



# Machine Learning: A new toolbox for Theoretical Physics

Juan Rojo

VU Amsterdam & Theory group, Nikhef

***D-ITP Advanced Topics in Theoretical Physics***

**02/12/2019**

# Today's lecture

- Unsupervised Learning: clustering
- Ensemble methods and bootstrapping
- Data visualisation and dimensional reduction
- Reinforcement Learning

## *Tutorial 3:*

- (a) *Clustering with Unsupervised Learning*
- (b) *Bagging and Random Forests for classification in the Ising Model*

# **Dimensional Reduction & Data Visualisation**

# Dimensional reduction

Efficient **data visualisation techniques** are essential to construct better models in ML applications eg by **identifying correlated, redundant, or irrelevant features**

Traditional data visualisation methods are not practical when the datasets involve a large number of features (such as images) and we need to project the data onto a lower-dimensional space, called the **latent space**, using **dimensional reduction**

*A good data visualisation strategy is very useful to identify the most suitable strategy to approach a ML problem*

# Dimensional reduction

Efficient **data visualisation techniques** are essential to construct better models in ML applications eg by **identifying correlated, redundant, or irrelevant features**

Traditional data visualisation methods are not practical when the datasets involve a large number of features (such as images) and we need to project the data onto a lower-dimensional space, called the **latent space**, using **dimensional reduction**

This is easier said than done: many pitfalls associated to **high-dimensionality datasets**

*``the curse of dimensionality''*

# Dimensional reduction

Efficient **data visualisation techniques** are essential to construct better models in ML applications eg by **identifying correlated, redundant, or irrelevant features**

Traditional data visualisation methods are not practical when the datasets involve a large number of features (such as images) and we need to project the data onto a lower-dimensional space, called the **latent space**, using **dimensional reduction**

This is easier said than done: many pitfalls associated to **high-dimensionality datasets**

💡 *High-dimensional data lives near the edge of the sample space*

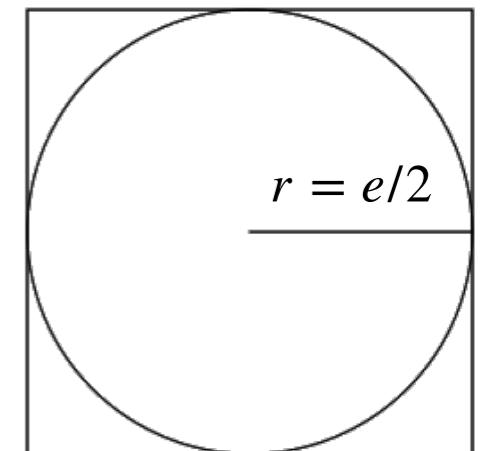
consider data distributed at random in a  $D$ -dimensional hypercube  $C = [-e/2, e/2]^D$

consider a  $D$ -dimensional sphere  $S$  of radius  $e/2$  centered at origin

probability that random point from  $C$  is sampled inside the sphere  $S$  is

$$P(x_i \in S) \simeq \frac{\pi^{D/2} (e/2)^D / \Gamma(D/2 + 1)}{e^D} = \simeq \frac{\pi^{D/2}}{2^D D^D} \rightarrow 0$$

so most of the data lies close to the hypercube **edge!**



# Dimensional reduction

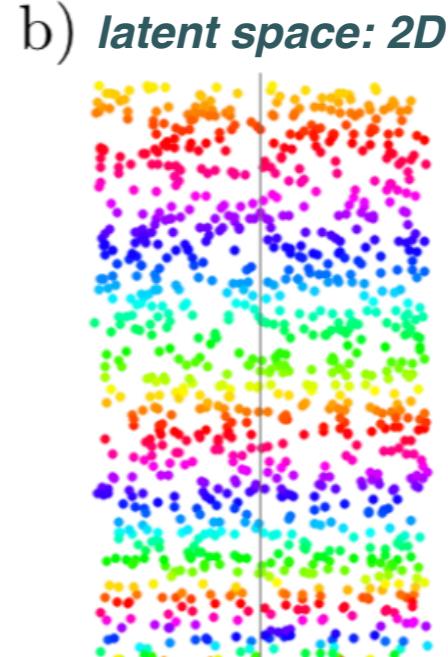
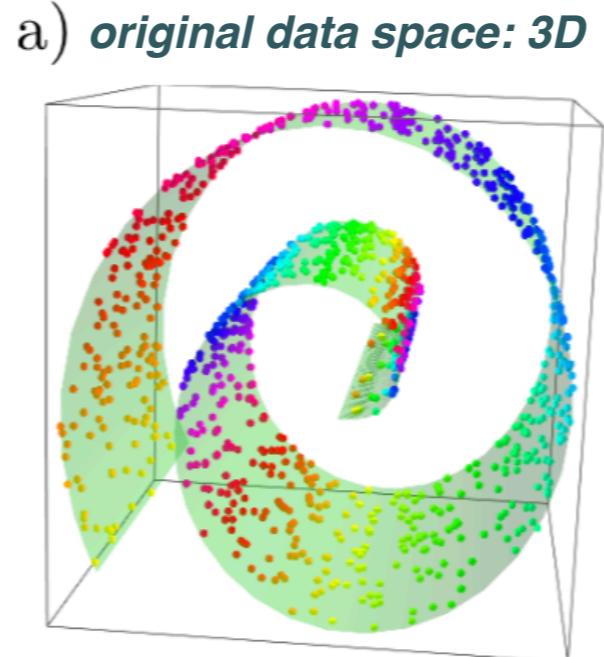
Efficient **data visualisation techniques** are essential to construct better models in ML applications eg by **identifying correlated, redundant, or irrelevant features**

Traditional data visualisation methods are not practical when the datasets involve a large number of features (such as images) and we need to project the data onto a lower-dimensional space, called the **latent space**, using **dimensional reduction**

This is easier said than done: many pitfalls associated to high-dimensionality datasets

- *High-dimensional data lives near the edge of the sample space*
- *We need to conserve information on original pair-wise distances or similarities when transforming to the latent space*

the minimum number of parameters needed to capture the original patterns is the **intrinsic dimensionality of the data**



# Dimensional reduction

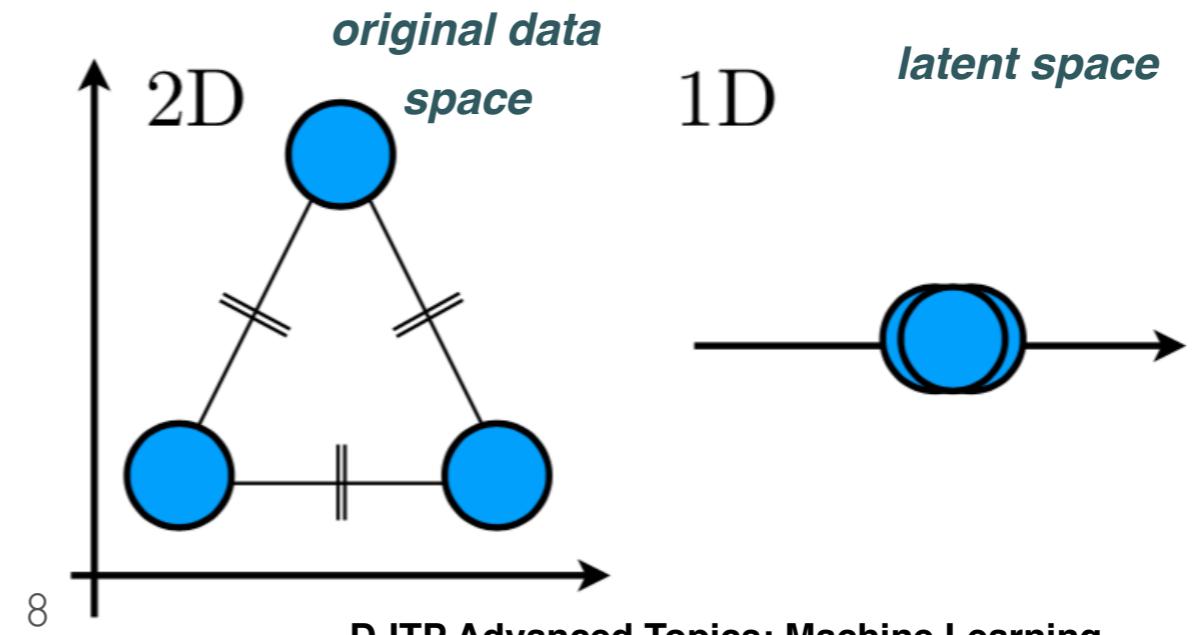
Efficient **data visualisation techniques** are essential to construct better models in ML applications eg by identifying correlated, redundant, or irrelevant features

Traditional data visualisation methods are not practical when the datasets involve a large number of features (such as images) and we need to project the data onto a lower-dimensional space, called the **latent space**, using **dimensional reduction**

This is easier said than done: many pitfalls associated to high-dimensionality datasets

- 📌 *High-dimensional data lives near the edge of the sample space*
- 📌 *We need to conserve information on original pair-wise distances or similarities when transforming to the latent space*
- 📌 *Dimensional reduction cannot be such to destroy info on original patterns in the data*

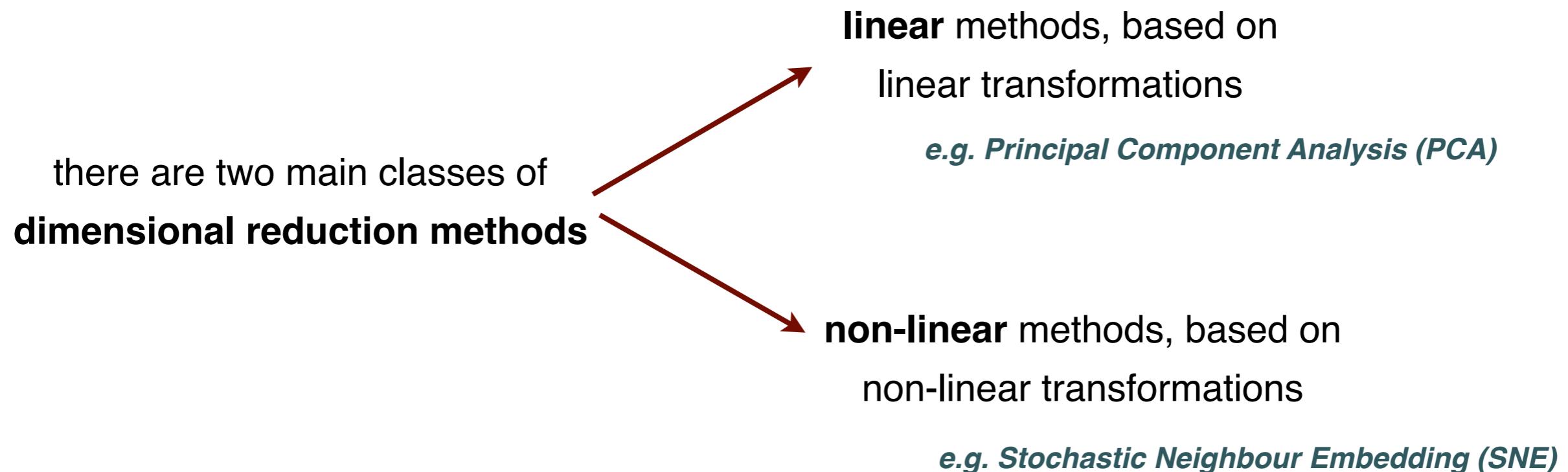
*“the crowding problem”*



# Dimensional reduction

Efficient **data visualisation techniques** are essential to construct better models in ML applications eg by identifying correlated, redundant, or irrelevant features

Traditional data visualisation methods are not practical when the datasets involve a large number of features (such as images) and we need to project the data onto a lower-dimensional space, called the **latent space**, using **dimensional reduction**



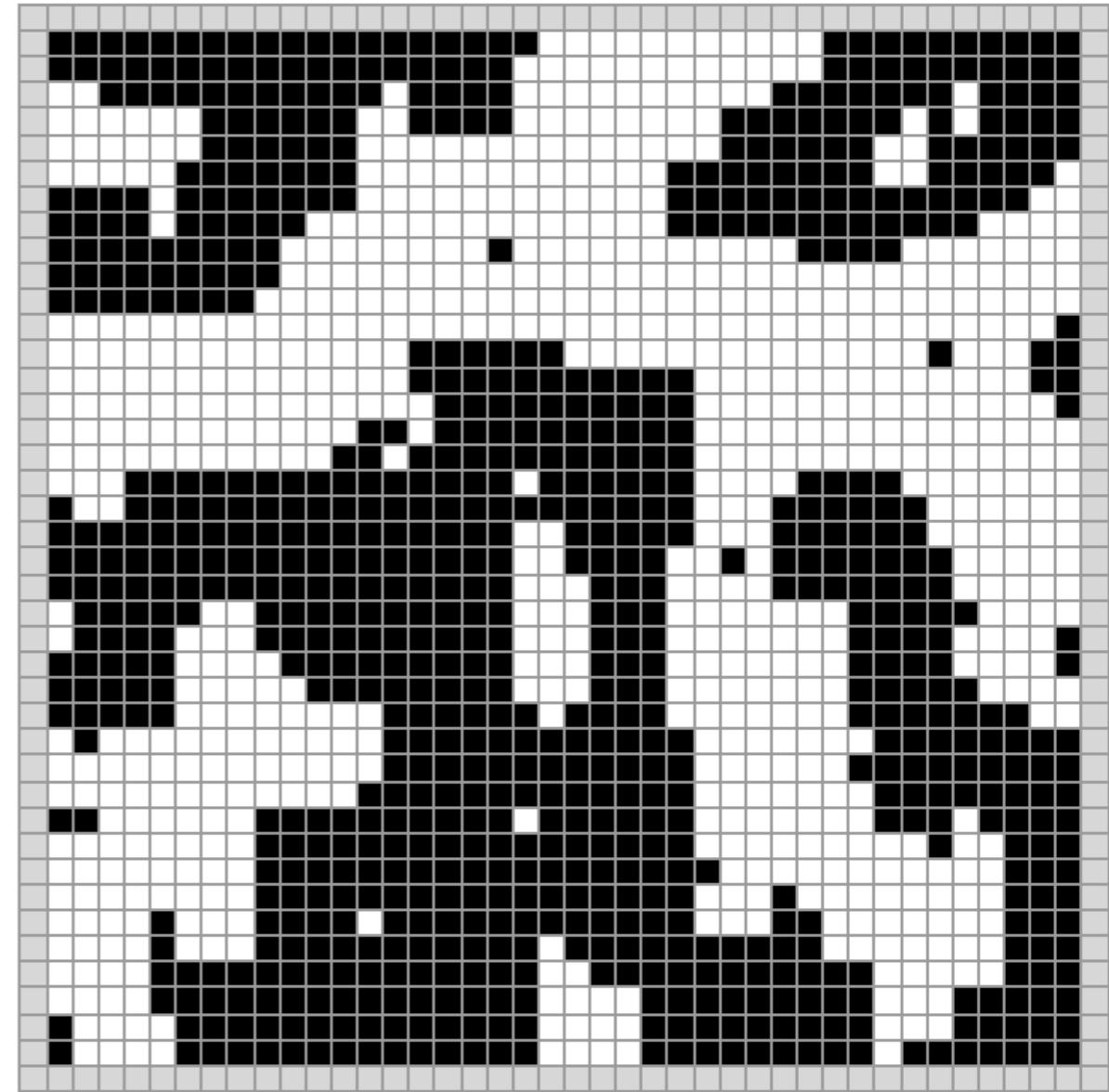
# Principal Component Analysis

PCA projections often capture the **large-scale structure** of high-dimensional datasets  
consider for example the **Ising Model in 2D** with 40 spins: 1600-dimensional space  
can we **measure ``order''** with few parameters?

*disordered (random) phase*



*ordered phase*



<https://demonstrations.wolfram.com/The2DIzingModelMonteCarloSimulationUsingTheMetropolisAlgorithm/>

# Principal Component Analysis

consider  $n$  data points that live in a  $p$ -dimensional feature space

$$\{\boldsymbol{x}_i\}_{i=1}^n \quad \boldsymbol{x}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,p})$$

assume for simplicity that the mean of these points vanishes

$$\bar{\boldsymbol{x}} = \frac{1}{n} \sum_{i=1}^n \boldsymbol{x}_i = 0$$

Now denote the **design matrix** as

$$\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_n]^T$$

*In a design matrix, each row represents an individual data point and each column one of the data features*

where rows are the data points and columns are features

$$\boldsymbol{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,p} \\ x_{2,1} & x_{2,2} & \dots & x_{2,p} \\ \dots & \dots & \dots & \dots \\ x_{n,1} & x_{n,2} & \dots & x_{n,p} \end{pmatrix} \xrightarrow{\text{data points}}$$

  
**features**

# Principal Component Analysis

the associated (symmetric)  $p$ -dimensional **covariance matrix** is then

$$\Sigma(X) = \frac{1}{n-1} X^T X \quad \rightarrow \quad \Sigma_{lm} = \frac{1}{n-1} \sum_{i=1}^n X_{li} X_{im}$$

*p-dimensional  
in space of features*      *sum over data points*

from where we see that  $\Sigma_{lm}$  measures the correlation between **features  $l$  and  $m$**

The goal of PCA is to rotate this matrix to a **new feature-basis** (different from the one present in the original data) that emphasises high-variability directions. This can be done by a linear transformation that **reduces the covariance between features**

recall the **eigenvalue decomposition** for a square matrix

$$A = Q \Lambda Q^T$$

Diagram illustrating the eigenvalue decomposition of a matrix  $A$ :

- Red arrows point from the text "columns are eigenvectors of  $A$ " to the columns of the matrix  $Q$ .
- A red arrow points from the text "diagonal matrix of eigenvalues" to the diagonal matrix  $\Lambda$ .

# Principal Component Analysis

the associated (symmetric)  $p$ -dimensional **covariance matrix** is then

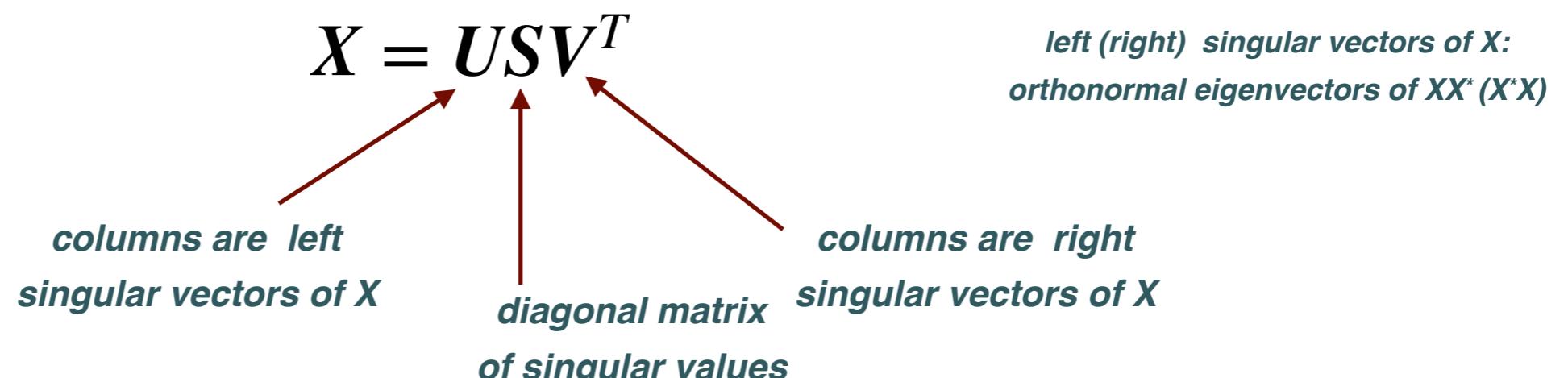
$$\Sigma(X) = \frac{1}{n-1} X^T X \quad \rightarrow \quad \Sigma_{lm} = \frac{1}{n-1} \sum_{i=1}^n X_{li} X_{im}$$

*p-dimensional  
in space of features*      *sum over data points*

from where we see that  $\Sigma_{lm}$  measures the correlation between **features  $l$  and  $m$**

The goal of PCA is to rotate this matrix to a **new feature-basis** (different from the one present in the original data) that emphasises high-variability directions. This can be done by a linear transformation that **reduces the covariance between features**

for this we will use **Singular Value Decomposition** (SVD), a factorisation of a real or complex matrix that generalizes the eigendecomposition of a positive semidefinite matrix



# Principal Component Analysis

Using SVD we can express the data covariance matrix as

$$\Sigma(X) = \frac{1}{n-1} VSU^T USV^T = V \left( \frac{S^2}{n-1} \right) V^T \equiv V \Lambda V^T$$

*U,V are unitary matrices*      *diagonal matrix with eigenvalues in decreasing order along diagonal*

columns of  $V$  → principal directions of  $\Sigma$

at this point we are ready to use PCA to reduce the dimensionality of data from  $p$  to  $p' < p$

$$\widetilde{Y} = X \widetilde{V}_{p'}$$

*reduced dataset: n points with  $p' < p$  features each in the rotated-feature matrix*      *original dataset: n points with  $p$  features each*      *projection matrix: select the  $p'$  largest eigenvectors*

with PCA only the  $p'$  directions with higher variability remain

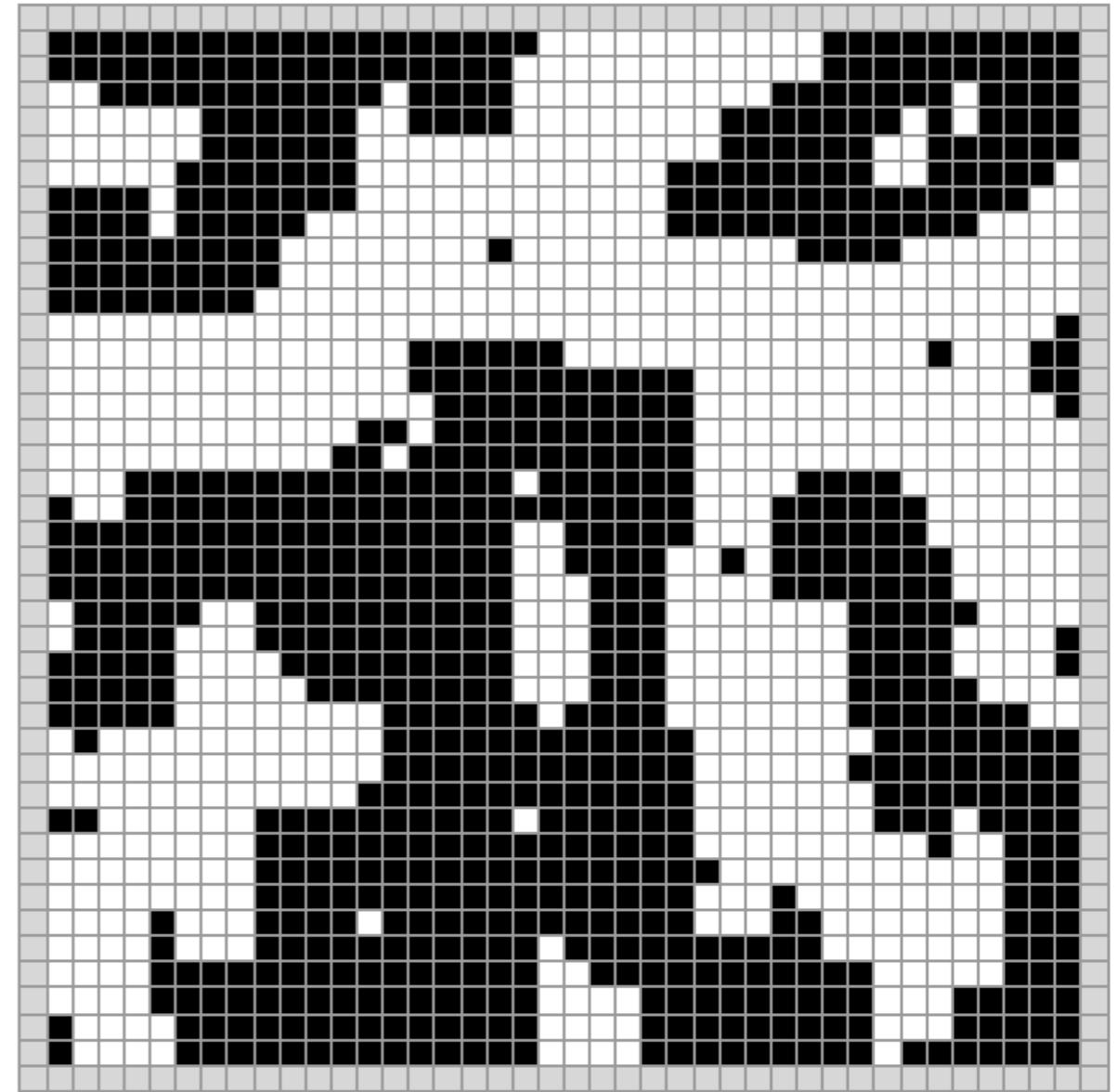
# Principal Component Analysis

PCA projections often capture the **large-scale structure** of high-dimensional datasets  
consider for example the **Ising Model in 2D** with 40 spins: 1600-dimensional space  
can we **measure ``order''** with few parameters?

