

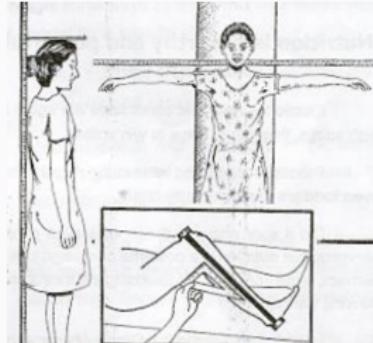
Non-linear Regression, Logistic Regression

Machine Learning 1 — Lecture 5

16th April 2023

Robert Peharz

Institute of Theoretical Computer Science
Graz University of Technology



knee height [cm]	arm span [cm]	height [cm]
50	166	171
56	172	175
52	174	168
...

- Last lecture, we discussed **regression**, i.e. predicting a continuous **target** variable y from an input vector x
- In particular, **linear regression** using an affine model:

$$\hat{y} = \mathbf{w}^T \mathbf{x} + b$$

- Least-squares loss:**

$$\mathcal{L}(\mathbf{w}, b) = \frac{1}{N} \sum_{i=1}^N (\hat{y}^{(i)} - y^{(i)})^2$$

- Collect all input vectors in the **design matrix**; include a “dummy feature” $x_0 \equiv 1$ to account for the bias b :

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}^{(1)T} \\ \mathbf{x}^{(2)T} \\ \vdots \\ \mathbf{x}^{(N)T} \end{pmatrix} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_D^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_D^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(N)} & x_2^{(N)} & \dots & x_D^{(N)} \end{pmatrix}$$

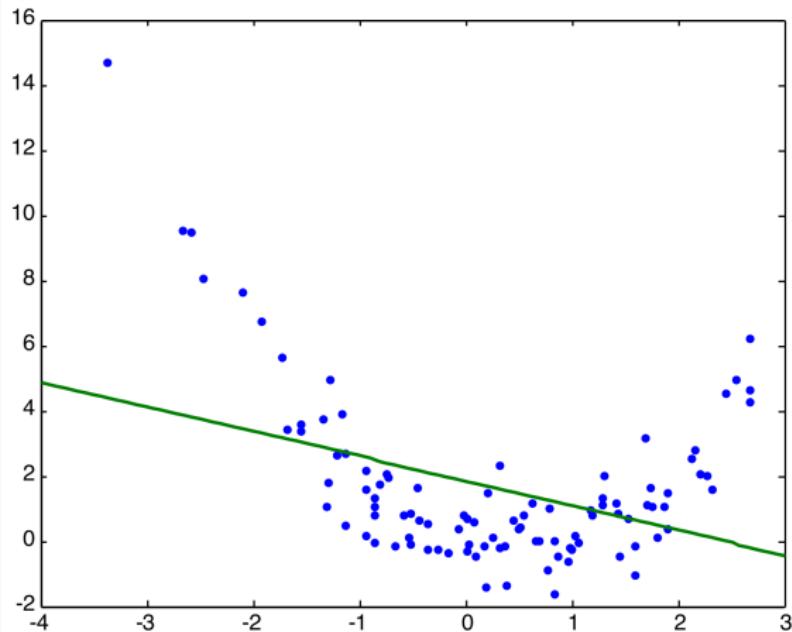
- Parameter vector $\theta = (\theta_0, \theta_1, \dots, \theta_D)^T = (b, w_1, \dots, w_D)^T$
- Least-squares solution** (minimizes $\mathcal{L}(\theta)$):

$$\theta^* = \underbrace{(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T}_{\text{Moore-Penrose Inverse}} \mathbf{y}$$

Non-linear Features

Linearity Assumption

Linear dependency is often a strong assumption:



$$\text{Linear fit: } f_{\theta^*}(x) = 2.1 - 0.76x$$

Non-Linear Features

$$\underbrace{\begin{pmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(N)} \end{pmatrix}}_{\text{prediction } \hat{y}} = \underbrace{\begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_D^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_D^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(N)} & x_2^{(N)} & \dots & x_D^{(N)} \end{pmatrix}}_{\text{design matrix } X} \underbrace{\begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_D \end{pmatrix}}_{\text{parameters } \theta}$$

Idea: We can apply any pre-processing (non-linear function) to the input features and **add them to the design matrix!**

E.g., in our body height example, we could include “creative” features like

- $x_3 := \cos(\text{knee height}^2) \times \text{arm span}^3$
- $x_4 := \exp(\text{knee height}) / \log(\text{arm span})$

Non-Linear Features cont'd

Replace the original D features x_1, x_2, \dots, x_D with K non-linear features $\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \dots, \phi_K(\mathbf{x})$.

features, basis functions: $\phi_k: \mathbb{R}^D \mapsto \mathbb{R}, 1 \leq k \leq K$

Parameter vector θ is now $(K + 1)$ -dimensional.

$$\underbrace{\begin{pmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(N)} \end{pmatrix}}_{\text{prediction } \hat{y}} = \underbrace{\begin{pmatrix} 1 & \phi_1(\mathbf{x}^{(1)}) & \dots & \phi_K(\mathbf{x}^{(1)}) \\ 1 & \phi_1(\mathbf{x}^{(2)}) & \dots & \phi_K(\mathbf{x}^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \phi_1(\mathbf{x}^{(N)}) & \dots & \phi_K(\mathbf{x}^{(N)}) \end{pmatrix}}_{\text{design matrix } \Phi} \underbrace{\begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_K \end{pmatrix}}_{\text{parameters } \theta}$$

Note that the constant “dummy” feature can also be interpreted as a basis function $\phi_0(\mathbf{x}) \equiv 1$.

Non-Linear Features cont'd

We can be creative and use **any** transformation as features, for example:

- **polynomials** $x_1^2, x_1x_2, x_2^2, \dots$
- **sinusoids** $\sin(\omega_i x_1), \cos(\omega_i x_1), \omega_i > 0$
- Localized features, like **radial basis functions** (Gaussian shaped)

$$\phi_i(\mathbf{x}) = \exp\left(-\frac{1}{\sigma^2} \|\mathbf{x} - \mathbf{c}_i\|^2\right)$$

- We can also include the original (raw) features x_1, x_2, x_3, \dots , i.e. **identity functions**

How to Learn with Non-linear Features?

