Lecture on:

Statistical methods for Machine Learning

From supervised to unsupervised learning of shallow models

Aurélien Decelle: Universidad Complutense de Madrid, Theoretical Physics

Outlines

The goal of this course is to have an introduction over many “elementary” regression methods

Both

→ at the level of “curve fitting”

→ and as a statistical method to embed complex dataset

- Supervised Learning: regression, classification and Bayesian inference
- Unsupervised Learning I: clustering
- Unsupervised Learning II: generative model

Machine Learning or ?

The first part of the lecture deals with regression and classifications. While it is not completely clear at this level the distinction between Machine Learning and Statistical method, we shall refer in the following as Machine Learning to make things simpler.

This course is not about deep-learning → see the lecture of Jean-Luc Parouty.
Still, we will share some vocabulary, method and building blocks

Supervised Learning

In what follow, we will name supervised learning, any learning tasks where the dataset is made of both

→ a set of inputs, usually named X, $\{\boldsymbol{x}_m\}_{m=1,\ldots,M}$

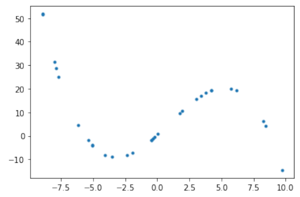
→ a set of targets or labels, usually named y $\{y_m\}_m$

The dataset will be made of many samples, for which we shall use the letter M as the total size of it, and a dimension N. The target will be often of dimension 1, to ease the notation.

Supervised Learning

Some toy examples of applications

Example of regression

$(x, y) =$ 

$\Rightarrow$ Adjust some parameters such that f(x)=y

In many different fields, the family of functions f is used to predict some interesting quantity

→ protein folding

→ weather forecast

→ ...

Example of classification

$x =$

$y =$ "cats"

$\Rightarrow$ Adjust some parameters such that f(x)='cat'

Supervised Learning

Let's discussed linear regression.

The problem can be simply explained as: Having a set of M observations, how can we find the parameters $w$that reproduce the best the data:

$$y^{(m)} = \sum_{i=1}^{N} w_i x_i^{(m)}$$

A classical setting for this problem is to declare that we wish to minimize the square of the different between l.h.s. and the r.h.s.

$$\hat{\boldsymbol{w}} = \operatorname{argmin}_{\boldsymbol{w}} \left\{ \sum_m \left(y^{(m)} - \sum_i w_i x_i^{(m)} \right)^2 \right\}$$

In such a case, we can find a direct expression for a minimizer, writing $\mathbf{X}$ the matrix of data:

$$\hat{\boldsymbol{w}}^T = (XX^T)^{-1} X^T \boldsymbol{y}$$

Supervised Learning

This approach is not limited to a linear set of basis functions. Linear regression can be written into a more general form as

$$y^{(m)} = \sum_{i=1}^{F} w_i \phi_i(\boldsymbol{x}^{(m)}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}^{(m)})$$

Where we use a set of F basis function ϕ. The solution still has a simple direct formulation

$$\hat{\boldsymbol{w}}^T = (\boldsymbol{\phi}^T \boldsymbol{\phi})^{-1} \boldsymbol{\phi}^T \boldsymbol{y}$$

This formulation is interesting because it includes many known cases, for instance using polynomial or Gaussian basis, or can be treated as the output of any automatic feature-crafting method (for instance a neural network)

$$\begin{aligned} \phi_i(x) &= a_i x^i \\ \phi_i(x) &= \exp\left(-\frac{(x-\mu_i)^2}{2\sigma^2}\right) \\ \phi_i(x) &= \sigma\left(\frac{x-\mu_i}{2s}\right), \text{ with } \sigma(x) = (1+\exp(-x))^{-1} \end{aligned}$$

SL: non-linear regression

While linear regression can be solved exactly, we can still grasp how to handle non-linear model.
We have now the following optimization problem:

$$\hat{\boldsymbol{\theta}} = \mathrm{argmin}_{\boldsymbol{w}} \left\{ \sum_m \left(y^{(m)} - f_\theta(\boldsymbol{x}^{(m)}) \right)^2 \right\}$$

Now, it is impossible to minimize it in general, we can use the gradient to define an iterative process minimizing locally the loss

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \eta \nabla_{\boldsymbol{w}} \left(\frac{1}{M} \sum_m \left(y^{(m)} - f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(m)}) \right) \right)^2 = \boldsymbol{\theta}^{(t)} + \frac{2}{M} \eta \sum_m \nabla_{\boldsymbol{w}} f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(m)}) \left(y^{(m)} - f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(m)}) \right)$$

Where we introduced the learning rate η

In practice, we can easily have many local minimum and therefore there is no guaranty to find “a good one”

SL: Bayesian approach

The loss function design previously was somehow ad-hoc. We can use a Bayesian approach that can helps us to understand the meaning of possible regularization and to define different estimators.

We first define a stochastic equation responsible of the generation process of the dataset

$$\tilde{y} = f(\boldsymbol{x}) + \xi, \quad \text{where} \quad \xi \sim \mathcal{N}(0, \sigma_\xi)$$

Using the elementary probability properties, we can write

$$p(\boldsymbol{x}, \tilde{y}|\boldsymbol{\theta}) = \int p(\boldsymbol{x}, \tilde{y}, \xi|\boldsymbol{\theta}) d\xi = \int p(\boldsymbol{x}, \tilde{y}|\xi, \boldsymbol{\theta}) p(\xi) d\xi$$

$$\text{and } p(\boldsymbol{x}, \tilde{y}|\xi, \boldsymbol{\theta}) = \delta(\xi - (\tilde{y} - f_{\boldsymbol{\theta}}(\boldsymbol{x}))$$

We obtain, given a set of parameters, the probability distribution on some dataset:

$$p(\boldsymbol{x}, \tilde{y}|\boldsymbol{\theta}) = \mathbb{E}_\xi \left[\delta(\xi - (\tilde{y} - f_{\boldsymbol{\theta}}(\boldsymbol{x}))\right] = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\tilde{y} - f_\theta(\boldsymbol{x}))^2}{2\sigma^2}\right)$$

SL: Bayesian approach

We obtain, given a set of parameters, the probability distribution on some dataset:

$$p(\boldsymbol{x}, \tilde{y}|\boldsymbol{\theta}) = \mathbb{E}_\xi\left[\delta(\xi - (\tilde{y} - f_{\boldsymbol{\theta}}(\boldsymbol{x})))\right] = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\tilde{y} - f_\theta(\boldsymbol{x}))^2}{2\sigma^2}\right)$$

Now using the Bayes theorem, we can evaluate the probability of our set of parameters given the dataset

$$P(A, B) = P(A|B)P(B) \;\; \text{which gives} \;\; P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

$$p(\boldsymbol{\theta}|\{\boldsymbol{x}^{(m)}\}_m, \{\tilde{y}^{(m)}\}_m) \propto p(\{\boldsymbol{x}^{(m)}\}_m, \{\tilde{y}^{(m)}\}_m|\boldsymbol{\theta})p(\boldsymbol{\theta})$$

Now, we can observe that our previous regression process consists in maximize the distribution ‘a posteriori’. In a case where the prior is uniform we have

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= \operatorname{argmin}_{\boldsymbol{\theta}}\left[\mathcal{L}(\boldsymbol{\theta}, \{\boldsymbol{x}^{(m)}\}_m, \{\tilde{y}^{(m)}\}_m)\right] = \operatorname{argmin}_{\boldsymbol{\theta}}\left[-\log\left(p(\{\boldsymbol{x}^{(m)}\}_m, \{\tilde{y}^{(m)}\}_m|\boldsymbol{\theta})\right)\right] \\ &= \operatorname{argmin}_{\boldsymbol{\theta}}\left[\sum_m \left(\tilde{y}^{(m)} - f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(m)})\right)^2\right],\end{aligned}$$

SL: Bayesian approach

Among the interesting properties of the Bayesian approach,
→ there is the possibility to compare different models (or family of functions)
→ and the interpretation of the regularization as a prior distribution

Prior distribution:

$p(\theta) \propto \exp\left(-\gamma\theta^2\right)$	**L2-regularization ↔ Gaussian prior**
$p(\theta) \propto \exp\left(-\gamma\|\theta\|\right)$	**L1-regularization ↔ Laplace prior**
$p(\theta) \propto p\delta(\theta) + (1-p)p(\theta)$	**Sparse prior**

SL: Bayesian approach

Among the interesting properties of the Bayesian approach,

→ there is the possibility to compare different model (or family of functions)

→ and the interpretation of the regularization as a prior distribution

Model comparison:

Assuming that I have two potential models to compare, can we compute P(M|D) ?

$$p(\mathcal{M}|\mathcal{D}) \propto p(\mathcal{D}|\mathcal{M})p(\mathcal{M})$$

We can explicit the first term as

$$p(\mathcal{D}|\mathcal{M}) = \int d\boldsymbol{\theta} p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M})p(\boldsymbol{\theta}|\mathcal{M}) = \int d\boldsymbol{\theta} p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M})$$

Which correspond to the normalization constant of the posterior distribution!

SL: Bayesian approach

$$p(\mathcal{D}|\mathcal{M}) = \int d\boldsymbol{\theta} p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M}) p(\boldsymbol{\theta}|\mathcal{M}) = \int d\boldsymbol{\theta} p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M})$$

Which correspond to the normalization constant of the posterior distribution!

→ it might seems intractable in general (it is). Yet we can approximate it by a Gaussian distribution aroung its max $\hat{\theta}$

$$\int d\boldsymbol{\theta} p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M}) \approx p(\hat{\boldsymbol{\theta}}|\mathcal{D}, \mathcal{M}) \int \exp\left[(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \Sigma^{-1} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})\right] \approx p(\hat{\boldsymbol{\theta}}|\mathcal{D}, \mathcal{M}) \frac{\det(\Sigma)^{1/2}}{(2\pi)^{D/2}}$$

...using the covariance matrix of the posterior

$$\Sigma_{ij}^{-1} = -\frac{\partial^2}{\partial\theta_i \partial\theta_j} p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M})|_{\hat{\boldsymbol{\theta}}}$$

Using these expressions we can compute the following ratio

$$r = \frac{p(\mathcal{M}_1|\mathcal{D})}{p(\mathcal{M}_2|\mathcal{D})} = \frac{p(\mathcal{D}|\mathcal{M}_1)}{p(\mathcal{D}|\mathcal{M}_2)} = \frac{p(\hat{\boldsymbol{\theta}}|\mathcal{D},\mathcal{M}_1)\det(\Sigma_1^{1/2})}{p(\hat{\boldsymbol{\theta}}|\mathcal{D},\mathcal{M}_2)\det(\Sigma_2^{1/2})}$$

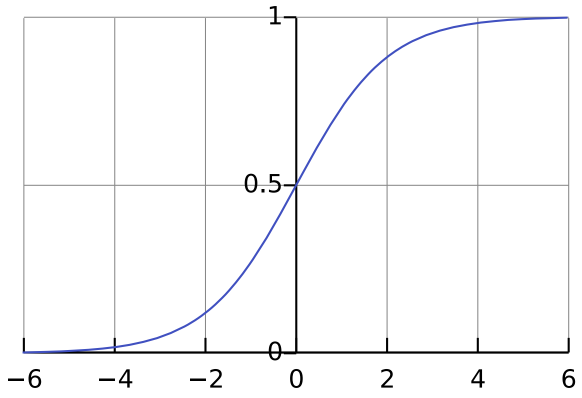
SL: Classification

It is easy to turn a regression task into a classification one, and the same Bayesian formalism can be used.
We need to define a proxy to estimate the probability to belong to a particular class. For a binary classification problem we can use the following:

Probability to be in the class (0) $p(\mathcal{C}_0|\mathcal{D}) = \frac{1}{1+\exp(\boldsymbol{\theta x})}$

Probability to be in the class (1) $p(\mathcal{C}_1|\mathcal{D}) = 1 - p(\mathcal{C}_0|\mathcal{D}) = \frac{\exp(\boldsymbol{\theta x})}{1+\exp(\boldsymbol{\theta x})} = \frac{1}{1+\exp(-\boldsymbol{\theta x})}$

Logistic or sigmoid function



By convention, we usually use the coordinate 0 to encode the bias (the center)

SL: Classification

Probability to be in the class (0) $p(\mathcal{C}_0|\mathcal{D}) = \frac{1}{1+\exp(\boldsymbol{\theta x})}$

Probability to be in the class (1) $p(\mathcal{C}_1|\mathcal{D}) = 1 - p(\mathcal{C}_0|\mathcal{D}) = \frac{\exp(\boldsymbol{\theta x})}{1+\exp(\boldsymbol{\theta x})} = \frac{1}{1+\exp(-\boldsymbol{\theta x})}$

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

We can now write the likelihood, using the true labels y

$$p(y|\boldsymbol{\theta}) = p(\mathcal{C}_0|\mathcal{D})^{1-y} p(\mathcal{C}_1|\mathcal{D})^{y} = \sigma(\boldsymbol{\theta x})^{y}(1-\sigma(\boldsymbol{\theta x}))^{1-y}$$

$$\mathcal{L}_{logistic} = -\sum_m \left[y^{(m)} \log(\sigma(\boldsymbol{\theta x}^{(m)})) + (1-y^{(m)}) \log(1-\sigma(\boldsymbol{\theta x}^{(m)}))\right]$$

Binary cross-entropy

And the gradient is given by

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}_{logistic} = \frac{1}{M}\sum_m \boldsymbol{x}^{(m)}(y^{(m)} - \sigma(\boldsymbol{\theta x}^{(m)}))$$

SL: Classification

Interesting properties:

1) Convexity: the Hessian of the loss function being positive definite, the likelihood is convex

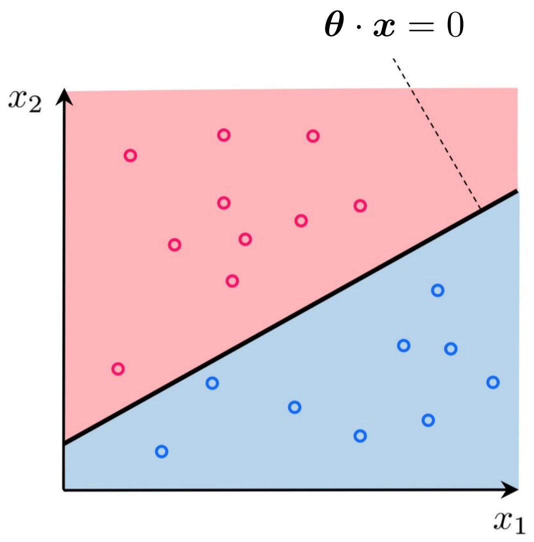
$$\mathrm{H}_{ij} = \frac{\partial \mathcal{L}_{logistic}}{\theta_i \theta_j} = \sum_m x_i^{(m)} \sigma^{(m)} (1 - \sigma^{(m)}) x_j^{(m)}$$

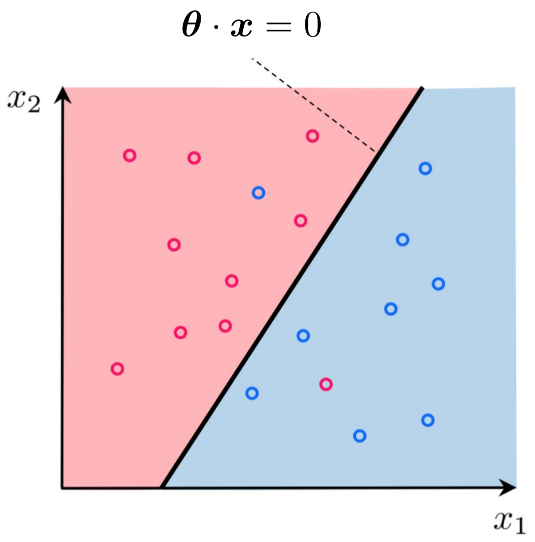
$$\sum_{i,j} u_i H_{ij} u_i > 0$$

2) Geometric interpretation

$$\frac{p(\mathcal{C}_0|\mathcal{D})}{p(\mathcal{C}_1|\mathcal{D})} = \exp(-\boldsymbol{\theta x})) = 1 \iff \boldsymbol{\theta x} = 0$$

An hyperplan of dimension N-1!