*disordered (random) phase*



*ordered phase*

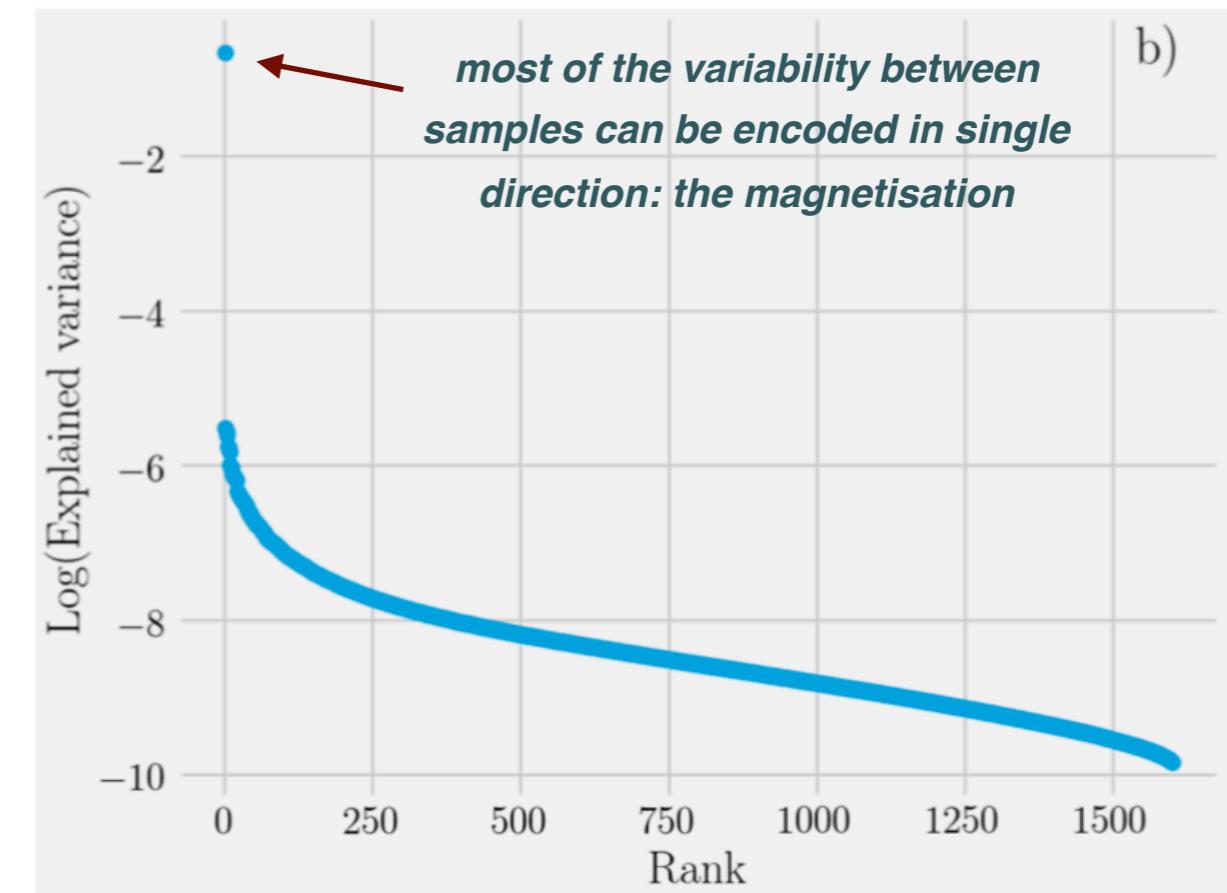
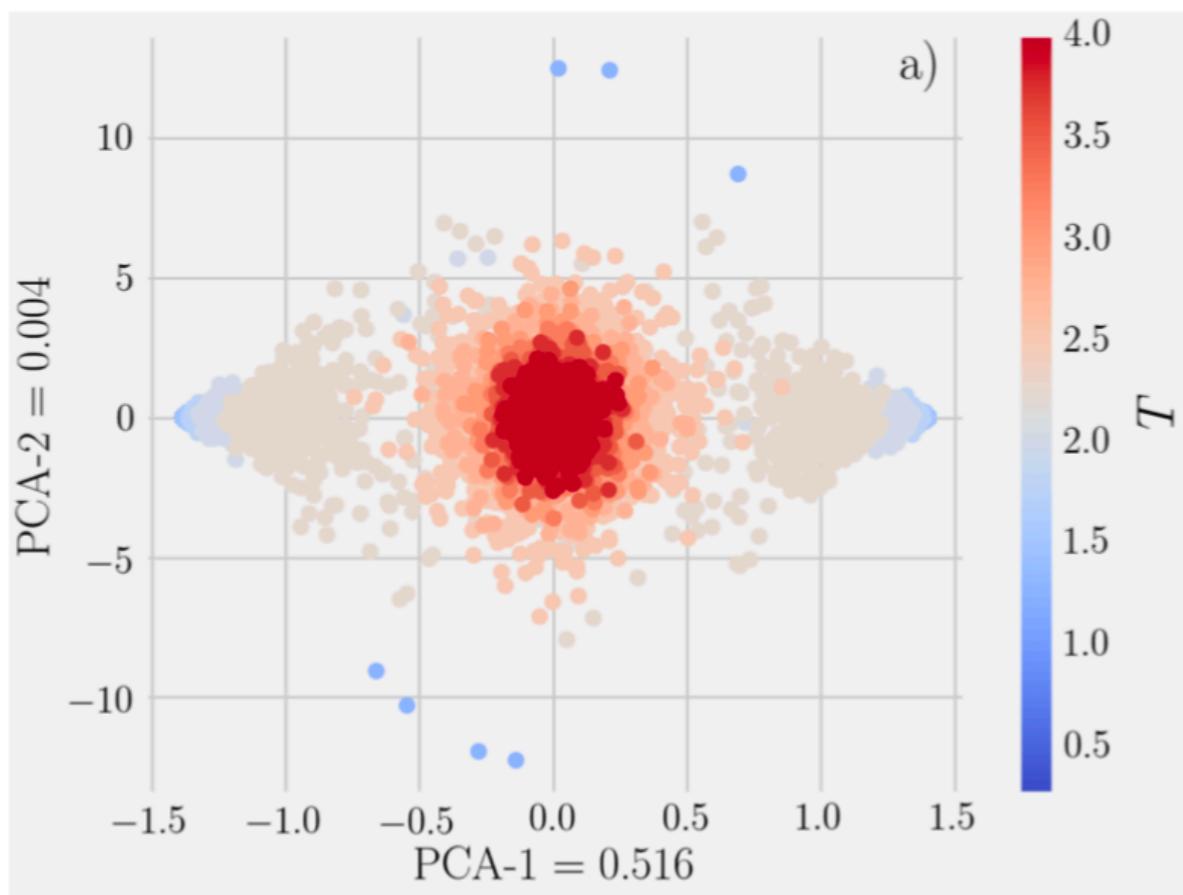


<https://demonstrations.wolfram.com/The2DIzingModelMonteCarloSimulationUsingTheMetropolisAlgorithm/>

# Principal Component Analysis

PCA projections often capture the **large-scale structure** of high-dimensional datasets  
consider for example the **Ising Model in 2D** with 40 spins: 1600-dimensional space  
can we **measure ``order''** with few parameters?

*1000 samples for each  $T$*

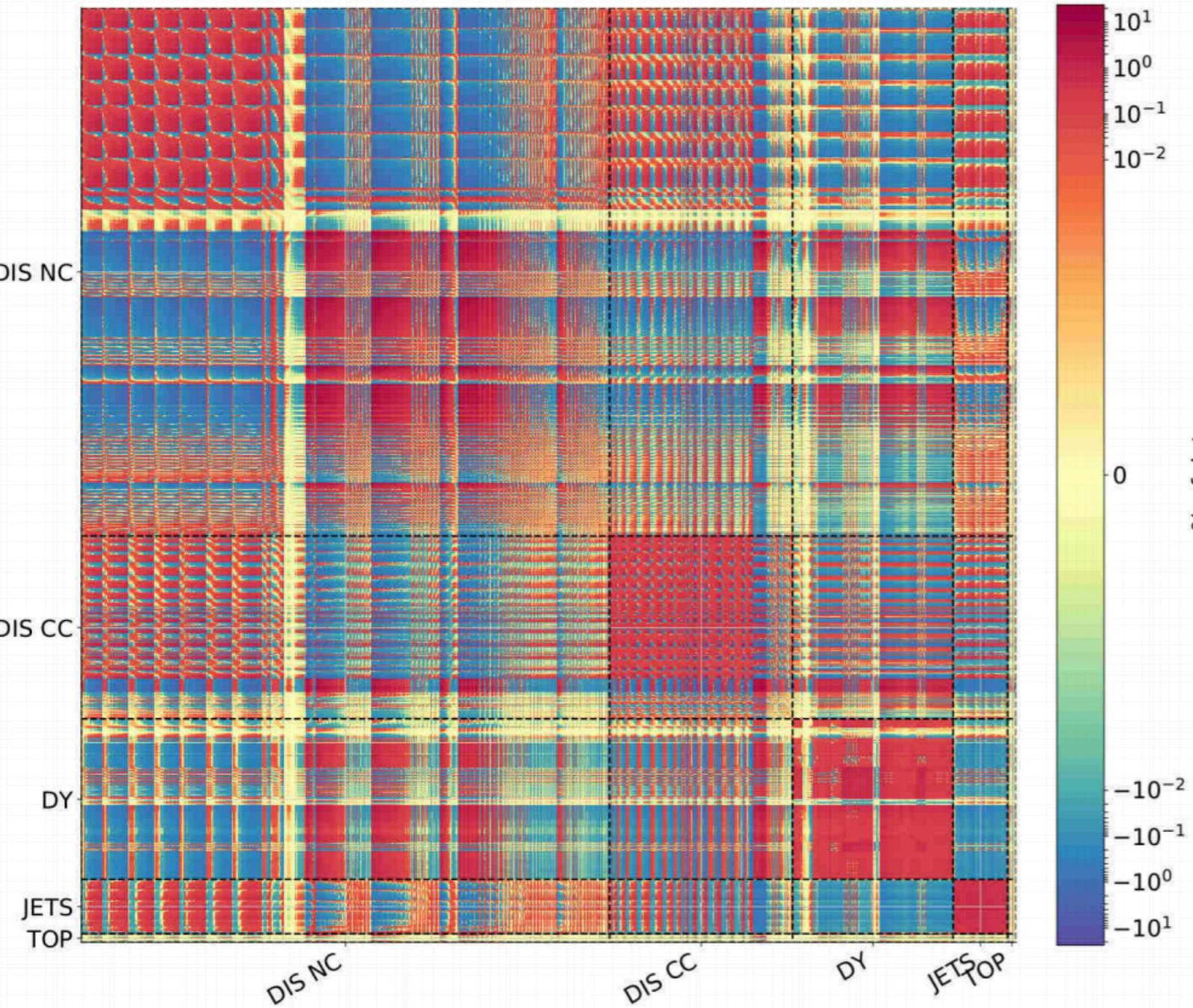


*The first principal component accounts for > 50% of total variability!*

This first PCA component corresponds to the **magnetisation order parameter**,  
which we have thus identified without any prior physical knowledge of the system

# PCA for propagation of theory errors

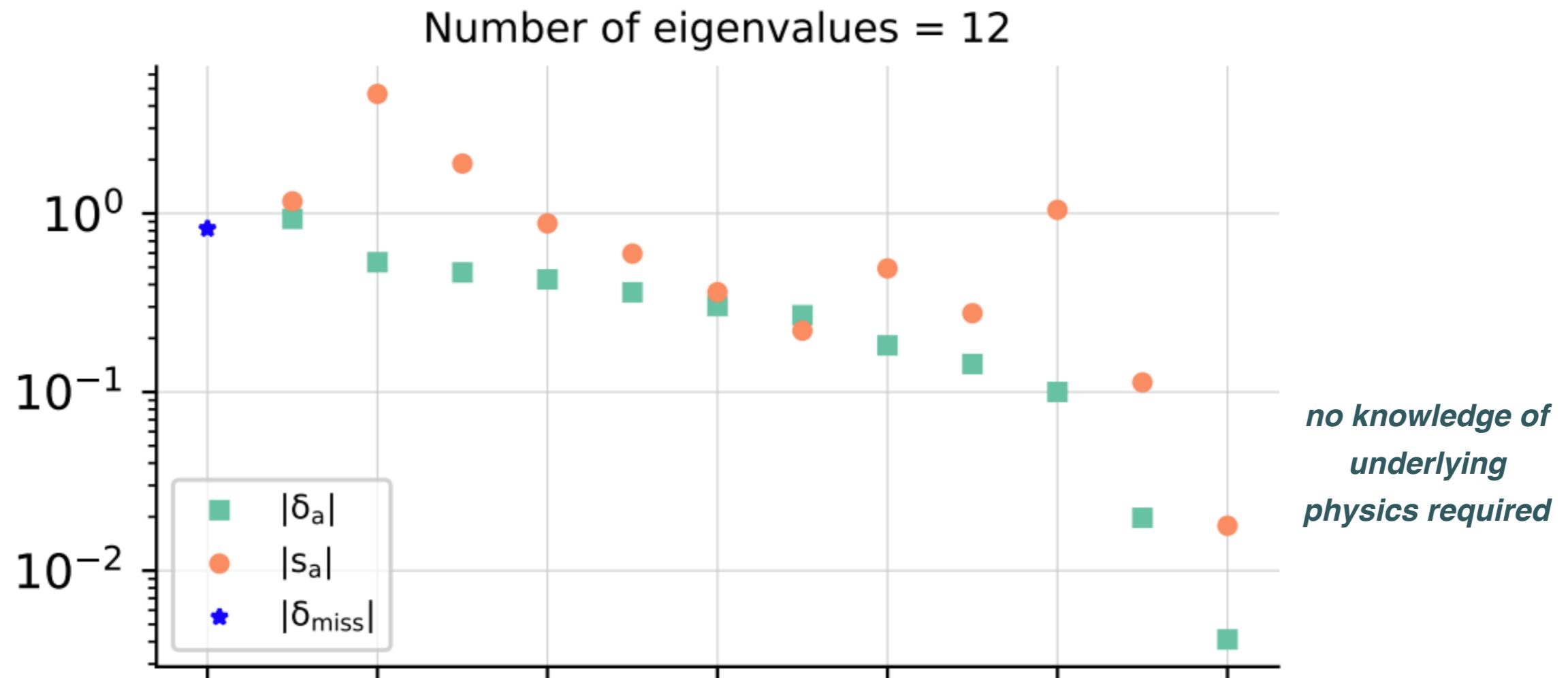
assume that some gives you the **covariance matrix of theory errors** of a global PDF fit (3000-dimensional space!). What are the **relevant components?**



Process Type	Dataset	Reference	$N_{\text{dat}}$	$N_{\text{dat}} (\text{total})$
DIS NC	NMC	[28, 29]	134	1593
	SLAC	[30]	12	
	BCDMS	[31, 32]	530	
	HERA $\sigma_{NC}^p$	[36]	886	
	HERA $\sigma_{NC}^c$	[37]	31	
DIS CC	NuTeV dimuon	[33, 34]	41	552
	CHORUS	[35]	430	
	HERA $\sigma_{CC}^p$	[36]	81	
DY	ATLAS $W, Z$ , 7 TeV 2010	[42]	30	484
	ATLAS $W, Z$ , 7 TeV 2011	[43]	34	
	ATLAS low-mass DY 2011	[44]	4	
	ATLAS high-mass DY 2011	[45]	5	
	ATLAS $Z$ $p_T$ 8 TeV ( $p_T^ll, M_{ll}$ )	[46]	44	
	ATLAS $Z$ $p_T$ 8 TeV ( $p_T^ll, y_Z$ )	[46]	48	
	CMS Drell-Yan 2D 2011	[51]	88	
	CMS $W$ asy 840 pb	[52]	11	
	CMS $W$ asy 4.7 pb	[53]	11	
	CMS $W$ rap 8 TeV	[54]	22	
	CMS $Z$ $p_T$ 8 TeV ( $p_T^ll, M_{ll}$ )	[55]	28	
	LHCb $Z$ 940 pb	[60]	9	
	LHCb $Z \rightarrow ee$ 2 fb	[61]	17	
	LHCb $W, Z \rightarrow \mu$ 7 TeV	[62]	29	
	LHCb $W, Z \rightarrow \mu$ 8 TeV	[63]	30	
	CDF $Z$ rap	[38]	29	
JET	D0 $Z$ rap	[39]	28	164
	D0 $W \rightarrow e\nu$ asy	[40]	8	
TOP	D0 $W \rightarrow \mu\nu$ asy	[41]	9	26
	ATLAS jets 2011 7 TeV	[47]	31	
	CMS jets 7 TeV 2011	[56]	133	
	ATLAS $\sigma_{tt}^{\text{top}}$	[48, 49]	3	
	ATLAS $t\bar{t}$ rap	[50]	10	26
	CMS $\sigma_{tt}^{\text{top}}$	[57, 58]	3	
	CMS $t\bar{t}$ rap	[59]	10	
	Total		2819	2819

# PCA for propagation of theory errors

assume that some gives you the **covariance matrix of theory errors** of a global PDF fit (3000-dimensional space!). What are the **relevant components**?



only **O(10)** directions encode all the variability of **3000-dimensional covmat**  
physical reason: theory errors are **highly correlated** among processes

# t-SNE

- In dimensional reduction and data visualisation techniques, it is often desirable to **preserve local structures** in high-dimensional datasets
- In many cases **non-linear methods** (unlike PCA, which is linear) are required
- One of these is **t-stochastic neighbour embedding** (t-SNE), where each high-dimensional training point is mapped to low-dimensional embedding coordinates optimized to preserve the local structures in the data

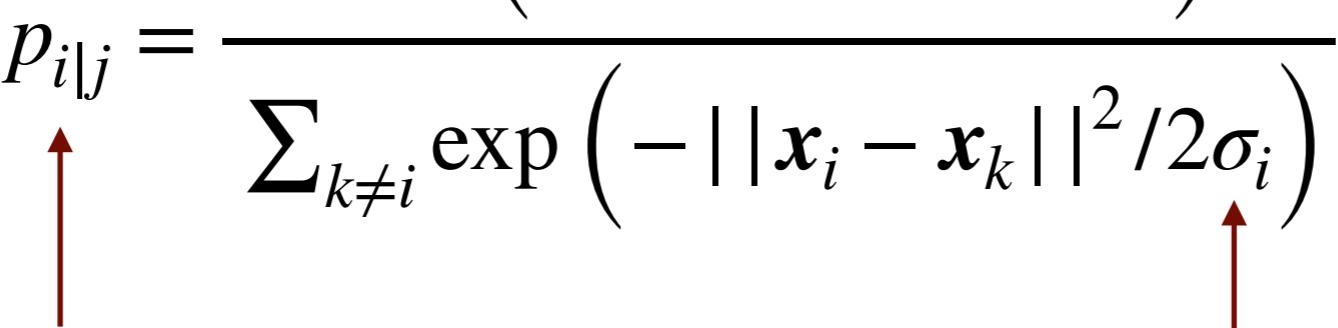
the main idea is to **associate a probability distribution** to the neighbour of each data point

$$p_{i|j} = \frac{\exp\left(-||x_i - x_j||^2/2\sigma_i\right)}{\sum_{k \neq i} \exp\left(-||x_i - x_k||^2/2\sigma_i\right)} \quad x_i \in \mathbb{R}^p$$

**Data Space**

*probability that j  
is neighbour of i*

*model parameter*



# t-SNE

the main idea is to **associate a probability distribution** to the neighbour of each data point

**Data Space**

$$p_{i|j} = \frac{\exp\left(-||x_i - x_j||^2/2\sigma_i\right)}{\sum_{k \neq i} \exp\left(-||x_i - x_k||^2/2\sigma_i\right)} \quad x_i \in \mathbb{R}^p$$

and then **constructing a similar probability** in the **lower-dimensional latent space**

**Latent Space**

$$q_{i|j} = \frac{\left(1 + ||y_i - y_j||^2\right)^{-1}}{\sum_{k \neq i} \left(1 + ||y_i - y_k||^2\right)^{-1}} \quad y_i \in \mathbb{R}^{p'}, \quad p' < p$$

the latent space coordinates are determined by minimising the **Kullback-Leibler divergence** between the two probability distributions

**cost function:**  $C(Y) = D_{\text{KL}}(p || q) \equiv \sum_{ij} p_{i|j} \log \frac{p_{i|j}}{q_{i|j}}$

*minimisation using  
e.g. Gradient Descent  
output of minimisation:  
latent-space coordinates  $\{y_i\}$   
of each data point  $\{x_i\}$*

# The Kullback-Leibler divergence

The KL divergence, a measure of the similarity between two probability distributions  $p(\mathbf{x})$  and  $q(\mathbf{x})$  plays an important role in machine learning applications

$$D_{KL}(p \parallel q) = \int d\mathbf{x} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} \quad D_{KL}(q \parallel p) = \int d\mathbf{x} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})}$$

which can be symmetrised to construct a **squared metric** (distance)

$$D_{JS}(p \parallel q) = \frac{1}{2} \left( D_{KL}\left(p \middle\| \frac{p+q}{2}\right) + D_{KL}\left(q \middle\| \frac{p+q}{2}\right) \right)$$

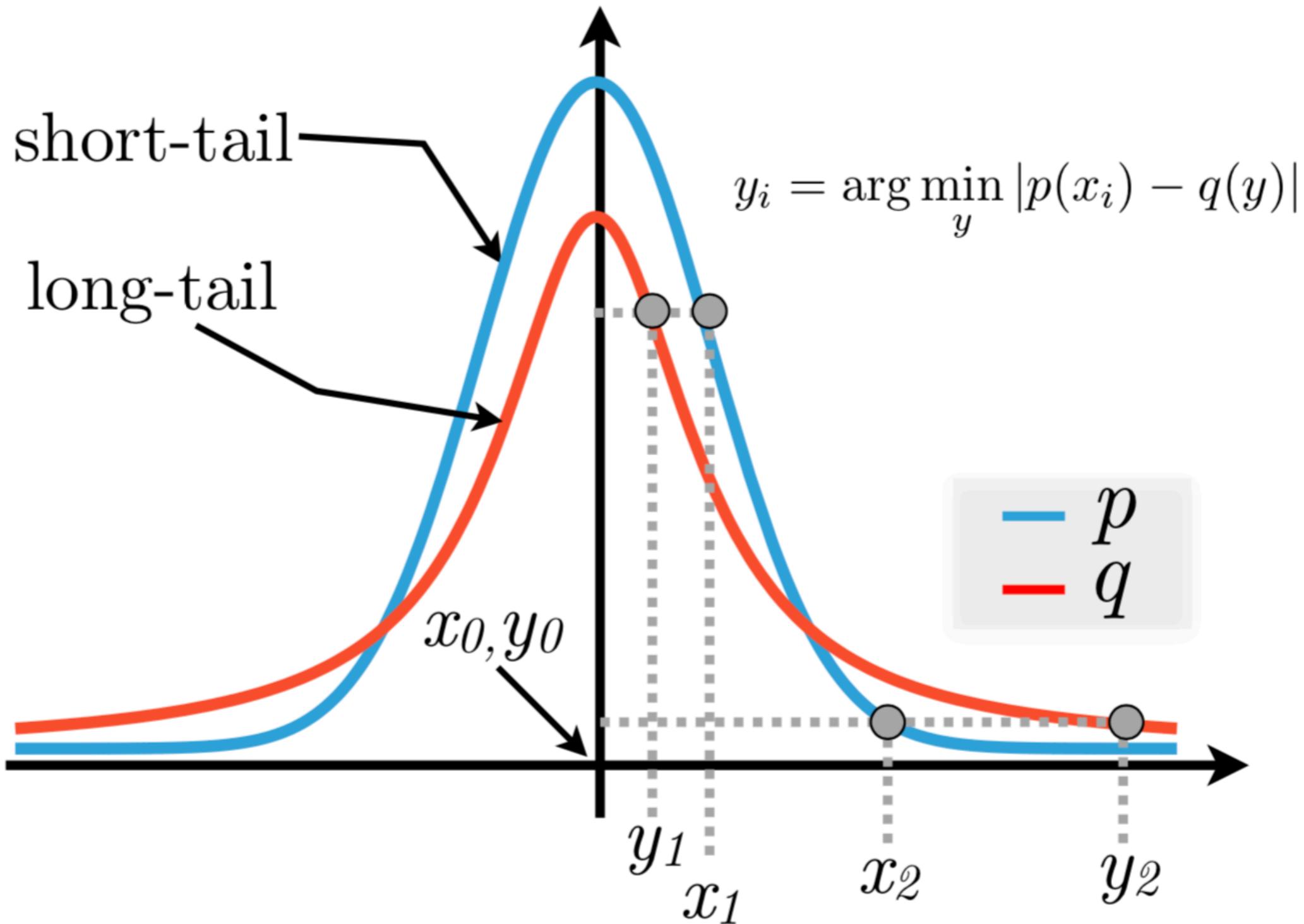
*Jensen-Shannon divergence*

The KL-divergence is **positive-definite**, and only vanishes when  $p(\mathbf{x})=q(\mathbf{x})$

$$D_{KL}(p \parallel q) \geq 0$$

in general the integral cannot be computed and one needs to sample the two probability distributions by means of a suitable binning