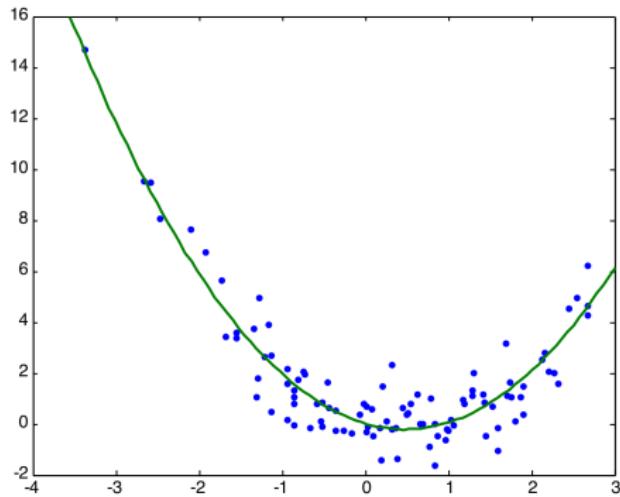
Regression with non-linear features works **exactly the same** as before. Simply replace \mathbf{X} with Φ .

$$\boldsymbol{\theta}^* = \underbrace{(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T}_{\text{Moore-Penrose Inverse}} \mathbf{y} \quad \Rightarrow \quad \boldsymbol{\theta}^* = \underbrace{(\Phi^T \Phi)^{-1} \Phi^T}_{\text{Moore-Penrose Inverse}} \mathbf{y}$$

If the features (columns of Φ) are linearly independent, $\Phi^T \Phi$ is invertible and the least-squares solution is unique.

Non-linear (Quadratic) Fit

Example



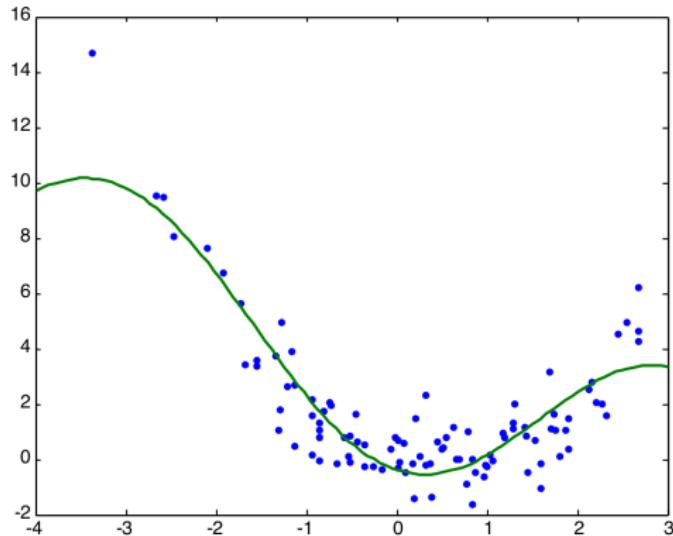
$$f_{\theta^*}(x) = 0.02 \cdot \underbrace{1}_{\phi_0} + 0.95 \cdot \underbrace{x}_{\phi_1} + 0.99 \cdot \underbrace{x^2}_{\phi_2}$$

The model is **linear in ϕ** but **non-linear in x !**

linear model + non-linear features = **non-linear model!**

Non-linear (Sinusoidal) Fit

Example



$$f_{\theta^*}(x) = 3.12 \cdot \underbrace{1}_{\phi_0} - 1.07 \cdot \underbrace{x}_{\phi_1} - 3.5 \cdot \underbrace{\cos(x)}_{\phi_2}$$

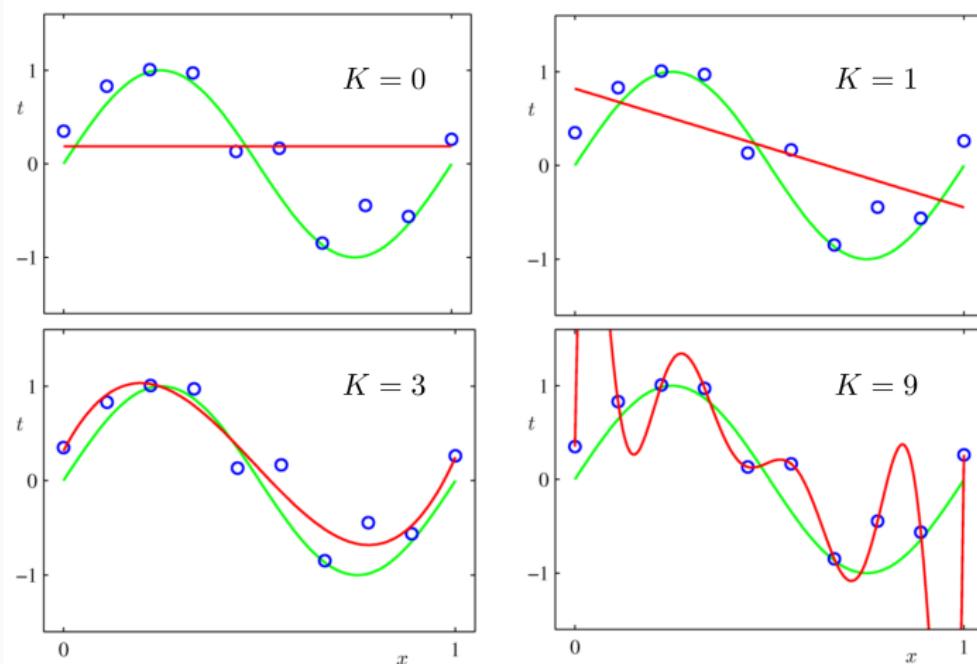
linear model + non-linear features = **non-linear model!**

