Montreal, July 8–12

DLMI2024

Basics of deep learning part 1

By

Pierre-Marc Jodoin

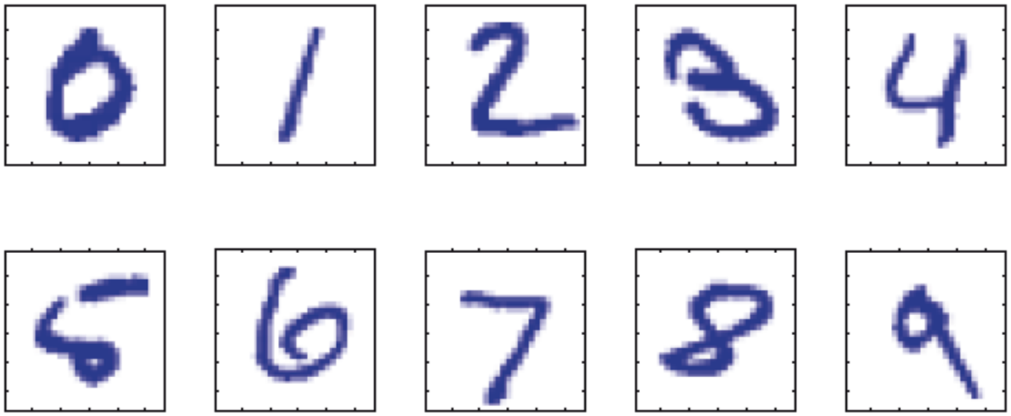
UNIVERSITÉ DE SHERBROOKE

Before we start

jodoin.github.io/dlmi2024

What is machine learning?

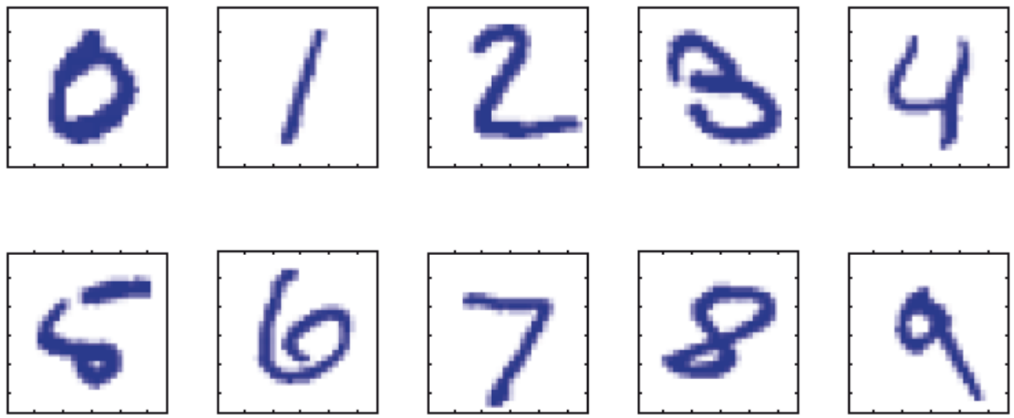
Question : how can one recognize hand written digits?



Answer : Design your own rules?

- A series of aligned pixels => ‘1’
- A circle of pixels => ‘0’
- Etc.

Question : how can one recognize hand written digits?

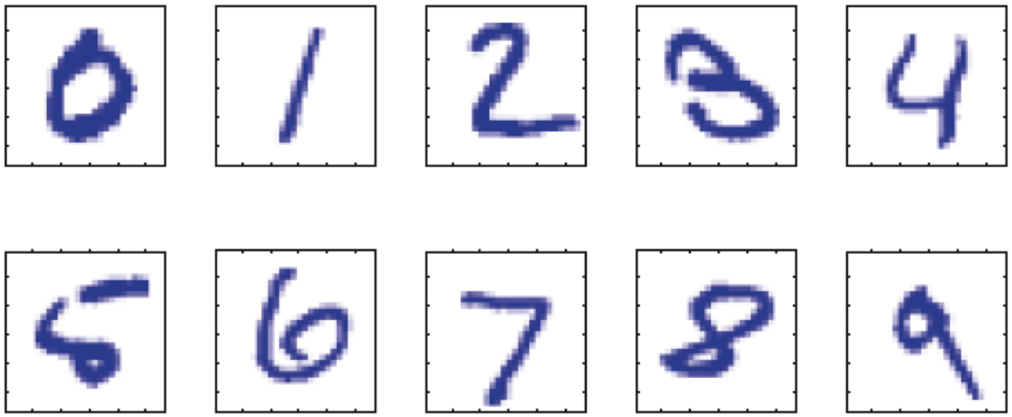


Answer : ~~Design your own rules?~~ **Wrong**

- Bad generalization

(Hugo Larochelle)

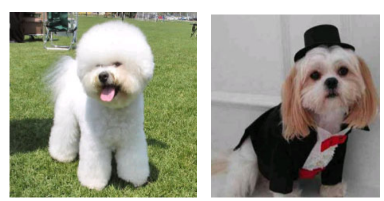
Question : how can one recognize hand written digits?



Answer : ~~Design your own rules?~~ **Wrong**

- Bad generalization

- Often difficult



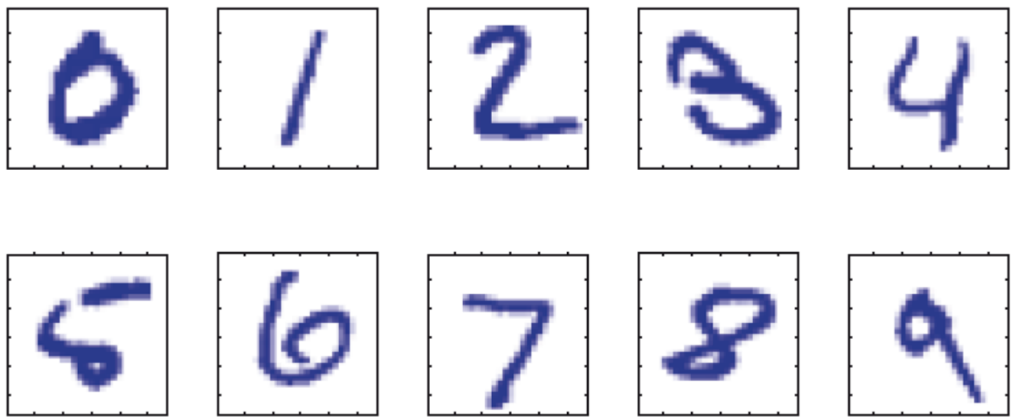
Dogs

Vs

Birds

(Hugo Larochelle)

Question : how can one recognize hand written digits?

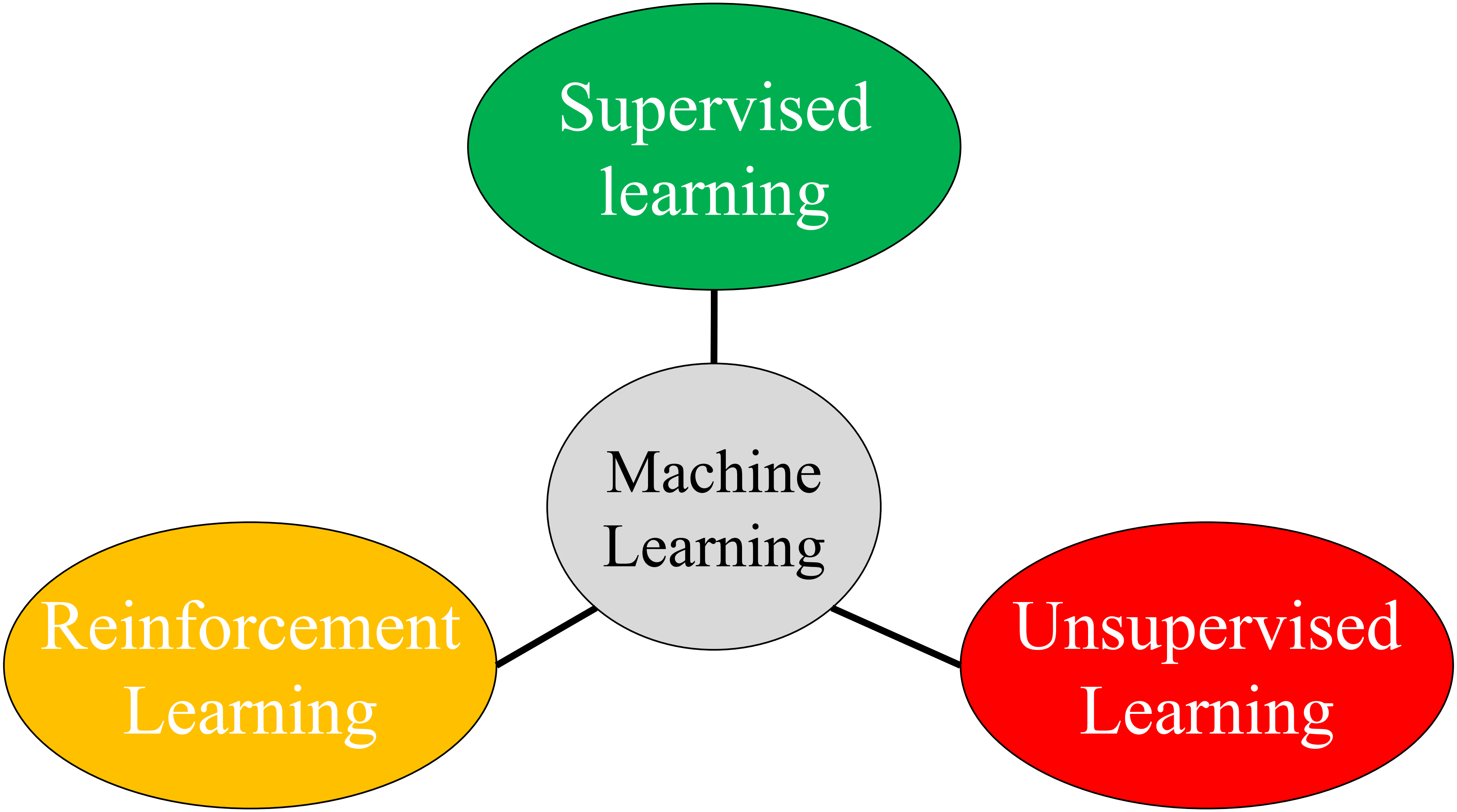


Answer : Let the computer « **learn** » the rules

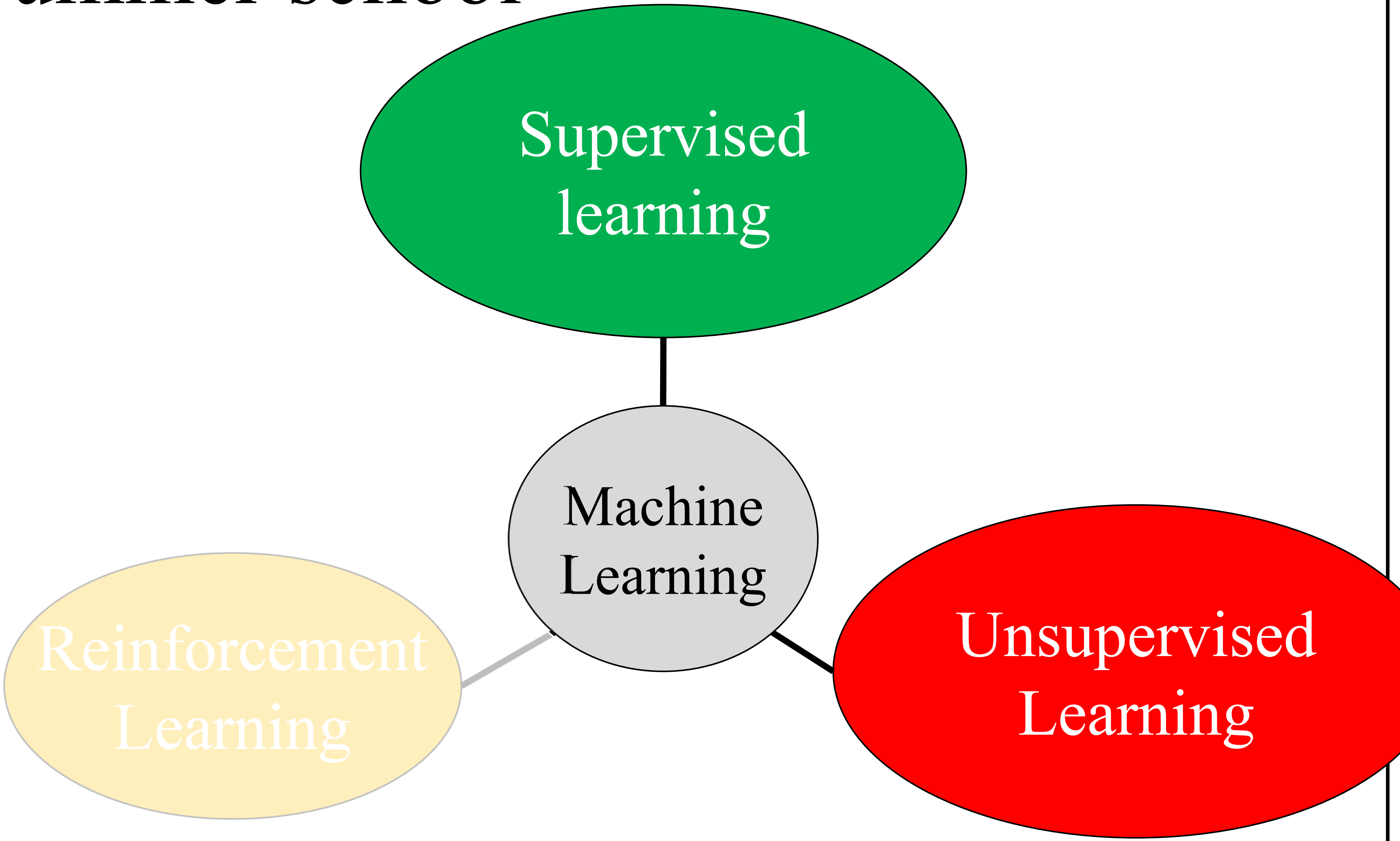
➢ *Main goal of machine learning*

(Hugo Larochelle)

Three large families



Summer school



Supervised learning

Provide the algorithm with **annotated training data**

‘0’ ‘0’ ‘0’ ‘1’ ‘0’ ‘1’ ‘1’

…and the algorithm returns a function capable of **generalizing** on new data

? ? ? ? ? ? ?

Supervised learning

Provide the algorithm with **annotated training data**

‘0’ ‘0’ ‘0’ ‘1’ ‘0’ ‘1’ ‘1’

The **training dataset**

$$D = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2), \ldots, (\vec{x}_N, t_N)\}$$

where $\vec{x}_i \in \Re^d$ is an **input** and t_i is a **target**

Goal of a supervised machine learning method

From a **training dataset:** $D = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2), \ldots, (\vec{x}_N, t_N)\}$

$\vec{x}_i \in \Re^d$ input data

t_i target associated to $\vec{x}_i$

the goal **is to learn** a function that may predict t_i given $\vec{x}_i$

$$y_W(\vec{x}_i) \rightarrow t_i$$

where W are the **parameters** of the model.

Supervised learning

Once the model $y_W(\vec{x})$ is trained, we use a **test set** D_{test} to gauge the **generalization** capabilities of the model.

‘0’ ‘0’ ‘1’ ‘1’ ‘1’ ‘0’ ‘0’ ‘1’

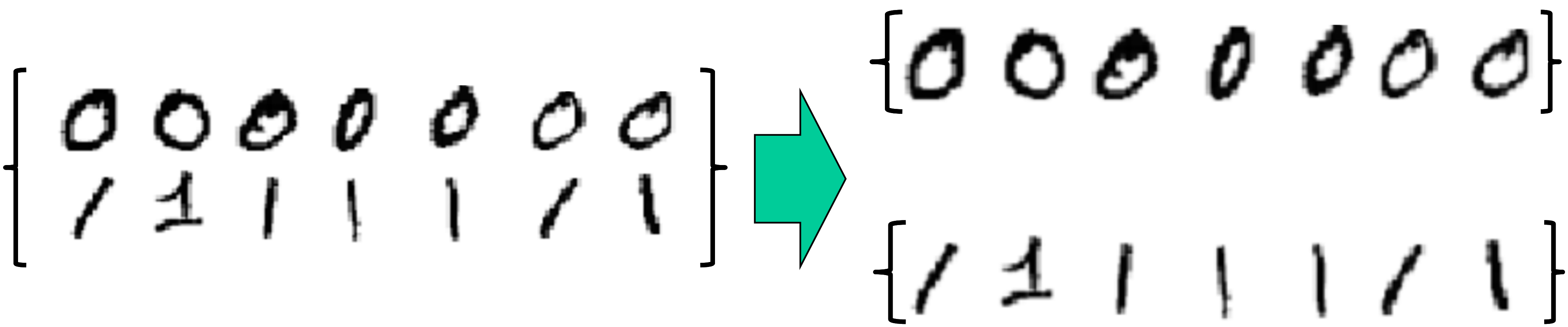
Supervised learning

Unsupervised learning

Unsupervised learning

When no target is explicitly provided

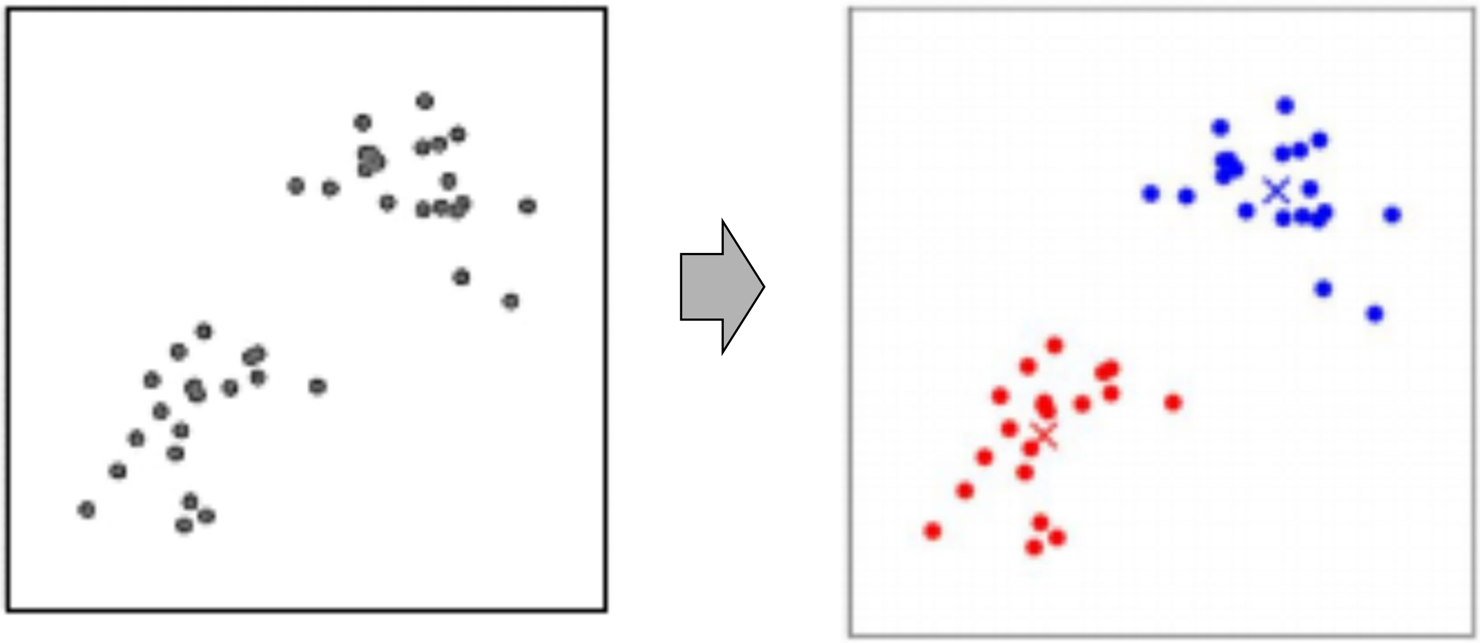
- E.g. data *clustering*



Unsupervised learning

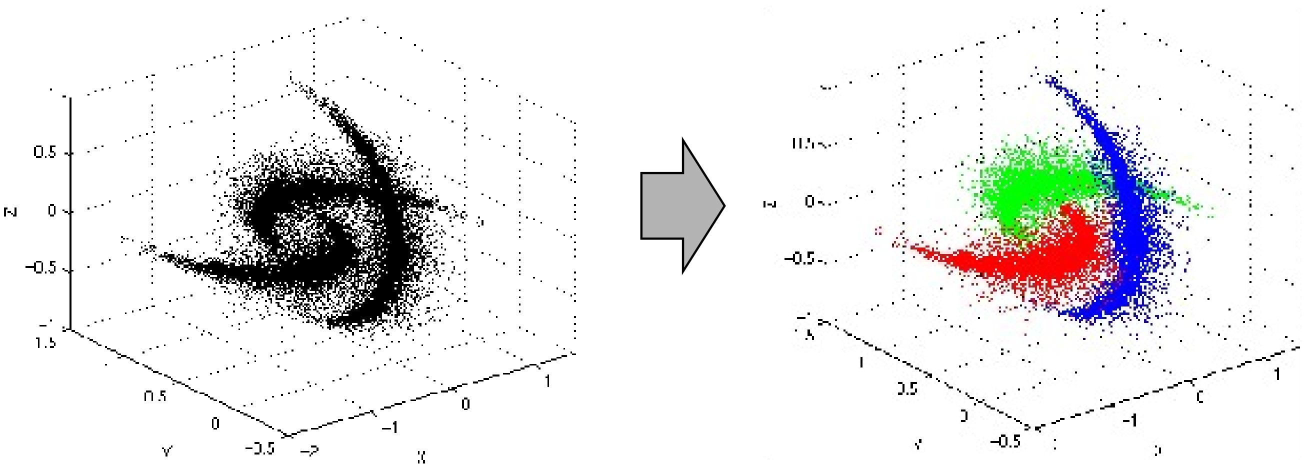
When no target is explicitly provided

- E.g. data *clustering*



Unsupervised learning

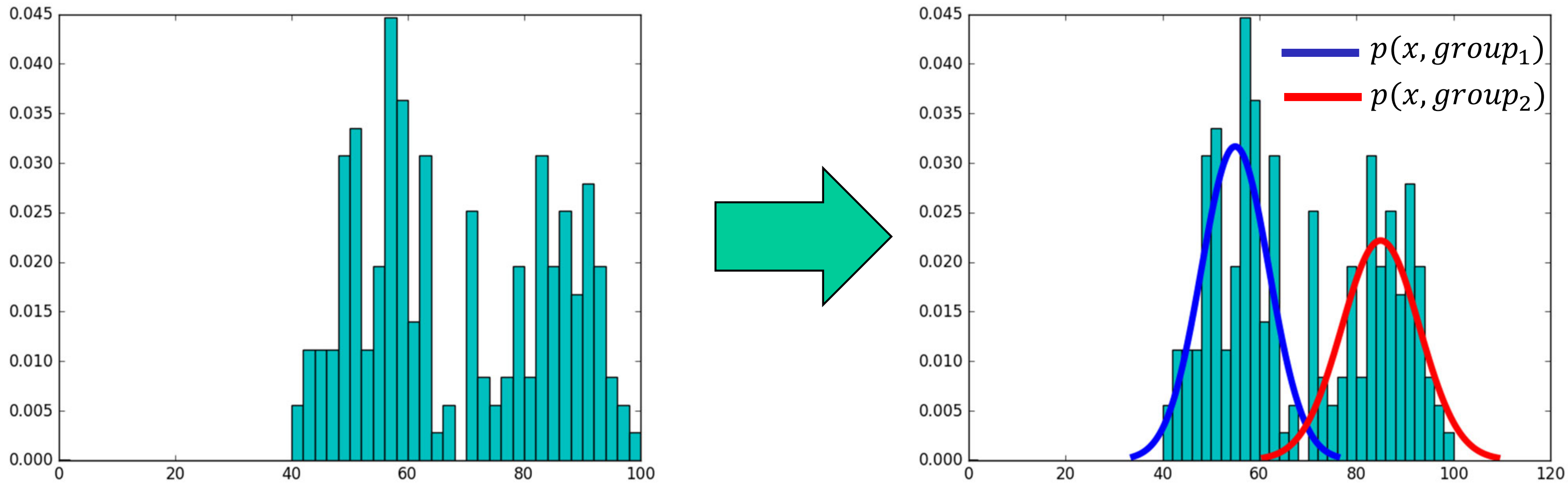
No limit to dimensionality. Could be 3D, 4D,…100kD



Unsupervised learning

Probability density function estimation

Example : find two groups of patients following a memory test



Supervised vs non-supervised

Main topic of the school

Supervised learning : there is a target

$$D = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2), \ldots, (\vec{x}_N, t_N)\}$$

Unsupervised learning : unknown target

$$D = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N\}$$

Supervised vs non-supervised

Supervised learning : there is a tar

$$D = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2)$$

Logistic regression
Perceptron
Multilayer perceptron
Convolutional neural networks
Recurrent neural networks
Semi-supervised learning
Graph Neural Nets
Transformers
Etc.

Unsupervised learning : unknown target

$$D = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N\}$$

Supervised vs non-supervised

Supervised learning : there is a target

$$D = \left\{ \left(\vec{x}_1, t_1 \right), \left(\vec{x}_2, t_2 \right), \ldots, \left(\vec{x}_N, t_N \right) \right\}$$

Unsupervised learning : unknown

Autoencoders
Variational autoencoders
GANs

$$D = \left\{ \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N \right\}$$

Supervised learning

Classification vs regression

Supervised learning

Two main applications

- **Classification :** the target is a class label $t \in \{1, \dots, K\}$
  - Exemple : disease recognition
    - ✓ $\vec{x}$: vector of medical measures, age, sex, etc.
    - ✓ t : {myocardial infarction, dilated cardiomyopathy, hypertrophic cardiomyopathy, normal}

- **Regression :** the target is a real number $t \in \mathbb{R}$
  - Exemple : prediction of life expectancy
    - ✓ $\vec{x}$: vector of medical measures, age, sex, etc.
    - ✓ t : number of months before death.

