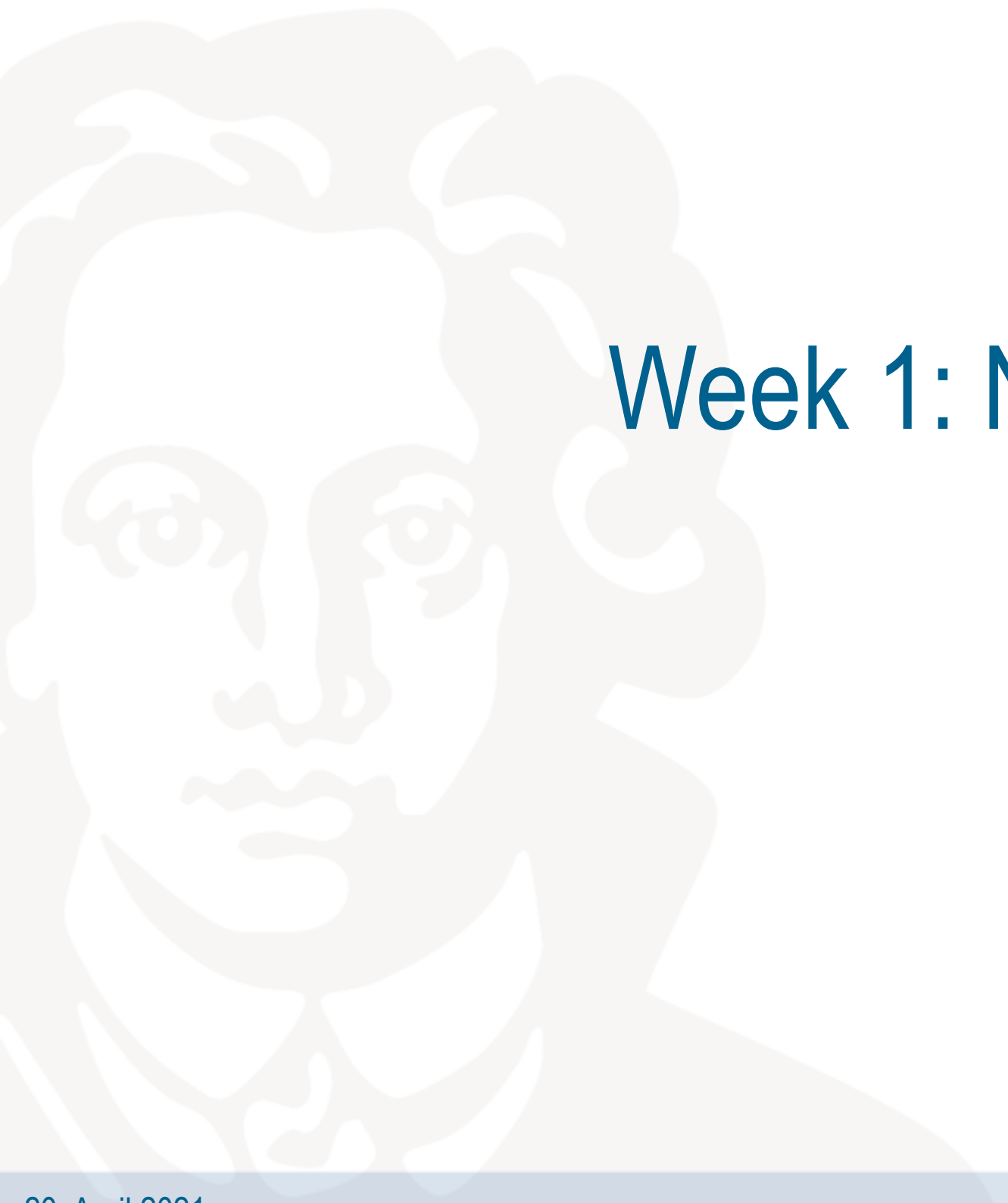


Achref Jaziri, Konrad Wartke, Prof. Dr. Visvanathan Ramesh

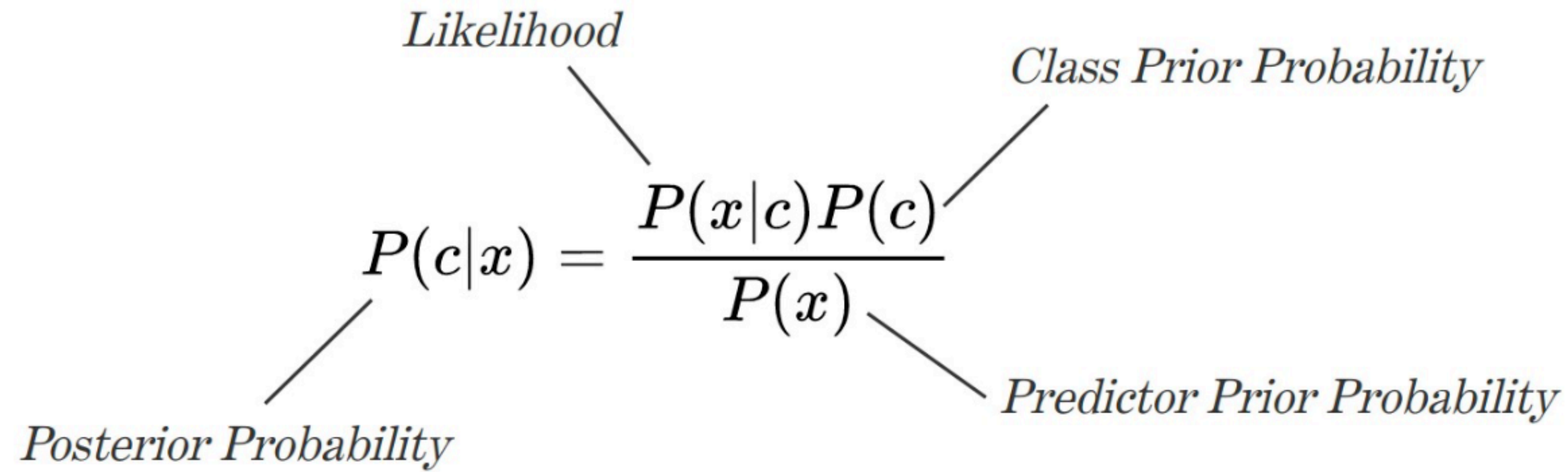
# Pattern Analysis & Machine Intelligence

## Praktikum: MLPR SS-21

### Week 1: Naive Bayes for Classification and Logistic Regression



# Bayes' Rule



The diagram shows the Bayes' Rule formula with labels pointing to its components. The formula is  $P(c|x) = \frac{P(x|c)P(c)}{P(x)}$ . The label 'Likelihood' points to  $P(x|c)$ . The label 'Class Prior Probability' points to  $P(c)$ . The label 'Posterior Probability' points to  $P(c|x)$ . The label 'Predictor Prior Probability' points to  $P(x)$ .

