Montreal, July 4–8

DLMI2022

Introduction to machine learning

By

Pierre-Marc Jodoin

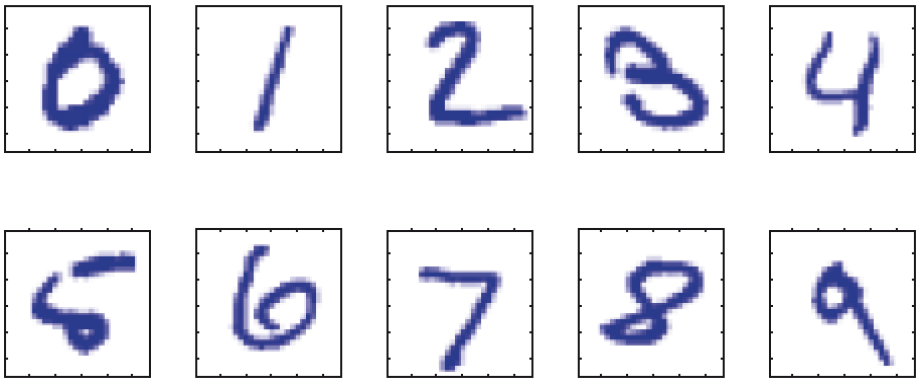
(Hugo Larochelle)

UNIVERSITÉ DE SHERBROOKE

Killer question…

What is machine learning?

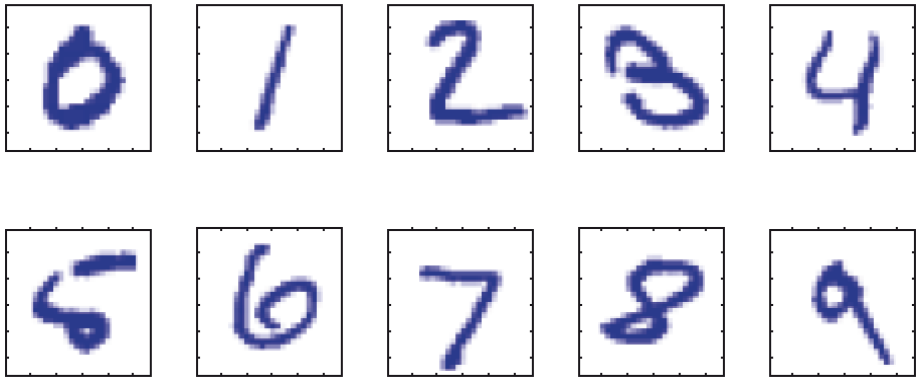
Question : how can one recognize hand written digits?



Answer : Design your own rules?

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

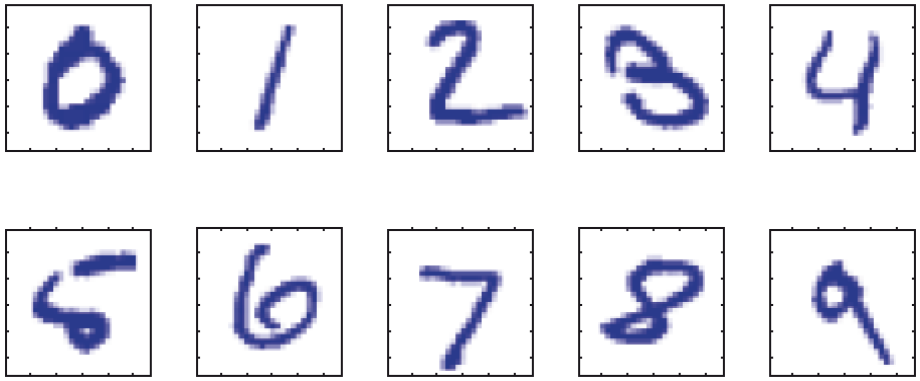
Question : how can one recognize hand written digits?



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

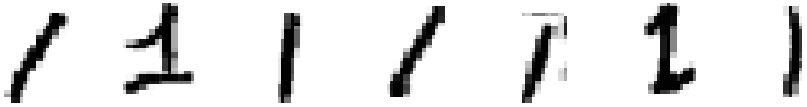
- Bad generalization

Question : how can one recognize hand written digits?

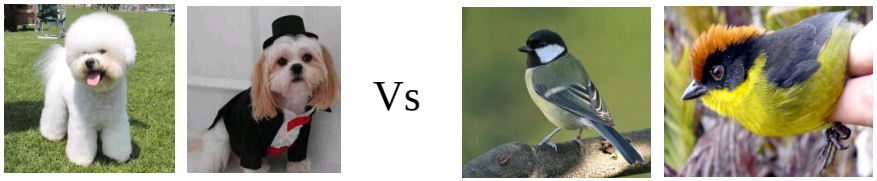


Answer : ~~Design your own rules?~~ **Wrong**

- Bad generalization

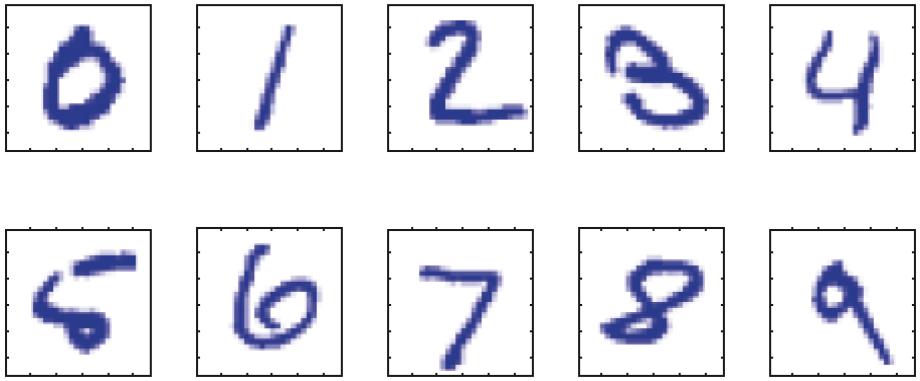


- Often difficult



Dogs Vs Birds

Question : how can one recognize hand written digits?



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

- *Main goal of machine learning*

Two large families

Supervised learning

Unsupervised learning

Two large families

Supervised learning

Unsupervised learning

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**

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

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 $\vec{x}_i$

$$y_{\vec{w}}(\vec{x}_i) \rightarrow t_i$$

where $\vec{w}$ are the **parameters** of the model.

Supervised learning

Once the model $y_{\vec{w}}(\vec{x})$ 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

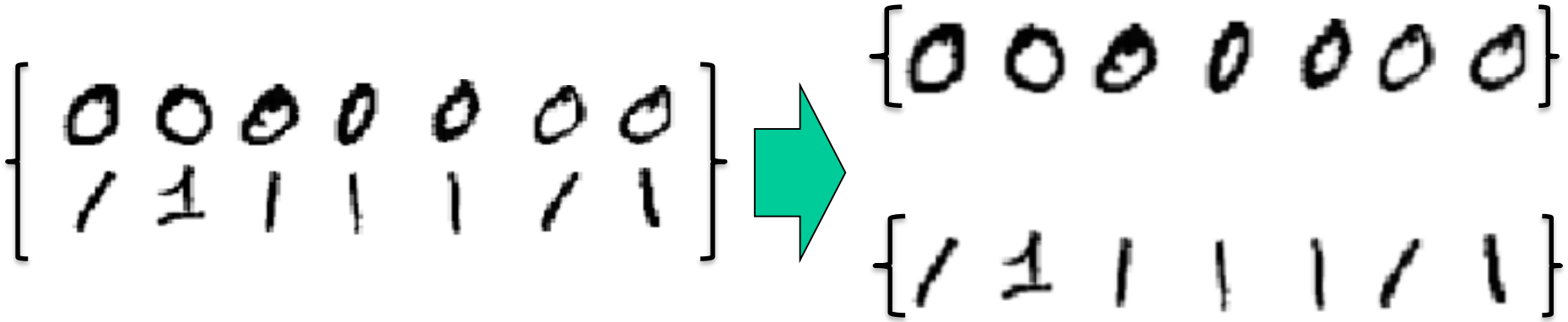
Supervised learning

Unsupervised learning

Unsupervised learning

When no target is explicitly provided

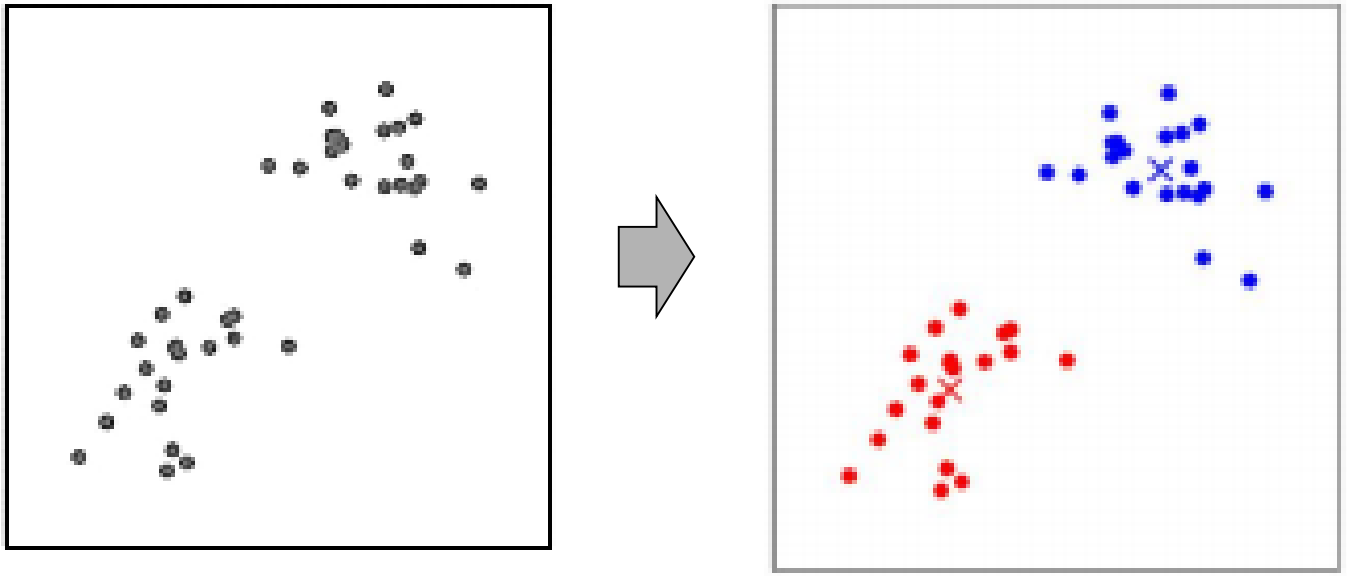
- E.g. data *clustering*



Unsupervised learning

When no target is explicitly provided

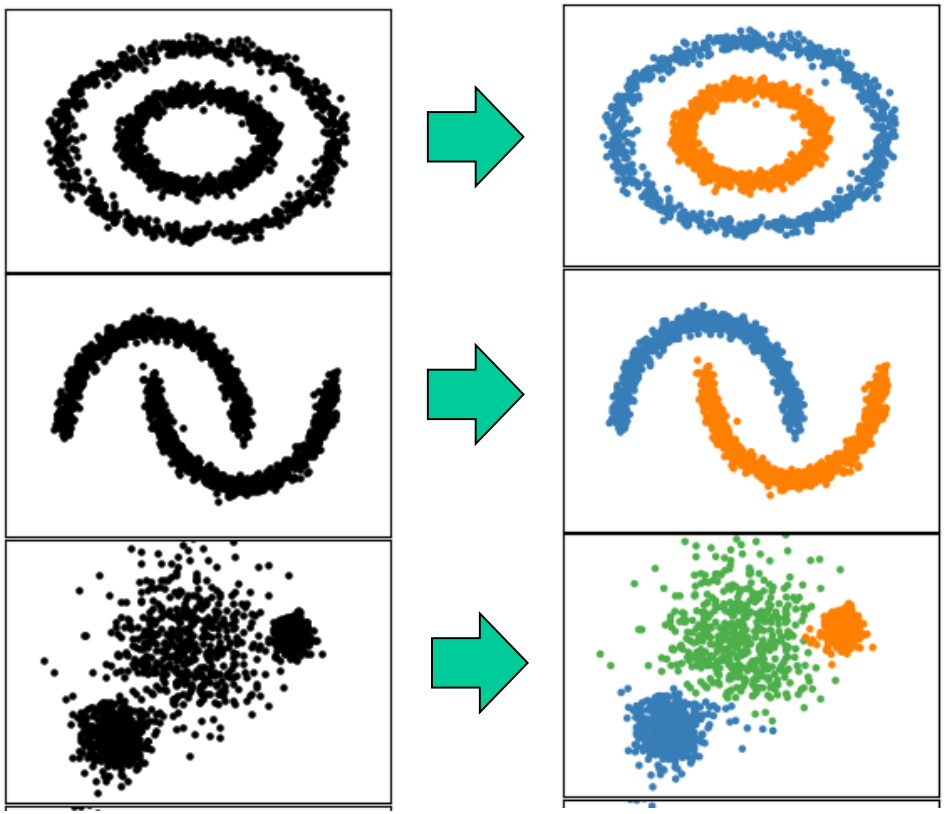
- E.g. data *clustering*



Unsupervised learning

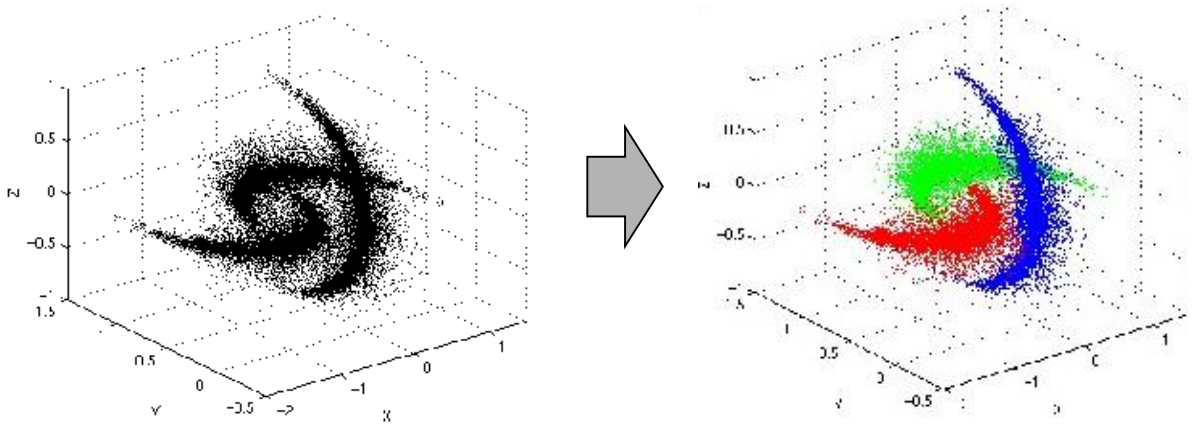
When no target is explicitly provided

- E.g. data *clustering*



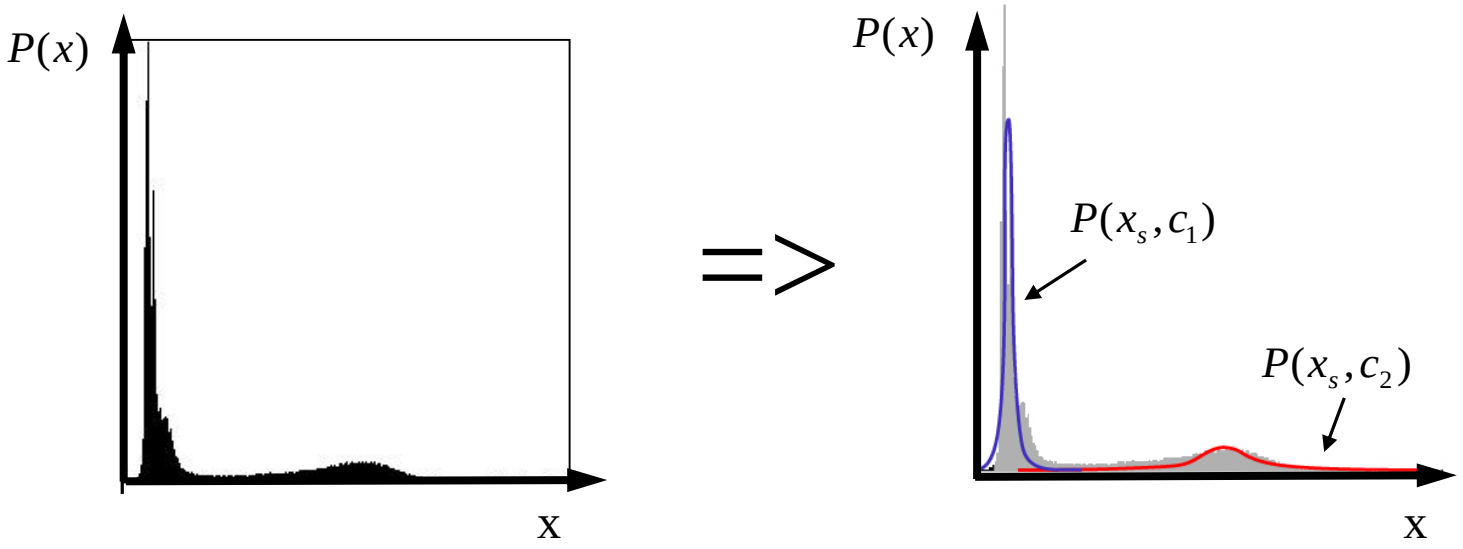
Unsupervised learning

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



Unsupervised learning

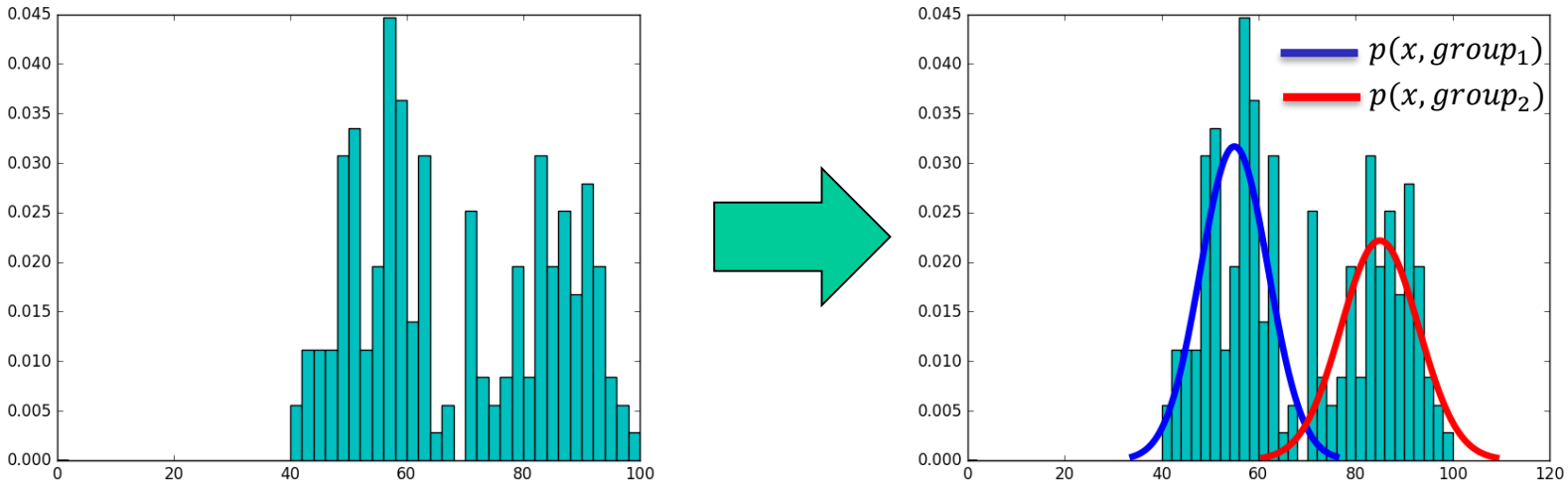
The goal is often to learn the distribution $p(x)$ from which the data has been generated from



Unsupervised learning

The goal is often to learn the distribution $p(x)$ from which the data has been generated from

Example : find two groupes of patients following a memory test



Unsupervised learning

The goal is often to learn the distribution $p(x)$ from which the data has been generated from

Other applications

- Data compression
- Data visualisation
- Population analysis
- Image segmentation
- etc.

