\boldsymbol{W}\right) - y_j^{(i)}\right)\boldsymbol{x}^{(i)T}$$

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

- By representing $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_K] \in \mathbb{R}^{d\times K}$, we have

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

  - $softmax\left(\boldsymbol{x}^{(i)}\boldsymbol{W}\right) = \left[softmax_1\left(\boldsymbol{x}^{(i)}\boldsymbol{W}\right), \cdots, softmax_K\left(\boldsymbol{x}^{(i)}\boldsymbol{W}\right)\right] \in \mathbb{R}^{1\times K}$ *is a row vector*
  - $\boldsymbol{x}^{(i)T} \in \mathbb{R}^{d\times 1}$ *is a column vector*

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