

# MACHINE LEARNING

## Linear Models: Logistic Regression

Last Update: 27th October 2022

Prof. Dr. Shadi Albarqouni

Director of Computational Imaging Research Lab. (Albarqouni Lab.)

**University Hospital Bonn | University of Bonn | Helmholtz Munich**



# STRUCTURE

## 1. Logistic Regression

# LOGISTIC REGRESSION

## Definition

**Logistic regression** is a widely used discriminative classification model  $p(y|\mathbf{x}; \theta)$ , where  $\mathbf{x} \in \mathbb{R}^D$  is a fixed-dimensional input vector,  $\mathbf{y} \in \{1, \dots, C\}$  is the class label, and  $\theta$  are the parameters.

if  $C = 2$ , this is known as **binary logistic regression**, and if  $C > 2$ , it is known as **multinomial logistic regression**, or alternatively, **multiclass logistic regression**.

# BINARY LOGISTIC REGRESSION

## Example: classifying Iris flowers (Code)

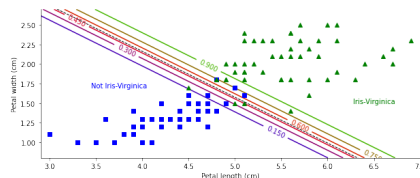
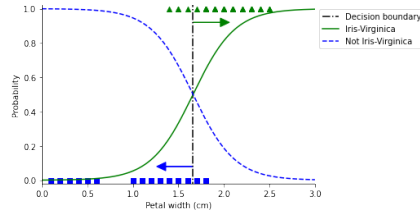
Binary Logistic Regression | Sigmoid function | Linear classifier | Objective function

Given some inputs  $x \in \mathcal{X}$  and a mapping function  $f(\cdot)$  that predict a binary variable  $y \in \{0, 1\}$ , the **conditional probability distribution**  $p(y|x; \theta) = \text{Ber}(y|f(x; \theta))$  where

$$p(y = 1|x; \theta) = f(x; \theta) \triangleq \sigma(\mathbf{w}^T \mathbf{x} + b)$$
$$\sigma(a) = \frac{1}{1+e^{-a}}$$

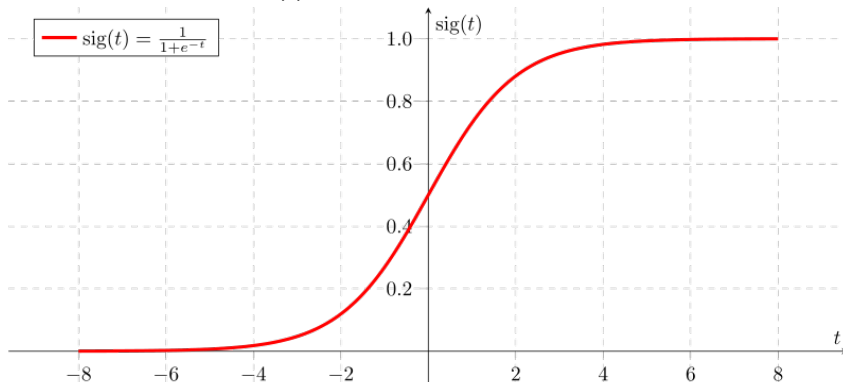
$a = \mathbf{w}^T \mathbf{x} + b$  is often called **logits** or **pre-activation**.

find  $\mathbf{w}$  and  $b$  for the given example.



## HYPOTHESIS REPRESENTATION -- SIGMOID FUNCTION

What happens when  $a \rightarrow \infty$ ?<sup>1</sup> or  $a > 0$ ?



