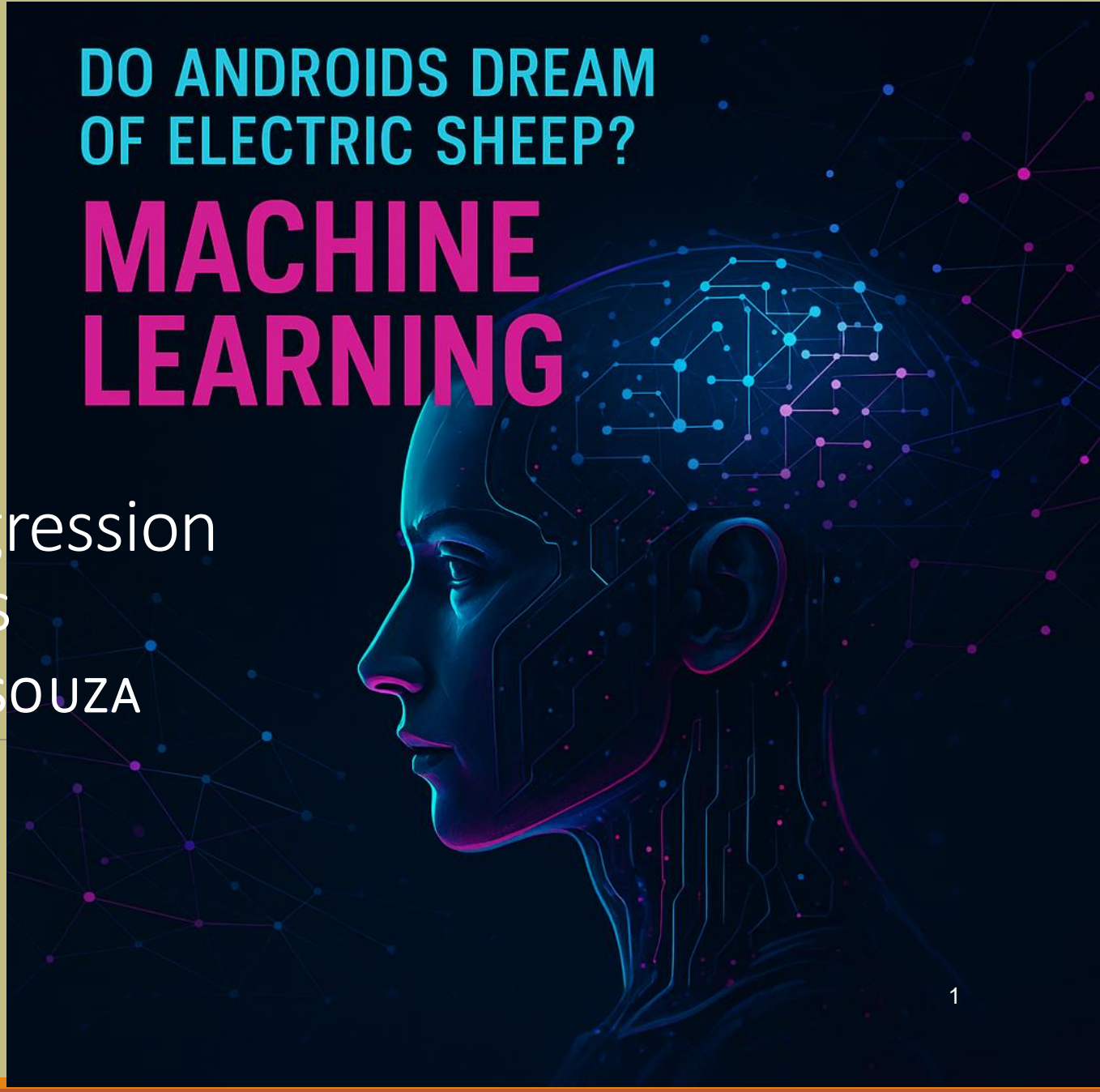
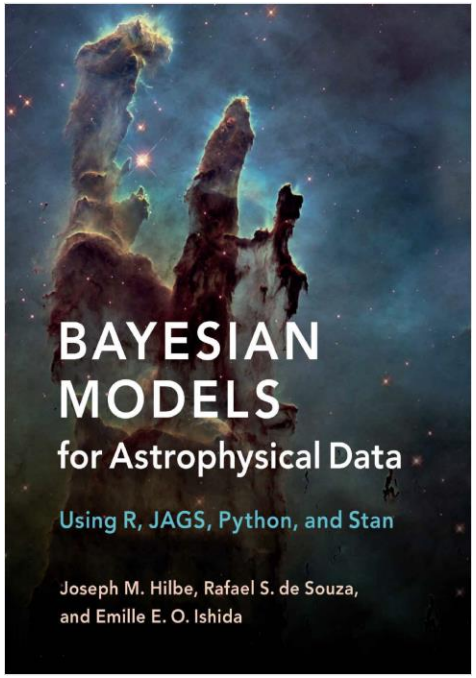
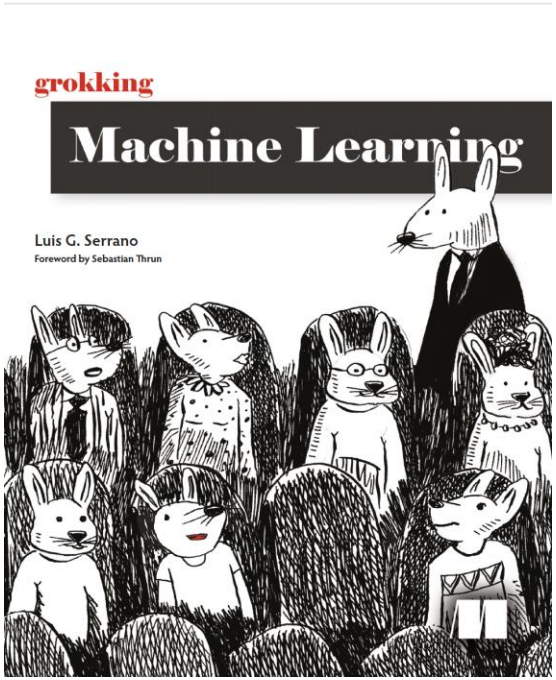
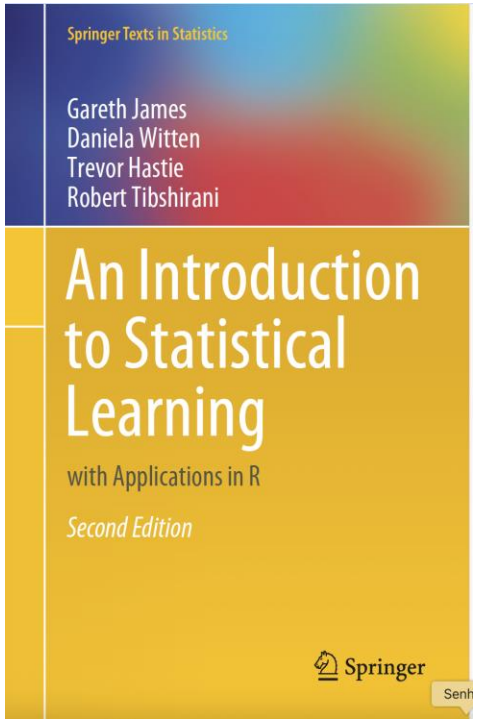
Advanced Regression Models

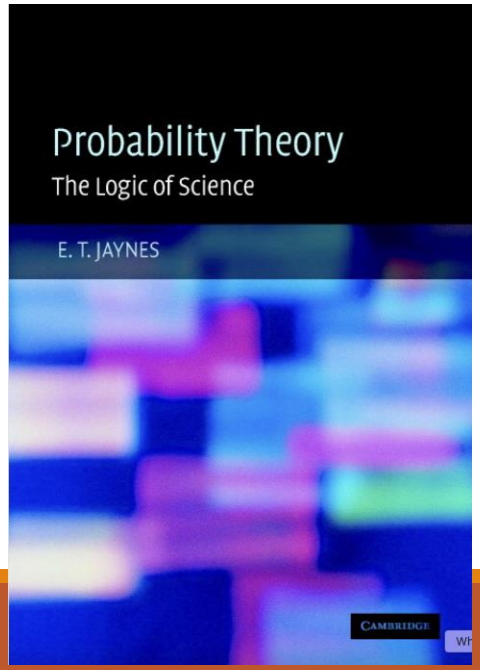
RAFAEL S. DE SOUZA

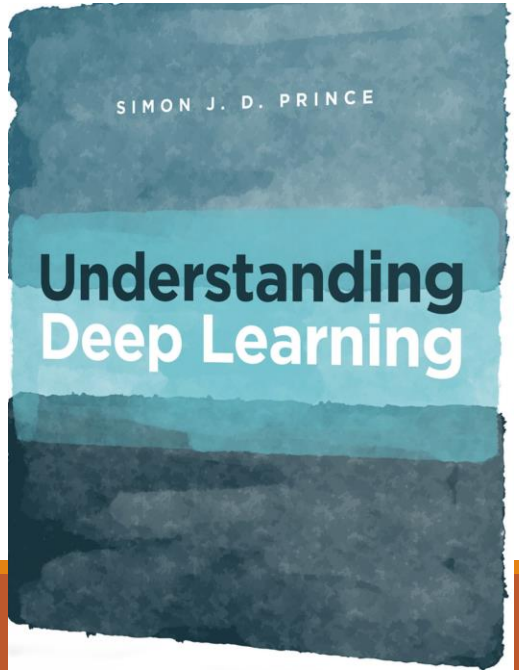


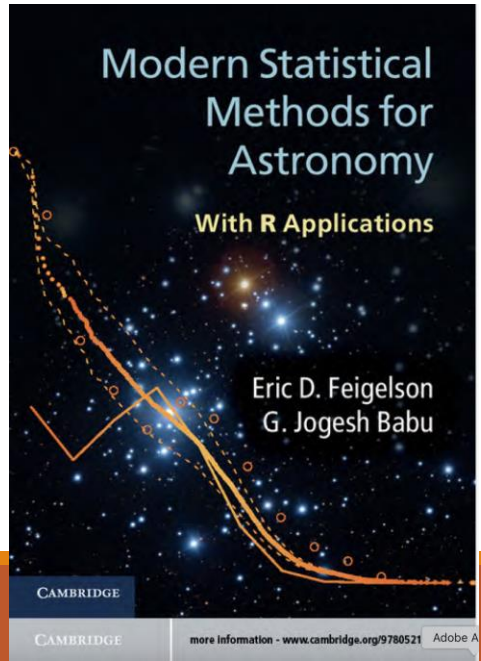

BAYESIAN
MODELS
for Astrophysical Data
Using R, JAGS, Python, and Stan
Joseph M. Hilbe, Rafael S. de Souza,
and Emille E. O. Ishida

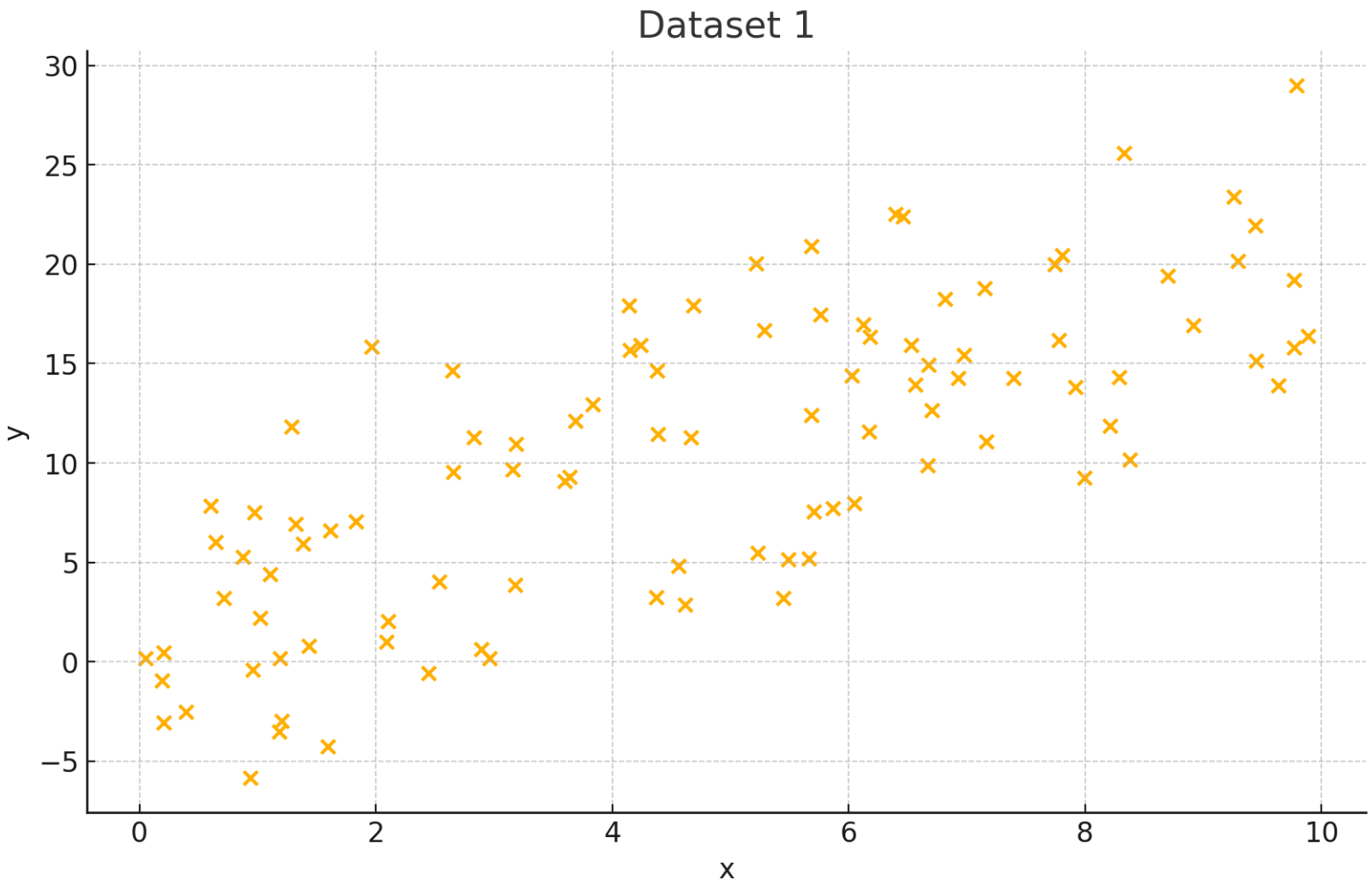

grokking
Machine Learning
Luis G. Serrano
Foreword by Sebastian Thrun

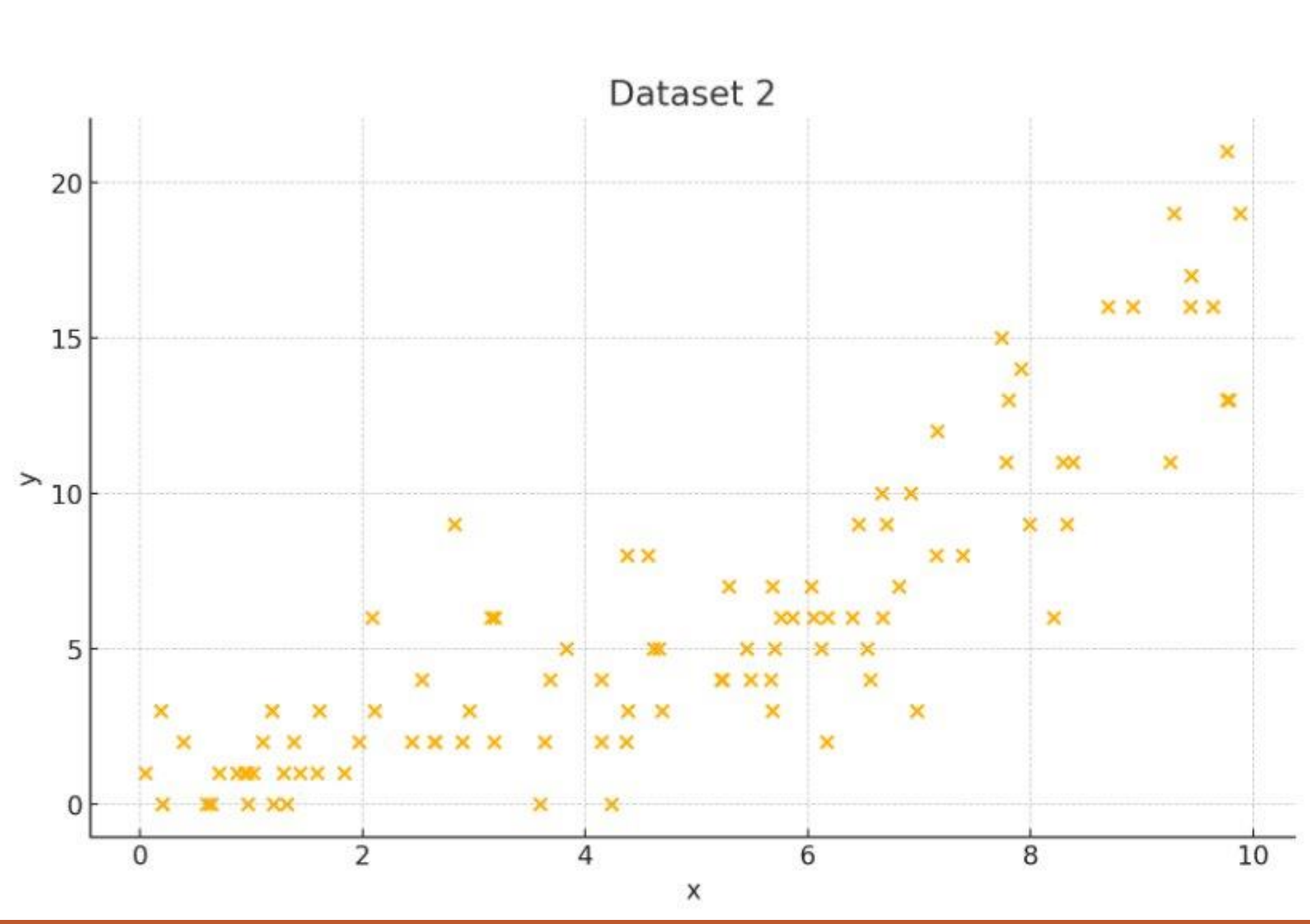

Springer Texts in Statistics
Gareth James
Daniela Witten
Trevor Hastie
Robert Tibshirani
An Introduction to Statistical Learning
with Applications in R
Second Edition
Springer

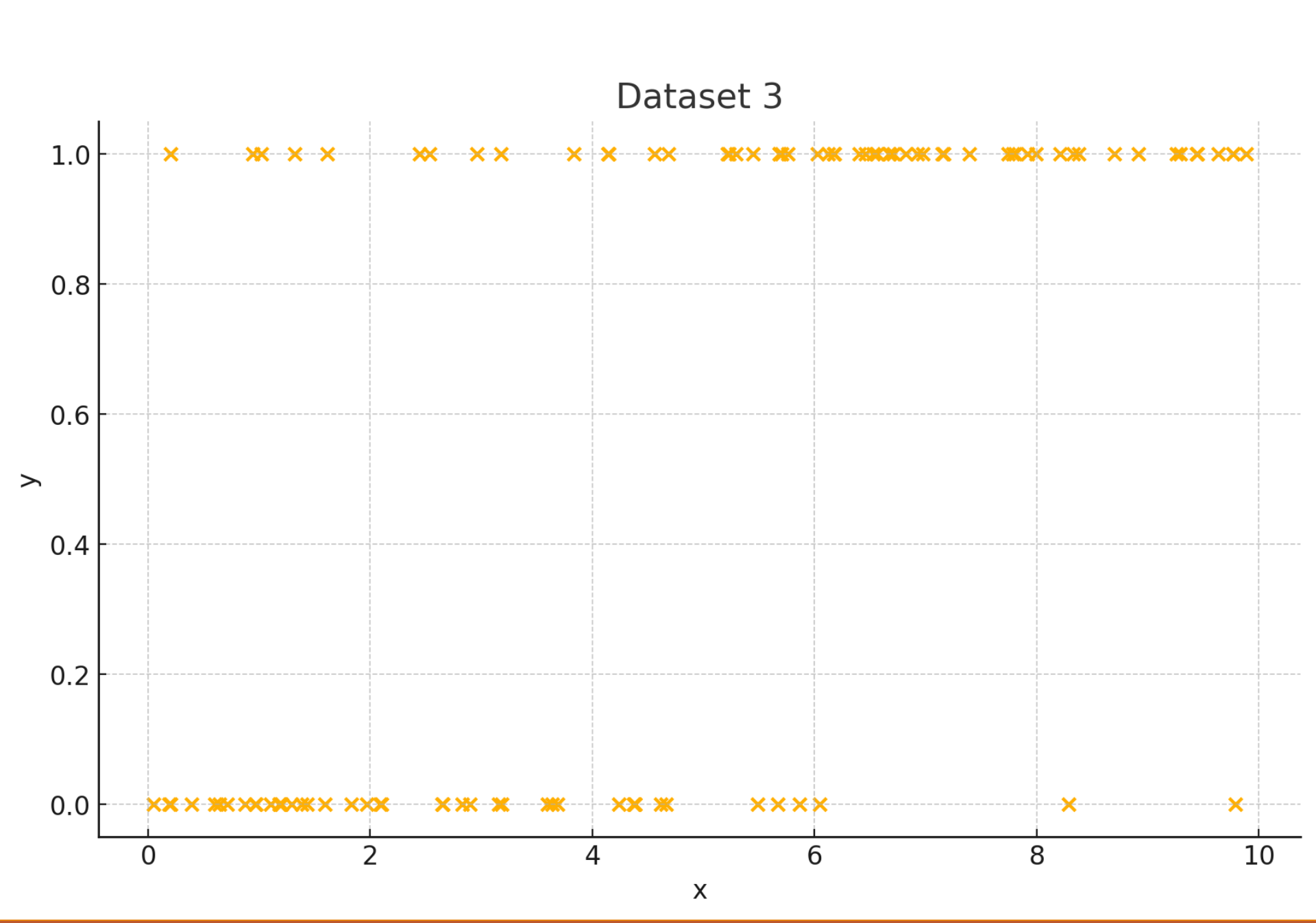

Probability Theory
The Logic of Science
E. T. JAYNES
CAMBRIDGE

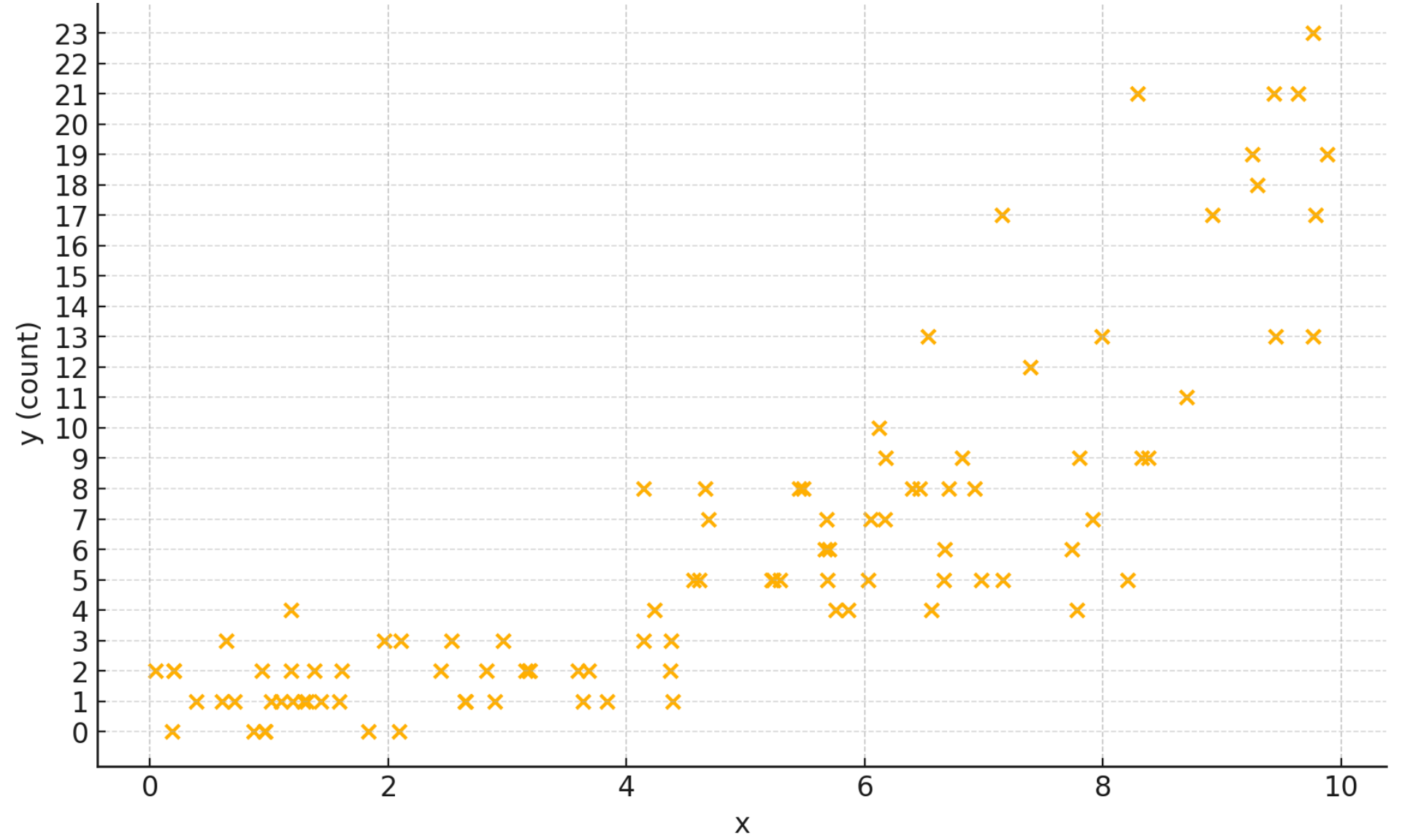

SIMON J. D. PRINCE
Understanding
Deep Learning


Modern Statistical Methods for Astronomy
With R Applications
Eric D. Feigelson
G. Jogesh Babu
CAMBRIDGE
CAMBRIDGE
more information - www.cambridge.org/9780521

