

# Music Machine Learning

---

## 1 – Machine Learning

Master ATIAM - Informatique

Philippe Esling ([esling@ircam.fr](mailto:esling@ircam.fr))

Maître de conférences – UPMC

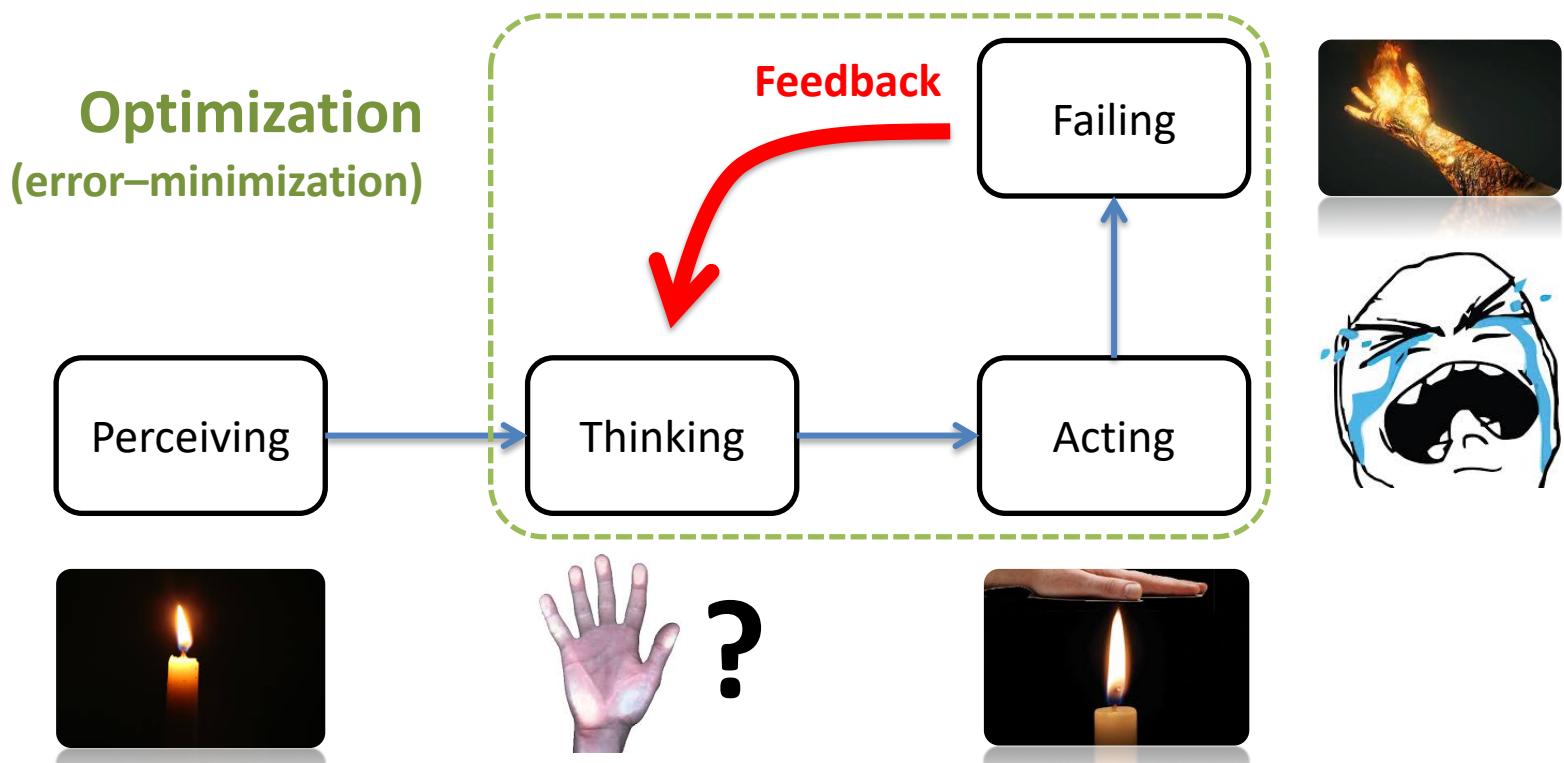
Equipe représentations musicales (IRCAM, Paris)



# Learning ?

---

# Learning ?



- **Goal-directed learning** = acquiring, modifying or reinforcing knowledge
  - Most problems can be summarized by a **trial-error process**
  - Is a nickname for **optimization** (through **error minimization**)

# Variational drums sequencer

C. Aouameur G. Hadjeres



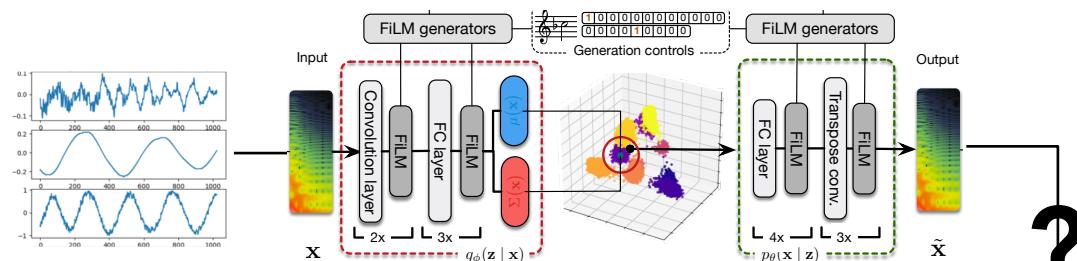
Sony CSL

## Variational learning for drums waveform

Drums are **mostly noisy**, therefore less adapted to spectrum

Can we perform end-to-end training with waveforms ?

Problems of WaveNet complexity and posterior collapse



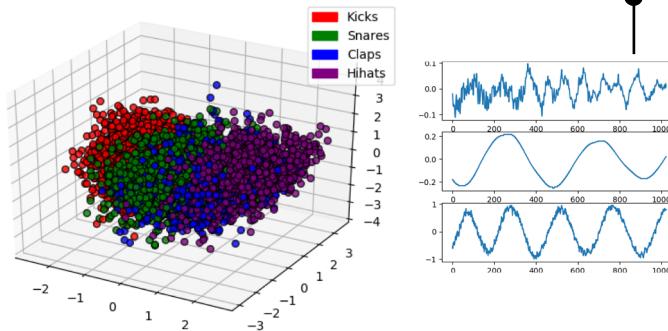
## Control space

User-friendly map

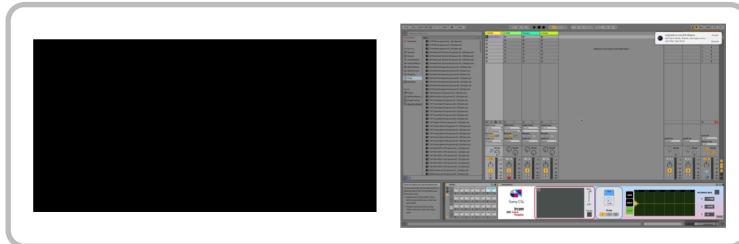
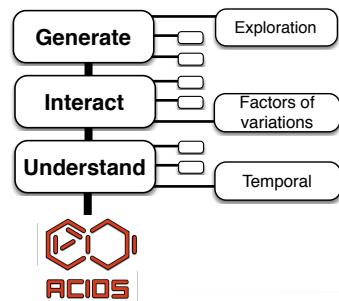
Allows **exploration**

**Generates waveform**

**Real-time inference**



Ableton plugin and web-based interface

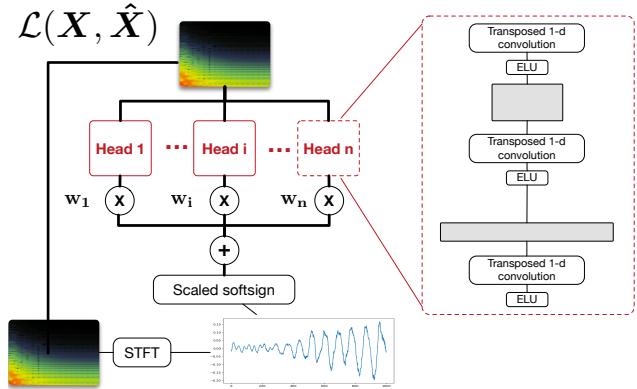


## Multi-head spectrum inversion

Fits the **natural properties of audio** signal

Allows a feed-forward spectrum inversion

Heads **unsupervisedly split into bands**



## Spectral convergence losses

$$\|\|\text{STFT}(s)\| - \|\text{STFT}(\hat{s})\|\|_F / \|\|\text{STFT}(s)\|\|_F,$$

$$\|\log(|\text{STFT}(s)| + \epsilon) - \log(|\text{STFT}(\hat{s})| + \epsilon)\|_1,$$

## Instantaneous phase loss

$$\left\| \frac{\partial}{\partial t} \phi(\text{STFT}(s)) - \frac{\partial}{\partial t} \phi(\text{STFT}(\hat{s})) \right\|_1,$$

Arik, S. Ö., Jun, H., & Diamos, G. (2019). Fast spectrogram inversion using multi-head convolutional neural networks. *IEEE Signal Processing Letters*, 26(1), 94-98.

## Real-time version demonstration

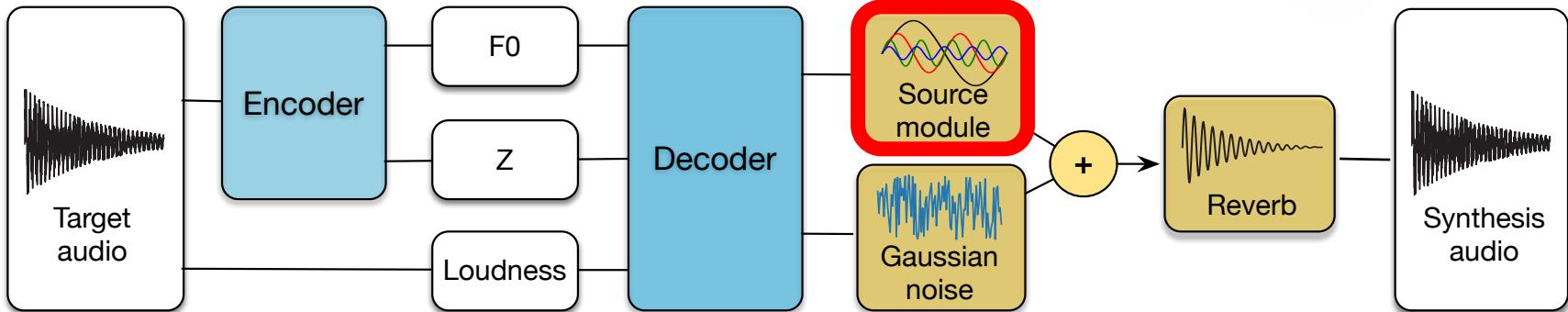
See you at the workshop ☺

# Differentiable Digital Signal Processing

A. Caillon



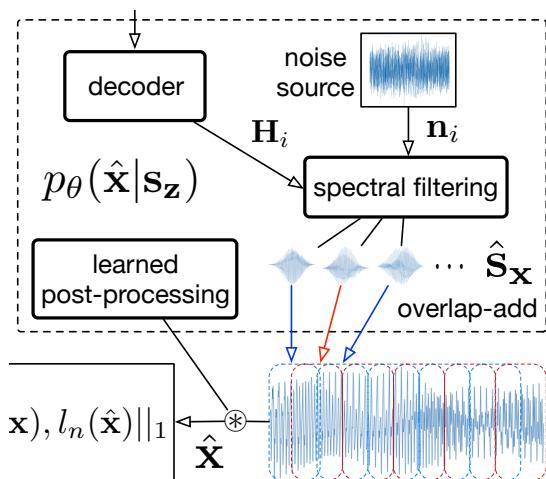
Engel, J. Hantrakul, L. Gu, C. and Roberts, A. – Differentiable Digital Signal Processing ICLR 2020



We can train a control model over it

Original model too restrictive

We implement a granular subtractive



Take traditional signal processing

And make it differentiable

Transfer results (vanilla)



Voice → Violin



Voice → Screech

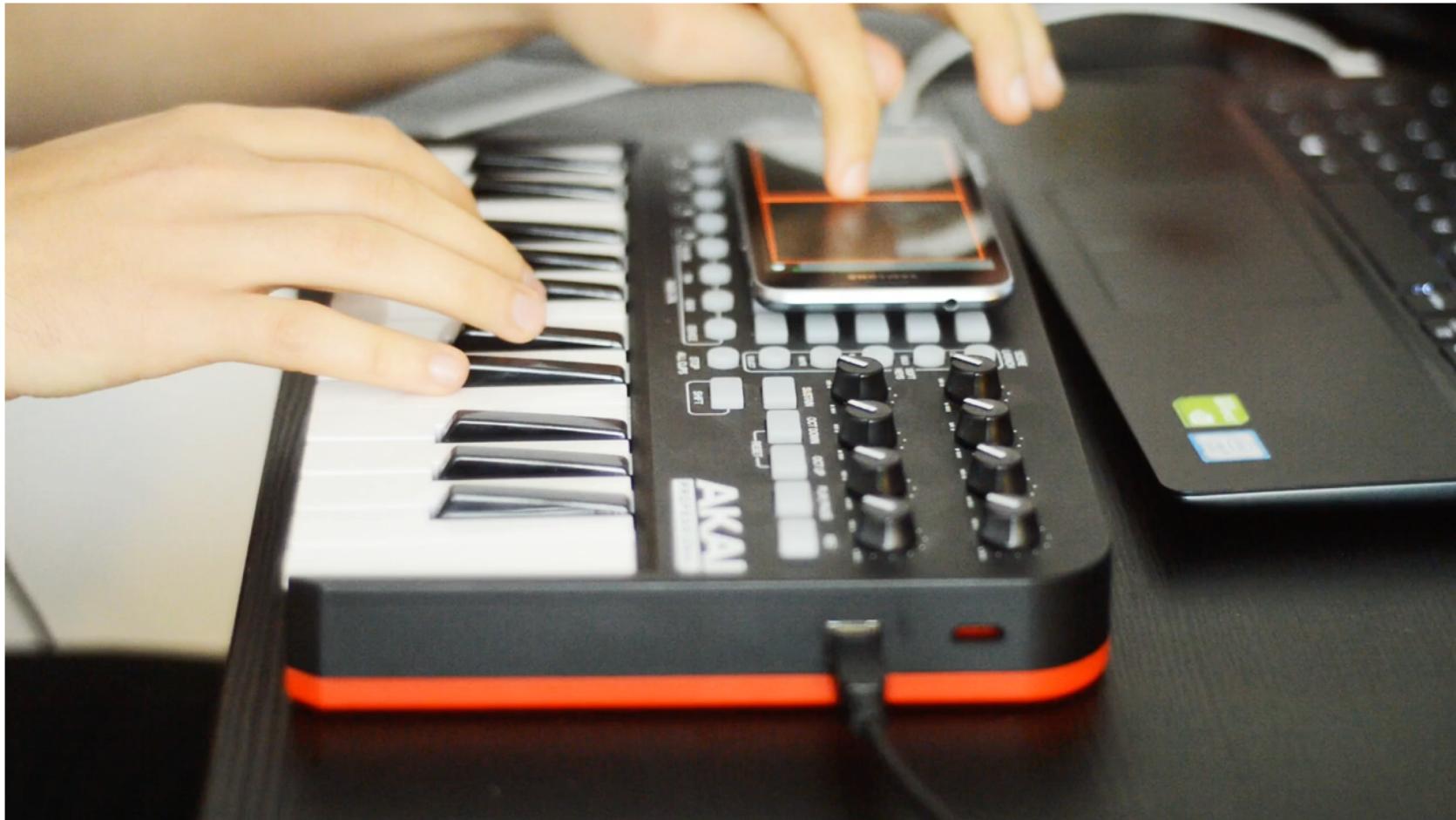


**Get Screechy**  
Get Lucky - voice transfer  
to screech and drills dataset

# Demo video

---

A. Caillon

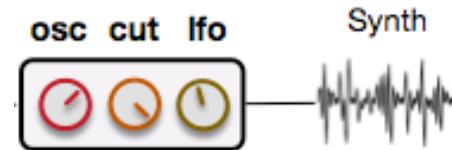


# Universal audio synthesis control



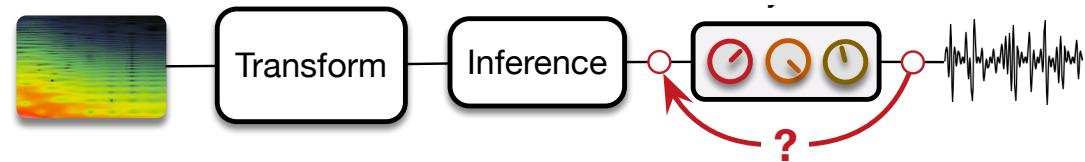
## Audio synthesizer are complex generation functions