Source: <https://commons.wikimedia.org/wiki/File:Sigmoid-function-2.svg>

---

$$^1\sigma(\cdot) \triangleq \text{sig}(\cdot), \text{ and } a \triangleq t$$

## LINEAR CLASSIFIER-- DECISION BOUNDARY

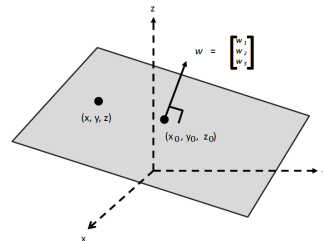
$$\begin{aligned} f(\mathbf{x}) &= \mathbb{I}(p(y=1|\mathbf{x}) > p(y=0|\mathbf{x})) \\ &= \mathbb{I}\left(\log \frac{p(y=1|\mathbf{x})}{p(y=0|\mathbf{x})} > 0\right) \\ &= \mathbb{I}(a > 0) \rightarrow \text{Perceptron} \end{aligned}$$

The inner product  $\langle \mathbf{w}, \mathbf{x} \rangle$  defines the **hyperplane** with a normal vector  $\mathbf{w}$  and offset  $b$ .

This plane  $\mathbf{w}^T \mathbf{x} + b = 0$  is often called the **decision boundary** separating the 3d space into two halves.

We call the data to be **linearly separable** if we can perfectly separate the training examples by such a **linear boundary**.

$$a = \mathbf{w}^T \mathbf{x} + b \triangleq b + \sum_{d=1}^D w_d x_d$$



More about dot products: watch this YouTube Video

# Machine Learning

## └ Logistic Regression

### └ Linear classifier– Decision boundary

#### LINEAR CLASSIFIER– DECISION BOUNDARY

$$f(\mathbf{x}) = \mathbb{I}(p(y=1|\mathbf{x}) > p(y=0|\mathbf{x})) \\ = \mathbb{I}\left(\log\frac{p(y=1|\mathbf{x})}{p(y=0|\mathbf{x})} > 0\right) \\ = \mathbb{I}(a > 0) \rightarrow \text{Perceptron}$$

The inner product  $\langle \mathbf{w}, \mathbf{x} \rangle$  defines the **hyperplane** with a normal vector  $\mathbf{w}$  and offset  $b$ . This plane  $\mathbf{w}^T \mathbf{x} + b = 0$  is often called the **decision boundary** separating the 3d space into two halves. We call the data to be **linearly separable** if we can perfectly separate the training examples by such a linear boundary.

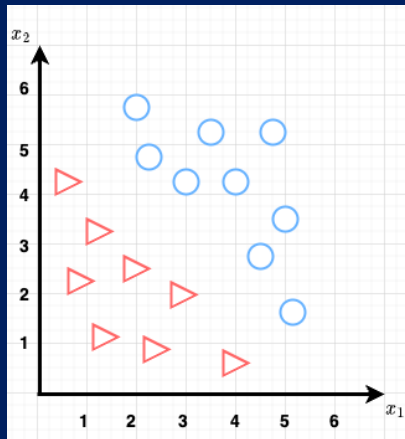


Example: Given the data points on the right hand side, what would be your optimal decision boundary to make the data linearly separable?

$$\sigma(a) = \sigma(w^T x + b)$$

$$a = b + \sum_{d=1}^D w_d x_d \triangleq b + w_1 x_1 + w_2 x_2 = 0$$

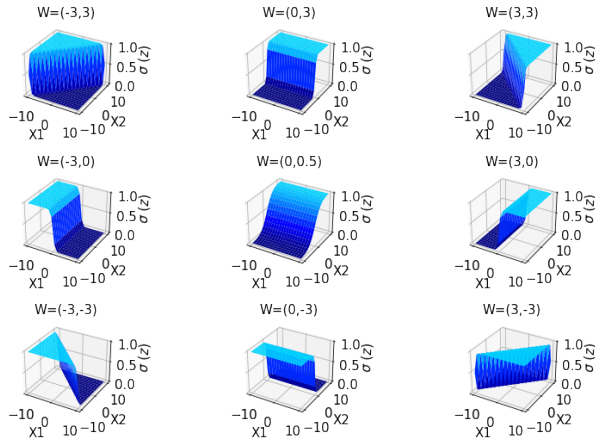
what happens if we have larger values of  $w$ ?





