



SAPIENZA  
UNIVERSITÀ DI ROMA

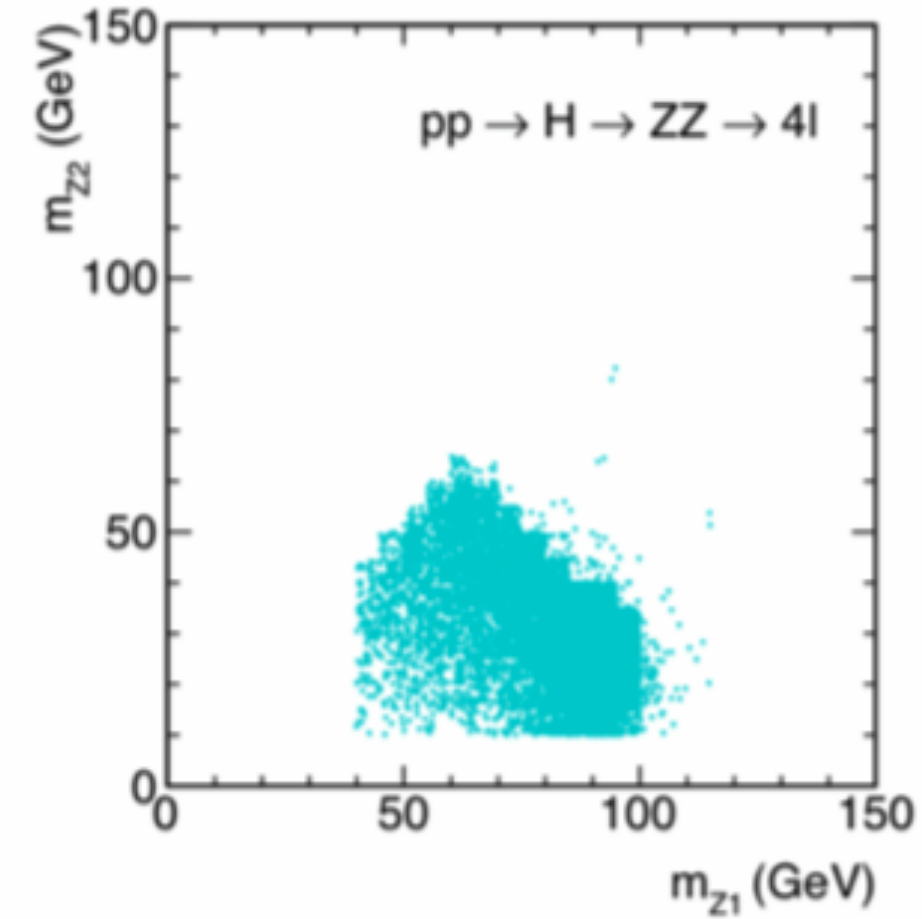
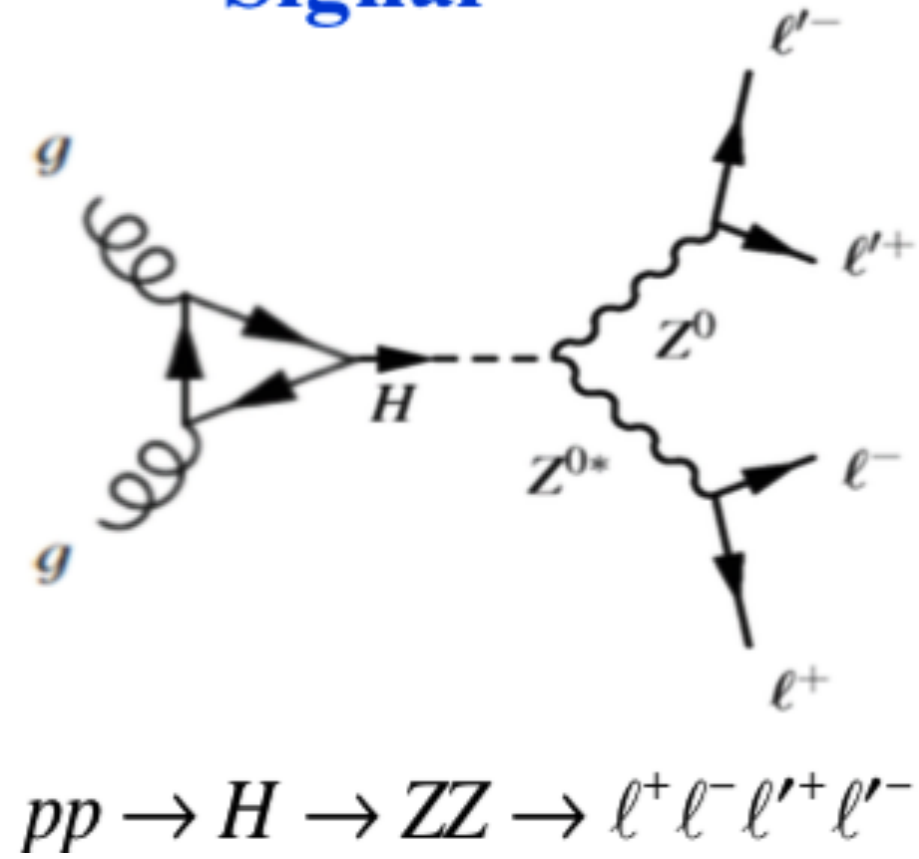
# METODI DI INTELLIGENZA ARTIFICIALE E MACHINE LEARNING IN FISICA

S. Giagu - AA 2019/2020

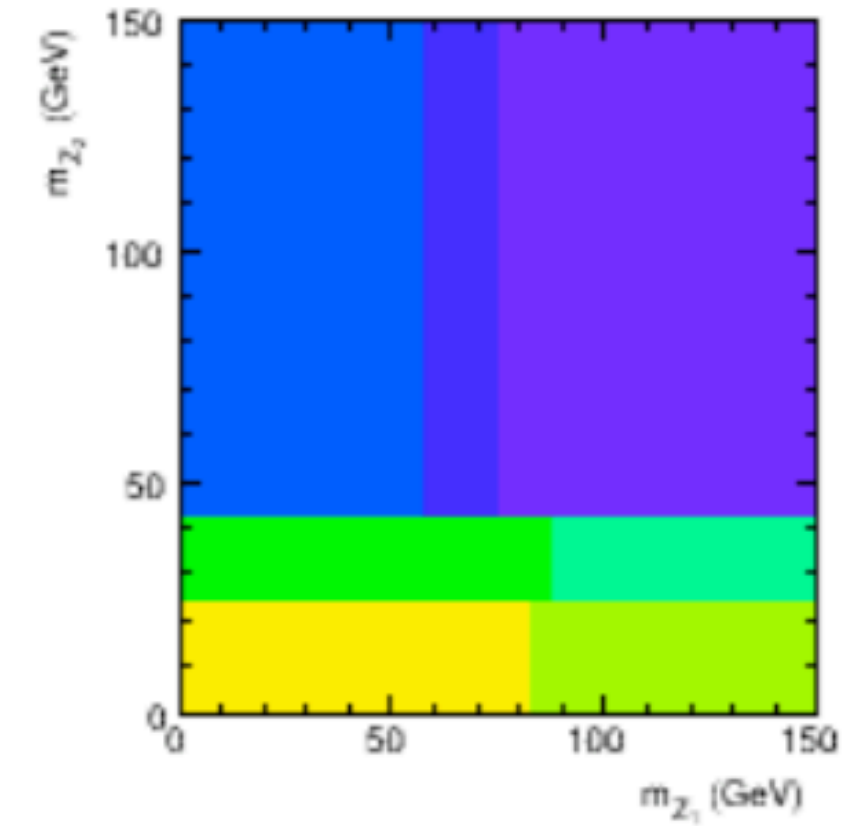
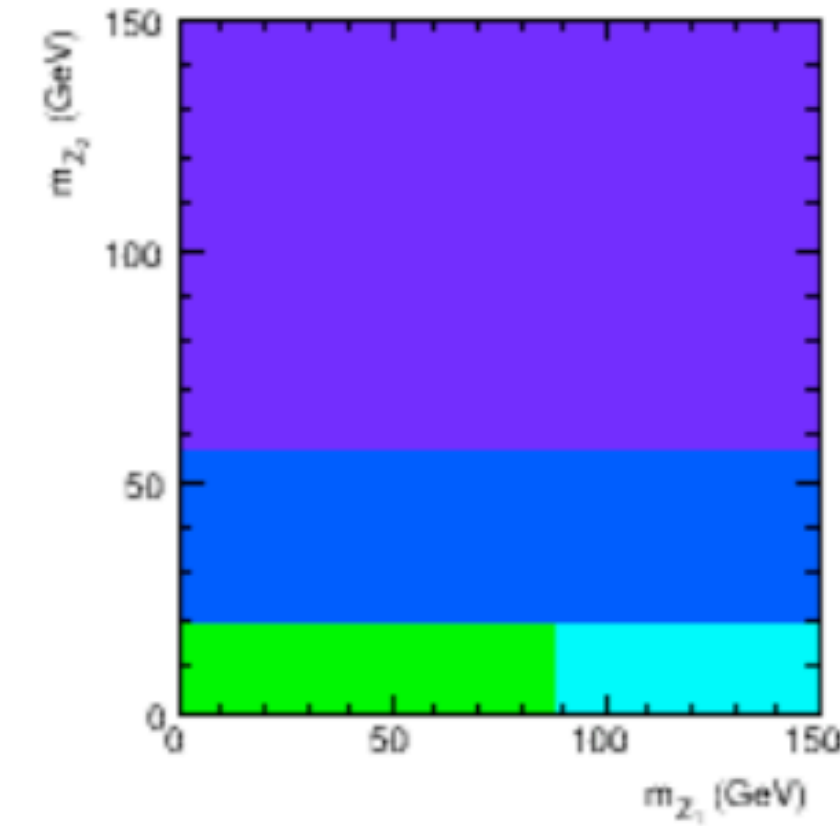
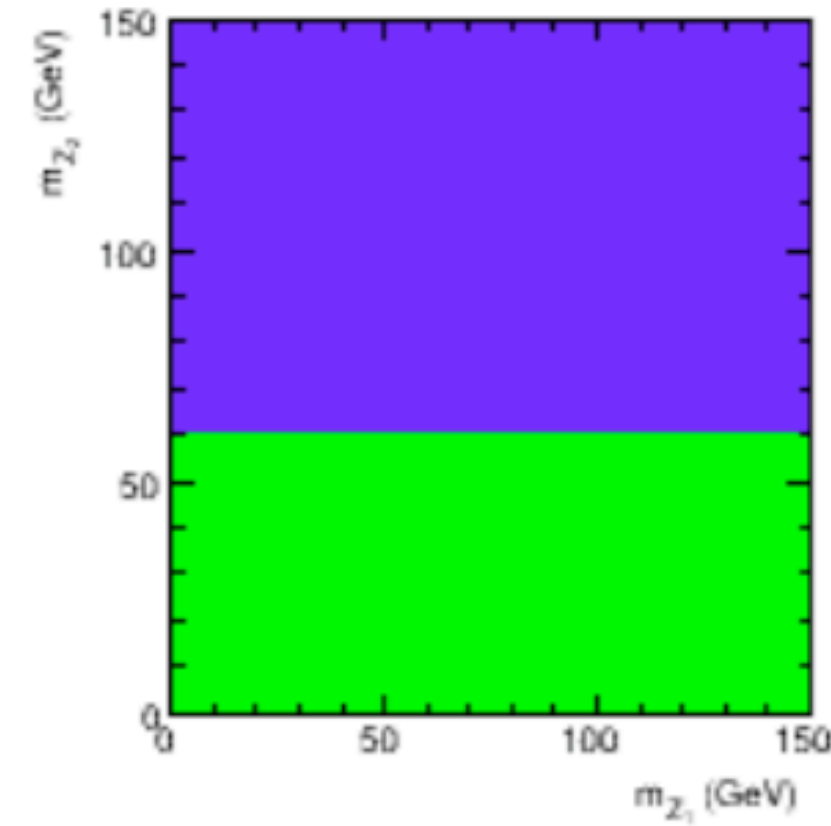
Lezione 10: 26.3.2020