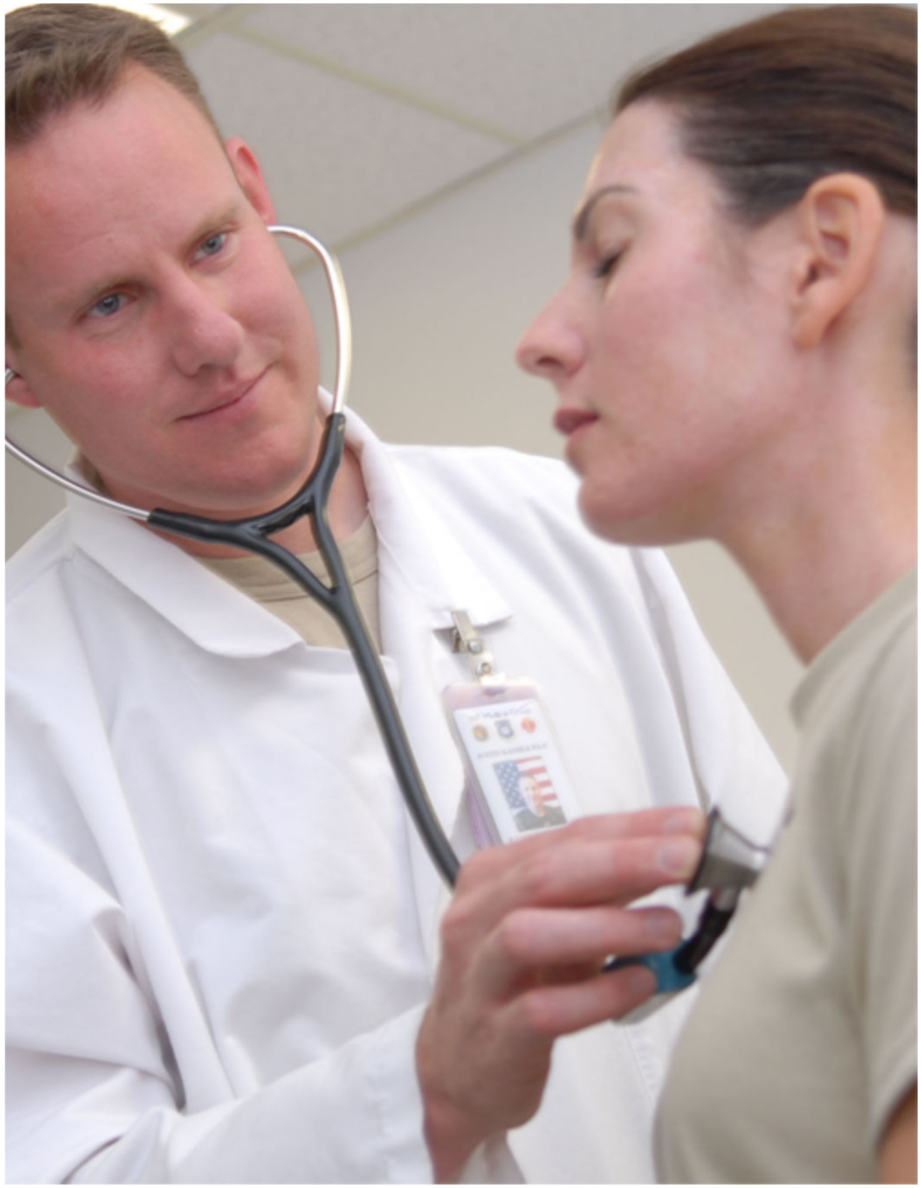
Supervised learning

Two main applications

- **Classification :** the target is a class label $t \in \{1, ... , K\}$
  - Exemple : disease recognition
    - ✓ $\vec{x}$: vector of medical measures, age, sex, etc.
    - ✓ t : myocardial infarction, dilated cardiomyopathy, hypertrophic cardiomyopathy, normal

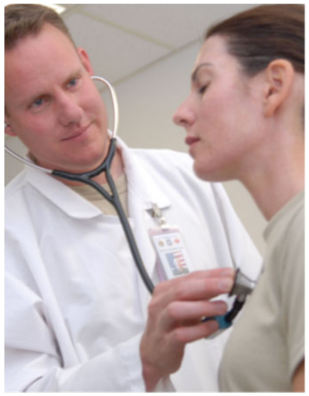
- **Régression :** the target is a real number $t \in \mathbb{R}$
  - Exemple : prediction of life expectancy
    - ✓ $\vec{x}$: vector of medical measures, age, sex, etc.
    - ✓ t : number of months before death.

Simple example of binary classification



From Wikimedia Commons
the free media repository

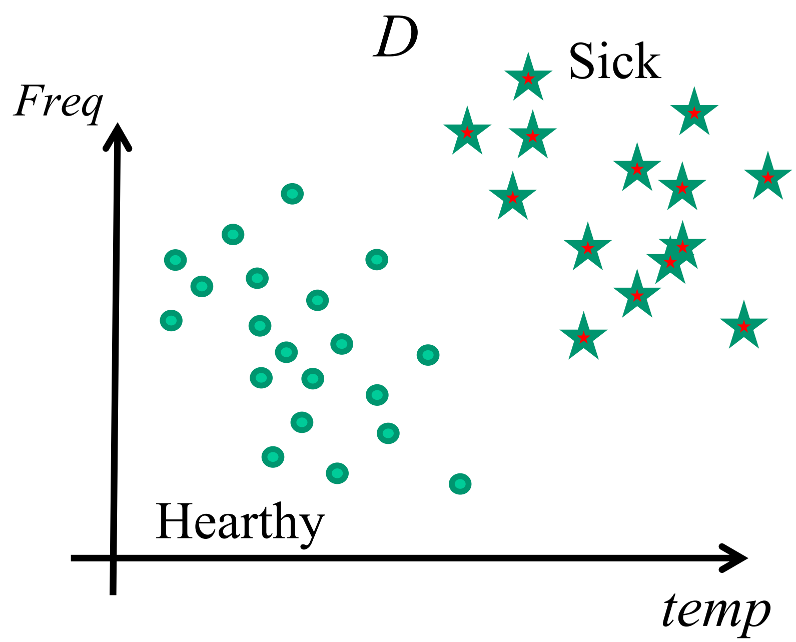
Simple example of binary classification



D

	(temp, freq)	Diagnostic
Patient 1	(37.5, 72)	hearthy
Patient 2	(39.1, 103)	sick
Patient 3	(38.3, 100)	sick
	(…)	…
Patient N	(36.7, 88)	hearthy

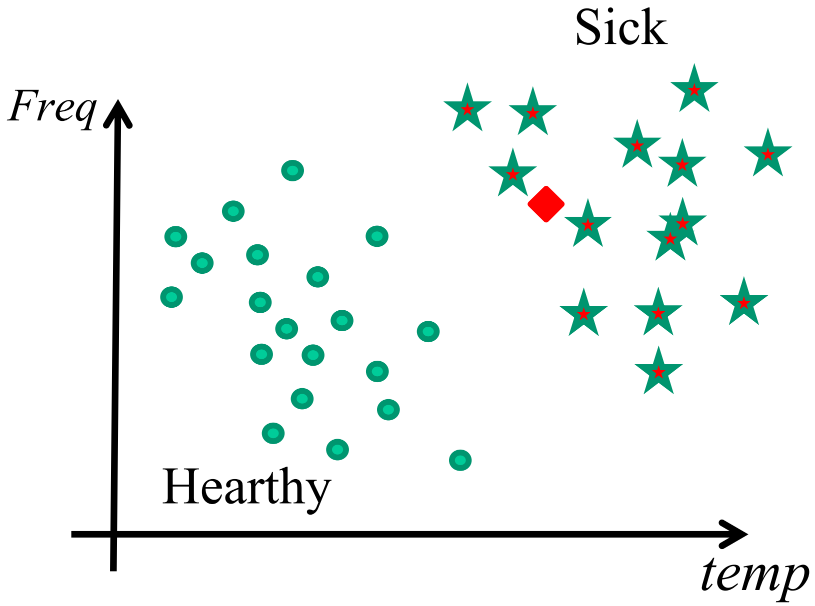
$\vec{x}$ t



Simple example of binary classification

From Wikimedia Commons
the free media repository

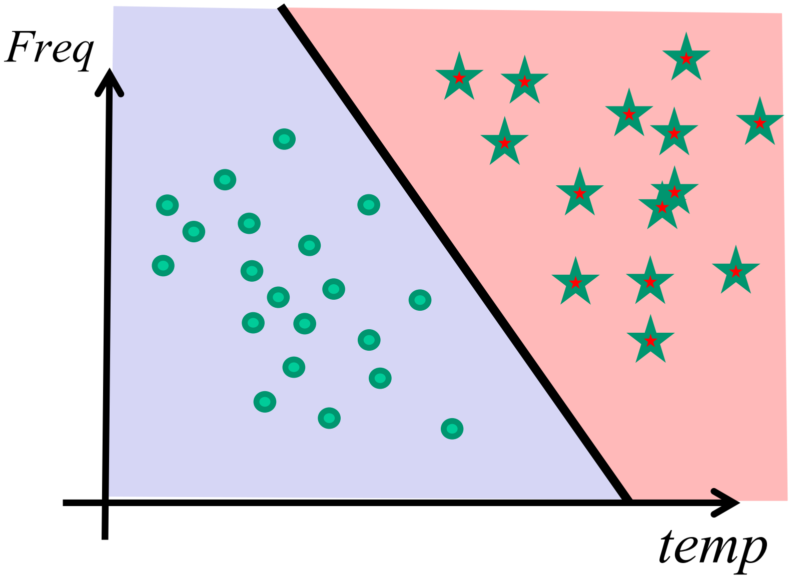
A new patient shows up at the hospital
How can we predict its state?



Solution

From Wikimedia Commons
the free media repository

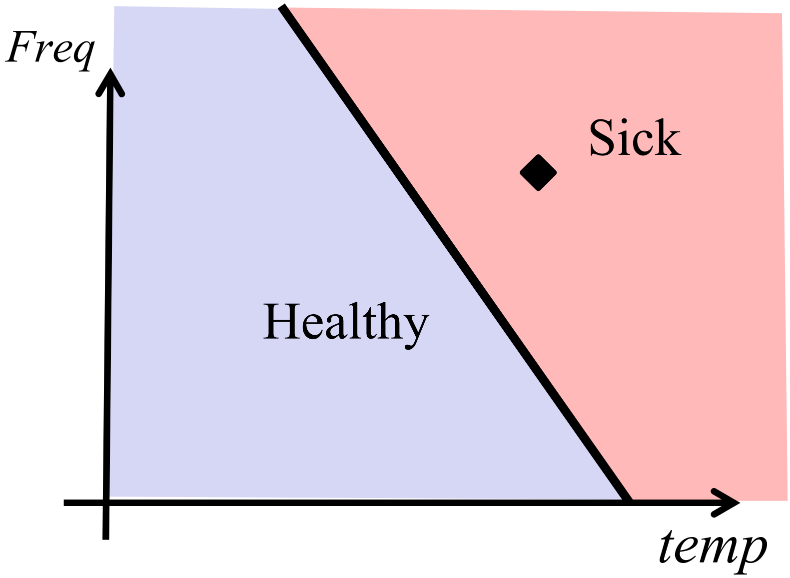
Divide the feature space in two regions : healthy and sick



Solution

From Wikimedia Commons
the free media repository

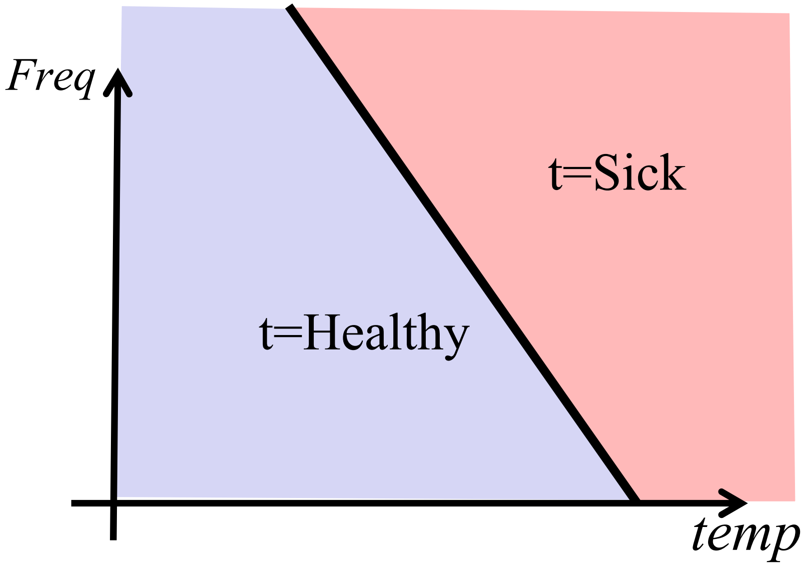
Divide the feature space in two regions : healthy and sick



More formally

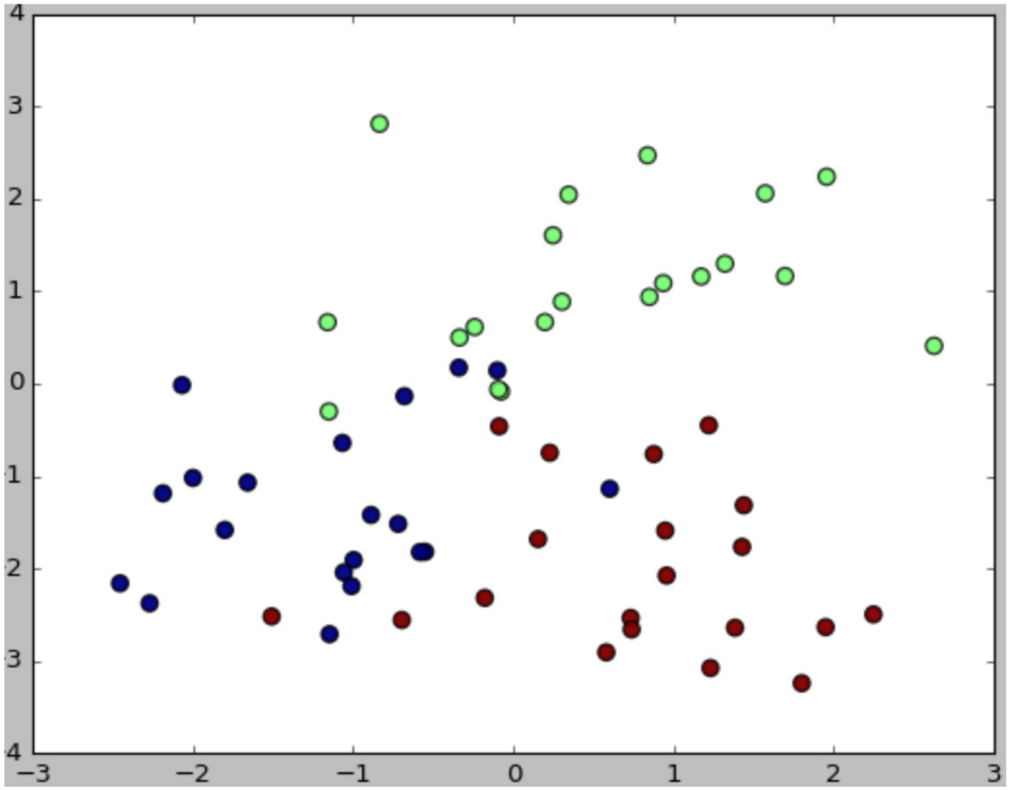
From Wikimedia Commons
the free media repository

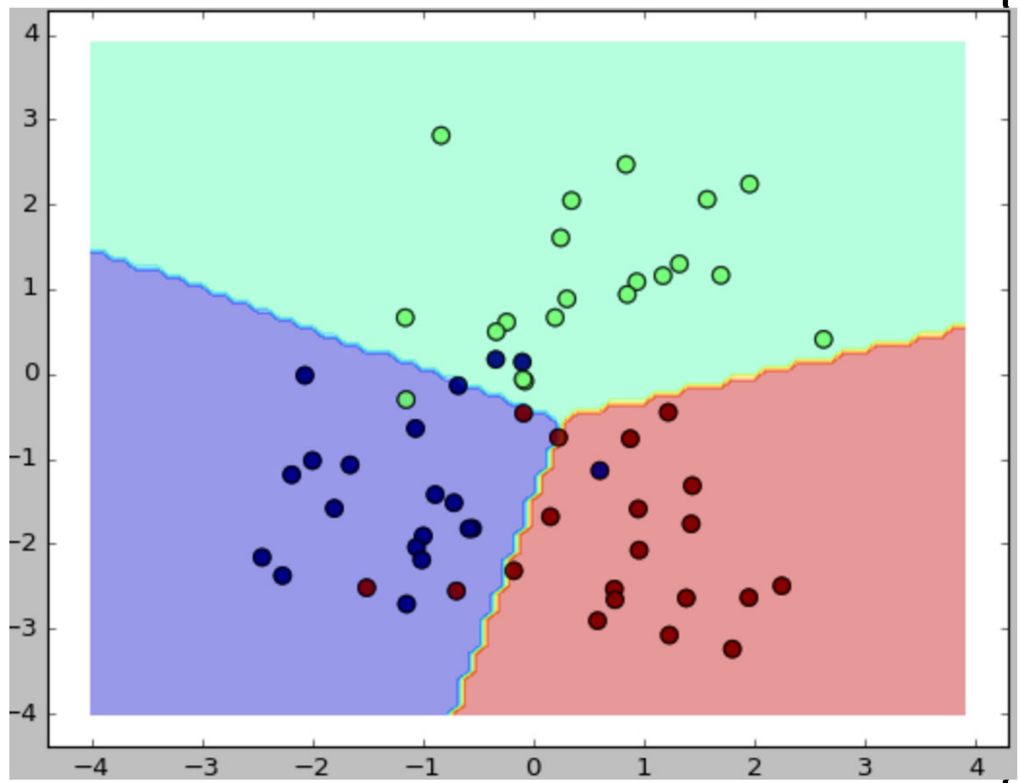
$$y_W(\vec{x}) = \begin{cases} Healthy \text{ if } \vec{x} \text{ is in the blue region} \\ Sick \quad \text{otherwise} \end{cases}$$



Classification

3-class case





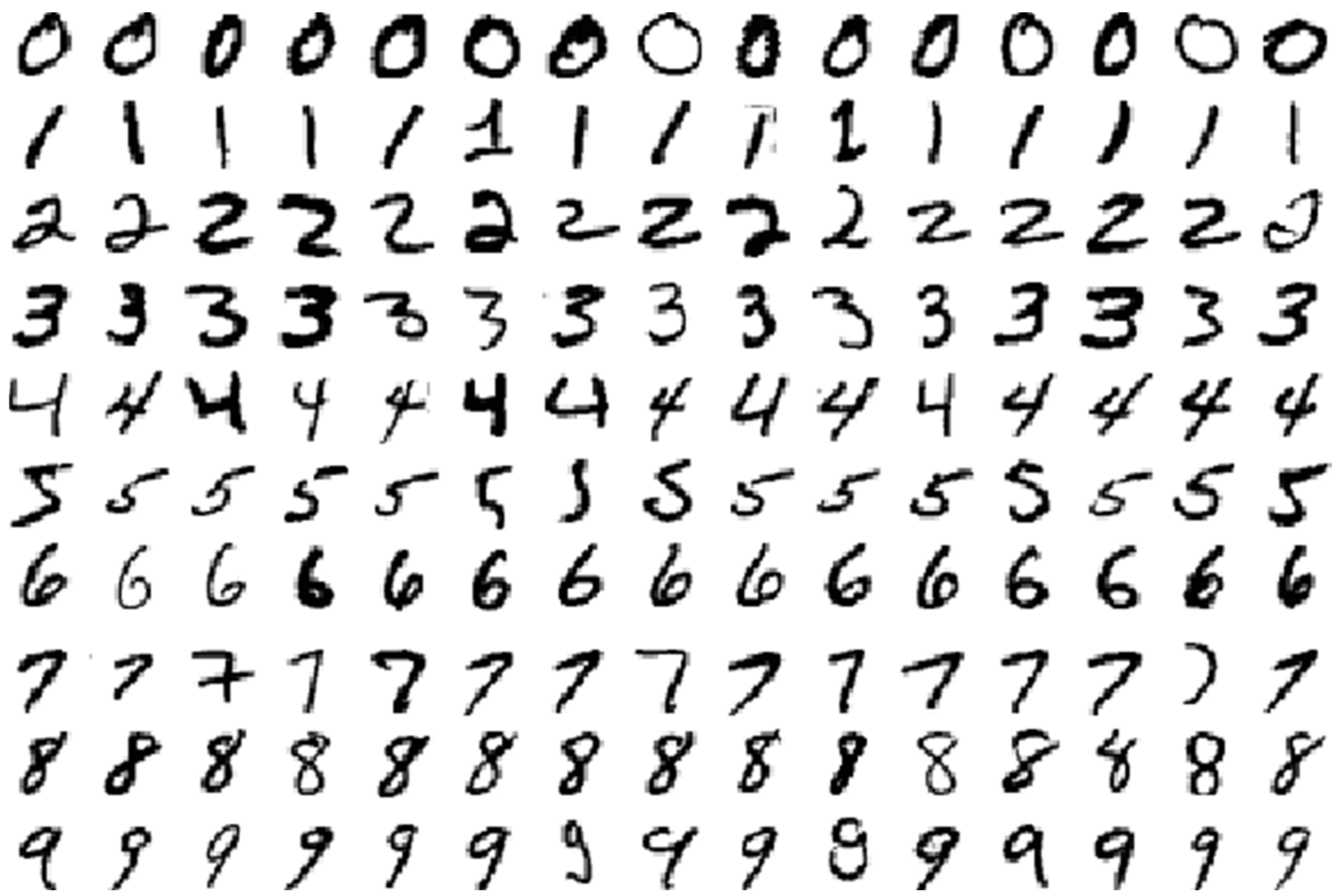
3 classes , , in a 2D feature space

Once training is over

$y_W(\ ) =$ class 1
$y_W(\ ) =$ class 2
$y_W(\ ) =$ class 3

Example of a classification dataset

MNIST



Example of a classification dataset

MNIST

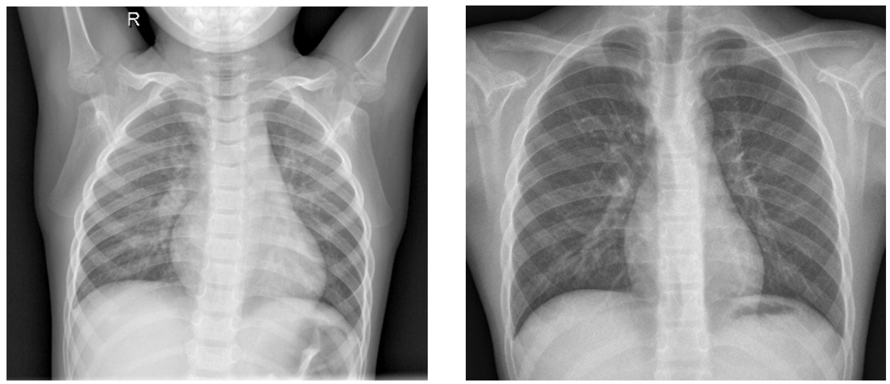
- 10 classes
- 70,000 images
  => 60,000 training
  => 10,000 test
- Images are in grayscale
  => 28x28

We can **vectorize these images** and represent it by a vector of size 28x28 = **784 dimensions.**

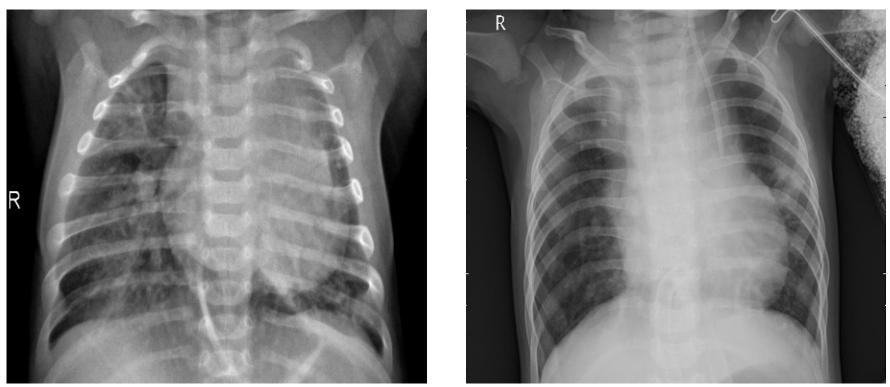
Example of a medical classification dataset