Polynomial regression: Basis functions are all powers of up to order K . For example, for $D = 2$ and $K = 3$:

$$\begin{aligned} f_{\theta}(x_1, x_2) = & \theta_0 1 + && \text{constant} \\ & \theta_1 x_1 + \theta_2 x_2 + && \text{linear} \\ & \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2 && \text{quadratic} \\ & \theta_6 x_1^3 + \theta_7 x_1^2 x_2 + \theta_8 x_1 x_2^2 + \theta_9 x_2^3 && \text{cubic} \end{aligned}$$

Polynomial Regression

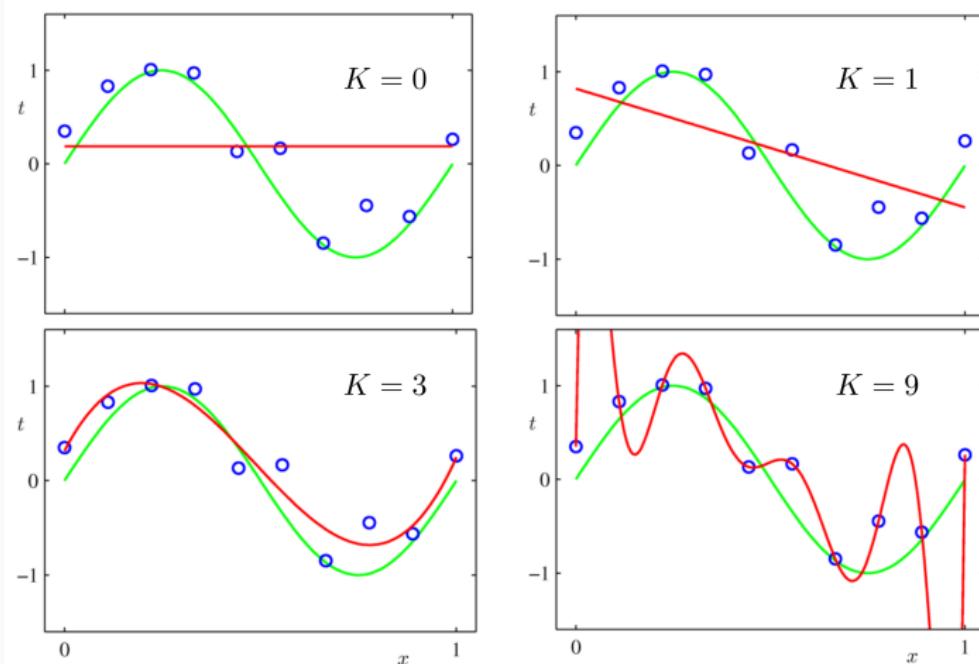
Example



What is going on with $K = 9$?

Polynomial Regression

Example



What is going on with $K = 9$?

⇒ To be discussed in a future lecture.

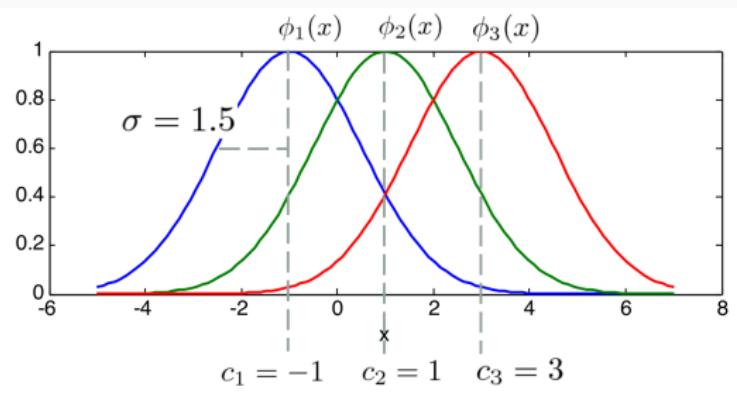
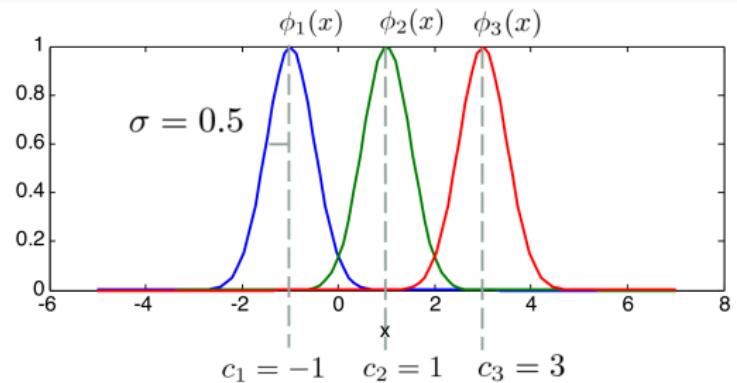
Radial basis functions (RBFs) are another prominent type of non-linear basis functions. They have the shape of the Gaussian bell curve and are parametrized by a **center vector** \mathbf{c}_i (i is the index of the feature). The width of the RBF is determined by a parameter σ .

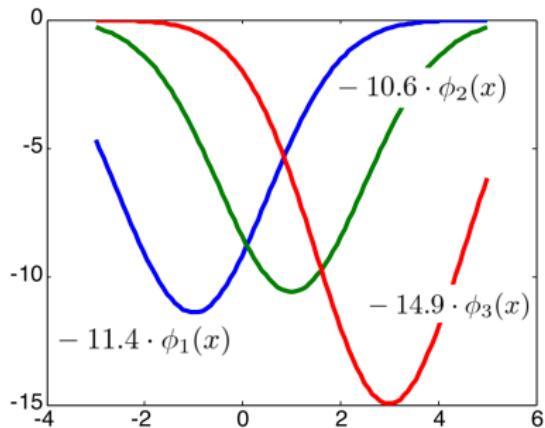
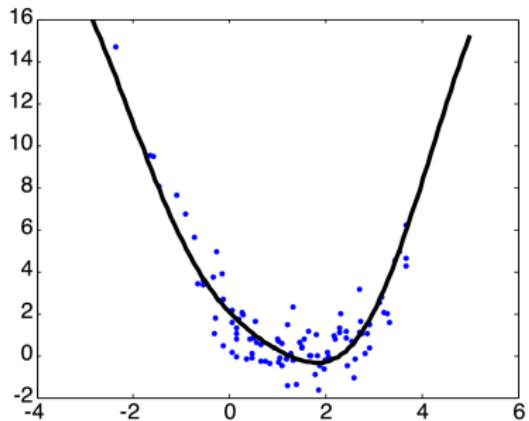
$$\phi_i(\mathbf{x}) = \exp\left(-\frac{1}{\sigma^2}\|\mathbf{x} - \mathbf{c}_i\|^2\right)$$

RBFs quickly approach 0 when moving away from the center vector – thus, they are so-called **local basis functions**.