# BDT: ESEMPIO HEP

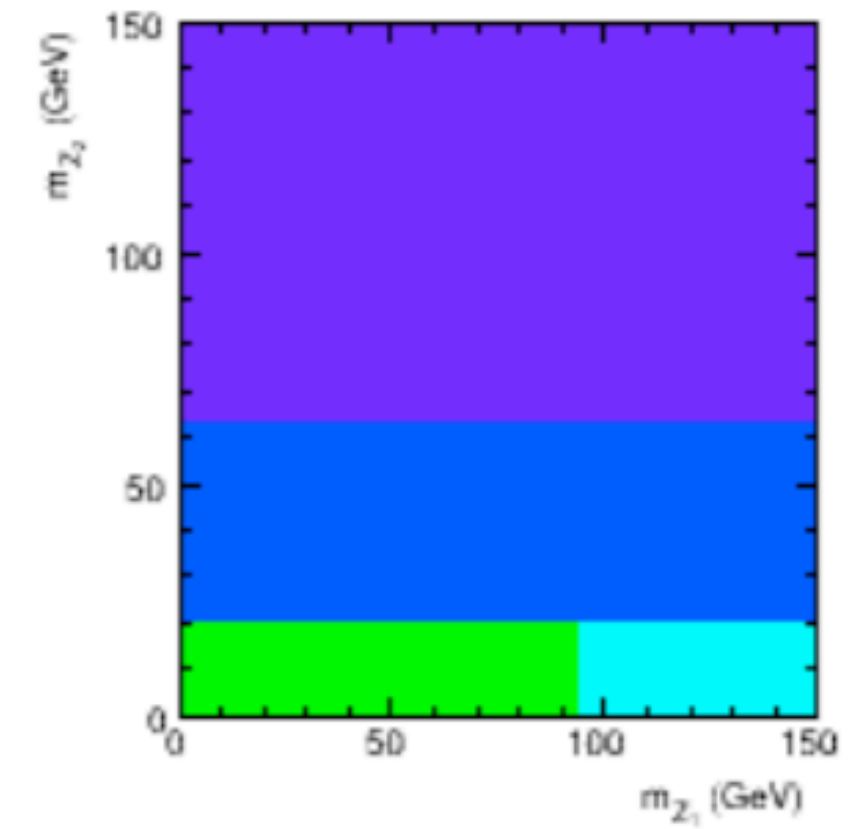
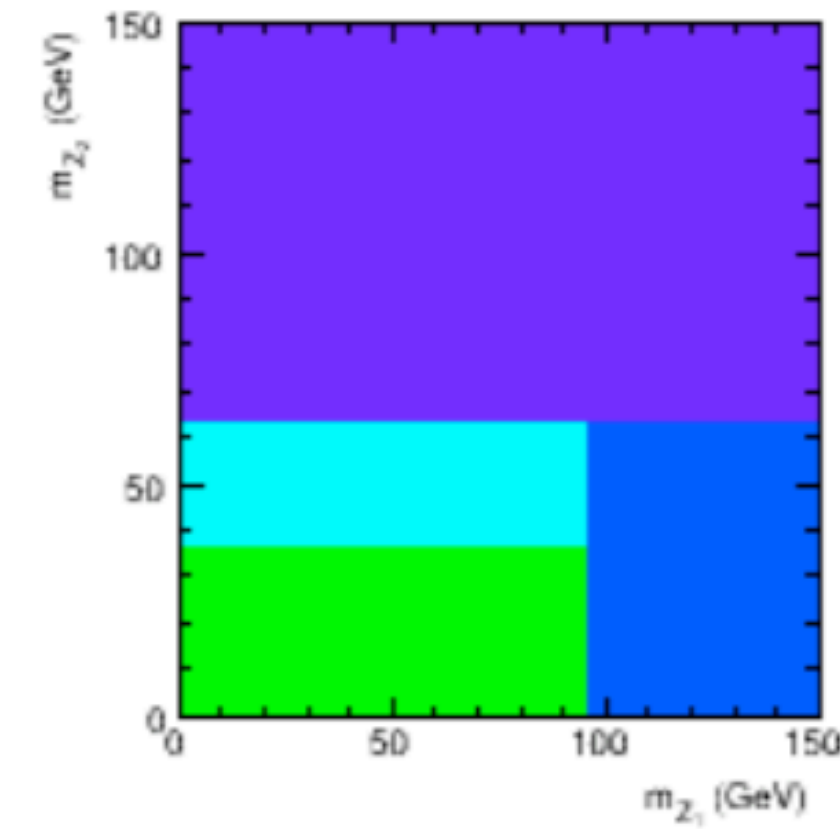
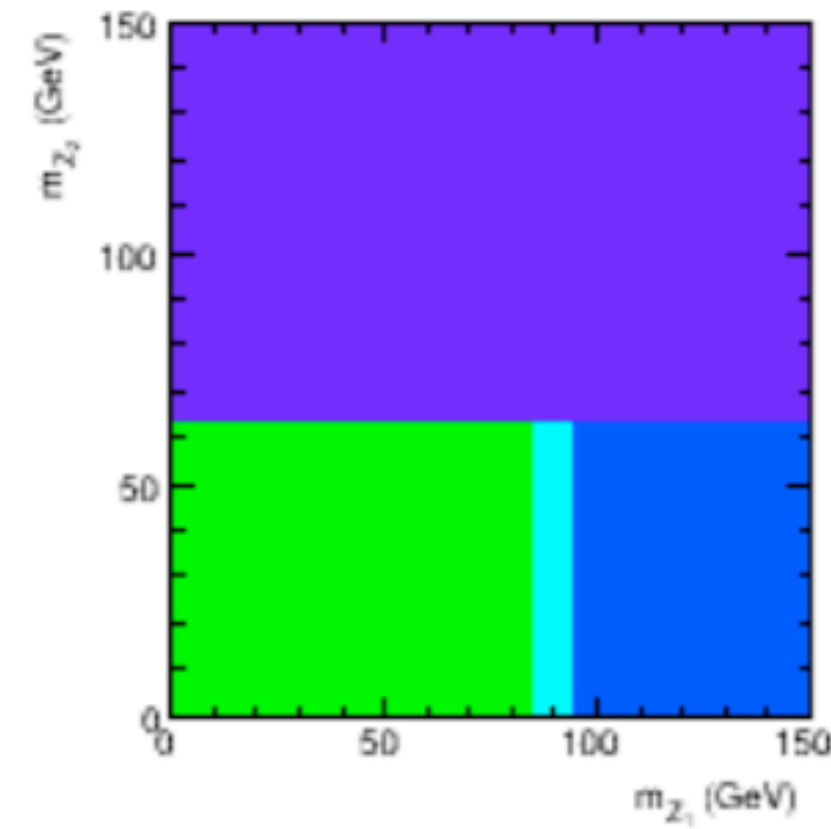
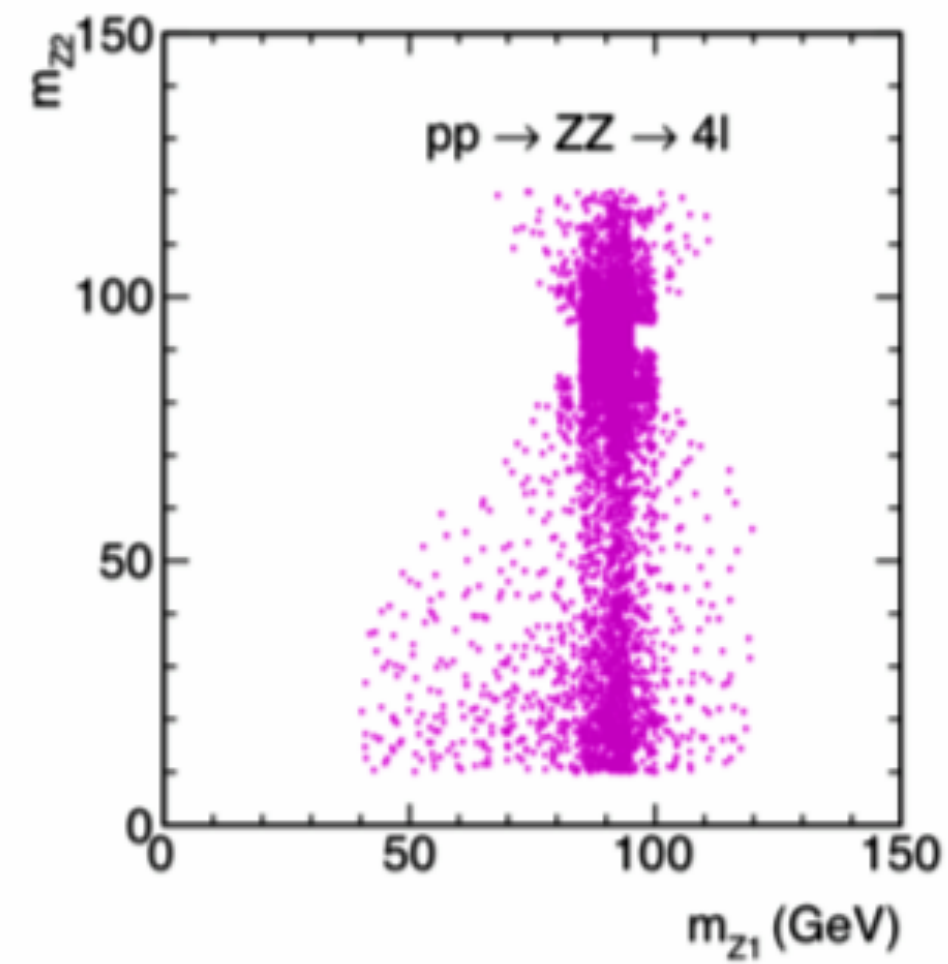
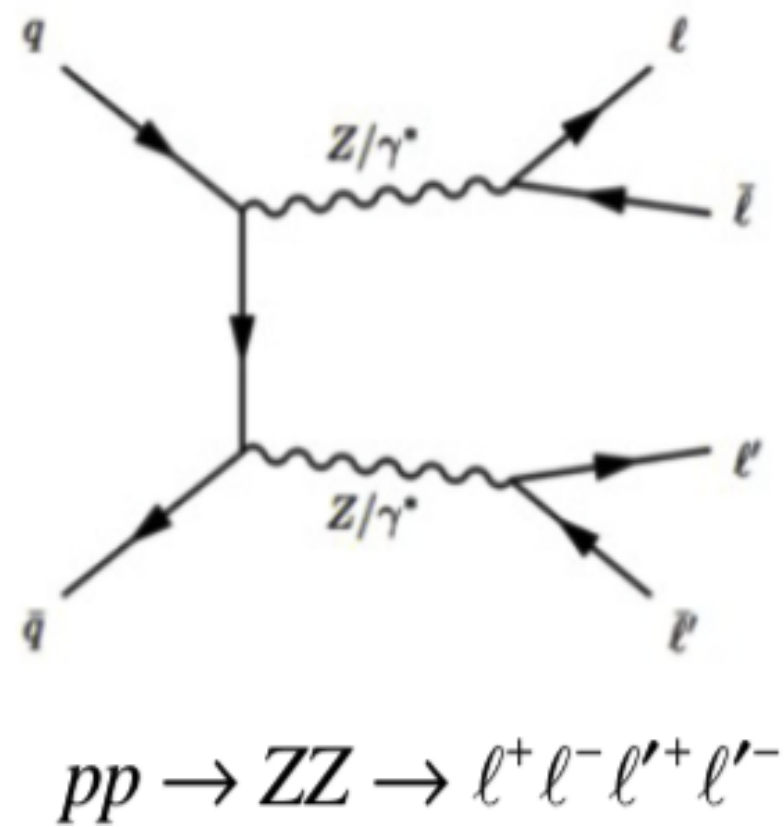
Signal