Impact of parameters is non-linear and complex  
Needs expert knowledge and hard work  
Does not correlate to perception



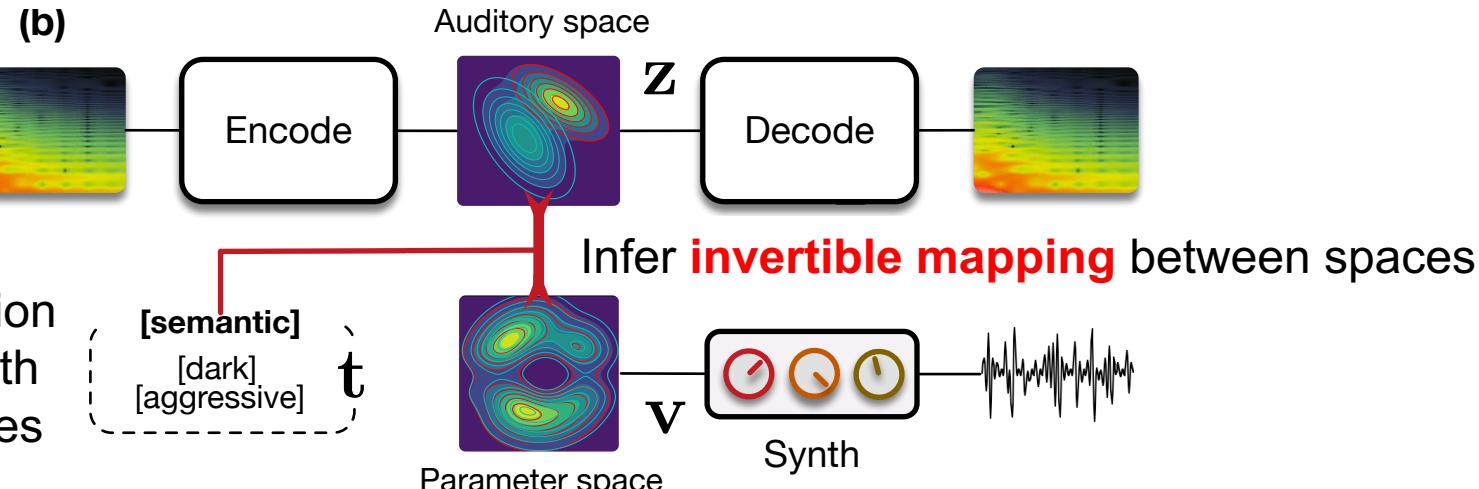
## Parameters inference models

Feed-forward inference  
But non-differentiable synth  
Also just input a waveform



## Our proposal = Universal synthesizer control with disentangling flows

Organize the auditory space of possibilities

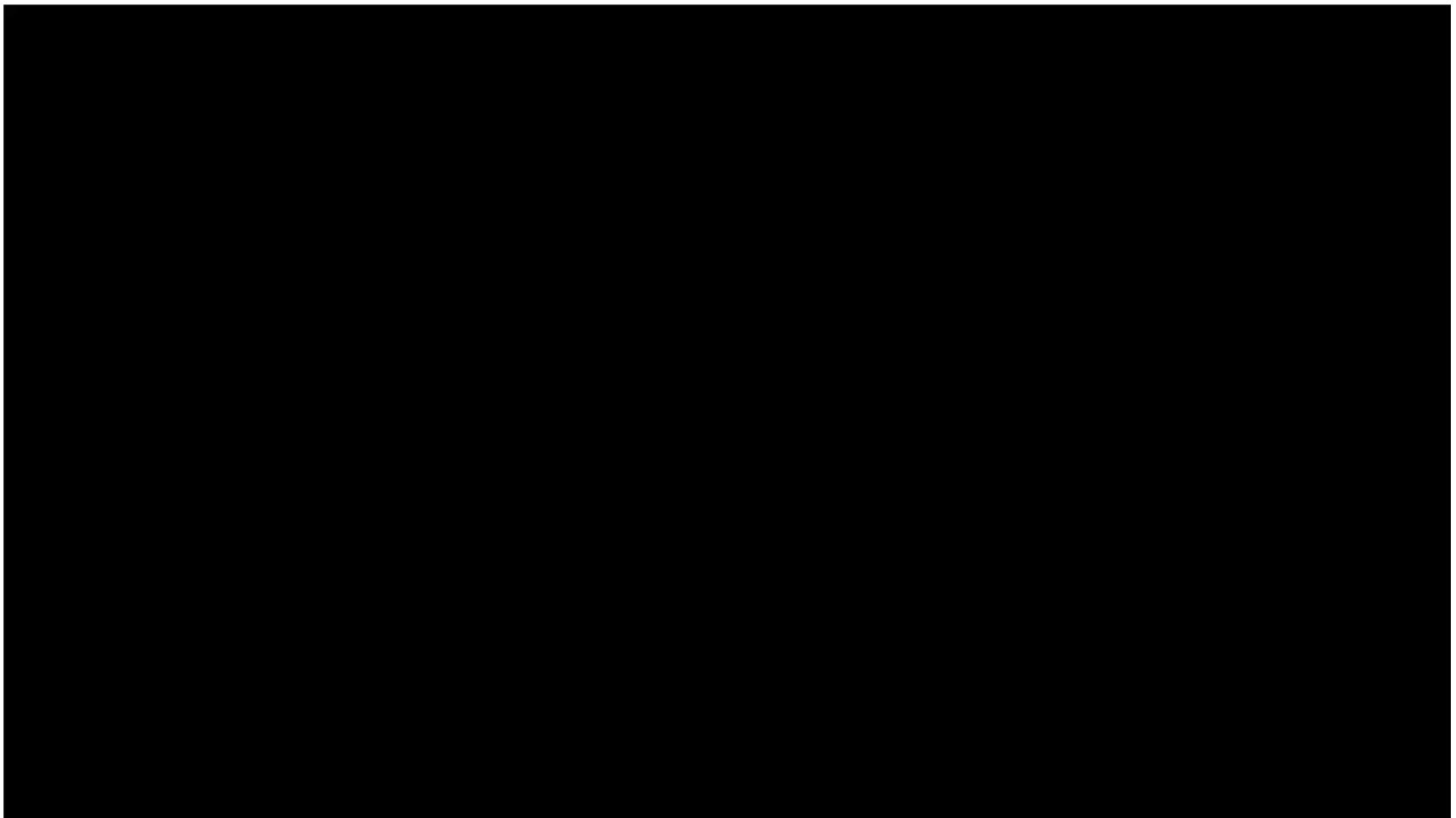


Control organization  
of dimensions with  
semantic attributes

Organize the parameters space

# Demo video

---



<https://www.youtube.com/watch?v=UufQwUitBlw>

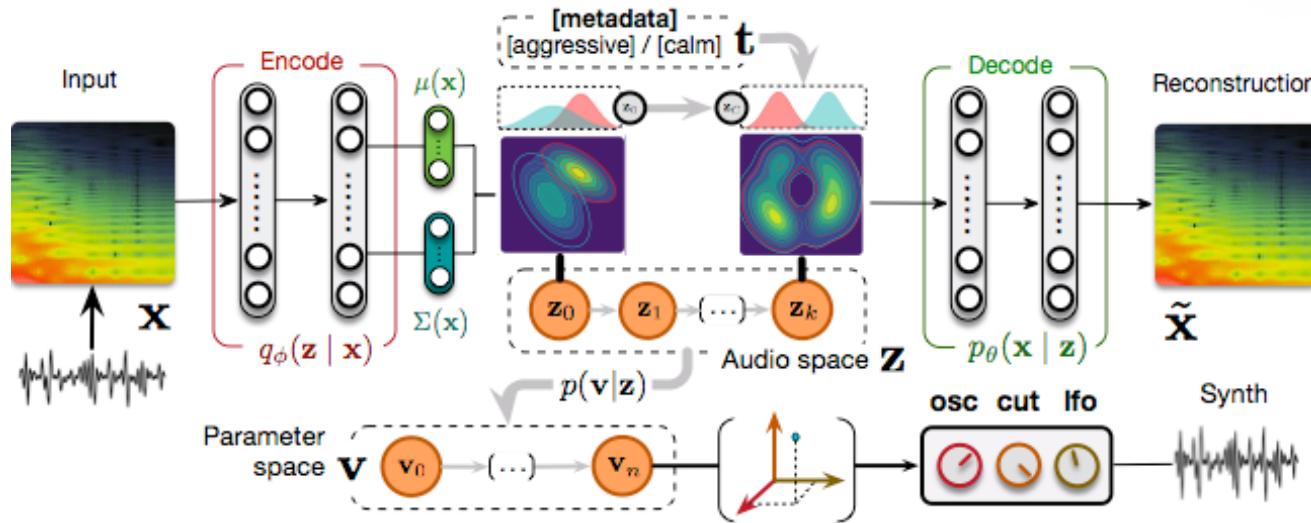
---

# Multiple flux synthesis (WavAE)

A. Caillon



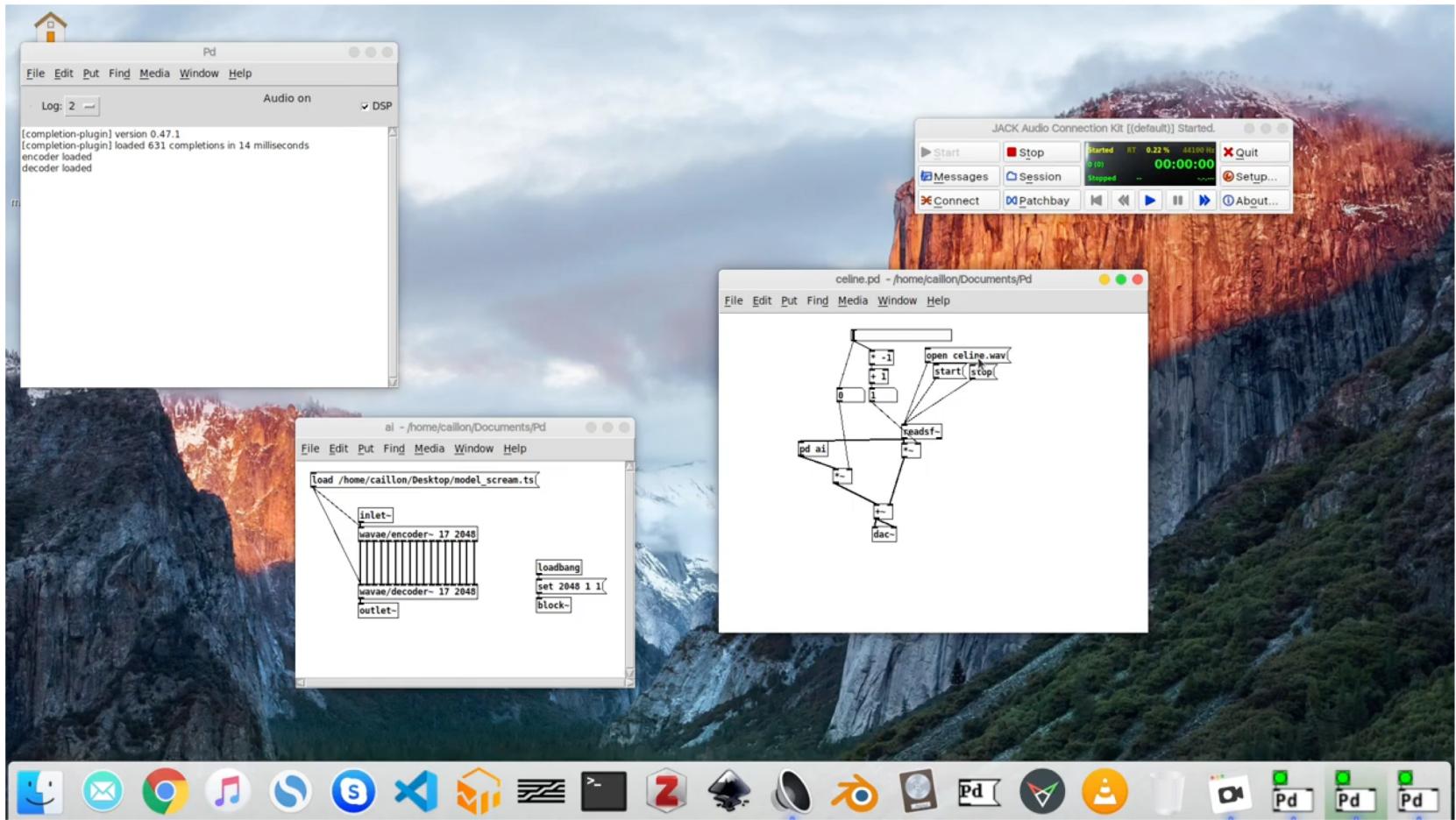
## Complete formalization



Similar to the flow synth overall approach

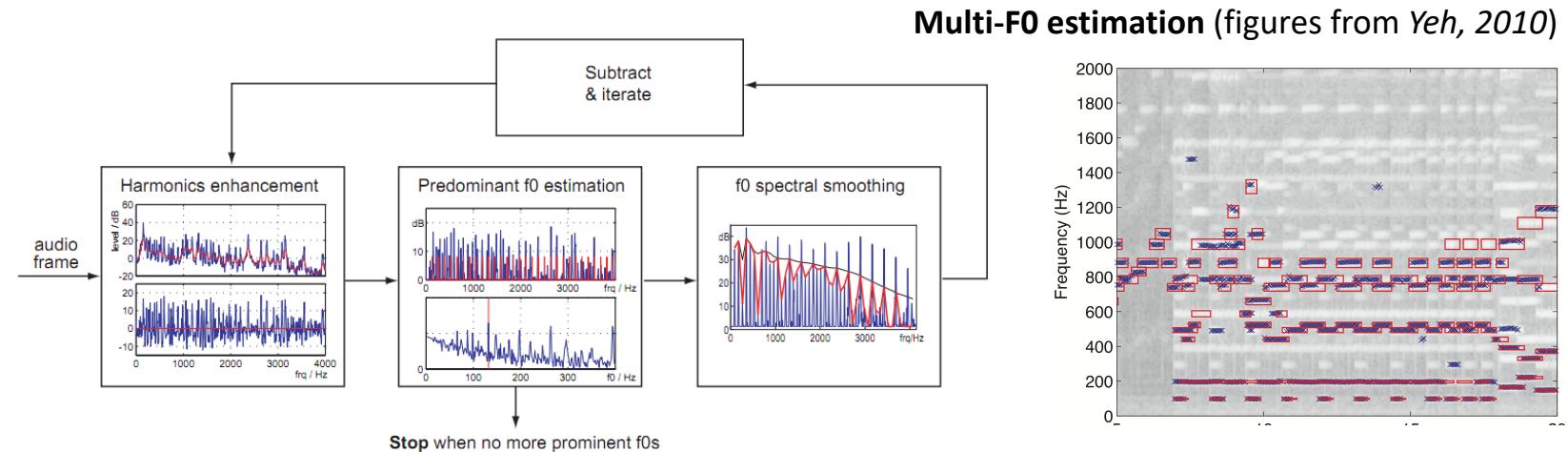
- Using different learning flux
- Allowing for multiple tasks and datasets
- In the future, simultaneous multiple waveform generation

# Demo video



# Why learning ?

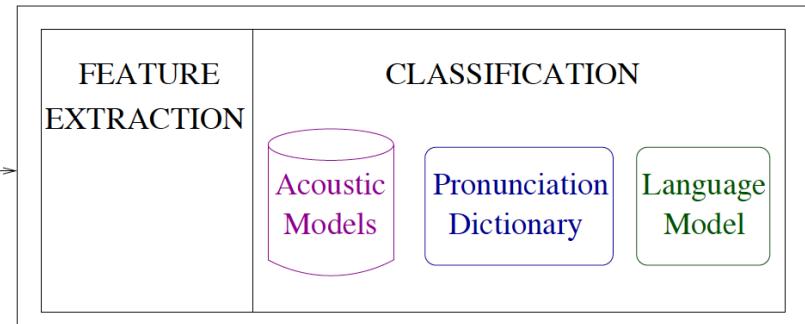
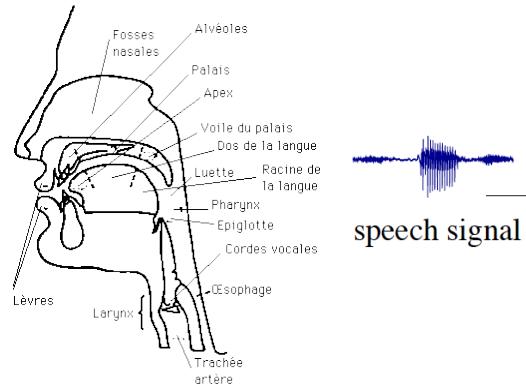
## Applications of pattern classification



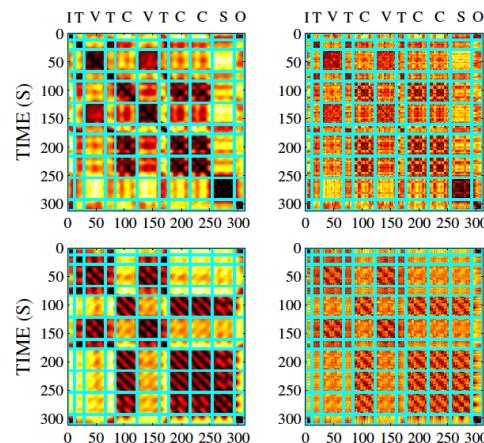
# Why learning ?

