Ciência de Dados Quântica 22/23

Classical Machine Learning:
Regression & Classification

LUÍS PAULO SANTOS

Material de Consulta

- Maria Schuld, Francesco Petruccione
  "***Machine Learning with Quantum Computers***"
  Cap. 2

- Luís G. Serrano
  "**grokking *Machine Learning***"
  Caps. 2, 3, 4, 5, 6, 7 e apêndice B

An attempt on definition

"Machine learning can be seen as one particular flavour of AI, which **designs computer algorithms** that generalise **from patterns found in data** to make predictions in unknown situations."

"The attraction of machine learning, therefore, lies in its capability to learn from **extremely large data sets** that only computers can process."

[Schuld2021 – chapter 2]

An attempt on definition

"Machine learning **starts** with a very **general, agnostic mathematical model**, and **uses data to adapt it** to the case, a process called "**training**". "

When **looking at the final model**, **we do not** necessarily **gain information on the physical mechanism or statistical relationships behind the data**, but consider it as a **black box** that has learned to **produce reliable outputs**.

[Schuld2021 – chapter 2]

Contrast this with **Explainable AI** (**XAI**): https://en.wikipedia.org/wiki/Explainable_artificial_intelligence

Fundamental concepts

- **Problem:**
  defined by an **unknown** function f^* that relates every point x in the **domain** $\mathcal{X}$ to a value y in the **codomain** $\mathcal{Y}$:

$$f^*: \mathcal{X} \rightarrow \mathcal{Y}$$

- **Regression:**
  $\mathcal{Y}$ is continuous

- **Classification:**
  $\mathcal{Y}$ is discrete

Fundamental concepts

- **Data set:**
  set of points $\mathcal{D}$ in either $\mathcal{X}$ or $\mathcal{X} \otimes \mathcal{Y}$ available to the machine learning algorithm
  - $\mathcal{D} \in \mathcal{X} = \{x^1, \cdots, x^M\}$: unlabelled data
  - $\mathcal{D} \in \mathcal{X} \otimes \mathcal{Y} = \{(x^1, y^1), \cdots, (x^M, y^M)\}$: labelled data

- **Features:**
  $\mathcal{X}$ is n-dimensional, meaning that $\mathrm{x} \in \mathcal{X}$ is a vector with n elements

$$x = (x_1, x_2, \cdots, x_n) \in \mathcal{X}$$

Fundamental concepts

- **Model:**

  $f: \mathcal{X} \rightarrow \mathcal{Y}$, which "*approximates*" f^*

- **Model Family:**

  $\mathcal{F} = \{f_\theta\}$, with $f_\theta: \mathcal{X} \rightarrow \mathcal{Y}$ and

  $\theta = \{\theta_0, \theta_1, \cdots, \theta_T\}$ is a T-dimensional vector that parameterizes f_θ

- **Loss:**

  $L(f_\theta(x), f^*(x) = y)$ quantifies the disparity between the model prediction $f_\theta(x)$ and the true value $y = f^*(x)$

Fundamental concepts

- **Risk (aka generalization error):**

  The risk of a model f_θ is the expected value of the loss over the whole domain $\mathcal{X}$:

$$R_{f_\theta} = \int_{\mathcal{X}} \mathrm{L}\big(f_\theta(x), f^*(x)\big)dx$$

- **ML goal:**

  find the model f_{θ^*} which minimizes the generalization error over $\mathcal{X}$:

$$f_{\theta^*} = \underset{f_\theta \in \mathcal{F}}{\operatorname{argmin}} \int_{\mathcal{X}} \mathrm{L}\big(f_\theta(x), f^*(x)\big)dx$$

Fundamental concepts

- **Empirical Risk :**

  the expected value of f_θ's loss over $\mathcal{D}$:

$$\hat{R}_{f_\theta}(\mathcal{D}) = \frac{1}{M}\sum_{i=1}^{M} \mathrm{L}\left(f_\theta(x^i), y^i\right)$$

- **ML's practical goal:**

  find the model $\hat{f}_\theta$ which minimizes the empirical risk $\hat{R}_{f_\theta}(\mathcal{D})$:

$$\hat{f}_\theta = \underset{f_\theta \in \mathcal{F}}{\operatorname{argmin}} \sum_{i=1}^{M} \mathrm{L}\left(f_\theta(x^i), y^i\right)$$

Classification of ML

- Supervised Learning : labelled data set
- Unsupervised Learning: unlabelled data set
- Reinforcement Learning: interactive based on feedback

Types of ML - **Supervised Learning**

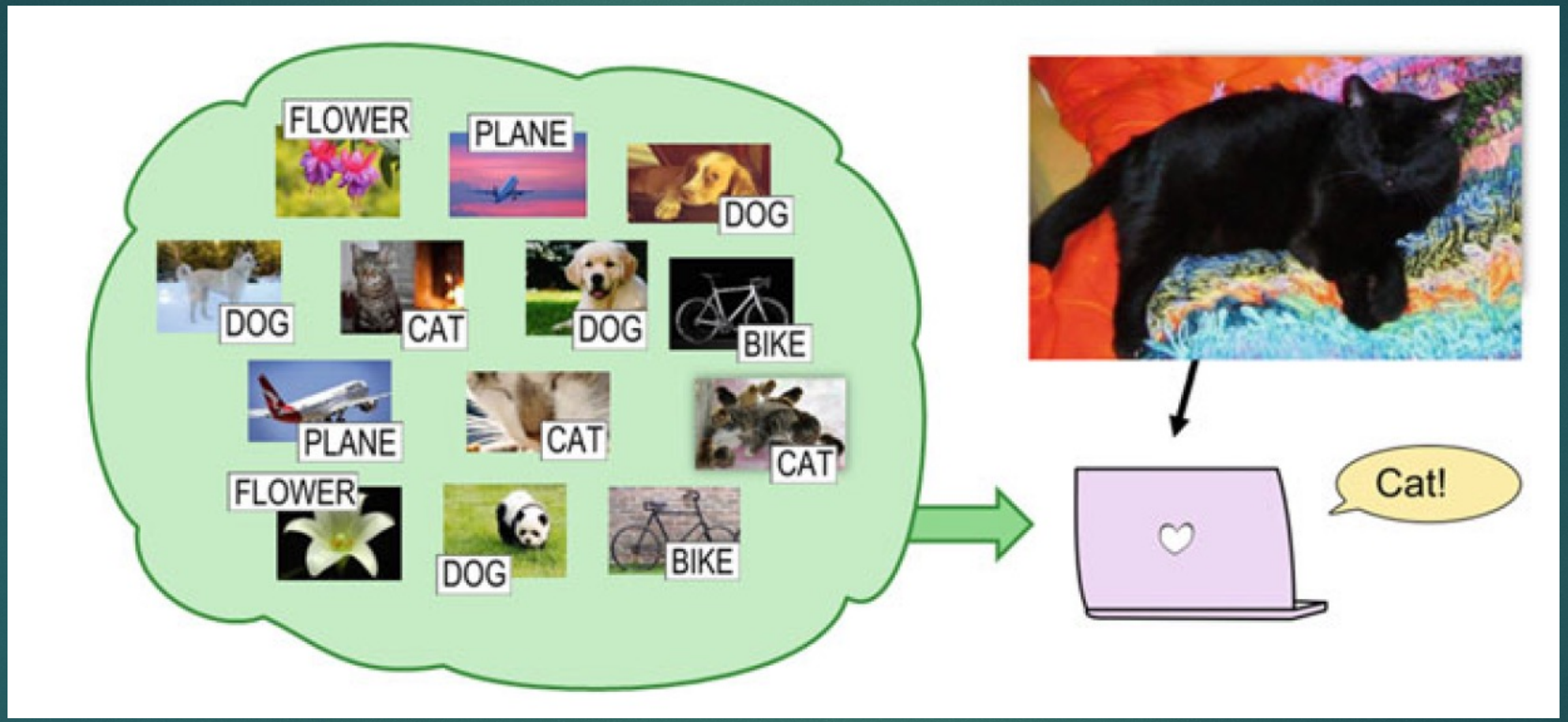
- **Definition:**

  Given a **labelled data set** $\mathcal{D} \in \mathcal{X} \otimes \mathcal{Y} = \{(x^1, y^1), \cdots, (x^M, y^M)\}$

  as well as a **new unclassified** point $\mathbf{x} \in \mathcal{X}$,

  predict the corresponding output $\mathrm{y} \in \mathcal{Y}$.

Supervised Learning Examples



[Schuld2021 – chapter 2]

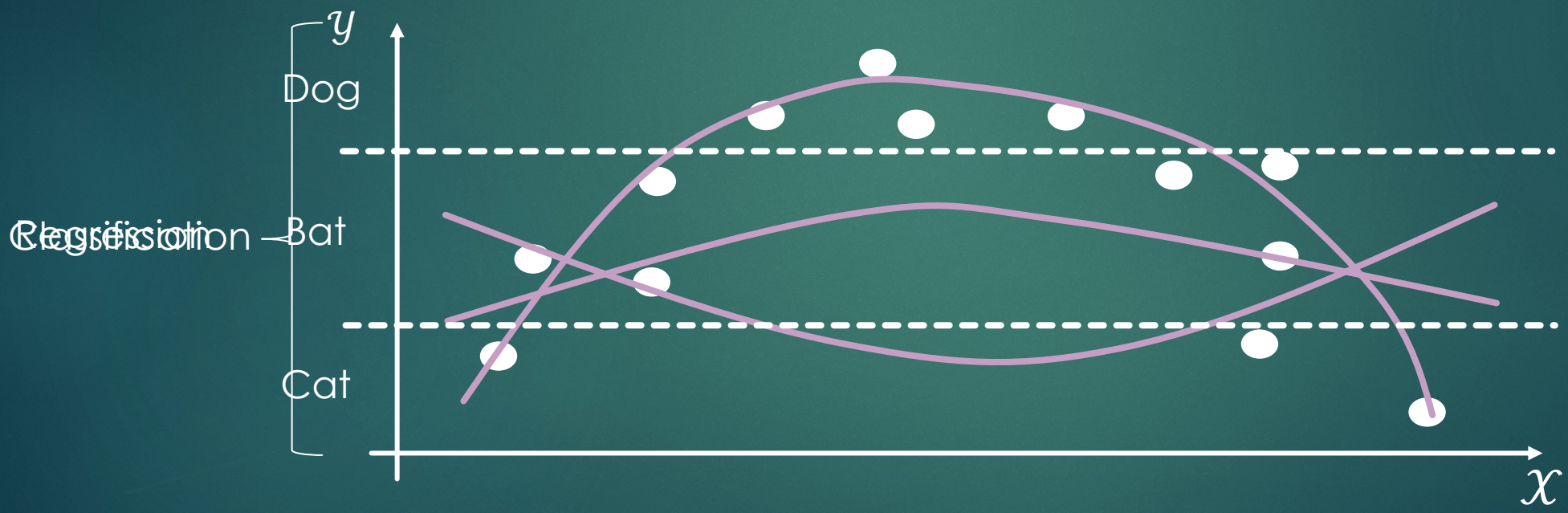
Supervised Learning Examples

- Time Series

What will be the value of C for time=20, given A and B ?

Supervised Learning

- Learns a model from the training data set which is used to predict unseen data points



Supervised Learning Algorithms

- Support-vector machines.
- Linear regression.
- Random forest.
- Naive Bayes.
- Linear discriminant analysis.
- Decision trees.
- K-nearest neighbor algorithm.
- Artificial Neural Networks

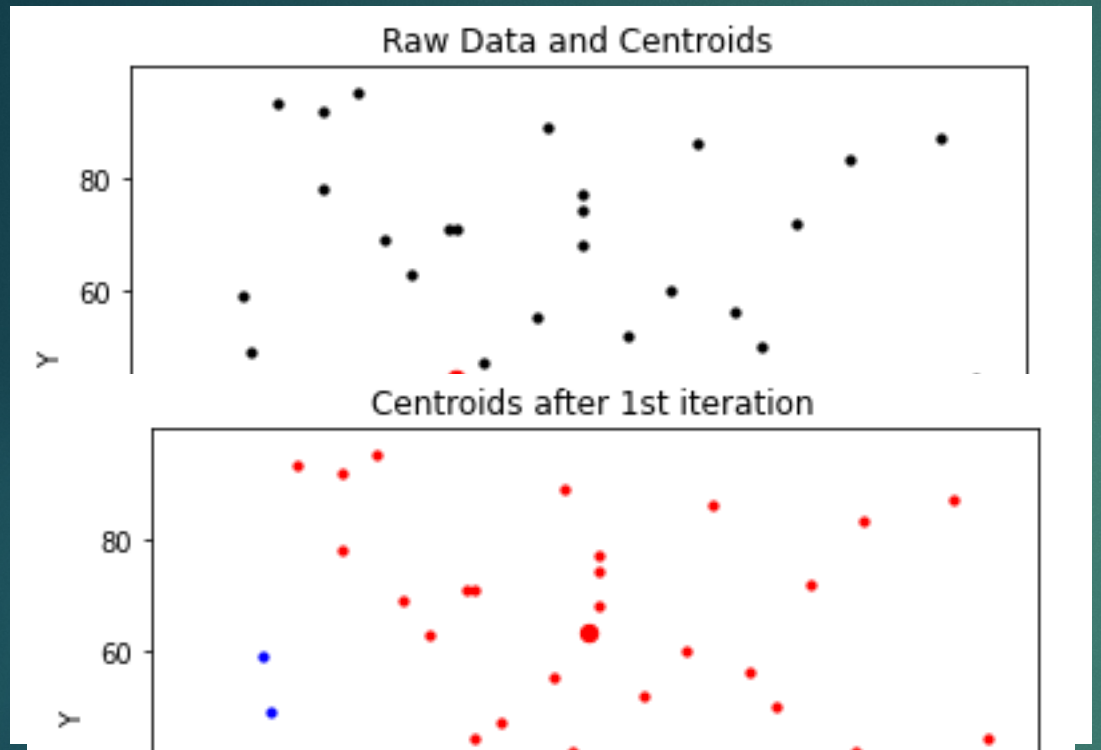
Types of ML - **Unsupervised Learning**

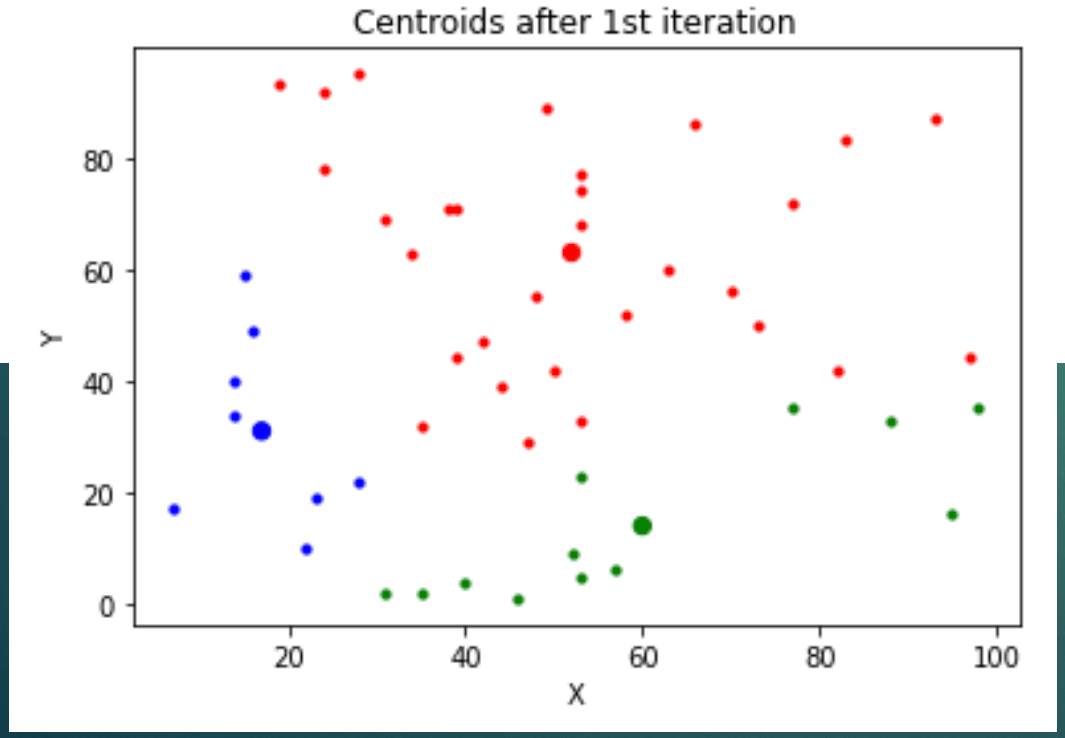
- **Definition:**

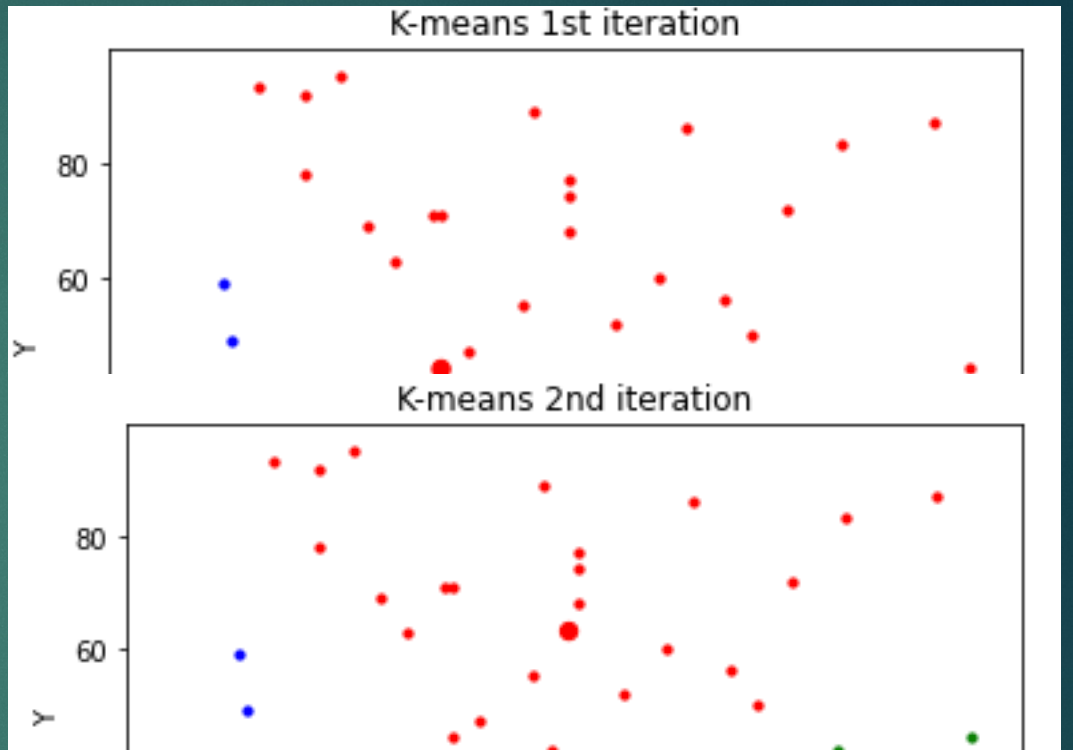
  Given an **unlabelled data set** $\mathcal{D} \in \mathcal{X} = \{x^1, \cdots, x^M\}$ drawn from a probability distribution $p(x)$:

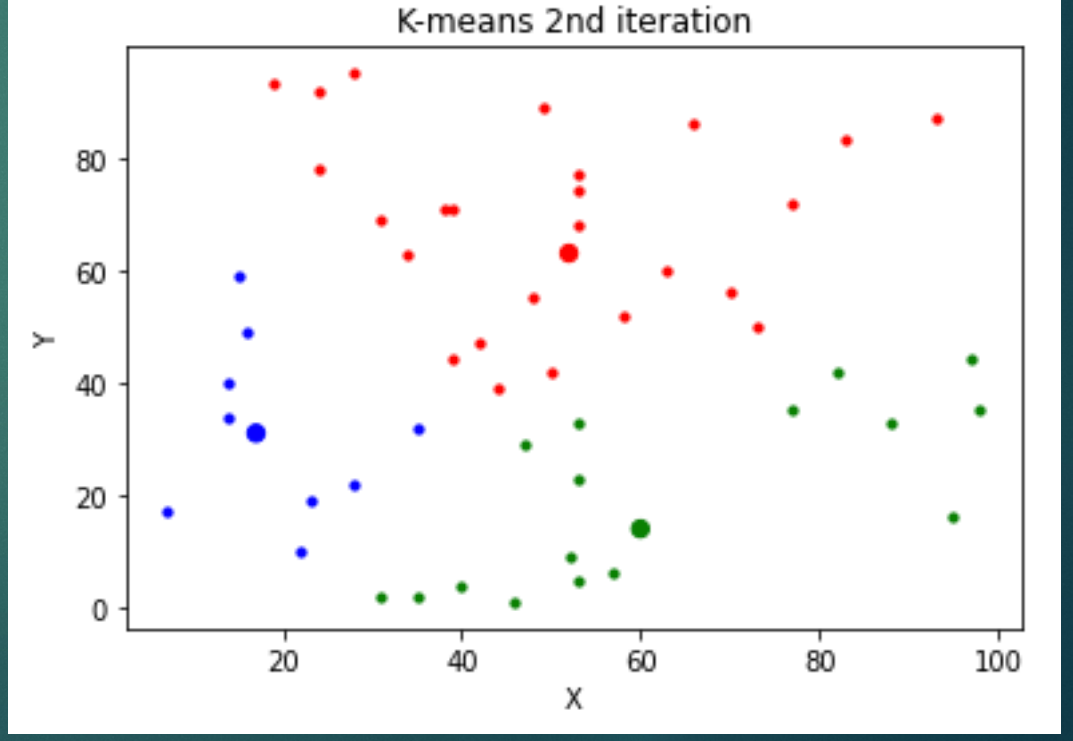
  - group the points according to some similarity measure – clustering
  - eliminate/combine features in $\mathcal{X}$ – dimensionality reduction
  - **draw a new sample** from $p(x)$ – generative approach

Unsupervised Learning - Clustering

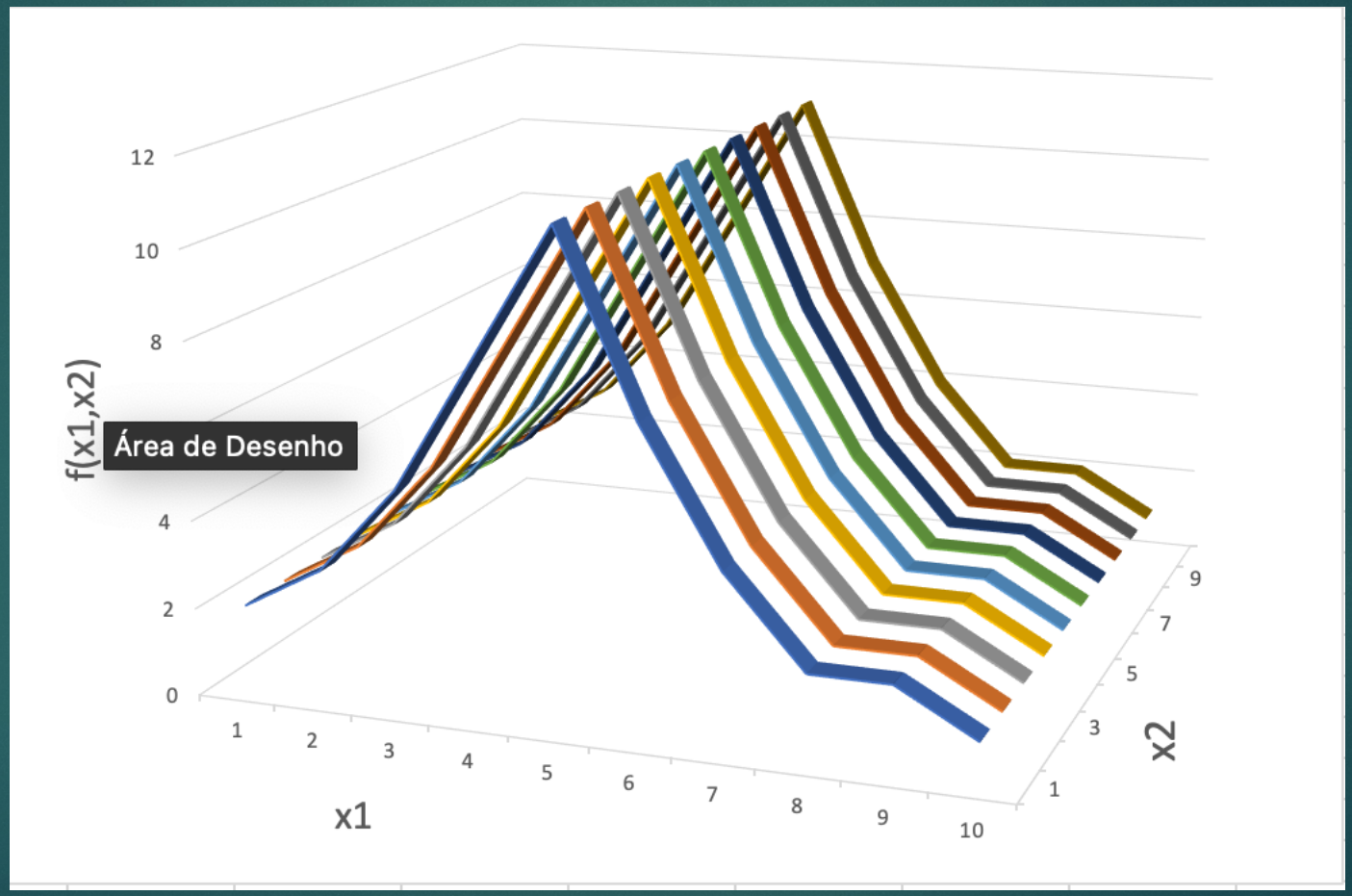








Unsupervised Learning – Dimensionality Reduction

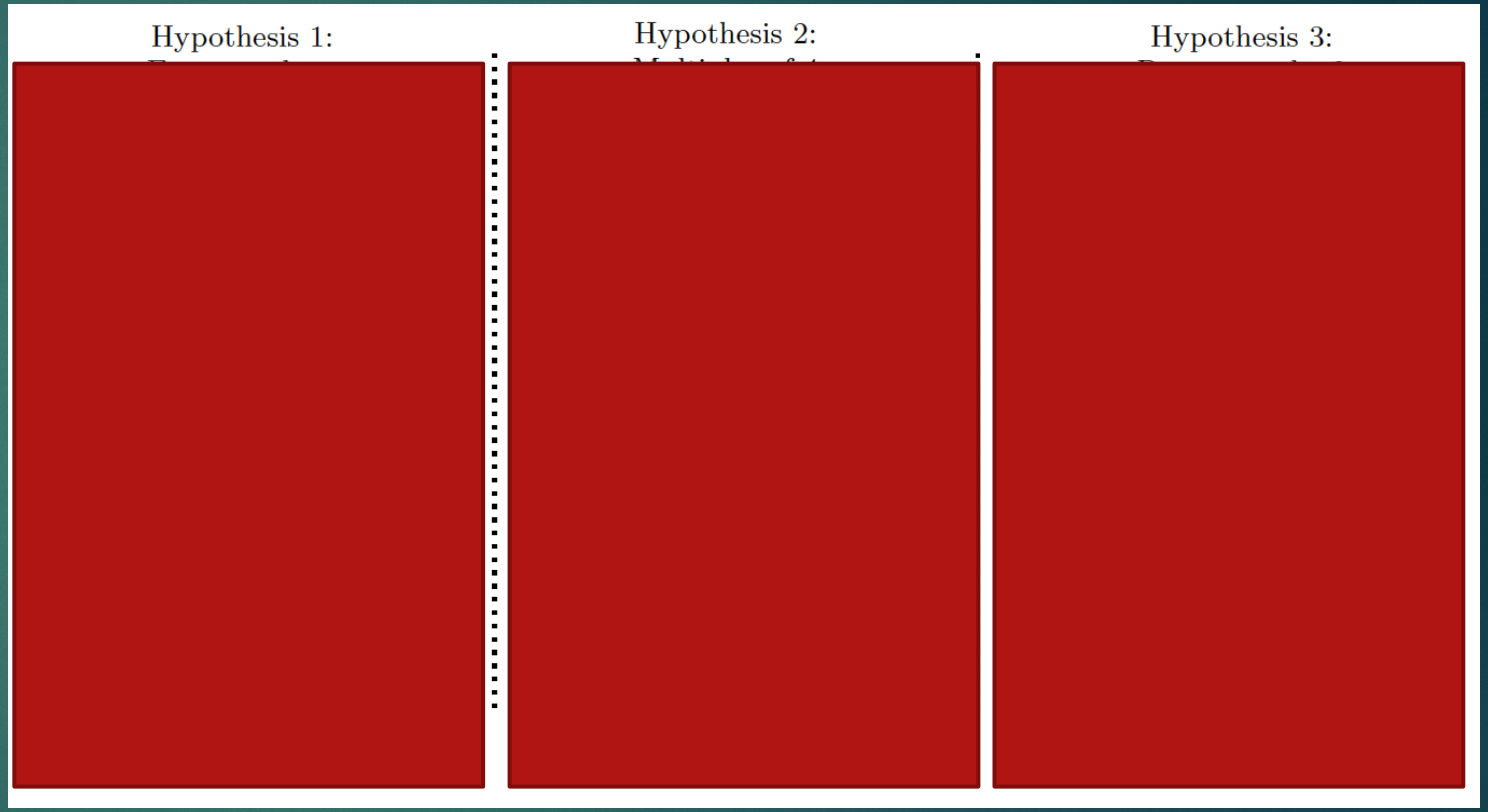


Unsupervised Learning - generative

- **Hypothesis guessing:**

In the range $i \in \{1 \cdots 100\}, i \in \mathbb{N}$, the values $\{4,16,36,100\}$ are given according to a (unknown) rule.

Generate another sample with the same rule.



Unsupervised Learning - generative

DALL-E

A photo of a white fur monster standing in a purple room

Chat GPT

podes escrever a receita de bacalhau à Brás no estilo de Luís de Camões?

"Oh, bacalhau, peixe digno de um prato
Que no mar do Norte é encontrado
E agora, em nossa terra, será trato
Com o tempero e o azeite profundo.

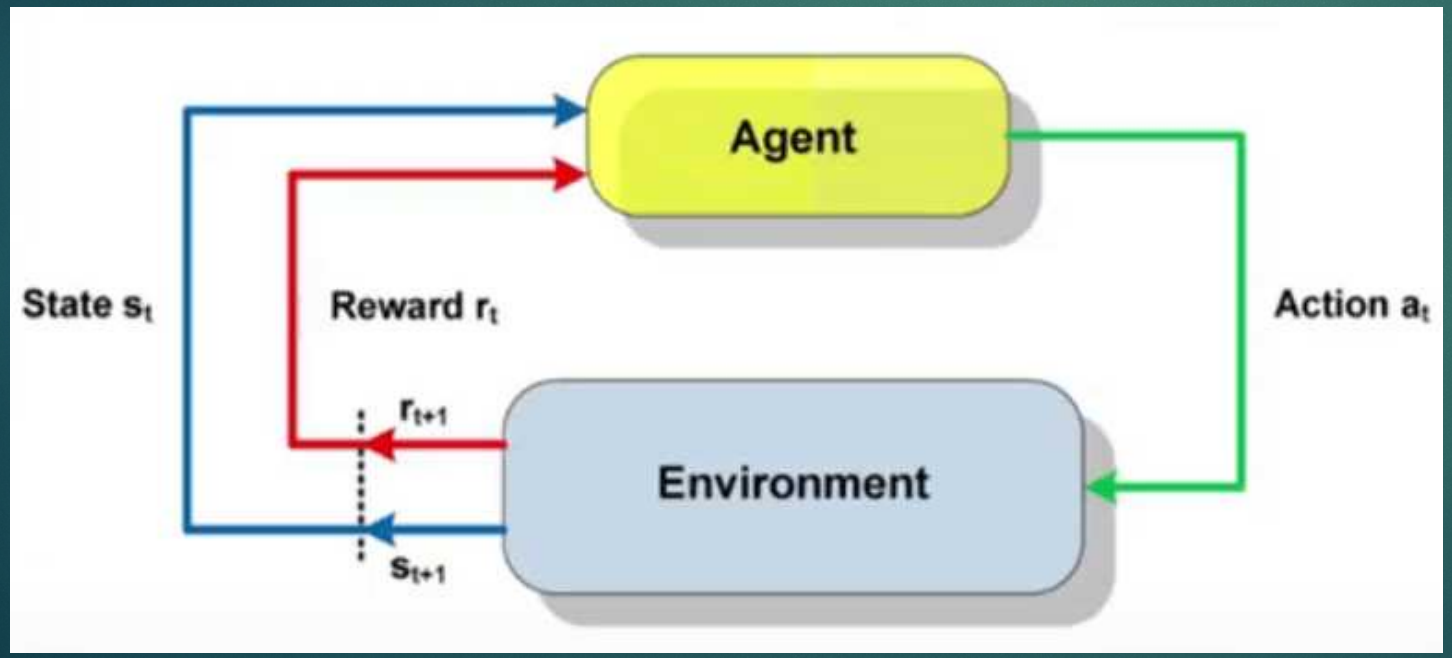
Com batatas, cebolas, alho e pimentões
Se misturarão em uma panela fervente
E em seu interior, ovos serão batidos
Até se tornarem cozidos.

E assim, a mistura será apresentada
Com sabor forte e aroma que encanta
E quem a experimentar, será conquistado
Pela iguaria de virtude tanta.

Sal e pimenta, temperos indispensáveis
Comprarão ainda mais brilho à refeição
E farão que o bacalhau, verdadeiro rei
Seja degustado com alegria e devoção."

Types of ML - **Reinforcement Learning**

- The agent learns good strategies by trial and error.
  Every time it acts on the environment it receives a reward.
  A good strategy maximizes the total received reward.
- The agent might learn a policy, indicating, for each state, which action will maximize the total (expected) reward.



