Application of the DBSCAN clustering algorithm in High Energy Physics

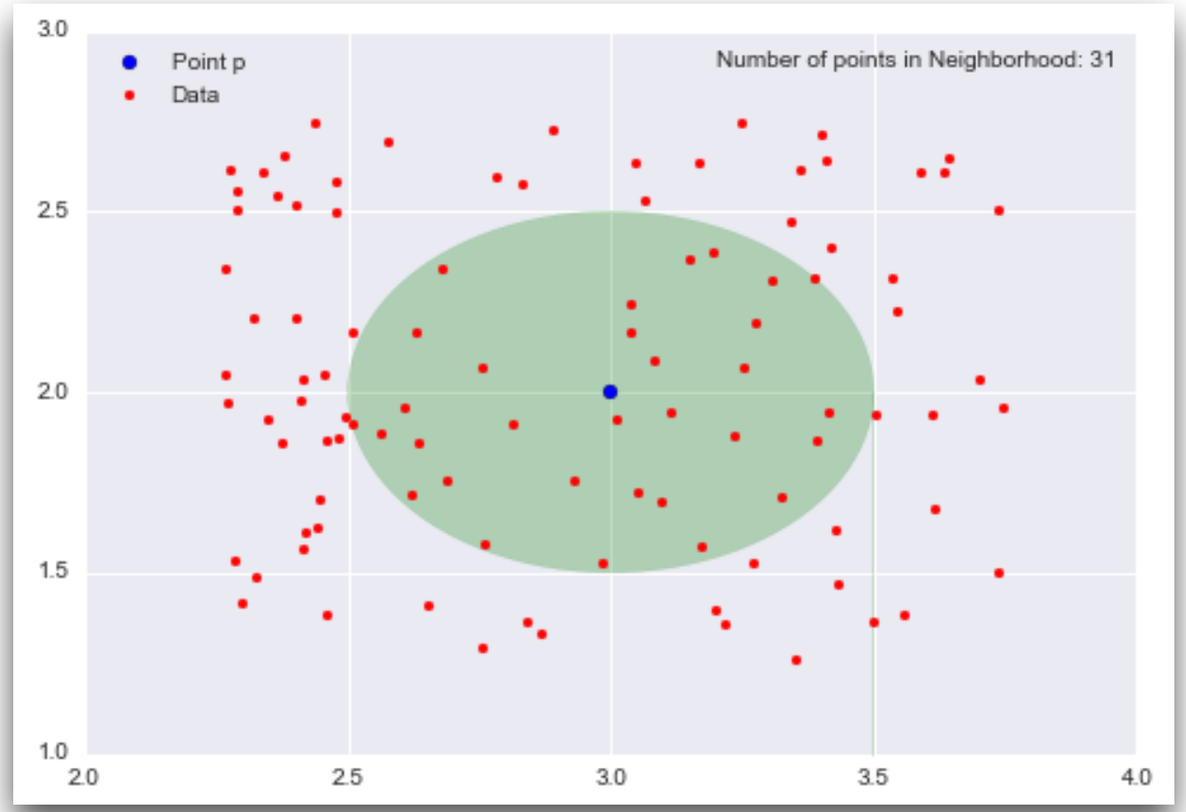
Unsupervised neural networks

Antonio Bianchi, Luca Barioglio,
Matteo Concas, Fabrizio Grosa

Neighbourhood density

- For some real value $\varepsilon > 0$ and some point p, the ε-neighbourhood of p is defined as the set of points that are at most distance ε away from p

$$Nbhd(p, \epsilon) = \{q : d(q, p) < \epsilon\}$$



- The neighbourhood of point p with $\varepsilon = 0.5$ is formed by 31 points

- The density of the neighbourhood is defined as the number of points contained within the neighbourhood (mass) divided by the volume of the neighbourhood:
  $\rho = 31/(\pi\varepsilon^2) \simeq 39.5$

https://blog.dominodatalab.com/topology-and-density-based-clustering/

Density-Based Spatial Clustering of Applications with Noise

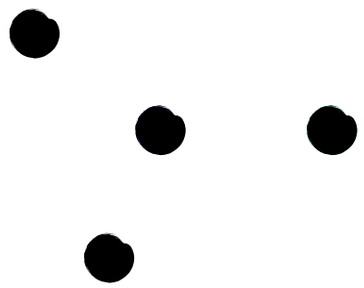
- DBSCAN is a density-based clustering algorithm, introduced in 1996 by Ester et. al. (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.71.1980)
- It does not require the number of clusters as a parameter
- It infers the number of clusters based on the data, and it can discover clusters of arbitrary shape
- To approximate the local density, two parameters are needed:
  - the radius ε of the neighbourhoods around a data point p
  - the minimum number of data points (*minPts*) we want in a neighbourhood to define a cluster
- Using these two parameters, DBSCAN categories the data points into three categories:
  - Core points
  - Border points
  - Outliers

Core points

- A data point *p* that belongs to a cluster is defined *core point* if the neighbourhood of *p* contains at least *minPts*

$$Nbhd(p, \epsilon) \geq minPts$$

- Example: let's consider a cluster obtained with *minPts* = 3 and radius ε

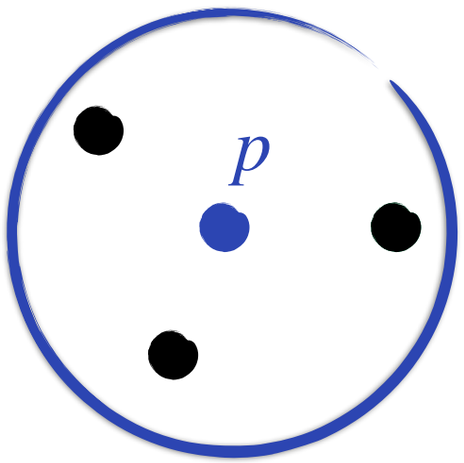


Core points

- A data point *p* that belongs to a cluster is defined *core point* if the neighbourhood of *p* contains at least *minPts*

$$Nbhd(p, \epsilon) \geq minPts$$

- Example: let's consider a cluster obtained with *minPts* = 3 and radius ε



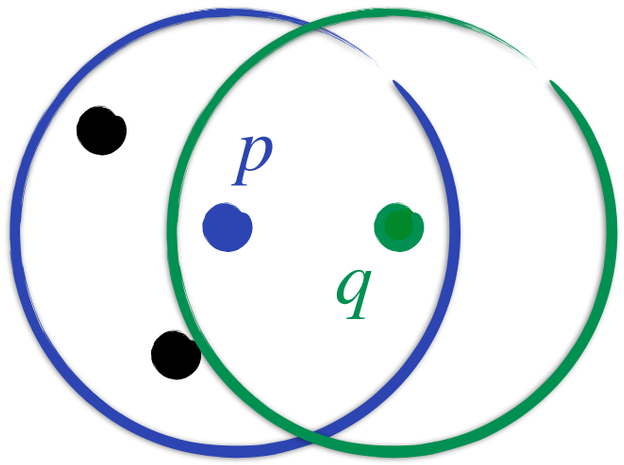
- *p* is a core point, since there are 3 points in its neighbourhood