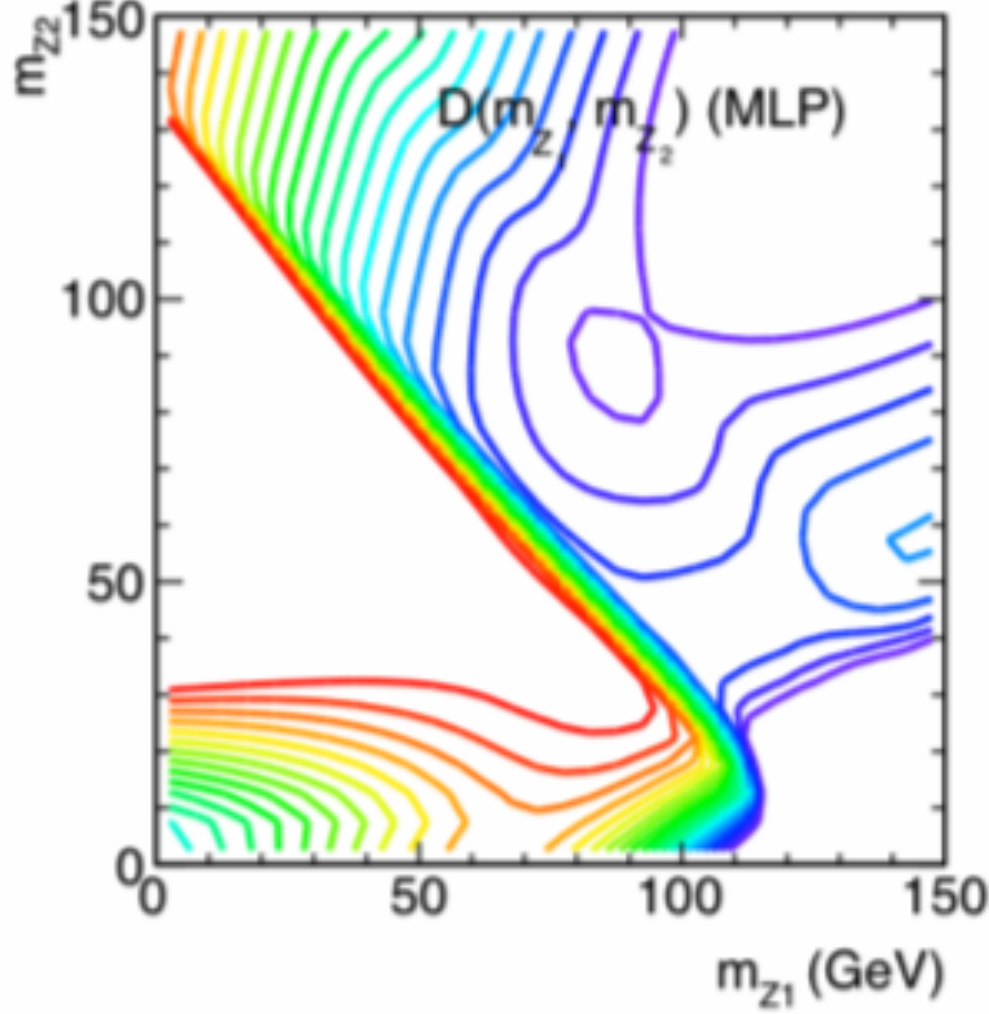
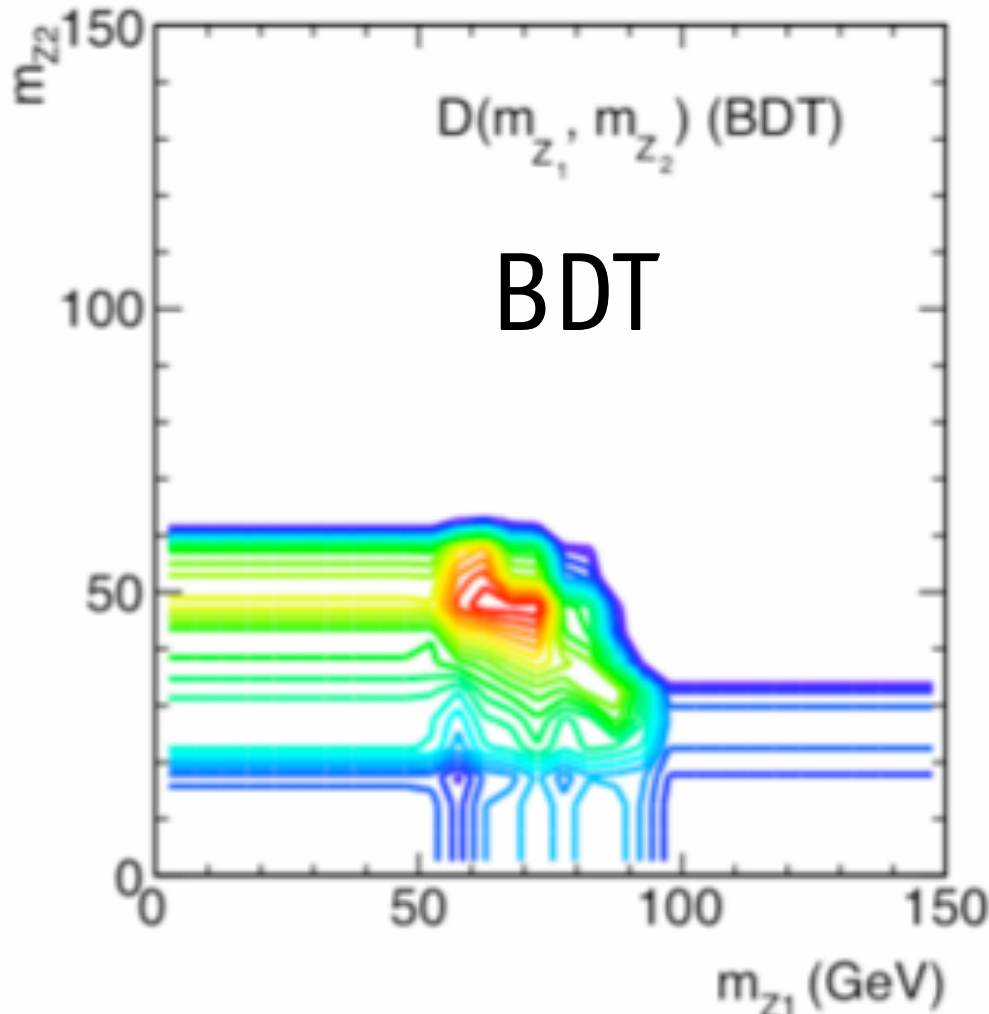
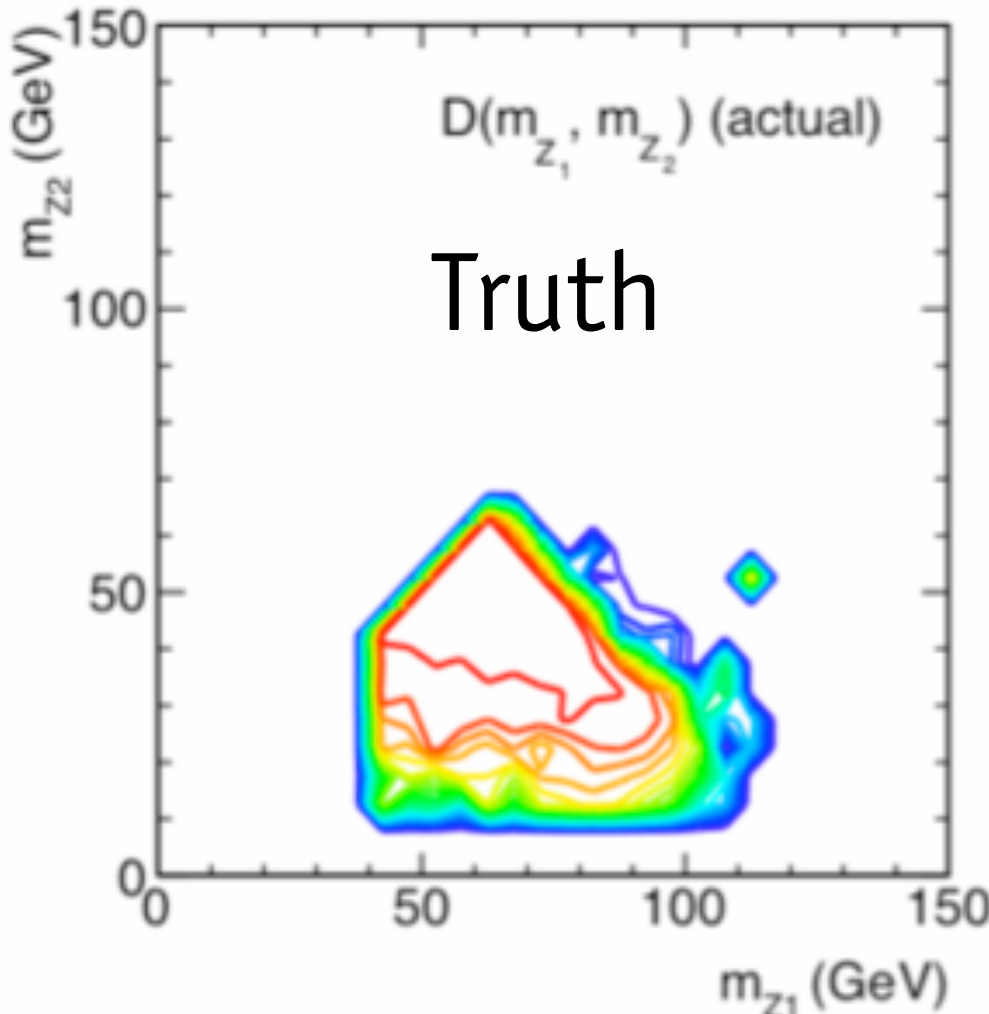
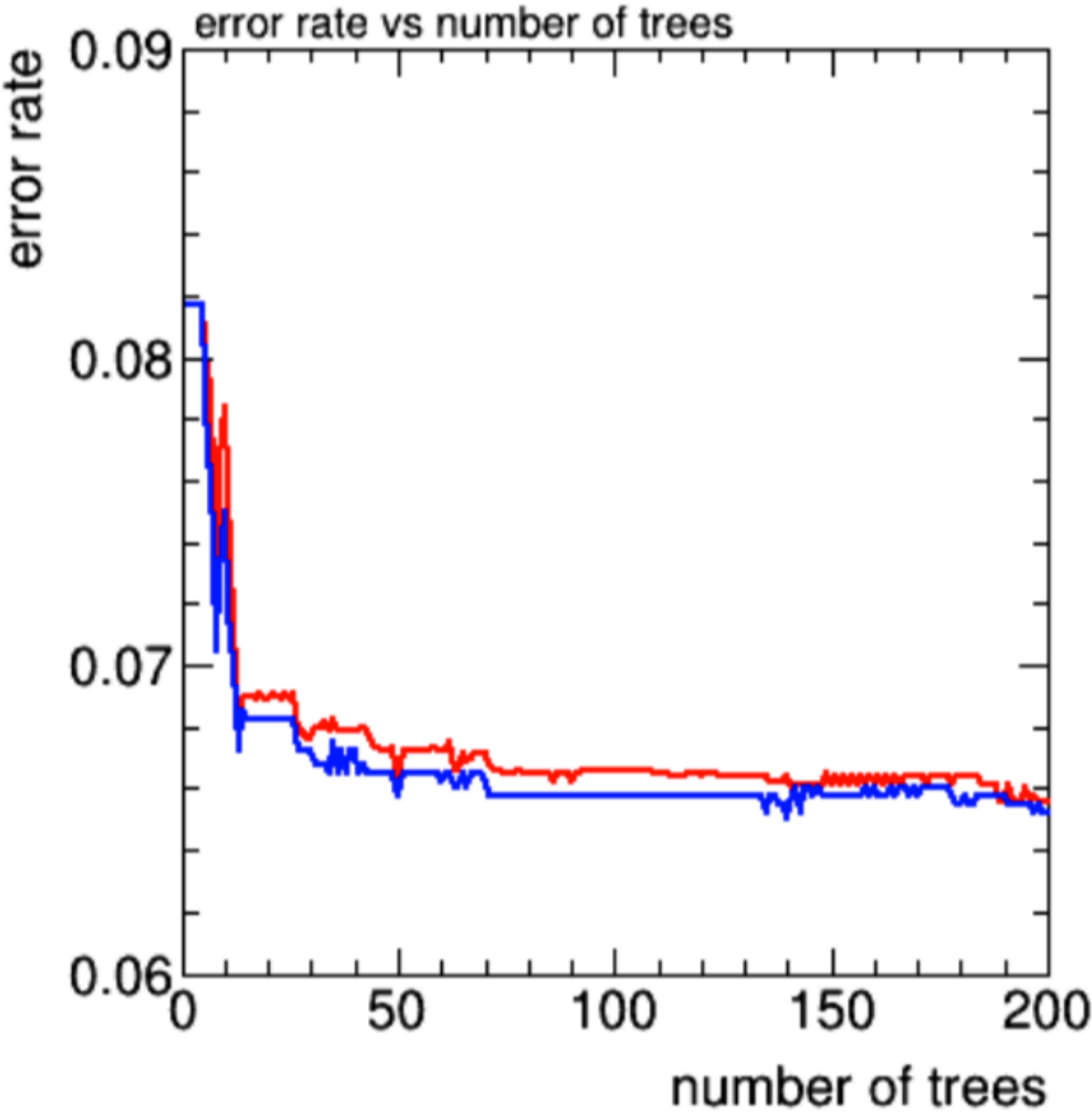
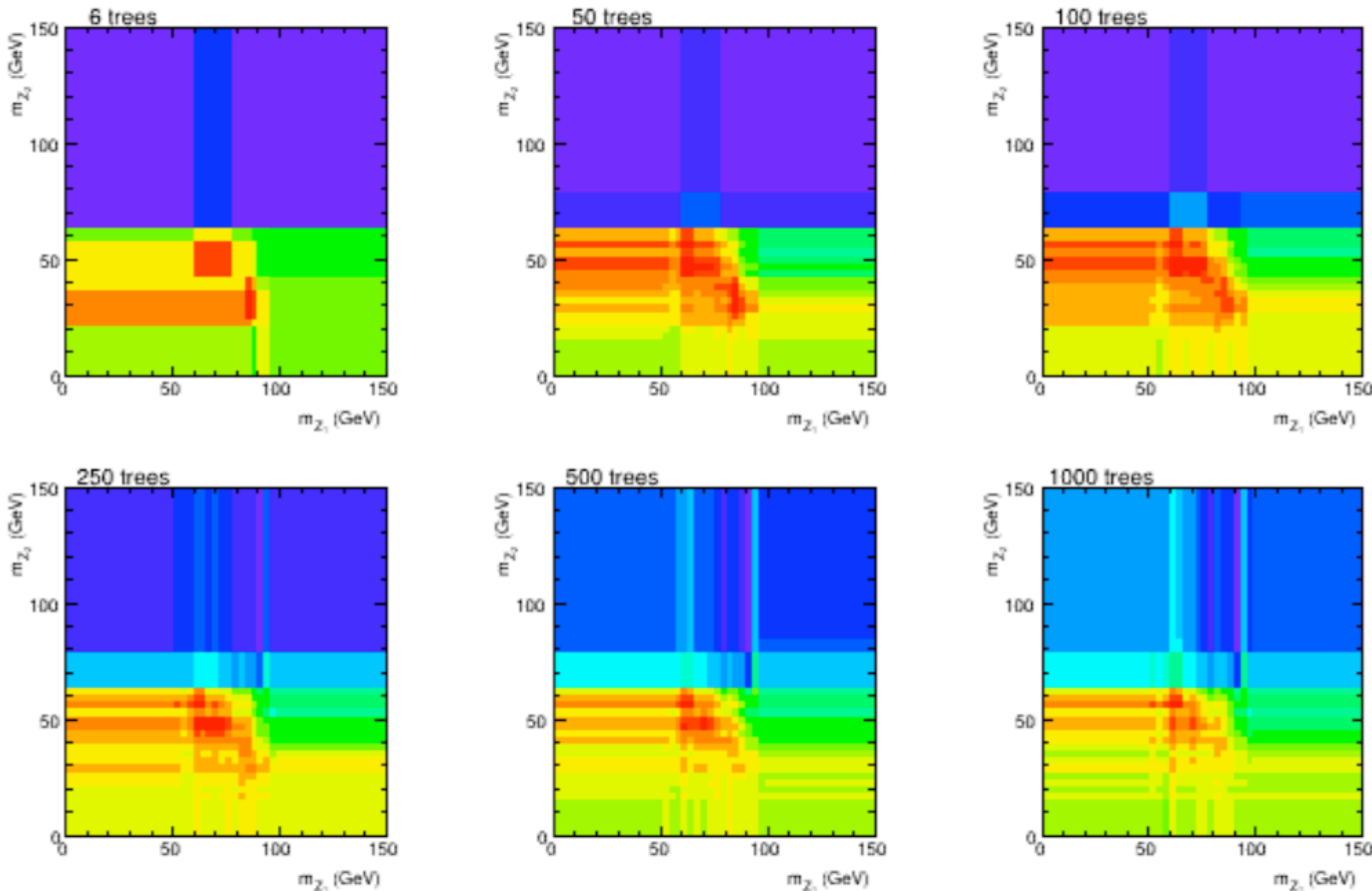
primi 6 alberi



Background



BDT: error rate vs #alberi



NN (shallow)

Risultati





# ADA-BOOST E OVERFITTING

si può dimostrare (Breiman, Friedman) che la tecnica di boosting può essere formalmente interpretata come una procedura di discesa lungo il gradiente di un algoritmo rispetto ad una appropriata loss function, i.e. la **minimizzazione di una funzione di “perdita” tra il modello di tree e il campione di dati di training**  $T = \{\mathbf{x}_i, y_i\}$