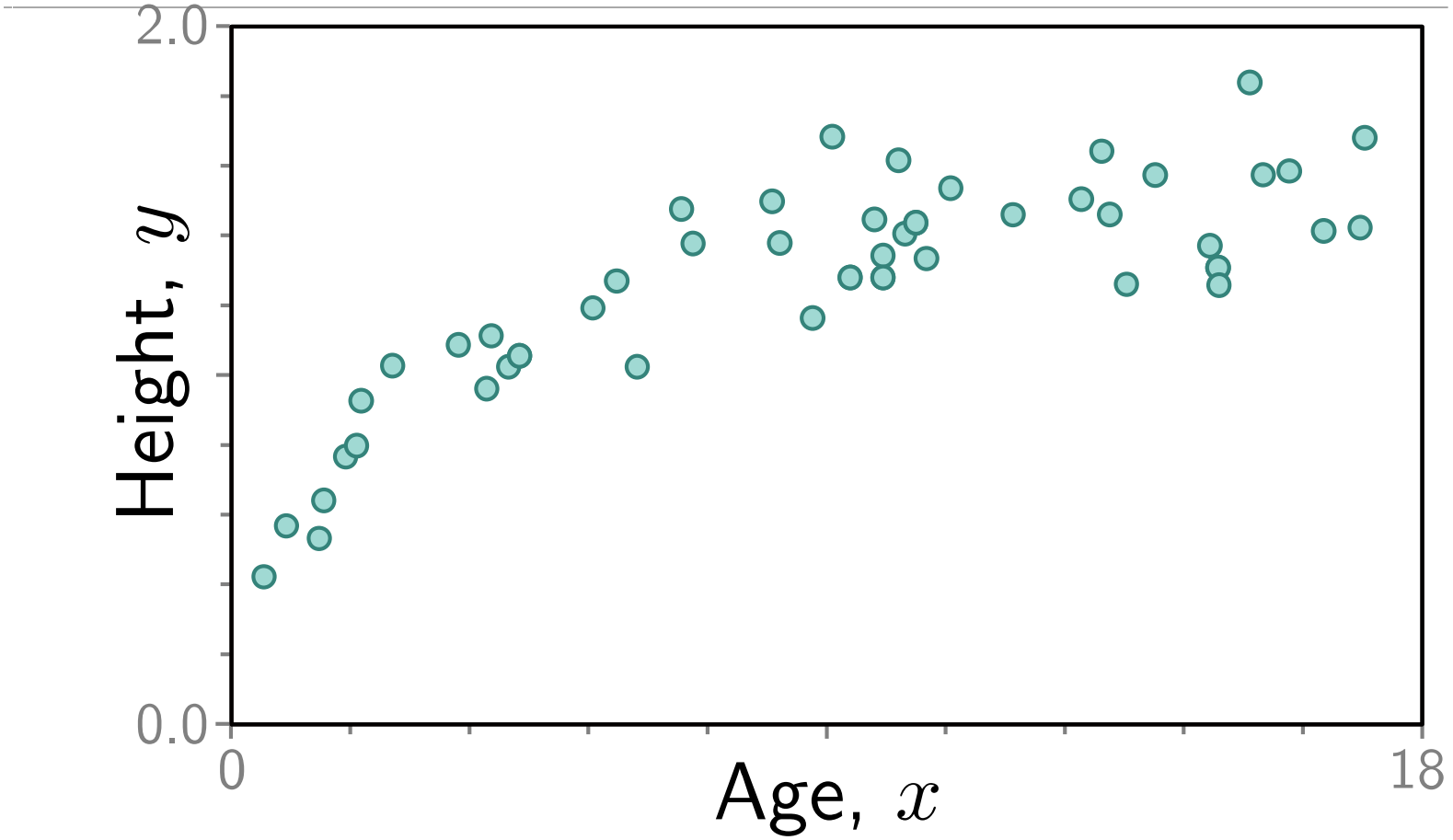

Dataset 1
30
25
20
15
10
5
0
−5
y
0
2
4
6
8
10
x

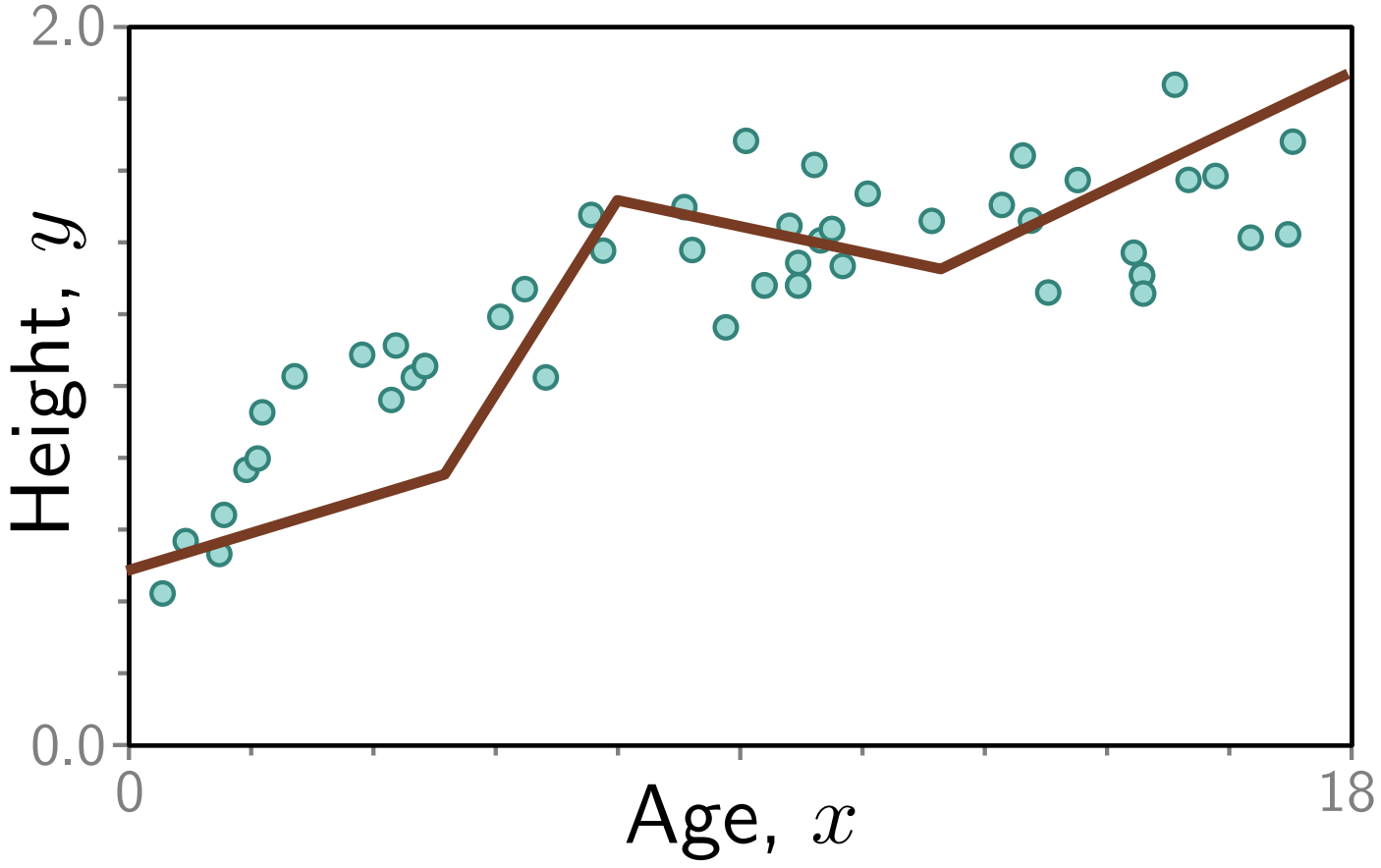

Dataset 2
20
15
10
5
0
y
0
2
4
6
8
10
x

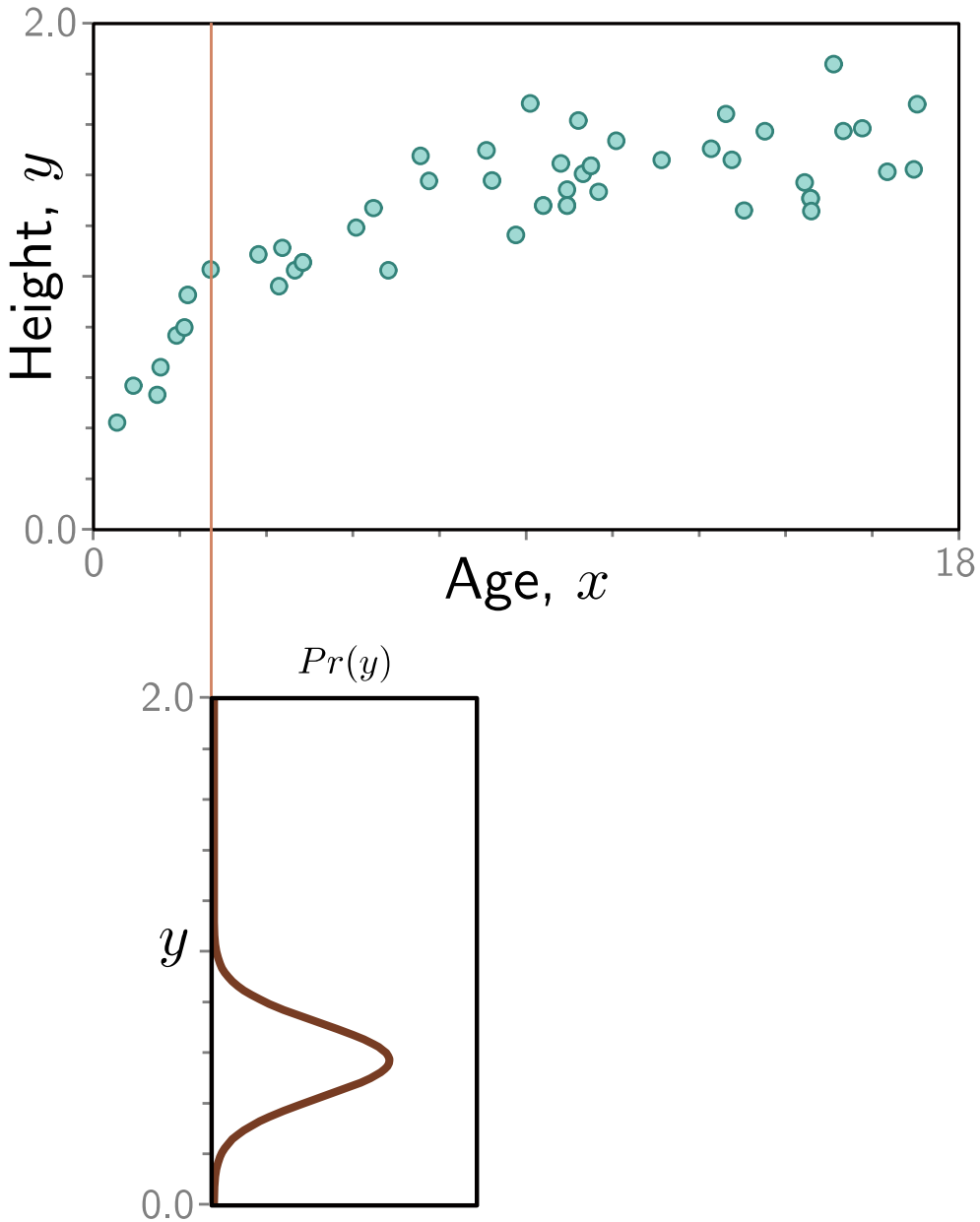

Dataset 3
1.0
0.8
0.6
0.4
0.2
0.0
y
0
2
4
6
8
10
x

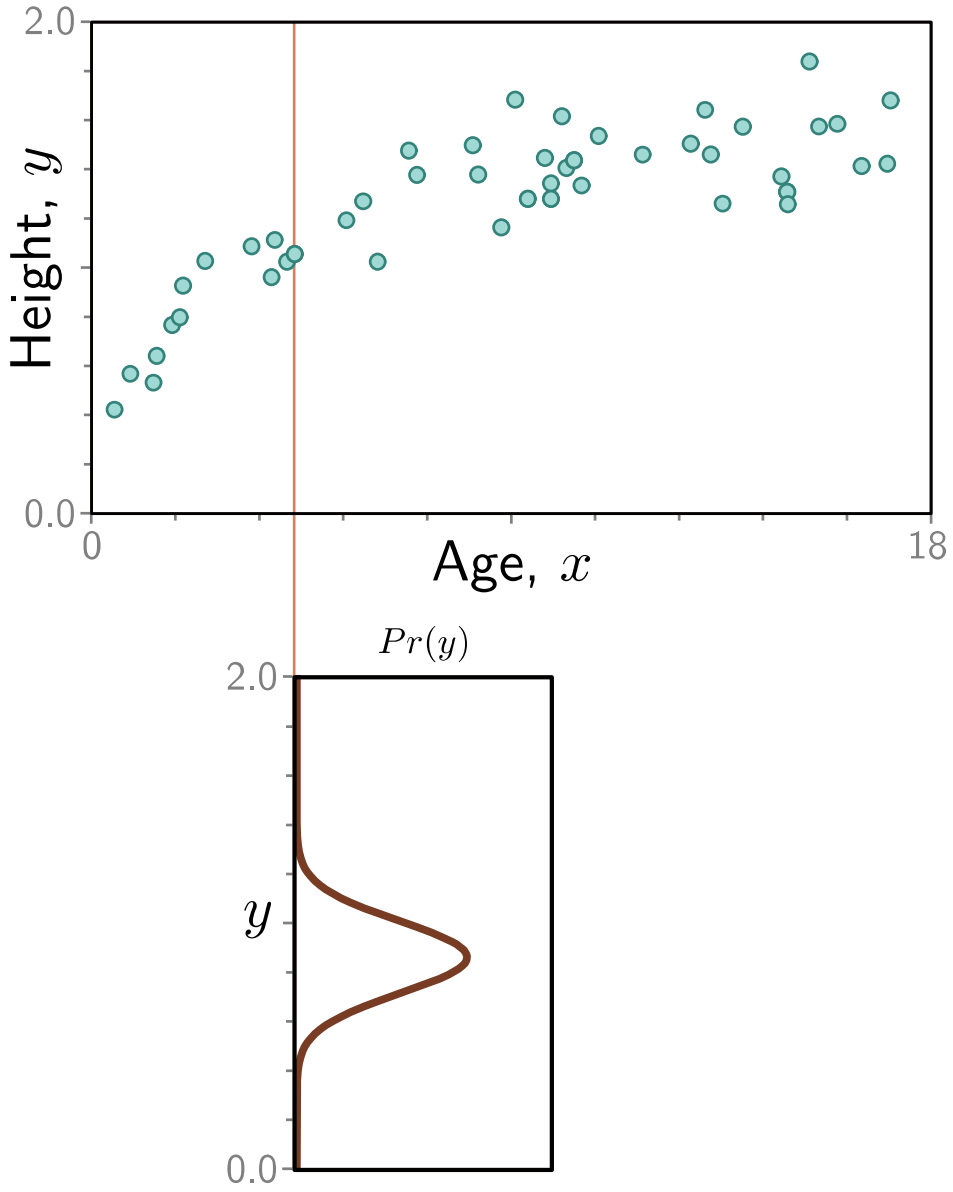

y (count)
x
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
0
2
4
6
8
10

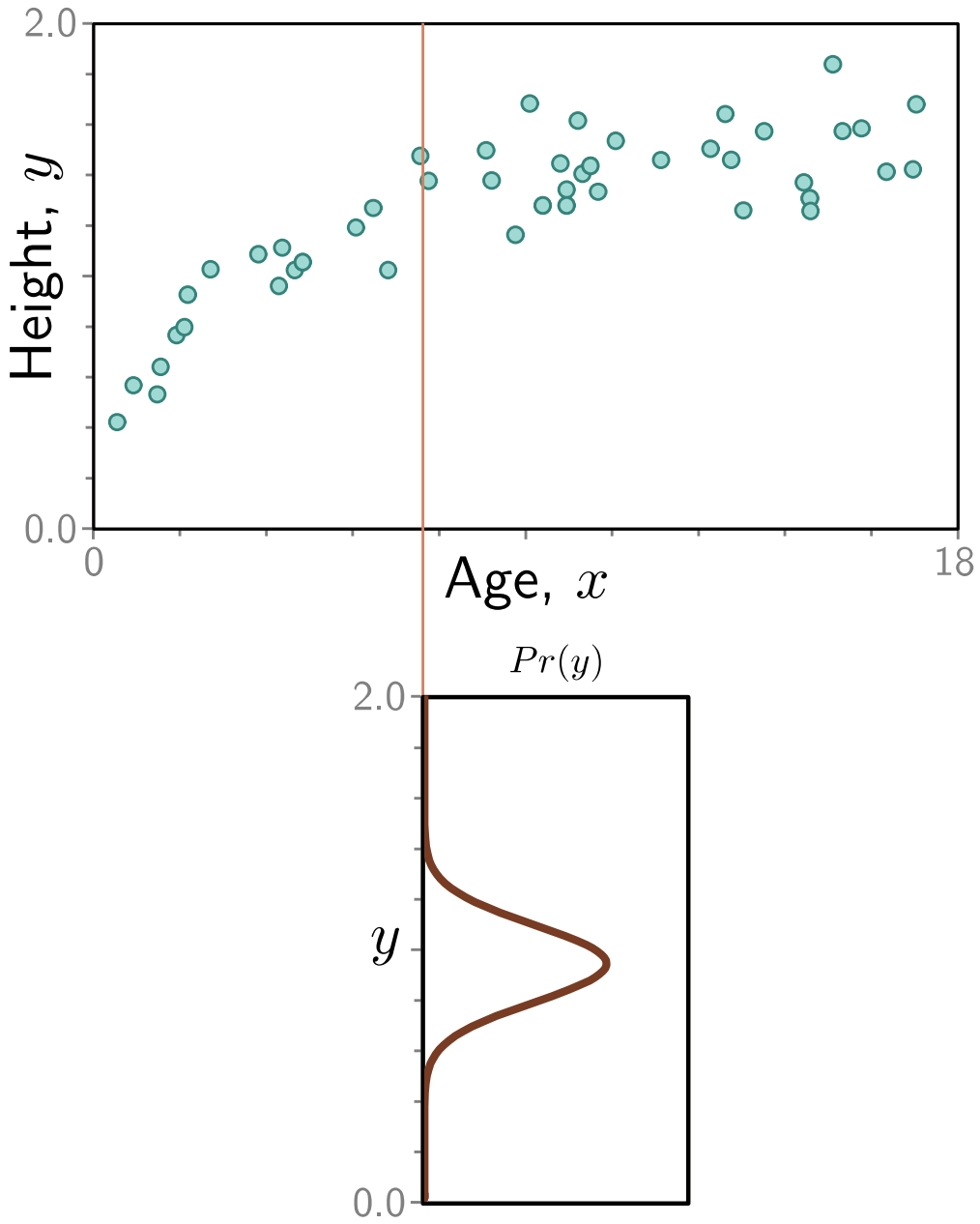

1.00
0.75
0.50
0.25
0.00
y
0
2
4
6
x1

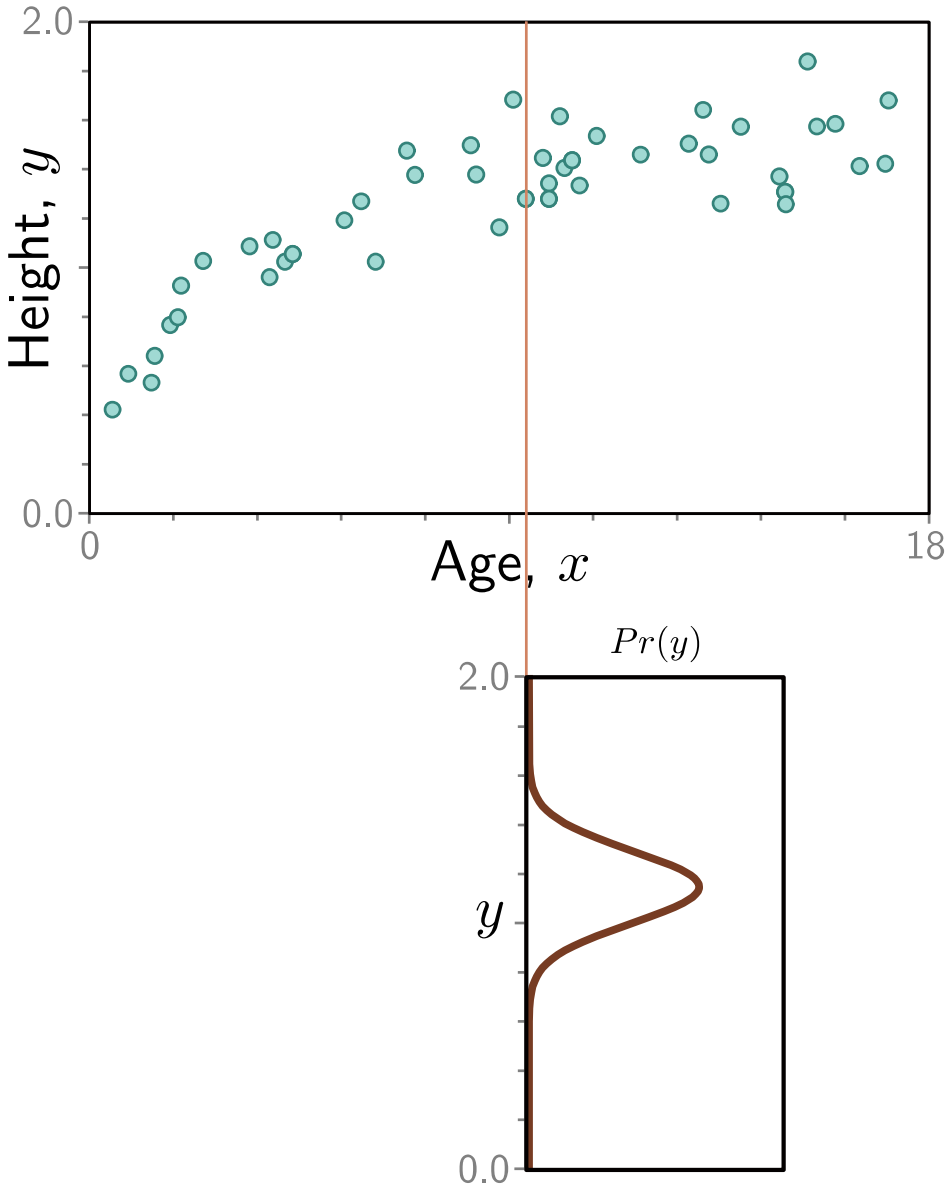

2.0
0.0
Height, y
0
18
Age, x

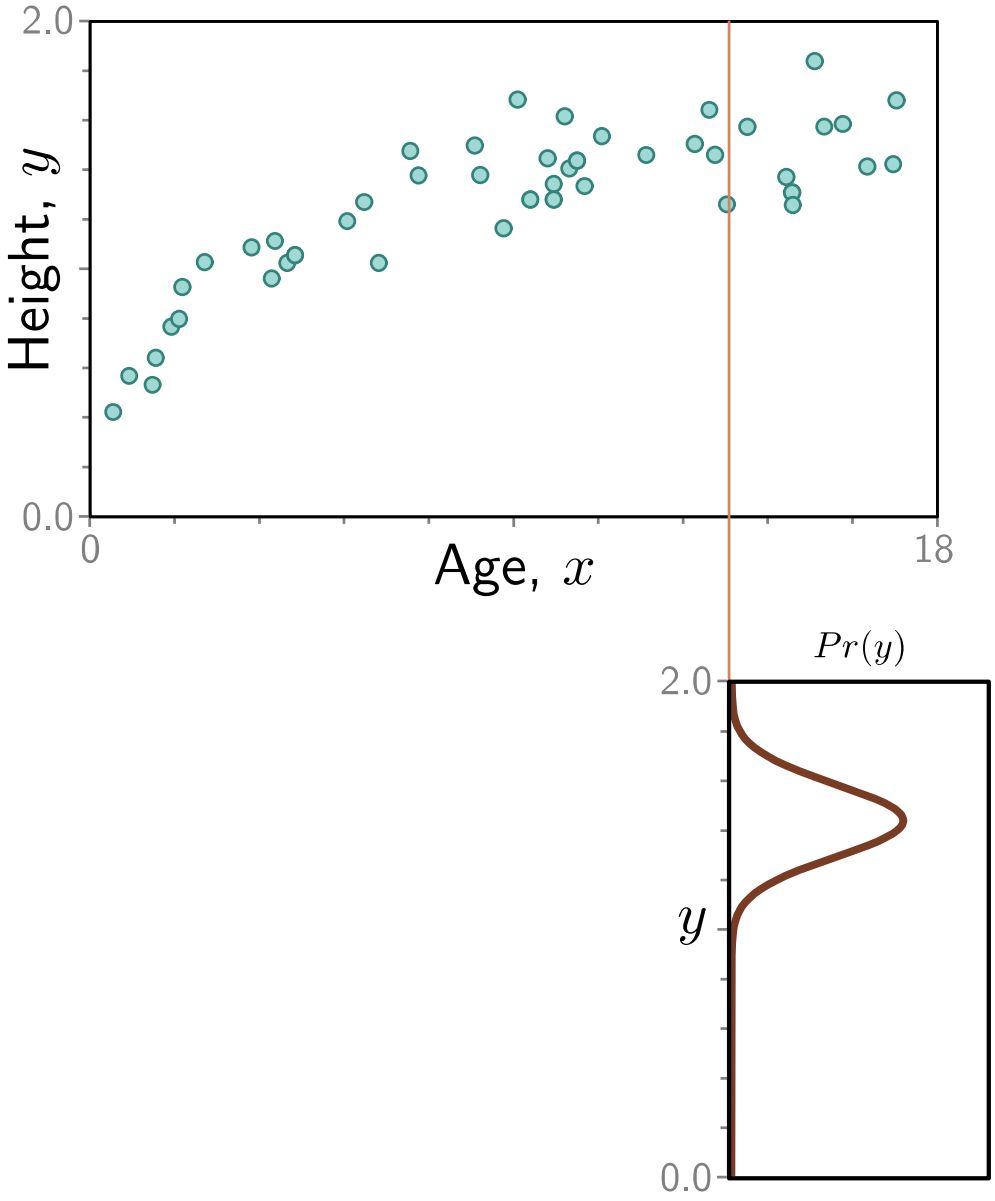

2.0
0.0
Height, y
0
18
Age, x

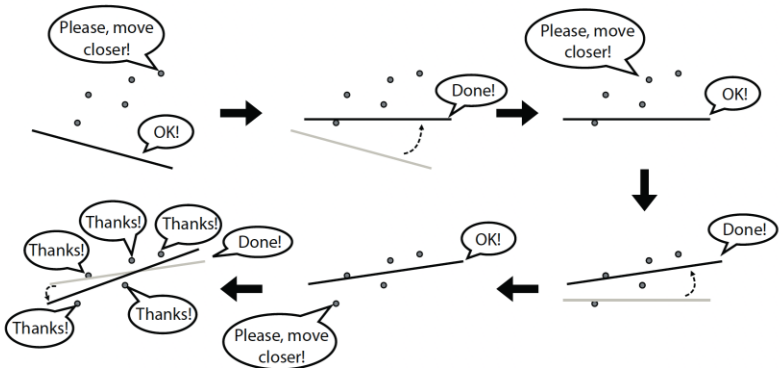

2.0
0.0
Height, y
0
18
Age, x
$Pr(y)$
2.0
y
0.0


2.0
0.0
Height, y
0
18
Age, x
$Pr(y)$
2.0
y
0.0


2.0
0.0
Height, y
0
18
Age, x
$Pr(y)$
2.0
y
0.0


2.0
0.0
Height, y
0
18
Age, x
$Pr(y)$
2.0
y
0.0


2.0
0.0
Height, y
0
18
Age, x
$Pr(y)$
2.0
y
0.0



Pseudocode for the linear regression algorithm (geometric)

Inputs: A dataset of points in the plane

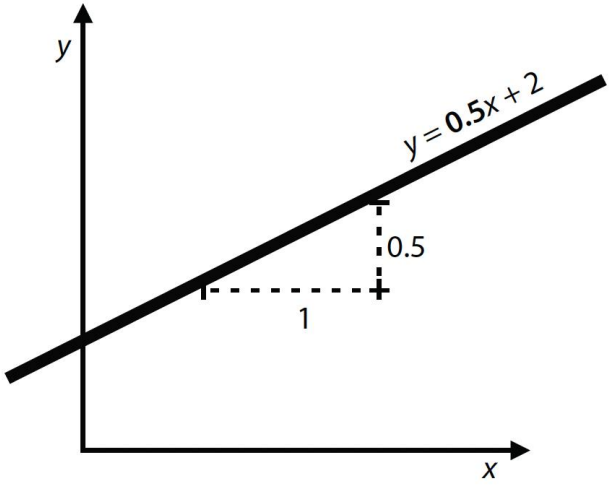
Outputs: A line that passes close to the points

Procedure:

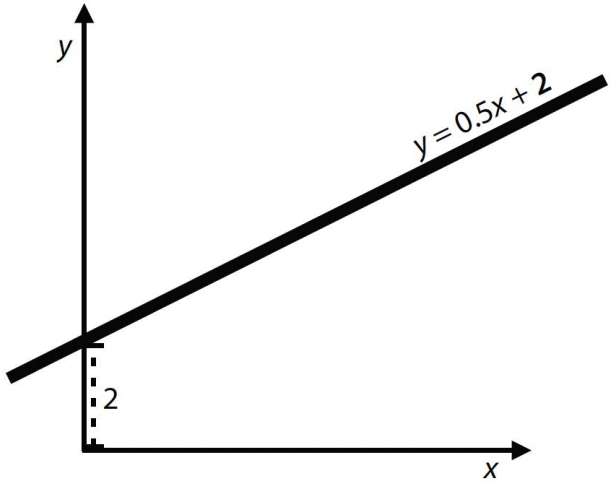
- Pick a random line.
- Repeat many times:
  - Pick a random data point.
  - Move the line a little closer to that point.
- **Return** the line you've obtained.

How to get the computer to draw this line:
The linear regression algorithm

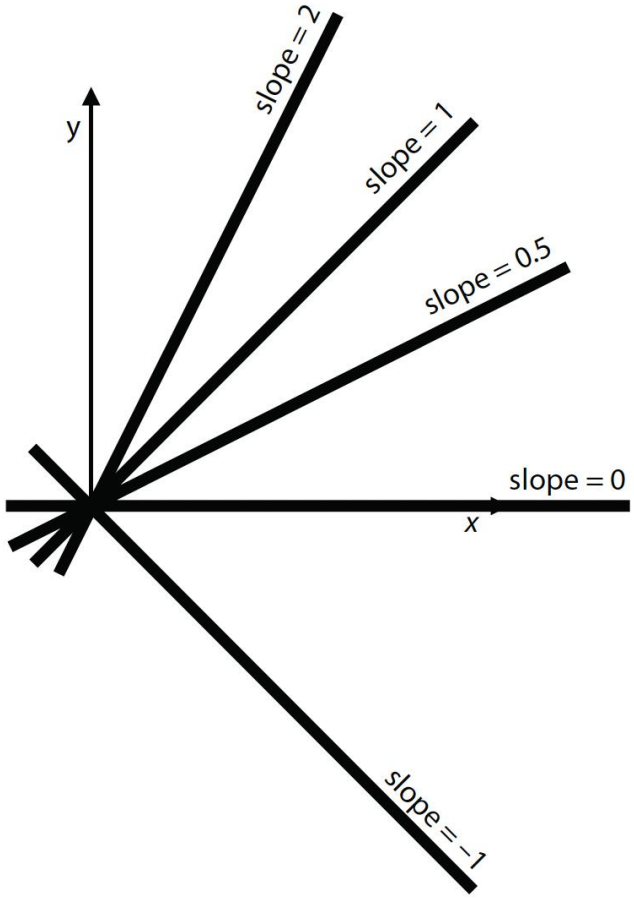
$$y = 0.5x + 2$$

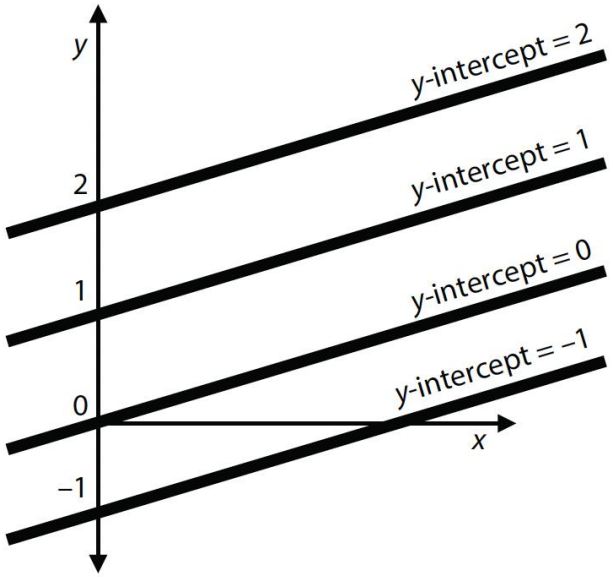


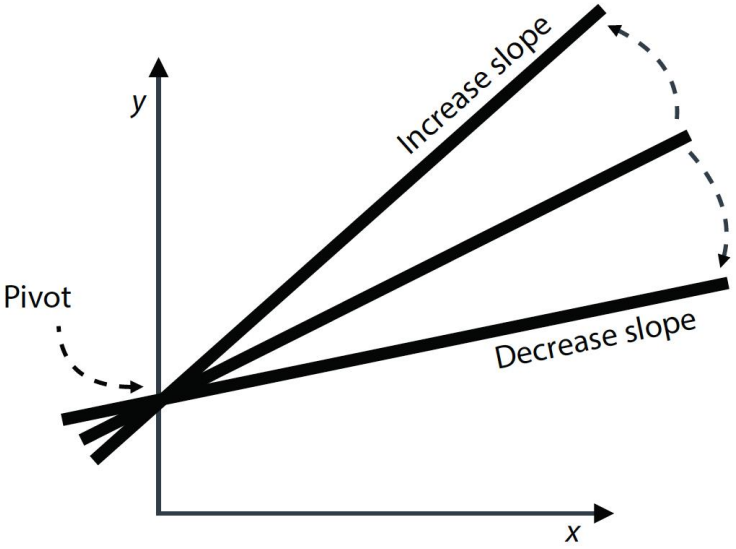
Slope = 0.5



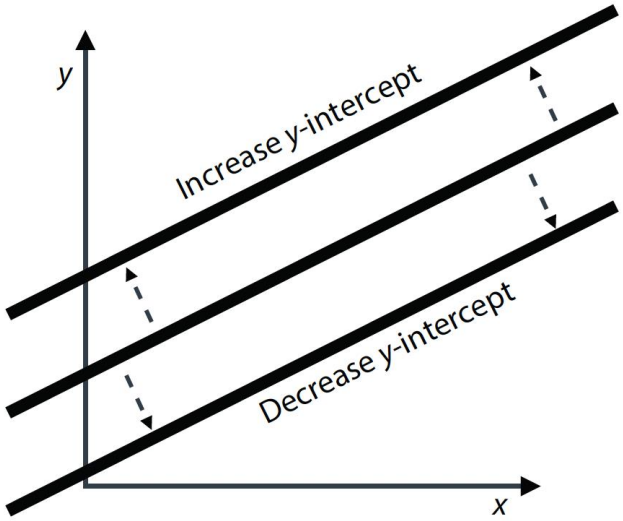
***y*-intercept = 2**


y
slope = 2
slope = 1
slope = 0.5
slope = 0
x
slope = –1

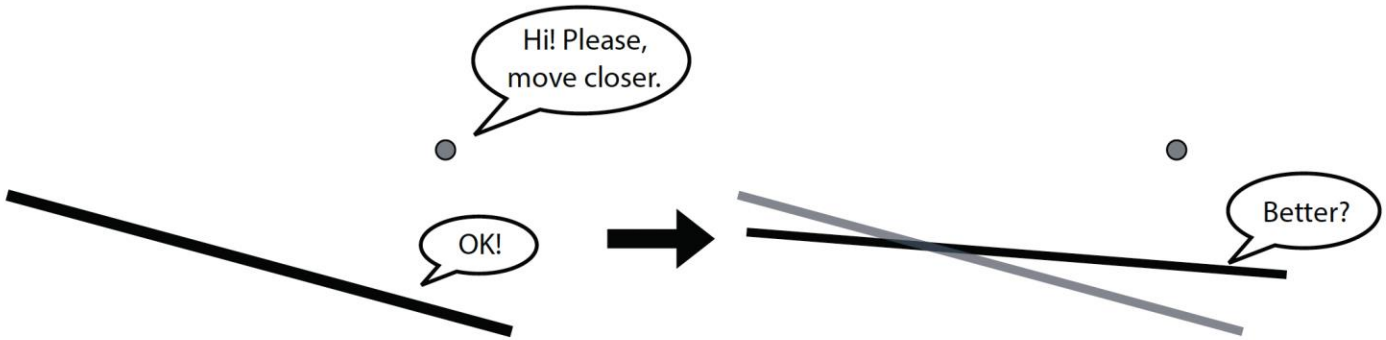

y
2
1
0
–1
y-intercept = 2
y-intercept = 1
y-intercept = 0
y-intercept = –1
x

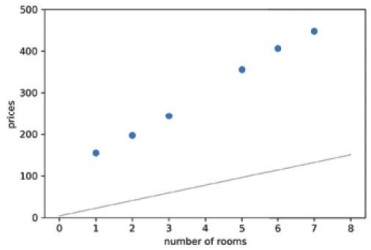


Rotate clockwise and counterclockwise

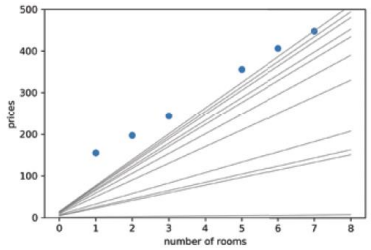


Translate up and down

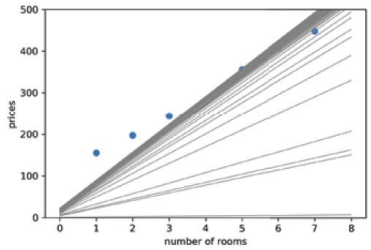

Hi! Please,
move closer.
OK!
Better?