Supervised vs non-supervised

Main topic of the school

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 target

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

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
Graph Neural Nets
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

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

Back to supervised learning

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.

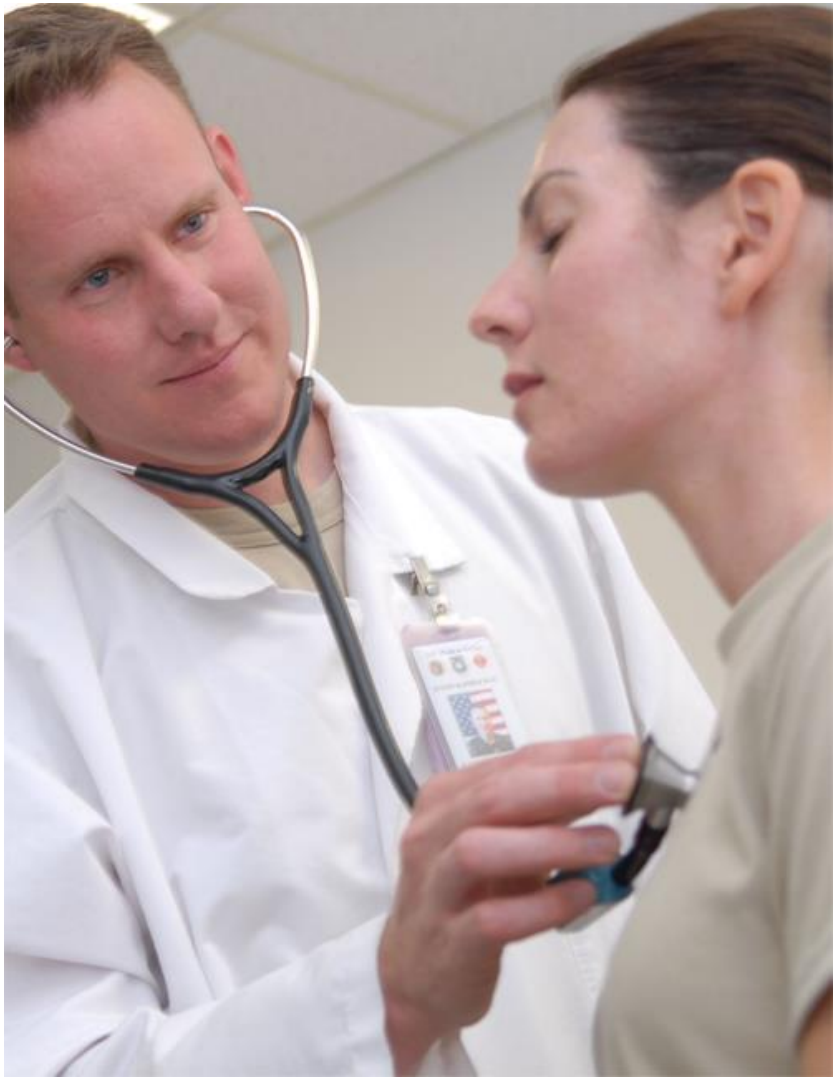
Supervised learning

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

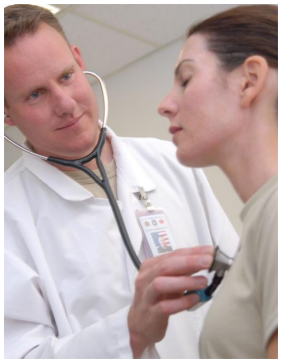
- **Régression :** the target is a real number $t \in \mathbb{R}$
  - Exemple : prediction of life expectancy
    - ✓ $\vec{x}$: vector of medical measures, age, sex, etc.
    - ✓ t : number of months before death.

Simple example of binary classification



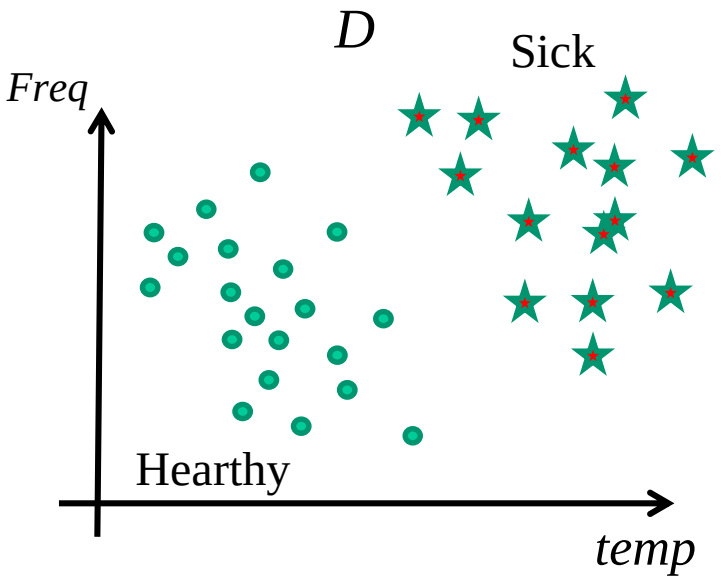
From Wikimedia Commons
the free media repository

Simple example of binary classification



D

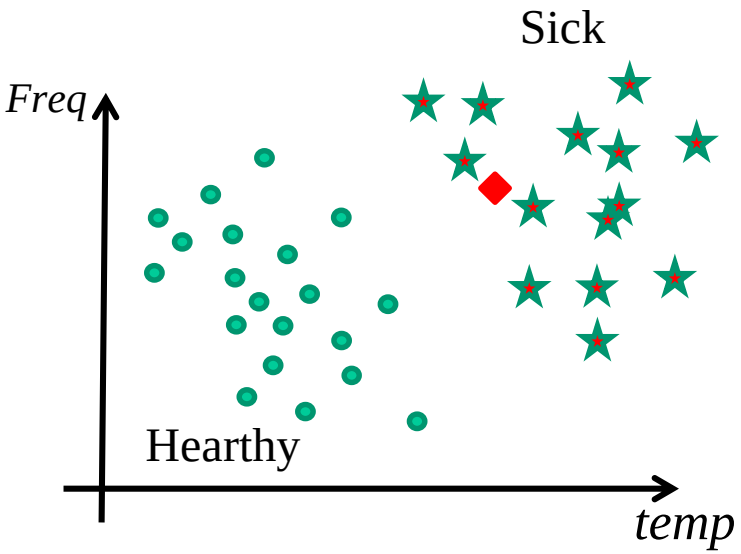
	(temp, freq)	Diagnostic
Patient 1	(37.5, 72)	hearthy
Patient 2	(39.1, 103)	sick
Patient 3	(38.3, 100)	sick
	(…)	…
Patient N	(36.7, 88)	hearthy
	$\vec{x}$	t



Simple example of binary classification

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

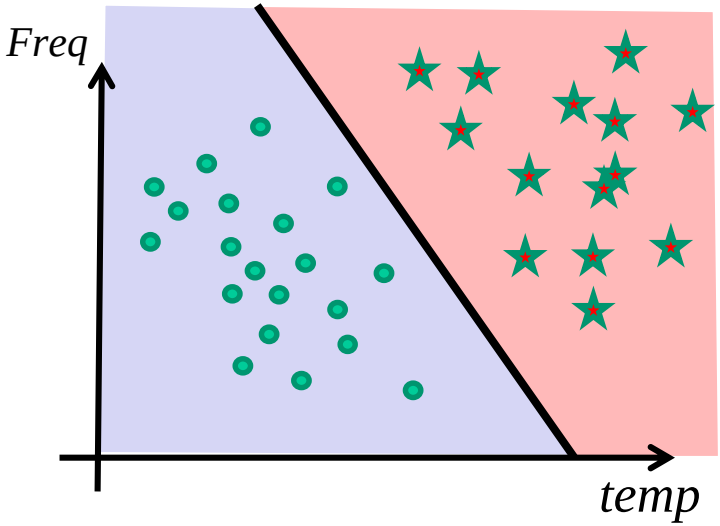
From Wikimedia Commons
the free media repository



Solution

From Wikimedia Commons
the free media repository

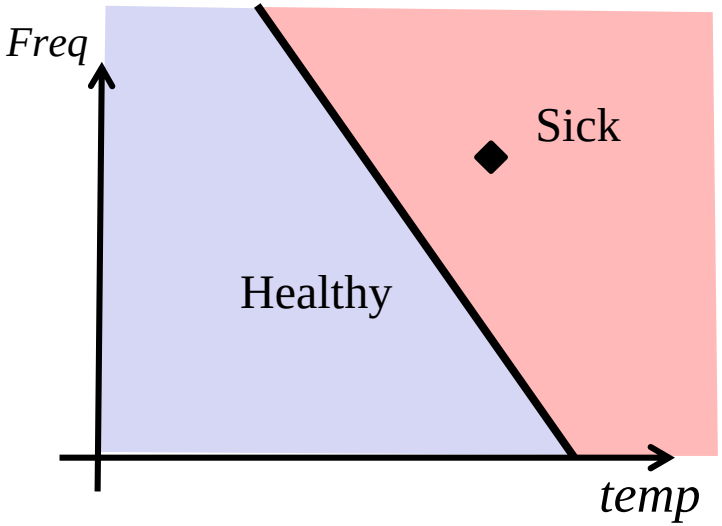
Divide the feature space in two regions : healthy and sick



Solution

From Wikimedia Commons
the free media repository

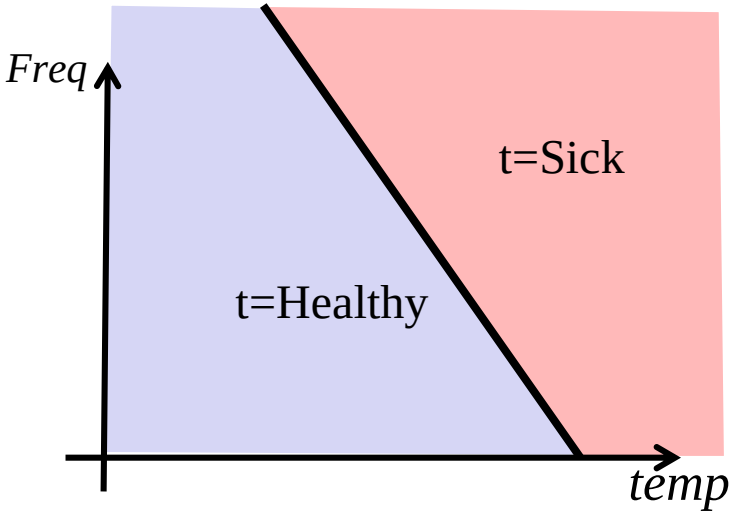
Divide the feature space in two regions : healthy and sick



More formally

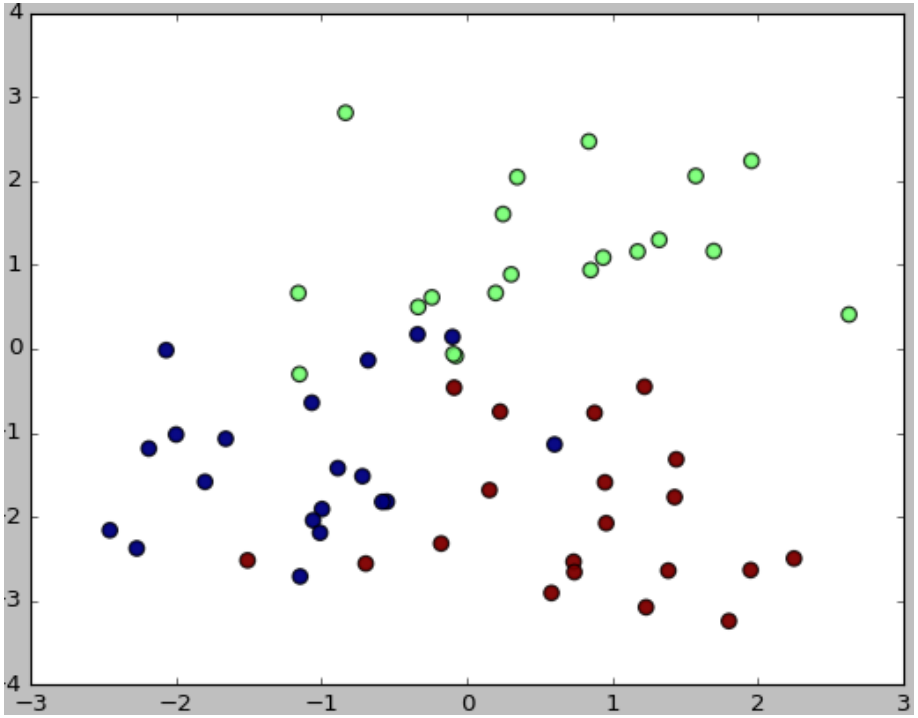
From Wikimedia Commons
the free media repository

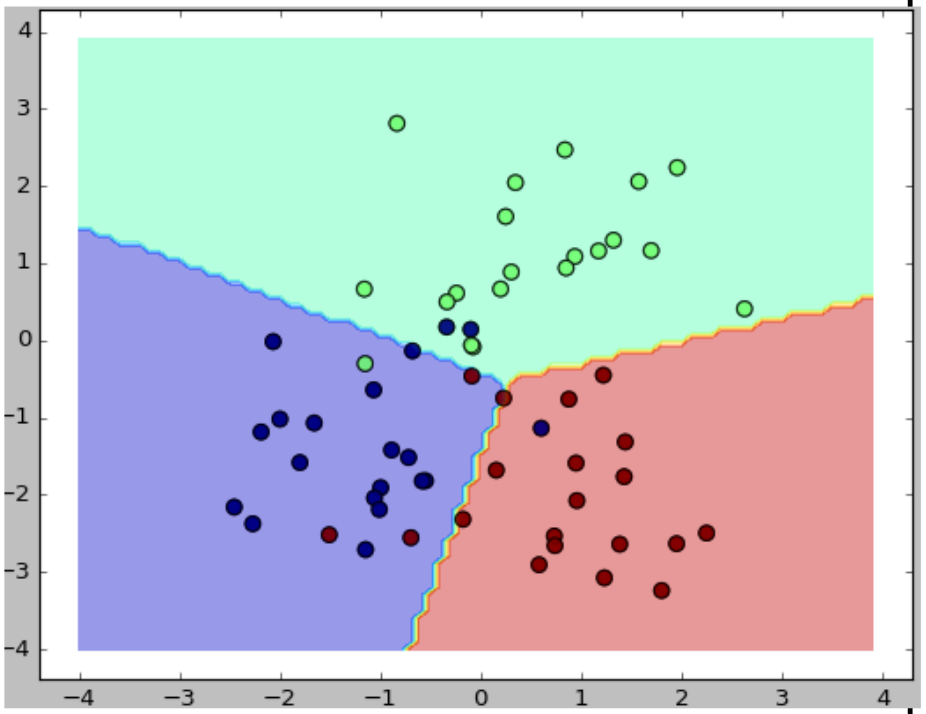
$$y_{\vec{w}}(\vec{x}) = \begin{cases} \textit{H}\text{ealthy if } \vec{x} \text{ is in the blue region} \\ \textit{Sick} \quad \text{otherwise} \end{cases}$$



Classification

3-class case





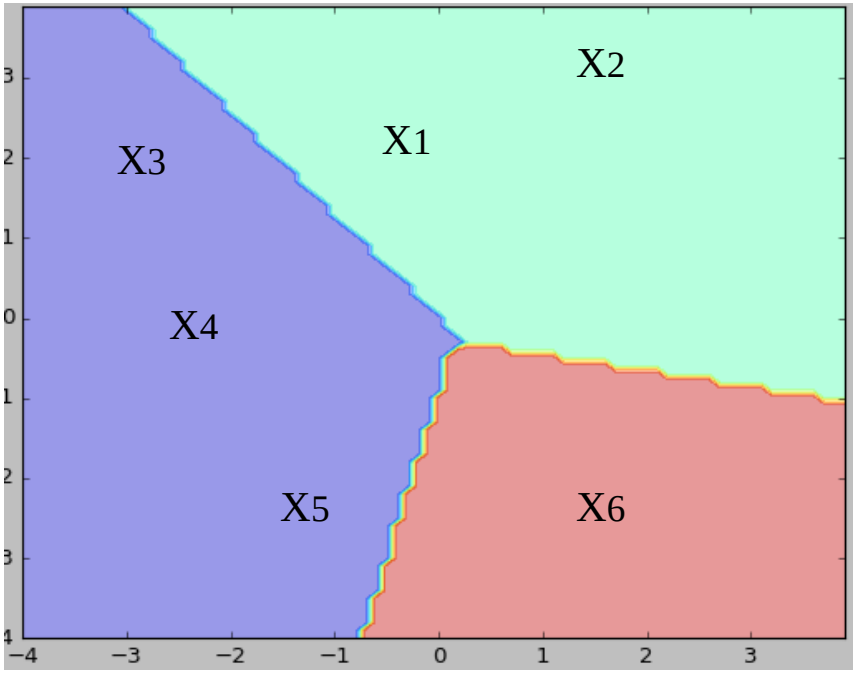
3 classes ●, ●, ● in a 2D feature space

Once training is over

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

Classification

Once training is over, we have a function y(x) that convert a point x into a class label



y(X1) => class
y(X2) => class
y(X3) => class
y(X4) => class
y(X5) => class
y(X6) => class