Core points

- A data point *p* that belongs to a cluster is defined *core point* if the neighbourhood of *p* contains at least *minPts*

$$Nbhd(p, \epsilon) \geq minPts$$

- Example: let's consider a cluster obtained with *minPts* = 3 and radius ε



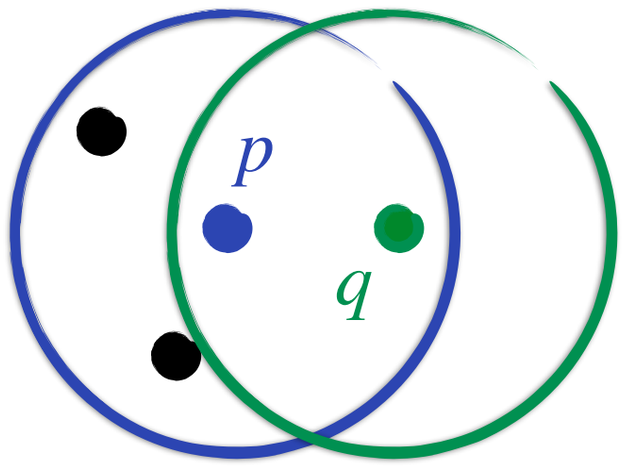
- *p* is a core point, since there are 3 points in its neighbourhood
- *q* is not a core point, since there is only 1 point in its neighbourhood

- Clusters are build around core points

Border points

- A data point q which belong to a cluster is defined *border point* if the neighbourhood of p contains at less points than *minPts*, but q is *reachable* from some *core point* p.

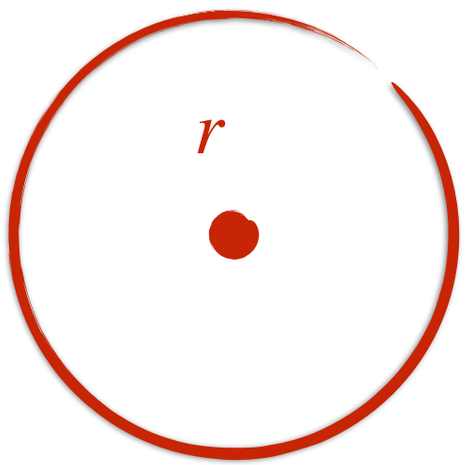
$$Nbhd(q, \epsilon) < minPts \wedge \exists p : q \in Nbhd(p, \epsilon) \geq minPts$$



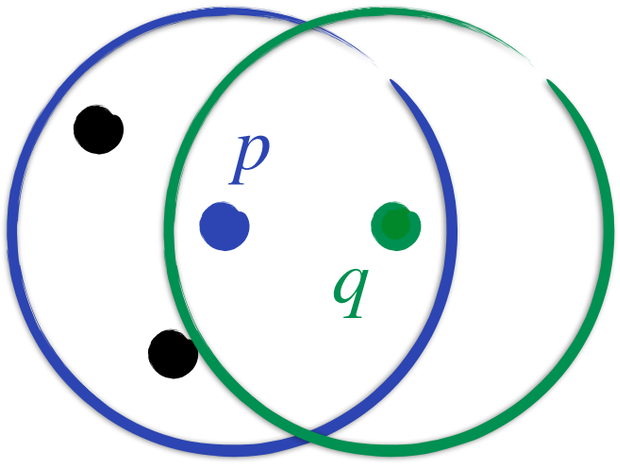
- p is a core point, since there are 3 points in its neighbourhood
- q is a border point, since there is only 1 points in its neighbourhood, but it belongs to the cluster of the core point p

Outliers

- A data point *r* which is neither a core or a border point, is defined as an *outlier*



- *r* is an outlier, since there are no other points in its neighbourhood and it does not belong to a cluster

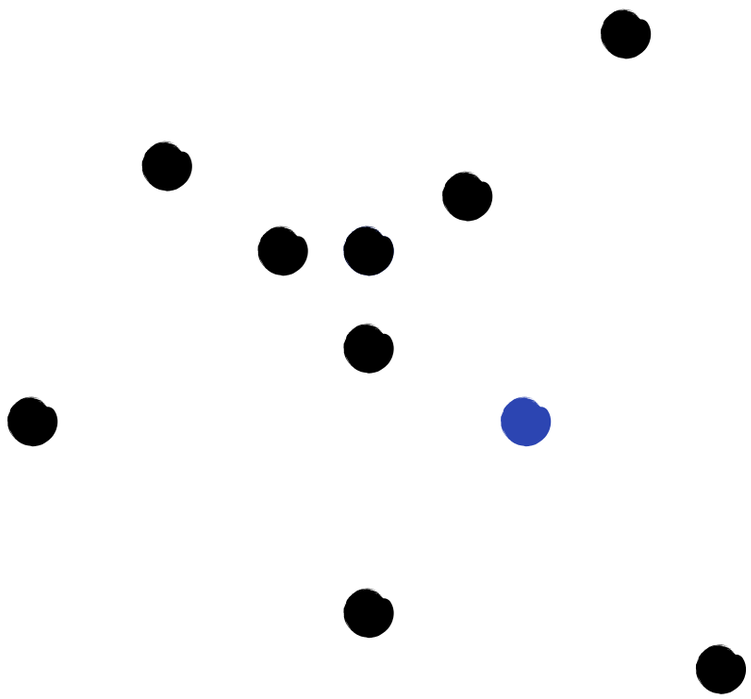


- *p* is a core point, since there are 3 points in its neighbourhood
- *q* is a border point, since there is only 1 points in its neighbourhood, but it belongs to the cluster of the core point *p*

DBSCAN algorithm

- The steps of the DBSCAN algorithm are:

  (i) Pick a random point that has not been assigned to a cluster or designated as an *outlier* and compute its neighbourhood to determine whether it's a *core point*