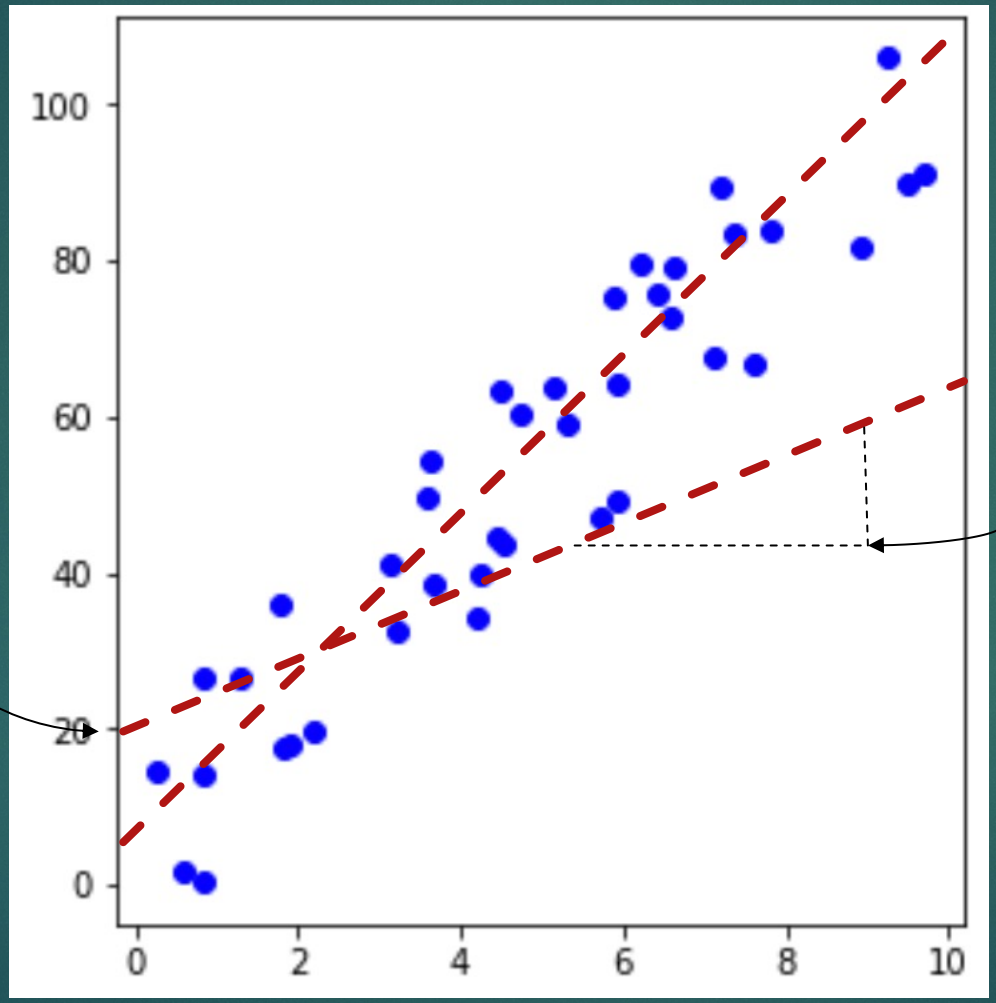
Regression

Supervised Learning: Regression

- Problem:
  given a labelled dataset $\mathcal{D} \in \mathcal{X} \otimes \mathcal{Y}$, taken from an unknown function $f^*: \mathcal{X} \to \mathcal{Y}$, **learn** a function $\hat{f}: \mathcal{X} \to \mathcal{Y}$ which approximates f^* and allows predicting $y = \hat{f}(x)$ for previously unseen (x, y), i.e. $(x, y) \notin \mathcal{D}$
- Learnable functions f_θ are parameterized with θ being a vector of parameters. **For linear regression** $\mathrm{y} = f_\theta(x) = \theta_0 + \sum_{i=1}^{n} \theta_i * x_i$,
  with the number of parameters $T = n + 1$, n being the number of features
- For uni-dimensional data (n=1) and linear regression lines are used:

$$y = f_\theta(x) = \theta_0 + \theta_1 * x_1$$

Supervised Learning: Regression



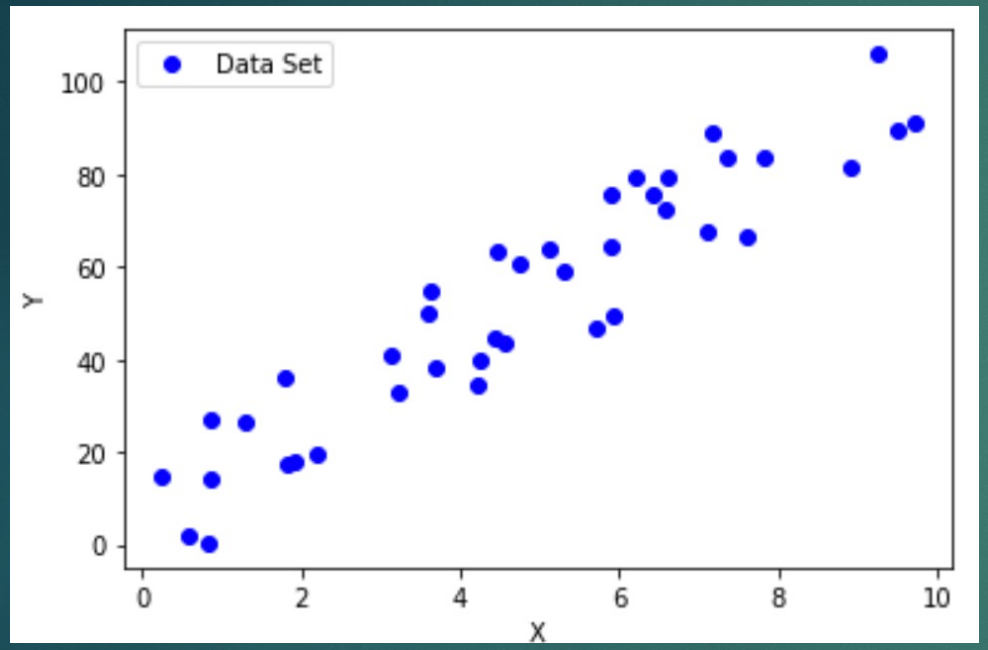
$$y = \theta_0 + \theta_1 * x_1$$

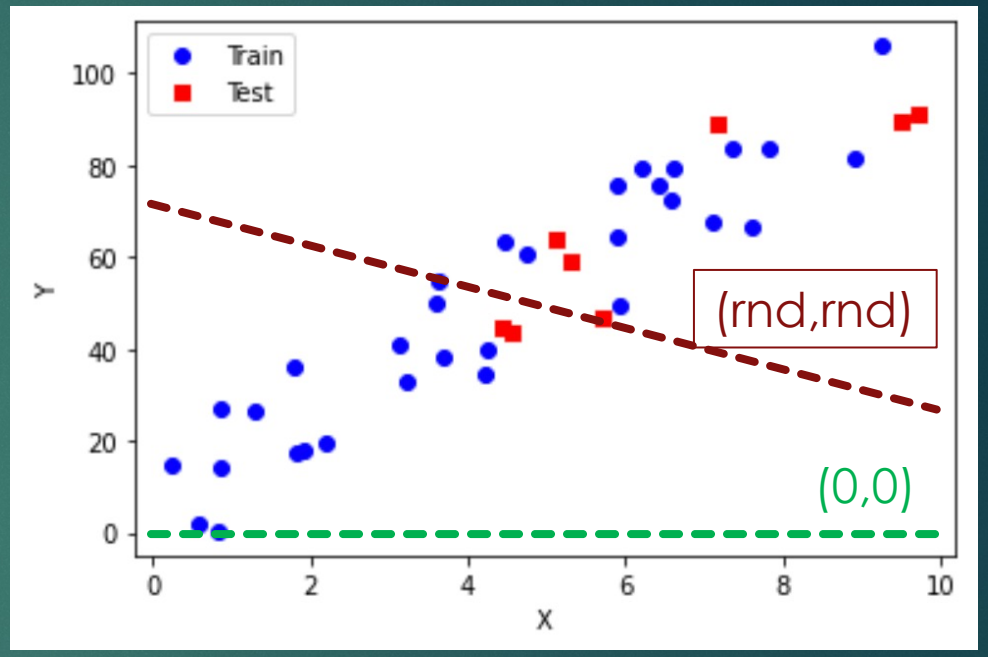
Linear Regression: algorithm

1. Dtrain, Dtest = separate (D, fraction)
2. θ = Initialize ()
3. Repeat many times
    1. take a subset Di from Dtrain
    2. θnew = update_theta (Di, θ)
    3. θ = θnew
4. verify (Dtest, θ)

Linear Regression: algorithm

1. `Dtrain, Dtest = separate (D, 0.8)`





2. θ = `Initialize ()`

Optimization: gradient descent

θnew = update_theta (Di, θ)

- The goal is to learn a model that minimizes the empirical risk $\hat{R}_{f_\theta}(\mathcal{D})$:

$$\hat{f}_\theta = \underset{f_\theta \epsilon \mathcal{F}}{\operatorname{argmin}} \sum_{i=1}^{M} \mathrm{L}\left(f_\theta(x^i), y^i\right)$$

- The loss quantifies the distance between each prediction and the true value. Let's use the square error as the example:

$$\mathrm{L}\left(f_\theta(x^i), y^i\right) = \left(f_\theta(x^i) - y^i\right)^2 = \left(\theta_0 + \theta_1 * x^i - y^i\right)^2$$

- The goal is to find θ that minimizes the objective function:

$$\sum_{i=1}^{M} \left(\theta_0 + \theta_1 * x^i - y^i\right)^2$$

Optimization: gradient descent

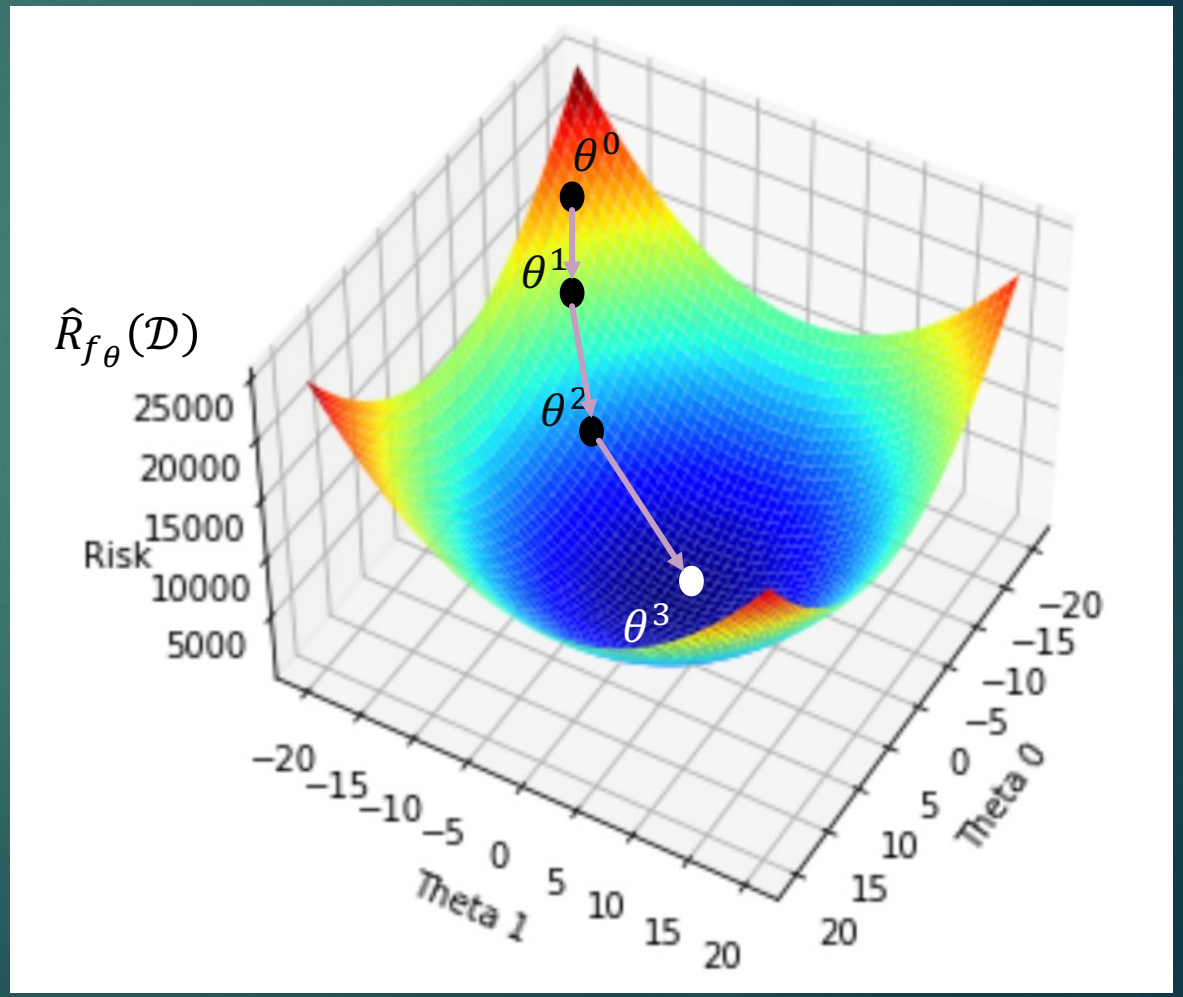
`θnew = update_theta (Di, θ)`

- The goal is to find θ that minimizes the objective function:

$$\hat{R}_{f_\theta}(\mathcal{D}) = \sum_{i=1}^{M} \left(\theta_0 + \theta_1 * x^i - y^i\right)^2$$

- At each iteration change θ slightly along the direction of maximum change

$$\theta^{t+1} = \theta^t - \eta * \nabla \sum_{i=1}^{M} \left(\theta_0^t + \theta_1^t * x^i - y^i\right)^2$$

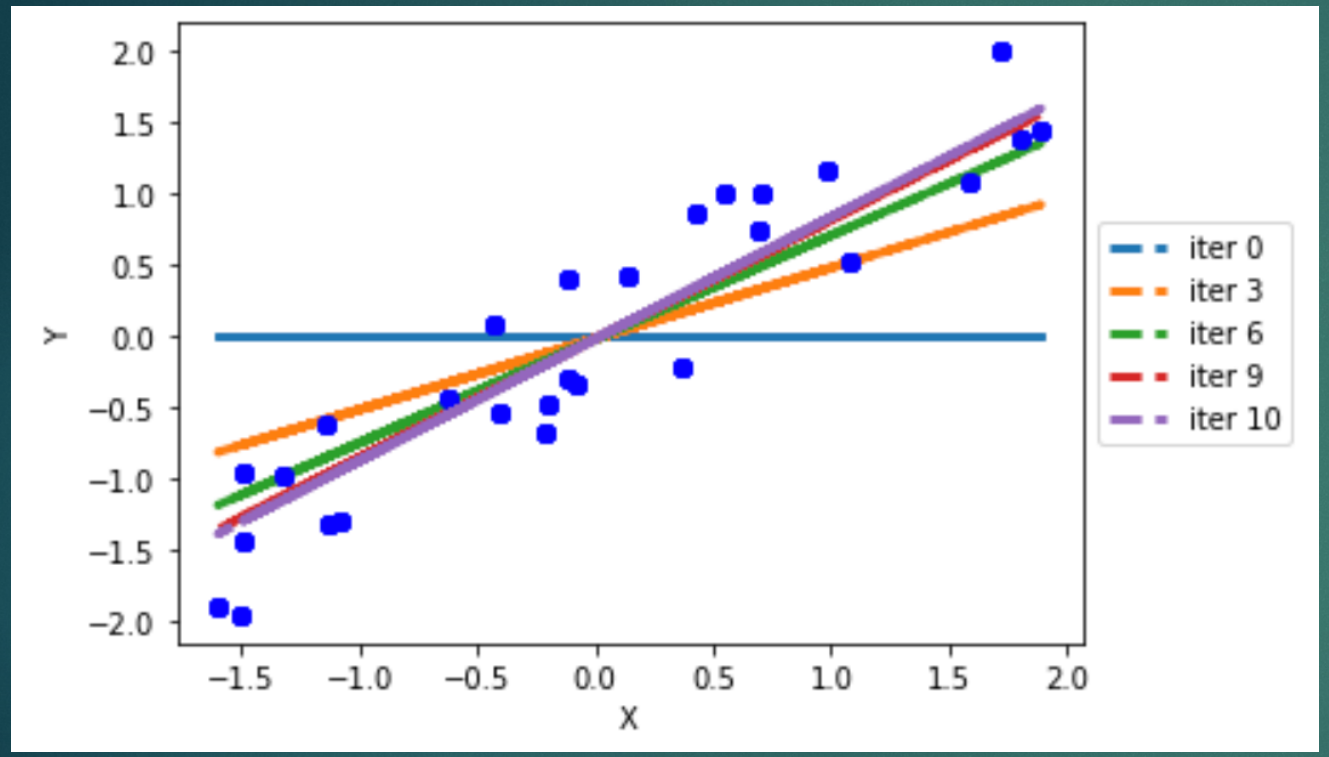


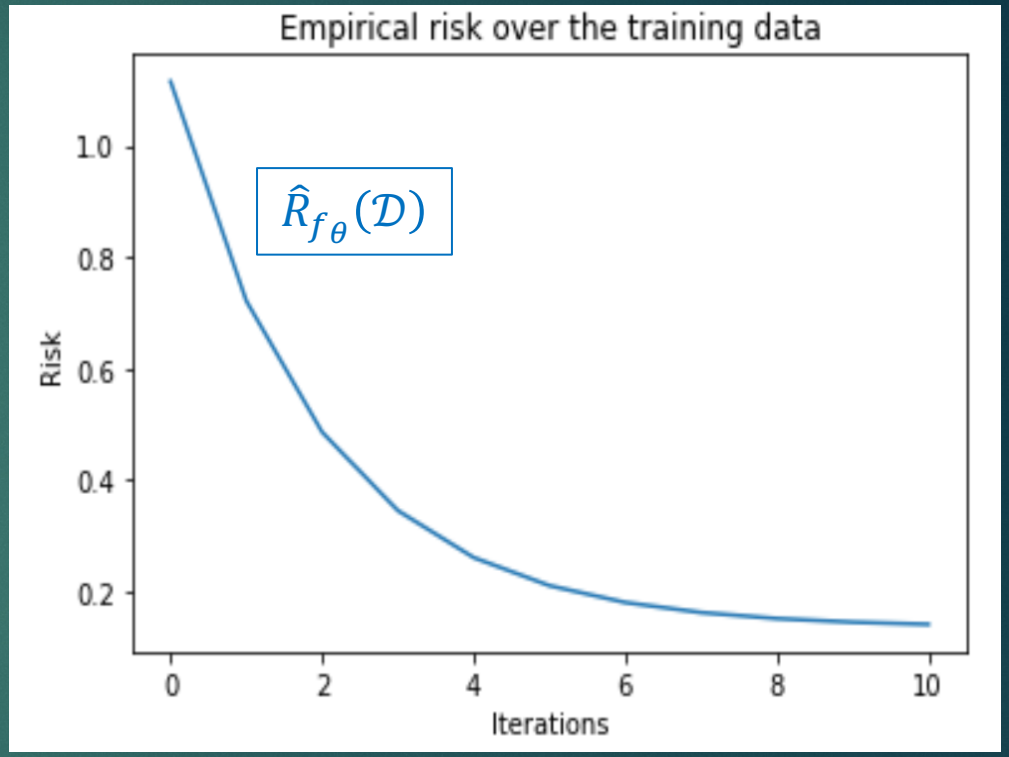
Optimization: gradient descent

$$\nabla \sum_{i=1}^{M} \left(\theta_0^t + \theta_1^t * x^i - y^i\right)^2 = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \\ \frac{\partial}{\partial \theta_1} \end{pmatrix} = \begin{pmatrix} 2\sum_{i=1}^{M} \left(\theta_0^t + \theta_1^t * x^i - y^i\right) \\ 2\sum_{i=1}^{M} \left(\theta_0^t + \theta_1^t * x^i - y^i\right) * x^i \end{pmatrix}$$

$$= \begin{pmatrix} 2\sum_{i=1}^{M} \left(f_{\theta^t}(x^i) - y^i\right) \\ 2\sum_{i=1}^{M} \left(f_{\theta^t}(x^i) - y^i\right) * x^i \end{pmatrix}$$

