Machine Learning – Linear Models

Artificial Intelligence for Digital Health (AID)

M.Sc. in Digital Health – University of Pisa

Davide Bacciu (davide.bacciu@unipi.it)

Lecture Outline

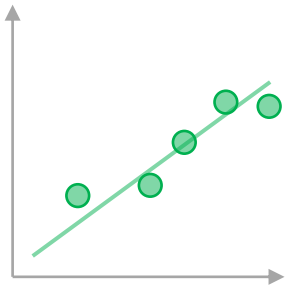
- Linear regression models
  - Formalization and interpretation
  - Training and closed form solutions
  - Regularization
- Logistic regression models
  - Binary classification
  - Training and gradient descent
- Towards neural networks

Basic Supervised (Linear) Models

Supervised Learning Learn $y = f(x)$ mapping

Continuous y

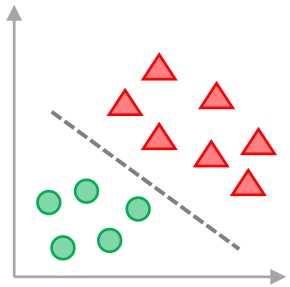
Y = 13.45, 73, 9.5, ...



Linear Regression

Categorical y

Y = *red*, *green*, *blue*, ...



Logistic Regression

Linear Regression

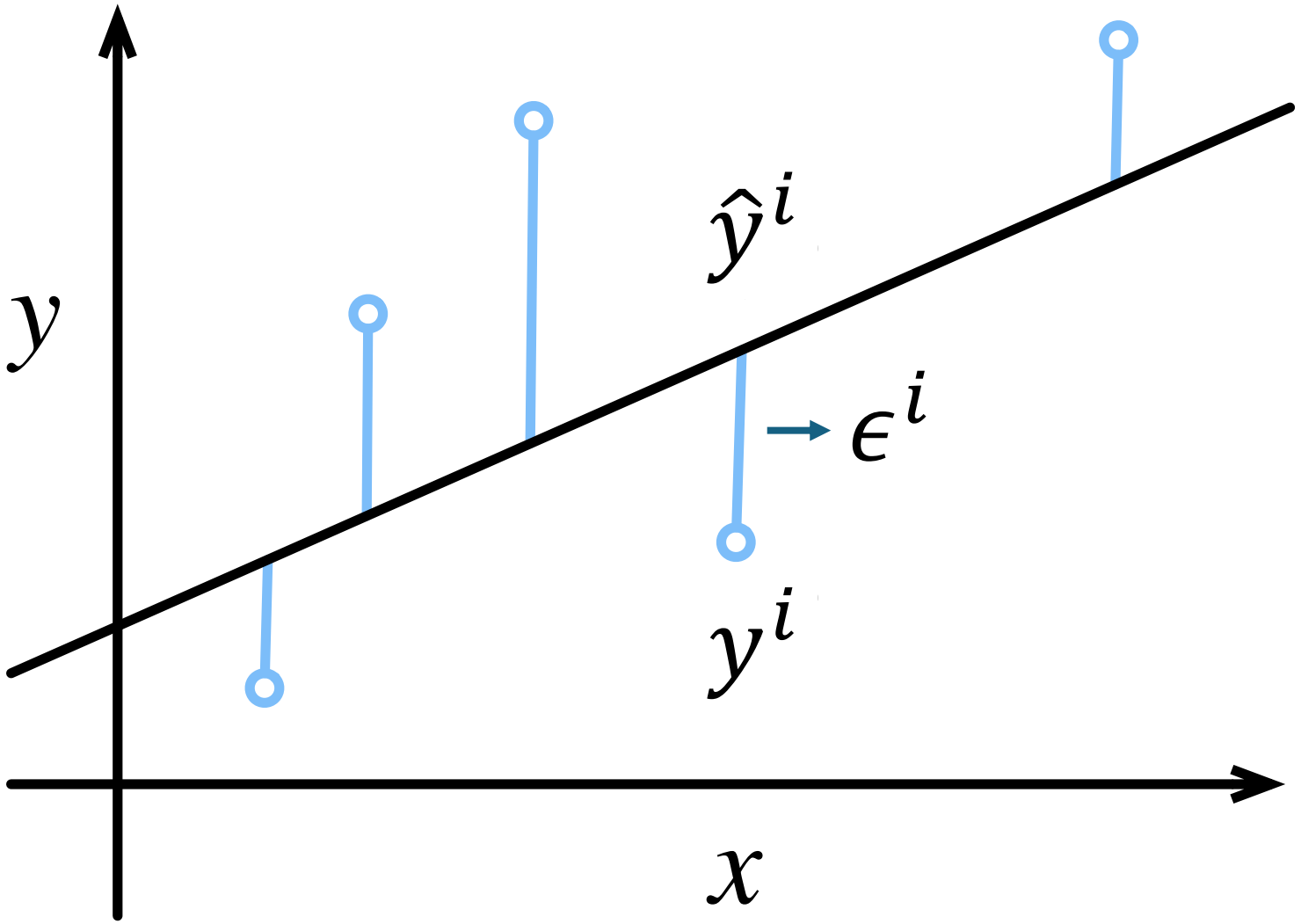
Simple Linear Regression Setting

- Given a collection of samples $D = \{(\boldsymbol{x}^1, y^1);\ \dots\ (\boldsymbol{x}^N, y^N)\}$ learn the unknown mapping $y = f(\boldsymbol{x})$ using a model h_θ
  - One or more input/free variables: i.e. $\boldsymbol{x}^n = [x_1^n, \dots, x_k^n, \dots, x_D^n] \in \mathbb{R}^D$
  - One output/response variable y
- The simplest possible h_θ for the job (high inductive bias) assumes input and output variables to be bound by a linear relationship

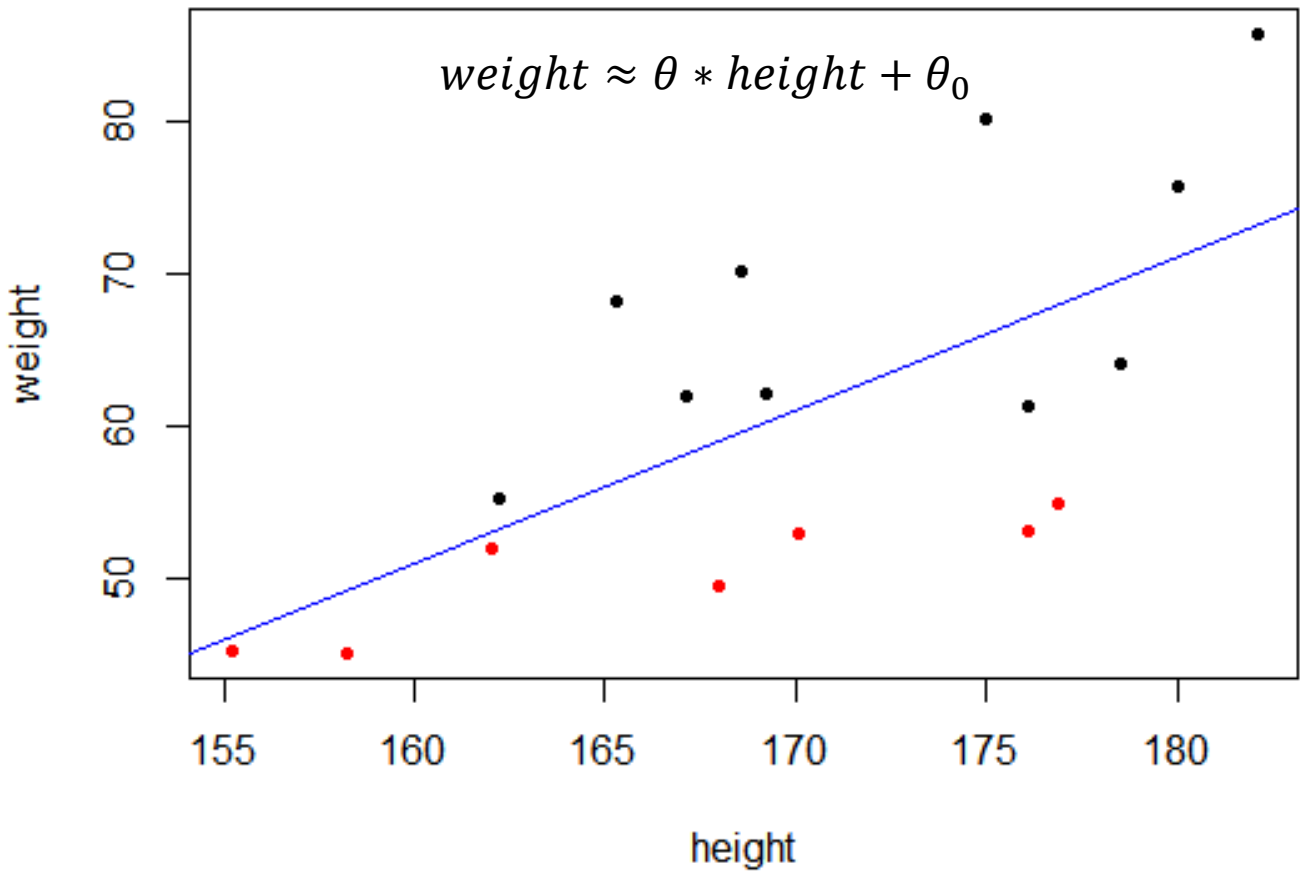
$$h_\theta \rightarrow y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_D x_D + \varepsilon = \sum_{k=1}^{D} \theta_k x_k + \epsilon$$