SL: Classification

Multiclass generalization

$$p(\boldsymbol{x} \in \mathcal{C}_a|\boldsymbol{\theta}) = y_a(\boldsymbol{x}) = \frac{\exp(\boldsymbol{\theta}_a \boldsymbol{x})}{\sum_b \exp(\boldsymbol{\theta}_b \boldsymbol{x})}$$

Softmax functino

Label: y = {1,2,...,K} → one-hot vector

$$\tilde{y}_a = \delta_{a,y} = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$\mathcal{L}_{multiclasse} = \sum_m \left[\sum_a \tilde{y}_a^{(m)} \log(y_a(\boldsymbol{x}^{(m)})) \right]$$

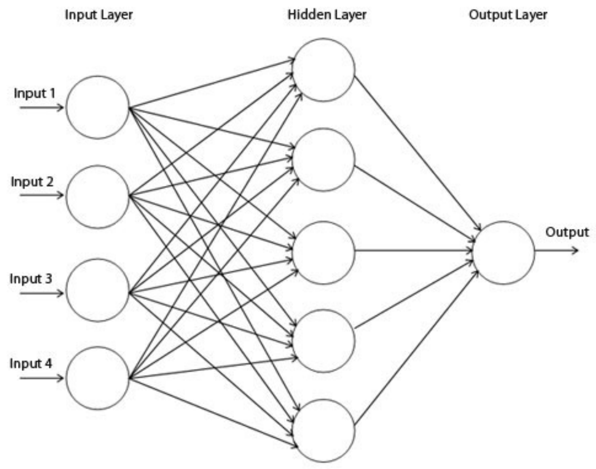
Categorial Cross-Entropy

$$\frac{\partial \mathcal{L}}{\partial \theta_{ia}} = x_i \left[\tilde{y}_a - y_a(\boldsymbol{x}) \right]$$

SL: one-hidden layer network

So far, we have discussed mostly (quasi) linear models.

We can replicate this construction by stacking linear+activation function



$\boldsymbol{\theta}^{(1)}$ set of weights between the first two layers

$\boldsymbol{\theta}^{(2)}$ set of weights between the last two layers

We define the layer's activation:

$$\boldsymbol{z} = f_{\text{act}}(\boldsymbol{\theta}^{(1)}\boldsymbol{x})$$

Input layer: $\boldsymbol{x}$

Intermediate layer: $\boldsymbol{z}$

Output layer: $\boldsymbol{y}$

regression or classif.

$$\begin{cases} y(\boldsymbol{z}) = \sum_i \theta_i^{(2)} z_i \\ y_a(\boldsymbol{z}) = \frac{\exp(\sum_i \theta_{ai}^{(2)} z_i)}{\sum_b \exp(\sum_i \theta_{bi}^{(2)} z_i)} \end{cases}$$

<u>**Universal approximator !**</u>

SL: CrossValidation-Overfitting

We discussed many different methods that are used to do regression or classification. Yet, we never talk about one crucial aspect of "curve fitting": how to evaluate the obtained models and how it is related to overfitting.

In general, we use the following metrics

→ for regression $E_{\text{reg}} = \frac{1}{M}\sum_m \left(\tilde{y}^{(m)} - y(\boldsymbol{x}^{(m)})\right)^2$

→ for classification $E_{\text{classif}} = \frac{1}{M}\sum_m \delta(\tilde{y}^{(m)} = \text{argmax}\{y_a(\boldsymbol{x}^{(m)})\})$

The problem: we are measuring the same object as the one we are minimizing !

→ we would like to understand how the model perform <u>on other data...</u>

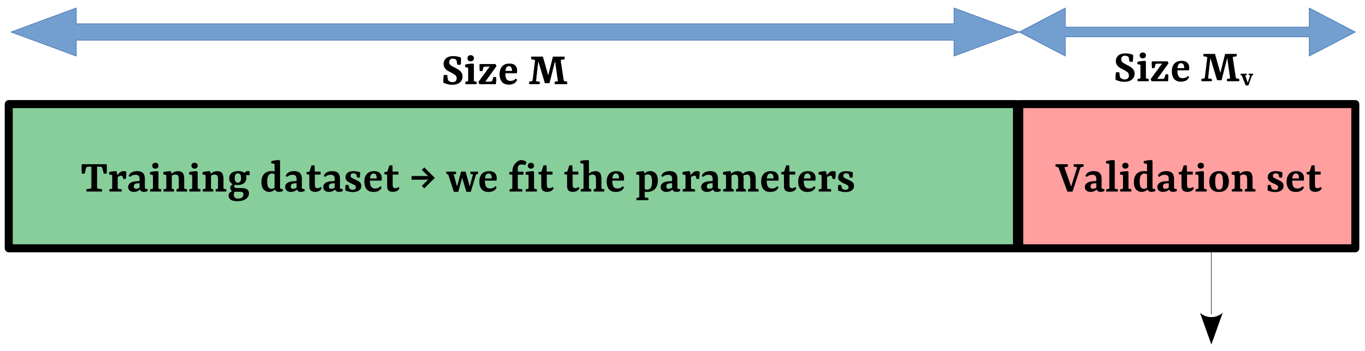
Ideally, we would like to measure

$$E_G = \mathbb{E}[(\tilde{y} - y(\boldsymbol{x}))^2] = \int (\tilde{y} - y(\boldsymbol{x}))^2 p(\boldsymbol{x}, \tilde{y}) d\boldsymbol{x} d\tilde{y}$$

We in general we do not have the distribution over the dataset ...

SL: CrossValidation-Overfitting

To do a non-biased estimate, we need to measure the loss on unseen data!



We evaluate the loss on !

$$E_G \approx \frac{1}{M_v} \sum_{m \in M_v} (\tilde{y}^{(m)} - y(\boldsymbol{x}^{(m)}))^2$$

It is good → uncorrelated to the training dataset but ...

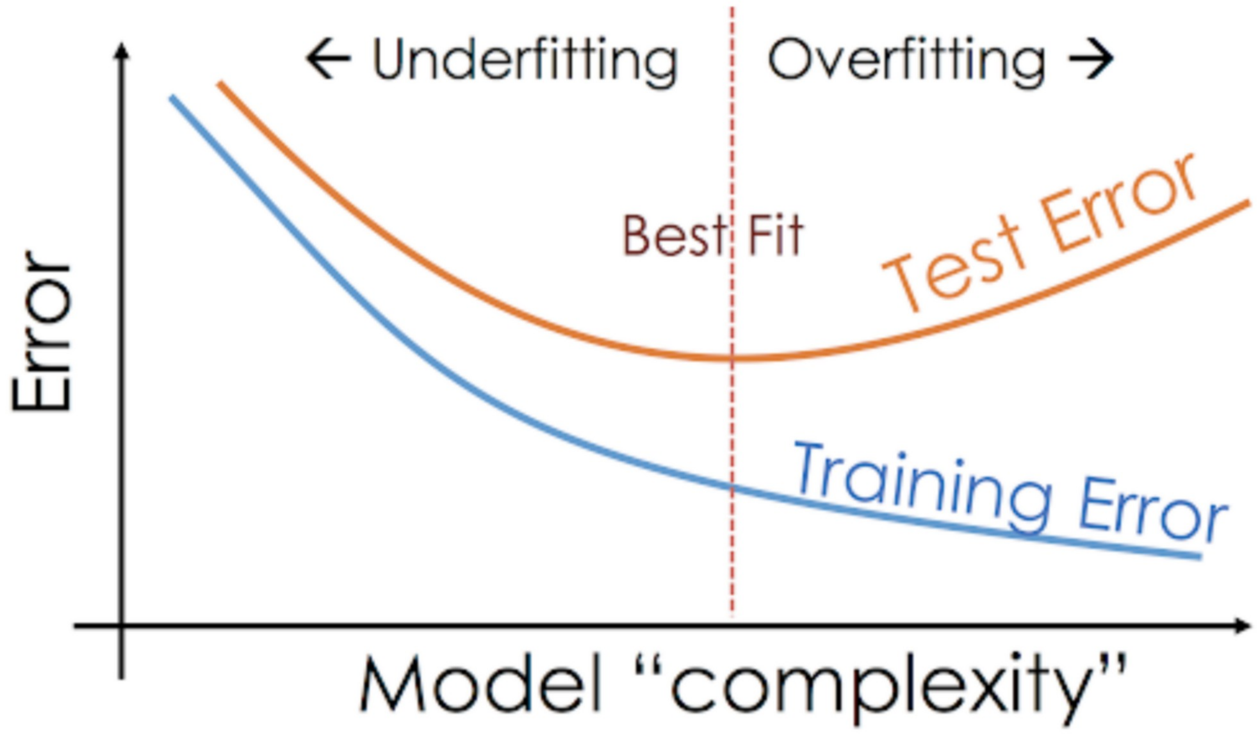
→ M need to be large to correctly learn the model

→ M$_v$ need to be large to correctly estimate the loss

→ We are not using all the data

This is a problem for small-moderate's size dataset

SL: CrossValidation-Overfitting



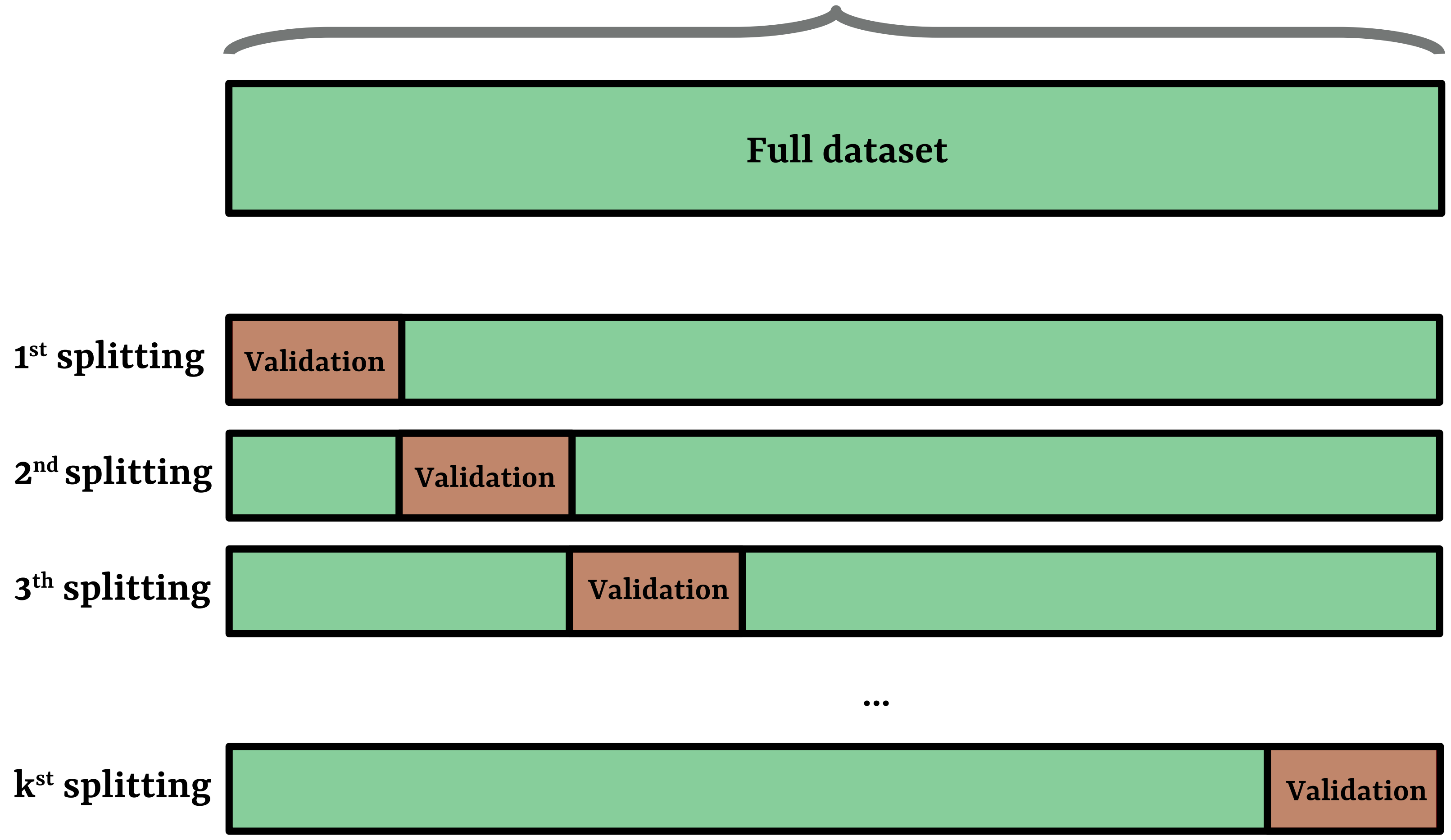
Error on the validation set

Error on the training set

→ we see that up to some point, the learning is getting better on unseen data, then it gets worse

SL: CrossValidation-Overfitting

To avoid having data that are not used in the learning, we can use
→ the k-fold cross-validation



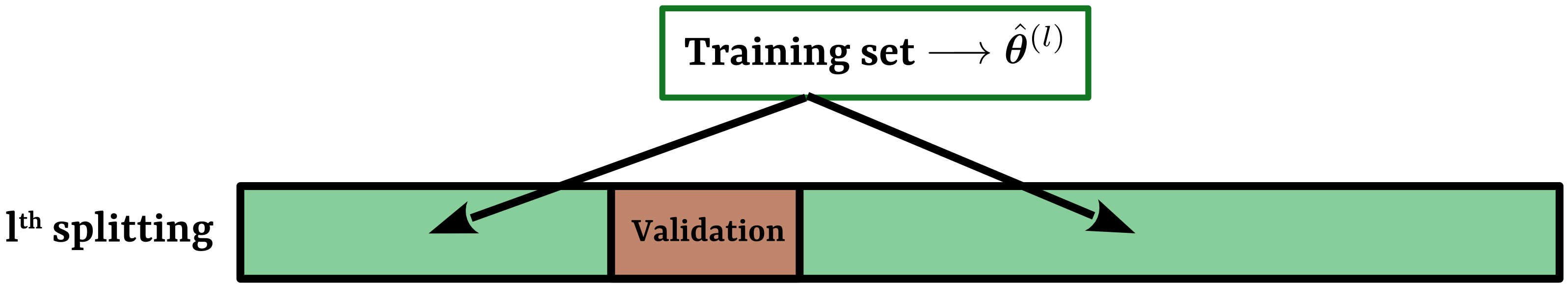
SL: CrossValidation-Overfitting

To avoid having data that are not used in the learning, we can use
→ the k-fold cross-validation

We approximate the generalization error as

$$E_G \approx E_{\text{k-fold}} = \frac{1}{k}\sum_{l=1}^{k} E^{(l)}(\boldsymbol{\theta}^{(l)})$$

$E^{(l)}$: loss computed on the l-th validation partitioning, computing on the model trained with the remaining dataset.