# LINEAR CLASSIFIER -- DECISION BOUNDARY

The vector  $\mathbf{w}$  defines the **orientation** of the decision boundary, and its magnitude,  $\|\mathbf{w}\|_2 = \sqrt{\sum_{d=1}^D w_d^2}$  controls the **steepness** of the sigmoid, and hence the **confidence** of the predictions.



Play with the code

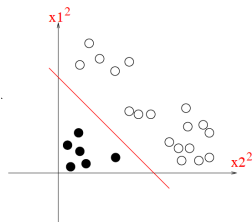
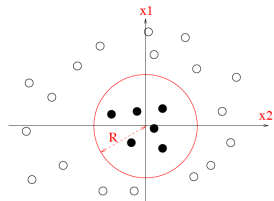
# NONLINEAR CLASSIFIER

We can often make a problem **linearly separable** by **preprocessing the inputs** in a suitable way.

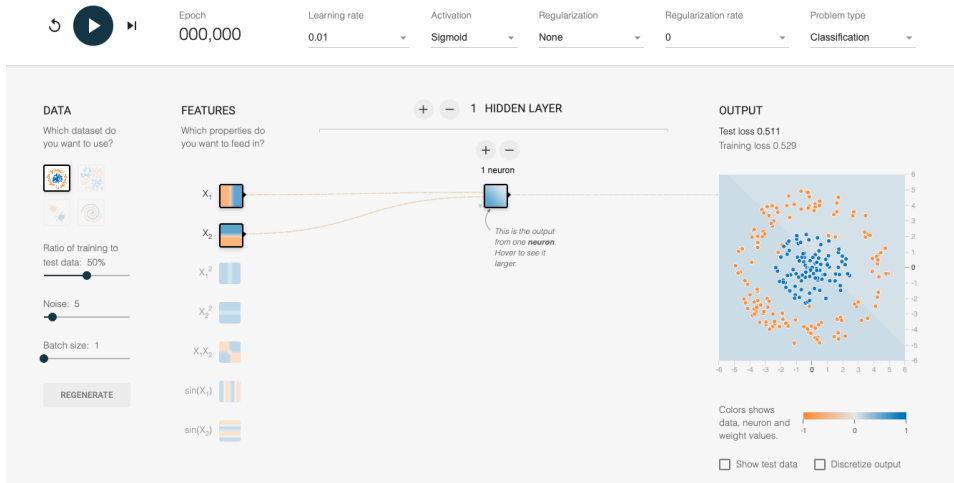
let  $\phi(x)$  be a transformed version of the input feature vector.

suppose we use  $\phi(x_1, x_2) = [1; x_1^2; x_2^2]$ , and we let  $w = [-R^2; 1; 1]$ .

$w^T \phi(x) = -R^2 + x_1^2 + x_2^2$ , so the decision boundary (where  $w^T \phi(x) = 0$ ) defines a circle with radius  $R$ .



## DEMO



# MAXIMUM LIKELIHOOD ESTIMATION (MLE)

## Maximum likelihood estimation (MLE)

It can be obtained by minimizing the **Negative Log Likelihood** as an objective function

$$\theta_{MLE} = \arg \min_{\theta} NLL(\theta)$$

The **Negative Log Likelihood (NLL)** for the binary classification is given by  $NLL(\mathbf{w}) = -\frac{1}{N} \log \prod_{n=1}^N \underbrace{\text{Ber}(y_n | f(\mathbf{x}_n; \mathbf{w}))}_{p(y_n | x_n; \theta)} \triangleq -\frac{1}{N} \log \prod_{n=1}^N \text{Ber}(y_n | \mu_n)$  where

$\mu_n = f(\mathbf{x}_n; \mathbf{w}) = \sigma(a_n)$  is the **prediction**

$a_n = \mathbf{w}^T \mathbf{x}_n = \sum_{d=0}^D w_d x_{nd}$  is the **logit**, with bias  $w_0 = b$  and  $x_0 = 1$ .