- Model parameters: $\boldsymbol{\theta} = [\theta_0, \theta_1, \dots, \theta_k]$ with bias θ_0 (the parameter corresponding to an input fixed to 1)
- (The mystical) Error term: $\epsilon \sim \mathcal{N}(0, \sigma^2)$ (a.k.a. Normally distributed with 0 mean and fixed variance σ^2)

Understanding Linear Regression



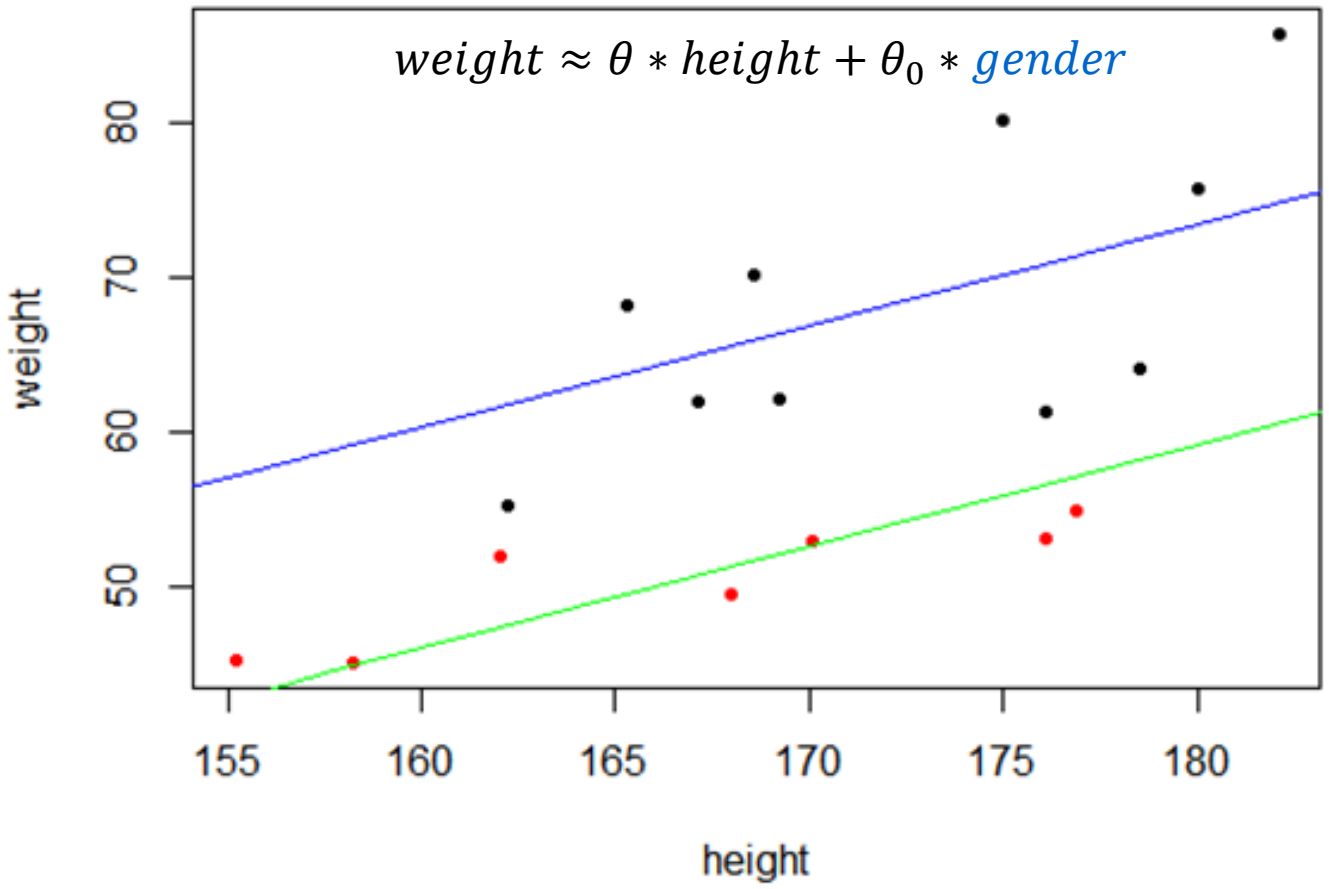
Matching conditions



Fitting a linear regressor without considering the influence of gender

• $= male$
• $= female$

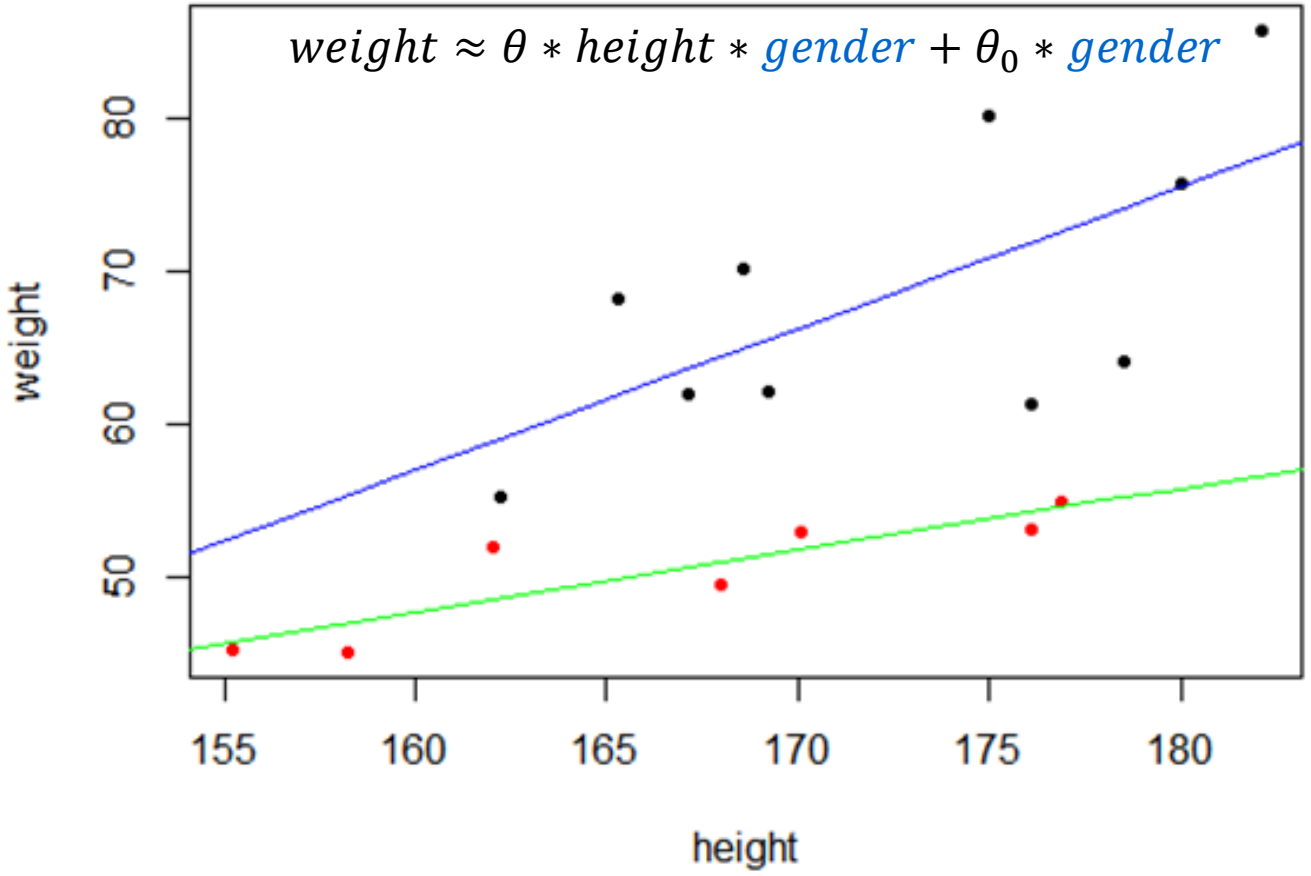
Matching conditions



A first approach would add gender as a dummy variable which determines the intercept of the linear regression (slope is not affected)

• $= male$
• $= female$

Matching conditions



Make both intercept and slope dependent on the categorical variable describing the condition to be matched

• $= male$
• $= female$

Vectorized Linear Regression

We can collate all inputs and responses of the dataset into matrices/vectors to obtain a more compact formulation

$$\boldsymbol{X} = \begin{bmatrix} x_1^1 & \dots & x_k^1 \\ \vdots & \ddots & \vdots \\ x_1^N & \dots & x_k^N \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} y^1 \\ \vdots \\ y^N \end{bmatrix} \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_k \end{bmatrix}$$

Vectorized formulation of the linear regression on dataset D

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\theta} + \boldsymbol{\epsilon}$$

with ϵ now being a column vector of normally distributed values

$$\boldsymbol{\epsilon} = [\epsilon^1, \dots, \epsilon^N]^T \text{ and } \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2)$$

