Montreal, July 8–12

DLMI2024

Basics of deep learning part 1

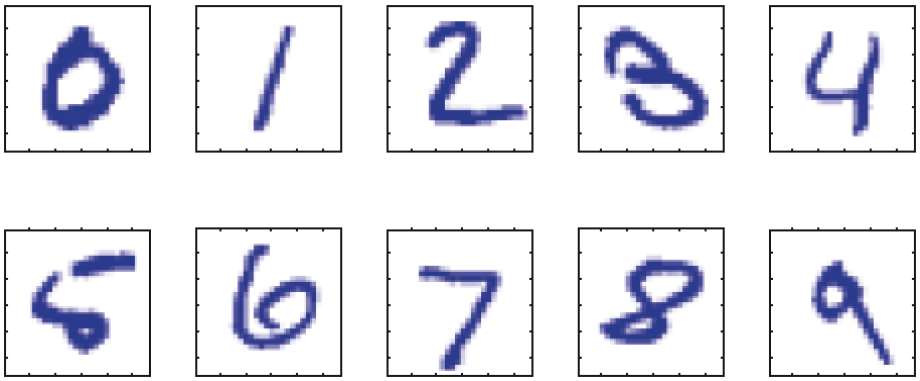
By

Pierre-Marc Jodoin

UNIVERSITÉ DE SHERBROOKE

What is machine learning?

Question : how can one recognize hand written digits?

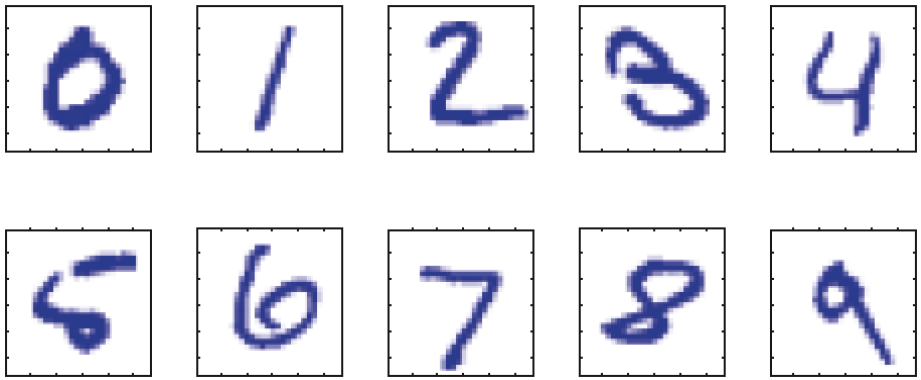


Answer : Design your own rules?

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

(Hugo Larochelle)

Question : how can one recognize hand written digits?

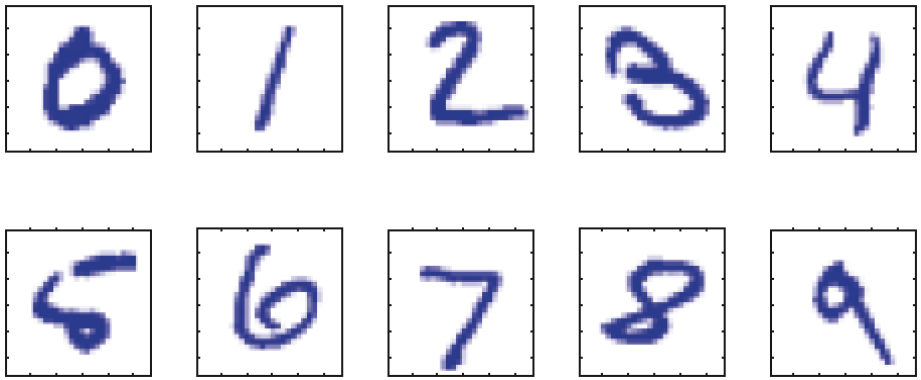


Answer : ~~Design your own rules?~~ **Wrong**

- Bad generalization

(Hugo Larochelle)

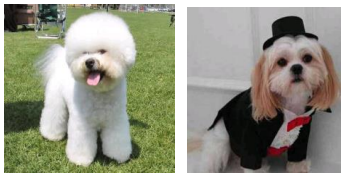
Question : how can one recognize hand written digits?



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

- Bad generalization

- Often difficult

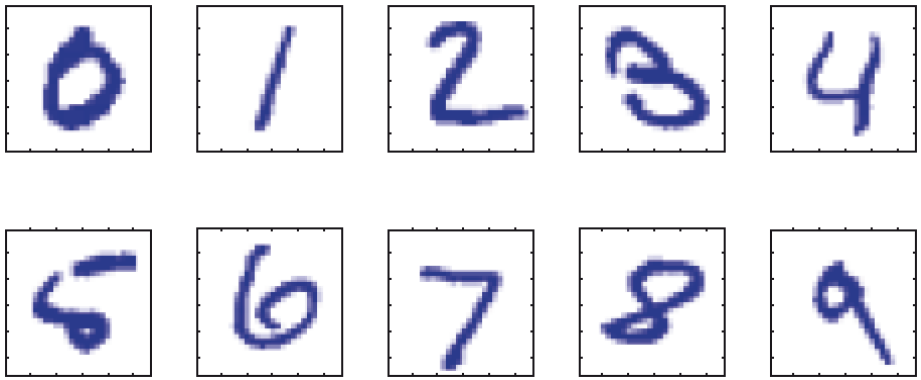


Vs

Dogs Birds

(Hugo Larochelle)

Question : how can one recognize hand written digits?

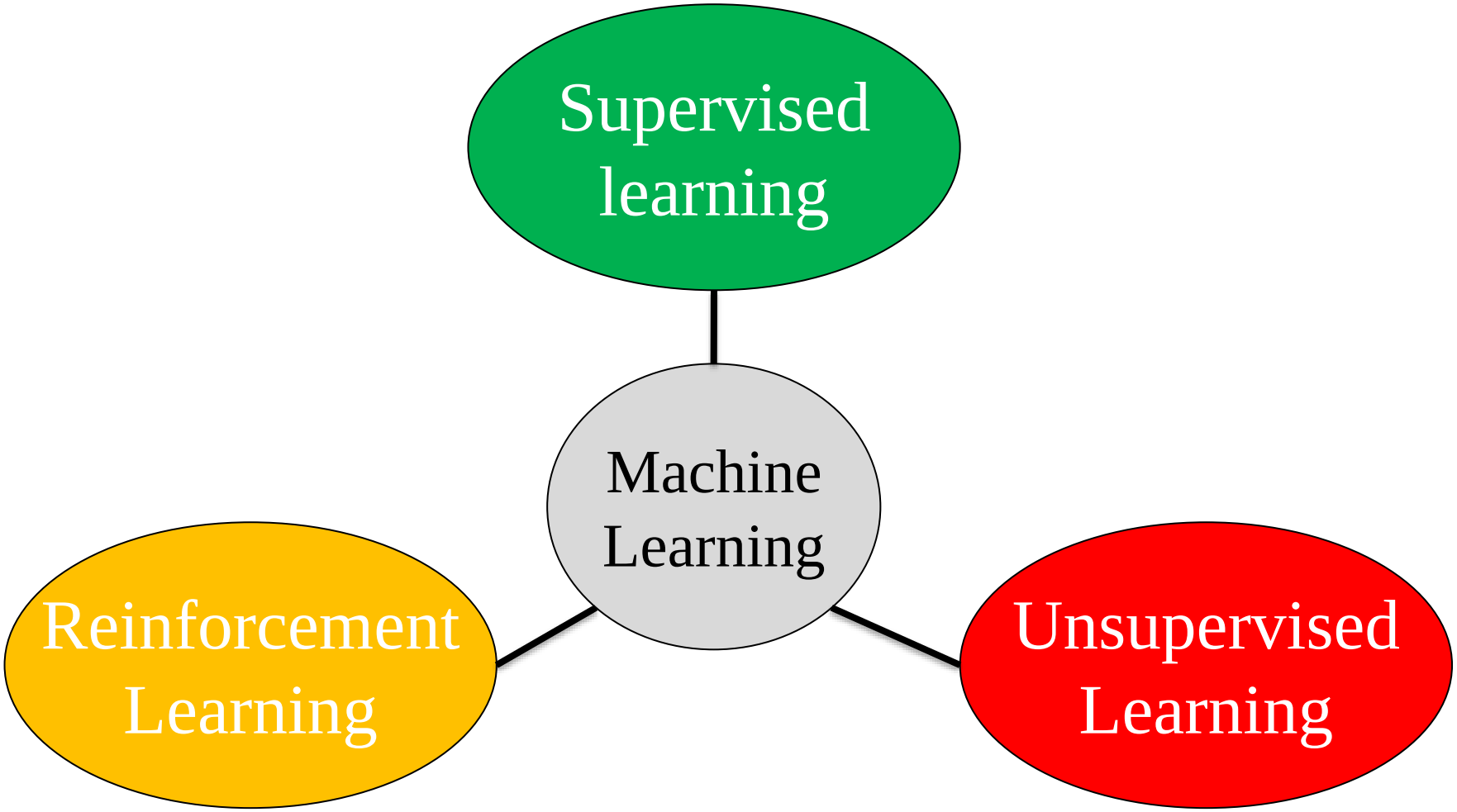


Answer : Let the computer « **learn** » the rules

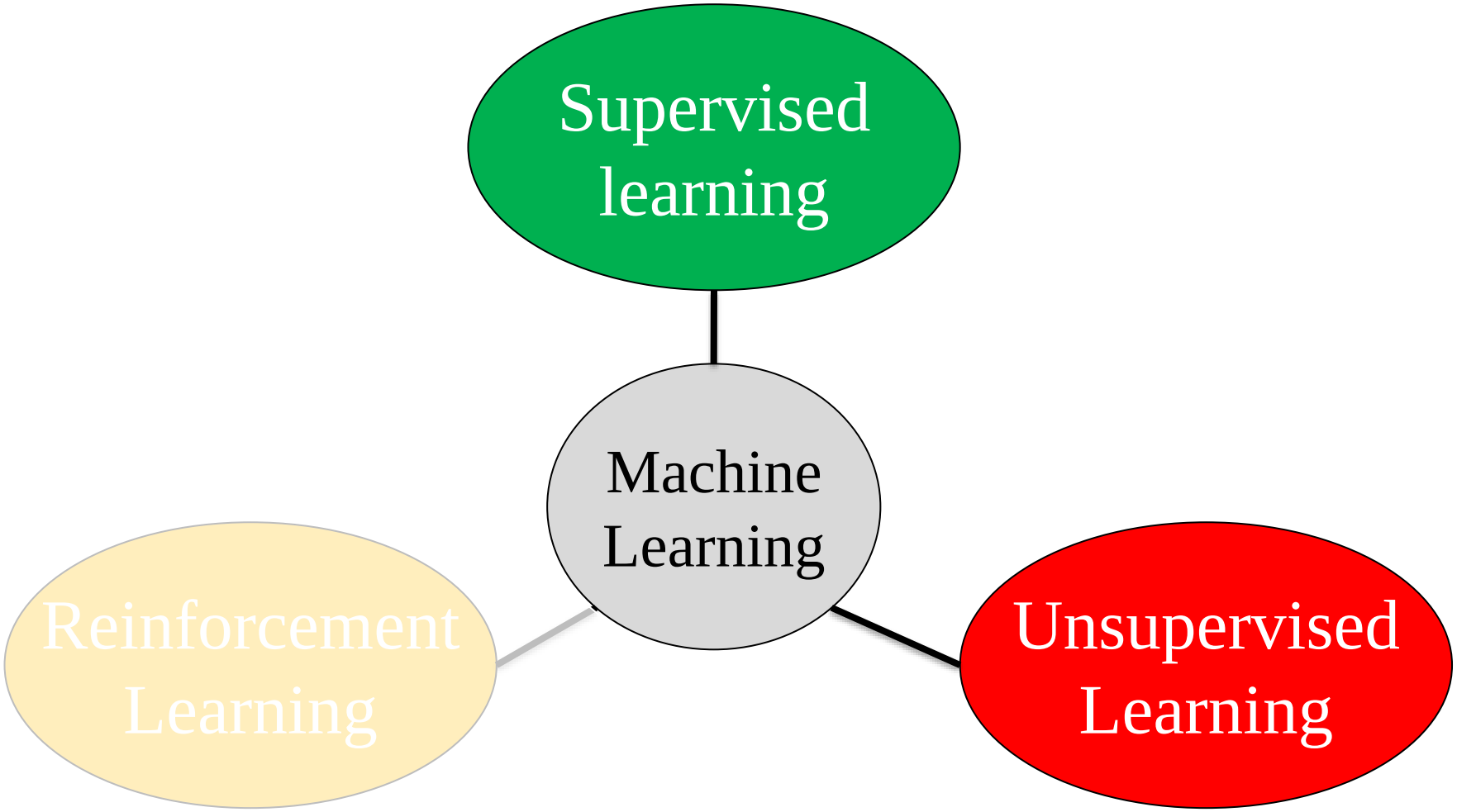
- *Main goal of machine learning*

(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’

Two large families

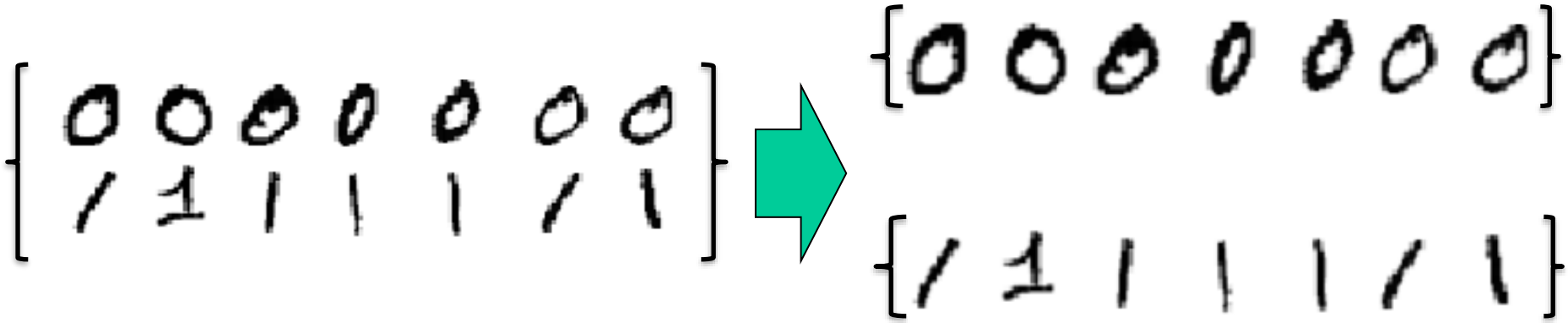
Supervised learning

Unsupervised learning

Unsupervised learning

When no target is explicitly provided

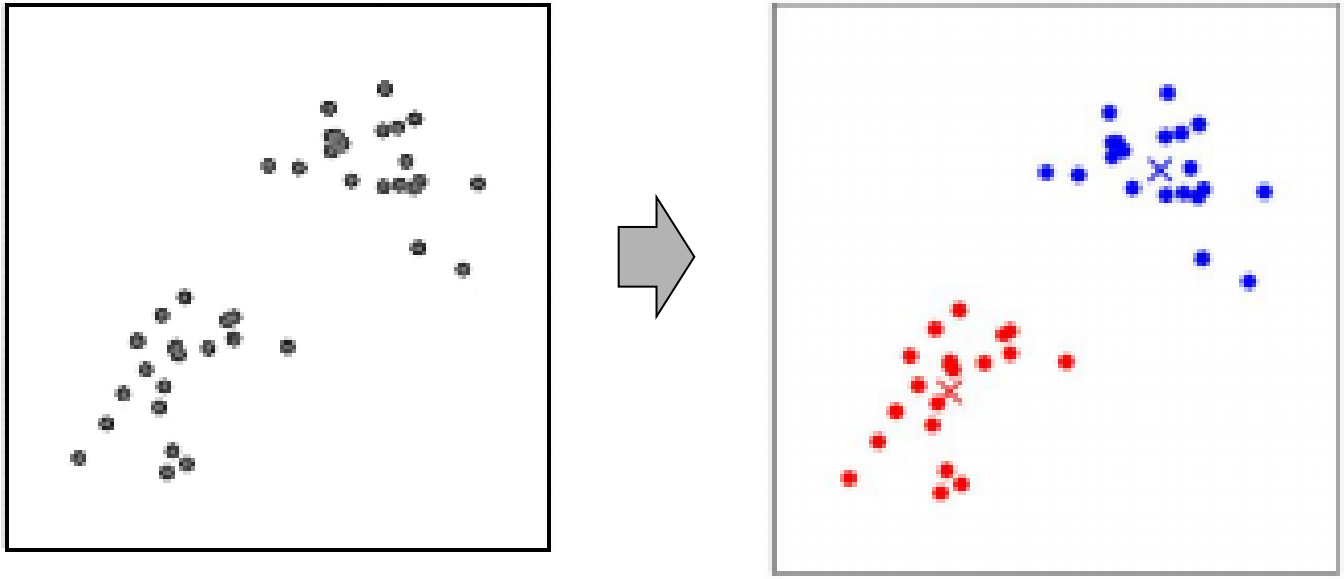
- E.g. data *clustering*



Unsupervised learning

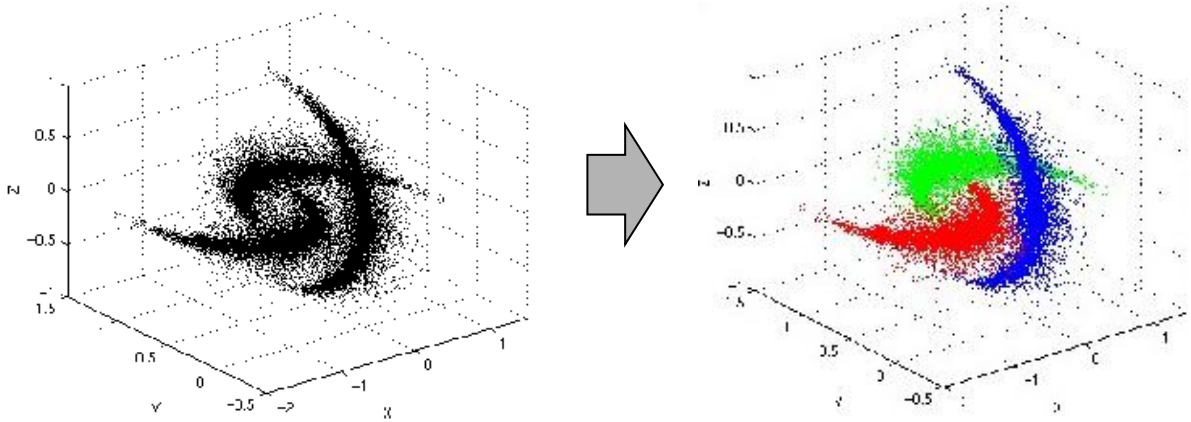
When no target is explicitly provided

- E.g. data *clustering*



Unsupervised learning

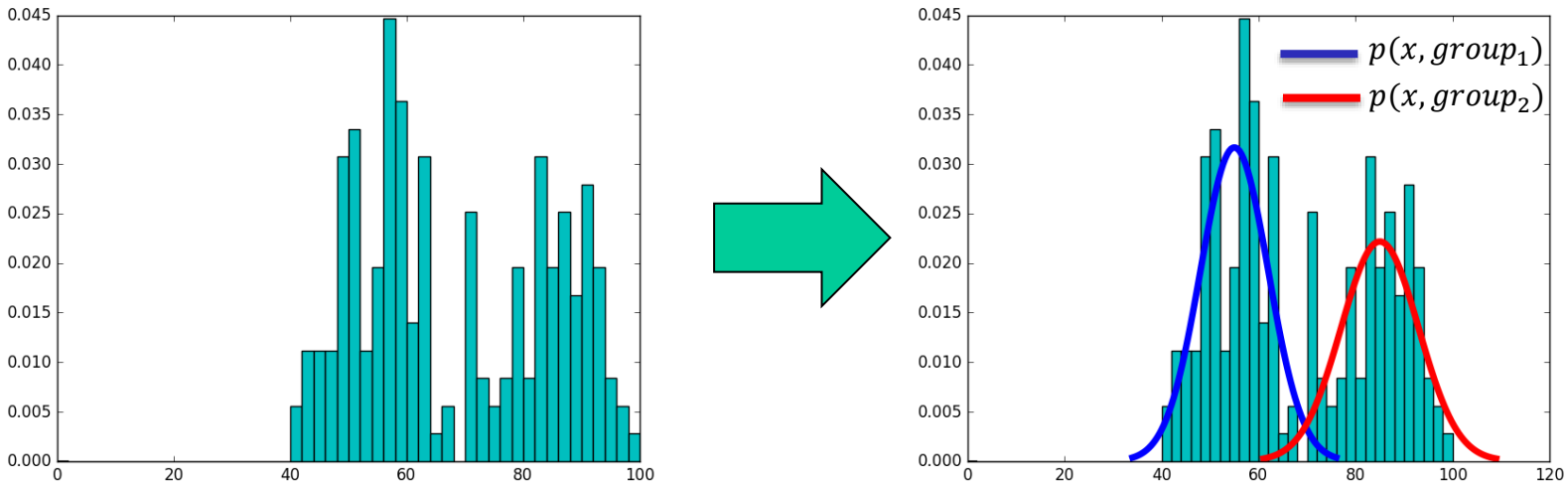
No limit to dimensionality. Could be 3D, 4D,…100kD



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

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

Unsupervised learning : unknown

Autoencoders
Variational autoencoders
GANs

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

Back to 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.

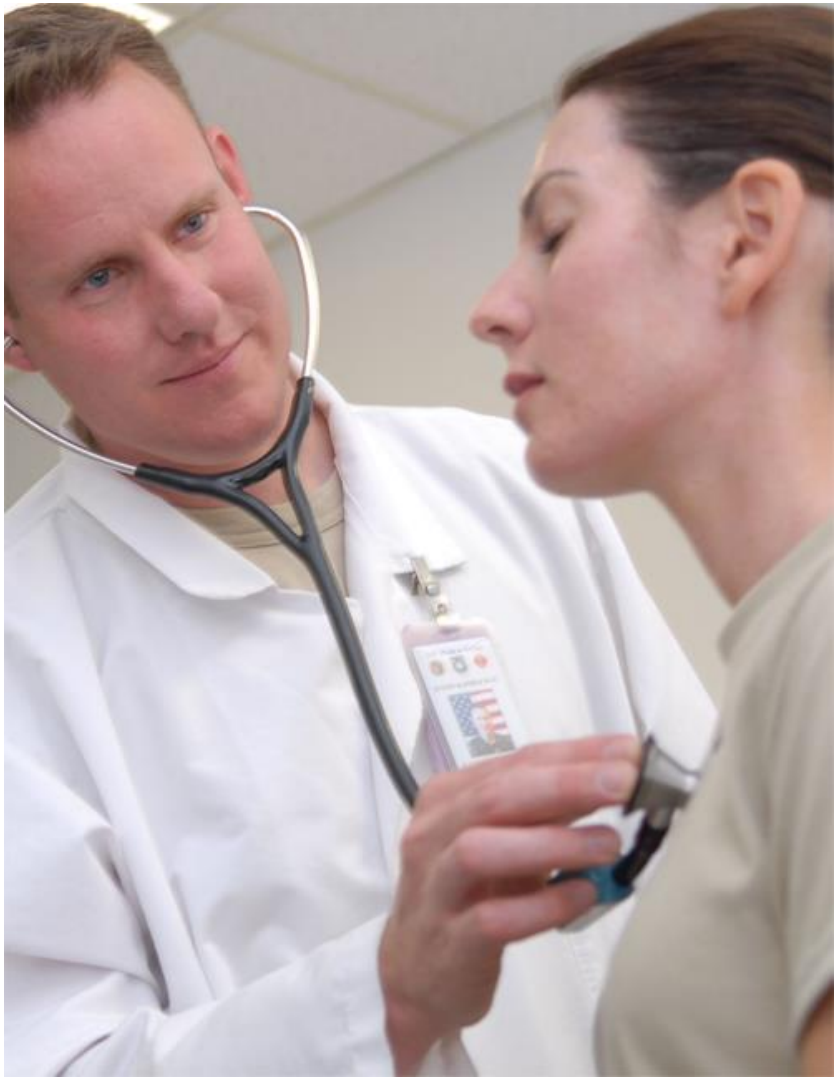
Supervised learning

Two main applications

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

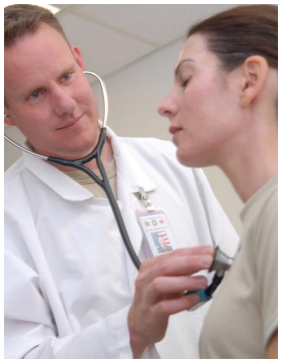
- **Régression :** the target is a real number $t \in \mathbb{R}$
  - Exemple : prediction of life expectancy
    - ✓ $\vec{x}$: vector of medical measures, age, sex, etc.
    - ✓ t : number of months before death.