## Applications of pattern classification

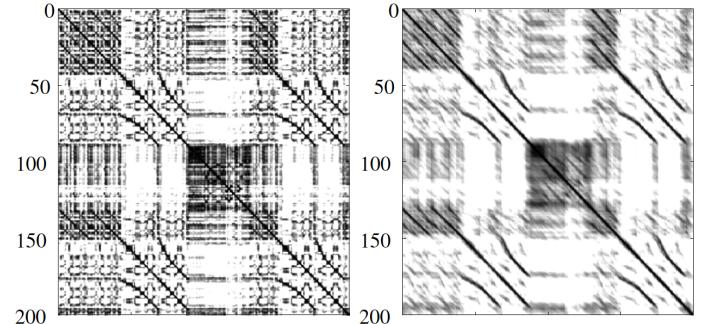
Speech recognition (figures from *HTK Documentation*)



"This is a..."  
word string

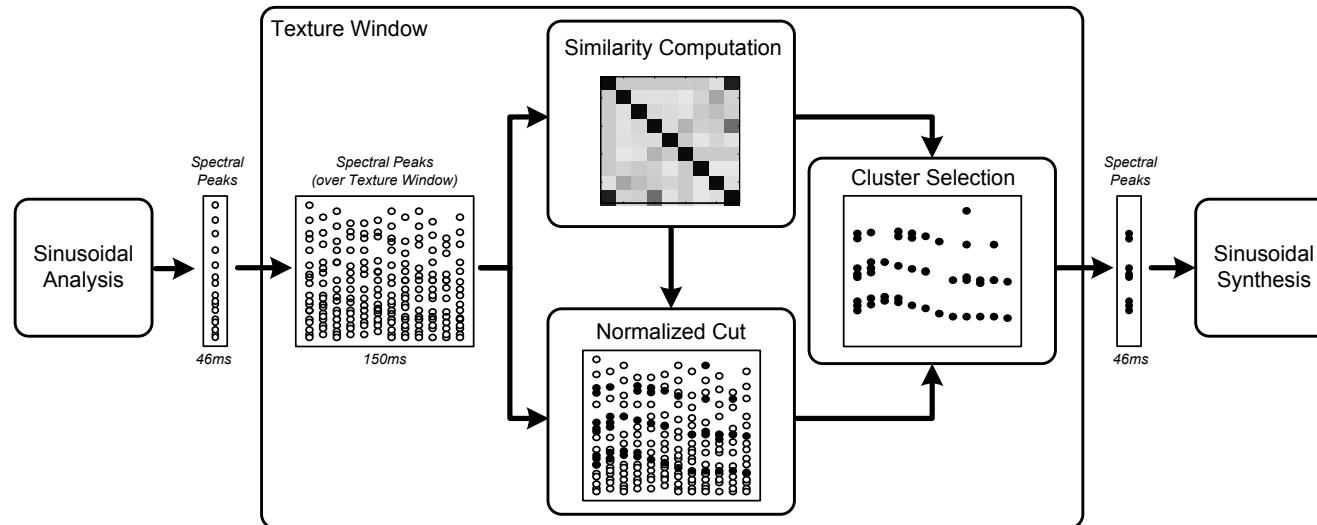
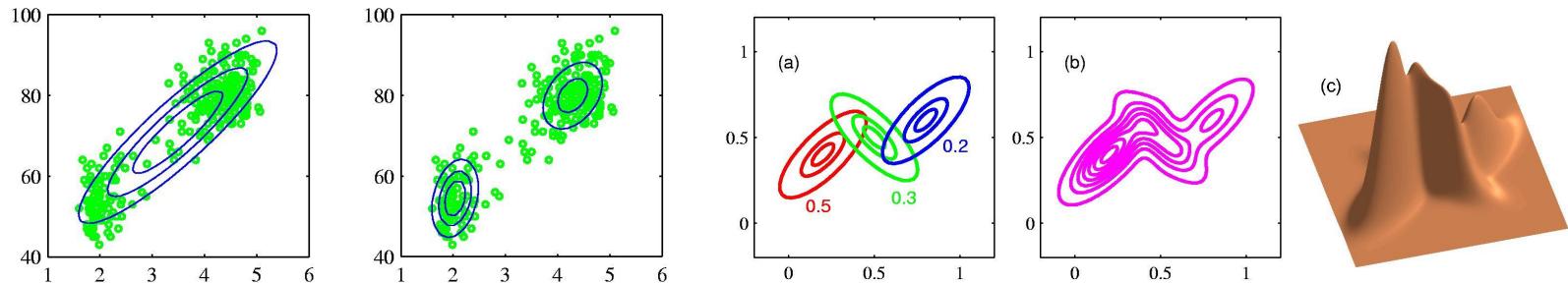


Structure discovery (figures from Paulus, 2010)



# Why learning ?

Typical learning problem: **clustering**

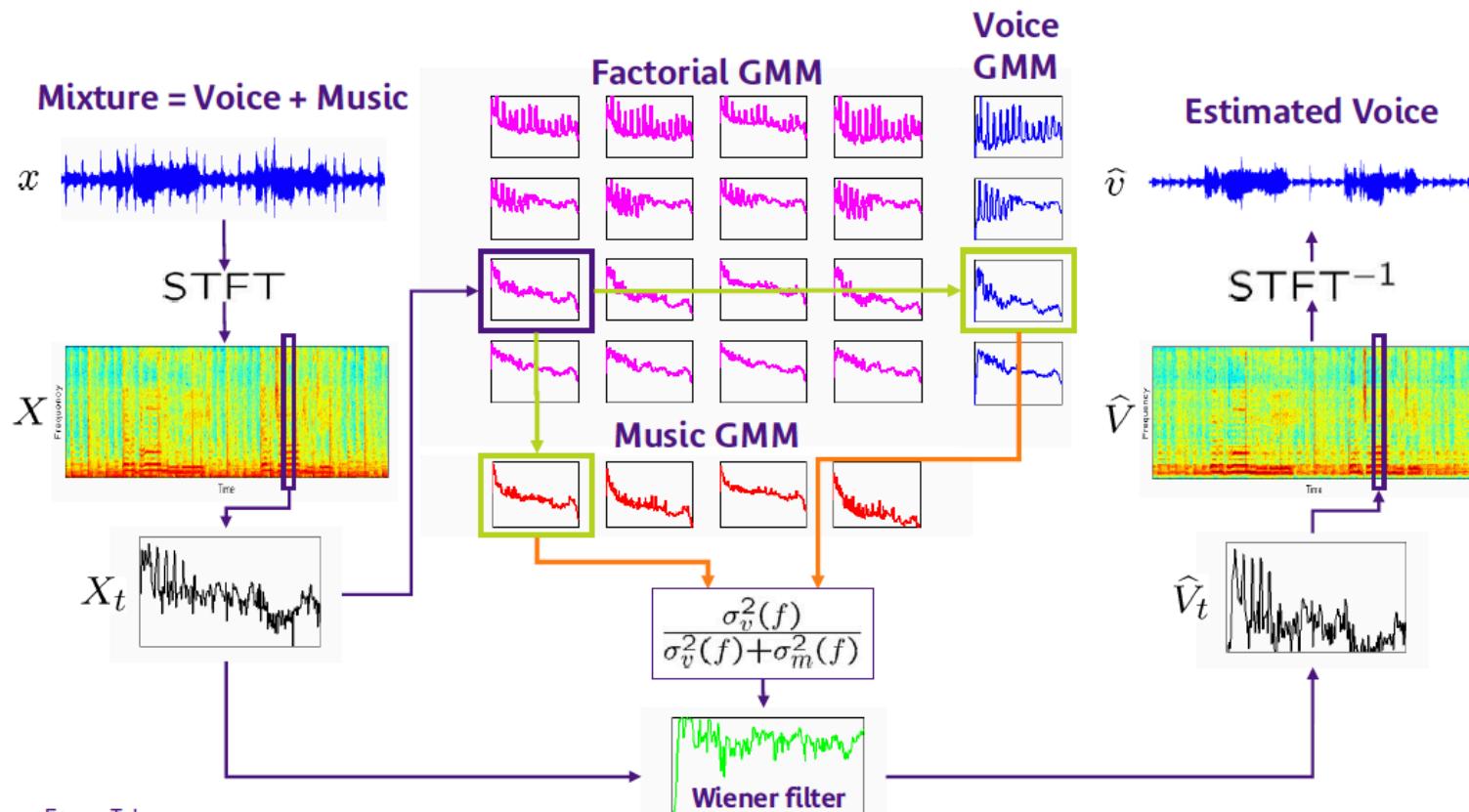


Computational Audio Scene Analysis (CASA) – Figures from Barker, 2011

# Why learning ?

## Applications of clustering

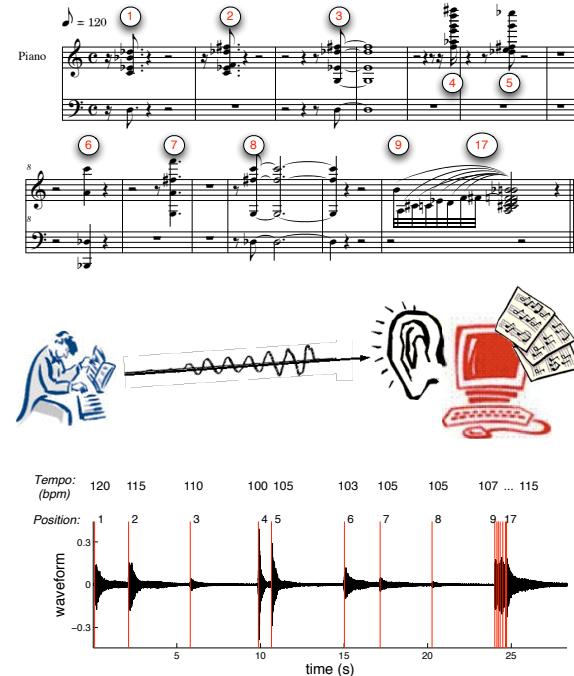
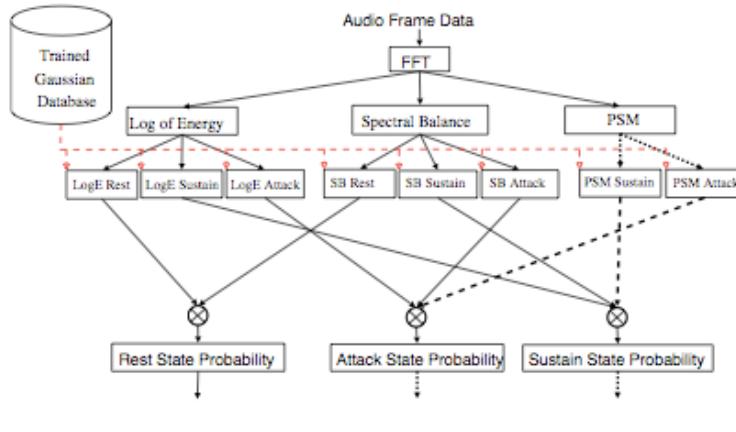
GMM-Based Sound Source Separation – Figures from Ozerov, 2005



# Why learning ?

## Sequential learning (temporal models)

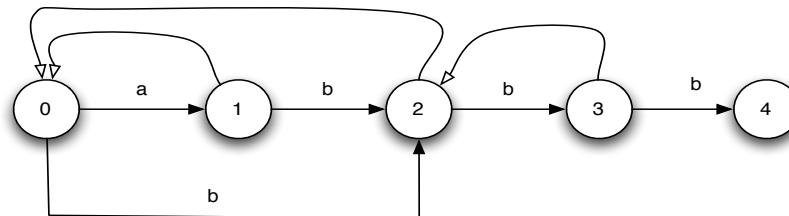
- The score following problem:
  - Need *observation models* at the front-end.
  - From audio frames to low-level state probs:



Figures from Cont, 2011

# Why learning ?

## Sequential learning (temporal models)



# Computational orchestration

L. Crestel

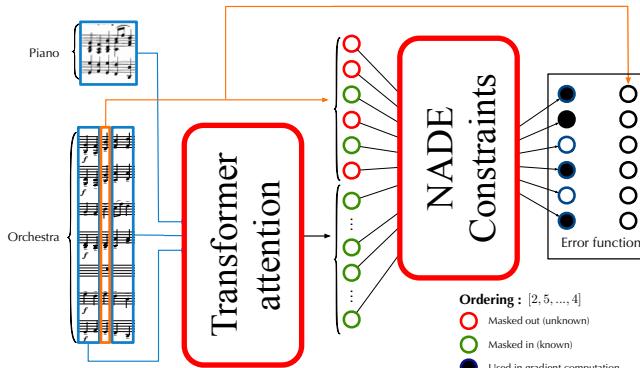
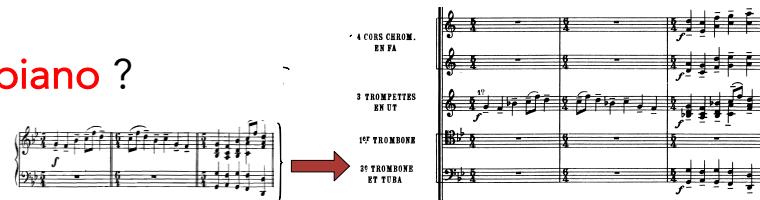


## Computer-assisted projective orchestration

Asking somewhat the inverse question ...

Can we **play the whole orchestra simply by playing a piano ?**

Targeting signal solely through symbolic information



## Multivariate predictive learning

Piano time series constraining a multivariate series

Mixing temporal, recurrent and feed-forward embeddings

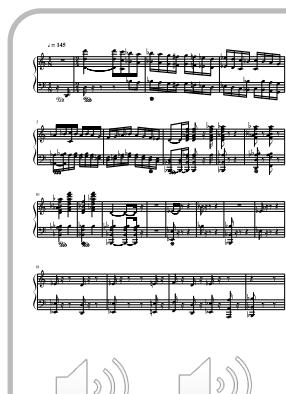
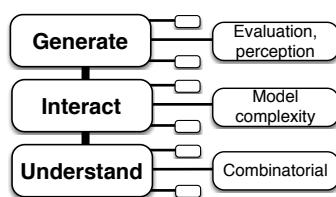
**Constrained generation** with Neural Distribution Estimator (NADE)

First ever dataset on the problem (LOP - <https://qsdfo.github.io/LOP>)

Hand-checked, corrected and entirely open data (~200 score pairs)

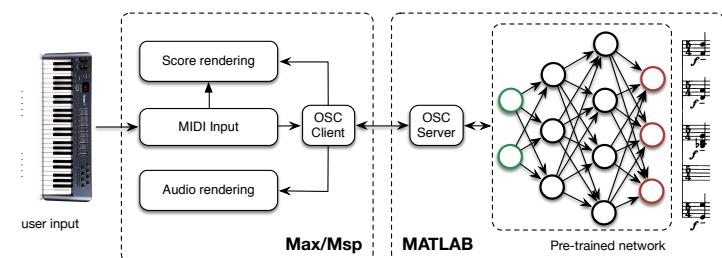
## Results

Liszt piano piece  
Symphony No. 3, S.464/3



## Real-time version demonstration

1st system to play orchestra on a keyboard  
See you at the workshop ☺



# Learning

---

3 major types of learning

- **Supervised learning** is inferring a function from labeled training data
- **Unsupervised learning** is trying to find hidden structure in unlabeled data
- **Reinforcement learning** is acting to maximize a notion of cumulative reward

What will we see along these courses?