Example of a classification dataset

MNIST

Example of a classification dataset

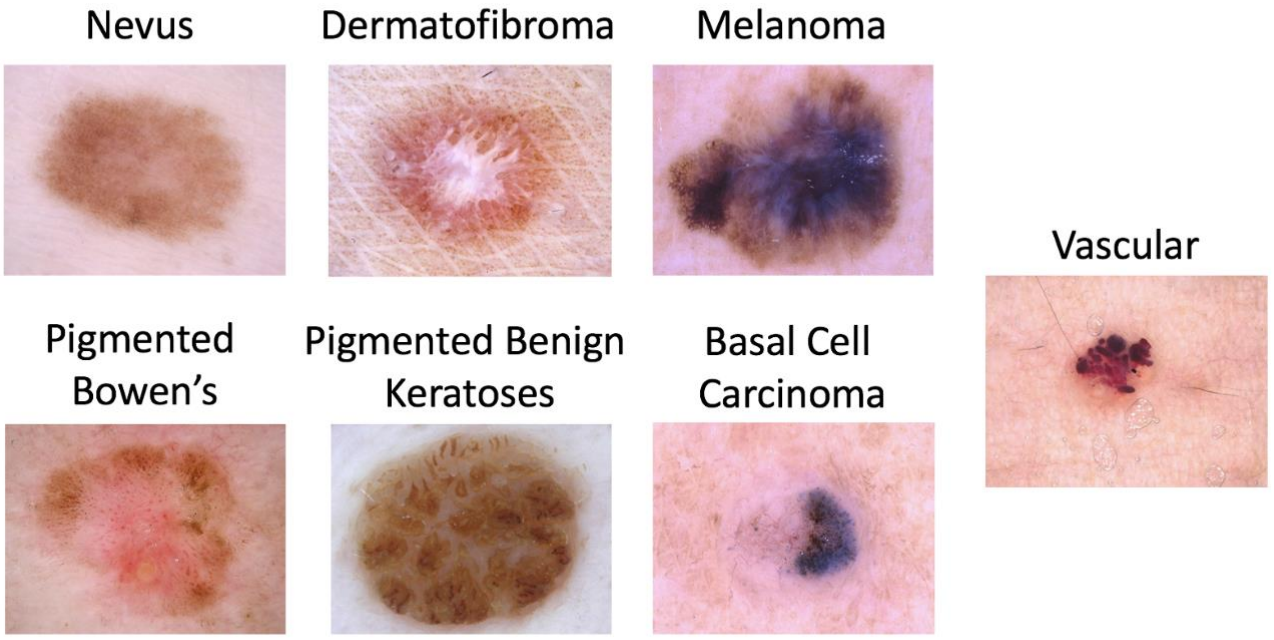
MNIST

- 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

ISIC melanoma classification challenge.



https://challenge.isic-archive.com/landing/2018/

Example of a medical classification dataset

ISIC melanoma classification challenge dataset

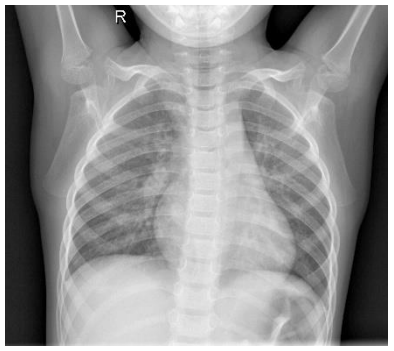
- 7 classes
- 11,527 images,
  => 10,015 training
  => 1,512 test
- Each image is in RGB
  => 628x417x3

We can **vectorize these images** and represent it by a vector of size 628x417x3 =**785,628 dimensions.**

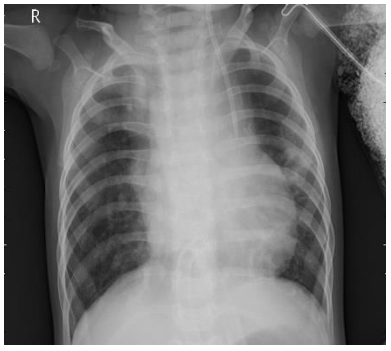
Example of a medical classification dataset

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

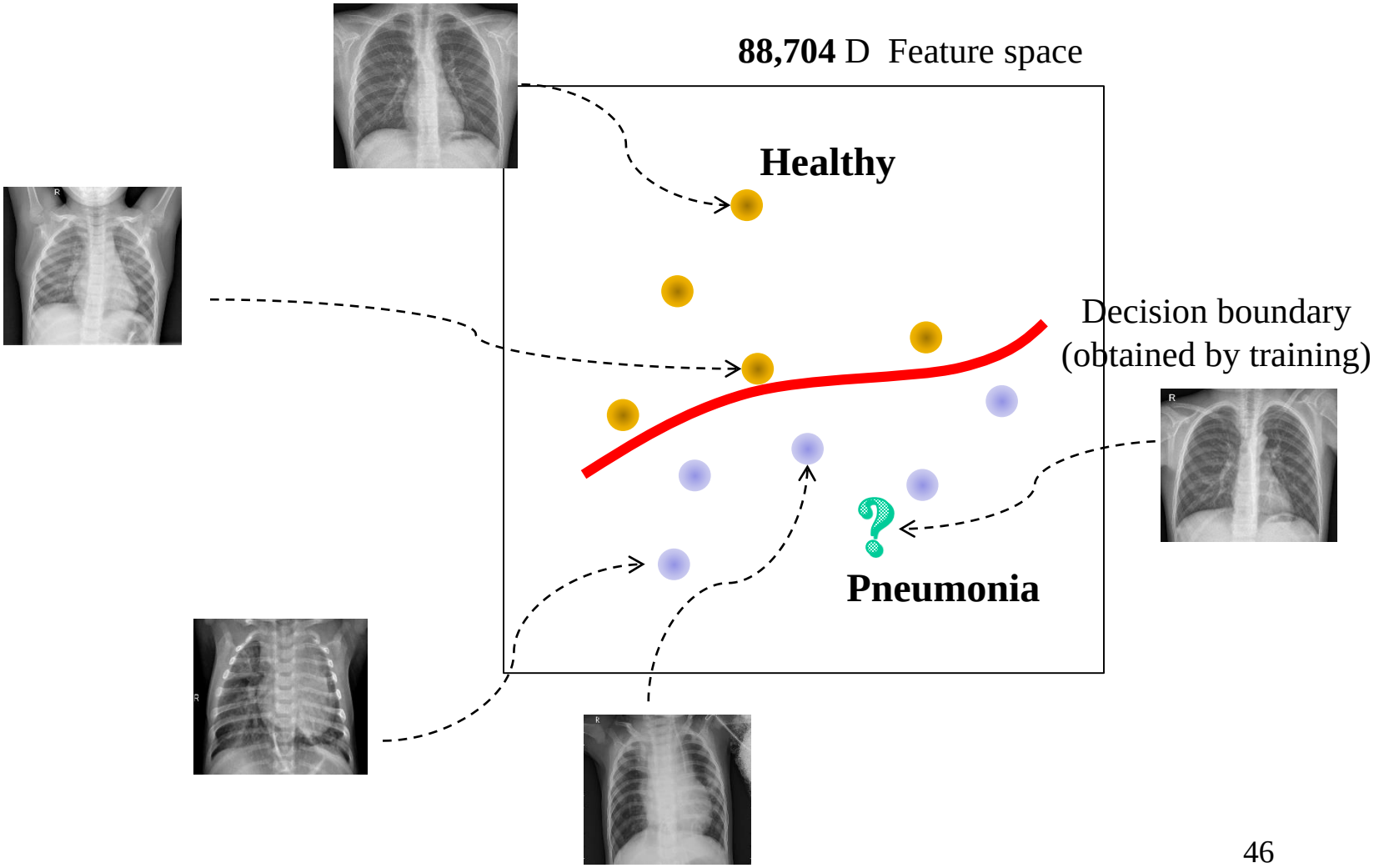
- 2 classes
- 5,840 images,
  => 5,216 training
  => 624 test
- Each image is in grayscale
  => 336 x 264*

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

* Rescaled version

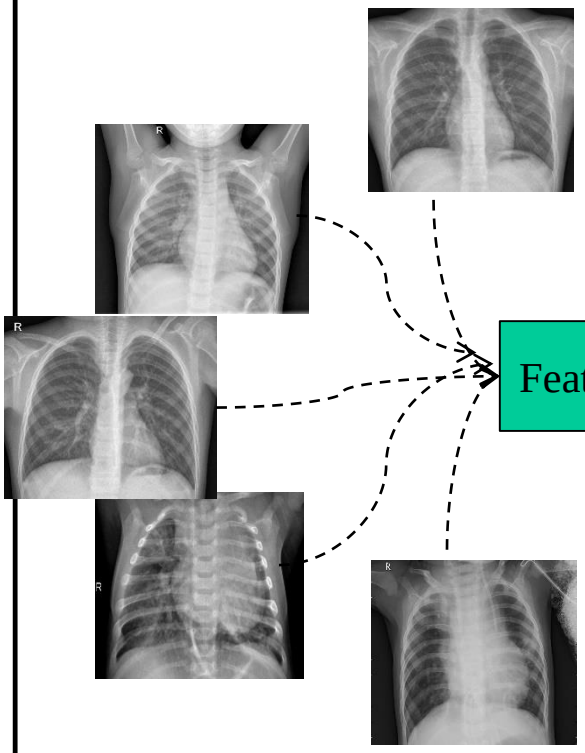
Supervised learning

Chess X-Ray Pneumonia



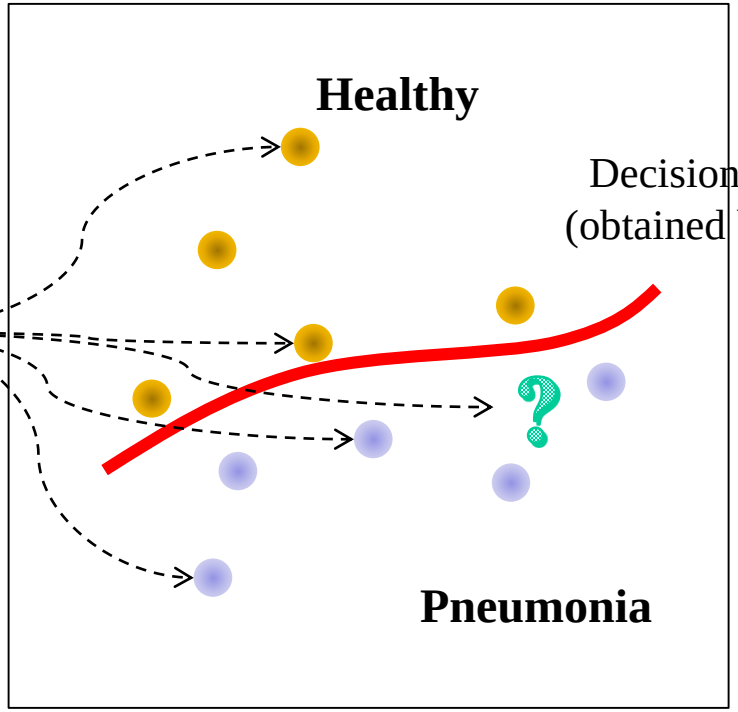
Very large feature spaces (like **88,704 dim) are problematic**.

Supervised learning



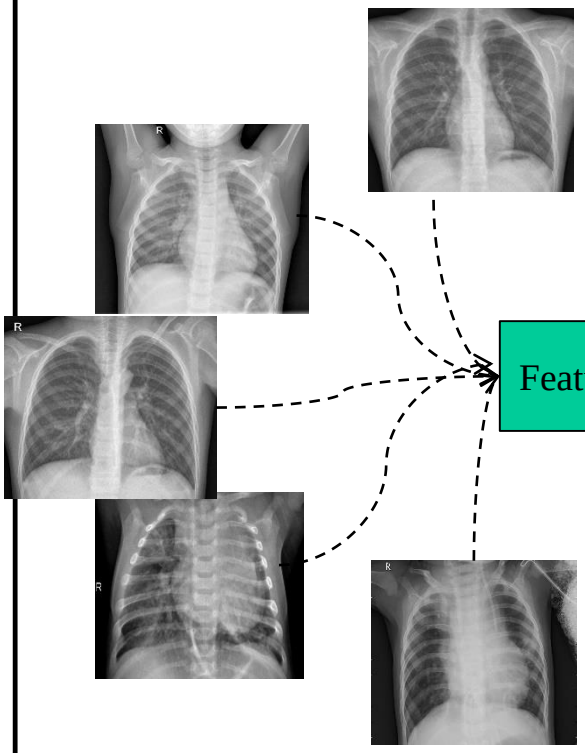
Feature extraction

K-D Feature space



Edge detection
Histograms
Filters of all kinds
Moments
Etc.

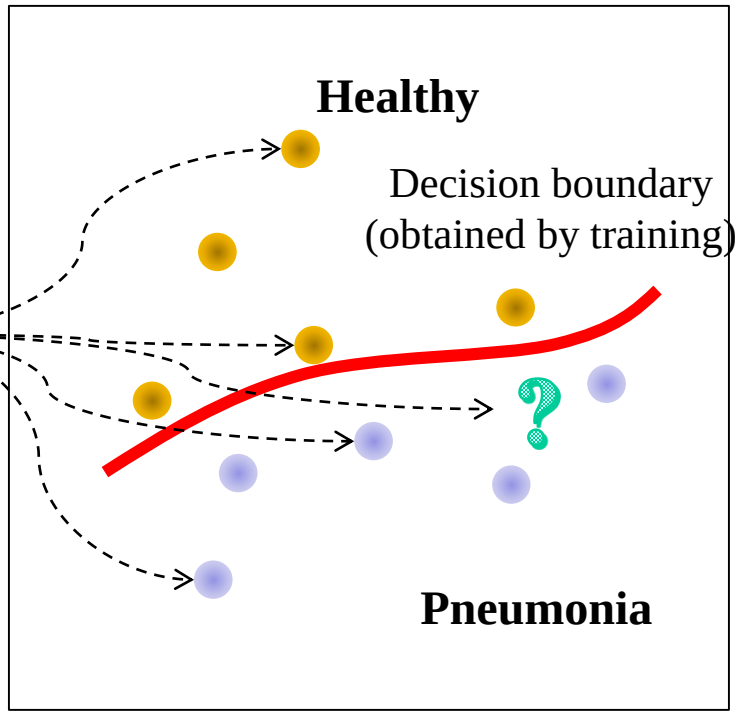
Supervised learning



Feature extraction

Dimensionality reduction

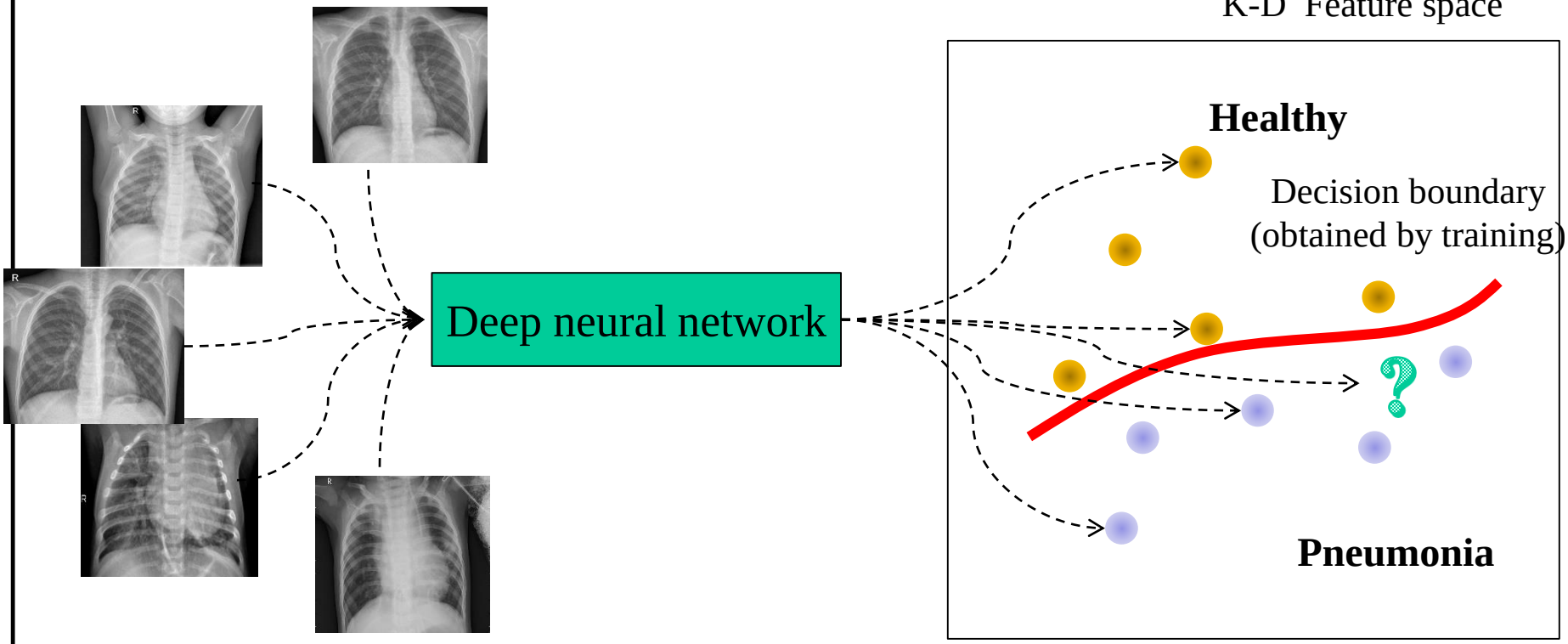
K-D Feature space



PCA
Kernel PCA
T-SNE
ISOMAPS
Etc.