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

*Likelihood*

*Class Prior Probability*

*Posterior Probability*

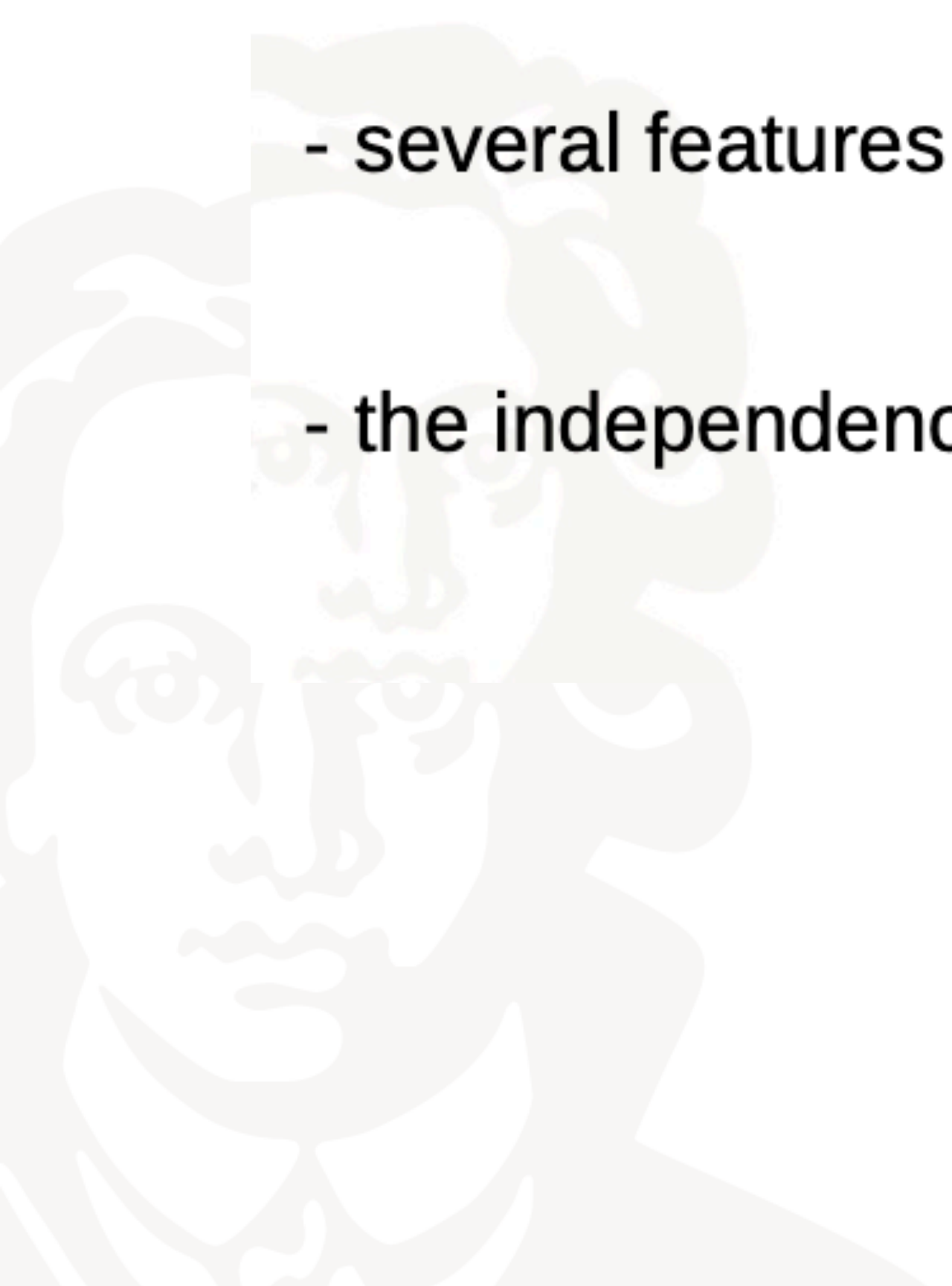
*Predictor Prior Probability*

<https://www.thelearningmachine.ai/naive>



# Bayes' rule: the naive assumption

- one feature case:  $P(C|X) \propto P(X|C)P(C)$
- several features case:  $P(C|X_1, X_2 \dots X_n) \propto P(X_1, X_2 \dots X_n|C)P(C)$
- the independency assumption:  $P(C|X_1, X_2 \dots X_n) \propto P(X_1|C)P(X_2|C) \dots P(X_n|C)P(C)$



# Naive Bayes Example

Feature 1	Feature 2	Classification
<b>Weather</b>	<b>Wind</b>	<b>Play</b>
Sunny	Strong	No
Overcast	Weak	Yes
Rainy	Moderate	Yes
Sunny	Weak	Yes
Sunny	Weak	Yes
Overcast	Weak	Yes
Rainy	Strong	No
Rainy	Strong	No
Sunny	Weak	Yes
Rainy	Weak	Yes
Sunny	Strong	No
Overcast	Weak	Yes
Overcast	Weak	Yes
Rainy	Moderate	No

$$\begin{aligned}
 P(C = \text{yes} | X_1 = \text{Moderate}, X_2 = \text{Sunny}) &\propto \\
 P(X_1 = \text{Moderate}, X_2 = \text{Sunny} | C = \text{yes}) P(C = \text{yes}) &= \\
 P(X_1 = \text{Moderate} | C = \text{yes}) P(X_2 = \text{Sunny} | C = \text{yes}) P(C = \text{yes}) &= \\
 \frac{1}{14} \frac{3}{14} \frac{9}{14} &= 0.0098
 \end{aligned}$$

$$\begin{aligned}
 P(C = \text{no} | X_1 = \text{Moderate}, X_2 = \text{Sunny}) &\propto \\
 P(X_1 = \text{Moderate}, X_2 = \text{Sunny} | C = \text{no}) P(C = \text{no}) &= \\
 P(X_1 = \text{Moderate} | C = \text{no}) P(X_2 = \text{Sunny} | C = \text{no}) P(C = \text{no}) &= \\
 \frac{1}{14} \frac{2}{14} \frac{5}{14} &= 0.0036
 \end{aligned}$$

$$P(C = \text{yes} | X_1 = \text{Moderate}, X_2 = \text{Sunny}) > P(C = \text{no} | X_1 = \text{Moderate}, X_2 = \text{Sunny})$$