Simple example of binary classification



From Wikimedia Commons
the free media repository

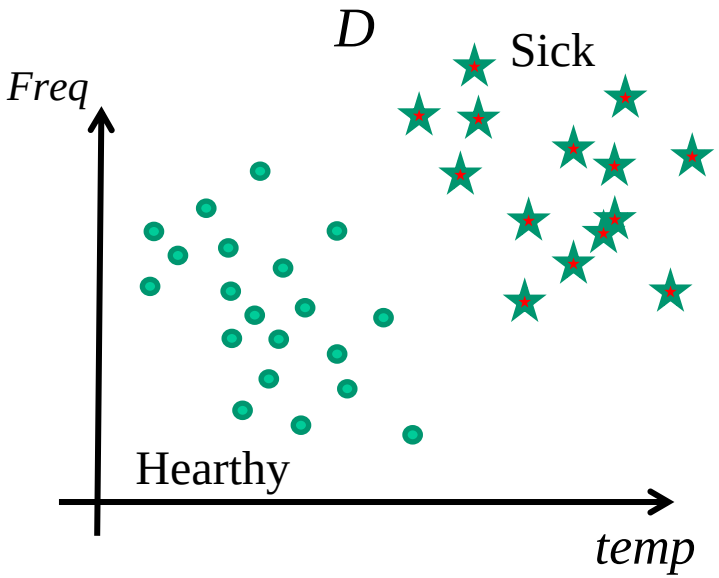
Simple example of binary classification



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

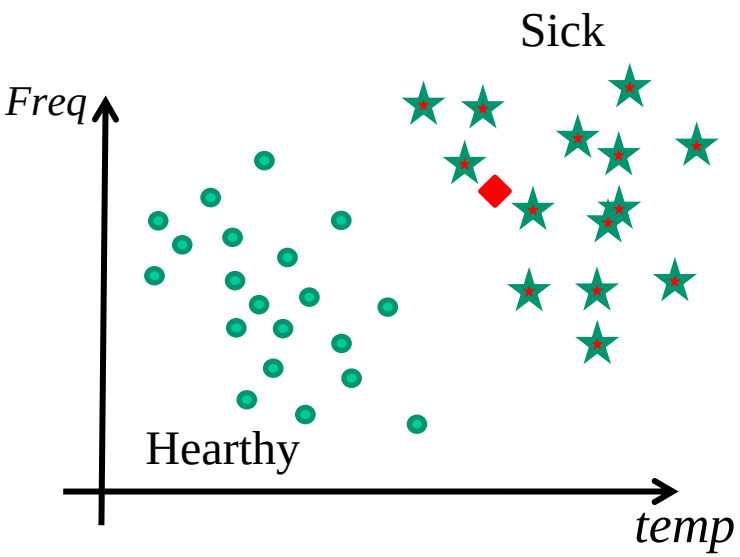
$\vec{x}$ t



Simple example of binary classification

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

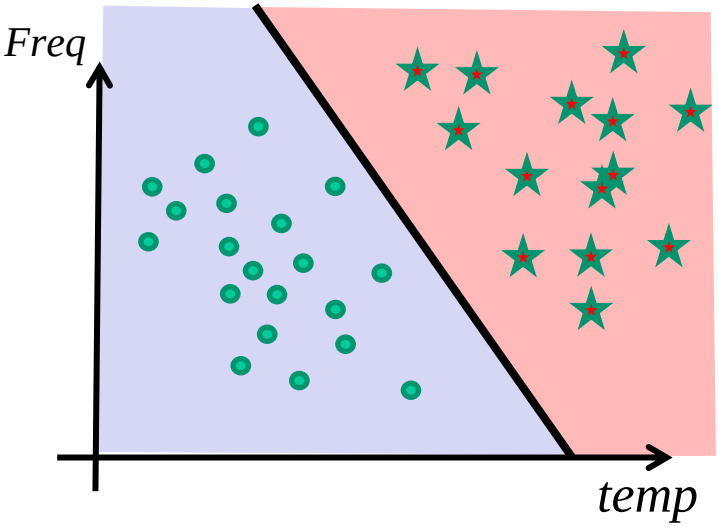
From Wikimedia Commons
the free media repository



Solution

From Wikimedia Commons
the free media repository

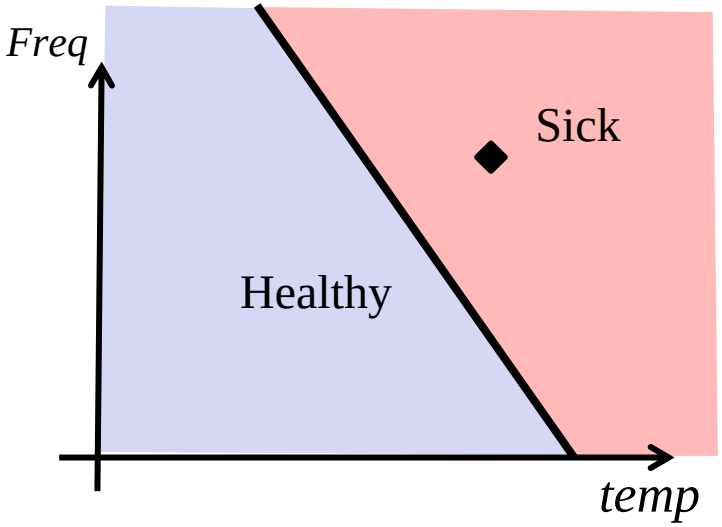
Divide the feature space in two regions : healthy and sick



Solution

From Wikimedia Commons
the free media repository

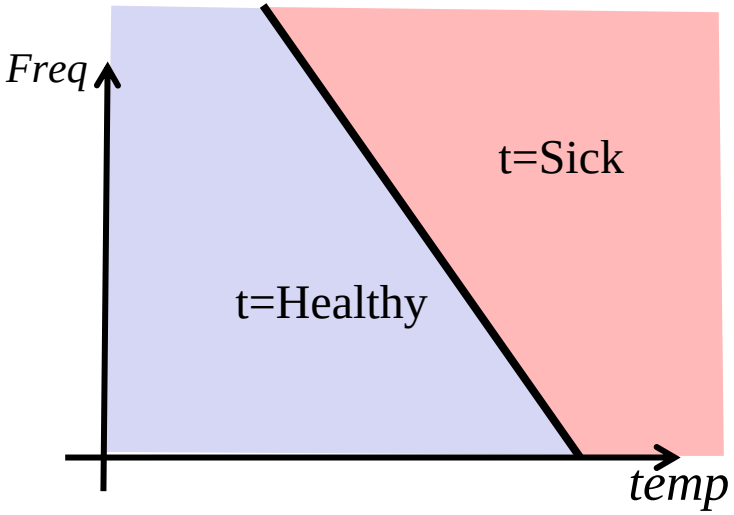
Divide the feature space in two regions : healthy and sick



More formally

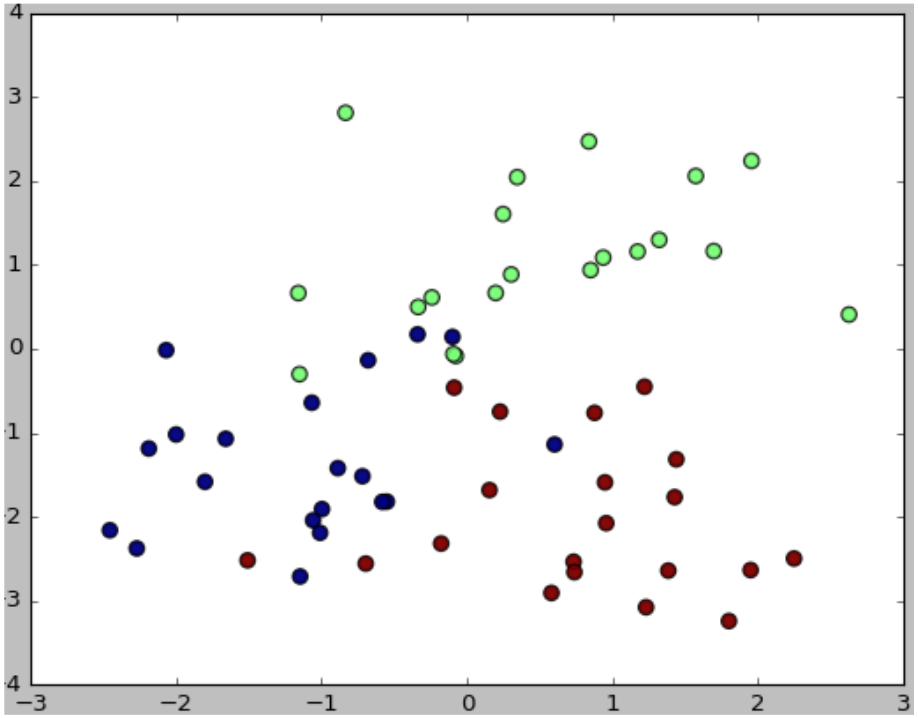
From Wikimedia Commons
the free media repository

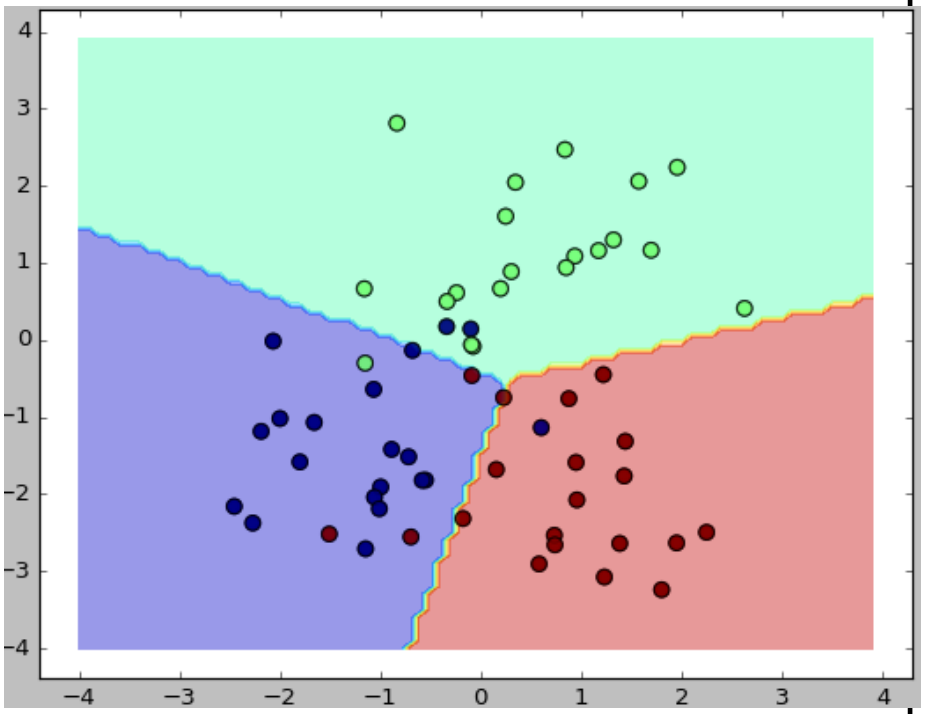
$$y_W(\vec{x}) = \begin{cases} \textit{H}\text{ealthy if } \vec{x} \text{ is in the blue region} \\ \textit{Sick} \quad \text{otherwise} \end{cases}$$



Classification

3-class case





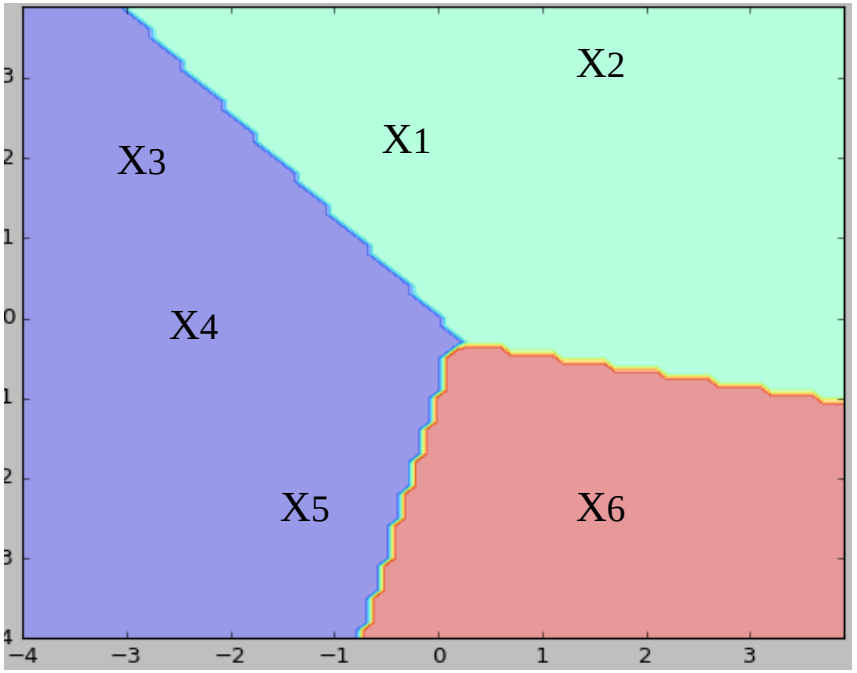
3 classes ●, ●, ● in a 2D feature space

Once training is over

y_w(●) = class 1
y_w(●) = class 2
y_w(●) = class 3

Classification

Once training is over, we have a function $y_W(\vec{x})$ that convert a point x into a class label



$y_W(x_1)$ => class

$y_W(x_2)$ => class

$y_W(x_3)$ => class

$y_W(x_4)$ => class

$y_W(x_5)$ => class

$y_W(x_6)$ => class

Example of a classification dataset

MNIST

Example of a classification dataset

MNIST

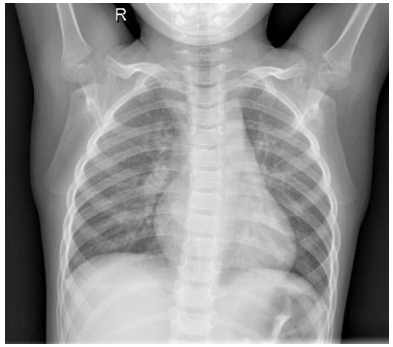
- 10 classes
- 70,000 images
  => 60,000 training
  => 10,000 test
- Images are in grayscale
  => 28x28

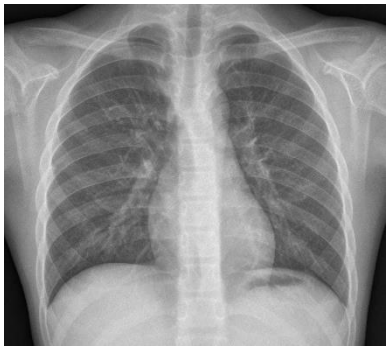
We can **vectorize these images** and represent it by a vector of size 28x28 = **784 dimensions.**

Example of a medical classification dataset

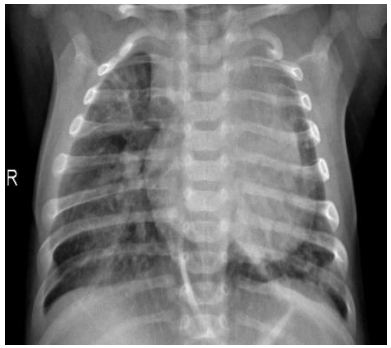
Chess X-Ray Pneumonia

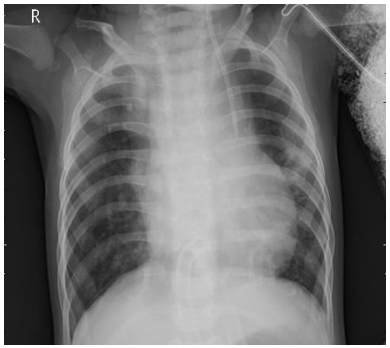
Healthy





Pneumonia





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

Example of a medical classification dataset

Chess X-Ray Pneumonia

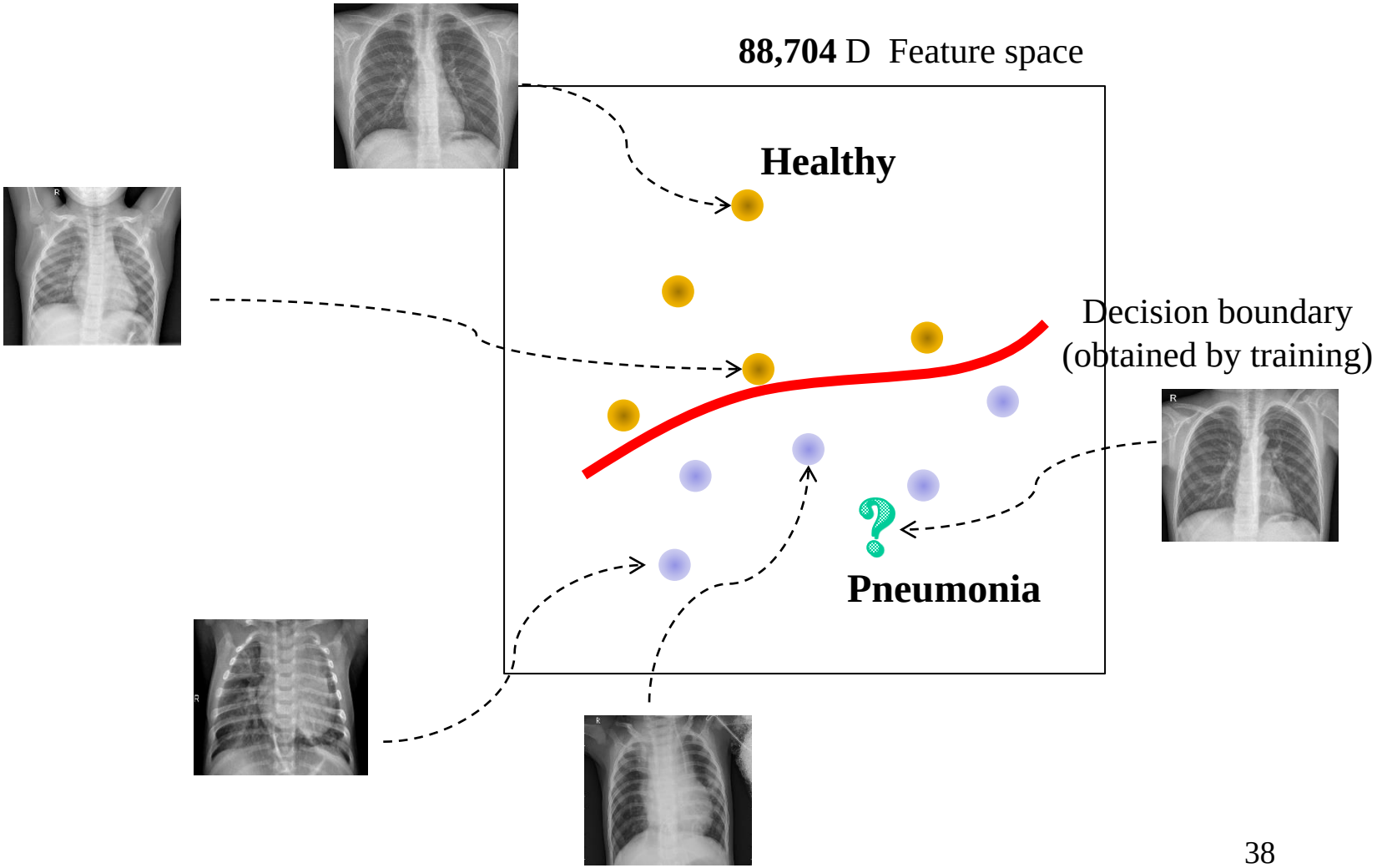
- 2 classes
- 5,840 images,
  => 5,216 training
  => 624 test
- Each image is in grayscale
  => 336 x 264*

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

* Rescaled version

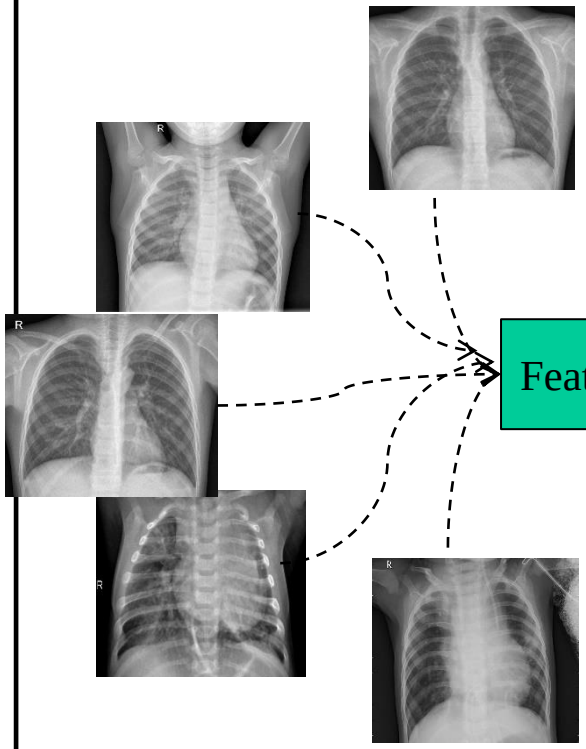
Supervised learning

Chess X-Ray Pneumonia



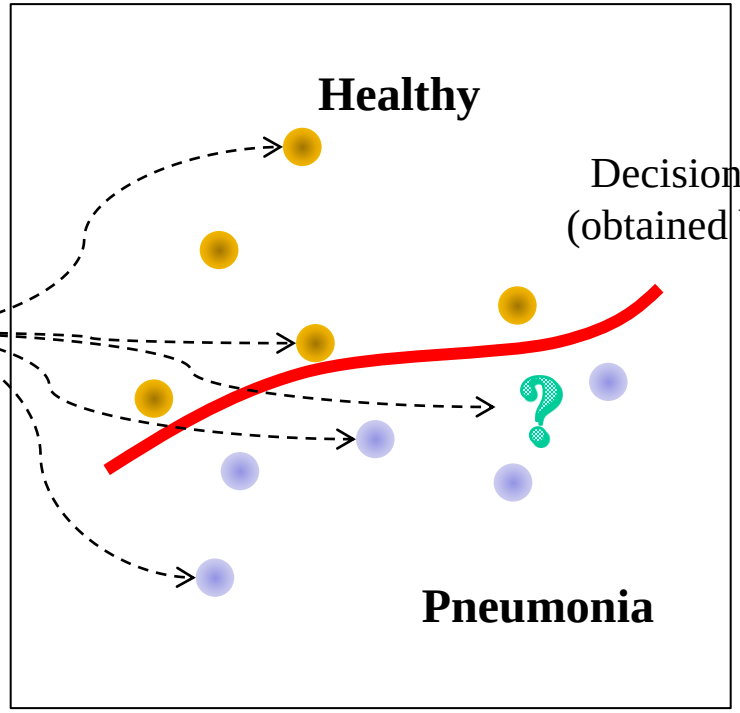
Very large feature spaces (like **88,704 dim) are problematic**.

Supervised learning



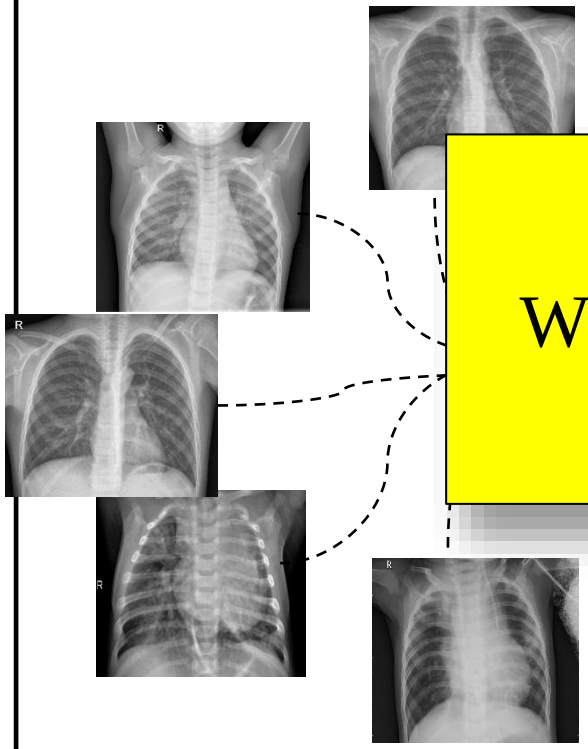
Feature extraction

K-D Feature space

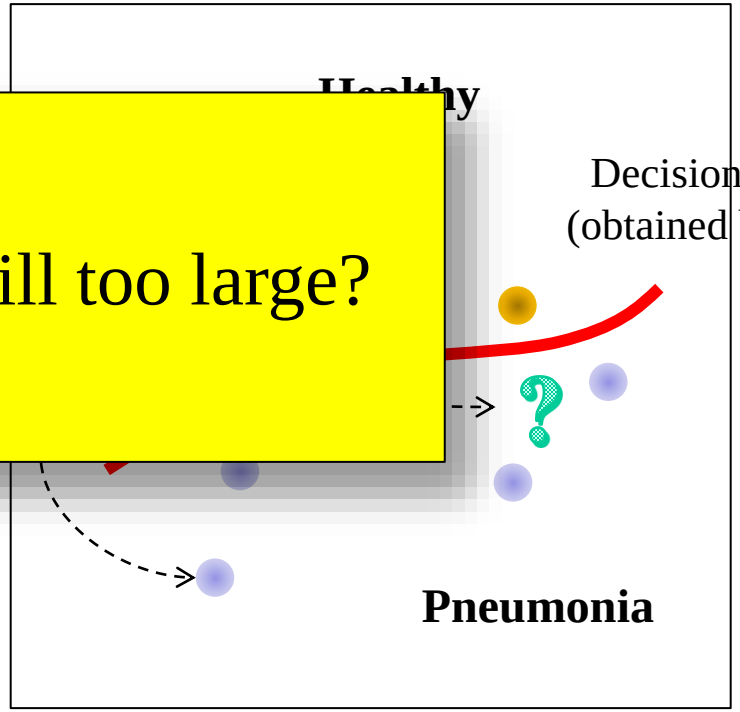


Edge detection
Histograms
Filters of all kinds
Moments
Etc.

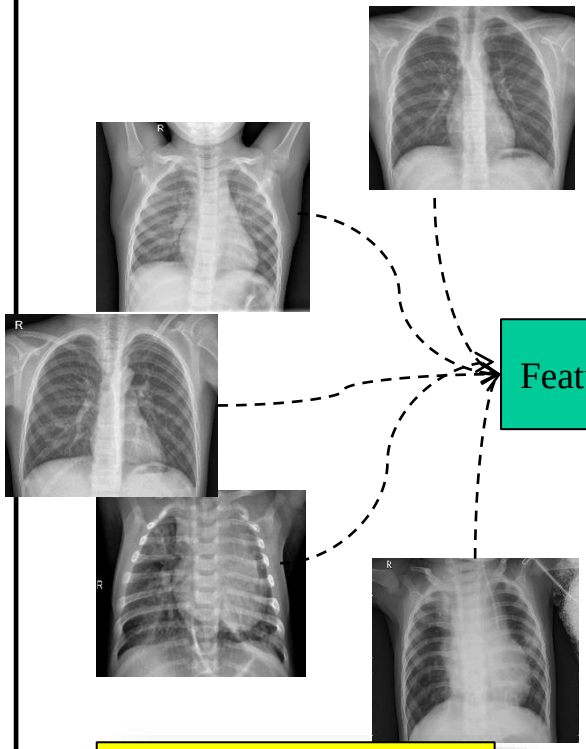
Supervised learning



What if K is still too large?



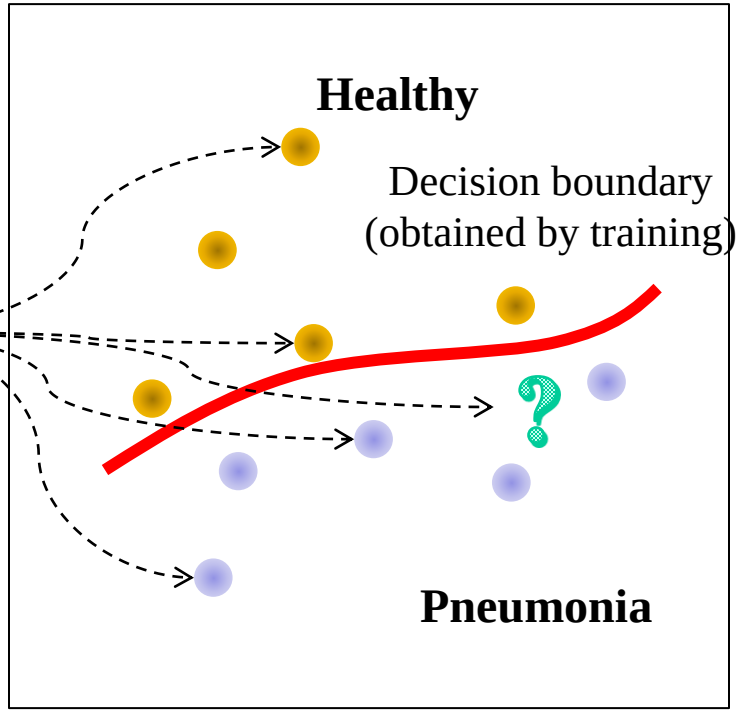
Supervised learning



Feature extraction

Dimensionality reduction

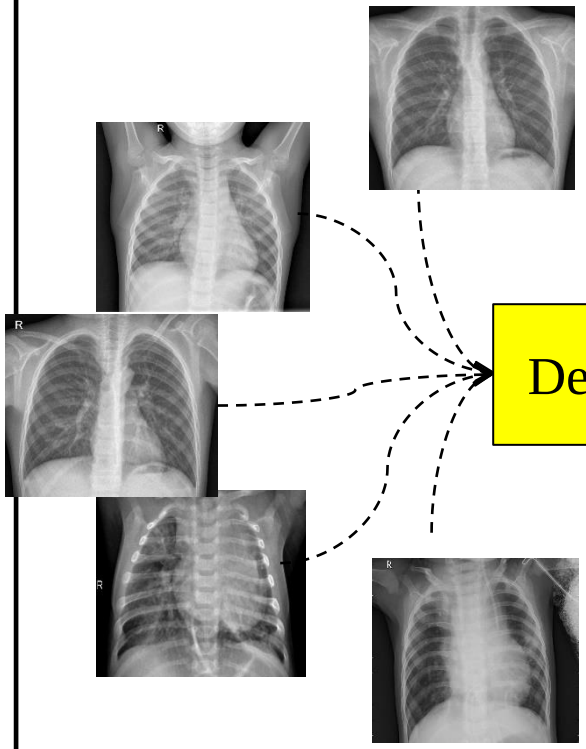
k-D Feature space



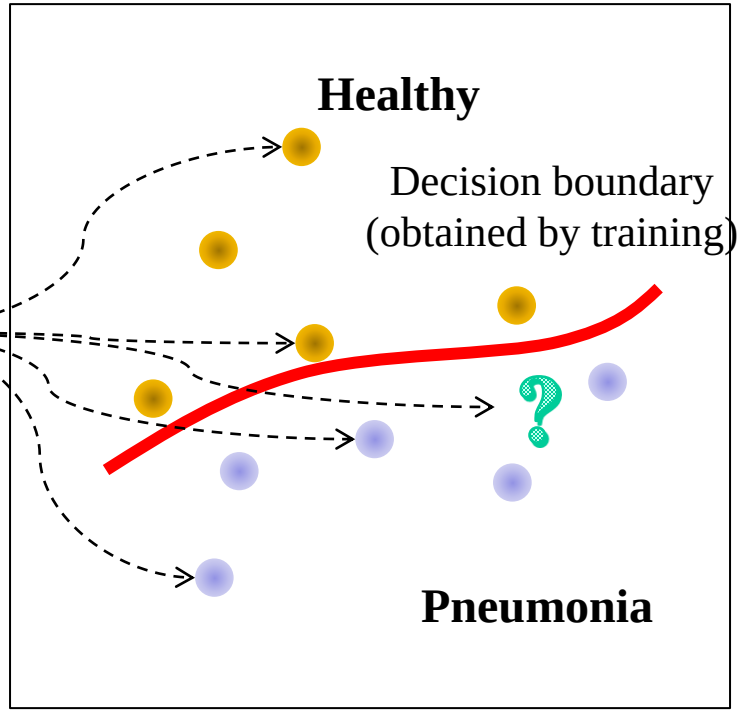
Feature selection
PCA
Kernel PCA
LLE
ISOMAPS
Etc.