Spoiler alert

In 2022…


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

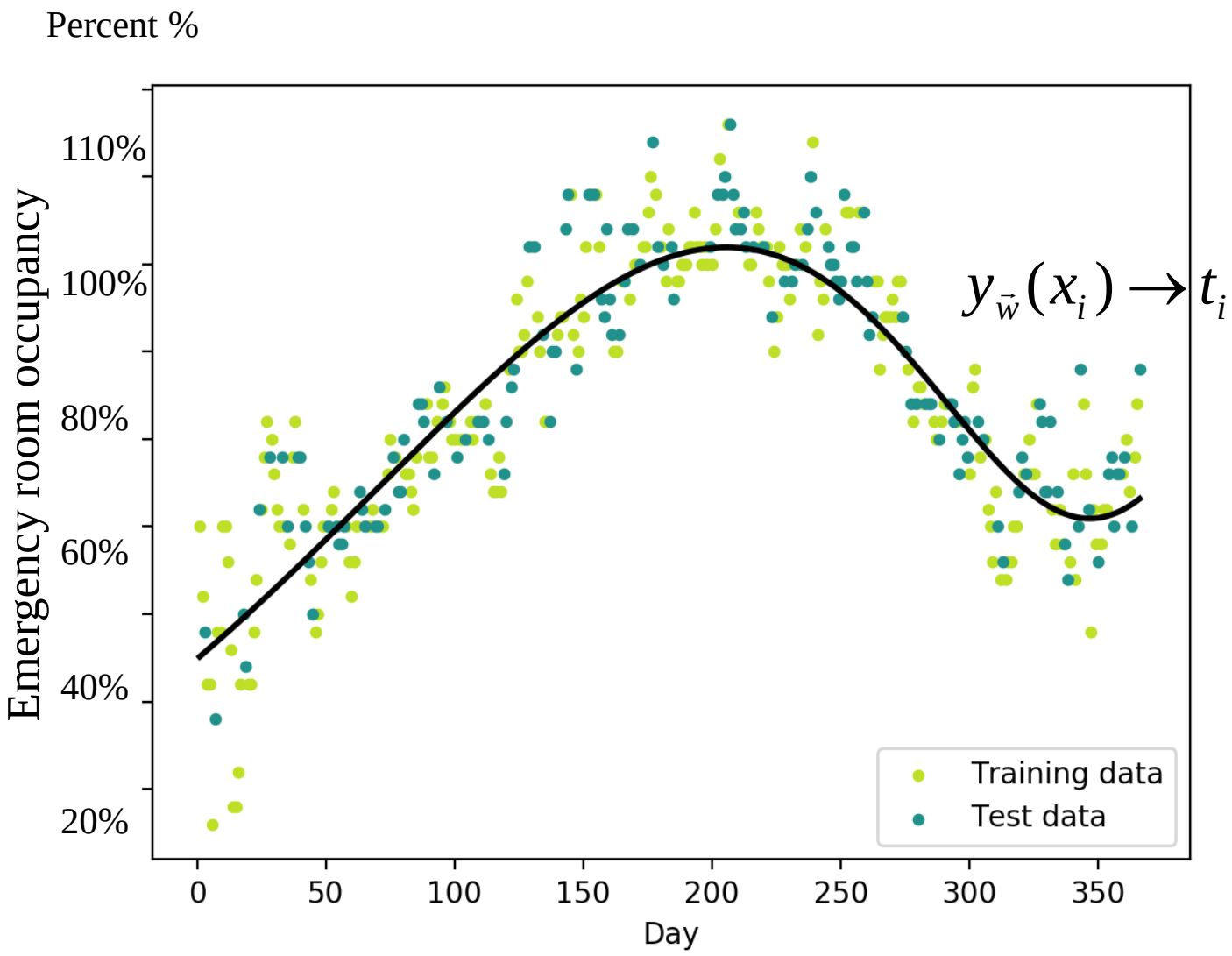
Supervised learning

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.

Example

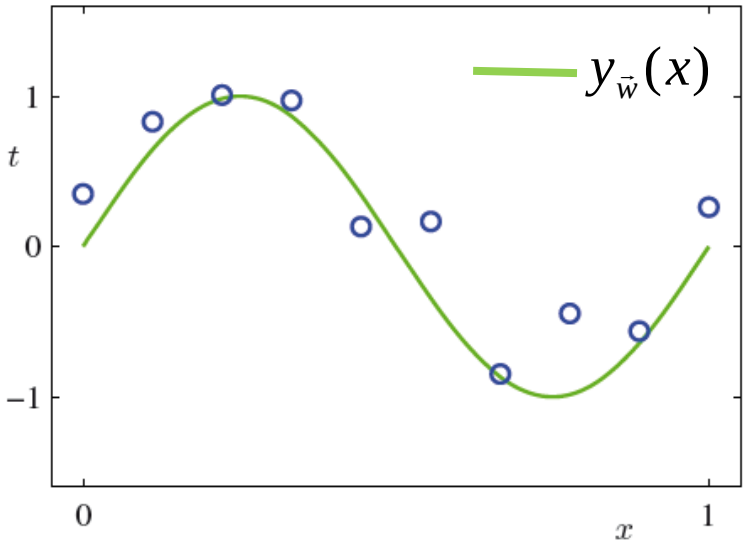

Percent %
Emergency room occupancy
110%
100%
80%
60%
40%
20%
0
50
100
150
200
250
300
350
Day
$y_{\vec{w}}(x_i) \rightarrow t_i$
Training data
Test data

Formal example:

1D regression

1D Regression

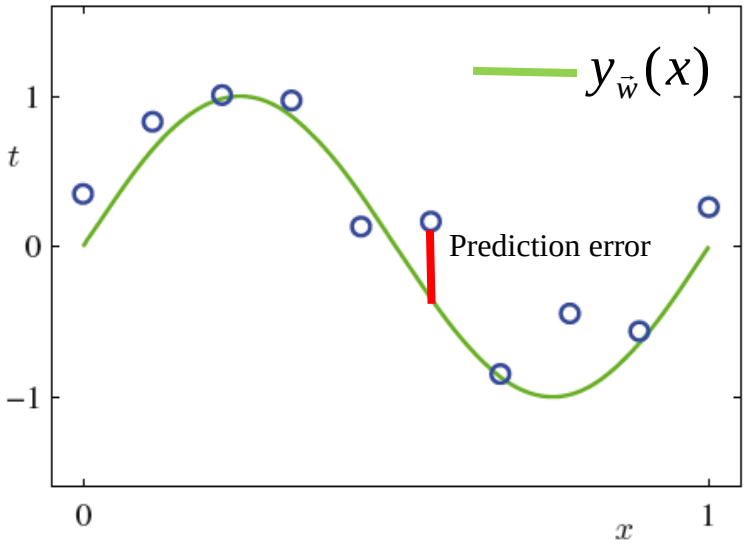
Simple example



- **Data**
  - ✓ input : scalar $x \in [0,1]$
  - ✓ target : scalar $t \in [-1,1]$
- **Training dataset** D contains:
  - ✓ $X = (x_1, \ldots, x_N)^T$
  - ✓ $T = (t_1, \ldots, t_N)^T$
- **Goal :**
  - ✓ Get a prediction $\hat{t}$ for each new data $\hat{x}$

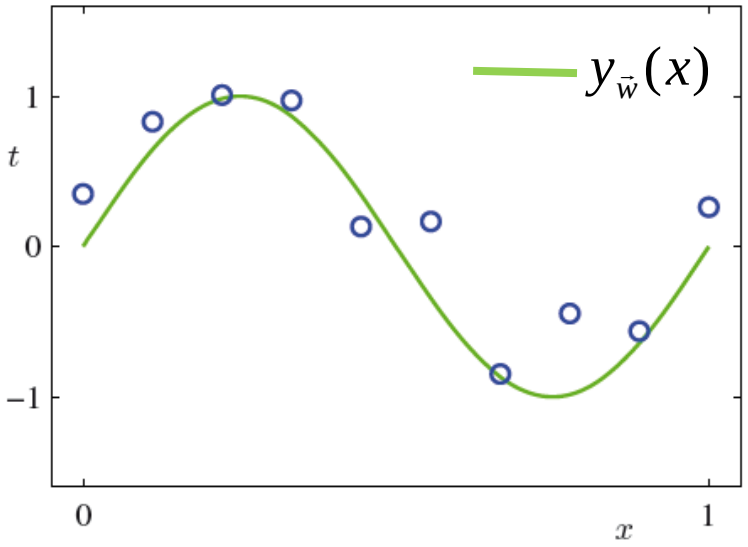
1D Regression

Simple example



- **Data**
  - ✓ input : scalar $x \in [0,1]$
  - ✓ target : scalar $t \in [-1,1]$
- **Training dataset** D contains:
  - ✓ $X = (x_1, \ldots, x_N)^T$
  - ✓ $T = (t_1, \ldots, t_N)^T$
- **Goal :**
  - ✓ Get a prediction $\hat{t}$ for each new data $\hat{x}$

Polynomial shape



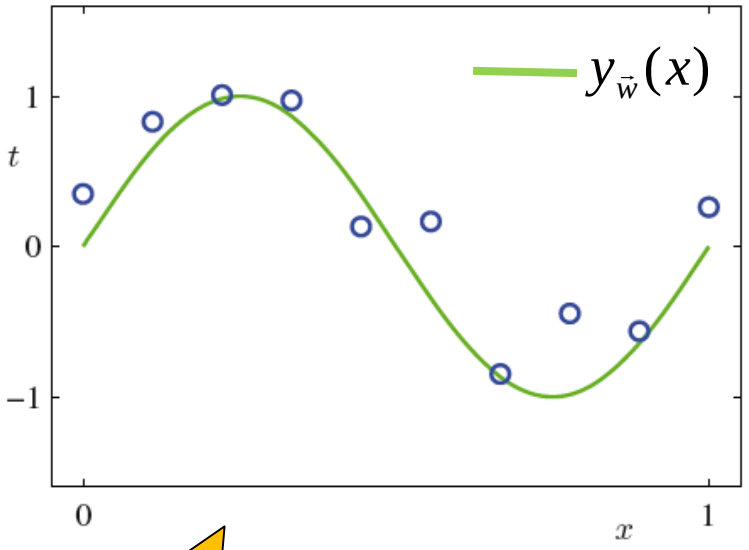
- Let's assume that our data has a **polynomial shape**

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

- $y_{\vec{w}}(x)$ is our **model**
  - ✓ It implicitly includes our hypothesies on the problem at hand
  - ✓ A model always has parameters that one needs to find (here $\vec{w}$)

Unknowns

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



Two unknowns

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

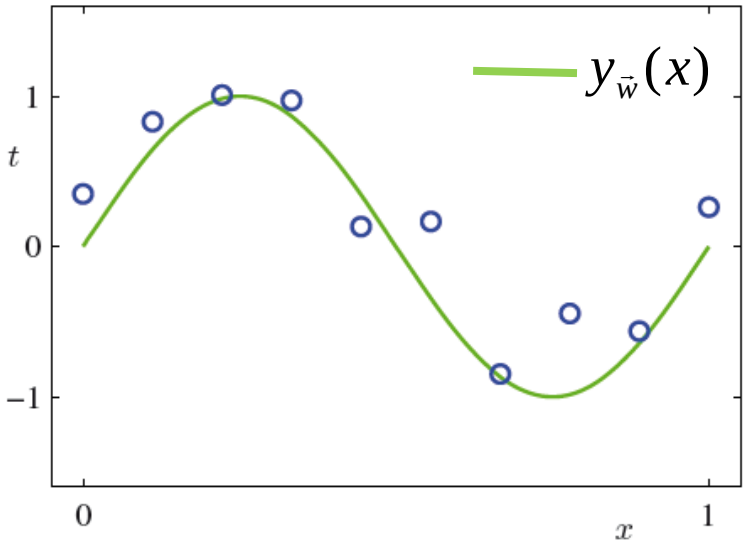
Parameters

$$M \in \mathrm{N}^{\geq 0}$$

Hyper-parameters

Training

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



Two unknowns

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

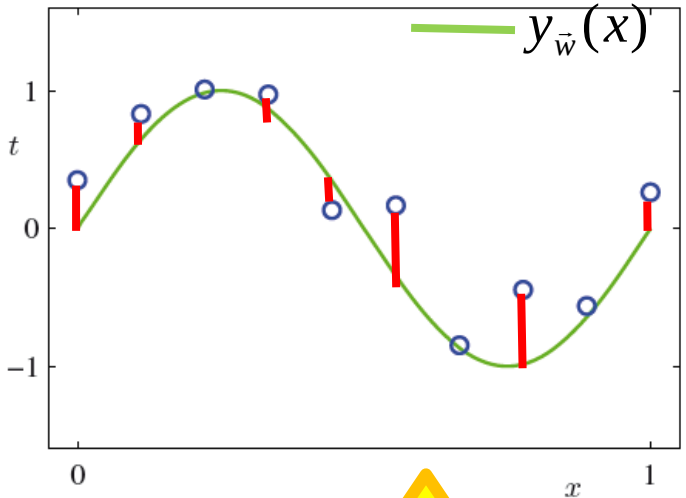
$$M \in \mathrm{N}^{\geq 0}$$

Training = find the « Best » w (and somtimes M) from X and T

Polynomial regression

- If ***M is given***, how can we find the best $\vec{w}$?

The « best » $\vec{w}$ is the one that minimizes the error (or the **loss**) on the training data



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

mean squared error (MSE)

- More formally

$$\vec{w} = \arg\min_{\vec{w}} E_D(\vec{w})$$

Hyperparameters

- How to find the best ***M***?

Hyperparameters cannot be **estimated** with classical **optimization algorithms** (gradient descent, Newton method, Simplex method, etc.) as for $\vec{w}$

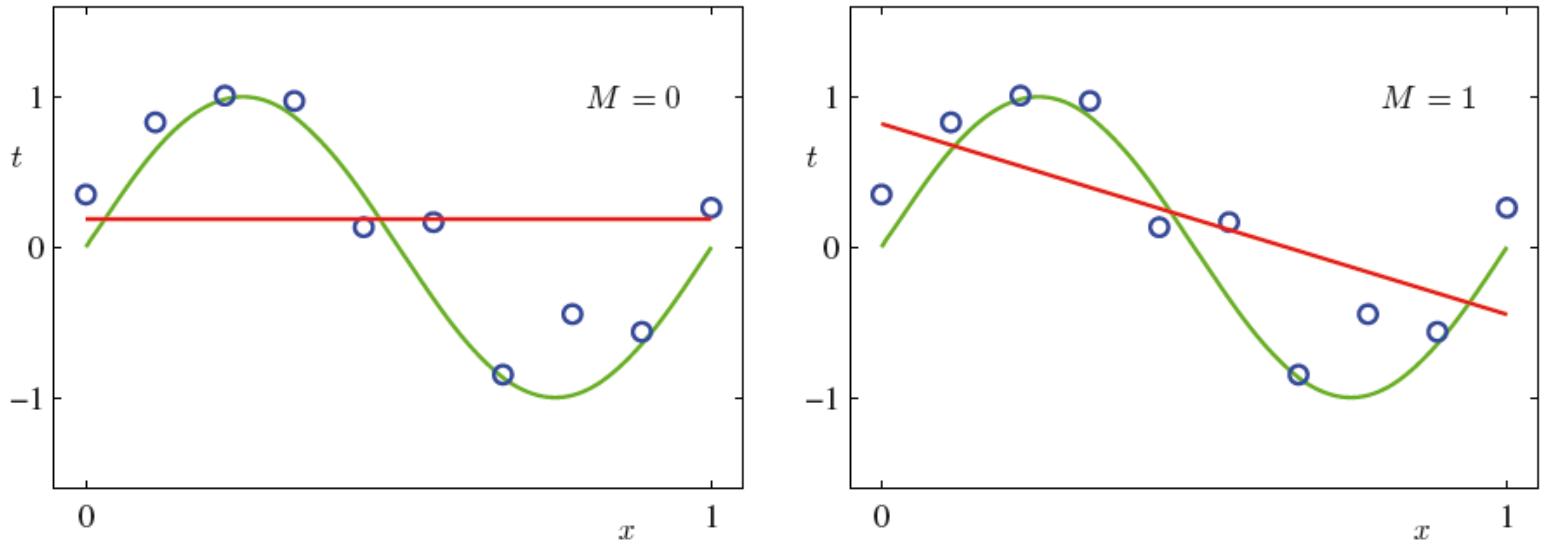
hyperparameters are often fixed manually.

BUT BEWARE!!! Their value greatly influence the **final results**.

Underfitting

$$M = 0 \Rightarrow y_{\vec{w}}(x) = w_0$$
$$M = 1 \Rightarrow y_{\vec{w}}(x) = w_0 + w_1 x$$

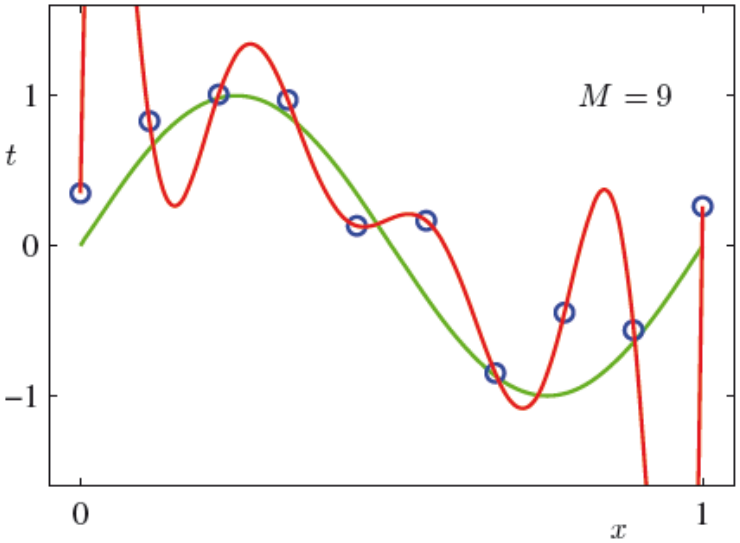
A small *M* gives a simplistic model that will **underfit** the data**.**



$$E_D(\vec{w}) \Rightarrow \text{large}$$
$$E_{D_{test}}(\vec{w}) \Rightarrow \text{large}$$

Overfitting

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

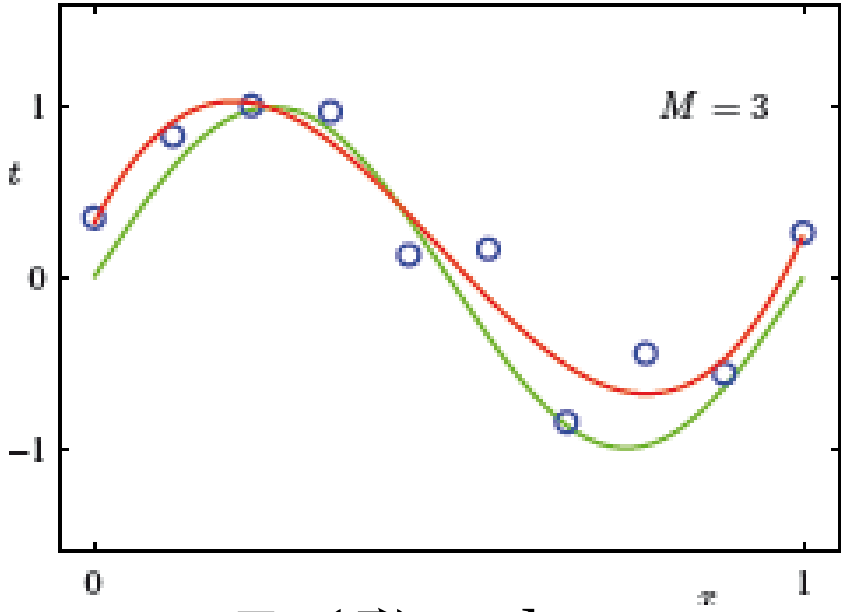


$$E_D(\vec{w}) \Rightarrow \text{VERY low}$$

$$E_{D_{test}}(\vec{w}) \Rightarrow \text{Large}$$

Over- and underfitting

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



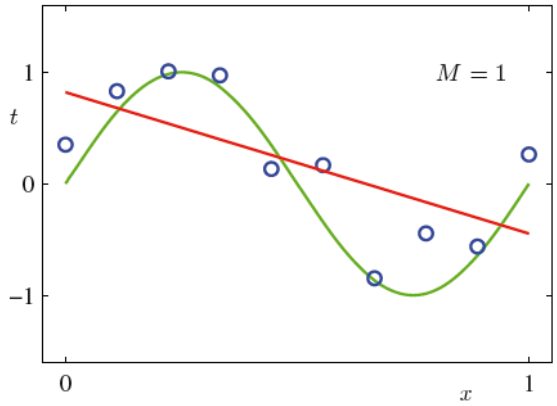
$$E_D(\vec{w}) \Rightarrow \text{low}$$

$$E_{D_{test}}(\vec{w}) \Rightarrow \text{low}$$

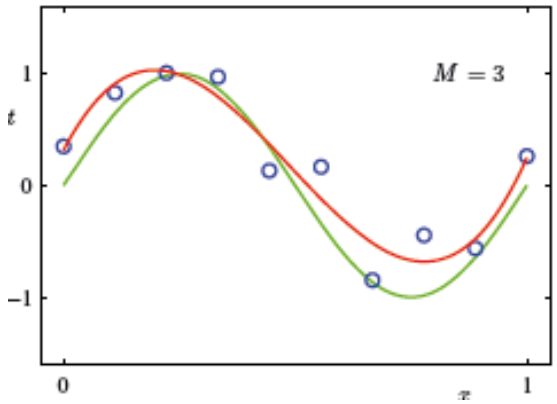
Hyperparameters often control the **capacity** of a model