Linear Regression

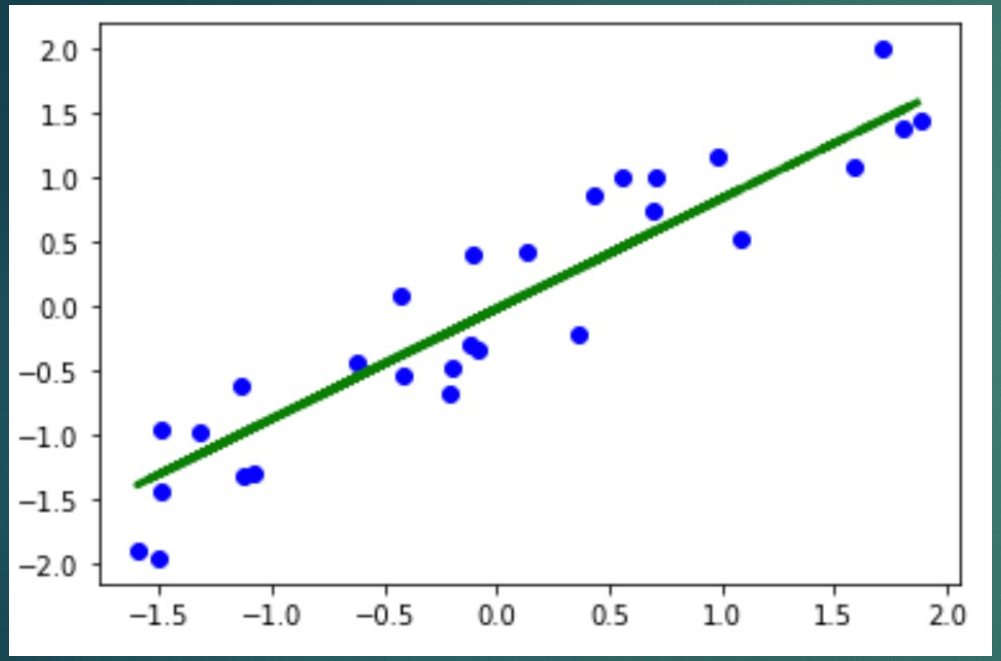




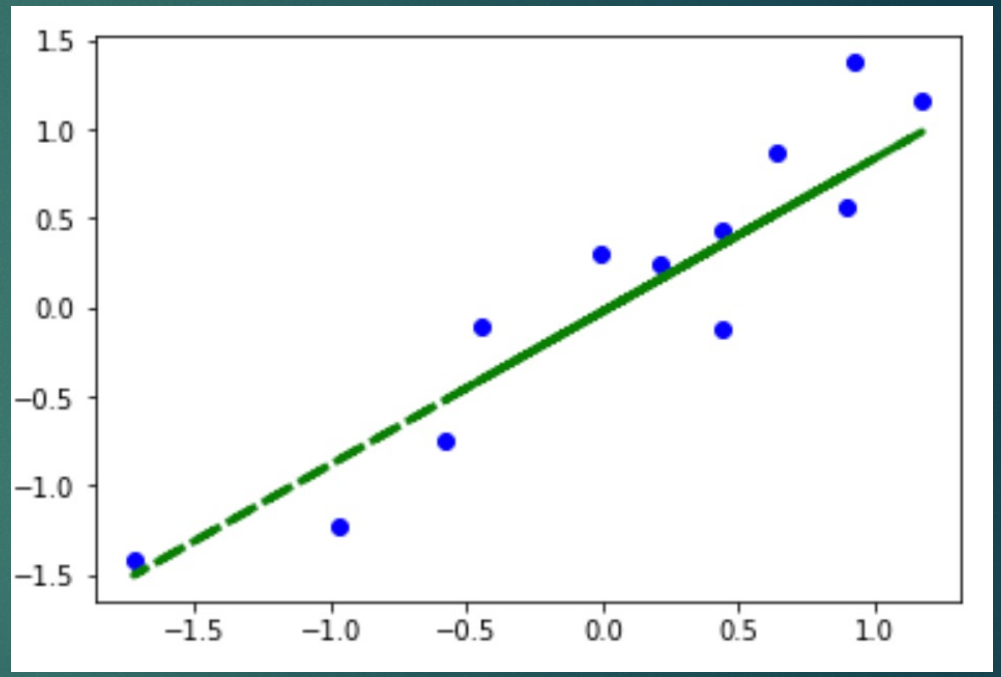
Linear Regression: testing

4. verify (Dtest, θ)

train data set



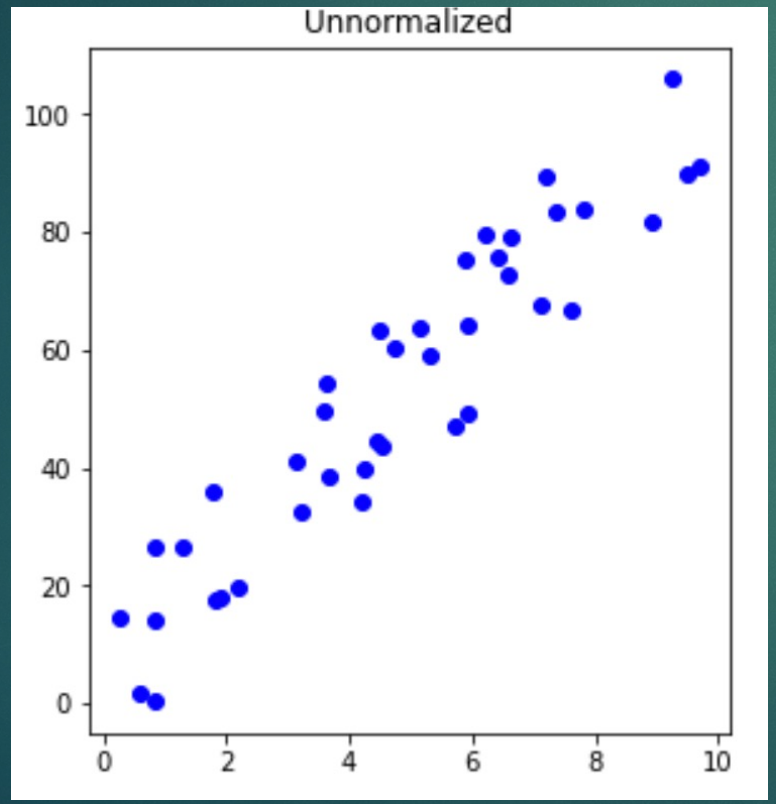
test data set

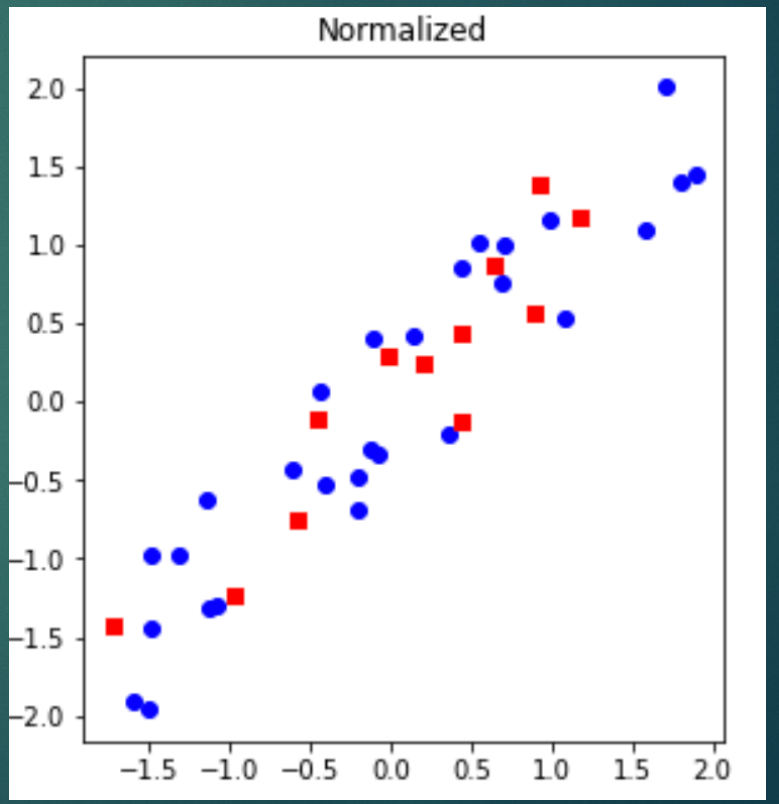


MSE = 0.0989

Linear Regression: data shifting and normalization

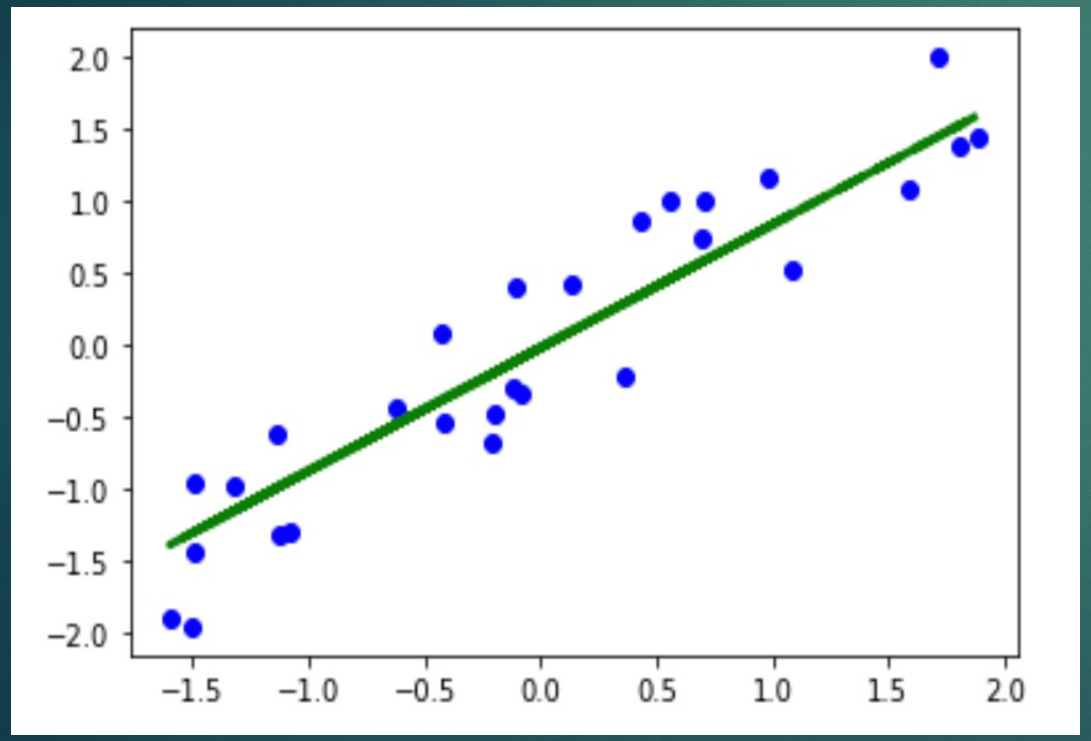
$$x = \frac{(x-\bar{x})}{\sigma_x} \qquad y = \frac{(y-\bar{y})}{\sigma_y} \qquad \forall (x, y) \in \mathcal{D}$$

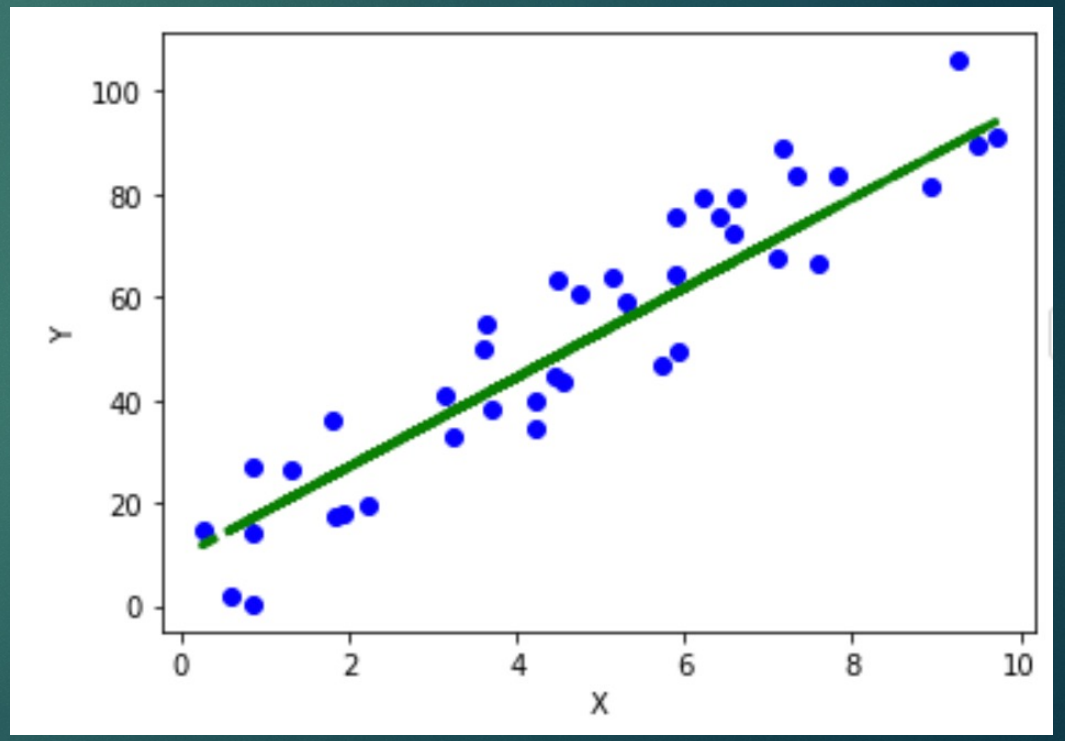




Linear Regression: data shifting and normalization

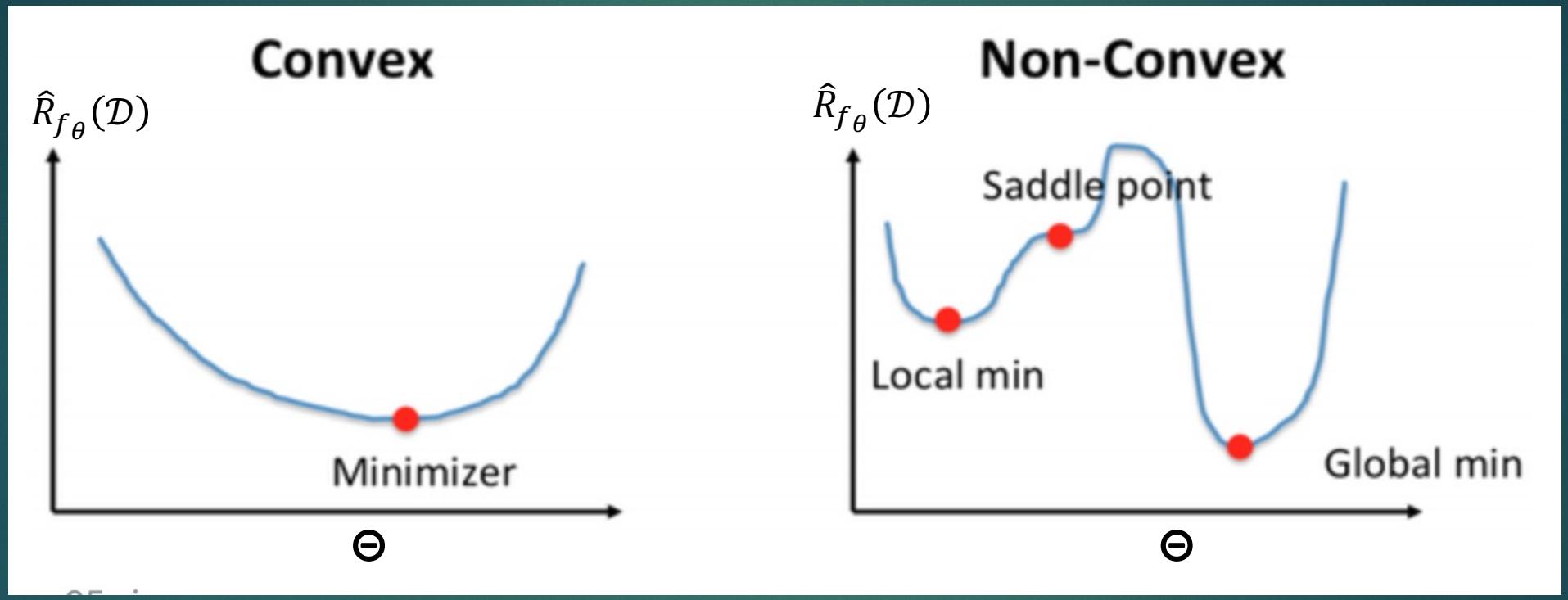
$$\theta_0 = \frac{\sigma_y(\theta_{0,n} - \theta_{1,n} * \bar{x})}{\sigma_x} + \bar{y} \qquad \theta_1 = \frac{\sigma_y * \theta_{1,n}}{\sigma_x}$$





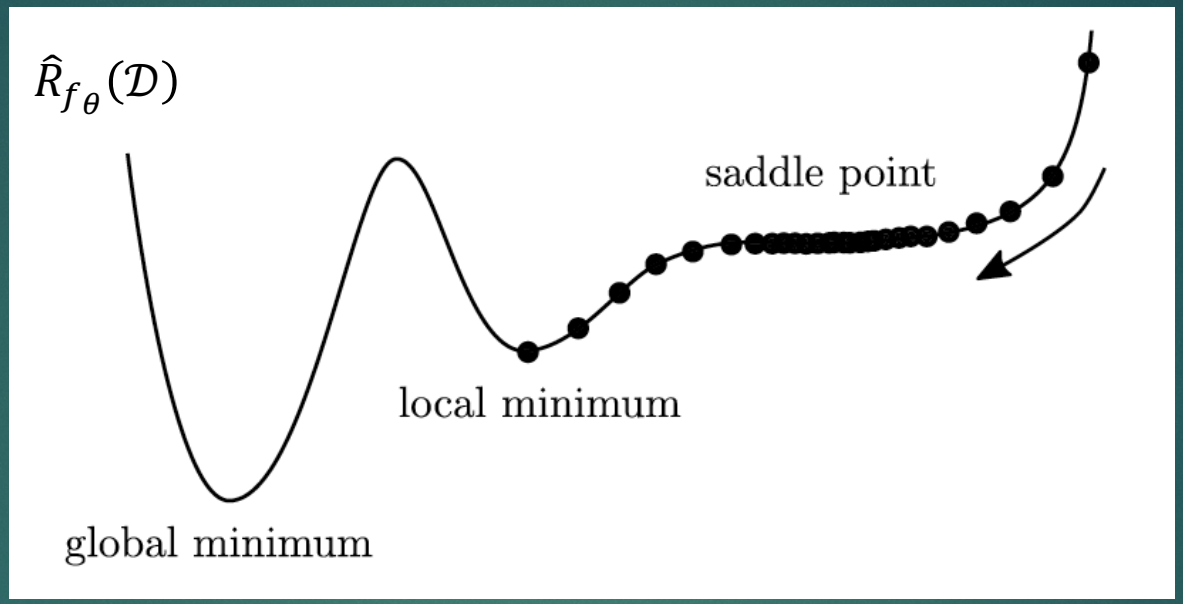
Optimization: convexity and local minima