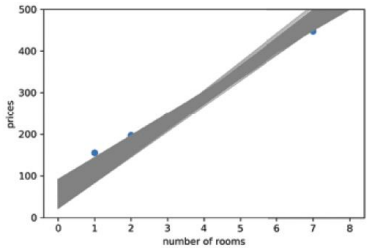
Starting point



Epochs 1–10

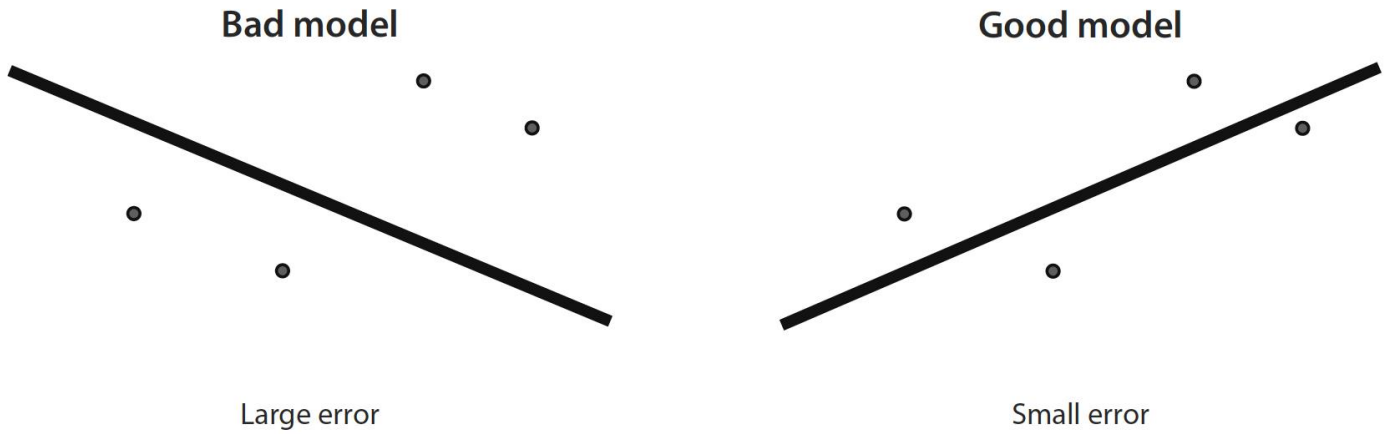


Epochs 1–50

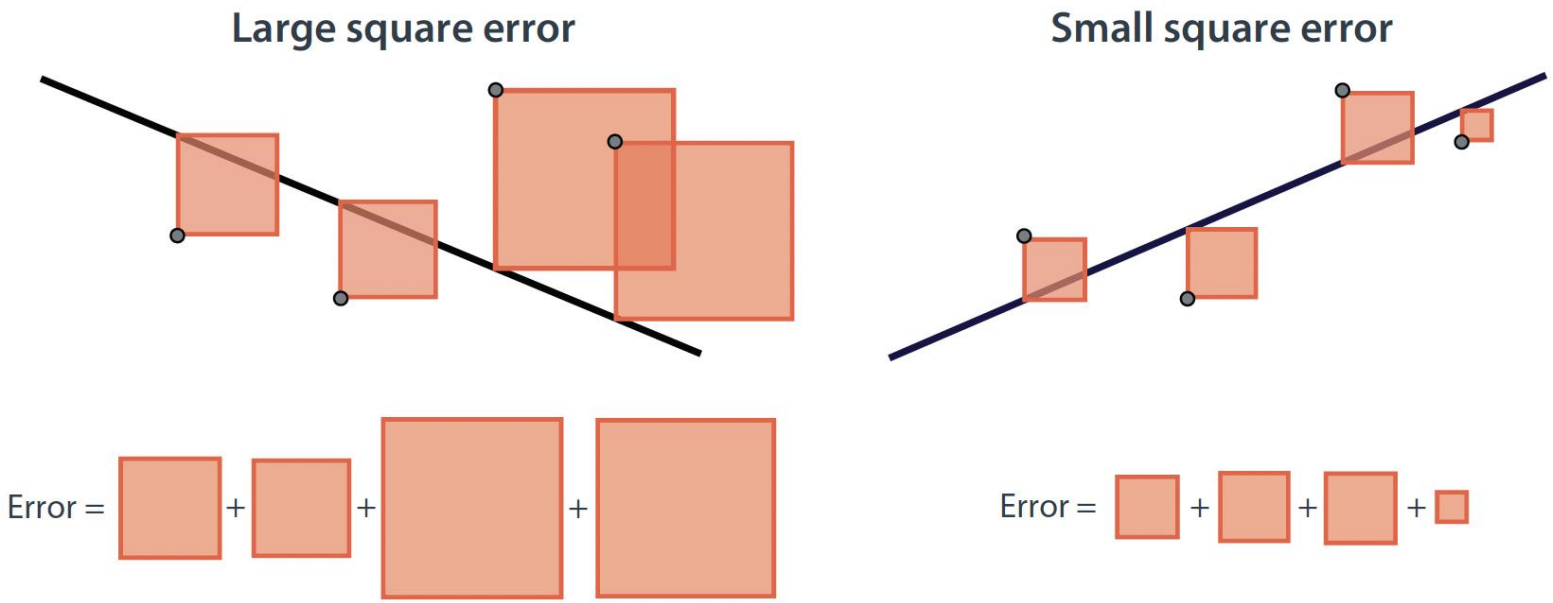


Epochs 51–10,000

How do we measure our results? The error function



The square error: A metric that tells us how good our model is by adding squares of distances



Loss function

Training dataset of I pairs of input/output examples:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{I}$$

Loss function or cost function measures how bad model is:

$$L\Big[\underbrace{\boldsymbol{\phi}, \mathrm{f}[\mathbf{x}, \boldsymbol{\phi}]}_{\text{model}}, \underbrace{\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{I}}_{\text{train data}}\Big]$$

Loss function

Training dataset of I pairs of input/output examples:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{I}$$

Loss function or cost function measures how bad model is:

or for short:

$$L[\boldsymbol{\phi}]$$

Returns a scalar that is smaller when model maps inputs to outputs better

Training

Loss function:

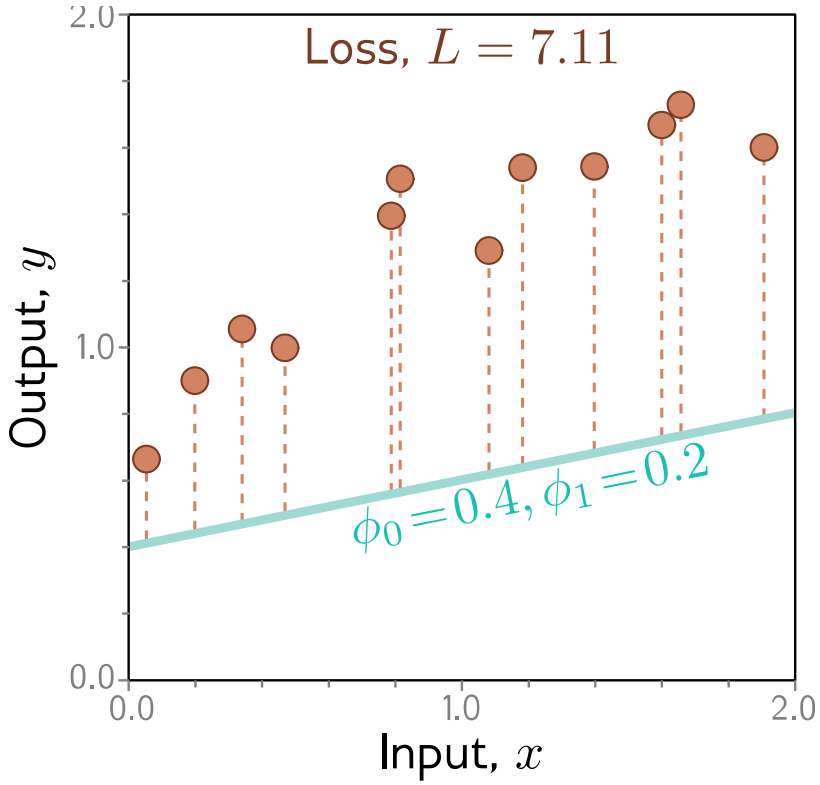
$$L\left[\boldsymbol{\phi}\right]$$

Returns a scalar that is smaller when model maps inputs to outputs better

Find the parameters that minimize the loss:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\Big[\mathrm{L}\left[\boldsymbol{\phi}\right]\Big]$$

Example: 1D Linear regression loss function



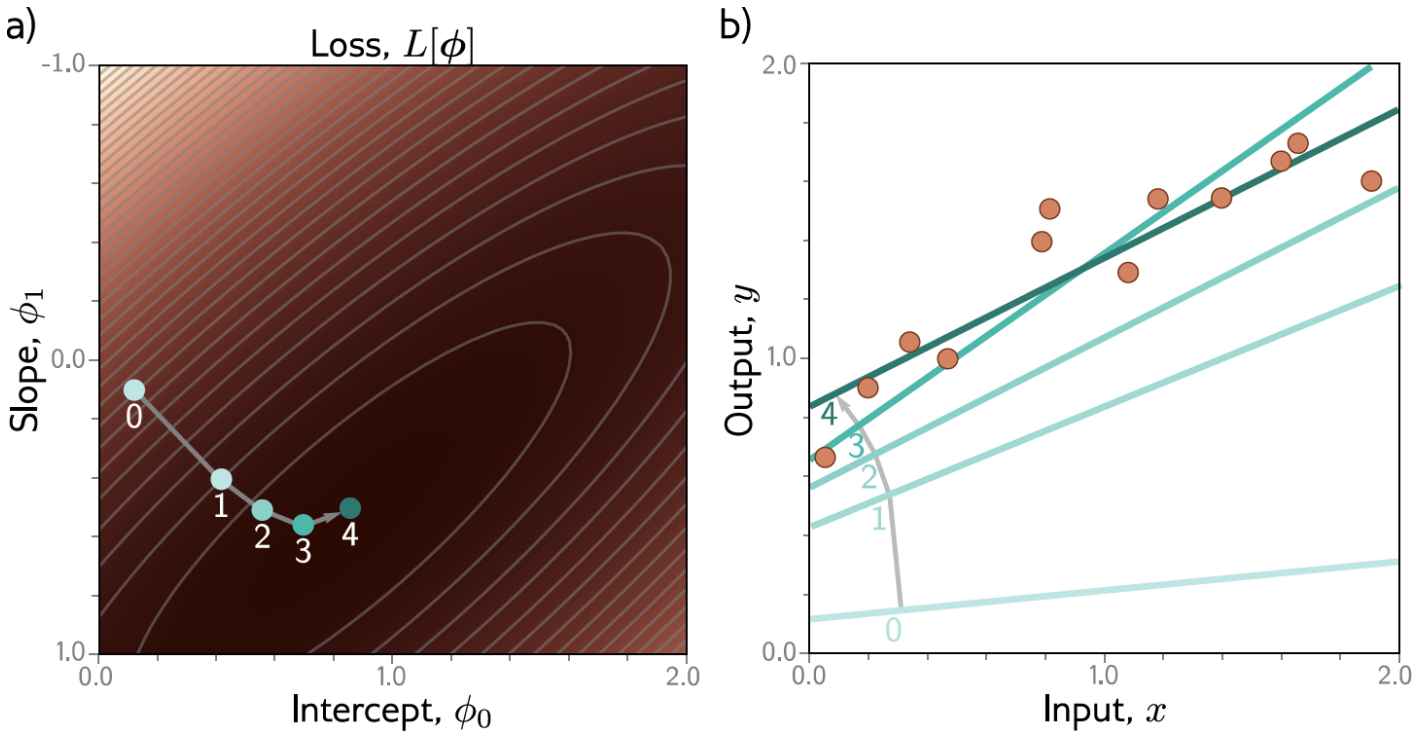
Loss function:

$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I} (\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i)^2$$

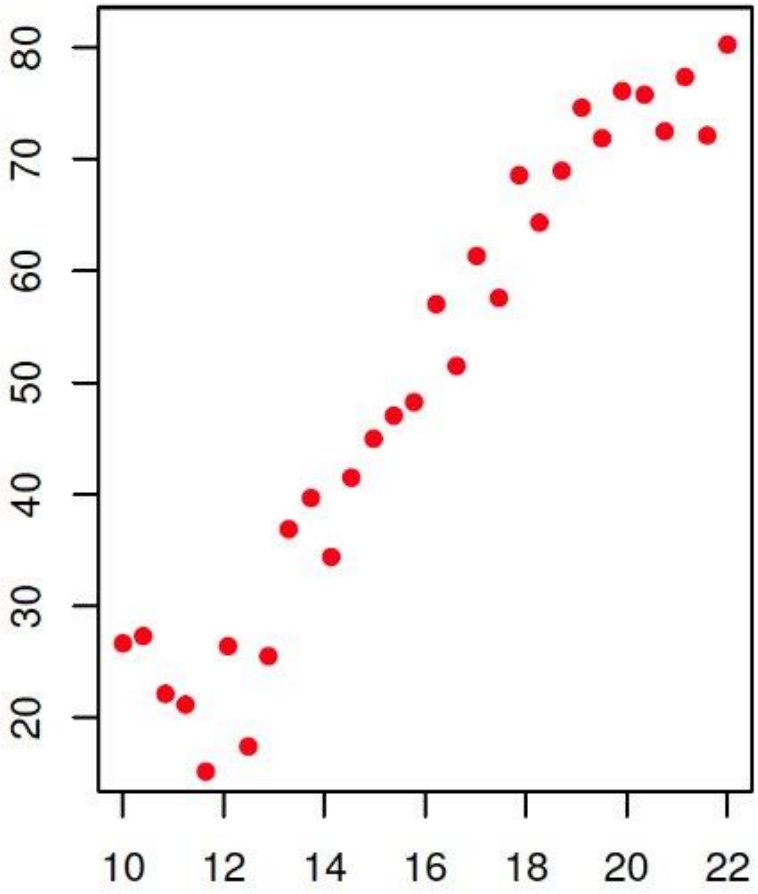
$$= \sum_{i=1}^{I} (\phi_0 + \phi_1 x_i - y_i)^2$$

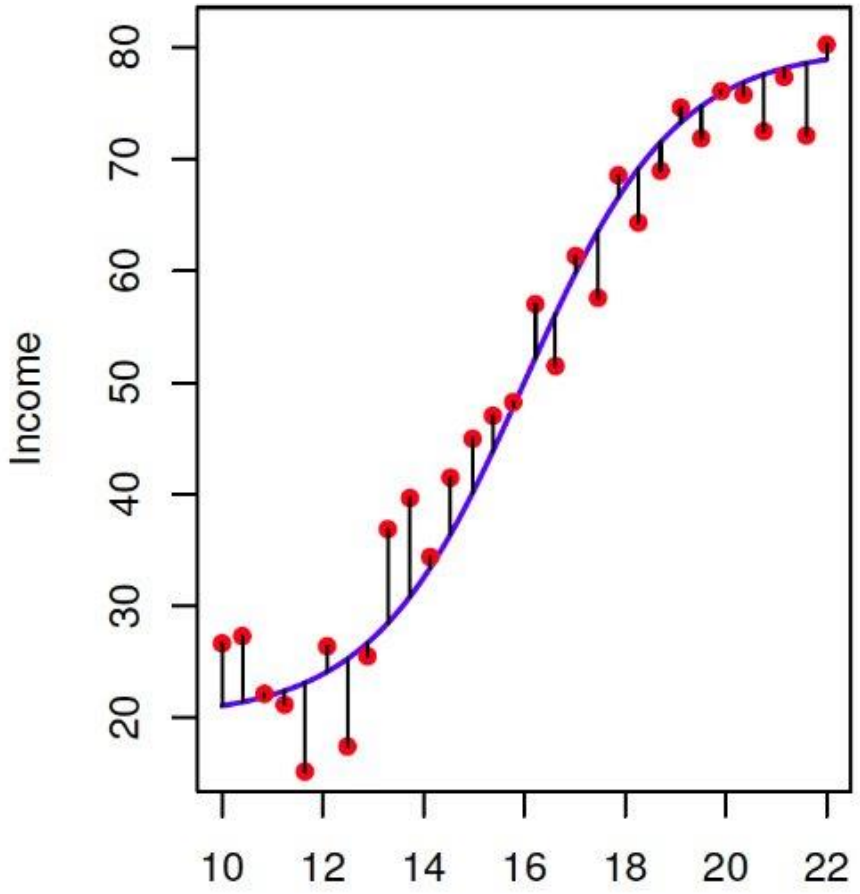
"Least squares loss function"

Example: 1D Linear regression training



This technique is known as gradient descent

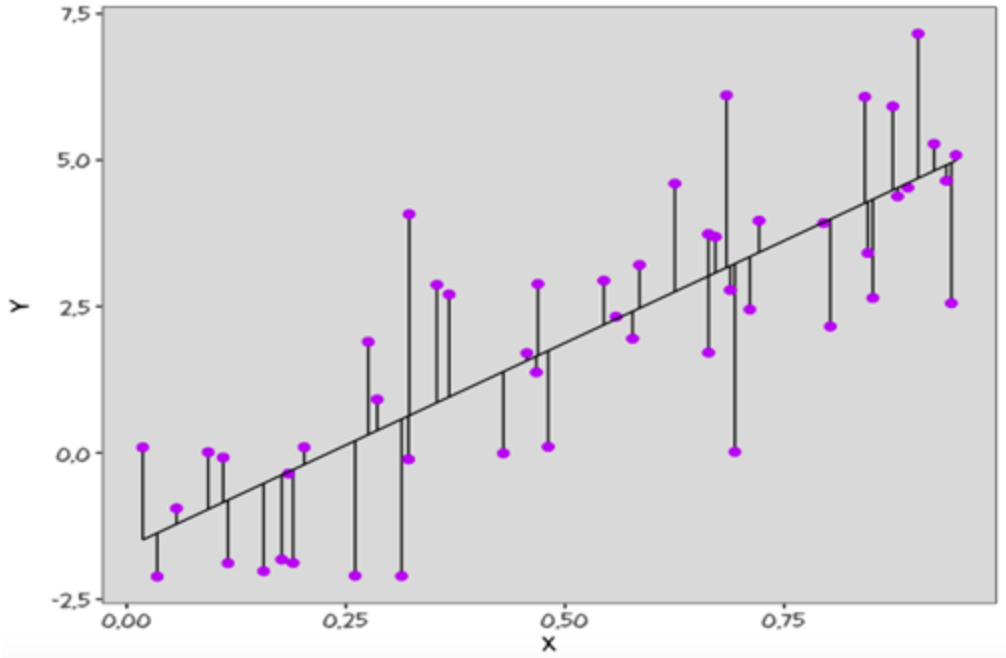

20
30
40
50
60
70
80
10
12
14
16
18
20
22

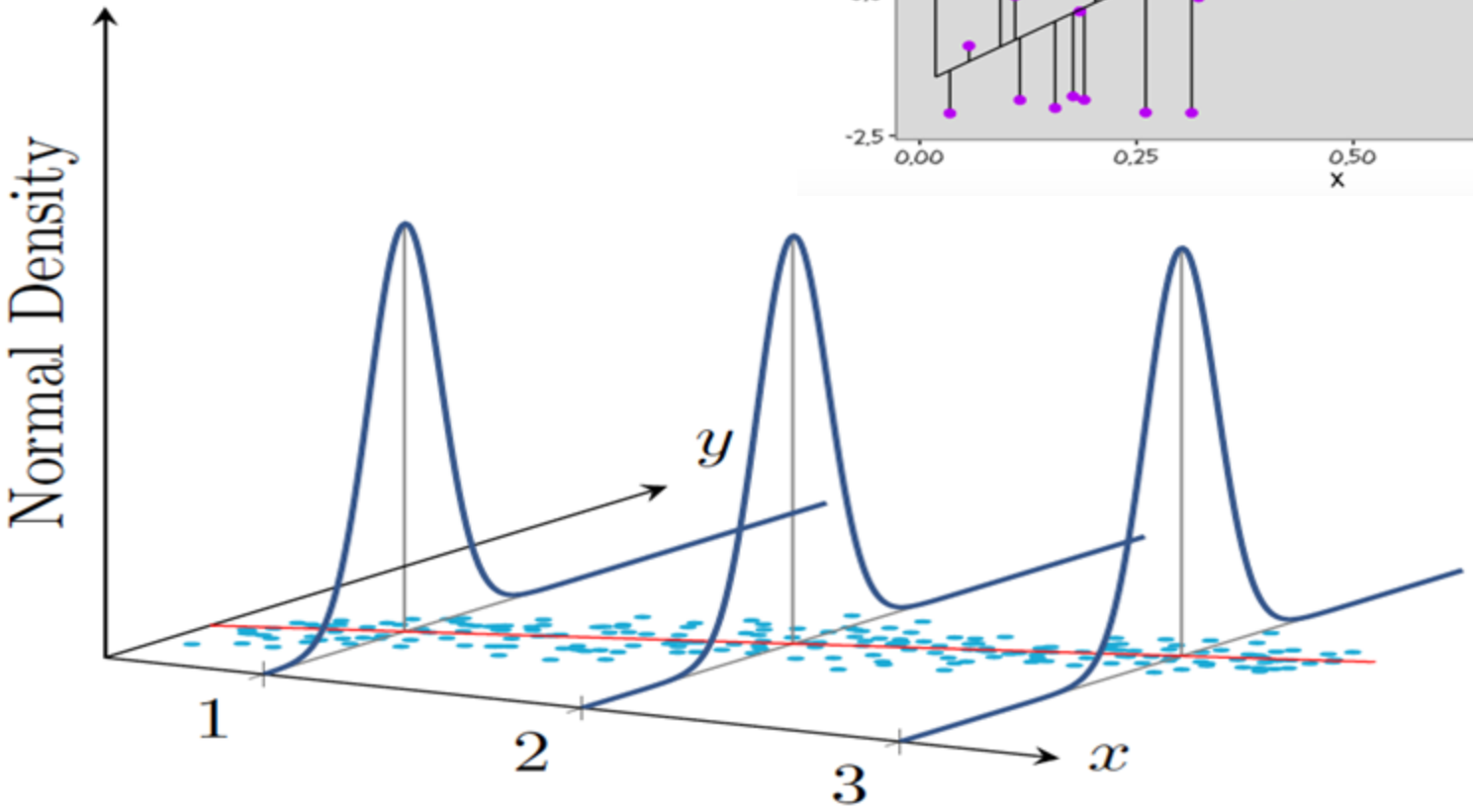

Income
20
30
40
50
60
70
80
10
12
14
16
18
20
22

Normal Linear Models

$$y = ax + b + \varepsilon$$

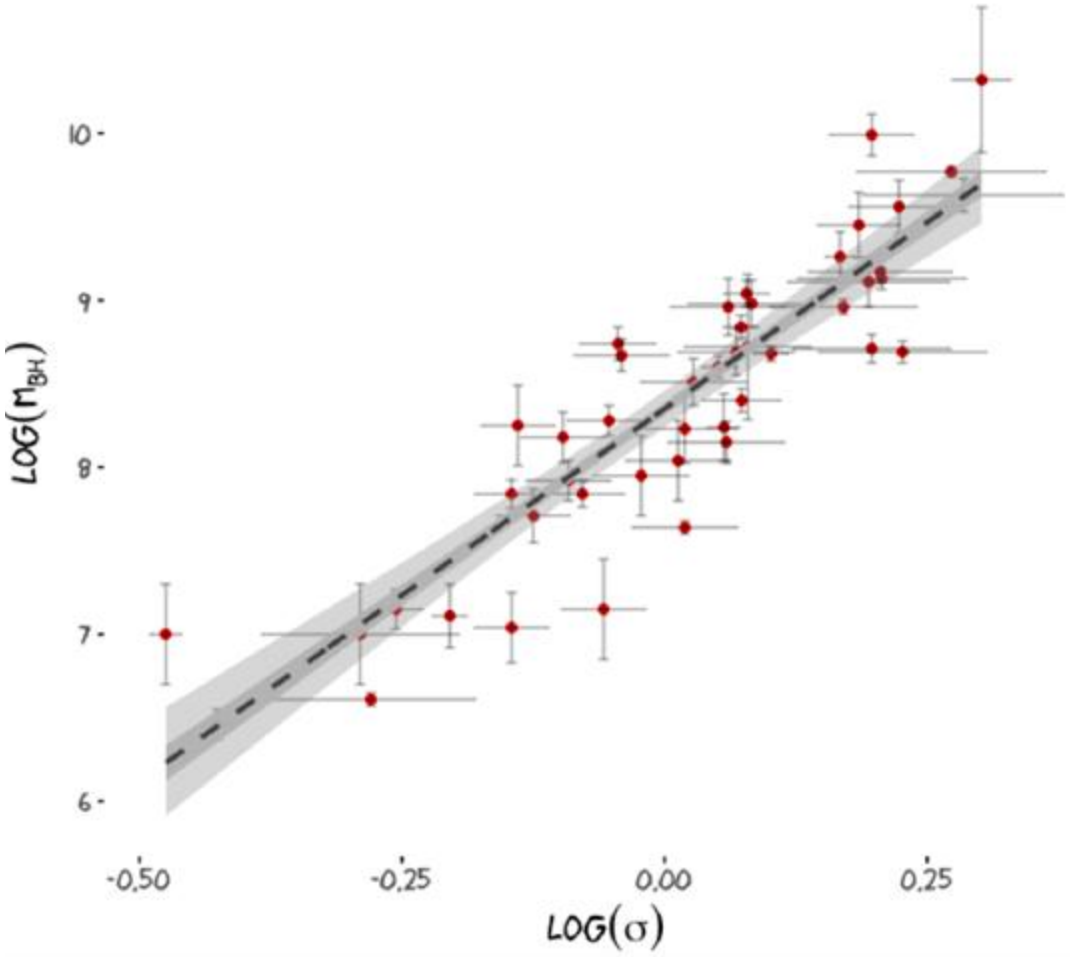
$$\varepsilon \sim N(0, \sigma^2)$$





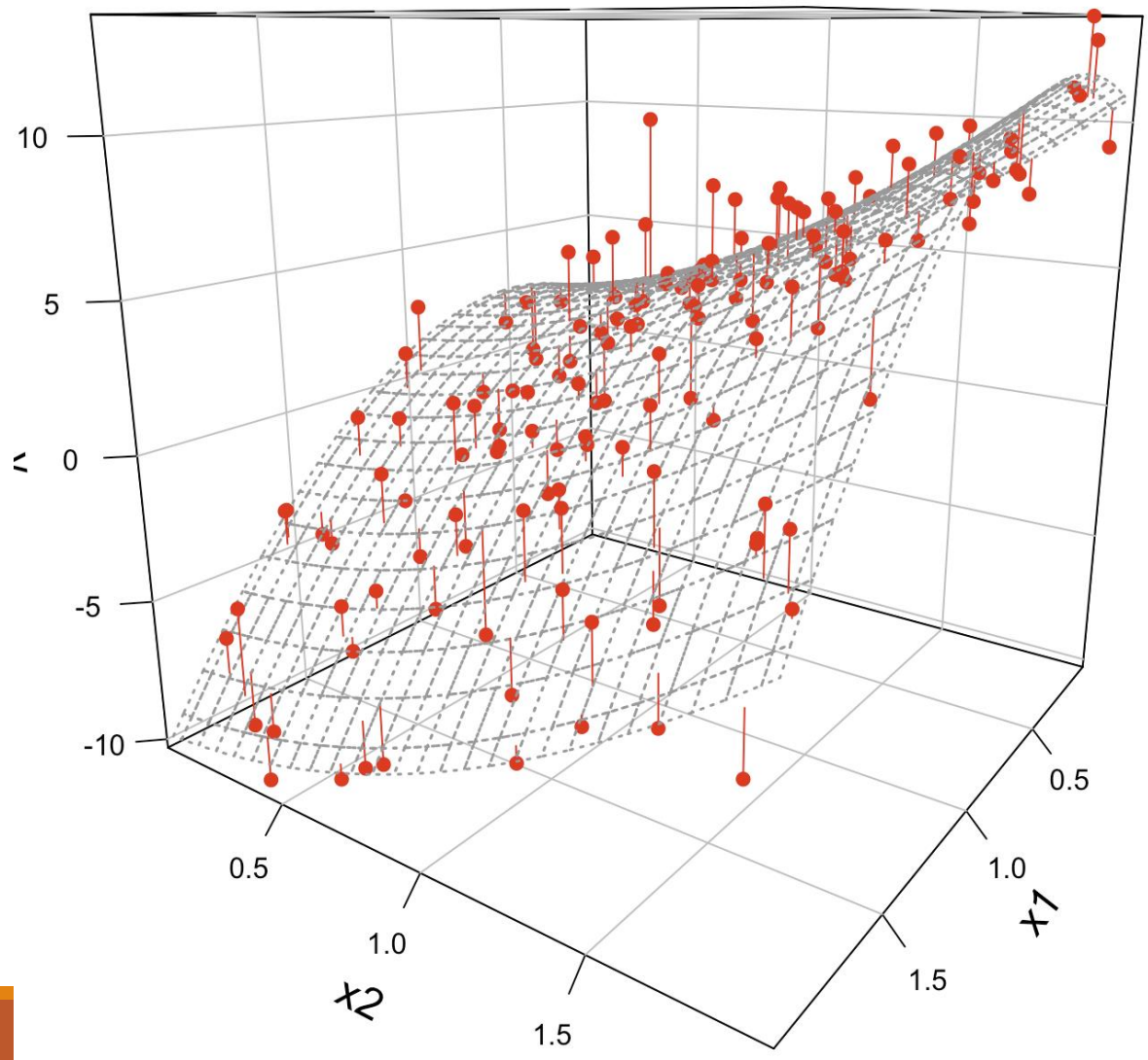
Errors-in-measurements: M-sigma relation

$$M_i \sim Normal(M_i^{\text{true}}, \varepsilon_{M;i}^2)$$
$$M_i^{\text{true}} \sim Normal(\mu_i, \varepsilon^2)$$
$$\mu_i = \alpha + \beta\sigma_i$$
$$\sigma_i \sim Normal(\sigma_i^{\text{true}}, \varepsilon_{\sigma;i}^2)$$
$$\sigma_i^{\text{true}} \sim Normal(0, 10^3)$$
$$\alpha \sim Normal(0, 10^3)$$
$$\beta \sim Normal(0, 10^3)$$
$$\varepsilon^2 \sim Gamma(10^{-3}, 10^{-3})$$
$$i = 1, \ldots, N$$