Radial Basis Functions cont'd

Definition





Using RBFs with $\sigma = 1.5$:

$$\begin{aligned}f_{\theta}(x) &= \theta_0 \cdot 1 + \theta_1 \cdot \phi_1(x) + \theta_2 \cdot \phi_2(x) + \theta_3 \cdot \phi_3(x) \\&= 21.7 \cdot 1 - 11.4 \cdot \phi_1(x) - 10.6 \cdot \phi_2(x) - 14.9 \cdot \phi_3(x)\end{aligned}$$

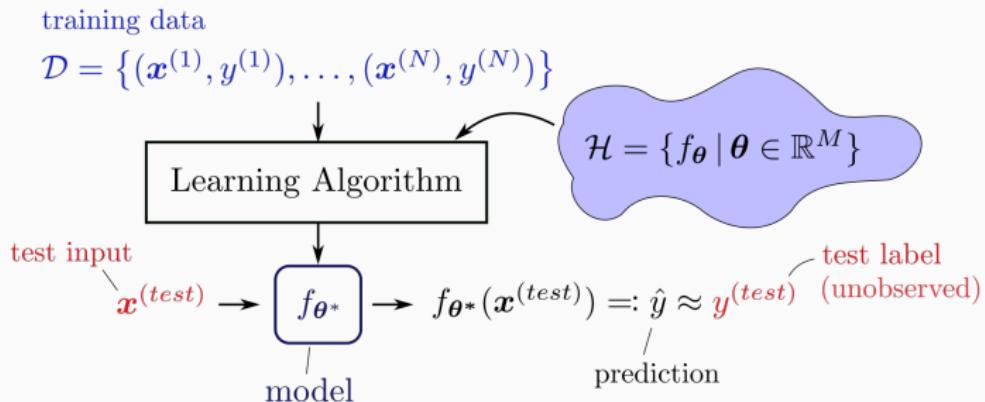
Classification, Logistic Regression

- **Regression:** Predict a **continuous** target y
- **Classification:** Predict a **categorical** target y (**label**, **class value**)
- Labels are **discrete** and assumed to have **no natural order**:
 - $\mathcal{C} = \{\text{'dog'}, \text{'cat'}, \text{'rabbit'}\}$
 - $\mathcal{C} = \{\text{'ice cream'}, \text{'chocolate'}, \text{'piece of cake'}, \text{'lollipop'}\}$
 - $\mathcal{C} = \{\text{'pear'}, \text{'apple'}\}$
- **Logistic regression:** a modification of linear regression for the classification problem
- *The* most widely used classification algorithm
- Confusing name: It's called logistic **regression**, even though it is a classification model 

Classification: The General Setup

- The true label y is a function $f: \mathbb{R}^D \mapsto \mathcal{C}$, where \mathcal{C} is the set of possible labels (e.g. $\mathcal{C} = \{\text{'dog'}, \text{'cat'}, \text{'rabbit'}\}$)
- f is sometimes called the **classification concept**
- We have a **training set** of **labelled examples** $(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(N)}, y^{(N)})$

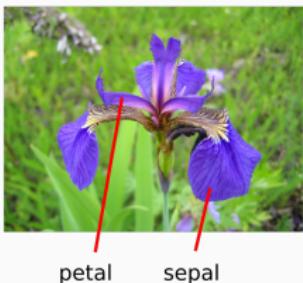
Similar picture as for regression (**supervised** learning):



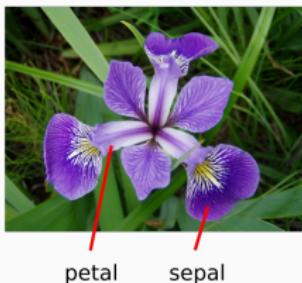
The Iris Dataset

Example

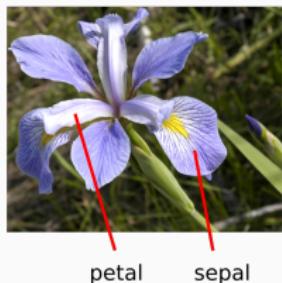
Iris Setosa



Iris Versicolor



Iris Virginica



The **Iris dataset** is a classical dataset collected by R. Fisher in the 1930's. It contains $50 \times 3 = 150$ samples of three types of iris flowers.

"sepallength", "sepalwidth", "petallength", "petalwidth", "class"

5.1, 3.5, 1.4, 0.2, Iris-setosa

4.9, 3.0, 1.4, 0.2, Iris-setosa

...

7.0, 3.2, 4.7, 1.4, Iris-versicolor

6.4, 3.2, 4.5, 1.5, Iris-versicolor

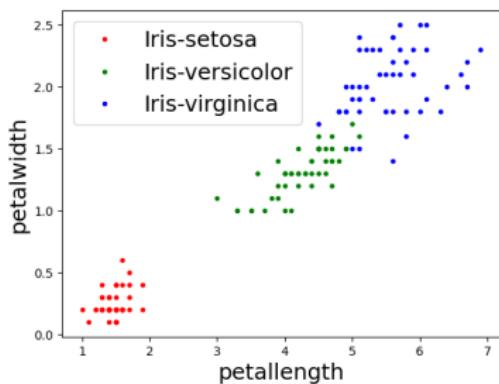
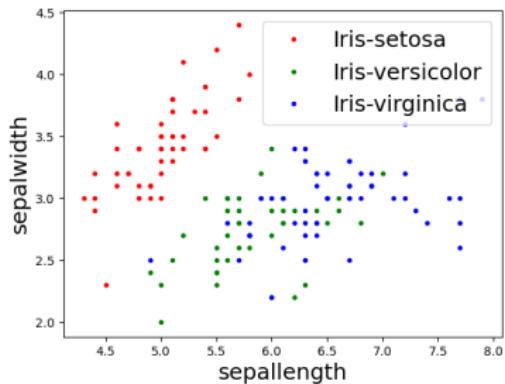
...