Spoiler alert

In 2024…



Deep neural network



Supervised learning

Two main applications

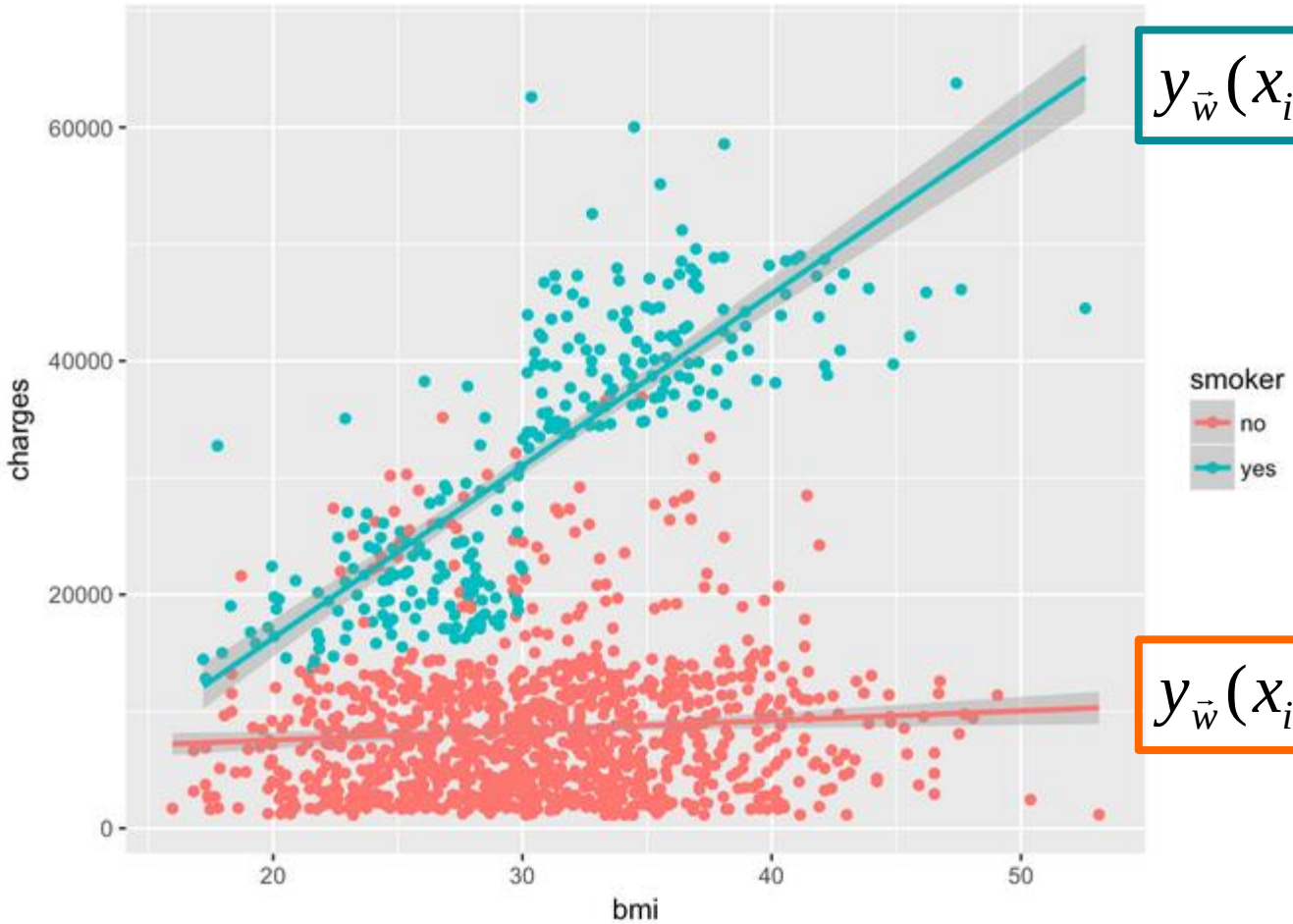
- **Classification :** the target is a class label $t \in \{1, \dots, K\}$
  - Exemple : disease recognition
    - ✓ $\vec{x}$: vector of medical measures, age, sex, etc.
    - ✓ t : myocardial infarction, dilated cardiomyopathy, hypertrophic cardiomyopathy, normal

- **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.

Example

(Linear)

Medical Cost Personal Datasets



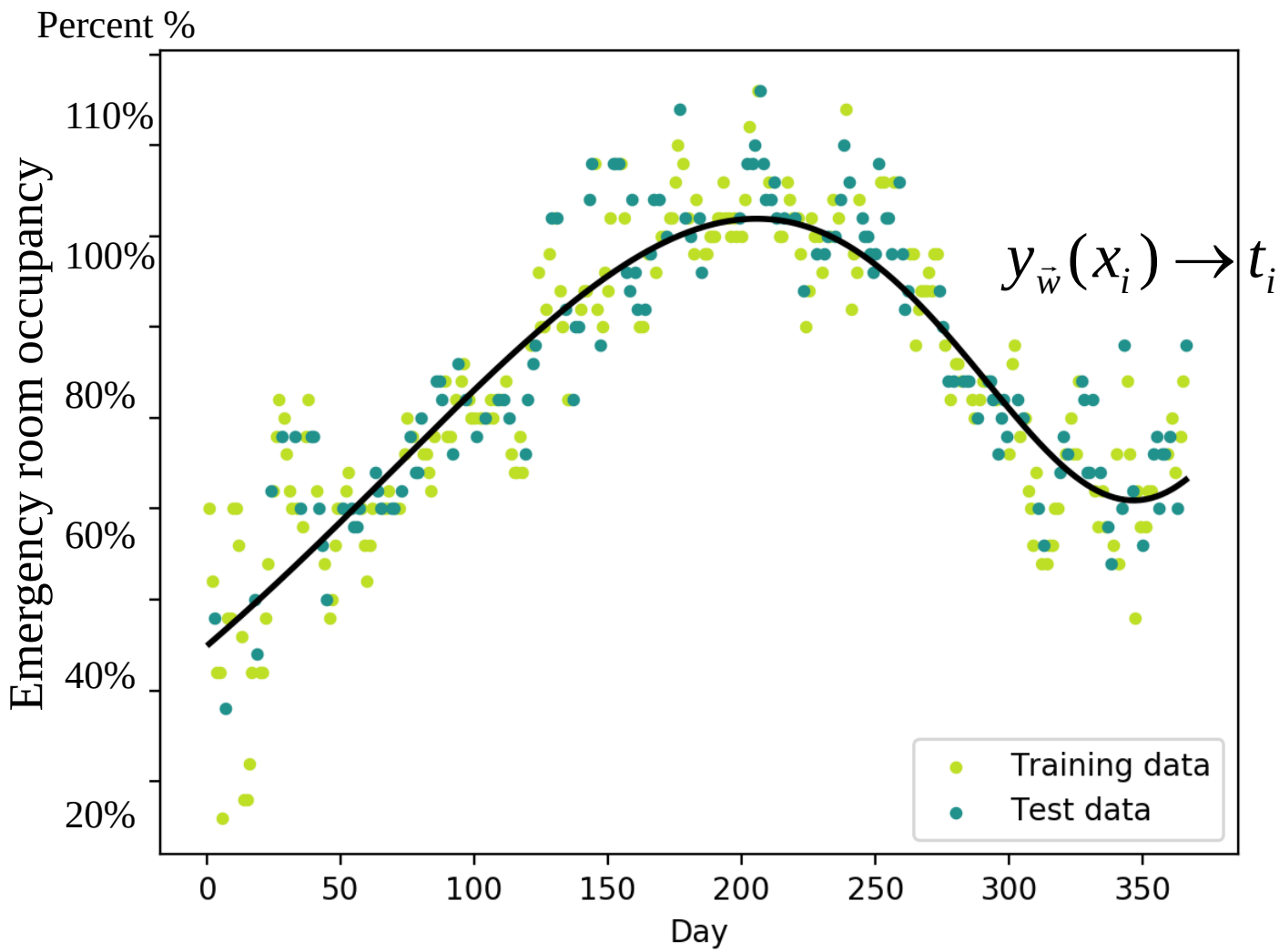
$$y_{\vec{w}}(x_i) \rightarrow t_i$$

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

https://rpubs.com/koki25ando/medicalcost

Example

(Nonlinear)



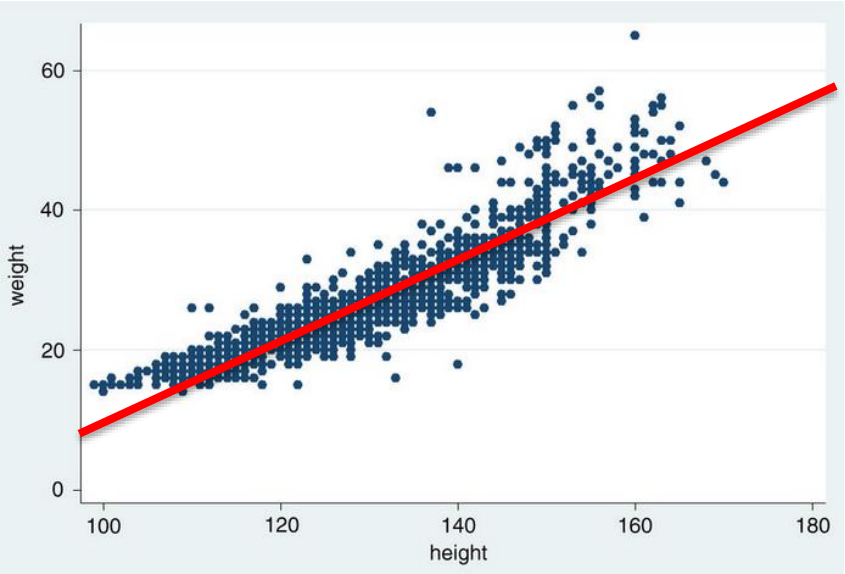
Before deep neural nets were … **linear models**

Vs
?

Linear models are to deep neural nets what atoms are to matter

Linear models are still relevant

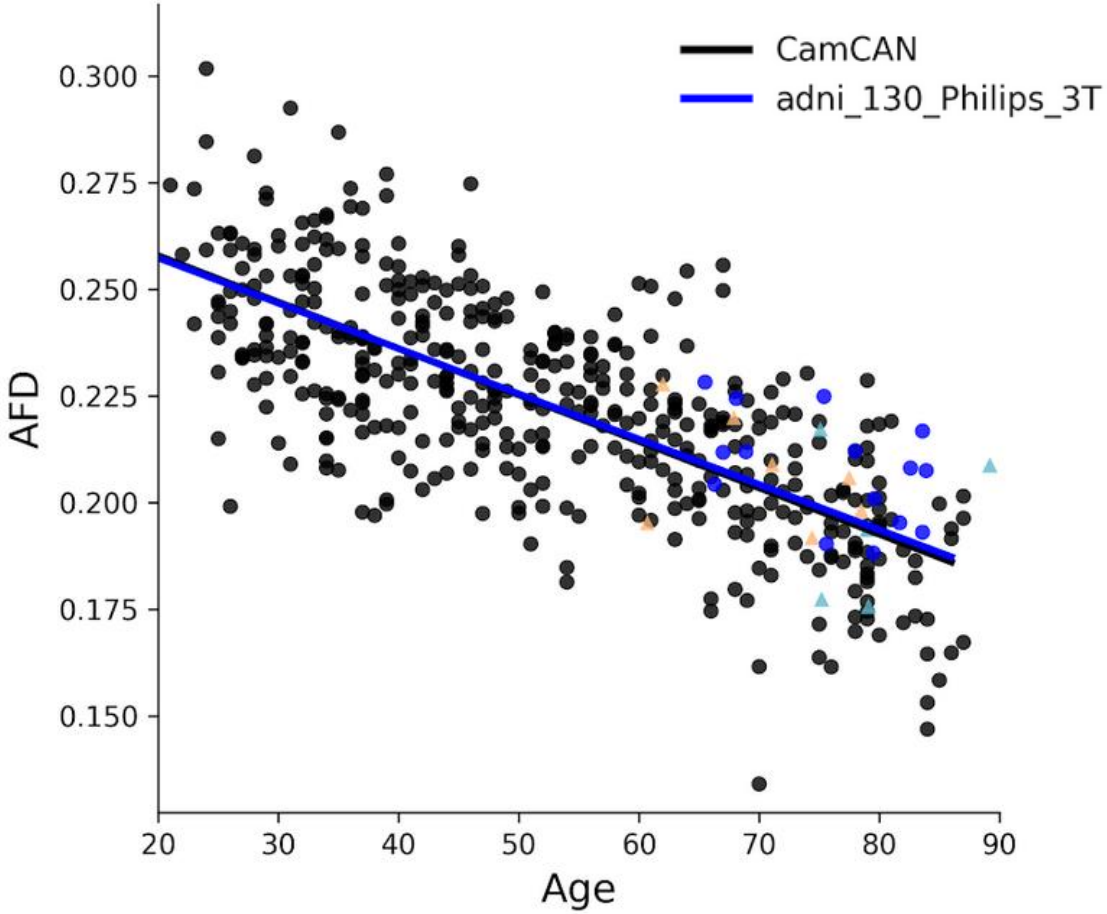
1,694 children surveyed in Tanzania.

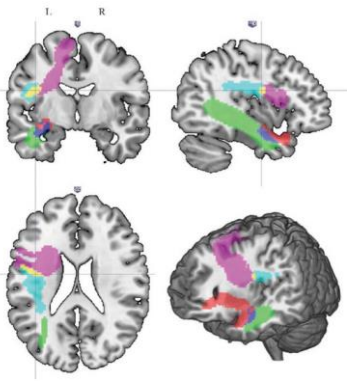


Nordin P, Poggensee G, Mtweve S, Krantz I. **From a weighing scale to a pole: a comparison of two different dosage strategies in mass treatment of Schistosomiasis haematobium**. Glob Health Action. 2014

Linear models are still relevant

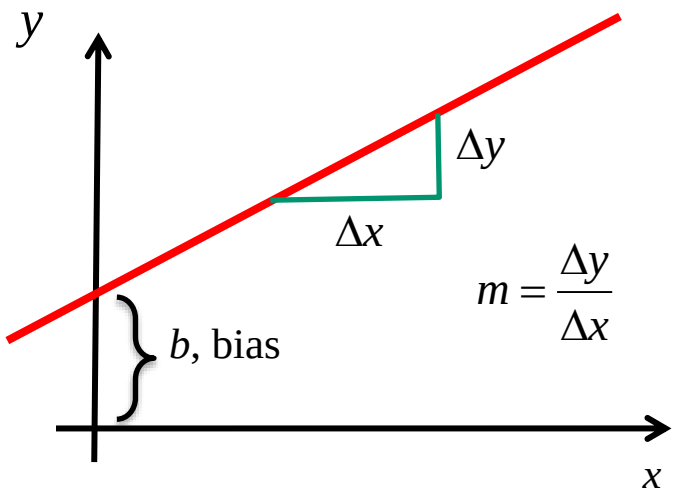
Apparent Fiber Density
in the white matter





https://commons.wikimedia.org/

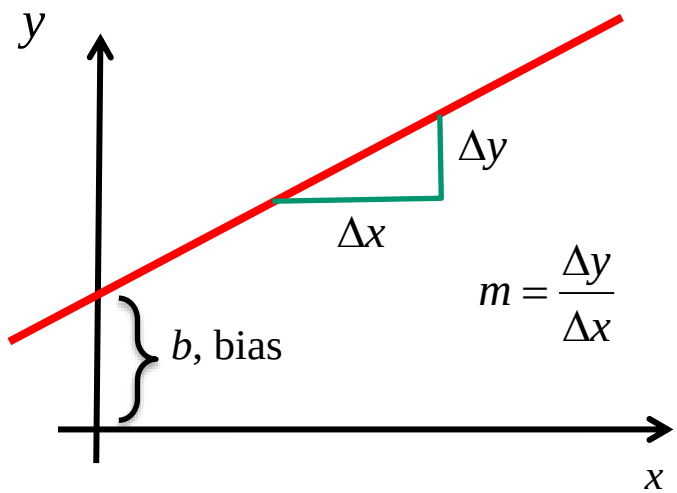
Definition … a line!



$$y = mx + b$$

slope bias

Definition … a line!



$$y = mx + b$$

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

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

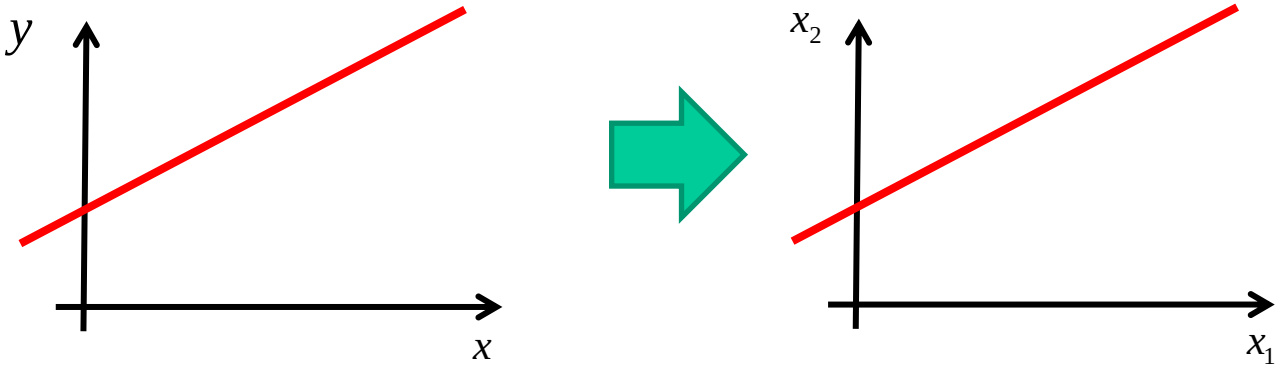
$$0 = \Delta y x - \Delta x y + b\Delta x$$

Rename variables

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

Rename variables

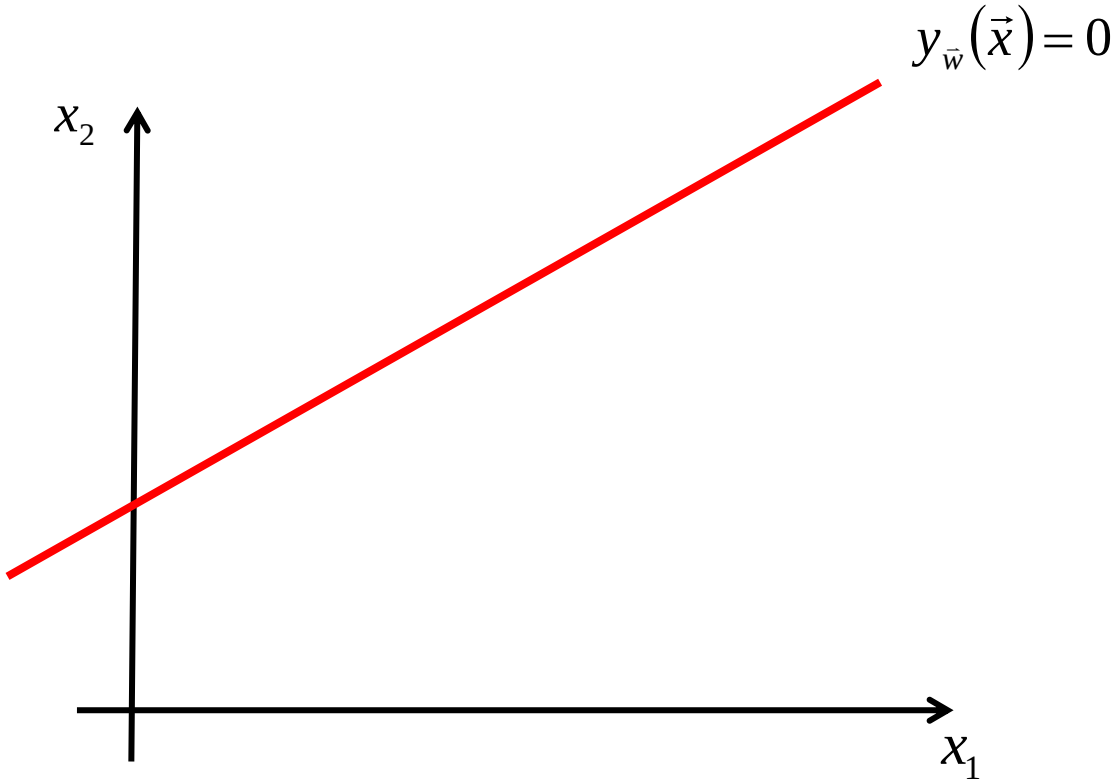
$$0 = w_1 x + w_2 y + w_0$$



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

Implicit function

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



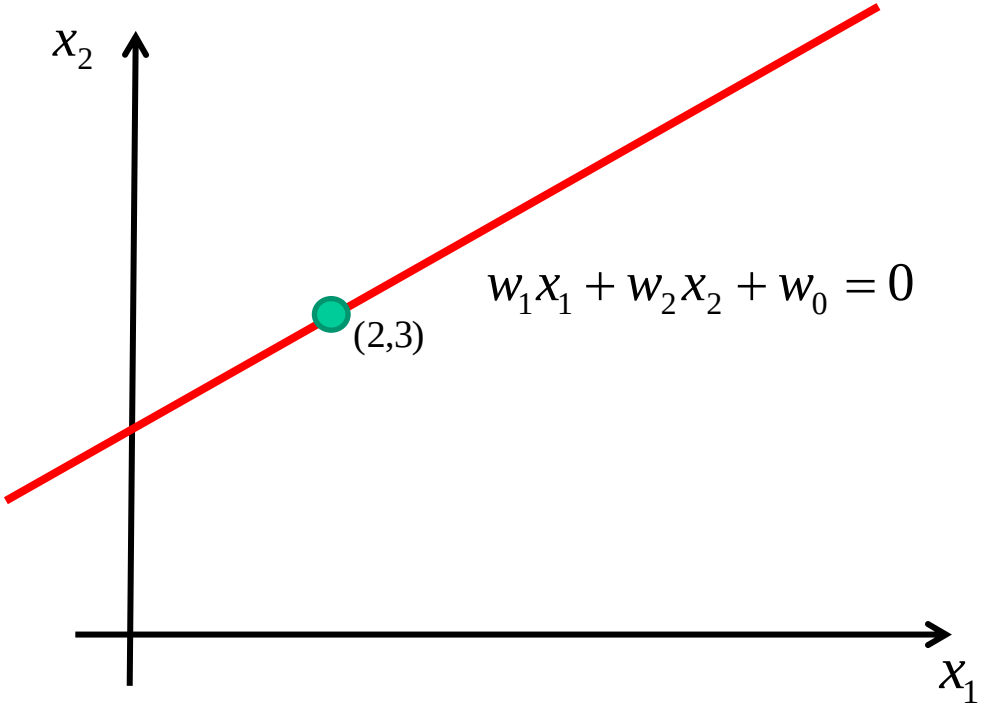
Implicit function

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

$w_1 = 1.0$

$w_2 = -2.0$

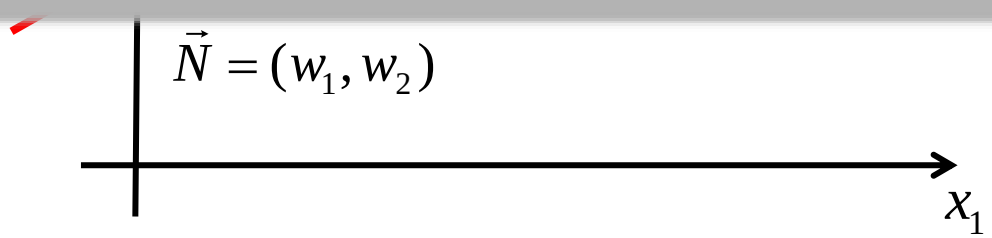
$w_0 = 4.0$



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



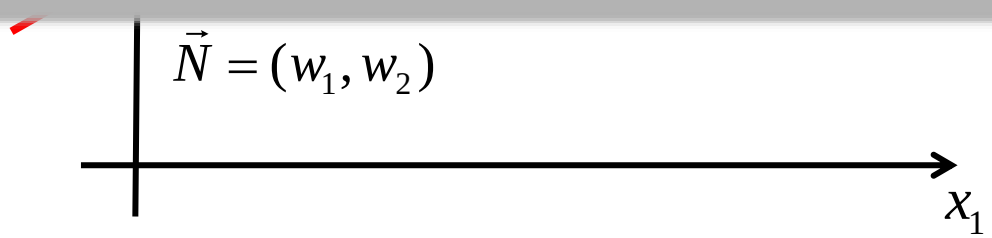
Implicit function

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

$$= (w_0, w_1, w_2) \cdot (1, x_1, x_2)$$

$$= \vec{w}^T \vec{x}'$$

DOT product



Equation of a line = dot product with bias included

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

Linear regression

- **Linear regression** model:

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

where $\vec{x} = \left(x_1, x_2, \ldots, x_d\right)^{\mathrm{T}}$

- The model is…
  - A **line** for *d=1*
  - A **plane** for *d=2*
  - A **hyperplan** for *d>2*