Training set $\longrightarrow \hat{\boldsymbol{\theta}}^{(l)}$
l^th splitting
Validation

SL: CrossValidation-Overfitting

How can we understand this phenomena into more details ?

<u>First</u>: lower bound on the generalization error

Under the Bayesian approach, the generalization error is given by

$$\mathbb{E}(\mathcal{L}) = \int \left(f_{\boldsymbol{\theta}}(\boldsymbol{x}) - y\right)^2 p(\boldsymbol{x}, y) d\boldsymbol{x} dy$$

Assuming a totally flexible model, the minimum on this loss is given by

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int y p(y|\boldsymbol{x}) dy = \mathbb{E}[y|\boldsymbol{x}]$$

Let’s expand the loss around this point $(f_{\boldsymbol{\theta}}(x) - \mathbb{E}[y|x] + \mathbb{E}[y|x] - y)^2$

$$\mathbb{E}(\mathcal{L}) = \underbrace{\int \left(f_{\boldsymbol{\theta}}(\boldsymbol{x}) - \mathbb{E}[y|\boldsymbol{x}]\right)^2 p(\boldsymbol{x}) d\boldsymbol{x}}_{\text{How well the model fits the data} \rightarrow \text{can be made arbitrarily small}} + \underbrace{\int \left(\mathbb{E}[y|\boldsymbol{x}] - y\right)^2 p(y|\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x}}_{\text{Intrisic noise of the dataset} \rightarrow \text{irreducible error}}$$

SL: CrossValidation-Overfitting

How can we understand this phenomena into more details ?

First: lower bound on the generalization error

Second: Bias-Variance decomposition

Now we can look at the estimator obtained by minimizing the loss function. To demonstrate our point, we consider the following setting.

We denote by $f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}$ the estimator obtained for a given dataset $\mathcal{D}$

Let's focus on the first term on the r.h.s of the following eq.

$$\mathbb{E}(\mathcal{L}) = \int \left(f_{\boldsymbol{\theta}}(\boldsymbol{x}) - \mathbb{E}[y|\boldsymbol{x}]\right)^2 p(\boldsymbol{x}) d\boldsymbol{x} + \int \left(\mathbb{E}[y|\boldsymbol{x}] - y\right)^2 p(y|\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x}$$

Let's expand around the solution the estimator average over all possible dataset

$$\mathrm{f}_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}} - \mathbb{E}[y|\boldsymbol{x}] = f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}} - \mathbb{E}_{\mathcal{D}}[f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}] + \mathbb{E}_{\mathcal{D}}[f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}] - \mathbb{E}[y|\boldsymbol{x}]$$

Using this decomposition, and replace the average over p(x,y) by the average over all possible dataset we have

$$\mathrm{E}_{\mathcal{D}}\left[\left(f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}(\boldsymbol{x}) - \mathbb{E}[y|\boldsymbol{x}]\right)^2\right] = \mathbb{E}_{\mathcal{D}}\left[\left(\mathbb{E}_{\mathcal{D}}[f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}(\boldsymbol{x})] - \mathbb{E}[y|\boldsymbol{x}]\right)\right]^2 + \mathbb{E}_{\mathcal{D}}\left[\left(f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}(\boldsymbol{x})]\right)^2\right]$$

SL: CrossValidation-Overfitting

How can we understand this phenomena into more details ?

First: lower bound on the generalization error

Second: Bias-Variance decomposition

Now we can look at the estimator obtained by minimizing the loss function. To demonstrate our point, we consider the following setting.

We denote by $f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}$ the estimator obtained for a given dataset $\mathcal{D}$

Let's focus on the first term on the r.h.s of the following eq.

$$\mathbb{E}(\mathcal{L}) = \int \left(f_{\boldsymbol{\theta}}(\boldsymbol{x}) - \mathbb{E}[y|\boldsymbol{x}]\right)^2 p(\boldsymbol{x})d\boldsymbol{x} + \int \left(\mathbb{E}[y|\boldsymbol{x}] - y\right)^2 p(y|\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}$$

Let's expand around the solution the estimator average over all possible dataset

$$\mathrm{f}_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}} - \mathbb{E}[y|\boldsymbol{x}] = f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}} - \mathbb{E}_{\mathcal{D}}[f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}] + \mathbb{E}_{\mathcal{D}}[f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}] - \mathbb{E}[y|\boldsymbol{x}]$$

Using this decomposition, and replace the average over p(x,y) by the average over all possible dataset we have

$$\mathrm{E}_{\mathcal{D}}\left[\left(f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}(\boldsymbol{x}) - \mathbb{E}[y|\boldsymbol{x}]\right)^2\right] = \mathbb{E}_{\mathcal{D}}\left[\left(\mathbb{E}_{\mathcal{D}}[f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}(\boldsymbol{x})] - \mathbb{E}[y|\boldsymbol{x}]\right)\right]^2 + \mathbb{E}_{\mathcal{D}}\left[\left(f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}(\boldsymbol{x})]\right)^2\right]$$

SL: CrossValidation-Overfitting

How can we understand this phenomena into more details ?

First: lower bound on the generalization error

Second: Bias-Variance decomposition

What does this mean ?

$$\mathrm{E}_{\mathcal{D}}\left[(f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}(\boldsymbol{x}) - \mathbb{E}[y|\boldsymbol{x}])^2\right] = \underbrace{\left[(\mathbb{E}_{\mathcal{D}}[f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}(\boldsymbol{x})] - \mathbb{E}[y|\boldsymbol{x}])\right]^2}_{\textbf{BIAS}} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[(f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}(\boldsymbol{x})])^2\right]}_{\textbf{VARIANCE}}$$

The Bias term tells you how well the model fit the dataset. If the model is very "complex", it can be made very small. At the opposite, simple model might have high bias.

The Variance term tells you about the dispersion of the estimation of your parameters when you vary the dataset. Complex model might change a lot the parameter from one dataset to another one, thus leading to high variance. Simple model at the opposite do not vary much

SL: CrossValidation-Overfitting

How can we understand this phenomena into more details ?

First: lower bound on the generalization error

Second: Bias-Variance decomposition

What does this mean ?

$$\mathrm{E}_{\mathcal{D}}\left[(f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}(\boldsymbol{x}) - \mathbb{E}[y|\boldsymbol{x}])^2\right] = \underbrace{\left[(\mathbb{E}_{\mathcal{D}}[f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}(\boldsymbol{x})] - \mathbb{E}[y|\boldsymbol{x}])\right]^2}_{\textbf{BIAS}} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[(f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[f_{\hat{\boldsymbol{\theta}}_{\mathcal{D}}}(\boldsymbol{x})])^2\right]}_{\textbf{VARIANCE}}$$

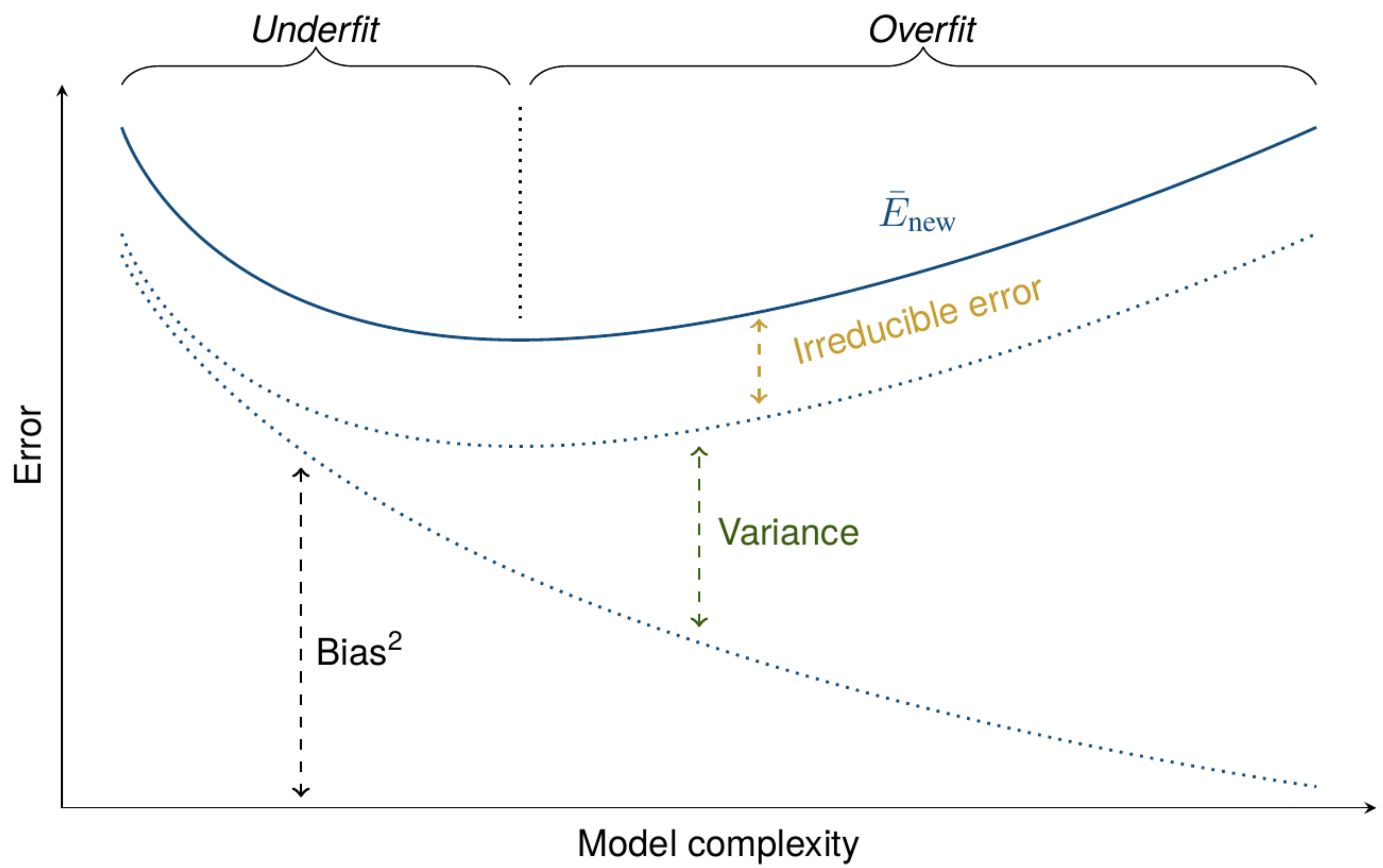
This is the Bias-Variance trade-off. When using very complex model, you can fit perfectly the training set, but at the cost of having bad generalization error due to the variance term.

SL: CrossValidation-Overfitting

How can we understand this phenomena into more details ?

First: lower bound on the generalization error

Second: Bias-Variance decomposition



SL: Overfitting in linear-reg

The Bias-Variance trade-off explain a part of the classical overfitting that we see sometimes.
→ still, deep-neural network have a HUGE number of parameters and seem not to suffer such problem.

In a linear-regression setting, we can get a nice understanding of why regularization helps.
We consider a teacher-student setting:
→ a dataset has been prepared, assuming some ground truth parameters:

$$y^{(m)} = \sum_i \theta_i^T x_i^{(m)} + \epsilon^{(m)}$$

Now, we can write the continuous limit of the gradient descent as

$$\tau \frac{\partial \boldsymbol{\theta}}{dt} = \boldsymbol{y} X - \boldsymbol{\theta}(X^T X)$$

SL: Overfitting in linear-reg

Now, we can write the continuous limit of the gradient descent as

$$\tau \frac{\partial \boldsymbol{\theta}}{dt} = \boldsymbol{y}X - \boldsymbol{\theta}(XX^T)$$

These equations can be diagonalized in the eigenbasis of the dataset ... what ?

We can always decompose a rectangular matrix as

$$X = V\Lambda U^T$$
$$C = XX^T = V\Lambda^2 V^T$$

The decomposition of X is called SVD for Singular Value Decomposition

We see how it is related to the covariance matrix and therefore the Principal Value Decomposition

Recall:

→ the eigenvectors V correspond to the directions of maximum variance

→ the eigenvalues of the PCA correspond to the value of the variance in those directions

SL: Overfitting in linear-reg

Now, we can write the continuous limit of the gradient descent as

$$\tau \frac{\partial \boldsymbol{\theta}}{dt} = \boldsymbol{y} X - \boldsymbol{\theta}(X^T X)$$

We can project these couple equations on the eigenvalues of the dataset

$$\begin{aligned} \tau \frac{dz_i}{dt} &= \lambda_i (z_i^T - z_i) + \tilde{\epsilon}_i \sqrt{\lambda_i} \\ z &= \boldsymbol{\theta} V \\ X &= V \Lambda U^T \end{aligned}$$

We can solve this equation and obtain the behavior of the generalization error in time

$$\mathrm{E}_g(t) = \int \rho(\lambda)(\sigma_{w^T}^2 + \sigma_0^2) \exp(-\tfrac{-2\lambda t}{\tau}) + \tfrac{\sigma_\epsilon^2}{\lambda \sigma_{w^T}^2} \left(1 - \exp(-\tfrac{-\lambda t}{\tau})\right)^2 d\lambda + \tfrac{\sigma_\epsilon^2}{\sigma_{w^T}^2}$$

$$\begin{aligned} \sigma_\epsilon^2 &: \text{ variance of the error term} \\ \sigma_{w^T}^2 &: \text{ variance of the teacher's weights} \\ \sigma_0^2 &: \text{ variance of the student's initial cdt} \\ \rho(\lambda) &: \text{ eigenvalues's density} \end{aligned}$$

SL: Overfitting in linear-reg

We can solve this equation and obtain the behavior of the generalization error in time

$$\mathrm{E}_g(t) = \int \rho(\lambda)(\sigma^2_{w^T} + \sigma^2_0)\exp(-\tfrac{-2\lambda t}{\tau}) + \tfrac{\sigma^2_\epsilon}{\lambda\sigma^2_{w^T}}\left(1 - \exp(-\tfrac{-\lambda t}{\tau})\right)^2 d\lambda + \tfrac{\sigma^2_\epsilon}{\sigma^2_{w^T}}$$

$$\begin{aligned} \sigma^2_\epsilon &: \text{variance of the error term} \\ \sigma^2_{w^T} &: \text{variance of the teacher's weights} \\ \sigma^2_0 &: \text{variance of the student's initial cdt} \\ \rho(\lambda) &: \text{eigenvalues's density} \end{aligned}$$

What does this equation tells us ?