6.3, 3.3, 6.0, 2.5, Iris-virginica

5.8, 2.7, 5.1, 1.9, Iris-virginica

The Iris Dataset

Example



How can we apply linear functions to the classification problem?

We focus first on the case of **binary classification**, i.e. $|\mathcal{C}| = 2$.

We might encode the labels as $y \in \{-1, +1\}$. For example:

$$\mathcal{C} = \{\text{'dog'}, \text{'cat'}\}$$

'dog': $y = -1$ **negative class**

'cat': $y = +1$ **positive class**

As **classifier** we use a linear function:

$$f(\phi) = \theta^T \phi$$

where $\phi = (1, \phi_1(\mathbf{x}), \dots, \phi_K(\mathbf{x}))$ is a feature vector.

Classification rule given by **half spaces**:

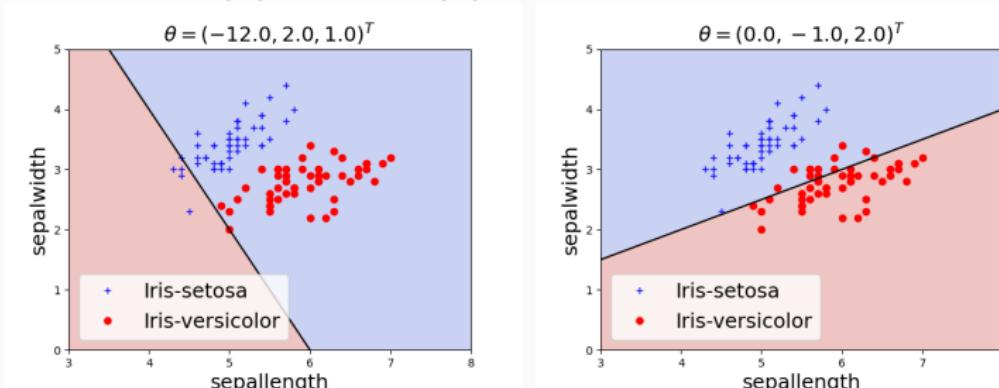
$$\theta^T \phi < 0 \Rightarrow \text{predict } \hat{y} = -1$$

$$\theta^T \phi \geq 0 \Rightarrow \text{predict } \hat{y} = +1$$

In short:

$$\hat{y} = \text{sign} (\theta^T \phi)$$

Two features $\phi_1(\mathbf{x}) = x_1$, $\phi_2(\mathbf{x}) = x_2$ and two classes from “Iris”:



- Linear classifiers yield a **linear decision boundary** (hyper-plane) in the **feature space** ϕ
- If ϕ includes non-linear features, the decision boundary becomes non-linear in the original feature space \mathbf{x} !
- A dataset is called **linearly separable** if there exists a linear classifier (hyper-plane) with 0 classification errors

How to learn θ ?

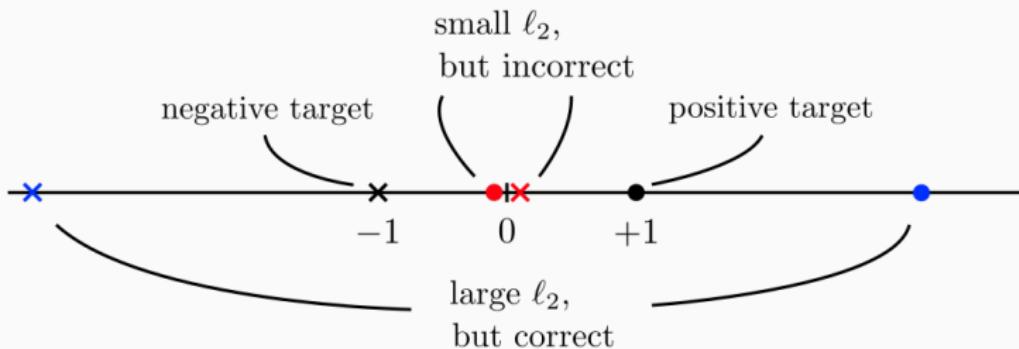
Idea: We could treat the problem as a regression problem (interpreting the labels $y = \pm 1$ as real numbers) and optimize squared loss. As we saw, squared loss is convex and optimization is easy.

But is this a good idea?



Squared Loss vs. Correct Classification

ℓ_2 -error in label space and correct classification are only loosely related:



Treating classification as a regression problem will work to a certain extend. But a loss functions reflecting the classification problem will deliver better results.

Minimizing Classification Error?

The most direct way seems to minimize the number of classification errors, or equivalently, the **classification error rate**:

$$\min_{\theta} CER = \frac{1}{N} \sum_{i=1}^N \mathbb{1} [\hat{y}^{(i)} \neq y^{(i)}]$$

where $\mathbb{1}[\cdot]$ is the **indicator function**: $\mathbb{1}[arg] = \begin{cases} 0 & \text{if } arg = \text{False} \\ 1 & \text{if } arg = \text{True} \end{cases}$

Equivalently, maximize the number of correctly classified examples, or the **classification accuracy**:

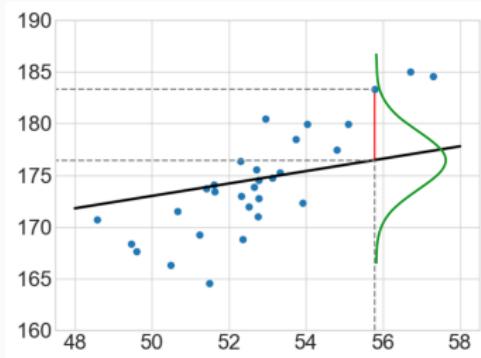
$$\max_{\theta} ACC = \frac{1}{N} \sum_{i=1}^N \mathbb{1} [\hat{y}^{(i)} = y^{(i)}]$$



Good idea, but non-convex and hard to optimize...