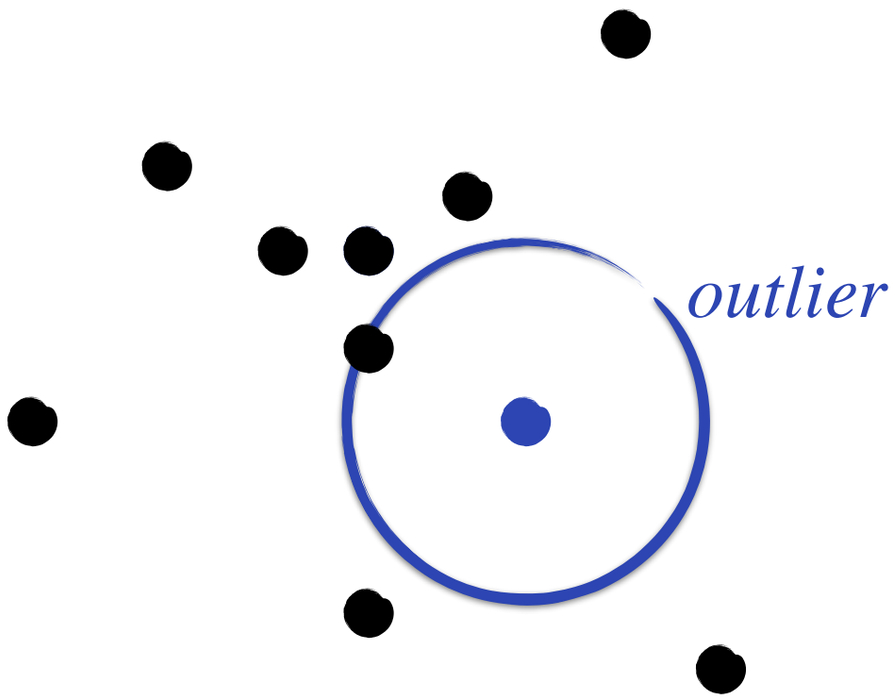


DBSCAN algorithm

- The steps of the DBSCAN algorithm are:

  (i) Pick a random point that has not been assigned to a cluster or designated as an *outlier* and compute its neighbourhood to determine whether it's a *core point*

if not, label the point as an *outlier*

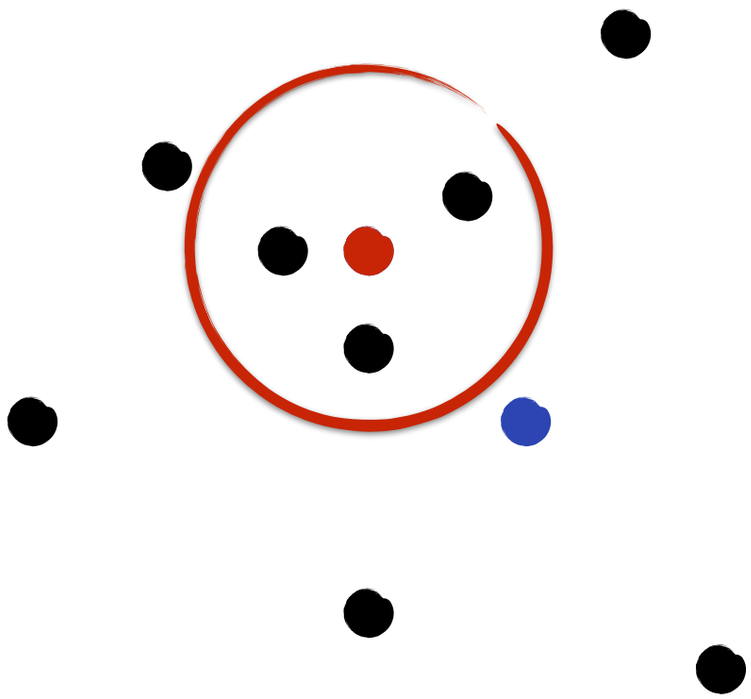


DBSCAN algorithm

- The steps of the DBSCAN algorithm are:

  (i) Pick a random point that has not been assigned to a cluster or designated as an *outlier* and compute its neighbourhood to determine whether it's a *core point*

if yes, start a cluster around this point

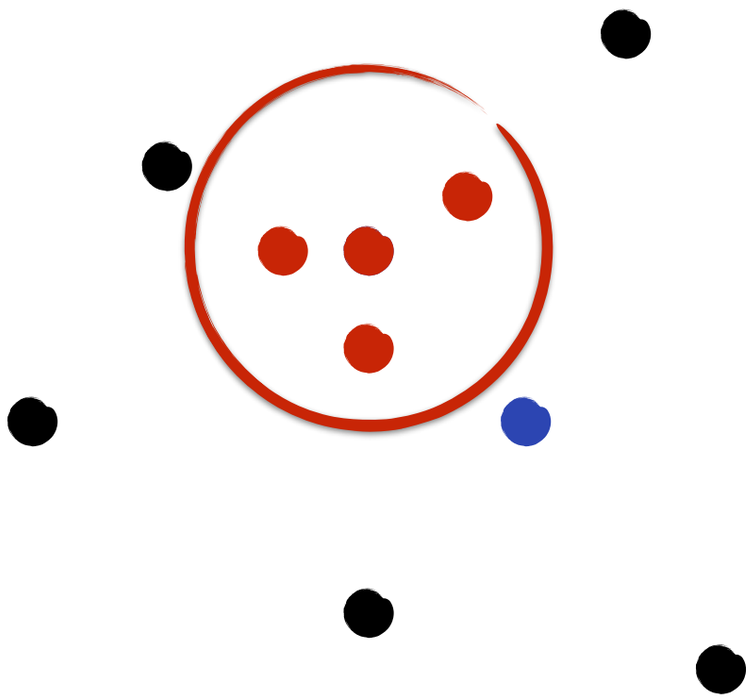


DBSCAN algorithm

- The steps of the DBSCAN algorithm are:

  (i) Pick a random point that has not been assigned to a cluster or designated as an *outlier* and compute its neighbourhood to determine whether it's a *core point*

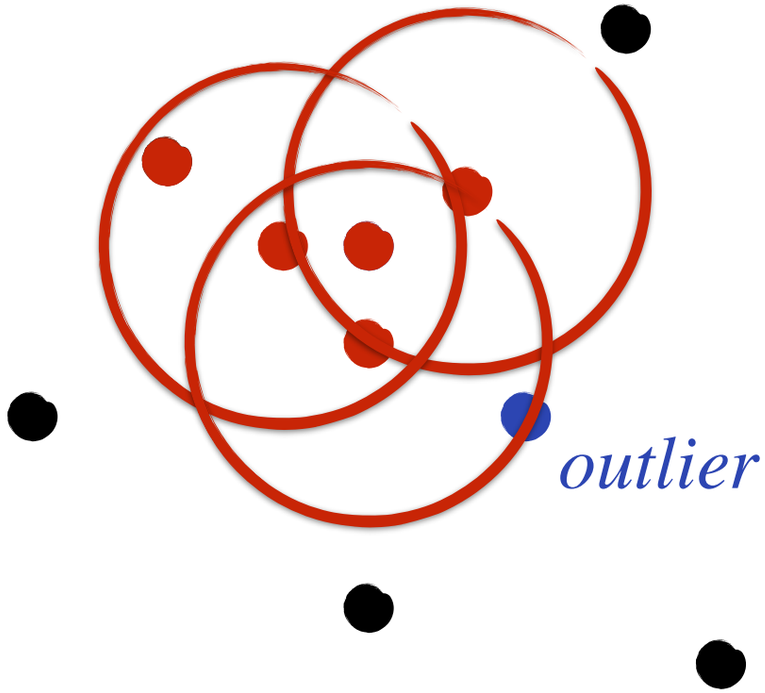
if yes, start a cluster around this point