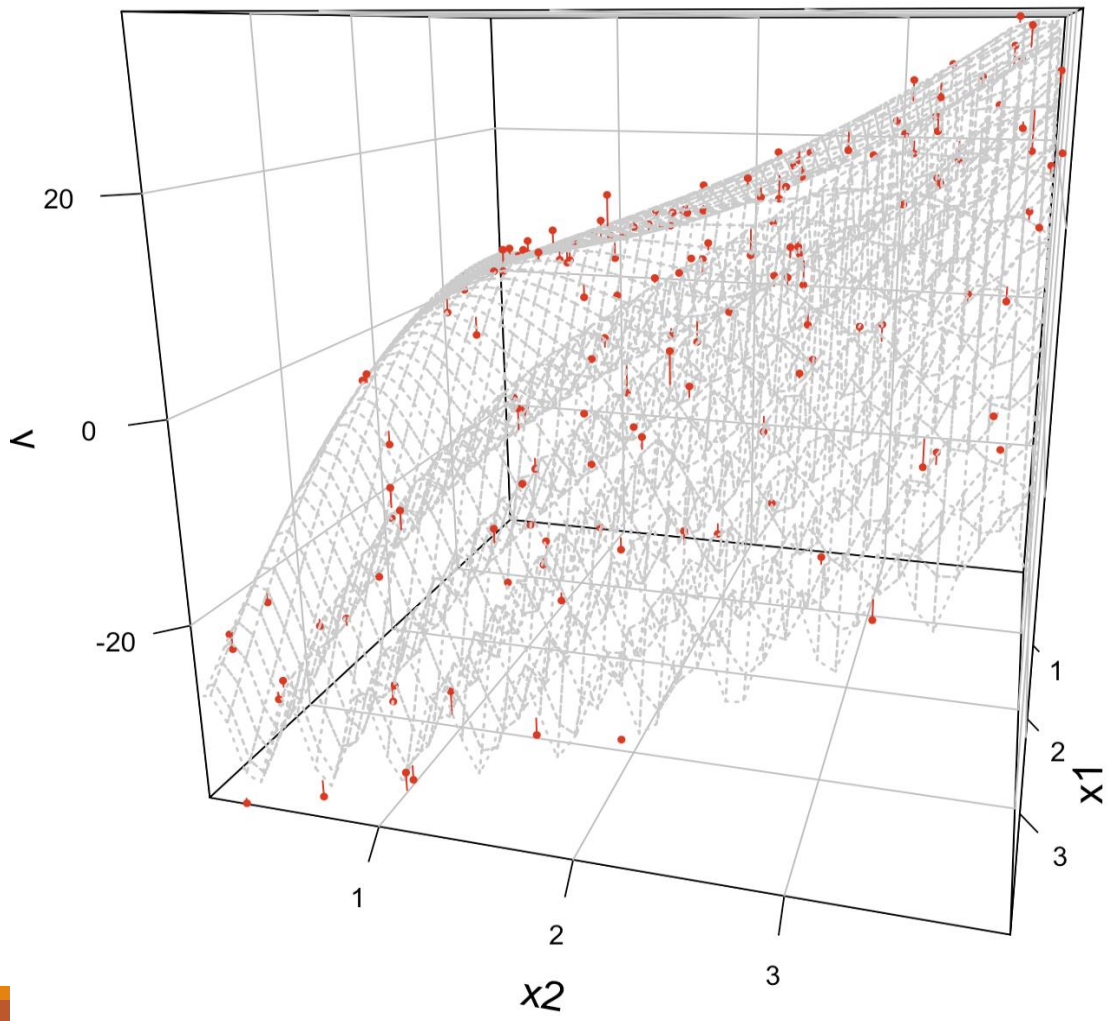
$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2 + \epsilon$$

$$\epsilon \sim N(0, \sigma^2)$$



$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2 + \beta_5 \sin(x_1^2 x_2) + \epsilon$$

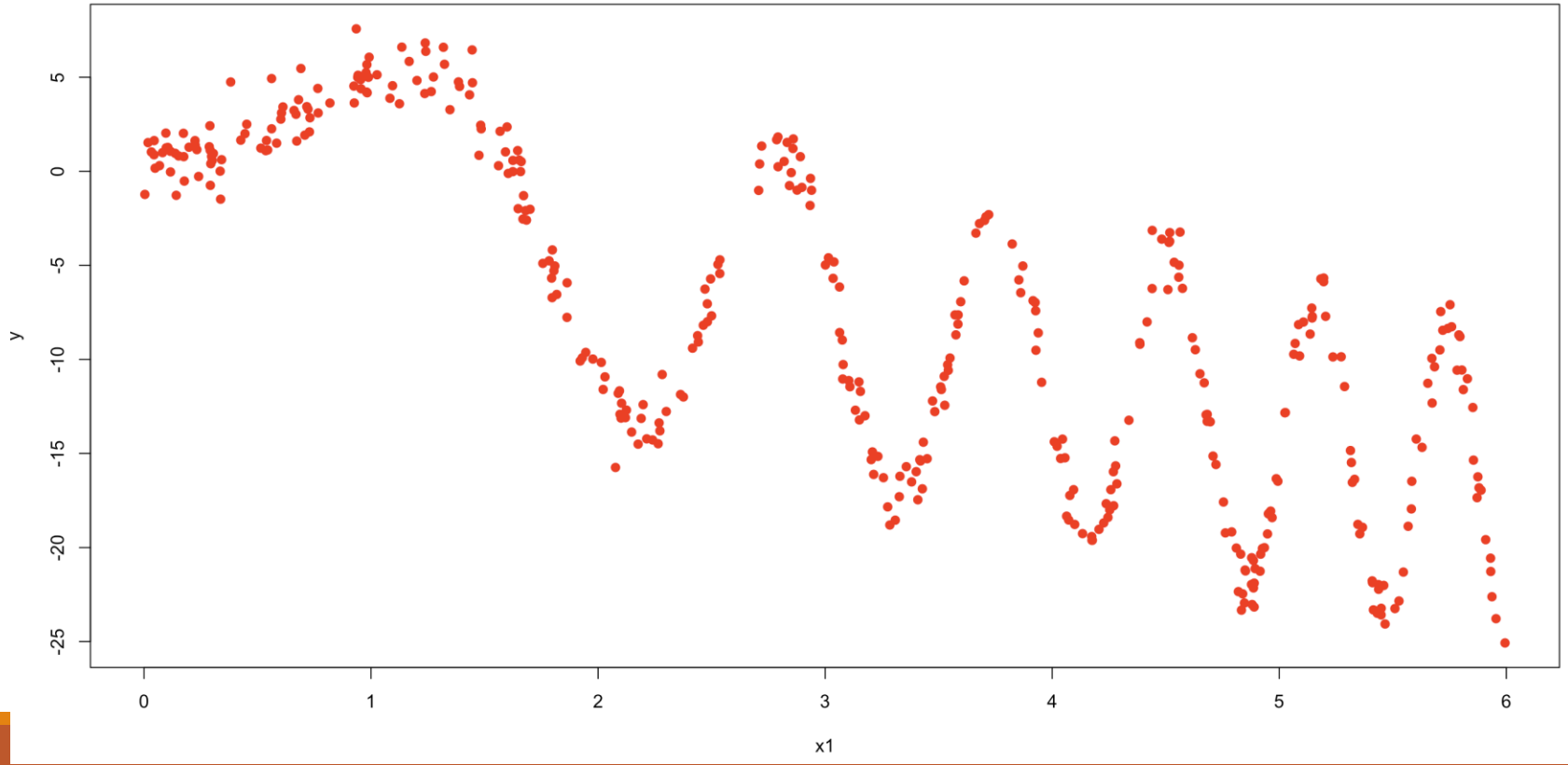
$$\epsilon \sim N(0, \sigma^2)$$



How would you model this data?

What is the underlying function?

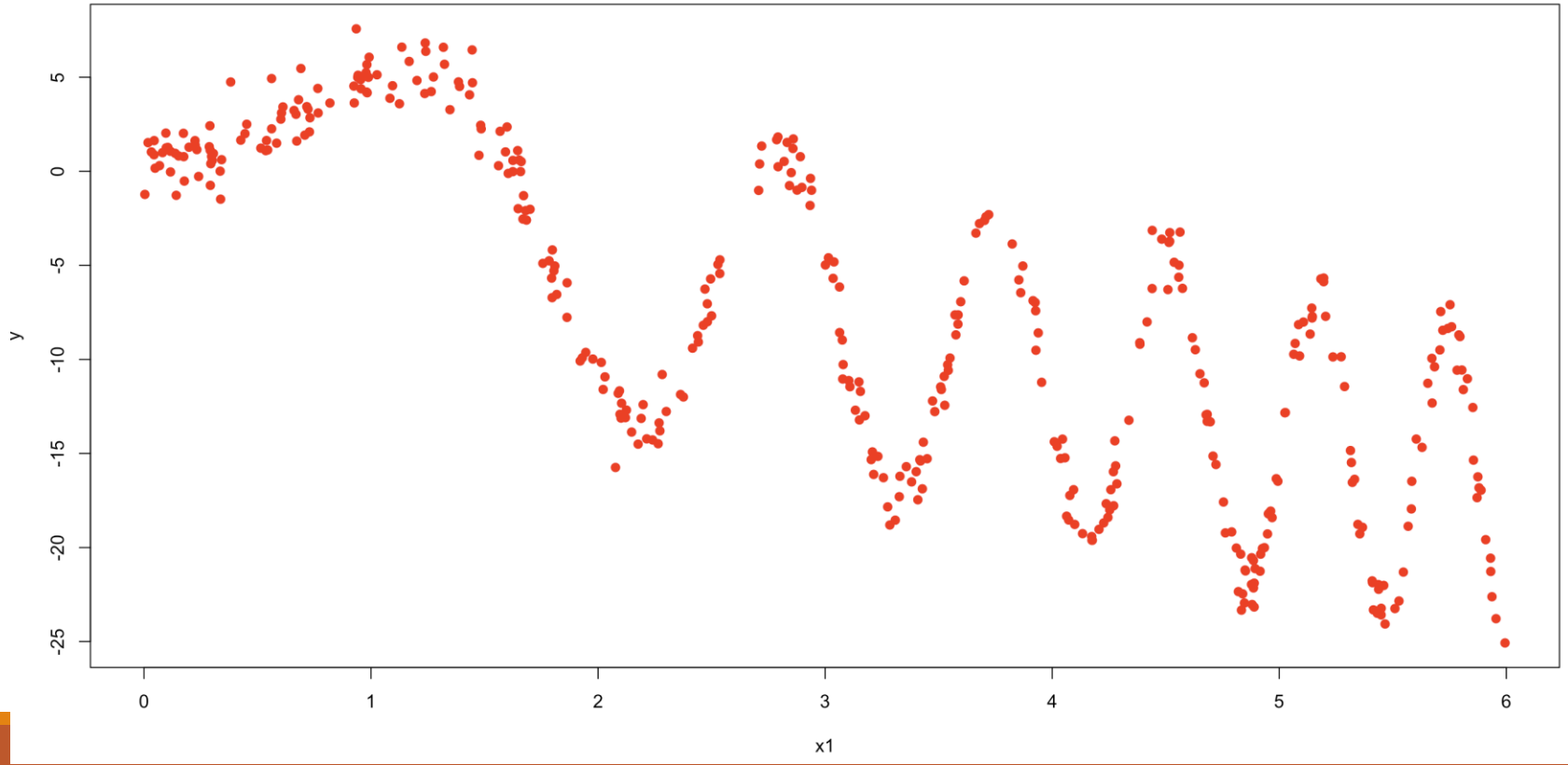
Can we construct a reliable approximation?



How would you model this data?

What is the underlying function?

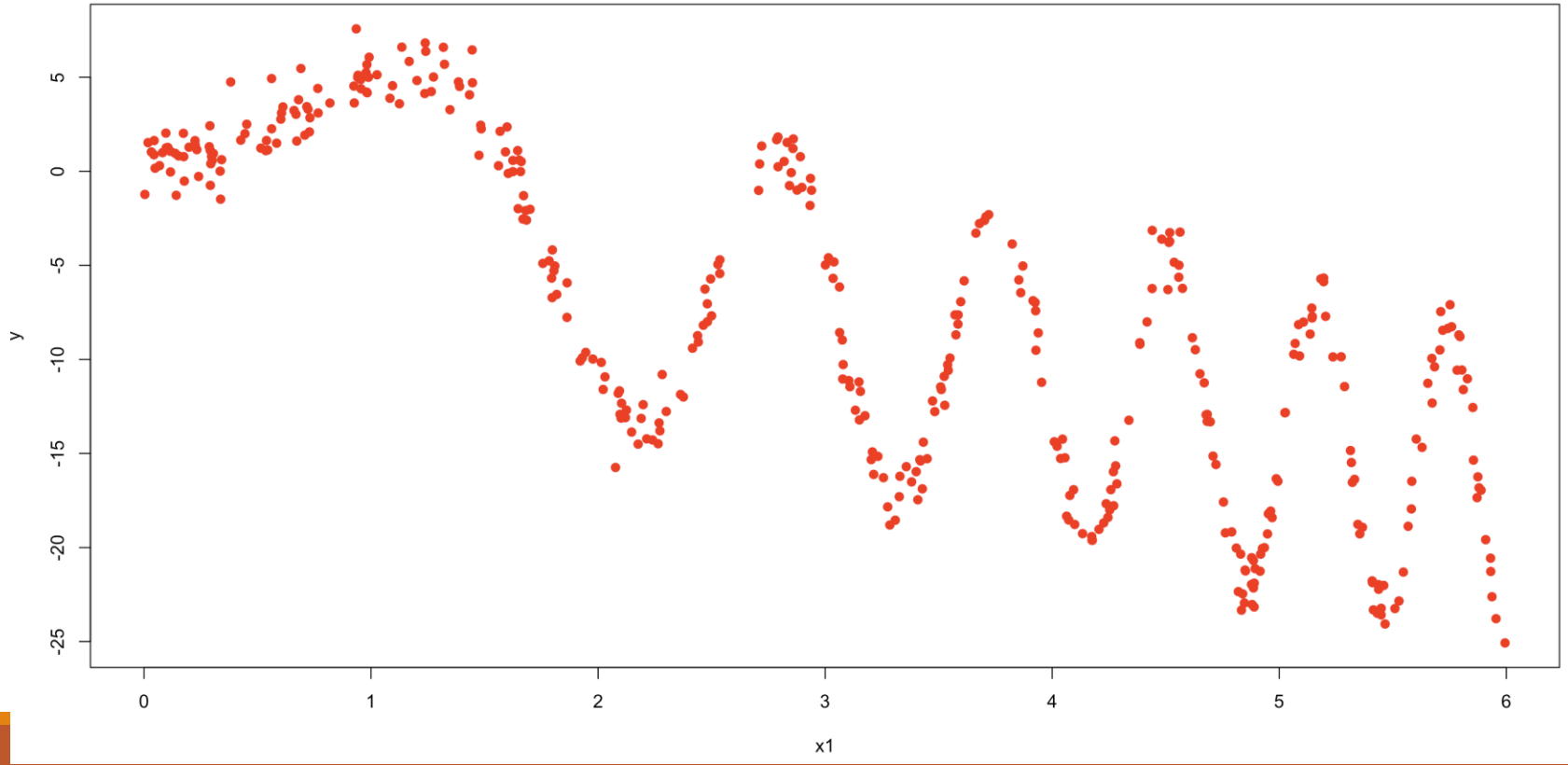
Can we construct a reliable approximation?



How would you model this data?

What is the underlying function?

Can we construct a reliable approximation?



$$f(\mathbf{x}) = \sum_{h=1}^{H} w_h \, \sigma\left(\sum_{j=1}^{p} v_{hj} x_j + b_h\right) + b_0$$

$$f(\mathbf{x}) = \frac{1}{T} \sum_{t=1}^{T} f^{(t)}(\mathbf{x})$$

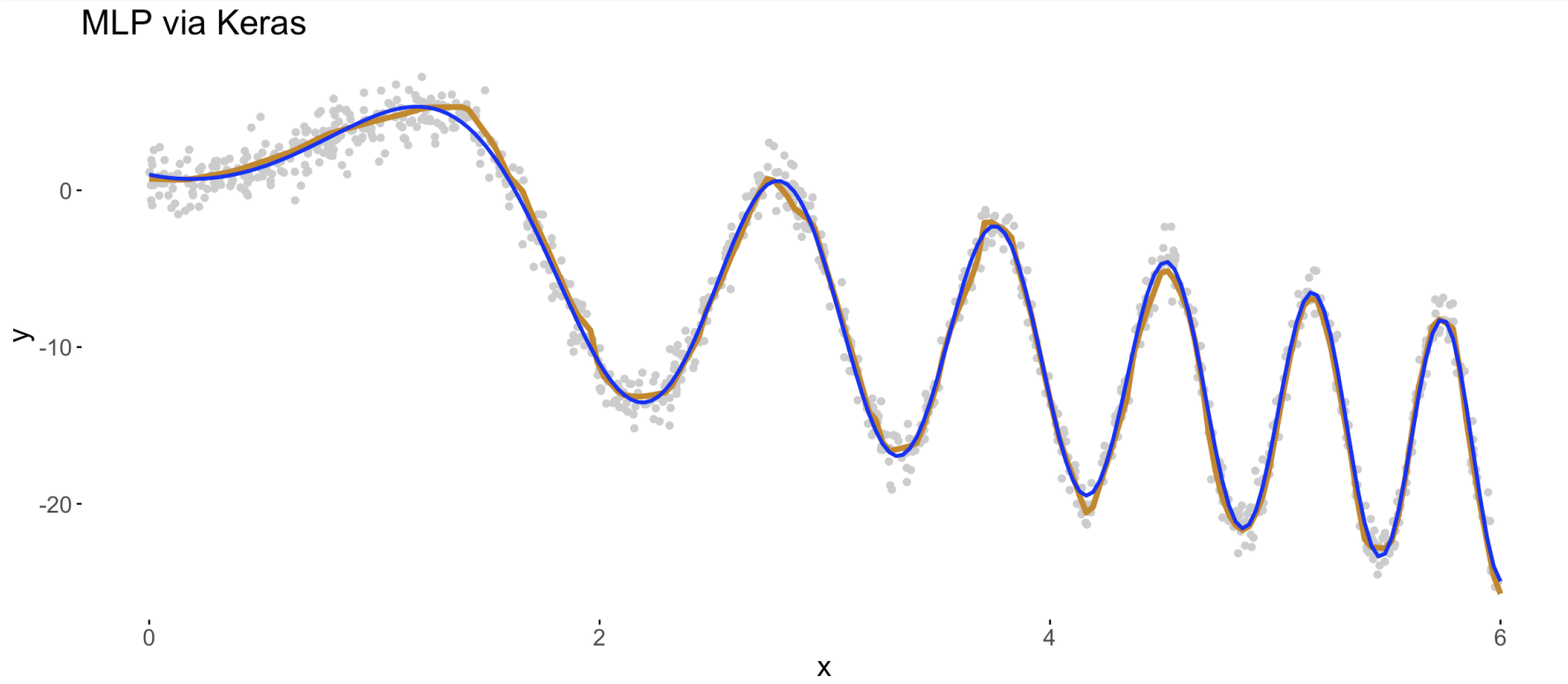
$$f^{(t)}(\mathbf{x}) = \sum_{m=1}^{M_t} c_m^{(t)} \cdot \mathbf{1}_{\{\mathbf{x} \in R_m^{(t)}\}}$$

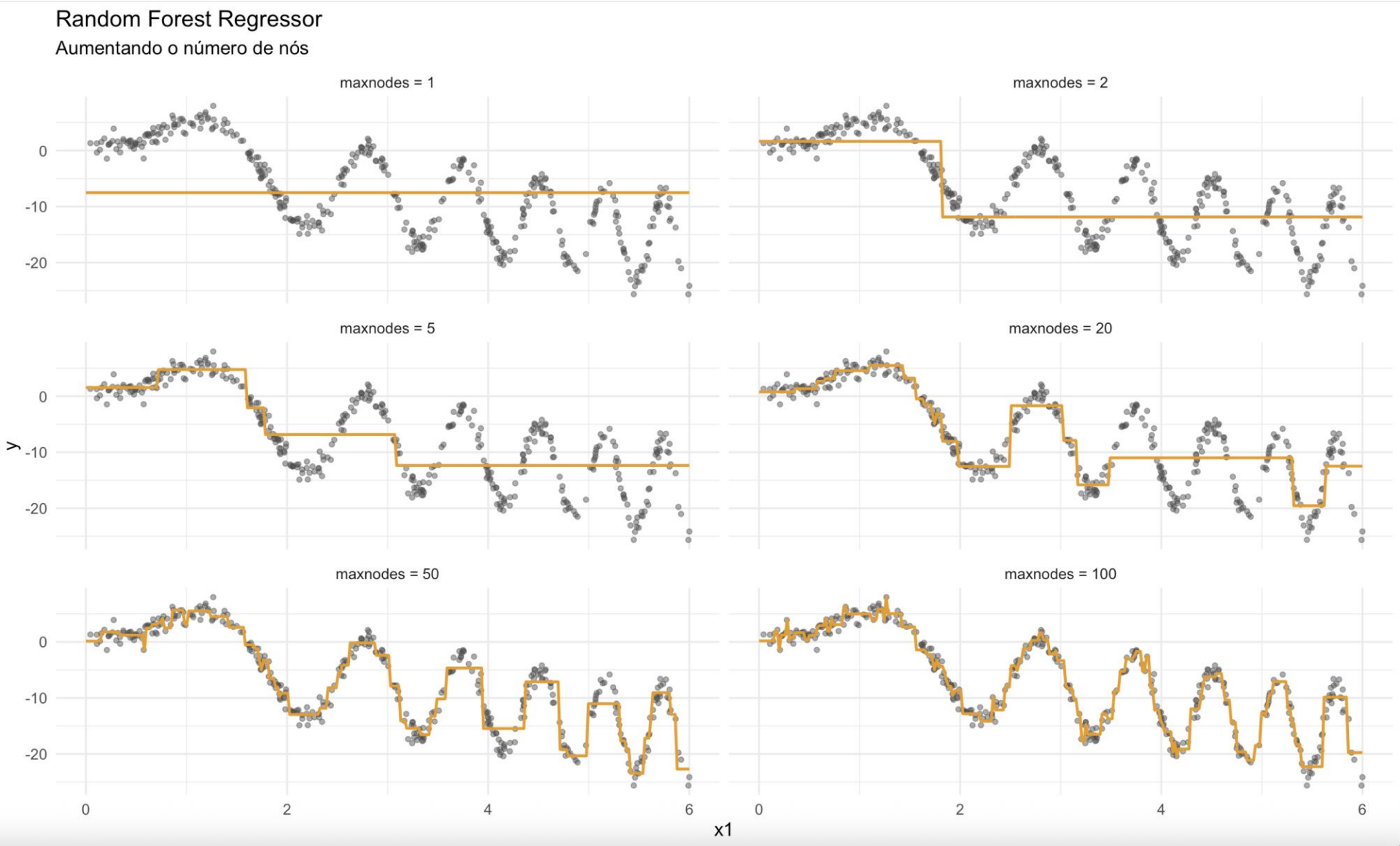
$$f(\mathbf{x}) = \beta_0 + \sum_{j=1}^{p} f_j(x_j)$$

$$f(\mathbf{x}) = \sigma\left(\sum_{j=1}^{p} w_j x_j + b\right) = \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right)$$

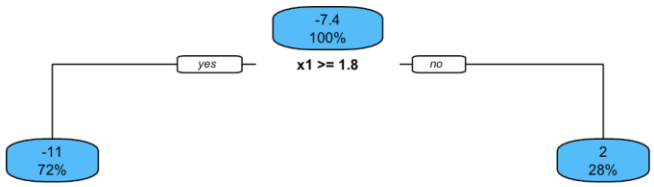
$$f(\mathbf{x}) = \beta_0 + \sum_{j=1}^{p} \beta_j x_j$$

Deep Learning Regression (aka Multilayer Perceptron)

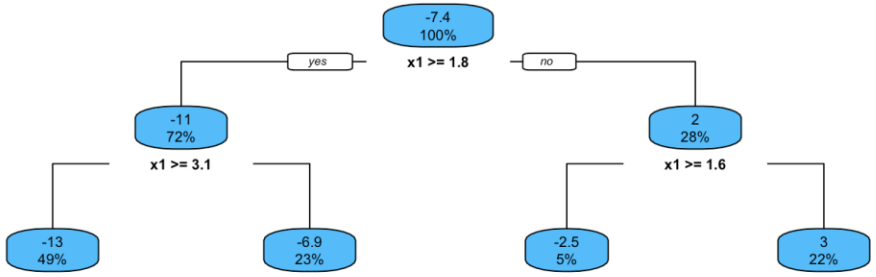



Random Forest Regressor
Aumentando o número de nós
maxnodes = 1
maxnodes = 2
maxnodes = 5
maxnodes = 20
maxnodes = 50
maxnodes = 100
y
x1
0
-10
-20
0
2
4
6