→ information on the dataset are contained in the principal values of the dataset (eigenvalues)

→ Each mode has a relaxation time that is re-scaled by the eigenvalue

In general, the lowest modes of the eigenvalues of the covariance matrix, correspond to noise in the dataset.

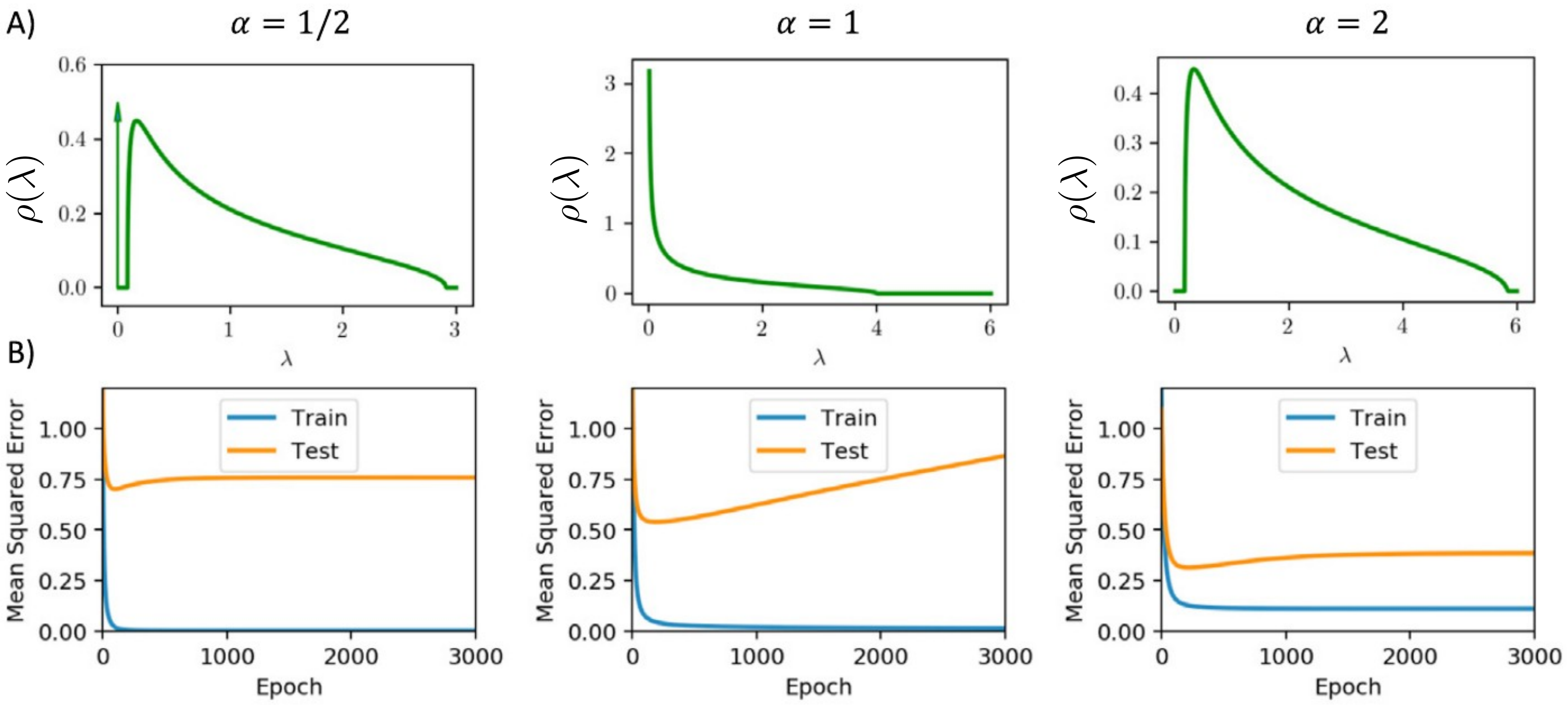
→ THEREFORE: the learning at large time is basically fitting the noise !

SL: Overfitting in linear-reg

We can solve this equation and obtain the behavior of the generalization error in time

$$\mathrm{E}_g(t) = \int \rho(\lambda)(\sigma^2_{w^T} + \sigma_0^2)\exp(-\tfrac{-2\lambda t}{\tau}) + \tfrac{\sigma_\epsilon^2}{\lambda\sigma^2_{w^T}}\left(1 - \exp(-\tfrac{-\lambda t}{\tau})\right)^2 d\lambda + \tfrac{\sigma_\epsilon^2}{\sigma^2_{w^T}}$$

$$\alpha = \frac{\#\text{samples}}{\#\text{parameters}}$$



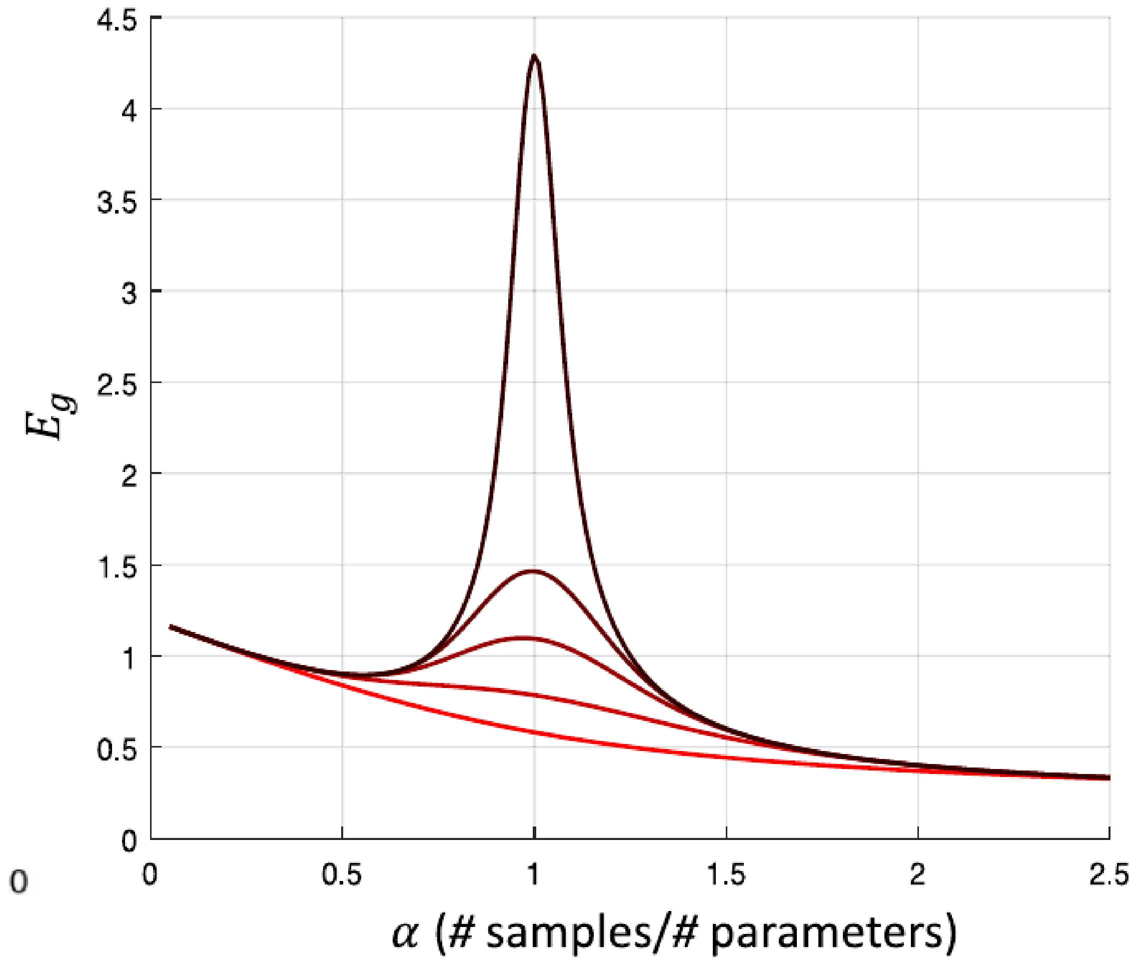
SL: Overfitting in linear-reg

We can solve this equation and obtain the behavior of the generalization error in time

$$\mathrm{E}_g(t) = \int \rho(\lambda)(\sigma^2_{w^T} + \sigma^2_0)\exp(-\tfrac{-2\lambda t}{\tau}) + \tfrac{\sigma^2_\epsilon}{\lambda\sigma^2_{w^T}}\left(1 - \exp(-\tfrac{-\lambda t}{\tau})\right)^2 d\lambda + \tfrac{\sigma^2_\epsilon}{\sigma^2_{w^T}}$$

C)

$$\alpha = \frac{\#\text{samples}}{\#\text{parameters}}$$



From black to red:

Training time increases !

Advani, Saxe 17'

SL: Overfitting in linear-reg

We can proceed to the same kind of analysis with and L2-regularization.

In such setting, we can write the generalization error at convergence:

$$\mathrm{p}(\boldsymbol{\theta}) \propto \exp\left(-\frac{\gamma}{2}\boldsymbol{\theta}^2\right)$$

$$\mathrm{E}_g = \int \rho(\lambda) \left[\sigma_{w^T}^2 \frac{\gamma^2}{(\gamma+\lambda)^2} + \frac{\lambda^2\sigma_\epsilon^2}{\lambda\sigma_{w^T}^2}\right] d\lambda + \frac{\sigma_\epsilon^2}{\sigma_{w^T}^2}$$

We can compute an optimal value of the L2-parameter

$$\gamma = \frac{\sigma_\epsilon^2}{\sigma_{w^T}^2}$$

In particular, it can be shown that the L2-regularization is optimal in this case, assuming Gaussian dataset etc.

SL: Take-Away message

- **Linear regression is simple but provide many crucial insight**
- **One needs to be careful to avoid overfitting**
- **The Bayesian formalism allows to interpret many aspects of the regression/classification**
- **Regularization of some sorts is needed in general**

Why do deep-learning is so good despite have billions of parameters ?

→ (in part) thanks to implicit and explicit regularization

L2-Reg

Stochastic Gradient (mini-batches)

Dropout, batch-norm, …

UnSupervised Learning

The goal is this part is to present some simple clustering methods.
We will follow the following path:

1) **Simple Geometrical clustering : k-means algorithm**
2) **A quick view on DBSCAN**
3) **Mixture models**

UnSupervised Learning

In unsupervised learning context, we are provided with a dataset but not label !
Therefore, we can not minimize some error given the dataset (well...).
Many approaches exist to adjust the parameters, we list a few of them

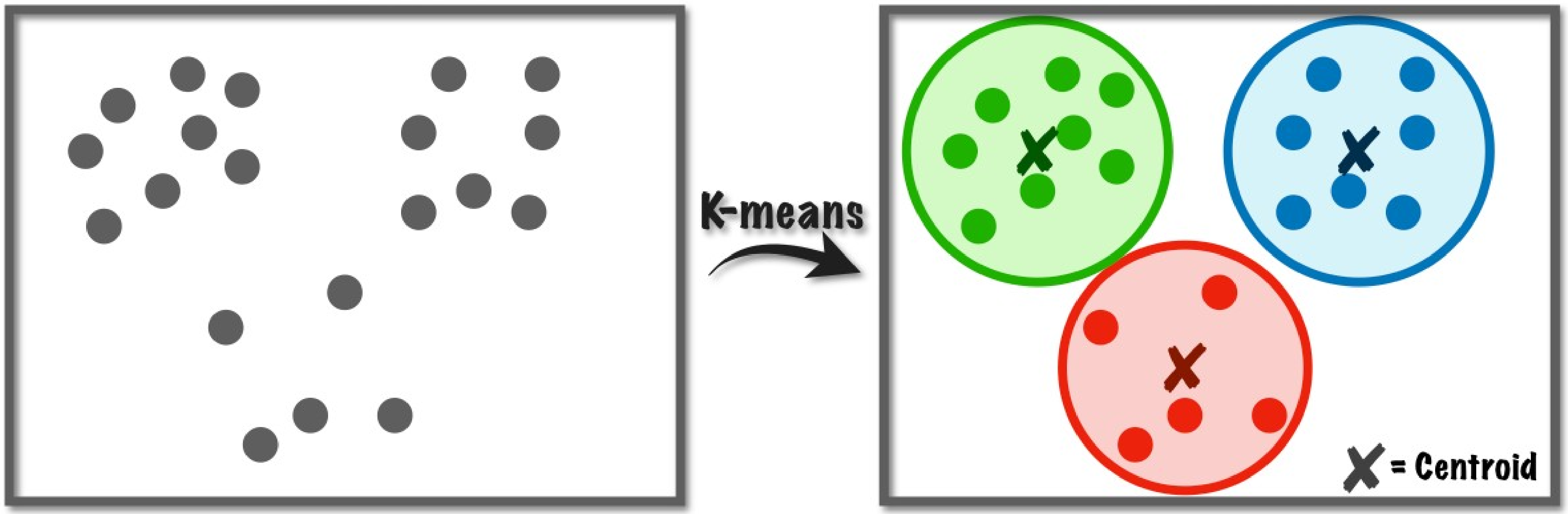
1) Defining a probability distribution and maximizing the likelihood.
The result depend on the chosen model and how it can describe the structure of the dataset.

2) Adversarial approach: the likelihood is implicitly improved using two competing models

3) Diffusion: we learn a diffusion process that map random points to en empirical distribution

UL: k-means

The idea of k-means is very simple.
Having some point lying in a D-dimensional space (with a metric), we will group them according to their closeness to some center.
Of course, we do not know the position of the centers, we will iteratively try to find them



UL: k-means

The algorithm works as follow. Assuming we have M datapoints and K centers.