One more step of vectorization

If we are predicting M response variables, we a have system of linear equations

$$\begin{cases} y_1 = \sum_{k=1}^{D} \theta_k^1 x_k + \epsilon_1 \\ \vdots \\ y_M = \sum_{k=1}^{D} \theta_k^M x_k + \epsilon_M \end{cases} \quad \text{with } \boldsymbol{\theta} = \begin{bmatrix} \theta_1^1 & \dots & \theta_1^M \\ \vdots & \ddots & \vdots \\ \theta_k^1 & \dots & \theta_k^M \end{bmatrix}$$

Which is vectorized for the whole dataset becomes

$$\boldsymbol{Y} = \boldsymbol{X\theta} + \boldsymbol{\epsilon}$$

$$\boldsymbol{Y} = \begin{bmatrix} y_1^1 & \dots & y_M^1 \\ \vdots & \ddots & \vdots \\ y_1^N & \dots & y_M^N \end{bmatrix}$$

where ϵ is a matrix of errors and $\boldsymbol{Y}$ a matrix of responses

Training a linear regression model

- Assume a single response variable and define a loss for the model
- Mean squared error (MSE)

$$E(h_\theta|D) = \frac{1}{N}\sum_{(\boldsymbol{x}^i,y^i)} L\left(h_\theta\left(\boldsymbol{x}^i\right), y^i\right) = \frac{1}{N}\sum_{(\boldsymbol{x}^i,y^i)} \left(y^i - \hat{y}^i\right)^2$$

- Measures the difference between predicted $\hat{y}^i = \boldsymbol{x}^i\boldsymbol{\theta}$ and actual y^i values (the error ϵ^i)
- Learning amounts to finding the minimum of the error function with respect to model parameters

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{N}\sum_{(\boldsymbol{x}^i,y^i)} \left(y^i - \boldsymbol{x}^i\boldsymbol{\theta}\right)^2$$

Ordinary Least Squares (LS) Solution

For a linear model with quadratic loss there exist a closed form solution for the error minimization problem

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{N} \sum_{(\boldsymbol{x}^i, y^i)} \left(y^i - \boldsymbol{x}^i \boldsymbol{\theta}\right)^2 = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{N} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^{\mathbf{2}}$$

Minimize squared regression error by taking the derivative

$$\frac{\partial(\boldsymbol{\epsilon})^2}{\partial\theta} = \frac{\partial(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^2}{\partial\theta} = \cdots = -2(\boldsymbol{X}^T\boldsymbol{y}) + 2(\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta}) = 0$$

Yields the ordinary least square solution

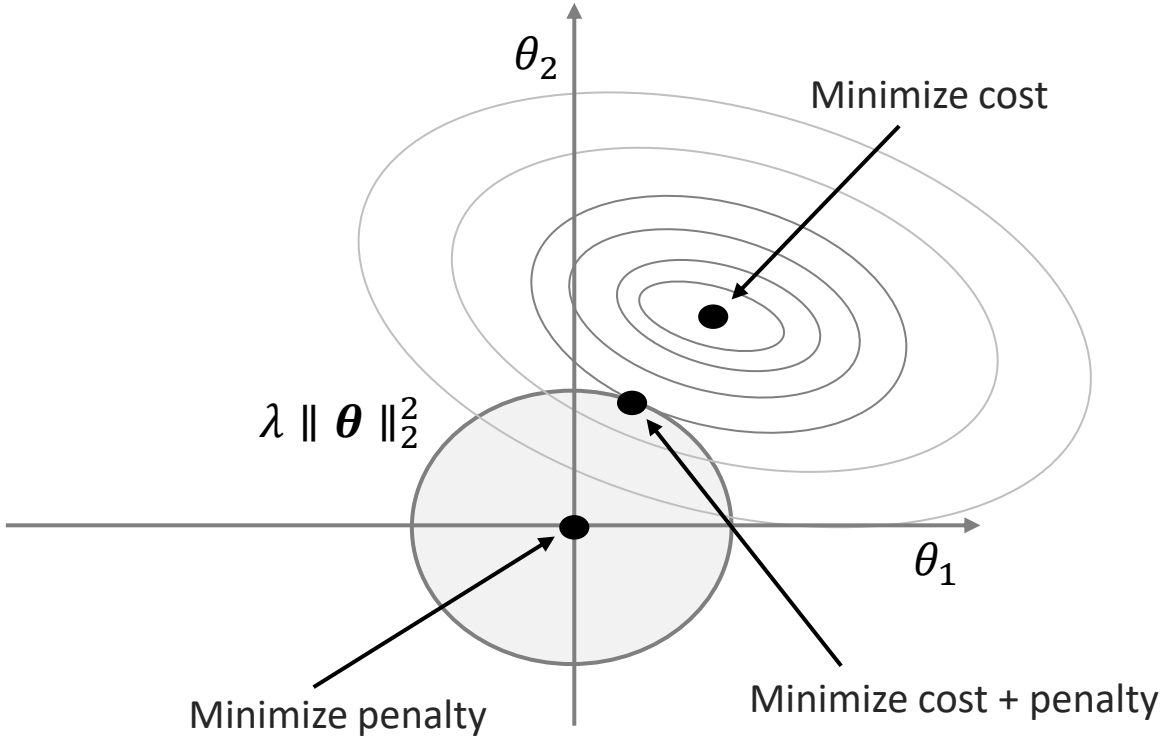
$$\boldsymbol{\theta} = [\underbrace{(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T}_{\text{Pseudo-inverse}}]\boldsymbol{y}$$

What can possibly go wrong?

- The objective in machine learning is to find those parameters that allow generalizing predictions to unseen data (avoiding overfitting)
- To achieve this, we need to control model complexity
- Regularized linear regression
  - L2 Regularization (Ridge)
  - L1 Regularization (Lasso)
  - Elastic Net (Combination of L1 & L2)

Ridge Regression (L2)

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{N} \sum_{(\boldsymbol{x}^i, y^i)} \left(y^i - \boldsymbol{x}^i \boldsymbol{\theta}\right)^2 + \lambda \parallel \boldsymbol{\theta} \parallel_2^2$$



- Adds squared weight penalties

$$\parallel \boldsymbol{\theta} \parallel_2^2 = \sum_k (\theta_k)^2$$

- Helps when data contains correlated features

Least Square Solution to Ridge Regression

A modified version of the ordinary LS

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{N} \sum_{(\boldsymbol{x}^i, y^i)} \left(y^i - \boldsymbol{x}^i \boldsymbol{\theta}\right)^2 + \lambda \parallel \boldsymbol{\theta} \parallel_2^2$$

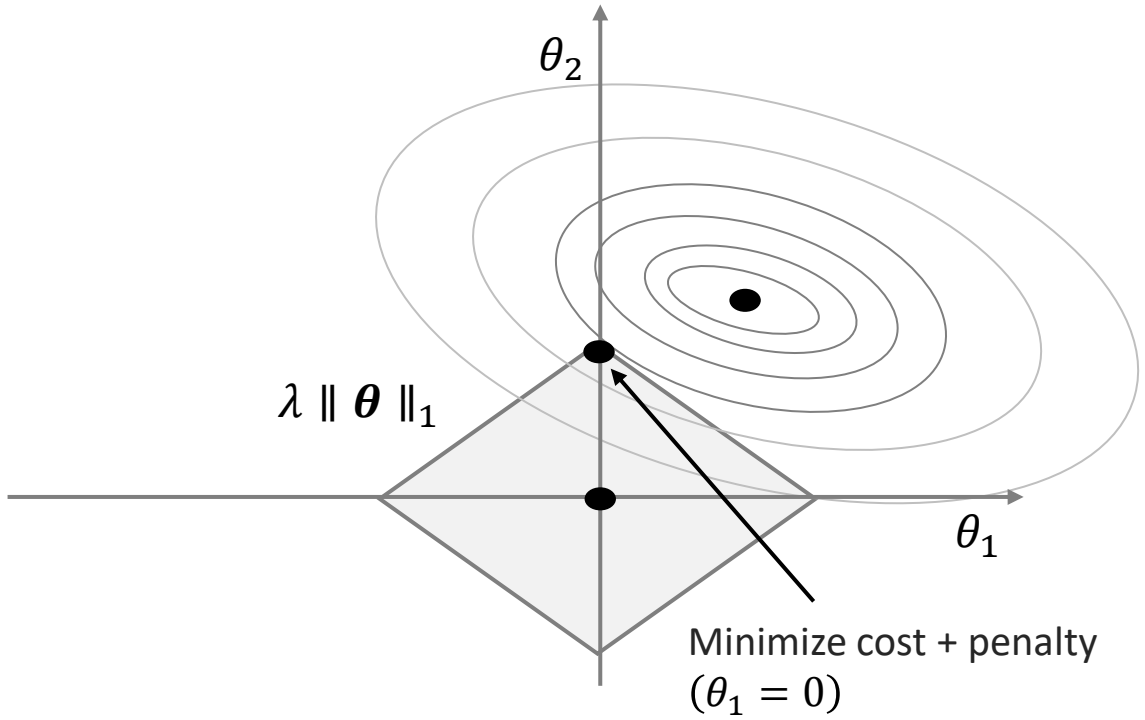
Yields to a slightly different solution

$$\boldsymbol{\theta} = [(\boldsymbol{X}^T \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^T] \boldsymbol{y}$$

Stabilizes numerically the solution by adding some λ weight to the diagonal of the (moment) matrix to be inverted