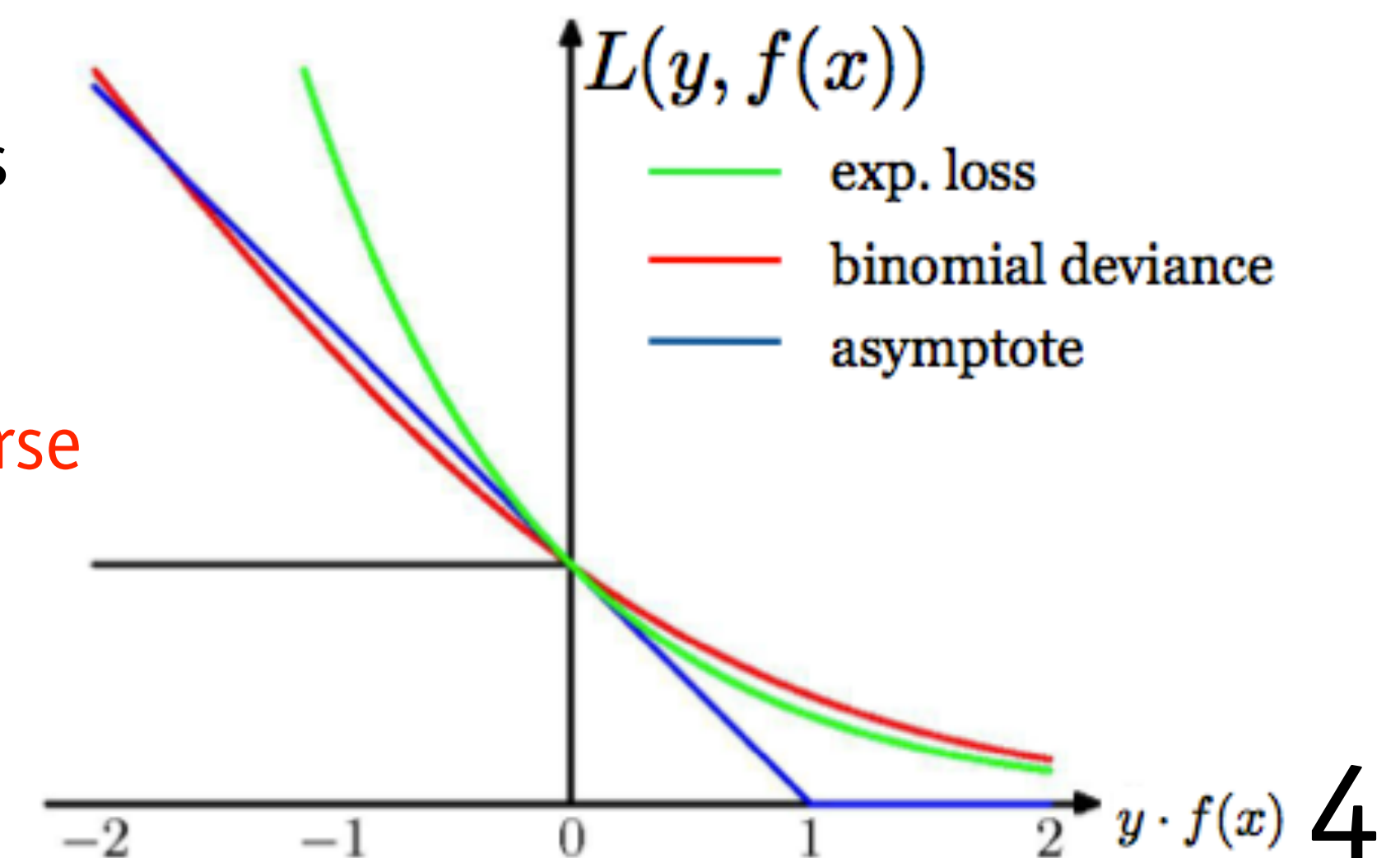
Friedman ha anche dimostrato che la funzione  $L()$  nel caso dell’ADA-Boost è equivalente alla funzione di “perdita esponenziale”:

$$L(y, f(x)) = \exp(-y \cdot f(x)); \quad y \in \{-1(bg), 1(sig)\}$$

$y_i \cdot f(x_i) > 0$  classificazione corretta  
 $y_i \cdot f(x_i) < 0$  classificazione errata

la tendenza all’overfitting dell’ ADA-boost è spiegata dalle caratteristiche della loss esponenziale di pesare fortemente outliers mal classificati nel training set

**una possibile soluzione al problema è quella di utilizzare funzioni di boosting diverse che possiedono migliori proprietà di generalizzazione, ottimizzare con tecniche di gradient boosting**



# GRADIENT BOOSTING

L'idea di base del Gradient Boosting è la stessa del boosting: combinare tanti classificatori deboli in un ensemble più forte con una tecnica iterativa

Esempio: regressione con metodo dei minimi quadrati

- vogliamo insegnare al nostro modello  $f$  a predire valori  $\hat{y} = f(x)$  minimizzando  $(\hat{y} - y)^2$  mediato sul campione di training  $T$
- procediamo iterativamente in  $N$  step (epoche). Ad ogni step  $j$  assumiamo di avere un modello approssimato di  $f$ :  $f_j$
- l'algoritmo GB migliora  $f_j$  aggiungendo a  $f_j$  un nuovo estimatore  $h(x)$ :

$$f_{j+1}(x) = f_j(x) + h(x) \text{ sia un modello migliore di } f_j(x)$$

- per trovare  $h(x)$  osserviamo che per un  $h(x)$  ideale (perfetto):

$$f_{j+1}(x) = f_j(x) + h(x) = y \rightarrow h(x) = (y - f_j(x))$$

- possiamo allora di trovare  $h(x)$  come funzione che fitti i residui di  $y - f_j(x)$
- ma i residui di  $(y - f(x))$  non sono altro che  $-\nabla$  rispetto a  $f(x)$  della funzione di perdita MSE:

$$L(f) = 1/2 (y - f(x))^2 \rightarrow -\partial_{f_j(x)} L(f) = (y - f_j(x))$$



il GB fondamentalmente consiste in un algoritmo di discesa lungo il gradiente nello spazio delle funzioni  $f(x)$

in modo più formale:

supponiamo di voler costruire un modello di regressione (regression tree) con **predictor** per il punto  $(\mathbf{x}_i, y_i)$  ottenuto sommando un ensemble di modelli (trees):

$$\hat{y}_i = g_{GB}(\mathbf{x}_i) = \sum_{j=1}^M g_j(\mathbf{x}_i)$$

consideriamo come **loss function** una funzione composta da due termini: uno che misura la bontà della predizione su ogni punto del campione di dati  $(\mathbf{x}_i, y_i)$   $i=1, \dots, N$ , che assumiamo sia differenziabile e convessa + uno (regolarizzatore) per ogni tree dell'ensemble che non dipende dai dati:

$$L(T, g_{GB}) = \sum_{i=1}^N l(y_i, \hat{y}_i) + \sum_{j=1}^M \Omega(g_j)$$

ad esempio xgBoost: square loss  $(y - \hat{y})^2$

in cui il regolarizzatore è scelto essere:

$$\Omega(g_j) = \gamma N_{foglie} + \frac{\lambda}{2} \sum_{t=1}^{N_{foglie}} w_{jt}^2$$

penalizza tree con  
troppe foglie

penalizza pesi troppo  
grandi per le foglie

NOTA:  $w_{jt}$  è il valore che il tree  $j$  associa alla foglia  $t$  del tree  
in pratica ogni albero  $g_j(x)$  è parametrizzato come una funzione  
 $q(x)$  che associa a  $x$  una delle foglie del tree  $(1, \dots, N_{foglie})$  + un  
vettore di pesi  $w$  che associa un valore ad ogni foglia:  $g_j(x) = w_{q(x)}$





formiamo l'ensemble in modo iterativo:  $\hat{y}_i^{(0)} = 0$   $\hat{y}_i^{(1)} = \hat{y}_i^{(0)} + g_1(\mathbf{x}_i)$  all'iterazione t:

$$\hat{y}_i^{(t)} = \sum_{j=1}^t g_j(\mathbf{x}_i) = \hat{y}_i^{(t-1)} + g_t(\mathbf{x}_i) \quad \text{NOTA: } \hat{y}_i^{(M)} = g_{GB}(\mathbf{x}_i)$$

l'idea centrale del gradient boosting è di ottimizzare la loss function per scegliere  $g_t(\mathbf{x})$  ad ogni iterazione:

$$L_t(g_t) = \sum_{i=1}^N l(y_i, \hat{y}_i^{(t-1)} + g_t(\mathbf{x}_i)) + \Omega(g_t) + \textit{termini costanti in } g_t$$

si osserva che per grandi t, ogni decision tree aggiunto  $g_t(\mathbf{x})$  costituisce solo una piccola perturbazione al predittore totale, il che ci permette di fare un'espansione in serie di Taylor nella loss function e di fermarci al secondo ordine:

$$L_t(g_t) \simeq \sum_{i=1}^N l(y_i, \hat{y}_i^{(t-1)}) + a_i g_t(\mathbf{x}_i) + \frac{1}{2} b_i g_t(\mathbf{x}_i)^2 + \Omega(g_t) + \textit{const in } g_t = \sum_{i=1}^N \left[ a_i g_t(\mathbf{x}_i) + \frac{1}{2} b_i g_t(\mathbf{x}_i)^2 \right] + \Omega(g_t) + \textit{const in } g_t$$

con:  $a_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$   $b_i = \partial_{\hat{y}_i^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)})$

il t-esimo decision tree  
scelto in modo da  
minimizzare  $L_t$

esempio loss quadratica  
usata in xgBoost:

$$L_t(g_t) \simeq \sum_{i=1}^N \left[ 2(y_i - \hat{y}_i^{(t-1)}) g_t(\mathbf{x}_i) + g_t^2(\mathbf{x}_i) \right] + \Omega(g_t) + \textit{const in } g_t$$



# XGBOOST

- algoritmo altamente ottimizzato per l'implementazione di foreste di alberi di decisione gradient-boost based:
  - <https://github.com/dmlc/xgboost>
- **molteplici tecniche di regolarizzazione:**
  - shrinkage:  $f_j(x) = f_{j-1}(x) + \eta h(x)$  → empiricamente è stato dimostrato che piccoli learning rate  $\eta < 0.1$  portano a sostanziali miglioramenti nelle proprietà di generalizzazione del modello rispetto al caso  $\eta = 1$  (pezzo: algoritmi più lenti)
  - penalizzazione della complessità degli alberi: varie forme di regolarizzazione L2, L1, L1+L2 ...
  - stochastic gradient descent
- **tecniche di velocizzazione:**
  - fast approximation della direzione di discesa del gradiente usando le derivate seconde della loss function
  - GPU acceleration
- **molteplici interfacce a librerie/framework per ML:**
  - python/C++/R/scikit-learn APIs ...





# ESEMPIO CLASSIFICAZIONE IN IN XGBOOST

Data: [https://www.dropbox.com/s/z654vjcy3qwcbe/heart\\_uci.data?dl=0](https://www.dropbox.com/s/z654vjcy3qwcbe/heart_uci.data?dl=0)

Python code: [https://www.dropbox.com/s/fk5v44pu2a1zhon/xGBoost\\_example.py?dl=0](https://www.dropbox.com/s/fk5v44pu2a1zhon/xGBoost_example.py?dl=0)

```
#!/usr/bin/python
```

```
from __future__ import division
```

```
import numpy as np
```

```
import xgboost as xgb
```

```
from sklearn.model_selection import KFold, train_test_split, GridSearchCV
```

```
from sklearn.metrics import confusion_matrix, mean_squared_error
```

```
from sklearn.datasets import load_iris, load_digits, load_boston
```

```
from sklearn.linear_model import LogisticRegression
```

```
np.random.seed(1234) #for reproducibility
```

UCI heart disease dataset

```
#use UCI heart disease dataset heart.cvi preprocessed by kaggle
```

```
# https://www.kaggle.com/ronitf/heart-disease-uci#heart.csv
```

```
# 14 features for bynary target (0 no presence, 1 presence of heart disease)
```

```
data = np.loadtxt('./heart_uci.data', delimiter=',')
```

```
np.random.shuffle(data)
```

```
sz = data.shape
```



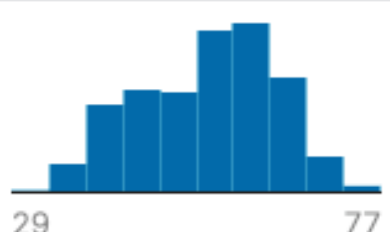


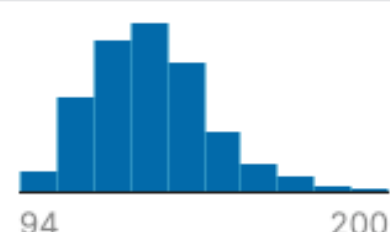



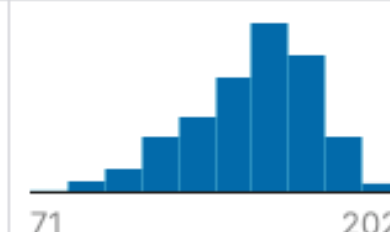

# UCI HEART DISEASE DATASET

<https://www.kaggle.com/ronitf/heart-disease-uci>

## Context

This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "goal" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4.

1. age
2. sex
3. chest pain type (4 values)
4. resting blood pressure
5. serum cholestoral in mg/dl
6. fasting blood sugar > 120 mg/dl
7. resting electrocardiographic results (values 0,1,2)
8. maximum heart rate achieved
9. exercise induced angina
10. oldpeak = ST depression induced by exercise relative to rest
11. the slope of the peak exercise ST segment
12. number of major vessels (0-3) colored by flourosopy
13. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
14. target (1 or 0)

	# age age in years	# sex (1 = male; 0 = female)	# cp chest pain type	# trestbps resting blood pressure (in mm Hg on admission to the hospital)	# chol serum cholestoral in mg/dl	# fbs (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)	# restecg resting electrocardiographic results	# thalach maximum heart rate achieved	# exang exercise inc (1 = yes; 0
									
	29 77	0 1	0 3	94 200	126 564	0 1	0 2	71 202	0
1	63	1	3	145	233	1	0	150	
2	37	1	2	130	250	0	1	187	
3	41	0	1	130	204	0	0	172	
4	56	1	1	120	236	0	1	178	
5	57	0	0	120	354	0	1	163	
6	57	1	0	140	192	0	1	148	





```
train = data[:int(sz[0] * 0.7), :]  
test = data[int(sz[0] * 0.7):, :]
```

divide data in training (70%) and test (30%)

```
X_train = train[:, :13]  
y_train = train[:, 14]
```

last entry on the UCI dataset is the label

```
X_test = test[:, :13]  
y_test = test[:, 14]
```

```
print(X_train)  
print(y_train)  
xgb_model = xgb.XGBClassifier().fit(X_train, y_train)  
predictions = xgb_model.predict(X_test)  
actuals = y_test
```

xGBoost provides also useful API for  
scikit-learn

```
print('Confusion matrix:')  
conf_m = confusion_matrix(actuals, predictions)  
print(conf_m)
```

```
Confusion matrix: [[31  8]  
                  [11 41]]
```

```
#get the accuracy from confusion matrix
```

```
accuracy = (conf_m[0][0] + conf_m[1][1]) / (conf_m[0][0] + conf_m[0][1] + conf_m[1][0] + conf_m[1][1])
```

```
print('Accuracy from confusion_matrix: ', accuracy)
```

```
print('Accuracy from scikit-learn: ', xgb_model.score(X_test, y_test))
```

```
logreg_model = LogisticRegression(C=1e5, solver='sag')
```

```
logreg_model.fit(X_train, y_train)
```

```
print('Accuracy for a linear classifier (LogisticRegression): ',
```

```
logreg_model.score(X_test, y_test))
```

```
Accuracy (xGBoost):      0.791
```

```
Accuracy (logisticReg):  0.692
```