Chess X-Ray Pneumonia

Healthy



Pneumonia



https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia

Example of a medical classification dataset

Chess X-Ray Pneumonia

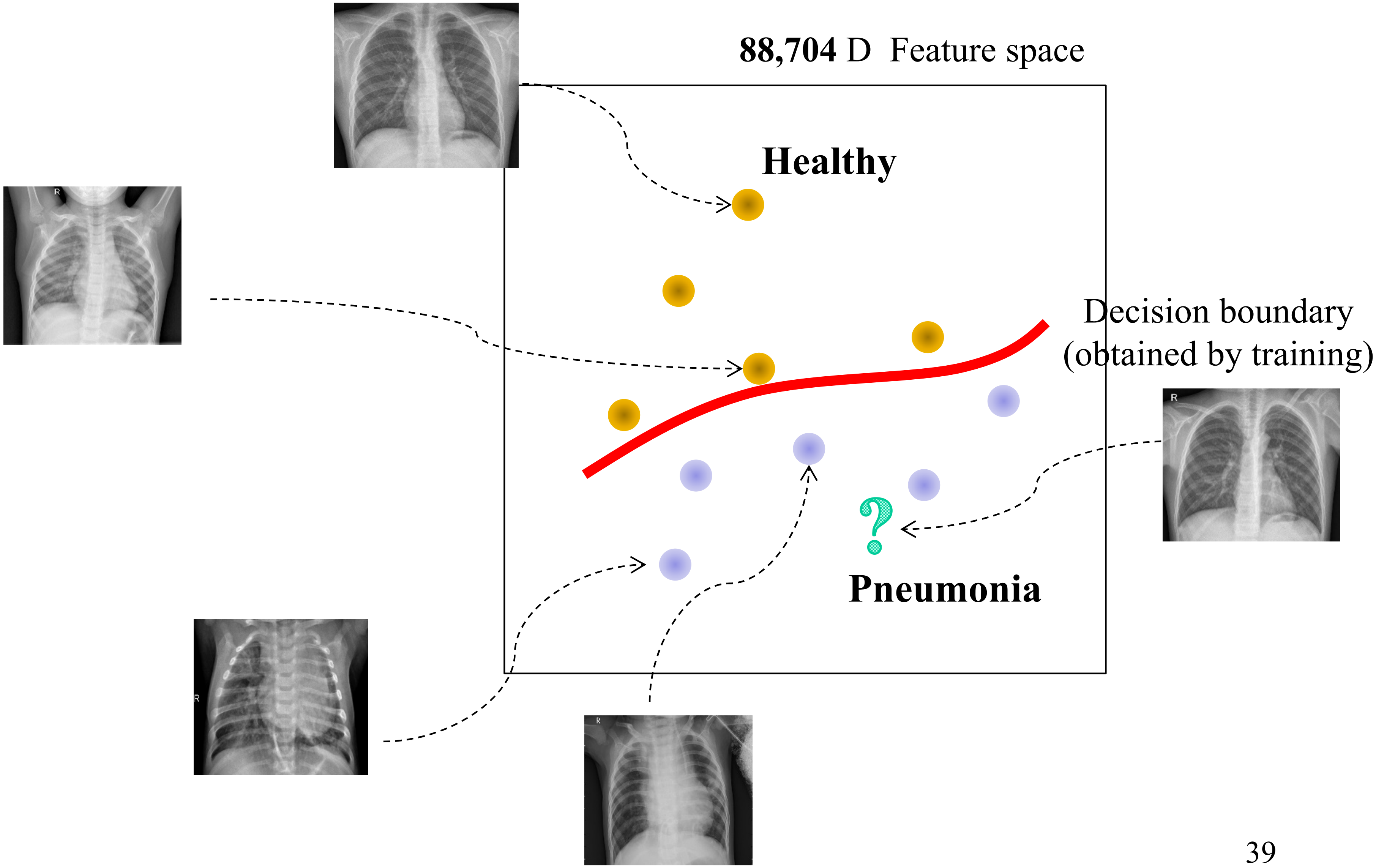
- 2 classes
- 5,840 images,
  => 5,216 training
  => 624 test
- Each image is in grayscale
  => 336 x 264*

We can **vectorize these images** and represent it by a vector of size 336x264 = **88,704 dimensions.**

* Rescaled version

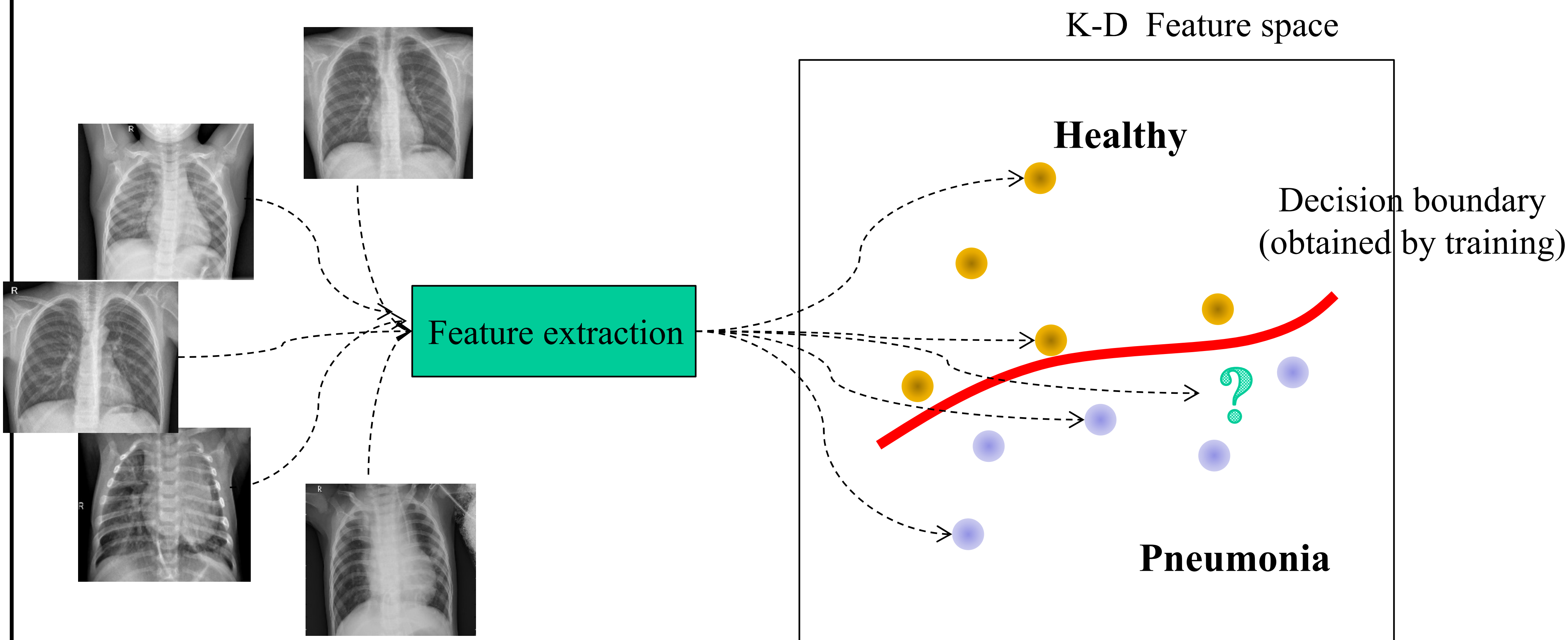
Supervised learning

Chess X-Ray Pneumonia



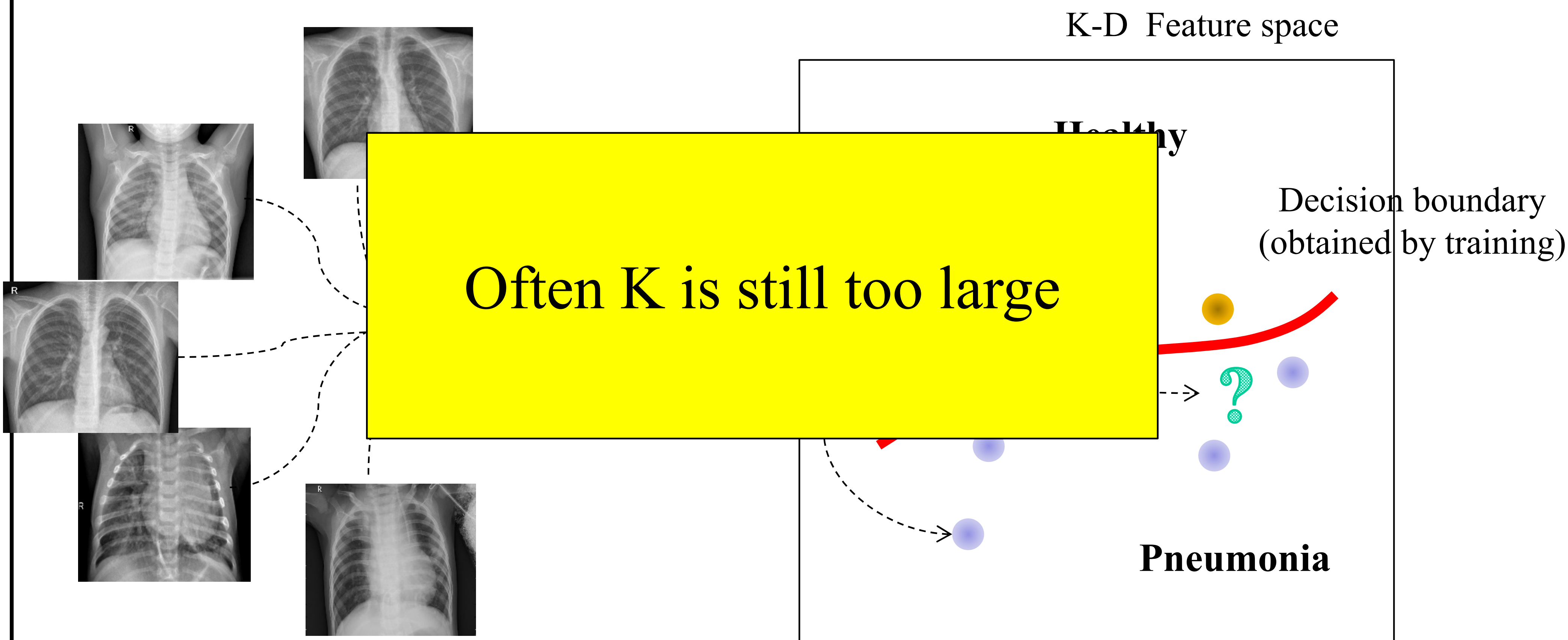
Very large feature spaces (like **88,704 dim) are problematic**.

Supervised learning



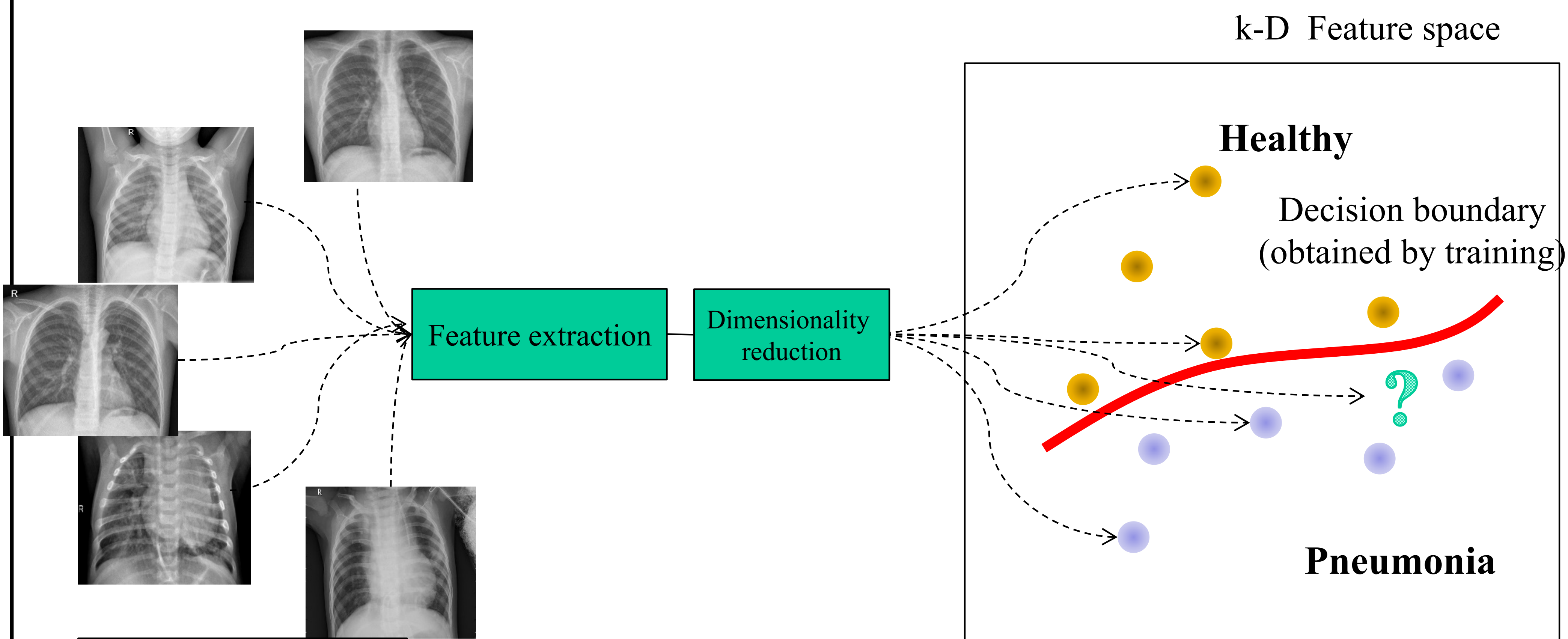
Edge detection
Histograms
Filters of all kinds
Moments
Etc.

Supervised learning



Often K is still too large

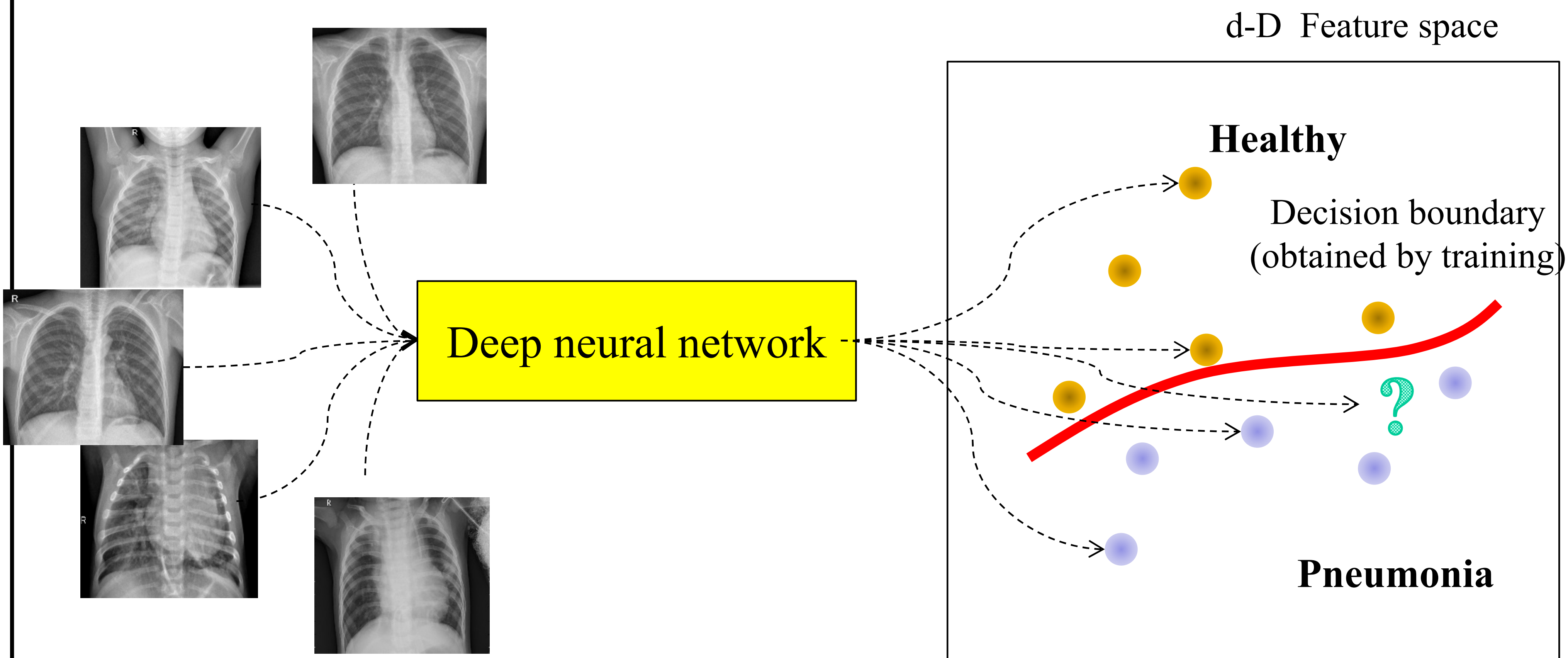
Supervised learning



Feature selection
PCA
Kernel PCA
LLE
ISOMAPS
Etc.

Spoiler alert

In 2024…



Deep neural network
d-D Feature space
Healthy
Decision boundary
(obtained by training)
?
Pneumonia

Supervised learning

Two main applications

- **Classification :** the target is a class label $t \in \{1, ... , K\}$
  - Exemple : disease recognition
    - ✓ $\vec{x}$: vector of medical measures, age, sex, etc.
    - ✓ t : myocardial infarction, dilated cardiomyopathy, hypertrophic cardiomyopathy, normal

- **Regression :** the target is a real number $t \in \mathbb{R}$
  - Exemple : prediction of life expectancy
    - ✓ $\vec{x}$: vector of medical measures, age, sex, etc.
    - ✓ t : number of months before death.

Example

(Linear)

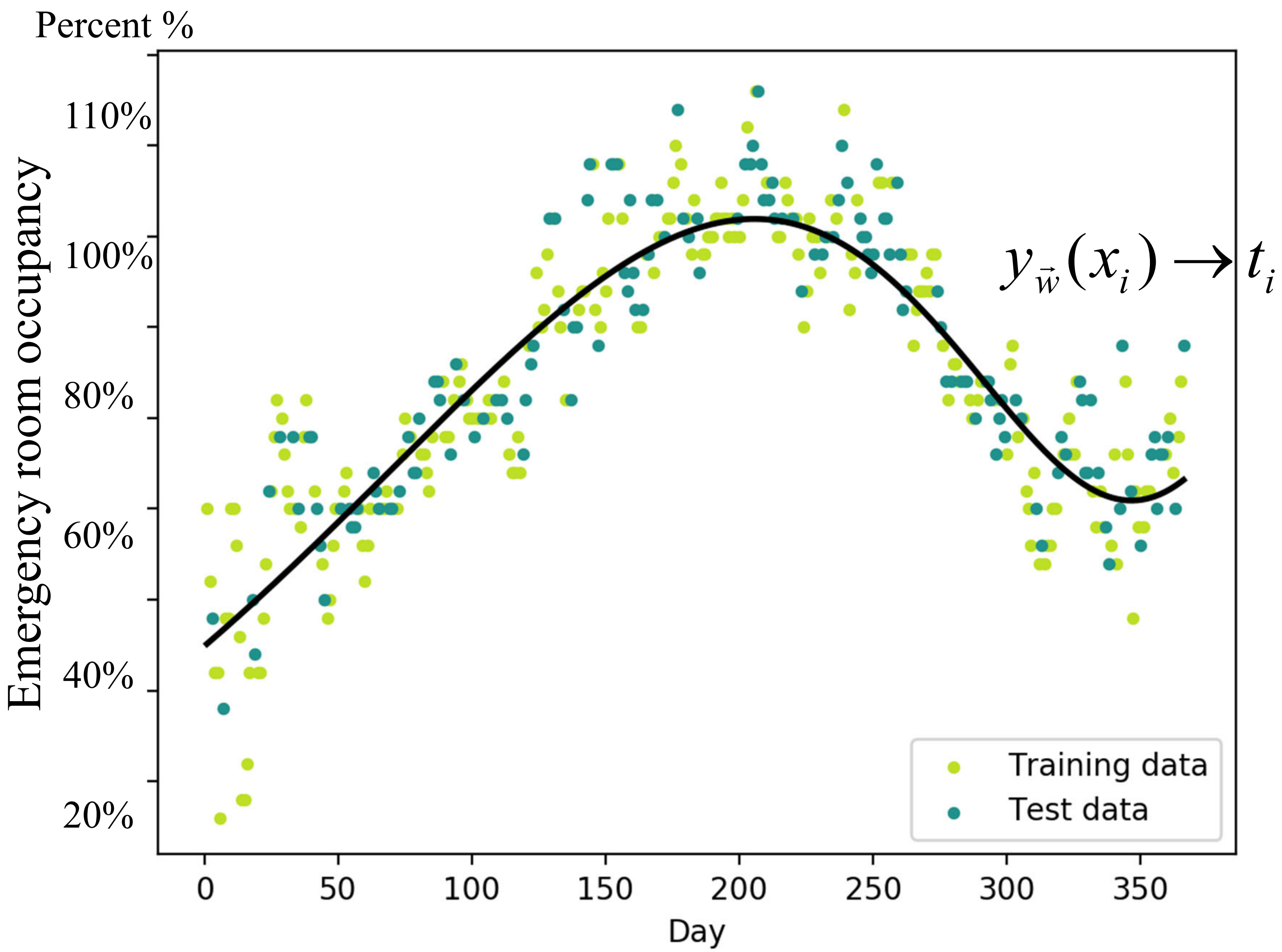
Medical Cost Personal Datasets

$$y_{\vec{w}}(x_i) \rightarrow t_i$$

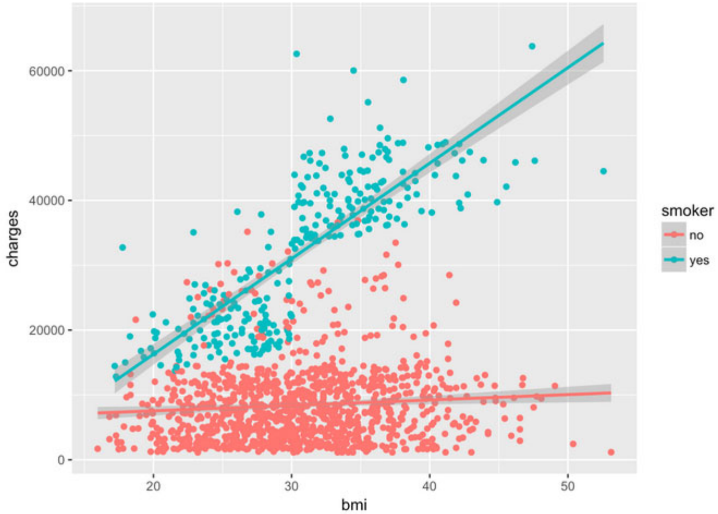
$$y_{\vec{w}}(x_i) \rightarrow t_i$$

https://rpubs.com/koki25ando/medicalcost

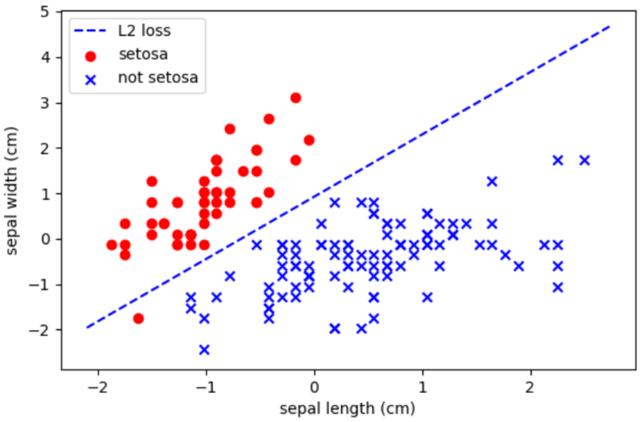
Example
(Nonlinear)



Linear models



https://rpubs.com/koki25ando/medicalcost



https://winder.ai/403-linear-classification/

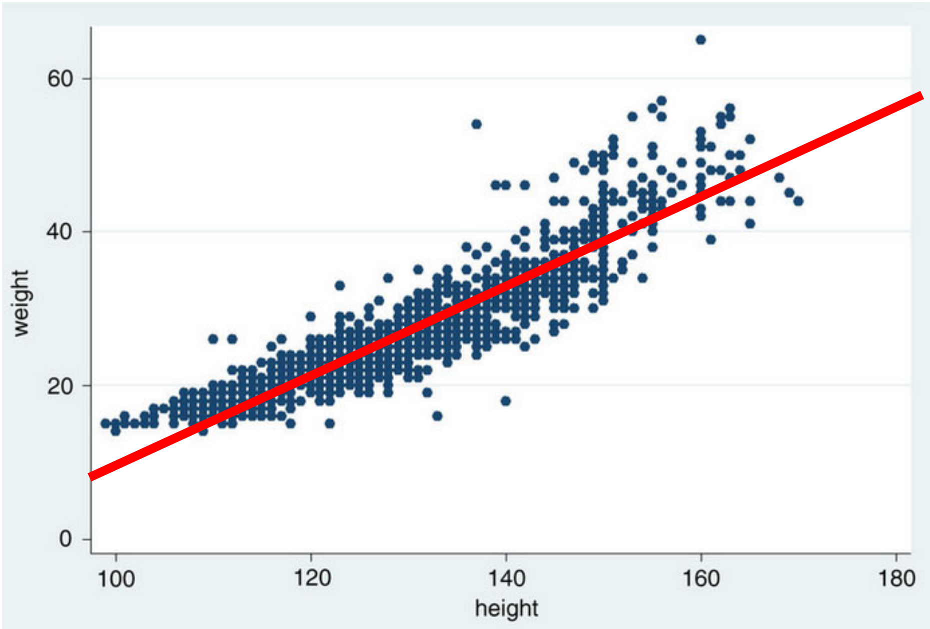
Deep neural nets … **linear models**

Vs
?