DBSCAN algorithm

The steps of the DBSCAN algorithm are:

(i) Pick a random point that has not been assigned to a cluster or designated as an *outlier* and compute its neighbourhood to determine whether it's a *core point*

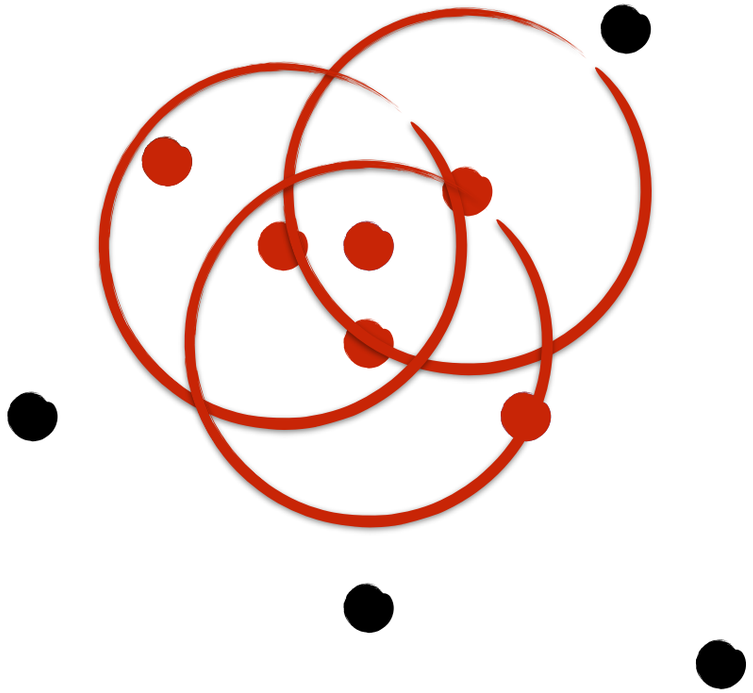


(ii) Once we find a *core point* and thus a cluster, expand the cluster by adding all *directly-reachable* points to the cluster. Perform "neighbourhood jumps" to find all *density-reachable* points and add them to the cluster.

DBSCAN algorithm

The steps of the DBSCAN algorithm are:

(i) Pick a random point that has not been assigned to a cluster or designated as an *outlier* and compute its neighbourhood to determine whether it's a *core point*

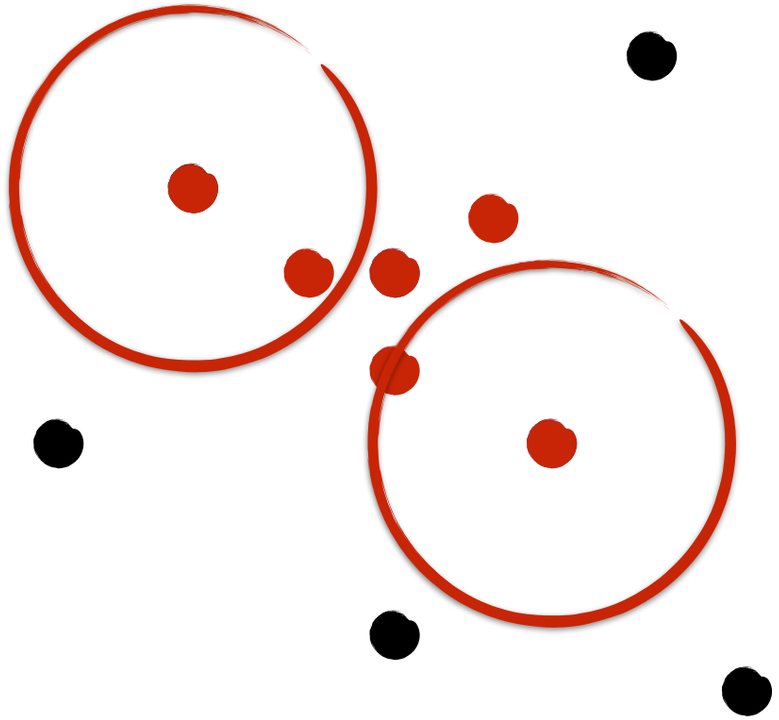


(ii) Once we find a *core point* and thus a cluster, expand the cluster by adding all *directly-reachable* points to the cluster. Perform "neighbourhood jumps" to find all *density-reachable* points and add them to the cluster. If an *outlier* is added, change that point's status from *outlier* to *border point*

DBSCAN algorithm

The steps of the DBSCAN algorithm are:

(i) Pick a random point that has not been assigned to a cluster or designated as an *outlier* and compute its neighbourhood to determine whether it's a *core point*