1. **Nearest neighbors**
2. **Neural Networks – Single and multilayers [+ advanced networks]**
3. **Support Vector Machines and Kernels**
4. **Probabilities and distributions**
5. **Probabilistic (Bayesian) inference**
6. **Latent models and Expectation-Maximization (EM)**
7. **Gaussian Mixture Models (GMM)**
8. **Approximate inference (sampling, MCMC and variational)**
9. **Deep learning**
10. **Variational auto-encoder and normalizing flows**
11. **Adversarial learning**
12. **Audio applications (project)**

# Learning

---

Using machine learning to model music ?

- **Supervised learning** is inferring a function from labeled training data
- **Unsupervised learning** is trying to find hidden structure in unlabeled data

We can define any problem as  $\hat{\mathbf{y}} = f_{\theta}(\mathbf{x})$

So we first need to define what **function** could *approximate* this process

$$f_{\theta} \in \mathcal{F} \quad \theta \in \Theta$$

Then we need to **evaluate the « quality »** of this approximation

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y} \mid \theta, f_{\theta})$$

|      |

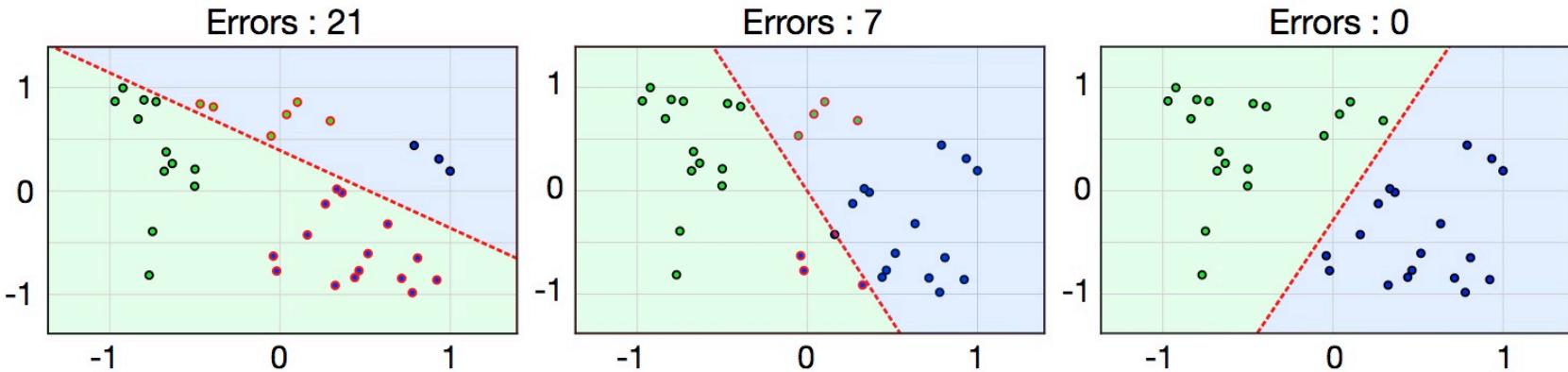
*Answer of the model    The « real » answer (groundtruth)*

**Expected risk**     $\mathcal{R}(f_{\theta}) = \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} [\mathcal{L}(f_{\theta}(\mathbf{x}), \mathbf{y})]$

The quality of our model **over the whole world of possibilities** ?

# Learning ?

- But how to learn the parameters  $\theta \in \Theta$  of our function  $f_\theta \in \mathcal{F}$  ?
- Learning is possible through weight *optimization*.



Example function :  $f_\theta(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$        $\theta = \{\mathbf{W}, \mathbf{b}\}$

- This is done through a **training phase**
- We have a set of input vectors  $\mathbf{X} = \{x_1, \dots, x_n\}$
- For which we know the desired output  $\mathbf{D} = \{d_1, \dots, d_n\}$
- Therefore we can compute the error  $\varepsilon = \sum_i (d_i - x_i)^2$
- So training amounts to adjust  $\theta \in \Theta$  so that  $y_i \sim d_i$
- Entering the beautiful world of *gradient descent*

# Gradient descent

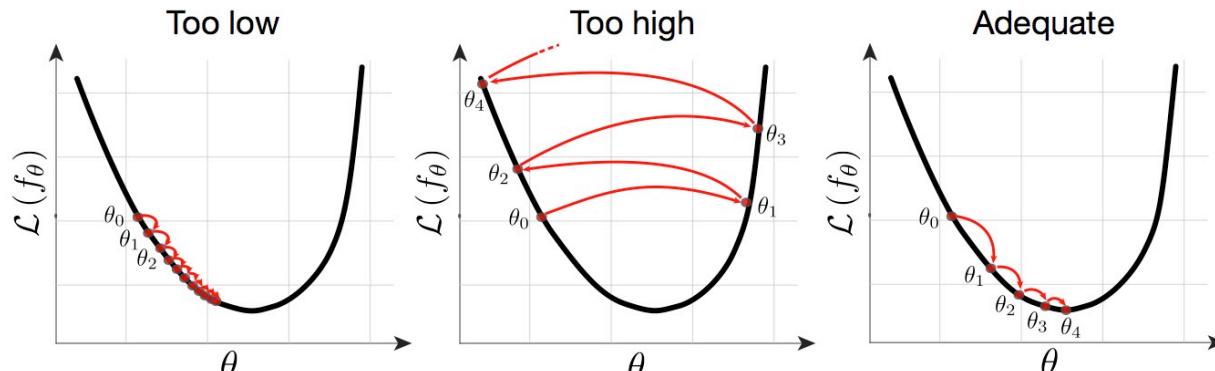
- Gradient descent is path following
- Need to find the lowest error point
- This amounts to error minimization
- Look around different directions
- Compute the difference in error

$$\Delta \bar{w} = r \left( \frac{\delta \mathcal{P}}{\delta w_1}, \frac{\delta \mathcal{P}}{\delta w_2} \right) \text{ « Go down hills » and update}$$

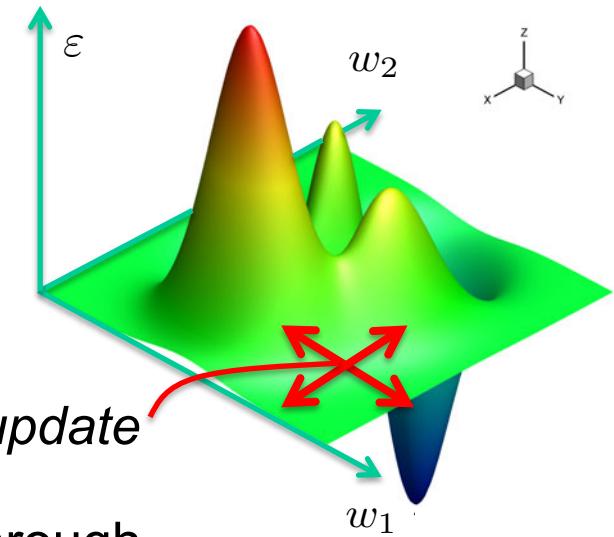
So we will iteratively update the parameters through

$$\theta_{n+1} = \theta_n - \eta \Delta_\theta \mathcal{L}(\theta_n)$$

with  $\Delta_\theta \mathcal{L}(\theta_n)$  the gradient of the loss function, and  $\eta$  the *learning rate*



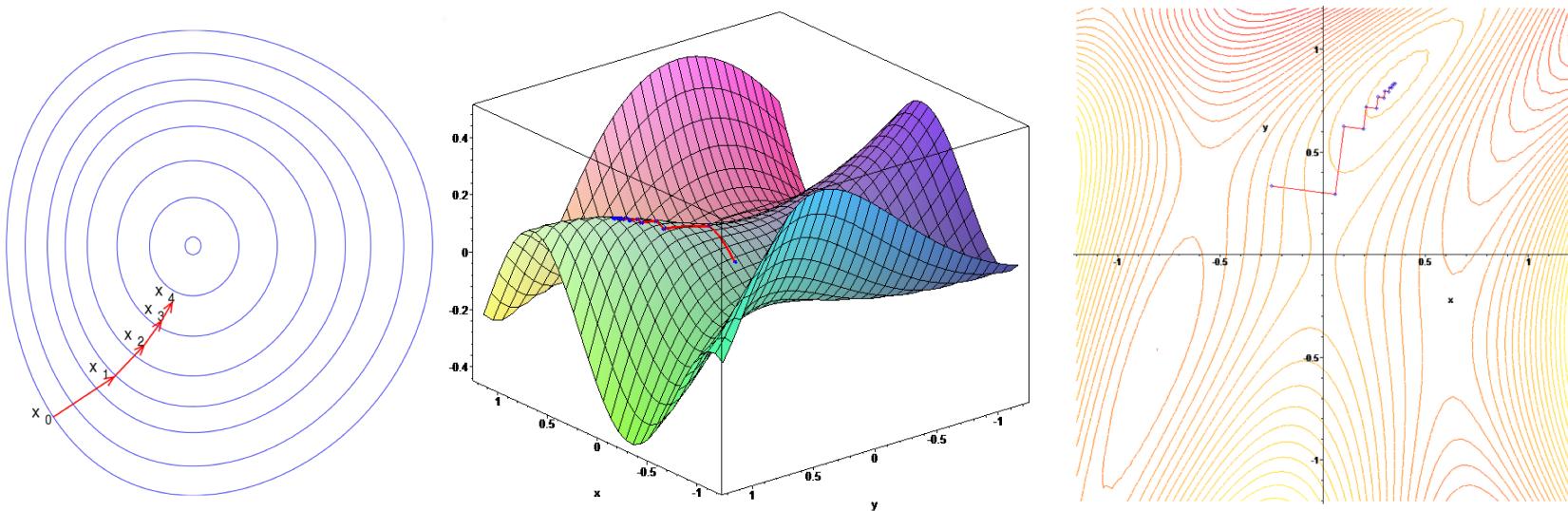
Effect of the learning rate  $\eta$



# Search (optimization)

A wide part of the optimization field is centered on search

- (Note that **not all ML approaches use gradient descent**)
- Depth-first (lexical order) - Breadth-first
- Heuristic – genetic algorithms



- Unfortunately, we will not spend too much time on *optimization* aspects
- But still explain everything that you need ☺

# Generic machine learning

- Natural processes relate some objects
- We observe relationships between  $\mathcal{X}$  and  $\mathcal{Y}$

Machine learning tries to approximate this as

$$\hat{\mathbf{y}} = f_{\theta}(\mathbf{x})$$

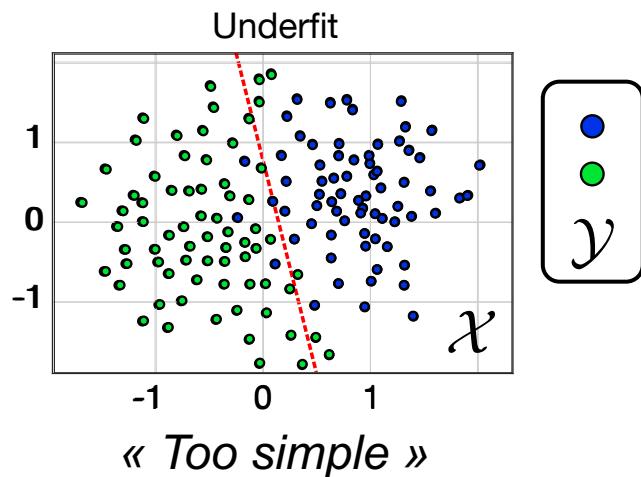
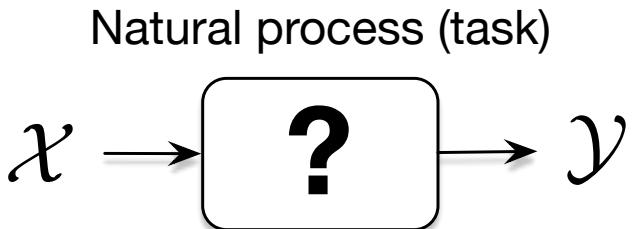
So we need to define what **function**  $f_{\theta} \in \mathcal{F}$

This function depends on some parameters  $\theta \in \Theta$

By evaluating the « quality » of this function  $\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y} \mid \theta, f_{\theta})$

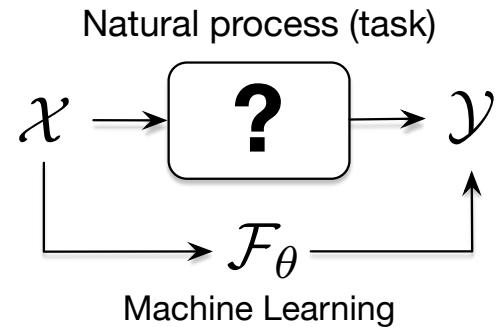
We can optimize parameters  $\theta \in \Theta$  to obtain the best approximation  $f_{\theta^*}$

So everything boils down to our choice of  $\mathcal{F}_{\theta}$



# Choice of approximation families

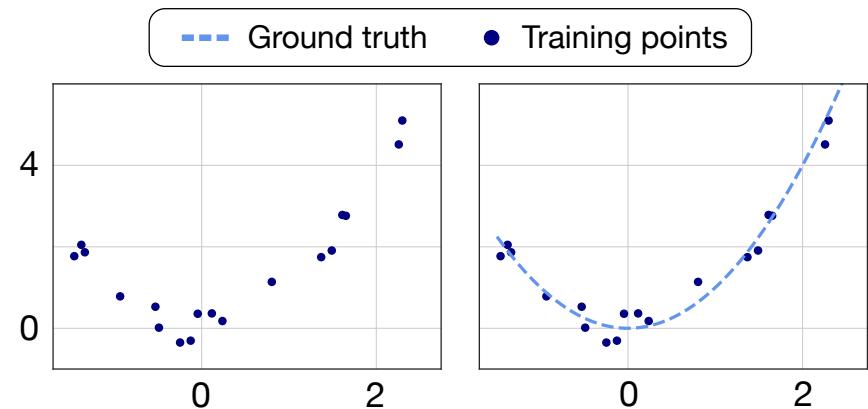
- Quality depends heavily on the *family*  $\mathcal{F}_\theta$
- Notion of *model capacity* ( $\sim$  complexity)
- There are some inclusions



## Approximation families

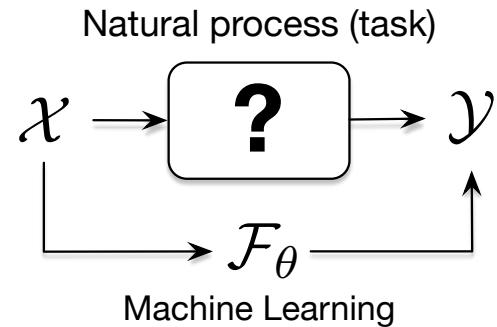
Constant  
 $\mathcal{F}_0(x) = \theta_0$

$\mathcal{F}_0$



# Choice of approximation families

- Quality depends heavily on the *family*  $\mathcal{F}_\theta$
- Notion of *model capacity* ( $\sim$  complexity)
- There are some inclusions