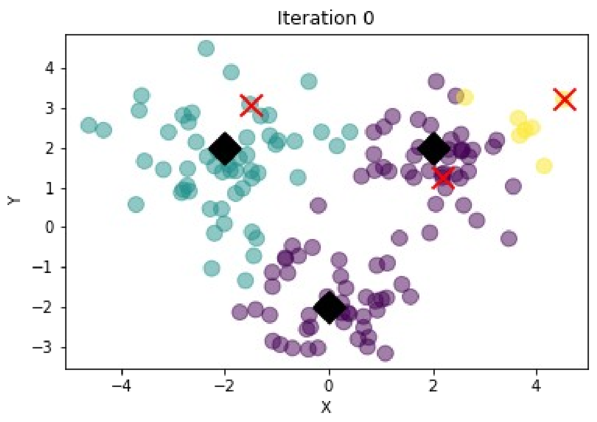
```
1) For each data m:
        - assign data m to the closest cluster $m_k$
2) For each cluster k:
        - change the position of the center k to its barycenter
```

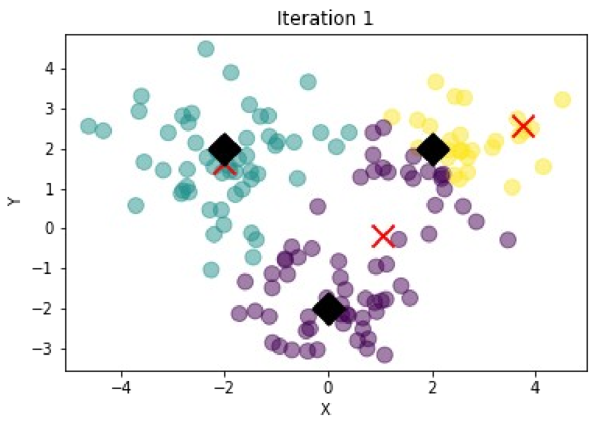
UL: k-means

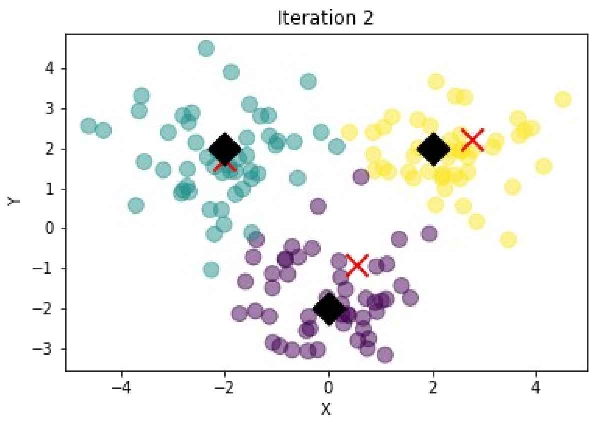
Action !

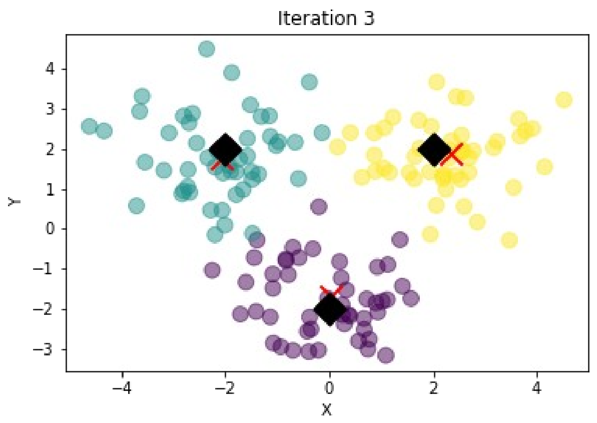
◊ true centers

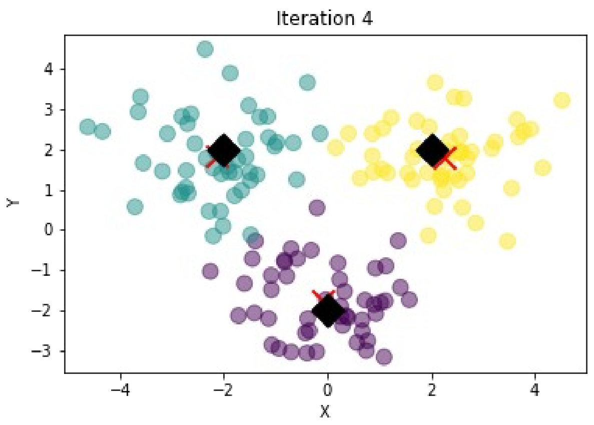
X k-means centers











UL: k-means

Pros:

- very simple

- very fast

Cons:

- the convergence depends on the initial conditions

- too simple: can not handle well clusters of different sizes

- how to fix the number of clusters ?

UL: DBSCAN

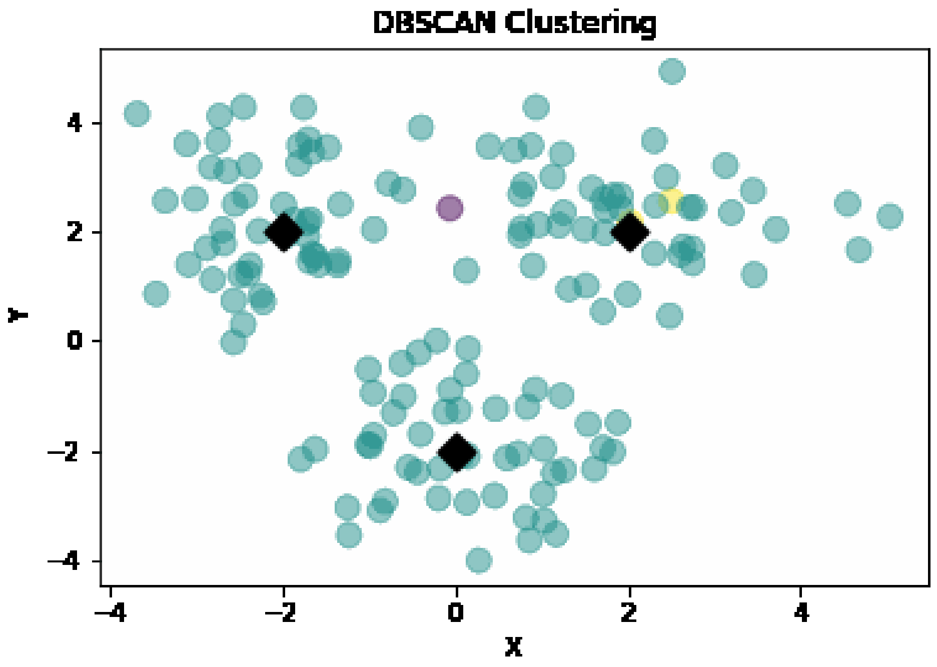
Before going into a more elaborate version of k-means, let's look at two other methods.

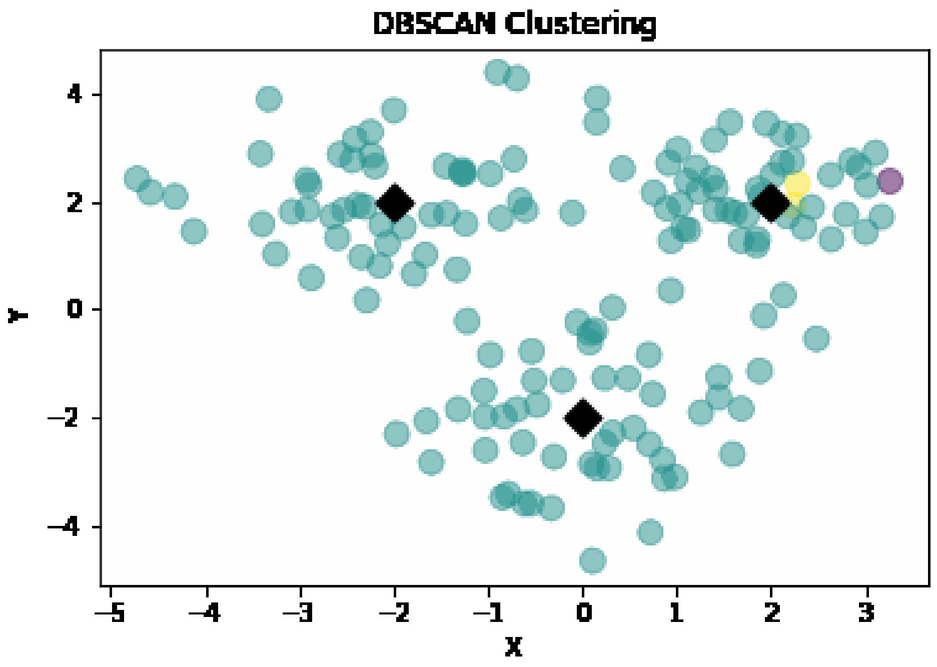
<u>DBSCAN:</u>

In practice, DBSCAN is grouping points together whenever the cluster is locally exceeding a given density threshold

It has two parameters:

- **The radius within which the density is computed**
- **The number of minimal points to form a dense region**





UL: DBSCAN

Pros:

- can adapt to various shapes
- the number of clusters is not predetermined

Cons:

- problems with clusters of different densities
- can be slow

UL: Mixture models

The k-means model is particularly appealing to develop a Bayesian version.
The general idea is to modelize the dataset by a mixture of distributions (usually Gaussian), for which we will learn the parameters (mean, variance,...).

The model that we consider is the following, for a given datapoint:

$$p(\boldsymbol{x}|\{\boldsymbol{\theta}\}) = \sum_a^K \rho_a \prod_i \left[\sqrt{\frac{1}{(2\pi\sigma_{ia})^2}} \exp\left(-\frac{(x_i-\mu_{ia})^2}{2\sigma_{ia}^2}\right)\right] = \sum_a \rho_a p(\boldsymbol{x}|\mathcal{C}_a, \boldsymbol{\theta}_a)$$

In this expression we have

$$\boldsymbol{\theta} = \begin{cases} \rho_a & : \text{ the probability to be in the cluster a} \\ \boldsymbol{\mu}_a & : \text{ the center of the cluster a} \\ \boldsymbol{\sigma}_a & : \text{ the variance of the cluster a} \end{cases}$$

For simplicity we considered diagonal covariance matrix.

UL: Mixture models

The idea is that again we will have to learn the parameters of the model using the Bayes theorem

$$\mathrm{p}(\{\boldsymbol{\theta}\}|\boldsymbol{x}) \propto p(\boldsymbol{x}|\{\boldsymbol{\theta}\})p(\boldsymbol{\theta})$$

The gradient with respect to the likelihood or the posterior distribution is not as simple as before. Therefore the following algorithm is used, called Expectation-Maximization, to maximize locally the likelihood (for more details, see the lecture of Ludovic Goudenège)

Step 1:

$$p(\boldsymbol{x}|\mathcal{C}_a, \boldsymbol{\theta}_a) = \prod_i \left[\sqrt{\frac{1}{(2\pi\sigma_{ia})^2}} \exp\left(-\frac{(x_i-\mu_{ia})^2}{2\sigma_{ia}^2}\right)\right]$$

$$r_a^{(m)} = \frac{\rho_a p(\boldsymbol{x}^{(m)}|\mathcal{C}_a,\boldsymbol{\theta}_a)}{\sum_b p(\boldsymbol{x}^{(m)}|\mathcal{C}_b,\boldsymbol{\theta}_b)} \text{ where we have that } \sum_a r_a^{(m)} = 1$$

Step 2:

$$\begin{cases} R_a & = \sum_{(m)} r_a^{(m)} \\ \boldsymbol{\mu}_a^{(t+1)} & = \sum_m \frac{r_a^{(m)}}{R_a} \boldsymbol{x}^{(m)} \\ \boldsymbol{\sigma}_a^{(t+1)} & = \sum_m \frac{r_a^{(m)}}{NR_a} (\boldsymbol{x}^{(m)} - \boldsymbol{\mu}_a)^2 \\ \rho_a^{(t+1)} & = \frac{R_a}{\sum_b R_b} \end{cases}$$

UL: Mixture models

The idea is that again we will have to learn the parameters of the model using the Bayes theorem

$$\mathrm{p}(\{\boldsymbol{\theta}\}|\boldsymbol{x}) \propto p(\boldsymbol{x}|\{\boldsymbol{\theta}\})p(\boldsymbol{\theta})$$

The gradient with respect to the likelihood or the posterior distribution is not as simple as before. Therefore the following algorithm is used, called Expectation-Maximization, to maximize locally the likelihood (for more details, see the lecture of Ludovic Goudenège)

Step 1:

$$p(\boldsymbol{x}|\mathcal{C}_a, \boldsymbol{\theta}_a) = \prod_i \left[\sqrt{\frac{1}{2\pi\sigma_{ia}^2}} \exp\left(-\frac{(x_i-\mu_{ia})^2}{2\sigma_{ia}^2}\right)\right]$$

$$r_a^{(m)} = \frac{\rho_a p(\boldsymbol{x}^{(m)}|\mathcal{C}_a,\boldsymbol{\theta}_a)}{\sum_b p(\boldsymbol{x}^{(m)}|\mathcal{C}_b,\boldsymbol{\theta}_b)} \text{ where we have that } \sum_a r_a^{(m)} = 1$$

Step 2:

$$\begin{cases} R_a & = \sum_m r_a^{(m)} \\ \boldsymbol{\mu}_a^{(t+1)} & = \sum_m \frac{r_a^{(m)}}{R_a}\boldsymbol{x}^{(m)} \\ \boldsymbol{\sigma}_a^{(t+1)} & = \sum_m \frac{r_a^{(m)}}{NR_a}(\boldsymbol{x}^{(m)} - \boldsymbol{\mu}_a)^2 \\ \rho_a^{(t+1)} & = \frac{R_a}{\sum_b R_b} \end{cases}$$

UL: Mixture models

The mixture model is quite flexible and very easy to use.