Linear regression

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

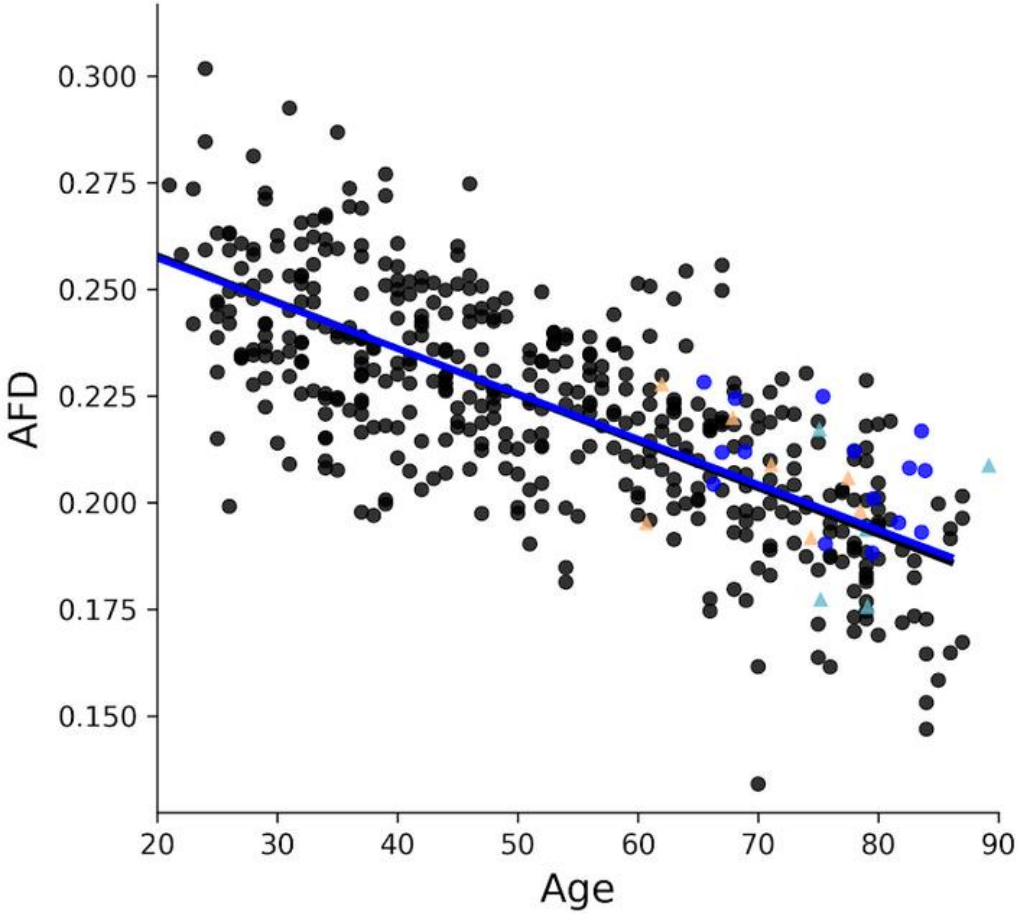
Bias

weights

A line : 1D regression

Example

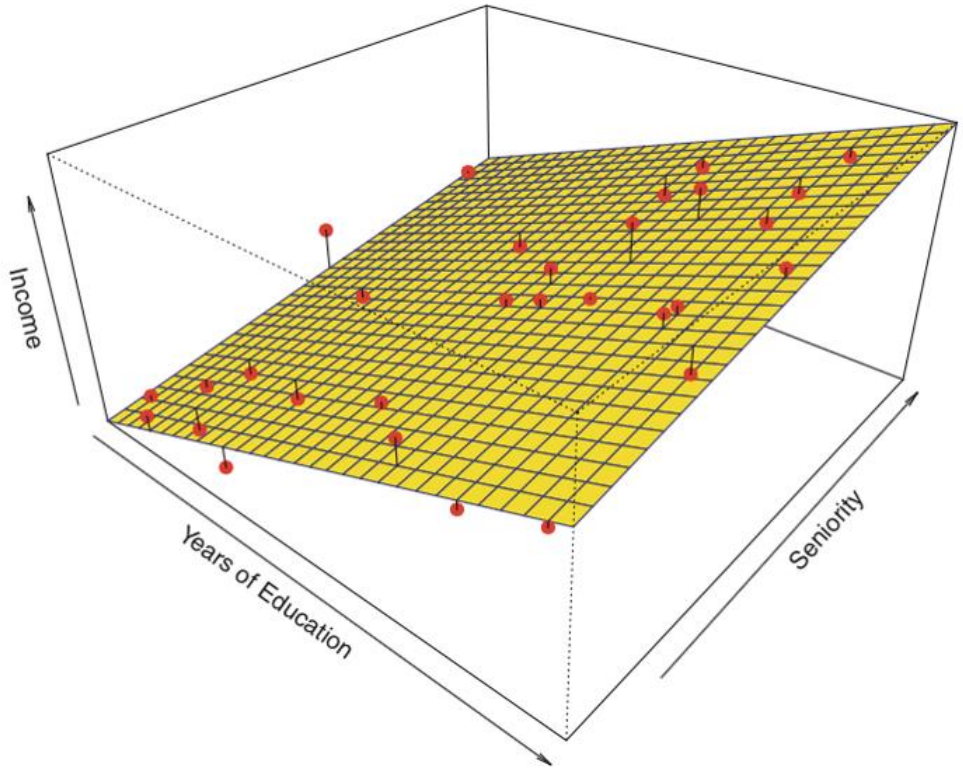
$$y_{\vec{w}}(x) = w_0 + w_1 x$$



A plane : 2D regression

Example

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



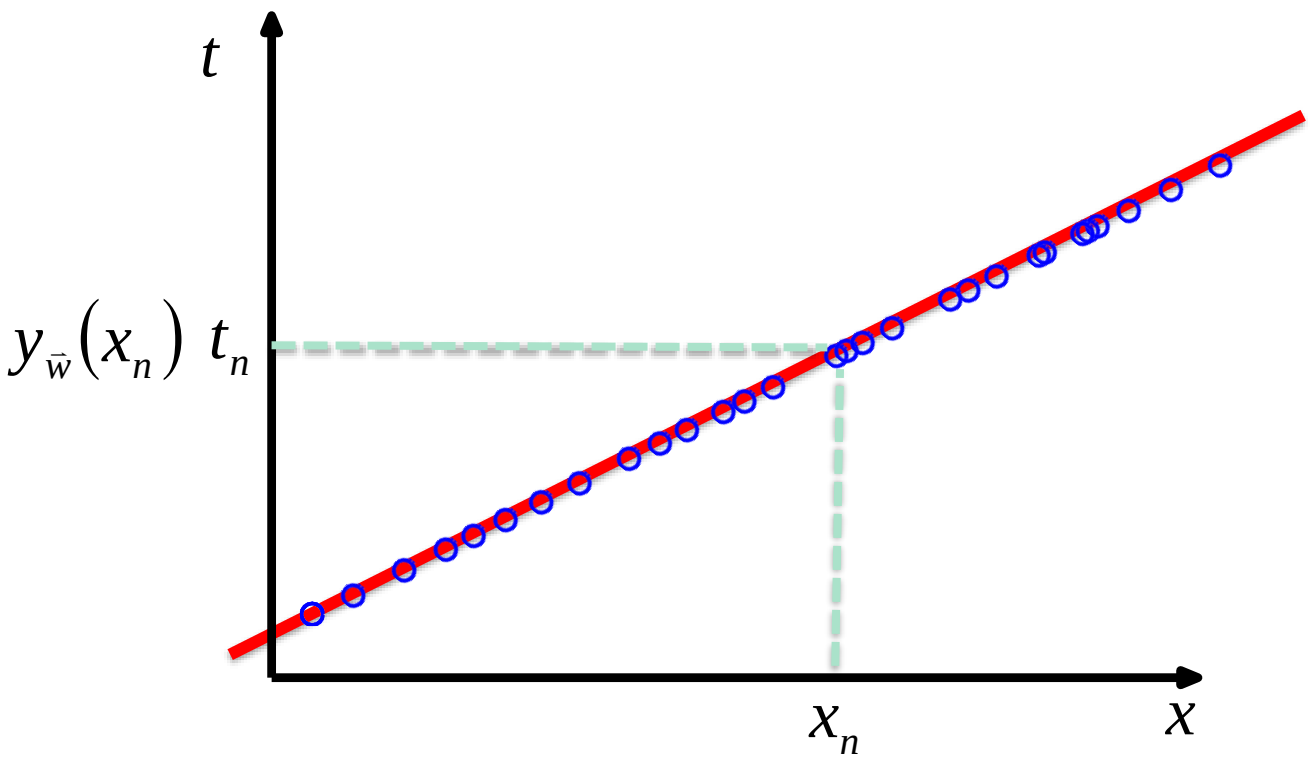
Credit : sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/R/R5_Correlation-Regression/R5_Correlation-Regression4.html

Problem to solve

Given a training example

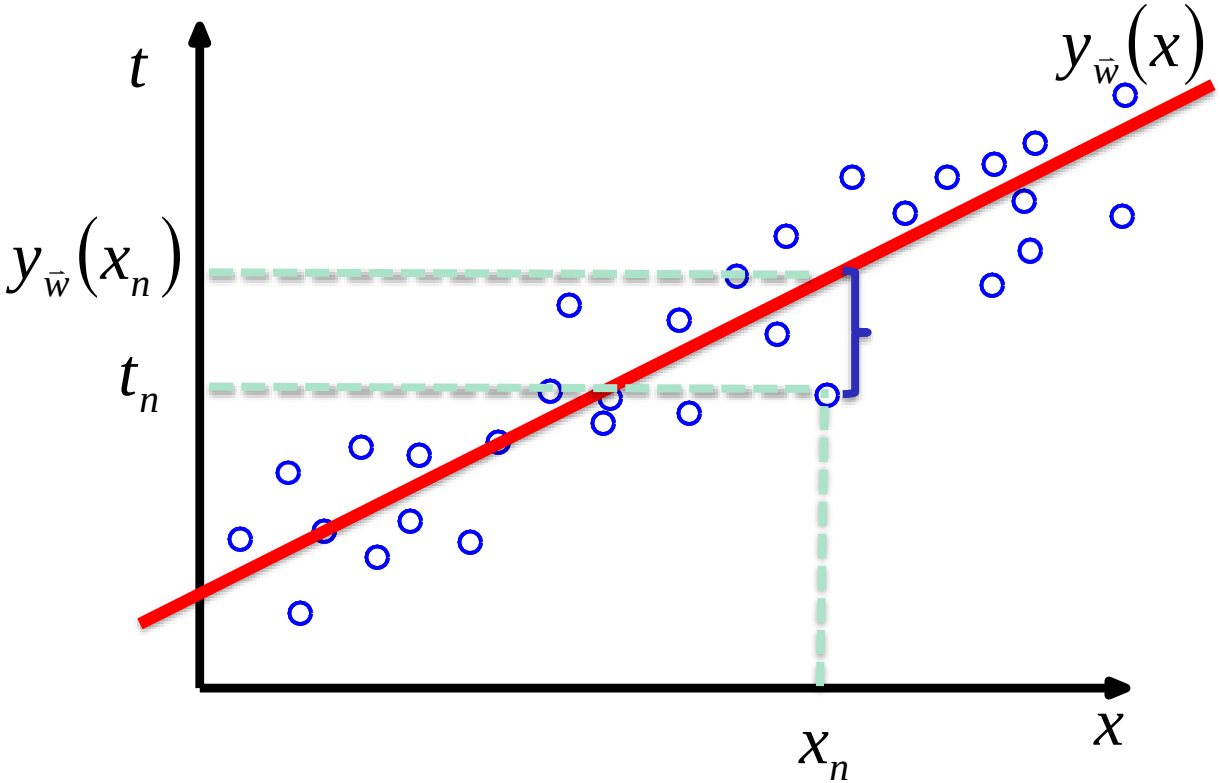
$$D = \{(x_1, t_1), (x_2, t_2), \ldots, (x_N, t_N)\}$$

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



Problem to solve

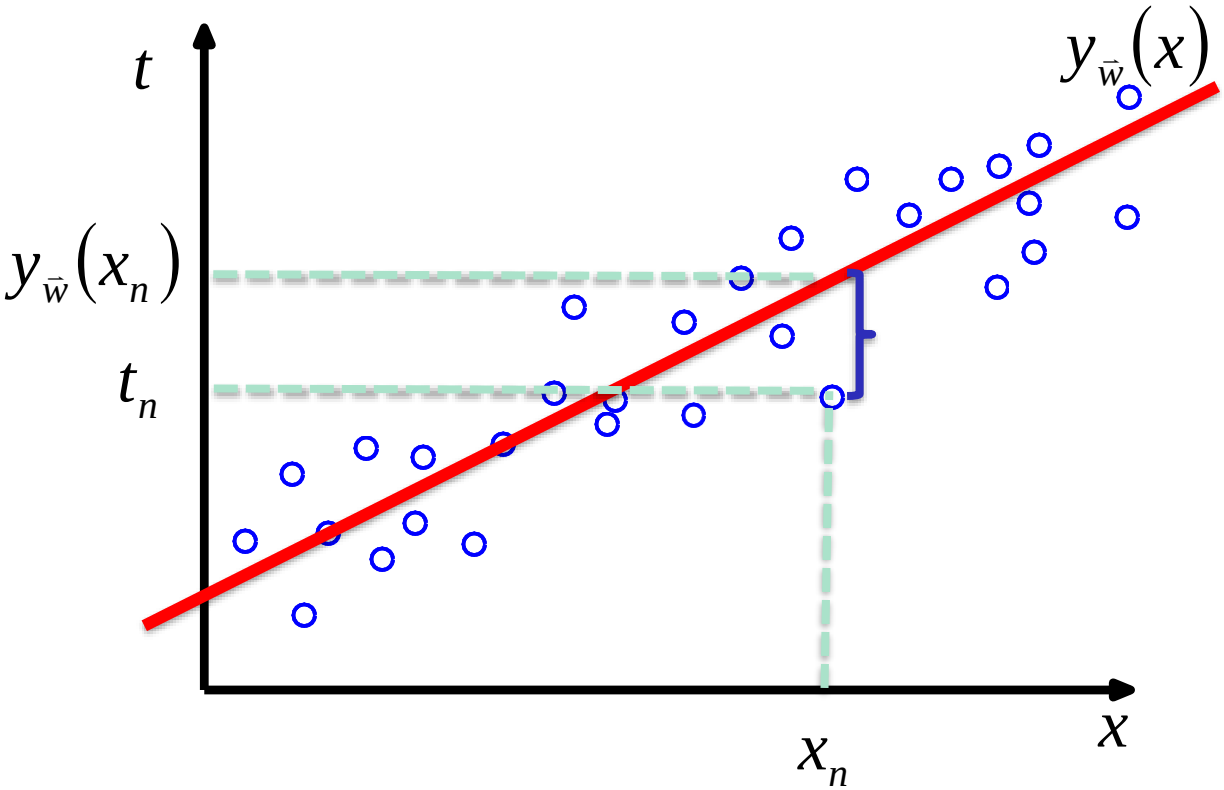
Unfortunately, real data are **noisy**



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

Problem to solve

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



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 **<u>minimizing the function</u>**

Problem to solve

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

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

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

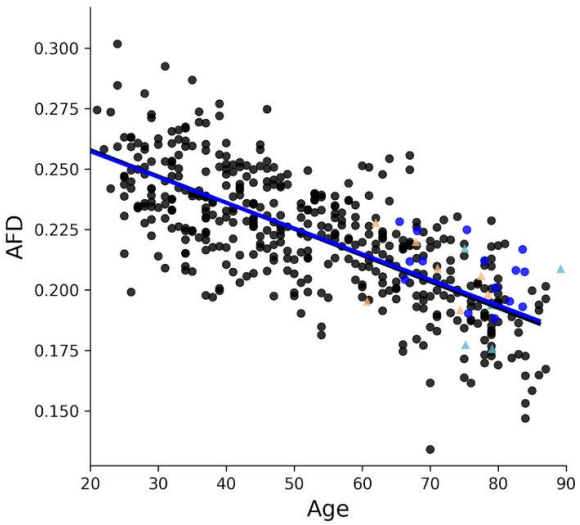
By **isolating** $\vec{w}$**,** we get

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

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



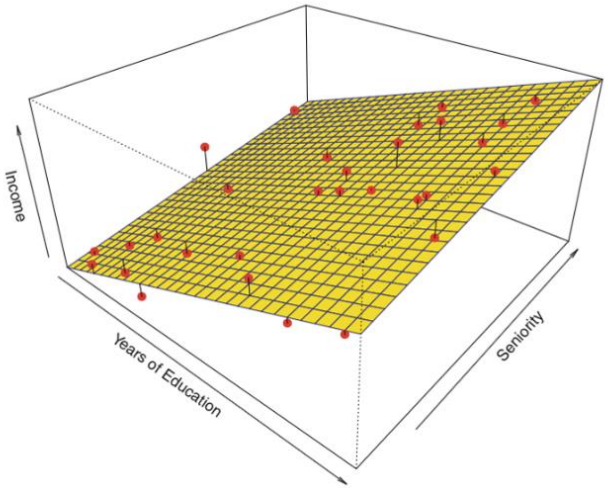
$$y_{\vec{w}}(x) = w_0 + w_1 x$$

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

where

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

For a 2D regression



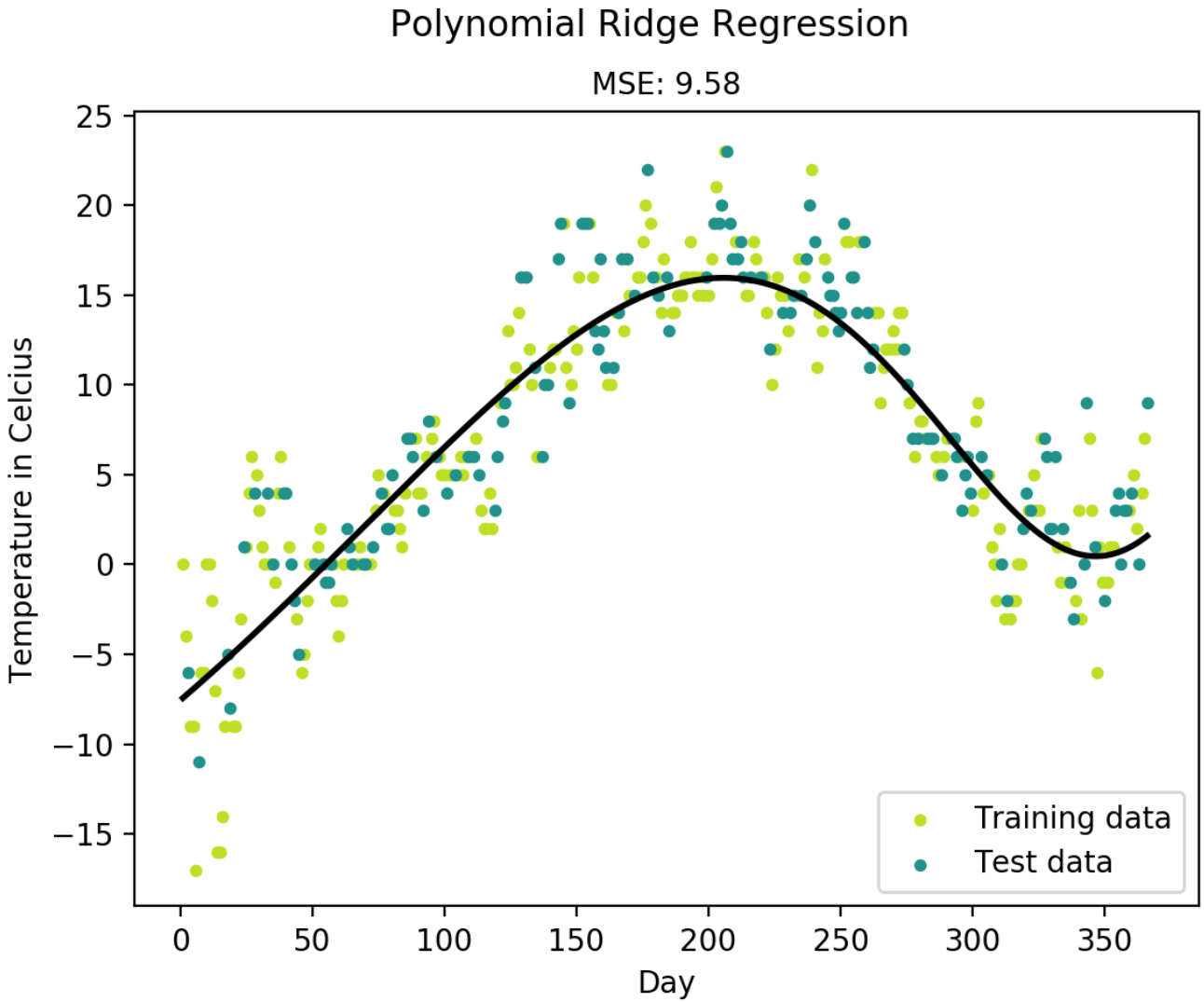
$$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2$$

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

where

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

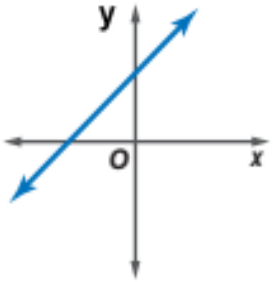
What about non-linear data?



What about non-linear data?

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

Linear function
Degree 1

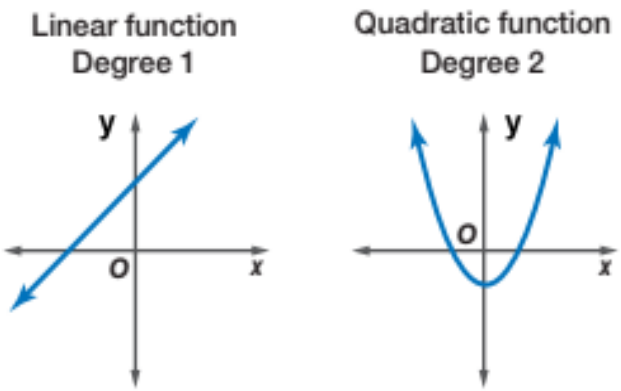


http://www.math.glencoe.com/

What about non-linear data?

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

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

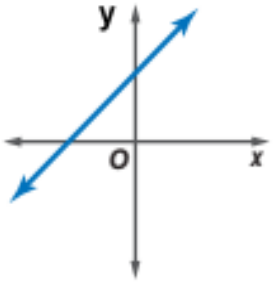


http://www.math.glencoe.com/

What about non-linear data?

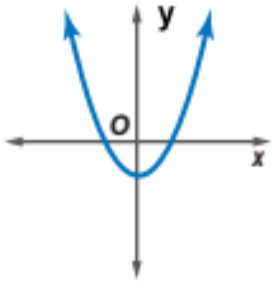
$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x$

Linear function
Degree 1

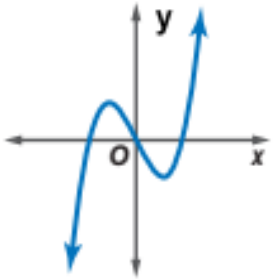


$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x + w_1 x^2$

Quadratic function
Degree 2



Cubic function
Degree 3



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

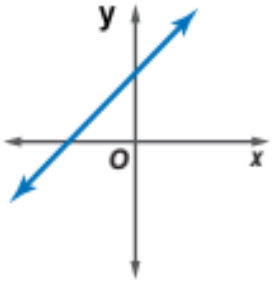
$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x + w_1 x^2 + w_1 x^3$

What about non-linear data?

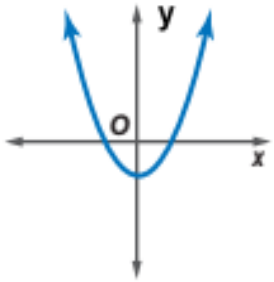
$y_{\vec{w}}(\vec{x}) = w_0 + w_1 x$

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

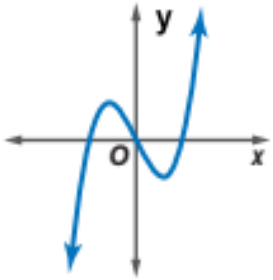
Linear function
Degree 1



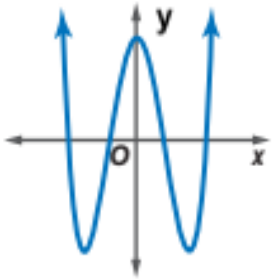
Quadratic function
Degree 2



Cubic function
Degree 3



Quartic function
Degree 4



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

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

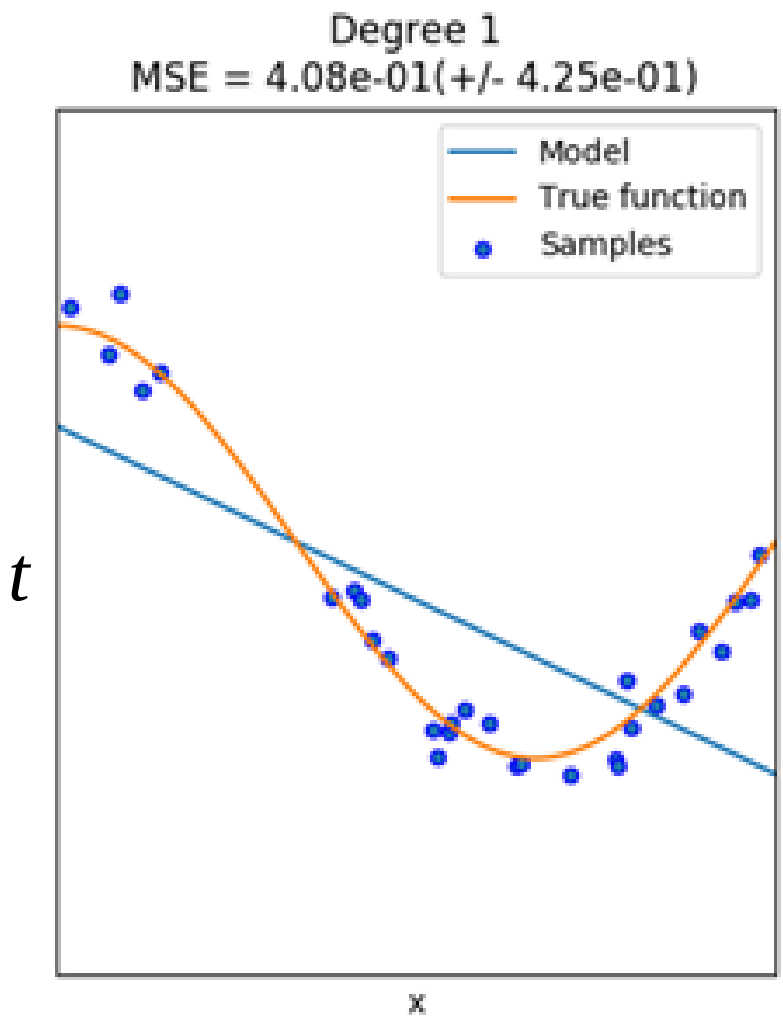
Example: Instead of a **1D regression**, lets do a **4D regression**

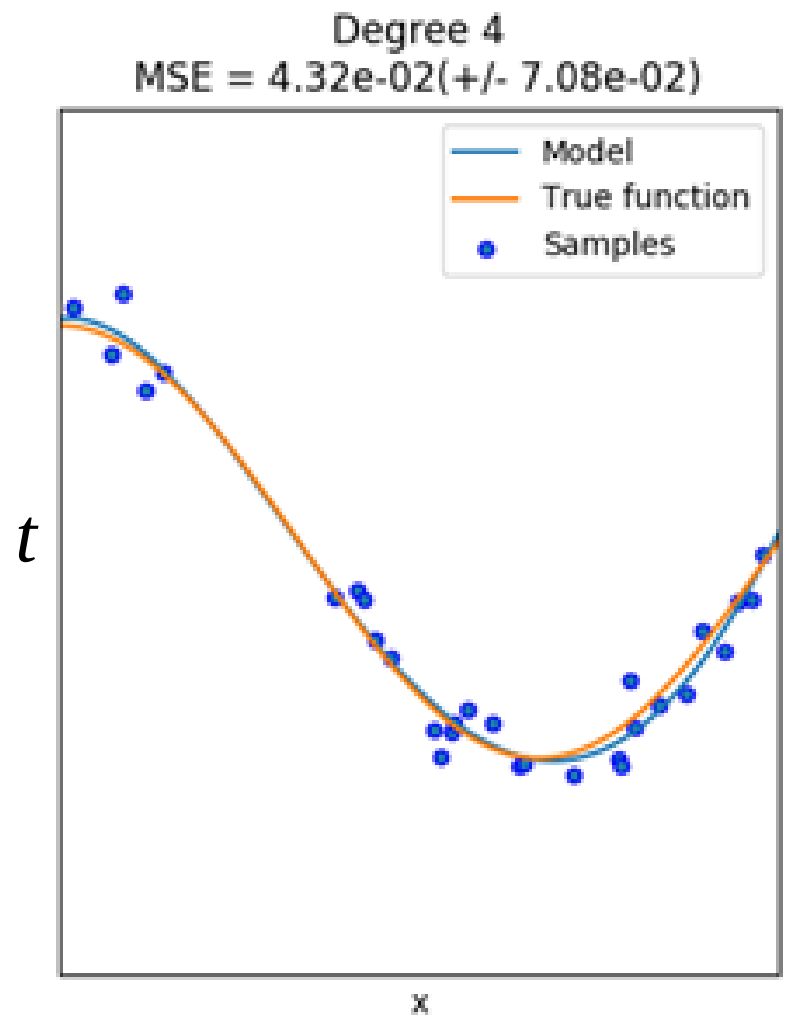
$$\varphi(x) \rightarrow (x, x^2, x^3, x^4)$$