<https://www.thelearningmachine.ai/naive>



# Bag of words representation

- **n**: number of words in a document
- **m**: vocabulary size
- **$x_j$** : frequency of a word  $j$  in a document
- **$\theta_j$** : probability of a word  $j$

$$P(x; \theta) = \frac{n!}{\prod_{j=1..m} x_j!} \prod_{j=1..m} \theta_j^{x_j}$$

<http://cseweb.ucsd.edu/~elkan/250B/topicmodels.pdf>

**Training data:** a corpus of messages/texts

**Task:** classify messages as spam or non-spam - **Attributes:**  
words in the messages

**Naive Bayes classifier:**

$$c_{NB} = \arg \max_{c_i \in \text{spam}, \text{non-spam}} P(c_i) \prod P(w_i | c_i)$$

[http://www.coli.uni-saarland.de/~crocker/Teaching/Connectionist/lecture10\\_4up.pdf](http://www.coli.uni-saarland.de/~crocker/Teaching/Connectionist/lecture10_4up.pdf)

# Generative vs. discriminative

- We first learned about Naïve Bayes -> a **generative** classification approach

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} \quad p(x, y) = p(x|y)p(y)$$

- The classifier is generative because we create a joint model of **p(x,y)** and can then condition on the observed features x.
- A **discriminative** classification approach is logistic regression
- The key difference is that we directly fit a model to **p(y|x)** and directly map x to y. We can discriminate, but we can no longer specify how to generate observed features x for each class.

# From linear to logistic regression (classification)

Before we can delve into the details of logistic regression, we first need to learn about two concepts:

- Linear regression (non-Bayesian)
- Empirical risk optimization – in this context: gradient descent

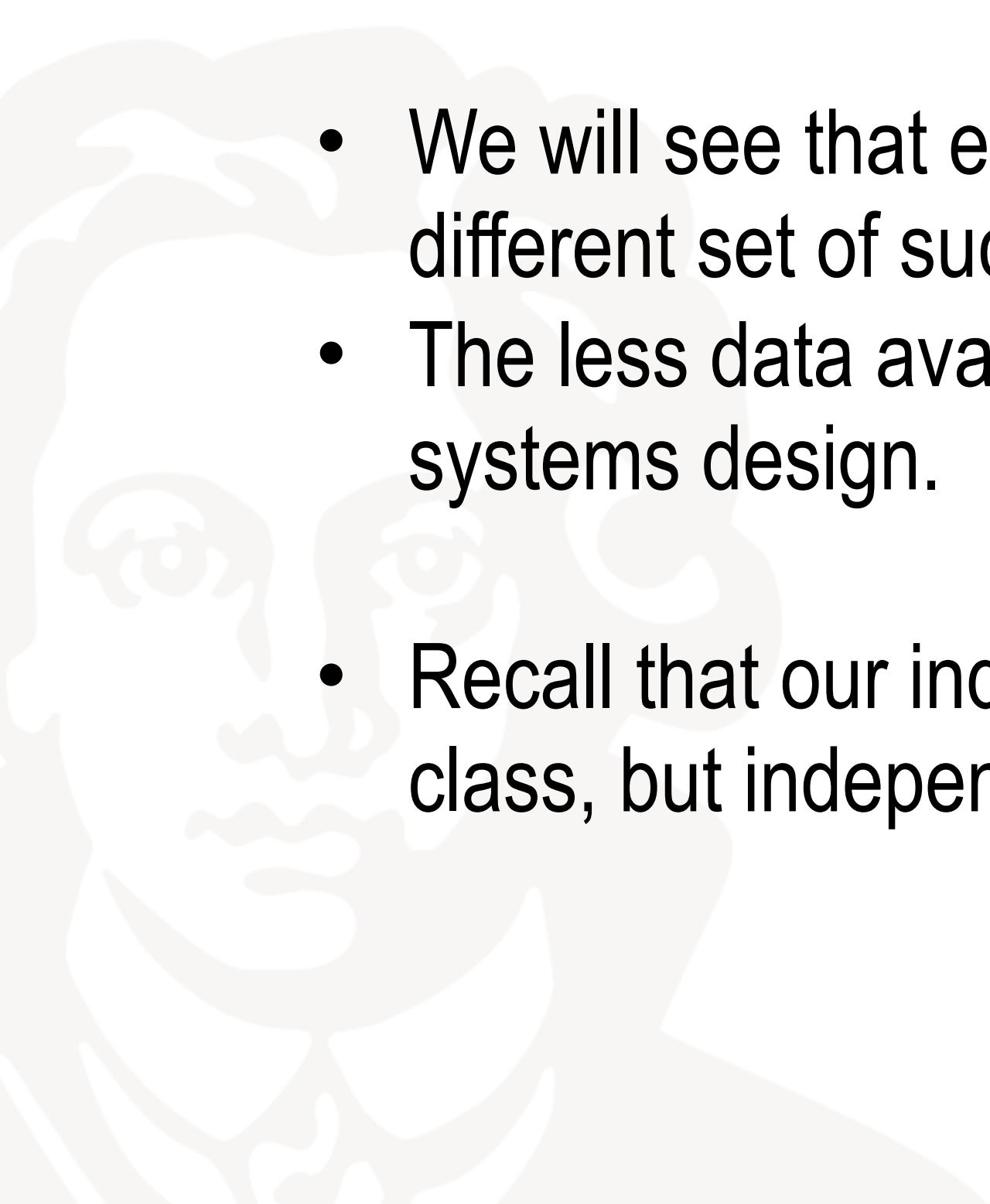
In machine learning we typically distinguish by the nature of our target:

- *Regression*, when our target is *continuous*
- *Classification*, when our target is *discrete*

We will see that logistic regression is actually a classification algorithm, but it is motivated and derived from linear regression.



- In Linear regression we make a strong assumption on the nature of the data distribution: we assume that the relationship between input and output is linear.
- We refer to such assumptions as *inductive bias*.
- We will see that each model that we will encounter in the course of our lecture comes with a different set of such assumptions.
- The less data available, the more assumptions are needed. This tradeoff is at the center of AI systems design.
- Recall that our inductive bias for **Naïve Bayes** was that each input is dependent only on the class, but independent from one another.