Lasso Regression (L1)

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{N} \sum_{(\boldsymbol{x}^i, y^i)} \left(y^i - \boldsymbol{x}^i \boldsymbol{\theta}\right)^2 + \lambda \parallel \boldsymbol{\theta} \parallel_1$$



- Adds absolute value penalties

$$\parallel \boldsymbol{\theta} \parallel_1 = \sum_k |\theta_k|$$

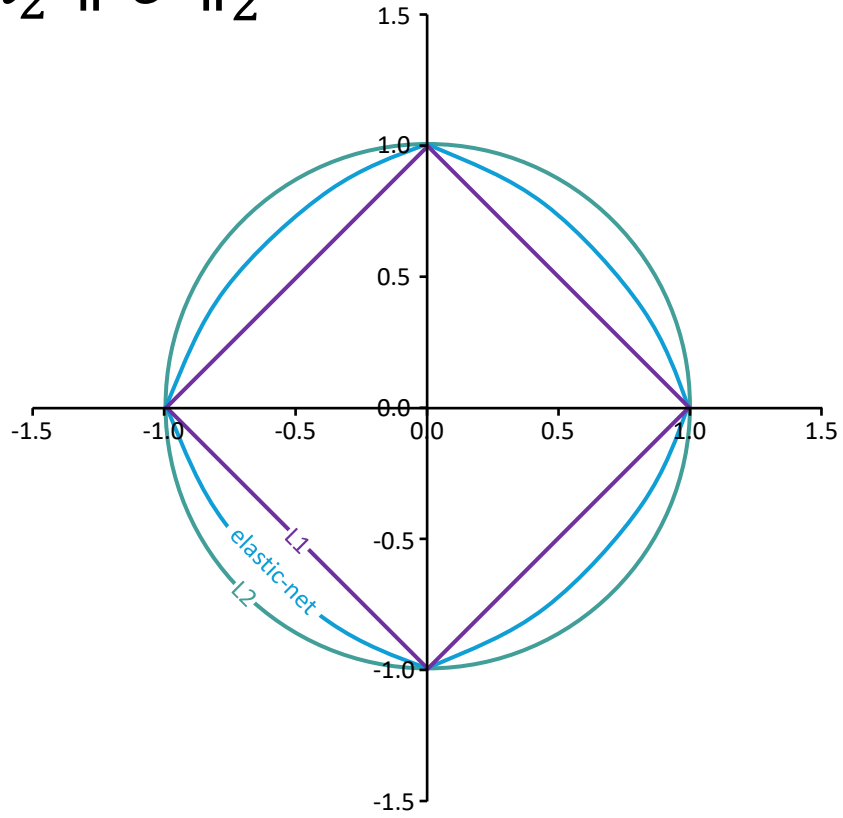
- Encourages sparsity
- Useful for feature selection in biomedical datasets

ElasticNet – Best of both worlds

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{N} \sum_{(\boldsymbol{x}^i, y^i)} \left(y^i - \boldsymbol{x}^i \boldsymbol{\theta}\right)^2 + \lambda_1 \parallel \boldsymbol{\theta} \parallel_1 + \lambda_2 \parallel \boldsymbol{\theta} \parallel_2^2$$

Elastic-net applies both the L2 norm and L1 norm at the same time, so the constraint is somewhere in the middle.

It reduces larger weights while making unimportant weight to 0



Alternative Loss Functions

- We can compute the loss using the absolute value yielding to the Mean Absolute Error (MAE)

$$E(h_\theta|D) = \frac{1}{N}\sum_{(\boldsymbol{x}^i,y^i)} L\left(h_\theta\left(\boldsymbol{x}^i\right), y^i\right) = \frac{1}{N}\sum_{(\boldsymbol{x}^i,y^i)} \left|y^i - \hat{y}^i\right|$$

MSE
- Differentiable and closed form solutions
- Penalizes larger errors more heavily due to squaring
- Sensitive to outliers

MAE
- Not (everywhere) differentiable and no closed form solutions
- Treats all errors equally
- Less sensitive to outliers

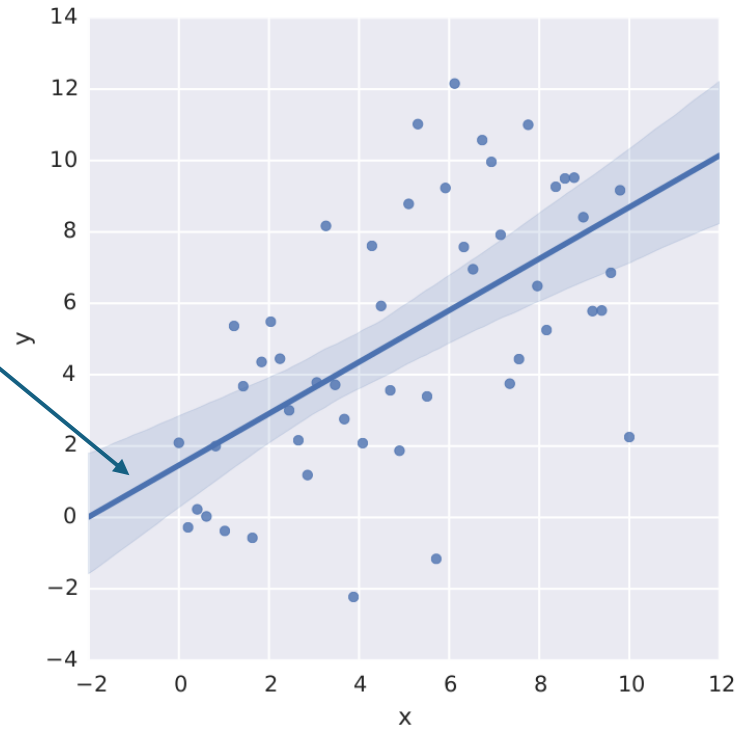
More on Regression Error Metrics

- $MSE = \frac{\sum_{i=1}^{N}(y_i - \widehat{y_i})^2}{N}$
- $MAE = \frac{\sum_{i=1}^{N}|y_i - \widehat{y_i}|}{N}$
- $RMSE = \sqrt{MSE}$ → Root Mean Squared Error
- $MAPE = \frac{100}{N} \times \sum_{i=1}^{N} \left| \frac{y_i - \widehat{y_i}}{y_i} \right|$ → Mean Absolute Precision Error

Confidence Intervals on Errors

Confidence intervals can be straightforwardly estimated for simple (1D) linear regression

$$CI = \hat{y} \pm z \cdot \overline{err}$$



Confidence Intervals on Errors

Confidence intervals can be straightforwardly estimated for simple (1D) linear regression

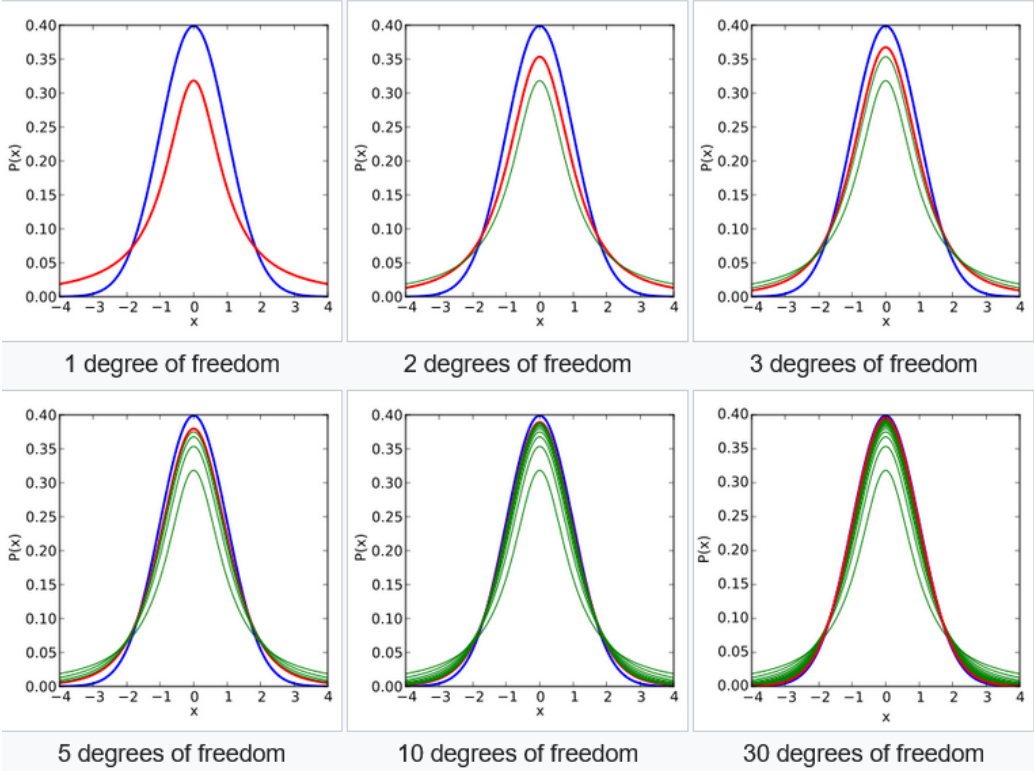
$$CI = \hat{y} \pm z \cdot \overline{err}$$