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

$$y_{\vec{w}}(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$$

$$= w_0 + \sum_{i=1}^{4} w_i \varphi_i(x)$$





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

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

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

Basis function

hyperparameter

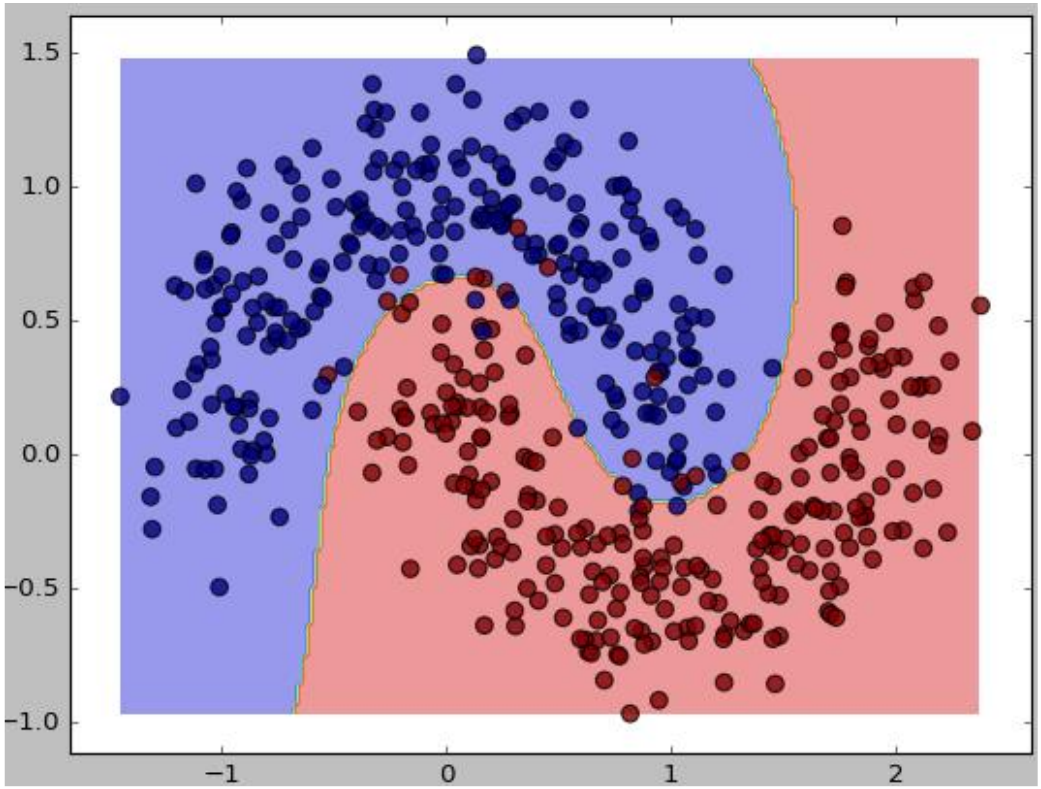
parameter

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

Basis function

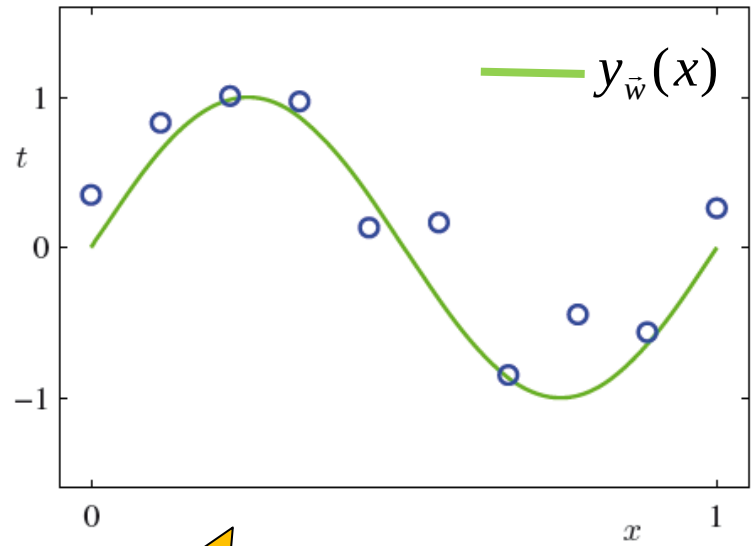
where $\varphi_0(\vec{x}) = 1$

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

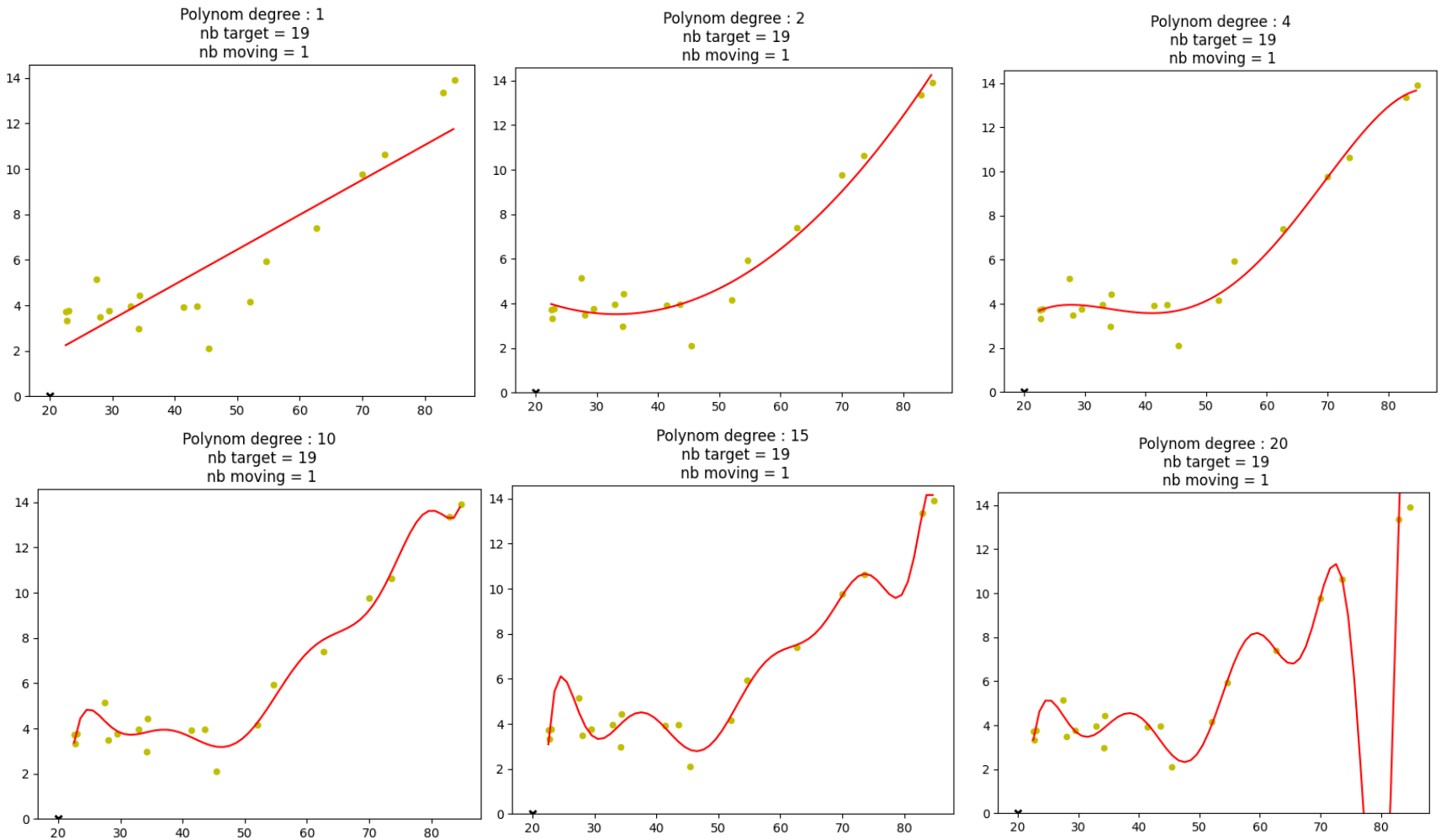
$$\vec{w} \in R^d$$

Parameters

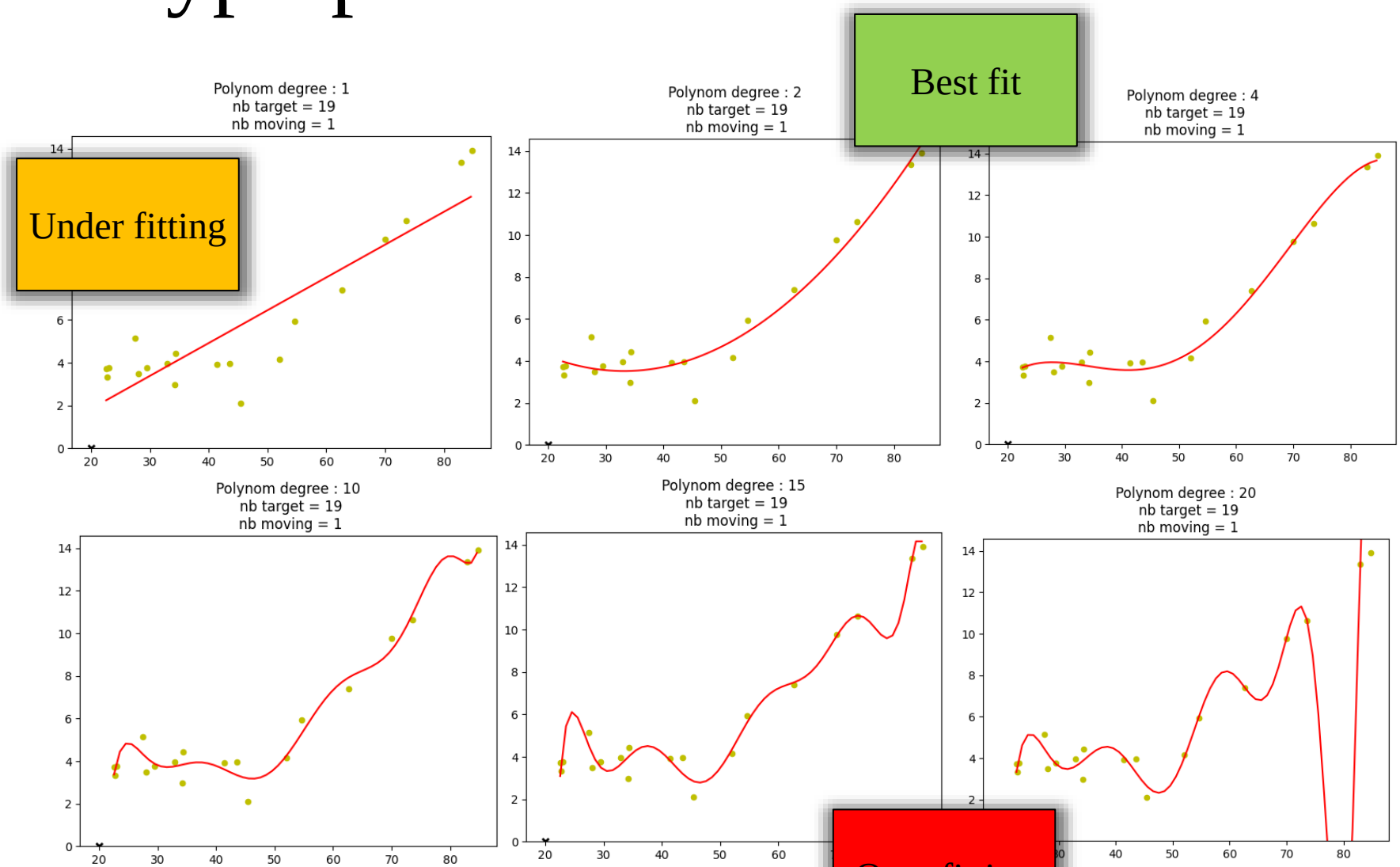
$$d \in N^{\geq 0}$$

Hyper-parameters

Hyperparameter *d*



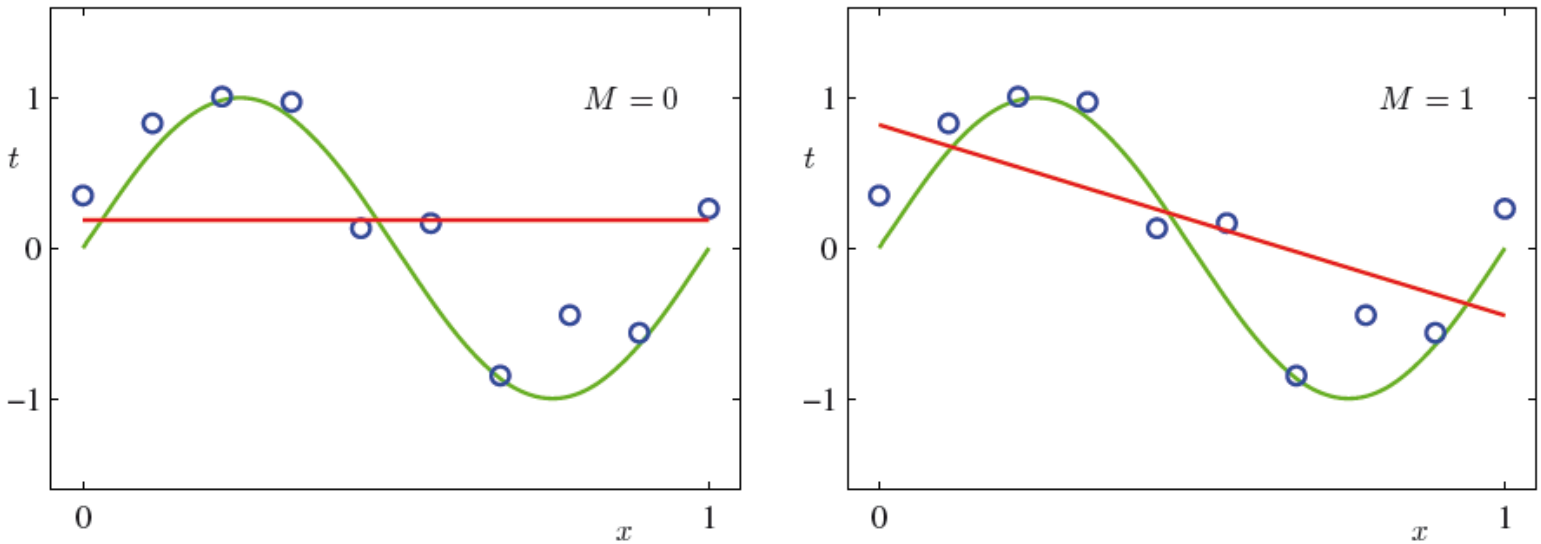
Hyperparameter *d*



Underfitting

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

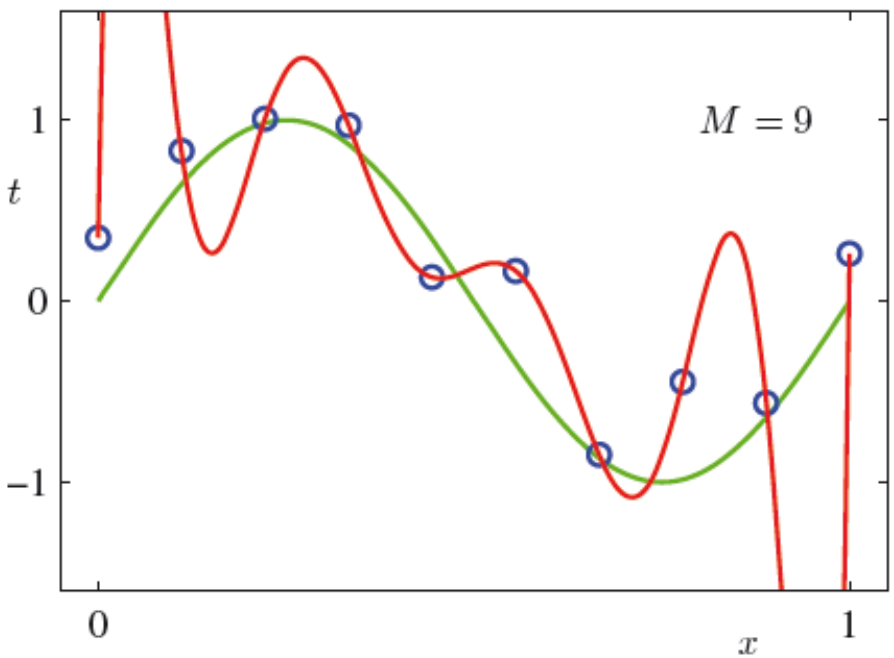
A small *d* gives a simplistic model that **underfits** the data**.**



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

Overfitting

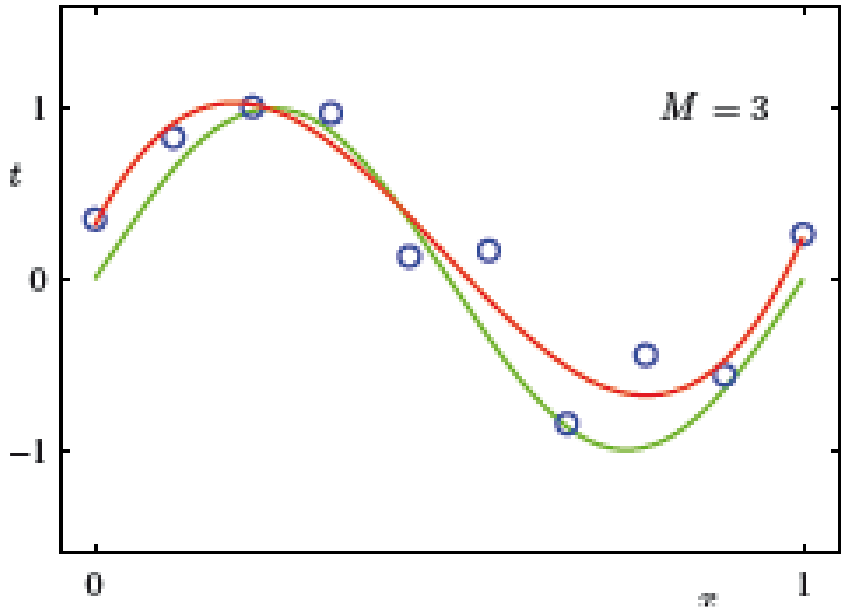
A large *d* gives a model that « **learn by heart** » and thus **overfits** training data



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

Over- and underfitting

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

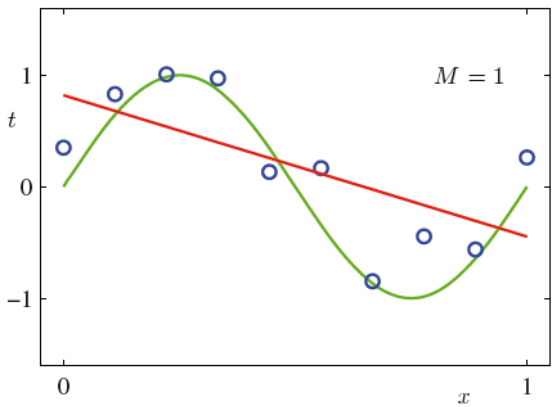


$$\mathcal{L}_D(\vec{w}) \Rightarrow \text{ low}$$
$$\mathcal{L}_{D_{test}}(\vec{w}) \Rightarrow \text{low}$$

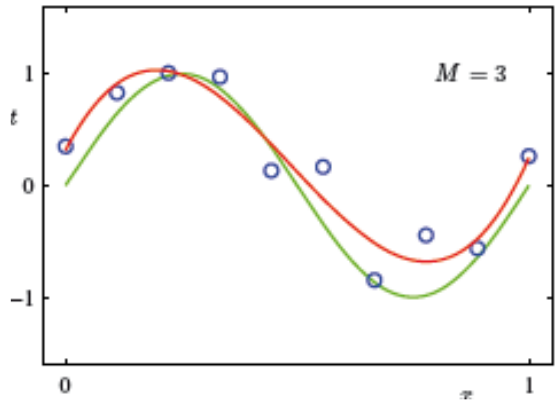
Hyperparameters often control the **capacity** of a model

Capacity: ability of a model to fit the training data

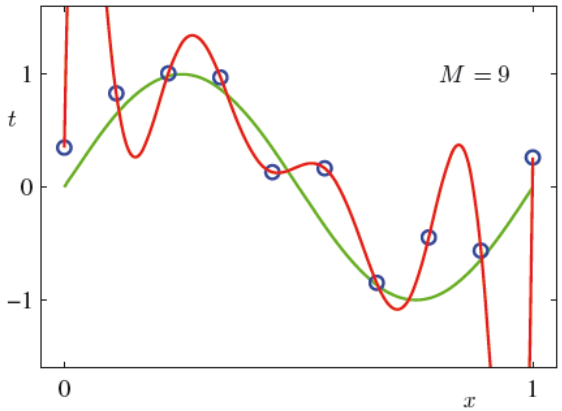
Low capacity



Medium capacity



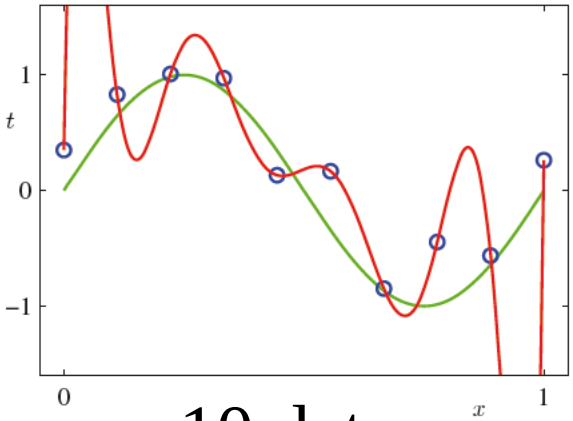
Large capacity



Generalization

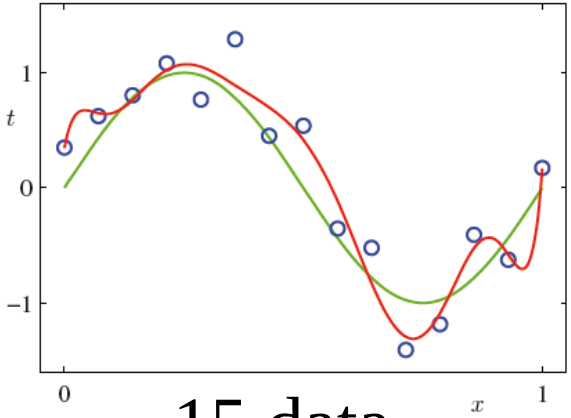
The more data you have, the better a high capacity model will generalize.

$d = 9$



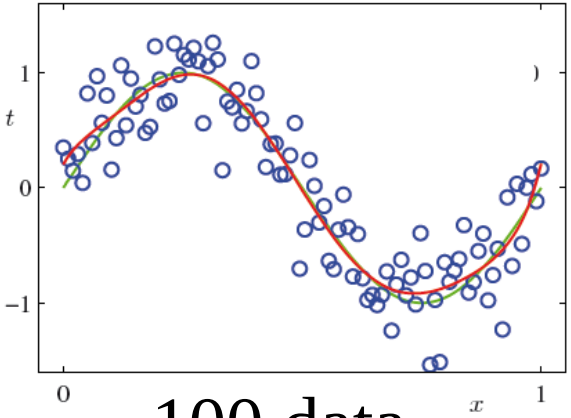
10 data

$d = 9$



15 data

$d = 9$



100 data

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

Regularization

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

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
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

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

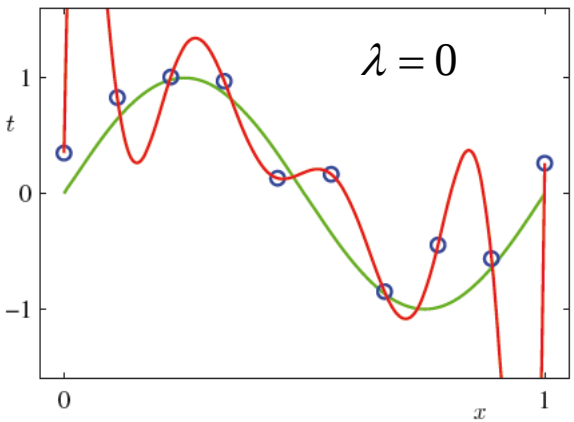
$$\|\vec{w}\|^2 = \vec{w}^{\mathrm{T}}\vec{w} = w_0^2 + w_1^2 + \ldots + w_d^2$$

Ridge model

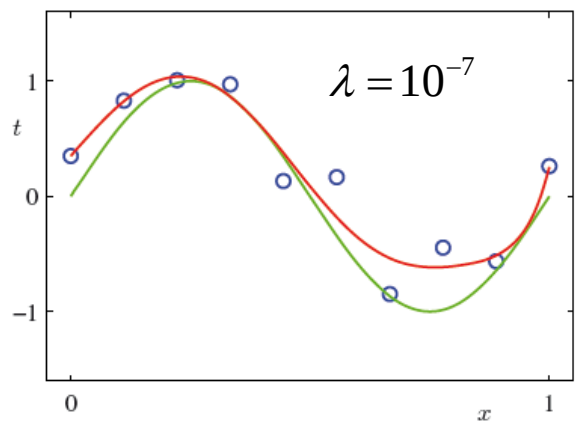
Regularization

Strong regularization= less capacity

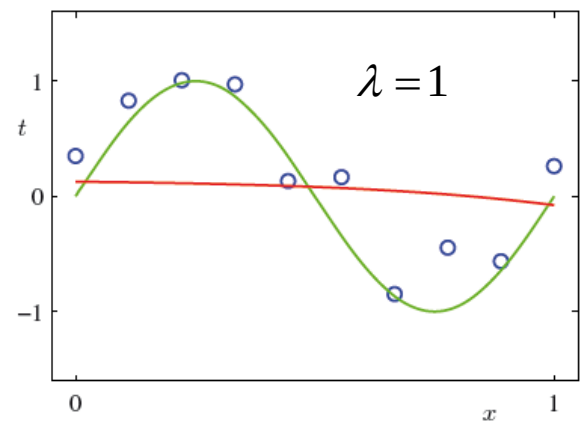
$d = 9$



$d = 9$

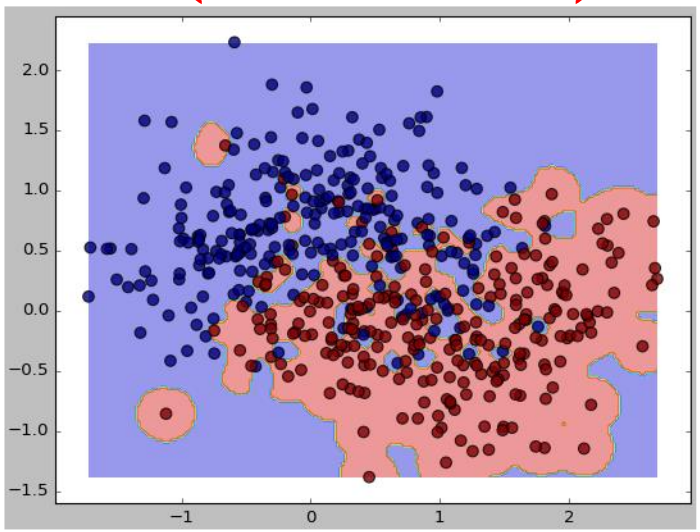


$d = 9$

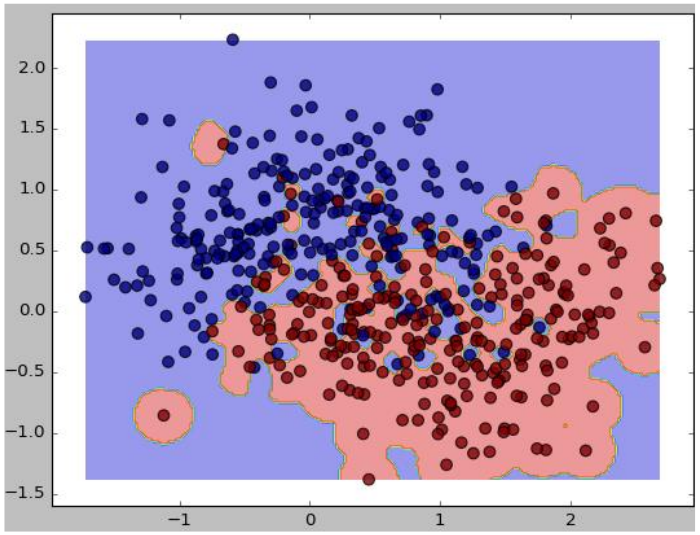


Over- and under-fitting also influence classification

Overfitting (Classification)

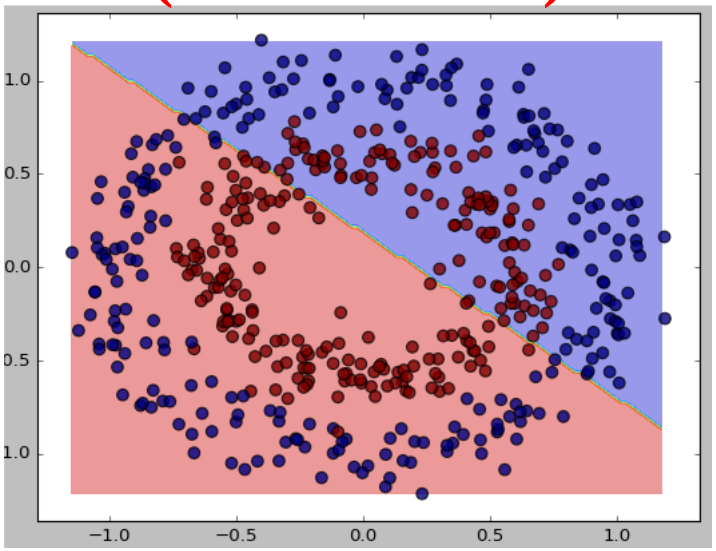


Training accuracy = 99.6%

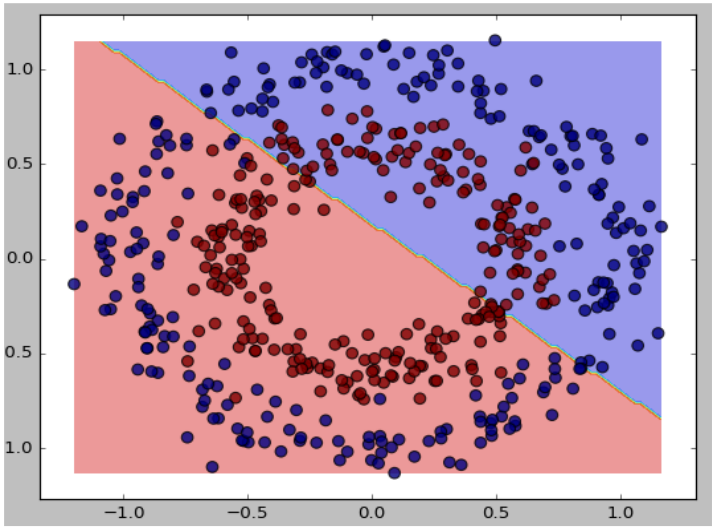


Testing accuracy = 78%

Underfitting (Classification)