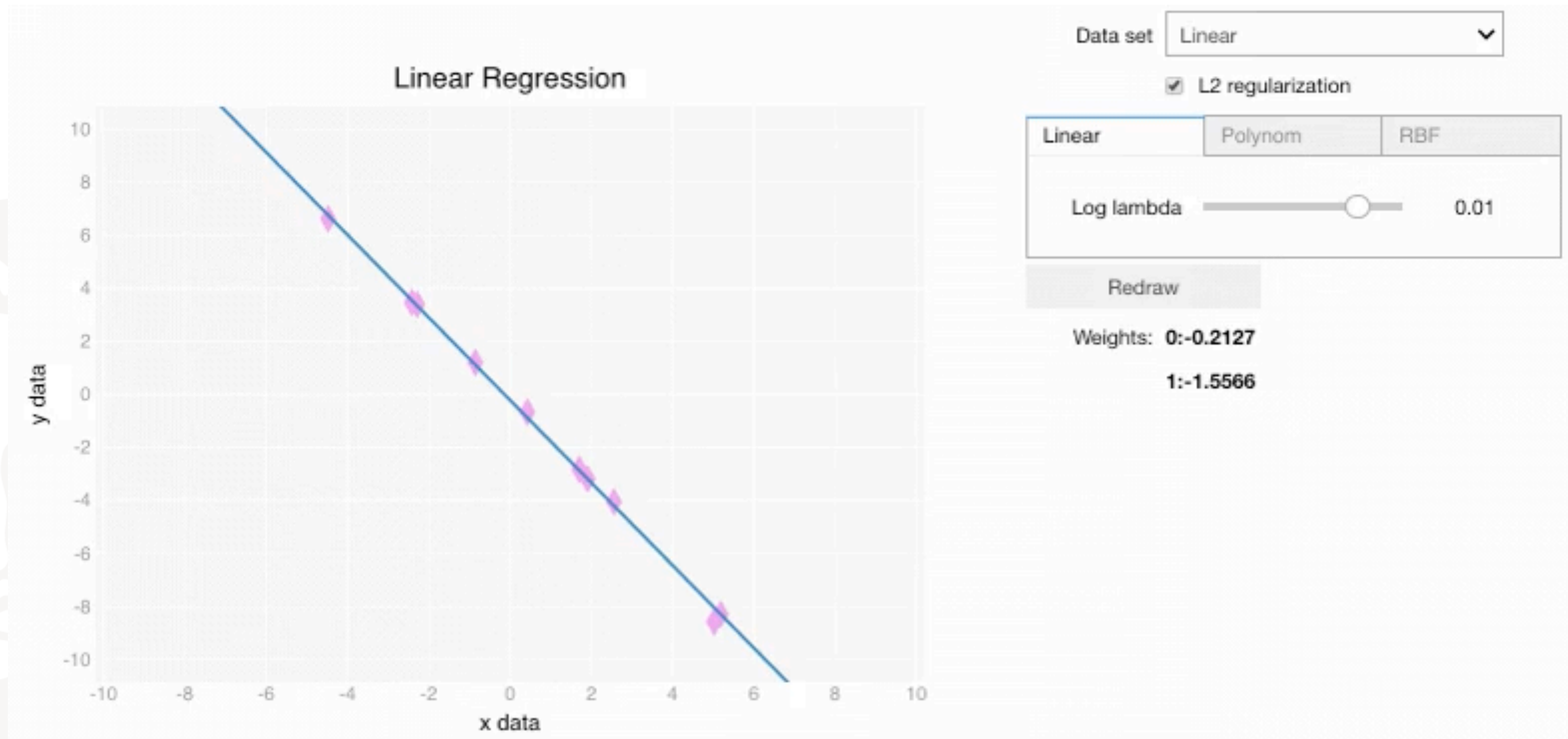
# Linear regression

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + \epsilon = \sum_{i=1}^D w_i x_i + \epsilon$$

- Here, we have the inner product between an input vector  $\mathbf{x}$  and the model's weight vector  $\mathbf{w}$  (often also referred to as the model parameters  $\theta$ )
- Epsilon is the residual error between our prediction and the true response.
- We generally assume epsilon to be Gaussian distributed:  $\epsilon \sim \mathcal{N}(\mu, \sigma^2)$
- Correspondingly, we can rewrite our model as:  $p(y | \mathbf{x}, \theta) = \mathcal{N}(y | \mu(\mathbf{x}), \sigma^2(\mathbf{x}))$
- In the simplest case, we assume  $\mu(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$  and sigma (noise) to be fixed
- We could model non-linear relationships by replacing  $\mathbf{x}$  with some non-linear function  
 $p(y | \mathbf{x}, \theta) = \mathcal{N}(y | \mathbf{w}^T \Phi(\mathbf{x}), \sigma^2)$  (-> basic function expansion, e.g. in polynomial regression)

# Linear regression

Let's take a look at an interactive example: <https://github.com/PyMLVizard/PyMLViz>



# From linear to logistic regression

- We can adapt linear regression to binary classification

1. Since the target  $y$  is binary  $\{0,1\}$ , replace Gaussian by Bernoulli:

$$p(y | \mathbf{x}, \boldsymbol{\theta}) = \text{Ber}(y | \mu(\mathbf{x})) \quad \text{with} \quad \mu(\mathbf{x}) = \mathbb{E}[y | \mathbf{x}] = p(y = 1 | \mathbf{x})$$

2. After the linear combination of inputs, we now pass through the sigmoid function, that ensures  $0 \leq \mu(\mathbf{x}) \leq 1$ .

$$\mu(\mathbf{x}) = \text{sigm}(\mathbf{w}^T \mathbf{x}) \quad \text{with} \quad \text{sigm}(z) = \frac{1}{1 + \exp(-z)} = \frac{e^z}{e^z + 1}$$

- If we threshold the obtained output probability at 0.5, we gain a *decision rule*:

$$\hat{y}(\mathbf{x}) = 1 \iff p(y = 1 | \mathbf{x}) > 0.5$$

- Note that this is still a **linear model**, but we can again make it non-linear/more complex with basis function expansion.