The **NLL** can be written as  $NLL(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^N y_n \log \mu_n + (1 - y_n) \log (1 - \mu_n)$

# Machine Learning

## └ Logistic Regression

## └ Maximum likelihood estimation (MLE)

### MAXIMUM LIKELIHOOD ESTIMATION (MLE)

#### Maximum likelihood estimation (MLE)

It can be obtained by minimizing the **Negative Log Likelihood** as an objective function

$$\theta_{MLE} = \underset{\theta}{\operatorname{argmin}} NLL(\theta)$$

The **Negative Log Likelihood (NLL)** for the binary classification is given by  $NLL(\mathbf{w}) = -\frac{1}{N} \log \prod_{i=1}^N \operatorname{Ber}(y_i | f(\mathbf{x}_i; \mathbf{w})) \hat{=} -\frac{1}{N} \log \prod_{i=1}^N \operatorname{Ber}(y_i | \mu_i)$  where  $f(\mathbf{x}_i; \mathbf{w}) = \mu_i$

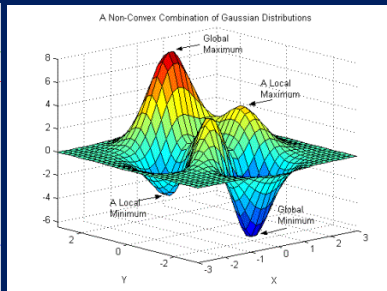
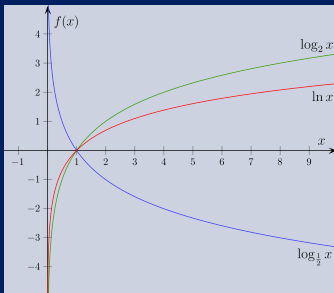
$\mu_i = f(\mathbf{x}_i; \mathbf{w}) = \sigma(a_i)$  is the **prediction**

$a_i = \mathbf{w}^T \mathbf{x}_i = \sum_{d=0}^D w_d x_{i,d}$  is the **logit**, with bias  $w_0 = b$  and  $x_0 = 1$ .

The **NLL** can be written as  $NLL(\mathbf{w}) = -\frac{1}{N} \sum_{i=1}^N y_i \log \mu_i + (1 - y_i) \log (1 - \mu_i)$

Why Negative Log Likelihood? Indeed, why we need to take the Log? and why we need to take the negative?

What about other loss functions, e.g., Mean Squared Error?



# Machine Learning

## └ Logistic Regression

### └ Maximum likelihood estimation (MLE)

## Maximum likelihood estimation (MLE)

It can be obtained by minimizing the **Negative Log Likelihood** as an objective function

$$\theta_{MLE} = \underset{\theta}{\operatorname{argmin}} NLL(\theta)$$

The **Negative Log Likelihood (NLL)** for the binary classification is given by  $NLL(\mathbf{w}) = -\frac{1}{N} \log \prod_{n=1}^N \frac{\operatorname{Ber}(y_n | f(\mathbf{x}_n; \mathbf{w}))}{p(y_n | \mu_n)}$  where

$\mu_n = f(\mathbf{x}_n; \mathbf{w}) = \sigma(a_n)$  is the **prediction**

$a_n = \mathbf{w}^T \mathbf{x}_n = \sum_{d=0}^D w_d x_{n,d}$  is the **logit**, with bias  $w_0 = b$  and  $x_{n,0} = 1$ .