Linear models are to deep neural nets what transistors are to modern processors

Linear models are still relevant

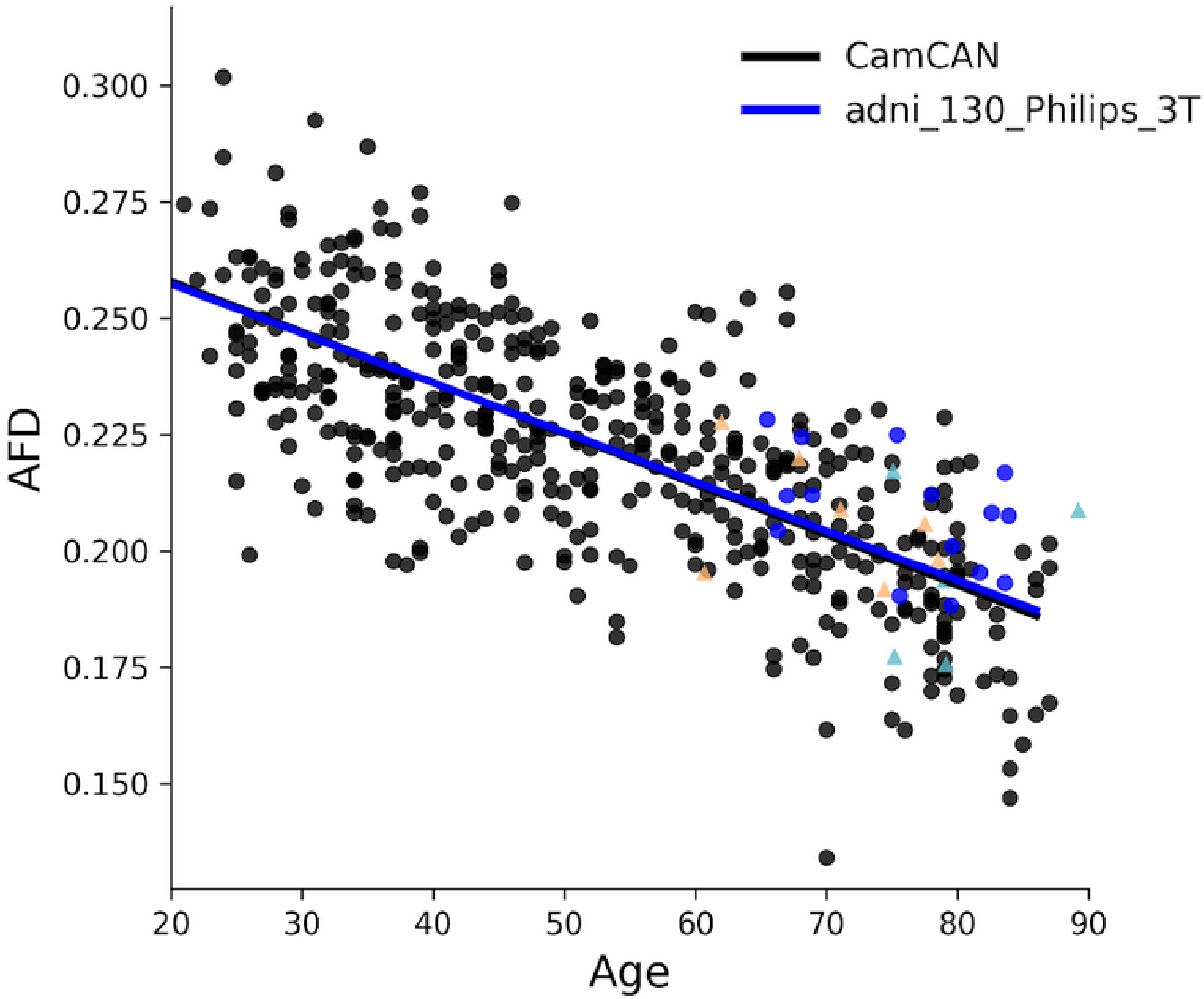
1,694 children surveyed in Tanzania.

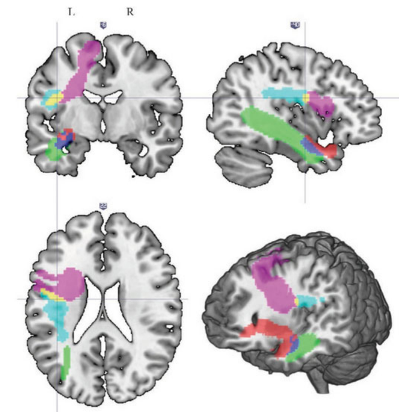


Nordin P, Poggensee G, Mtweve S, Krantz I**. From a weighing scale to a pole: a comparison of two different dosage strategies in mass treatment of Schistosomiasis haematobium**. Glob Health Action. 2014

Linear models are still relevant

Apparent Fiber Density
in the white matter





https://commons.wikimedia.org/

Linear models

Classification

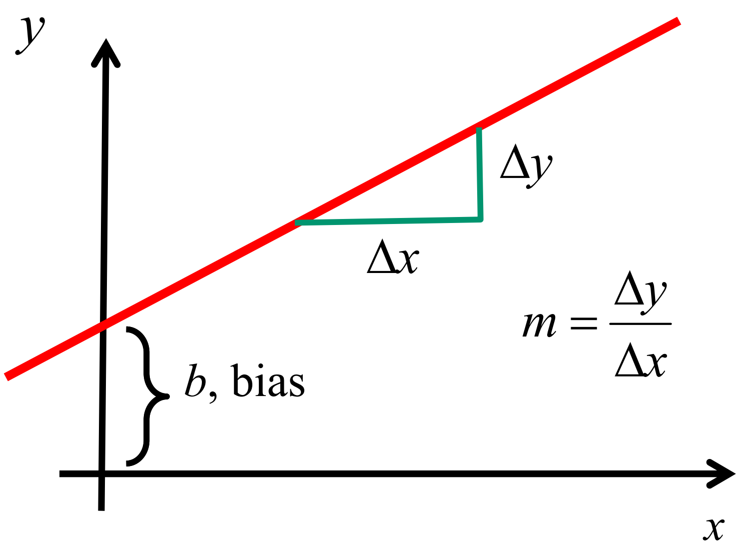
Regression

Linear models

Classification

Regression

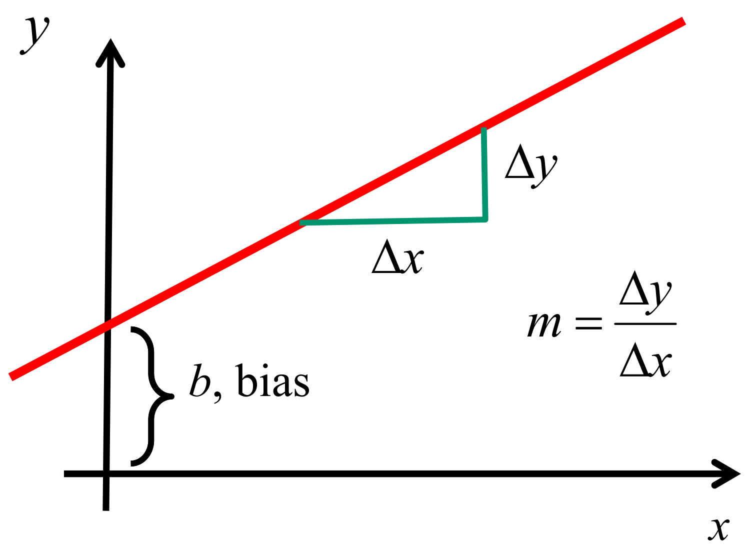
Definition … a line!



$$y = mx + b$$

slope   bias

Definition … a line!



$$y = mx + b$$

$$y = \frac{\Delta y}{\Delta x} x + b$$

$$y\Delta x = \Delta y x + b\Delta x$$

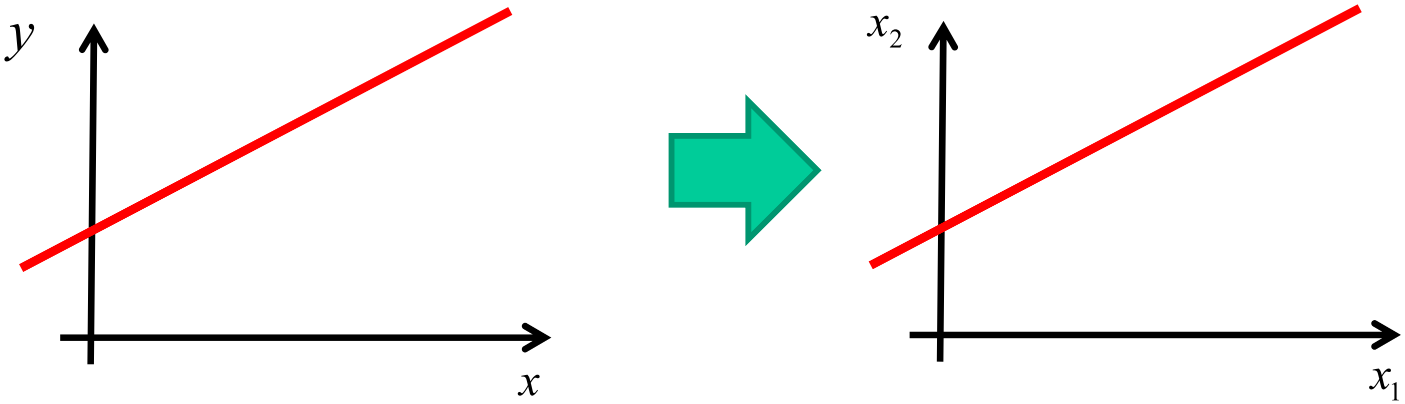
$$0 = \Delta y x - \Delta x y + b\Delta x$$

Rename variables

$$0 = \underbrace{\Delta y}_{w_1} x - \underbrace{\Delta x}_{w_2} y + \underbrace{b \Delta x}_{w_0}$$

Rename variables

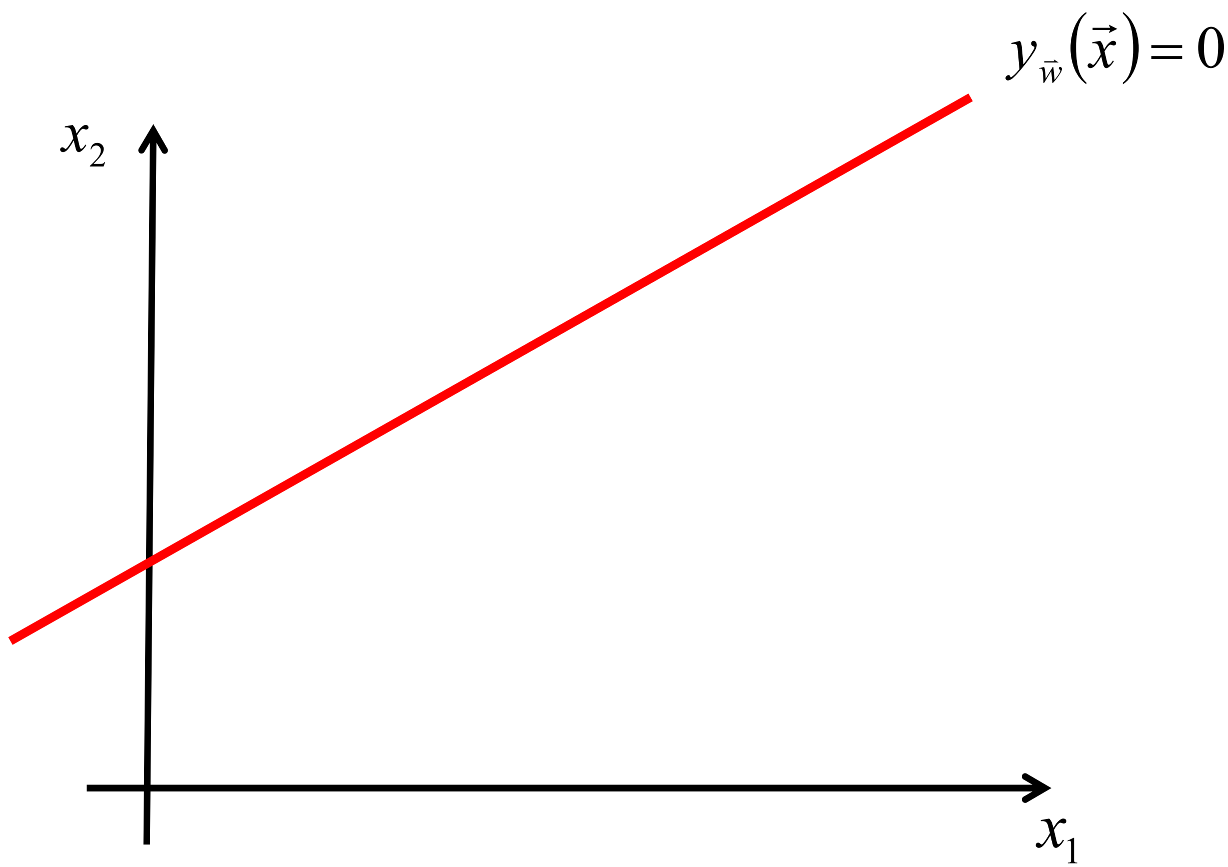
$$0 = w_1 x + w_2 y + w_0$$



$$0 = w_1 x_1 + w_2 x_2 + w_0$$

Implicit function

$$y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0$$



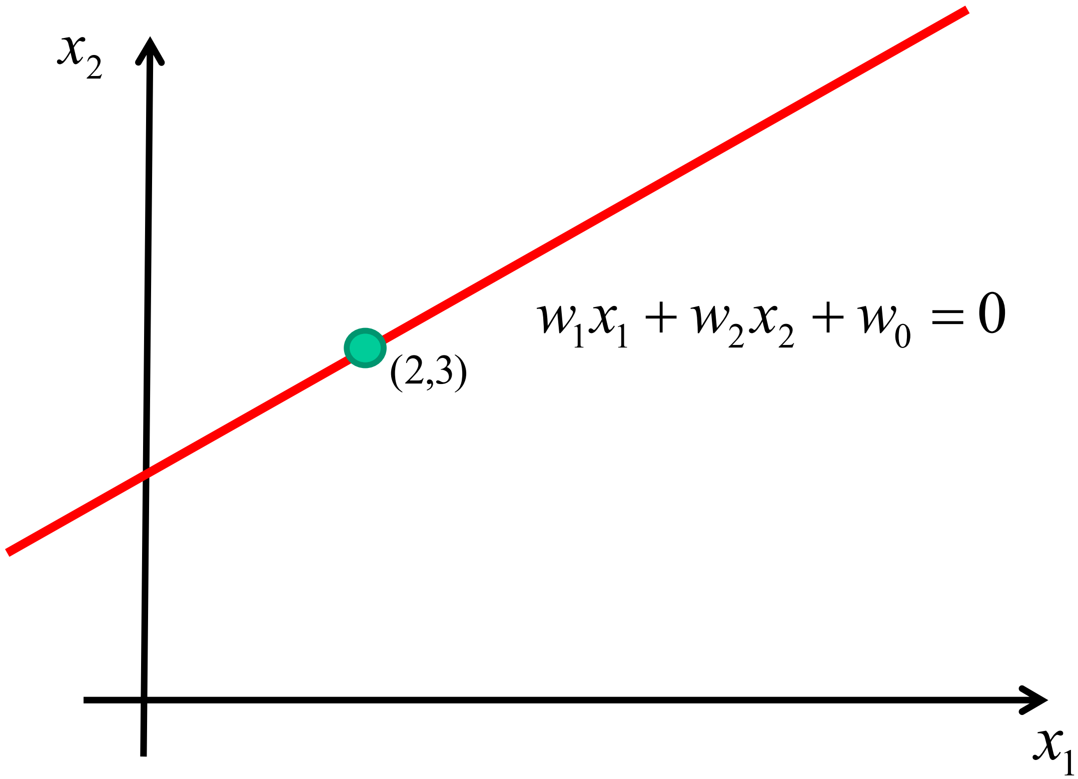
Implicit function

$$y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0$$

$w_1 = 1.0$

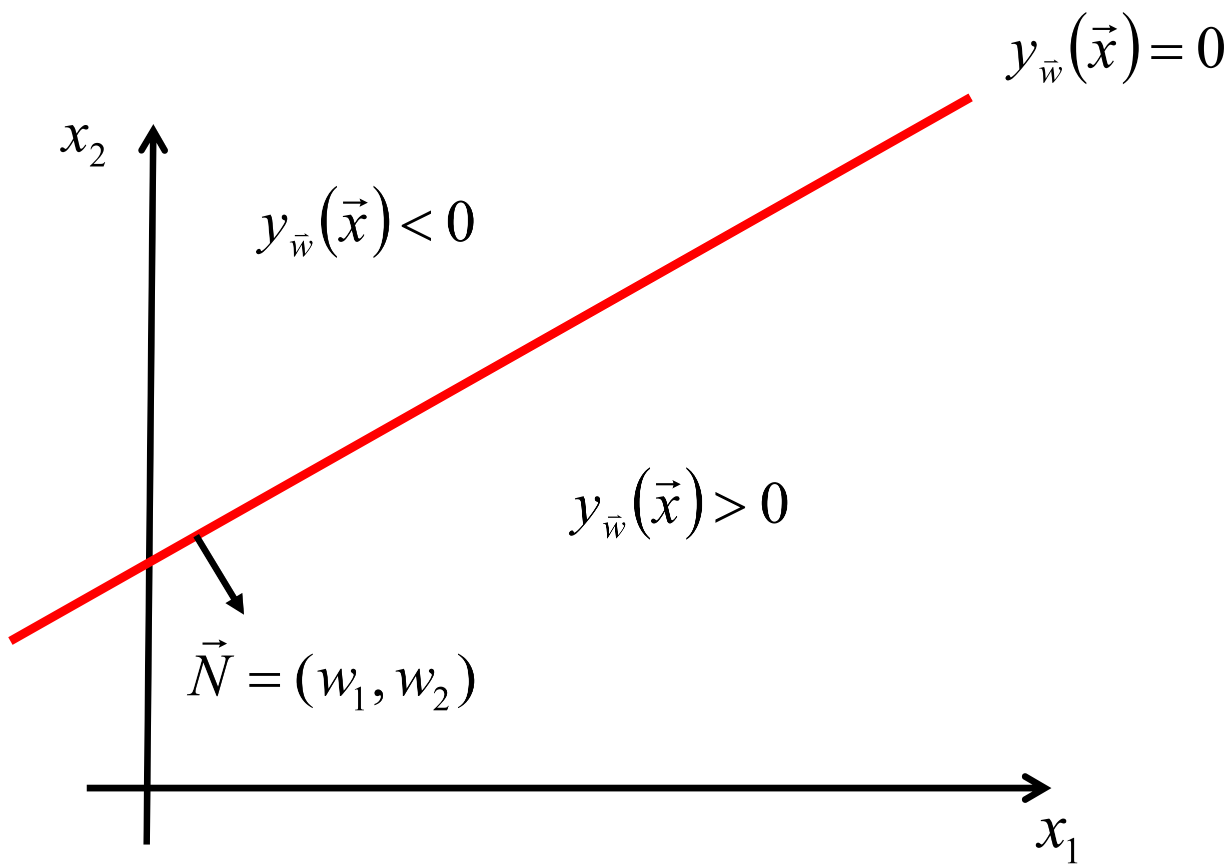
$w_2 = -2.0$

$w_0 = 4.0$



Classification function

$$y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0$$



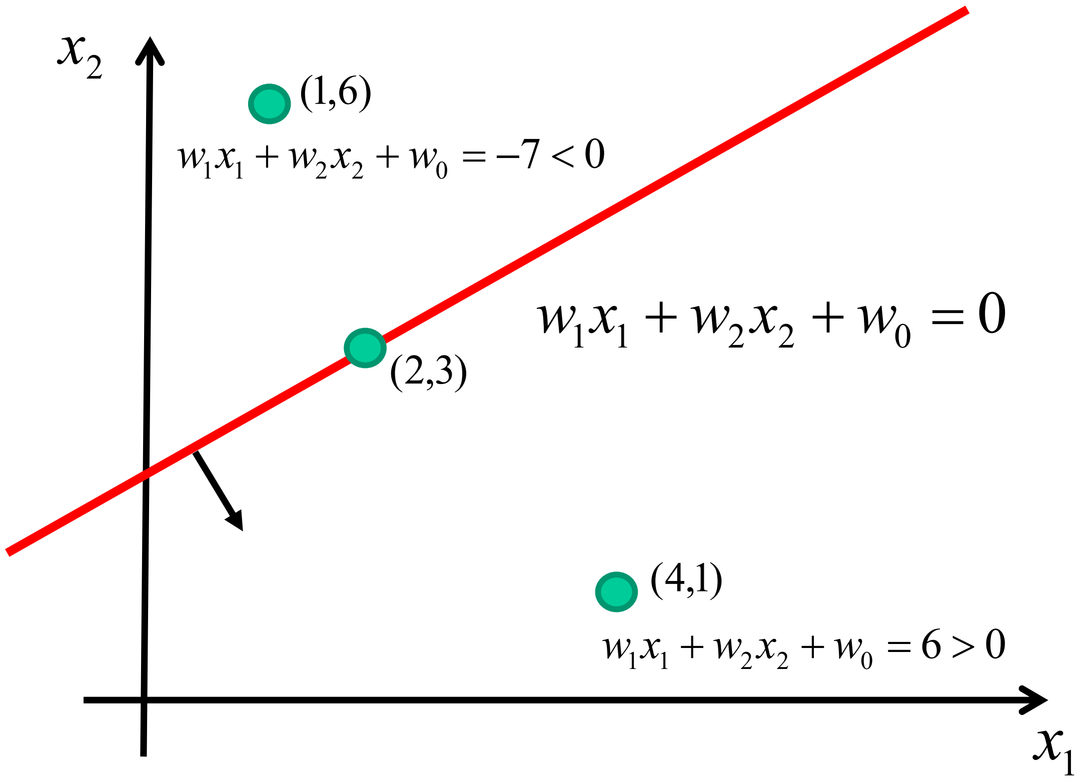
Classification function

$$y_{\vec{w}}(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0$$

$w_1 = 1.0$

$w_2 = -2.0$

$w_0 = 4.0$

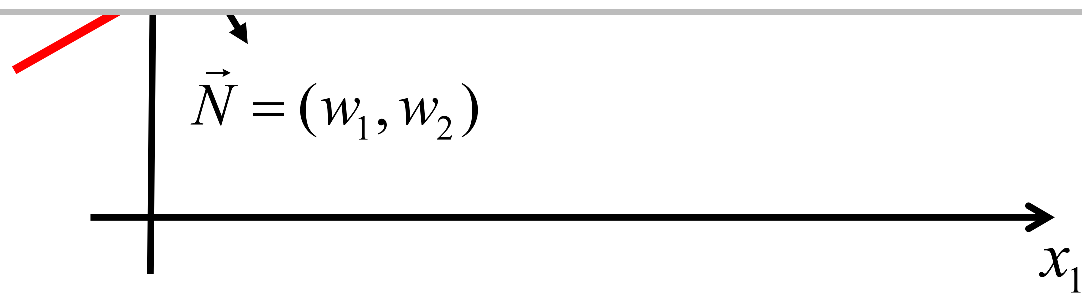


Implicit function

$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2$$

$$= \underbrace{(w_0, w_1, w_2)}_{\vec{w}} \cdot \underbrace{(1, x_1, x_2)}_{\vec{x}\,'}$$

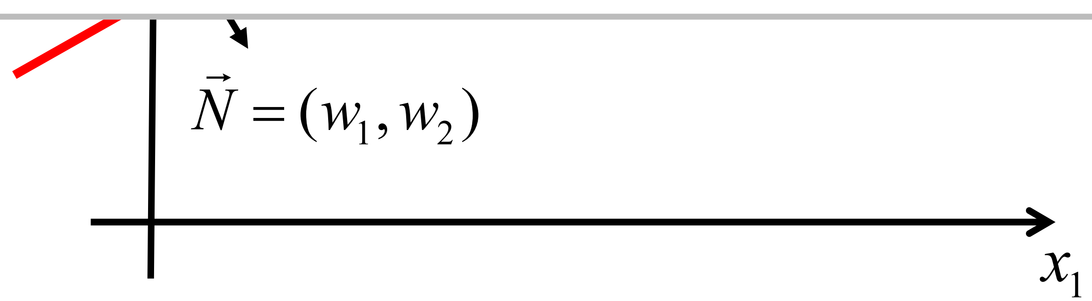
DOT product



Implicit function

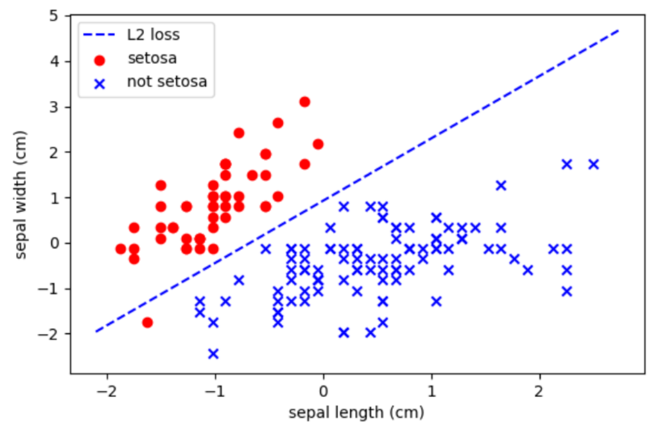
$$
\begin{aligned}
y_{\vec{w}}(\vec{x}) &= w_1 x_1 + w_2 x_2 + w_0 \\
&= (w_0, w_1, w_2) \cdot (1, x_1, x_2) \\
&= \vec{w}^T \vec{x}'
\end{aligned}
$$

DOT product



Linear classifier = dot product with bias included

$$y_{\vec{w}}(\vec{x}) = \vec{w}^T \vec{x} = \begin{cases} > 0 & \text{if in front} \\ < 0 & \text{otherwise} \end{cases}$$