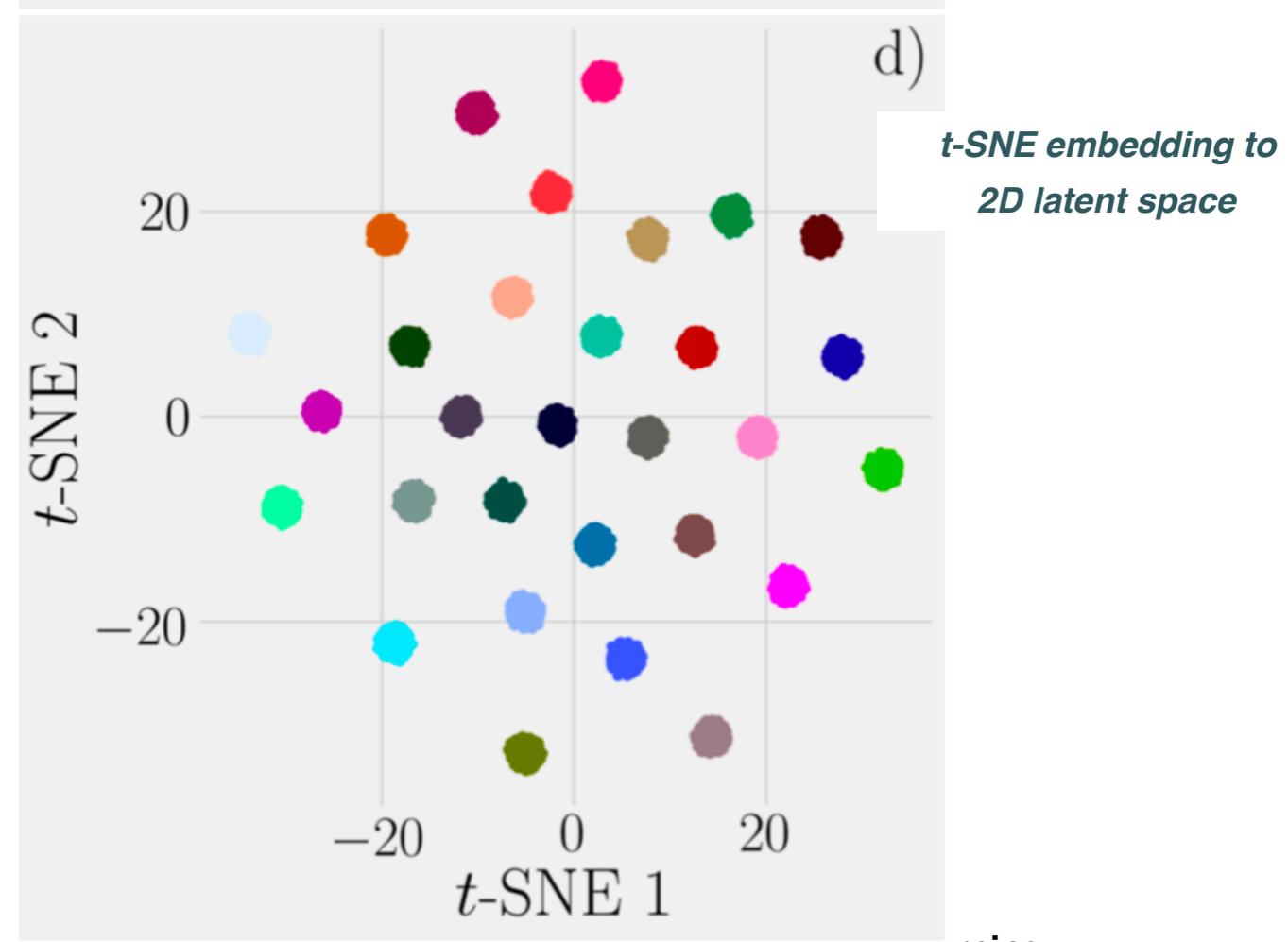
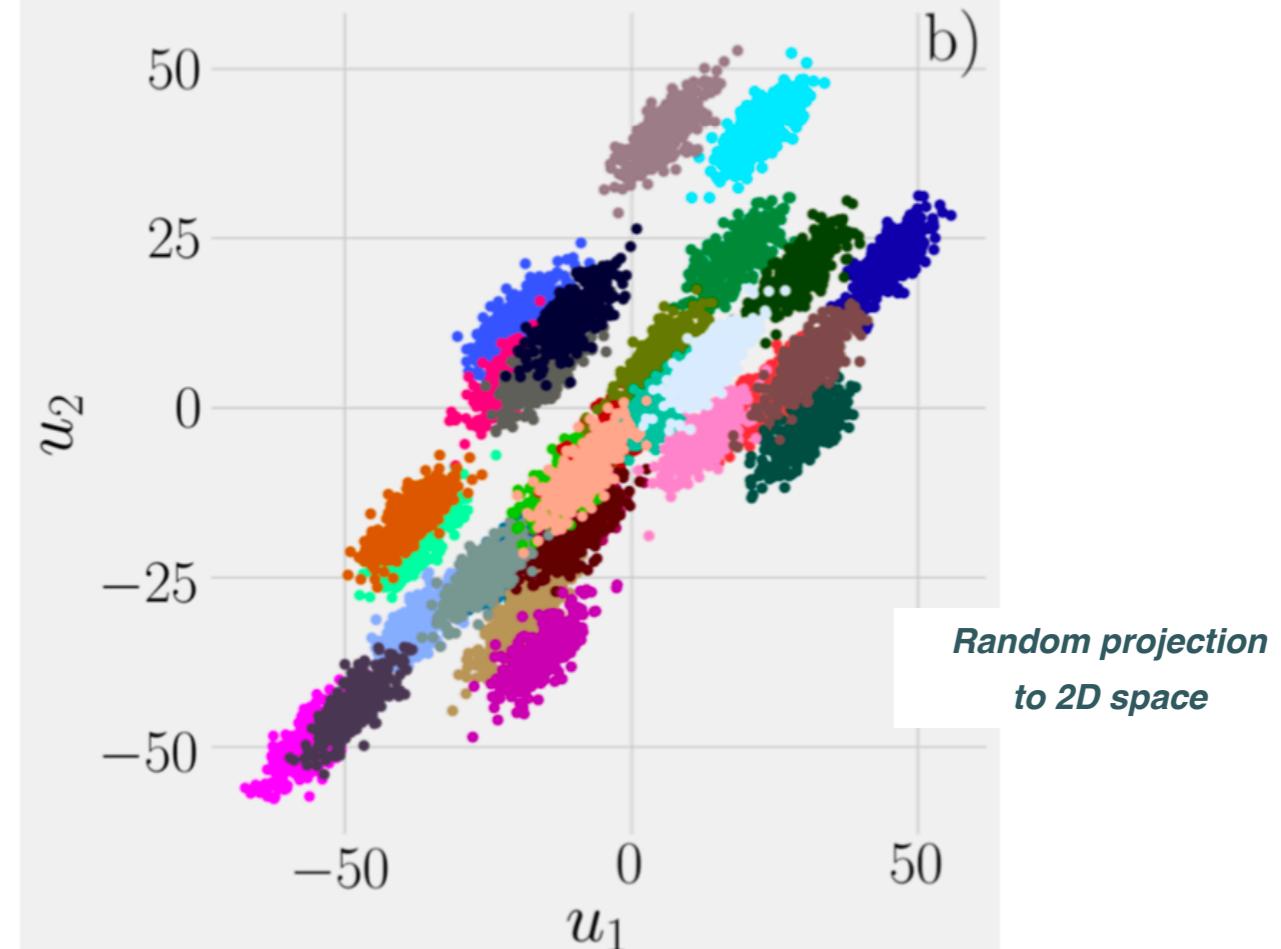
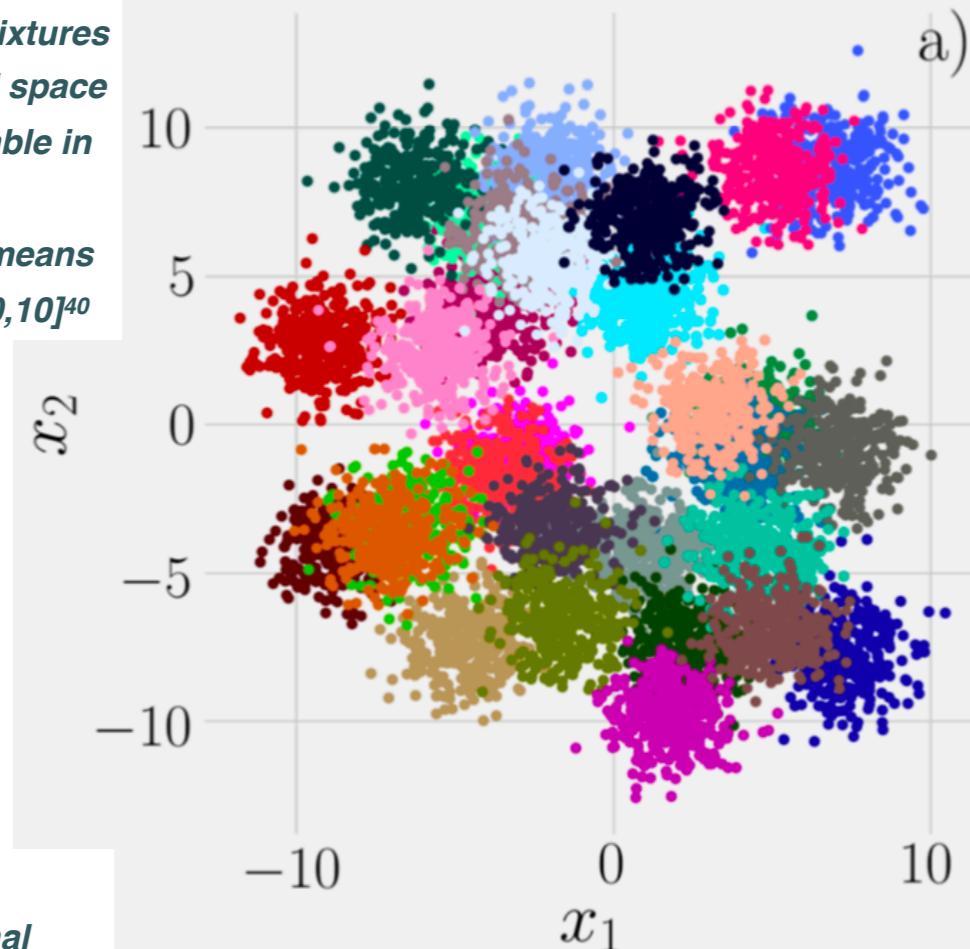
# t-SNE



Note that  $q$  is a **long-tail distribution** (Cauchy): this preserves short distance information while repelling two points that are far apart in the original space

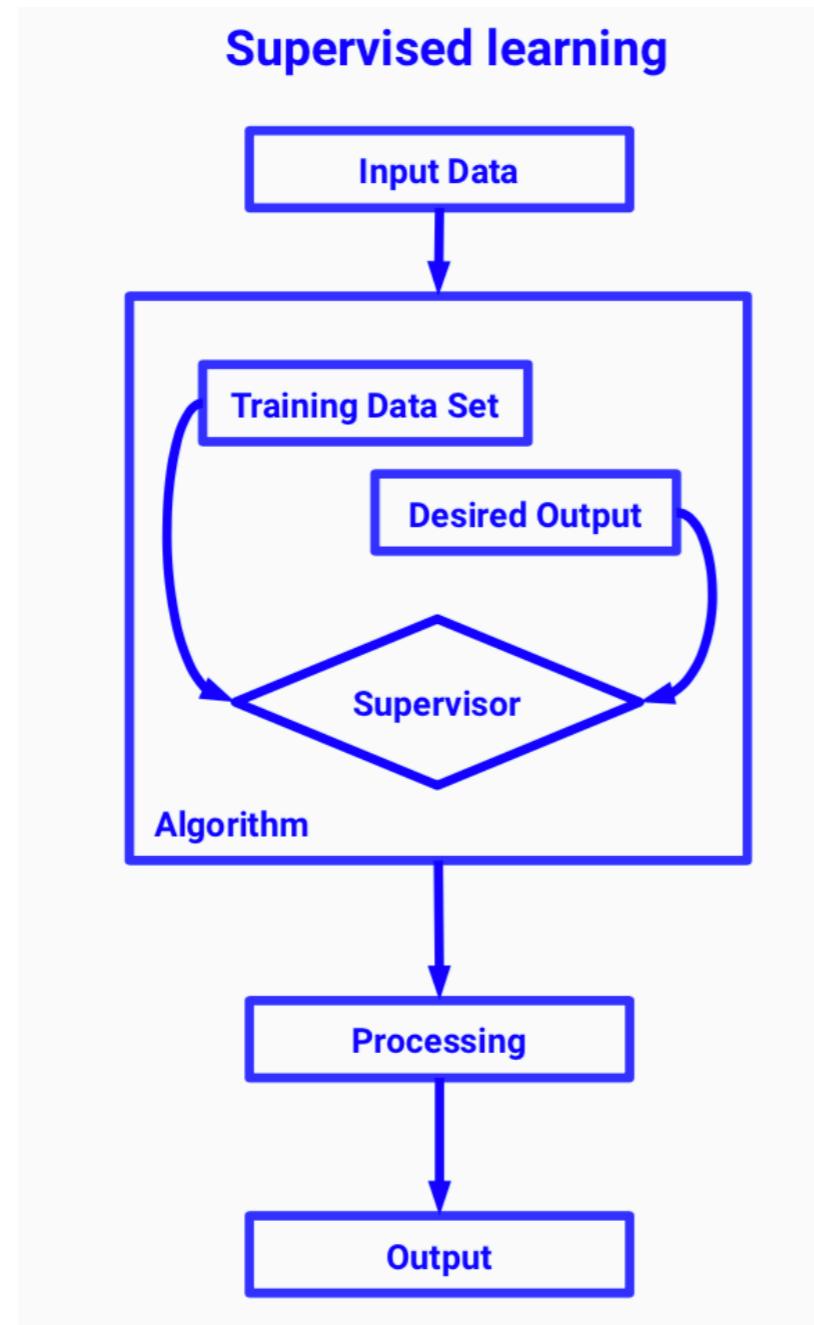
**K=30 Gaussian mixtures  
in 40-dimensional space  
(labels not available in  
real case!)**

**Same variance, means  
at random in  $[-10, 10]^{40}$**



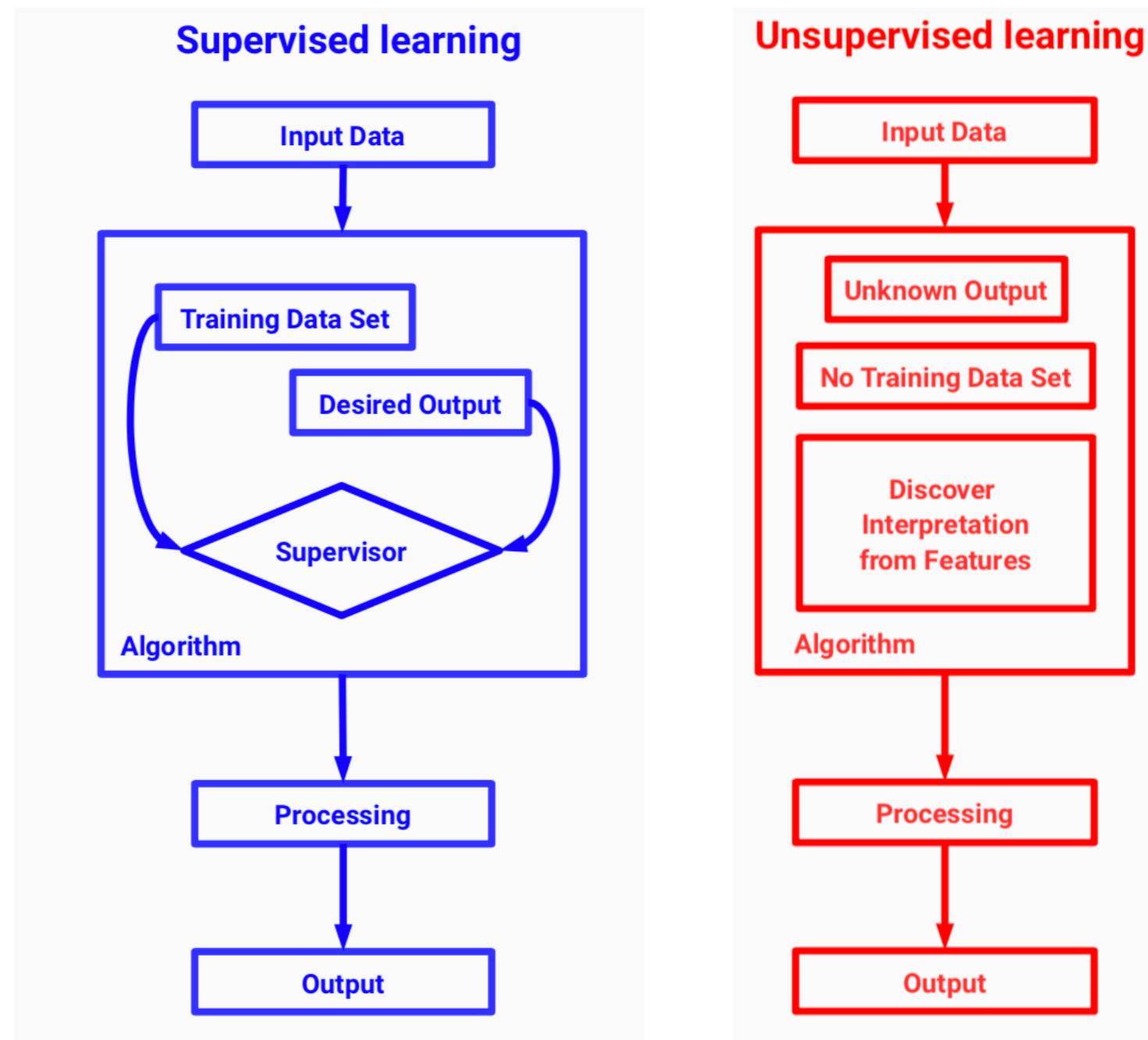
# Unsupervised Learning

# Supervised vs Unsupervised Learning



unlike Supervised Learning, where we aim to find a model that reproduces the underlying law of a set of input/output patterns, in Unsupervised Learning there are no labels: **aim is to identify underlying structures and connections in the data**

# Supervised vs Unsupervised Learning



unlike Supervised Learning, where we aim to find a model that reproduces the underlying law of a set of input/output patterns, in Unsupervised Learning there are no labels: **aim is to identify underlying structures and connections in the data**

# Unsupervised Learning

In ML context, **unsupervised learning** is concerned with discovering underlying structures in **unlabelled data**

an important example of unsupervised learning is **clustering**: the aim is to group unlabelled data into clusters using some **distance or similarity measure**

let us illustrate these ideas with **K-means clustering**

$$\{\boldsymbol{x}_n\}_{n=1}^N \quad \boldsymbol{x}_n = (x_{n,1}, x_{n,2}, \dots, x_{n,p})$$

*unlabelled dataset: N points with p features each*

$$\{\boldsymbol{\mu}_k\}_{k=1}^K \quad \boldsymbol{\mu}_k = (\mu_{k,1}, \mu_{k,2}, \dots, \mu_{k,p})$$

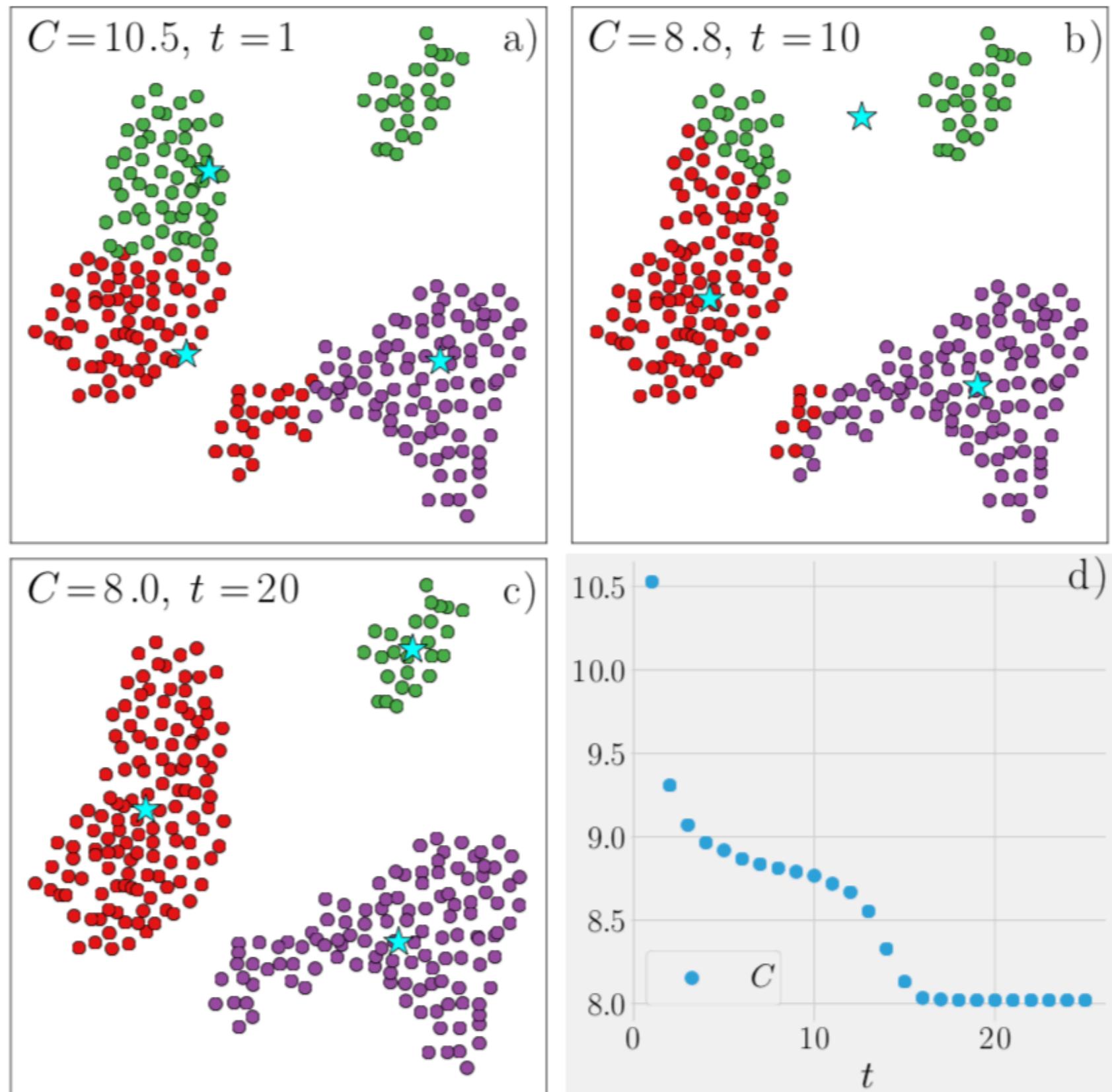
**cluster means:** K clusters with p features each

the intuitive idea is that the cluster means represent the **main features of each cluster**, to which the data points will be assigned in the clustering procedure