Cons:

- we still have to fix the number of clusters
- the strong dependence on the initial conditions

Let's see one interesting phenomena for practical use of the GMM, and an example of application using a prior on the centers position

UL: Mixture models

The partitioning process depends on the initial conditions and on the number of clusters
We can design a process that gradually split the system into more and more clusters as a function some control parameters.
To do that, we need to consider the case where the variance is <u>kept fixed</u> and the same in all direction – <u>spherical symmetry</u>:

$$p(\boldsymbol{x}|\mathcal{C}_a, \boldsymbol{\theta}_a) = \prod_i \left[\sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{(x_i-\mu_{ia})^2}{2\sigma^2}\right)\right]$$

From this, we can check that when using value of the variance parameter that is much bigger than the variance of the whole dataset, all clusters should center at the barycenters, independently of the number of clusters. That is we can check that it correspond to a fix-point of the EM equations:

$$\sum_m \frac{(\boldsymbol{x}^{(m)}-\boldsymbol{\mu}_a)\exp(-\beta||\boldsymbol{x}^{(m)-\boldsymbol{\mu}_a}||^2)}{\sum_b \exp(-\beta||\boldsymbol{x}^{(m)-\boldsymbol{\mu}_b}||^2)} = 0$$

Rose, 98
Bonnaire et al., 2021

UL: Mixture models

In the limit $\beta \to 0$ **we obtain, and taking the center to be a small perturbation of the center of mass** $\boldsymbol{\mu}_a \approx 0 + \boldsymbol{\epsilon}_a$

1) Check that $\boldsymbol{\mu}_a = 0$ **is a solution**

$$\sum_m \frac{(\boldsymbol{x}^{(m)} \exp(-\beta||\boldsymbol{x}^{(m)}||^2)}{\sum_b \exp(-\beta||\boldsymbol{x}^{(m)}||^2)} = \sum_m \boldsymbol{x}^{(m)}/K = 0$$

2) Look for its stability by linear perturbation

$$\boldsymbol{\epsilon}_a^{(t+1)} \approx 2\beta C \boldsymbol{\epsilon}_a^{(t)} \text{ where } C = \sum_m \boldsymbol{x}^{(m)} (\boldsymbol{x}^{(m)})^T$$

We see that a perturbation will be stable until

$$\boxed{\beta_c = \frac{1}{2\lambda_{\max}}}$$

$\lambda_{\max}$ is the highest eigenvalue of C

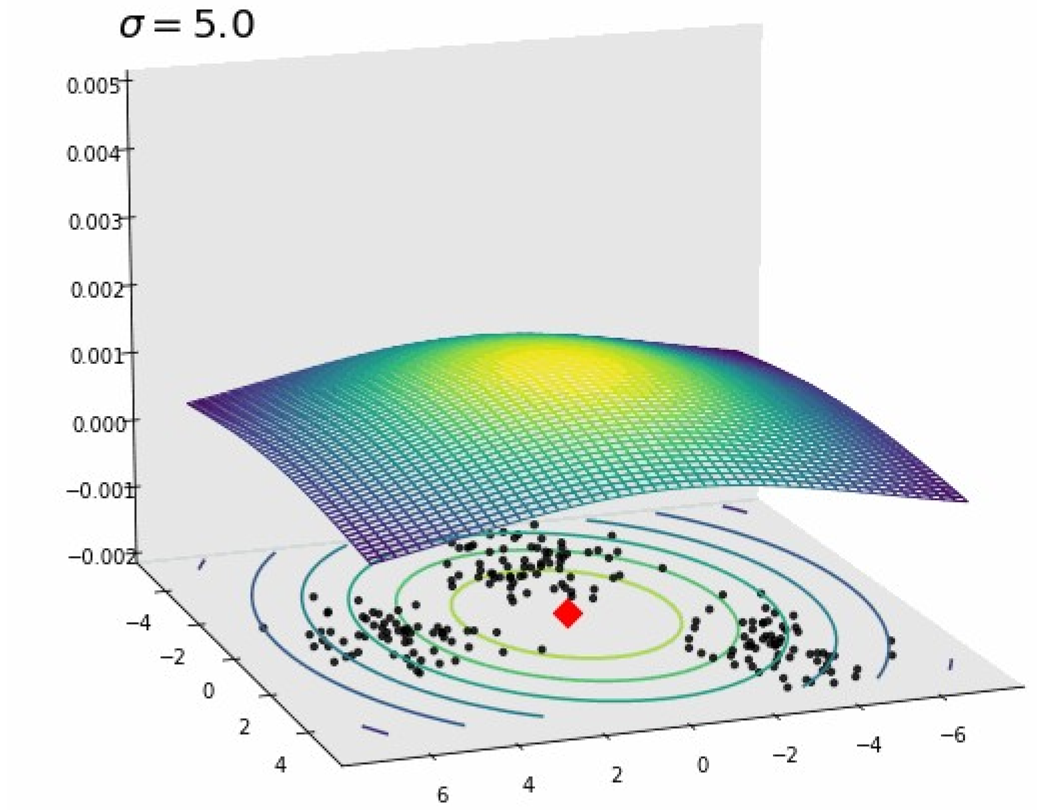
UL: Mixture models

What does it mean ?
1) starting the clustering process with many centers and low variance → all centers converge to the center of mass
2) decreasing the variance up to highest eigenvalue of C will start splitting the centers into two part, in the direction of the corresponding eigenvectors.

This is true, but it is clear that when having separated cluster, this will also be true for each separate cluster

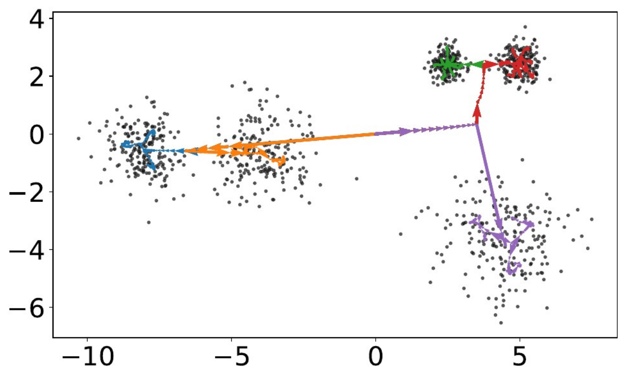
UL: Mixture models

Very simple toy example:

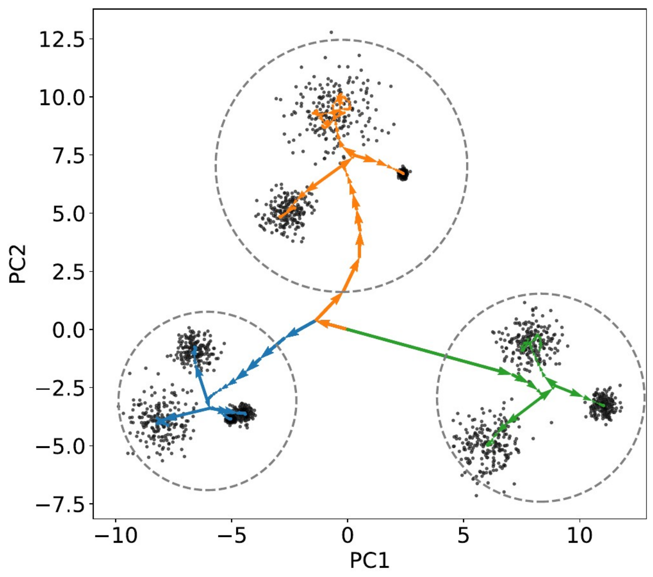


UL: Mixture models

Examples – toy datasets

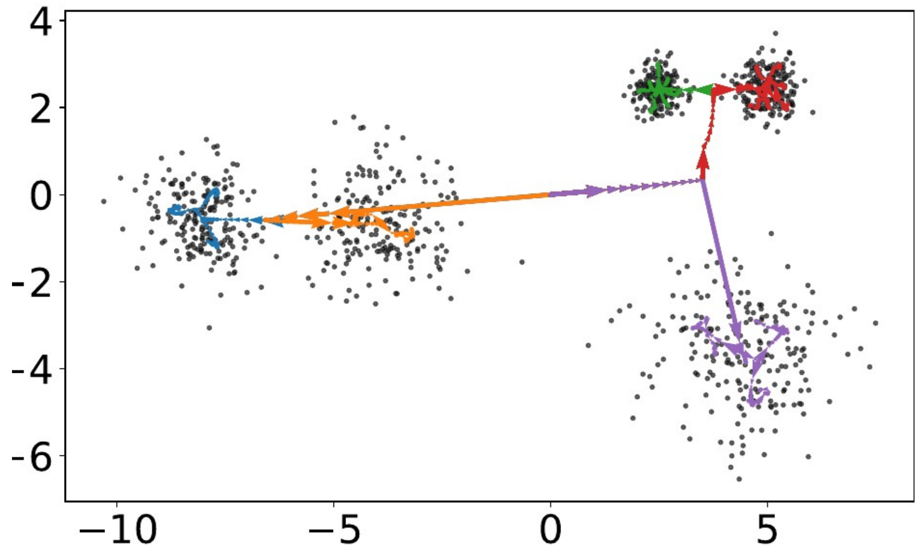


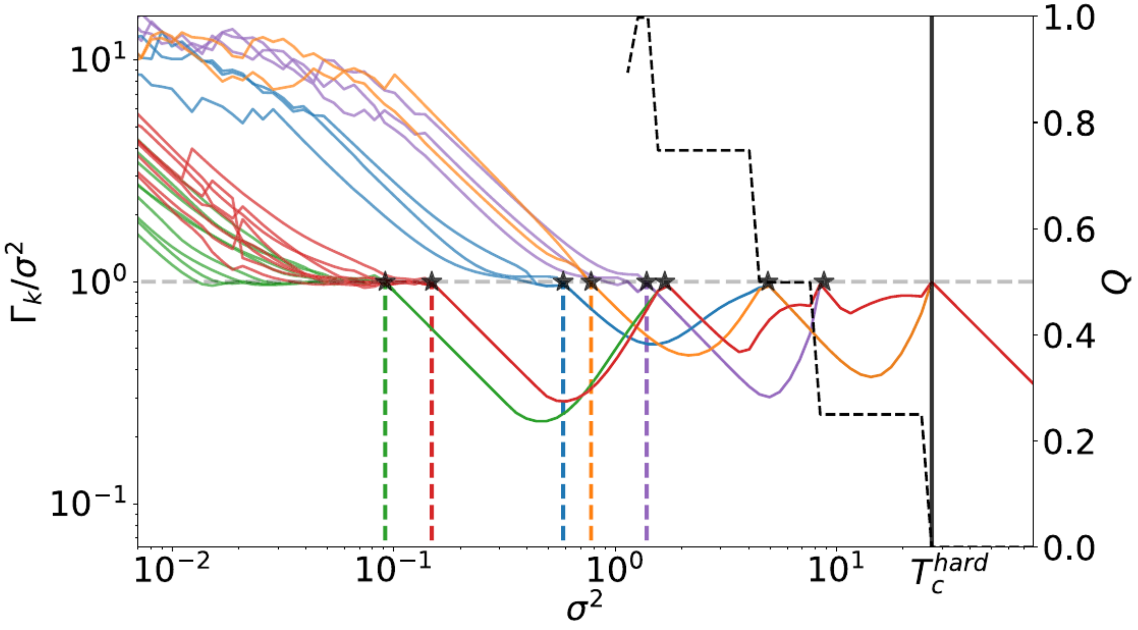
5D Gaussian clusters



UL: Mixture models

Examples – toy datasets : (ask Tony for a gif !)