The **NLL** can be written as  $NLL(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^N y_n \log \mu_n + (1 - y_n) \log (1 - \mu_n)$

Given  $\operatorname{Ber}(y|\theta) \triangleq \theta^y(1-\theta)^{1-y}$ , the  $NLL(\mathbf{w}) = -\frac{1}{N} \log \prod_{n=1}^N \operatorname{Ber}(y_n | \mu_n)$ , the objective function can be written as:

$$\begin{aligned} NLL(\mathbf{w}) &= -\frac{1}{N} \log \prod_{n=1}^N \operatorname{Ber}(y_n | \mu_n) \\ &= -\frac{1}{N} \log \prod_{n=1}^N \mu_n^{y_n} (1 - \mu_n)^{1-y_n} \\ &= -\frac{1}{N} \sum_{n=1}^N \log [\mu_n^{y_n} (1 - \mu_n)^{1-y_n}] \\ &= -\frac{1}{N} \sum_{n=1}^N \underbrace{y_n \log \mu_n + (1 - y_n) \log (1 - \mu_n)}_{\mathbb{H}_{ce}(y_n, \mu_n) \text{ is the binary cross entropy}} \end{aligned}$$

## MAXIMUM LIKELIHOOD ESTIMATION (MLE)

Given the objective function, we aim to find the MLE solution by computing the **gradient** and solving

$$g(\mathbf{w}) = \nabla_{\mathbf{w}} NLL(\mathbf{w}) = 0$$

$$\nabla_{\mathbf{w}} NLL(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N (\mu_n - y_n) \mathbf{x}_n$$

$$\nabla_{\mathbf{w}} NLL(\mathbf{w}) = \frac{1}{N} (\mathbf{1}_N^T (\text{diag}(\mu - \mathbf{y}) \mathbf{X}))^T \text{ in a matrix form}$$

To ensure the objective function is **convex**, we must prove the **hessian** is positive semi-definite;

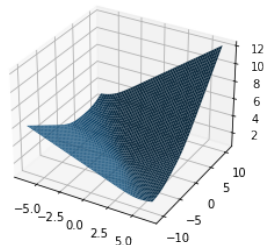
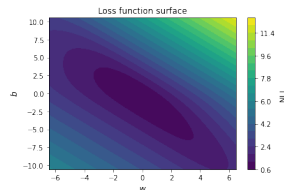
$$\mathbf{H} = \nabla_w \nabla_w NLL(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N (\mu_n (1 - \mu_n) x_n) x_n^T$$

$$\mathbf{H} = \frac{1}{N} \mathbf{X}^T \mathbf{S} \mathbf{X} \text{ in a matrix form where}$$

$$\mathbf{S} \triangleq \text{diag}(\mu_1(1 - \mu_1), \dots, \mu_N(1 - \mu_N))$$

It can be shown that for any non-zero vector,  $\mathbf{v}$ ;

$$\mathbf{v}^T \mathbf{H} \mathbf{v} = \mathbf{v}^T \mathbf{X}^T \mathbf{S} \mathbf{X} \mathbf{v} = (\mathbf{v}^T \mathbf{X}^T \mathbf{S}^{\frac{1}{2}}) (\mathbf{S}^{\frac{1}{2}} \mathbf{X} \mathbf{v}) = \|\mathbf{S}^{\frac{1}{2}} \mathbf{X} \mathbf{v}\|_2^2 > 0$$



Play with the code

## MAXIMUM LIKELIHOOD ESTIMATION (MLE)

Given the gradient  $\frac{1}{N}(1_N^T(\text{diag}(\mu - \mathbf{y})\mathbf{X}))^T$  and the hessian  $\frac{1}{N}\mathbf{X}^T\mathbf{S}\mathbf{X}$  of the objective function, one can compute the **stochastic gradient descent** (Sec. 8.4) using

first-order method:

$$\begin{aligned}\omega_{t+1} &= \omega_t - \eta_t \mathbf{g}_t \triangleq \omega_t - \eta_t \frac{1}{N} (1_N^T (\text{diag}(\mu_t - \mathbf{y}) \mathbf{X}))^T \\ &\triangleq \omega_t - \eta_t \frac{1}{N} \sum_{n=1}^N (\mu_n - y_n) \mathbf{x}_n\end{aligned}$$