Capacity: ability of a model to fit the training data

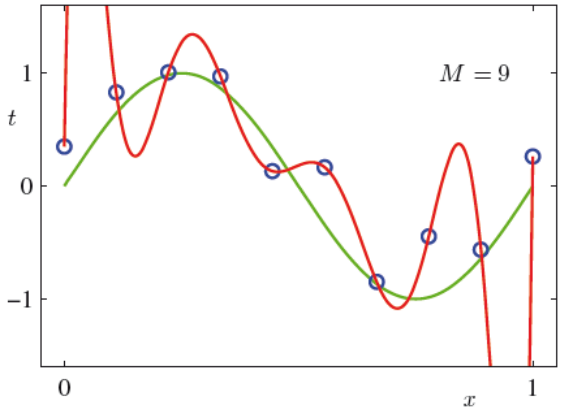
Low capacity



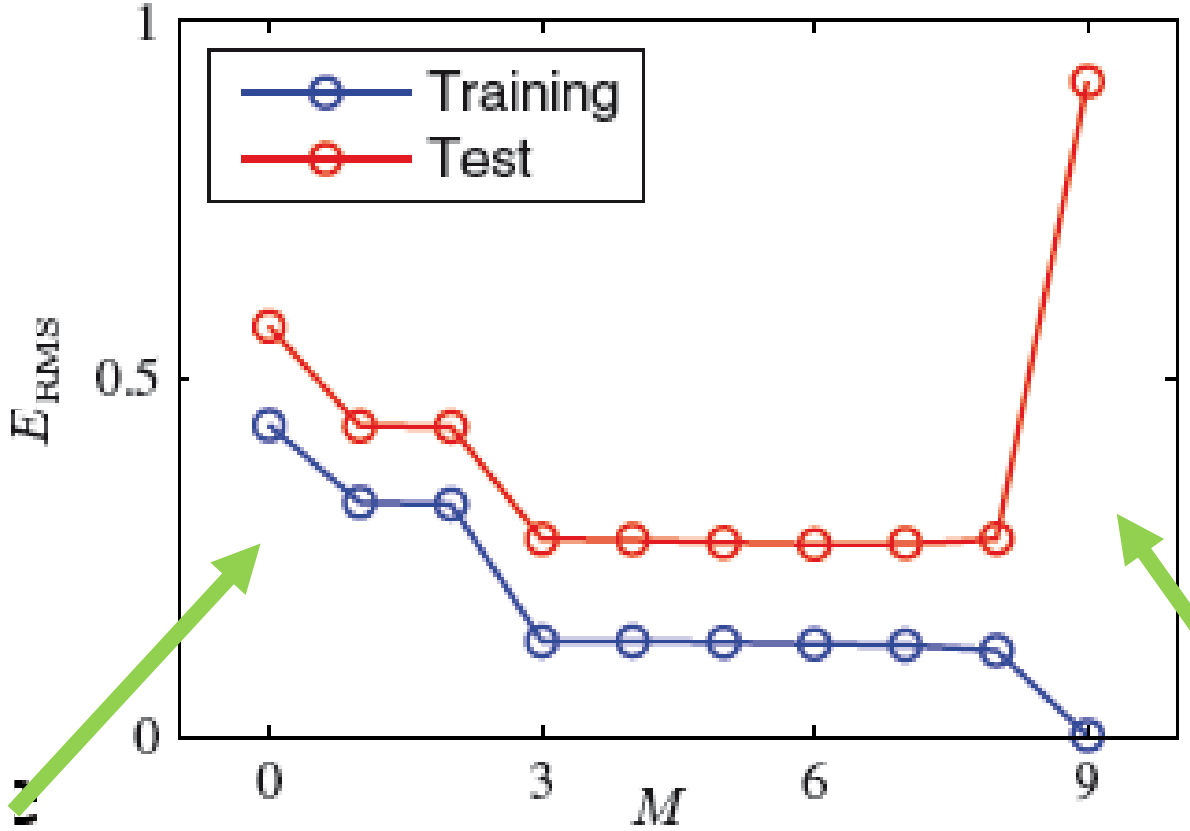
Medium capacity



Large capacity



Over- and underfitting



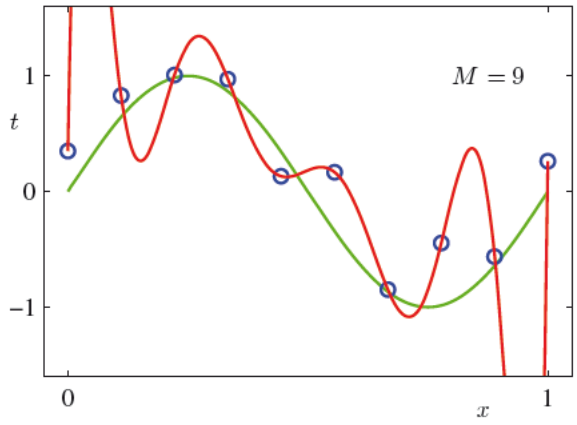
Underfitting

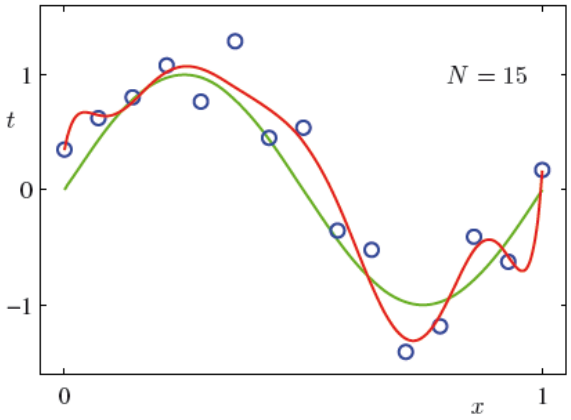
Overfitting

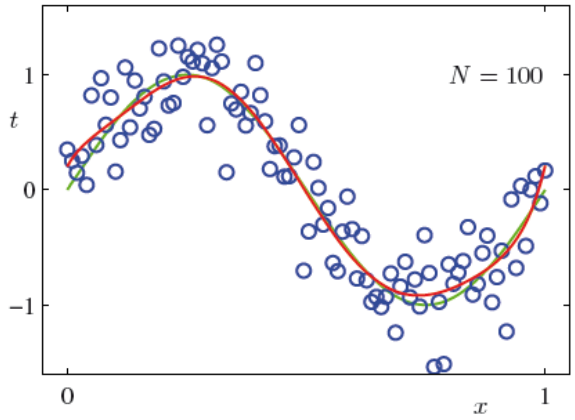
$$E_{RMS} = \sqrt{\frac{E(\vec{w})}{N}}$$

Generalization

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







If we have a **large M** and **few training data**, how do we prevent our model from 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 « M »
2. Reduce capacity by **regularization**

Exemple : penalise the **L2 norm**

Constante that controls regularization

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

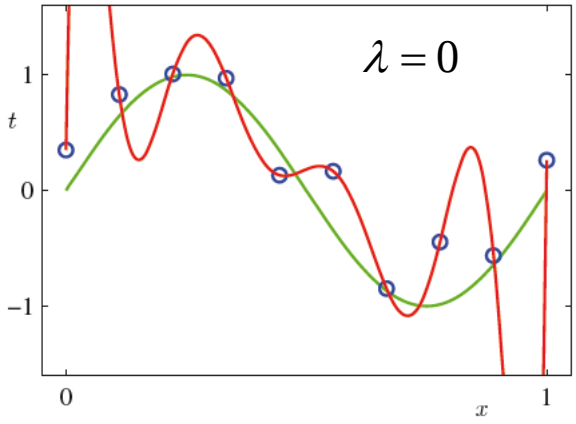
$$\|\vec{w}\|^2 = \vec{w}^{\mathrm{T}}\vec{w} = w_0^2 + w_1^2 + \ldots + w_M^2$$

Ridge model

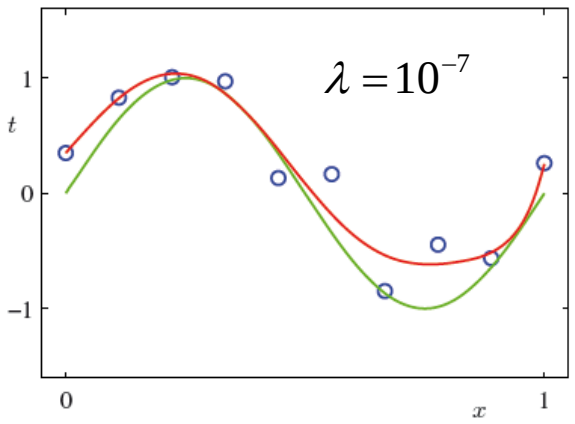
Regularization

Strong regularization= less capacity

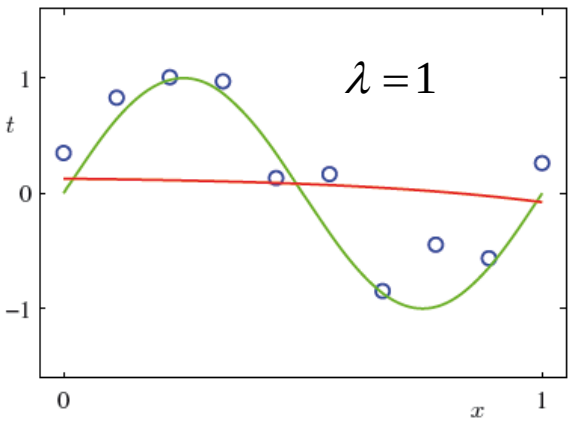
M=9



M=9

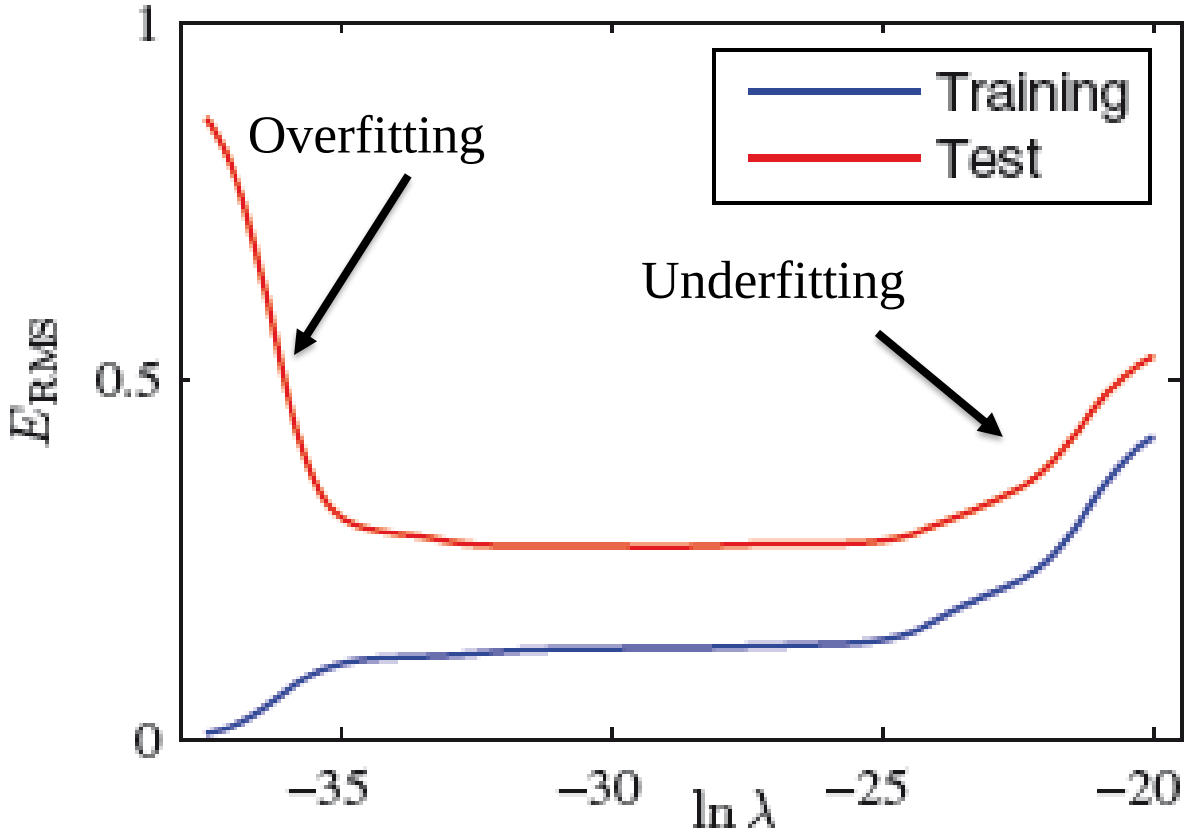


M=9



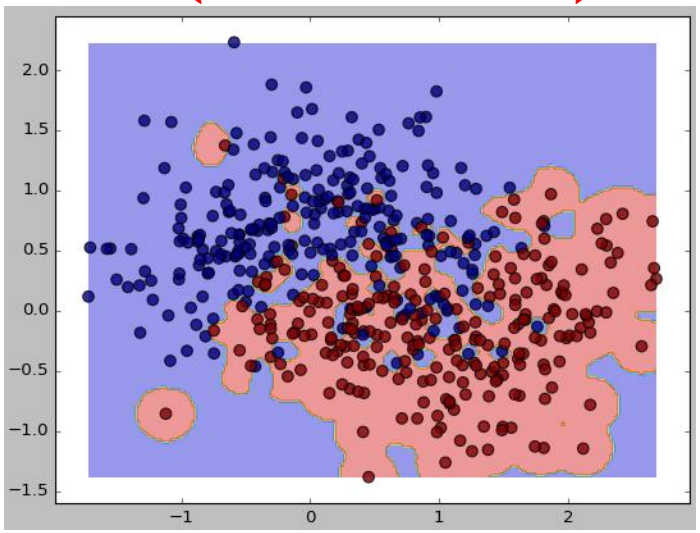
Regularization

Influence on the training and testing error

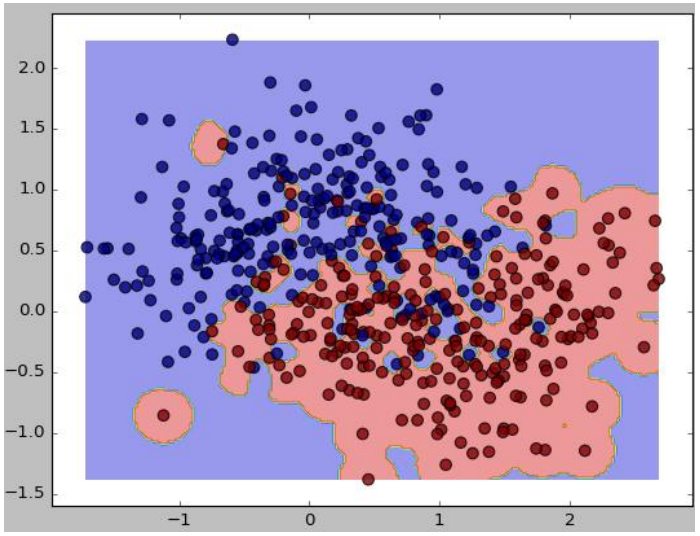


Over- and under-fitting also influence classification

Overfitting (Classification)

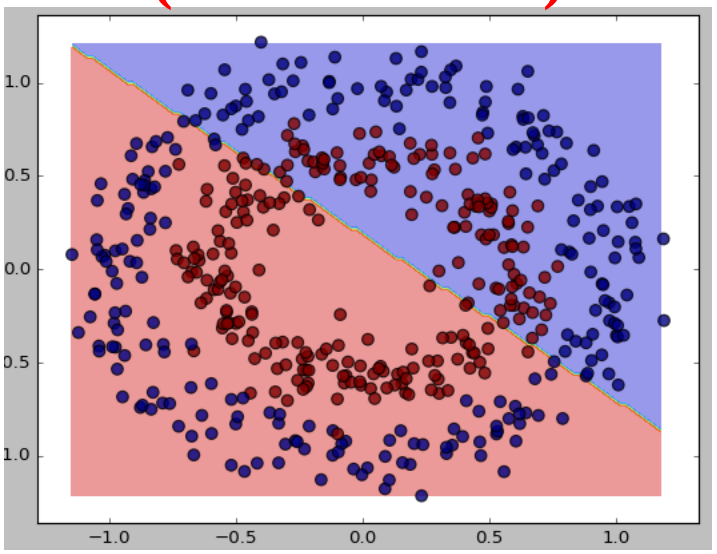


Training accuracy = 99.6%

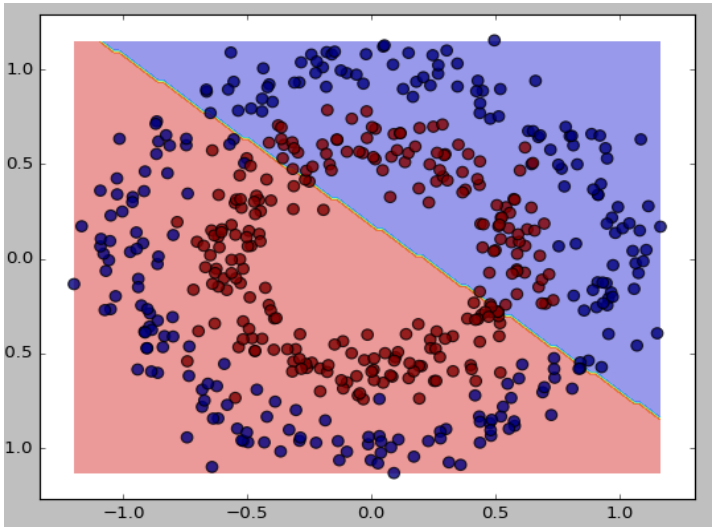


Testing accuracy = 78%

Underfitting (Classification)