# Clustering

**2D example of clustering:** each colour represents a cluster, with stars indicating their **centers**

how is this clustering achieved **in practice?**



# Clustering

in  $K$ -means clustering, the **cluster means** and the **data point assignments** are determined from the minimisation of a cost function:

$$C(\mathbf{x}; \boldsymbol{\mu}) = \sum_{n=1}^N \sum_{k=1}^K r_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k)^2$$

*binary assignment variable*

*Euclidean distance between  $n$ -th data point and  $k$ -th cluster centre*

$r_{nk} = 1 \longrightarrow$  *the  $n$ -th point is assigned to the  $k$ -th cluster*

$r_{nk} = 0 \longrightarrow$  *the  $n$ -th point is not assigned to the  $k$ -th cluster*

furthermore since **clustering is exclusive** one needs to impose:

$$\sum_{k=1}^K r_{nk} = 1 \quad \forall n$$

one sees that  $K$ -means clustering aims to **minimise the variance within each cluster**

*other distances possible:  
the user needs to define  
what ``closeness'' means!*

# Clustering

Let's describe an algorithm that implements  $K$ -means clustering by minimising the cost function

$$C(\mathbf{x}; \boldsymbol{\mu}) = \sum_{n=1}^N \sum_{k=1}^K r_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k)^2$$

this algorithm alternates iteratively between two main steps:

• (1) **Expectation:** starting from set of cluster assignments  $\{r_{nk}\}$  minimise  $C$  wrt cluster means

$$\frac{\partial}{\partial \boldsymbol{\mu}_k} C(\mathbf{x}; \boldsymbol{\mu}) = 0 \quad \rightarrow \quad \boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^N r_{nk} \mathbf{x}_n \quad N_k = \sum_{n=1}^N r_{nk}$$

*number of points  
in k-th cluster*

• (2) **Maximization:** given the  $K$  cluster centers, the assignments  $\{r_{nk}\}$  should minimise  $C$ . This can be achieved by assigning each data point to its closest cluster-mean

$$r_{nk} = 1 \quad \text{if} \quad k = \arg \min_{k'} (\mathbf{x}_n - \boldsymbol{\mu}_{k'})$$

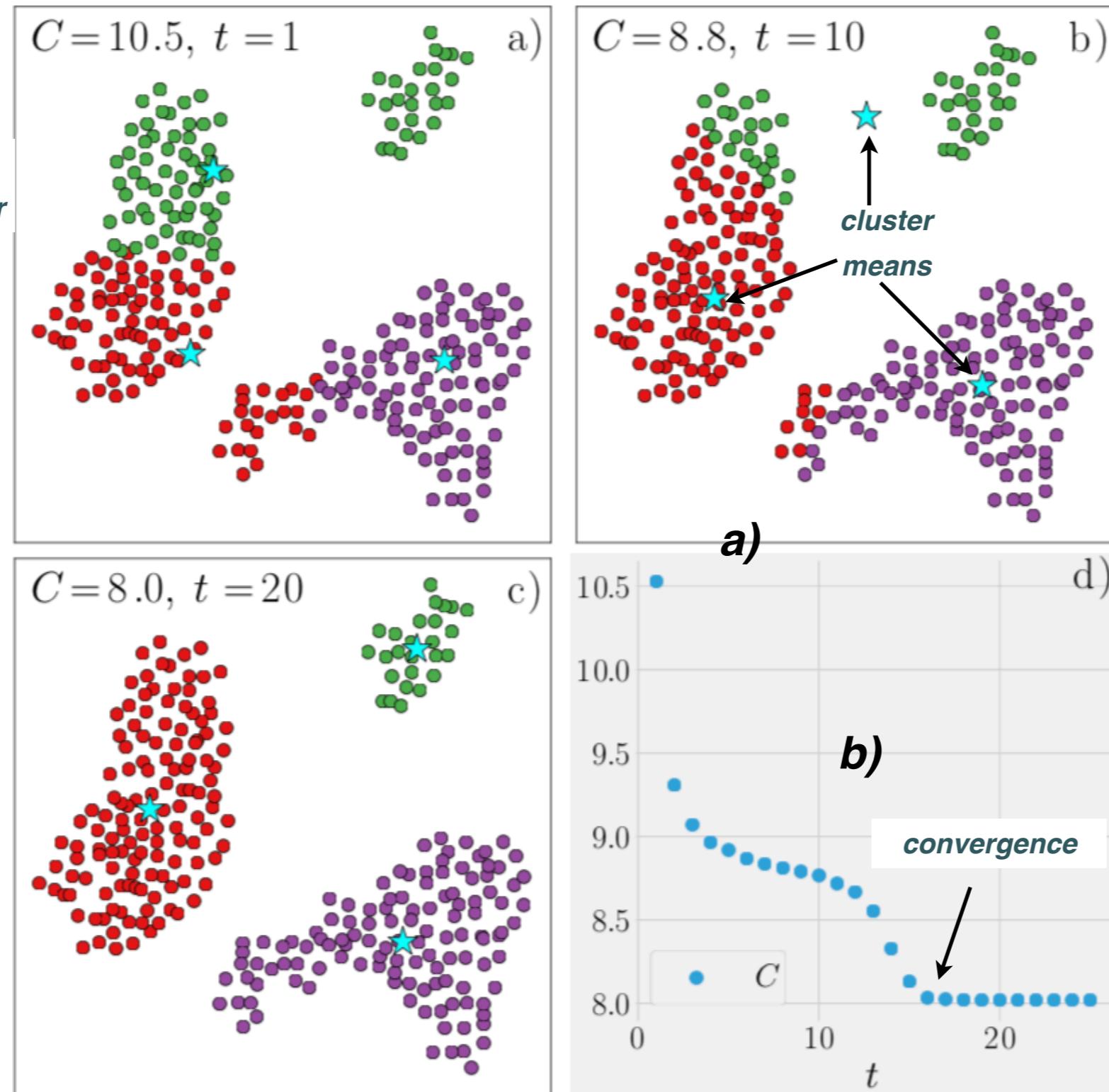
*iterate until convergence  
achieved!*

$$r_{nk} = 0 \quad \text{if} \quad k \neq \arg \min_{k'} (\mathbf{x}_n - \boldsymbol{\mu}_{k'})$$

*note that here GD not  
required, optimisation is  
semi-analytical*

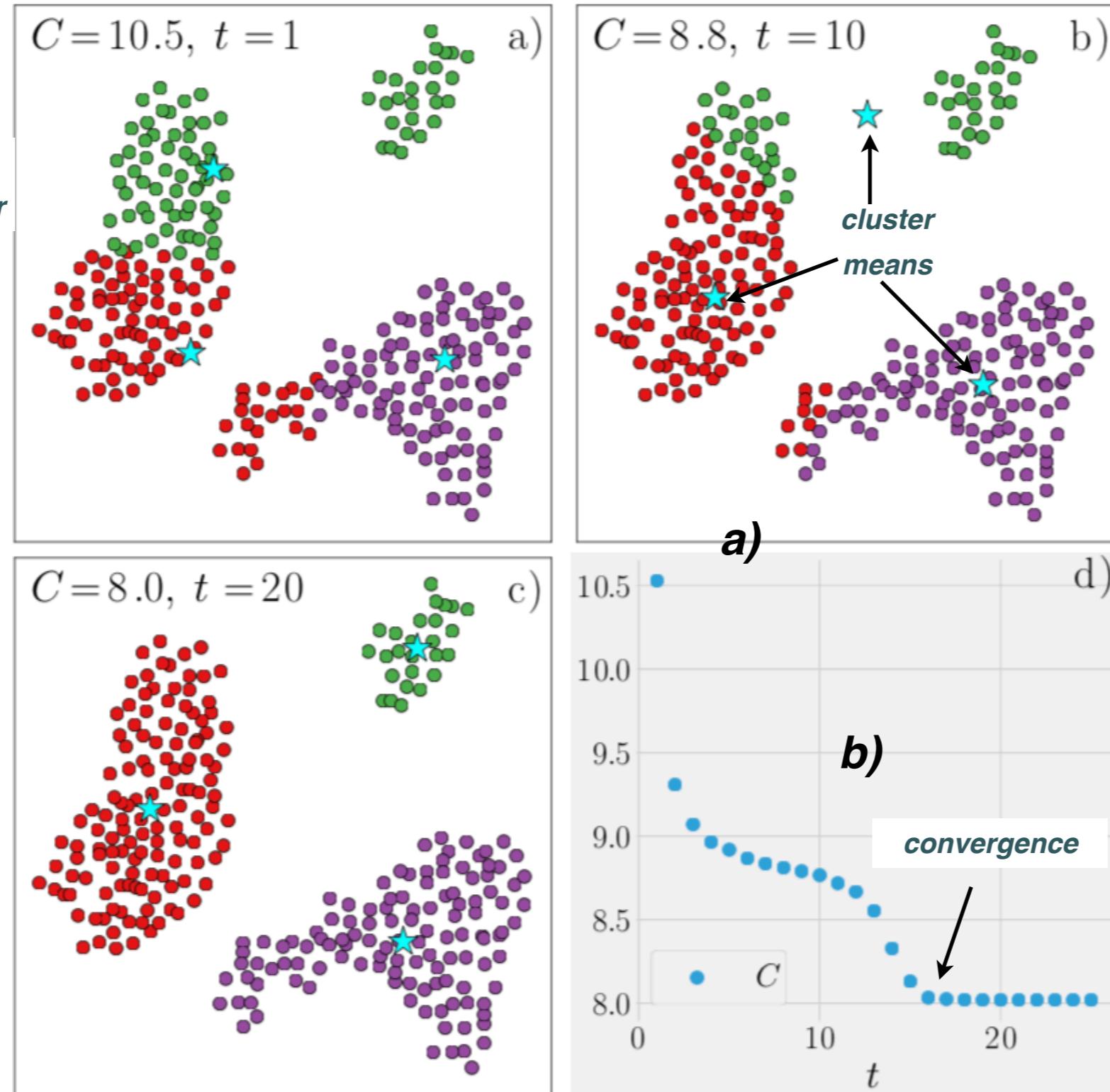
# Clustering

these two steps are iterated until some **convergence criterion** is achieved, e.g. when the change in the cost function between two iterations is below some threshold



# Clustering

these two steps are iterated until some **convergence criterion** is achieved, e.g. when the change in the cost function between two iterations is below some threshold



*K-means clustering can lead to spurious results since the **underlying assumption** is that the latent model has uniform variances*

*fails if the underlying clusters have different variances!*

# Hierarchical clustering

- Another approach to clustering is based on **agglomerative methods**, where one starts from small clusters which are progressively **merged into bigger clusters**
- This hierarchical structure provides information on the relations between clusters and the **subcomponents of individual clusters**
- As before, we need to specify a **distance**, this time between two clusters  $X, Y$

$$d(X, Y) \in \mathcal{R}$$

- At each iteration, the two clusters closer to each other (quantified by  $d$ ) are *merged*

the **agglomerative cluster algorithm** works as follows:

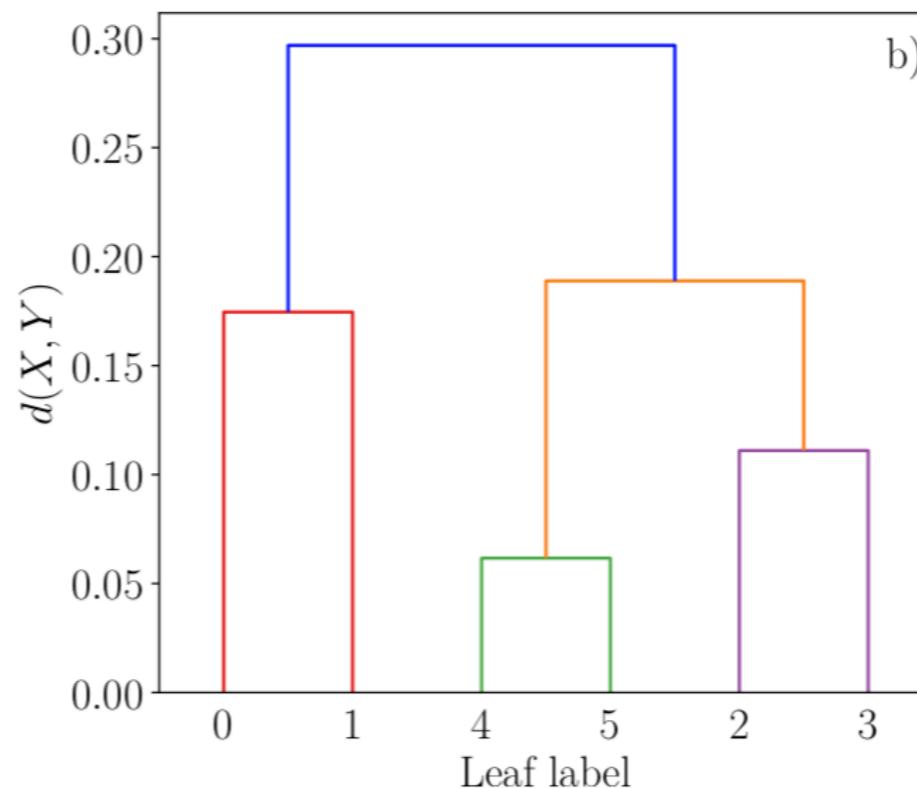
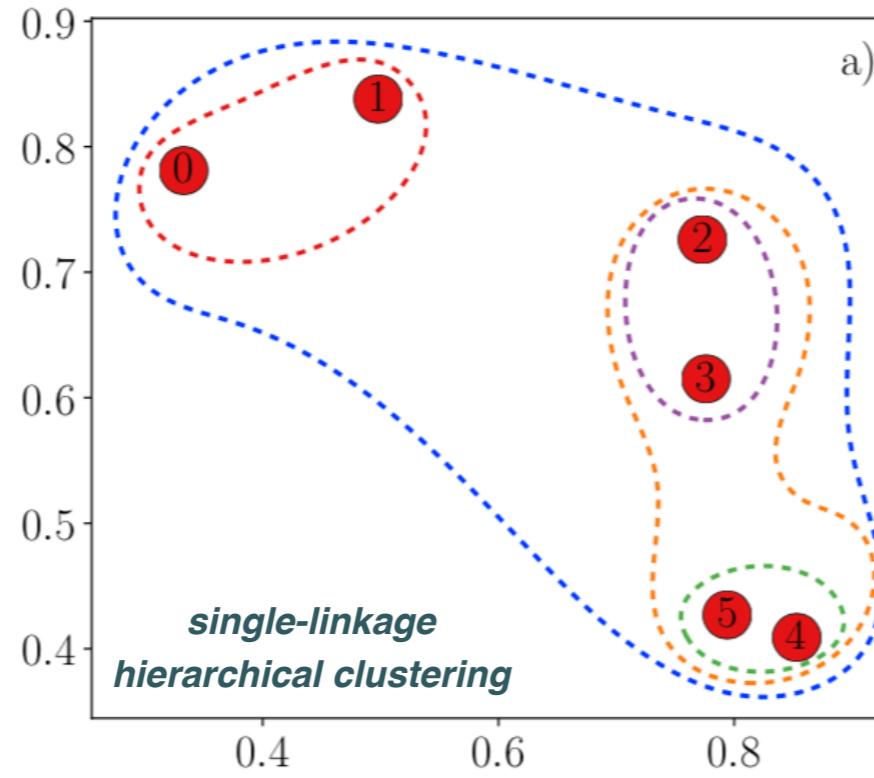
- (1) Assign **each data point to be its own cluster**
- (2) Given the resulting set of  $K$  clusters, find the closest pair
$$(X_i, X_j) \text{ such that } (i, j) = \arg \min_{i'j'} d(X_{i'}, X_{j'})$$
- (3) Merge the pair into a single cluster. Iterate (2) and (3) until a single cluster remains

# Hierarchical clustering

Clearly the results of hierarchical clustering depend on the **choice of distance**

single linkage  $\longrightarrow d(X_i, X_j) = \min_{x_i \in X_i, x_j \in X_j} \|x_i - x_j\|_2$  Euclidean  
distance

complete linkage  $\longrightarrow d(X_i, X_j) = \max_{x_i \in X_i, x_j \in X_j} \|x_i - x_j\|_2$  Euclidean  
distance

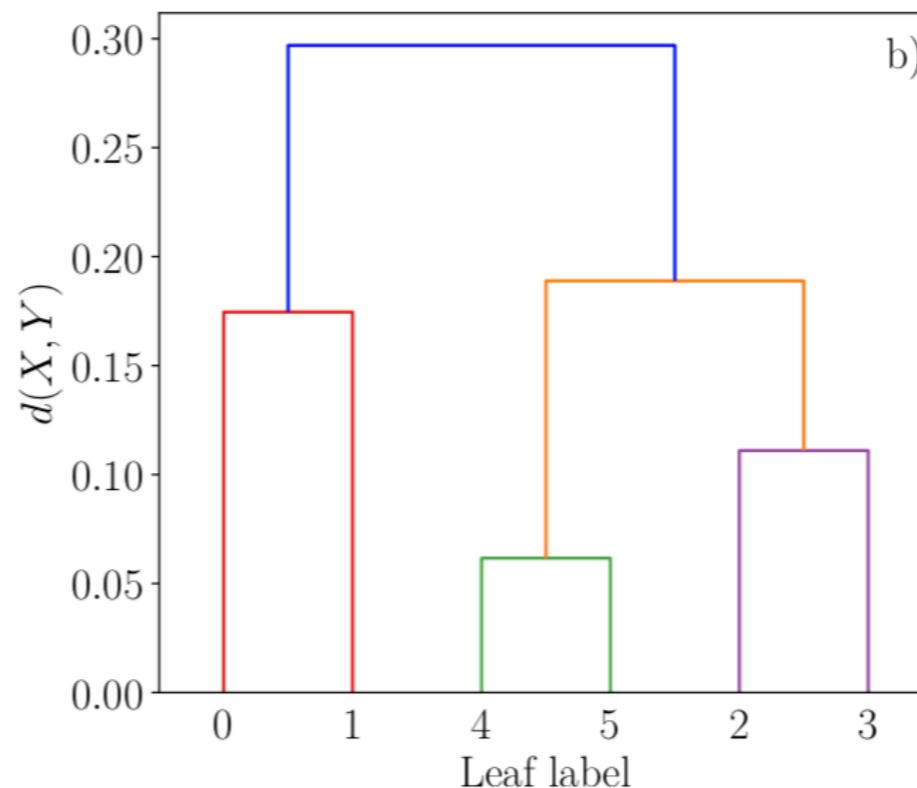
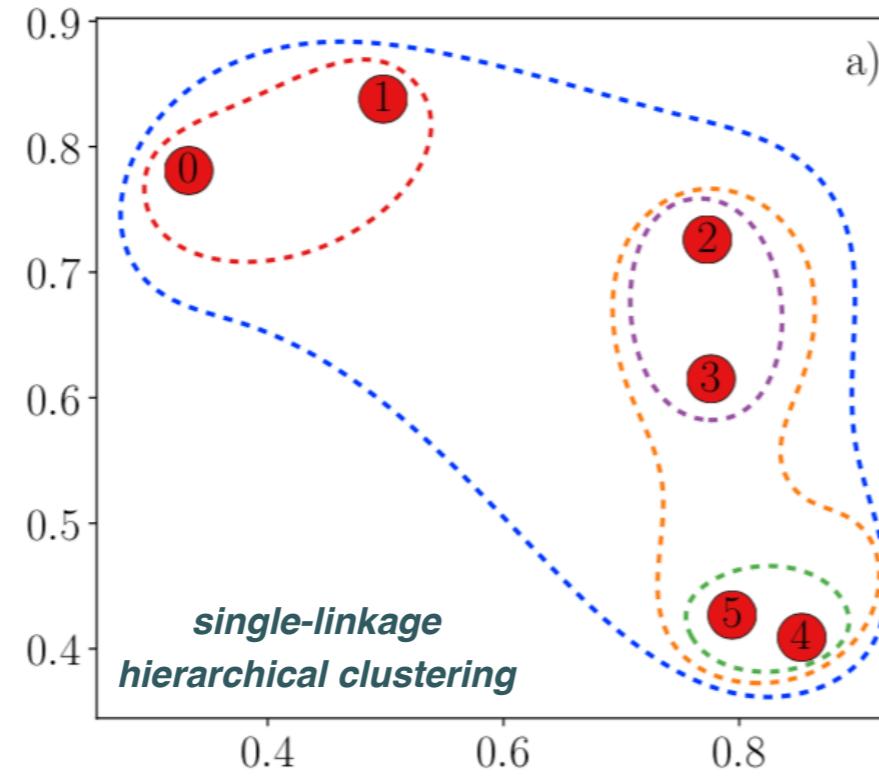


# Hierarchical clustering

Clearly the results of hierarchical clustering depend on the **choice of distance**

single linkage  $\longrightarrow d(X_i, X_j) = \min_{x_i \in X_i, x_j \in X_j} \|x_i - x_j\|_2$  Euclidean  
distance

complete linkage  $\longrightarrow d(X_i, X_j) = \max_{x_i \in X_i, x_j \in X_j} \|x_i - x_j\|_2$  Euclidean  
distance



hierarchical clustering methods do not scale well for large  $N$ , so they are typically **combined with K-means clustering** in the initial steps to define small clusters

# Clustering and Latent variables

A central concept in **Unsupervised Learning** is that of a **latent or hidden variable**: not directly observable, but still they influence visible structure of data

*e.g. in clustering, a latent variable is the cluster identity of each datapoint*

One can think of clustering as an algorithm to learn **the most probable value of a latent variable**

a common feature of all Unsupervised Learning algorithms is the need for assumptions about the underlying probability distribution of the data: the **generative model**

*e.g. in K-means clustering, we assume that the points of each cluster are generated Gaussianly with respect to its mean (center)*

$$C(x; \mu) = \sum_{n=1}^N \sum_{k=1}^K r_{nk} (x_n - \mu_k)^2$$

different **generative models** lead to different types of clustering algorithms

# Gaussian Mixture Models

in **Gaussian Mixture Models (GMM)**, a generative model used in clustering applications, points are drawn from  $K$  gaussians with different means and covariances

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \sim \exp \left[ -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})^T \right] \quad \text{one of the } K \text{ gaussians}$$

The probability of generating a point  $\mathbf{x}$  in a GMM is given by

$$p(\mathbf{x}, \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k\}) = \sum_{k=1}^K \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \pi_k$$

*probability of drawing a point from mixture k*

$$\theta = \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k\}$$

*GMM parameters*

the probability that a data point  $\mathbf{x}$  is associated to the  $k$ -th cluster is