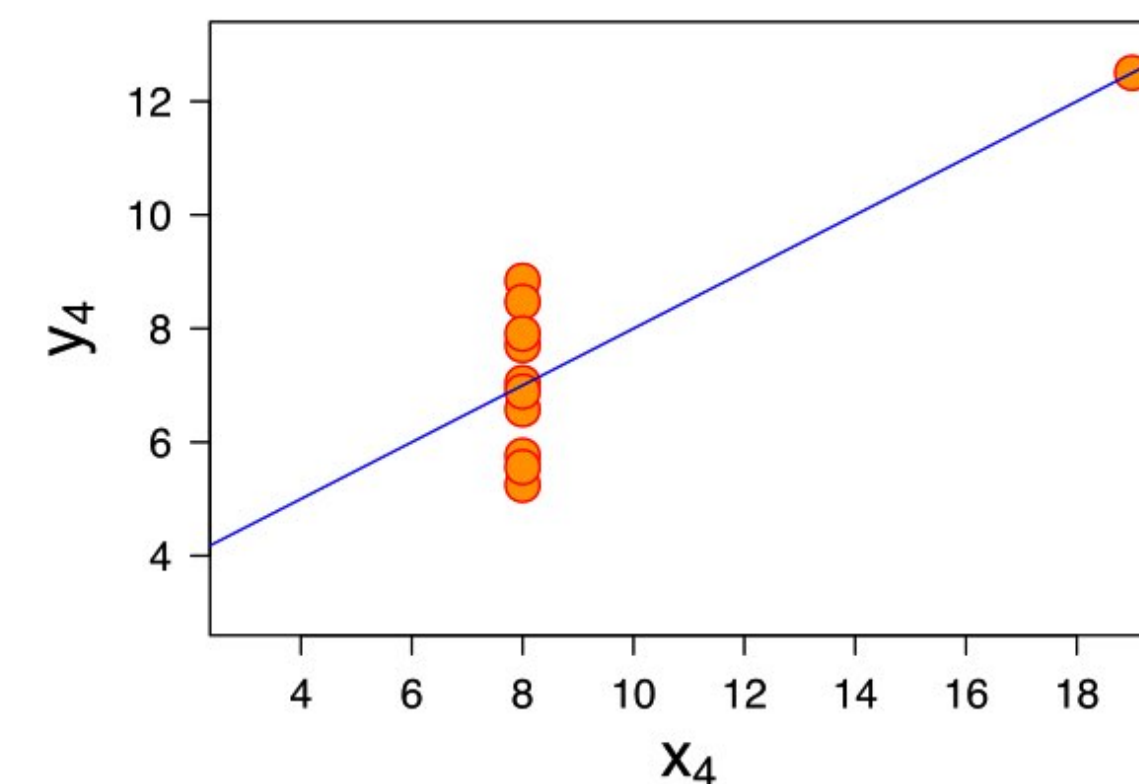
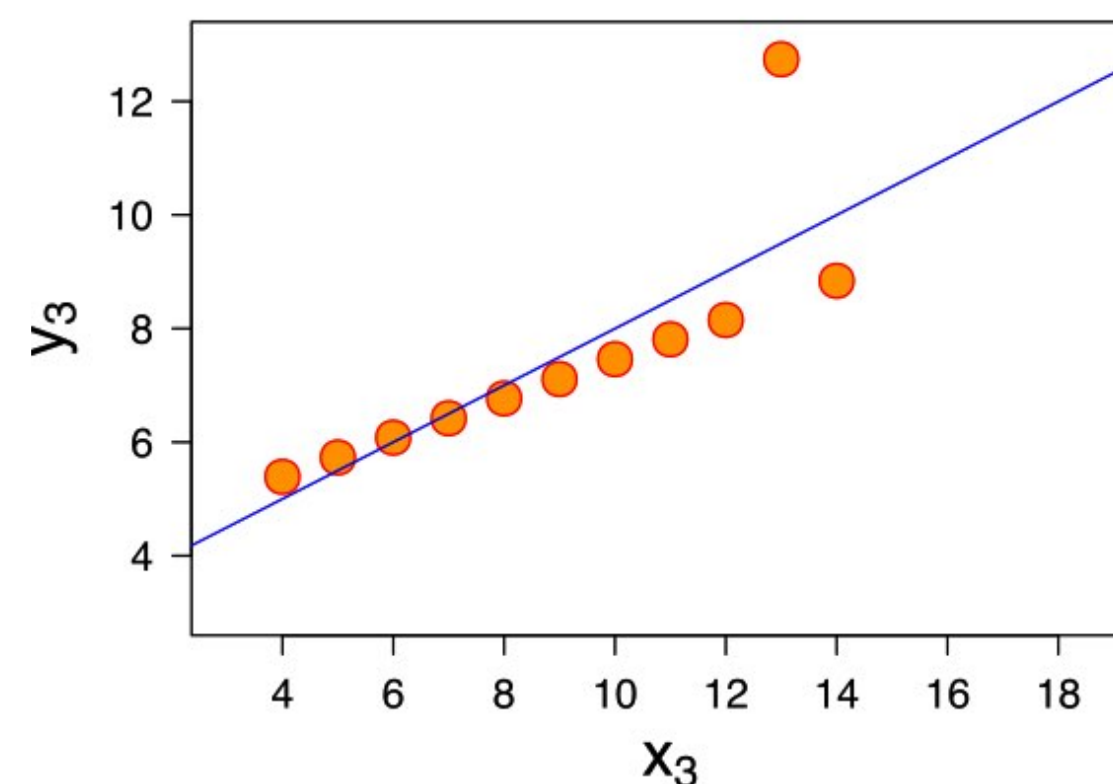
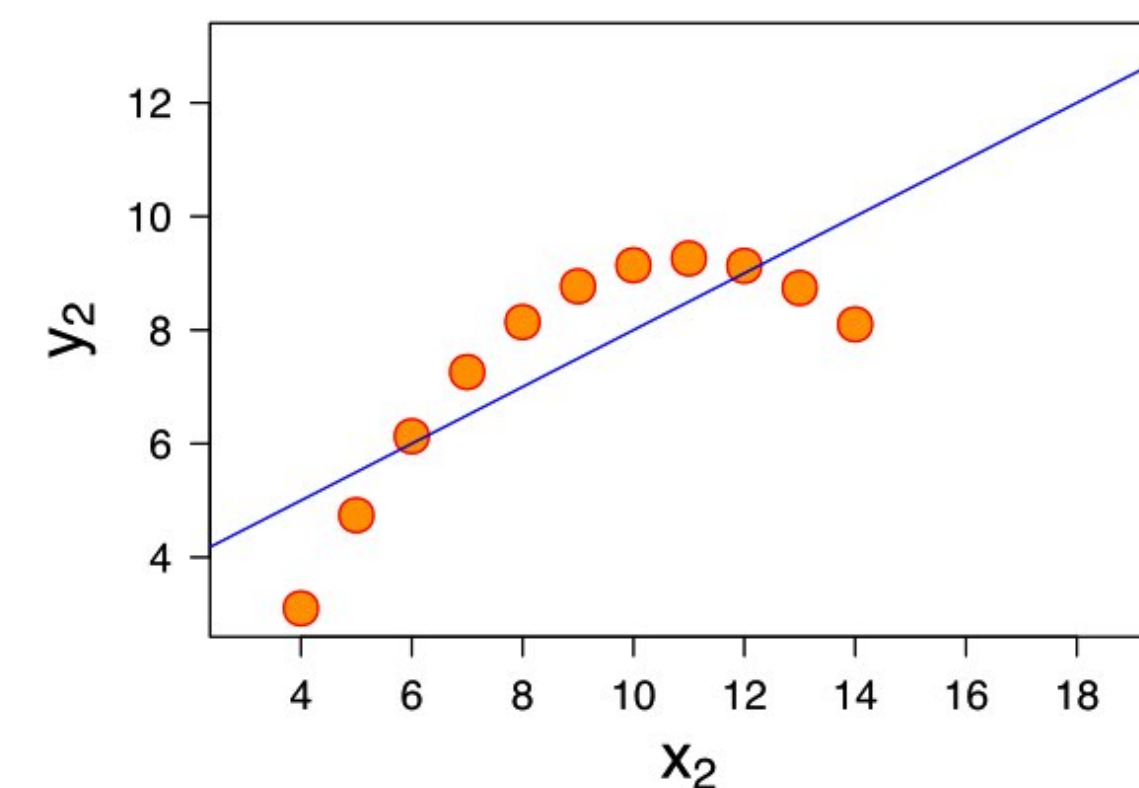
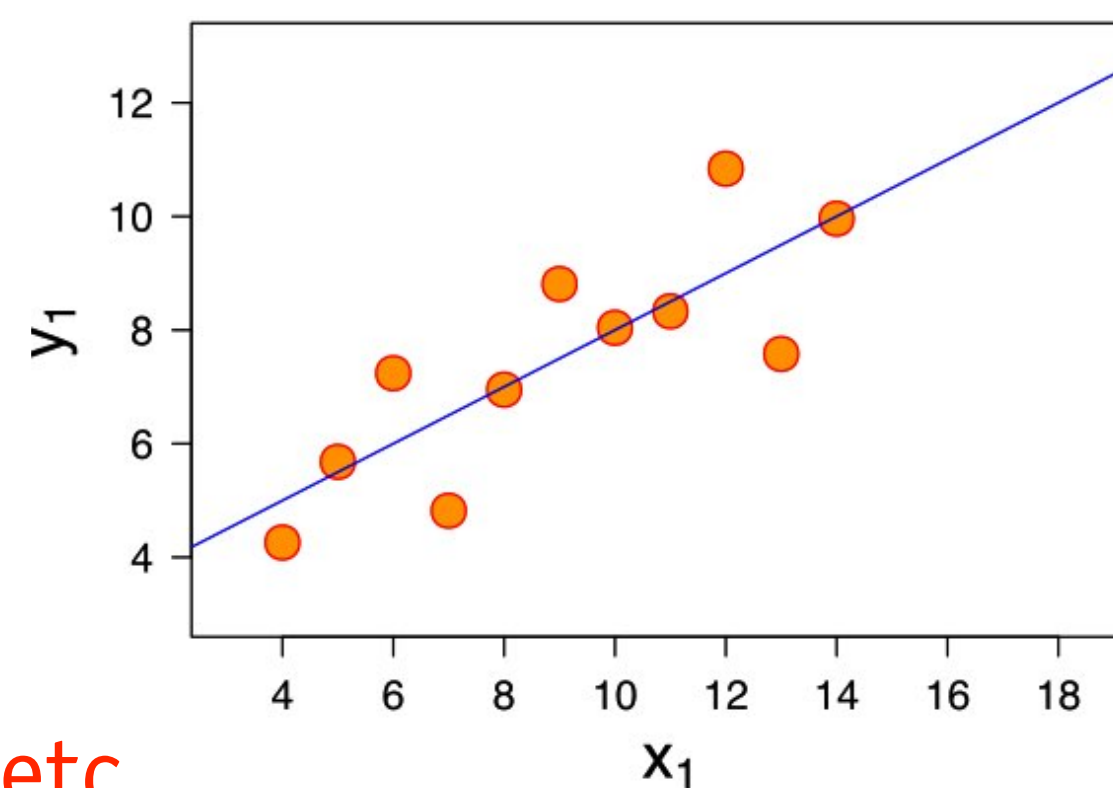


# VISUALIZZAZIONE DEI DATI E RIDUZIONE DIMENSIONALE

- la fase di **esplorazione e visualizzazione dei dati** rappresenta il primo passo in qualsiasi progetto di Machine Learning, saltare tale fase significa sprecare molto tempo con strategie e modelli sbagliati e spesso non riuscire ad arrivare ad un modello finale soddisfacente

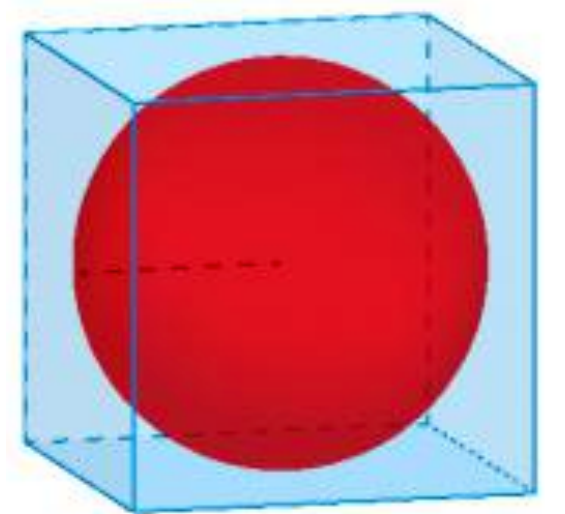
- esempio: i dataset mostrati hanno tutti indicatori delle summary statistic identici (valori medi in x e y, varianze, indici di correlazione, parametri dei best fit lineari, ...)
- solo **l'analisi visuale permette di identificare specifici comportamenti nei dati, possibili artefatti, rumore, outliers, etc...**  
ed aiuta a scegliere il modello più appropriato per analizzare i dati stessi

- metodi di riduzione-dimensionale aiutano nella visualizzazione di dati multidimensionali**



# VISUALIZZAZIONE DEI DATI E RIDUZIONE DIMENSIONALE

- visualizzare dati con un basso numero di dimensioni è facile ed intuitivo, per esempio andando a guardare le correlazioni tra coppie di osservabili → non praticabile al crescere della dimensionalità dei dati analizzati → **riduzione dimensionale**
- **problema**: all'operazione di riduzione delle dimensioni è associata una **perdita di informazione**
- **goal**: minimizzare tale perdita preservando le informazioni più importanti contenute nel dataset analizzato
- **Caratteristiche particolari dei dati ad alta-dimensione:**
  - **i dati ad alta dimensione tendono ad addensarsi vicino ai bordi dello spazio campionato** (la geometria in alte dimensioni ha proprietà contro-intuitive):
    - esempio:
      - dati distribuiti in modo uniforme su un ipercubo D-dimensionale di lato  $l$ :  $C = [-l/2, l/2]^D$
      - ipersfera  $S$  di raggio  $l/2$  centrata nell'origine e contenuta in  $C$
      - la probabilità che un dato  $x$  estratto in modo random e uniformemente da  $C$  sia contenuto in  $S$  è ben approssimato dal rapporto tra i volumi di  $S$  e  $C$ :  $p(x \in S) \sim (1/2)^D$ , quindi per  $D \rightarrow \infty \Rightarrow p \rightarrow 0$  con velocità esponenziale e tutti i punti andranno a concentrarsi negli angoli dell'ipercubo (è l'osservazione che in fisica spiega molte delle proprietà dei gas ideali (distribuzione di Maxwell, equipartizione, ...))