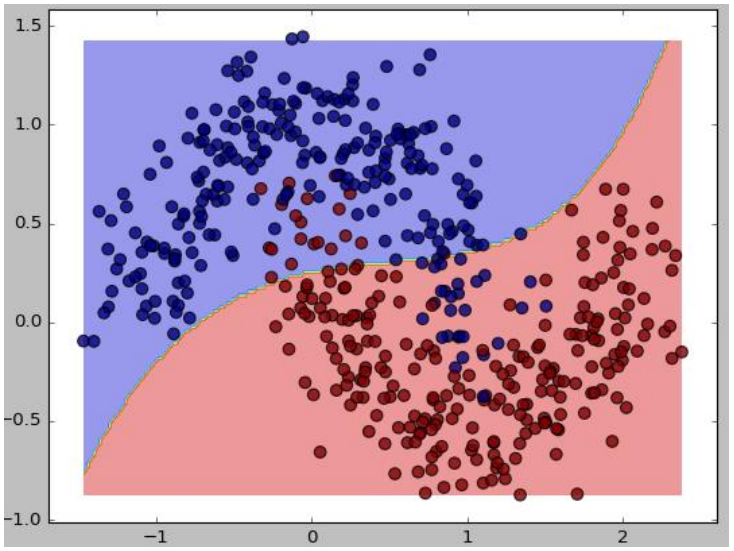


Training accuracy =52.2%

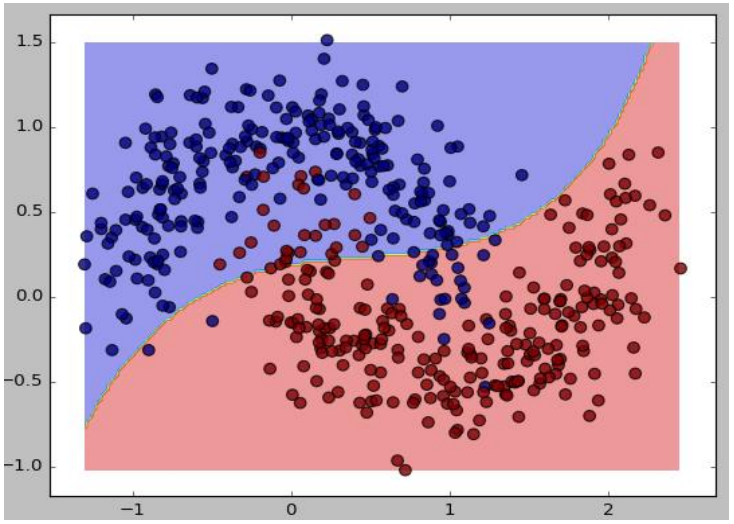


Testing accuracy = 51.2%

Could be better…

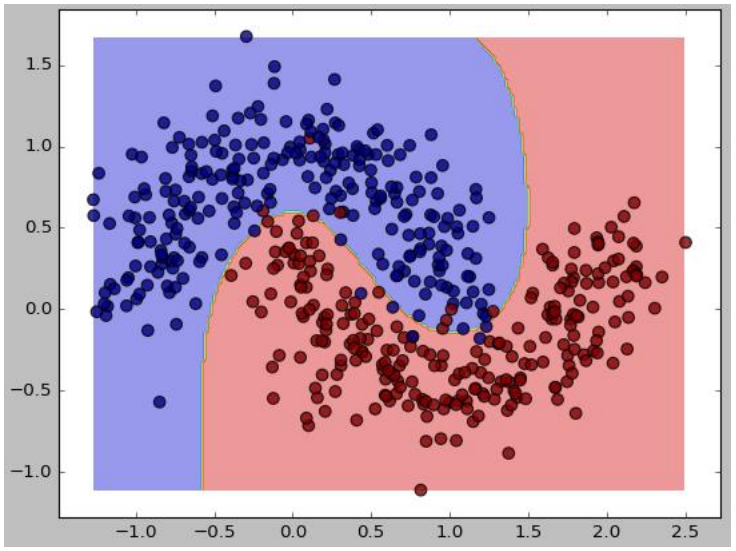


Training accuracy =82%

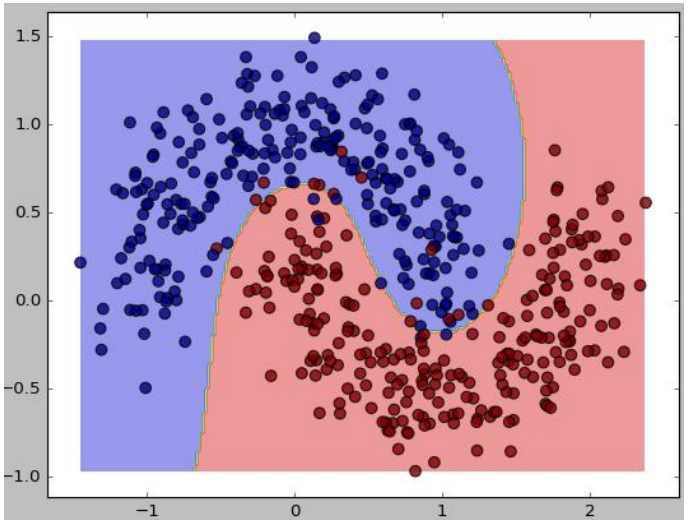


Testing accuracy = 80%

Wonderful !!!



Training accuracy =97.8%



Testing accuracy = 96.2%

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

$$\left\| \vec{w} \right\|^2 = \vec{w}^T \vec{w} = w_0^2 + w_1^2 + \ldots + w_M^2$$

Model selection

How to find the right hyper-parameters?

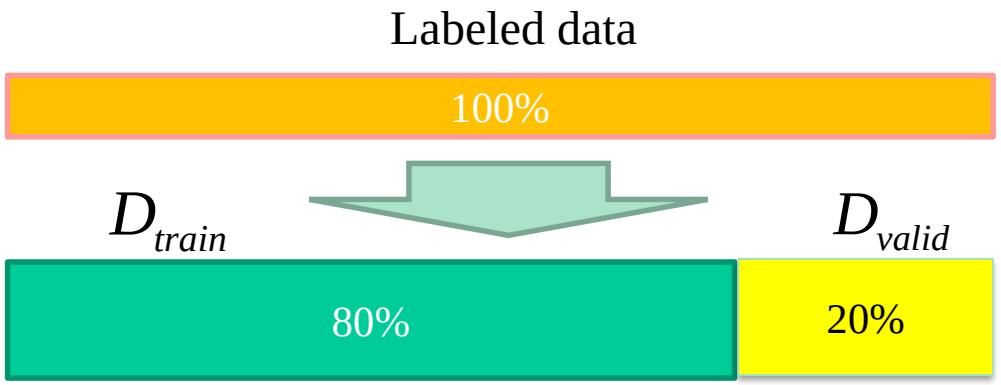
M and λ

How to find the right M and the right λ ?

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

k-fold cross-validation

FOR M from M_{min} to M_{max}
FOR λ from λ_{min} to λ_{max}
FOR j from 0 à K

Divide the labeled data in 2 groups => D_{train} D_{valid}

Train the model on D_{train}
Compute error on D_{valid}

Keep (M,λ) with the lowest **mean validation error**

K-fold cross-validation with K = 10

Mean validation error

STD

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

BEST!

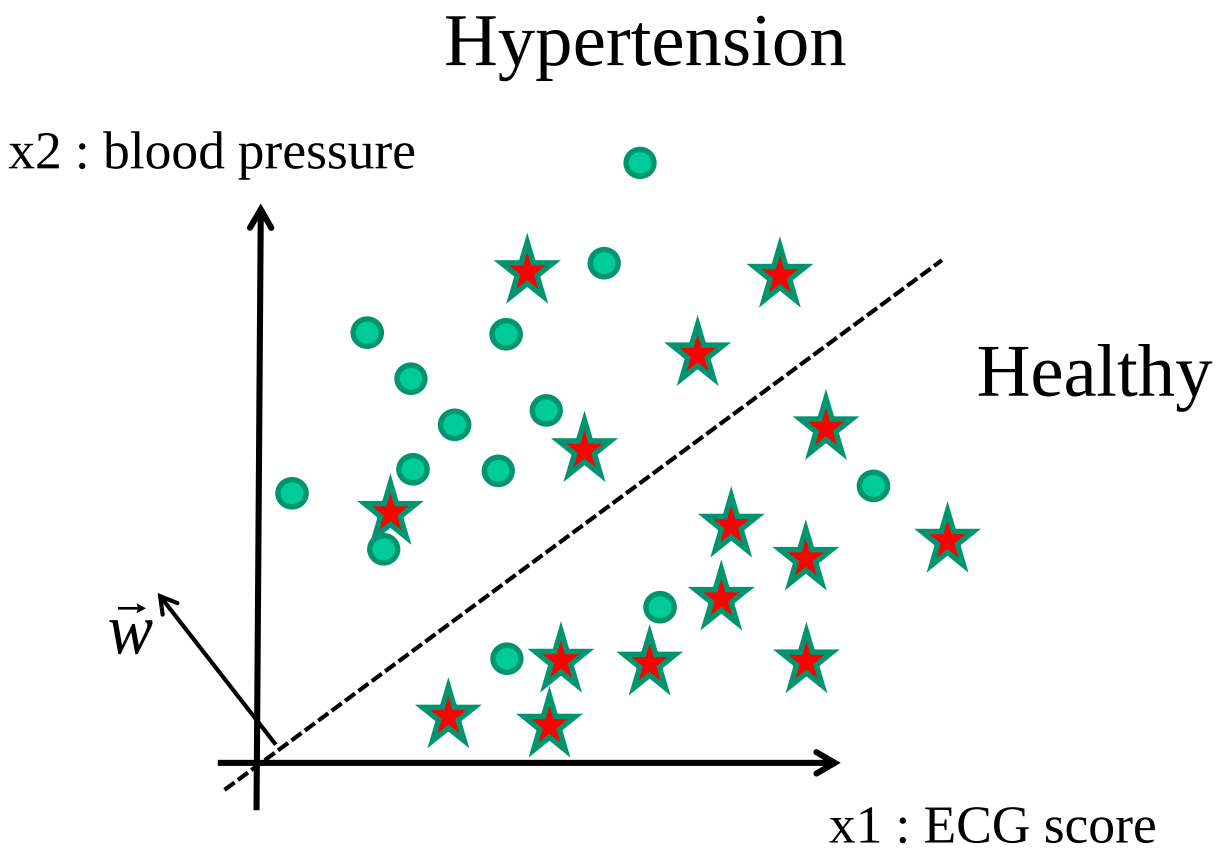
M=5,
$\lambda = 1$

In short

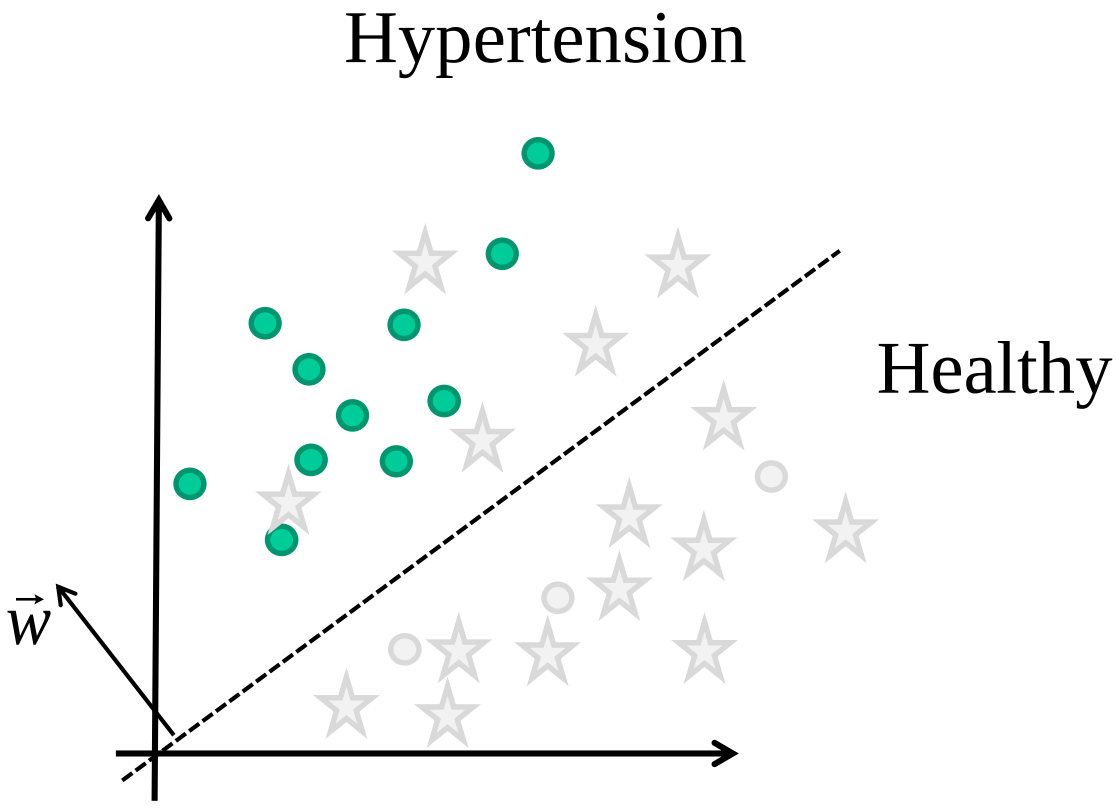
- ✓ The goal is to **train** a **model** on a **training dataset** with good **generalization** capabilites
- ✓ Has **hyper-parameters** that control the **capacity** of the model, choisis à l'aide d'une procédure de **sélection de modèle**
- ✓ mesure sa performance de **généralisation** sur un **ensemble 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

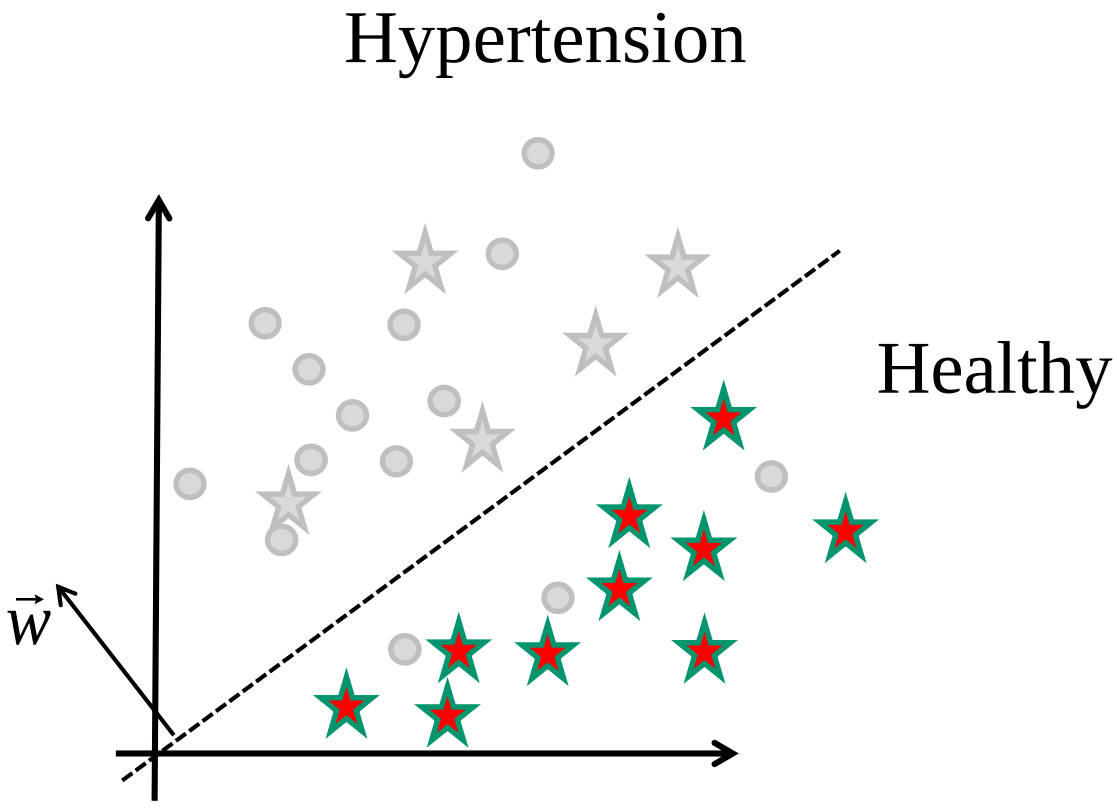
How to evaluate a model?



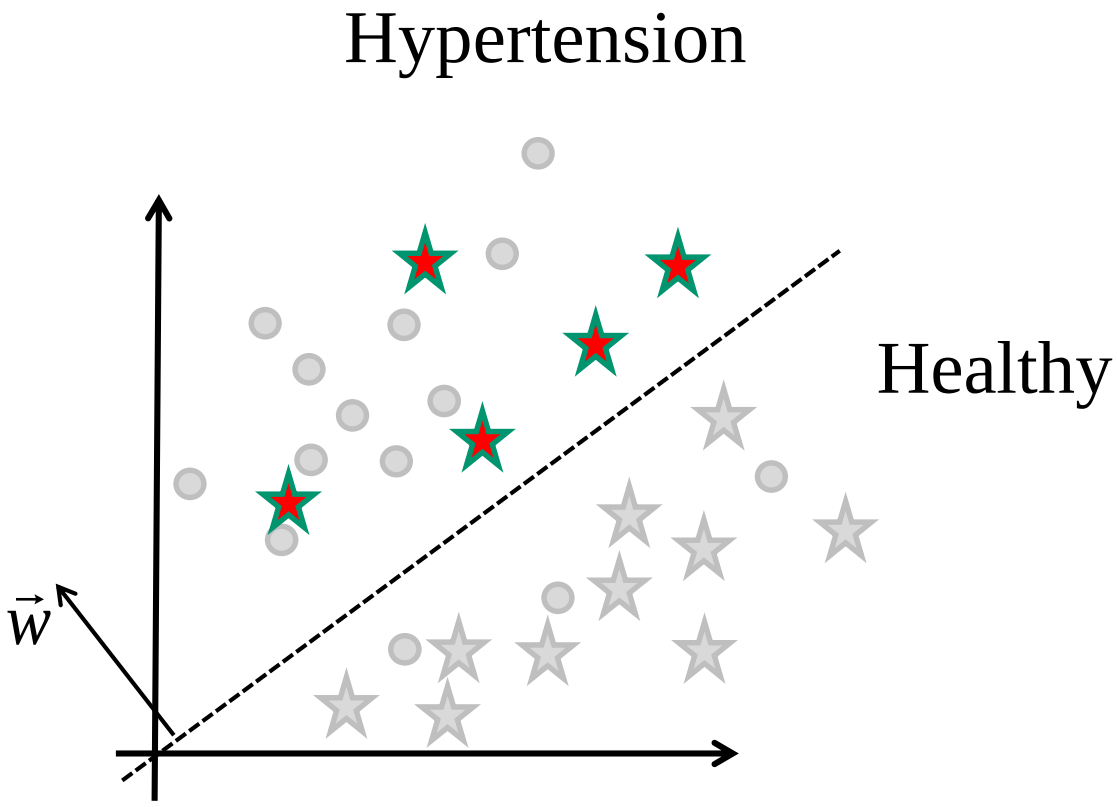
True positive (11)