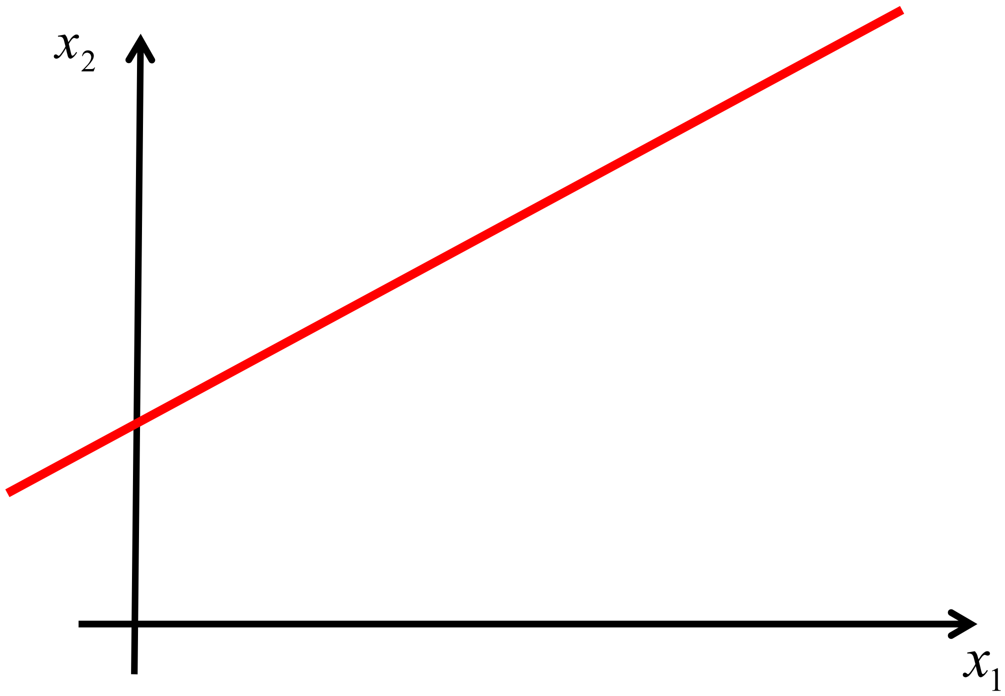


Linear models

Classification

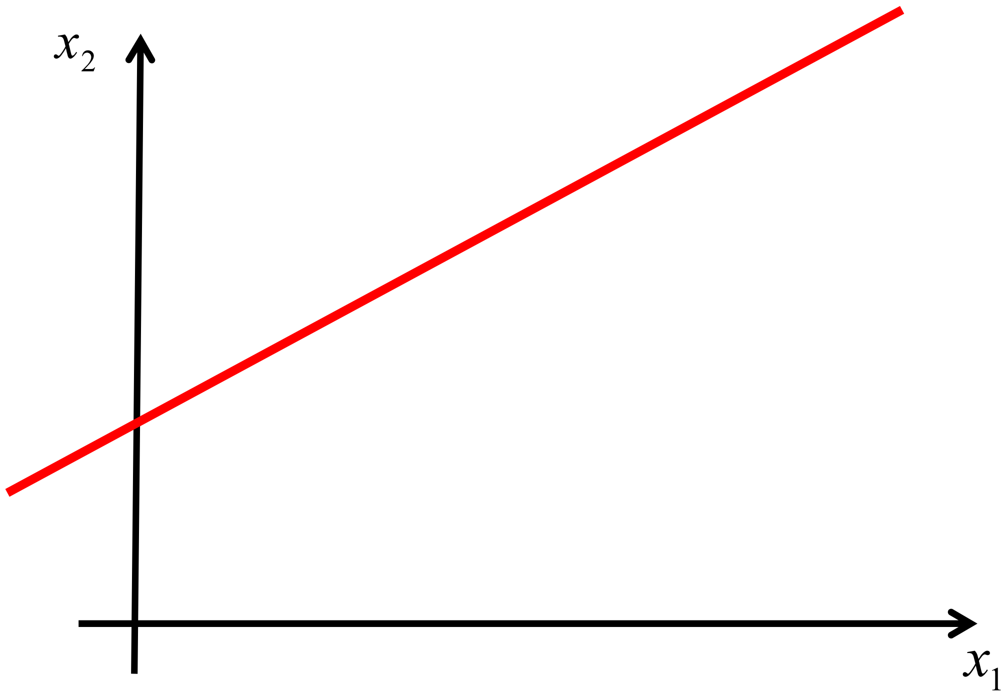
Regression

Classification : implicit function



$$0 = w_1 x_1 + w_2 x_2 + w_0$$

Explicit function

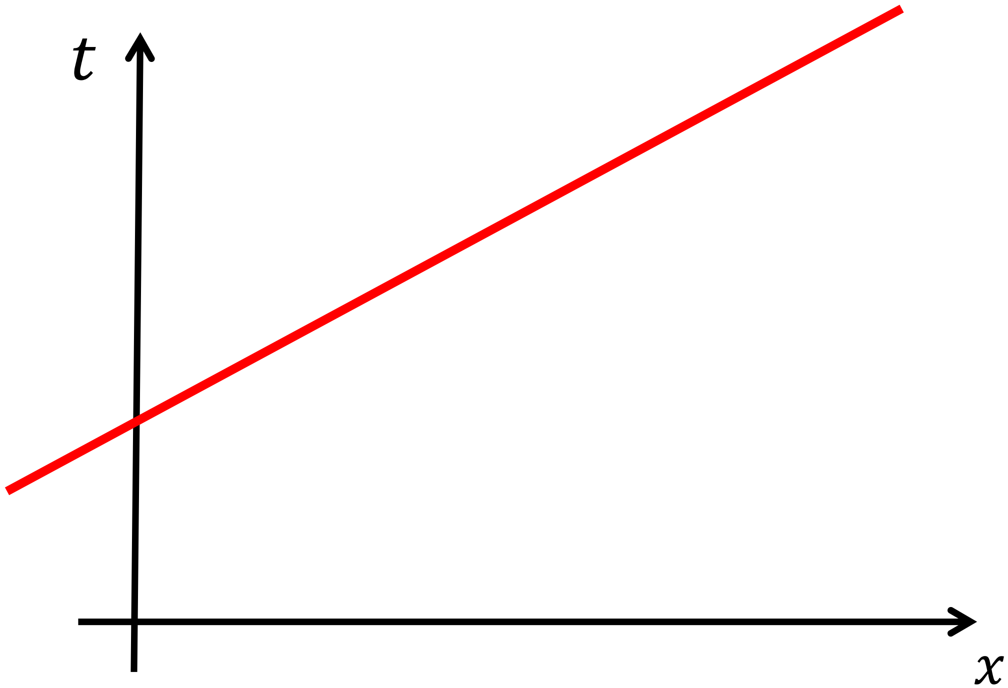


$$0 = w_1 x_1 + w_2 x_2 + w_0$$
$$x_2 = \frac{1}{-w_2}(w_1 x_1 + w_0)$$
$$x_2 = w'_1 x_1 + w'_0$$

Rename, t for target

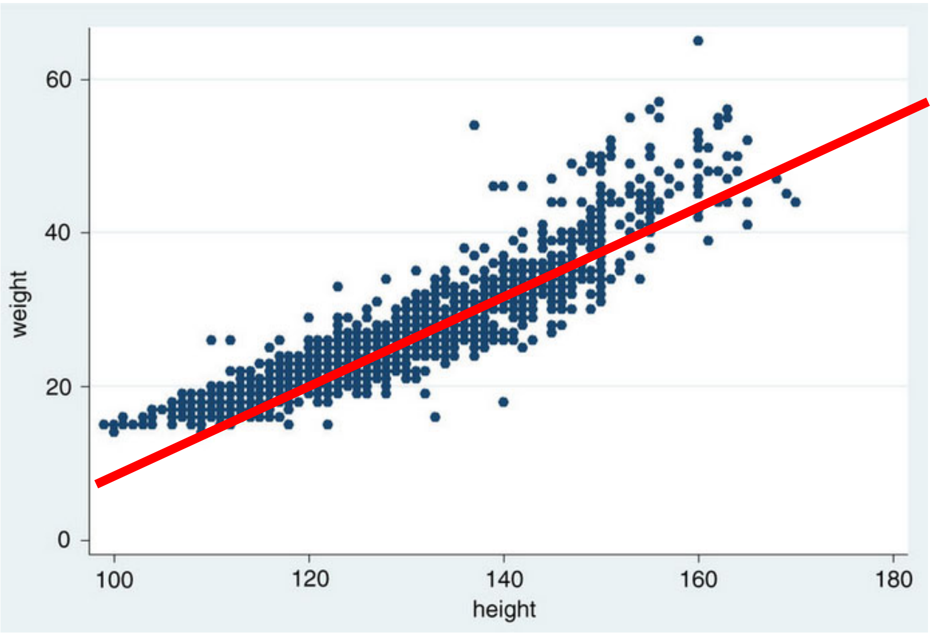
$x_2 \rightarrow t$

$t = w_1 x + w_0$



Goal : predict the target *t* given x

1,694 children surveyed in Tanzania.



$$y_{\vec{w}}(x) = w_1 x + w_0$$

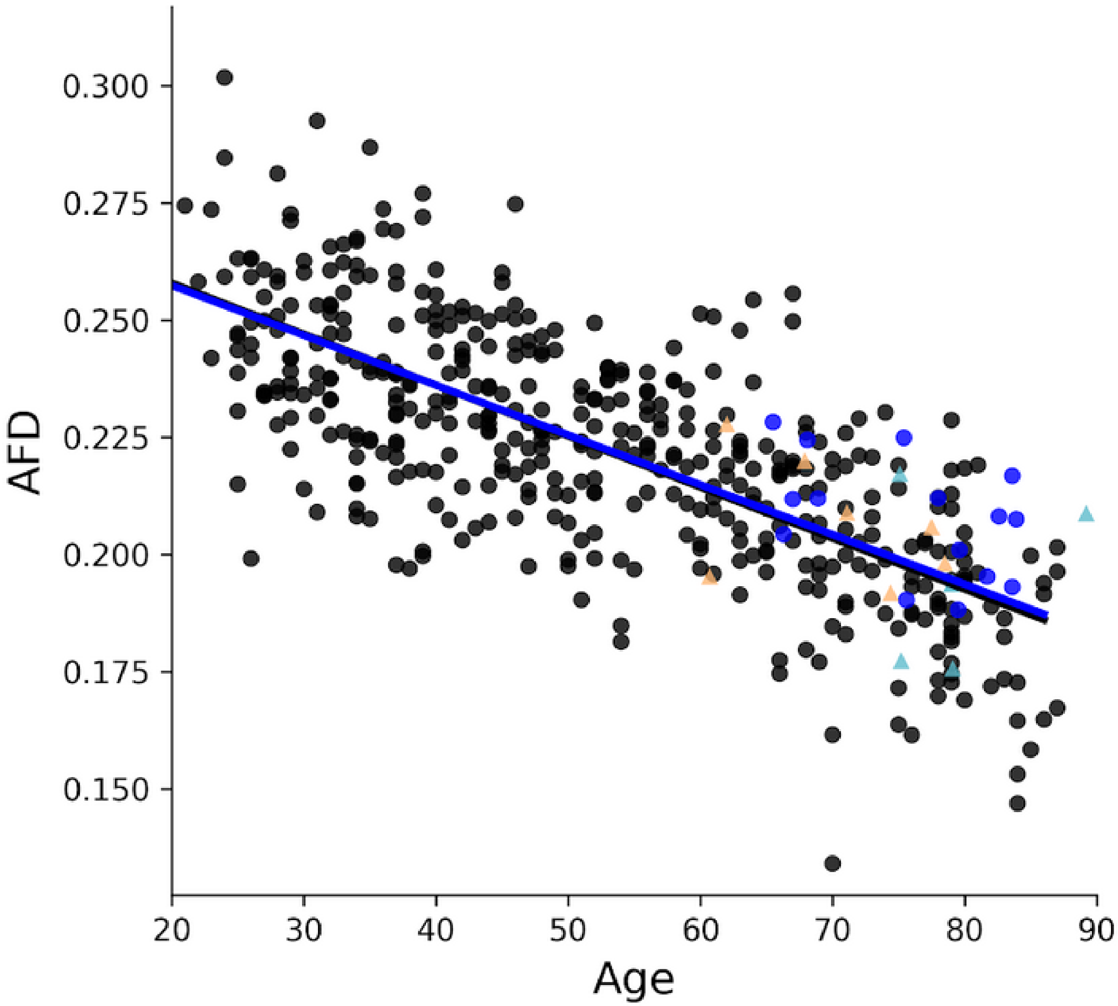
$$y_{\vec{w}}(x_i) = t_i \quad \forall i$$

Nordin P, Poggensee G, Mtweve S, Krantz I**. From a weighing scale to a pole: a comparison of two different dosage strategies in mass treatment of Schistosomiasis haematobium**. Glob Health Action. 2014

A line : 1D regression

Example

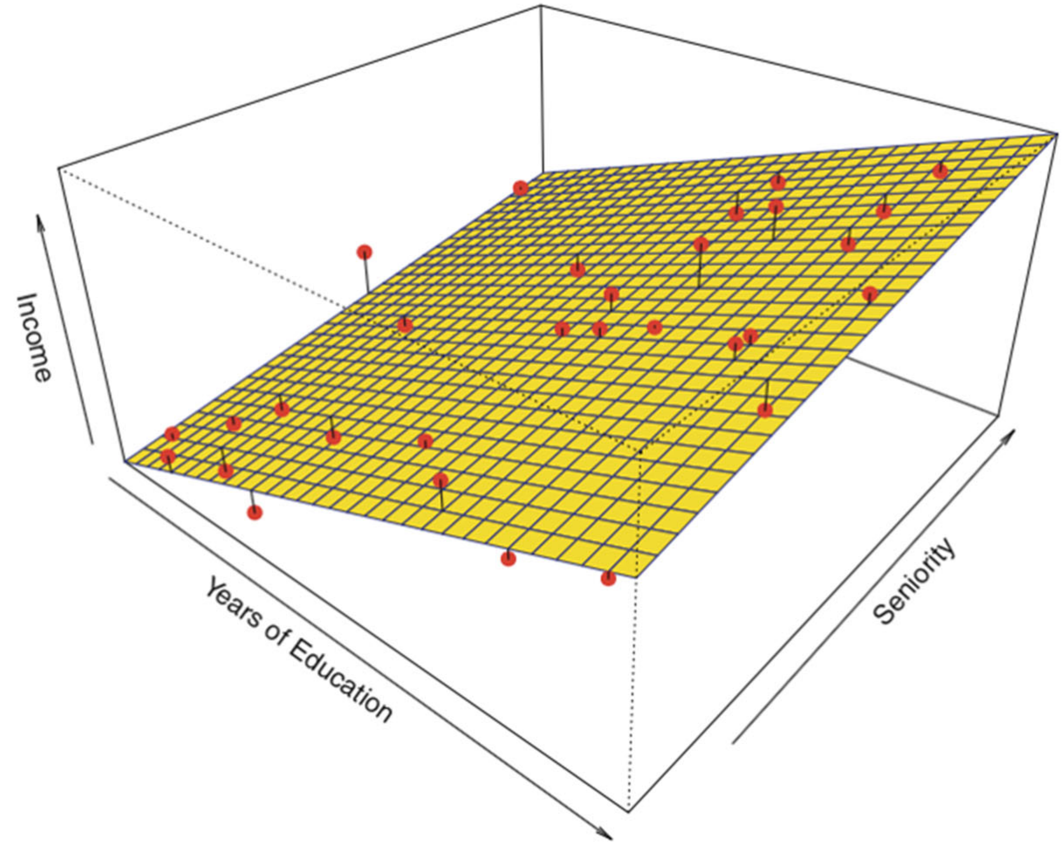
$$y_{\vec{w}}(x) = w_0 + w_1 x$$



A plane : 2D regression

Example

$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2$$



Credit : sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/R/R5_Correlation-Regression/R5_Correlation-Regression4.html

A hyper plane : dD regression

$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$$

$$= \vec{w}^{\mathrm{T}} \vec{x}$$

Dot product

Linear models

Classification

$$y_{\vec{w}}(\vec{x}) = \vec{w}^T \vec{x} = \begin{cases} > 0 & \text{if in front} \\ < 0 & \text{otherwise} \end{cases}$$

Regression

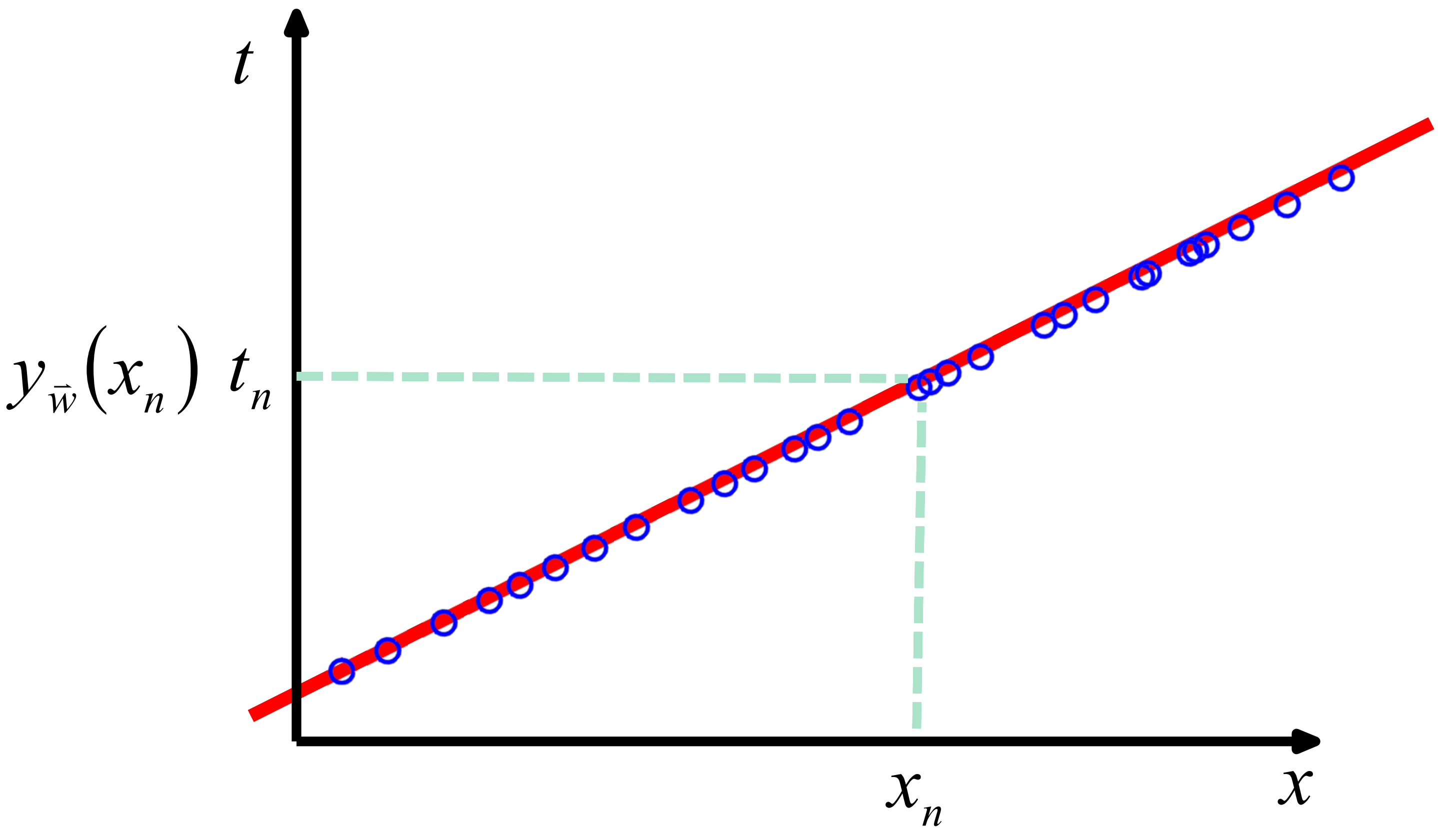
$$y_{\vec{w}}(\vec{x}) = \vec{w}^T \vec{x} = t$$

Problem to solve

Given a training example

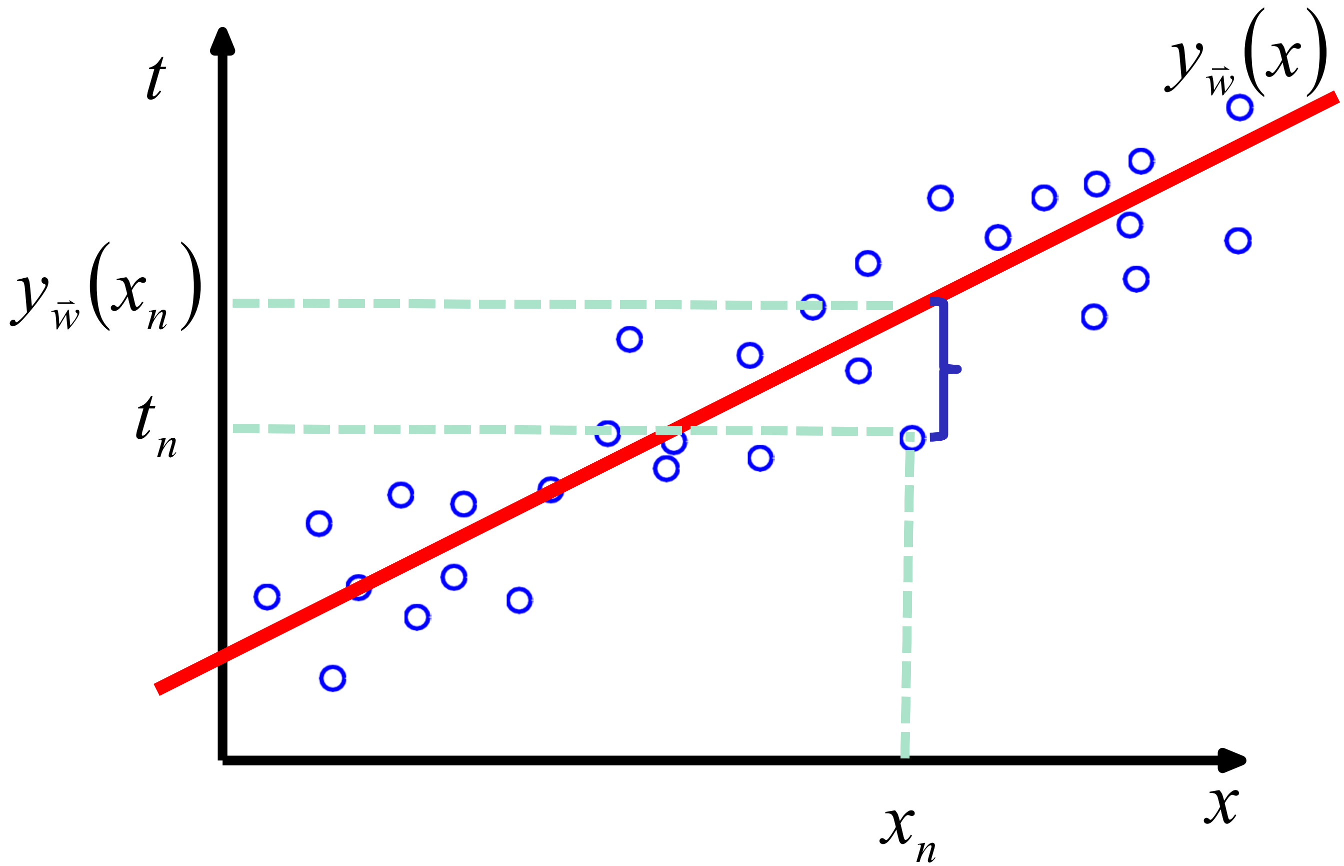
$$D = \{(x_1, t_1), (x_2, t_2), \ldots, (x_N, t_N)\}$$

Ideally, we wish $y_{\vec{w}}(x_i) = t_i$



Problem to solve

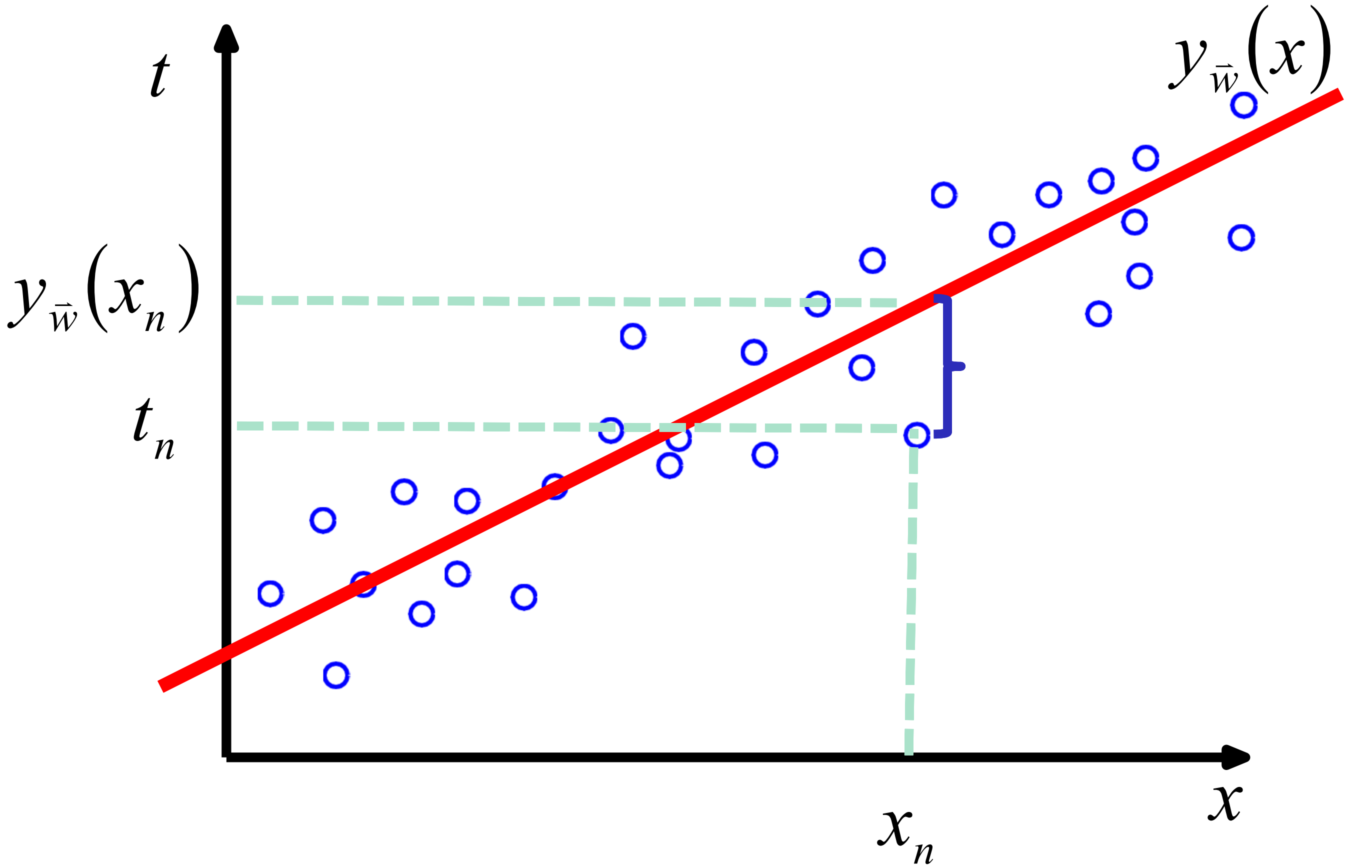
Unfortunately, real data are **noisy**



Here the goal is to make **small mistakes**.