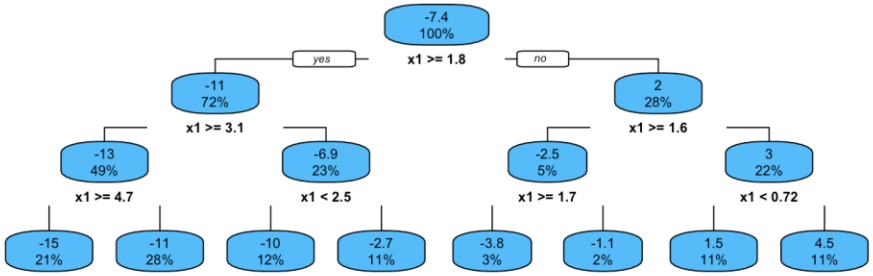
maxnodes ≈ 1



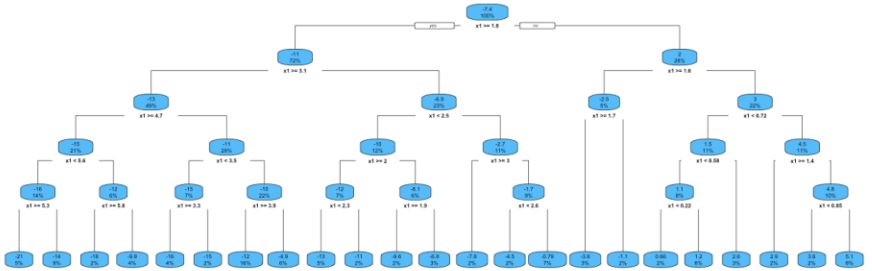
maxnodes ≈ 2



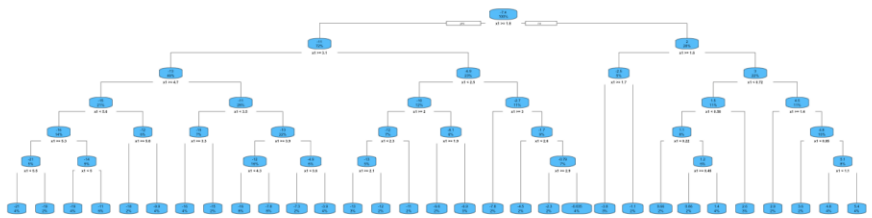
maxnodes ≈ 5



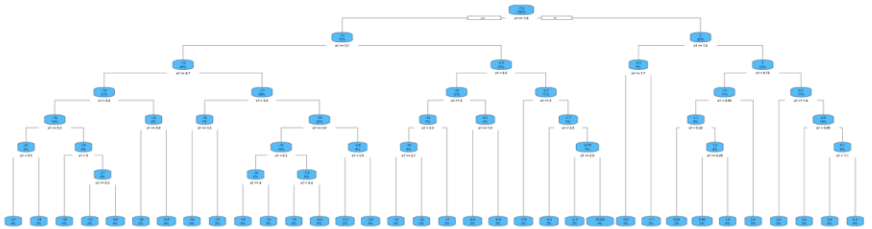
maxnodes ≈ 20



maxnodes ≈ 50

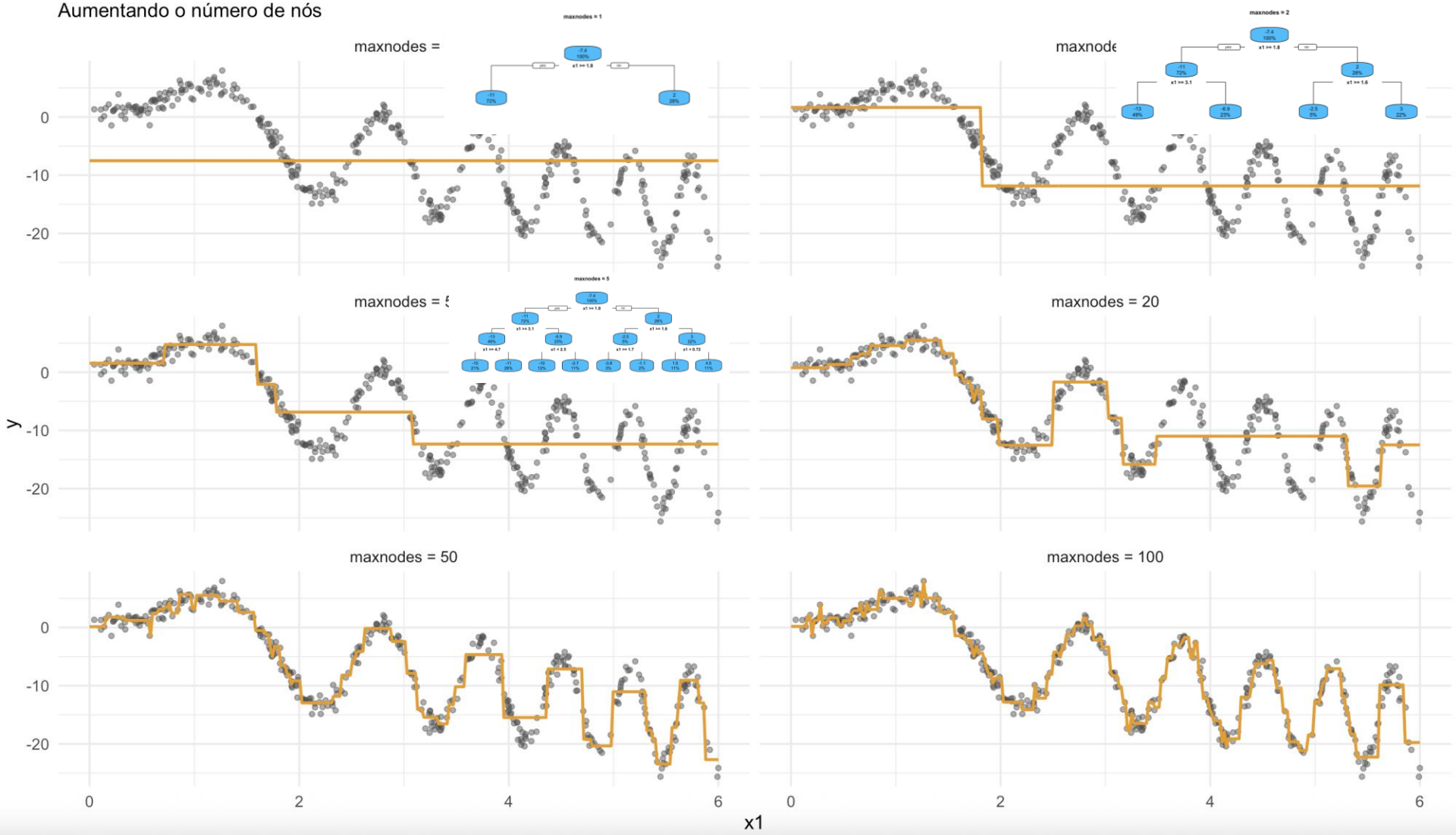


maxnodes ≈ 100

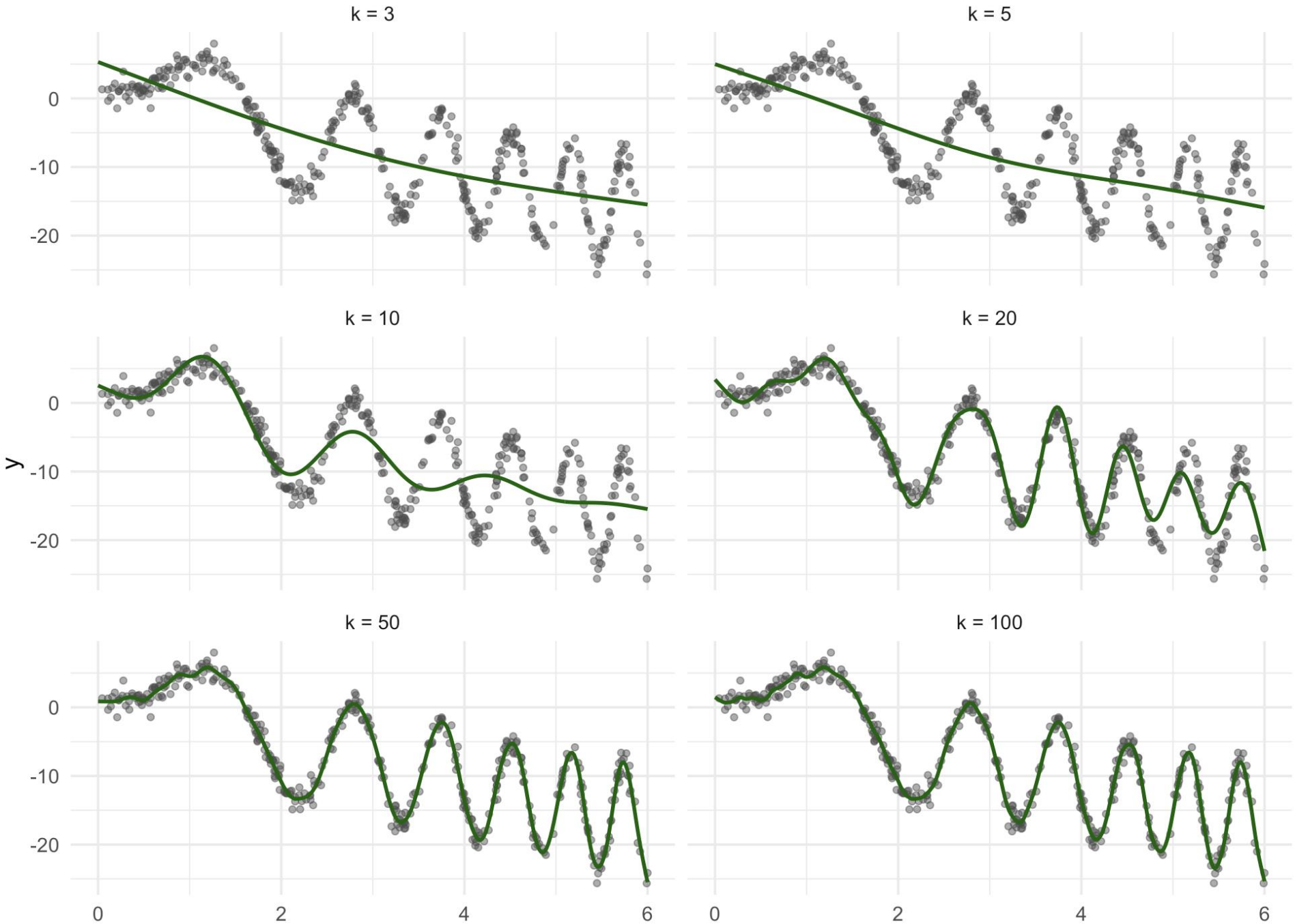


Random Forest Regressor

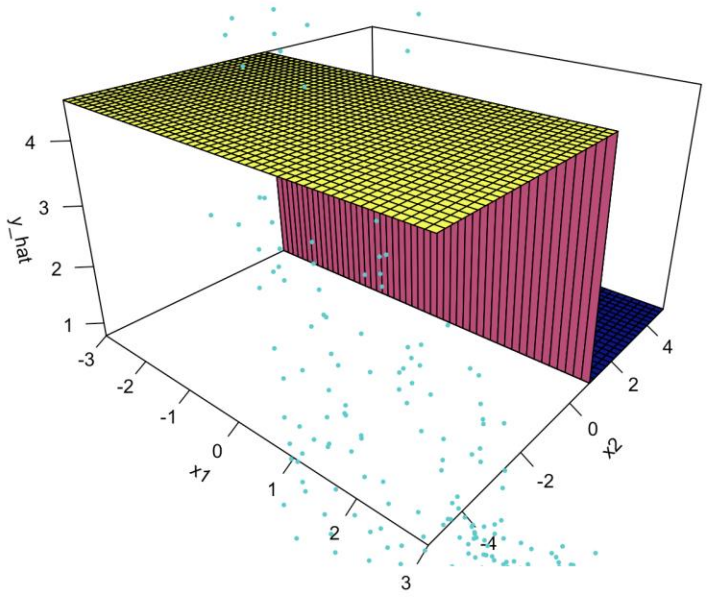
Aumentando o número de nós



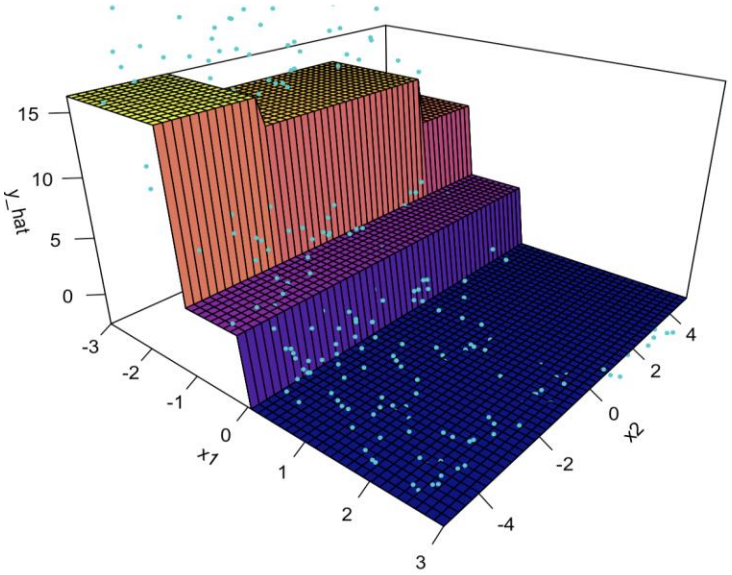
Generalized Additive Models



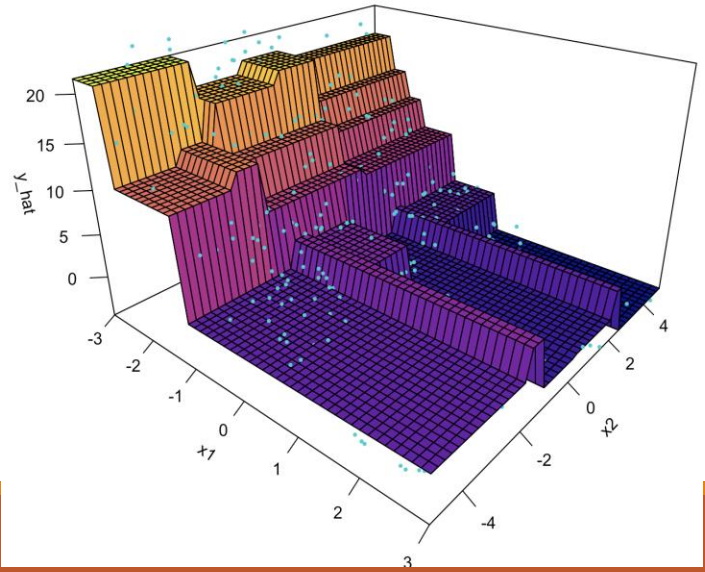
Nodes = 2



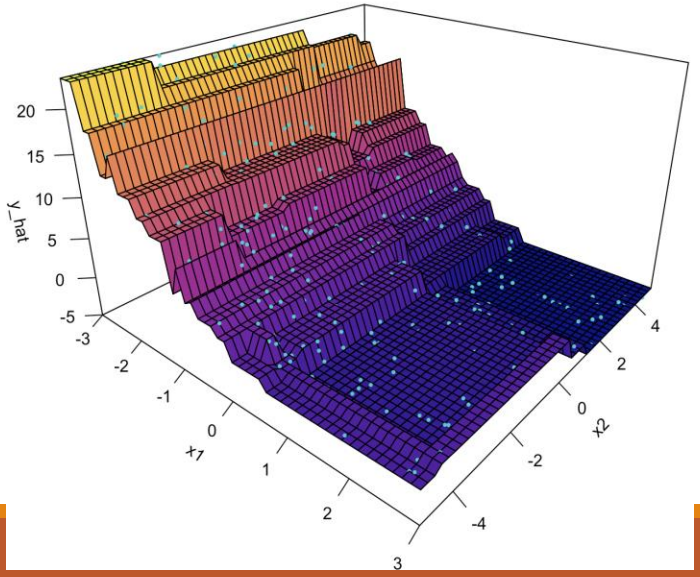
Nodes = 5



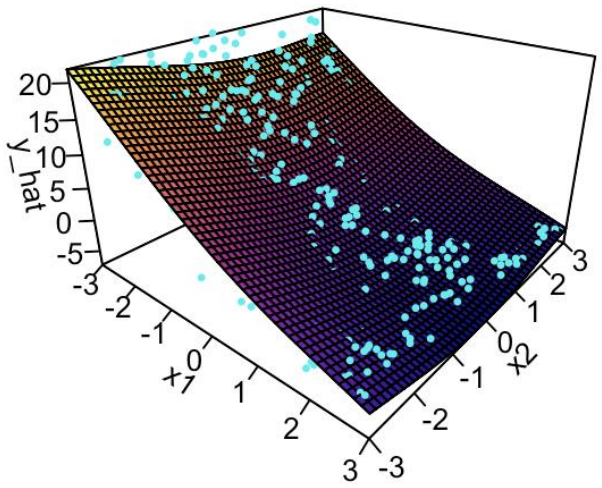
Nodes = 20



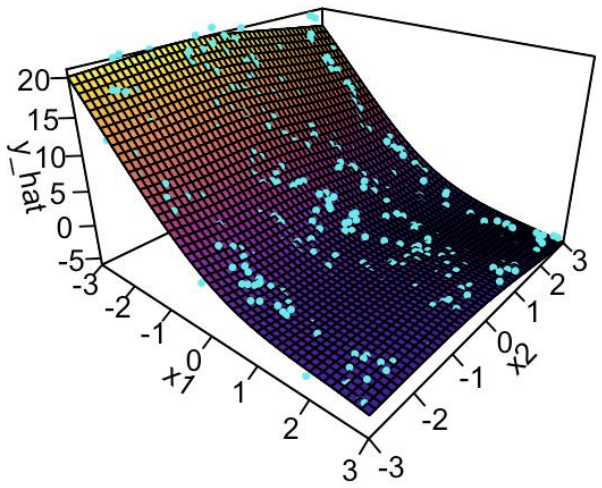
Nodes = 50



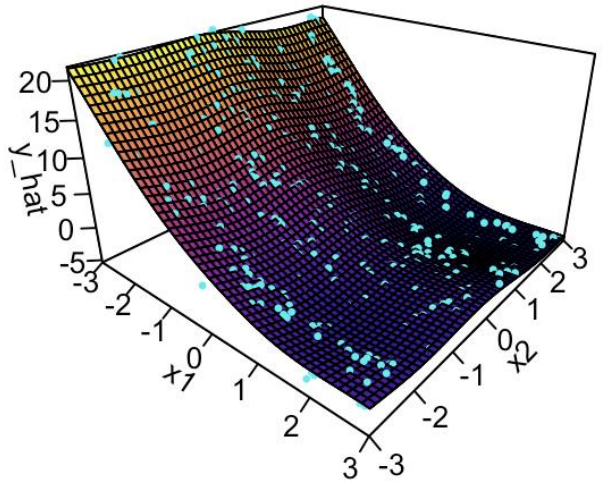
GAM - k = 5



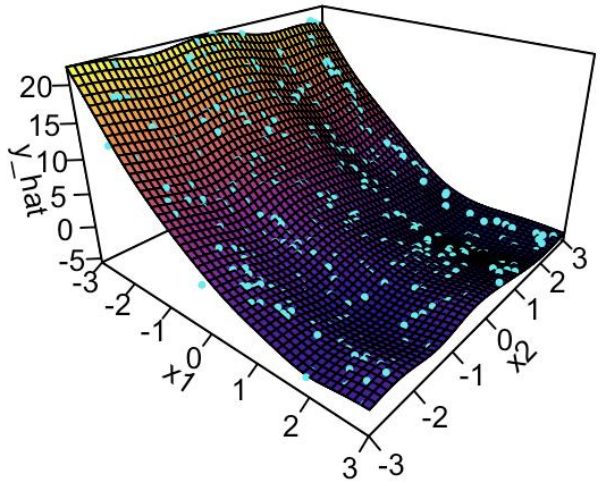
GAM - k = 10



GAM - k = 20

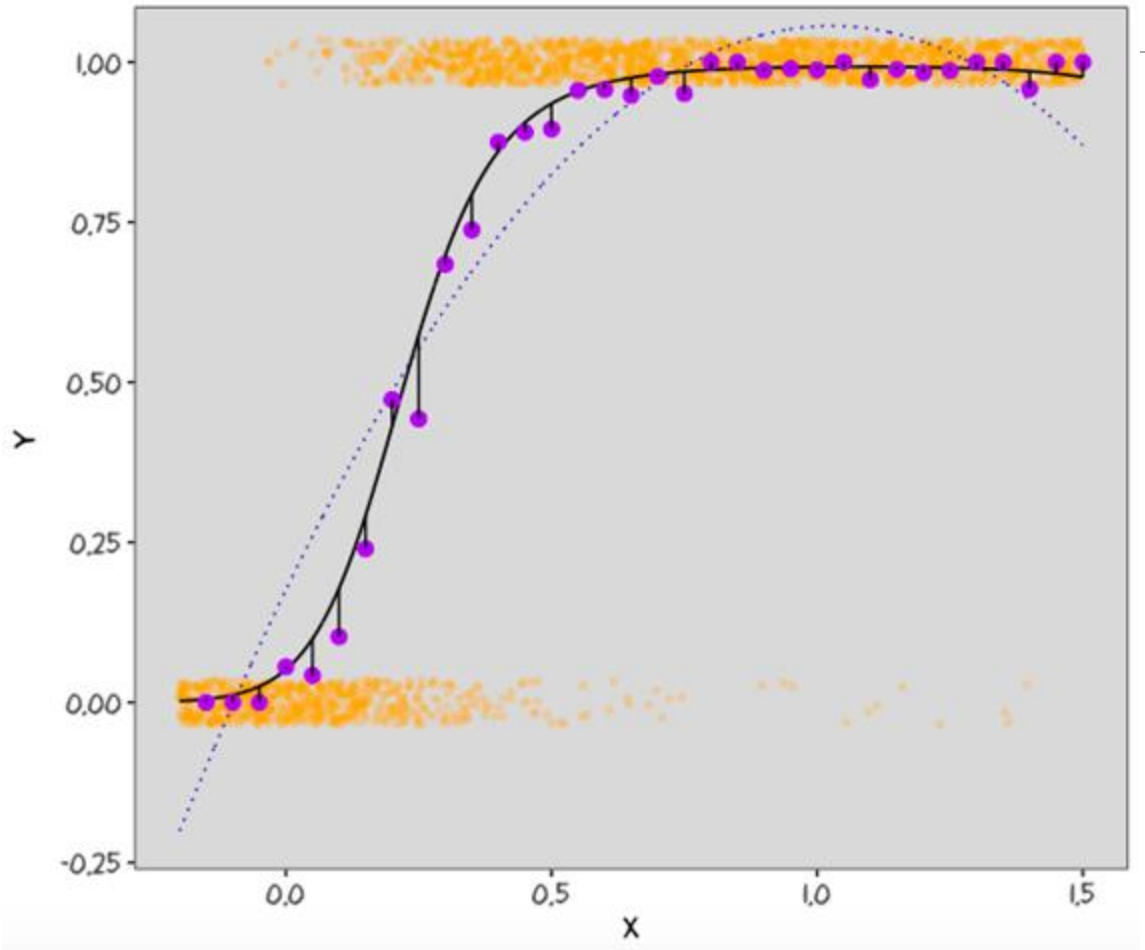


GAM - k = 50

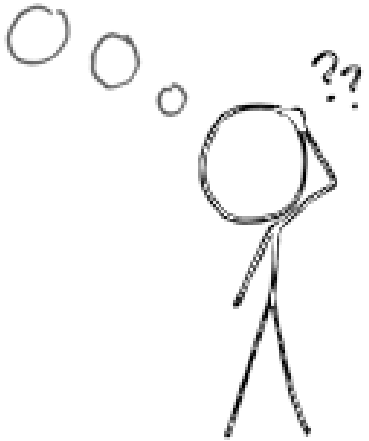


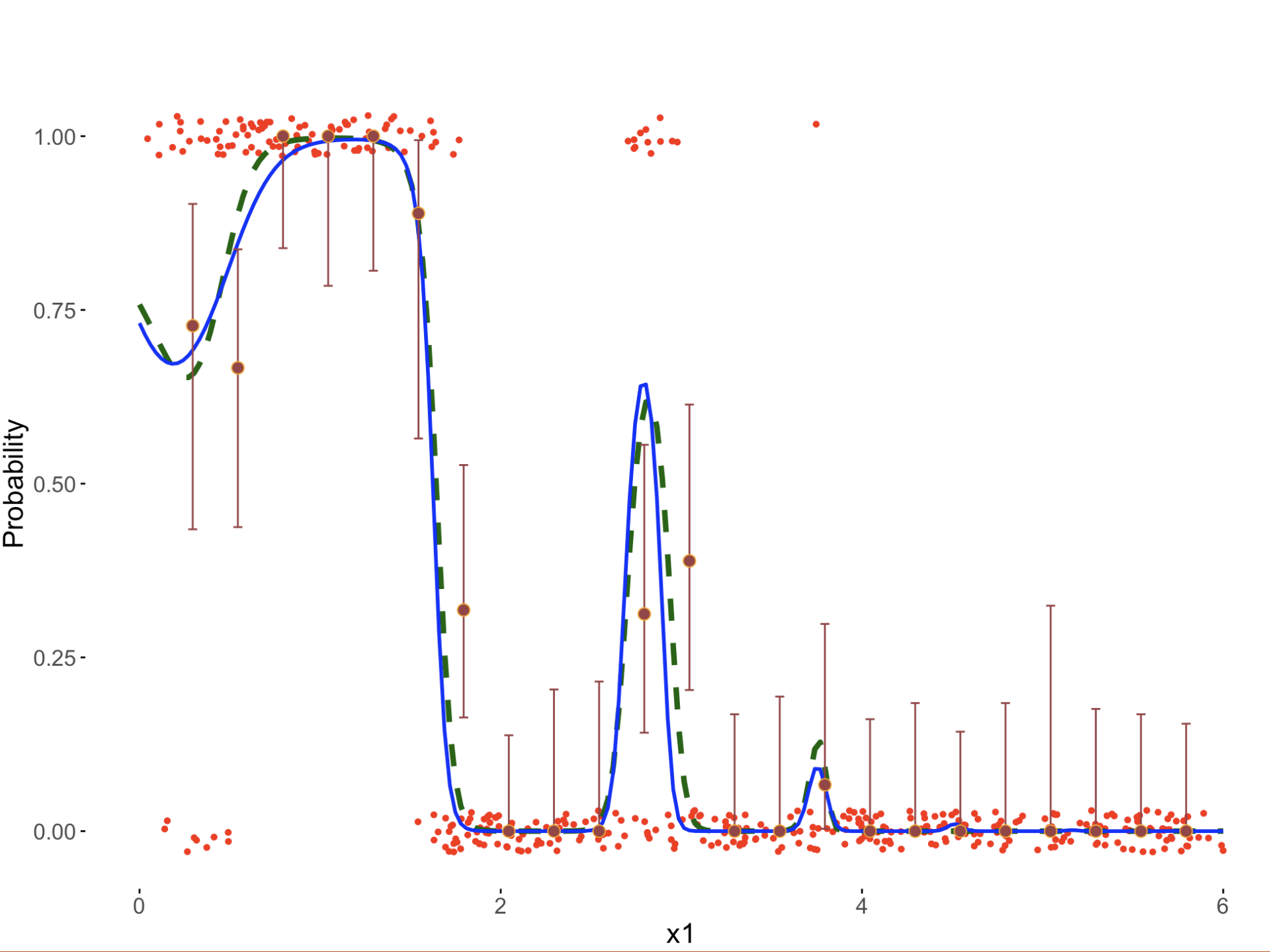

1.00
0.75
0.50
0.25
0.00
y
0
2
4
6
x1

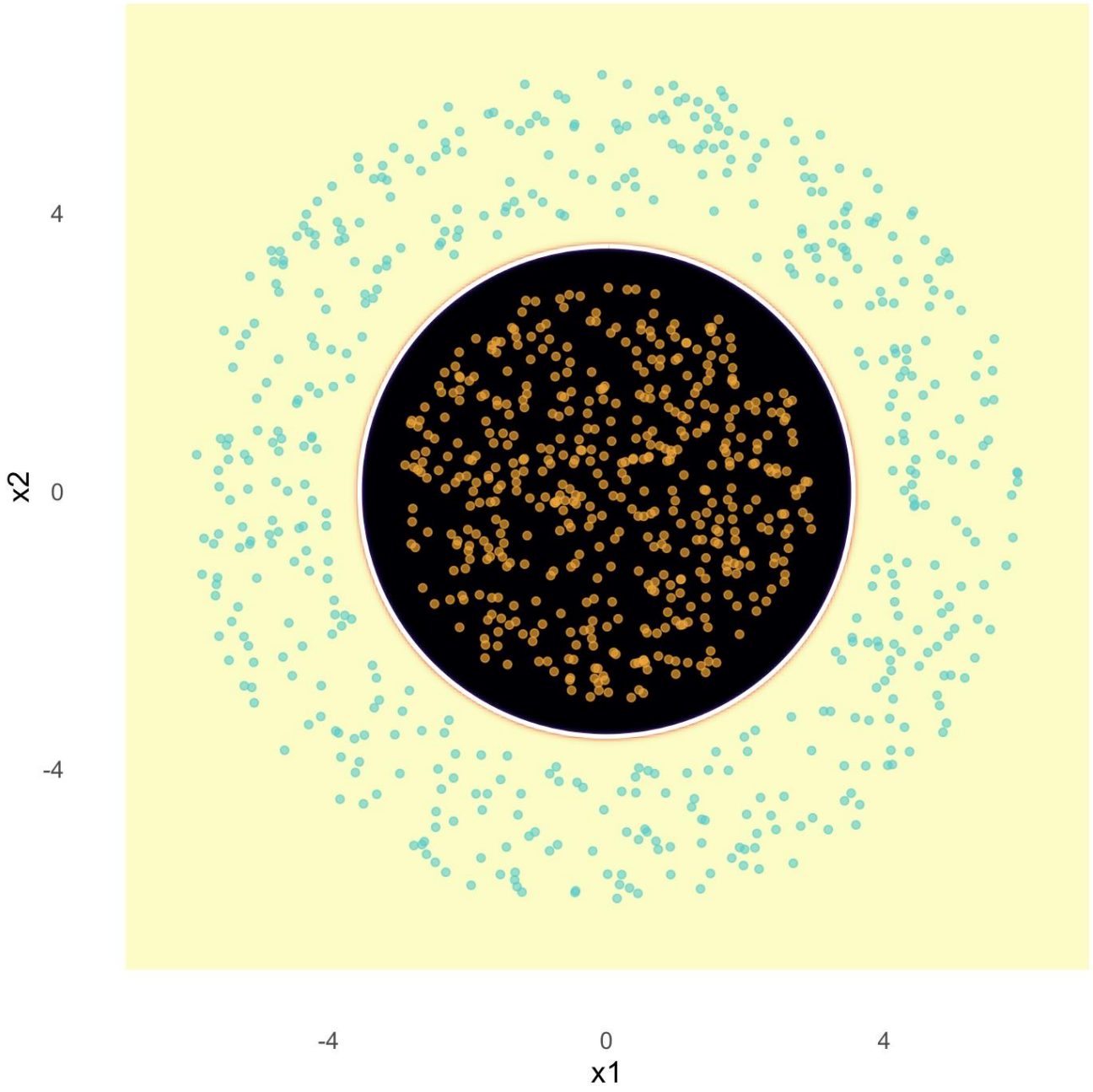
Linear Models: Limitations

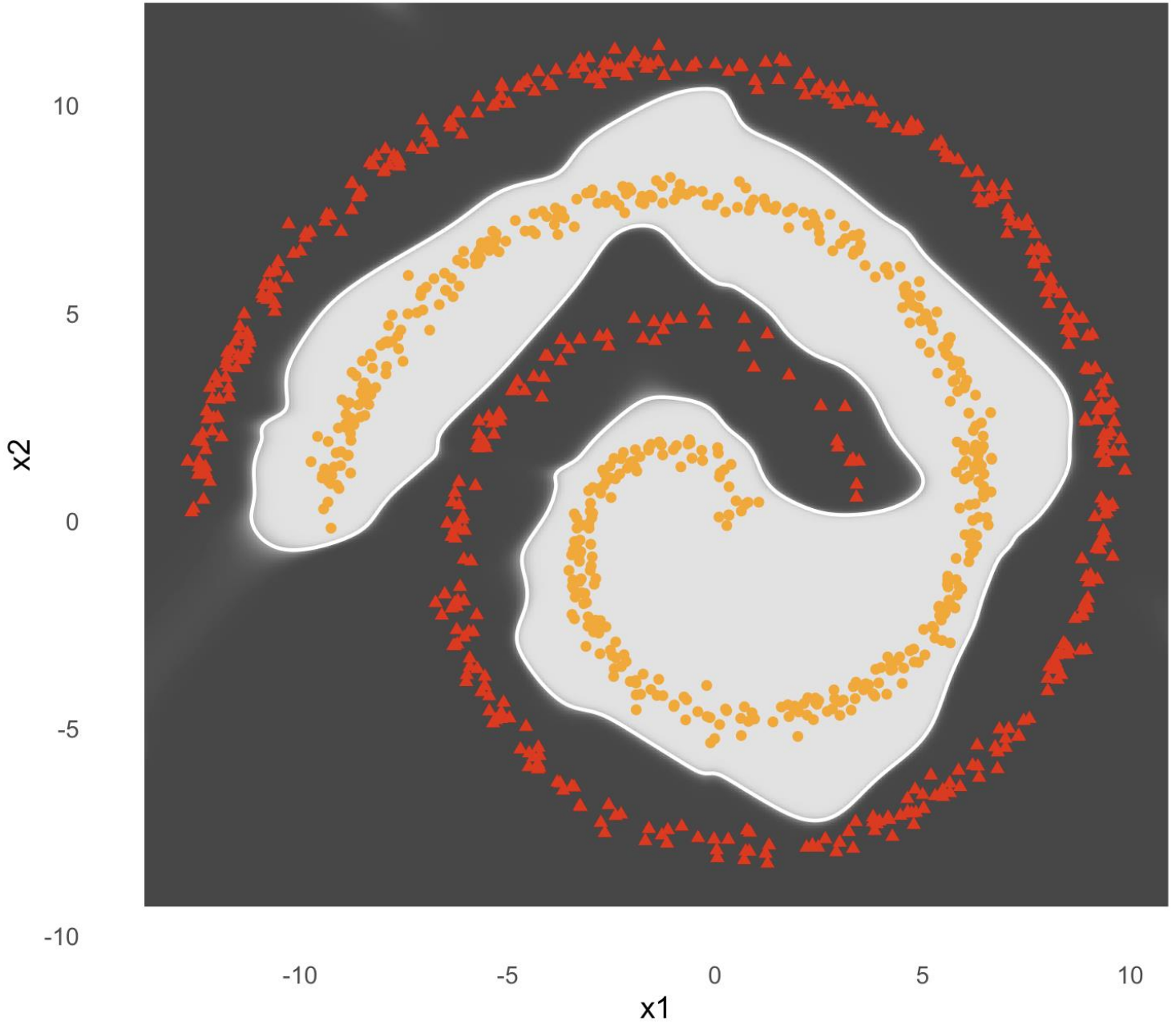


Binary or fractional data




1.00
0.75
0.50
0.25
0.00
Probability
0
2
4
6
x1


4
0
-4
x2
-4
0
4
x1


10
5
0
-5
-10
x2
-10
-5
0
5
10
x1

Bernoulli regression (Logistic Regression)

Describes a random variable which takes the value 1 with success probability of p and 0 with failure probability of $1 - p$.