Recall that least-squares regression is equivalent to **maximum likelihood** under a **conditional Gaussian model**:

$$\theta^* = \arg \max_{\theta} \sum_{i=1}^N \log p(y^{(i)} | \phi^{(i)}, \theta)$$



For classification, we could simply take a different distribution, reflecting that $y^{(i)}$ are binary. 🤔

Y is a RV with state space $\mathcal{Y} = \{-1, 1\}$ and PMF

$$p_Y(y; \pi) = \begin{cases} 1 - \pi & \text{if } y = -1 \\ \pi & \text{if } y = 1 \end{cases}$$

Bernoulli distribution with success probability π .



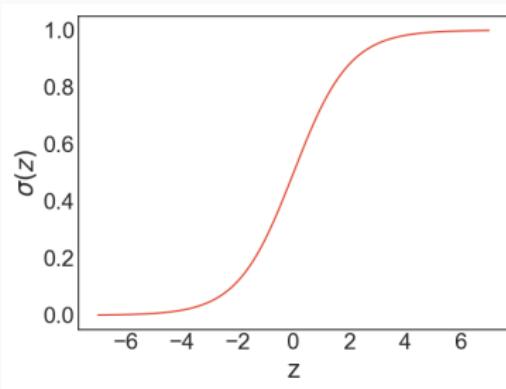
Jakob Bernoulli, 1654–1705

- We want to make the Bernoulli distribution conditional on input ϕ
- That is, the success probability π should be a function of $\theta^T \phi$
- Note that we require $0 \leq \pi \leq 1$

The **logistic function (sigmoid function)** is defined as

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

It maps from the real numbers to the interval $[0, 1]$, thus it is also sometimes called **squashing function**:



The name **logistic regression** stems from this function.

- The output for the i^{th} sample is $\theta^T \phi^{(i)}$
- The outputs are often called **logits**
- We use the logistic function to convert logits into Bernoulli parameters (success probabilities):

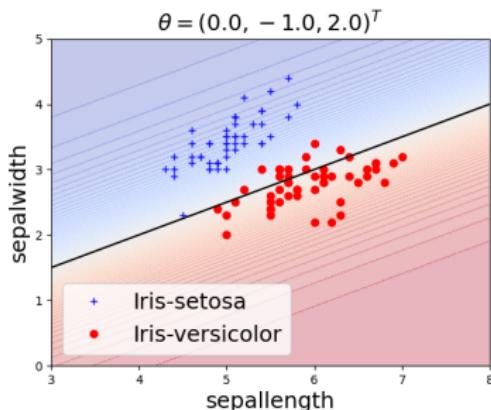
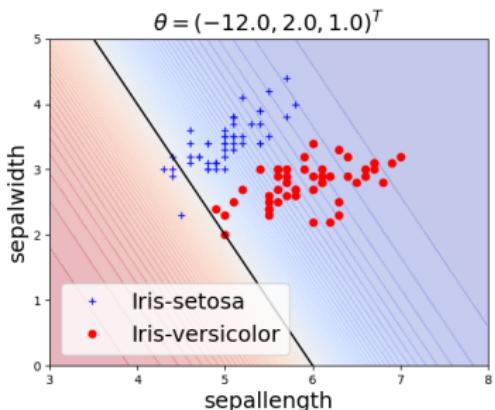
$$\pi^{(i)} := p_{\theta}(y^{(i)} = 1) = \frac{1}{1 + \exp(-\theta^T \phi^{(i)})}$$

- $\pi^{(i)}$ the **model's probability (belief)** that the input sample belongs to the **positive class**
- For **negative class**: $p_{\theta}(y^{(i)} = -1) = 1 - \pi^{(i)}$

i	$\theta^T \phi^{(i)}$	$\pi^{(i)}$	$1 - \pi^{(i)}$
1	3	0.953	0.047
2	1.5	0.818	0.182
3	-2	0.12	0.88
\vdots	\vdots	\vdots	\vdots

Model Probabilities

Example



blue shade where $\pi = p(y = 1) = \sigma(\boldsymbol{\theta}^T \phi)$ is high

red shade where $\pi = p(y = 1) = \sigma(\boldsymbol{\theta}^T \phi)$ is low (or $1 - \pi$ is high)

Maximum Likelihood Objective

The log-likelihood (under i.i.d. assumption) is

$$\begin{aligned}\log p(\mathcal{D}; \theta) &= \sum_{i=1}^N \log p(y^{(i)} | \phi^{(i)}, \theta) \\&= \sum_{i=1}^N \mathbb{1}[y^{(i)} = 1] \log \pi^{(i)} + \mathbb{1}[y^{(i)} = -1] \log(1 - \pi^{(i)}) \\&= \sum_{i=1}^N \mathbb{1}[y^{(i)} = 1] \log \sigma(\theta^T \phi^{(i)}) + \mathbb{1}[y^{(i)} = -1] \log(1 - \sigma(\theta^T \phi^{(i)}))\end{aligned}$$

We might also write this as

$$\log p(\mathcal{D}; \theta) = \underbrace{\sum_{i: y^{(i)} = 1} \log \sigma(\theta^T \phi^{(i)})}_{\text{sum over positive samples}} + \underbrace{\sum_{i: y^{(i)} = -1} \log(1 - \sigma(\theta^T \phi^{(i)}))}_{\text{sum over negative samples}}$$