Problem to solve

$$\vec{w} = \arg\min_{\vec{w}} \sum_{n=1}^{N} (y_{\vec{w}}(x_n) - t_n)^2$$



Problem to solve

$$\vec{w} = \arg\min_{\vec{w}} \sum_{n=1}^{N} \left(\vec{w}^{\mathrm{T}} \vec{x}_n - t_n\right)^2$$

If the **data is linear** + the **noise is Gaussian**, the best possible weights are those **minimizing this function**

Problem to solve

$$\boxed{\vec{w} = \arg\min_{\vec{w}} \sum_{n=1}^{N} \left(\vec{w}^{\mathrm{T}} \vec{x}_n - t_n\right)^2}$$

$$\underbrace{\sum_{n=1}^{N} \left(\vec{w}^{\mathrm{T}} \vec{x}_n - t_n\right)^2}_{\mathcal{L}_D(\vec{w})}$$

the « best » $\vec{w}$ is the one for which the gradient is **zero**

$$\nabla_{\vec{w}} \mathcal{L}_D(\vec{w}) = \sum_{n=1}^{N} 2\left(\vec{w}^{\mathrm{T}} \vec{x}_n - t_n\right) \vec{x}_n^{\mathrm{T}} = 0$$

$$\vec{w}^{\mathrm{T}} \sum_{n=1}^{N} \vec{x}_n \vec{x}_n^{\mathrm{T}} - \sum_{n=1}^{N} t_n \vec{x}_n^{\mathrm{T}} = 0$$

Problem to solve

$$\vec{w}^{\mathrm{T}} \sum_{n=1}^{N} \vec{x}_n \vec{x}_n^{\mathrm{T}} - \sum_{n=1}^{N} t_n \vec{x}_n^{\mathrm{T}} = 0$$

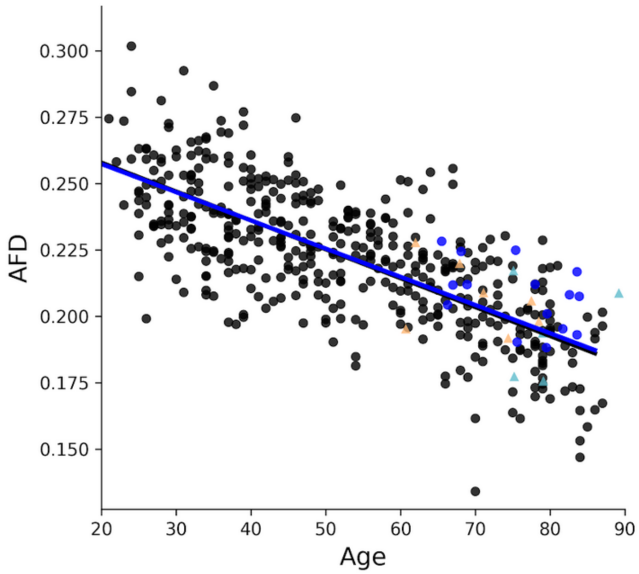
By **isolating** $\vec{w}$, we get

$$\boxed{\vec{w} = \left(X^{\mathrm{T}} X\right)^{-1} X^{\mathrm{T}} T}$$

where

$$X = \begin{pmatrix} 1 & x_{1,1} & \cdots & x_{1,d} \\ 1 & x_{2,1} & \cdots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N,1} & \cdots & x_{N,d} \end{pmatrix} \qquad T = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{pmatrix}$$

For a 1D regression



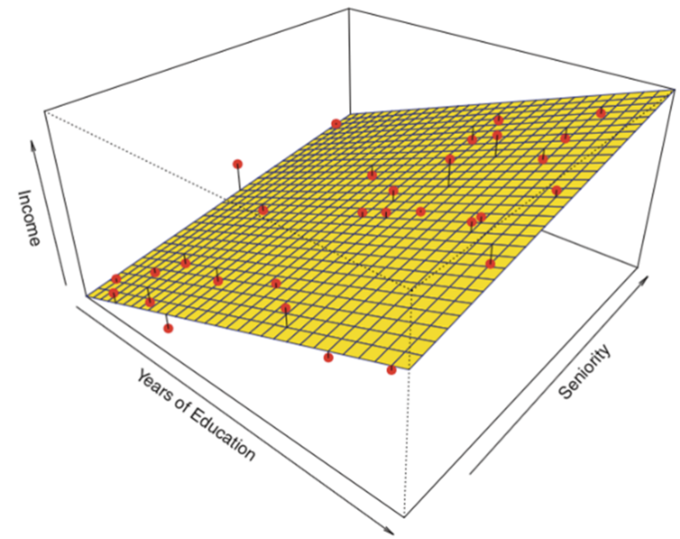
$$y_{\vec{w}}(x) = w_0 + w_1 x$$

$$\boxed{\vec{w} = \left(X^{\mathrm{T}} X\right)^{-1} X^{\mathrm{T}} T}$$

where

$$X = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{pmatrix} \; T = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{pmatrix}$$

For a 2D regression



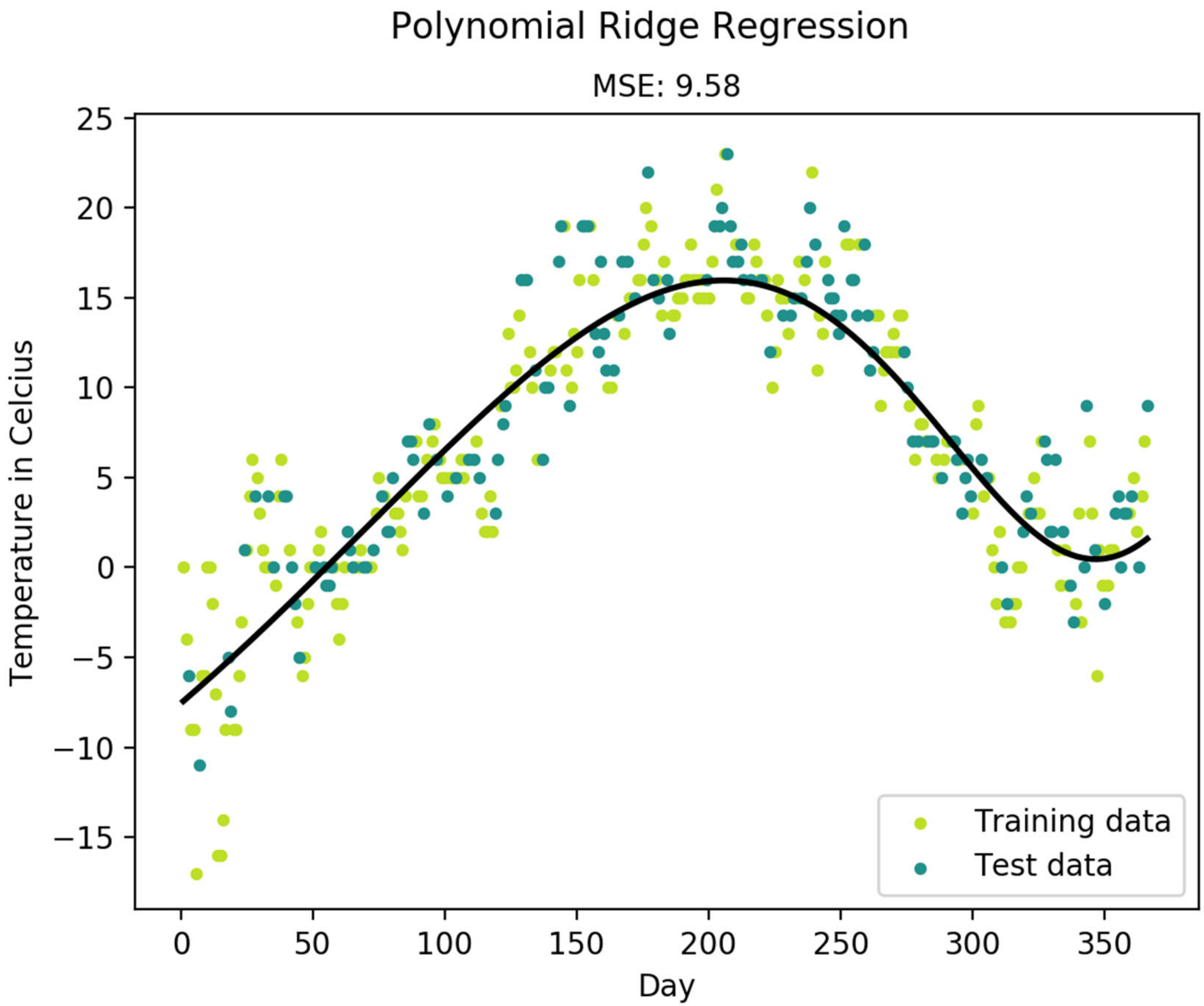
$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2$$

$$\boxed{\vec{w} = \left(X^{\mathrm{T}} X\right)^{-1} X^{\mathrm{T}} T}$$

where

$$X = \begin{pmatrix} 1 & x_{1,1} & x_{1,2} \\ 1 & x_{2,1} & x_{2,2} \\ \vdots & \vdots & \vdots \\ 1 & x_{N,1} & x_{N,2} \end{pmatrix} \quad T = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{pmatrix}$$

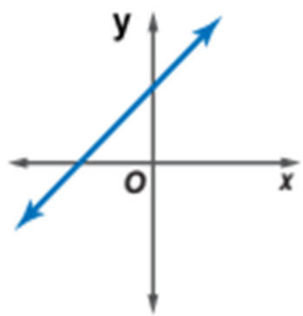
What about non-linear data?



What about non-linear data?

$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x$

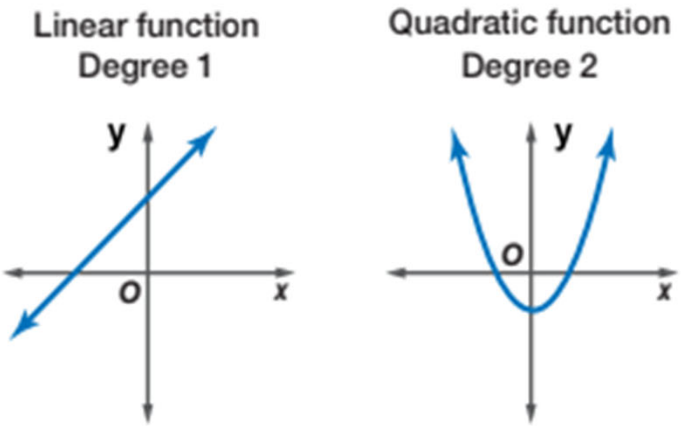
Linear function
Degree 1



http://www.math.glencoe.com/

What about non-linear data?

$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x$ $\qquad$ $y_{\vec{w}}(\vec{x}) = w_0 + w_1 x + w_1 x^2$



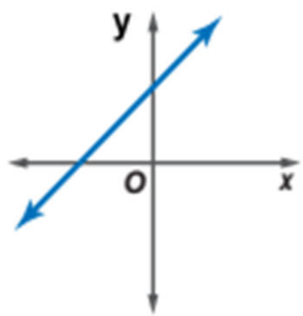
http://www.math.glencoe.com/

What about non-linear data?

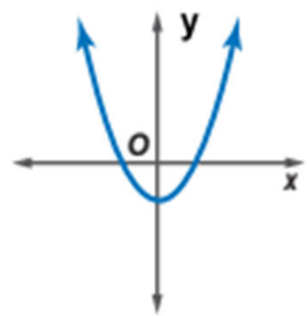
$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x$$

$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x + w_1 x^2$$

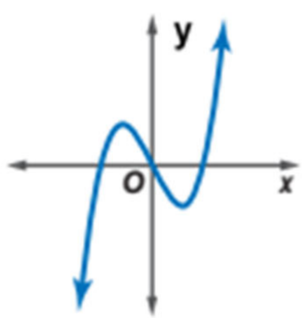
Linear function
Degree 1



Quadratic function
Degree 2



Cubic function
Degree 3



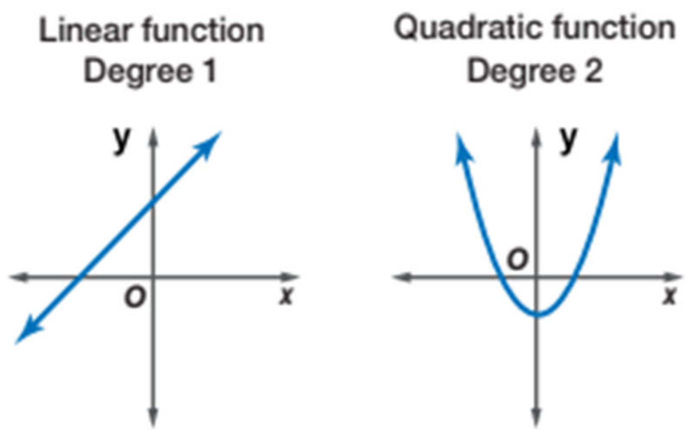
http://www.math.glencoe.com/

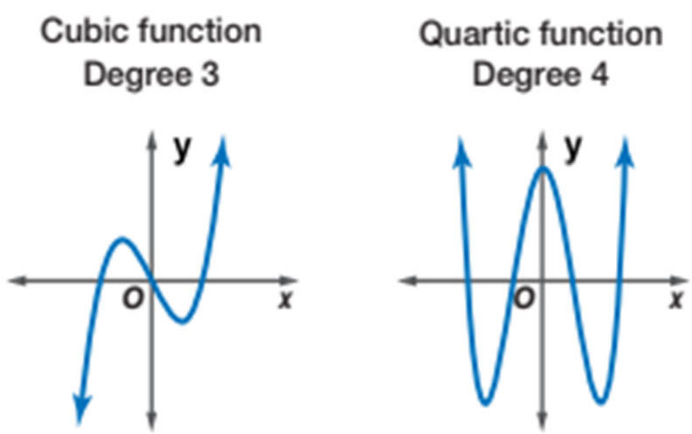
$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x + w_1 x^2 + w_1 x^3$$

What about non-linear data?

$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x$

$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x + w_1 x^2$





$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x + w_1 x^2 + w_1 x^3 + w_1 x^4$

http://www.math.glencoe.com/

$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x + w_1 x^2 + w_1 x^3$

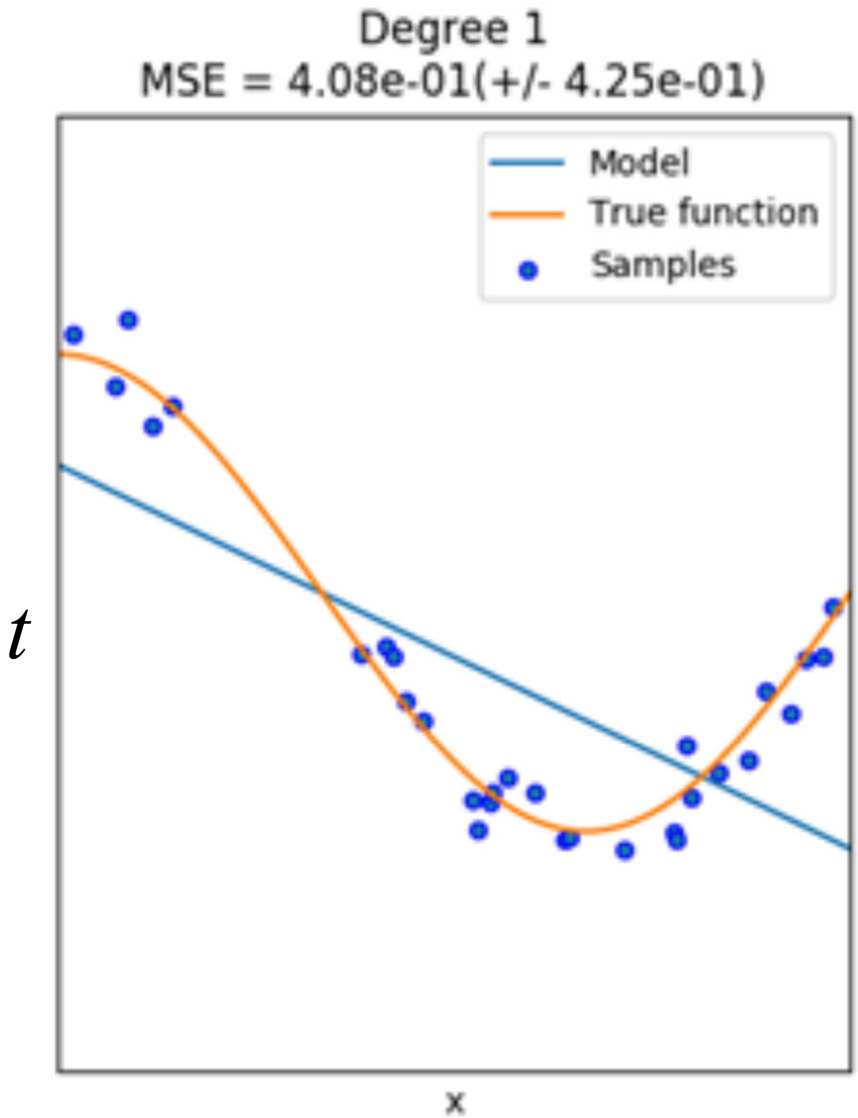
Basis function

Example: Instead of a **1D regression**, lets do a **4D regression**

$$\varphi(x) \to (x, x^2, x^3, x^4)$$

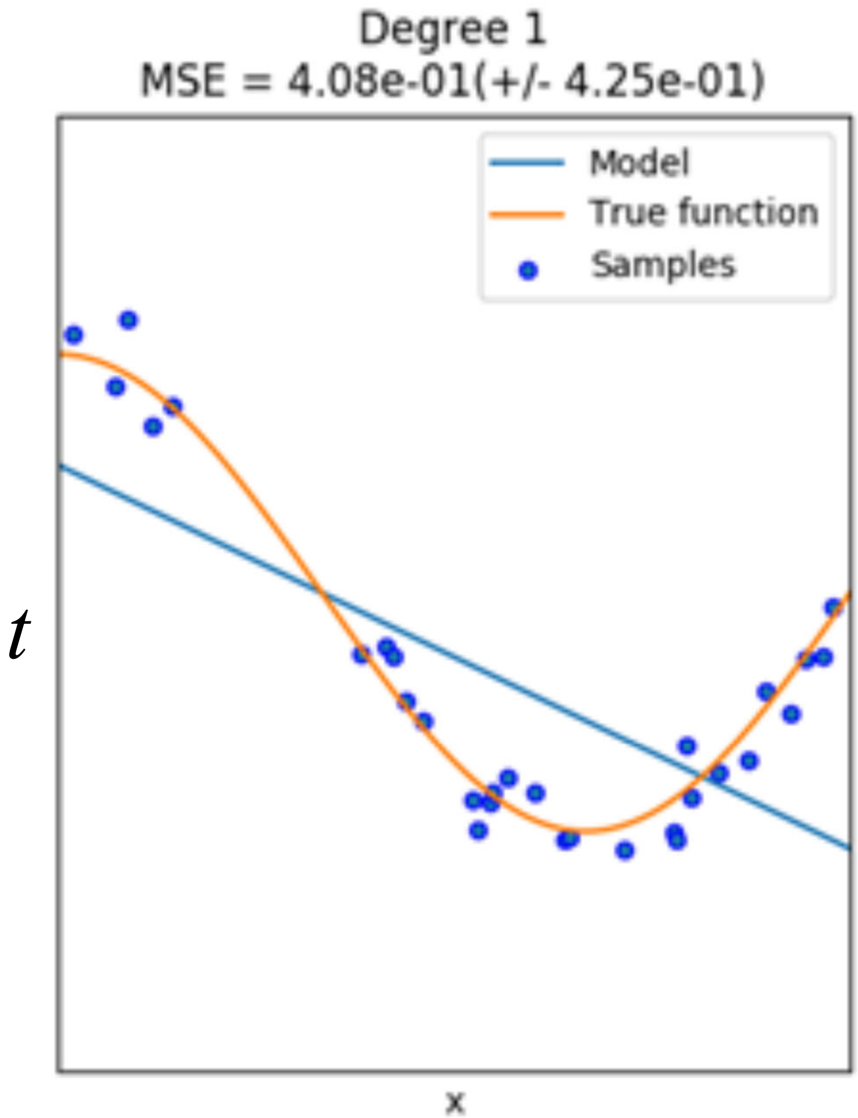
$$y_{\vec{w}}(x) = w_0 + w_1 x$$

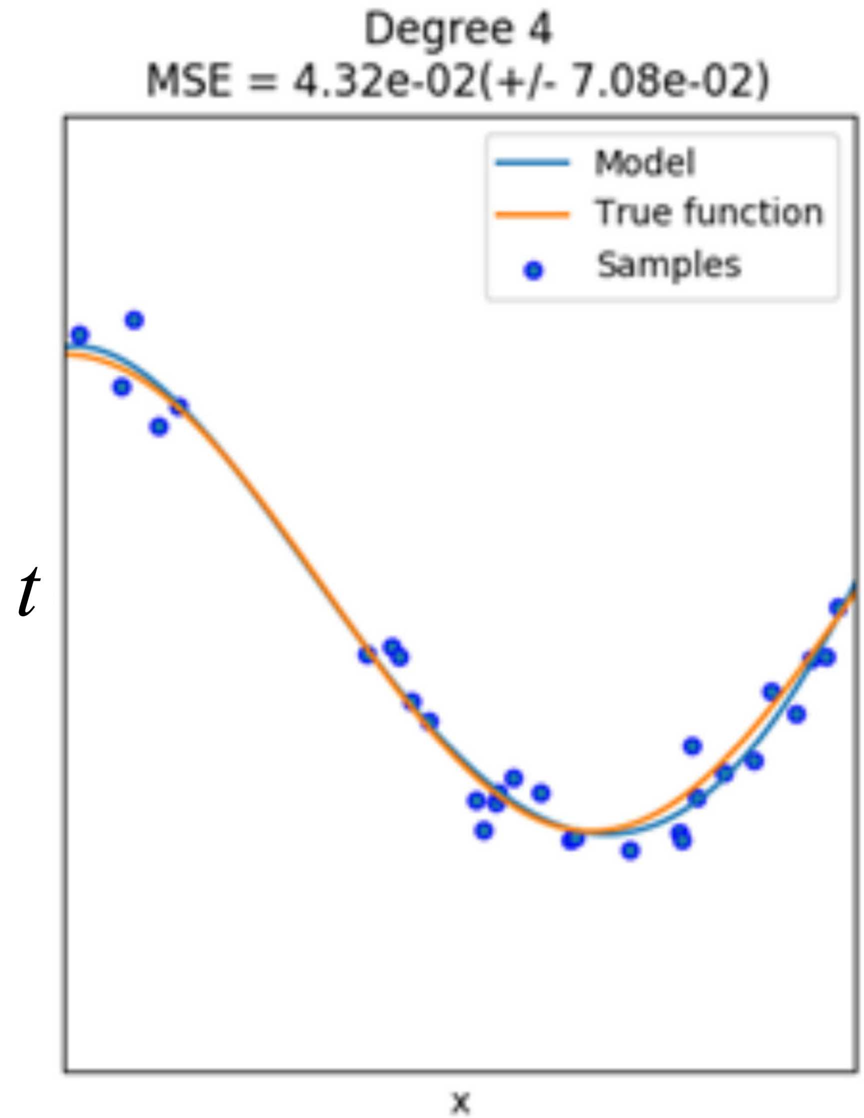
$$\begin{aligned} y_{\vec{w}}(x) &= w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 \\ &= w_0 + \sum_{i=1}^{4} w_i \varphi_i(x) \end{aligned}$$



$$y_{\vec{w}}(x) = w_0 + w_1 x$$

Source : stats.stackexchange.com/questions/305619/artificial-neural-networks-equivalent-to-linear-regression-with-polynomial-featu





$$y_{\vec{w}}(x) = w_0 + w_1 x$$

$$y_{\vec{w}}(x) = w_0 + \sum_{i=1}^{4} w_i \phi_i(x)$$

Source : stats.stackexchange.com/questions/305619/artificial-neural-networks-equivalent-to-linear-regression-with-polynomial-featu