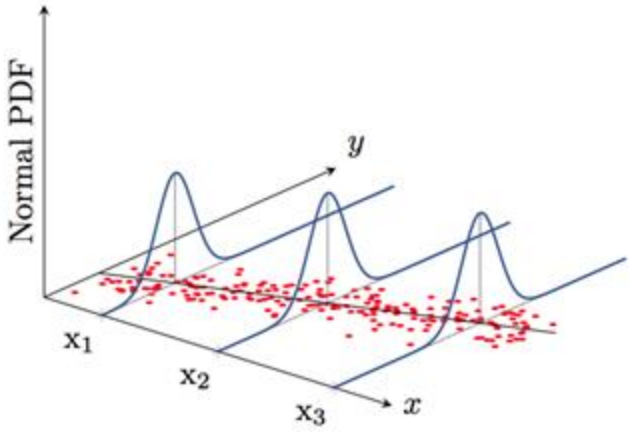
Logistic Model

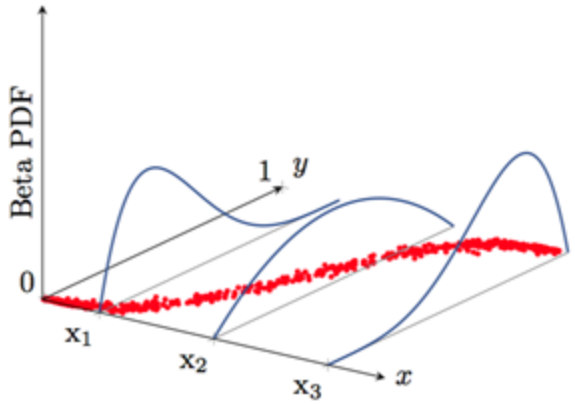
$$Y_i \sim \text{Bernoulli}(p) \equiv p^y (1-p)^{1-y}$$

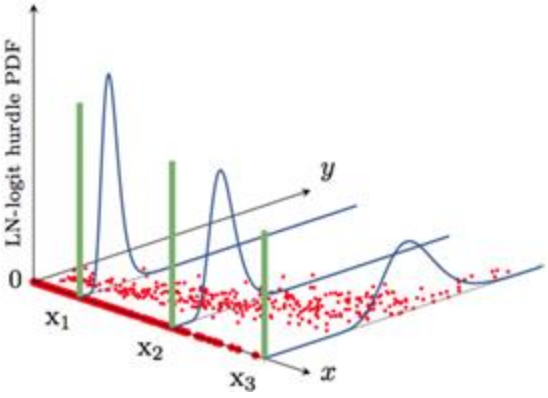
$$\log \frac{p}{1-p} = \eta$$

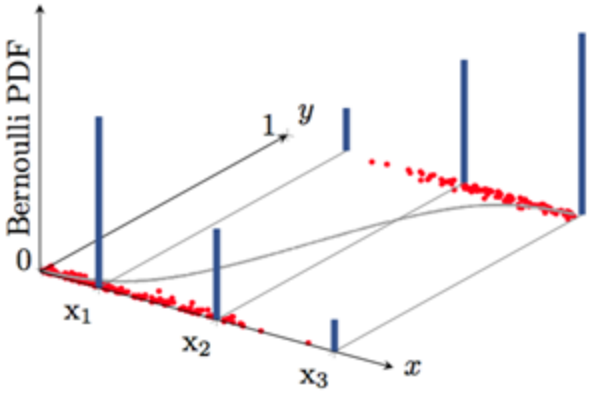
$$\eta = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k$$

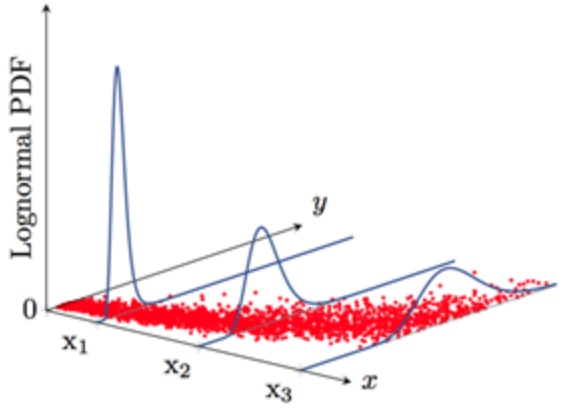
Generalized Linear Models and extensions

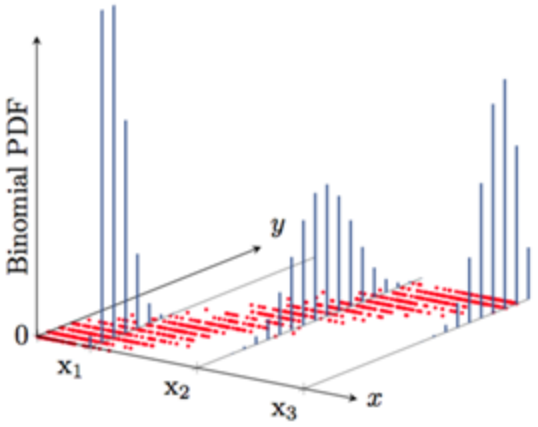












Example: Poisson regression

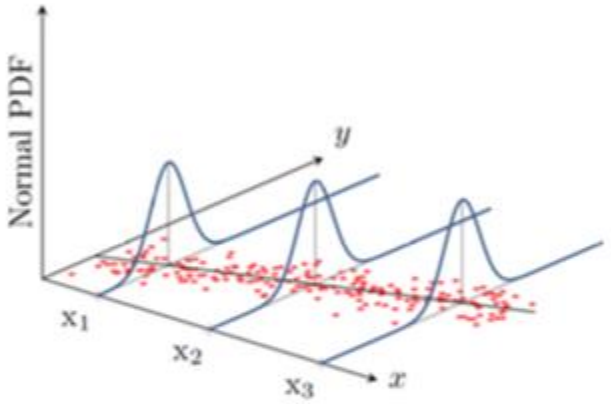
The Poisson distribution model the number of times an event occurs in an interval of time or space.

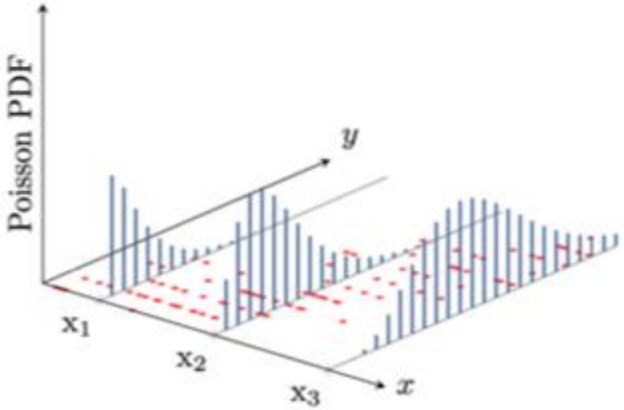
Linear Models

$$Y \sim \text{Normal}(\mu, \sigma^2)$$
$$\mu = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k$$

Poisson Model

$$Y \sim \text{Poisson}(\mu) \equiv \frac{\mu^y e^{-\mu}}{y!}$$
$$\log(\mu) = \eta$$
$$\eta = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k$$



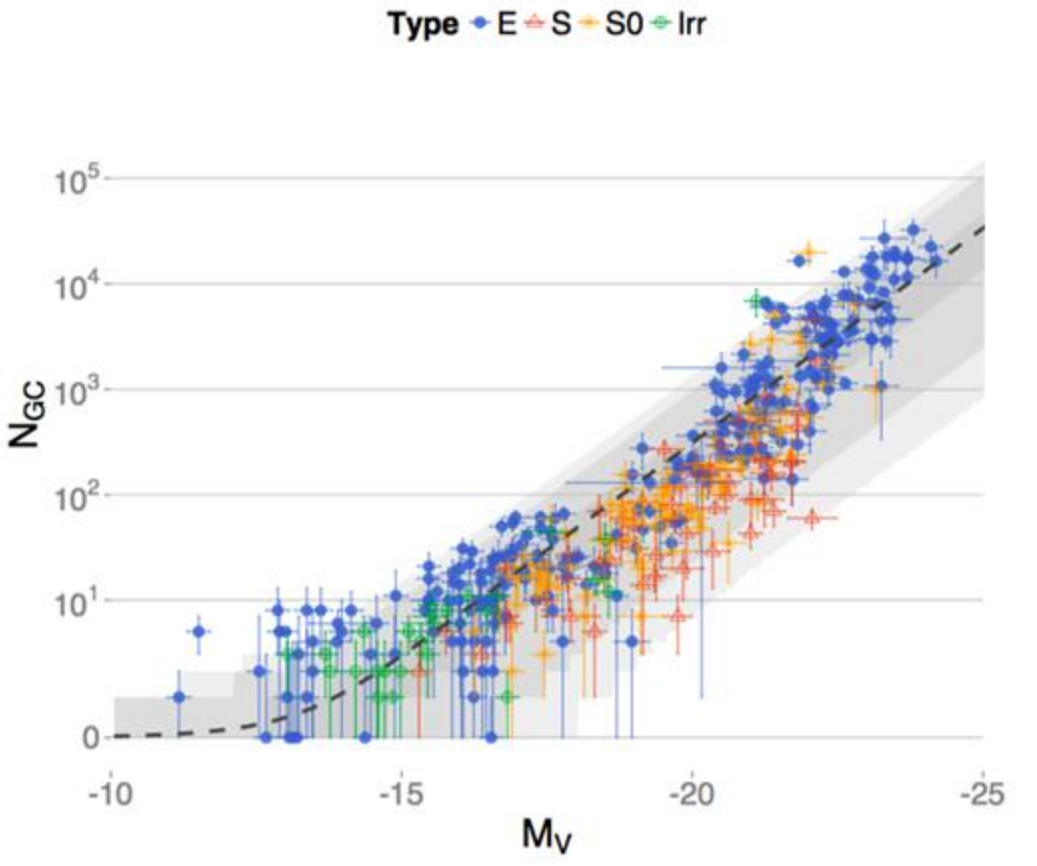


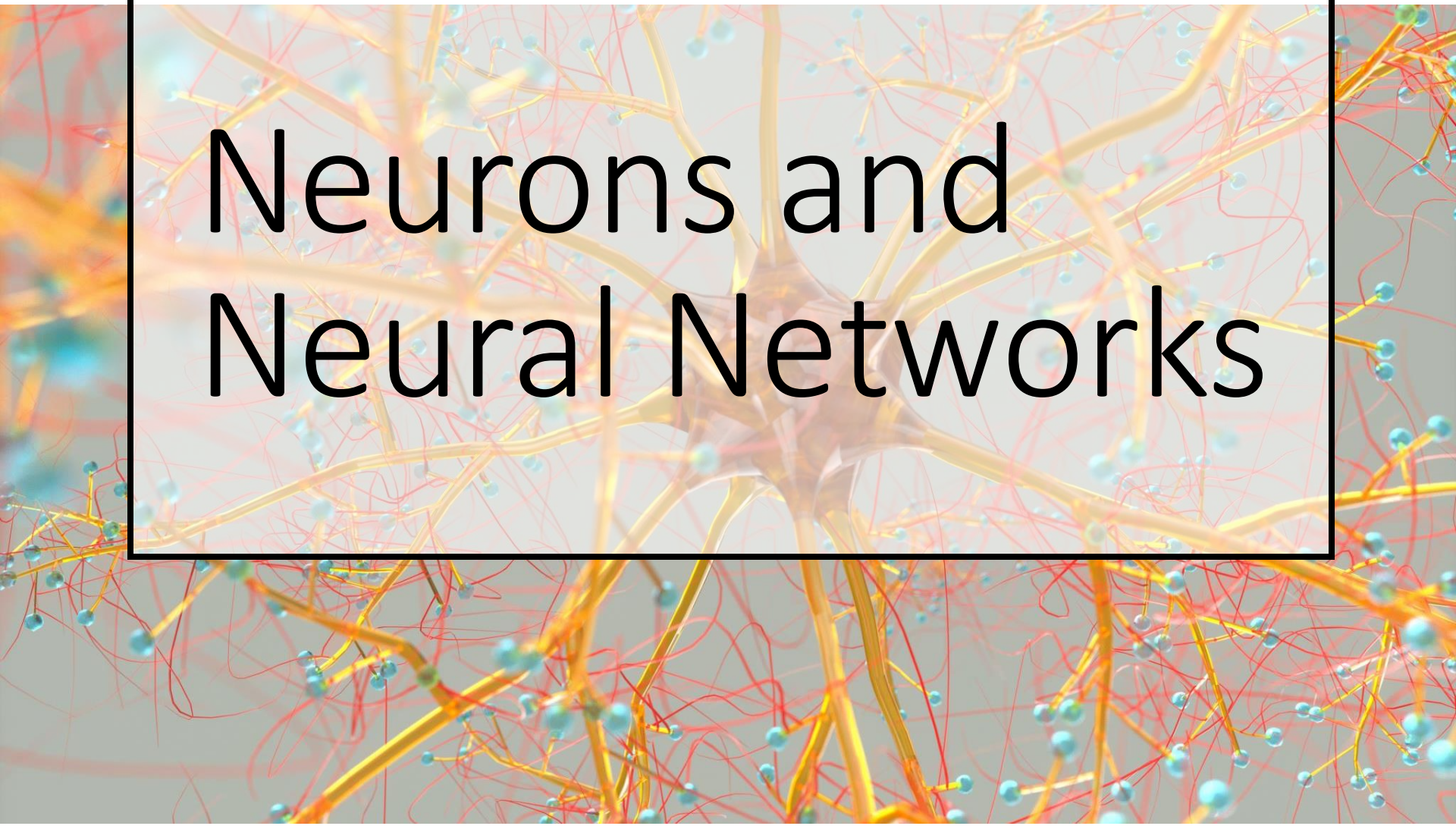
The overlooked potential of generalized linear models in astronomy – III. Bayesian negative binomial regression and globular cluster populations

R. S. de Souza ✉, J. M. Hilbe ✉, B. Buelens, J. D. Riggs, E. Cameron, E. E. O. Ishida, A. L. Chies-Santos, M. Killedar

Monthly Notices of the Royal Astronomical Society, Volume 453, Issue 2, 21 October 2015, Pages 1928–1940,
https://doi.org/10.1093/mnras/stv1825

$$
\begin{aligned}
&N_{GC;i} \sim \mathrm{NB}(\mathrm{p_i}, \mathrm{k});\\
&p_i = \frac{k}{k+\mu_i};\\
&\mu_i = e^{\eta_i} + \epsilon_{N_{GC};i};\\
&\eta_i = \beta_0 + \beta_1 \times M^*_{V;i};\\
&k \sim \mathcal{U}(0,5);\\
&M_{V;i} \sim \mathcal{N}(M^*_{V;i}, e^2_{M_V;i});\\
&\epsilon_{N_{GC};i} \sim \mathcal{B}(0.5, 2e_{N_{GC};i}) - e_{N_{GC};i};\\
&\beta_0 \sim \mathcal{N}(0, 10^6);\\
&\beta_1 \sim \mathcal{N}(0, 10^6);\\
&M^*_{V;i} \sim \mathcal{U}(-26, -10);\\
&i = 1, \cdots, N.
\end{aligned}
$$

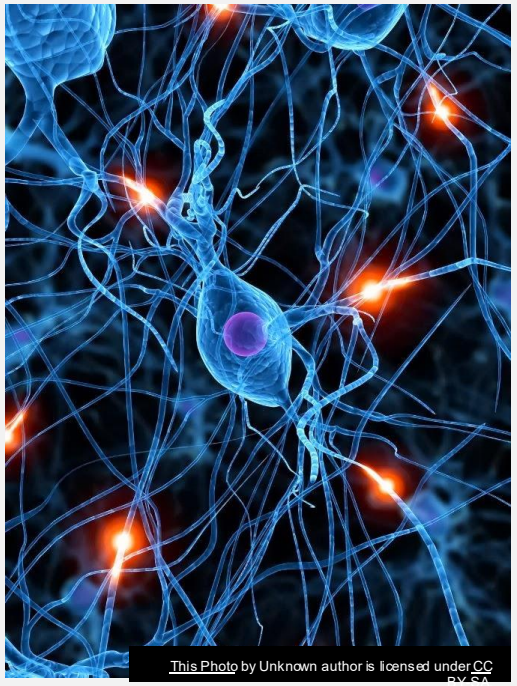



Neurons and
Neural Networks

Connectionism

A movement in cognitive science that hopes to explain intellectual abilities using artificial neural networks (also known as “neural networks” or “neural nets”).

- Stanford Encyclopedia of Philosophy



Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

F. ROSENBLATT

Cornell Aeronautical Laboratory

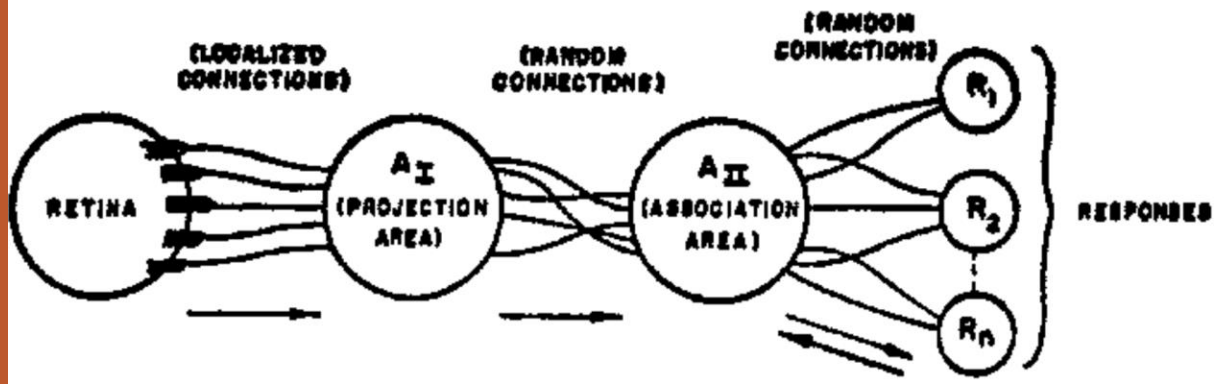
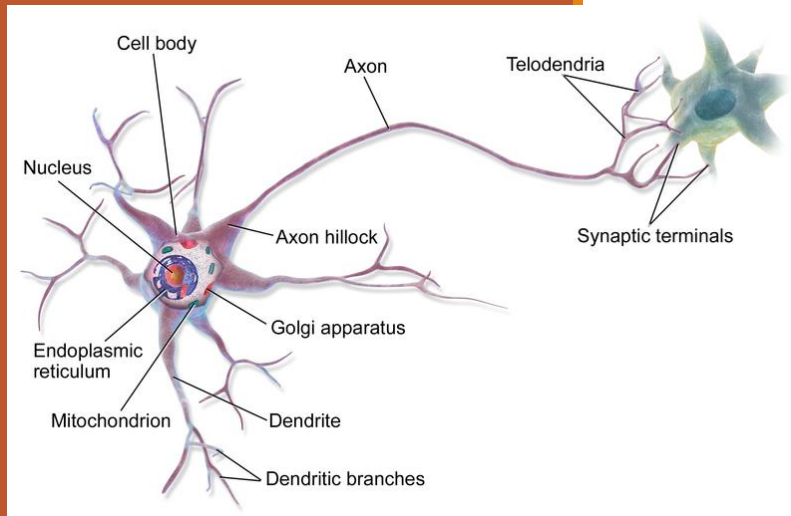
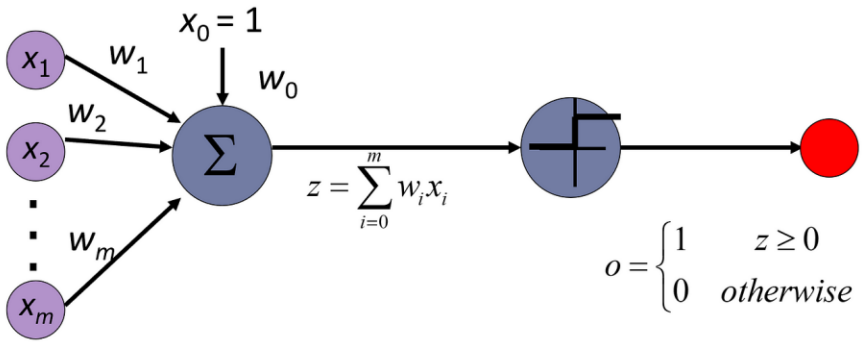


FIG. 1. Organization of a perceptron.

Biological Neuron



PERCEPTRON



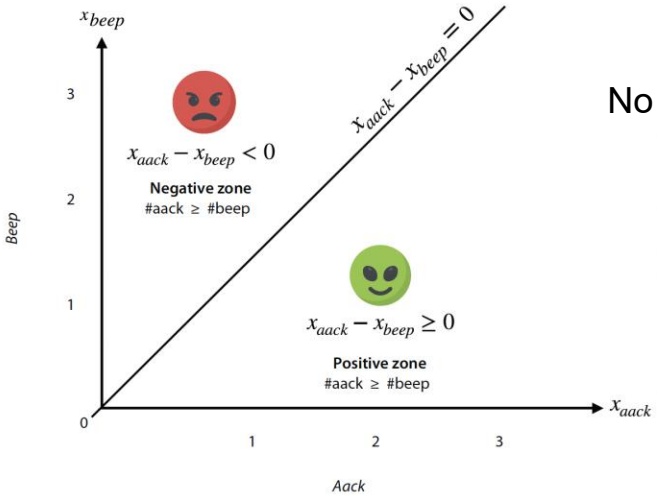
$$\hat{y} = step(ax_1 + bx_2 + c).$$

Given a sentence, assign the following weights and bias to the words:

Weights:

- *Crack*: one point
- *Doink*: one point

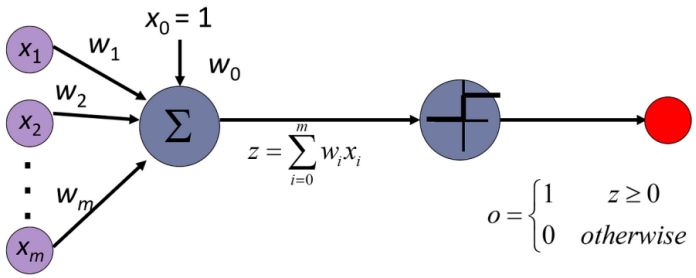
Bias: –3.5 points

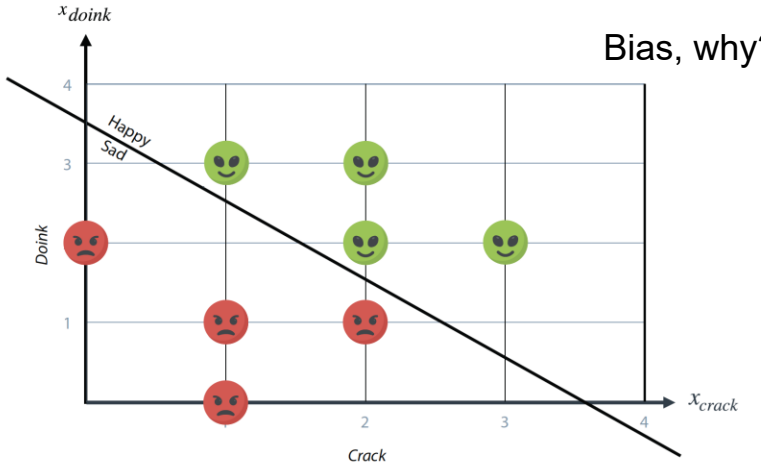


No bias

Positive zone: the area on the plane for which $x_{crack} + x_{doink} - 3.5 \geq 0$

Negative zone: the area on the plane for which $x_{crack} + x_{doink} - 3.5 < 0$





Bias, why?

Using the principle of Taylor series expansion, polynomial regression provides a simple yet effective method for approximating unknown functions, fitting data with polynomial expressions.

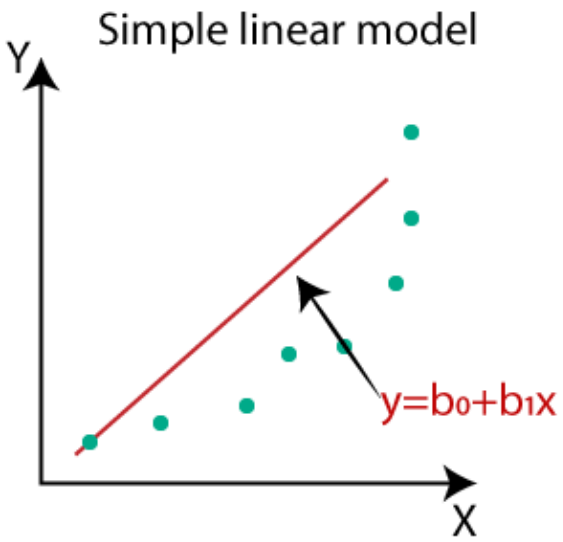
$$\frac{1}{1-x} = 1 + x + x^2 + x^3 \cdots = \sum_{n=0}^{\infty} x^n$$

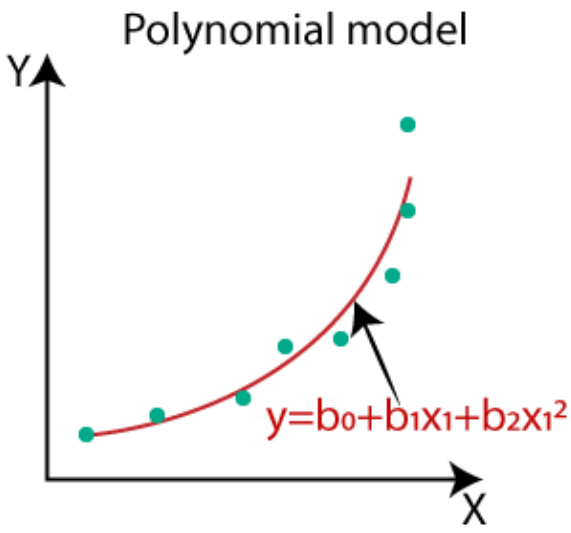
$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} \cdots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

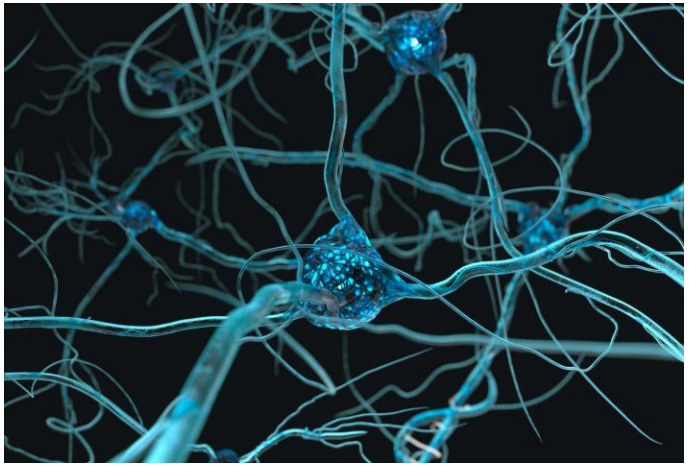
$$sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

$$cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}$$

$$ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{n+1}}{n+1}$$







Multilayer Perceptrons aka Deep Learning

Kolmogorov-Arnold Representation theorem

Any continuous function $f : [0,1]^n \to \mathbb{R}$ *can be written as*

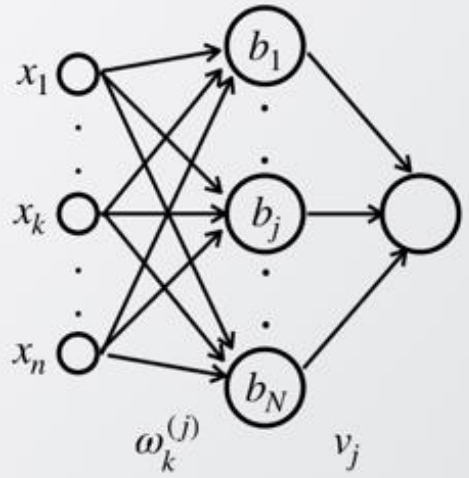
$$f(x) = f(x_1, .., .., x_n) = \sum_{q=1}^{Z_m} \phi_q \left(\sum_{q=1}^{m} \Psi_q(x_q) \right)$$

The universal approximation theorem states that any continuous function $f : [0,1]^n \to [0,1]$ can be approximated arbitrarily well by a neural network with at least 1 hidden layer with a finite number of weights, which is what we are going to illustrate in the next subsections.