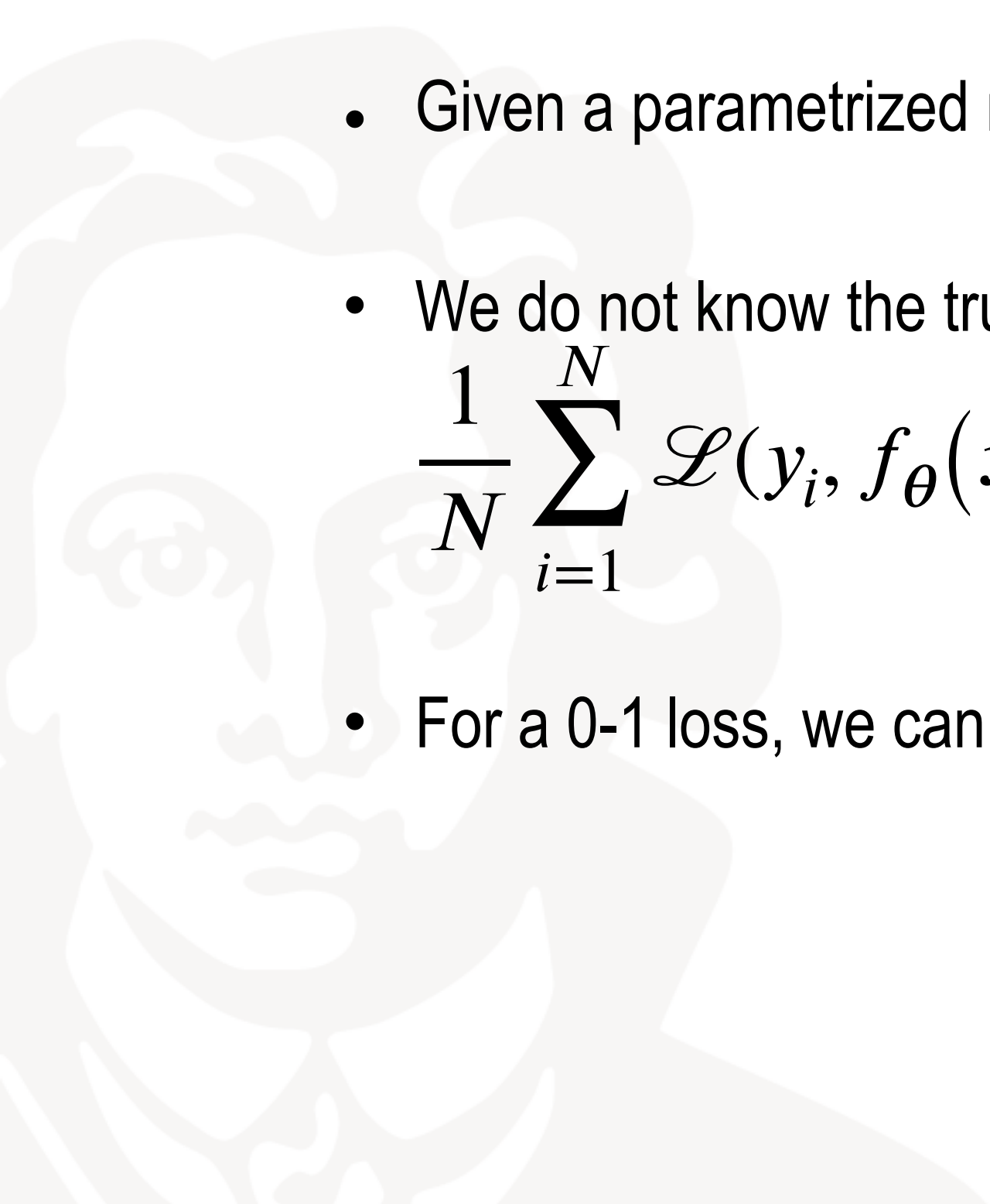


How do we interpret logistic regression?  $p(y | \mathbf{x}, \mathbf{w}) = \text{Ber}(y | \text{sigm}(\mathbf{w}^T \mathbf{x}))$

- For binary classification we can define the *log odds*:  $\log\left(\frac{p(y = 1 | \mathbf{x})}{p(y = 0 | \mathbf{x})}\right)$
- It neither outcome is favored  $\rightarrow \log \text{ odds} = 0$
- If one outcome is favored by  $\log \text{ odds} = a$ , the other is disfavored by  $-a$
- Example - two features: 1. number of cigarettes smoked per day, 2. minutes of exercise a day  
If we estimate our model parameters/weights as 1.3, -1.1, your chance of getting cancer increases by  $e^{1.3}$  for every smoked cigarette.
- How do we estimate the parameters/weights?

# Logistic regression and empirical optimization

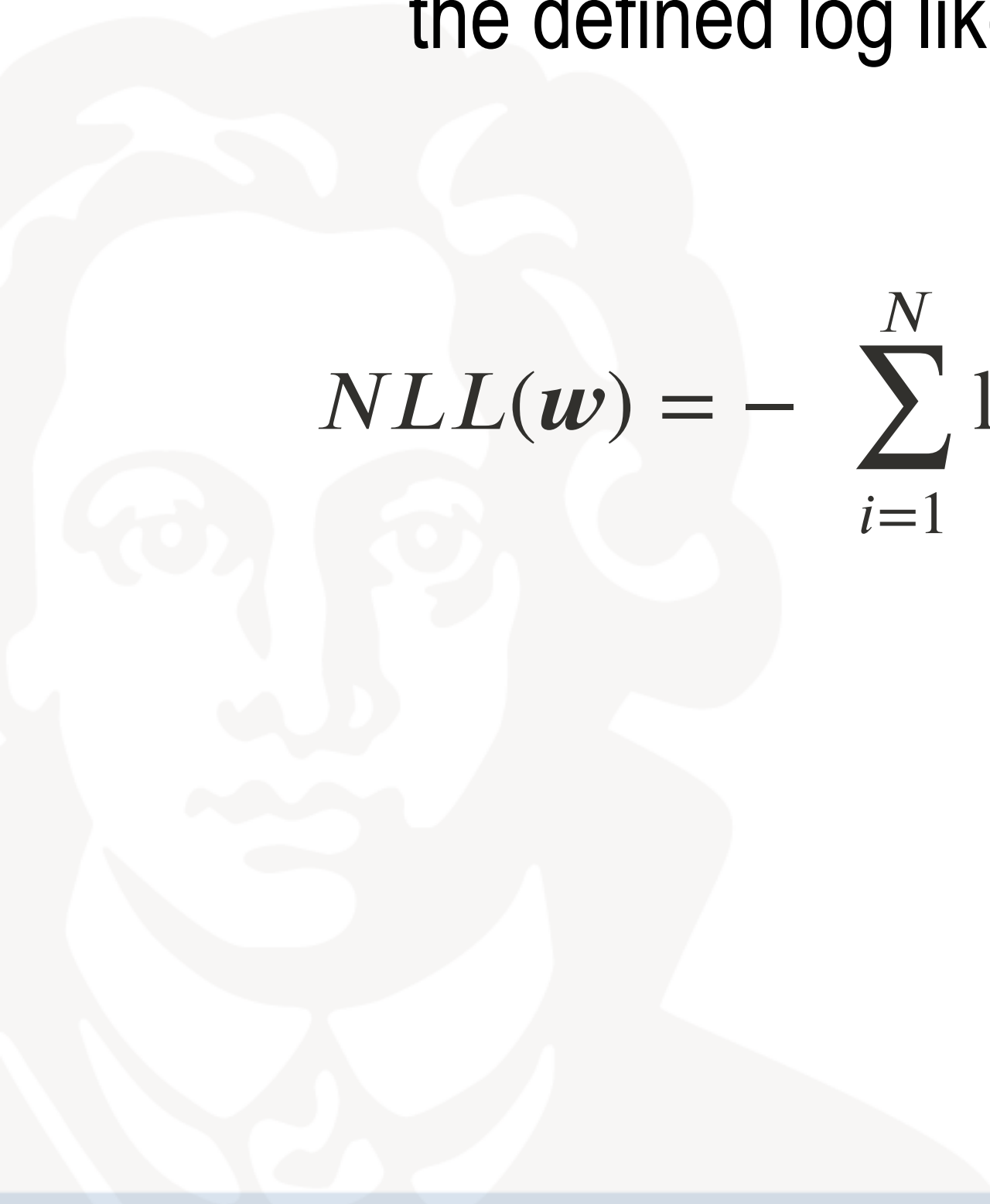
- Unfortunately, in contrast to linear regression, we cannot directly solve our equations to obtain the best set of parameters.
- In contrast to fully Bayesian approaches, we do not have a prior, and hence also no posterior. We have a decision likelihood.
- However, we can *measure the difference between our prediction  $\hat{y}$  and our true outcome  $y$* .
- Given a parametrized model  $f_{\theta}$  we can formulate a loss function:  $\mathbb{E}_{(\mathbf{x}, y) \sim p^*} [\mathcal{L}(y, f_{\theta}(\mathbf{x}))]$
- We do not know the true data distribution  $p^*$ , we only have  $N$  data samples, so we rely on an empirical estimate:  
$$\frac{1}{N} \sum_{i=1}^N \mathcal{L}(y_i, f_{\theta}(\mathbf{x}_i))$$
- For a 0-1 loss, we can refer to this as a misclassification rate, which we would like to minimize!