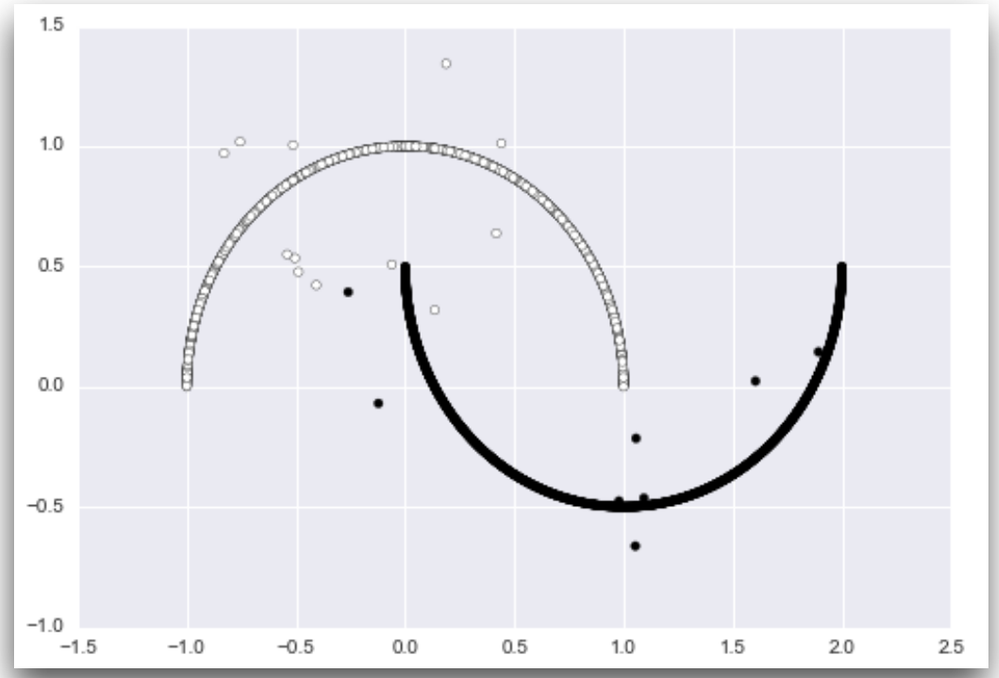


(ii) Once we find a *core point* and thus a cluster, expand the cluster by adding all *directly-reachable* points to the cluster. Perform "neighbourhood jumps" to find all *density-reachable* points and add them to the cluster. If an *outlier* is added, change that point's status from *outlier* to *border point*

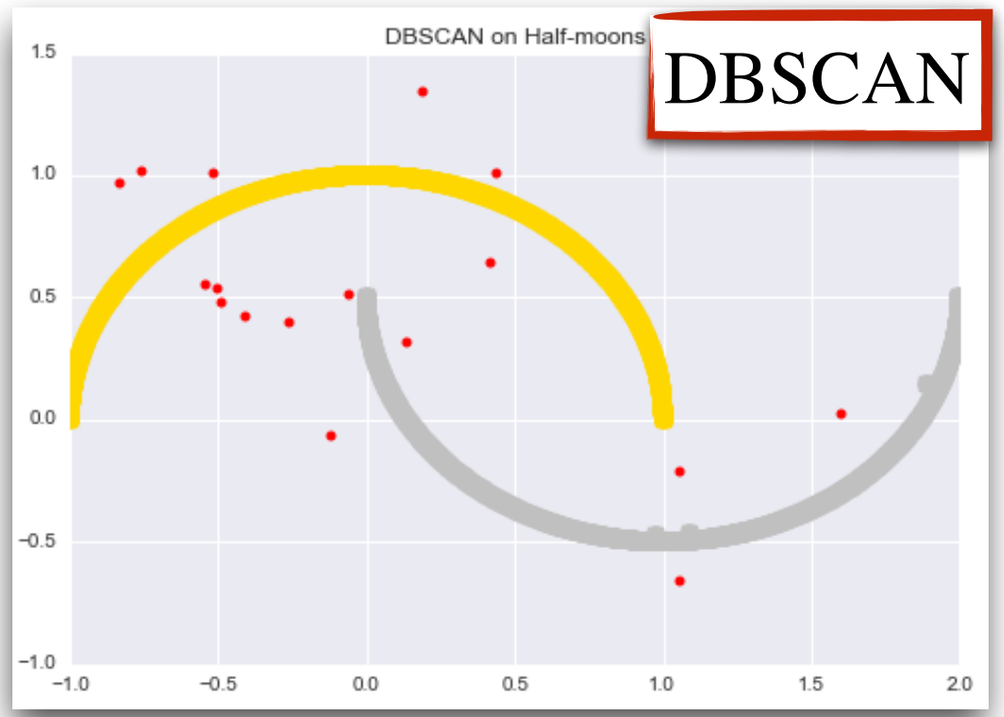
(iii) Repeat these two steps until all points are either assigned to a cluster or designated as an *outlier*

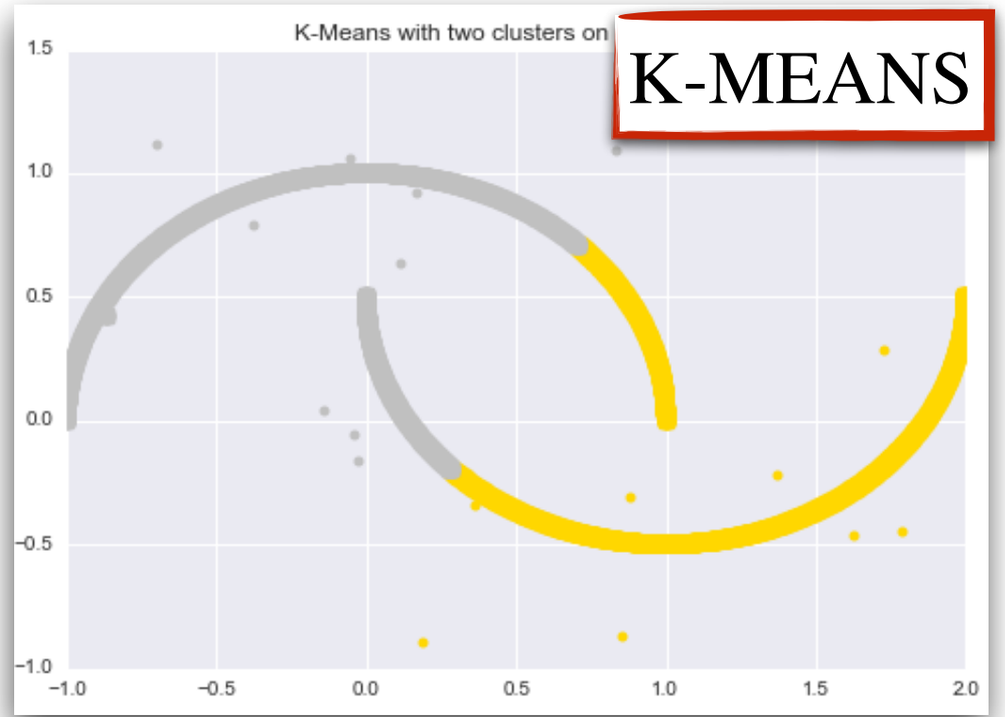
DBSCAN vs. k-MEANS

- Example: consider the following dataset:
  - (i) two “half-moon” clusters
  - (ii) some noise points