# VISUALIZZAZIONE DEI DATI E RIDUZIONE DIMENSIONALE

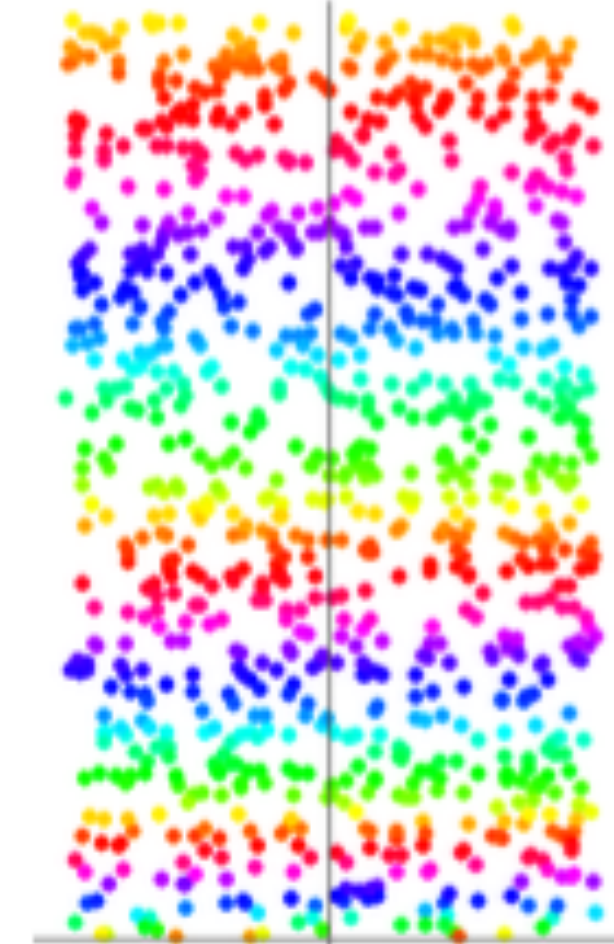
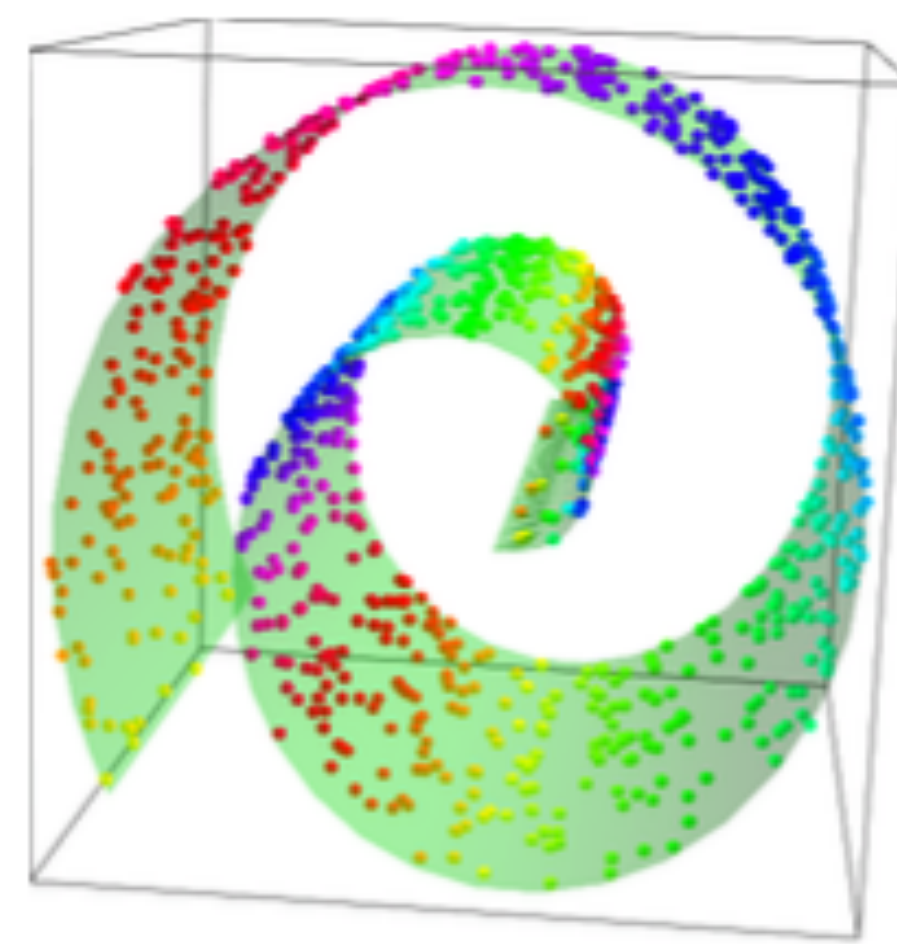
- **in realtà i dati reali non seguono generalmente distribuzioni uniformi:** i dati reali vivono in spazi di dimensione inferiore a quello dello spazio in cui le feature sono state misurate e risultano quindi mediati ("smoothed") localmente, il che significa che una variazione locale nei dati misurati spesso non comporterà un cambiamento sensibile nel target
- idea centrale nella fisica statistica e nelle teorie di campo, in cui le proprietà dei sistemi con un numero enorme di gradi di libertà possono essere ben caratterizzate da "parametri di ordine" a bassa dimensione o da effettivi gradi di libertà (esempio le proprietà di insieme di un gas possono essere rappresentate da poche variabili termodinamiche  $(T, P, \dots)$  invece che da un enorme numero di coppie  $(x, p)$  di ogni molecola del gas)



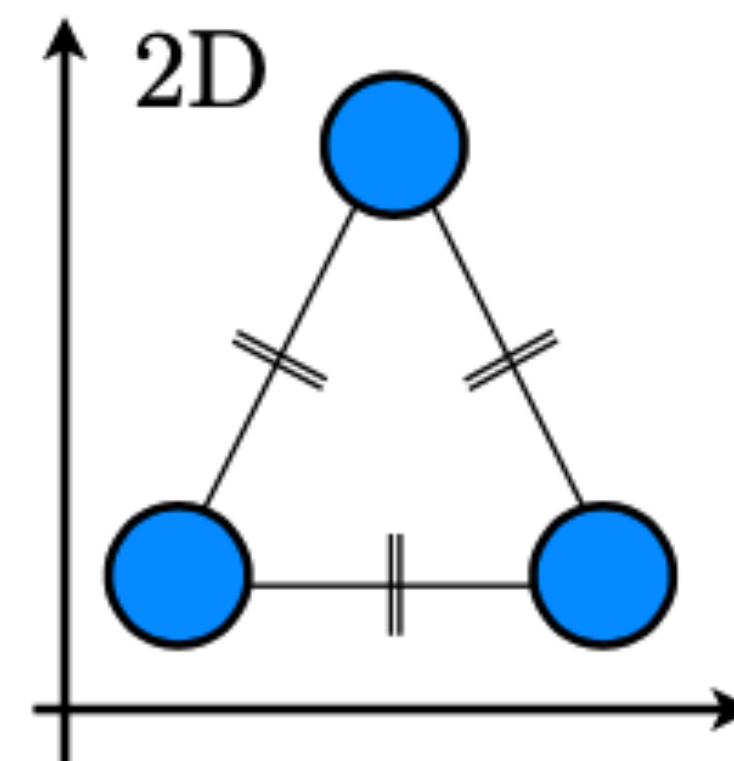


# VISUALIZZAZIONE DEI DATI E RIDUZIONE DIMENSIONALE

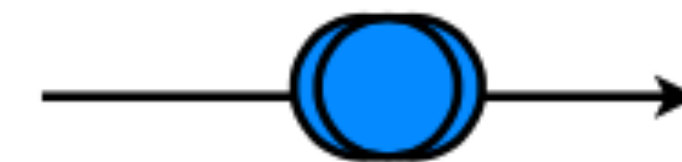
- **crowding problem**: quando si cerca di ridurre la dimensione di un campione **cercando di preservare le distanze relative tra i punti nello spazio originale**, e si scende sotto la dimensionalità intrinseca del campione stesso (quella che cattura l'insieme completo delle informazioni dei dati) si rischia di creare un problema di sovrappollamento dei dati
- soluzione o rilasciare i vincoli di similitudine dello spazio originale oppure tSNE