# Empirical optimization: surrogate loss function

- An additional challenge we face is that 0-1 loss is not smooth. It can thus be hard to optimize.
- We can define a *surrogate loss function*. In the case of logistic regression we can make use of the defined log likelihoods and the corresponding sigmoid probability distribution.

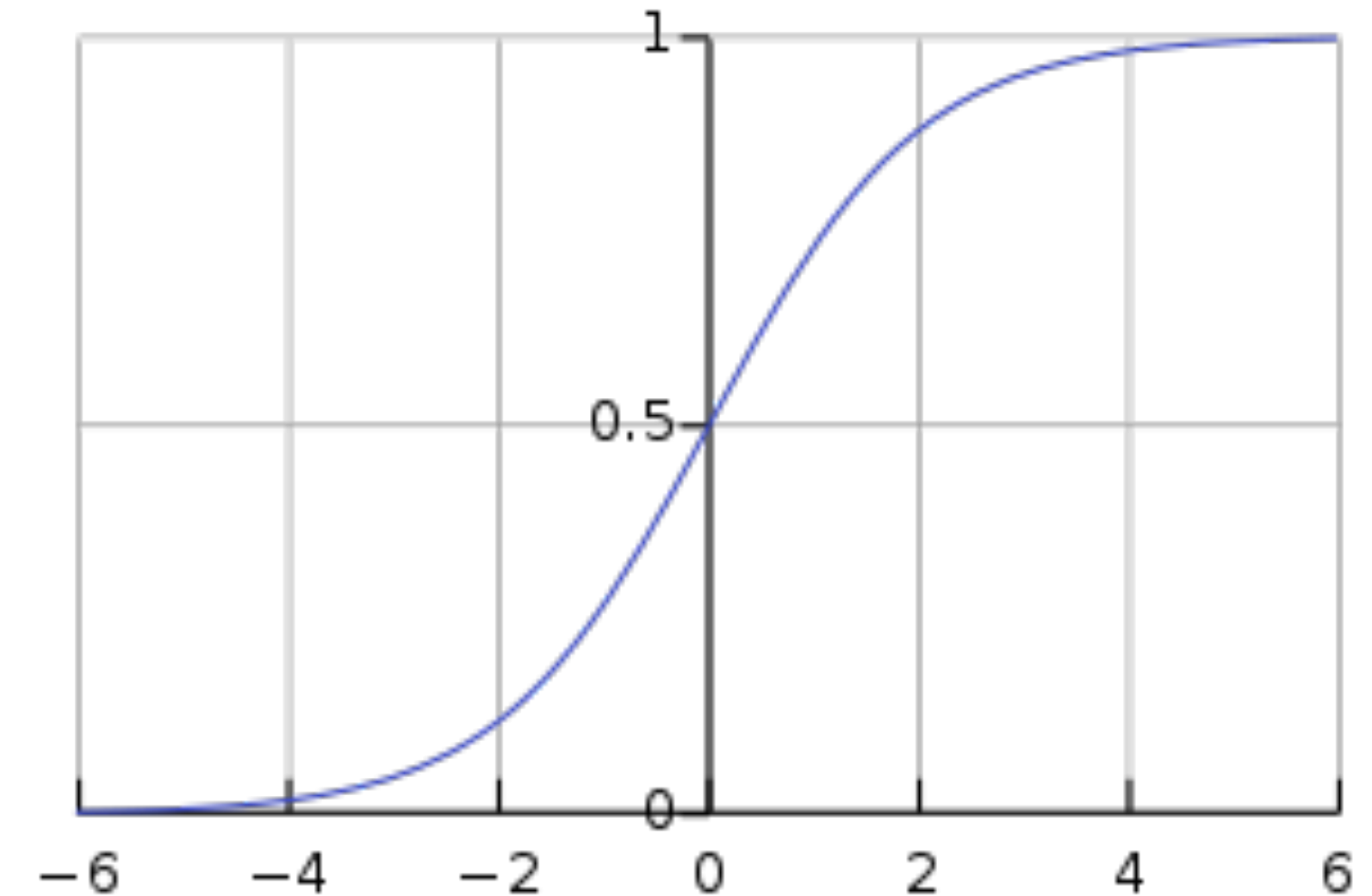
$$NLL(\mathbf{w}) = - \sum_{i=1}^N \log[\mu_i^{\mathbb{I}(y_i=1)} \times (1 - \mu_i)^{\mathbb{I}(y_i=0)}] = - \sum_{i=1}^N \left[ y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i) \right]$$



# Learning logistic regression

$$\Rightarrow P(y = 1 \mid \mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + \mathbf{x}^T \mathbf{w}_1)}}$$

- We can find the maximum log likelihood by setting the derivatives of the log-likelihood with respect to the weights (parameters) to zero.



[https://en.wikipedia.org/wiki/Sigmoid\\_function](https://en.wikipedia.org/wiki/Sigmoid_function)

$$\frac{\partial}{\partial w_i} \frac{1}{N} \sum_{i=1}^N [y_i \log(P(y_i = 1 \mid x_i, w)) + (1 - y_i) \log(1 - P(y_i = 1 \mid x_i, w))] = 0$$

- We need to derive above expression -> apply the chain rule multiple times. Because above expression isn't closed form, we have to solve it numerically by maximizing the likelihood or using e.g. *gradient descent* to minimize the negative thereof.