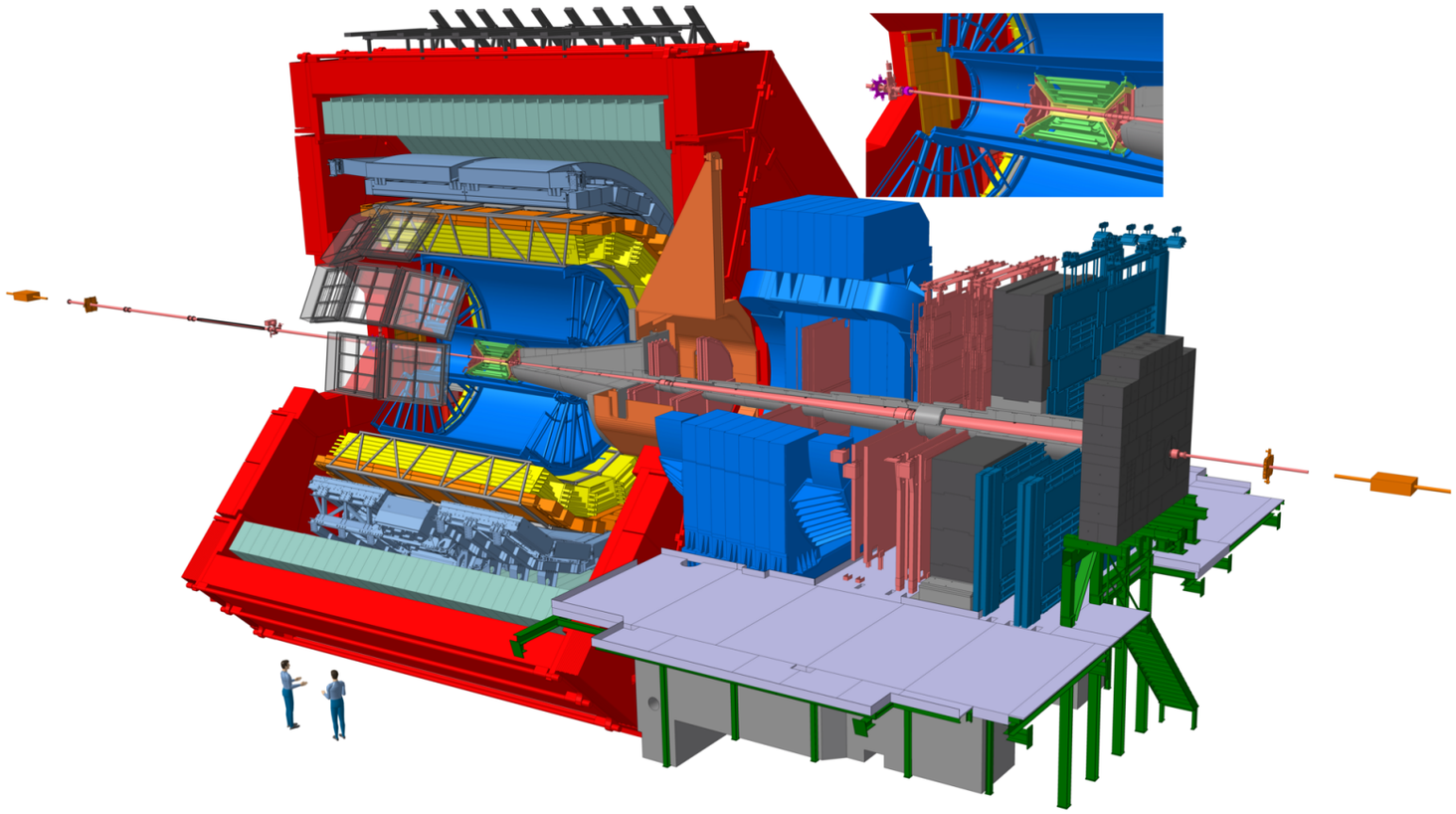
- Using DBSCAN the two clusters are recognised and most of the noise points tagged as “outliers”