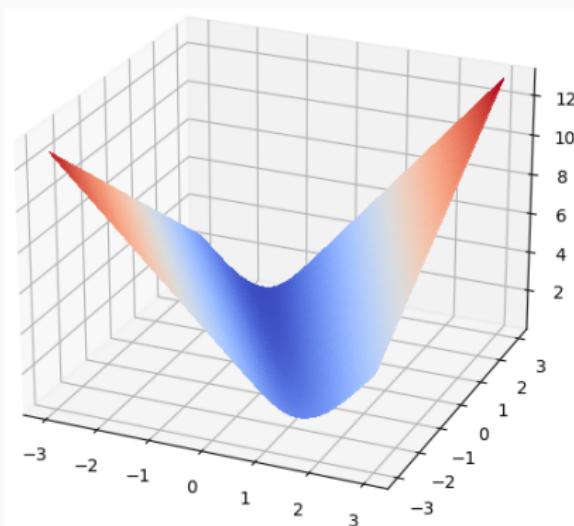
Cross-Entropy

- Usually, one performs two cosmetic modifications of the log-likelihood objective:
 - multiply times -1 and replace maximization by minimization
 - divide by N
- The resulting **equivalent** objective is called **cross-entropy (CE)** loss:

$$\begin{aligned}\mathcal{L}_{CE}(\theta) = & -\frac{1}{N} \sum_{i=1}^N \mathbf{1}[y^{(i)}=1] \log \sigma(\theta^T \phi^{(i)}) \\ & + \mathbf{1}[y^{(i)}=-1] \log(1 - \sigma(\theta^T \phi^{(i)}))\end{aligned}$$

- \mathcal{L}_{CE} is the most widely used objective for binary classification
- In general, cross-entropy compares two probability distributions—here the label is interpreted as an “extreme” distribution, assigning all probability to a single class

- No closed form solution for \mathcal{L}_{CE}
- But, \mathcal{L}_{CE} is differentiable and convex, thus we can use gradient descent to find a **global minimum**
- For example: cross-entropy loss for two input features and two classes of the Iris dataset:



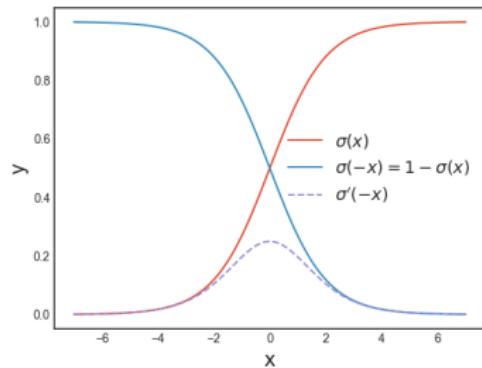
Gradient Descent for Logistic Regression

- We are going to derive the gradient of the cross-entropy loss for the logistic regression model, i.e. the vector of all partial derivatives $\frac{\partial \mathcal{L}_{CE}}{\partial \theta_i}$
- We will use the **chain rule of calculus**, since the loss is of the form

$$\mathcal{L}_{CE} = \text{crossentropy}(\text{logistic function}(\theta^T x))$$

- To this end, we first need the derivative of the logistic function $\sigma(\cdot)$

Derivative of Sigmoid Function



$$\begin{aligned}\sigma'(z) &= \left(\frac{1}{1 + e^{-z}} \right)' = - (1 + e^{-z})^{-2} (-e^{-z}) \\&= \frac{e^{-z}}{(1 + e^{-z})^2} \\&= \frac{1}{1 + e^{-z}} \frac{e^{-z}}{1 + e^{-z}} \\&= \sigma(z) \frac{1}{e^z + 1} \\&= \underline{\sigma(z) \sigma(-z)} \\&= \underline{\sigma(z) (1 - \sigma(z))}\end{aligned}$$

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

Gradient of Cross Entropy Loss

$$\mathcal{L}_{CE} = -\frac{1}{N} \left[\sum_{i: y^{(i)}=1} \log \sigma(\boldsymbol{\theta}^T \boldsymbol{\phi}^{(i)}) + \sum_{i: y^{(i)}=-1} \log (1 - \sigma(\boldsymbol{\theta}^T \boldsymbol{\phi}^{(i)})) \right]$$

$$\frac{\partial \mathcal{L}_{CE}}{\partial \theta_k} = -\frac{1}{N} \left[\sum_{i: y^{(i)}=1} \frac{\partial}{\partial \theta_k} \log \sigma(\boldsymbol{\theta}^T \boldsymbol{\phi}^{(i)}) + \sum_{i: y^{(i)}=-1} \frac{\partial}{\partial \theta_k} \log (1 - \sigma(\boldsymbol{\theta}^T \boldsymbol{\phi}^{(i)})) \right]$$

$$\begin{aligned} \frac{\partial}{\partial \theta_k} \log \sigma(\boldsymbol{\theta}^T \boldsymbol{\phi}^{(i)}) &= \overbrace{\frac{1}{\sigma(\boldsymbol{\theta}^T \boldsymbol{\phi}^{(i)})}}^{\log'} \overbrace{\sigma'(\boldsymbol{\theta}^T \boldsymbol{\phi}^{(i)})}^{\text{prediction error}} \overbrace{(1 - \sigma(\boldsymbol{\theta}^T \boldsymbol{\phi}^{(i)}))}^{\sigma'} \phi_k^{(i)} \\ &= \overbrace{(1 - \sigma(\boldsymbol{\theta}^T \boldsymbol{\phi}^{(i)}))}^{\text{prediction error}} \phi_k^{(i)} \end{aligned}$$

Gradient of Cross Entropy Loss cont'd

$$\begin{aligned}\frac{\partial}{\partial \theta_k} \log \left(1 - \sigma(\boldsymbol{\theta}^T \phi^{(i)})\right) &= \overbrace{\frac{1}{1 - \sigma(\boldsymbol{\theta}^T \phi^{(i)})}}^{\text{log}'} \overbrace{-\sigma(\boldsymbol{\theta}^T \phi^{(i)}) \left(1 - \sigma(\boldsymbol{\theta}^T \phi^{(i)})\right)}^{-\sigma'} \phi_k^{(i)} \\ &= \underbrace{\left(0 - \sigma(\boldsymbol{\theta}^T \phi^{(i)})\right)}_{\text{prediction error}} \phi_k^{(i)}\end{aligned}$$

Collecting all partial derivatives gives the sample-wise the gradient:

- positive: $\nabla_{\boldsymbol{\theta}} \log \sigma(\boldsymbol{\theta}^T \phi^{(i)}) = (1 - \sigma(\boldsymbol{\theta}^T \phi^{(i)})) \phi^{(i)}$
- negative: $\nabla_{\boldsymbol{\theta}} \log (1 - \sigma(\boldsymbol{\theta}^T \phi^{(i)})) = (0 - \sigma(\boldsymbol{\theta}^T \phi^{(i)})) \phi^{(i)}$