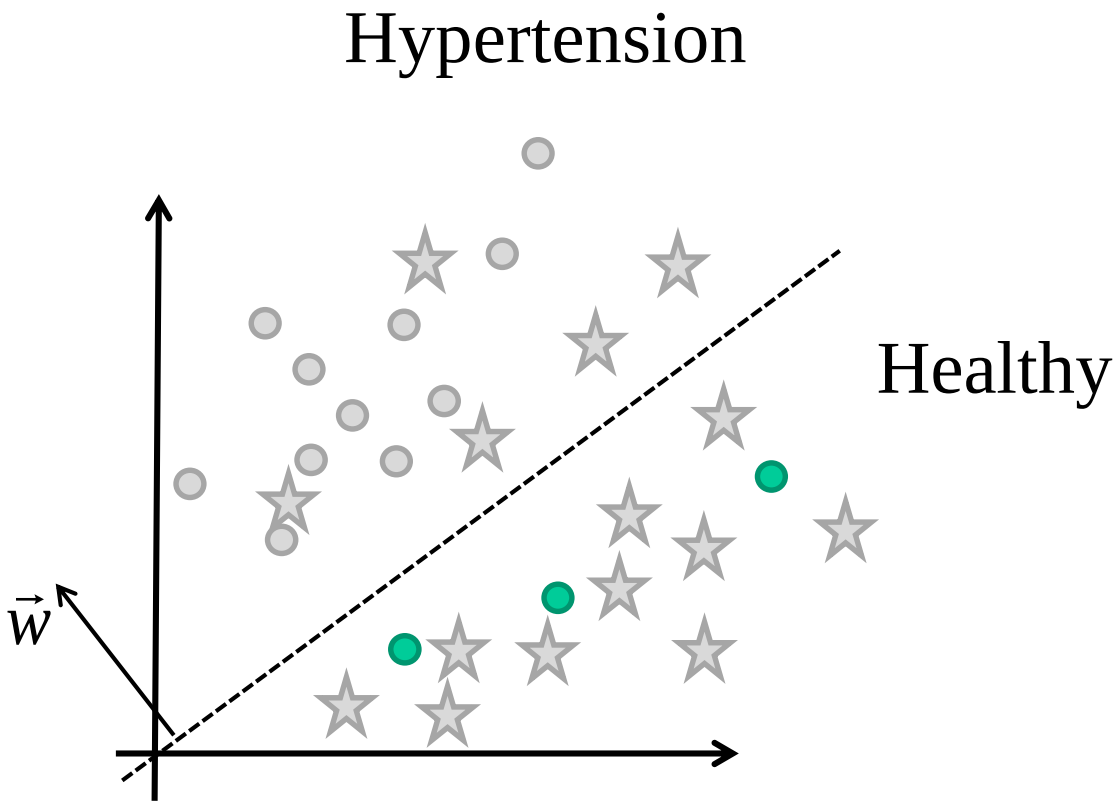
True negative (10)



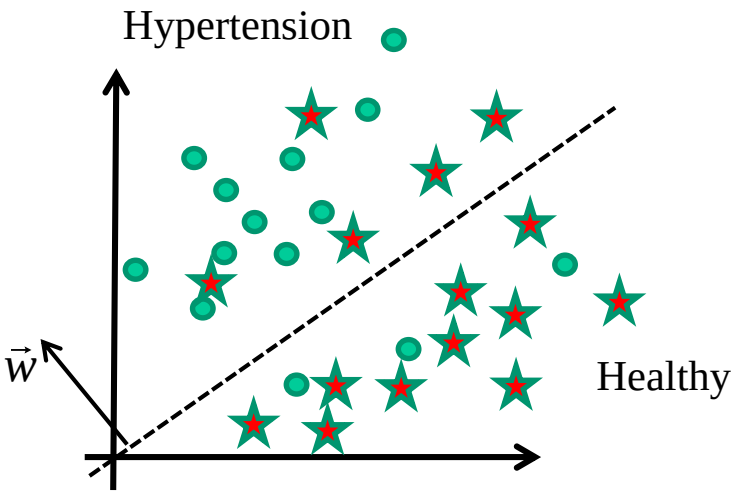
False positive (5)



False negative (3)



Confusion matrix



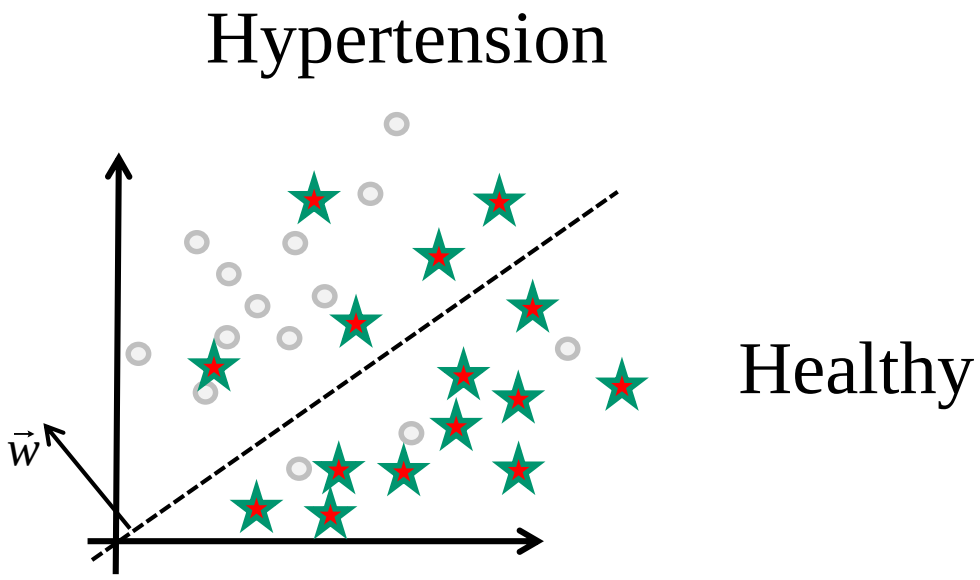
Ground truth

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

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

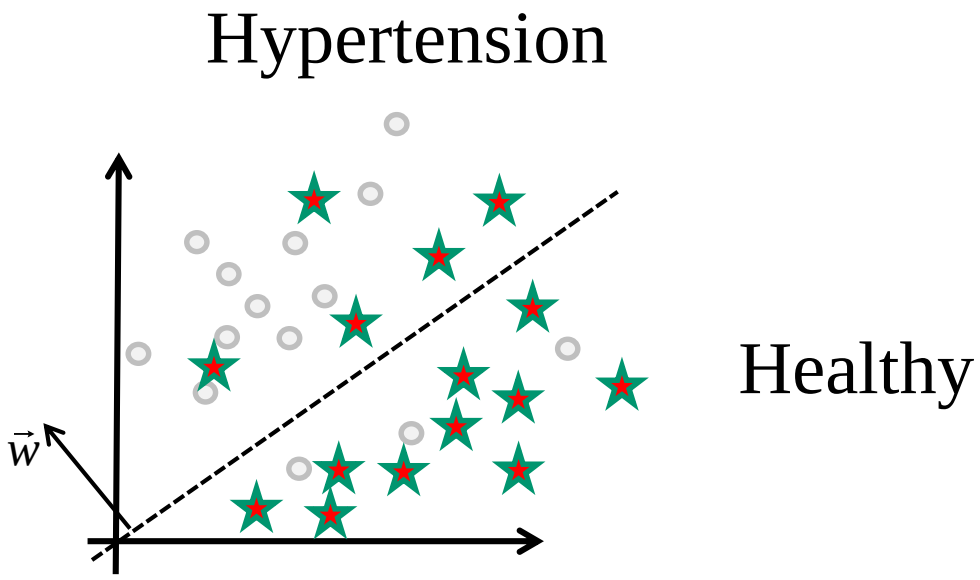
TP + FP = 16 = TOTAL # of patients classified +1
FN + TN = 13= TOTAL # of patients classified -1

False positive rate



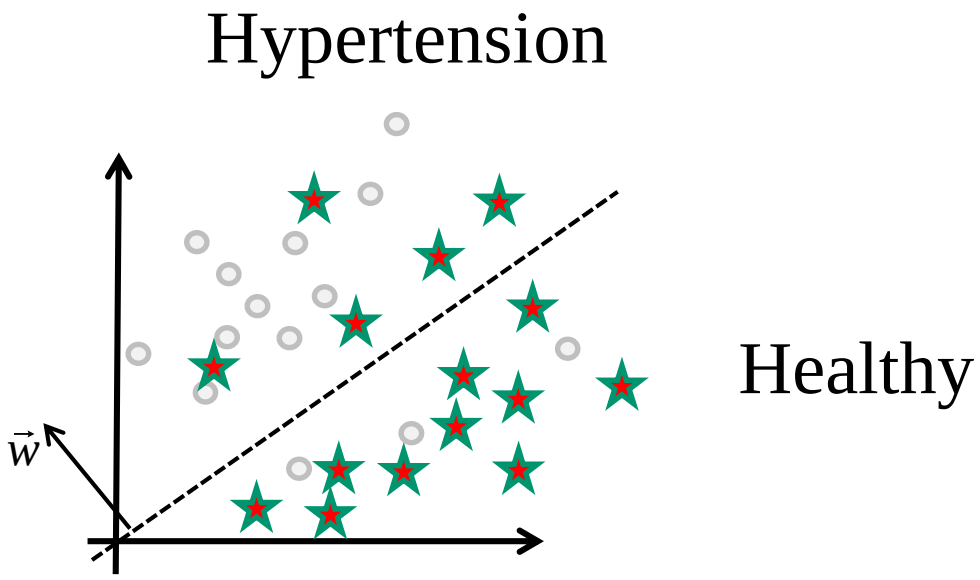
$$FPR = FP/(FP+TN) = 5/15$$

Specificity



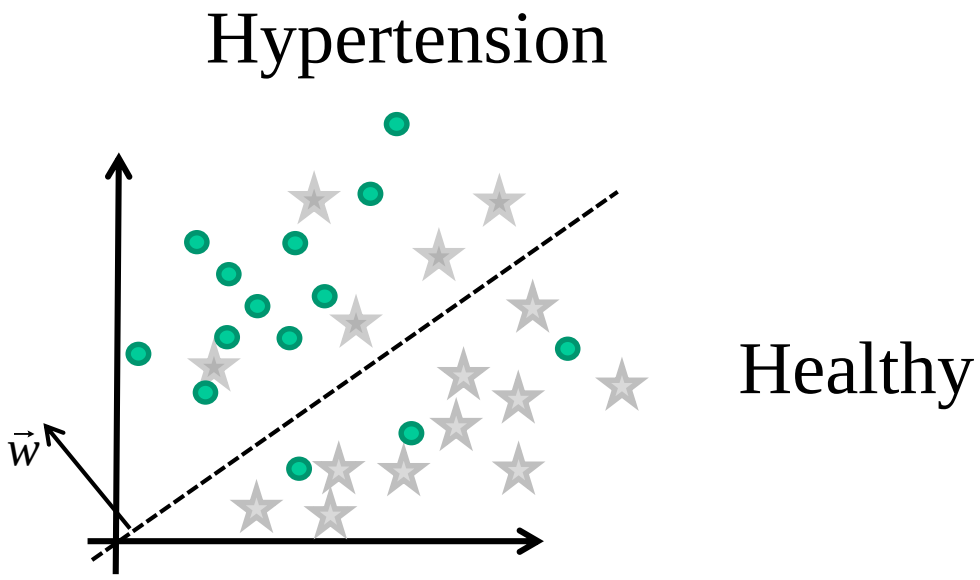
Sp = TN/(FP+TN)=11/15

Specificity



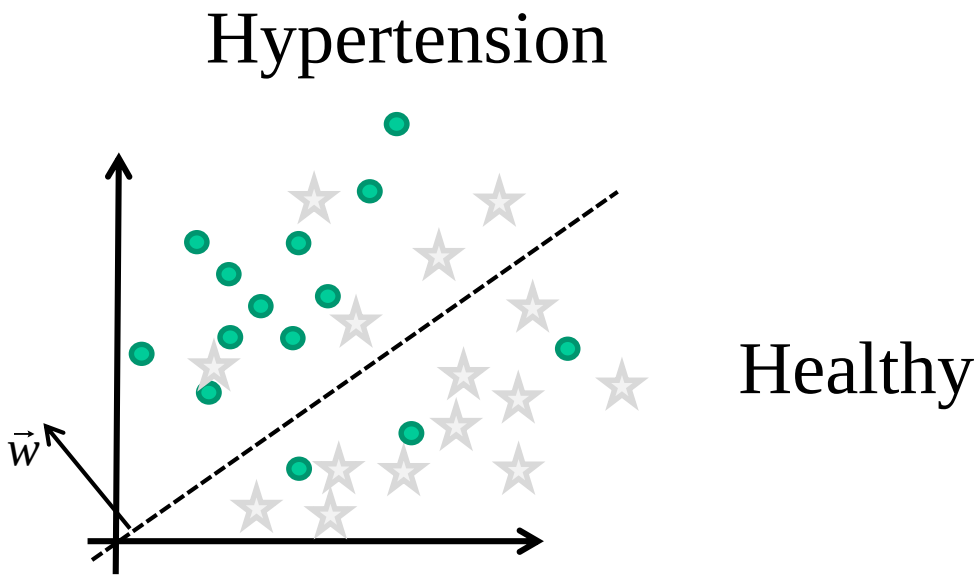
Sp = TN/(FP+TN)=1-FNR=11/15

False negative rate



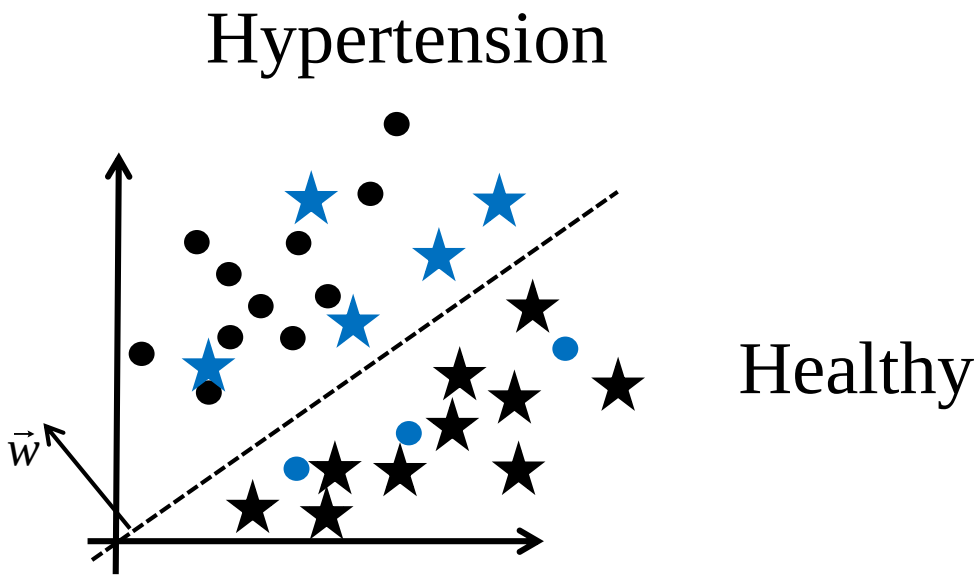
FNR = FN/(FN+TP) = 3/14

Recall



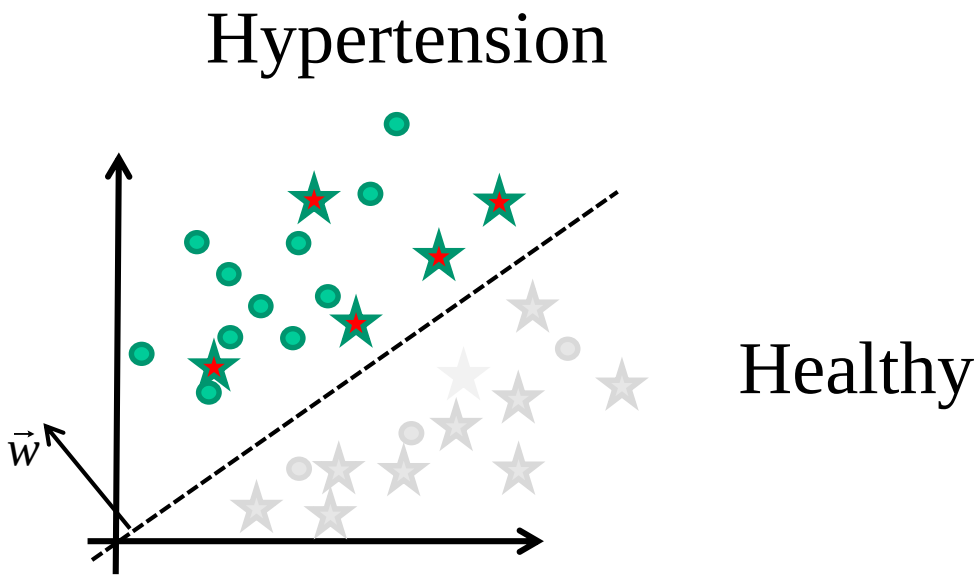
Re = TP/(FN+TP)=1-FNR=11/14

Accuracy



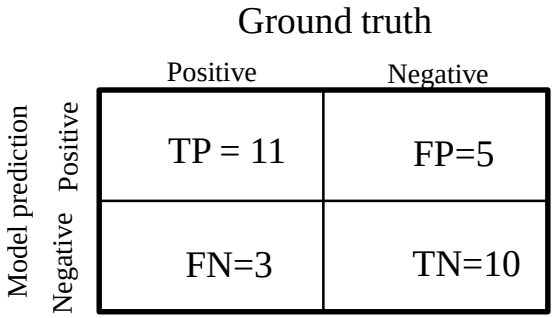
Rate of good classification = (TP+TN)/(FP+FN+TP+TN) = 21/29

Precision



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

In short



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

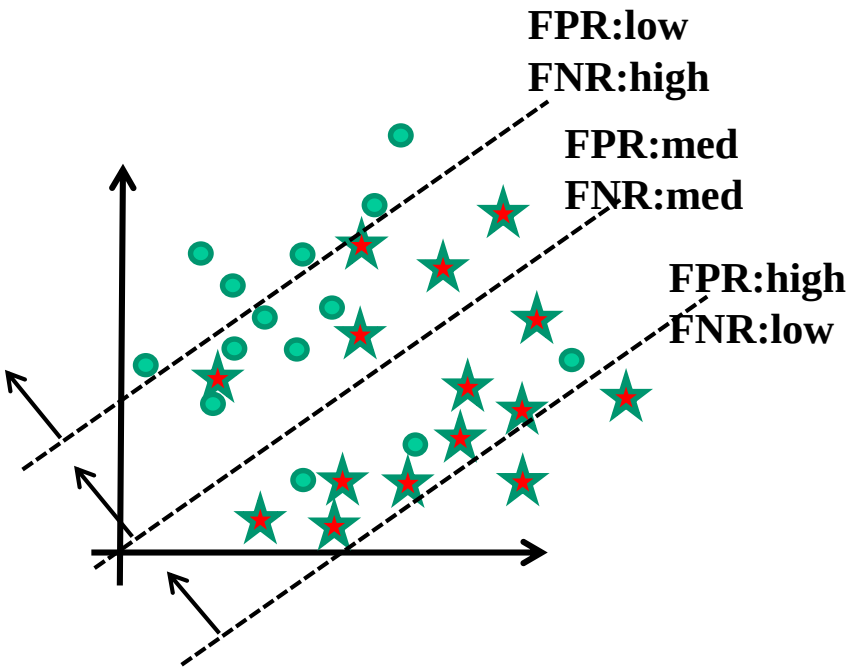
False positive rate = FP/(FP+TN) = 5/15
False negative rate = FN/(FN+TP) = 3/14