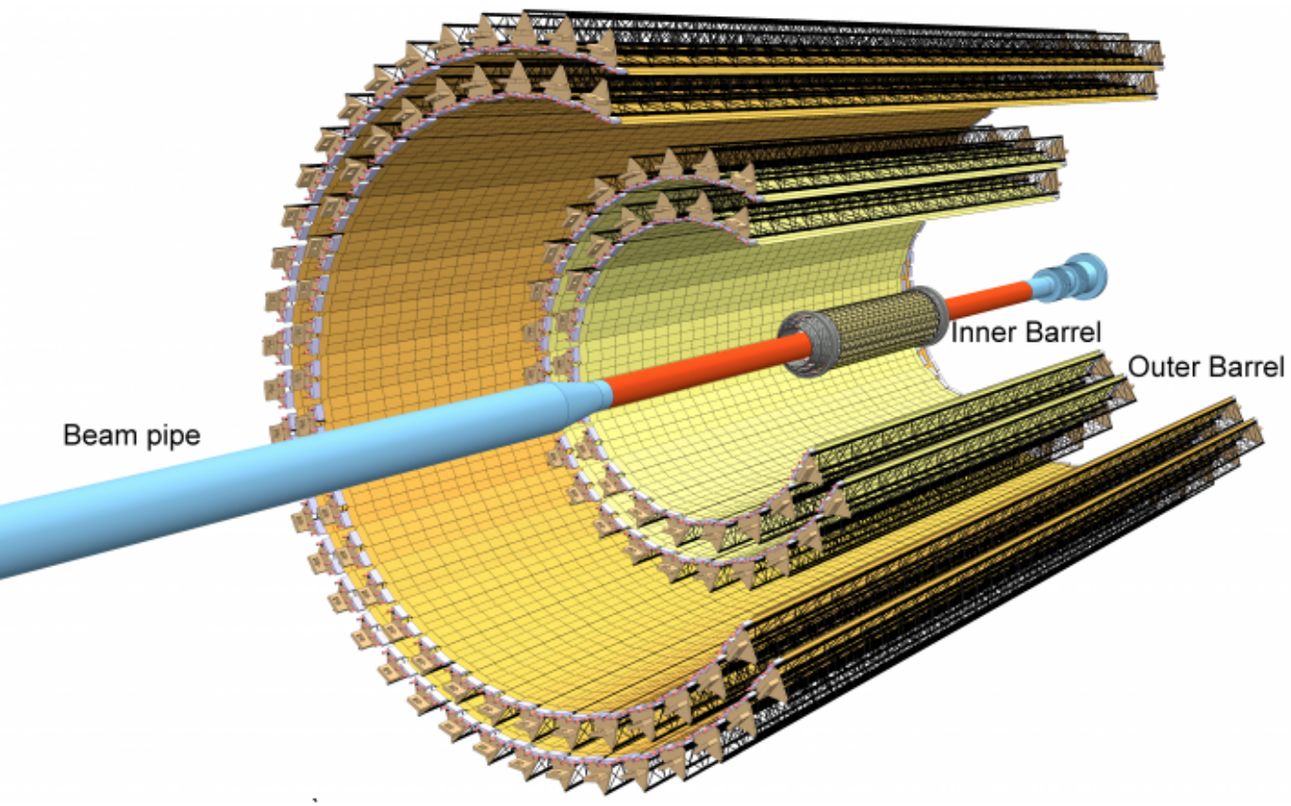


A Large Ion Collider Experiment

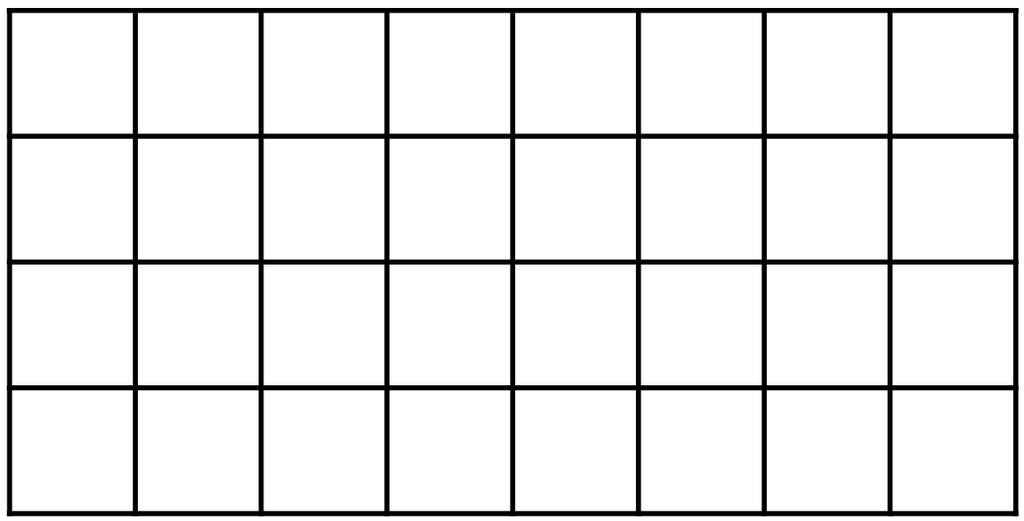
- ALICE is one of the four LHC experiments at CERN
- It is designed to study the matter created in ultra-relativistic heavy-ion collisions
- In each collision many particles are created
- Trajectories of charged particles are reconstructed using various detectors
- The closest detector to the interaction point is the Inner Tracking System (ITS), which consists of several cylindrical layers of silicon sub-detectors



Detection of charged particles with the ITS

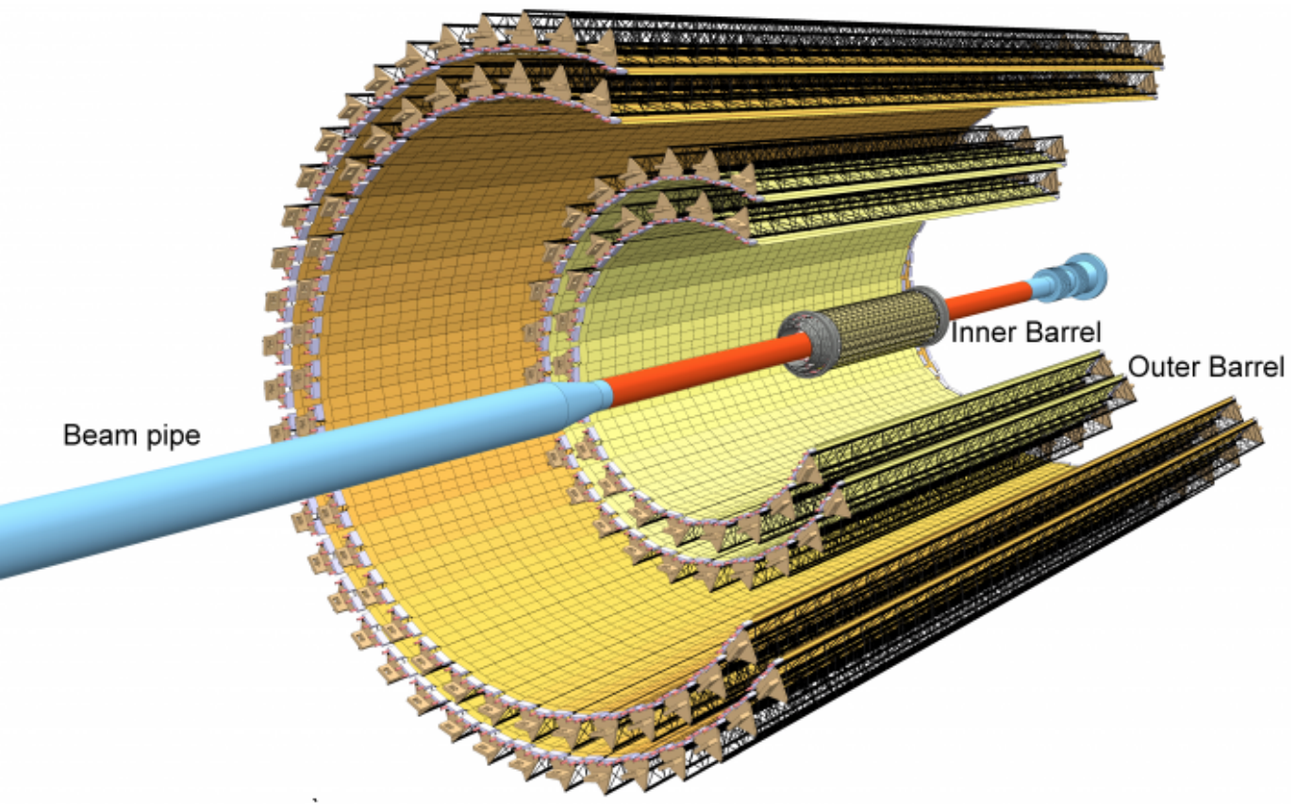


→ When the charged particles pass through each layer, they fire a certain amount of pixels, depending on their energy loss