Convexity: non-convex objective functions are much harder to optimize



Many machine learning problems are high-dimensional (large number of parameters) non convex

Optimization: local minima



- clever initialization of θ
- random restart
- clever choice of the learning rate η
- stochastic gradient descent: use a stochastic subset of the training data at each step
- natural gradient: use geometric information on the curvature of the landscape

Multivariate Linear Regression

- **Loss**

$$\mathrm{L}(f_\theta(x), \mathrm{y}) = (f_\theta(x) - y)^2 = \left(\theta_0 + \sum_{i=1}^{n} (\theta_i * x_i) - y\right)^2$$

- **Empirical Risk**

$$\hat{R}_{f_\theta}(\mathcal{D}) = \frac{1}{M}\sum_{j=1}^{M} \mathrm{L}(f_\theta(x^{(j)}), \mathrm{y}^{(j)}) = \frac{1}{M}\sum_{j=1}^{M} \left(\theta_0 + \sum_{i=1}^{n} \left(\theta_i * x_i^{(j)}\right) - \mathrm{y}^{(j)}\right)^2$$

- **Gradient**

$$\nabla \hat{R}_{f_\theta}(\mathcal{D}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \\ \frac{\partial}{\partial \theta_1} \\ \vdots \\ \frac{\partial}{\partial \theta_{T-1}} \end{pmatrix} = \begin{pmatrix} 2\sum_{j=1}^{M} (f_\theta(x^{(j)}) - y^{(j)}) \\ 2\sum_{j=1}^{M} (f_\theta(x^{(j)}) - y^{(j)}) * x_1^{(j)} \\ \vdots \\ 2\sum_{j=1}^{M} (f_\theta(x^{(j)}) - y^{(j)}) * x_{T-1}^{(j)} \end{pmatrix}$$

Polynomial Regression

- **Loss**

$$\mathrm{L}(f_\theta(x), \mathrm{y}) = (f_\theta(x) - y)^2 = \left(\theta_0 + \sum_{i=1}^{T-1}(\theta_i * x^i) - y\right)^2$$

x to the power of i

- **Empirical Risk**

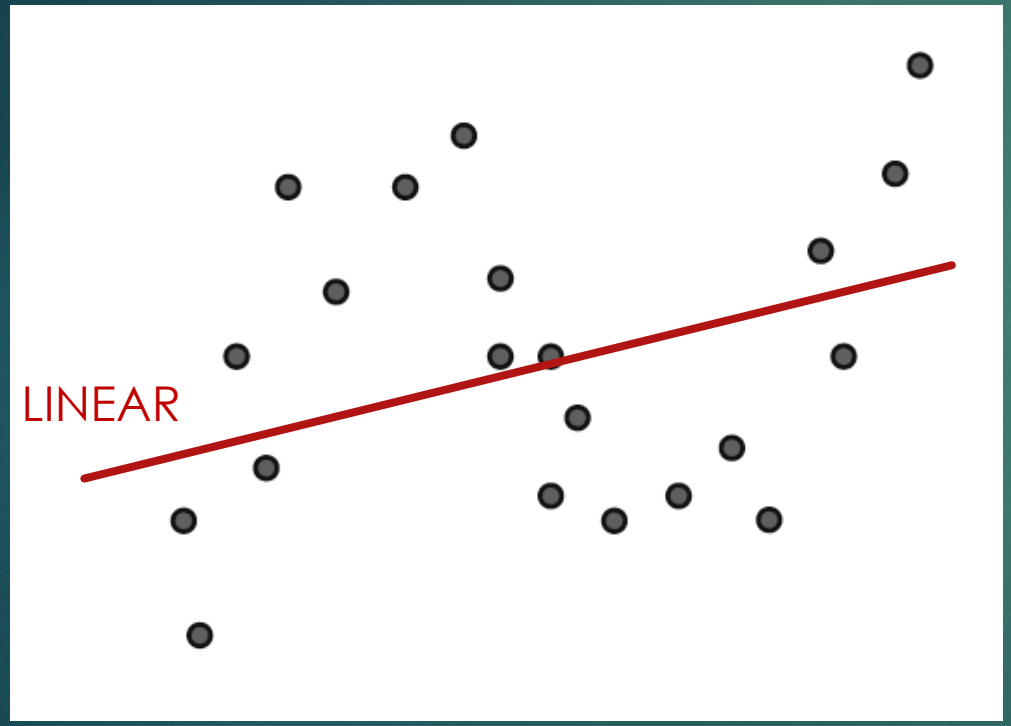
$$\hat{R}_{f_\theta}(\mathcal{D}) = \frac{1}{M}\sum_{j=1}^{M}\mathrm{L}(f_\theta(x^{(j)}), \mathrm{y}^{(j)}) = \frac{1}{M}\sum_{j=1}^{M}\left(\theta_0 + \sum_{i=1}^{T-1}\left(\theta_i * {x^{(j)}}^i\right) - \mathrm{y}^{(j)}\right)^2$$

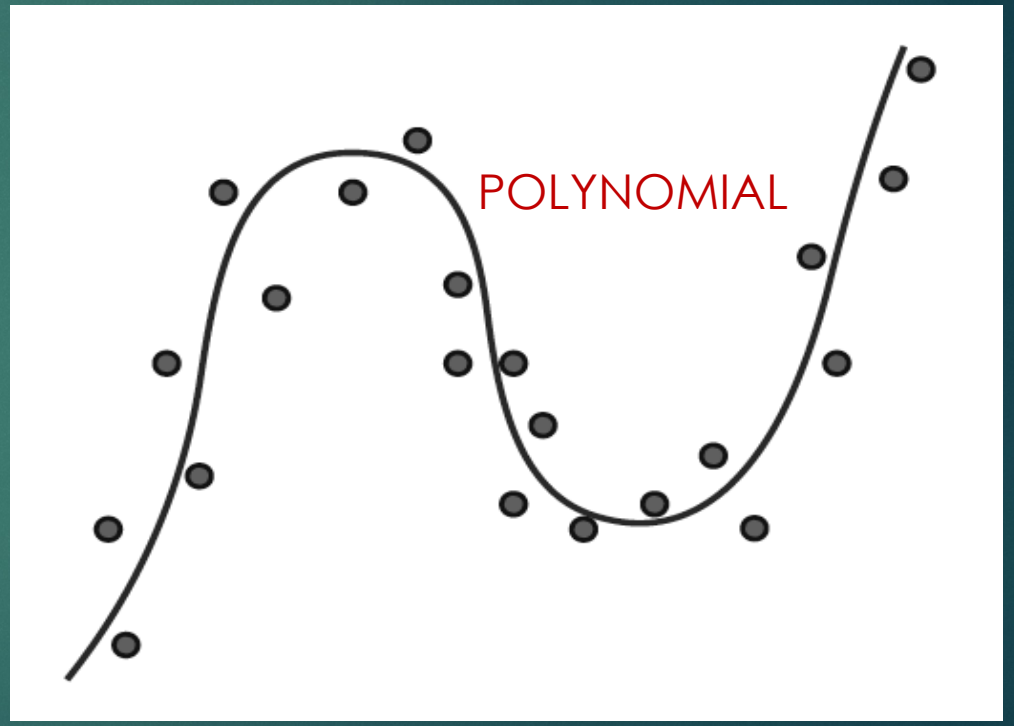
$x^{(j)}$ to the power of i

- **Gradient**

$$\nabla\hat{R}_{f_\theta}(\mathcal{D}) = \begin{pmatrix} \frac{\partial}{\partial\theta_0} \\ \frac{\partial}{\partial\theta_1} \\ \vdots \\ \frac{\partial}{\partial\theta_{T-1}} \end{pmatrix} = \begin{pmatrix} 2\sum_{j=1}^{M}(f_\theta(x^{(j)}) - y^{(j)}) \\ 2\sum_{j=1}^{M}(f_\theta(x^{(j)}) - y^{(j)}) * {x^{(j)}}^1 \\ \vdots \\ 2\sum_{j=1}^{M}(f_\theta(x^{(j)}) - y^{(j)}) * {x^{(j)}}^{T-1} \end{pmatrix}$$

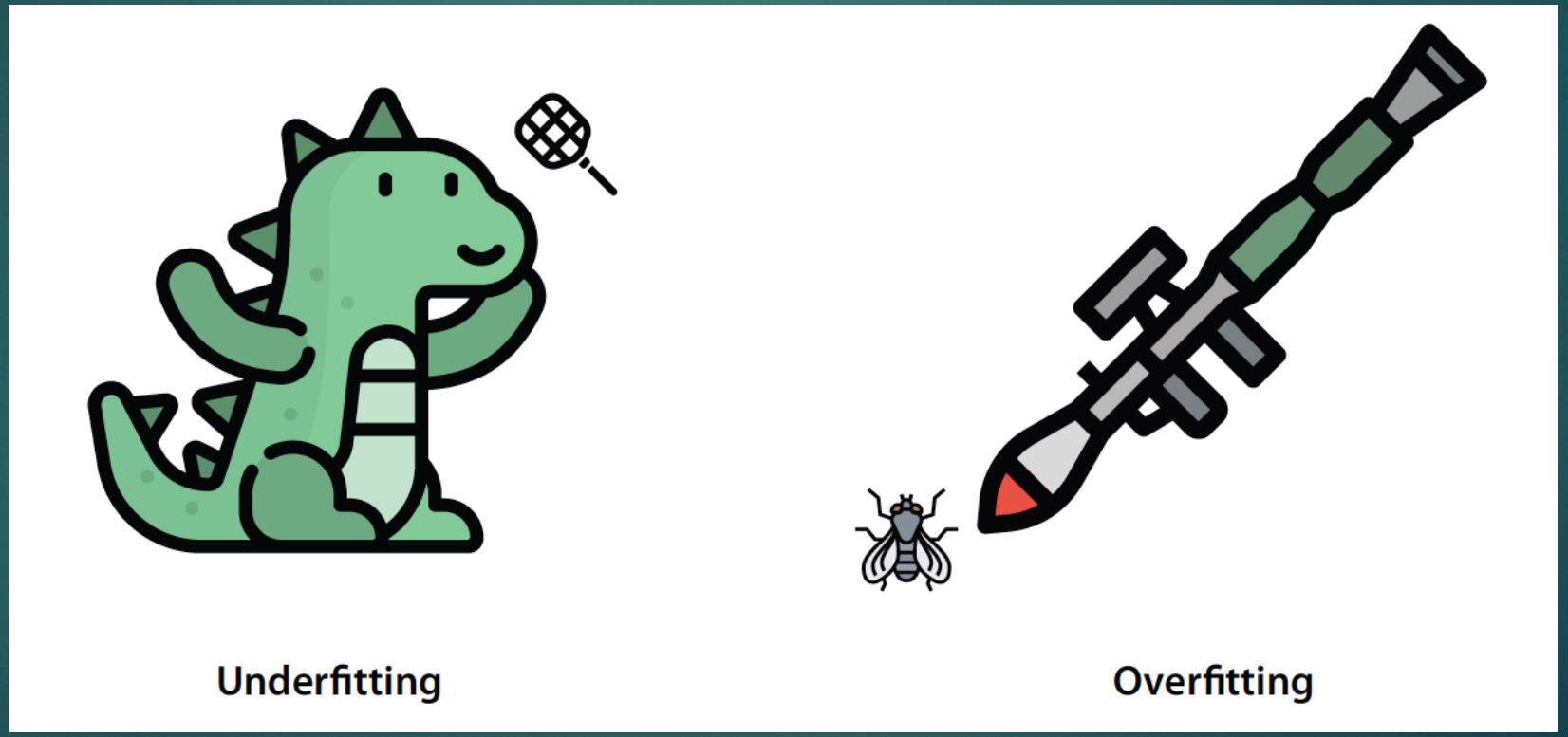
Polynomial Regression





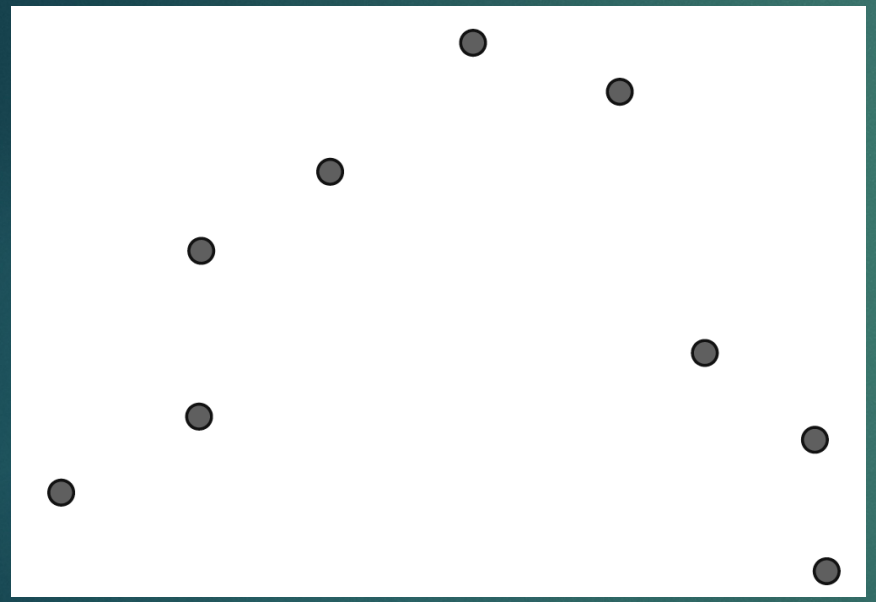
Model Learning: under and over fitting

- **underfitting** occurs when **a too simple** model is used
- **overfitting** occurs when a **too complex** model is used

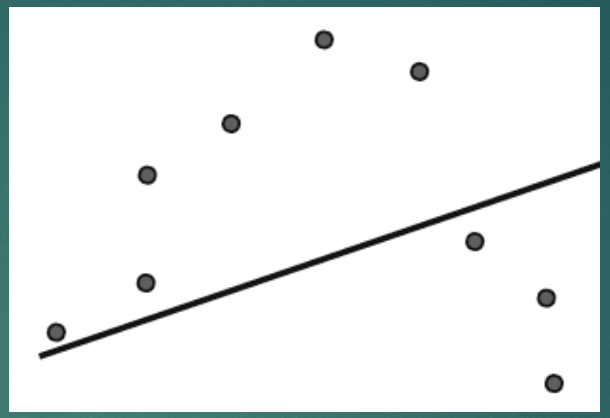


[Serrano2021]

Model Learning: under and over fitting

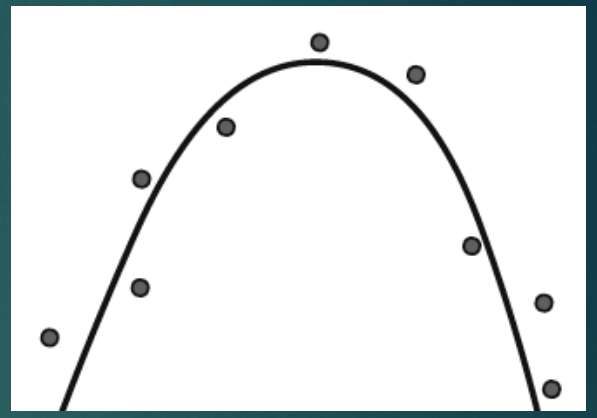


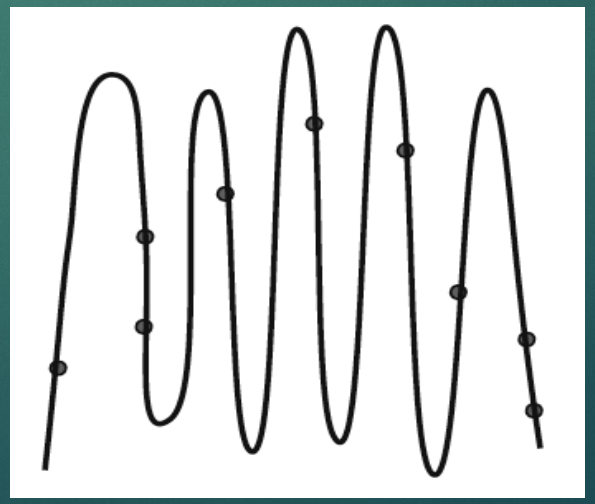
[Serrano2021]



LINEAR

DEGREE 2

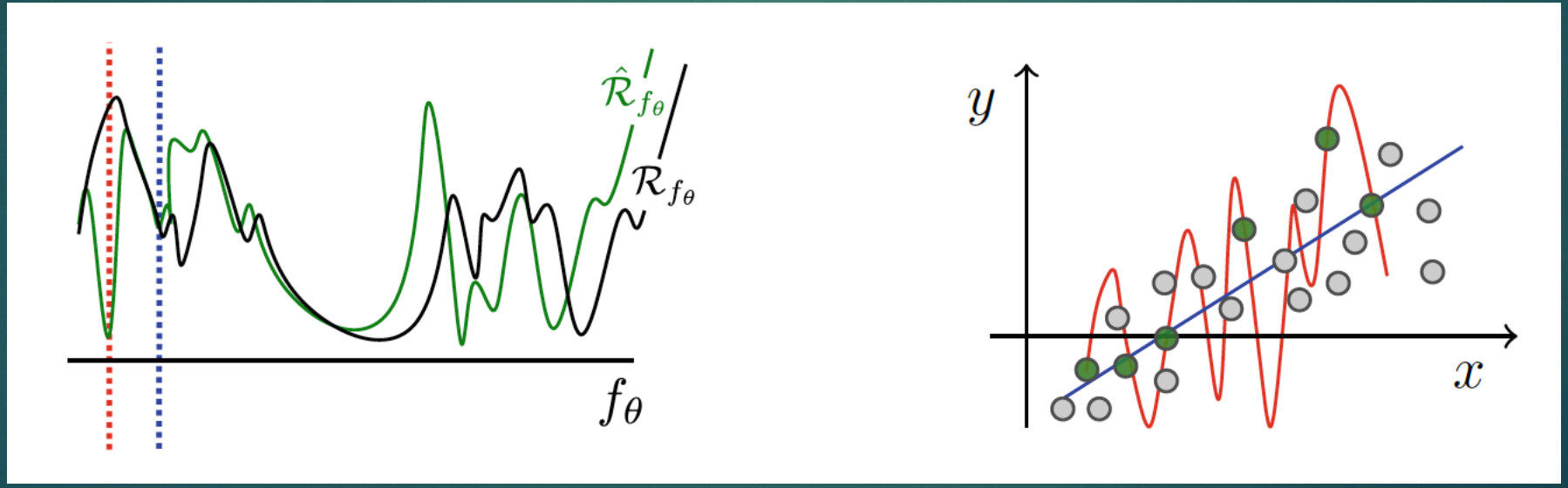




DEGREE 10

Overfitting

- What if empirical risk minimization does not lead to actual risk minimization?
- What if the trained model does really well on the data it was trained with, but performs poorly on new data?