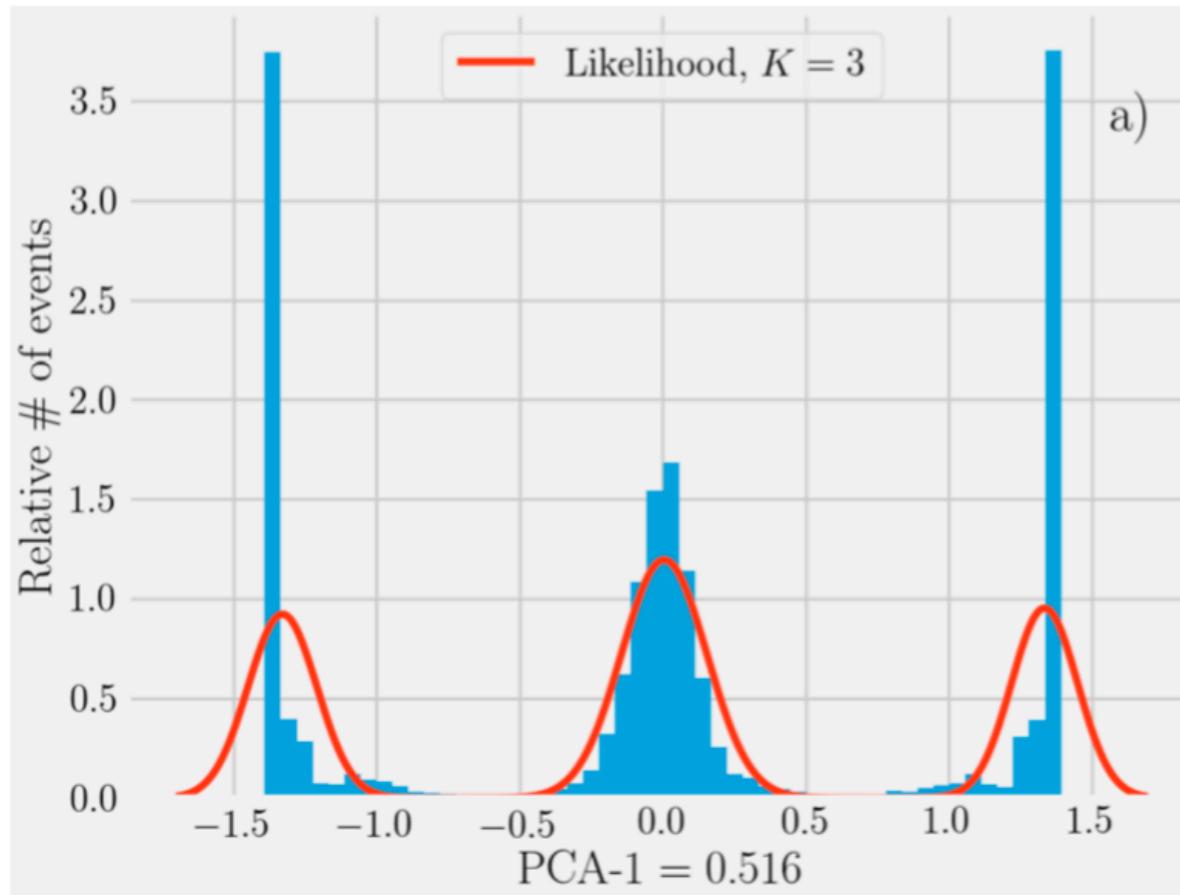
$$\gamma_k(\mathbf{x}) \sim \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \pi_k$$

model parameters found by maximising likelihood using SGD

$$\hat{\theta} = \arg \max_{\theta} \log p(\mathbf{X} | \theta)$$

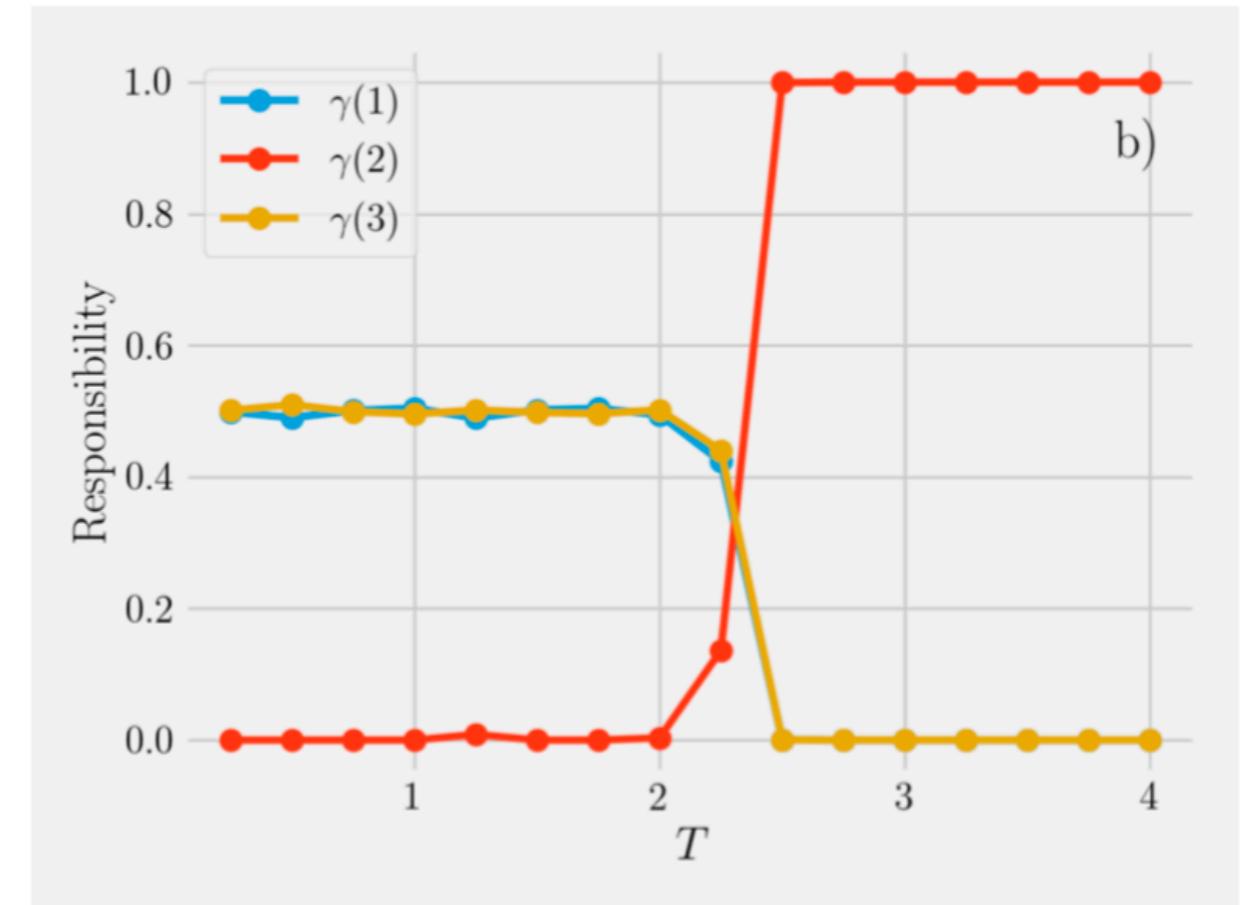
# Gaussian Mixture Models

*Ising dataset fitted to 3-component GGM*



*1st principal component  
(magnetisation)*

*Probability of being on each phase*



*probability coincide at critical point*

recall that this model has been trained only on examples: **no knowledge of the underlying physical mechanisms** whatsoever

# Tutorial 3 Exercise 3a

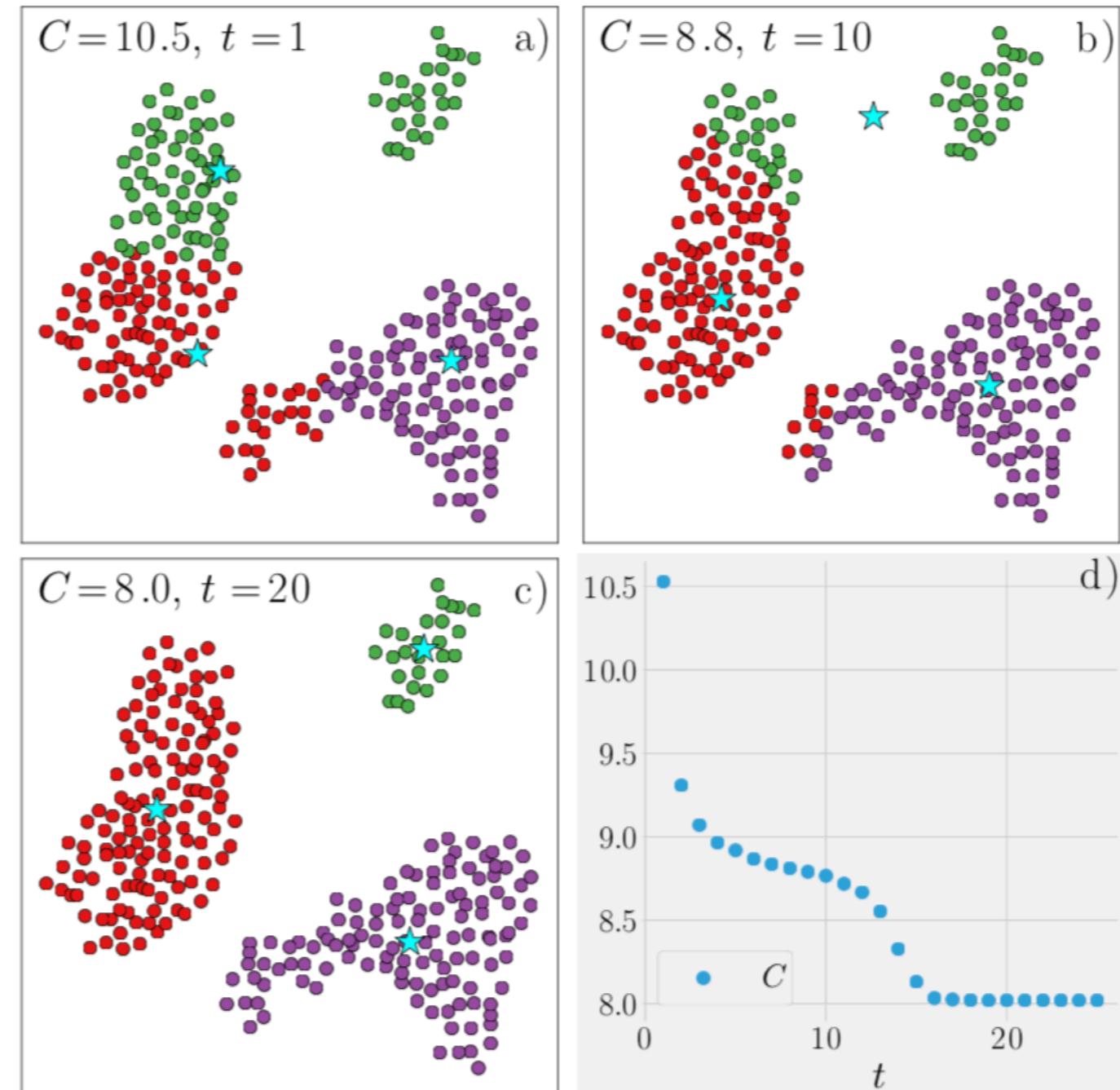
starting point is the **Python script** that you will find in

<https://github.com/juanrojochacon/ml-ditp-attp/blob/master/Tutorials/Tutorial3/>

- 💡 Become familiar with the ideas underlying Unsupervised Learning with their application to **clustering**

- 💡 Compare the performance of different clustering algorithms, and study different ways of representing the training data

- 💡 How do results depend on your assumptions concerning the definition of the distance between points



# Ensemble Methods

# Combining models

a powerful strategy in machine learning is that of **ensemble methods** that **combine predictions from multiple statistical models** to improve predictive performance

- 💡 Key aspect: assess the **degree of correlation** between the models of the ensemble
- 💡 The reason is that **combining correlated models reduces the overall variance less** than in the uncorrelated case
- 💡 Also these correlations can **increase the bias of the combined model**, offsetting potential reductions in variance from ensemble averaging

# Combining models

a powerful strategy in machine learning is that of **ensemble methods** that **combine predictions from multiple statistical models** to improve predictive performance

- Key aspect: assess the **degree of correlation** between the models of the ensemble
- The reason is that **combining correlated models reduces the overall variance less** than in the uncorrelated case
- Also these correlations can **increase the bias of the combined model**, offsetting potential reductions in variance from ensemble averaging
  - e. g. assume that your model predicts  $X$ , and you have  $n$  models. If each of these models has variance  $\sigma$ , then the variance of their sum is

$$\text{Var} \left( \sum_{i=1}^n X_i \right) = \sum_{i=1}^n \text{Var}(X_i) + 2 \sum_{1 \leq i \leq j \leq n} \text{Cov}(X_i, X_j)$$

$$\text{Var}(\bar{X}) = \text{Var} \left( \frac{1}{n} \sum_{i=1}^n X_i \right) = \frac{1}{n^2} \sum_{i=1}^n \text{Var}(X_i)$$

$$\text{Var}(\bar{X}) = \frac{\sigma^2}{n} + \frac{n-1}{n} \rho \sigma^2$$

*reduction as  $n$  increased*

*correlations increase variance*

# Combining models

a powerful strategy in machine learning is that of **ensemble methods** that **combine predictions from multiple statistical models** to improve predictive performance

- 💡 Key aspect: assess the **degree of correlation** between the models of the ensemble
- 💡 The reason is that **combining correlated models reduces the overall variance less** than in the uncorrelated case
- 💡 Also these correlations can **increase the bias of the combined model**, offsetting potential reductions in variance from ensemble averaging
  - e. g. assume that your model predicts  $X$ , and you have  $n$  models. If each of these models has variance  $\sigma$ , then the variance of their sum is

$$\text{Var}(\bar{X}) = \frac{\sigma^2}{n} + \frac{n-1}{n}\rho\sigma^2$$

*for fully correlated models*   $\text{Var}(\bar{X}) = \frac{\sigma^2}{n} + \frac{n-1}{n}\sigma^2 \rightarrow \sigma^2$

*no reduction of variance in the combination*

# Bagging

**Bootstrap AGGregation (Bagging) is a popular ensemble combination method**

we start from a large dataset that is **partitioned** into  $M$  smaller datasets:

$$\mathcal{L} \rightarrow \{\mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_M\} \quad \sum_{m=1}^M \mathcal{L}_m = \mathcal{L}$$

so that each of the  $M$  datasets is large enough to train a predictor. The **aggregate predictor** is then constructed from those trained in each separate dataset

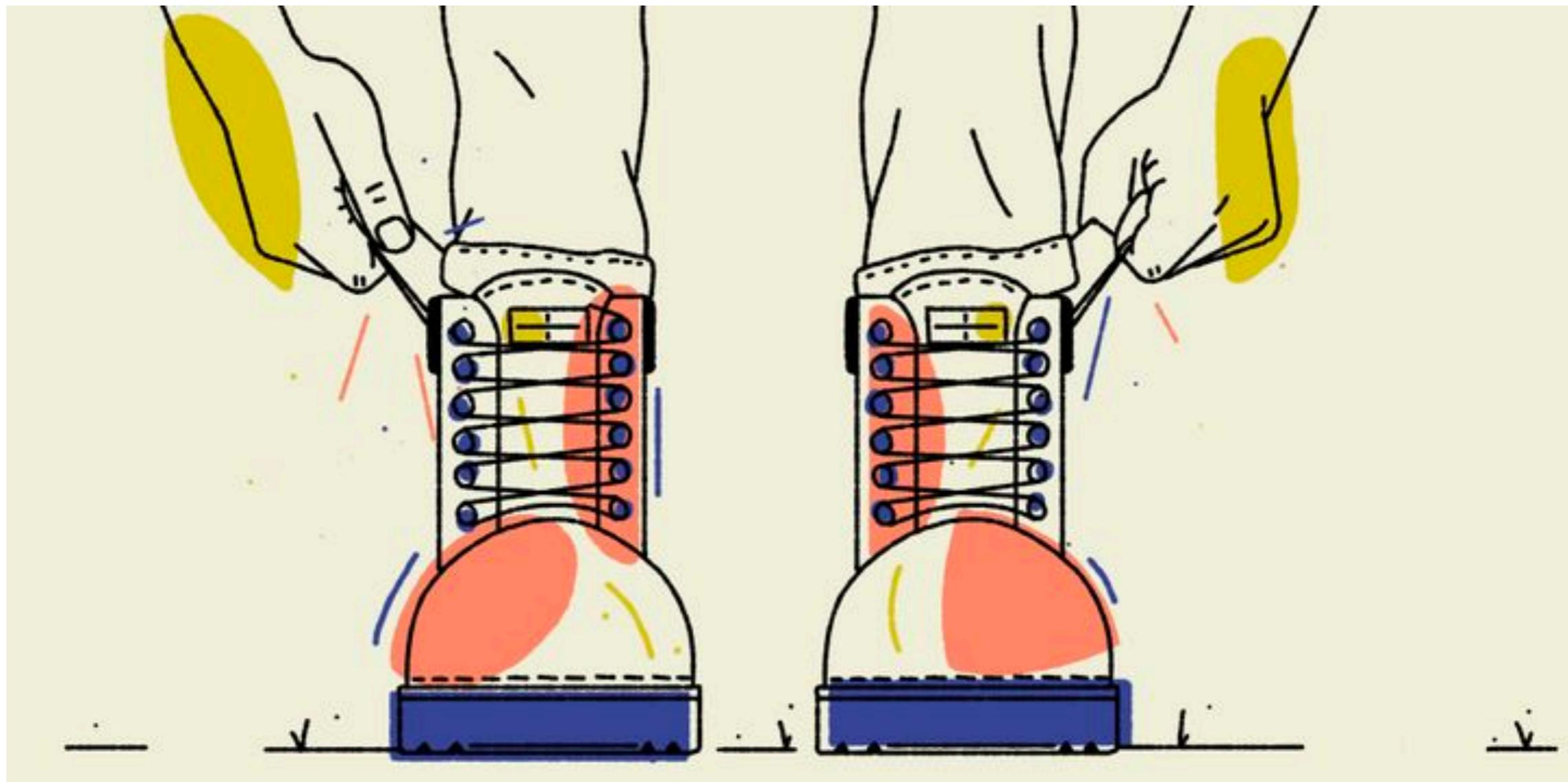
$$\hat{g}_{\mathcal{L}}^A(\mathbf{x}) = \frac{1}{M} \sum_{i=1}^M g_{\mathcal{L}_i}^A(\mathbf{x}) \quad \textit{for continuous datasets, analogous expressions for discrete classifiers}$$

such aggregation can **reduce the variance without increasing the bias**

But what should we do if we don't have a very large dataset? If each partitioned set one has few data points then the prediction will be poor ...

*and if using more data/examples is not possible?*

# Bootstrapping



Bootstrapping quantifies properties of an estimator by measuring those properties when **sampling from an approximating distribution** (e.g. a subset of the original data)

# Bootstrapping

- Assume we are given a training dataset and we want to compute e.g. confidence intervals

$$\mathcal{D} = \{X_1, \dots, X_n\}$$

- This can be done by **sampling  $n$  points with replacement** to get  **$B$  new datasets**

$$\begin{aligned}\mathcal{D}^{*(1)} &= \{X_1^{*(1)}, \dots, X_n^{*(1)}\} \\ &\vdots \\ \mathcal{D}^{*(B)} &= \{X_1^{*(B)}, \dots, X_n^{*(B)}\}\end{aligned}$$

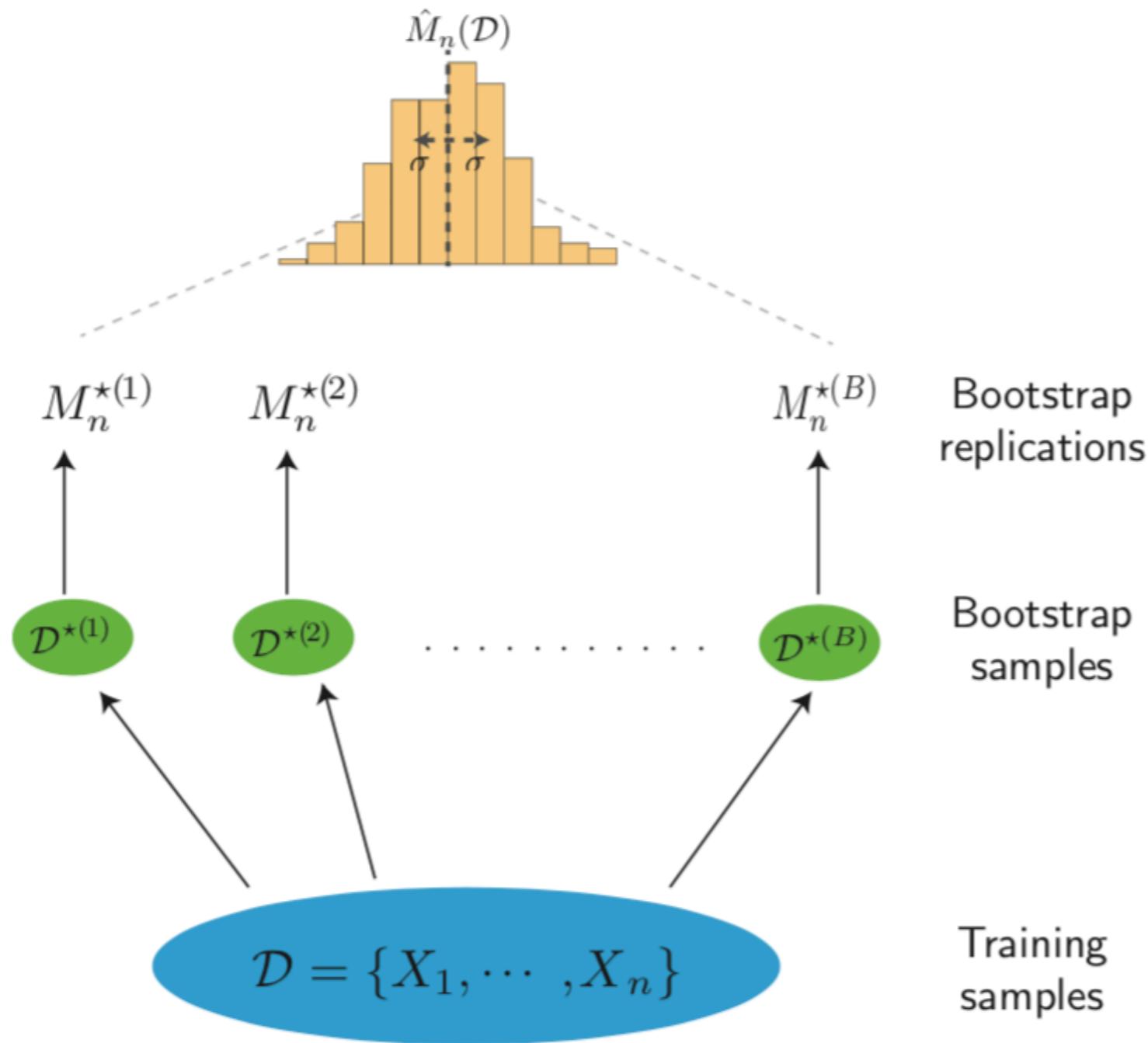
*bootstrap samples  
(with repeated elements)*

- With the **bootstrapped samples** we can construct statistical quantities of interest:

$$\widehat{\text{Var}}_B(M_n) = \frac{1}{B-1} \sum_{k=1}^B \left( M_n^{*(k)} - \bar{M}_n^* \right)^2 \quad \bar{M}_n^* = \frac{1}{B} \sum_{k=1}^B M_n^{*(k)}$$

it can be shown that in the large  $n$  limit the **bootstrap distributions** approximate well the **sampling distributions** from which the training dataset was obtained

# Bootstrapping



In ML applications, bootstrapping is frequently used to **assign measures of accuracy** (defined in terms of bias, variance, correlations etc to sample estimates

It can be shown that in the large  $n$  limit the **bootstrap distributions** approximate well the **sampling distributions** from which the training dataset was obtained