- z is the critical value corresponding to the expected confidence level α (e.g. $\alpha = 95\%$) assuming a Student distribution with N-2 degrees of freedom

$$z = t_{1-\alpha,N-2}$$

Student Distribution

Simply put: the generalization of a Normal distribution



One-sided	75%	80%	85%	90%	95%	97.5%	99%	99.5%	99.75%	99.9%	99.95%
Two-sided	50%	60%	70%	80%	90%	95%	98%	99%	99.5%	99.8%	99.9%
1	1.000	1.376	1.963	3.078	6.314	12.706	31.821	63.657	127.321	318.309	636.619
2	0.816	1.061	1.386	1.886	2.920	4.303	6.965	9.925	14.089	22.327	31.599
3	0.765	0.978	1.250	1.638	2.353	3.182	4.541	5.841	7.453	10.215	12.924
4	0.741	0.941	1.190	1.533	2.132	2.776	3.747	4.604	5.598	7.173	8.610
5	0.727	0.920	1.156	1.476	2.015	2.571	3.365	4.032	4.773	5.893	6.869
6	0.718	0.906	1.134	1.440	1.943	2.447	3.143	3.707	4.317	5.208	5.959
7	0.711	0.896	1.119	1.415	1.895	2.365	2.998	3.499	4.029	4.785	5.408
8	0.706	0.889	1.108	1.397	1.860	2.306	2.896	3.355	3.833	4.501	5.041
9	0.703	0.883	1.100	1.383	1.833	2.262	2.821	3.250	3.690	4.297	4.781
						• • •					
30	0.683	0.854	1.055	1.310	1.697	2.042	2.457	2.750	3.030	3.385	3.646
40	0.681	0.851	1.050	1.303	1.684	2.021	2.423	2.704	2.971	3.307	3.551
50	0.679	0.849	1.047	1.299	1.676	2.009	2.403	2.678	2.937	3.261	3.496
60	0.679	0.848	1.045	1.296	1.671	2.000	2.390	2.660	2.915	3.232	3.460
80	0.678	0.846	1.043	1.292	1.664	1.990	2.374	2.639	2.887	3.195	3.416
100	0.677	0.845	1.042	1.290	1.660	1.984	2.364	2.626	2.871	3.174	3.390
120	0.677	0.845	1.041	1.289	1.658	1.980	2.358	2.617	2.860	3.160	3.373
∞	0.674	0.842	1.036	1.282	1.645	1.960	2.326	2.576	2.807	3.090	3.291

Image credits to Wikipedia

Confidence Intervals on Errors

Confidence intervals can be straightforwardly estimated for simple (1D) linear regression

$$CI = \hat{y} \pm z \cdot \overline{err}$$

- z is the critical value corresponding to the expected confidence level α (e.g. $\alpha = 95\%$) assuming a Student distribution with N-2 degrees of freedom

$$z = t_{1-\alpha,N-2}$$

- $\overline{err}$ is the standard estimate of error

$$\overline{err} = \sqrt{MSE\left(\frac{1}{\mathrm{N}} + \frac{(x - \bar{\mu}_x)^2}{\sum_{i=1}^{N}(x_i - \bar{\mu}_x)^2}\right)}$$

Logistic Regression

Logistic regression setting

Binary classification task: given an input $\boldsymbol{x}$ assign a class in $y \in \{0,1\}$ according the unknown function $y = f(\boldsymbol{x})$ using a model h_θ

- Usual dataset $D = \{(\boldsymbol{x}^1, y^1);\ \dots\ (\boldsymbol{x}^N, y^N)\}$
- Input/free variables: i.e. $\boldsymbol{x}^n = [x_1^n, \dots, x_k^n, \dots, x_D^n] \in \mathbb{R}^D$
- Output/response variable y

features →

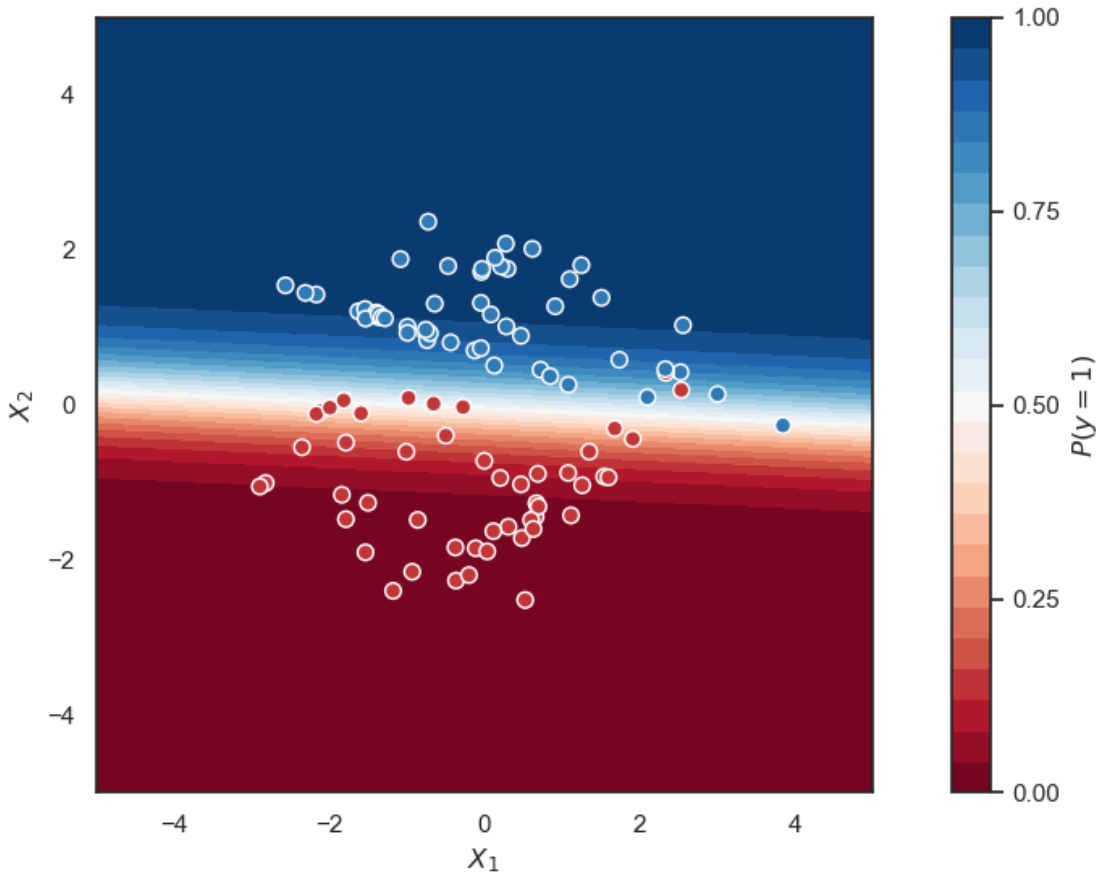
samples ↓

	x_0	x_1	x_2	...	x_n	y (class)
$\boldsymbol{x}^{(1)}$	1	0.1	0.4	...	-0.3	1
$\boldsymbol{x}^{(2)}$	1	-0.3	0.5	...	1.0	0
...	...	1	...	...	...	...
$\boldsymbol{x}^{(i)}$	1	0.4	0.5	...	-0.8	0
...	...	1	...	...	...	...
$\boldsymbol{x}^{(m)}$	1	1.2	-0.8	...	-0.3	1

Understanding Logistic Regression

Learns a decision boundary separating two classes

- Assigns an input $\boldsymbol{x}$ to the probability of being in class 1, i.e. $P(y = 1|\boldsymbol{x})$
- We check on which side of the boundary the sample falls into and assign the class accordingly
- Distance from the boundary affects the probability



Building the logistic regression

We start again from a linear model

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_D x_D = \sum_{k=1}^{D} \theta_k x_k = \boldsymbol{x\theta}$$

Ideally

- $\theta_k > 0$: the feature "is related to" class 1
- $\theta_k < 0$: the feature "is related to" class 0
- $\theta_k = 0$: the feature is irrelevant

We sum the features weighted by the parameters.

- Positive result: I assign class 1.
- Negative result: I assign class 0.

In practice..

- Things are a bit more complex
- Logistic regression assigns a probability to each sample:
- A value in the range **(0,1)**
  - **1**: I am certain input **x** belongs to class **1**
  - **0**: I am certain input **x** does **not** belong to class **1**
  - Everything in between represents the **degree of confidence** in class **1**
- Problem: I need to squash the $\boldsymbol{x\theta}$ (unbound) regression in [0,1] respecting the sum-to-1 constraint of probabilities. How?