Regularization

- Strategies to avoid overfitting
  - Choose less flexible families of models
  - Stop training prematurely
  - Add a regularization term: $\mathrm{C}(\theta) = \hat{R}_{f_\theta}(\mathcal{D}) + g(\theta)$ and minimize $\mathrm{C}(\theta)$

Regularizers impose constraints on the model:

- $g_{\ell 1}(\theta) = \sum_{i=1}^{T} |\theta_i|$ - the $\ell 1$ regularizer leads to sparser parameters' vectors
- $g_{\ell 2}(\theta) = \sum_{i=1}^{T} {\theta_i}^2$ - the $\ell 2$ regularizer leads to shorter parameters' vectors, i.e. parameters with smaller absolute values

ℓ_2 Regularized Multivariate Linear Regression

- **Cost Function**

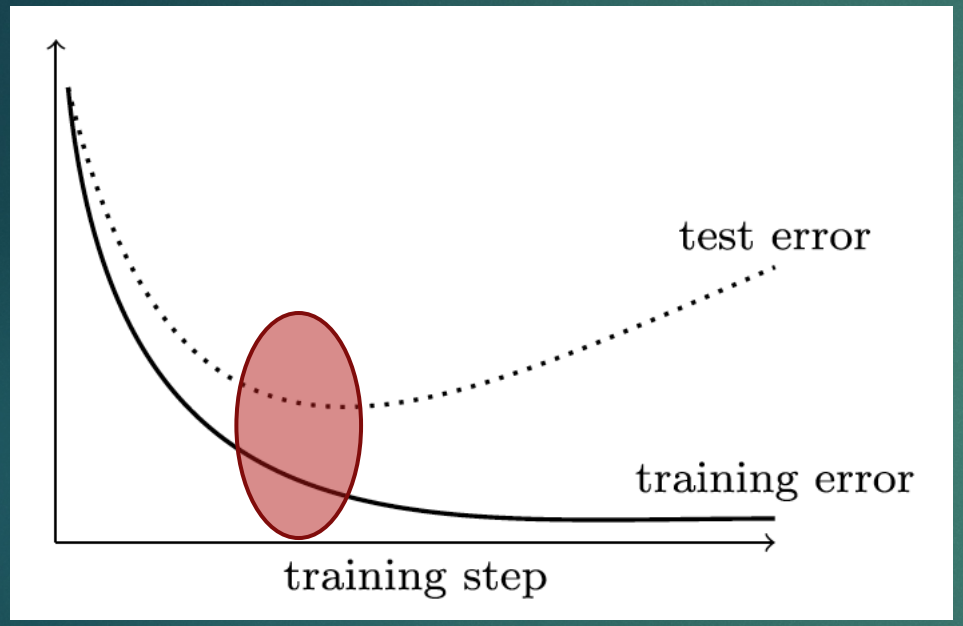
$$\mathrm{C}(\theta) = \hat{R}_{f_\theta}(\mathcal{D}) + g(\theta) = \frac{1}{M}\sum_{j=1}^{M}\left(\theta_0 + \sum_{i=1}^{n}\left(\theta_i * x_i^{(j)}\right) - \mathrm{y}^{(j)}\right)^2 + \sum_{i=0}^{T-1}{\theta_i}^2$$

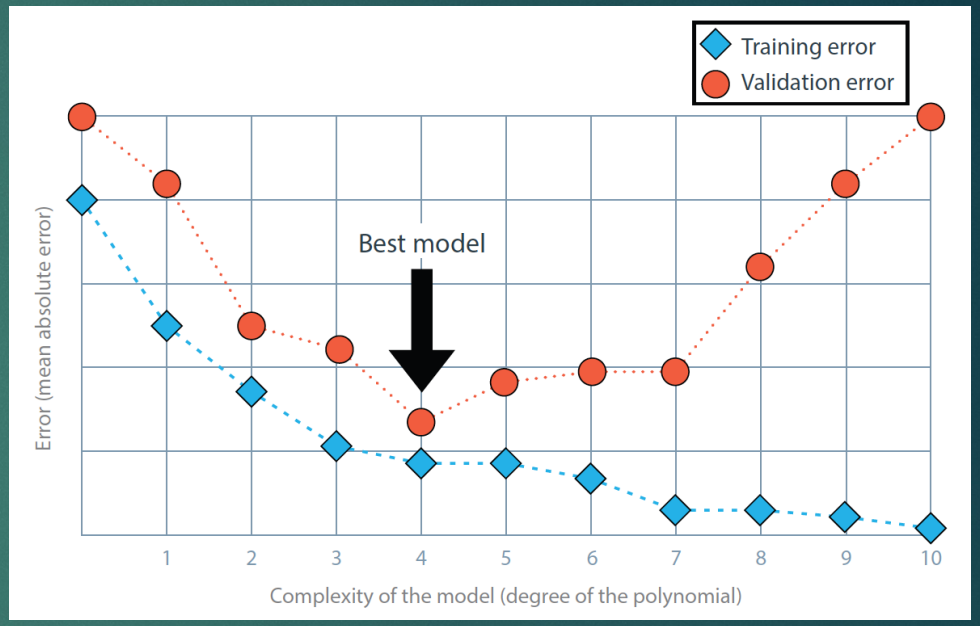
- **Gradient**

$$\nabla\hat{R}_{f_\theta}(\mathcal{D}) = \begin{pmatrix} \frac{\partial}{\partial\theta_0} \\ \frac{\partial}{\partial\theta_1} \\ \vdots \\ \frac{\partial}{\partial\theta_{T-1}} \end{pmatrix} = \begin{pmatrix} 2\sum_{j=1}^{M}\left(f_\theta\left(x^{(j)}\right) - y^{(j)}\right) + 2\theta_0 \\ 2\sum_{j=1}^{M}\left(f_\theta\left(x^{(j)}\right) - y^{(j)}\right) * x_1^{(j)} + 2\theta_1 \\ \vdots \\ 2\sum_{j=1}^{M}\left(f_\theta\left(x^{(j)}\right) - y^{(j)}\right) * x_{T-1}^{(j)} + 2\theta_{T-1} \end{pmatrix}$$

Generalization: training vs. test/validation errors

- the input data $\mathcal{D}$ is divided into a training set and a test set





- the model is overfitting the training data when the training error keeps decreasing, while the test error starts increasing.
- The optimal point to stop training is near the test error inflection point OR choose the model to use based on the model complexity graph

Classification

Supervised Learning: Classification

- Problem:
  given a labelled dataset $\mathcal{D} \in \mathcal{X} \otimes \mathcal{Y}$, taken from an unknown function $f^*: \mathcal{X} \to \mathcal{Y}$, **learn** a function $\hat{f}: \mathcal{X} \to \mathcal{Y}$ which approximates f^* and allows predicting $\hat{y} = \hat{f}(x)$ for previously unseen (x, y), i.e. $(x, y) \notin \mathcal{D}$

- **Classification** entails that $\mathcal{Y}$ is a discrete set of **labels**

- **Binary Classification** entails that $\mathcal{Y}$ is a discrete set of **two labels**
  Without loss of generality $\mathcal{Y} = \{0,1\}$

Perceptron Classifier

- The perceptron classification *line* is expressed as an identity to 0 (small difference compared to the regression in the previous section)

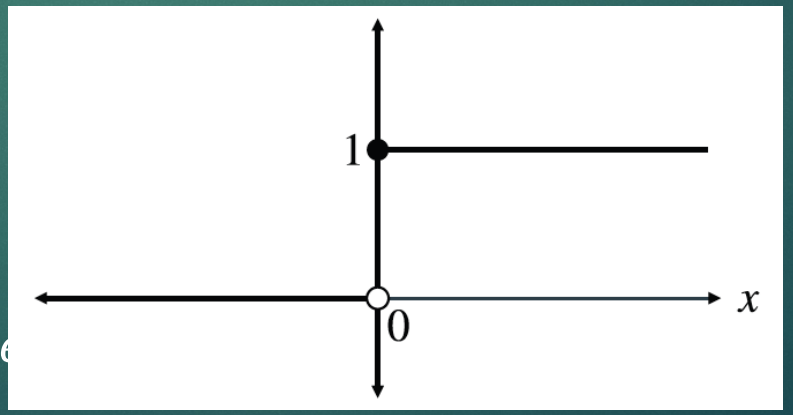
- If there are n features x_i per data point x and $T = n + 1$ parameters θ_t then the classifier produces a score:

$$score(x) = \theta_0 + \sum_{i=1}^{n} \theta_i x_i$$

- $\hat{y}(x) = \begin{cases} 0 & \leftarrow score(x) < 0 \\ 1 & \leftarrow score(x) \geq 0 \end{cases}$

- $\hat{y}(x) = step\big(score(x)\big)$

$= ste$



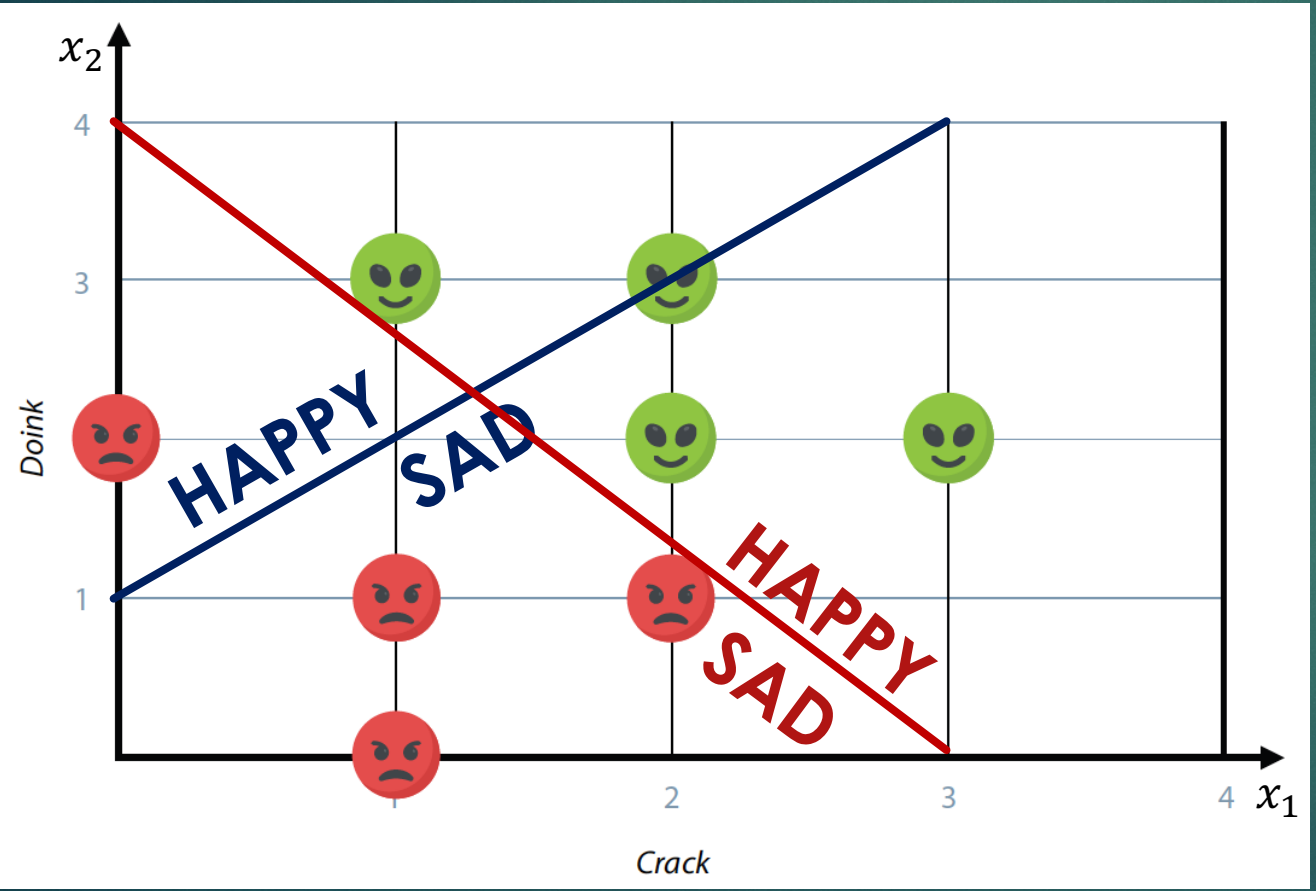
Activation function:

$$step(x) = \begin{cases} 0 & \leftarrow x < 0 \\ 1 & \leftarrow x \geq 0 \end{cases}$$

The "alien" dataset
[Serrano'2021-cap 5]

Sentence	*Crack*	*Doink*	Mood
Crack!	1	0	Sad
Doink doink!	0	2	Sad
Crack doink!	1	1	Sad
Crack doink crack!	2	1	Sad
Doink crack doink doink!	1	3	Happy
Crack doink doink crack!	2	2	Happy
Doink doink crack crack crack!	3	2	Happy
Crack doink doink crack doink!	2	3	Happy

The "alien" dataset
[Serrano'2021-cap 5]



$$\theta^{(0)} = [1., 0.5, 0.5]$$

$$score_{\theta^{(0)}}(x) = 1. + \frac{x_1}{2} + \frac{x_2}{2}$$

$$\hat{y}_{\theta^{(0)}}(x) = step\left(1. + \frac{x_1}{2} + \frac{x_2}{2}\right)$$

$$\theta^{(t)} = [-4., {}^{4}/_{3}, 1.]$$

$$score_{\theta^{(t)}}(x) = -4. + \frac{4x_1}{3} + x_2$$

$$\hat{y}_{\theta^{(t)}}(x) = step\left(-4. + \frac{4x_1}{3} + x_2\right)$$

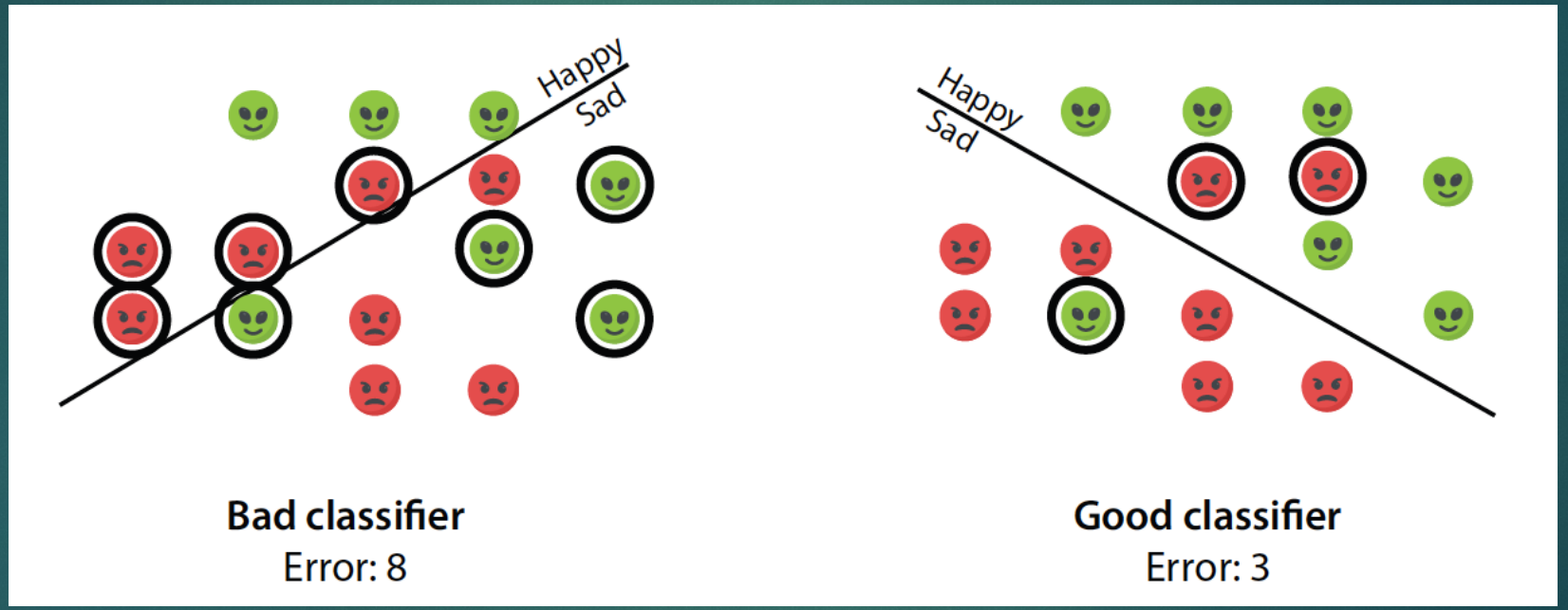
Hyper-Planes

$$score(x) = \theta_0 + \sum_{i=1}^{n} \theta_i x_i$$

- The classifier is represented by an hyperplane with $(n - 1)$-dimensionality

Perceptron Error Function

- Option 1: count the number of errors

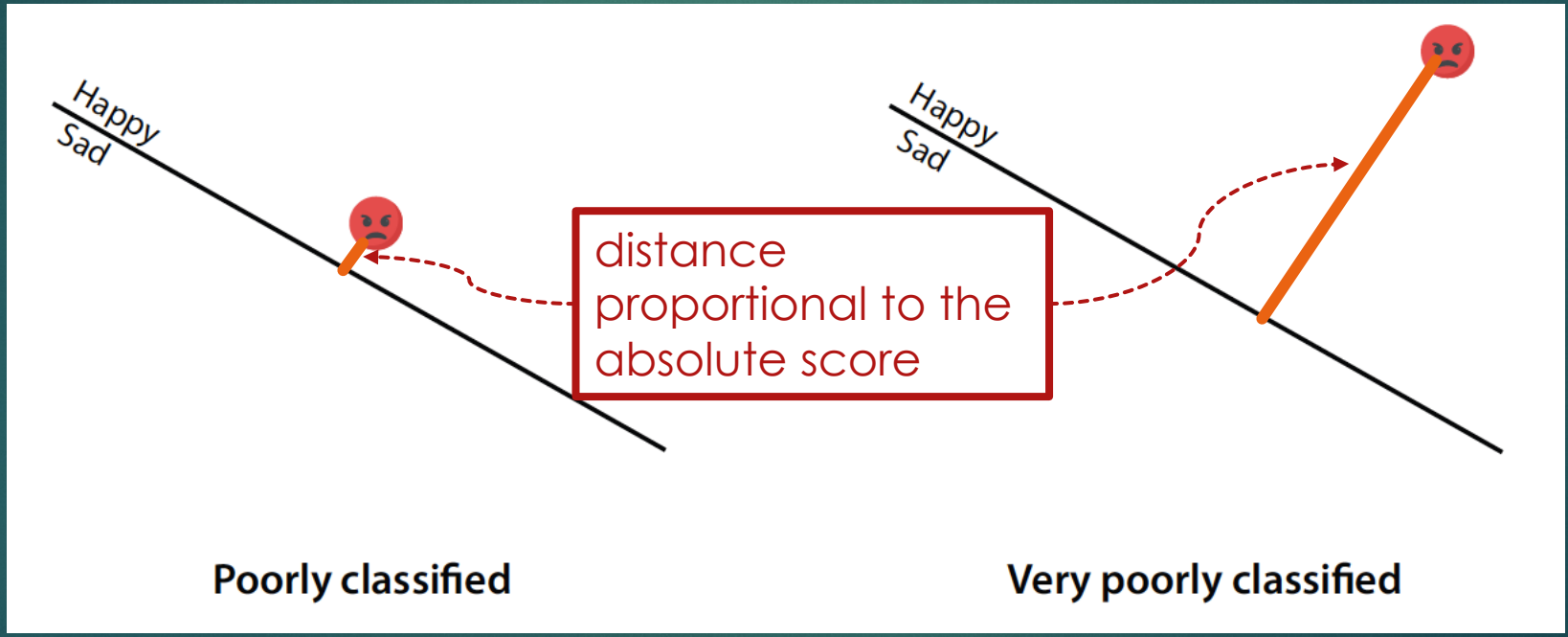


Does not capture **how misclassified** each point was!