Gradient Descent for Cross Entropy

The gradient of the overall loss is given as:

$$\nabla_{\theta} \mathcal{L}_{CE} = -\frac{1}{N} \left[\sum_{i: y^{(i)}=1} \left(1 - \sigma \left(\theta^T \phi^{(i)} \right) \right) \phi^{(i)} + \sum_{i: y^{(i)}=-1} -\sigma \left(\theta^T \phi^{(i)} \right) \phi^{(i)} \right]$$

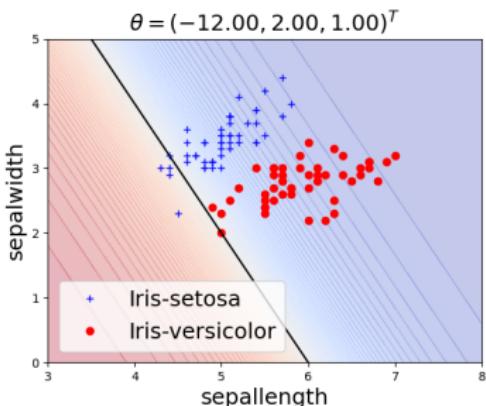
- Initialize θ arbitrarily
- Repeat until convergence

$$\theta \leftarrow \theta - \eta \nabla_{\theta} \mathcal{L}_{CE}$$

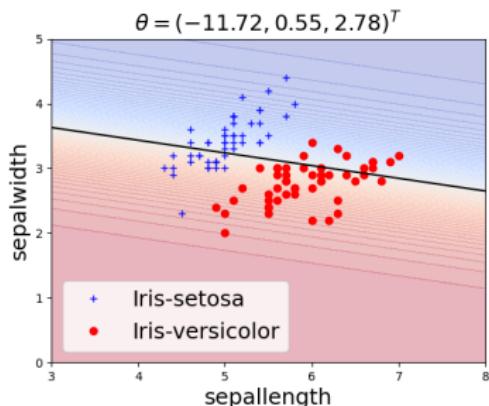
Minimizing Cross-Entropy: Iris Dataset

Example

- using 2 features and 2 classes
- gradient descent with step-size $\eta = 0.1$



Iteration 0

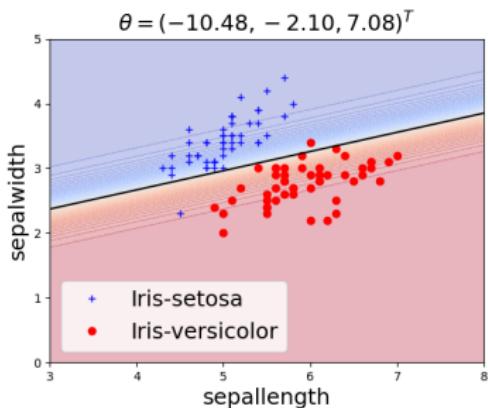


Iteration 100

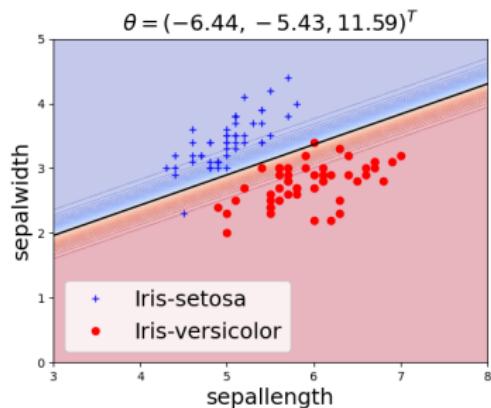
Minimizing Cross-Entropy: Iris Dataset

Example

- using 2 features and 2 classes
- gradient descent with step-size $\eta = 0.1$



Iteration 1000

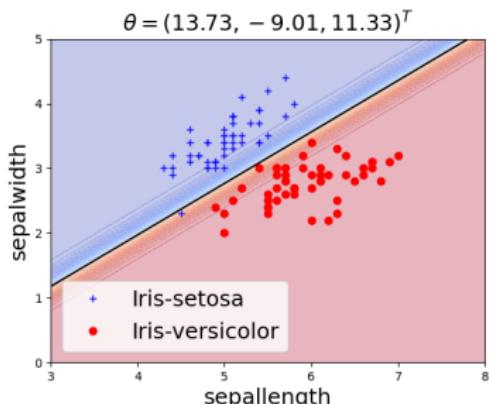


Iteration 10000

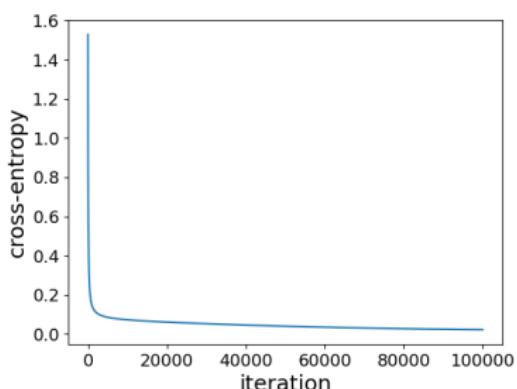
Minimizing Cross-Entropy: Iris Dataset

Example

- using 2 features and 2 classes
- gradient descent with step-size $\eta = 0.1$



Iteration 100000



cross-entropy over iteration

- Linear models with non-linear features give non-linear models
- For linear regression, the closed form solution is the same – simply replace design matrix \mathbf{X} with a design matrix Φ including non-linear features
- **Logistic regression**: linear model for classification
- Linear **decision boundary** in feature space ϕ (non-linear in \mathbf{x} if $\phi(\mathbf{x})$ is non-linear)
- **Cross-entropy loss**: equivalent to maximum likelihood for conditional Bernoulli, using the **logistic function** to convert **logits** into Bernoulli parameters
- No closed-form solution for logistic regression, so we used gradient descent