# Logistic Regression cost function derivation

$$\begin{aligned}
 \frac{\partial J(\theta)}{\partial \theta_j} &= \frac{\partial}{\partial \theta_j} \frac{-1}{m} \sum_{i=1}^m [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))] \\
 &\stackrel{\text{linearity}}{=} \frac{-1}{m} \sum_{i=1}^m \left[ y^{(i)} \frac{\partial}{\partial \theta_j} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \frac{\partial}{\partial \theta_j} \log(1 - h_\theta(x^{(i)})) \right] \\
 &\stackrel{\text{chain rule}}{=} \frac{-1}{m} \sum_{i=1}^m \left[ y^{(i)} \frac{\frac{\partial}{\partial \theta_j} h_\theta(x^{(i)})}{h_\theta(x^{(i)})} + (1 - y^{(i)}) \frac{\frac{\partial}{\partial \theta_j} (1 - h_\theta(x^{(i)}))}{1 - h_\theta(x^{(i)})} \right] \\
 &\stackrel{h_\theta(x) = \sigma(\theta^\top x)}{=} \frac{-1}{m} \sum_{i=1}^m \left[ y^{(i)} \frac{\frac{\partial}{\partial \theta_j} \sigma(\theta^\top x^{(i)})}{h_\theta(x^{(i)})} + (1 - y^{(i)}) \frac{\frac{\partial}{\partial \theta_j} (1 - \sigma(\theta^\top x^{(i)}))}{1 - h_\theta(x^{(i)})} \right] \\
 &\stackrel{\sigma'}{=} \frac{-1}{m} \sum_{i=1}^m \left[ y^{(i)} \frac{\sigma(\theta^\top x^{(i)}) (1 - \sigma(\theta^\top x^{(i)})) \frac{\partial}{\partial \theta_j} (\theta^\top x^{(i)})}{h_\theta(x^{(i)})} - (1 - y^{(i)}) \frac{\sigma(\theta^\top x^{(i)}) (1 - \sigma(\theta^\top x^{(i)})) \frac{\partial}{\partial \theta_j} (\theta^\top x^{(i)})}{1 - h_\theta(x^{(i)})} \right] \\
 &\stackrel{\sigma(\theta^\top x) = h_\theta(x)}{=} \frac{-1}{m} \sum_{i=1}^m \left[ y^{(i)} \frac{h_\theta(x^{(i)}) (1 - h_\theta(x^{(i)})) \frac{\partial}{\partial \theta_j} (\theta^\top x^{(i)})}{h_\theta(x^{(i)})} - (1 - y^{(i)}) \frac{h_\theta(x^{(i)}) (1 - h_\theta(x^{(i)})) \frac{\partial}{\partial \theta_j} (\theta^\top x^{(i)})}{1 - h_\theta(x^{(i)})} \right] \\
 &\stackrel{\frac{\partial}{\partial \theta_j} (\theta^\top x^{(i)}) = x_j^{(i)}}{=} \frac{-1}{m} \sum_{i=1}^m [y^{(i)} (1 - h_\theta(x^{(i)})) x_j^{(i)} - (1 - y^{(i)}) h_\theta(x^{(i)}) x_j^{(i)}] \\
 &\stackrel{\text{distribute}}{=} \frac{-1}{m} \sum_{i=1}^m [y^i - y^i h_\theta(x^{(i)}) - h_\theta(x^{(i)}) + y^{(i)} h_\theta(x^{(i)})] x_j^{(i)} \\
 &\stackrel{\text{cancel}}{=} \frac{-1}{m} \sum_{i=1}^m [y^{(i)} - h_\theta(x^{(i)})] x_j^{(i)} \\
 &= \frac{1}{m} \sum_{i=1}^m [h_\theta(x^{(i)}) - y^{(i)}] x_j^{(i)}
 \end{aligned}$$

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

Taken from:

<https://stats.stackexchange.com/questions/278771/how-is-the-cost-function-from-logistic-regression-derived>

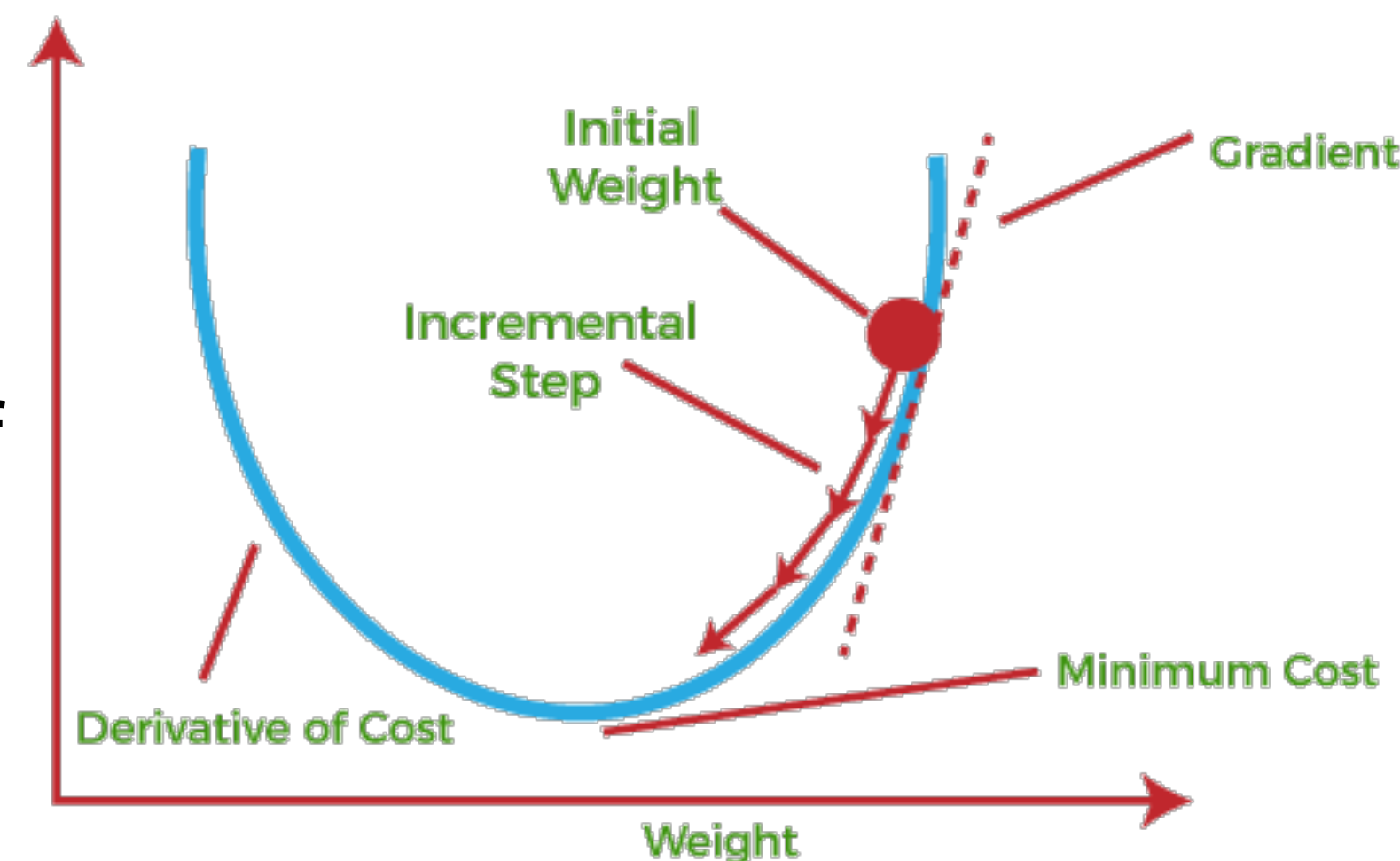
# Logistic Regression and Gradient Descent