Perceptron Error Function

- Option 1: count the number of errors

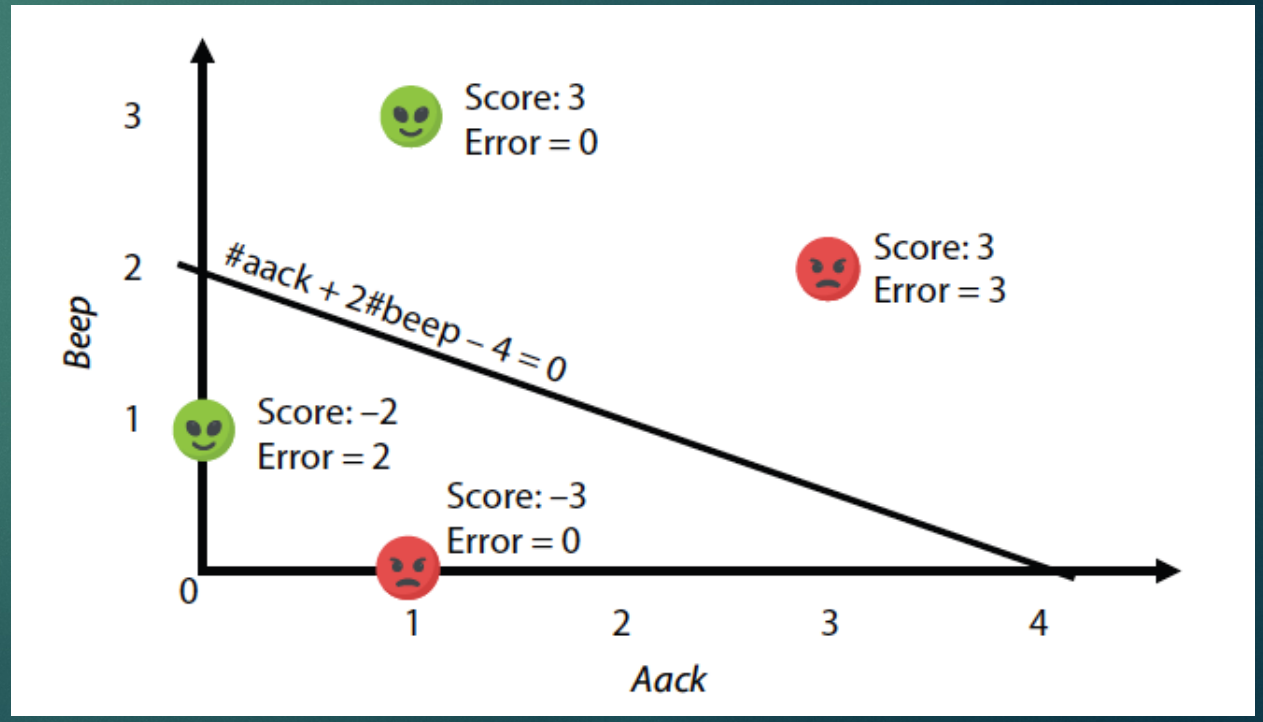
  Does not capture **how misclassified** each point was!



Perceptron Error Function

- Option 2: absolute score of misclassified points

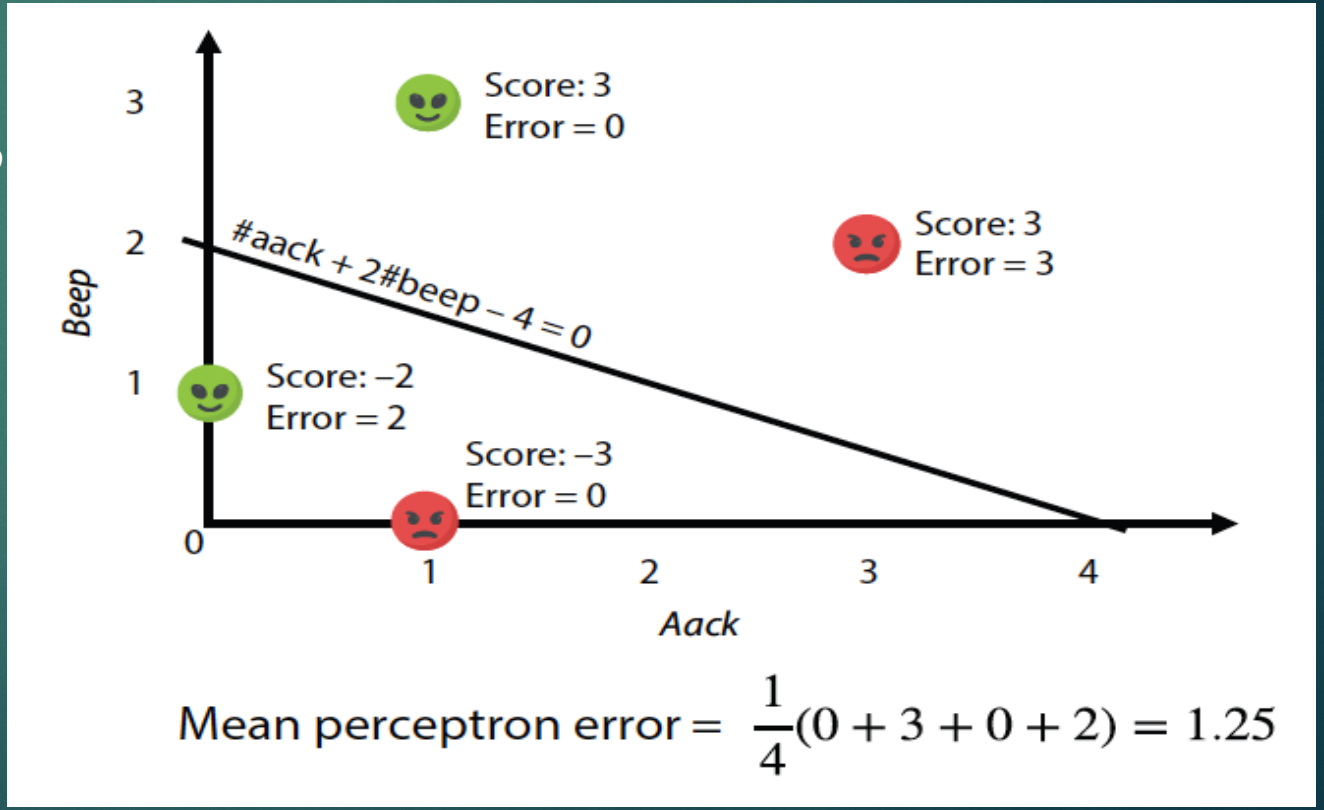
$$L(\hat{y}(x), y) = \begin{cases} 0 & \leftarrow \hat{y}(x) = y \\ \left|\theta_0 + \sum_{i=1}^{n} \theta_i x_i\right| & \leftarrow \hat{y}(x) \neq y \end{cases}$$



Mean Perceptron Error Function (Risk)

- average over the dataset of the absolute scores of misclassified points

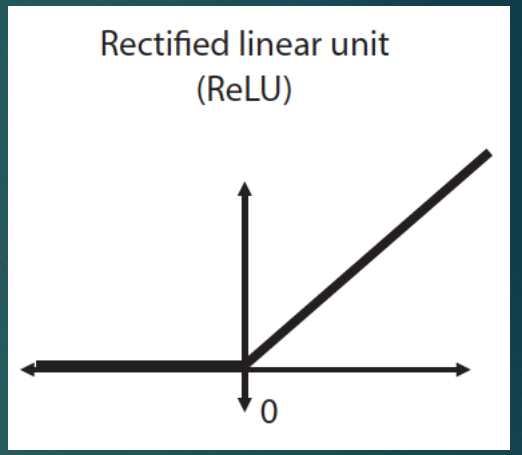
$$\hat{\mathcal{R}}_\theta(\mathcal{D}) = \frac{1}{M} \sum_{x^{(j)} \in \mathcal{D}}$$



Gradient Descent

$$\hat{\mathcal{R}}_\theta(\mathcal{D}) = \frac{1}{M}\sum_{j=1}^{M} L\left(\hat{y}\left(x^{(j)}\right), y^{(j)}\right) \qquad L(\hat{y}(x), y) = \begin{cases} 0 & \leftarrow \hat{y}(x) = y \\ |\theta_0 + \sum_{i=1}^{n} \theta_i x_i| & \leftarrow \hat{y}(x) \neq y \end{cases}$$

$$\frac{\partial ReLU(f(x))}{\partial x} = step(f(x))\frac{\partial f(x)}{\partial x} \qquad ReLU(x) = \begin{cases} 0 & \leftarrow x < 0 \\ x & \leftarrow x \geq 0 \end{cases}$$



$$L(\hat{y}(x), y) = y * ReLU(-\theta_0 - \sum_{i=1}^{n} \theta_i x_i) + (1 - y) * ReLU(\theta_0 + \sum_{i=1}^{n} \theta_i x_i)$$

$$\frac{\partial L(\hat{y}(x), y)}{\partial \theta_{>0}} = y * step\big(-score(x)\big) * (-x_i) + (1 - y) * step\big(score(x)\big) * x_i$$

Gradient Descent

$$L(\hat{y}(x), y) = y * ReLU(-\theta_0 - \sum_{i=1}^{n} \theta_i x_i) + (1 - y) * ReLU(\theta_0 + \sum_{i=1}^{n} \theta_i x_i)$$

$$\frac{\partial L(\hat{y}(x), y)}{\partial \theta_{>0}} = y * step\big(-score(x)\big) * (-x_i) + (1 - y) * step\big(score(x)\big) * x_i =$$

$$= \Big(-y * step\big(-score(x)\big) + (1 - y) * step\big(score(x)\big)\Big) x_i$$

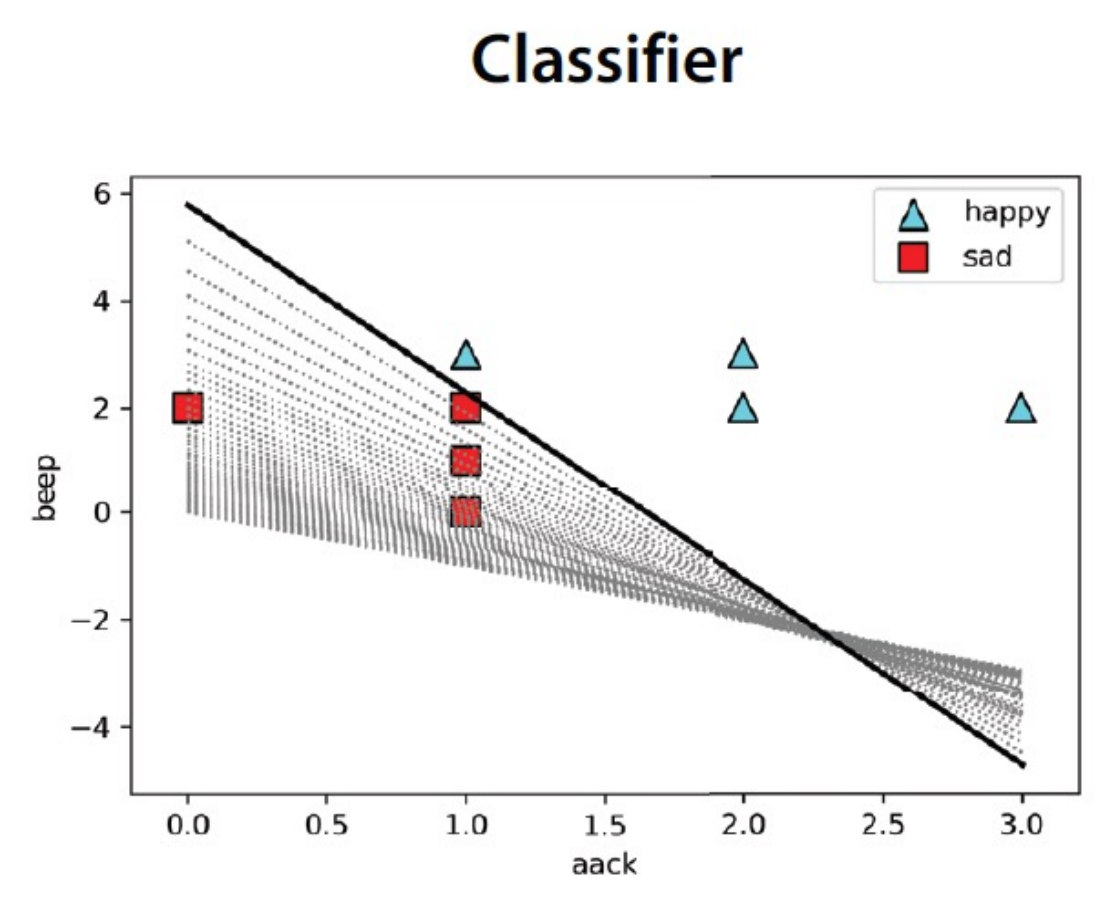
$$\frac{\partial L(\hat{y}(x), y)}{\partial \theta_0} = -y * step\big(-score(x)\big) + (1 - y) * step\big(score(x)\big)$$

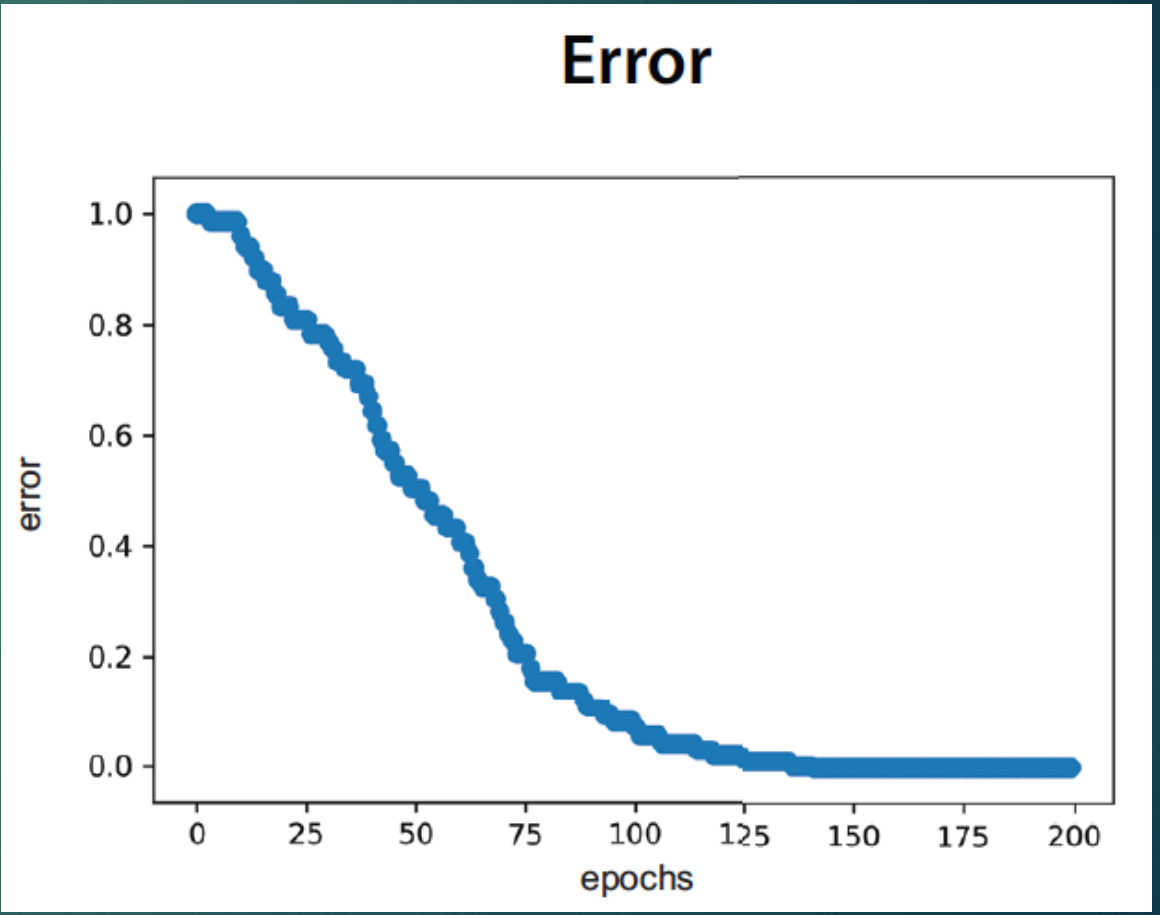
$\hat{y}(x)$	y	$score(x)$	$\frac{\partial L(\hat{y}(x), y)}{\partial \theta_{>0}}$
0	0	< 0	0
0	1	< 0	$-x_i$
1	0	> 0	x_i
1	1	> 0	0

$$\hat{\mathcal{R}}_\theta(\mathcal{D}) = \frac{1}{M} \sum_{j=1}^{M} L\big(\hat{y}\big(x^{(j)}\big), y^{(j)}\big)$$

$$\nabla \hat{R}_{f_\theta}(\mathcal{D}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \\ \frac{\partial}{\partial \theta_1} \\ \vdots \\ \frac{\partial}{\partial \theta_{T-1}} \end{pmatrix} = \begin{pmatrix} \frac{1}{M} \sum_{j=1}^{M} \Big[-y^{(j)} * step\Big(-score\big(x^{(j)}\big)\Big) + \big(1 - y^{(j)}\big) * step\Big(score\big(x^{(j)}\big)\Big)\Big] \\ \frac{1}{M} \sum_{j=1}^{M} \Big[-y^{(j)} * step\Big(-score\big(x^{(j)}\big)\Big) + \big(1 - y^{(j)}\big) * step\Big(score\big(x^{(j)}\big)\Big)\Big] * x_1^{(j)} \\ \vdots \\ \frac{1}{M} \sum_{j=1}^{M} \Big[-y^{(j)} * step\Big(-score\big(x^{(j)}\big)\Big) + \big(1 - y^{(j)}\big) * step\Big(score\big(x^{(j)}\big)\Big)\Big] * x_{T-1}^{(j)} \end{pmatrix}$$