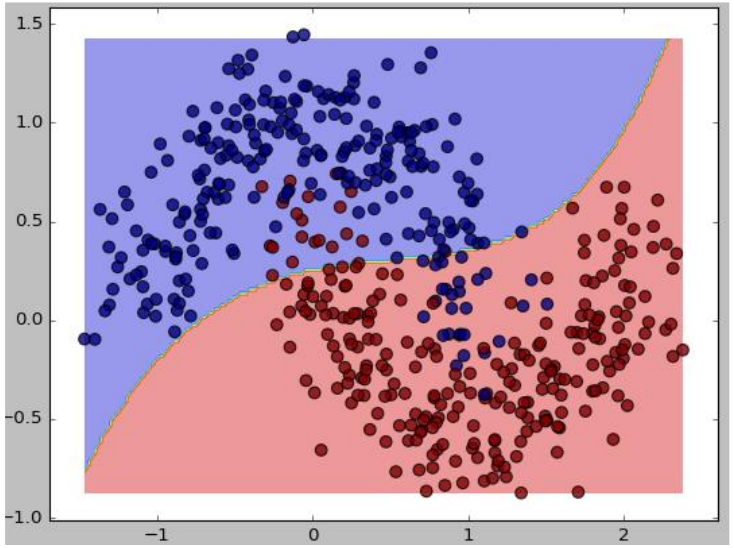


Training accuracy =52.2%

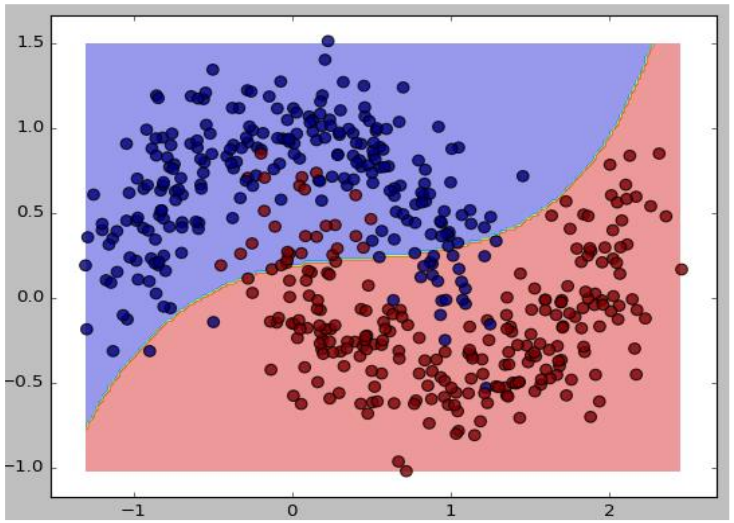


Testing accuracy = 51.2%

Could be better…

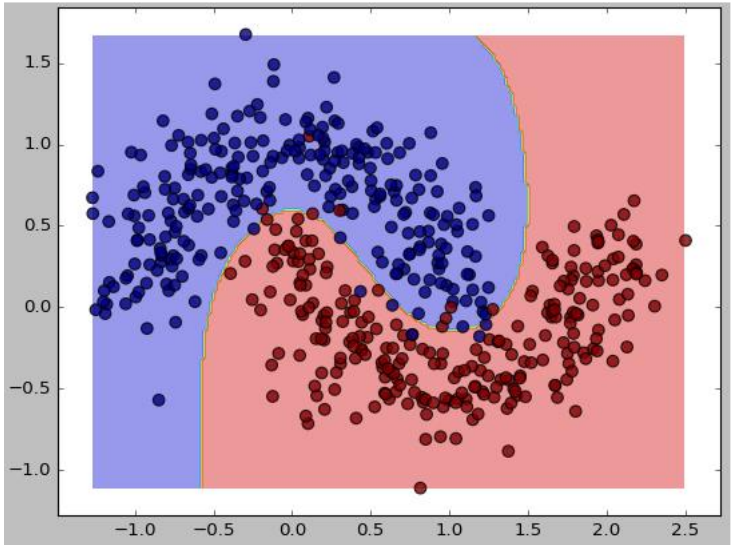


Training accuracy =82%

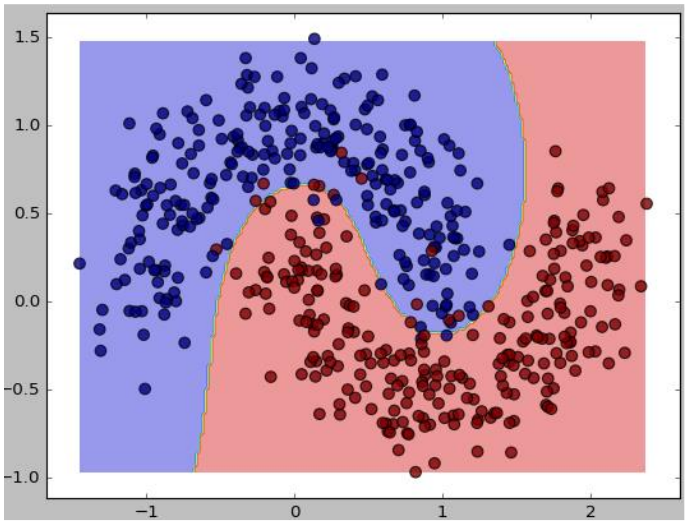


Testing accuracy = 80%

Wonderful !!!



Training accuracy =97.8%



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

<u>How to find the right hyper-parameters?</u>

d and λ

How to find the right d and the right λ ?

- **Very bad idea** : choose randomly

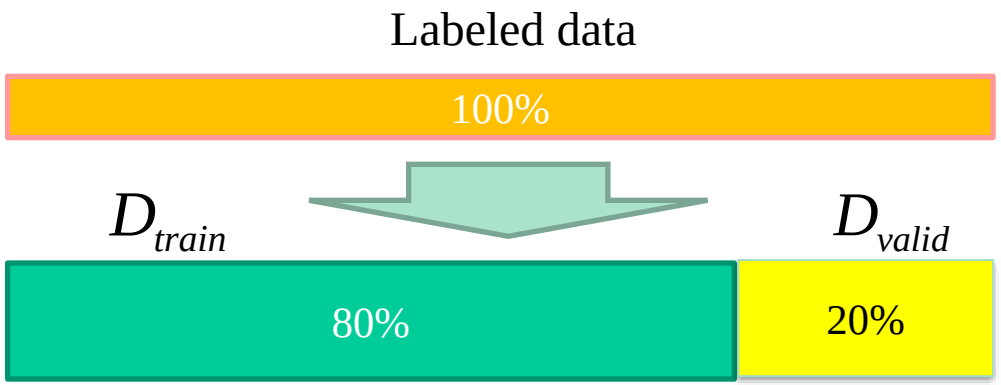
- **Bad idea** : take many (d, λ) and keep the one with the lowest training error
  - ➢ overfitting

- **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

- **Good solution** : take many (d, λ) and keep the one with the lowest **validation error**

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!

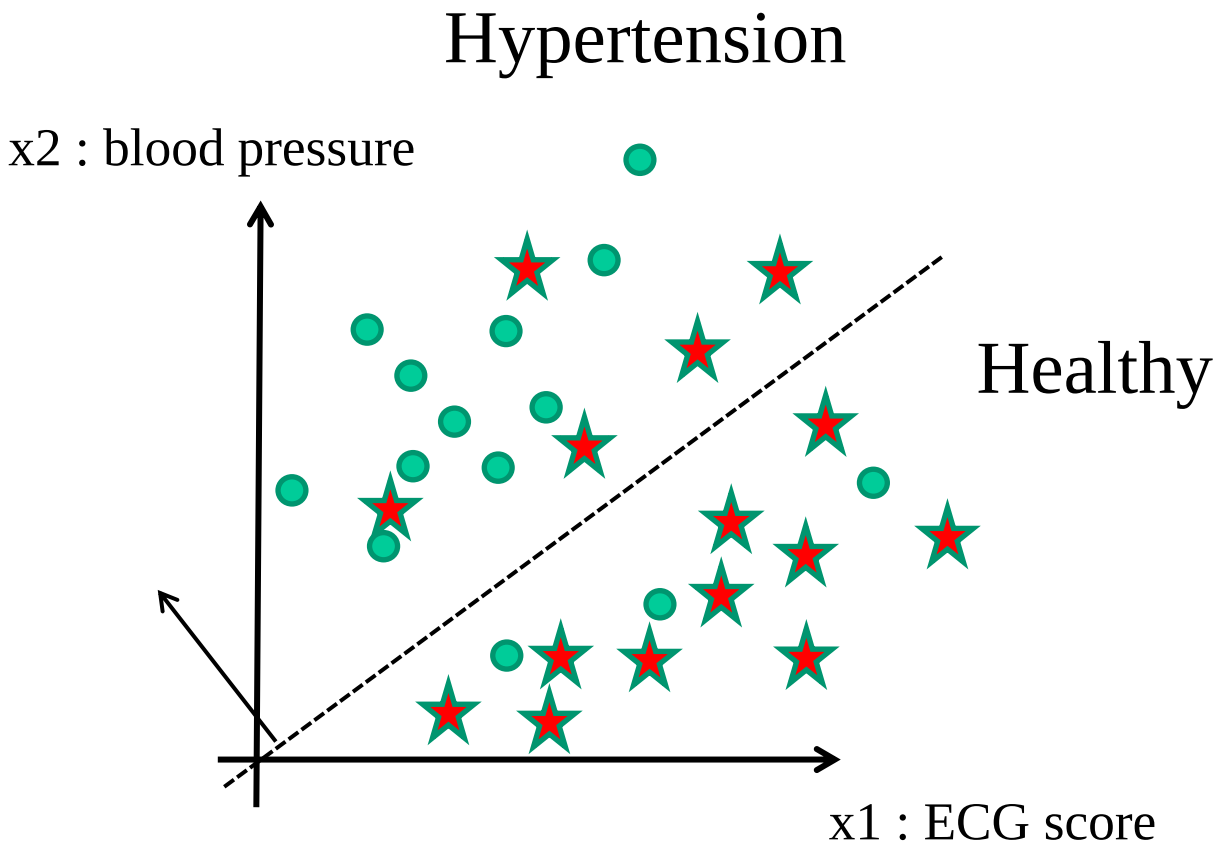
$d=5,$
$\lambda = 1$

In short

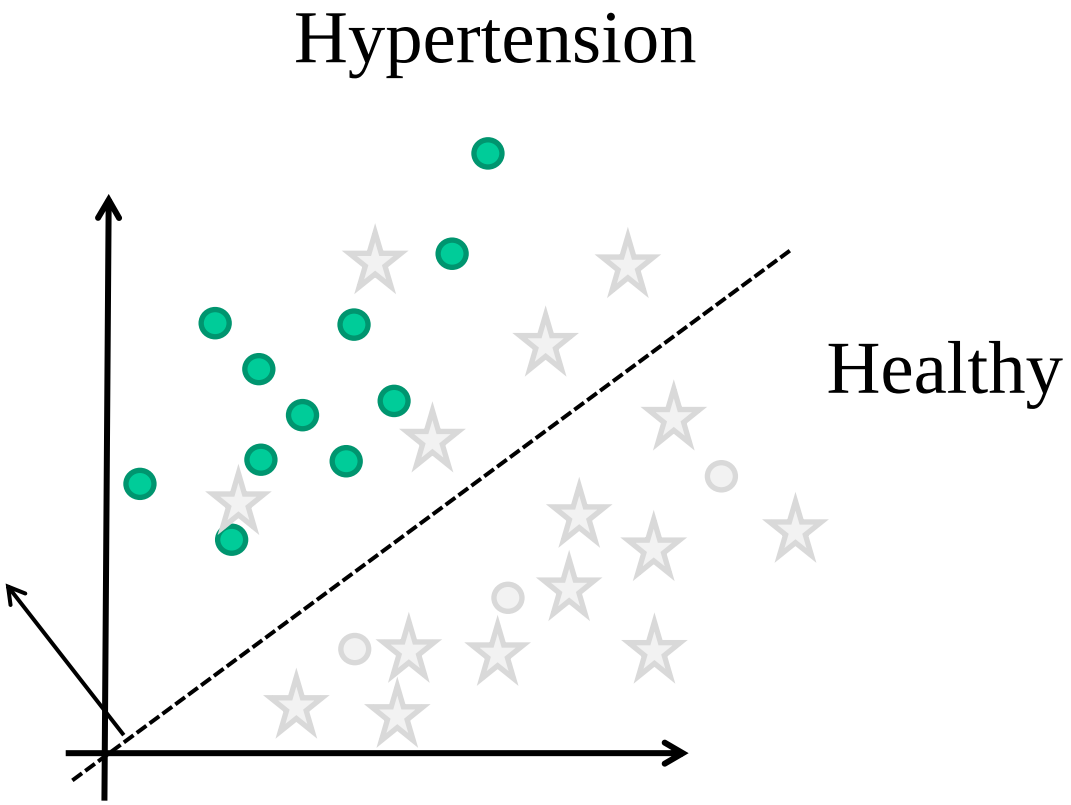
- ✓ The goal is to **train** a **model** on a **training dataset** with good **generalization** capabilites
- ✓ Training = minimization of a **loss function**
- ✓ Has **hyper-parameters** that control the **capacity** of the model, choisis à l'aide d'une procédure de **sélection de modèle**
- ✓ mesure 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)

Evaluation metrics

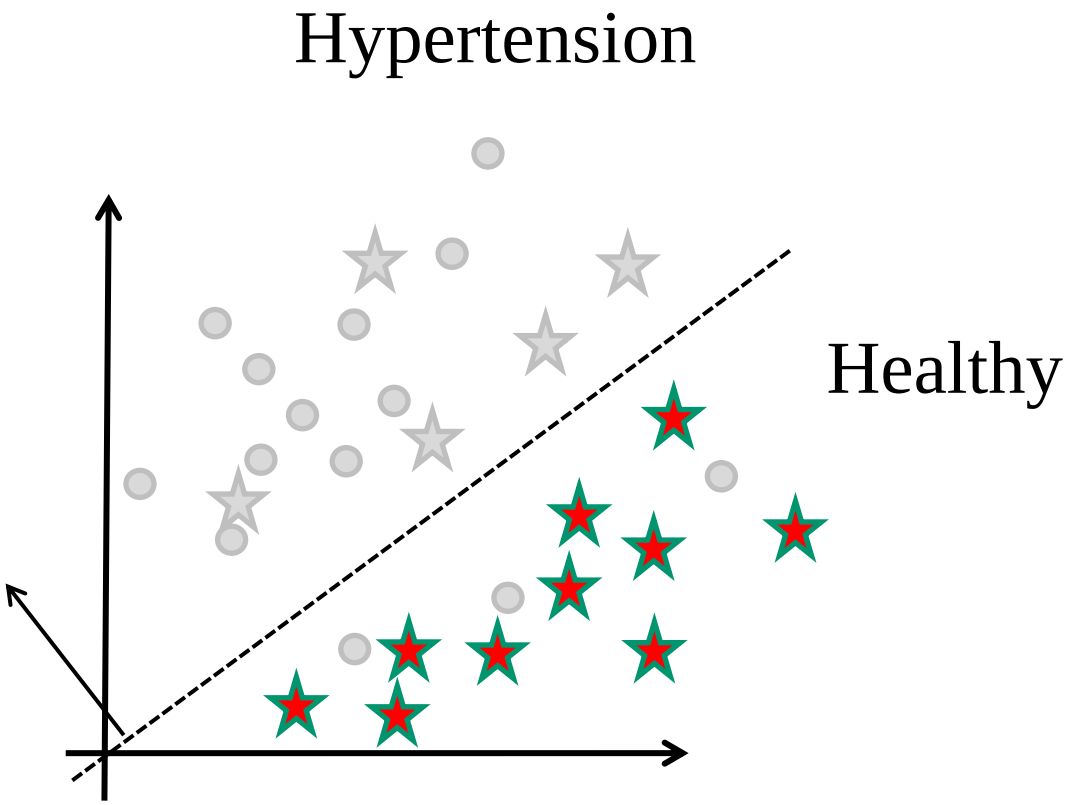
How to evaluate a model?



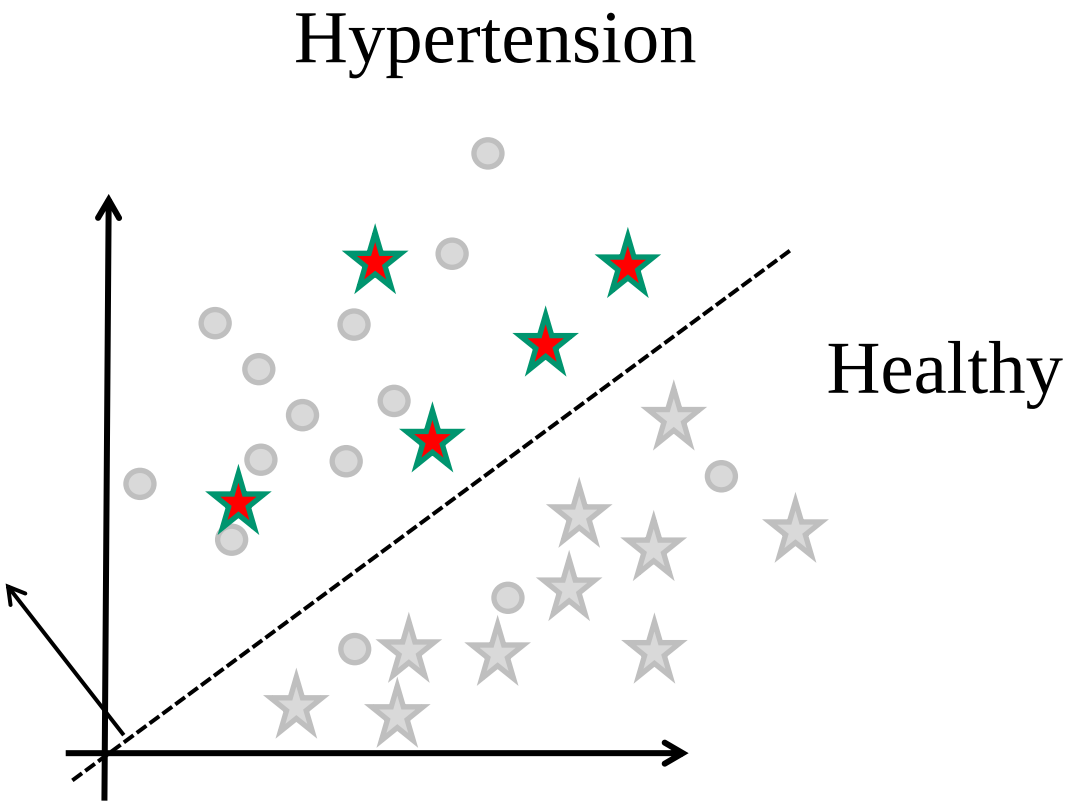
True positive (11)



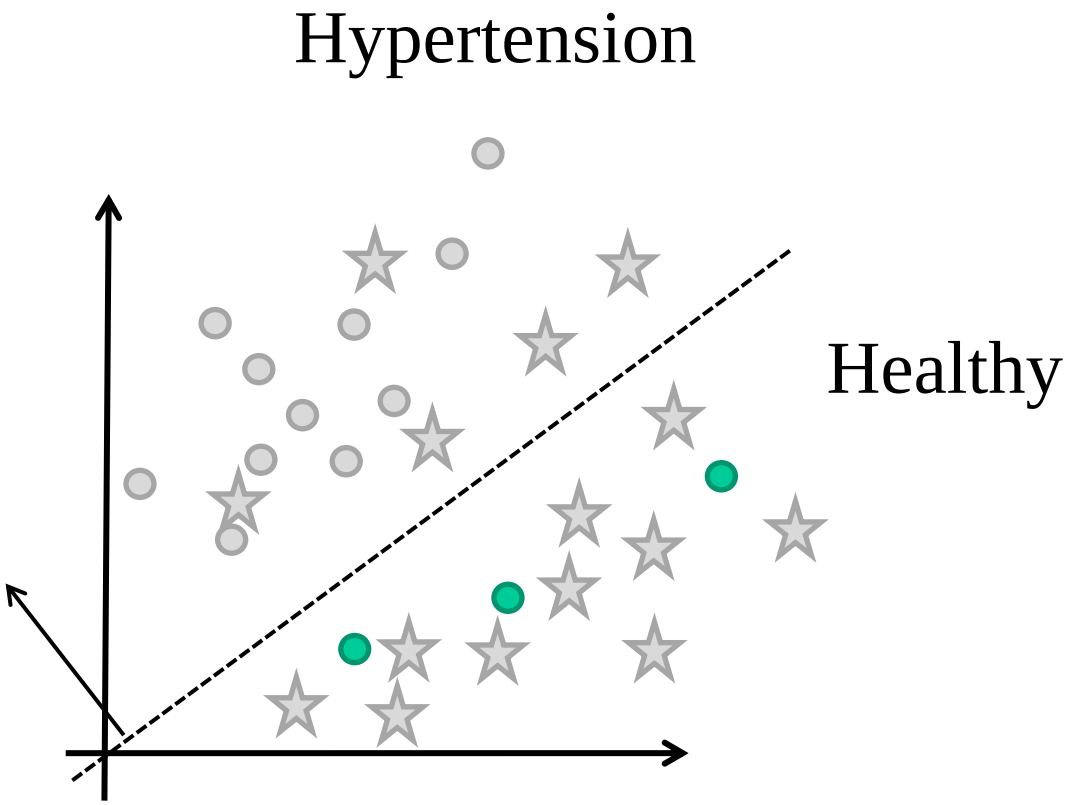
True negative (10)



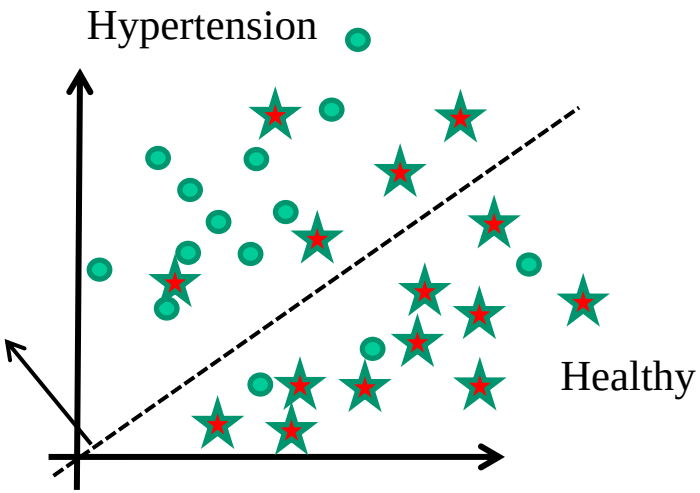
False positive (5)



False negative (3)



Confusion matrix



Ground truth

Model prediction	Positive	Negative
Positive	TP = 11	FP=5
Negative	FN = 3	TN=10

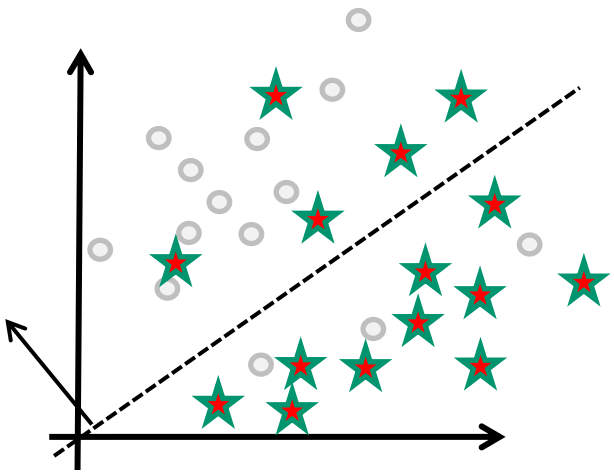
TP + FN = 14 = TOTAL # positive
TN + FP = 15 = TOTAL # negative

TP + FP = 16 = TOTAL # of patients classified +1
FN + TN = 13= TOTAL # of patients classified -1

False positive rate

TP = 11	FP=5
FN = 3	TN=10

Hypertension



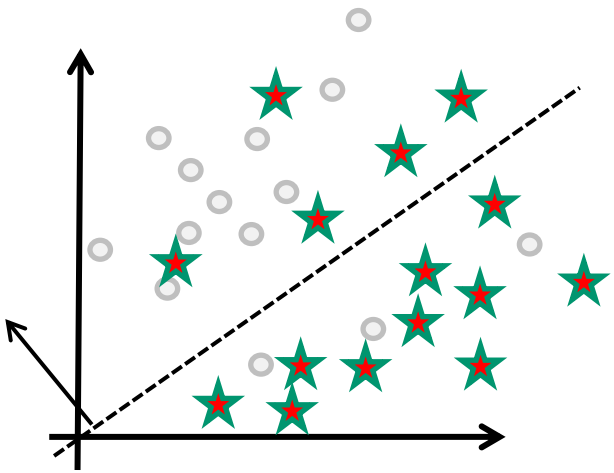
Healthy

$$FPR = FP/(FP+TN) = 5/15$$

Specificity
(true negative rate)

TP = 11	FP=5
FN = 3	TN=10

Hypertension

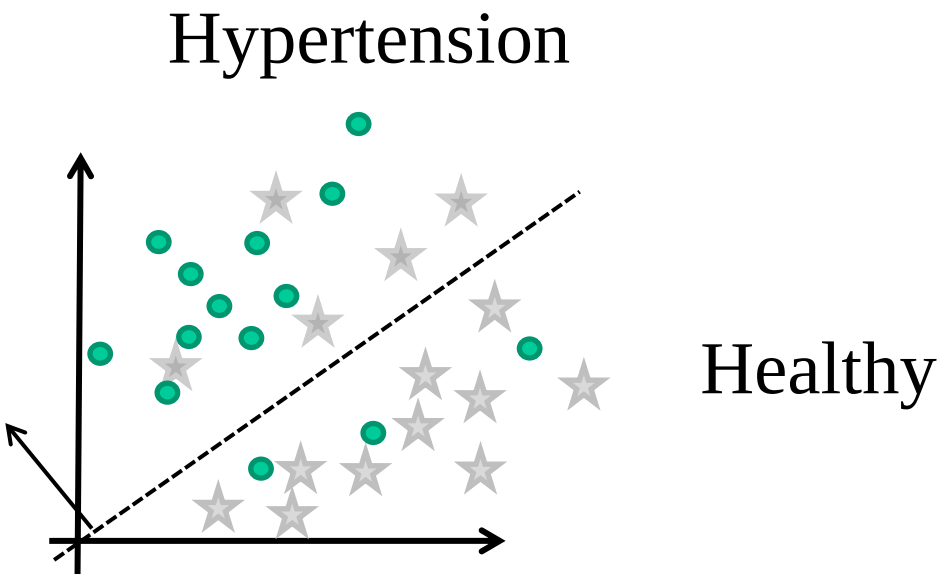


Healthy

Sp = TN/(FP+TN)=11/15

False negative rate

TP = 11	FP=5
FN = 3	TN=10

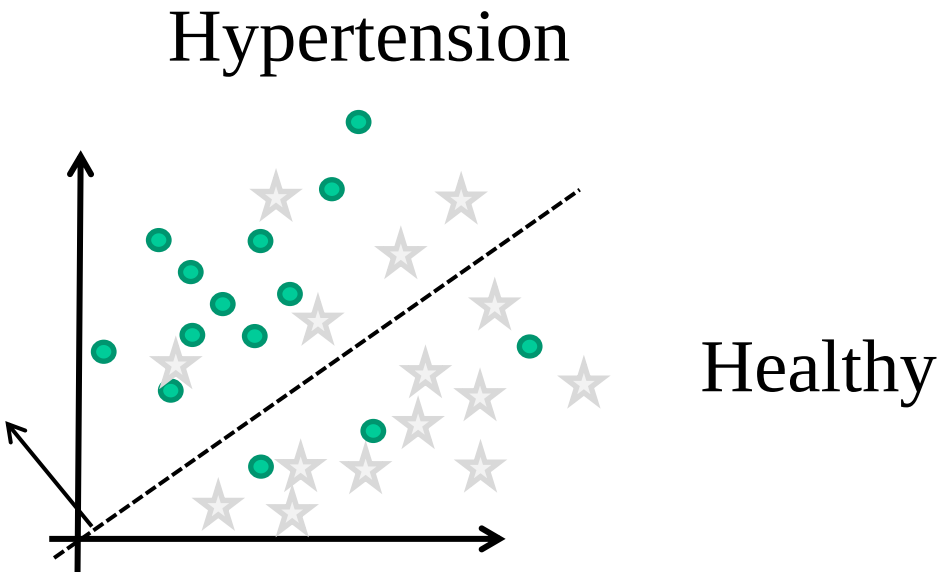


FNR = FN/(FN+TP) = 3/14

Recall

(True positive rate)