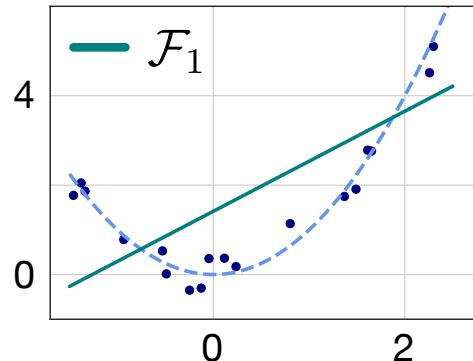
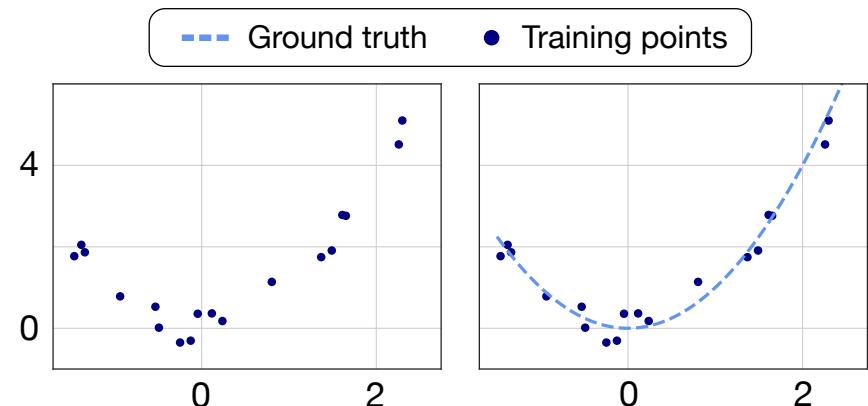
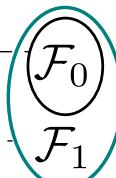
## Approximation families

Constant

$$\mathcal{F}_0(x) = \theta_0$$

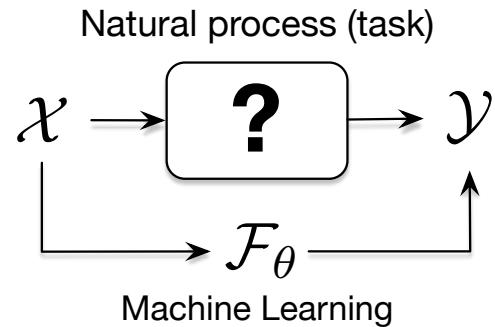
Linear

$$\mathcal{F}_1(x) = \theta_1 x + \theta_0$$



# Choice of approximation families

- Quality depends heavily on the *family*  $\mathcal{F}_\theta$
- Notion of *model capacity* ( $\sim$  complexity)
- There are some inclusions



## Approximation families

Constant

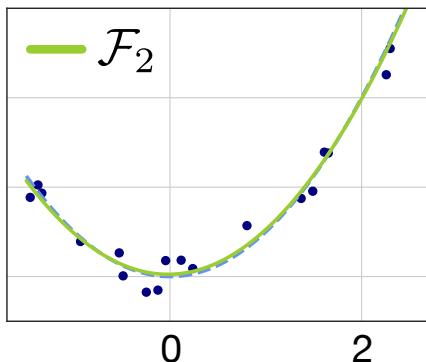
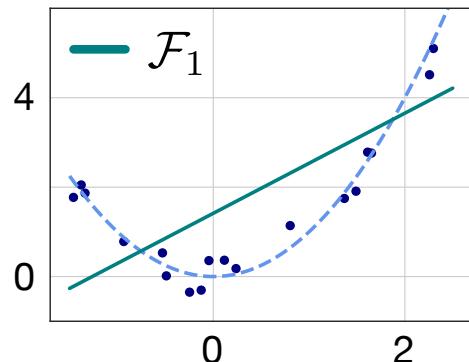
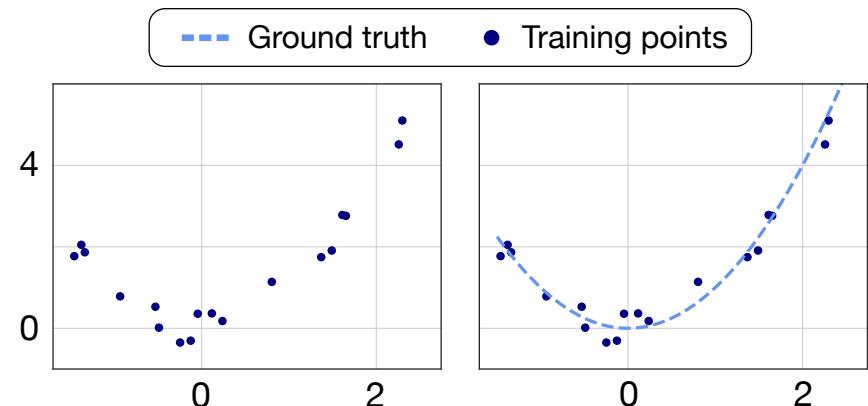
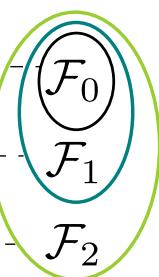
$$\mathcal{F}_0(x) = \theta_0$$

Linear

$$\mathcal{F}_1(x) = \theta_1 x + \theta_0$$

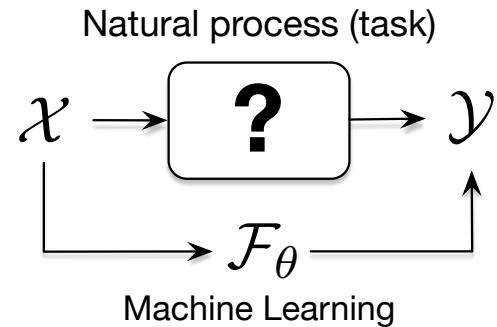
Square

$$\mathcal{F}_2(x) = \theta_2 x^2 + \theta_1 x + \theta_0$$



# Choice of approximation families

- Quality depends heavily on the *family*  $\mathcal{F}_\theta$
- Notion of *model capacity* ( $\sim$  complexity)
- There are some inclusions



## Approximation families

Constant

$$\mathcal{F}_0(x) = \theta_0$$

Linear

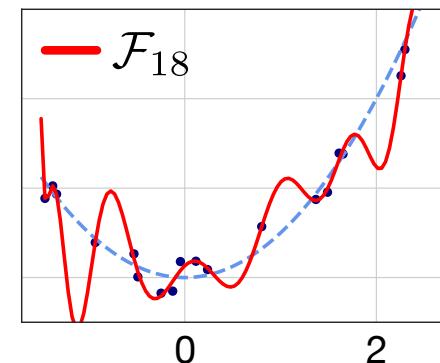
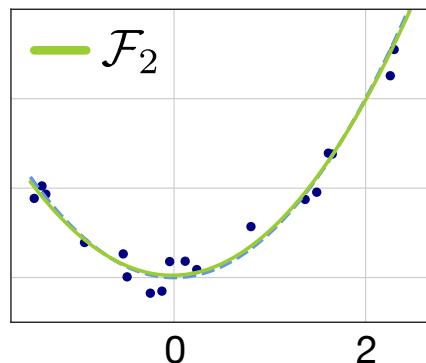
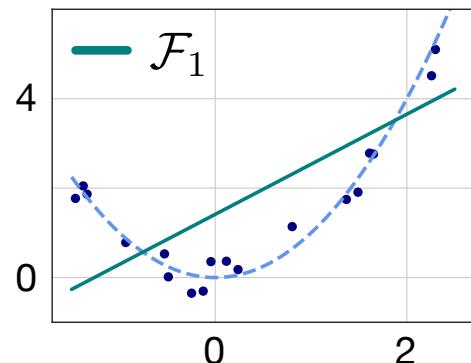
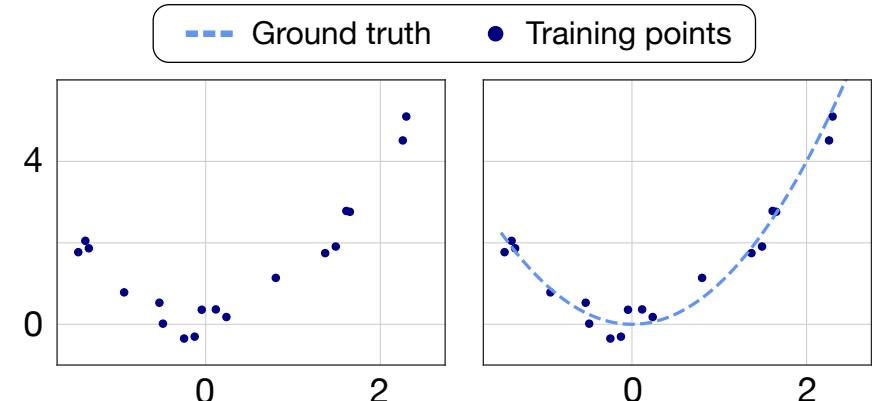
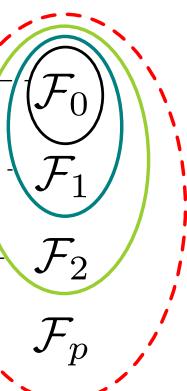
$$\mathcal{F}_1(x) = \theta_1 x + \theta_0$$

Square

$$\mathcal{F}_2(x) = \theta_2 x^2 + \theta_1 x + \theta_0$$

Polynomial

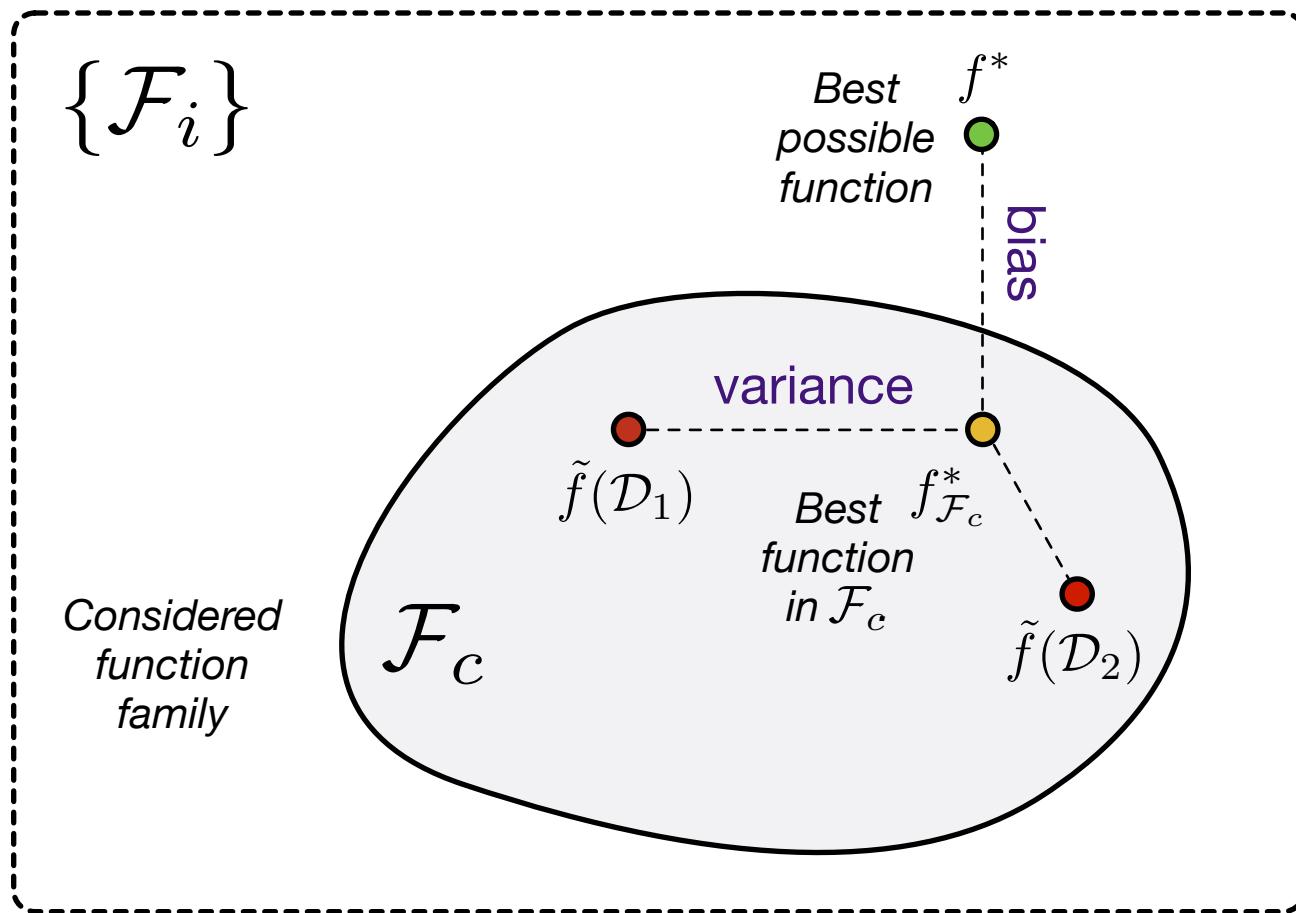
$$\mathcal{F}_p(x) = \theta_p x^p + \dots + \theta_1 x + \theta_0$$



# Bias and variance

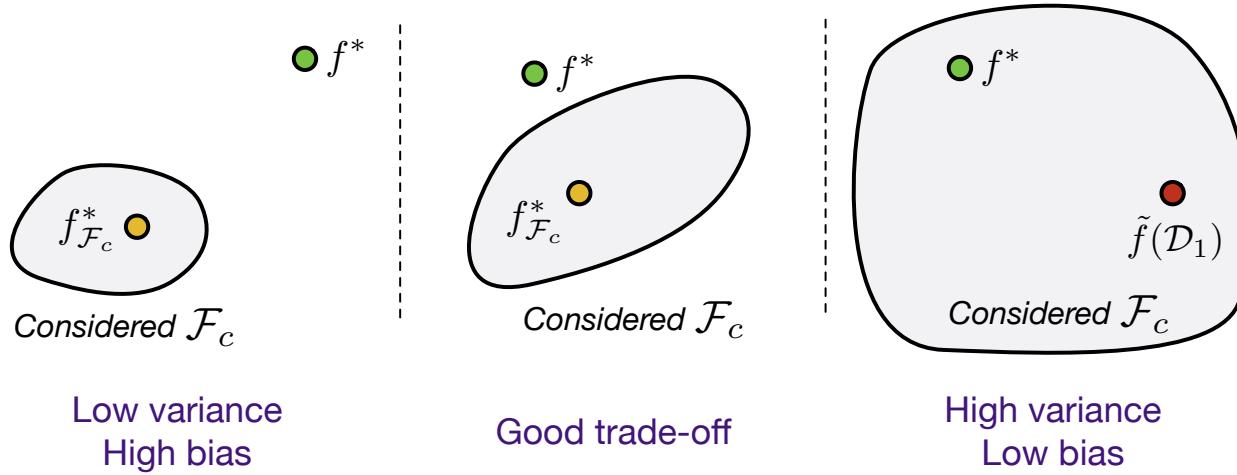
This problem lies between the *bias* and *variance* of our approximation

*Space of all possible functions*



# Bias and variance - details

So how can we adequately choose the family ?