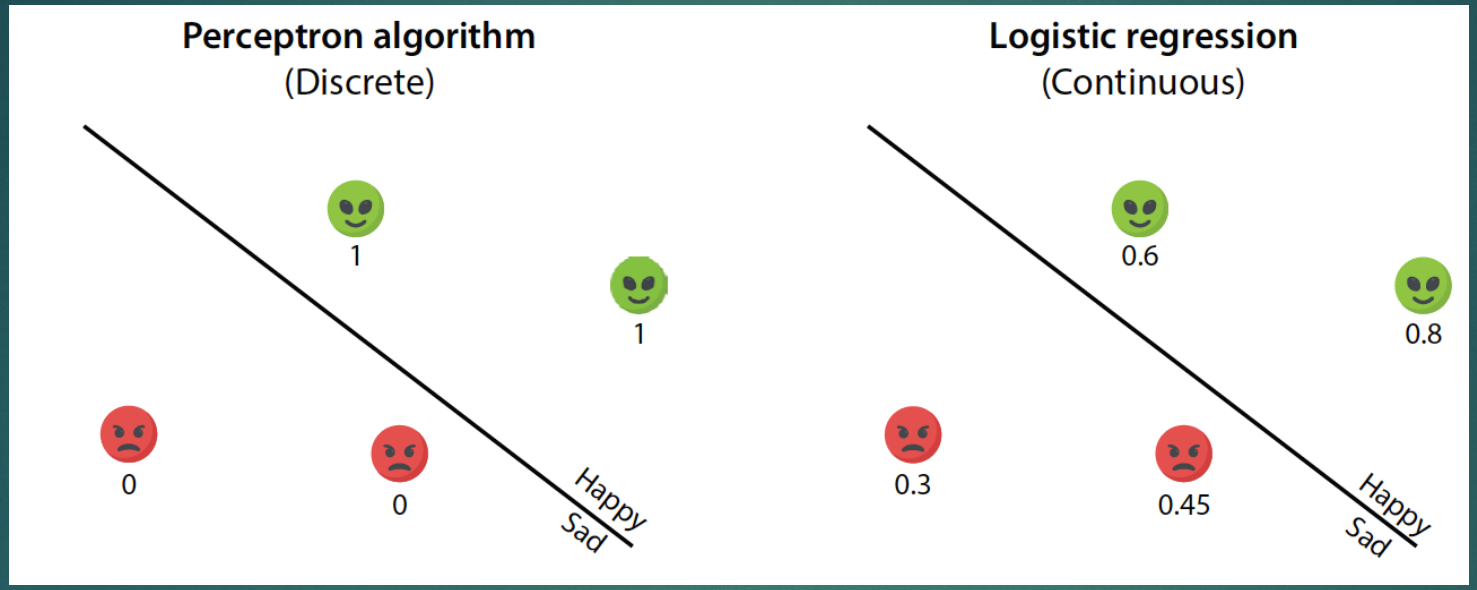
Perceptron Classifier





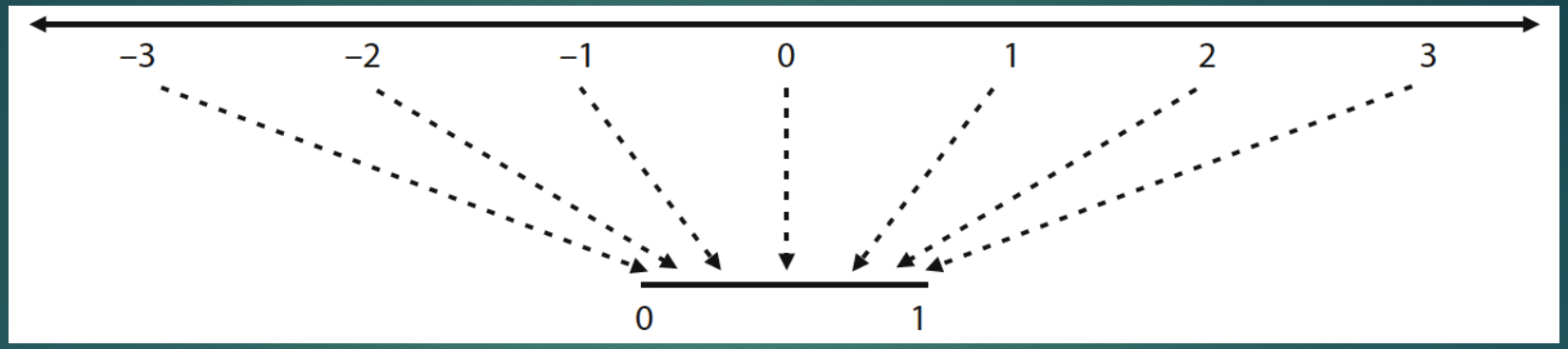
Logistic Classifiers

- Binary classifiers which return a continuous value in [0..1]



- The number assigned to each data point is the probability that its label is y=1: $p_1(x)$
- The probability of being y=0 is $p_0(x)$=1- $p_1(x)$

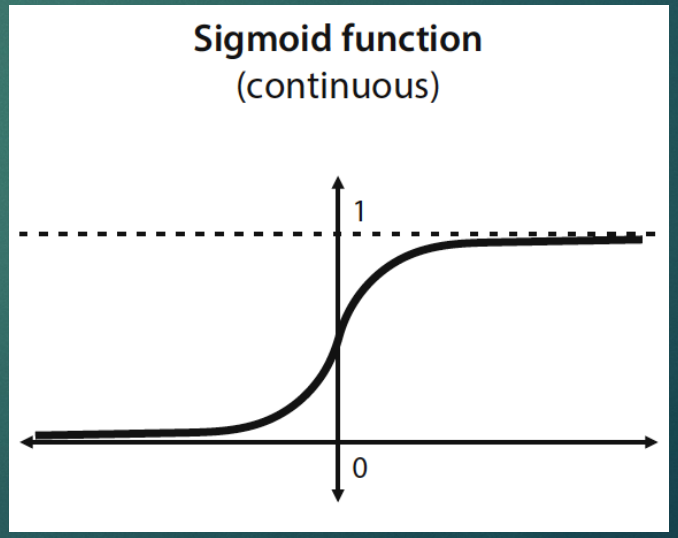
Logistic Classifiers



Sigmoid function: the score ∈ ℝ is compressed into [0..1]

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

$$\hat{y}(x) = \sigma\big(score(x)\big)$$
$$= \sigma\left(\theta_0 + \sum_{i=1}^{n} \theta_i x_i\right)$$



Logistic Classifier Error: log loss

- If $y = 1$ then $L(\hat{y}(x), y) = -\ln(\hat{y}(x))$
- If $y = 0$ then $L(\hat{y}(x), y) = -\ln(1 - \hat{y}(x))$

$$L(\hat{y}(x), y) = -\mathrm{y} * \ln(\hat{y}(x)) - (1 - y) * \ln(1 - \hat{y}(x))$$

$$\hat{\mathcal{R}}_\theta(\mathcal{D}) = \sum_{j=1}^{M} L(\hat{y}(x^{(j)}), y^{(j)})$$

Point	Label	Predicted label	Probability of being its label	Log loss
1	0 (Sad)	0.953	0.047	–ln(0.047) = 3.049
2	1 (Happy)	0.731	0.731	–ln(0.731) = 0.313
3	1 (Happy)	0.119	0.119	–ln(0.119) = 2.127
4	0 (Sad)	0.119	0.881	–ln(0.881) = 0.127

Gradient Descent

$$\hat{y}(x) = \sigma\left(\theta_0 + \sum_{i=1}^{n} \theta_i x_i\right)$$

$$L(\hat{y}(x), y) = -\mathrm{y} * \ln\left(\hat{y}(x)\right) - (1-y) * \ln\left(1-\hat{y}(x)\right)$$

$$\frac{\partial \sigma\left(f(\theta)\right)}{\partial \theta} = \sigma\left(f(\theta)\right)\left(1-\sigma\left(f(\theta)\right)\right)\frac{\partial f(\theta)}{\partial \theta}$$

$$\frac{\partial ln\left(f(z)\right)}{\partial z} = \frac{1}{f(z)}\frac{\partial f(z)}{\partial z}$$

$$\frac{\partial \hat{y}(x)}{\partial \theta_{>0}} = \sigma\left(\theta_0 + \sum_{i=1}^{n} \theta_i x_i\right)\left(1-\sigma\left(\theta_0 + \sum_{i=1}^{n} \theta_i x_i\right)\right)x_i = \hat{y}(x)(1-\hat{y}(x))x_i$$

$$\frac{\partial L(\hat{y}(x),y)}{\partial \theta_{>0}} = -\mathrm{y} * \frac{1}{\hat{y}(x)}\frac{\partial \hat{y}(x)}{\partial \theta_{>0}} - (1-y) * \frac{-1}{1-\hat{y}(x)}\frac{\partial \hat{y}(x)}{\partial \theta_{>0}} = \cdots = (\hat{y}-y)\, x_i$$

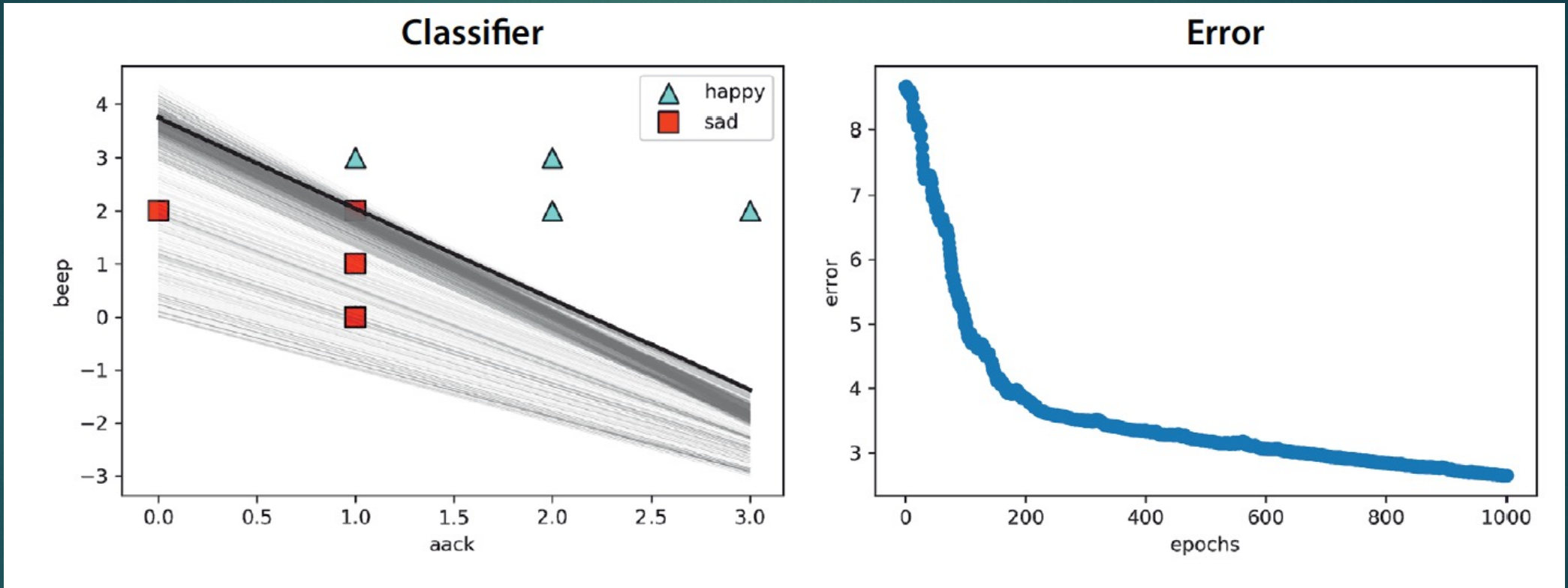
$$\frac{\partial L(\hat{y}(x), y)}{\partial \theta_0} = (\hat{y}-y)$$

Gradient Descent

$$\hat{\mathcal{R}}_\theta(\mathcal{D}) = \sum_{j=1}^{M} L\left(\hat{y}\left(x^{(j)}\right), y^{(j)}\right)$$

$$\nabla \hat{R}_{f_\theta}(\mathcal{D}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \\ \frac{\partial}{\partial \theta_1} \\ \vdots \\ \frac{\partial}{\partial \theta_{T-1}} \end{pmatrix} = \begin{pmatrix} \Sigma_{j=1}^{M}\left[\hat{y}\left(x^{(j)}\right) - y^{(j)}\right] \\ \Sigma_{j=1}^{M}\left[\hat{y}\left(x^{(j)}\right) - y^{(j)}\right] * x_1^{(j)} \\ \vdots \\ \Sigma_{j=1}^{M}\left[\hat{y}\left(x^{(j)}\right) - y^{(j)}\right] * x_{T-1}^{(j)} \end{pmatrix}$$

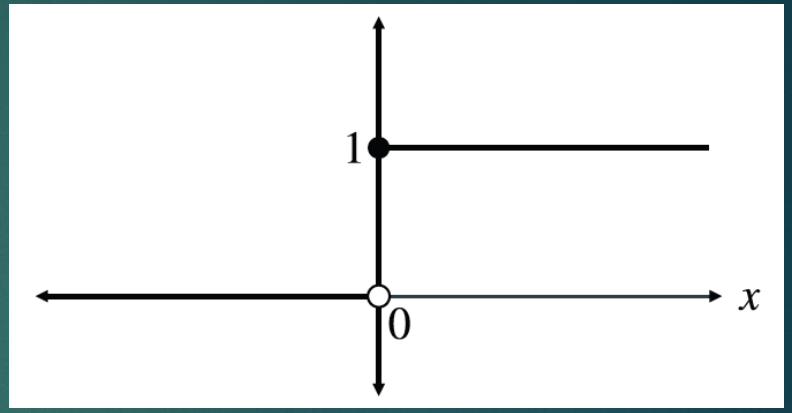
Logistic Classifier



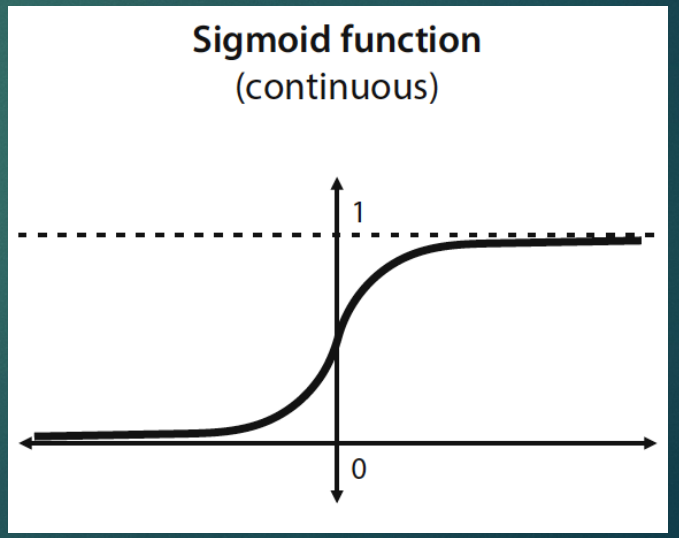
Perceptrons

Perceptron Classifier:

$f_{activation}(\quad) = step(\quad)$



$$\hat{y}(x) = f_{activation}\left(\theta_0 + \sum_{i=1}^{n} \theta_i x_i\right)$$



Logistic Classifier:

$f_{activation}(\quad) = \sigma(\quad)$

Multi Label Classifier

- Train one classifier for each class $l_i \in \mathcal{Y} = \{l_0, \cdots, l_{L-1}\}, i = 0 \ldots L-1$
- To classify a data point x obtain the L different scores
- Compute a probability distribution over $\mathcal{Y}$ by normalizing the scores

Multi Label Classifier: Example

- $\mathcal{Y} = \{"dog","cat","bird"\},\ L = 3$
- $scores(x) = \{3,2,-1\}$
- Get a probability distribution:
  - normalize $p(\hat{y}(x)) = \{3/5\,,2/5\,,-1/5\}$
  - use softmax $p(\hat{y}_i(x)) = \frac{e^{\hat{y}_i(x)}}{\sum_{l=0}^{L-1} e^{\hat{y}_l(x)}}$

$$p(\hat{y}(x)) = \{0.721,0.265,0.013\}$$