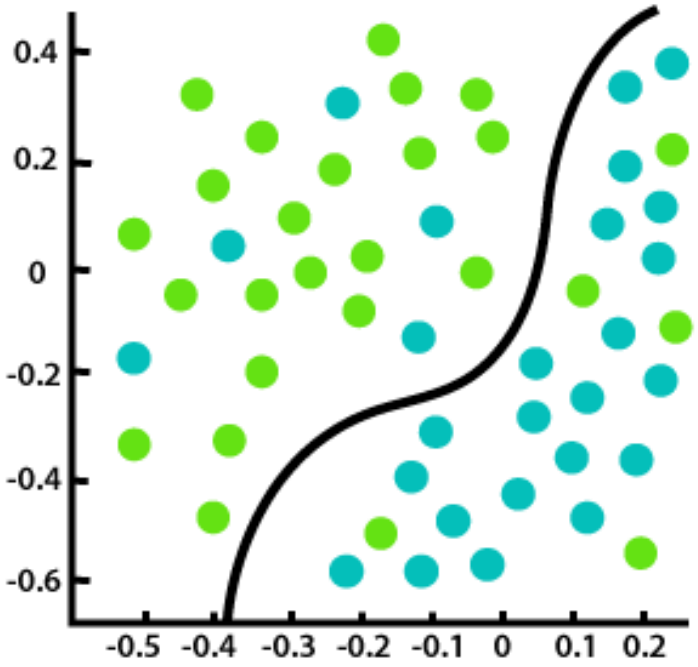
The function f is approximated by the finite sum

For each set of inputs $\vec{x} = (x_1, x_2, \cdots x_n)$,

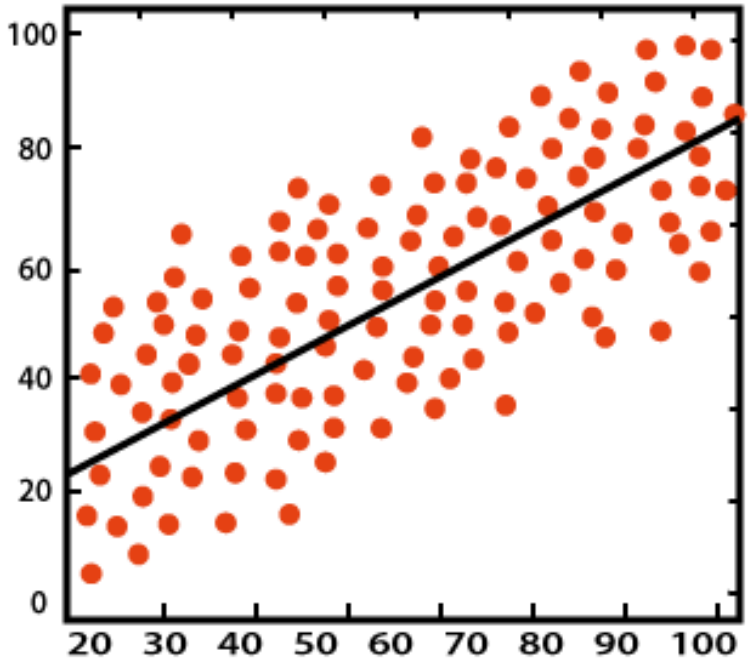
$$f(\vec{x}) = \sum_{j=1}^{N} \omega_k^{(j)} v_j + \text{A small term}$$



Universal Approximators

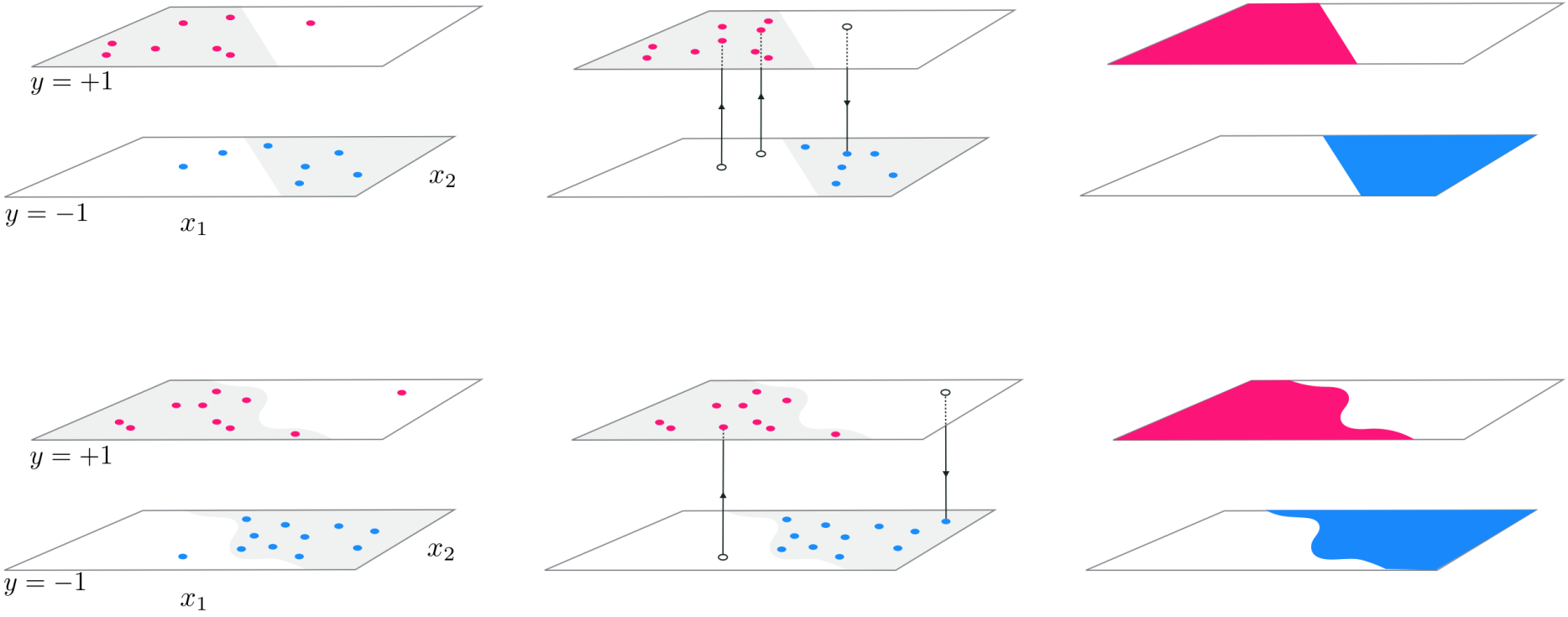


Classification

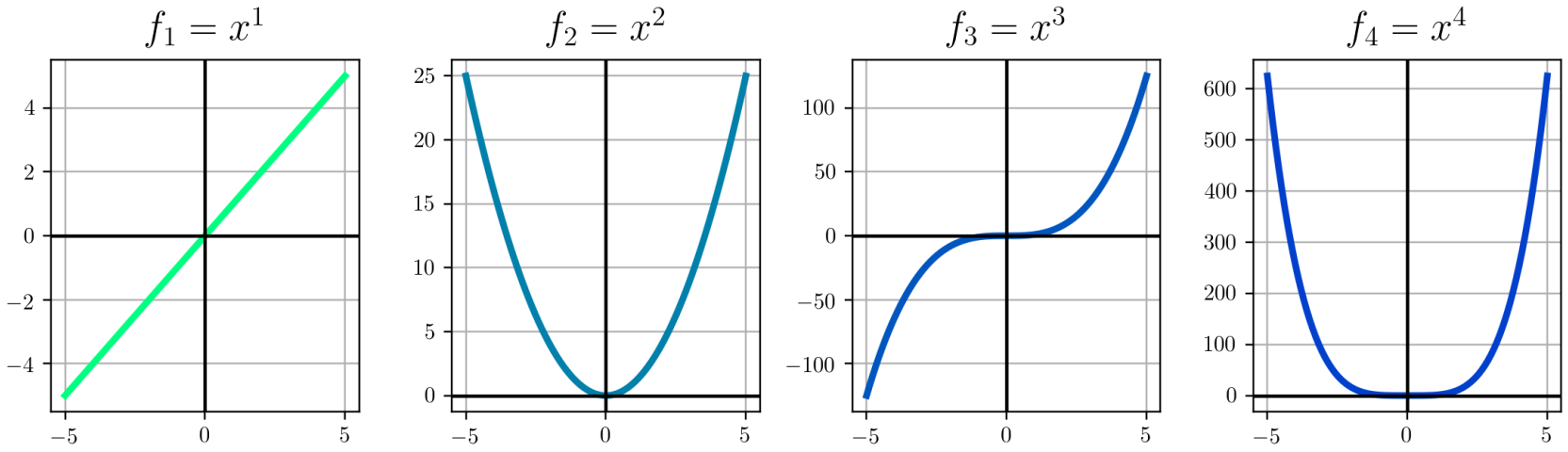


Regression

For any classification problem, our goal is to learn the optimal decision boundary that discriminates between classes in an arbitrarily large (i.e., high-dimensional) and complex space (manifold).



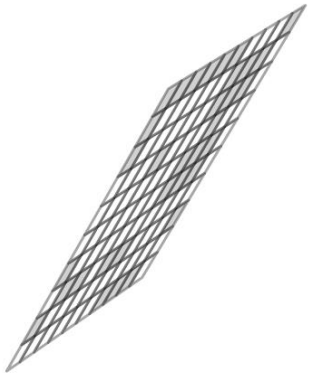
$$f_1(x) = x, \quad f_2(x) = x^2, \quad f_3(x) = x^3, \quad \text{etc.},$$



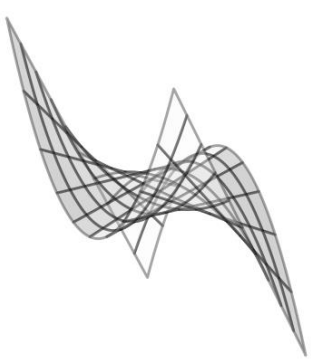
Polynomial regression is particularly powerful in approximate complex functions can approximate a

$$f_1(x) = x, \;\; f_2(x) = x^2, \;\; f_3(x) = x^3, \;\; \text{etc.},$$

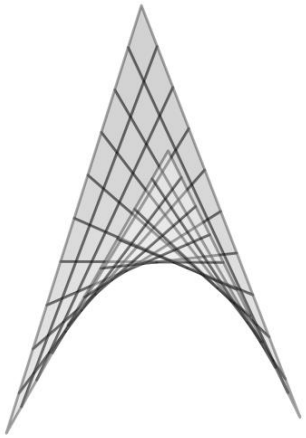
$f_1 = x_2$



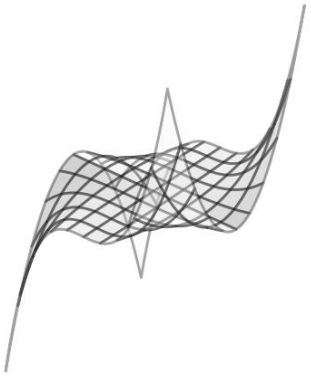
$f_2 = x_1 x_2^2$



$f_3 = x_1 x_2$

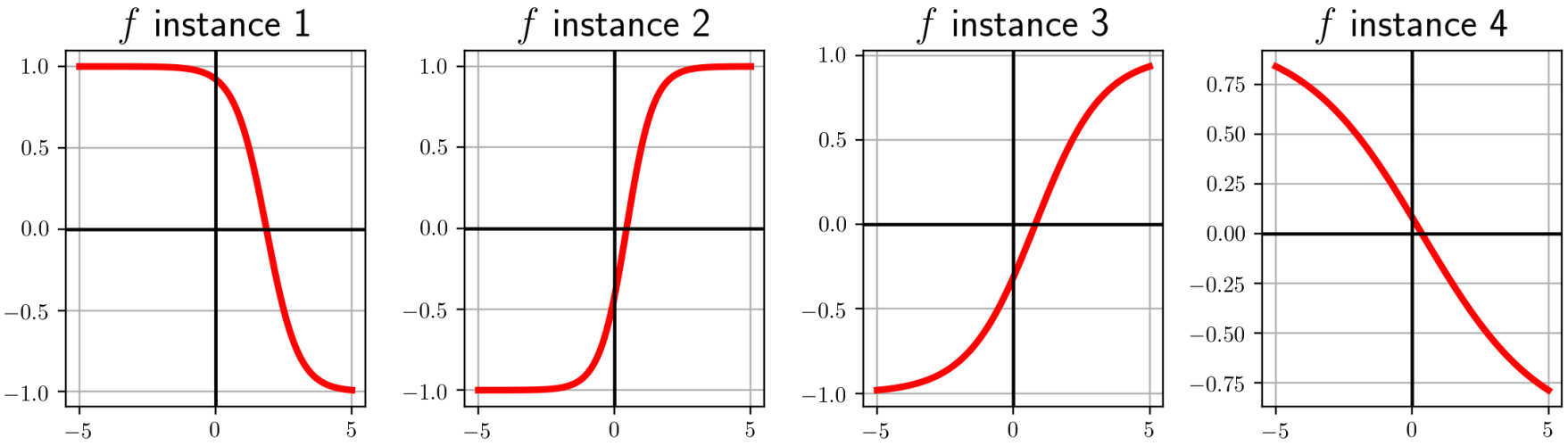


$f_4 = x_1^2 x_2^3$



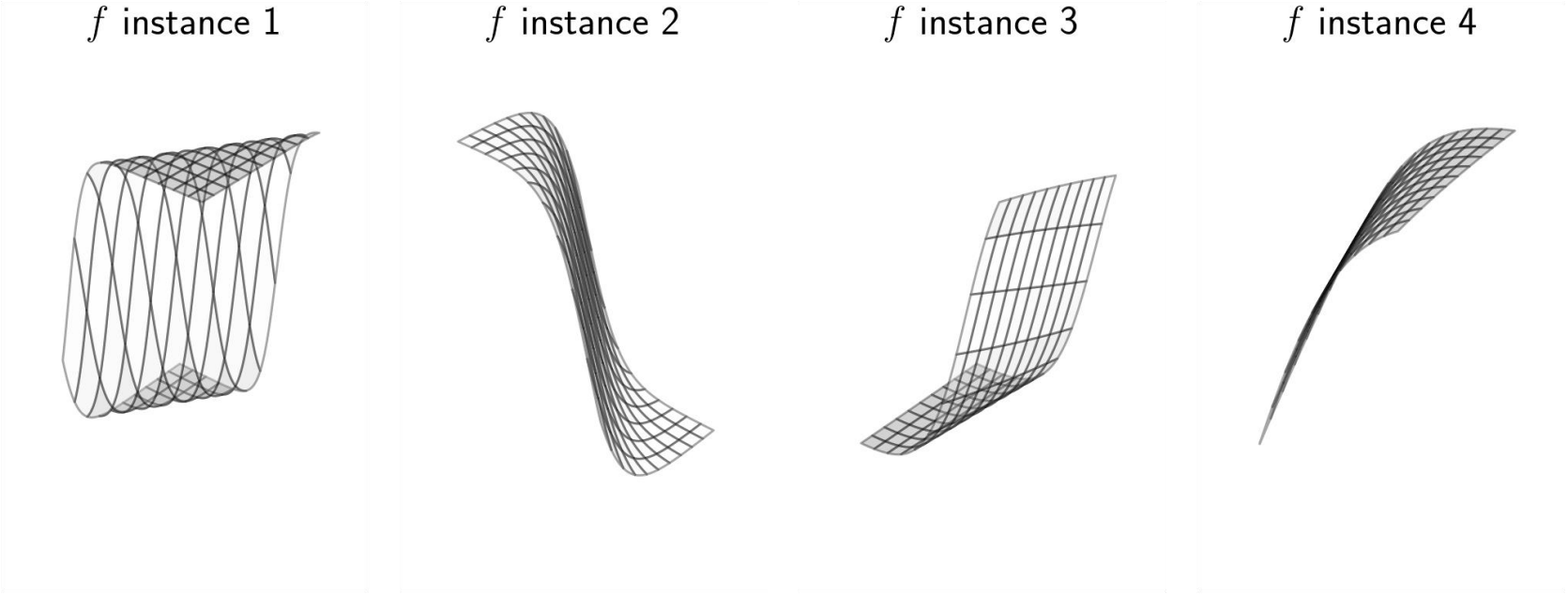
Neural Networks as Universal approximators

$$f_1(x) = \tanh\left(w_{1,0} + w_{1,1}x\right), \quad f_2(x) = \tanh\left(w_{2,0} + w_{2,1}x\right), \quad \text{etc.}$$



Neural Networks as Universal approximators

$$f_1(x) = \tanh\left(w_{1,0} + w_{1,1}x\right), \quad f_2(x) = \tanh\left(w_{2,0} + w_{2,1}x\right), \quad \text{etc.}$$

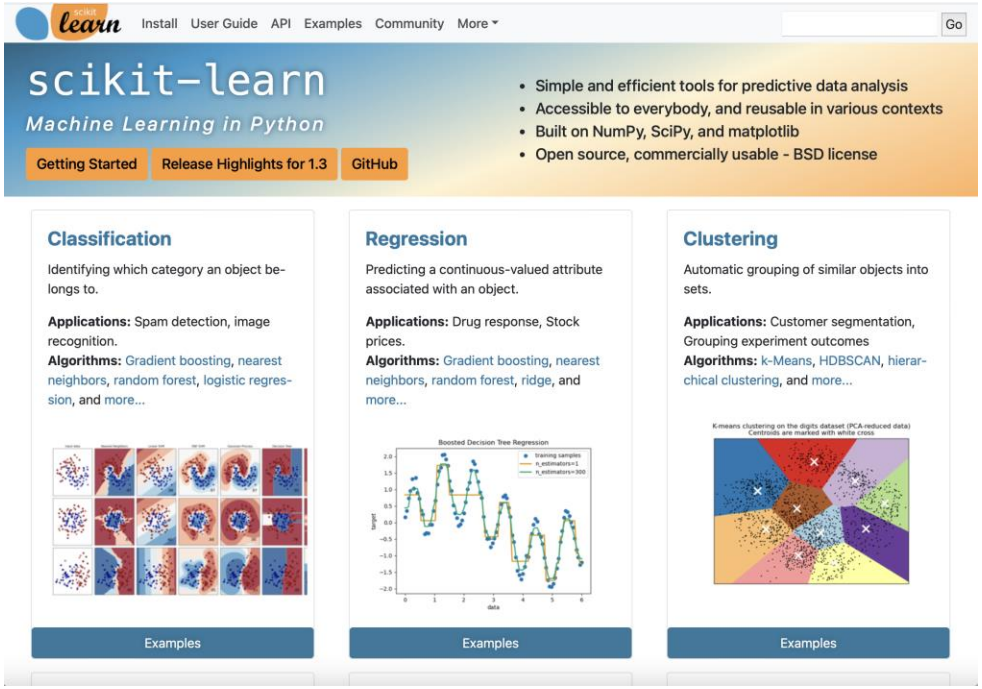


What MLP can solve?

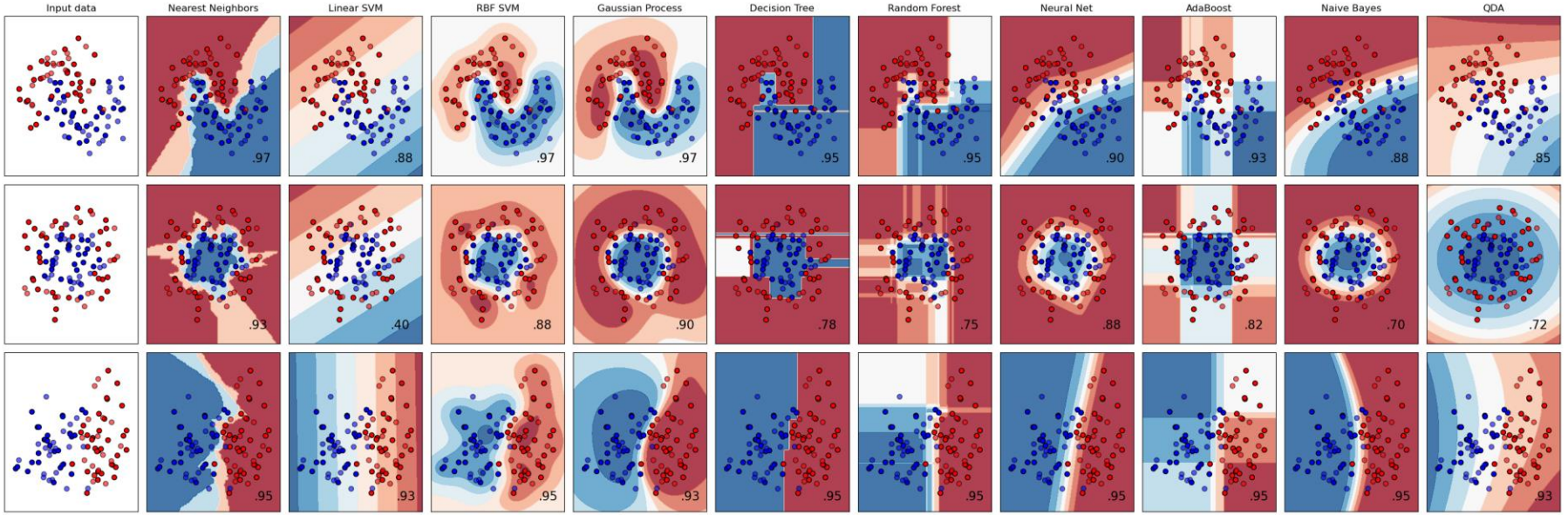
- MLP can deal with both regression and classification models, and will be competitive in general against other traditional ML models such as:
- Random Forest, Gaussian Process, (Kernel) Support Vector Machines, Gradient Boosting, nearest neighbors and so on.

Message to take home

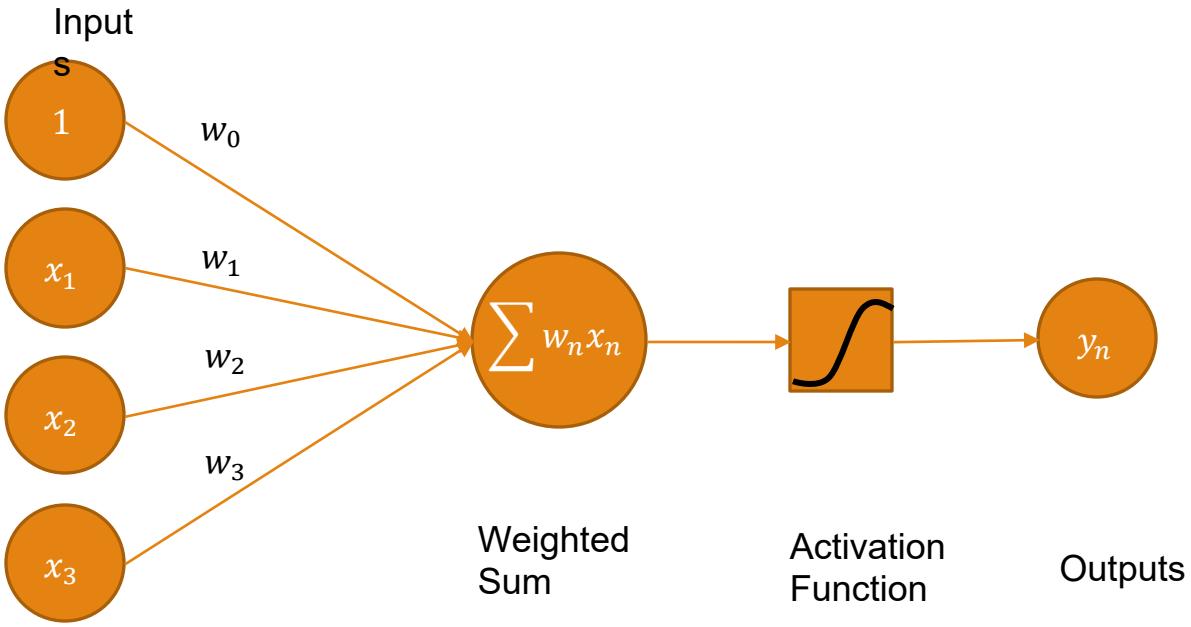
For most of the traditional regression and classification problems, in which your data is a table, you will do just fine using e.g. https://scikit-learn.org/stable/