The Sigmoid

Defined as

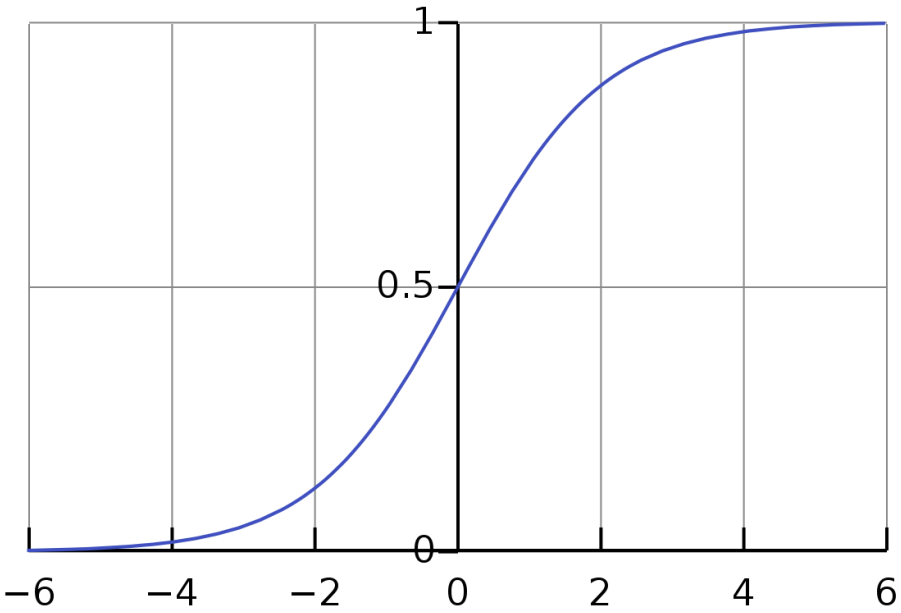
$$\sigma(\mathrm{x}) = \frac{1}{1 + e^{-x}}$$

Behaviour:

- $x \geq 6 \rightarrow \sigma(\mathrm{x}) \approx 1$
- $x \leq -6 \rightarrow \sigma(\mathrm{x}) \approx 0$
- $x = 0 \rightarrow \sigma(\mathrm{x}) = 0.5$

So, we can have:

- If $\boldsymbol{x\theta}$ is very positive, $\sigma(\boldsymbol{x\theta}) \approx 1$
- If $\boldsymbol{x\theta}$ is very negative, $\sigma(\boldsymbol{x\theta}) \approx 0$



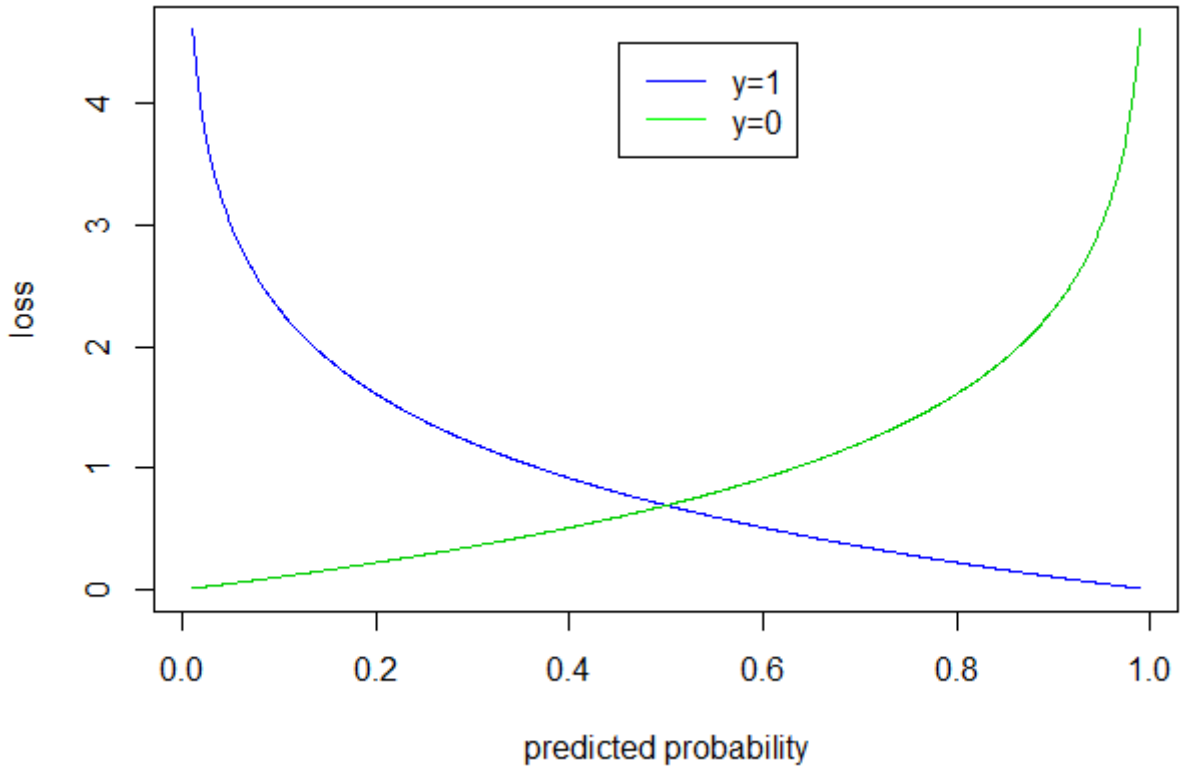
Summarizing our logistic regression model is

$$\mathrm{h}_{\theta}(\mathrm{x}) = \sigma(\boldsymbol{x\theta})$$

Training the logistic regression

Choose a suitable loss: the binary cross-entropy (BCE)

$$L(h_\theta(\boldsymbol{x}), y) = -\mathrm{y}\log\big(h_\theta(\boldsymbol{x})\big) - (1-y)\log\big(1-h_\theta(\boldsymbol{x})\big)$$



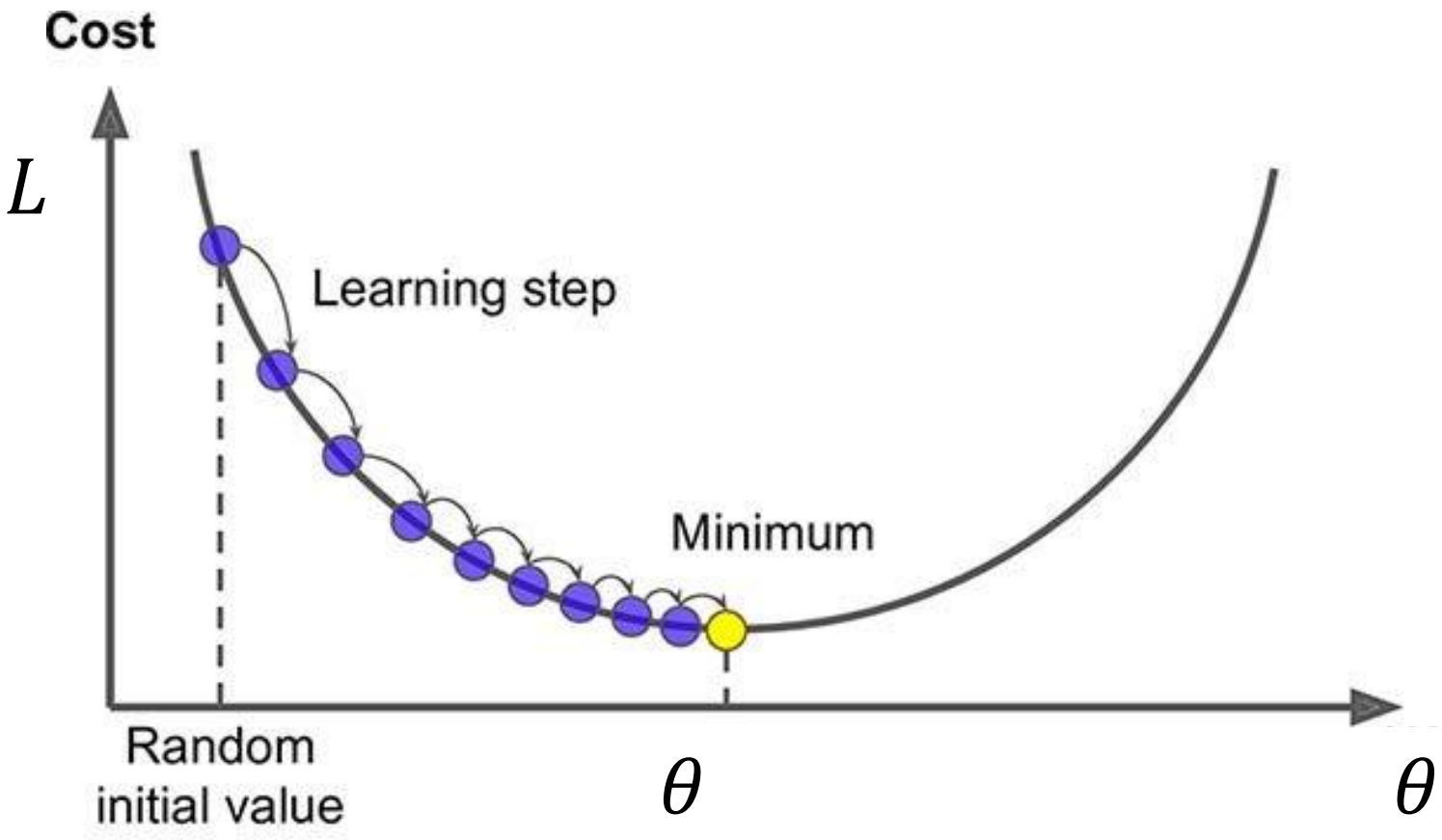
A Gradient Based Approach

- The gradient is the vector of partial derivatives of the loss function with respect to the weights $\boldsymbol{\theta}$

$$\nabla_{\boldsymbol{\theta}} L = \left.\frac{\partial L}{\partial \theta_k}\right|_{k=1\ldots\mathrm{D}} = \begin{bmatrix} \frac{\partial L}{\partial \theta_1} & \ldots & \frac{\partial L}{\partial \theta_d} \end{bmatrix}^T$$

- The gradient tells us how to modify the parameters in a way that increases the loss.
- To decrease the loss, we need to update the parameters in the opposite direction of the gradient

Gradient Descent



BCE Gradient

For Binary Cross-Entropy (BCE), the gradient is given by

$$\nabla_\theta L = \boldsymbol{x}(h_\theta(\boldsymbol{x}) - y)$$

Interpretation:

- $(h_\theta(\boldsymbol{x}) - y)$ is the error made when predicting y with the current parameters $\boldsymbol{\theta}$
- $\boldsymbol{x}(h_\theta(\boldsymbol{x}) - y)$ is the contribution of each feature to the error

Gradient Descent Algorithm

- Gradient Descent is an iterative algorithm used to find the minimum of any function.
- We use it to update the parameters $\boldsymbol{\theta}$ in a way that progressively reduces the loss.