Basis function

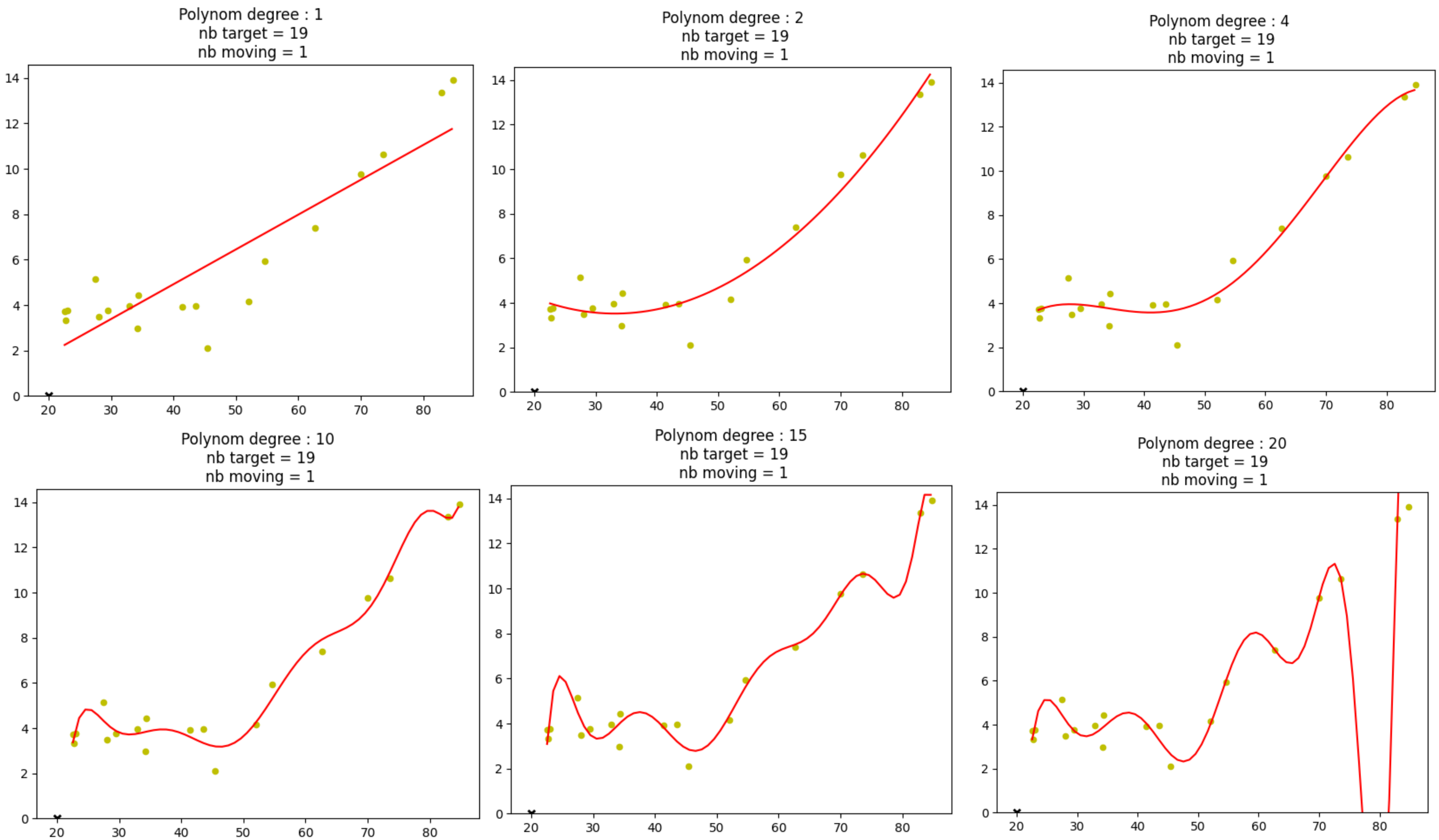
parameter

$$y_{\vec{w}}(\vec{x}) = \sum_{i=0}^{d-1} w_i \varphi_i(\vec{x})$$

Basis function

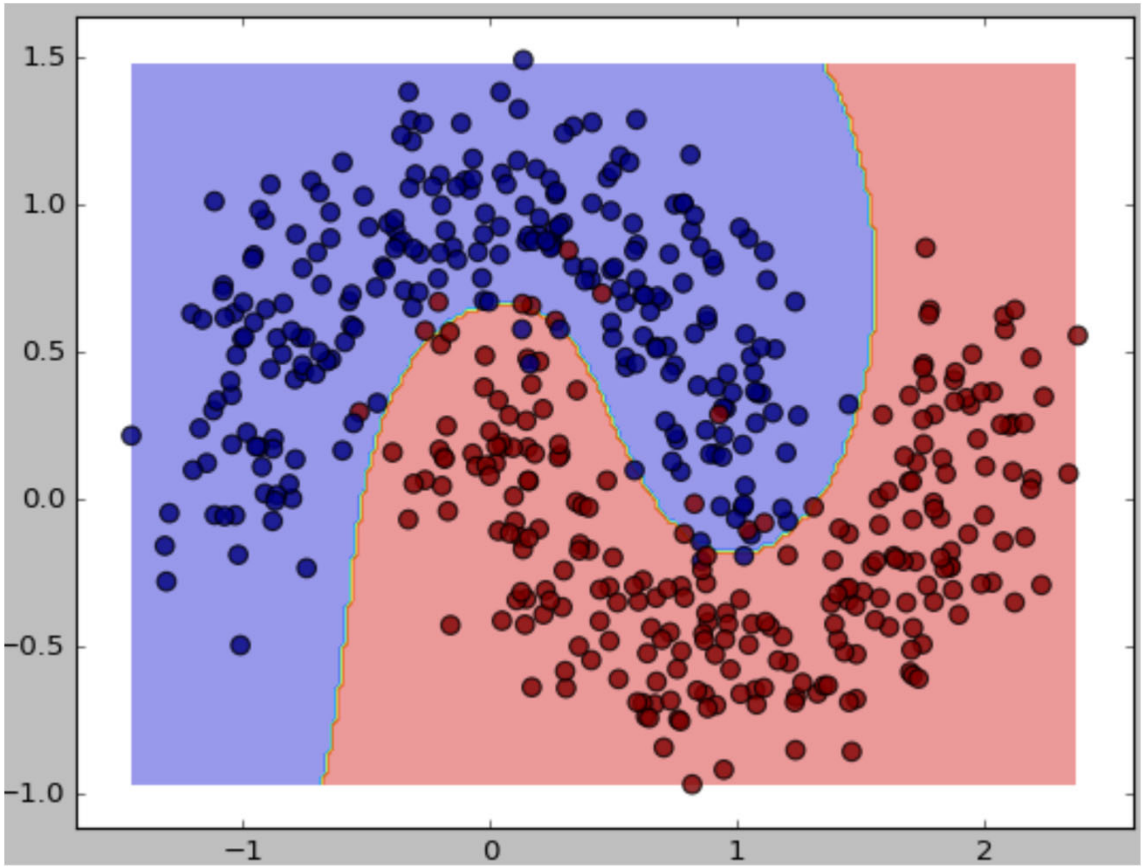
where $\varphi_0(\vec{x}) = 1$

Regression



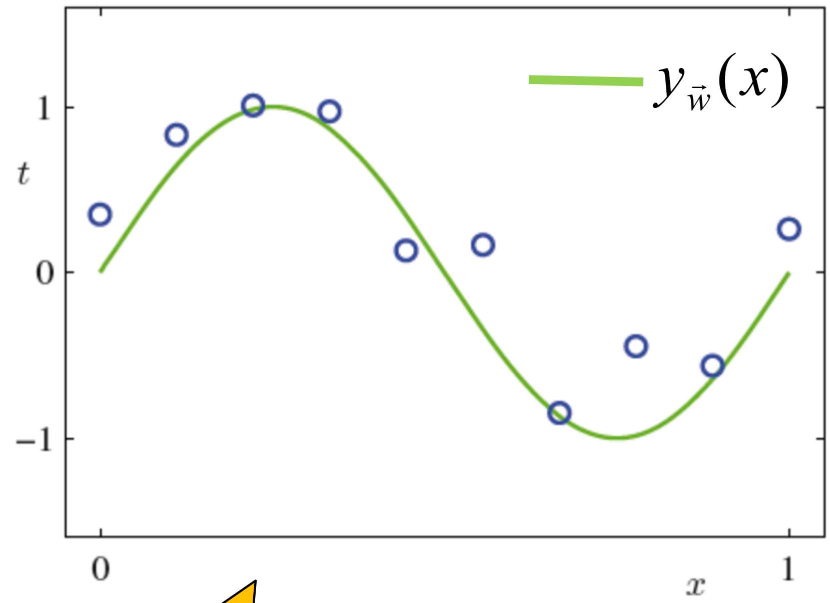
Polynom degree : 1
nb target = 19
nb moving = 1
Polynom degree : 2
nb target = 19
nb moving = 1
Polynom degree : 4
nb target = 19
nb moving = 1
Polynom degree : 10
nb target = 19
nb moving = 1
Polynom degree : 15
nb target = 19
nb moving = 1
Polynom degree : 20
nb target = 19
nb moving = 1

Similar approach for classification



Unknowns

$$y_{\vec{w}}(x) = w_0 + w_1 x + w_2 x^2 + \cdots + w_d x^d$$
$$= \sum_{i=0}^{M} w_i x^i$$



Two unknowns

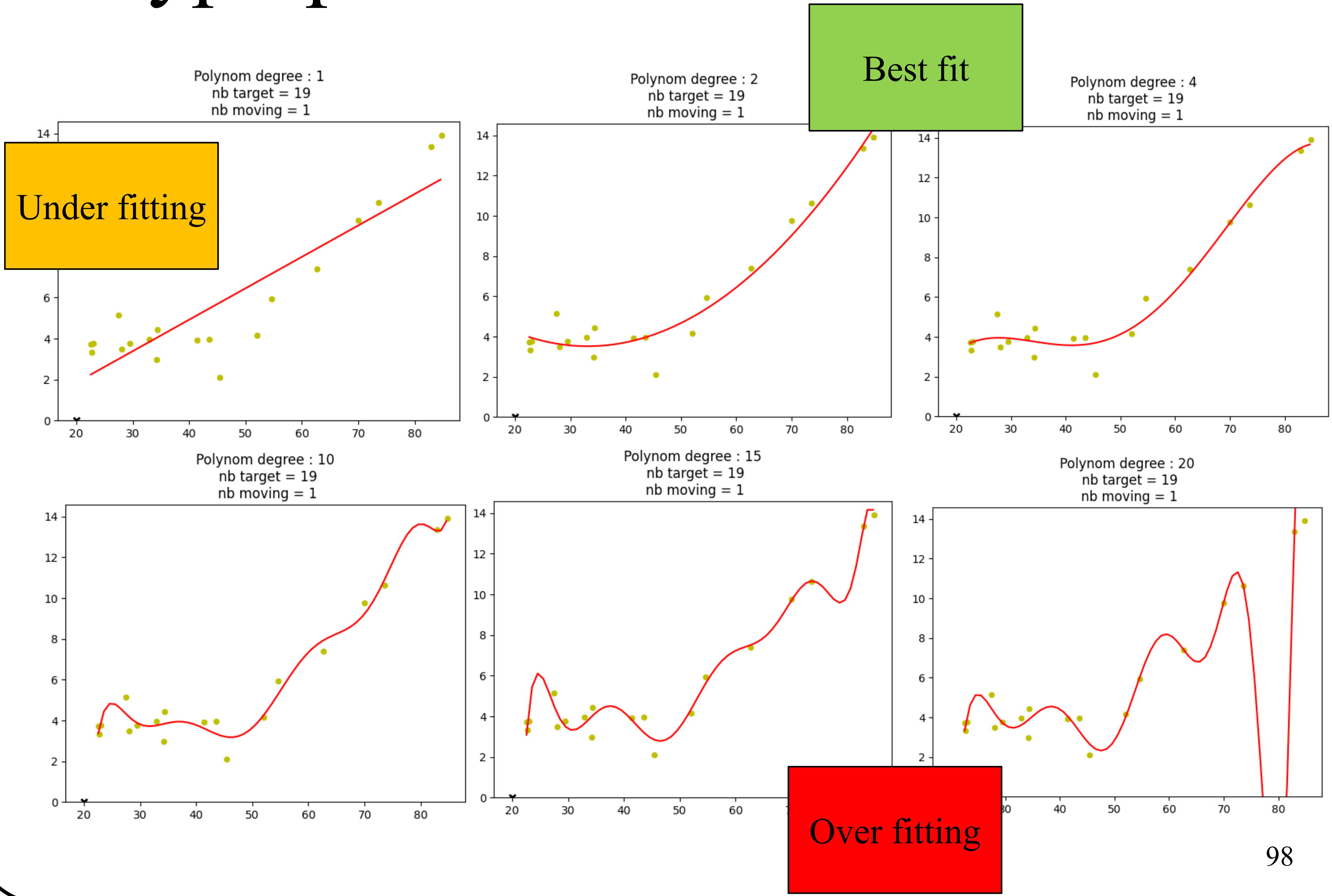
$$\vec{w} \in R^d$$

Parameters

$$d \in N^{\geq 0}$$

Hyper-parameters

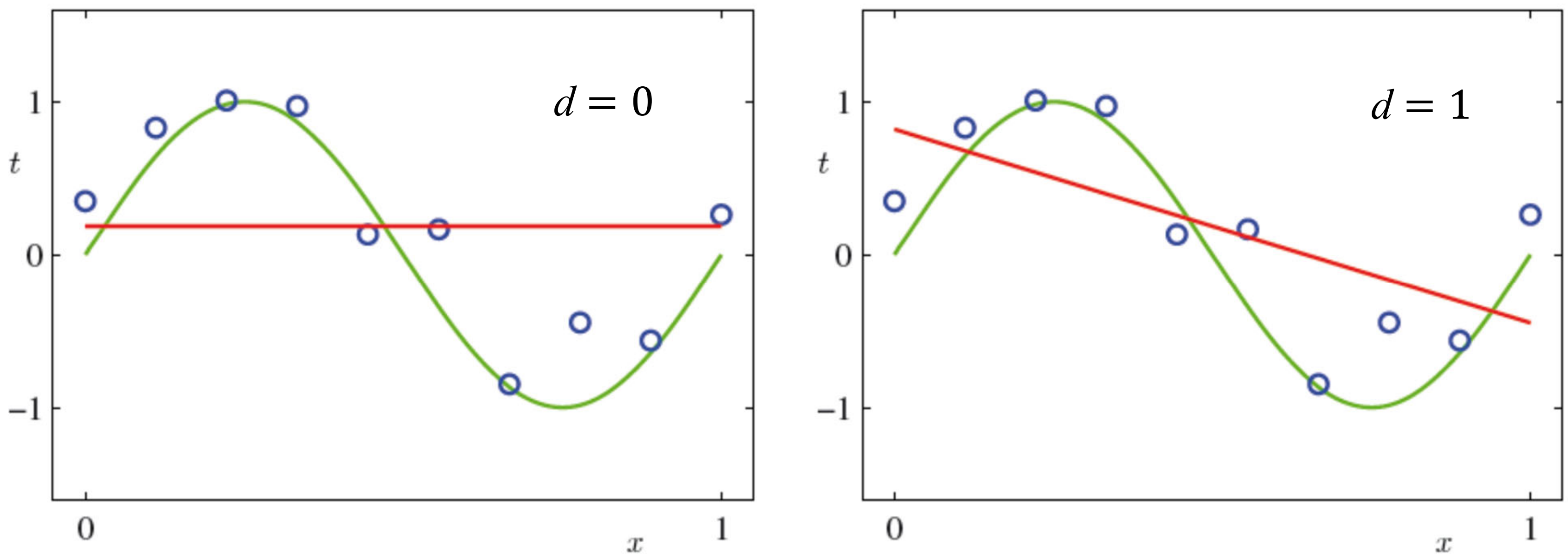
Hyperparameter *d*



Underfitting

$d = 0 \quad => \quad y_{\vec{w}}(x) = w_0$
$d = 1 \quad => \quad y_{\vec{w}}(x) = w_0 + w_1 x$

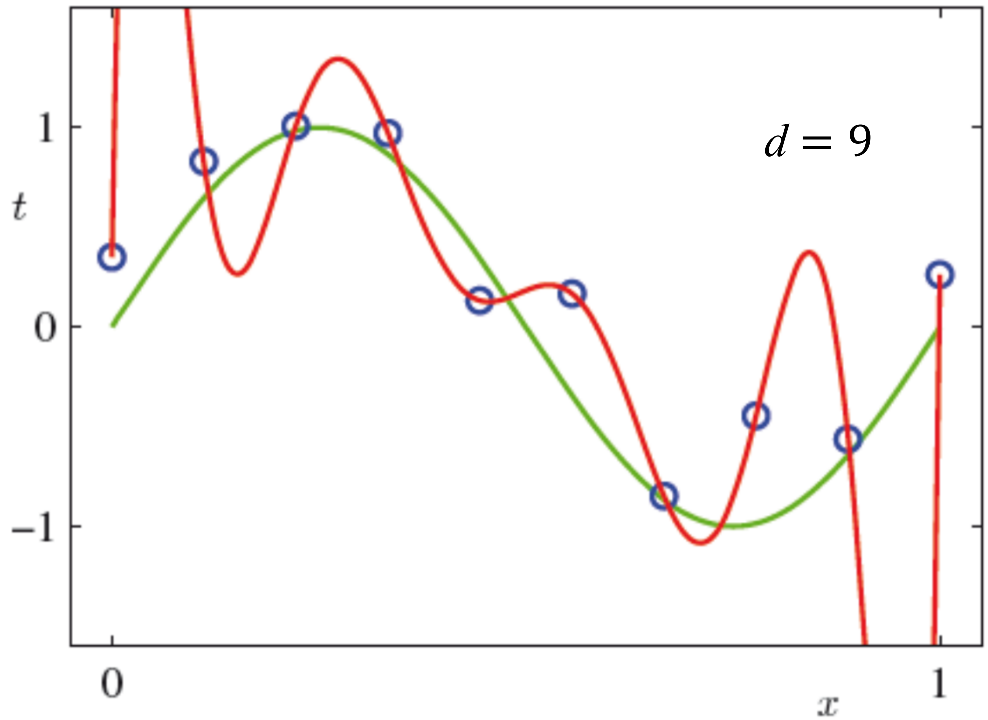
A small *d* gives a simplistic model that **underfits** the data.



$$\mathcal{L}_D(\vec{w}) \Rightarrow \text{large}$$
$$\mathcal{L}_{D_{test}}(\vec{w}) \Rightarrow \text{large}$$

Overfitting

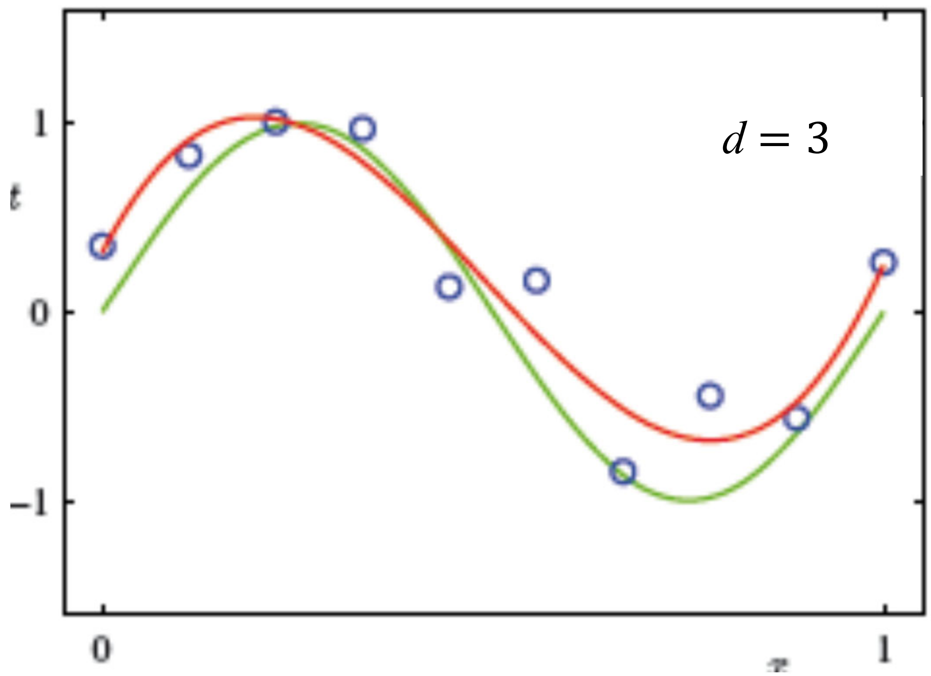
A large d gives a model that « **learn by heart** » and thus **overfits** training data



$$\mathcal{L}_D(\vec{w}) \Rightarrow \text{VERY low}$$
$$\mathcal{L}_{D_{test}}(\vec{w}) \Rightarrow \text{large}$$

Over- and underfitting

Need for an **intermediate value** for which the training and the testing errors are low

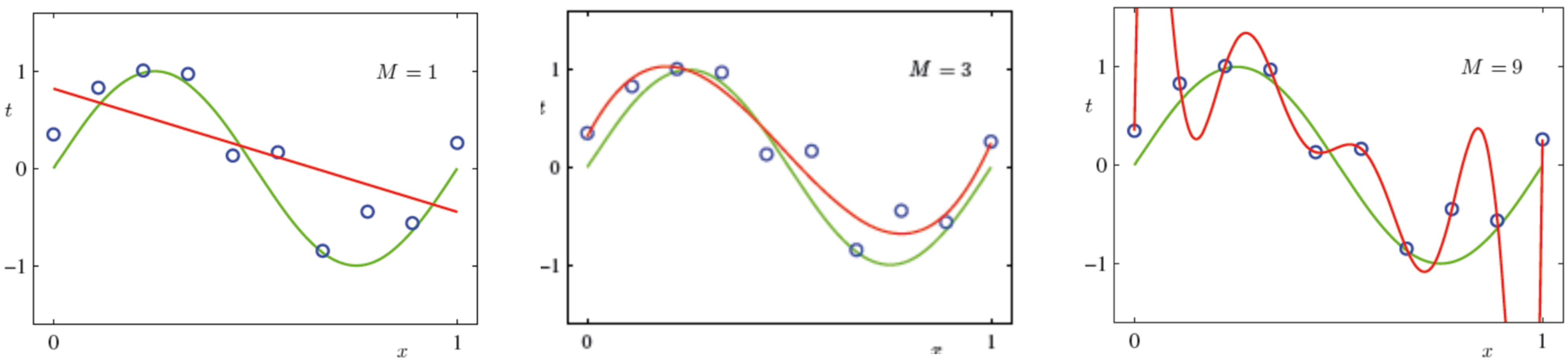


$$\mathcal{L}_D(\vec{w}) \Rightarrow \text{ low}$$
$$\mathcal{L}_{D_{test}}(\vec{w}) \Rightarrow \text{low}$$

Hyperparameters often control the **capacity** of a model

Capacity: ability of a model to fit the training data

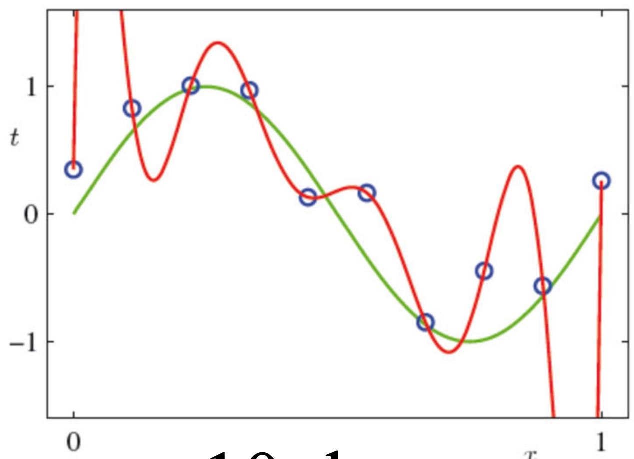
Low capacity | Medium capacity | Large capacity



Generalization

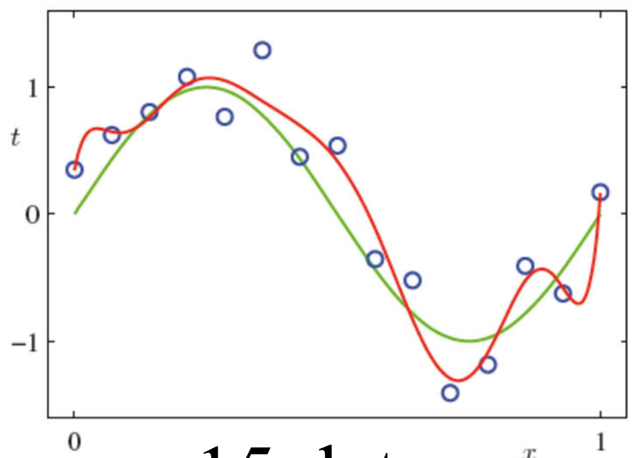
The more data you have, the better a high capacity model will generalize.