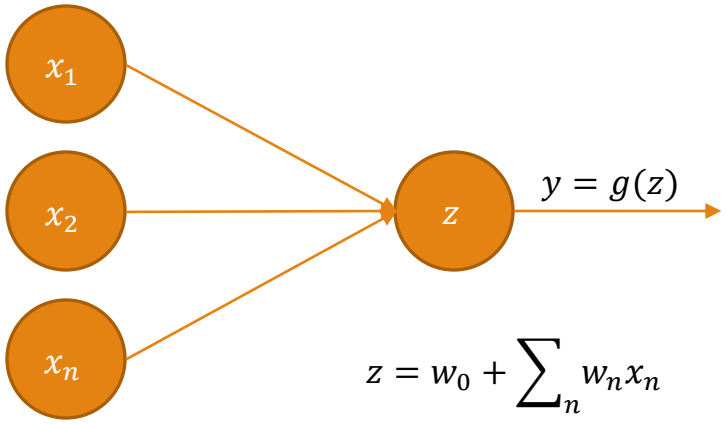
Classifier comparison



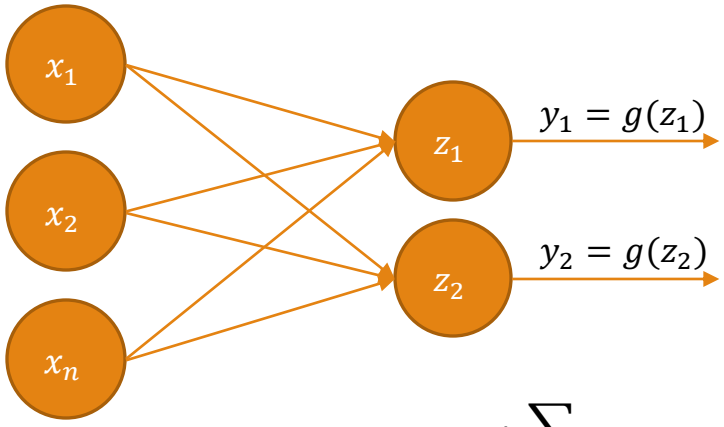
Perceptrons to Neural Networks



Perceptrons to Neural Networks

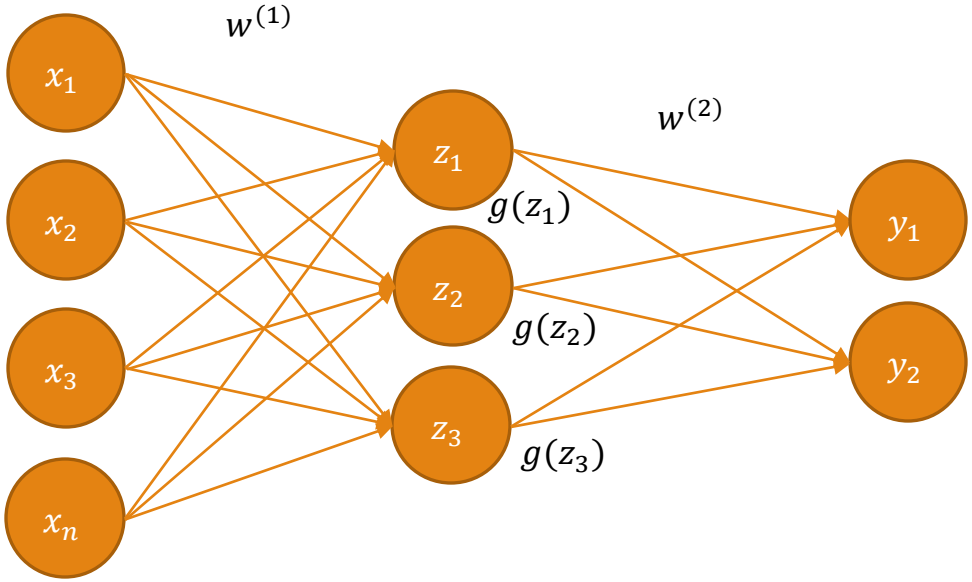


Perceptrons to Neural Networks



$$z_i = w_{i,0} + \sum_n w_{i,n} x_n$$

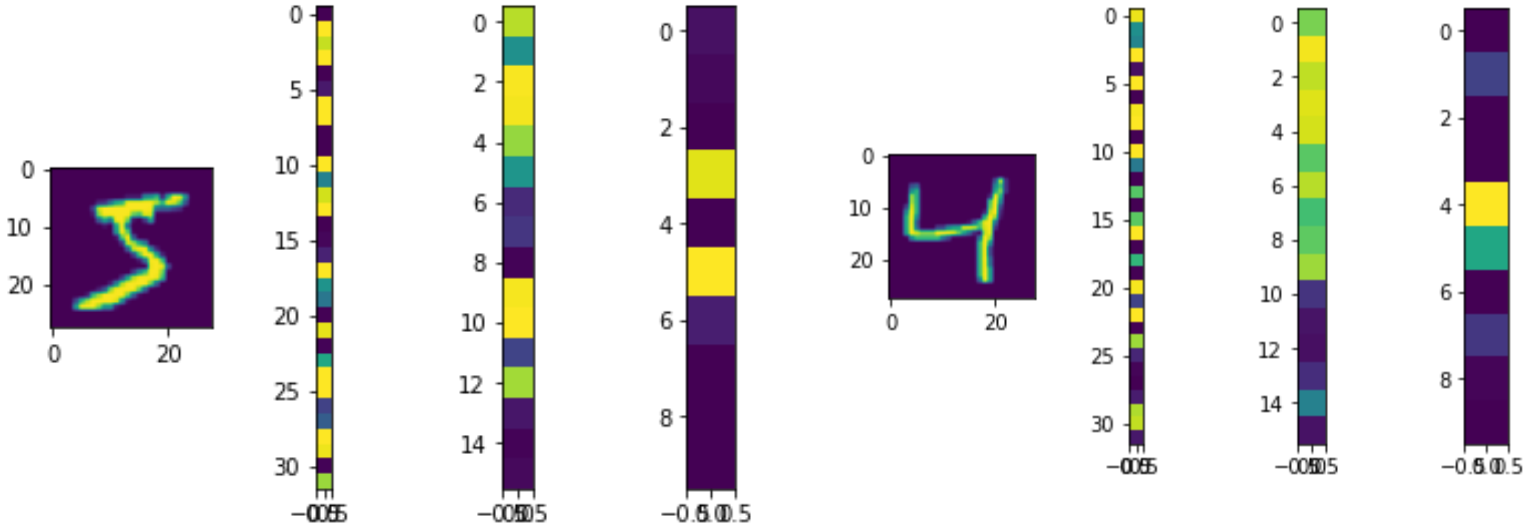
Single Hidden Layer Network



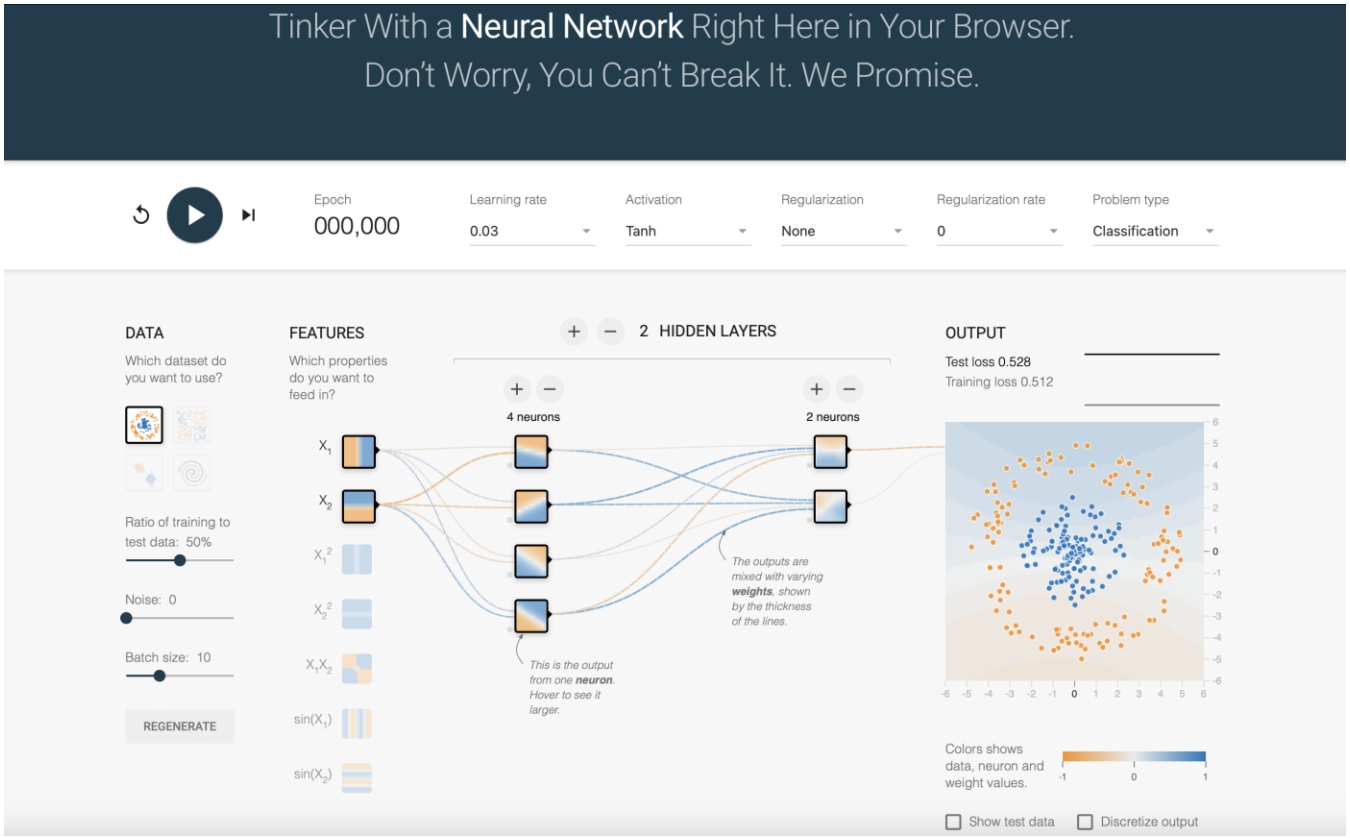
$$z_i = w_{i,0}^{(1)} + \sum_j w_{i,j}^{(1)} x_j$$

$$y_i = g\left(w_{i,0}^{(2)} + \sum_j w_{i,j}^{(2)} g(z_j) \right)$$

What happens inside a DNN?



Tensor Flow


Tinker With a Neural Network Right Here in Your Browser.
Don't Worry, You Can't Break It. We Promise.
Epoch
000,000
Learning rate
0.03
Activation
Tanh
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
2 HIDDEN LAYERS
4 neurons
2 neurons
The outputs are mixed with varying weights, shown by the thickness of the lines.
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.528
Training loss 0.512
Colors shows data, neuron and weight values.
Show test data
Discretize output

Non-Linearity and Activation Functions

Why do we need non-linearity

What is linearity

$$f(x+y) = f(x) + f(y)$$

$$f(ax) = af(x)$$

Why do we need non-linearity

What is linearity

$$f(x) = 10x$$

$$f(2) = f(-2) + f(4)$$

$$20 = -20 + 40$$

Why do we need non-linearity

Non-linearity

$$f(x) = 10x^2$$

$$f(2) \neq f(-2) + f(4)$$

$$40 \neq 40 + 160$$

Linearity in Neural Networks

$$y = f\left(b + \sum_{i=1}^{n} w_i x_i\right)$$

$$Layer\ 1$$

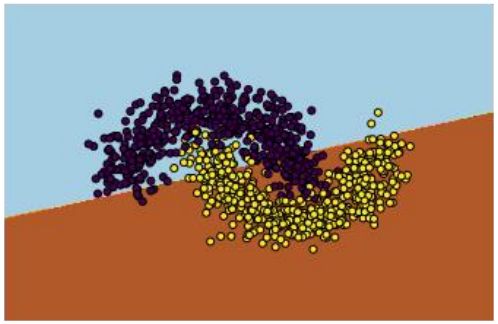
$$y_1 = w_{11}x_1 + w_{12}x_2$$
$$y_2 = w_{21}x_1 + w_{22}x_2$$

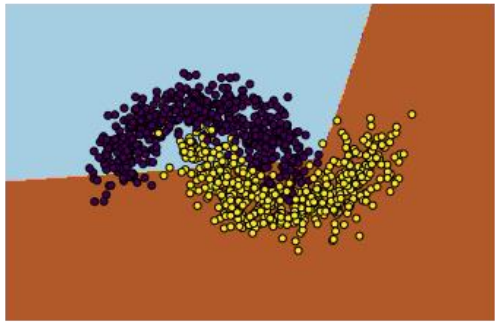
$$Layer\ 2$$

$$y_3 = w_1 y_1 + w_2 y_2$$
$$y_3 = w_1(w_{11}x_1 + w_{12}x_2) + w_2(w_{21}x_1 + w_{22}x_2)$$

Why deep learning?

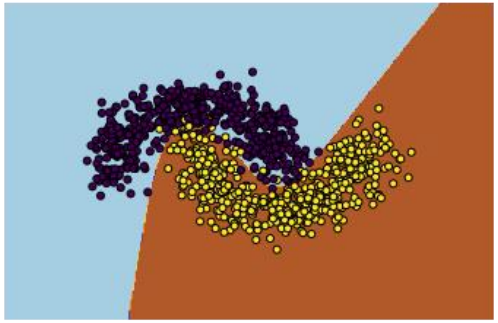
$N_{Neurons}$=1
Activation=Sigmoid

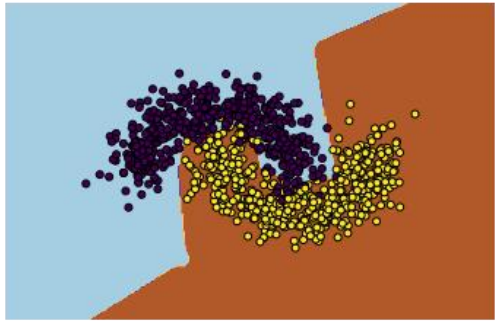




$N_{Neurons}$=2
Activation=Sigmoid

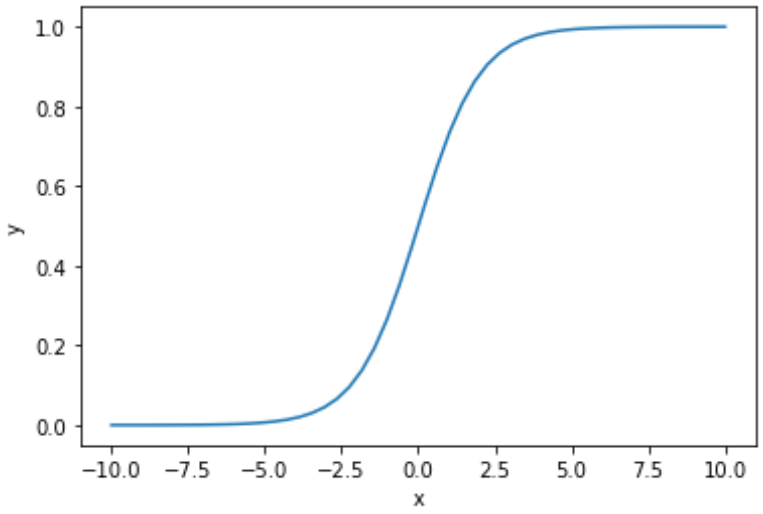
$N_{Neurons}$=3
Activation=Sigmoid





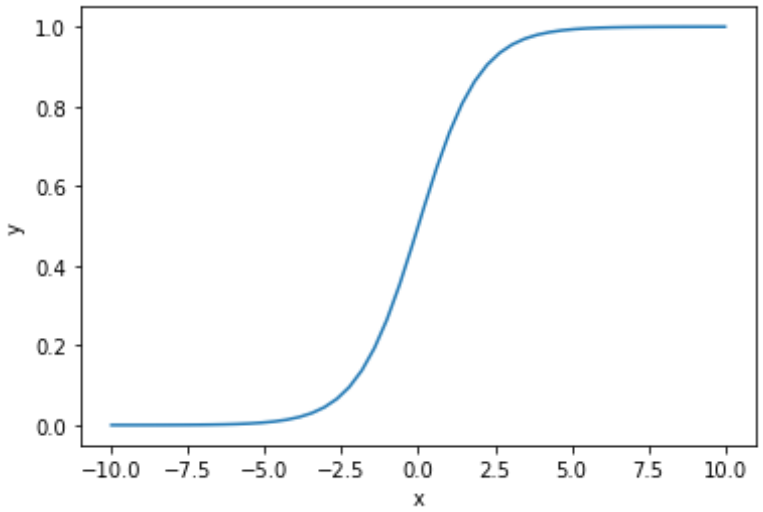
$N_{Neurons}$=8
Activation=Sigmoid

Sigmoid Function

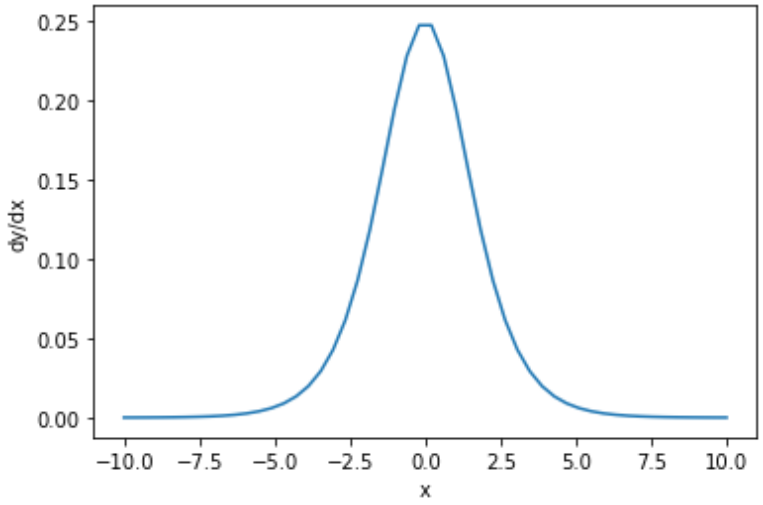


$$y = s(x) = \frac{1}{1 - e^{-x}}$$

Sigmoid Function

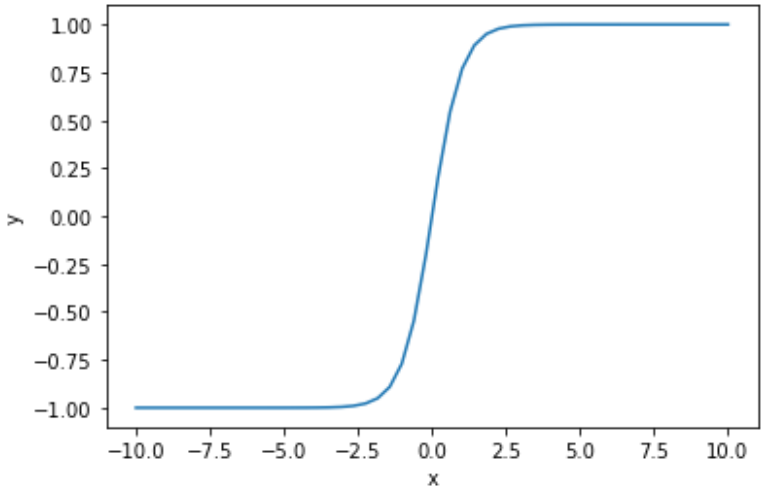


$$y = s(x) = \frac{1}{1 + e^{-x}}$$



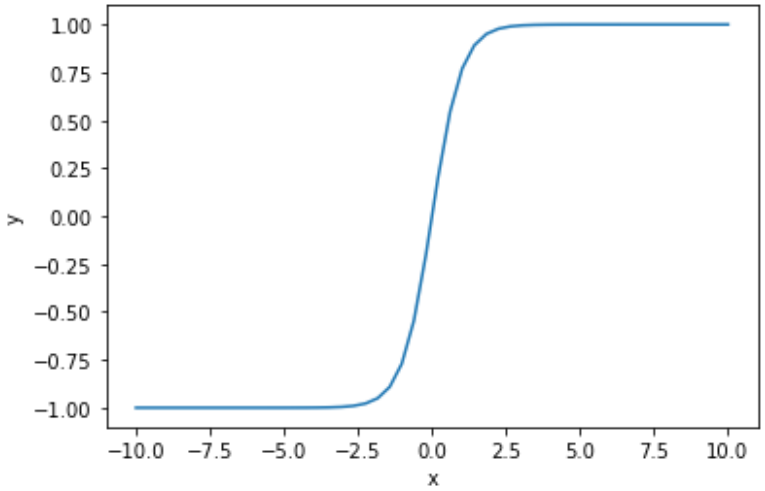
$$\frac{dy}{dx} = s(x)(1 - s(x))$$

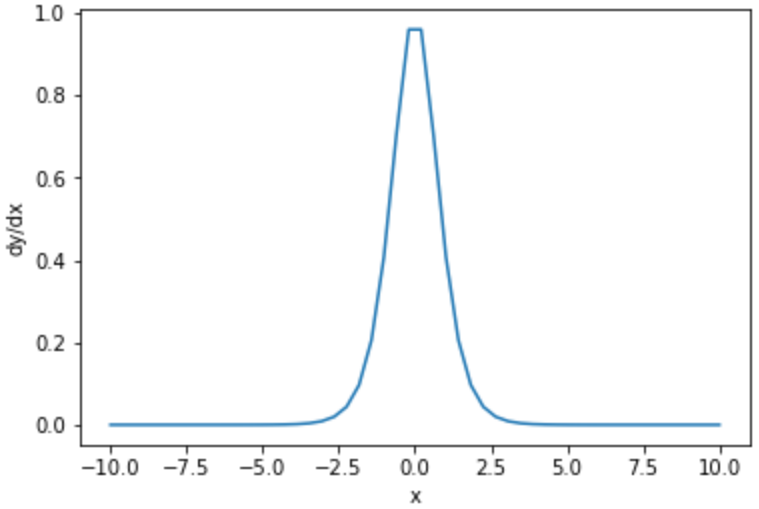
Hyperbolic Tan Function (Tanh)



$y = \tanh(x)$

Hyperbolic Tan Function (Tanh)

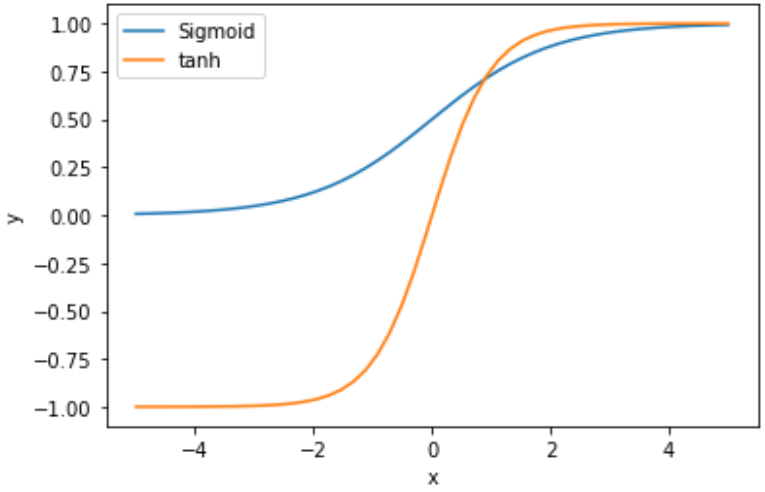


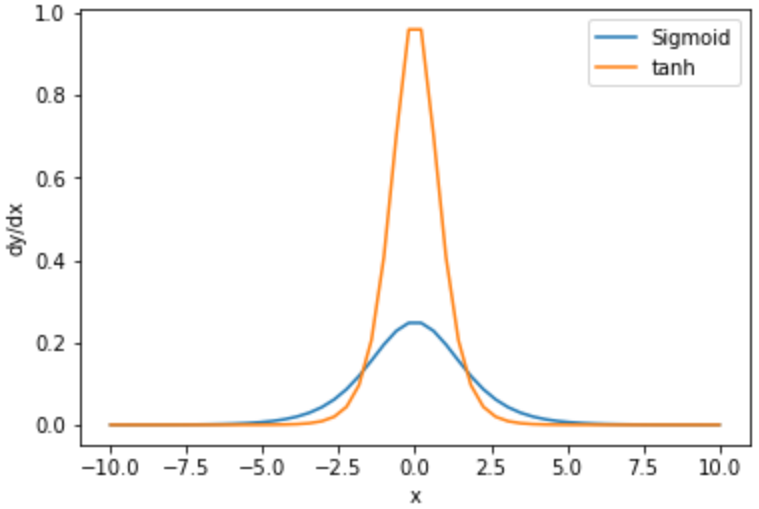


$$y = \tanh(x)$$

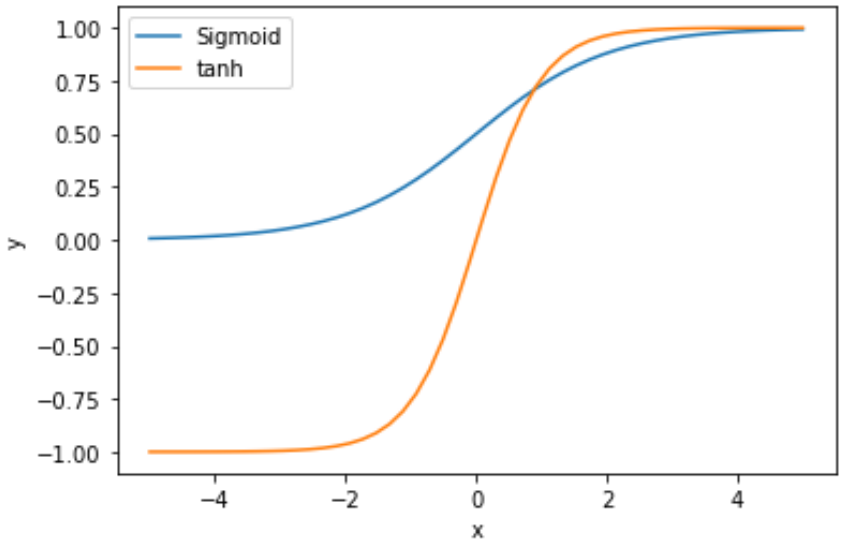
$$\frac{dy}{dx} = 1 - \tanh^2(x)$$

Sigmoid vs Tanh





Sigmoid vs Tanh

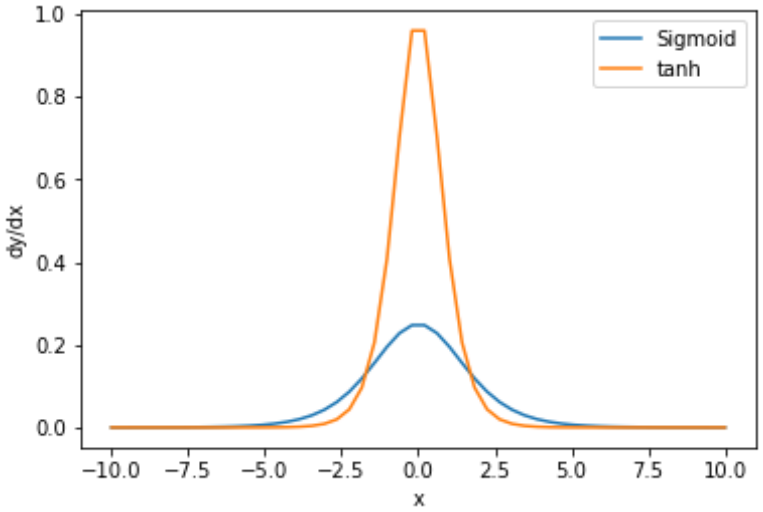


Both networks have an S-curve behaviour in their output values. The sigmoid function returns values between $(0,1)$ while the tanh function returns values between $(-1,1)$

The bounded outputs keep the gradients constrained, and smooth out the input range

The sigmoid function returns values that look like probabilities, and switches off the signal of a neuron when the output approaches 0

Sigmoid vs Tanh

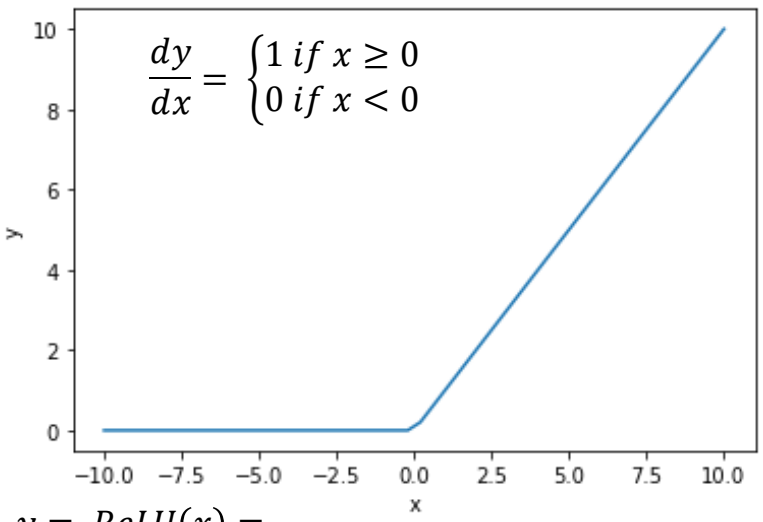


The tanh function has much stronger gradients than the sigmoid function

They both have the highest gradients when the input value is around 0, which has the effect of pushing the output value towards the boundaries of the functions

The gradient of the tanh function if four times stronger than the sigmoid function when $x = 0$

Rectified Linear Unit - ReLU



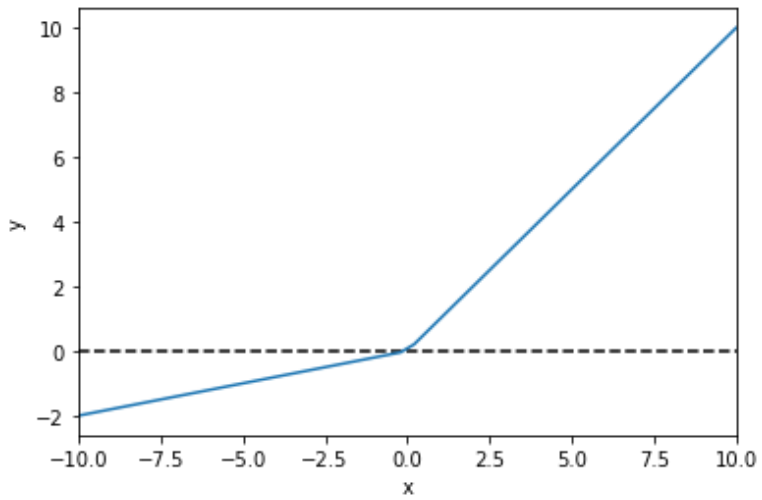
$$y = ReLU(x) = \begin{cases} x \; if \; x \geq 0 \\ 0 \; if \; x < 0 \end{cases}'$$

The ReLU unit is widely used, particularly in computer vision applications.

It is less affected by the vanishing gradient problem, because the gradients are only suppressed in one direction

Is computationally easy to calculate both the output and gradient

Leaky ReLU



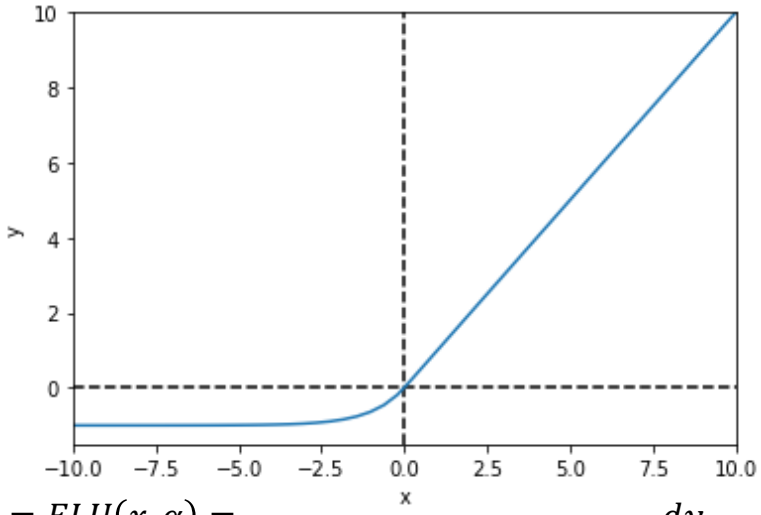
Sometimes the ReLU activated neurons can "Die", meaning they get stuck in the state of returning a value of 0. Because the ReLU unit has no gradient when the input is less than 0, the unit cannot recover from this state.

The Leaky ReLU function serves to prevent this issue, by allowing small gradients for negative input values. The scale of the gradients is determined prior to training.

$$y = Leaky\ ReLU(x, \alpha) = \begin{cases} x\ if\ x \geq 0 \\ \alpha x\ if\ x < 0 \end{cases}'$$

$$\frac{dy}{dx} = \begin{cases} 1\ if\ x \geq 0 \\ \alpha\ if\ x < 0 \end{cases}$$

Exponential Linear Unit - ELU



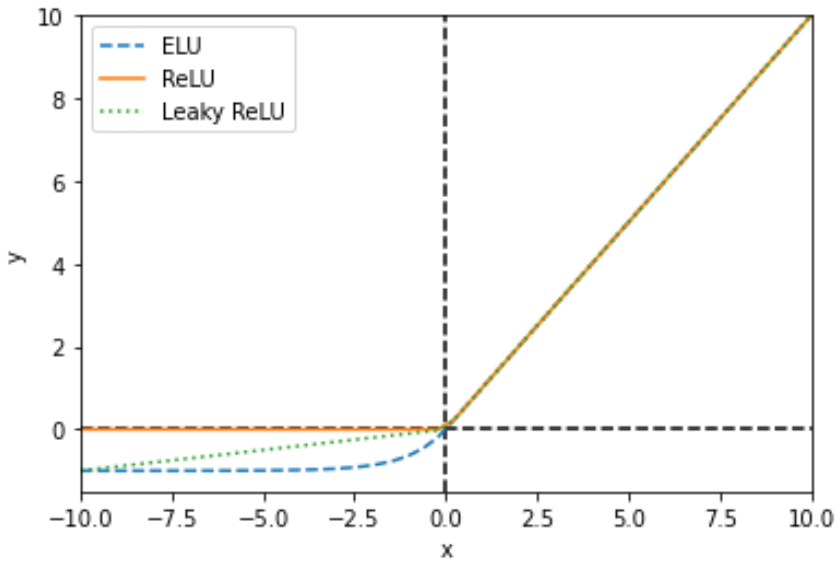
$$y = ELU(x, \alpha) = \begin{cases} x \; if \; x \geq 0 \\ \alpha(e^x - 1) \; if \; x < 0' \end{cases}$$

$$\frac{dy}{dx} = \begin{cases} 1 \; if \; x \geq 0 \\ \alpha e^x \; if \; x < 0 \end{cases}$$

In the same family as ReLU and Leaky ReLU, is the ELU function. Like Leaky ReLU, this function allows for negative inputs by scaling them with an exponential function.

The negative gradients smoothly approach 0, unlike ReLU which abruptly changes.

ELU, ReLU and Leaky ReLU



All of these functions train quickly, due to their unconstrained gradients, but likewise can have exploding outputs that approach ∞.

Should only be used for the hidden layers in a network, due to the partial linearities in their outputs.

Computationally cheap compared to Sigmoid and Tanh.

Softmax

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

The Softmax function is not one that is used in hidden layers of neural networks, but is often applied to the final layer of a multi-class classification network.

For each element in the output tensor, $\boldsymbol{z}$, the softmax function calculates the exponential of that element, scaled by the sum of the exponential of all elements in the output.

The resulting tensor has the property of always summing up to 1—this makes the softmax function crucial for calculating probabilities of multi-class predictions.