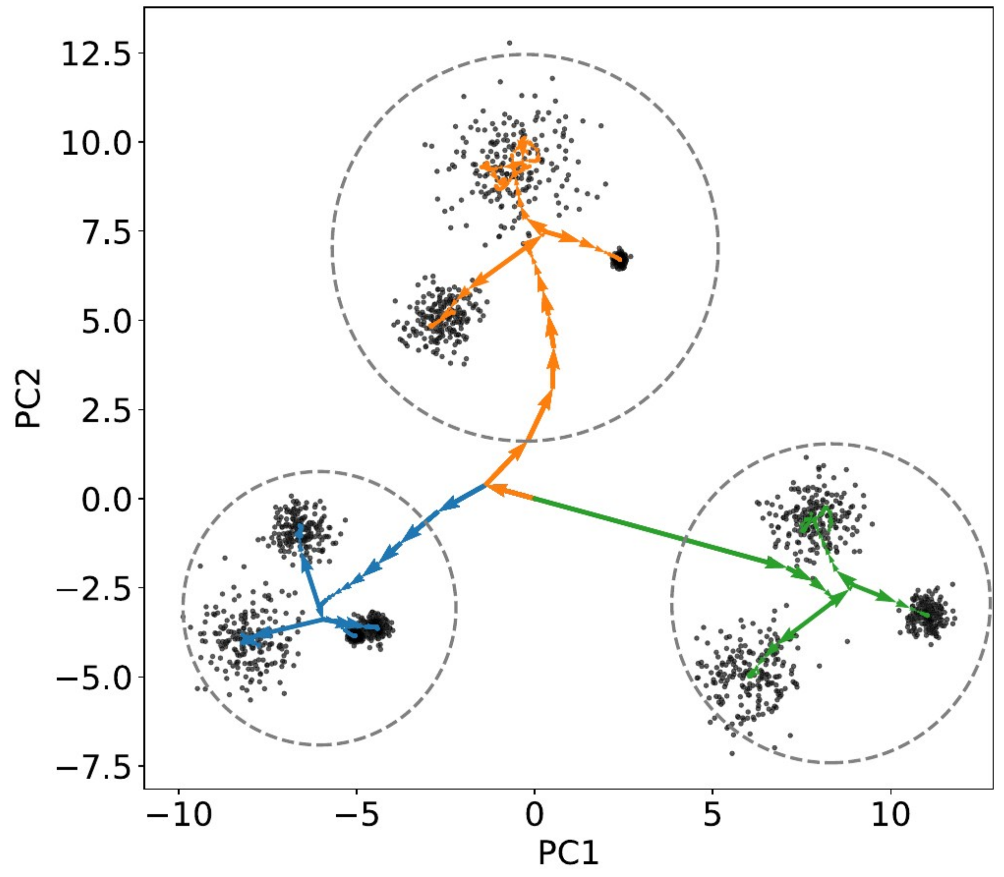


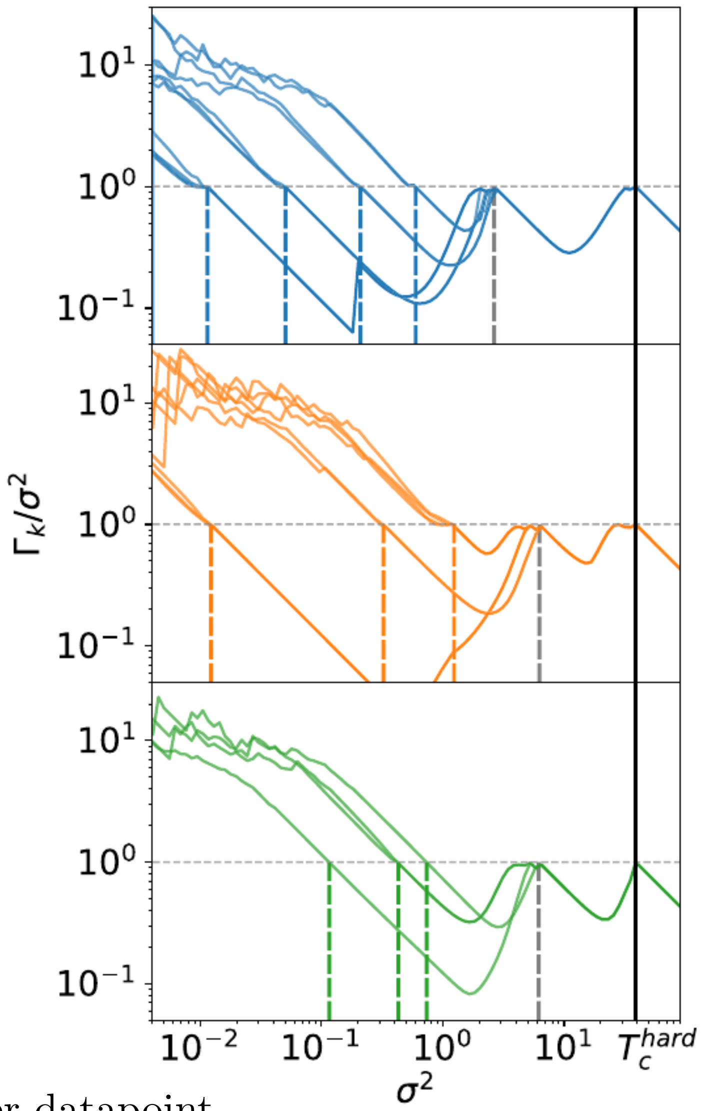


Γ_k : variance measure in the intra-cluster datapoint

UL: Mixture models

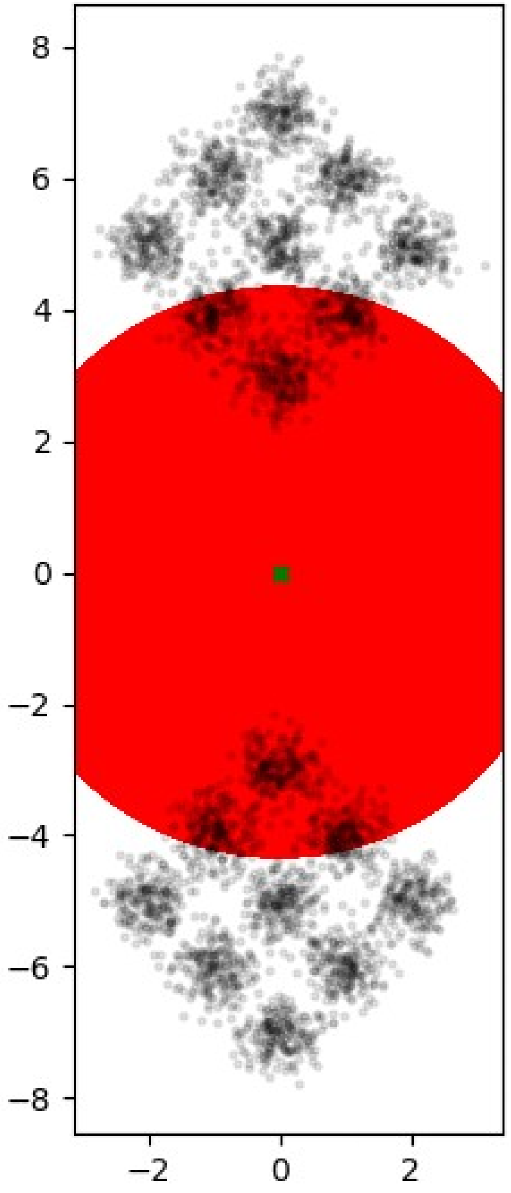
Examples – toy datasets



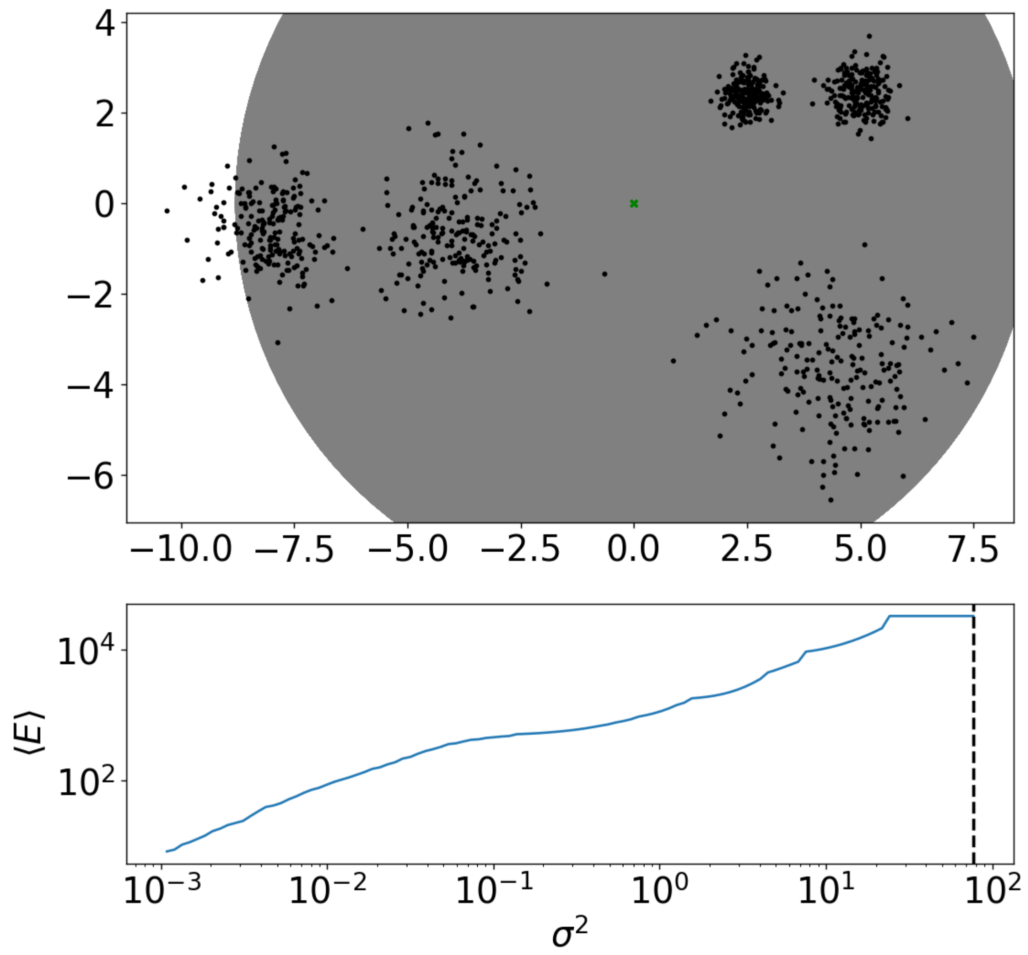


Γ_k : variance measure in the intra-cluster datapoint

UL: Mixture models



UL: Mixture models



UL: Mixture models

We can imagine many refinement over this setting:

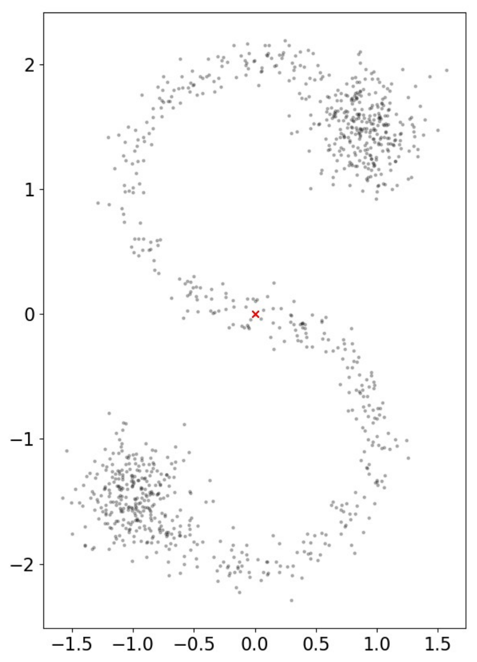
→ using a soft constraint instead of a hard-spherical one by using the conjugate prior, the inverse-gamma distribution

$$\ln p(\sigma_a^2) = -\lambda_\sigma \left[\ln \sigma_a^2 + \frac{\sigma}{\sigma_a}\right]$$

→ using a graph-prior on the center's position to infer the principal graph of some dataset

$$\ln p(\{\boldsymbol{\mu}\}) = -\frac{\mu}{2} \sum_{a,b} A_{ab} ||\boldsymbol{\mu}_a - \boldsymbol{\mu}_b||^2$$

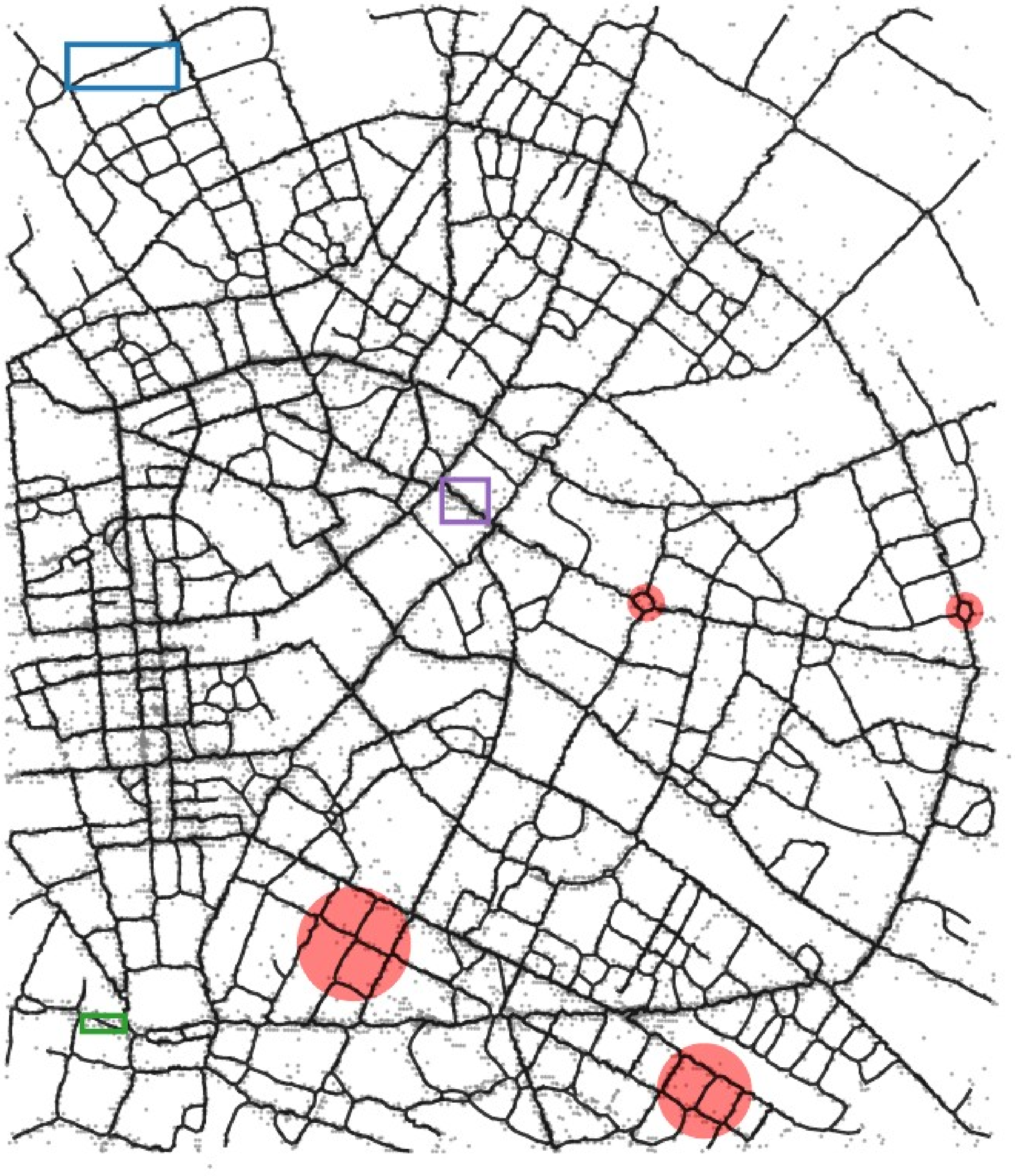
where A is some adjacency matrix



UL: Mixture models

Some applications:

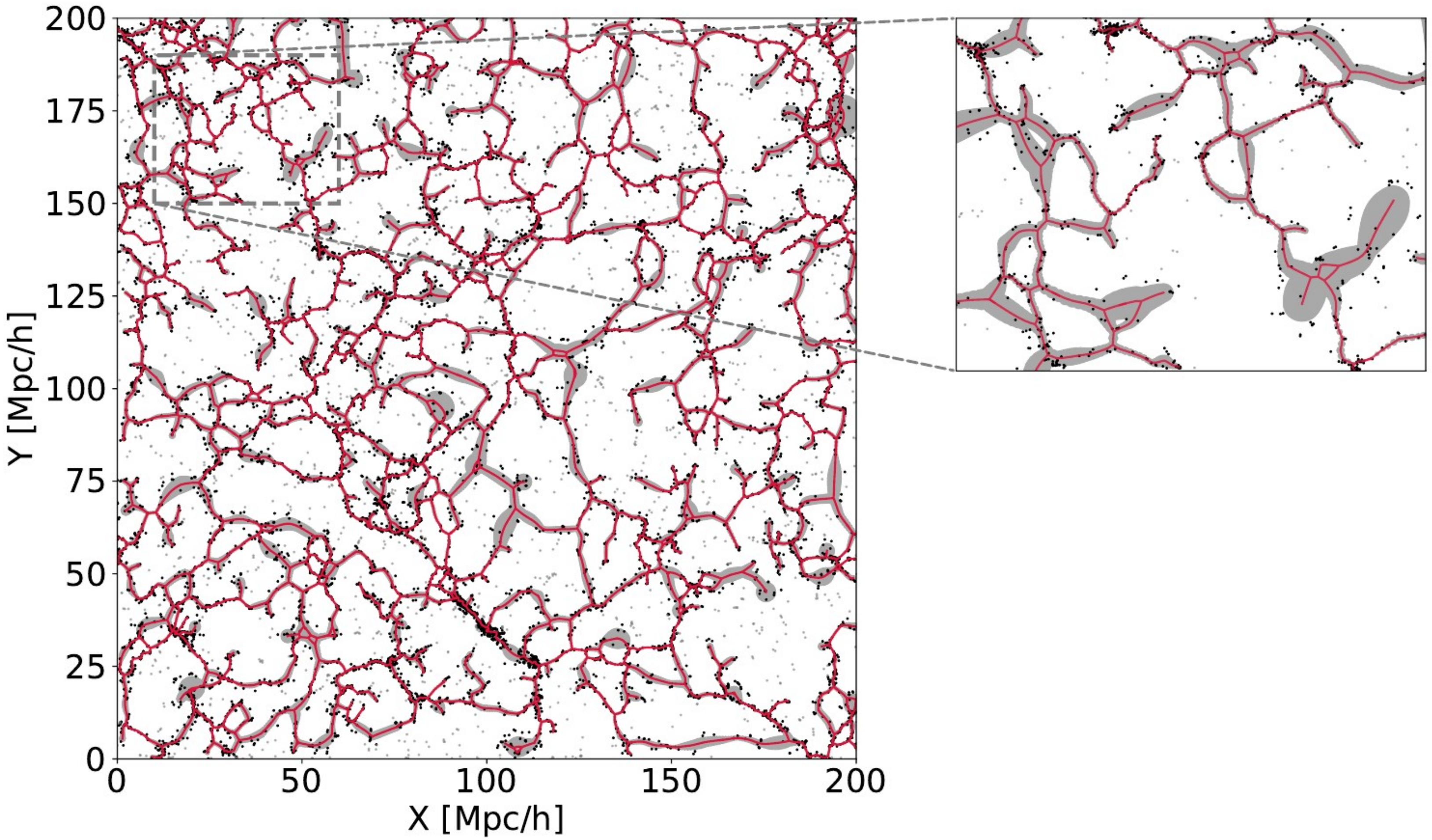
1) the GPS signals on Berlin



UL: Mixture models

Some applications:

2) Density of dark matter in the universe at very large scale



UL: Take-Away message

Clustering method such as GMM can be applied in various situation.