Steps of the Algorithm

1. Initialize $\boldsymbol{\theta}$ with random values.
2. Compute the loss using the assigned $\boldsymbol{\theta}$ (call it $\boldsymbol{\theta}_{old}$)
3. Compute the gradient of the loss

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}}\boldsymbol{L} = \boldsymbol{x}(h_{\theta}(\boldsymbol{x}) - y)$$

3. Update $\boldsymbol{\theta}$ using the rule

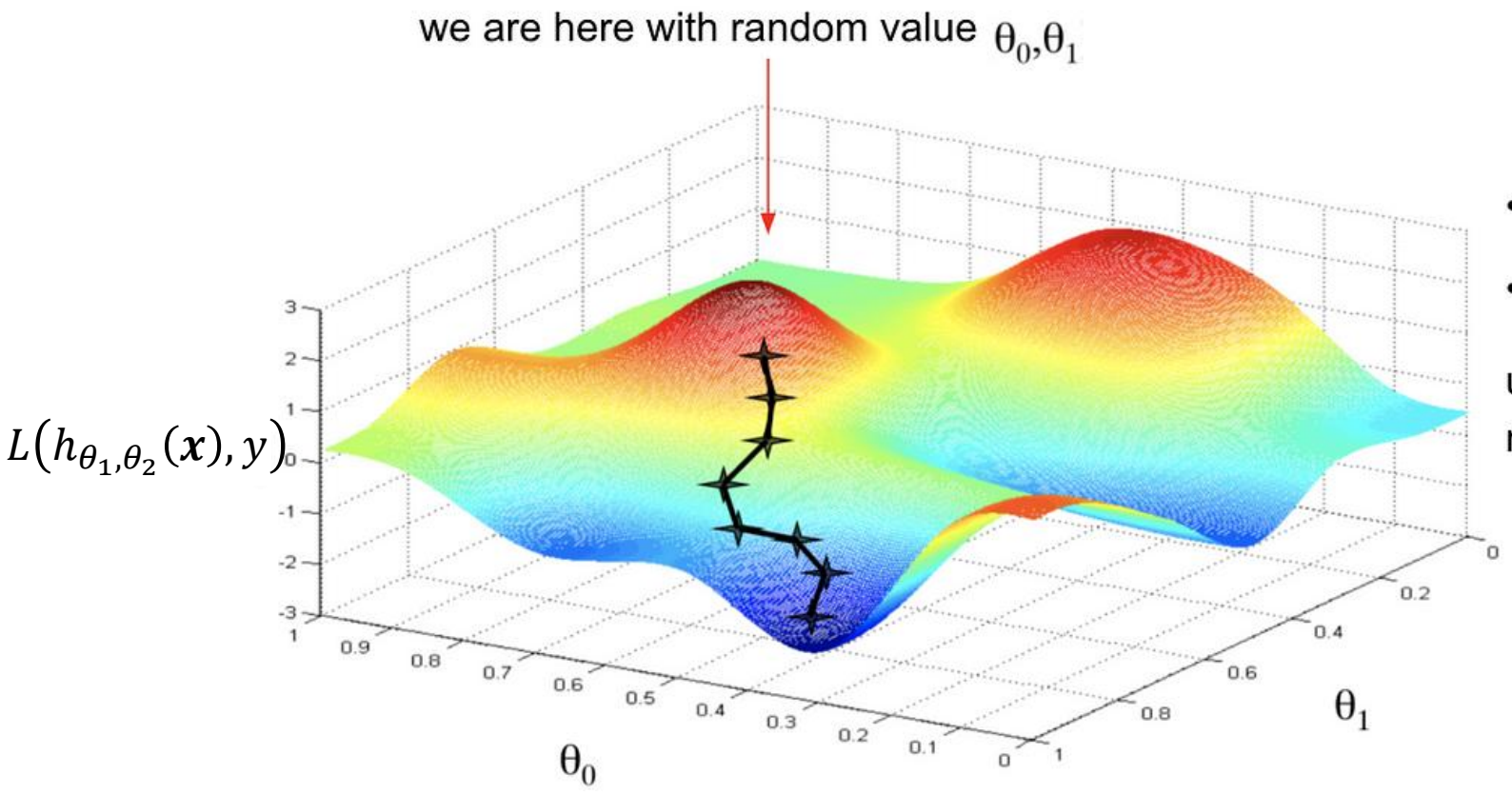
$$\boldsymbol{\theta}_{new} = \boldsymbol{\theta}_{old} - \eta\boldsymbol{\nabla}_{\boldsymbol{\theta}}\boldsymbol{L}$$

5. Repeat until reaching the minimum of the loss function.

Step Size (η - Learning Rate)

- η (learning rate) controls how big the update step is
- It is usually < 1 to ensure stable convergence

Gradient Descent on a Loss Landscape



- Start with some θ_0,θ_1
- Keep changing θ_0,θ_1 to reduce $L(h_{\theta_1,\theta_2}(\boldsymbol{x}),y)$

until we hopefully end up at a minimum

Training Algorithm Summary

For a certain number of iterations (epochs), the algorithm updates the parameters Θ based on the training data D_{train}.

For each training pair $\left(\boldsymbol{x}^i, y^i\right) \in D_{train}$:

1. Compute the prediction $h_\theta\left(\boldsymbol{x}^i\right) = \sigma(\boldsymbol{x}^i\boldsymbol{\theta})$
2. Compute the loss L of the prediction $h_\theta\left(\boldsymbol{x}^i\right)$ compared to the true label y^i
3. Compute the gradient of the loss $\boldsymbol{\nabla_\theta L}$
4. Update the parameters $\boldsymbol{\theta}_{new} = \boldsymbol{\theta} - \eta\boldsymbol{\nabla_\theta L}$
5. Use the updated parameters in the next iteration $\boldsymbol{\theta} = \boldsymbol{\theta}_{new}$

This process repeats for multiple epochs, allowing the model to progressively minimize the loss and improve its predictions

What if I add some regularization?

- Need to update the learning equations (descends again from taking the derivative of the error)
- Weight update with L1 (LASSO)

$$\boldsymbol{\theta}_{new} = \boldsymbol{\theta} - \eta(\boldsymbol{\nabla}_{\boldsymbol{\theta}}\boldsymbol{L} + \lambda\, sign(\boldsymbol{\theta}))$$

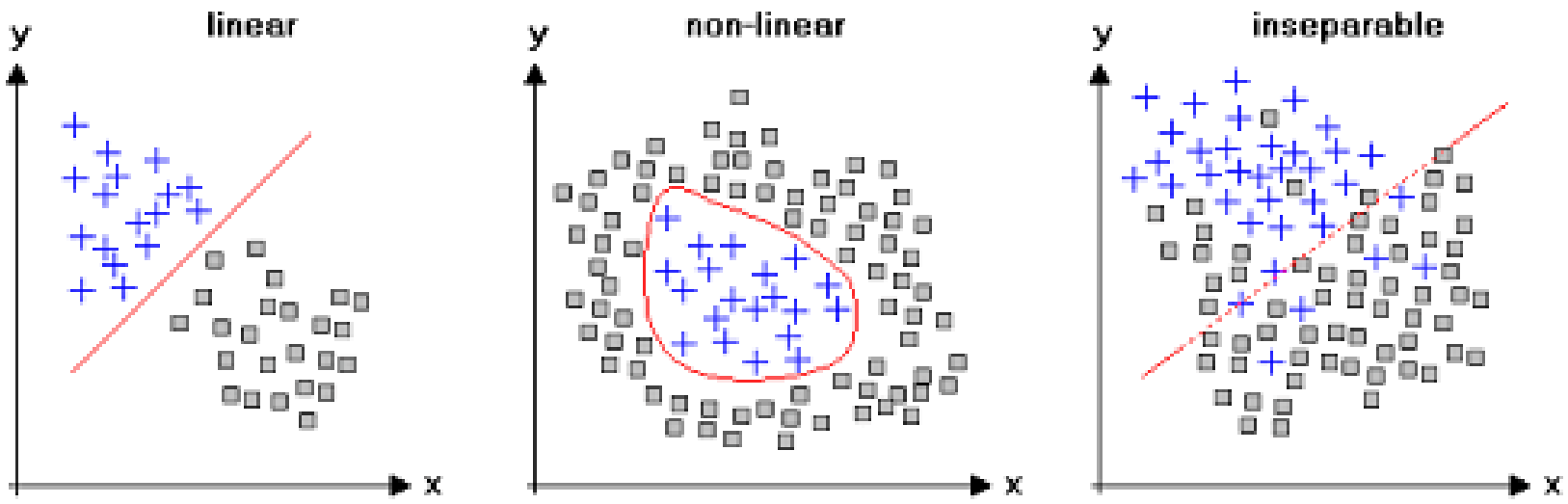
- Weight update with L2 (Ridge)

$$\boldsymbol{\theta}_{new} = \boldsymbol{\theta} - \eta(\boldsymbol{\nabla}_{\boldsymbol{\theta}}\boldsymbol{L} + 2\lambda\boldsymbol{\theta})$$