OK



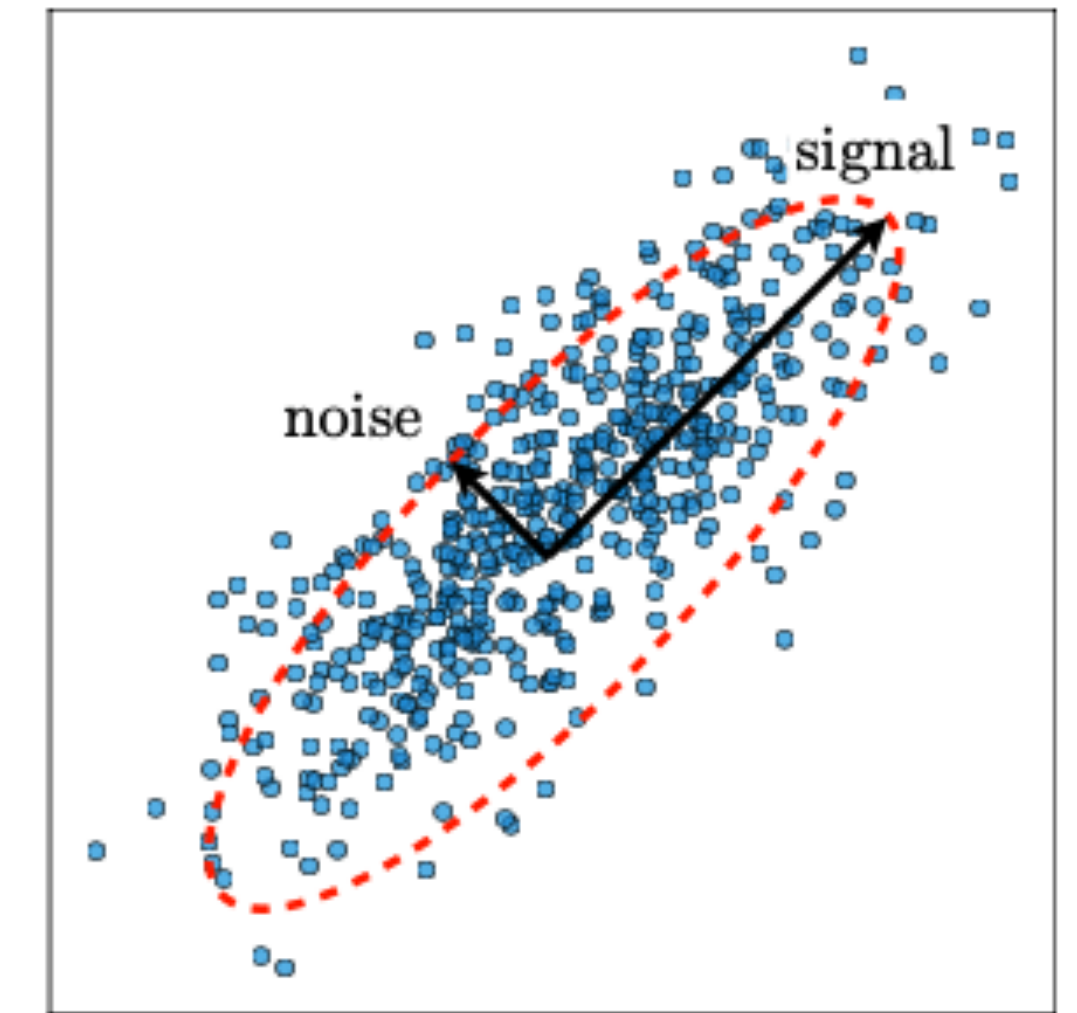
1D



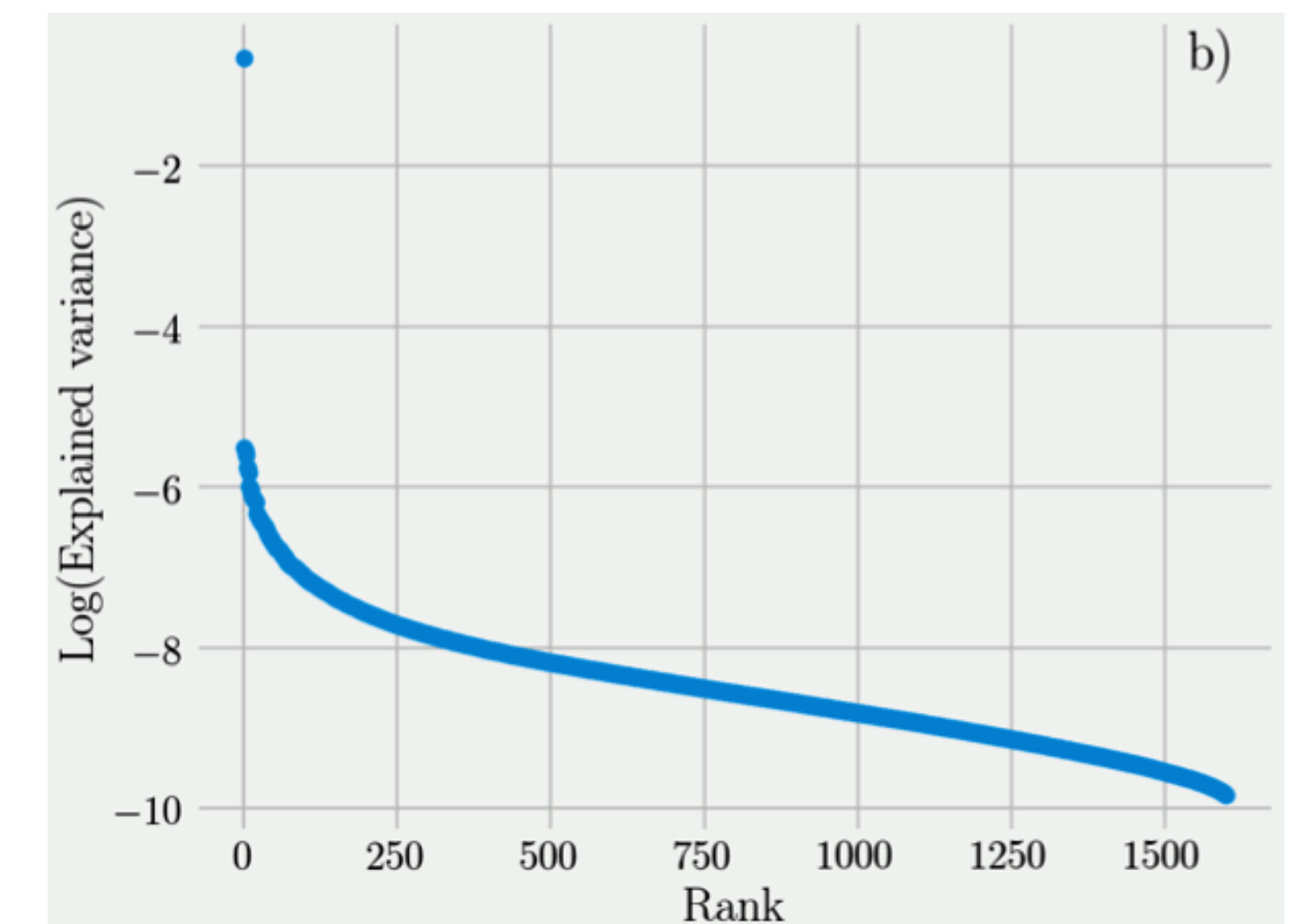
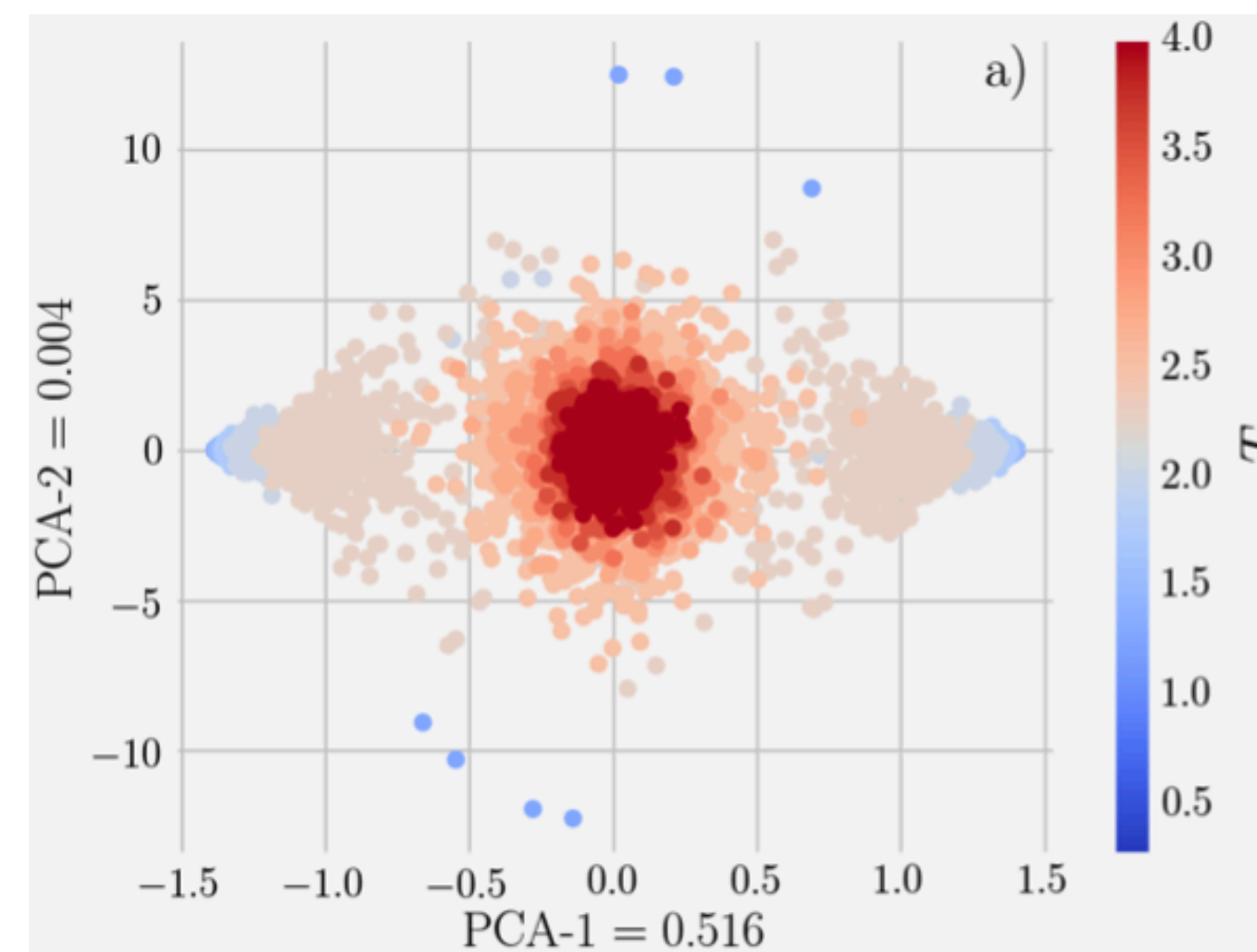
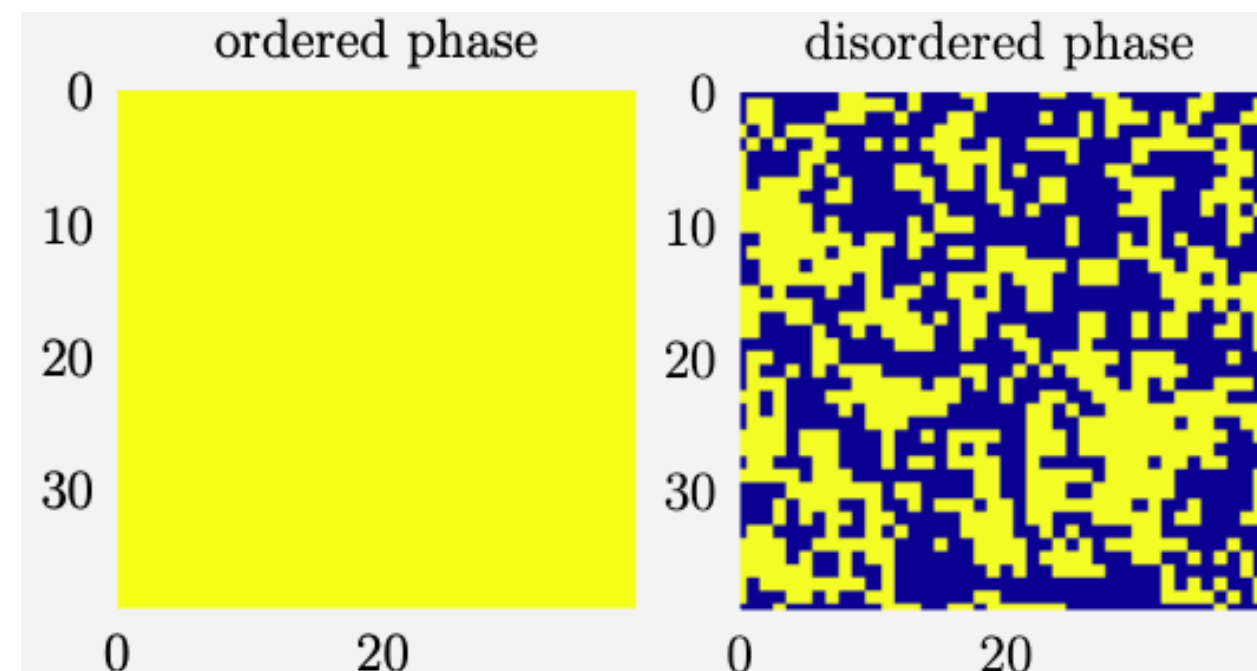
CROWDING

# PCA

- **analisi a componenti principali (PCA)** è il metodo classico più noto di riduzione dimensionale
- consiste nel:
  1. applicare una trasformazione ortogonale ai dati D-dimensionali in modo da individuare le direzioni di massima varianza dei dati
  2. ordinare tali direzione in ordine di importanza (da quella con massima varianza a quella con minima)
  3. prendere le prime  $d < D$  direzioni come rappresentanti il campione e considerare come “noise” le restanti
- in grado di identificare la maggior parte delle strutture di grande scala in molti campioni di dati diversi



**esempio: Ising 2D dataset**  
**prime due component principali**



# PCA

- più in dettaglio: supponiamo di avere  $N$  dati  $\{\mathbf{x}_i\}$   $i=1,\dots,N$  con  $\mathbf{x}_i$  feature vector  $d$ -dimensionale dell'evento  $i$ -esimo del set e supponiamo (per semplicità) che la media empirica sia zero
- sia  $\mathbf{X}$  la matrice  $N \times d$ :  $\mathbf{X}=[\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N]^t$  con righe i punti dei dati e colonne le feature

matrice di covarianza (simmetrica:  $d \times d$ ) associata: 
$$\Sigma(\mathbf{X}) = \frac{1}{N-1} \mathbf{X}^t \mathbf{X}$$
 con  $\Sigma_{ii}$  varianza della feature  $i$   
 $\Sigma_{ij}$  covariant tra feature  $i$  e  $j$

si cerca la trasformazione lineare che riduce la covarianza tra feature diverse, via singular value decomposition si diagonalizza la matrice di covarianza:

$$\mathbf{X} = \mathbf{U} \mathbf{S} \mathbf{V}^t \implies \Sigma(\mathbf{X}) = \frac{1}{N-1} \mathbf{V} \mathbf{S} \mathbf{U}^t \mathbf{U} \mathbf{S} \mathbf{V}^t = \mathbf{V} \left( \frac{\mathbf{S}^2}{N-1} \right) \mathbf{V}^t = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^t$$

con  $\mathbf{\Lambda}$  matrice diagonale con autovalori  $\lambda_i = s_i^2 / (N-1)$

le colonne di  $\mathbf{V}$  sono direzioni principali di  $\Sigma$ , si selezionano le matrici di proiezione  $\tilde{\mathbf{V}}$  corrispondenti ai valori singolari massimi ( $\lambda_i$  massimi) e si proiettano i dati lungo tali direzioni:

$$\tilde{\mathbf{X}} = \mathbf{X} \tilde{\mathbf{V}}$$





# tSNE

- tSNE (t-stochastic neighbor embedding) è un metodo alternativo basato su trasformazioni non-lineari e che quindi preserva meglio strutture locali presenti nei dati
- è diventato uno dei metodi standard per la visualizzazione di dati multidimensionali
- idea di base:
  - si associa una distribuzione di probabilità nello spazio originale che regola le relazioni tra punti vicini
  - si cerca di ri-creare uno spazio a dimensione più piccola che sia in grado di seguire tale probabilità nel modo più accurato

1. si prende un punto  $x_i$  del dataset e si associa una distribuzione di probabilità dei suoi vicini (un altro punto  $x_j$ ) proporzionale ad una gaussiana centrata in  $x_i$

$$p_{ij} = \frac{e^{\frac{-||x_i - x_j||^2}{2\sigma_i^2}}}{\sum_{k \neq i} e^{\frac{-||x_i - x_k||^2}{2\sigma_i^2}}}$$

$\sigma_i$  sono parametri liberi da determinare in modo tale che il numero di vicini sia circa lo stesso (perplexity) per ogni punto per evitare che vi sia un punto che pesi in modo sproporzionato rispetto agli altri

2. nello spazio ridotto  $\{y\}$  in principio vorremmo che la distribuzione di probabilità dei vicini in ogni punto assomigli a  $p_{ij}$ :

$$q_{ij} = \frac{e^{\frac{-||y_i - y_j||^2}{2\sigma_i^2}}}{\sum_{k \neq i} e^{\frac{-||y_i - y_k||^2}{2\sigma_i^2}}}$$

non va bene a causa del problema del crowding!

la gaussian ha code piccole → alta probabilità che eventi vicini in  $x$  vengano compressi in  $y$



- soluzione: usiamo una distribuzione con struttura simile ma code più grandi, la t-student con 1 grado di libertà (distribuzione di Cauchy):

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq i} (1 + ||y_i - y_k||^2)^{-1}}$$

- per trovare le coordinate dello spazio latente  $\{y\}$  tSNE minimizza la divergenza di Kullback-Leibler tra  $p_{ij}$  e  $q_{ij}$ :

$$L(y) = D_{KL}(p || q) = \sum_{ij} p_{ij} \log \left( \frac{p_{ij}}{q_{ij}} \right)$$

$D_{KL}(p||q)$ : misura la dissimilarità tra due distribuzioni di probabilità  $p$  e  $q$   
 $D_{KL}(p||q) \geq 0$  (=0 se e solo se  $p=q$ )