- Let's solve our logistic regression with *gradient descent* (steepest descent). This will be useful because what we have just seen and will learn about gradient descent, will be our first building block when we go on to e.g. neural networks later.
- What is *gradient descent*?
  - \* In its simplest form, it is a first order optimization algorithm designed to find the minimum of a differentiable function.
  - \* This is achieved by iteratively taking (small) steps towards the negative of the gradient. For a function  $f(x)$  this is the direction in which it decreases the fastest.

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \lambda \nabla f(\mathbf{x}_n)$$

- With a step size  $\lambda$  (called learning rate in ML) we can find the minimum of the function

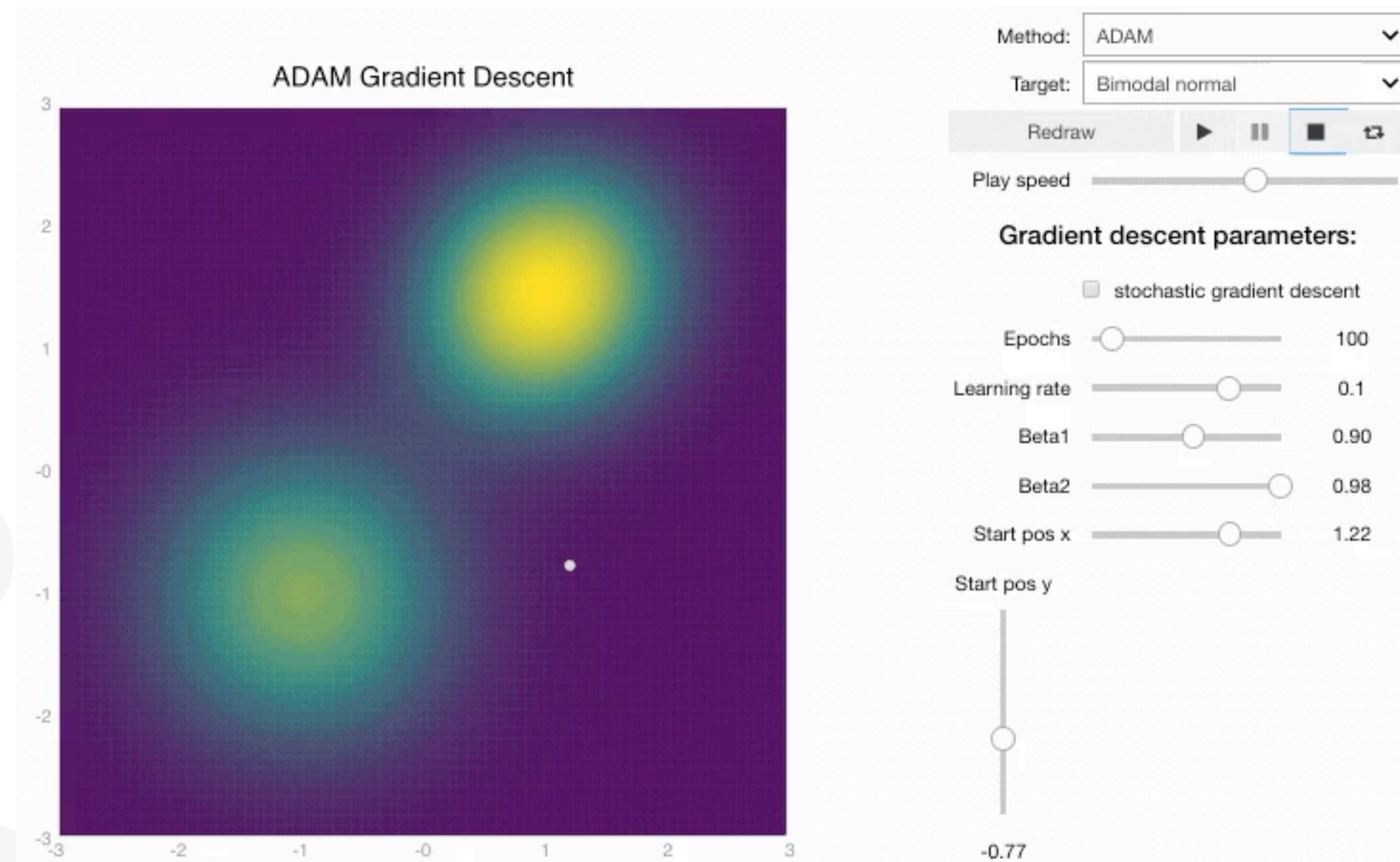
$$f(\mathbf{x}_0) \geq f(\mathbf{x}_1) \geq \dots \geq f(\mathbf{x}_n)$$





# Gradient Descent

We can again look at interactive examples: <https://github.com/PyMLVizard/PyMLViz>



# Gradient Descent: the ML perspective

- The previous slides gave an intuition of how to apply gradient descent to a function.  
How do we use gradient descent for ML?

=> We calculate the gradient based on the loss function with our model's predictions and iteratively learn a good set of parameters  $\theta$

(in ML we generally optimize  $\theta$  as there can be many more parameters than just the weights)

- In the simplest form we can directly update our weights (parameters) with the gradients:

$$= \theta - \lambda \nabla \mathcal{L}(\theta) = \theta - \lambda \frac{1}{N} \sum_{n=1}^N \nabla \mathcal{L}_n(\theta)$$

- There is many pitfalls, limitations, but also ways around it with variants and stochastic approximations. -> let's look at them together interactively