Problem in the **variance** aspect is the *discrepancy* lying between

**Expected risk**  
(what we want)

$$\mathcal{R}(f_\theta) = \mathbb{E}_{p(\mathbf{x}, \mathbf{y})} [\mathcal{L}(f_\theta(\mathbf{x}), \mathbf{y})]$$

**Empirical risk**  
(what we have)

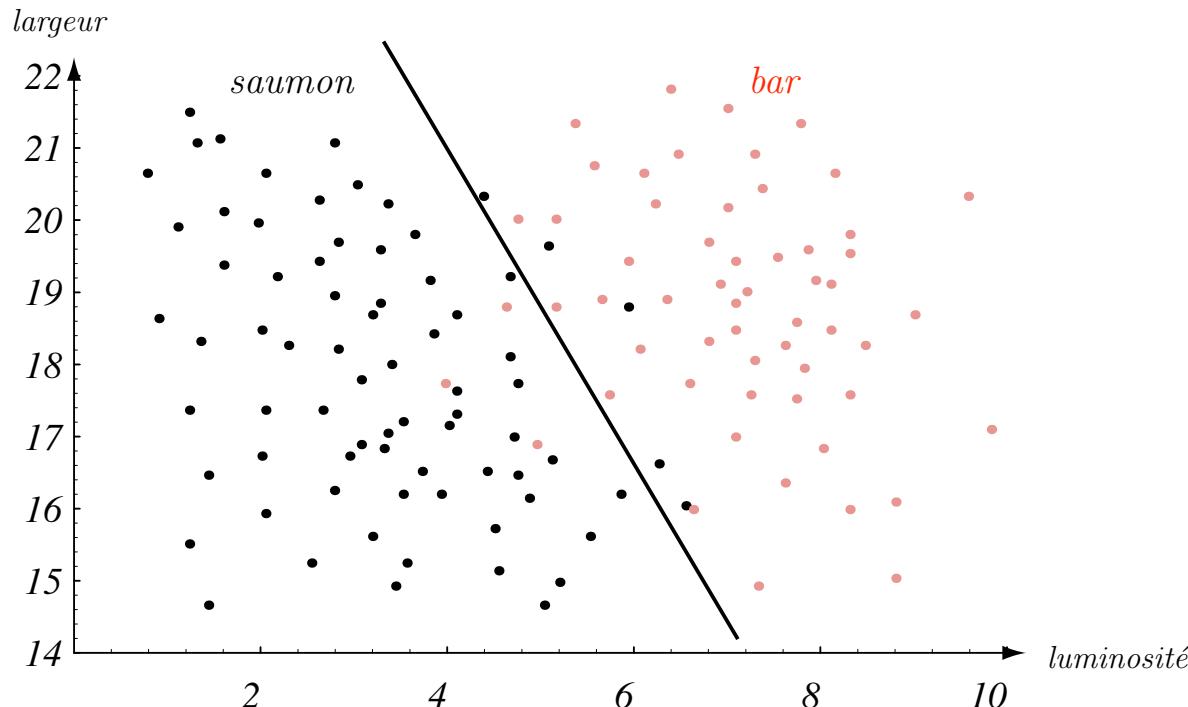
$$\hat{\mathcal{R}}(f_\theta, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \mathcal{L}(f_\theta(\mathbf{x}), \mathbf{y})$$

# Defining Machine Learning

Ex: 2D classification

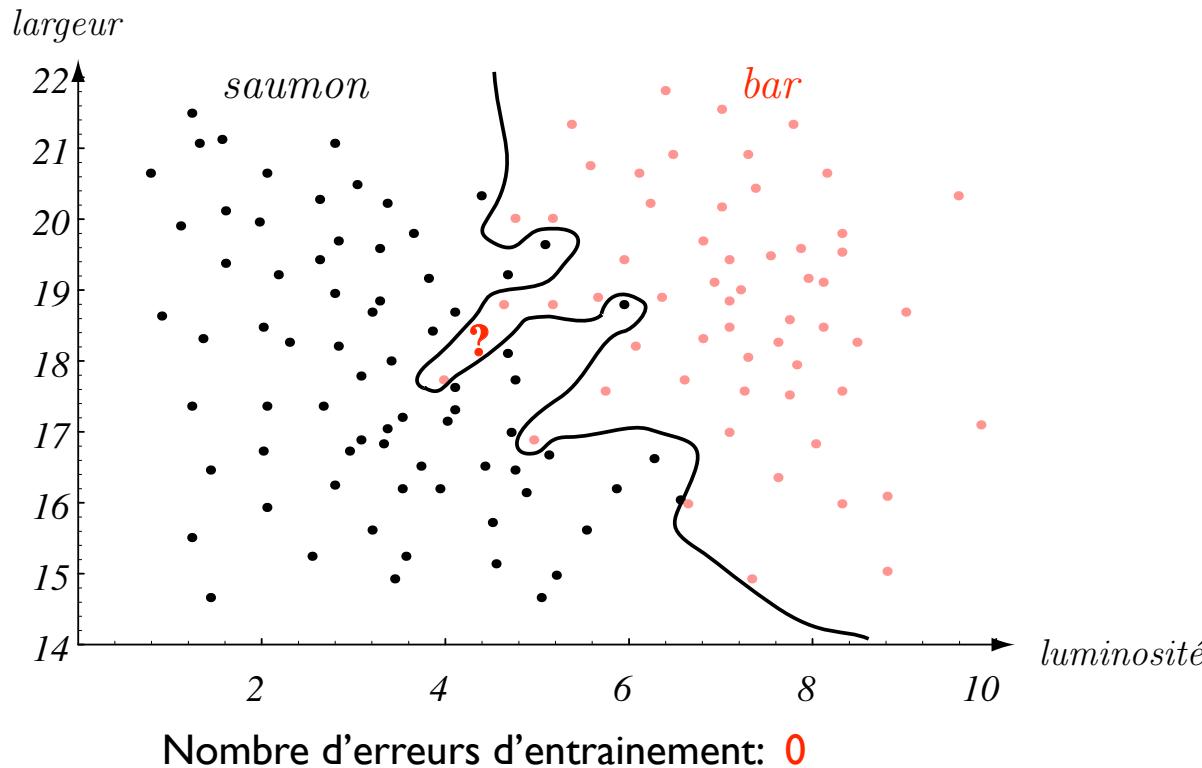
*Linear classifier*

- Function family too poor  
(too inflexible)
- = **Capacity too low** for this problem  
(relative to number of examples)
- => Under-fitting



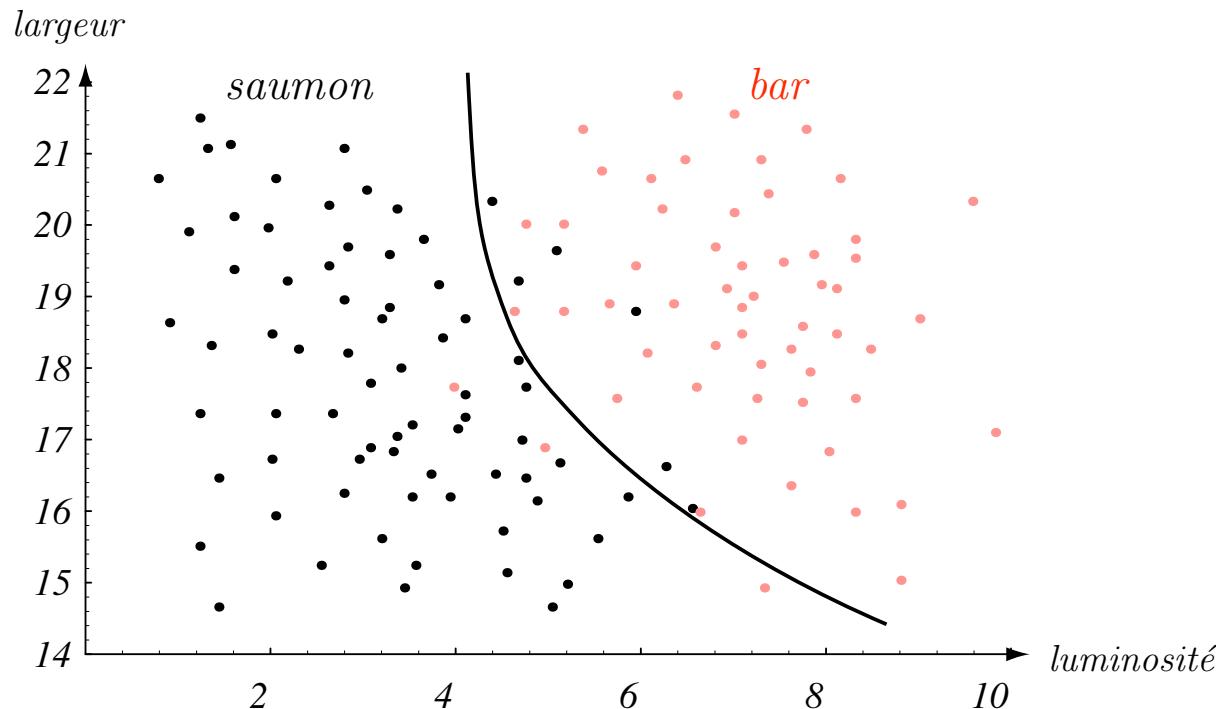
# Defining Machine Learning

- Function family too rich  
(too flexible)
- = **Capacity too high** for this problem  
(relative to the number of examples)
- => **Over-fitting**



# Defining Machine Learning

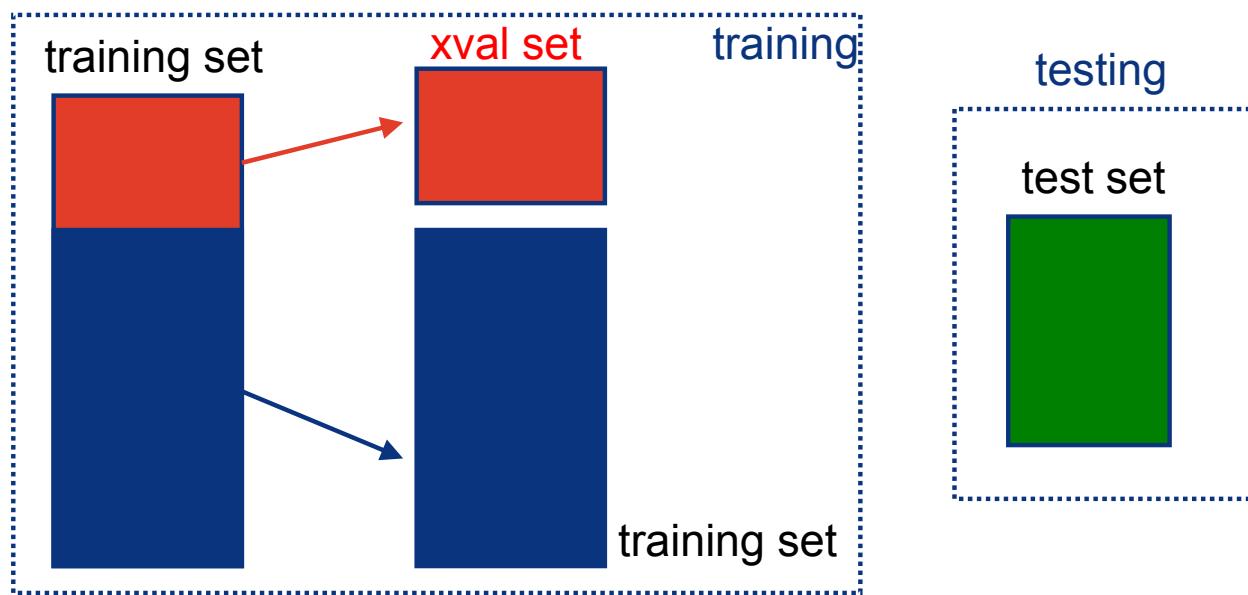
- **Optimal capacity** for this problem  
(par rapport à la quantité de données)
- => Best generalization  
(on future test points)



# Cross-validation

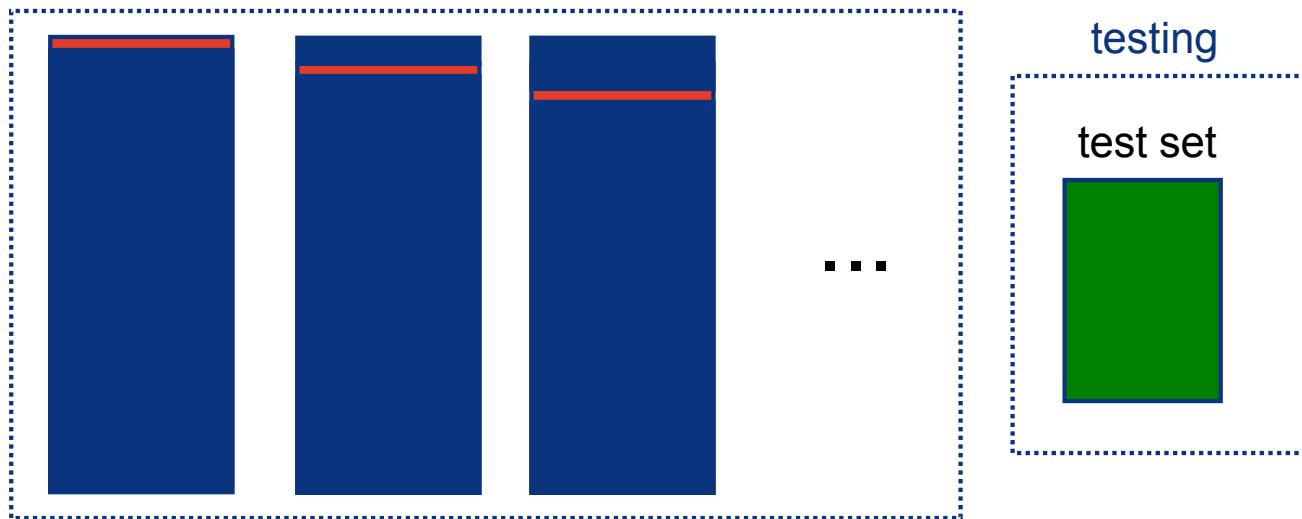
► basic idea:

- leave some data out of your training set (cross validation set)
- train with different parameters
- evaluate performance on cross validation set
- pick best parameter configuration



# Cross-validation

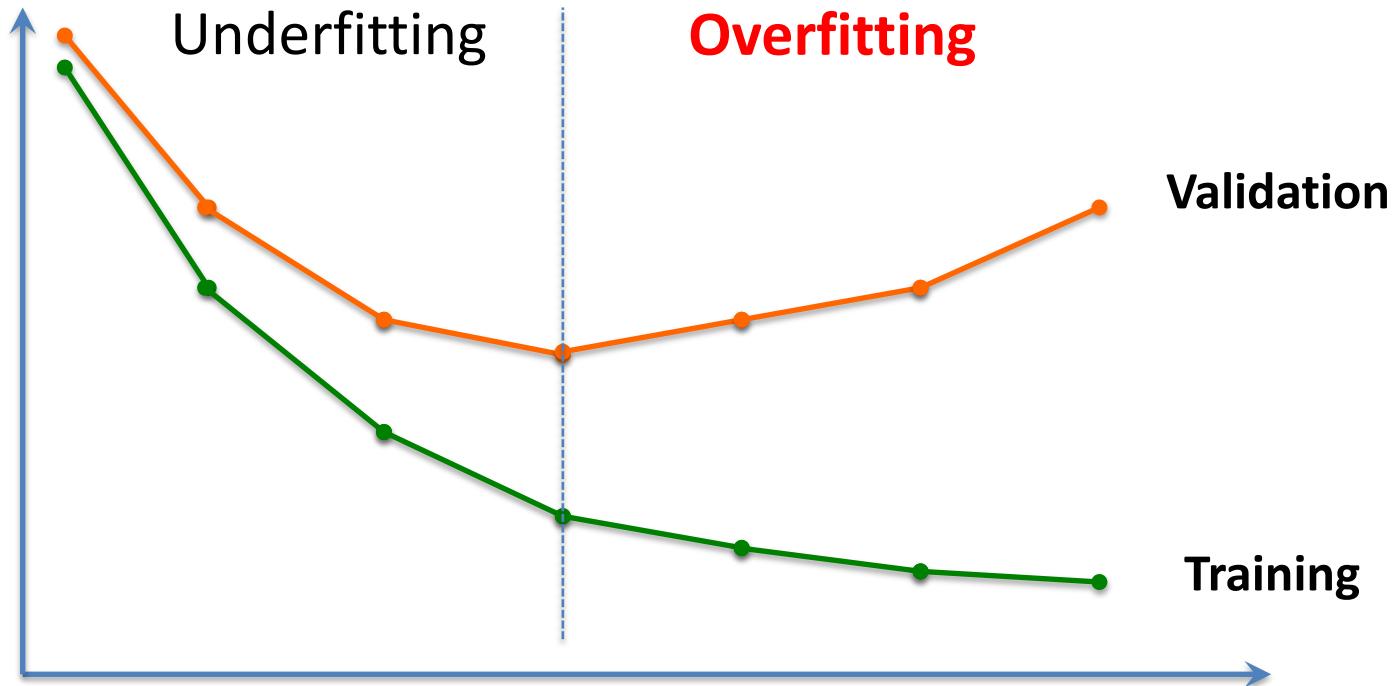
- ▶ many variations
- ▶ leave-one-out CV:
  - compute n estimators of  $P_X(x)$  by leaving one  $X_i$  out at a time
  - for each  $P_X(x)$  evaluate  $P_X(X_i)$  on the point that was left out
  - pick  $P_X(x)$  that maximizes this likelihood



# Knowing when to stop

## Early stopping

Stop the learning when the error increases on the validation set  
(Approximates the generalization power)

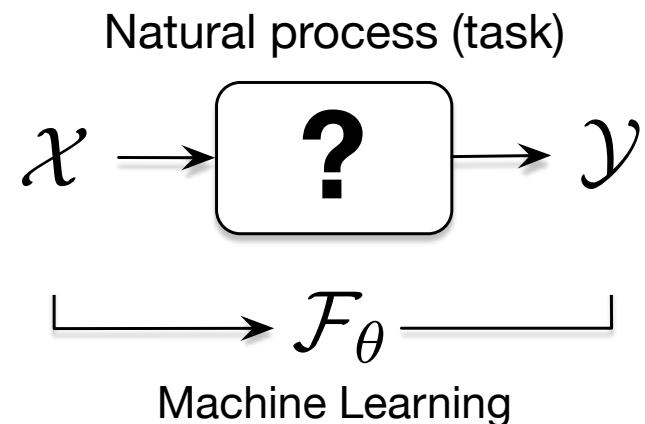


# Understanding the input space

- Natural processes relate some objects
- We observe relationships between  $\mathcal{X}$  and  $\mathcal{Y}$