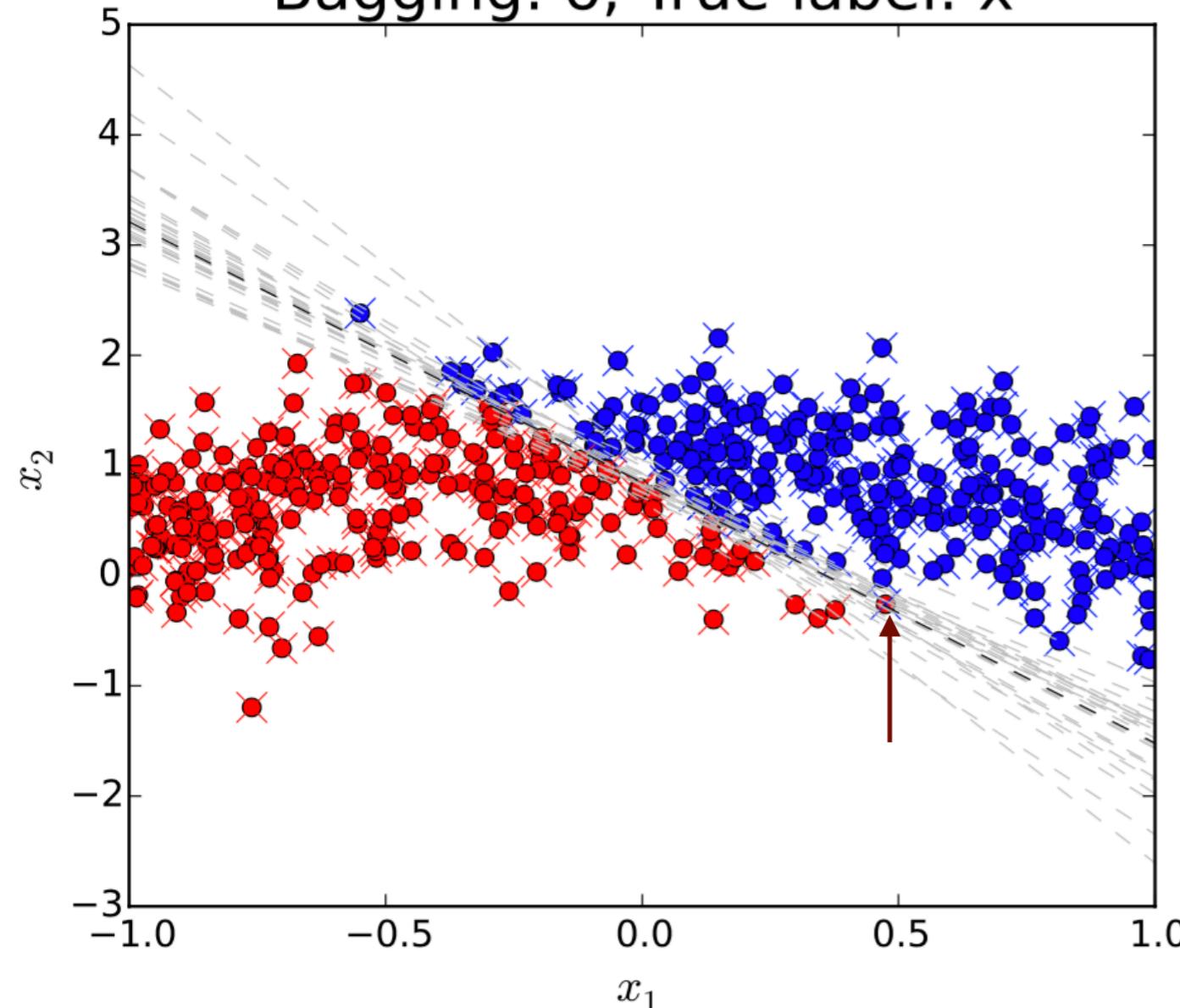
# Bagging with bootstrap

same as before, but now the datasets have been partitioned with bootstrapping

$$\hat{g}_{\mathcal{L}}^{\text{BS}}(\mathbf{x}) = \frac{1}{M} \sum_{i=1}^M g_{\mathcal{L}_i^{\text{BS}}}^A(\mathbf{x})$$

which can lead to a **variance reduction** to the price of an **increase in the bias**

Bagging: o, True label: x



ex: bagging with bootstrap (**2D classification**)

$n=500, B=25$  (50 points each)

grey dashed: bootstrap predictions

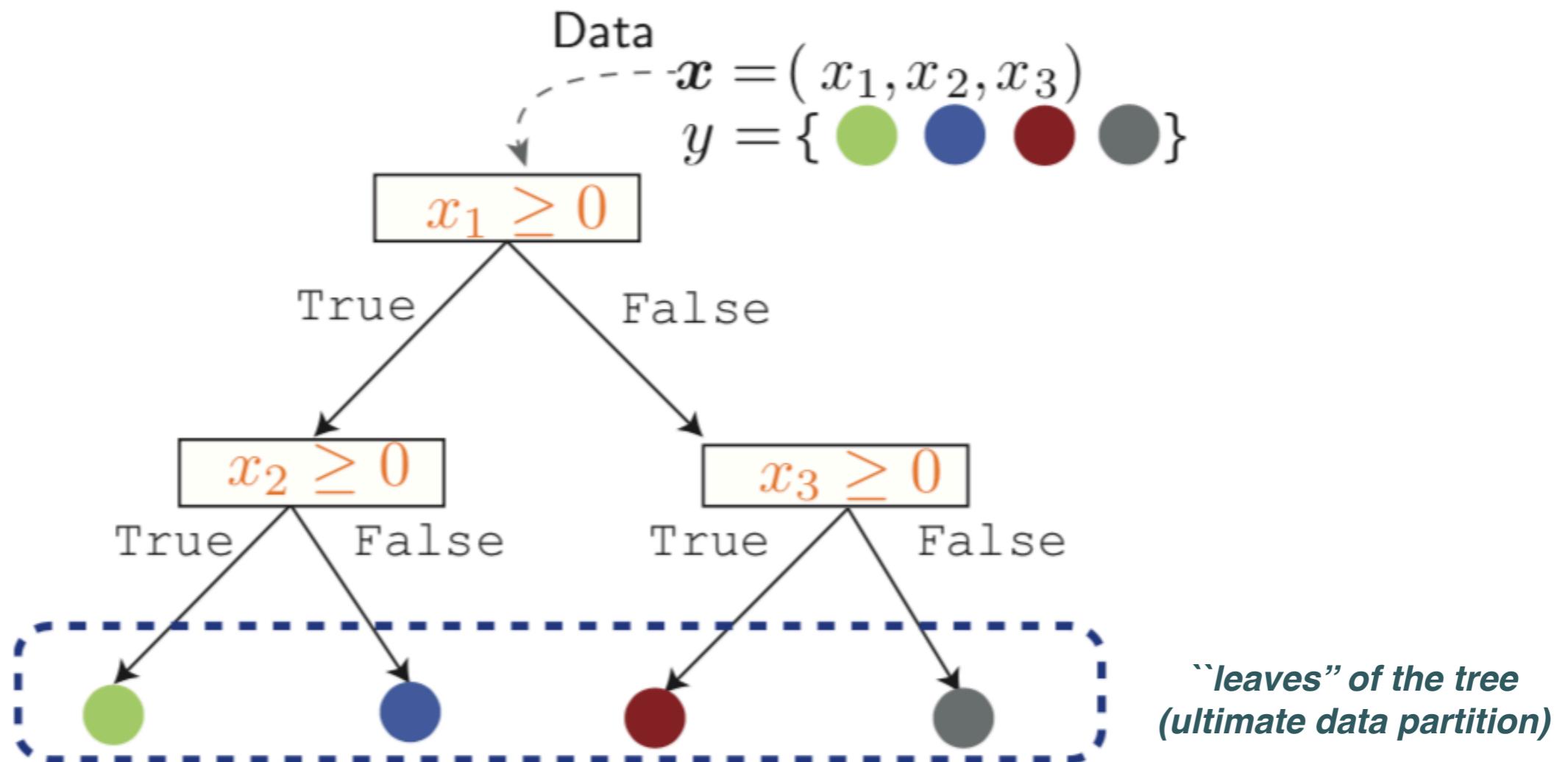
black dashed: bagging average

individual predictors poor, bagging much better!

bagging is specially useful for **unstable learning algorithms** where small changes in the training dataset result in large changes in the prediction

# Random Forests

Ensemble method widely used in complex classification tasks  
constructed from **randomised tree-based classifier decision trees**



In ML, a **decision tree** is an algorithm which uses a series of questions to hierarchically partition the data, where each branch of the tree splits the data into smaller subsets

# Random Forests

individual trees have often **high variance** and are weak classifiers:  
we can improve by incorporating them in an ensemble method

→ We need an ensemble of **randomised decision trees** (minimised correlations)

- 💡 (1) train each decision tree on a different bootstrapped dataset: **bagged decision tree**
- 💡 (2) use different random subset of features at each split: **random forest**

*reduces correlations between trees that arise  
when only few features are strongly predictive*

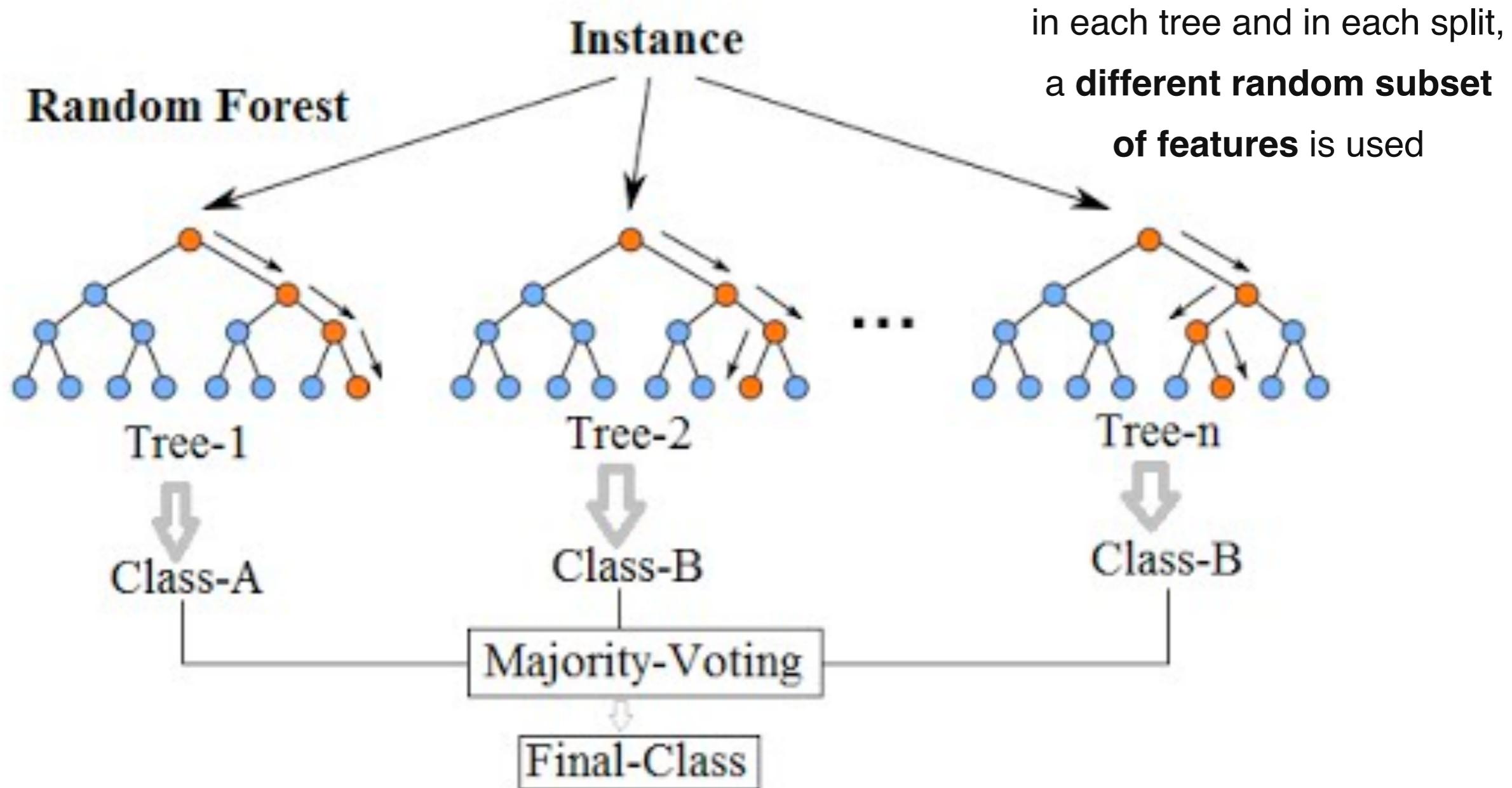
as other ML learning classifiers also Random Forests require some **regularisation**, for example a maximum depth of the tree, to control complexity and prevent overfitting

*typically, a classification problem with  $p$  features, in RFs only  $p^{1/2}$  features are used in each split*

Random forests have other attractive features, for example, they can be used to **rank the importance of variables** in a regression or classification problem in a natural way

# Random Forests

each decision tree in the ensemble is built upon a **random bootstrap sample** of the original data



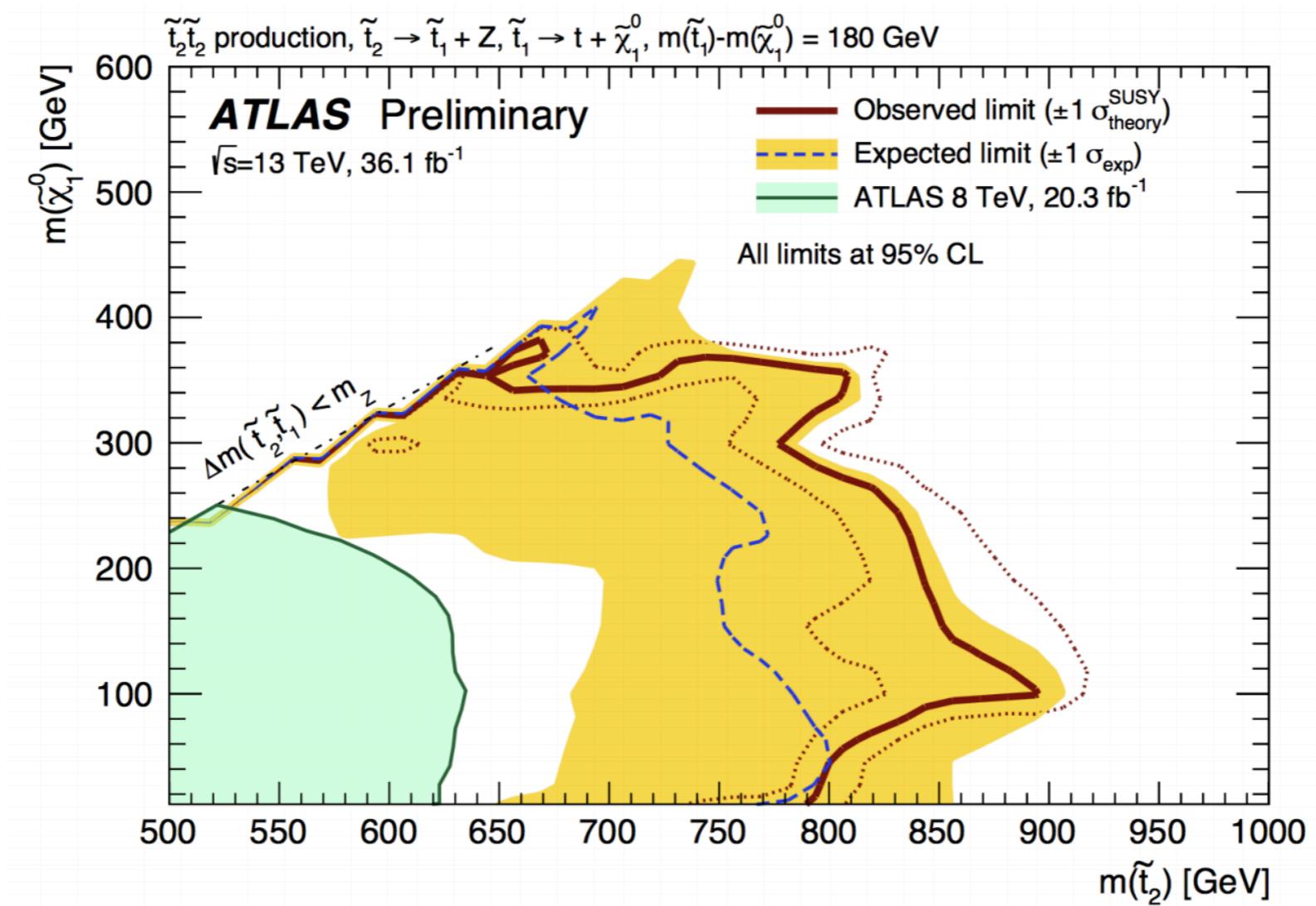
the final categorisation is assigned e.g. from **majority voting**

# Case Study: limit setting in high-dimensional models with decision trees

*S. Caron, J. S. Kim, K. Rolbiecki, R. Ruiz de Austri and B. Stienen,  
‘The BSM-AI project: SUSY-AI–generalizing LHC limits on supersymmetry with machine learning’  
Eur. Phys. J. C 77, no. 4, 257 (2017), [arXiv:1605.02797 [hep-ph]].*

# Harvesting LHC data for BSM signals

- In the absence of new particles and/or interactions, LHC searches for BSM physics are used to derive exclusion ranges for specific scenarios
- Results presented as excluded ranges in subset of the full parameter space

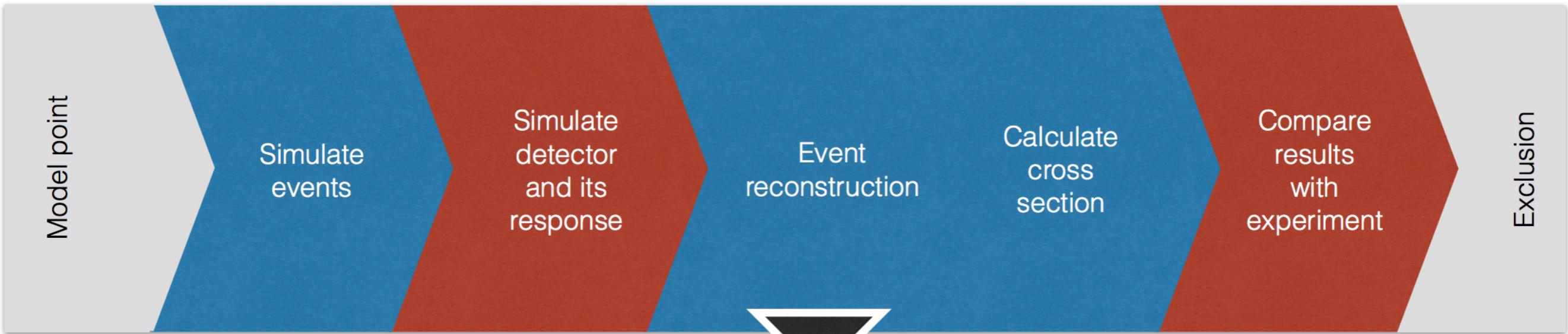


However this is only a **small part of the information contained by the LHC measurements**, ideally we would like the exclusion ranges in the full parameter space of the theory: e.g. **19 params in the pMSSM**

# Harvesting LHC data for BSM signals

**Exploring the full parameter space** of a given BSM scenario is in general CPU time consuming

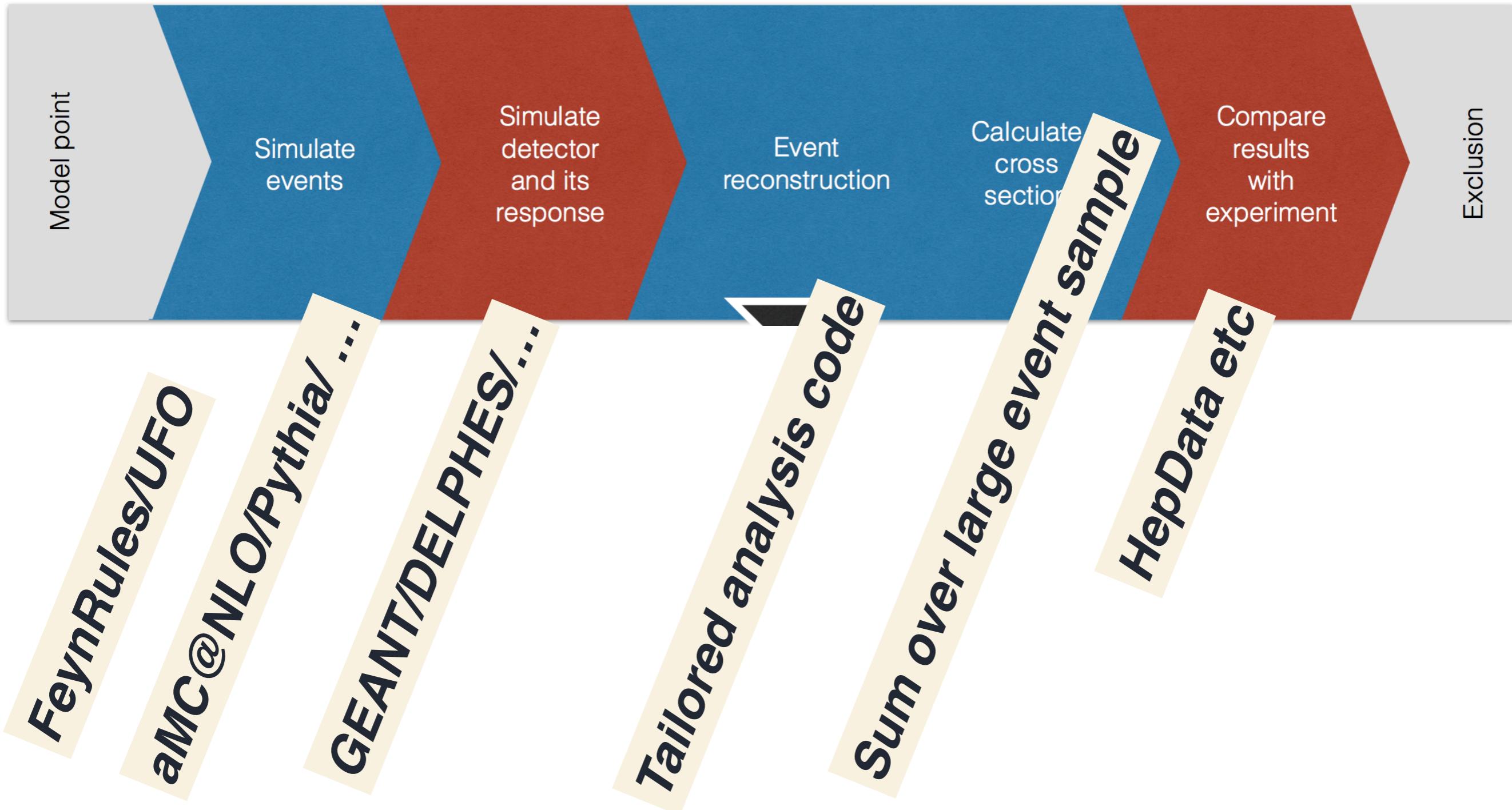
Time = O(hours)



# Harvesting LHC data for BSM signals

Exploring the full parameter space of a given BSM scenario is in general CPU time consuming

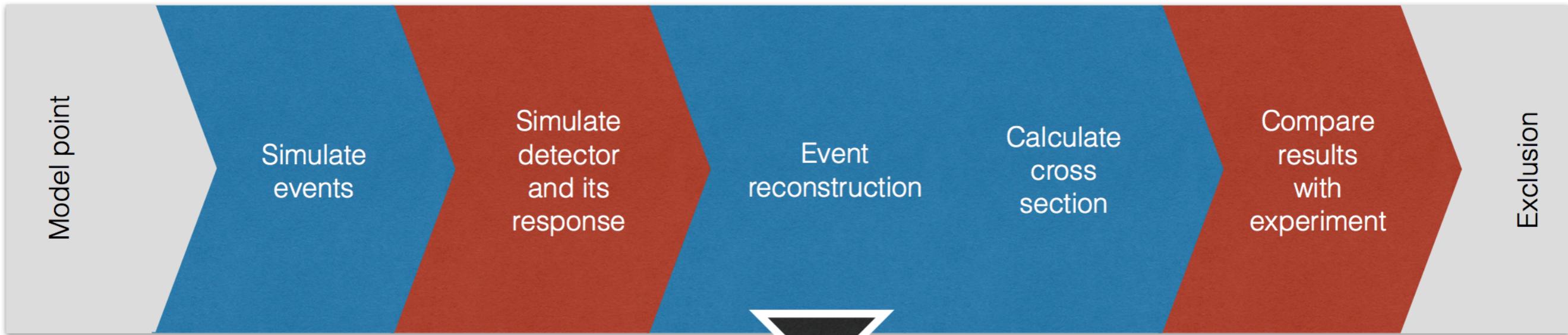
Time = O(hours)



# Harvesting LHC data for BSM signals

Exploring the full parameter space of a given BSM scenario is in general CPU time consuming