Limitations of Logistic Regression

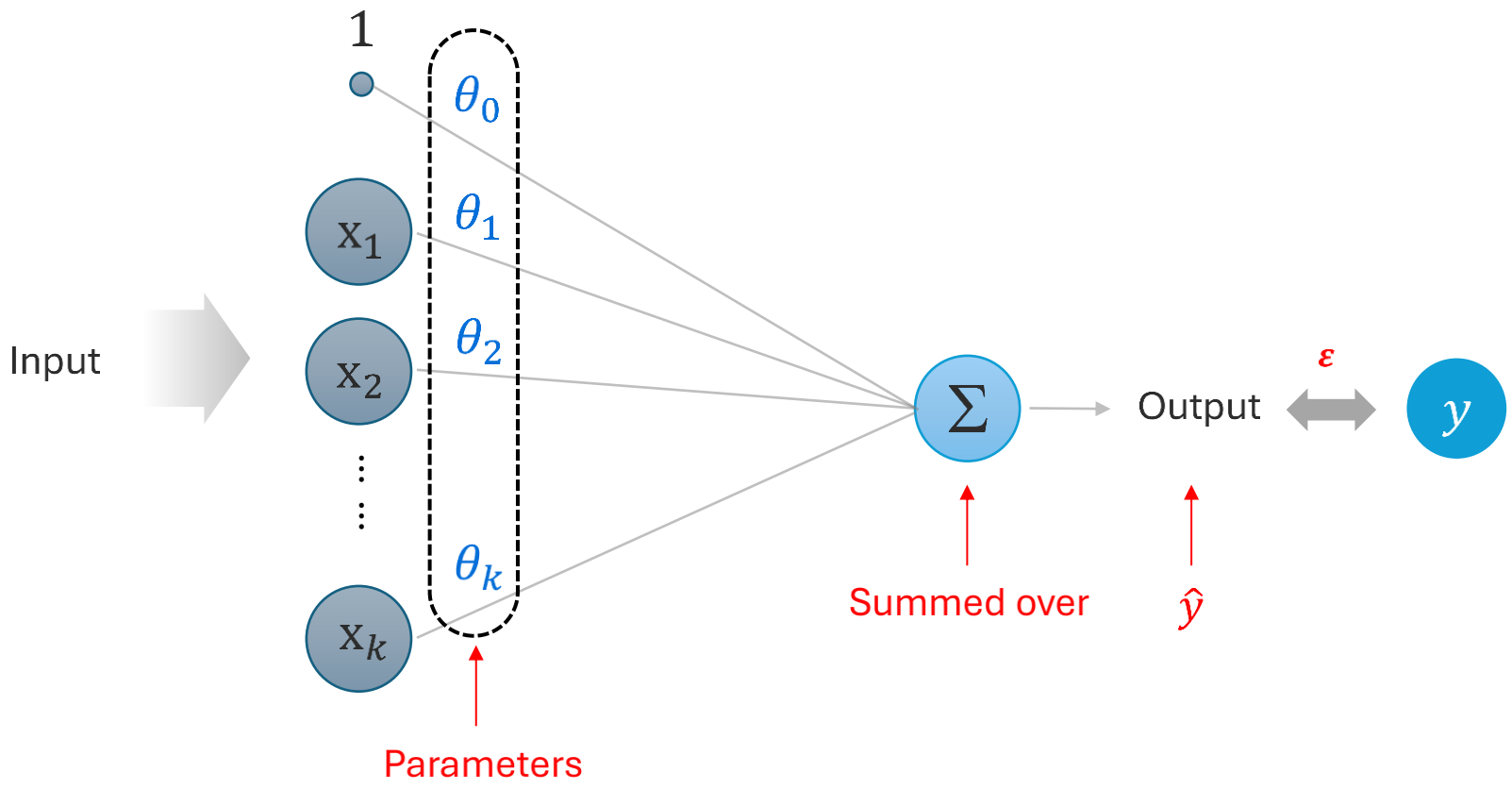
The decision boundary of logistic regression is linear (literally, a plane that separates the two classes).

- If the classes are linearly separable, logistic regression works perfectly.
- What if the classes are not separable?

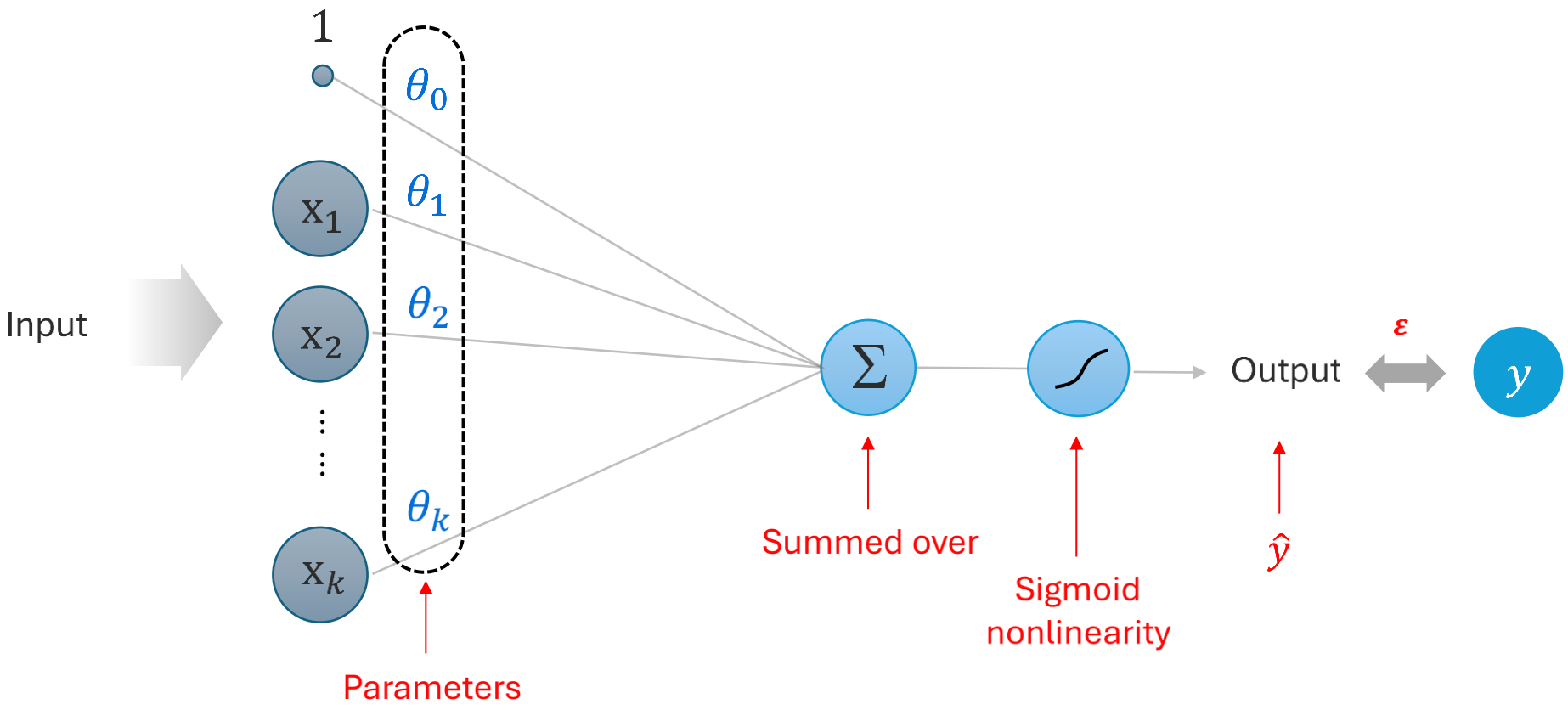


Wrap-up

Linear Regression towards Neural Networks



Logistic Regression towards Neural Networks



Take home lessons

- Linear models as your first ML method
  - Parameters are coefficients of the linear combination
  - Have a (potentially misleading) interpretation n terms of input feature importance
  - Not so far from a neural network
- Regularization into action as parameter value penalties
  - Ridge (L2-norm) – Smoothens collinearity issues; closed form learning solutions
  - Lasso (L1-norm) – Promotes feature sparsity; typically, gradient-based learning
  - ElasticNet – Best of both worlds
- Logistic Regression as first binary classifier providing with a probability of class membership
  - Widely used in early biomedical applications
- Model losses: MSE, MAE, BCE, …

Next Lecture

- Lab tutorials
- Introduction to neural networks (next week)
  - Modeling the artificial neuron
  - Artificial neural networks and the multilayer perceptron
    - Layered structure
    - Activation functions
    - Outputs and losses
  - Training Artificial neural networks
    - Backpropagation algorithm
    - Loss optimization