$d = 9$



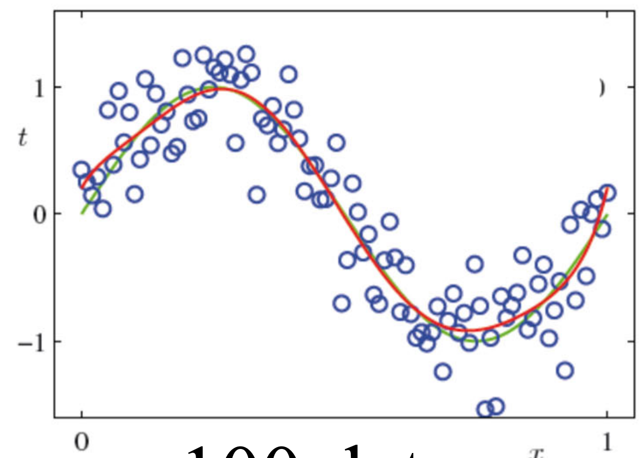
10 data

$d = 9$



15 data

$d = 9$



100 data

How do we prevent our model from under- and overfitting?

Regularization

Parameter values $\vec{w}$ for different d **without** regularization

	$d=0$	$d=1$	$d=3$	$d=9$
w_0	0.19	0.82	0.31	0.35
w_1		-1.27	7.99	232.37
w_2			-25.43	-5321.83
w_3			17.37	48568.31
w_4				-231639.30
w_5				640042.26
w_6				-1061800.52
w_7				1042400.18
w_8				-557682.99
w_9				125201.43

Regularization

To prevent over-fitting

1. Choose a small « d »

2. Reduce capacity by **regularization**

Exemple : penalyse the **L2 norm**

Constant that controls regularization

$$E_D(\vec{w}) = \frac{1}{N}\sum_{n=1}^{N}\left(t_n - y_{\vec{w}}(\vec{x})\right)^2 + \lambda||\vec{w}||^2$$

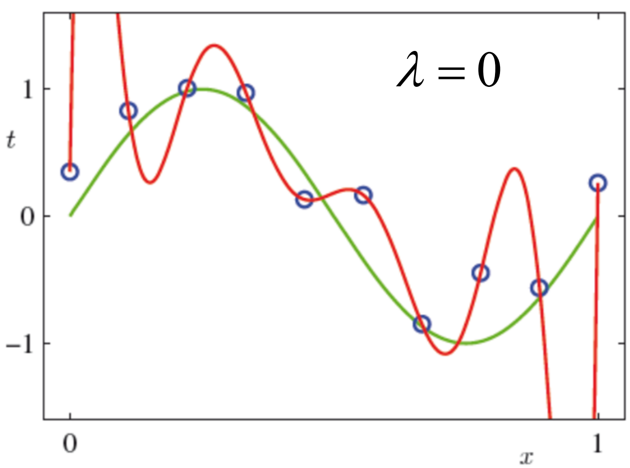
$$||\vec{w}||^2 = \vec{w}^{\mathrm{T}}\vec{w} = w_0^2 + w_1^2 + \ldots + w_d^2$$

Ridge model

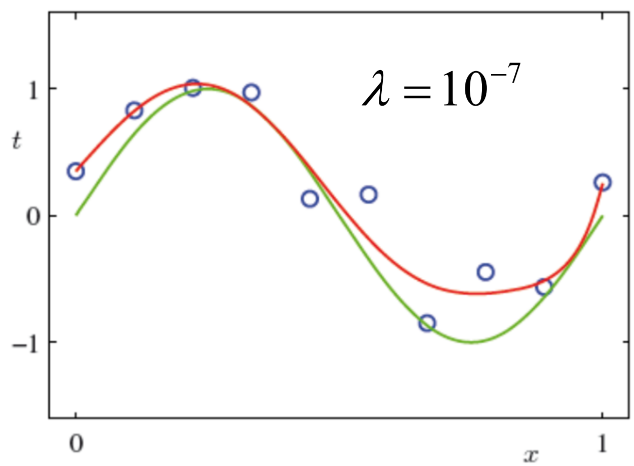
Regularization

Strong regularization= less capacity

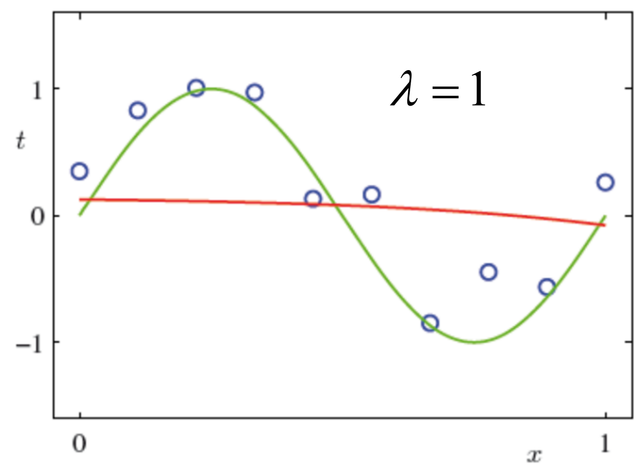
$d = 9$



$d = 9$

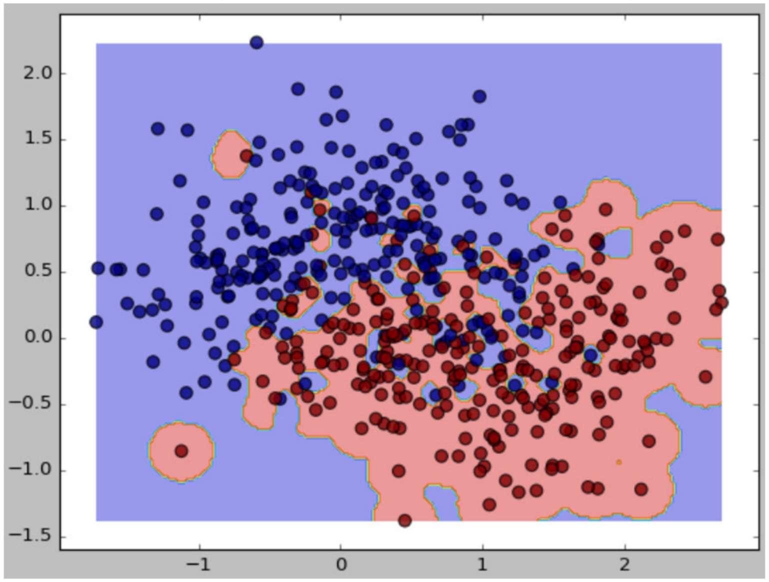


$d = 9$

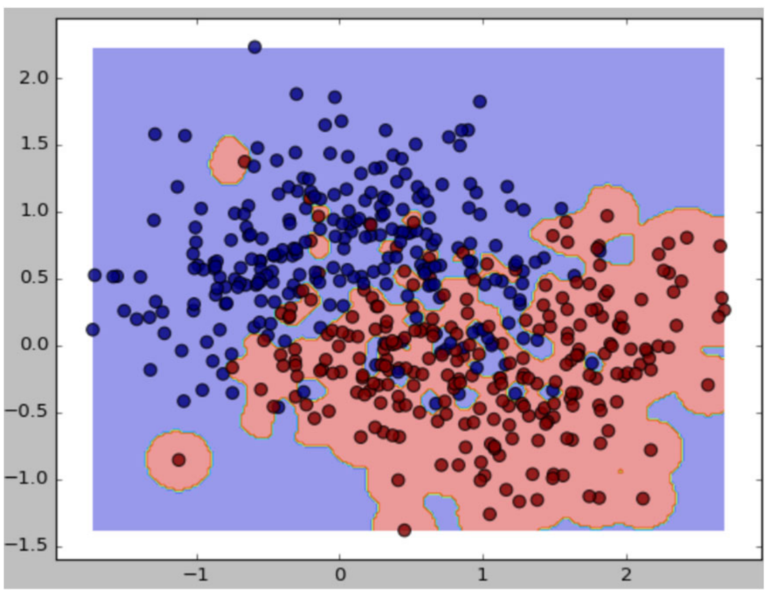


Over- and under-fitting also influence classification

Overfitting (Classification)

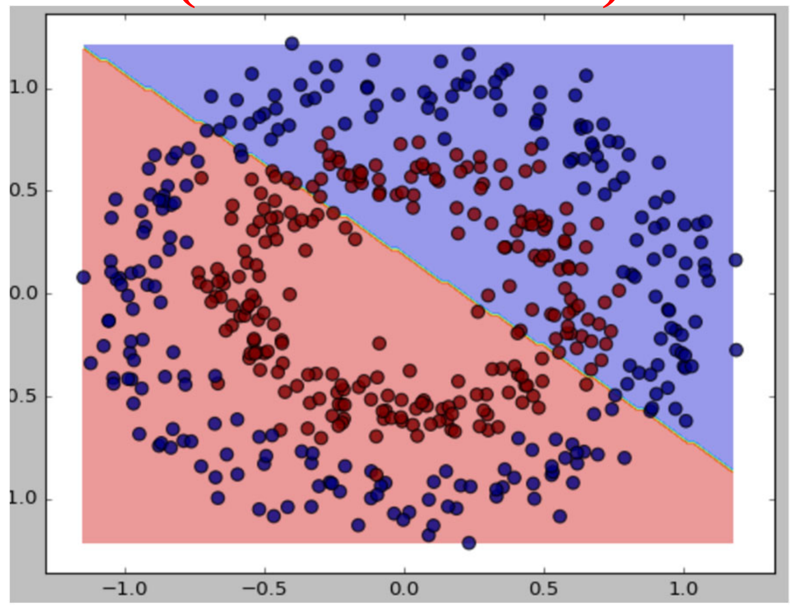


Training accuracy = 99.6%

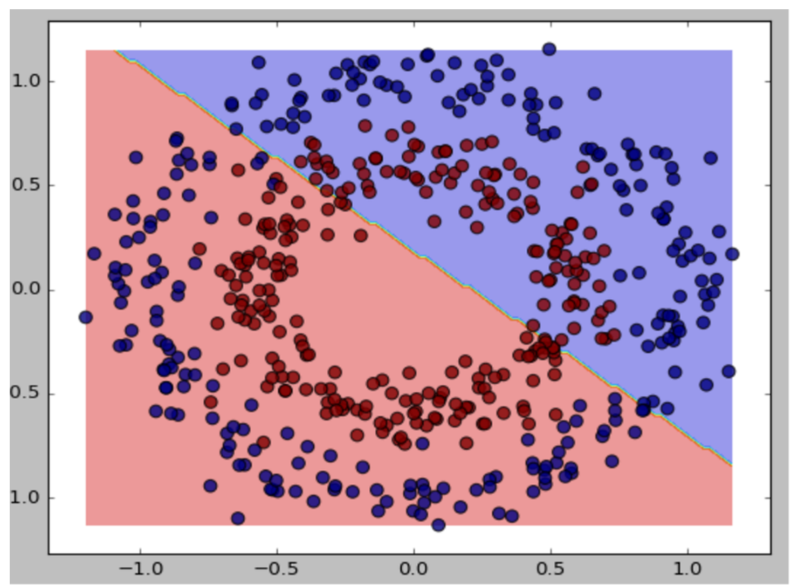


Testing accuracy = 78%

Underfitting (Classification)



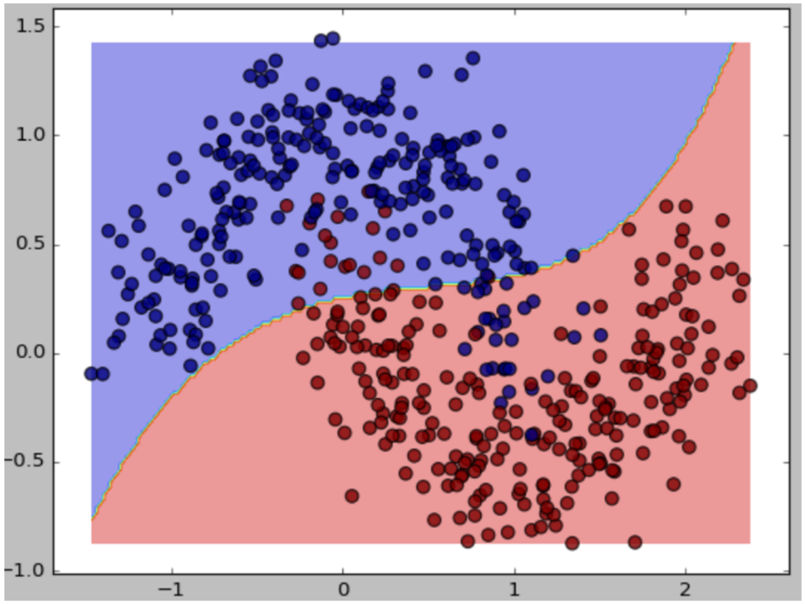
Training accuracy =52.2%

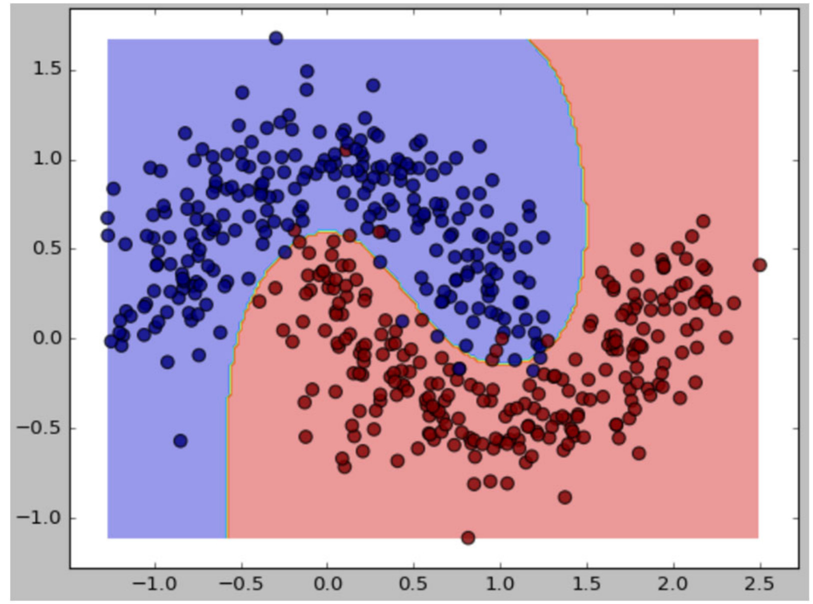


Testing accuracy = 51.2%

Could be better…

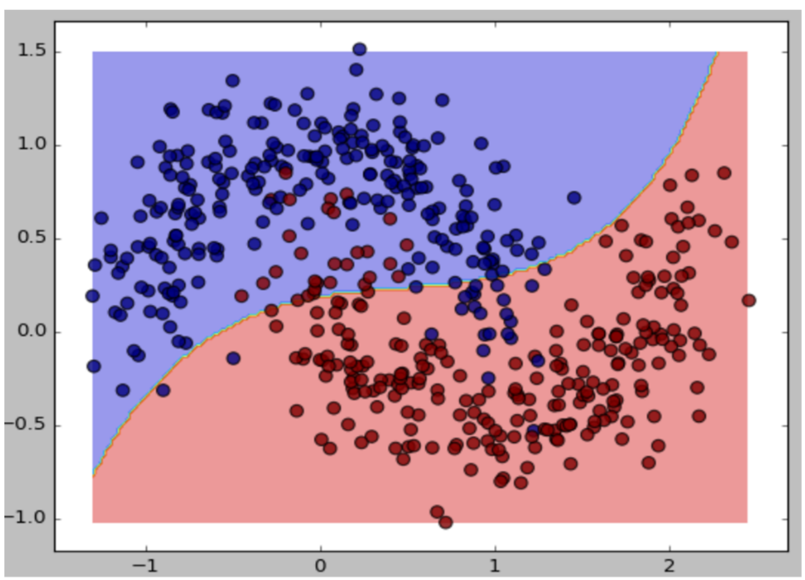
Wonderful !!!

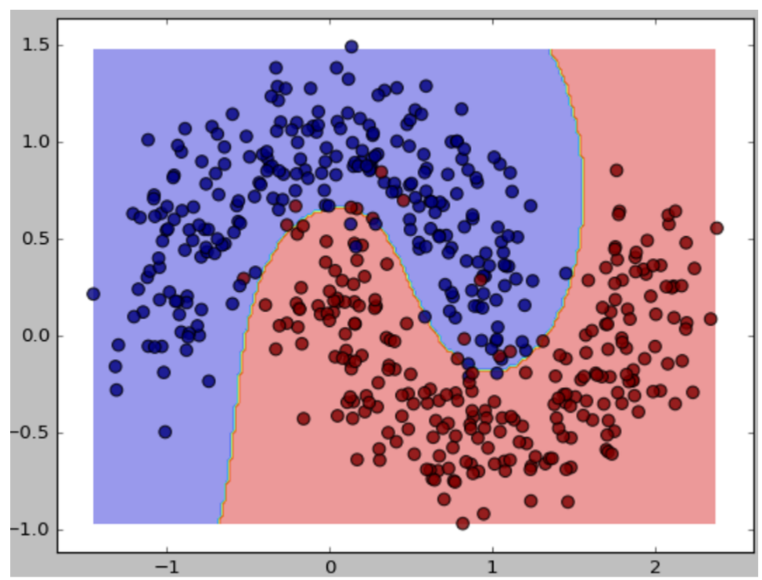




Training accuracy =82%

Training accuracy =97.8%





Testing accuracy = 80%

Testing accuracy = 96.2%

$$E_D(\vec{w}) = \frac{1}{N}\sum_{n=1}^{N}(y_{\vec{w}}(x_n) - t_n)^2 + \lambda\|\vec{w}\|^2$$

$$\|\vec{w}\|^2 = \vec{w}^T\vec{w} = w_0^2 + w_1^2 + \ldots + w_d^2$$

Model selection

How to find the right hyper-parameters?

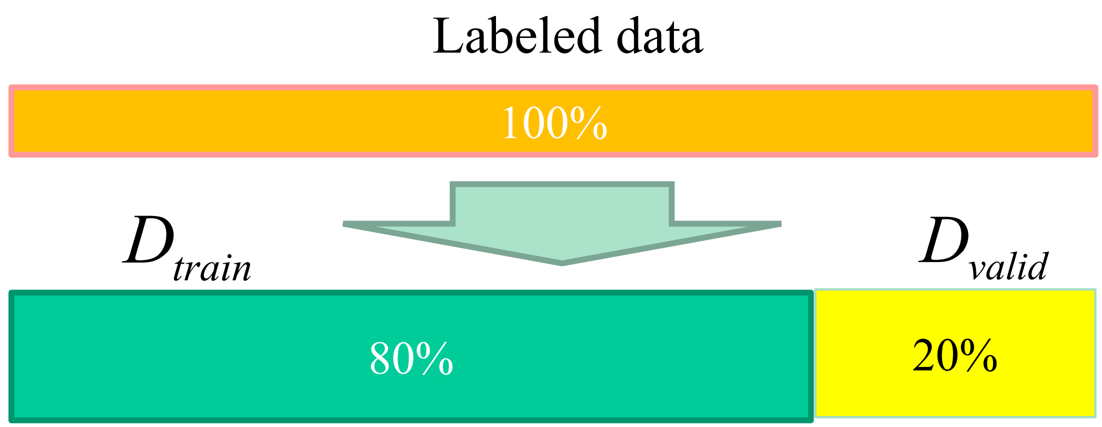
d and λ

How to find the right d and the right λ ?

- **Very bad idea** : choose randomly
- **Bad idea** : take many (d, λ) and keep the one with the lowest training error
  - ➢ overfitting
- **Bad idea** : take many (d, λ) and keep the one with the lowest testing error
  - ➢ D_{test} should NEVER be used to train a model
- **<u>Good solution</u>** : take many (d, λ) and keep the one with the lowest **<u>validation error</u>**

Cross-validation

1- Randomly devide data in 2 groups



2- FOR M from $M_{\min}$ to $M_{\max}$

FPR λ from $\lambda_{\min}$ to $\lambda_{\max}$

Train the model on D_{train}

Compute error on D_{valid}

3- Keep (M,λ) with the lowest <u>**validation error**</u>

K-fold cross-validation with K = 10

Mean validation error

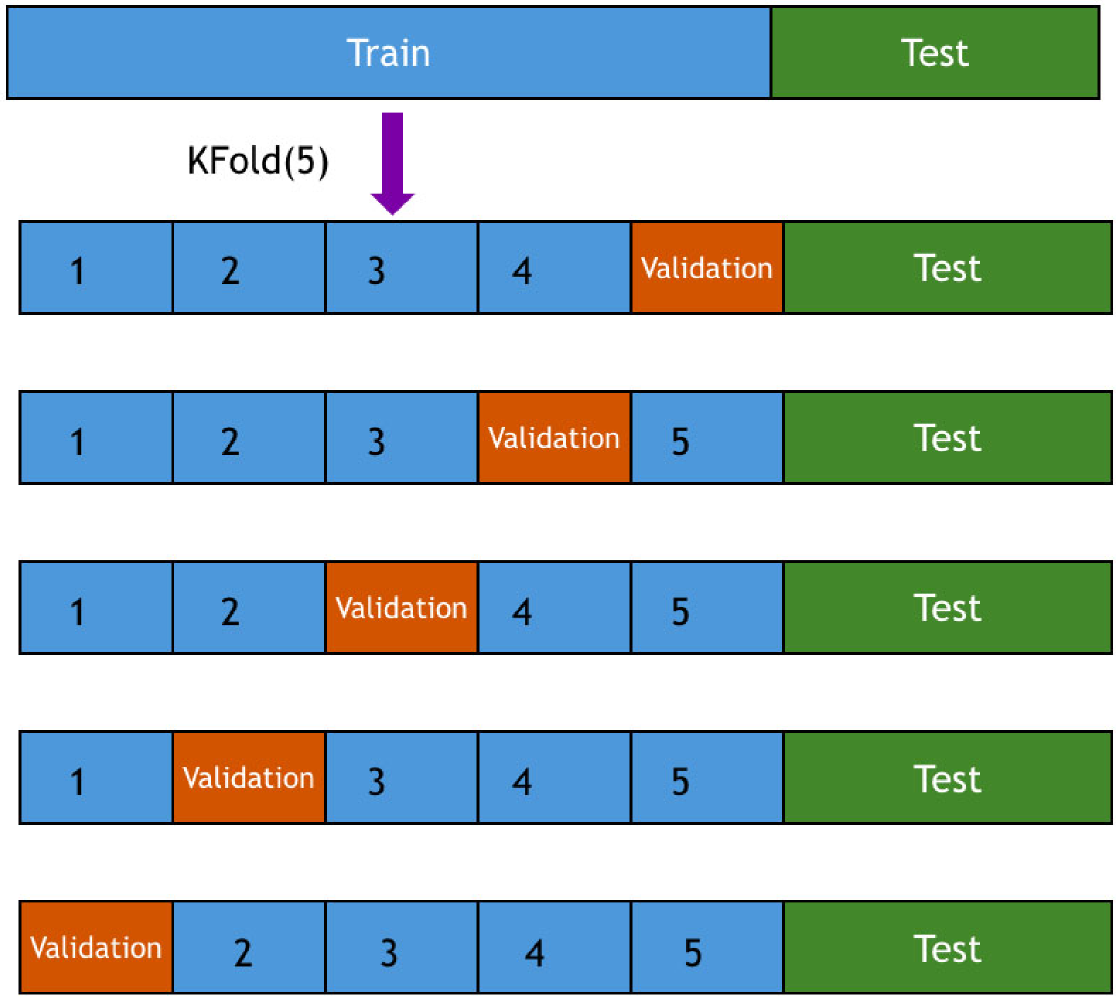
STD

```
2.832 (+/-0.116) for {'regression': 'poly', 'd': 3, 'lambda': 0.01}
1.854 (+/-0.072) for {'regression': 'poly', 'd': 3, 'lambda': 0.1}
1.910 (+/-0.065) for {'regression': 'poly', 'd': 3, 'lambda': 1}
1.902 (+/-0.077) for {'regression': 'poly', 'd': 3, 'lambda': 10}
2.844 (+/-0.101) for {'regression': 'poly', 'd': 4, 'lambda': 0.01}
2.864 (+/-0.089) for {'regression': 'poly', 'd': 4, 'lambda': 0.1}
1.910 (+/-0.065) for {'regression': 'poly', 'd': 4, 'lambda': 1}
1.894 (+/-0.086) for {'regression': 'poly', 'd': 4, 'lambda': 10}
2.848 (+/-0.080) for {'regression': 'poly', 'd': 5, 'lambda': 0.01}
1.904 (+/-0.064) for {'regression': 'poly', 'd': 5, 'lambda': 0.1}
0.916 (+/-0.069) for {'regression': 'poly', 'd': 5, 'lambda': 1}
1.870 (+/-0.072) for {'regression': 'poly', 'd': 5, 'lambda': 10}
2.846 (+/-0.090) for {'regression': 'poly', 'd': 6, 'lambda': 0.01}
2.906 (+/-0.062) for {'regression': 'poly', 'd': 6, 'lambda': 0.1}
1.904 (+/-0.075) for {'regression': 'poly', 'd': 6, 'lambda': 1}
2.858 (+/-0.112) for {'regression': 'poly', 'd': 6, 'lambda': 10}
```

BEST!

$d=5$, $\lambda = 1$

k-fold cross-validation



In short

- ✓ The goal is to **train** a **model** on a **training dataset** with good **generalization** capabilites
- ✓ Training = minimization of a **loss function**
- ✓ Has **hyper-parameters** that control the **capacity** of the model, choisis à l’aide d’une procédure de **sélection de modèle**
- ✓ mesure sa performance de **généralisation** sur un **ensemble de test**
- ✓ Aura une meilleure performance de généralisation si la **quantité de données d’entraînement augmente**
- ✓ Peut souffrir de **sous-apprentissage** (pas assez de capacité) ou de **sur-apprentissage** (trop de capacité)

Thank you!