Time =  $O(\text{hours})$



Model point

Exclusion

Machine Learning

Exclusion

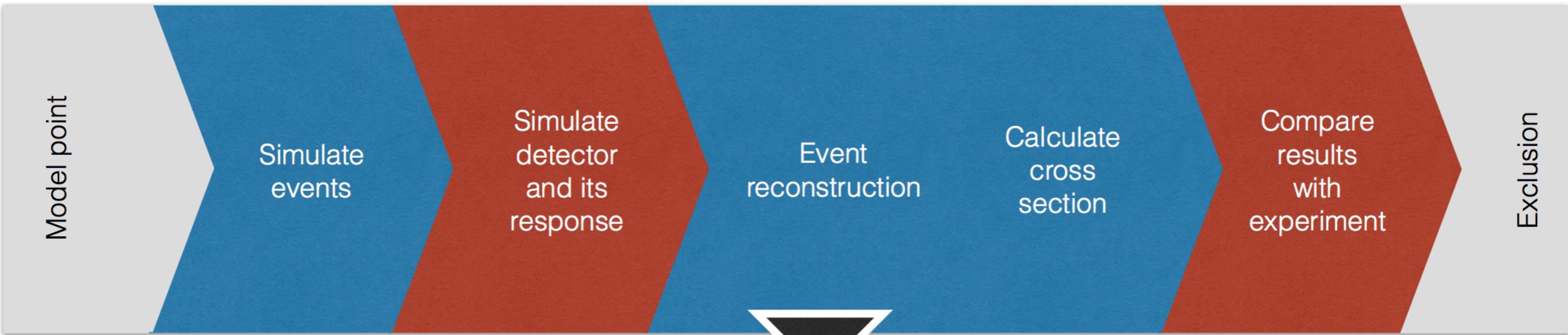
Time =  $O(\text{ms})$

Use ML tools one can **speed-up the limit-setting procedure by orders of magnitude**, making possible an efficient exploration of the full parameter space

# Harvesting LHC data for BSM signals

Exploring the full parameter space of a given BSM scenario is in general CPU time consuming

Time =  $O(\text{hours})$



Model point

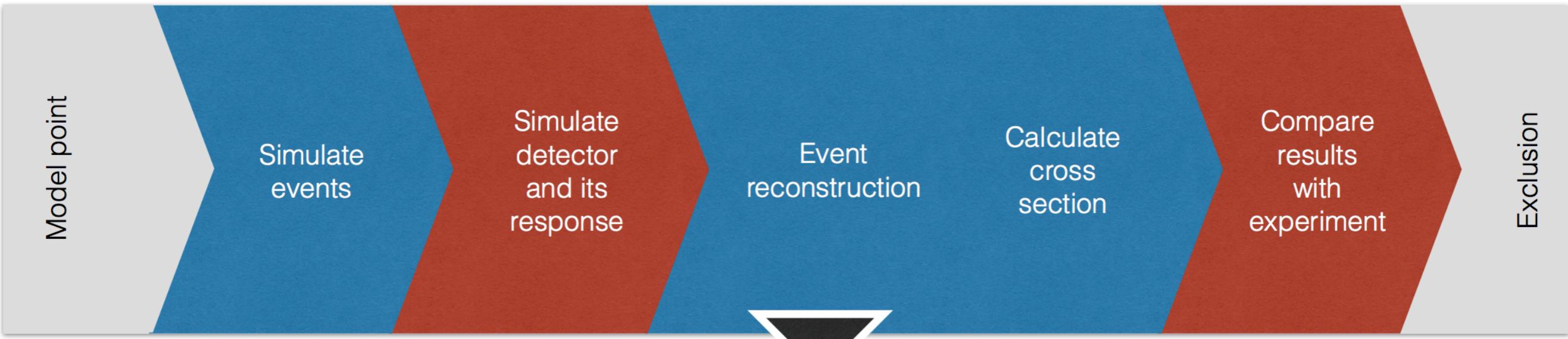
Time =  $O(\text{ms})$

Use ML tools one can **speed-up the limit-setting procedure by orders of magnitude**, making possible an efficient exploration of the full parameter space

# Harvesting LHC data for BSM signals

Exploring the full parameter space of a given BSM scenario is in general CPU time consuming

Time =  $O(\text{hours})$



Model point

Exclusion

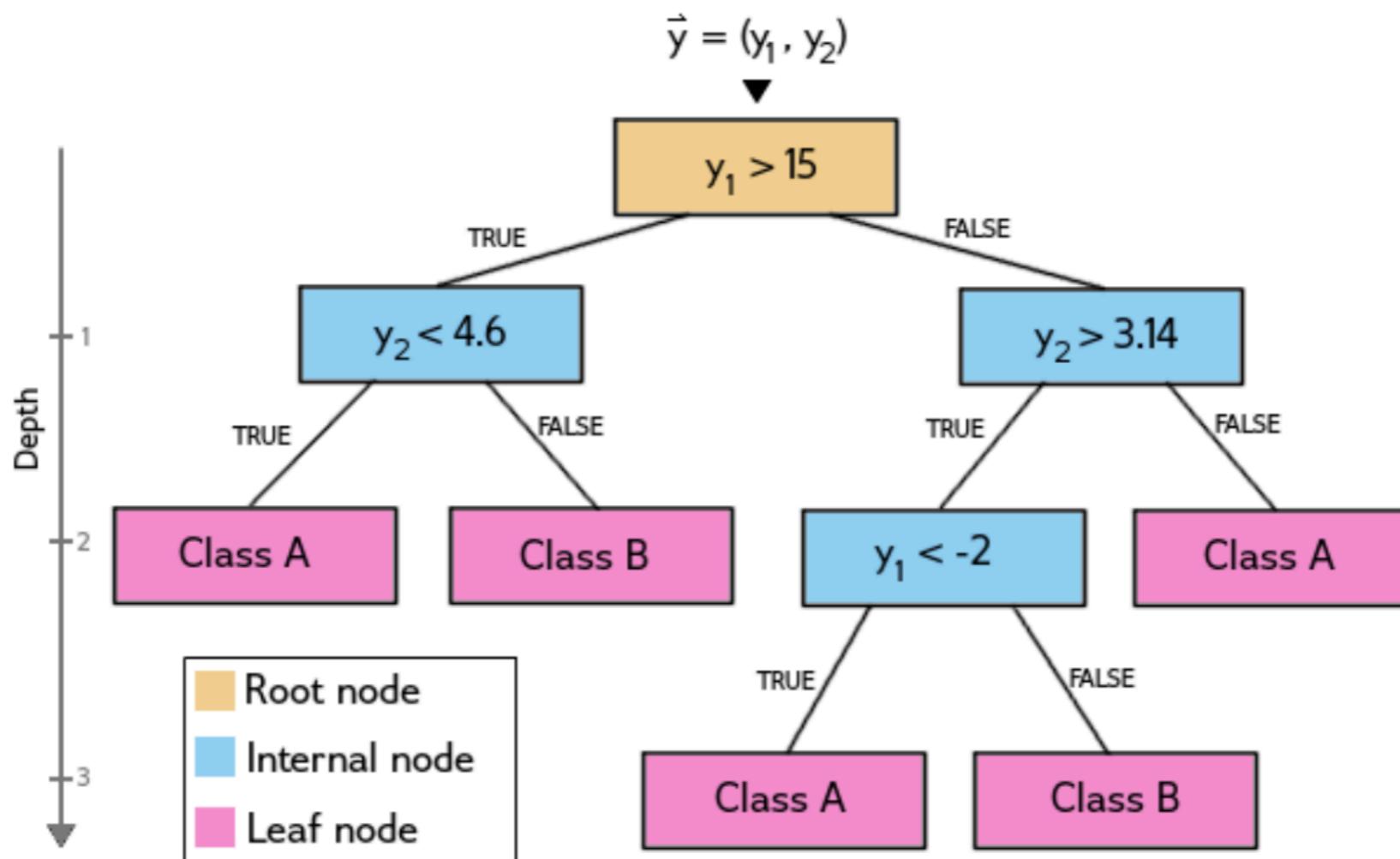
explore the ability of ML algorithms to **generalise from examples**  
without any need to be aware of the underlying physics

Time =  $O(\text{ms})$

Use ML tools one can **speed-up the limit-setting procedure by orders of magnitude**, making possible an efficient exploration of the full parameter space

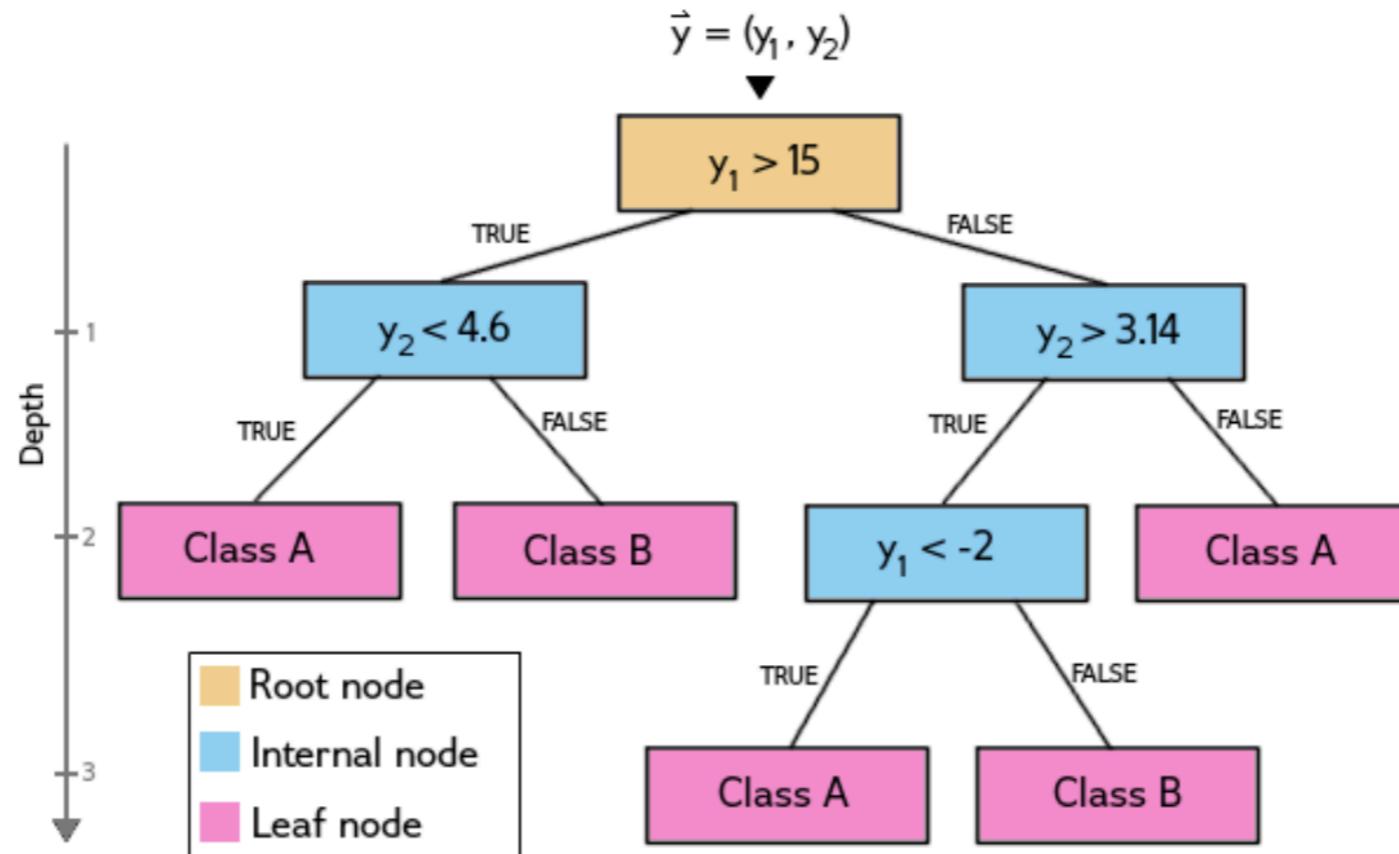
# Classification with Decision Trees

- This is an example of a discrete Classification problem: a given point in bSM parameter space can be either **allowed** or **excluded**, with no options in between
- Decision Trees classifiers, such as **Random Forest classifier**, exhibit good performance here



Procedure starts by presenting parameter sets and class labels, to **learn the patterns that the input data follow**

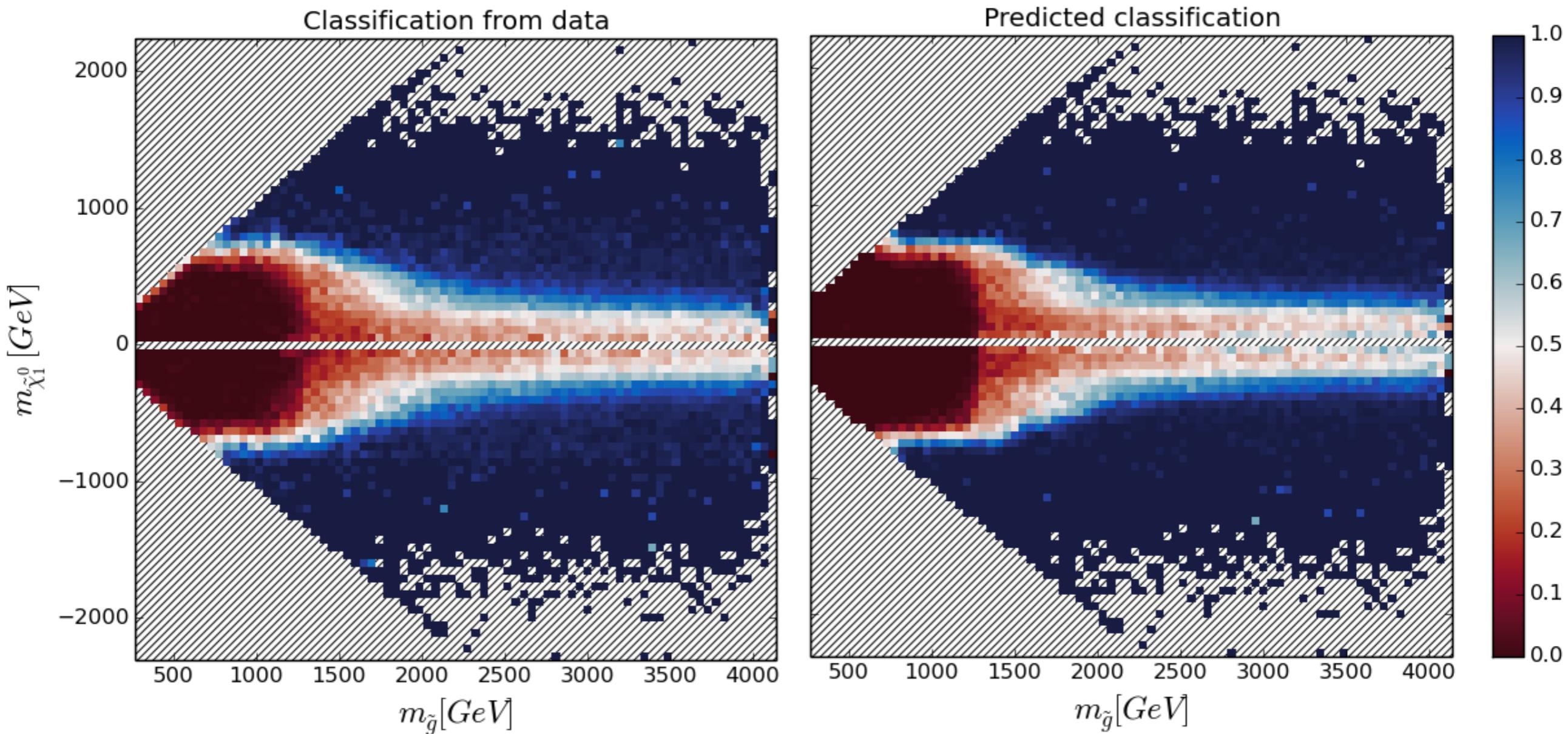
# Classification with Decision Trees



- Each node in tree specifies a **test performed on the arriving attribute**
- The result of this test determines to which node the attribute set is sent next.
- Process is repeated until the **final leaf node is reached**,
- At final node a **class label is assigned to the set**, specifying its class
- Tree works on the **entire parameter space**: every test performed interpreted as a cut in this space.
- The **parameter space is split into disjunct regions**, each having borders defined by the cuts in the root and internal nodes, and a classification defined by a leaf node.

# Harvesting LHC data for BSM signals

Compare classification (allowed vs excluded) in **real data** vs the **ML-trained classifier**



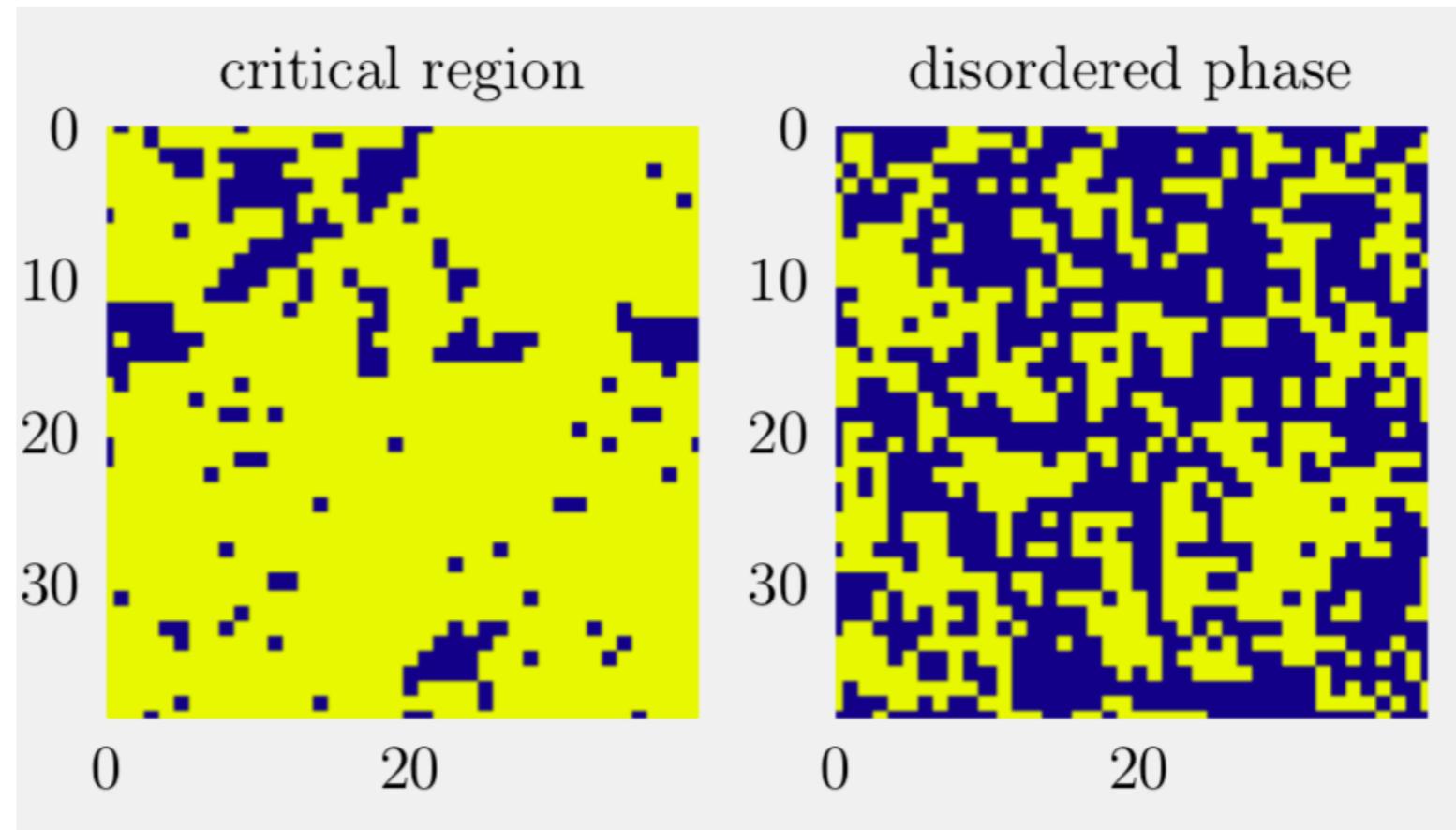
- Efficient exploration of the **full bSM parameter space**
- Can be **projected** in any of the dimensions of the 19-parameter space of the pMSSM
- Generalise to points outside training dataset in  $O(ms)$  as opposed to  $O(h)$

# Tutorial 3 Exercise 3b

starting point is the **Python script** that you will find in

<https://github.com/juanrojochacon/ml-ditp-attp/blob/master/Tutorials/Tutorial3/>

- ✿ Construct a **classifier based on Random Forests** and test it on the 2D Ising dataset
- ✿ What are advantages and disadvantages as compared to a classifier based on **logistic regression**?
- ✿ Investigate different methods to **visualise the results**, including with principal component analysis



# Reinforcement Learning

with input from *An overview of Reinforcement Learning and Multi-Agent Systems*,  
Alberto Guffanti, NNPDF meeting August 2019, <https://indico.cern.ch/event/825705/>

# Reinforcement Learning

So far we have considered **two main paradigms** in Machine Learning problems

**Supervised Learning:** starting from a training dataset with **labelled examples**,  $\{x_i, y_i\}_{i=1,N}$ , produce a **model  $f(x)$**  that predicts and generalises the info in the training sample. The labels  $y_i$  can be continuous (underlying law is function) or discrete (classification)

**Unsupervised Learning:** starting from a training dataset with **unlabelled examples**,  $\{x_i\}_{i=1,N}$ , produce a **model** that takes a sample as input and as output produces the solution of a practical problem, such as **clustering**, **dimensional reduction**, or **outlier detection**

# Reinforcement Learning

So far we have considered **two main paradigms** in Machine Learning problems

**Supervised Learning:** starting from a training dataset with **labelled examples**,  $\{x_i, y_i\}_{i=1,N}$ , produce a **model  $f(x)$**  that predicts and generalises the info in the training sample. The labels  $y_i$  can be continuous (underlying law is function) or discrete (classification)

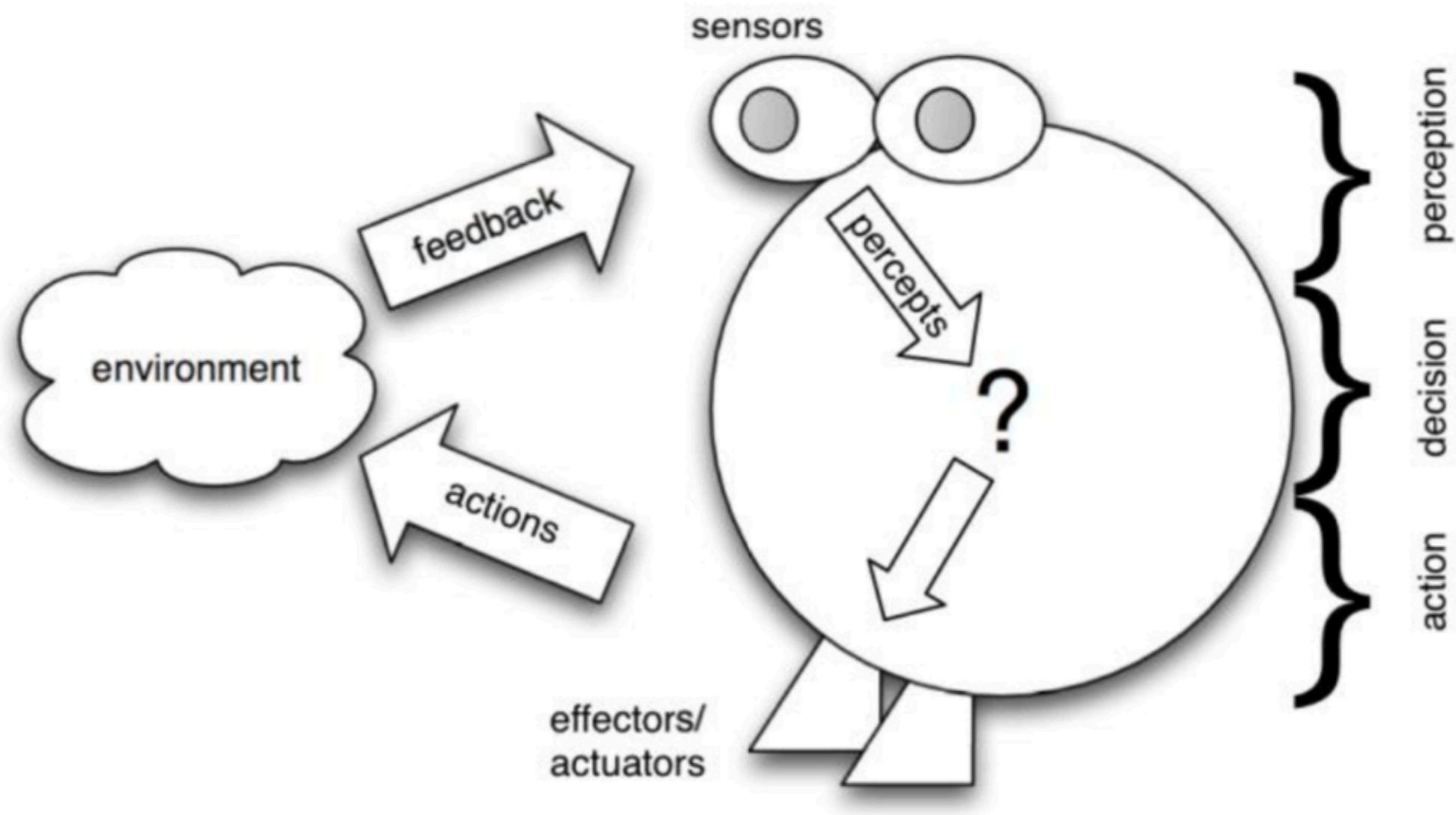
**Unsupervised Learning:** starting from a training dataset with **unlabelled examples**,  $\{x_i\}_{i=1,N}$ , produce a **model** that takes a sample as input and as output produces the solution of a practical problem, such as **clustering**, **dimensional reduction**, or **outlier detection**

now we want to discuss a **third ML paradigm**

**Reinforcement Learning:** given a complex task in a complex environment (dynamic, non deterministic, only partly accessible) train an **agent** that carry out **autonomous action** in this environment and complete the requested task

# Agents in Reinforcement Learning

In the context of **Reinforcement Learning**, an **agent** is a computer system capable **autonomous action** in some environment, in order to achieve its delegated goals



# Agents in Reinforcement Learning

In the context of **Reinforcement Learning**, an **agent** is a computer system capable **autonomous action** in some environment, in order to achieve its delegated goals



*trivial agents: thermostat, e-mail daemons, ....*

# Agents in Reinforcement Learning

In the context of **Reinforcement Learning**, an **agent** is a computer system capable **autonomous action** in some environment, in order to achieve its delegated goals

*non-trivial agents should exhibit the following properties:*

- ✿ **Reactive:** interact with environment and react its changes
- ✿ **Proactive:** recognise opportunities and take initiative
- ✿ **Social:** cooperate with other agents (and humans!) via cooperation, negotiation, coordination
- ✿ **Rational:** the agent will always act to fulfil its goals
- ✿ **Adaptability:** the agent is able to improve its performance over time

# Agents in Reinforcement Learning

**Environments in RL** can exhibit the following features:

- ➊ **Accessible or Inaccessible:** can the agent obtain updated and accurate information about the state of the environment?
- ➋ **Deterministic or non-deterministic :** has each action that the agent perform always associated the same effect?
- ➌ **Static vs dynamics:** is the environment stable expect for the action of the agent?
- ➍ **Discrete vs continuous:** are there a finite or infinite number of actions possible?