- usando la discesa lungo il gradiente:

$$\partial_{y_i} L(y) = \sum_{j \neq i} 4p_{ij}q_{ij}Z_i(y_i - y_j) - \sum_{j \neq i} 4q_{ij}^2Z_i(y_i - y_j) = F_{attrattiva}(i) - F_{repulsiva}(i)$$

$$Z_i = \frac{1}{\sum_{k \neq i} (1 + ||y_i - y_k||^2)^{-1}}$$

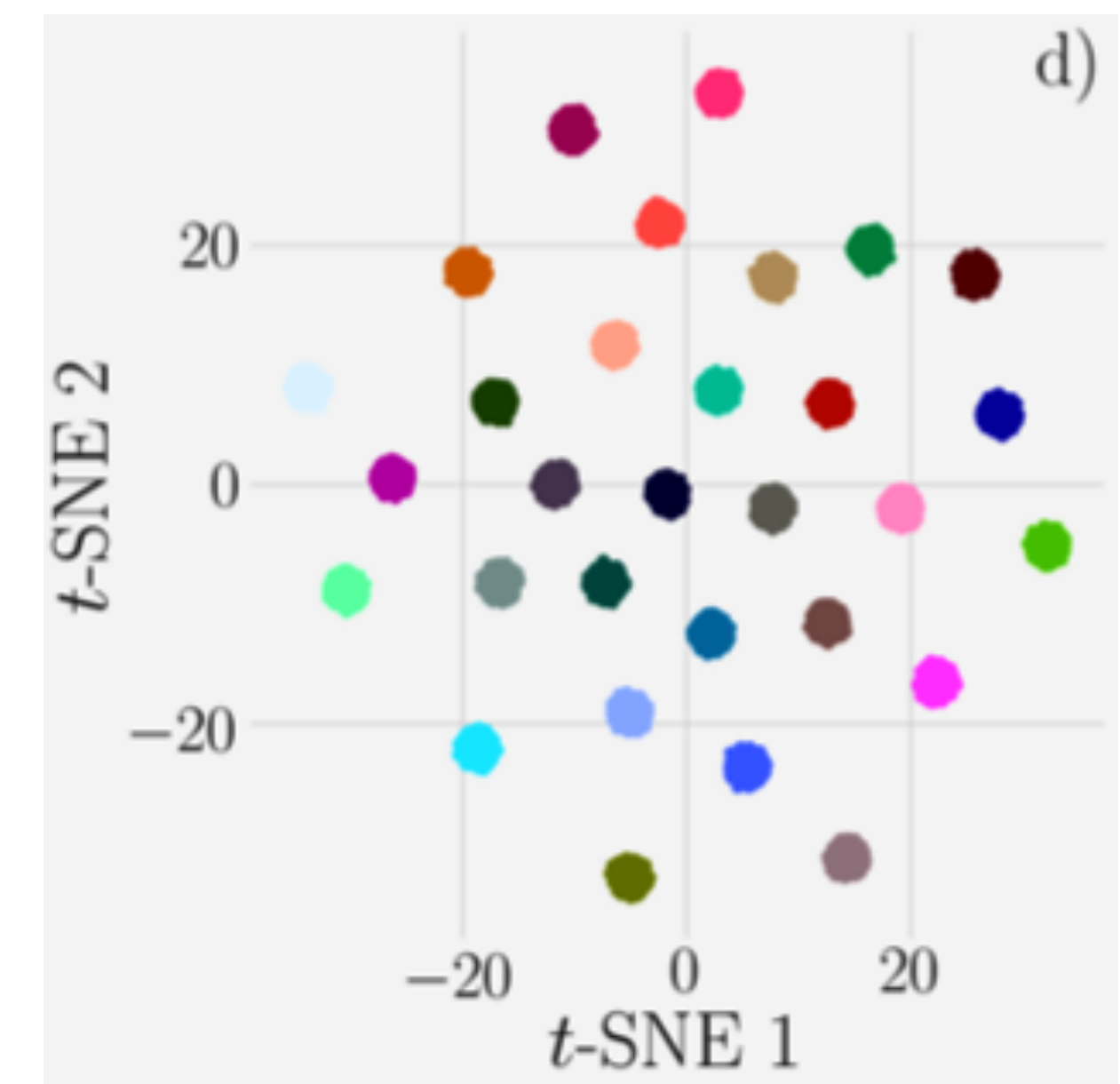
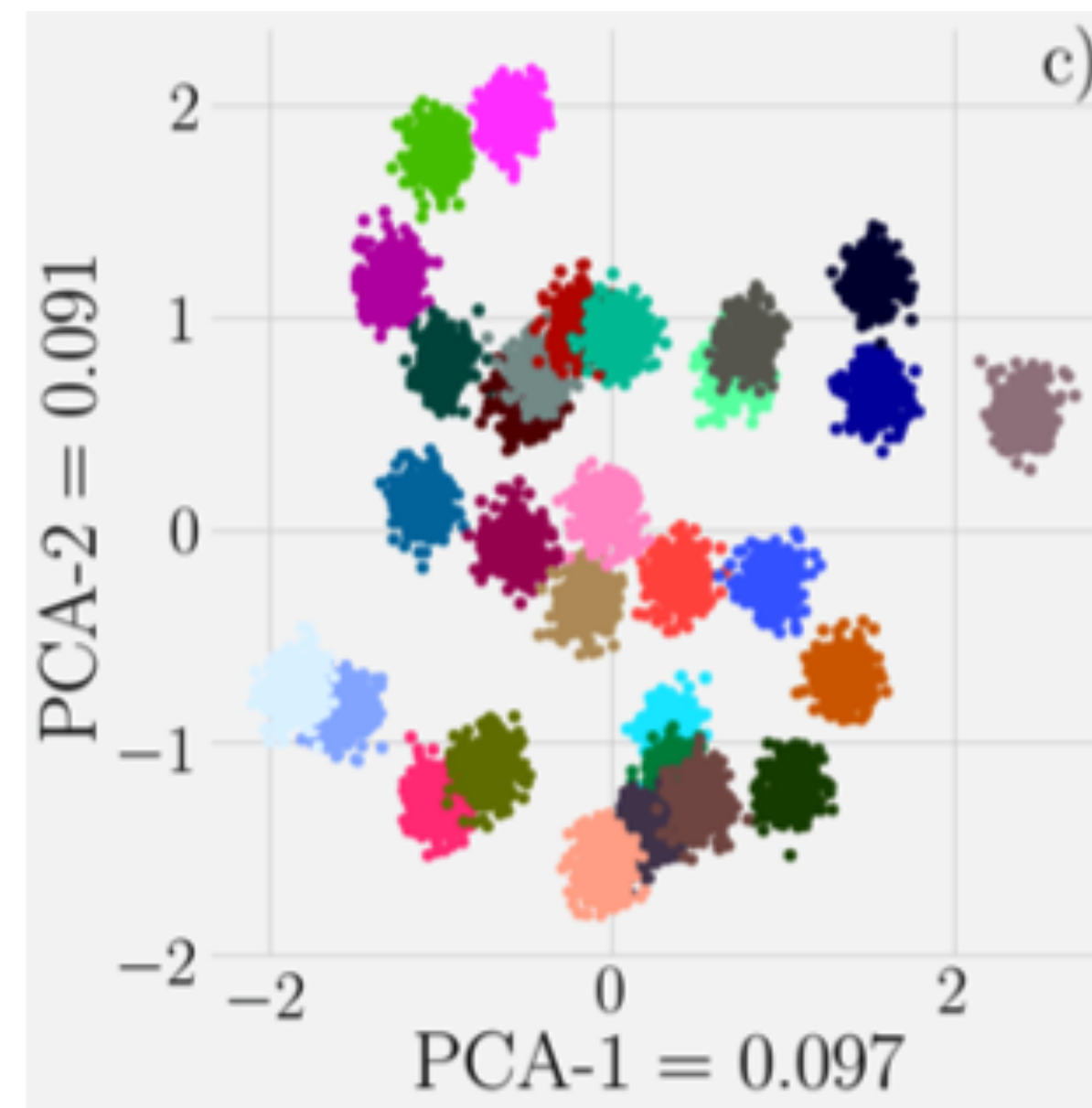
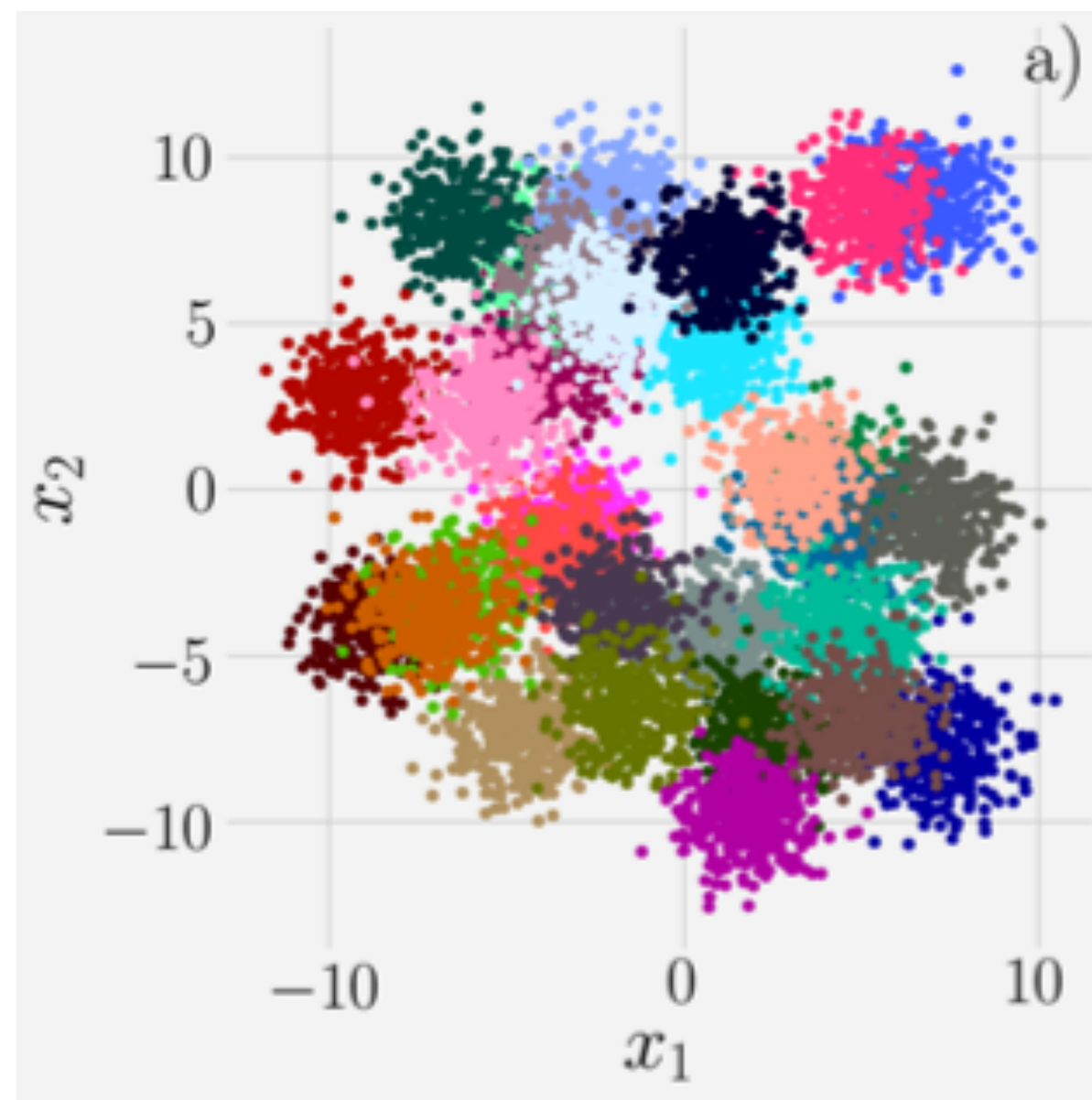
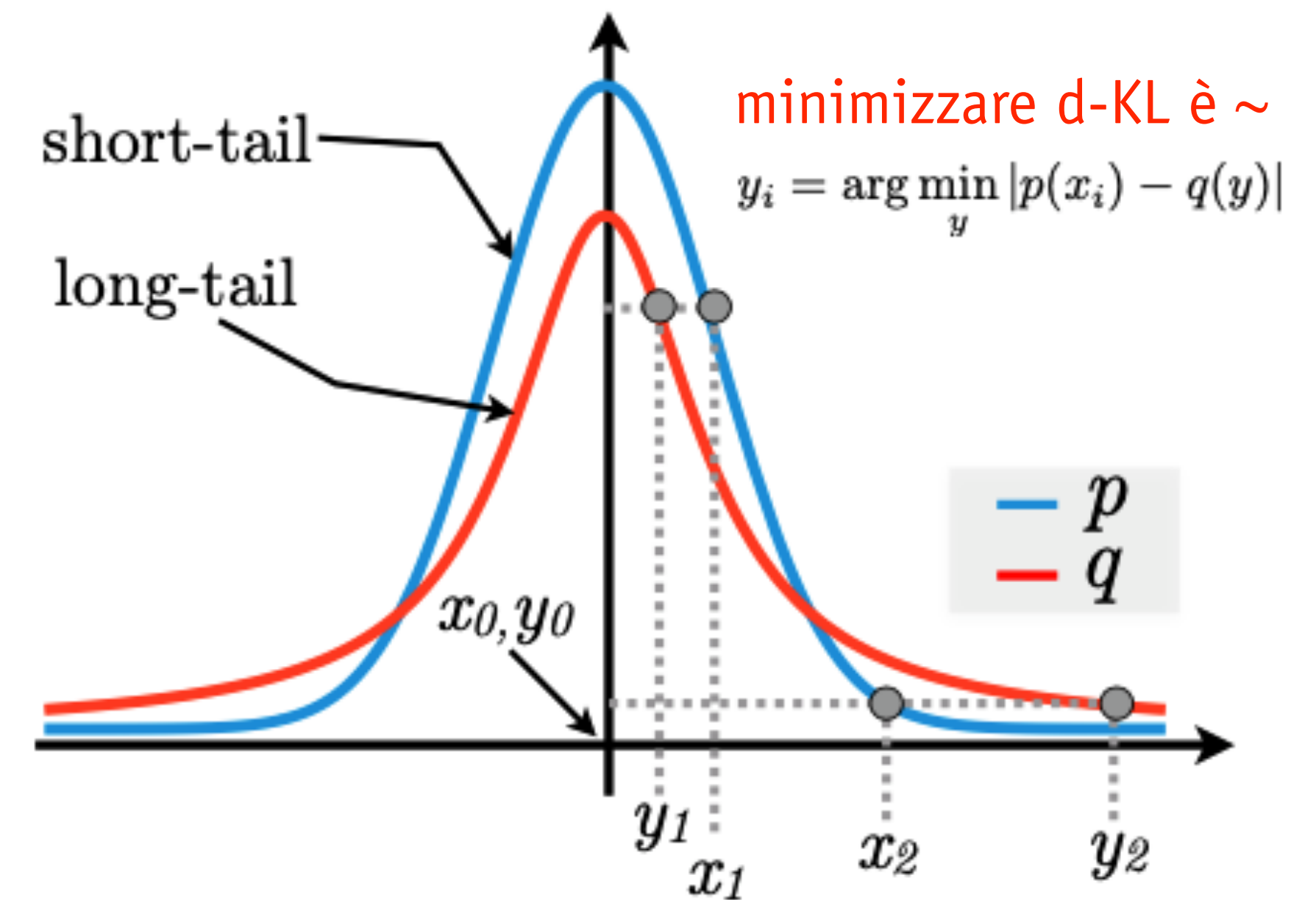
- gradiente costituito da un termine attrattivo e uno repulsivo
- il termine attrattivo induce una forza attrattiva significativa solo per punti che sono vicini nello spazio originale ( $p_{ij}$ )
- trovare  $\{y\}$  corrisponde al trovare la configurazione di equilibrio di particelle che interagiscono tramite le due forze in gioco ...



# tSNE

tSNE preserva le informazioni a breve distanza: punti vicini nello spazio originale lo sono anche in quello ridotto (non conserva le distanze quantitativamente però)

Esempio: mistura di 40 gaussiane in  $D=40$   
(stessa covarianza,  $\mu$  uniforme random in  $[-10,10]^{40}$ )



$x_1, x_2$ : prime due coordinate

PCA  $d=2$

tSNE  $d=2$