**slow convergence, when gradient is small**

```
1:  $w \leftarrow 0, \eta \leftarrow 1$ 
2: repeat
3:   for  $n = 1 : N$  do
4:      $a_n \leftarrow \omega^T \mathbf{x}_n$ 
5:      $\mu_n \leftarrow \sigma(a_n)$ 
6:      $e_n \leftarrow (\mu_n - y_n)$ 
7:   end for
8:    $\mathbf{E} \leftarrow \text{diag}(e_{1:N})$ 
9:    $\omega \leftarrow \omega - \eta \frac{1}{N} \mathbf{X}^T \mathbf{E}$ 
10: until Converged
```



## MAXIMUM LIKELIHOOD ESTIMATION (MLE)

Given the gradient  $\frac{1}{N}(1_N^T(\text{diag}(\mu - \mathbf{y})\mathbf{X}))^T$  and the hessian  $\frac{1}{N}\mathbf{X}^T\mathbf{S}\mathbf{X}$  of the objective function, one can compute the **stochastic gradient descent** (Sec. 8.4) using

second-order method:

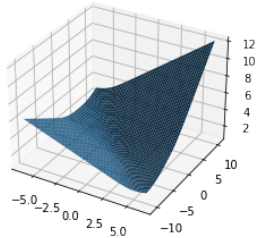
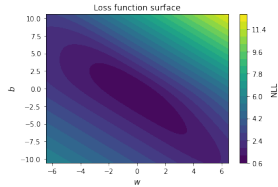
$$\omega_{t+1} = \omega_t - \eta_t \mathbf{H}_t^{-1} \mathbf{g}_t \triangleq \eta_t (\mathbf{X}^T \mathbf{S}_t \mathbf{X})^{-1} \mathbf{X}^T \mathbf{S}_t \mathbf{z}_t$$

where  $\mathbf{z}_t \triangleq \mathbf{X}\omega_t + \mathbf{S}_t^{-1}(\mathbf{y} - \mu_t)$

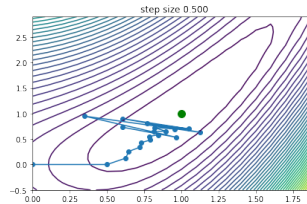
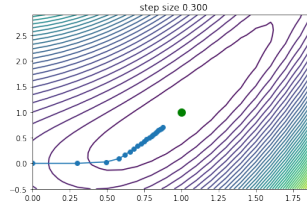
It is often called **Iteratively reweighted least squares (IRLS)**

```
1:  $w \leftarrow 0, \eta \leftarrow 1$ 
2: repeat
3:   for  $n = 1 : N$  do
4:      $a_n \leftarrow \omega^T \mathbf{x}_n$ 
5:      $\mu_n \leftarrow \sigma(a_n)$ 
6:      $s_n \leftarrow \mu_n(1 - \mu_n)$ 
7:      $z_n \leftarrow a_n + \frac{y_n - \mu_n}{s_n}$ 
8:   end for
9:    $\mathbf{S} \leftarrow \text{diag}(s_{1:N})$ 
10:   $\omega \leftarrow \eta(\mathbf{X}^T \mathbf{S} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{S} \mathbf{z}$ 
11: until Converged
```