Pros:

- they are quite simple to do

- they are fast

- the Bayesian formalist allow for interesting refinemen

Cons:

- still hard to evaluate the partitioning

- need a notion of distance

The DBSCAN can be a good alternative for density-like cluster !

UnSupervised Learning

In this final part, we turn on generative model. We will consider a simple shallow neural network capable of reproducing complex dataset.

A generative model's task is to, given a dataset, be able to reproduce statistically similar data, but not identical to the dataset
Among the famous examples:
- Generative Adversarial Network → https://this-person-does-not-exist.com/en
- ChatGPT

UL: Generative Model

We will consider a statistical model that is very close to the “classical” Ising model.

Let’s start with the Ising inverse case and show how we can improve over it.

Let’s assume that you have a set of samples that are either

→ continuous $\boldsymbol{X} \in \mathbb{R}^{N \times M}$

→ discrete $\boldsymbol{X} \in \{\pm 1\}^{N \times M}$

Can we define a probability distribution that reproduce the statistical properties of the dataset ?

UL: Generative Model

Let's assume that you have a set of samples that are eigher

- **continuous** $\boldsymbol{X} \in \mathbb{R}^{N\times M}$
- **discrete** $\boldsymbol{X} \in \{\pm 1\}^{N\times M}$

Can we define a probability distribution that reproduce the statistical properties of the dataset ?

The most simple ones, that include correlations, are given by

$$p(\boldsymbol{x}) \propto \exp\left(-\mathcal{H}\right) = \exp\left(-\tfrac{1}{2}\textstyle\sum_{i,j} J_{ij} x_i x_j - \sum_i b_i x_i\right)$$

When the variables are continuous, we have a direct expression for the parameters

- $J_{ij} = C_{ij}^{-1} \text{ with } C_{ij} = \dfrac{1}{M}\displaystyle\sum_m x_i^{(m)} x_j^{(m)}$

  $h_i = m_i \text{ with } m_i = \dfrac{1}{M}\displaystyle\sum_m x_i^{(m)}$

For discrete variables, there is not explicit solution *in general*. Yet we can maximize the likelihood

$$\frac{\partial \mathcal{L}}{\partial J_{ij}} = C_{ij} - \langle x_i x_j \rangle_p$$

$$\frac{\partial \mathcal{L}}{\partial h_i} = m_i - \langle x_i \rangle_p$$

$\text{where} \quad \langle f(\boldsymbol{x})\rangle_p = \sum_{\boldsymbol{x}} f(\boldsymbol{x}) p(\boldsymbol{x})$

UL: Generative Model

Problem: we can show that this kind of generative model can only adjust for the first two moment of the empirical distribution of the dataset.
We will be able to reproduce perfectly the
- mean <x_i>
- covariance matrix <$x_i x_j$>
But not higher-order statistics.

A "hidden-variable" approach, has been introduced ~ 30 years ago that can handle this problem.
→ Hinton, Sejnowski 85 and Smolensky 86' introduced the Restricted Boltzmann Machine.

UL: Generative Model

<u>The Restricted Boltzmann Machine</u> : a competitive shallow-neural network

We define the "energy" function of the probability distribution as

$$\mathcal{H}[\boldsymbol{x}, \boldsymbol{\tau}; \boldsymbol{w}, \boldsymbol{\eta}, \boldsymbol{\theta}] = -\sum_{ia} x_i w_{ia} \tau_a - \sum_i \eta_i x_i - \sum_a \theta_a \tau_a$$

There is two types of variables :

→ x **, that correspond to the degree of liberties,**

→ τ **, that encode latent features of the distribution.**

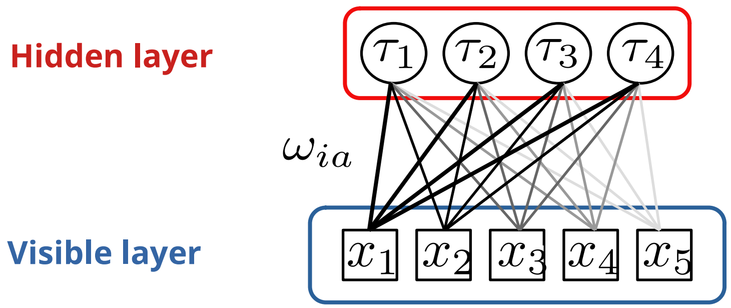
We can then then define the probability distribution over the input variable's as

$$p(\boldsymbol{x}) = \frac{1}{Z} \sum_{\{\boldsymbol{\tau}\}} \exp(-H[\boldsymbol{x}, \boldsymbol{\tau}; \boldsymbol{w}, \boldsymbol{\eta}, \boldsymbol{\theta}])$$

UL: Generative Model

The Restricted Boltzmann Machine : a competitive shallow-neural network

Topology of the system:



We can then then define the probability distribution over the input variable's as

$$p(\boldsymbol{x}) = \frac{1}{Z}\sum_{\{\boldsymbol{\tau}\}} \exp(-H[\boldsymbol{x},\boldsymbol{\tau};\boldsymbol{w},\boldsymbol{\eta},\boldsymbol{\theta}])$$

$$\mathcal{H}[\boldsymbol{x},\boldsymbol{\tau};\boldsymbol{w},\boldsymbol{\eta},\boldsymbol{\theta}] = -\sum_{ia} x_i w_{ia}\tau_a - \sum_i \eta_i x_i - \sum_a \theta_a \tau_a$$

As in the previous case for discrete variables, we can learn the parameters of the model by maximizing the likelihood:

$$\hat{\boldsymbol{\theta}} = \text{argmax}_{\boldsymbol{w},\boldsymbol{\eta},\boldsymbol{\theta}} \left\{\prod_m p(\boldsymbol{x}^{(m)})\right\}$$

UL: Generative Model

The Restricted Boltzmann Machine : a competitive shallow-neural network

The gradient take a similar form:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_{ia}} &= \frac{1}{M}\sum_m \langle x_i^{(m)} \tau_a \rangle_{p(\tau_a|\boldsymbol{x}^{(m)})} - \langle x_i \tau_a \rangle_p \\ \frac{\partial \mathcal{L}}{\partial \eta_i} &= m_i - \langle x_i \rangle_p \\ \frac{\partial \mathcal{L}}{\partial \theta_a} &= \frac{1}{M}\sum_m \langle \tau_a \rangle_{p(\tau_a|\boldsymbol{x}^{(m)})} - \langle \tau_a \rangle_p\end{aligned}$$

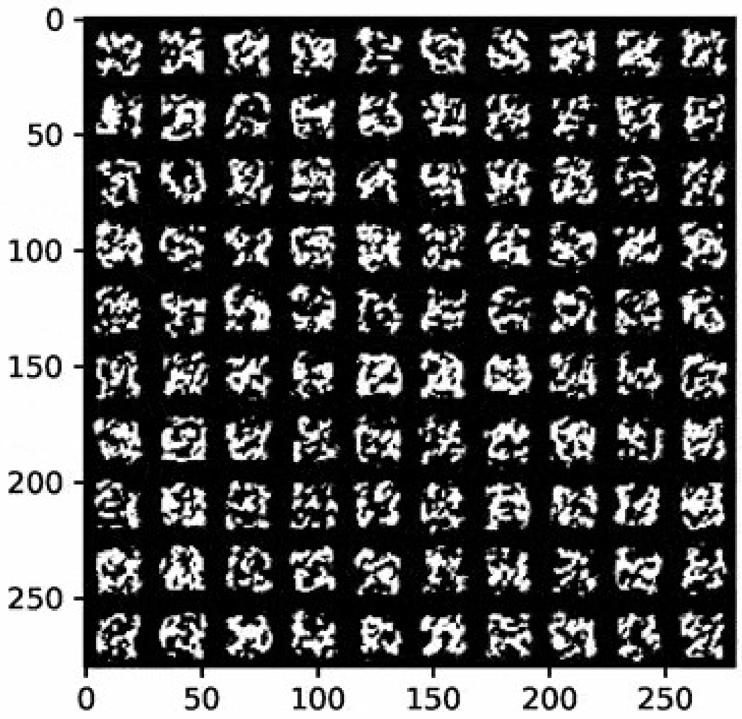
There is still a computational difficulty:

→ the first can be computed directly

→ the second one typically depends on Monte Carlo estimate which can be very slow

UL: Generative Model

Toy example (yet real dataset): MNIST



28x28 pixels

10.000 samples

UL: Generative Model

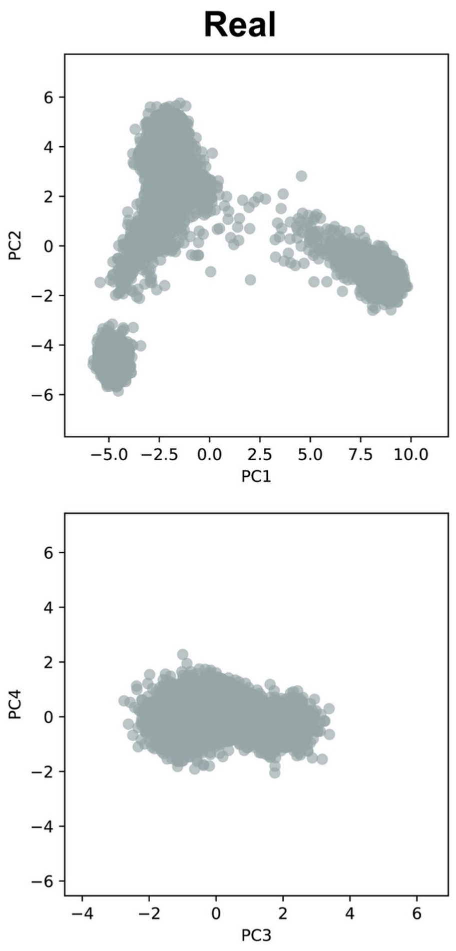
Can be applied to various type of dataset. Example on population genetics

Human Genome Project:

- subsamples of one chromosome (800 pairs)
- 2500 individuals (~5000 samples)
- 0 or 1, if mutation or not

A typical problem: genome are private!

→ use the RBM to generate fake data



Projection

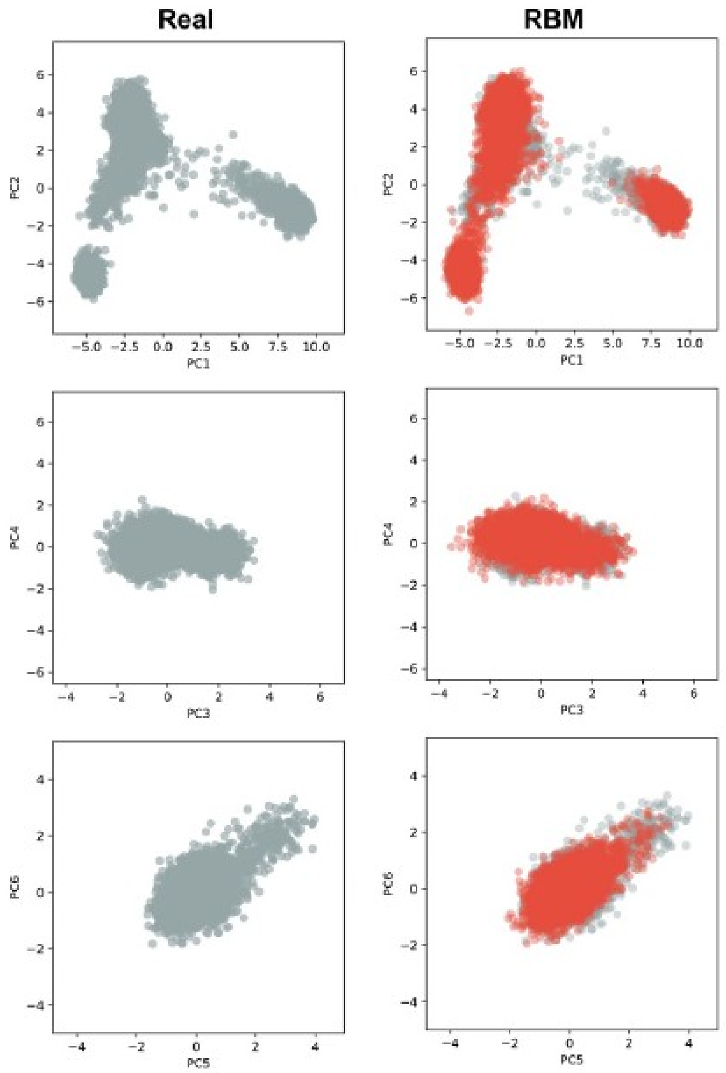
along the PCA

Yelmen et al, 21

UL: Generative Model

Can be applied to various type of dataset. Example on population genetics

Projection on the first directions of the PCA.
→ the RBM reproduce correctly the empirial distribution



UL: Take way message

Simple generative model (such as RBM) can model complex dataset.

Pros

- easy to program,
- not a too-black box, it is possible to look at the learn features
- familiar to some known model (Ising model)

Cons

- the training using MC method can be cumbersome
- large scale dataset are not so easy to handle, notably with the lack of convolutional layer

The end

Questions and remarks ?