# A Reinforcement Learning system

The ultimate goal of **Reinforcement Learning** is to

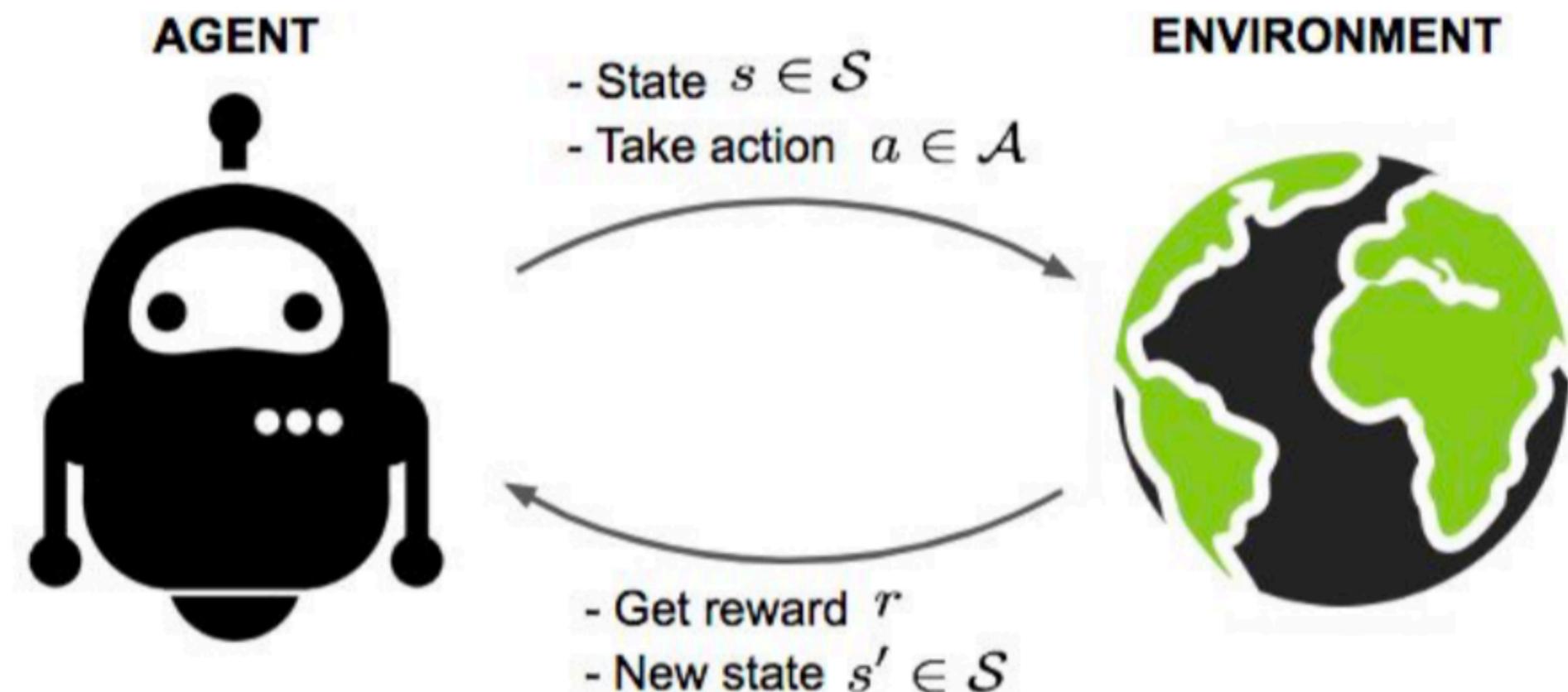
design an agent that **performs complex tasks** and **takes autonomous action** to fulfil its design goals, in an environment that is: partly inaccessible, non-deterministic, non-episodic, dynamic and continuous (*i.e.* the real world!).

- The agent receives the state of the environment as a **vector of features** (inputs)
- The agent can execute actions in every state, with different actions bringing **different rewards**
- Goal: **learn a policy**, *i.e.* a function that maps the features of an state vector to an optimal action to be taken in that stage
- An action is optimal if it **maximizes the expected average reward**
- In RL **decision making is sequential and the goal is long-term** (*i.e.* game playing, robotics, resource management, ...)

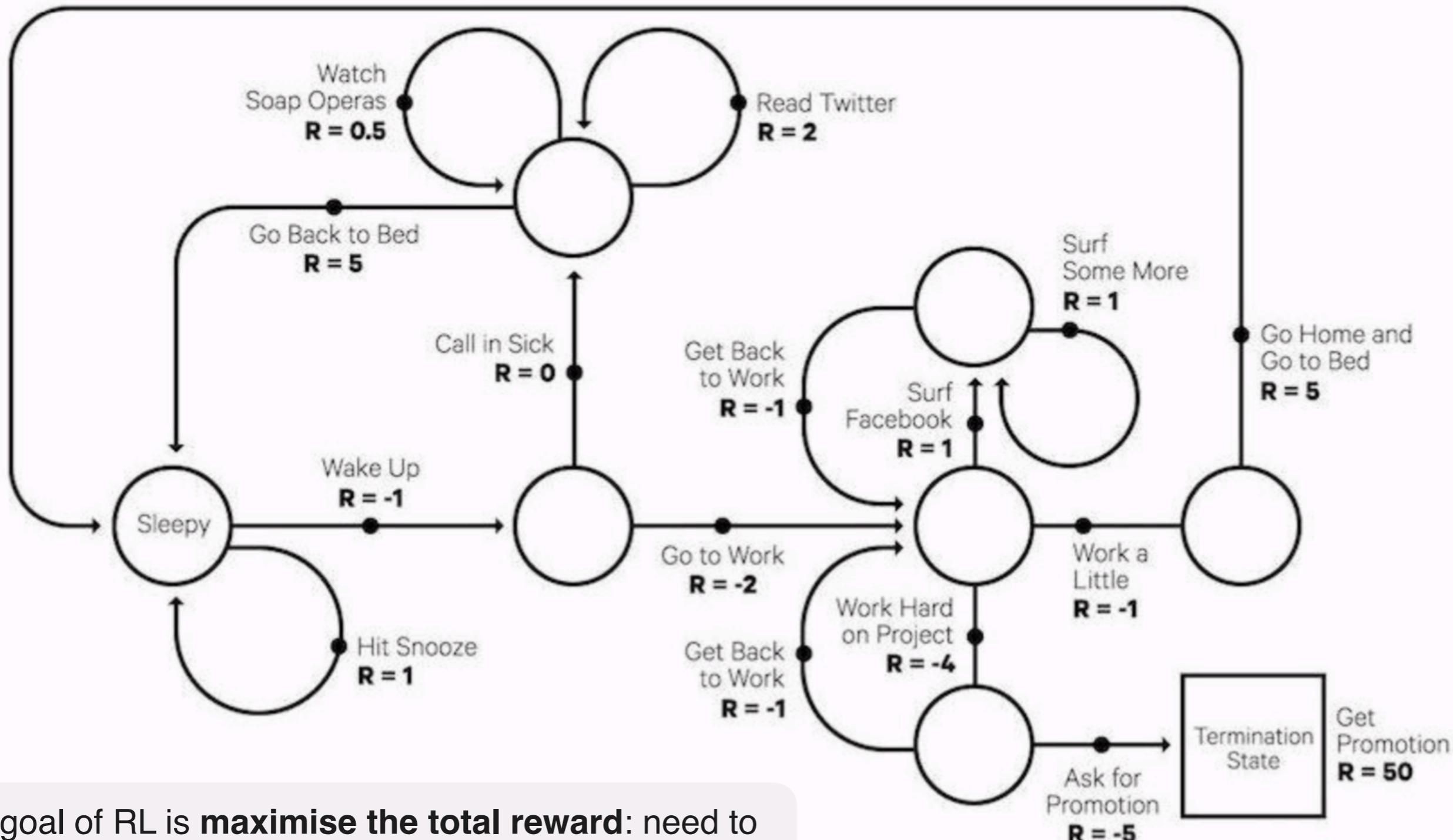
# Agents in Reinforcement Learning

The ultimate goal of **Reinforcement Learning** is to

design an agent that **performs complex tasks** and **takes autonomous action** to fulfil its design goals, in an environment that is: partly inaccessible, non-deterministic, non-episodic, dynamic and continuous (*i.e.* the real world!).



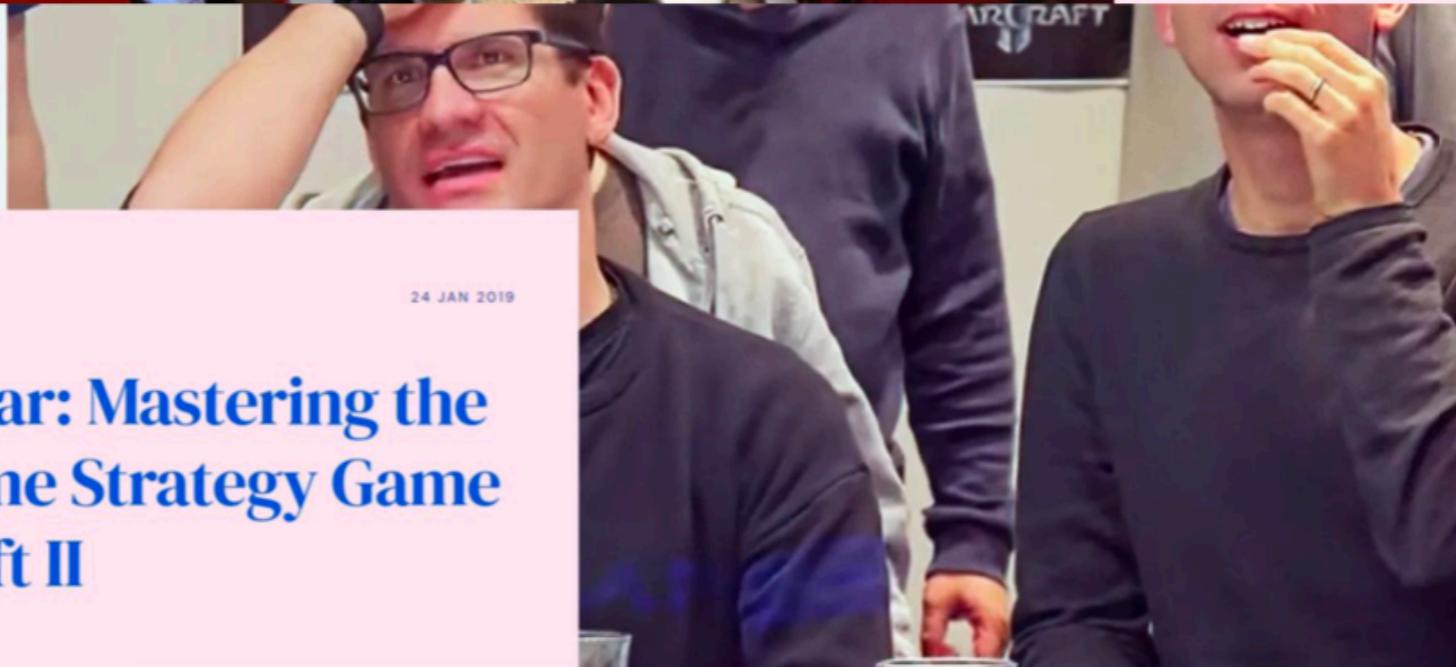
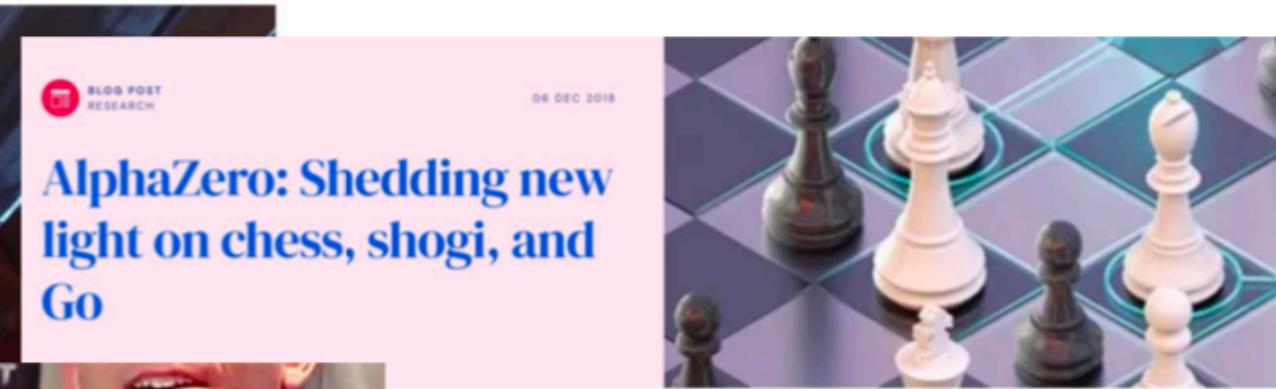
# A Reinforcement Learning system



the goal of RL is **maximise the total reward**: need to explore all possible options to determine the best policy for each action that it might need to carry

# Reinforcement Learning for Games

AlphaGo: using machine learning to master the ancient game of Go



In late 2017 we [introduced AlphaZero](#), a single system that taught itself from scratch how to master the games of chess, [shogi](#) (Japanese chess), and [Go](#), beating a world-champion program in each case. We were excited by the preliminary results and thrilled to see the response from members of the chess community, who saw in AlphaZero's games a ground-breaking, highly dynamic and "[unconventional](#)" style of play that differed from any chess playing engine that came before it.

# Reinforcement Learning for Games



REINFORCEMENT LEARNING DEMO

# Q-learning

Q-learning is a **model-free reinforcement learning algorithm**, which aims to learn a **policy** about what actions should the agent carry out for different circumstances

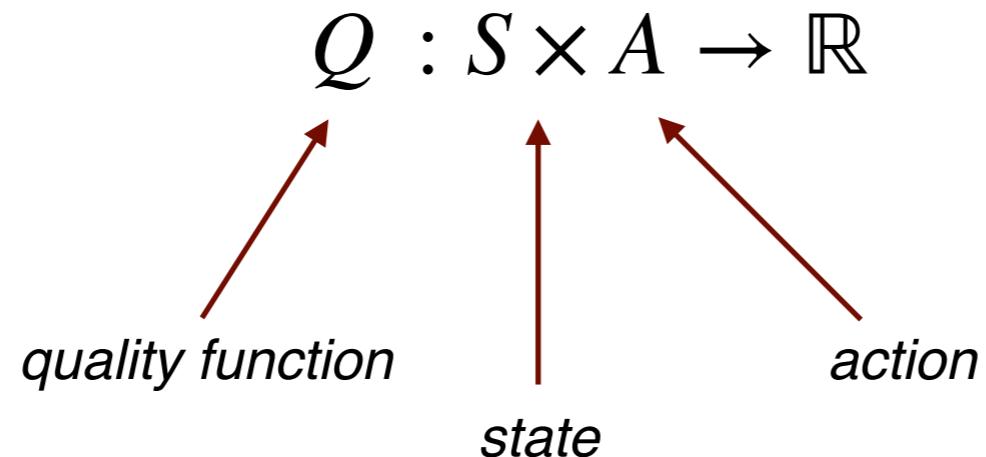
*no model of the environment need: the agent learns to maximise its future reward by repeatedly interacting with the environment*

In Q-learning, the weight for a step into the future is calculated by the **discount factor**

$$0 \leq \gamma^{\Delta t} \leq 1$$

*earlier rewards valued higher than later ones*

analog of cost function in Supervised Learning is the **quality of state-action combination**



the goal of Q-learning is to determine the actions that the agent should take for each state in order to **maximise the total reward**

# Q-learning

Schematically, at **each iteration of the Q-learning algorithm** the following steps take place:

- The agent selects an **action  $a_t$**
- As a consequence of this action, the agent observes a **reward  $r_t$**
- The agent then enters into a **new state  $s_t$**
- The quality function (cost function)  **$Q$**  is updated

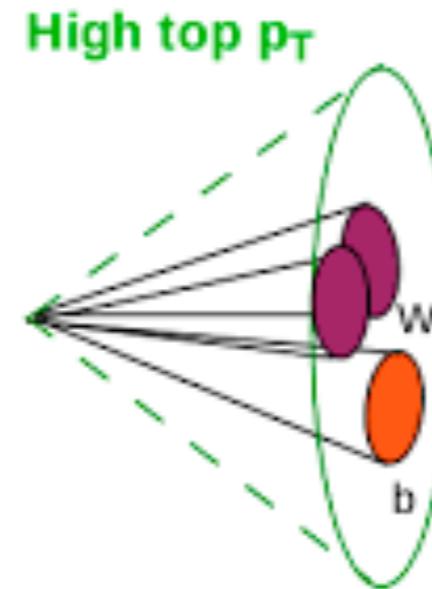
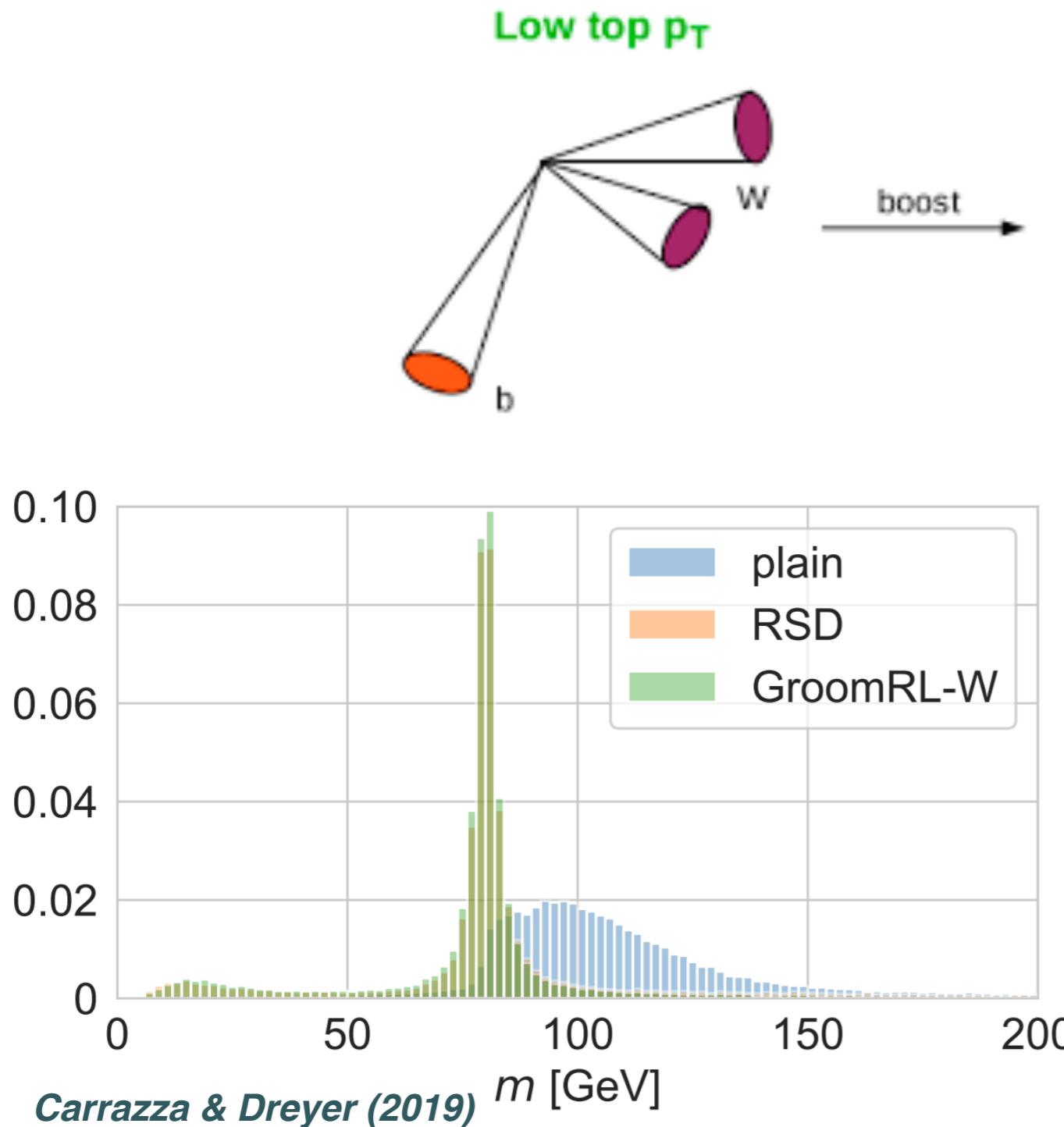
$$Q^{\text{new}}(s_t, a_t) \leftarrow (1 - \alpha) \times Q^{\text{old}}(s_t, a_t) + \alpha \left( r_t + \gamma \times \max_a Q(s_{t+1}, a) \right)$$

The diagram illustrates the Q-learning update rule. It shows the formula  $Q^{\text{new}}(s_t, a_t) \leftarrow (1 - \alpha) \times Q^{\text{old}}(s_t, a_t) + \alpha \left( r_t + \gamma \times \max_a Q(s_{t+1}, a) \right)$ . Red arrows with labels indicate the components: 'learning rate' points to the term  $(1 - \alpha)$ ; 'discount factor' points to the term  $\gamma$ ; and two arrows labeled '(estimate of) future optimal value' point to the term  $\max_a Q(s_{t+1}, a)$ .

After training, the agent has a policy  **$Q$**  which tells it how to act for each circumstance

# Reinforcement learning for jets

jet substructure allows separating interesting jets (collimated hadrons in high energy collisions) from background processes e.g. jets from top quark decay vs QCD jets



use RL to train an agent to ``groom'' jets  
removing background contamination: tune  
the **reward** to the **physics requirements**

*bypasses need for algorithmic  
implementation of a grooming algorithm*