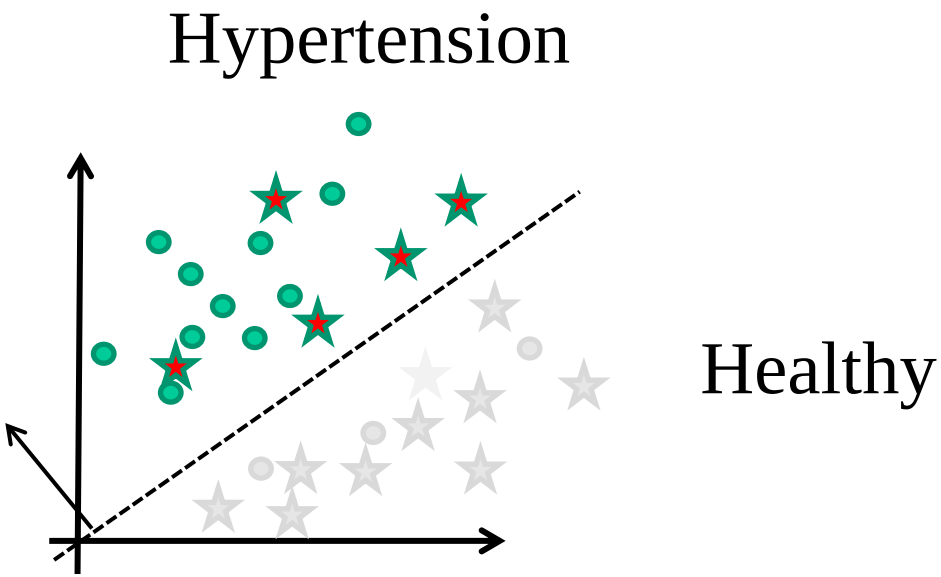
TP = 11	FP=5
FN = 3	TN=10



Re = TP/(FN+TP)=1-FNR=11/14

Precision

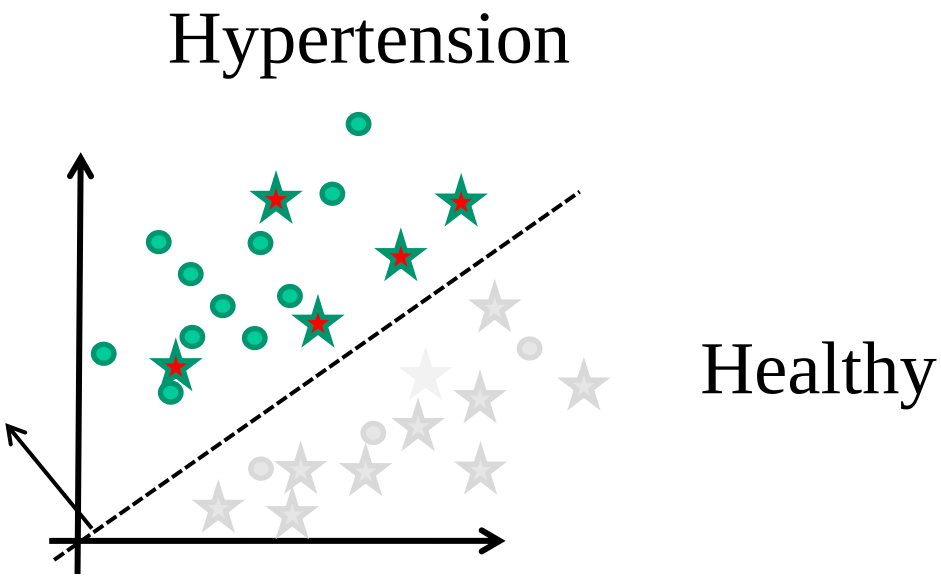
TP = 11	FP=5
FN = 3	TN=10



Pr = TP/(TP+FP) = 11/16

False Discovery Rate

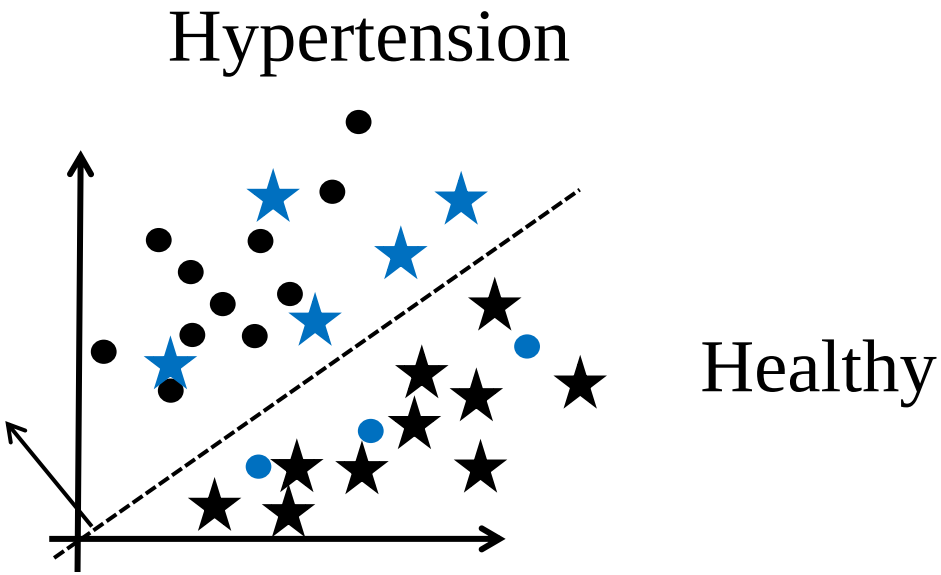
TP = 11	FP=5
FN = 3	TN=10



FDR = FP/(TP+FP) = 5/16

Accuracy

TP = 11	FP=5
FN = 3	TN=10



Rate of good classification = (TP+TN)/(FP+FN+TP+TN) = 21/29

In short

		Ground truth: Positive	Ground truth: Negative
Model prediction	Positive	TP = 11	FP=5
	Negative	FN=3	TN=10

TN + FP = 15 = TOTAL # negative
TP + FN = 14 = TOTAL # positive

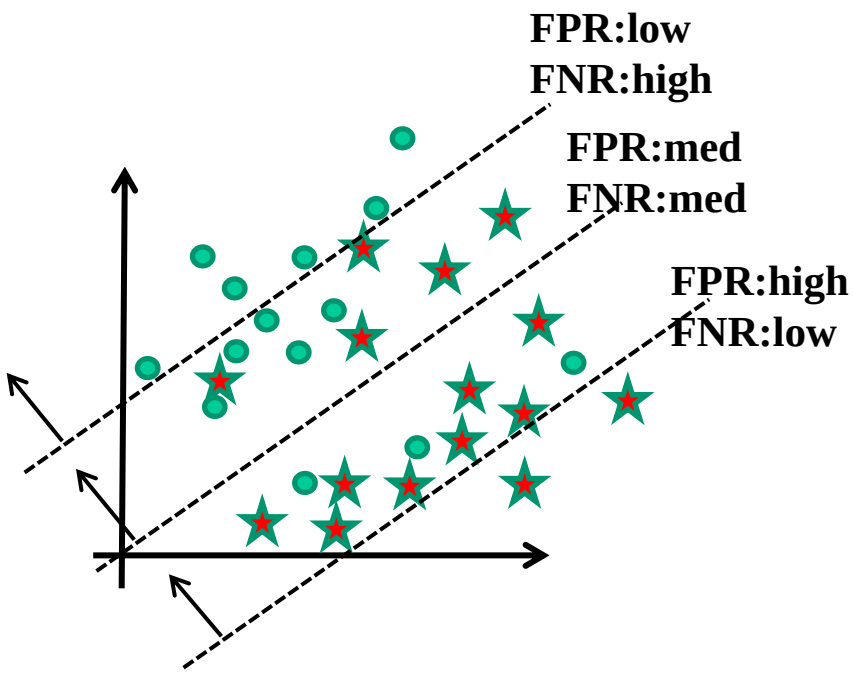
TP + FP = 16 = TOTAL # of patients classified +1
FN + TN = 13= TOTAL # of patients classified -1

False positive rate = FP/(FP+TN) = 5/15
False negative rate = FN/(FN+TP) = 3/14

Specificity (**Sp**) = TN/(FP+TN)=1-FPR=10/15
Recall (**Re**) = TP/(TP+TN)=11/14
Precision (**Pr**) = TP/(TP+FP) = 11/16

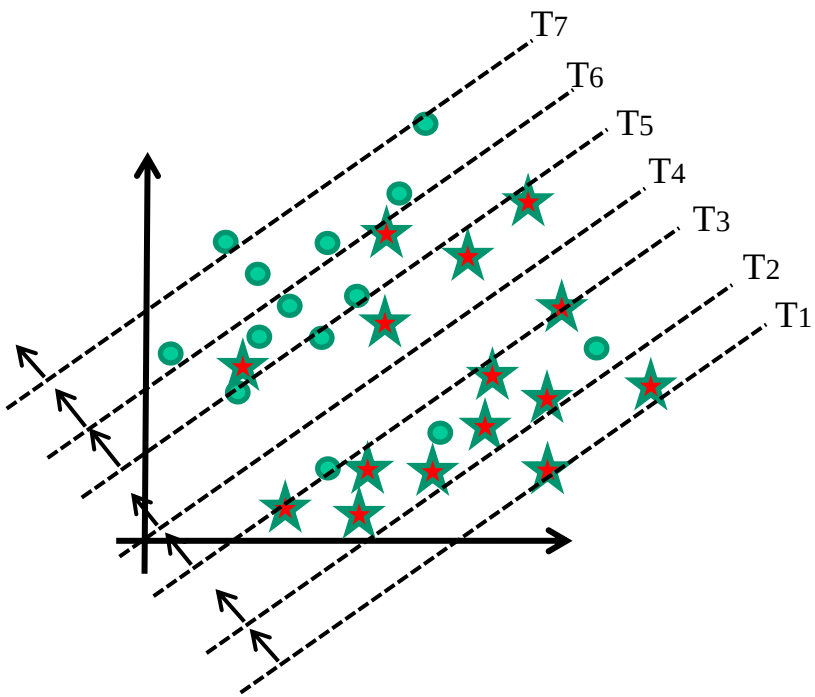
Accuracy = (TP+TN)/(FP+FN+TP+TN) = 21/29
F-measure = 2* (Re*Pr)/(Pr+Re)=0.73

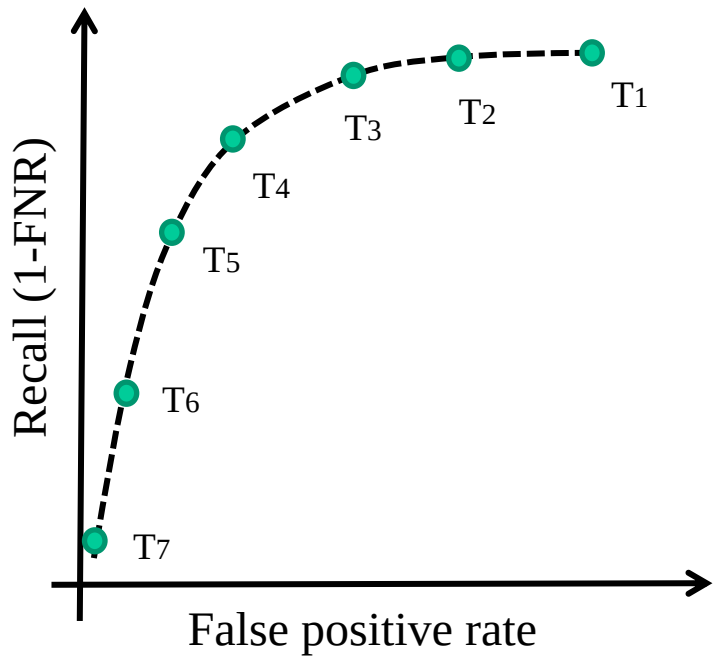
Different thresholds, different results



ROC curves

Compute Recall and FPR for **different thresholds**

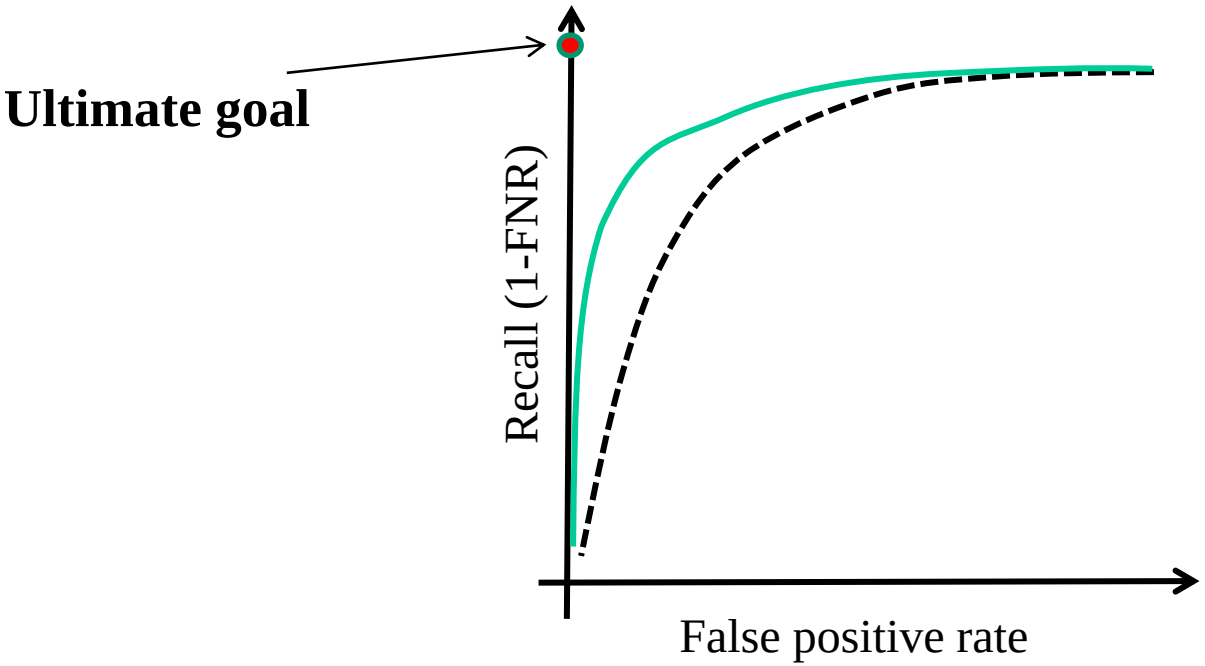




ROC curves

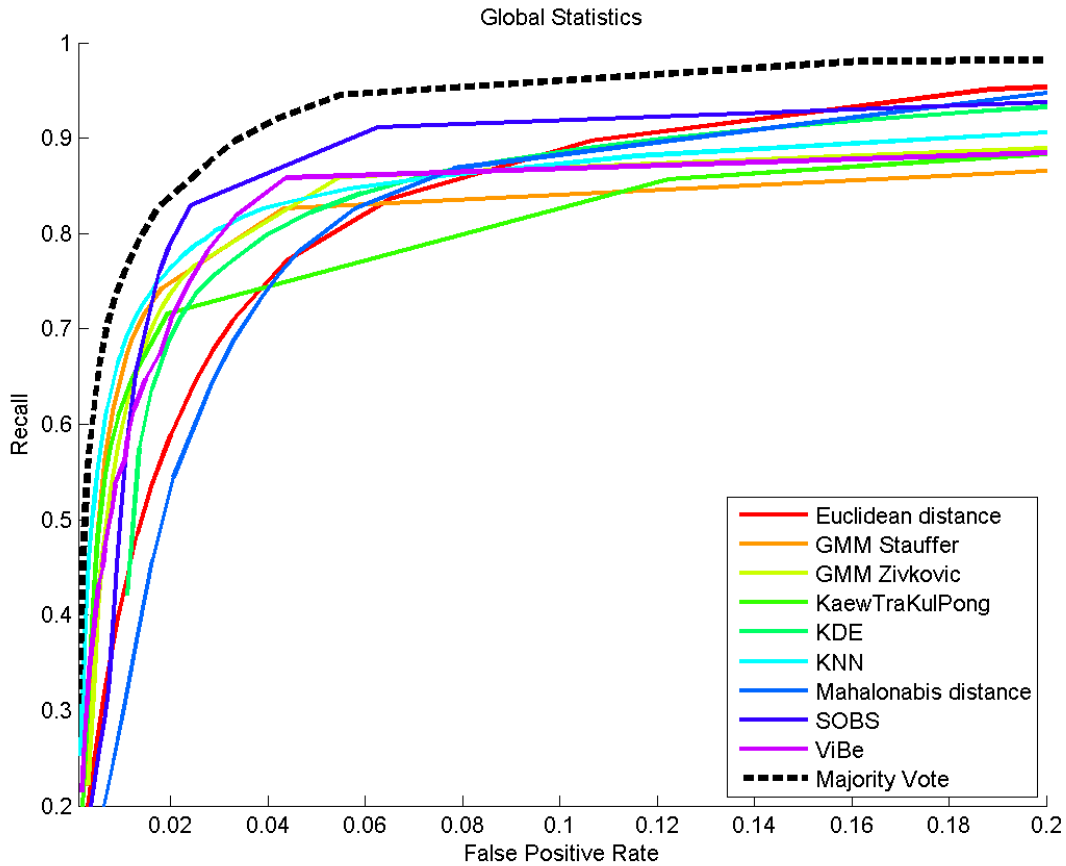
Good way for comparing methods

Which method is best?



ROC curves

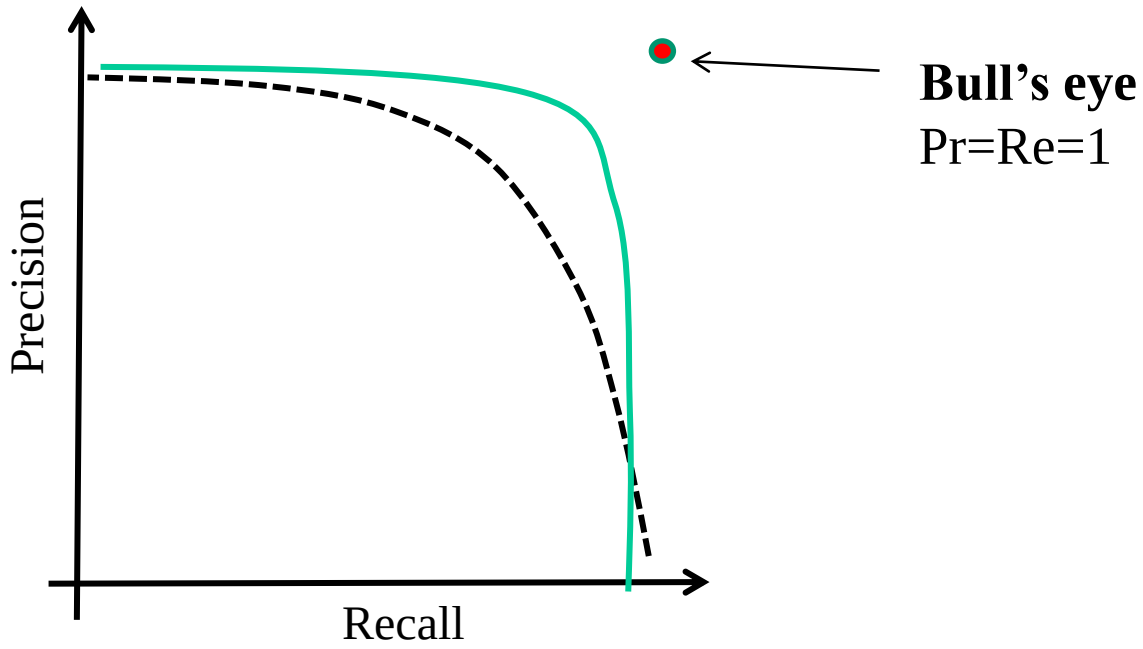
Example : 10 motion detection methods



Precision recall curve

Sampe spirit that the ROC curve

Which one is best?



Segmentation metrics

Sørensen–Dice coefficient

		Ground truth	
		Positive	Negative
Model prediction	Positive	TP = 11	FP=5
	Negative	FN=3	TN=10

$$\text{Dice} = 2TP/(TP+FP+FN)$$
$$= 2*11/(2*11+5+10)$$
$$= 0.59$$

Segmentation metrics

Sørensen–Dice coefficient

Useful when TN is very large

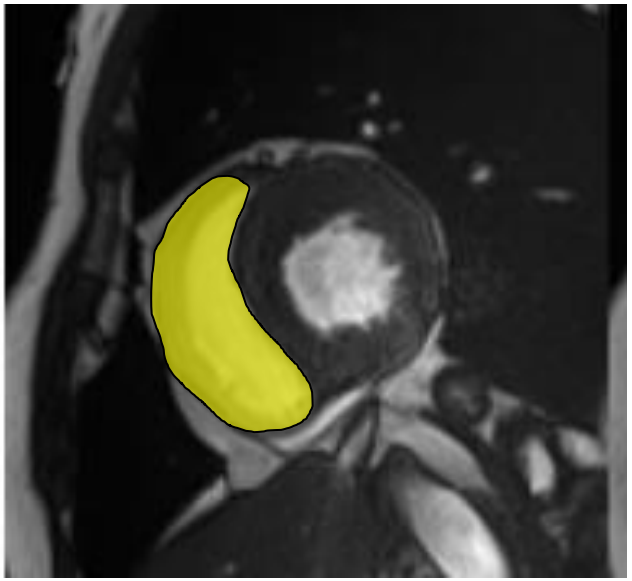
		Ground truth	
		Positive	Negative
Model prediction	Positive	TP = 11	FP=5
	Negative	FN=3	**TN=10,0000**