Machine learning tries to approximate this as

$$\hat{\mathbf{y}} = f_{\theta}(\mathbf{x})$$



So everything depends on the nature of the input space

$$\mathbf{X} = \{x_1, \dots, x_n\}$$

# Nearest-Neighbor

---

I have input sounds and want to dissociate:

- Male singing from female singing

# Nearest-Neighbor

---

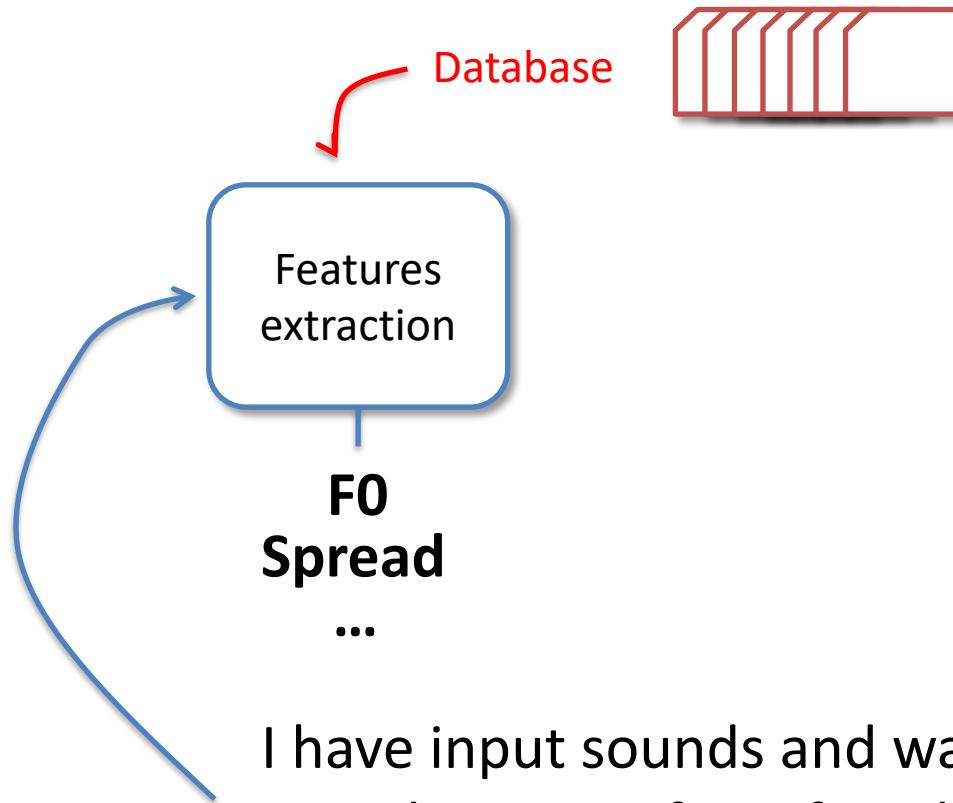
Database



I have input sounds and want to dissociate:

- Male singing from female singing

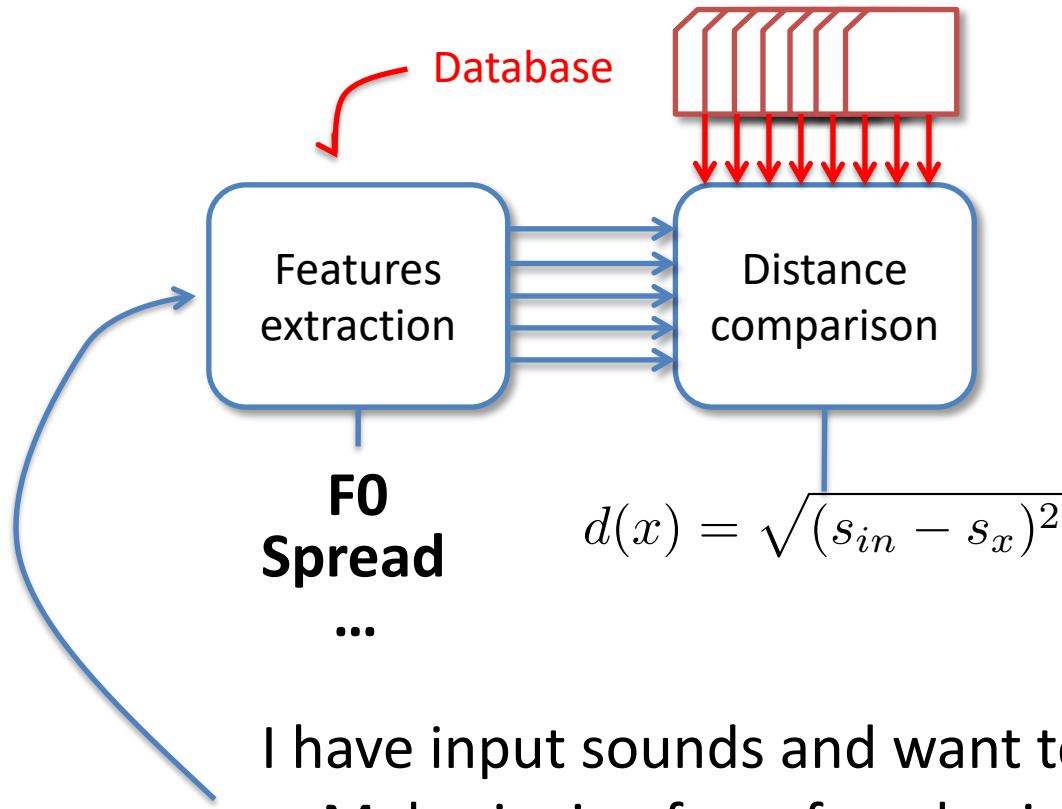
# Nearest-Neighbor



I have input sounds and want to dissociate:

- Male singing from female singing

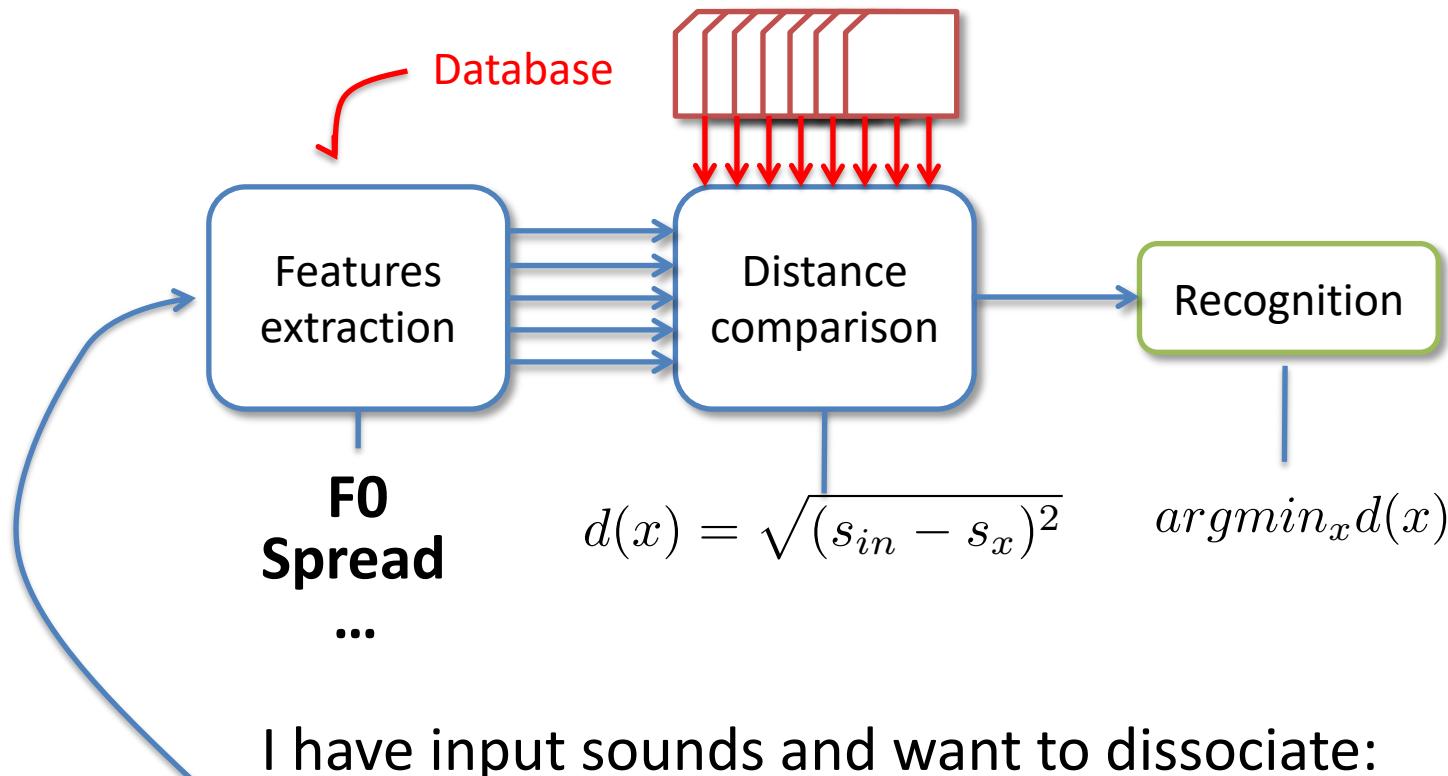
# Nearest-Neighbor



I have input sounds and want to dissociate:

- Male singing from female singing

# Nearest-Neighbor

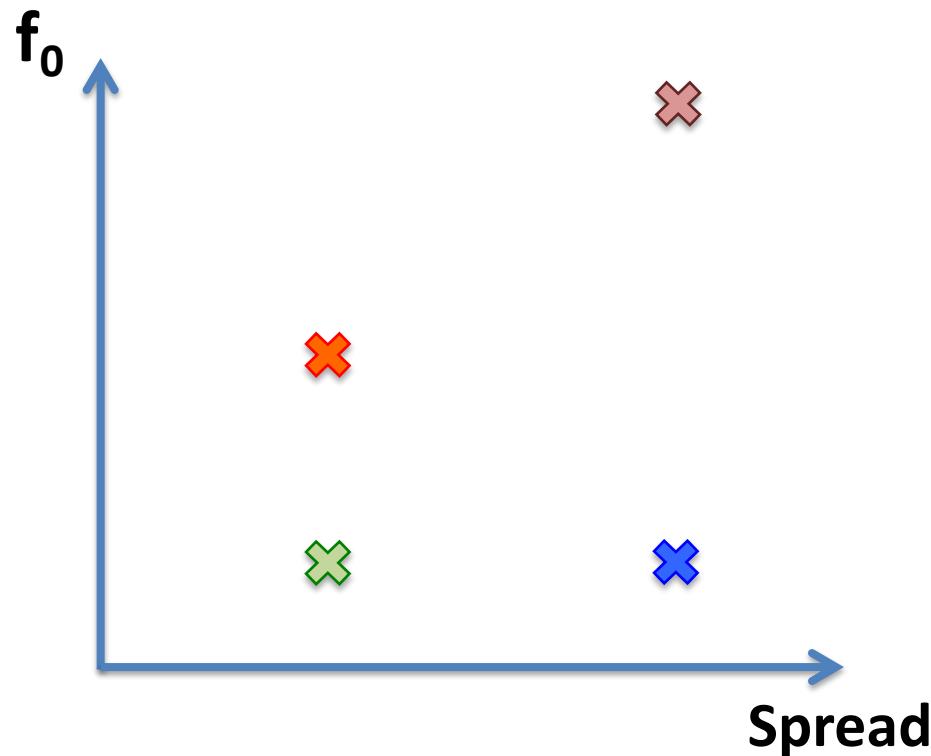


I have input sounds and want to dissociate:

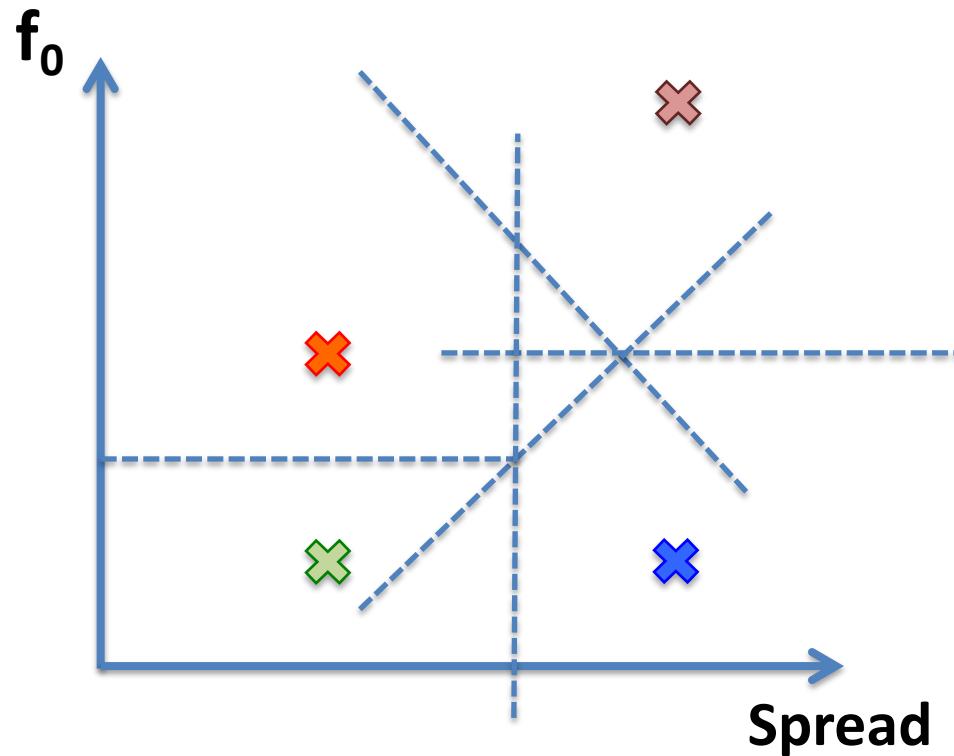
- Male singing from female singing

# Nearest-Neighbor

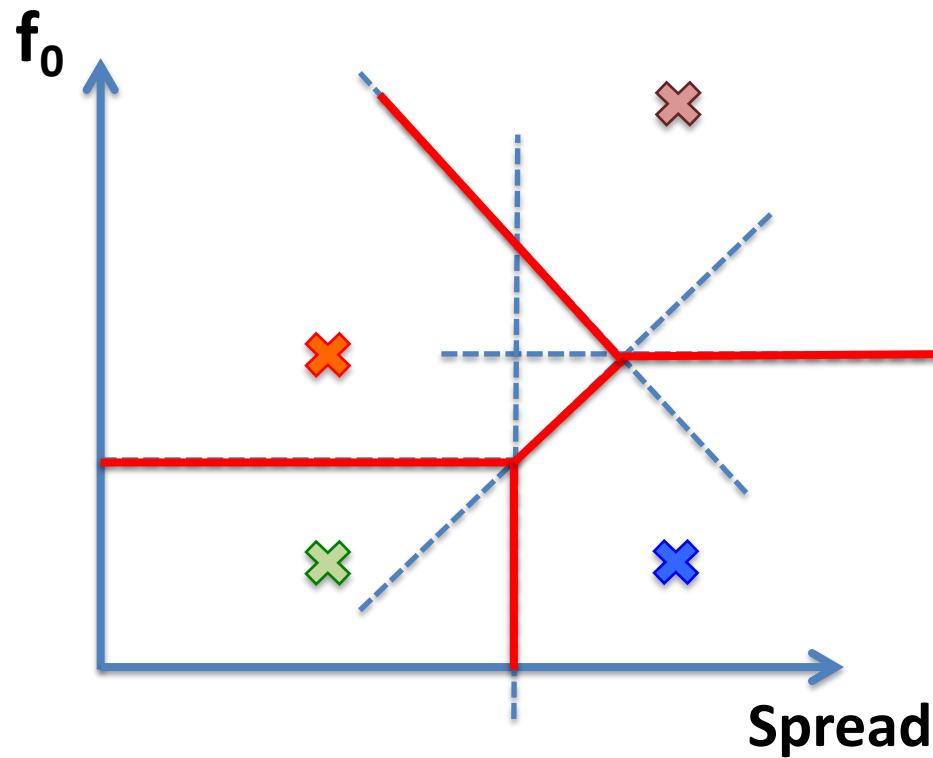
---



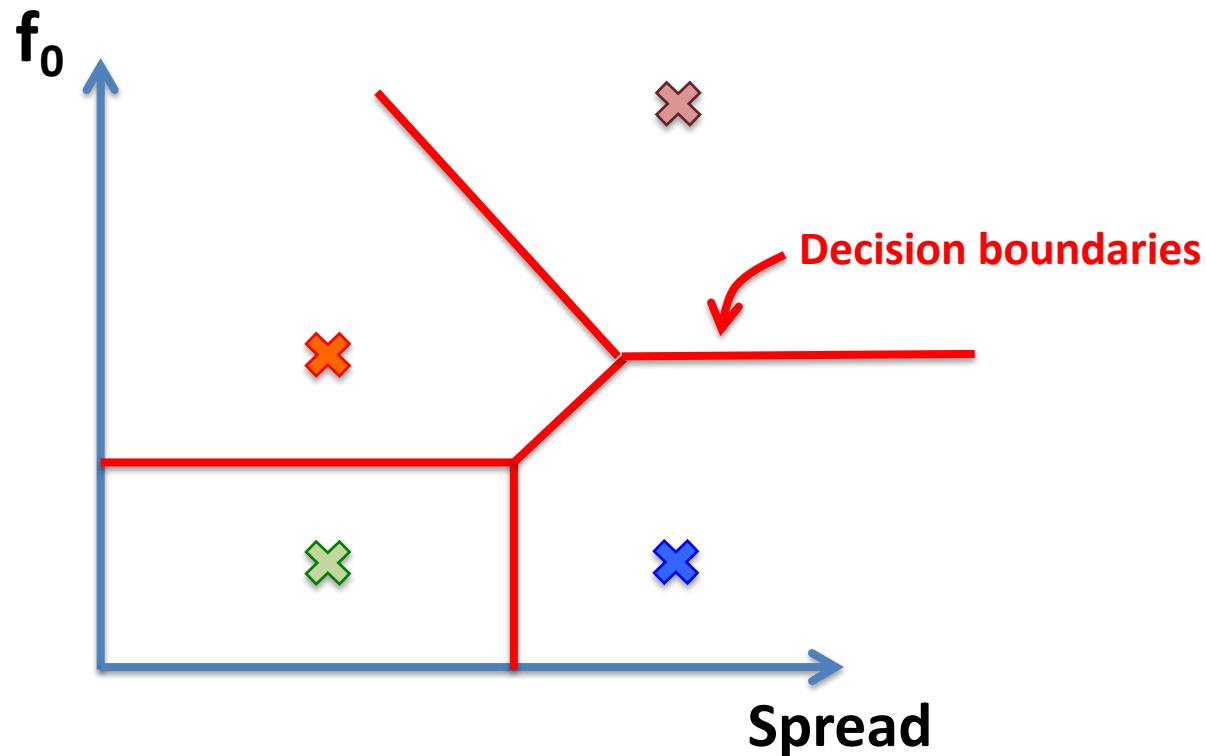
# Nearest-Neighbor



# Nearest-Neighbor

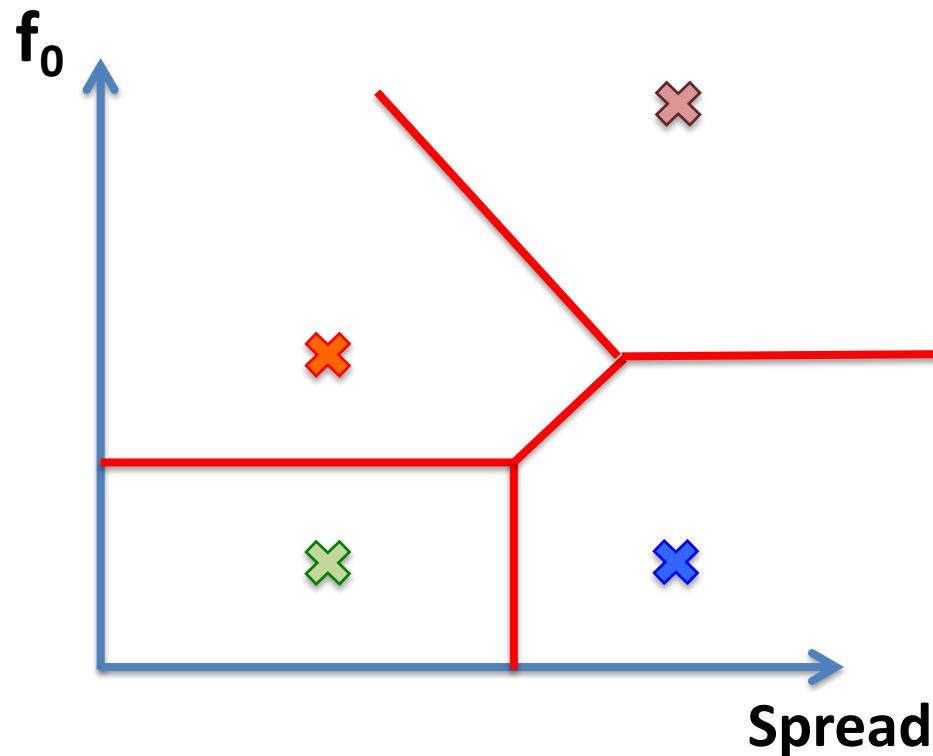


# Nearest-Neighbor



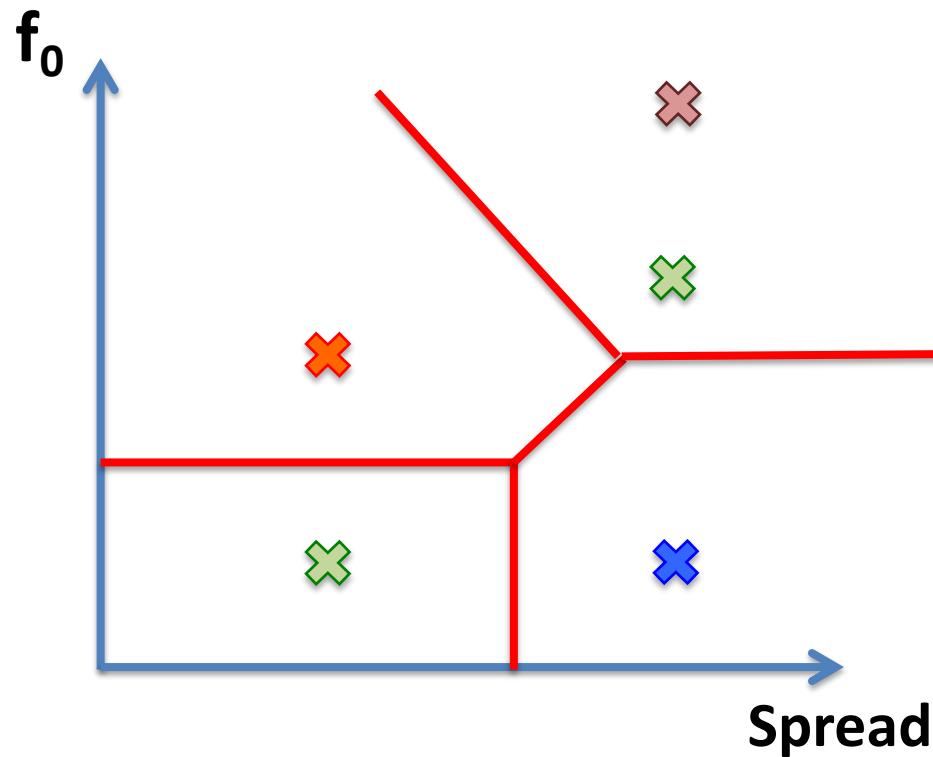
# Nearest-Neighbor

---

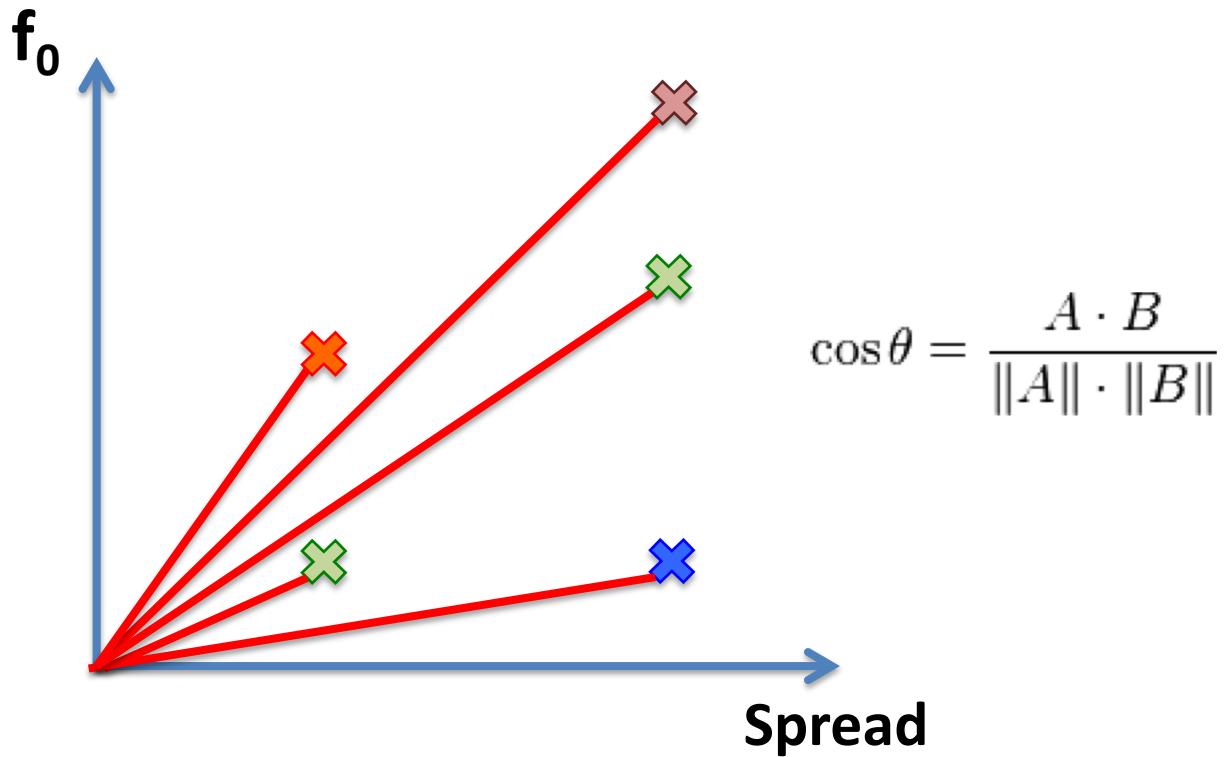


# Nearest-Neighbor

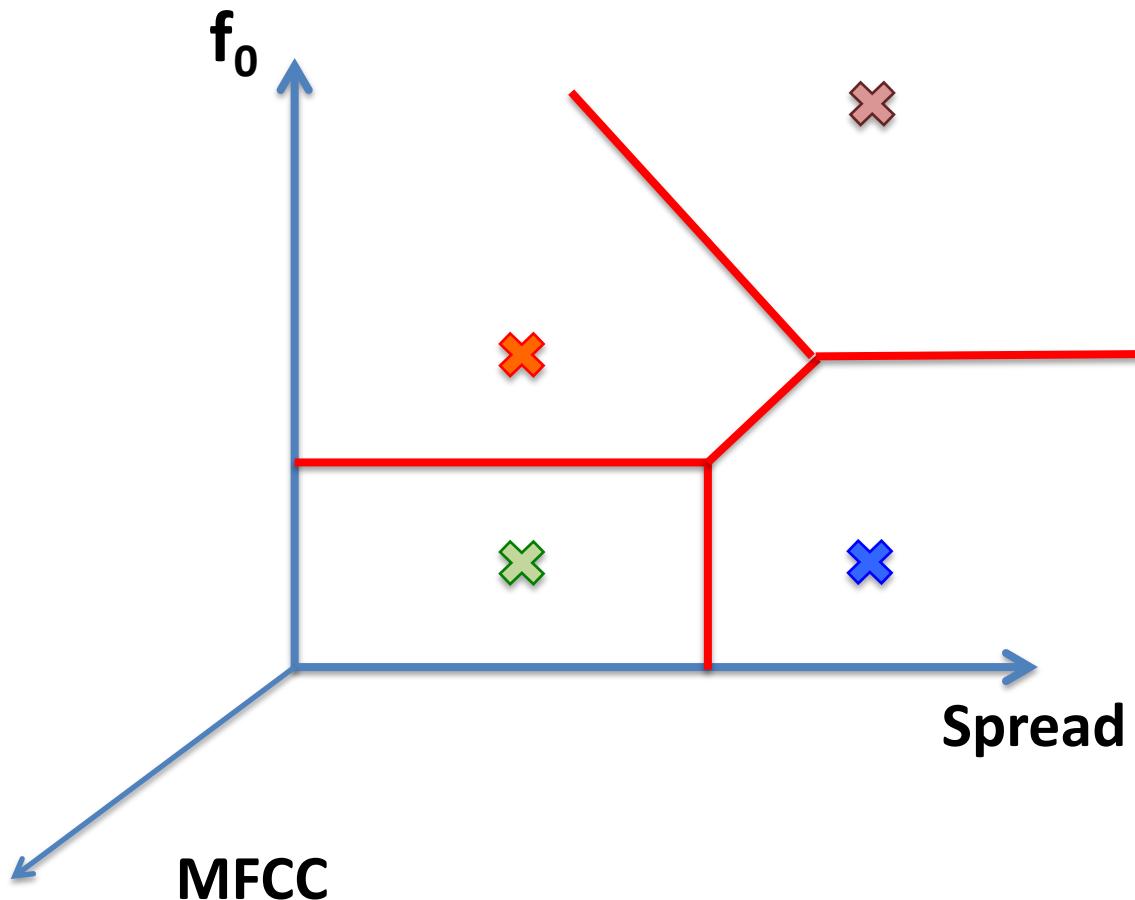
---



# Nearest-Neighbor



# Nearest-Neighbor



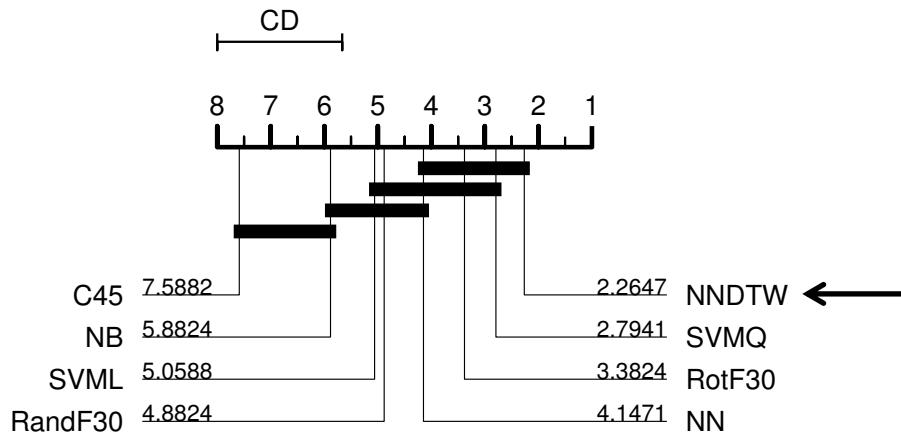
# Nearest-Neighbor

## Typical turnarounds of Nearest-Neighbor

- ⇒ Improve your features (make them problem-specific)
- ⇒ Add dimensions (without overspecifying the problem)
- ⇒ Rely on non-standard metrics (cosine, Bregman, ...)
- ⇒ Go from 1-NN to k-NN (eventually with variable k)

- Nearest-Neighbor might seem like a quite dull approach
- However recent researches show that features are more important than classifier
- A dull classifier works perfectly if features are great

Cf. the ant's path and Occam's razor



Results from [Bagnall et. al 2012] on the **17** datasets from the UCR time series archive

1-NN with DTW is statistically superior to other classification methods (even though not *critical* with SVMQ and RotF30)