# VISUALIZATION



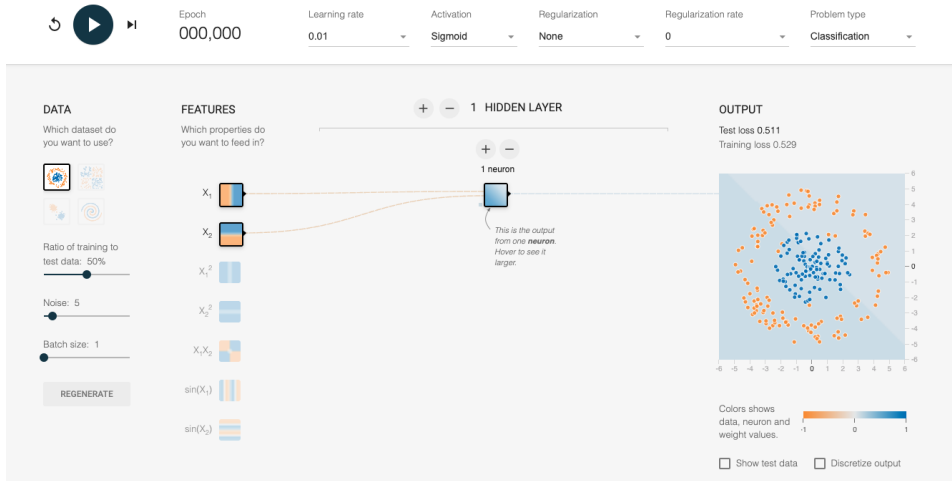
Play with the code



Play with the code

Source: <https://towardsdatascience.com/animations-of-logistic-regression-with-python-31f8c1>

## OVERFITTING



# MAXIMUM A POSTERIOR (MAP)

## Maximum A Posterior (MAP)

It can be obtained by minimizing the **Penalized Negative Log Likelihood** as an objective function

$$\theta_{MAP} = \arg \min_{\theta} NLL(\theta) + \lambda \|\theta\|_2^2$$

where  $\|\theta\|_2^2 = \sum_{d=1}^D w_d^2$  is the  $\ell_2$ -**regularization** or **weight decay** and  $\lambda$  is the **regularization rate/parameter**.

The **Penalized Negative Log Likelihood (PNLL)** is quite desirable to avoid overfitting. The gradient and hessian are given as:

$$\begin{aligned}\nabla_{\mathbf{w}} PNLL(\mathbf{w}) &= \nabla_{\mathbf{w}} NLL(\mathbf{w}) + 2\lambda \mathbf{w} \\ \nabla_{\mathbf{w}} \nabla_{\mathbf{w}} PNLL(\mathbf{w}) &= \nabla_{\mathbf{w}} \nabla_{\mathbf{w}} NLL(\mathbf{w}) + 2\lambda \mathbf{I}\end{aligned}$$

→ Standardization (Sec. 10.2.8)!

# MULTINOMINAL LOGISTIC REGRESSION

## Definition

**Multinomial logistic regression** is a discriminative classification model  $p(y|\mathbf{x}; \theta) = \text{Cat}(y|\text{softmax}(W\mathbf{x} + \mathbf{b}))$ , where  $\mathbf{x} \in \mathbb{R}^D$  is a fixed-dimensional input vector,  $y \in \{1, \dots, C\}$  is the class label with  $C > 2$ , and  $\theta = (W, \mathbf{b})$  are the parameters with  $W$  as the weight matrix of  $C \times D$ , and  $\mathbf{b}$  as the  $C$ -dimensional bias vector.

$$p(y_c|\mathbf{x}; \theta) = \frac{e^{a_c}}{\sum_{c'=1}^C e^{a_{c'}}}$$

Since  $\sum_{c=1}^C p(y_c|\mathbf{x}; \theta) = 1$ , one can ignore the weight vector  $w_C$  for the last class  $C$ , so the weight matrix  $W$  becomes of size  $(C - 1) \times D$ .

When the labels are not mutually exclusive, then an input could have multiple output, i.e., **multi-label classification**, e.g, image tagging. In this particular case,  $p(y_c|\mathbf{x}; \theta) = \prod_{c=1}^C \text{Ber}(y_c|\sigma(\mathbf{w}_c^T \mathbf{x}))$ .

## SUMMARY

	Binary logistic Regression	Multinomial logistic regression
Probability $p(y \mathbf{x}; \theta)$	$\text{Ber}(y \sigma(w^T x + b))$	$\text{Cat}(y_c \text{softmax}(w^T x + b))$
Activation function $\sigma(\cdot)$	sigmoid	softmax
Cost function $H_{ce}$	$-[y \log \mu + (1 - y) \log (1 - \mu)]$	$-\sum_{c=1}^C y_c \log \mu_c$
Gradient	–	–
Hessian	–	–

# DEMO



# Questions