$$\text{Dice} = 2TP/(TP+FP+FN)$$
$$= 2*11/(2*11+5+10)$$
$$= 0.59$$

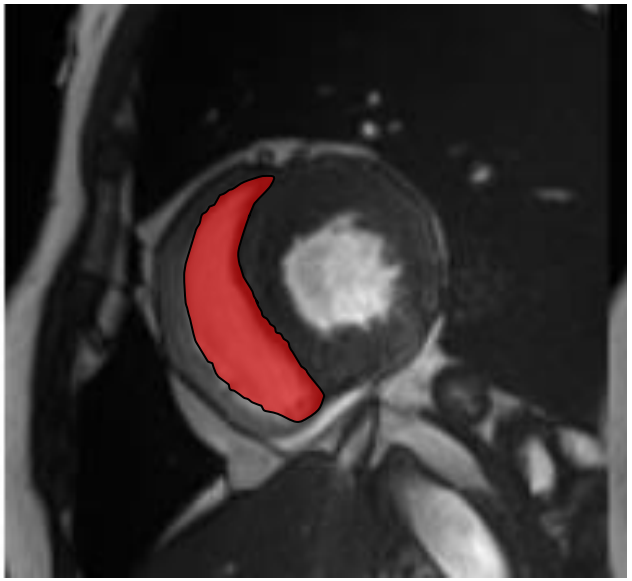
Right ventricle segmentation

Sørensen–Dice coefficient

Prediction

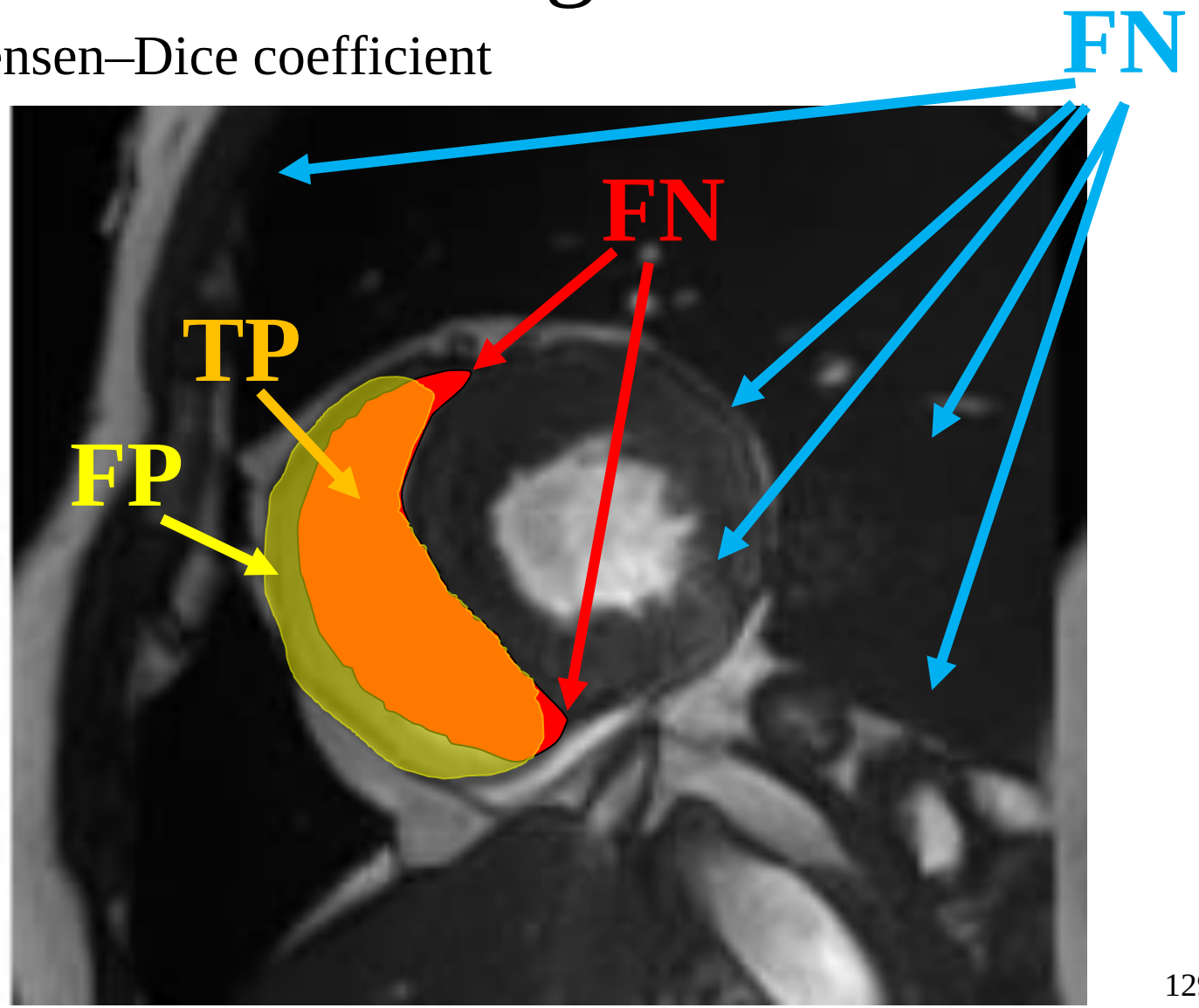


Ground truth



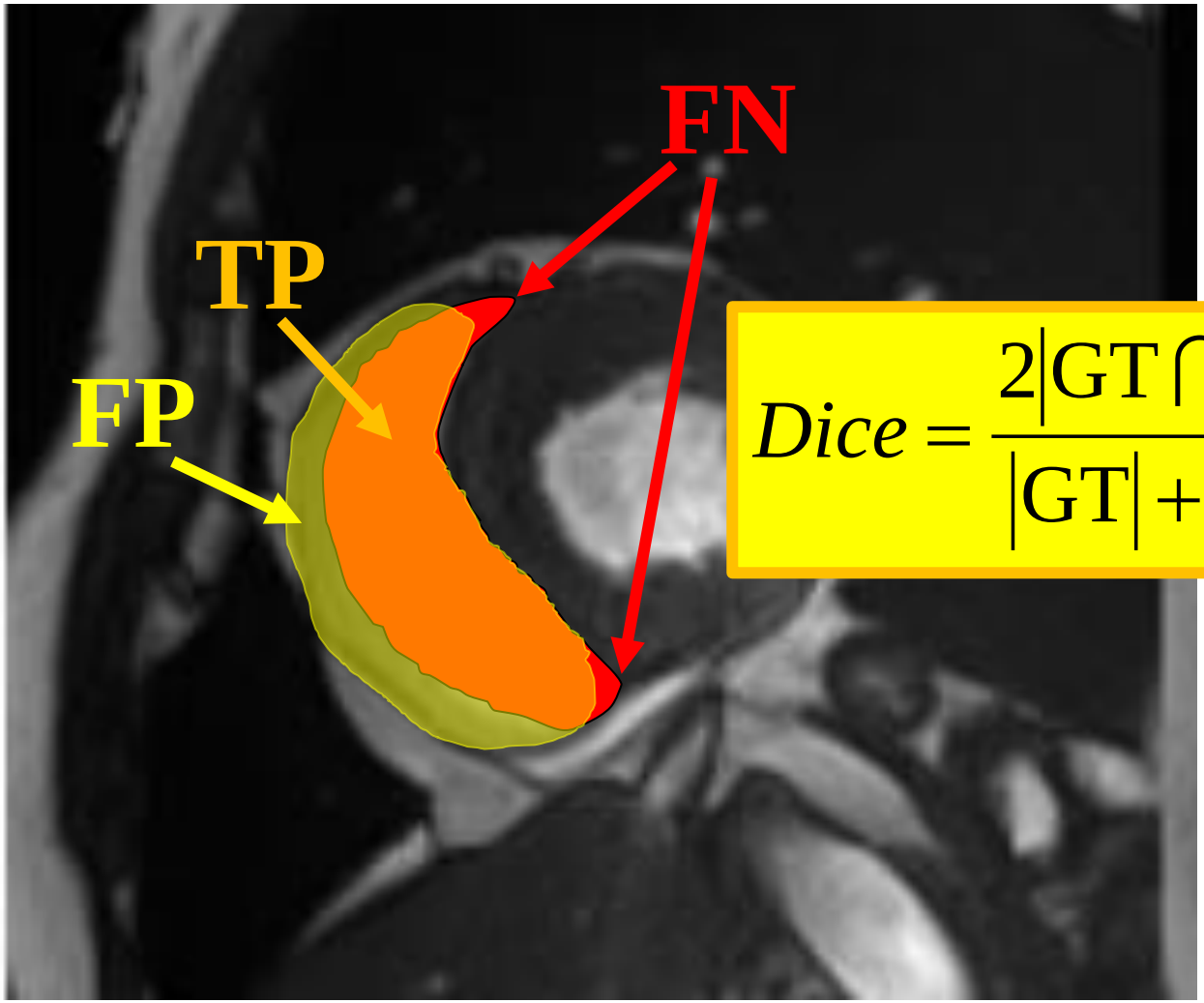
Right ventricle segmentation

Sørensen–Dice coefficient



Right ventricle segmentation

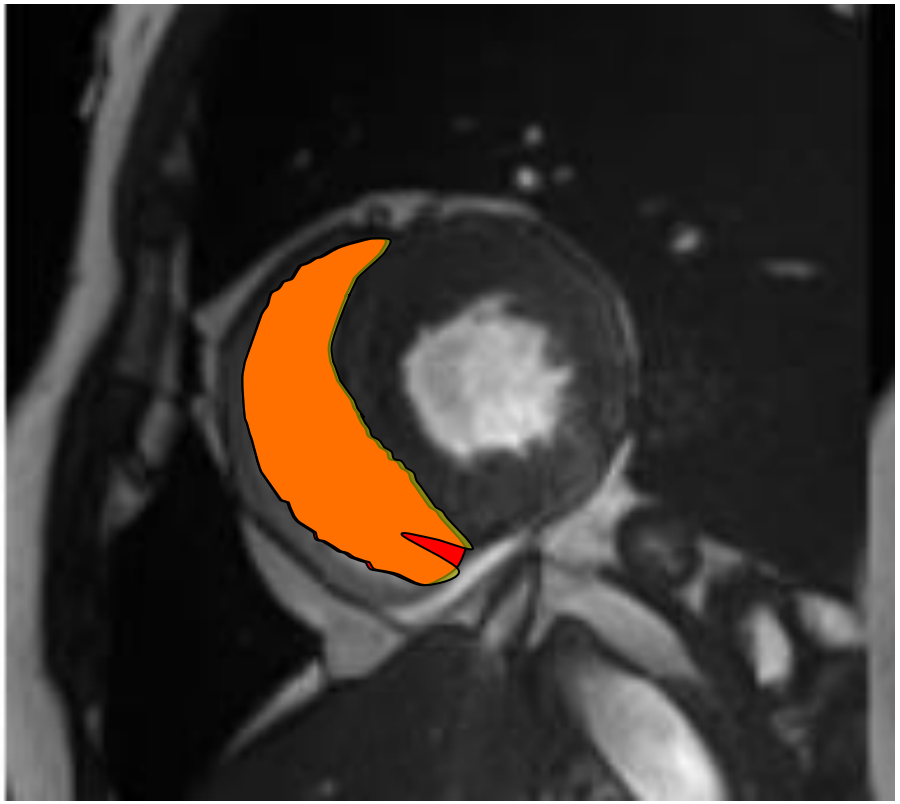
Sørensen–Dice coefficient



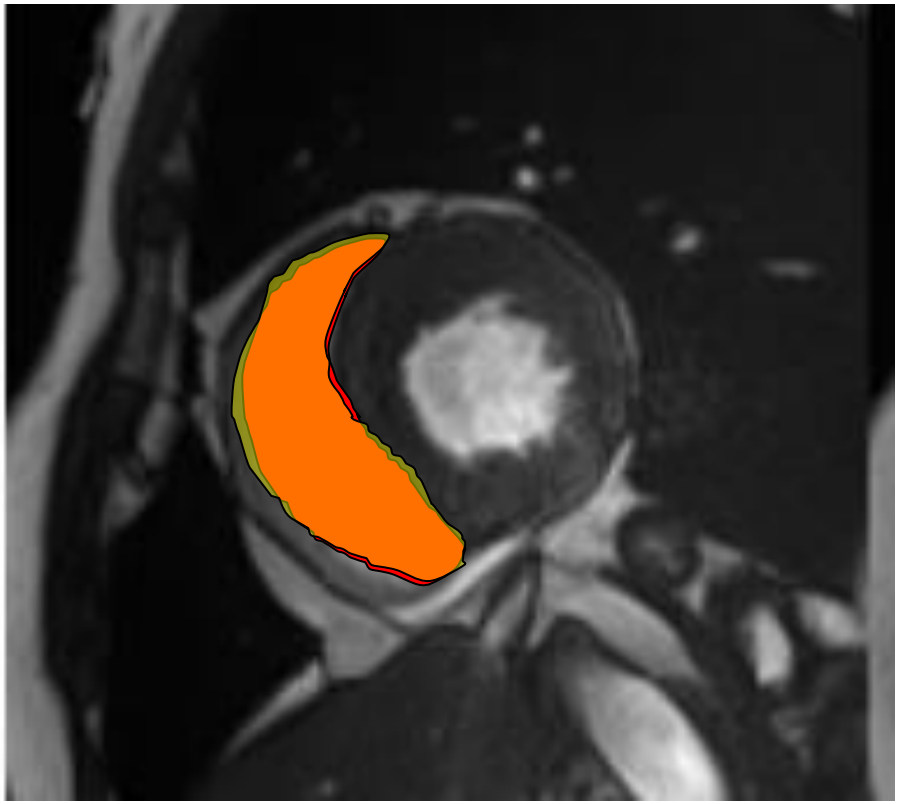
$$Dice = \frac{2|\mathrm{GT} \cap \mathrm{Pred}|}{|\mathrm{GT}| + |\mathrm{Pred}|}$$

Right ventricle segmentation

Limit of the Dice coefficient



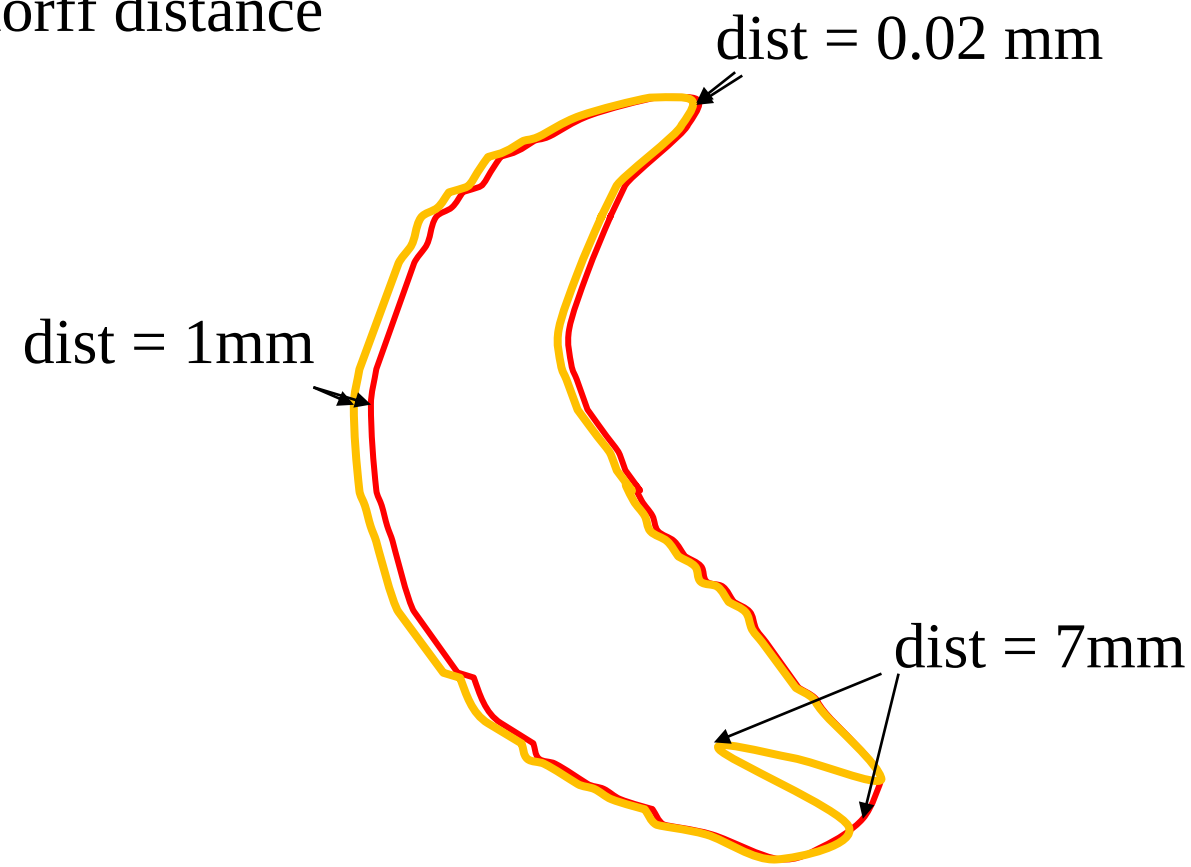
Dice=0.95



Dice=0.95

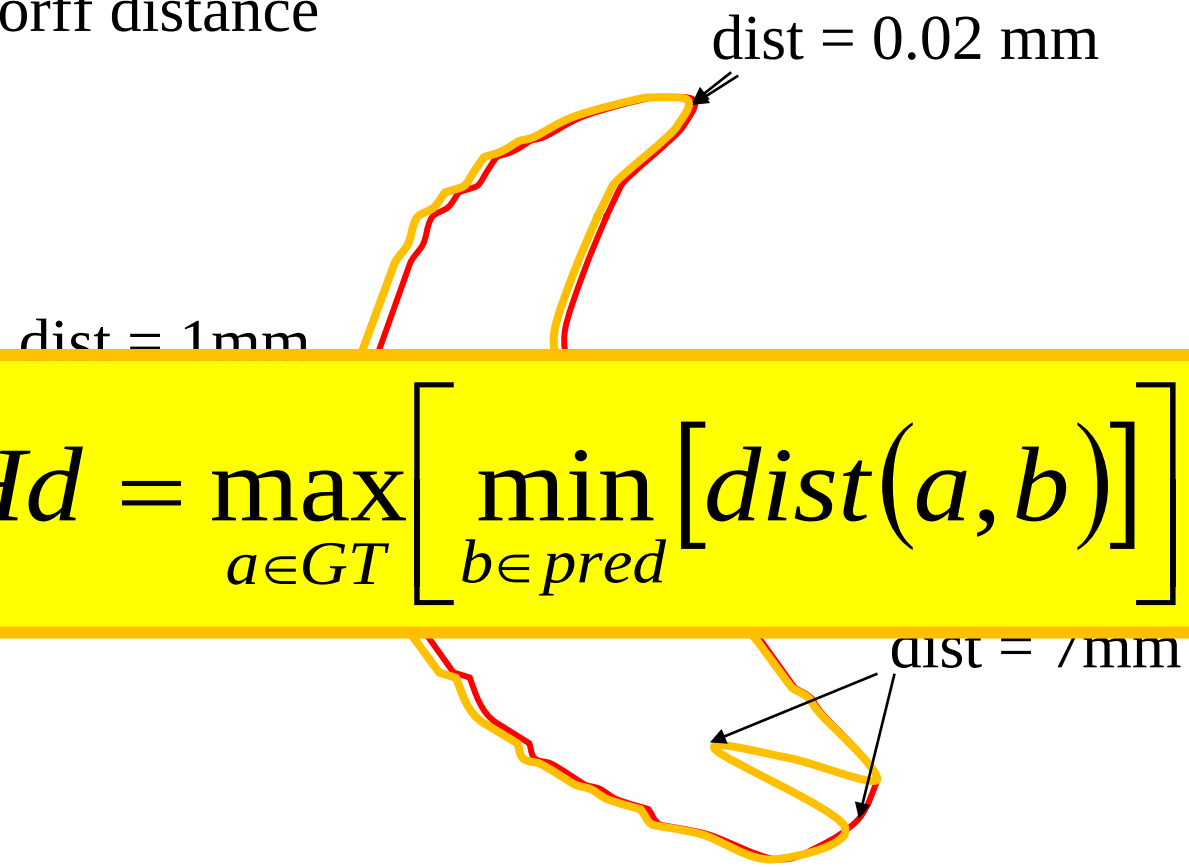
Right ventricle segmentation

Hausdorff distance



Right ventricle segmentation

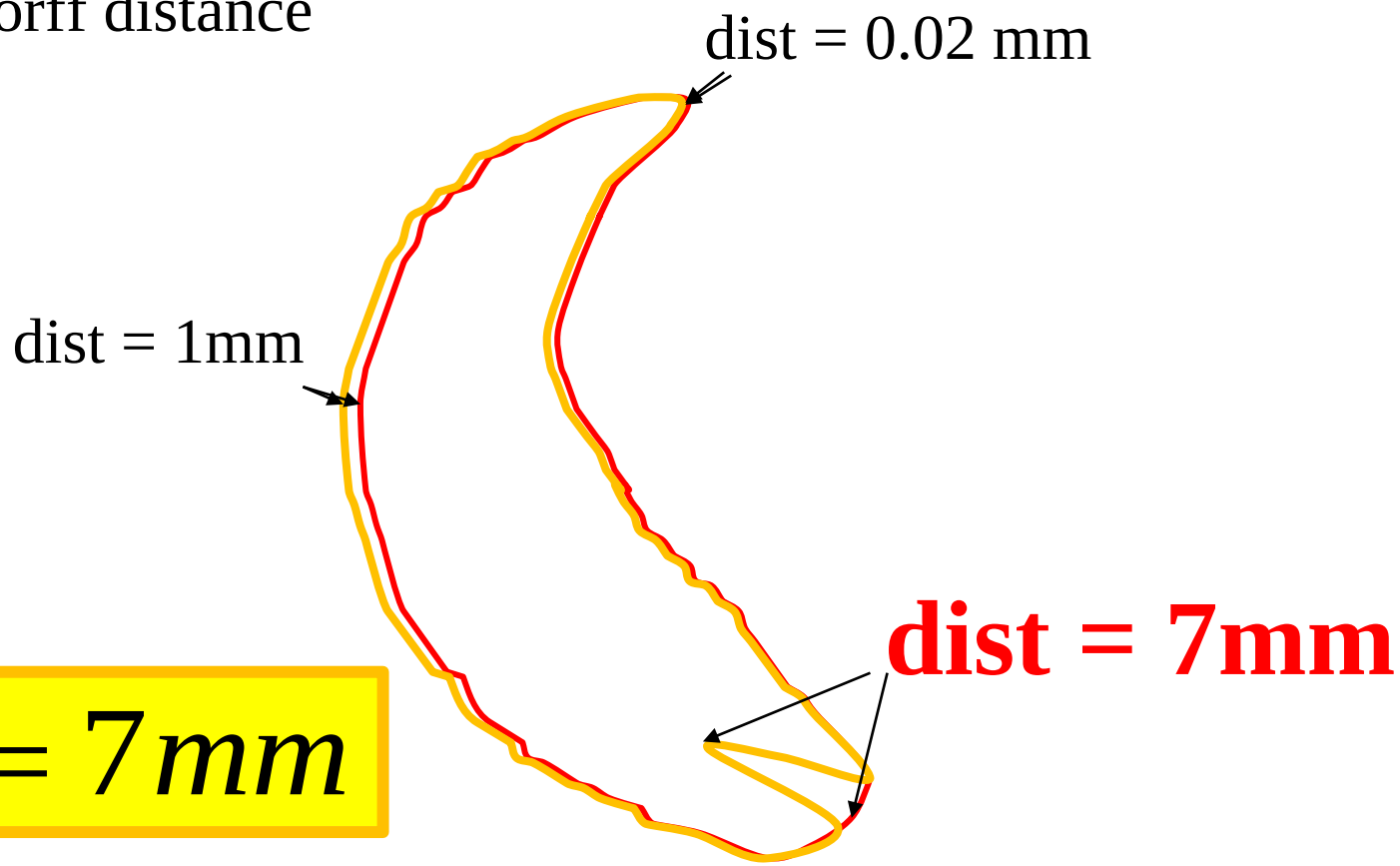
Hausdorff distance



$$Hd = \max_{a \in GT}\left[\min_{b \in pred}\left[dist(a,b)\right]\right]$$

Right ventricle segmentation

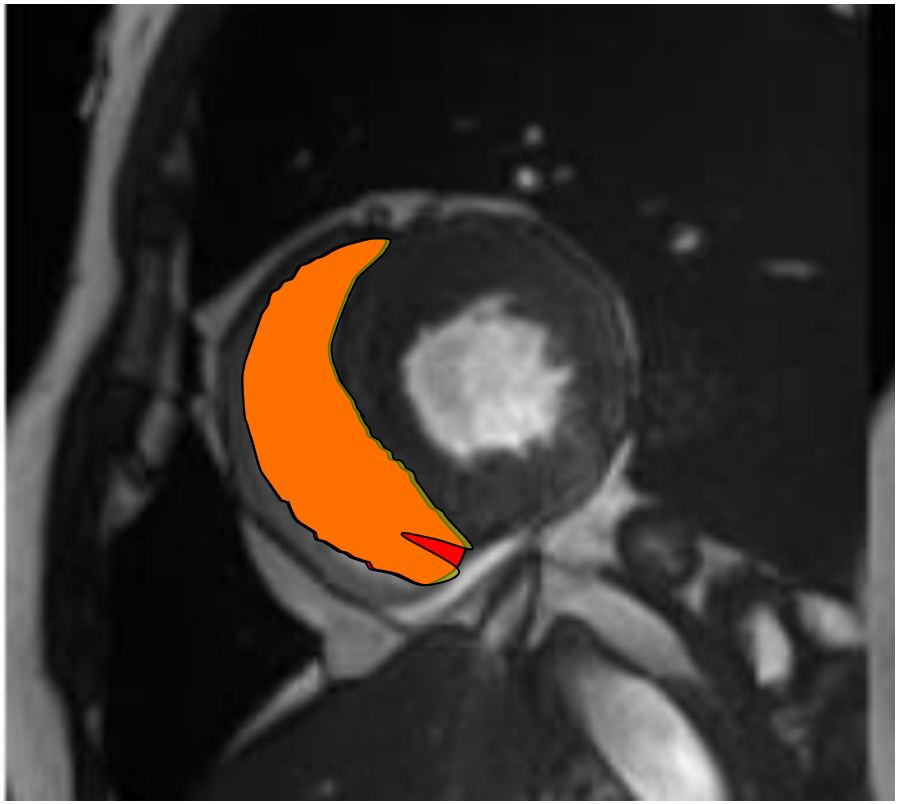
Hausdorff distance



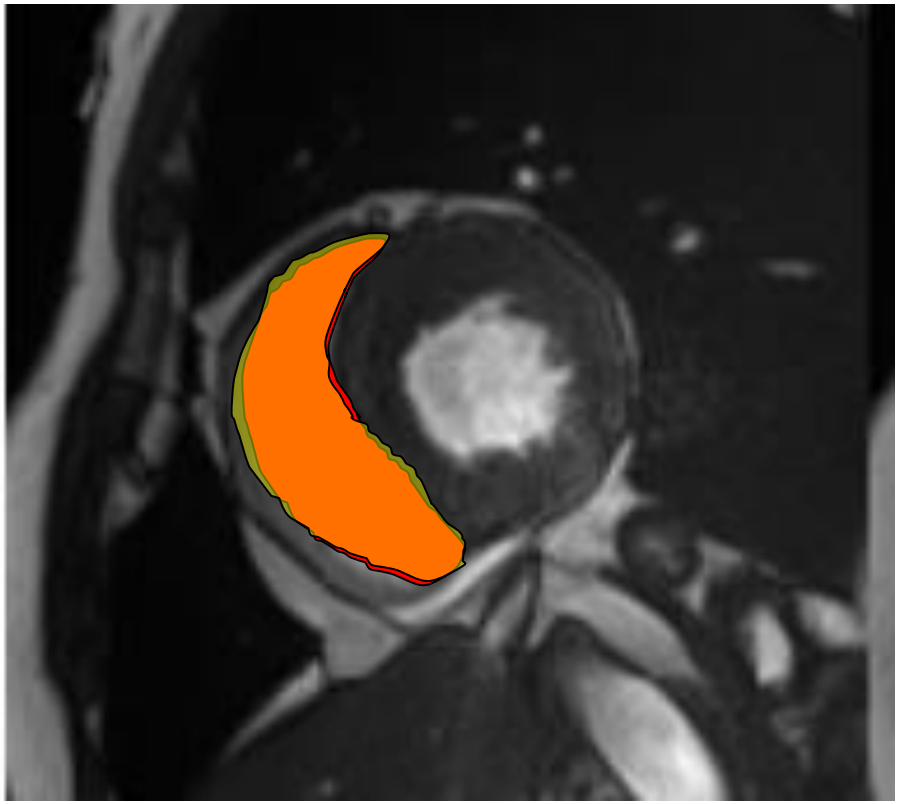
$$Hd = 7mm$$

Right ventricle segmentation

Hausdorff distance



Hd=7mm



Hd=1.6mm

In short we have seen

- Supervised vs unsupervised learning
- Regression vs Classification
- Linear vs non-linear models
- Parameters vs hyper-parameters
- Over vs Underfitting
- Cross-validation
- Metrics

Thank you!

Model ensembling

Why use only one model?

Does combining several model works well?

In practice … **yes!**

Combining several models is often called **ensembling**

Why use only one model?

Combining what?

- Several **different** models
- The **same model** trained with different **hyperparameters**
- The **same model** trained on different **data**.

Typically 2 ways of combining models

- ***Bagging*** : good for models with a **large capacity**
- ***Boosting*** : good for models with a **low capacity**

Typically 2 ways of combining models

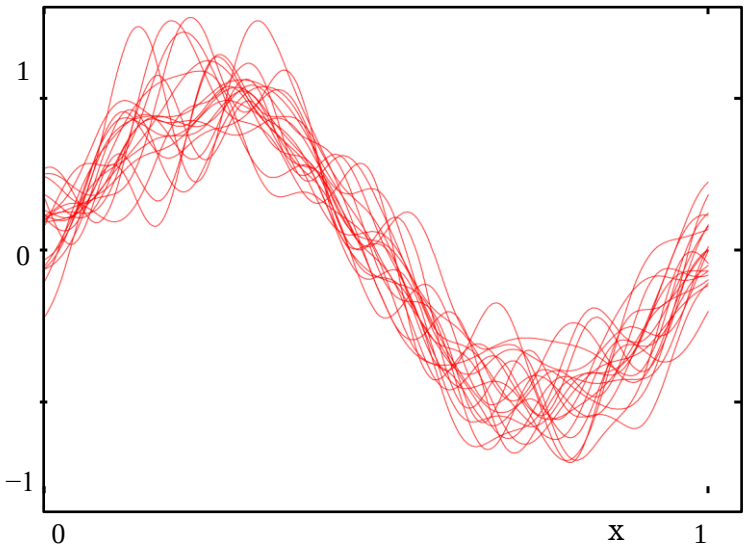
- ***Bagging*** : good for models with a **large capacity**
- ***Boosting*** : good for models with a **low capacity**

Bagging

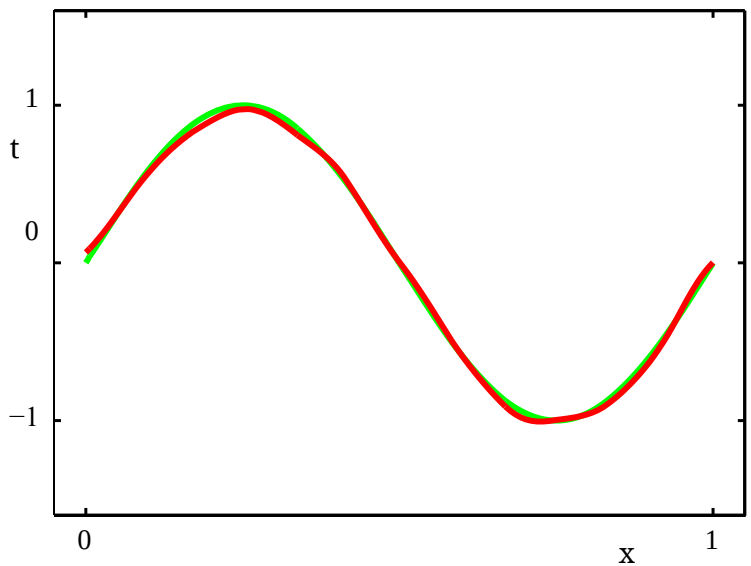
- A simple way of combining several models
  - FOR i = 1 to *m*
    - Train model $y_{W,i}(\vec{x})$
- ***Ensemble*** the *m* models
  - For **regression**
    - $y_{COM}(\vec{x}) = \frac{1}{m}\sum_{i=1}^{m} y_{W,i}(\vec{x})$
  - For a **classification**
    - **majority vote**

Ex: polynomial regression M=25

100 models trained on
100 different training sets



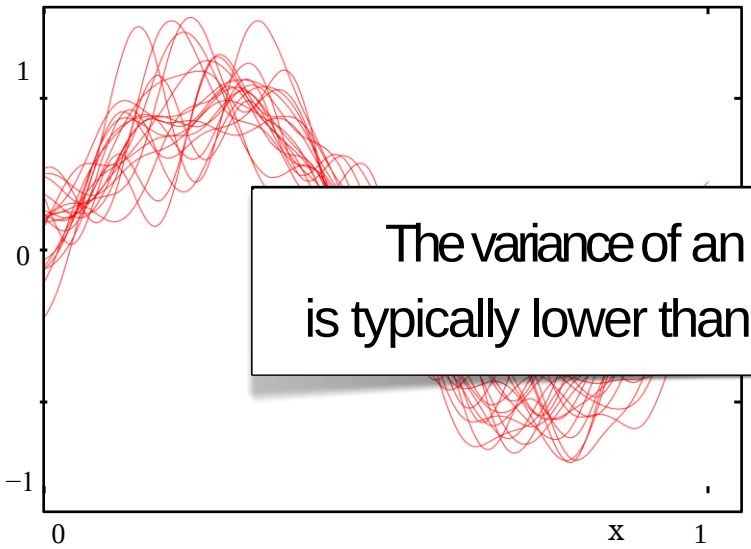
Ensemble of 100 models
Vs
True model

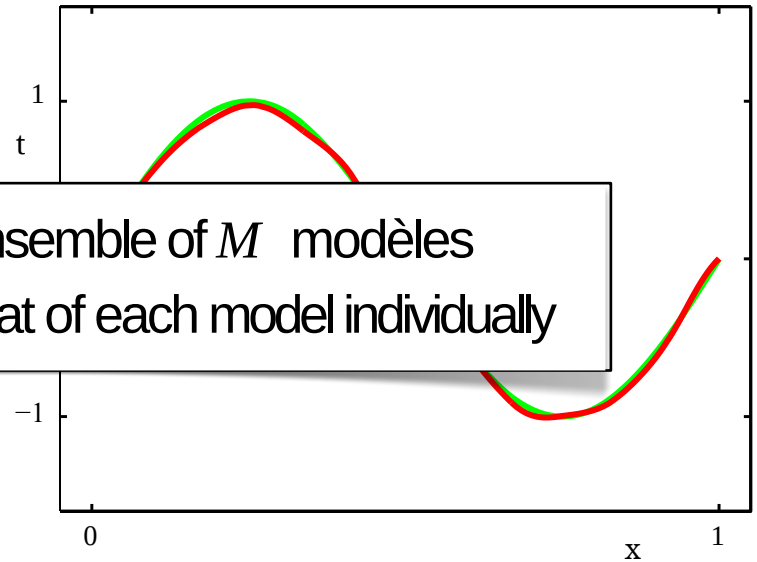


Ex: polynomial regression M=25

100 models trained on
100 different training sets

Ensemble of 100 models
Vs
True model





The variance of an ensemble of M modèles
is typically lower than that of each model individually

Bootsrap

What if you do not have enough data for building 100 trainingsets? What can you do???

Solution 1 : **Data augmentation**

Solution 2 : ***Bootsraping***.

FOR j from 1 to 100 DO

$$D_{bootstrap} = \{ \quad \}$$

FOR N iterations

- Choose randonly a natural number between *1* and *N*

- $D_{bootstrap} = D_{bootstrap} \cup \{ (\vec{x}_n, t_n) \}$

$$D_{train,j} = D_{bootstrap}$$

Train j-th model

In short we have seen

- Supervised vs unsupervised learning
- Regression vs Classification
- Linear vs non-linear models
- Parameters vs hyper-parameters
- Over vs Underfitting
- Cross-validation
- Metrics
- Ensembling – Bootsrapping

Thank you!