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

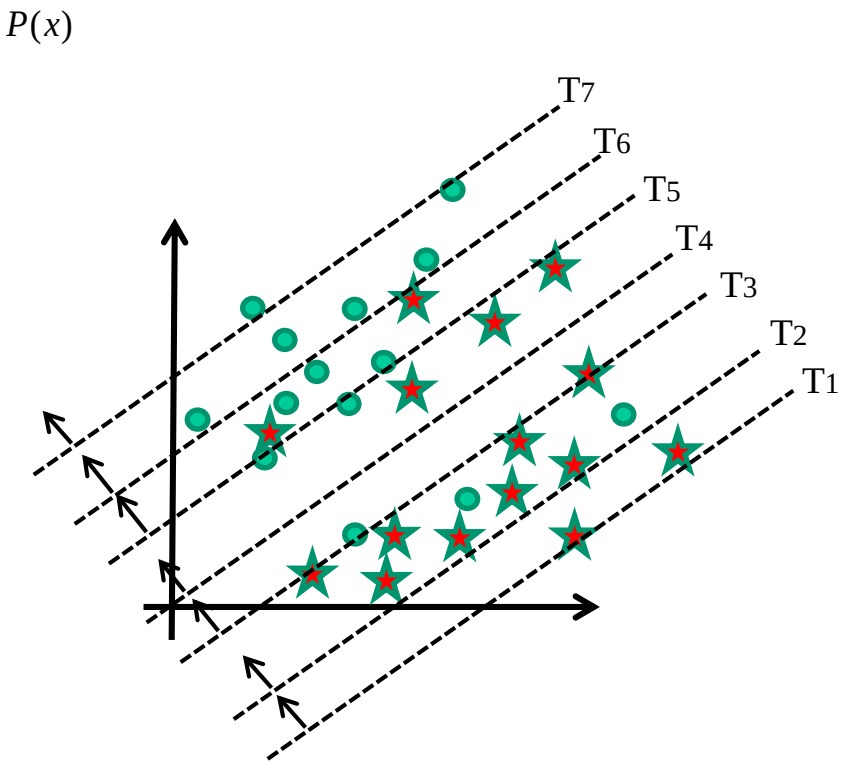
Different thresholds, different results

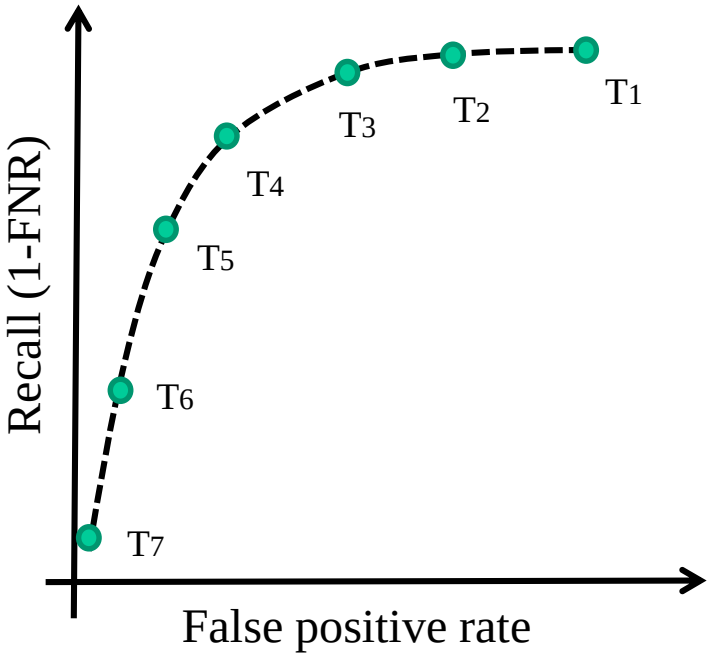
$P(x)$



ROC curves

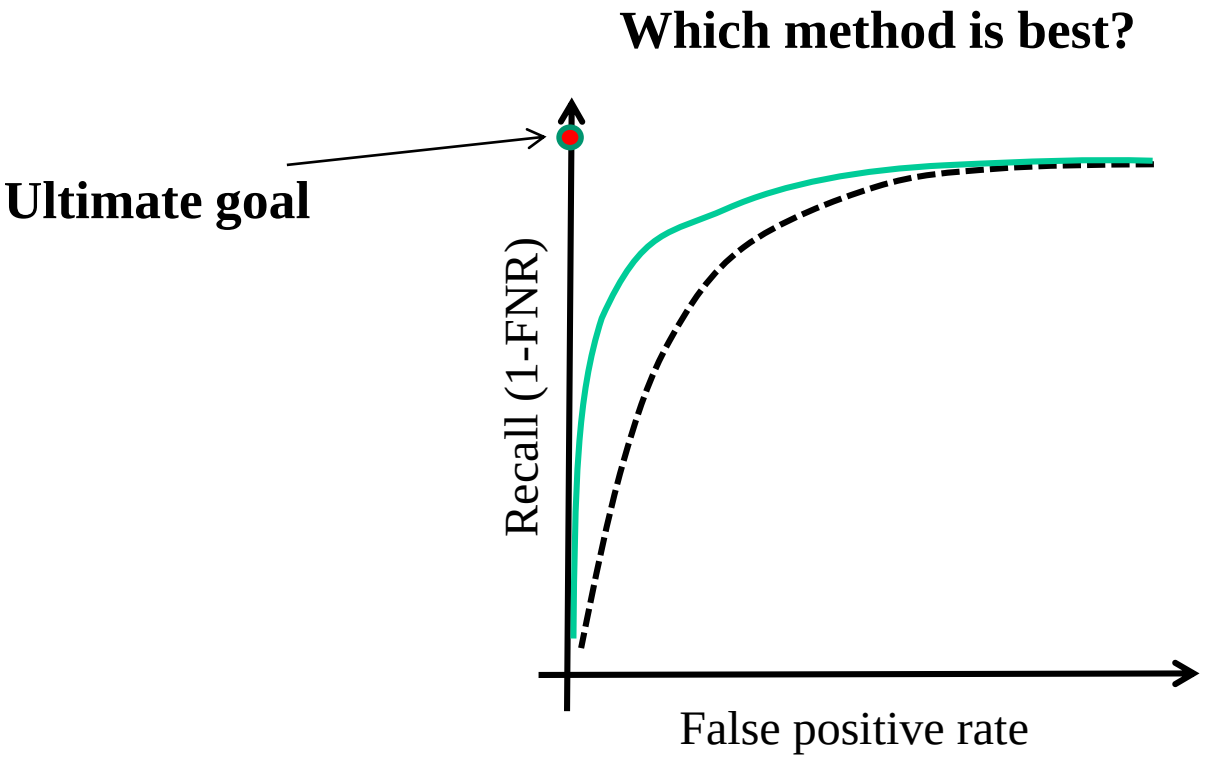
Compute Recall and FPR for **different thresholds**





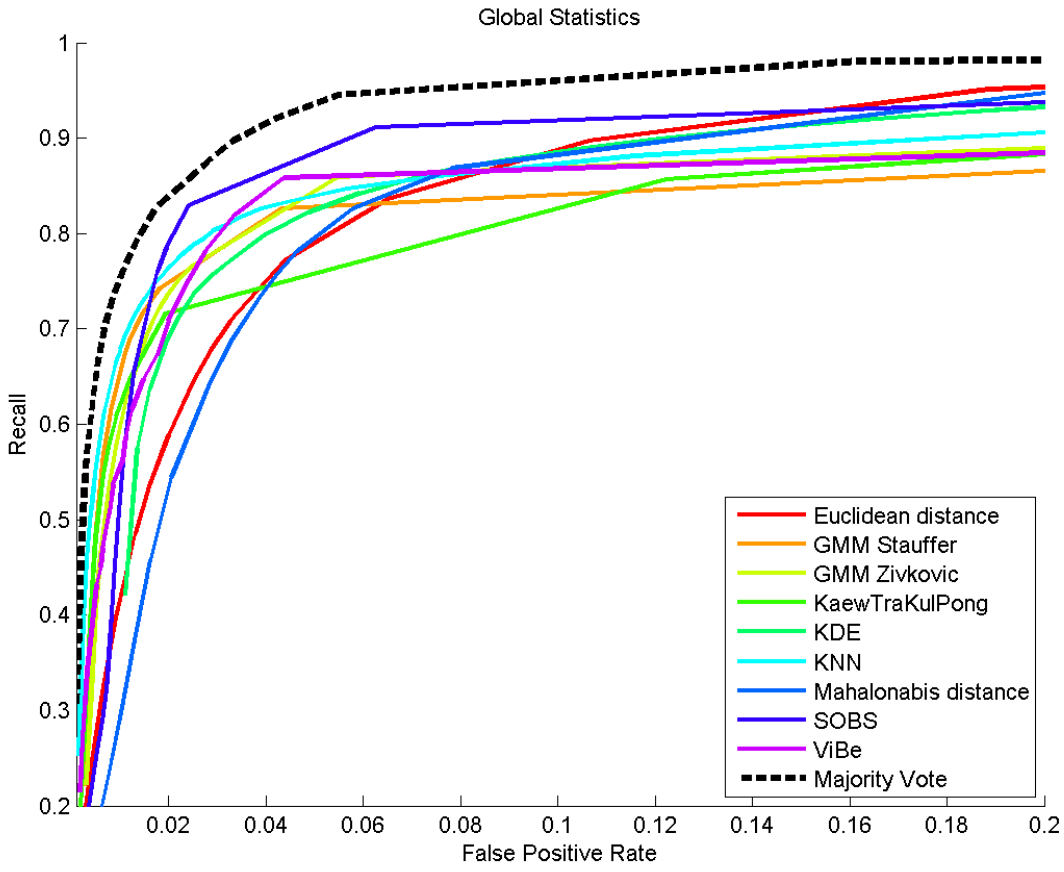
ROC curves

Good way for comparing methods



ROC curves

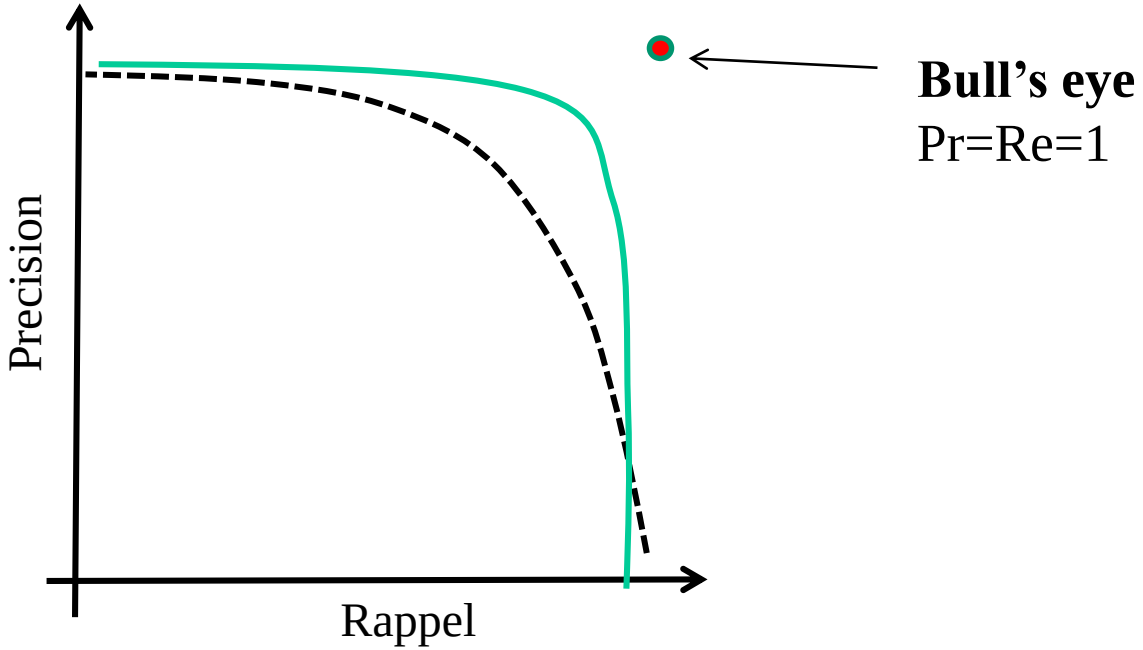
Example : 10 motion detection methods



Precision recall curve

Sampe spirit that the ROC curve

Which one is best?



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

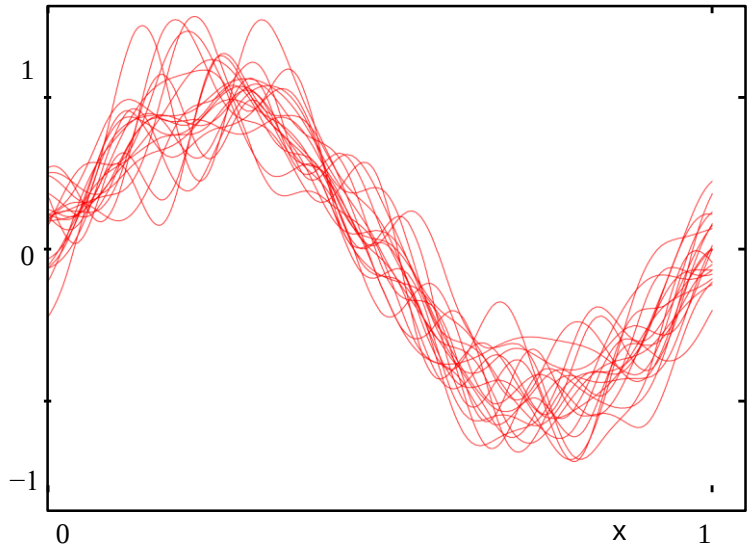
- ***Bagging*** : good for models with a **large capacity**
- ***Boosting*** : good for models with a **low capacity**

Bagging

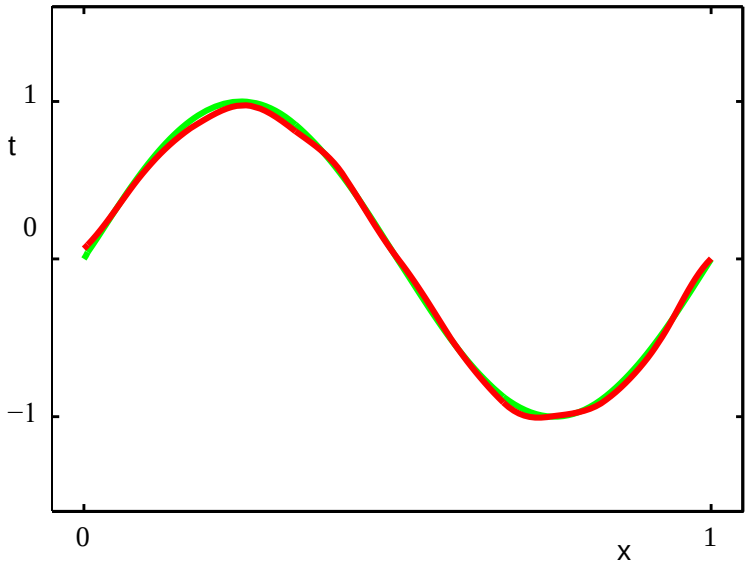
- A simple way of combining several models
  - FOR i = 1 to *m*
    - Train model $y_{W,i}(\vec{x})$
- ***Ensemble*** the *m* models
  - For **regression**
    - $y_{COM}(\vec{x}) = \frac{1}{m}\sum_{i=1}^{m} y_{W,i}(\vec{x})$
  - For a **classification**
    - **majority vote**

Ex: polynomial regression M=25

100 models trained on
100 different training sets



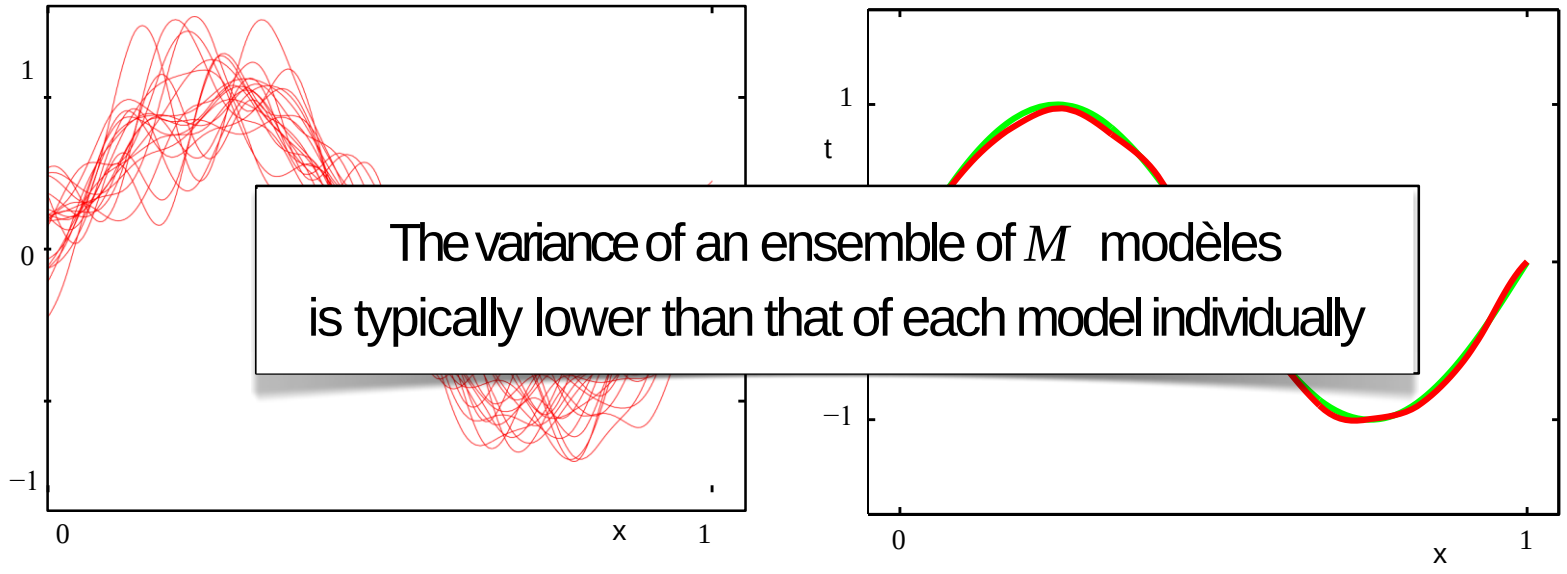
Ensemble of 100 models
Vs
True model



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

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

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!