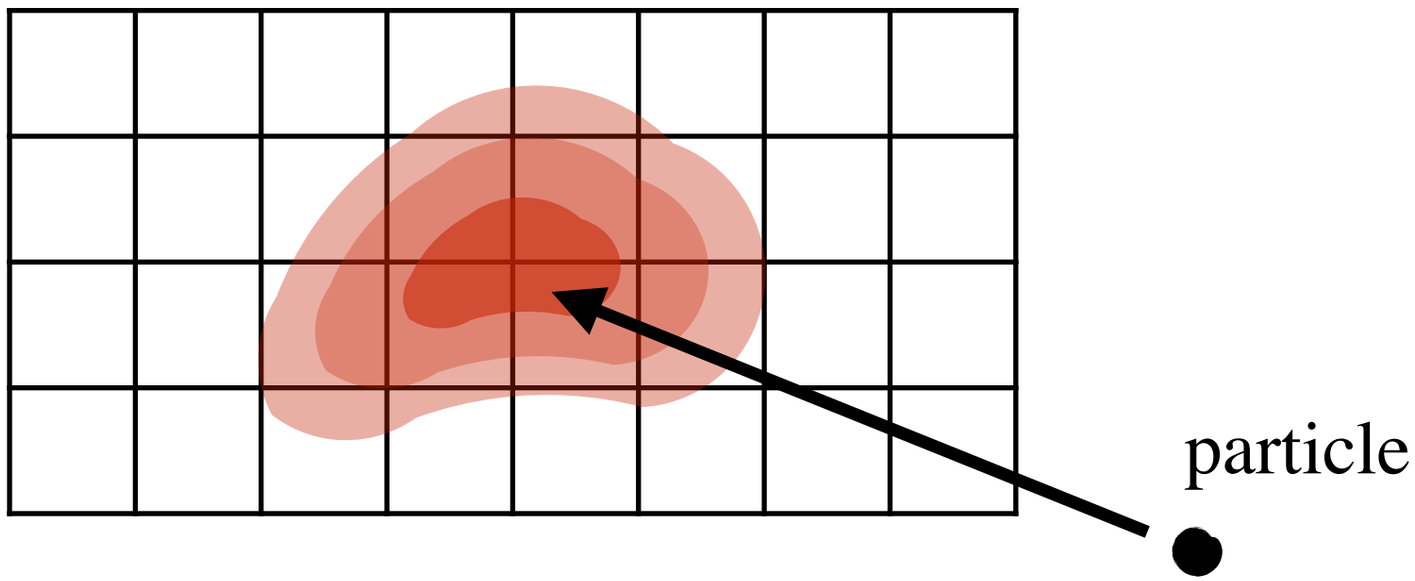


particle

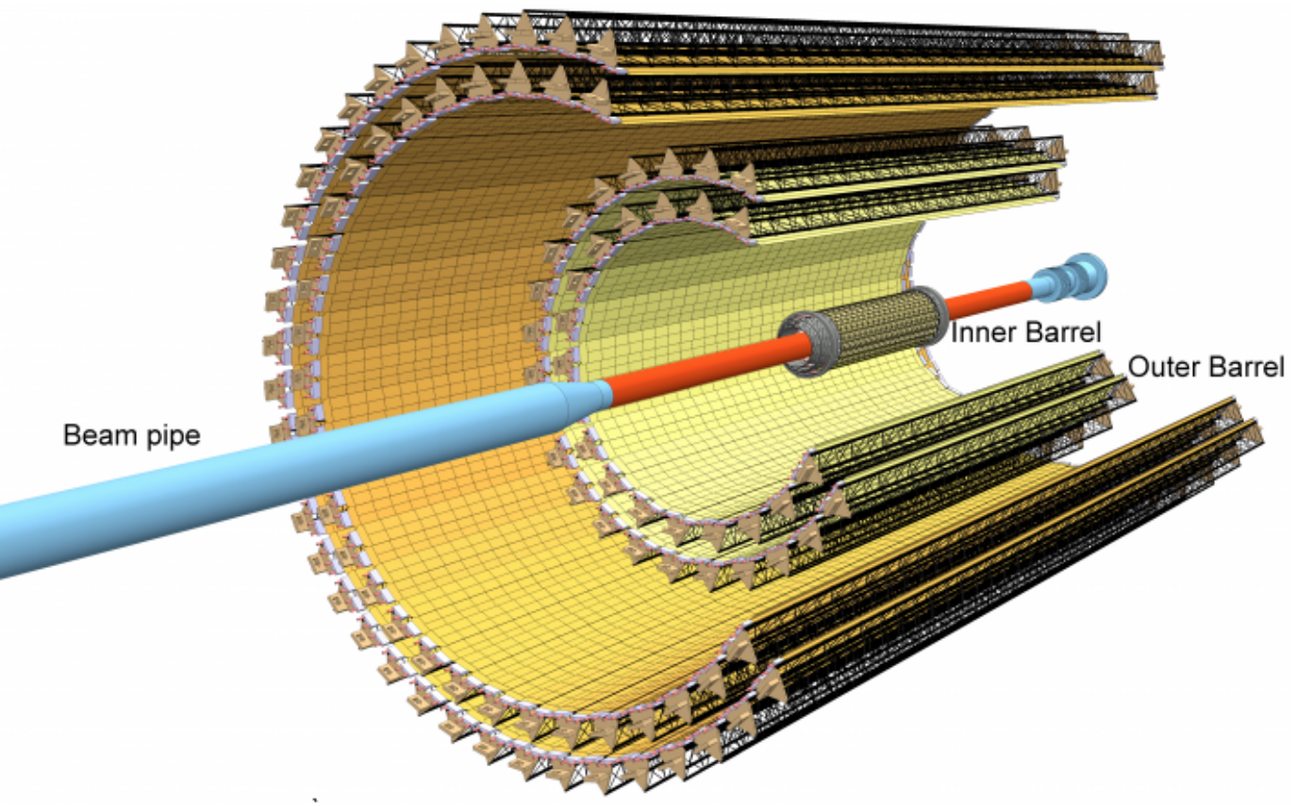
Detection of charged particles with the ITS



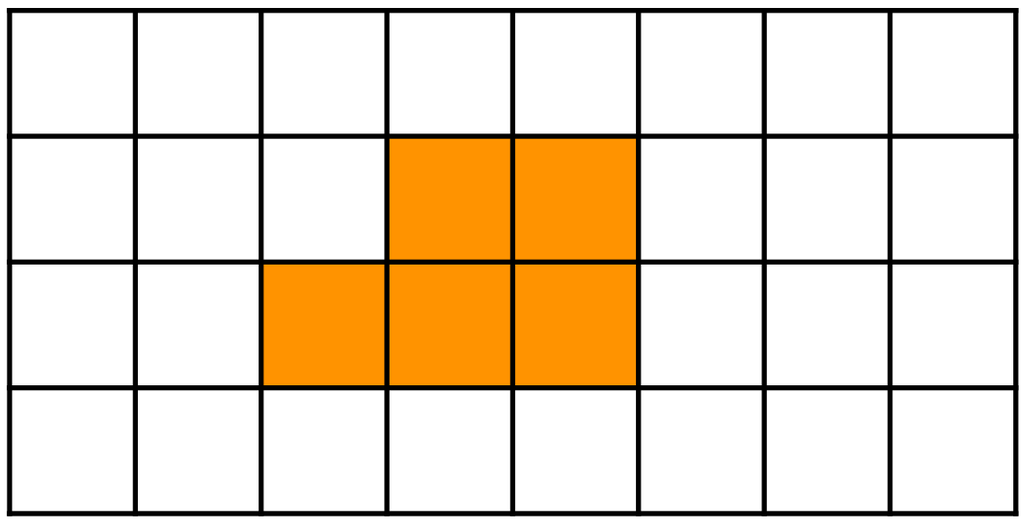
- When the charged particles pass through each layer, they fire a certain amount of pixels, depending on their energy loss



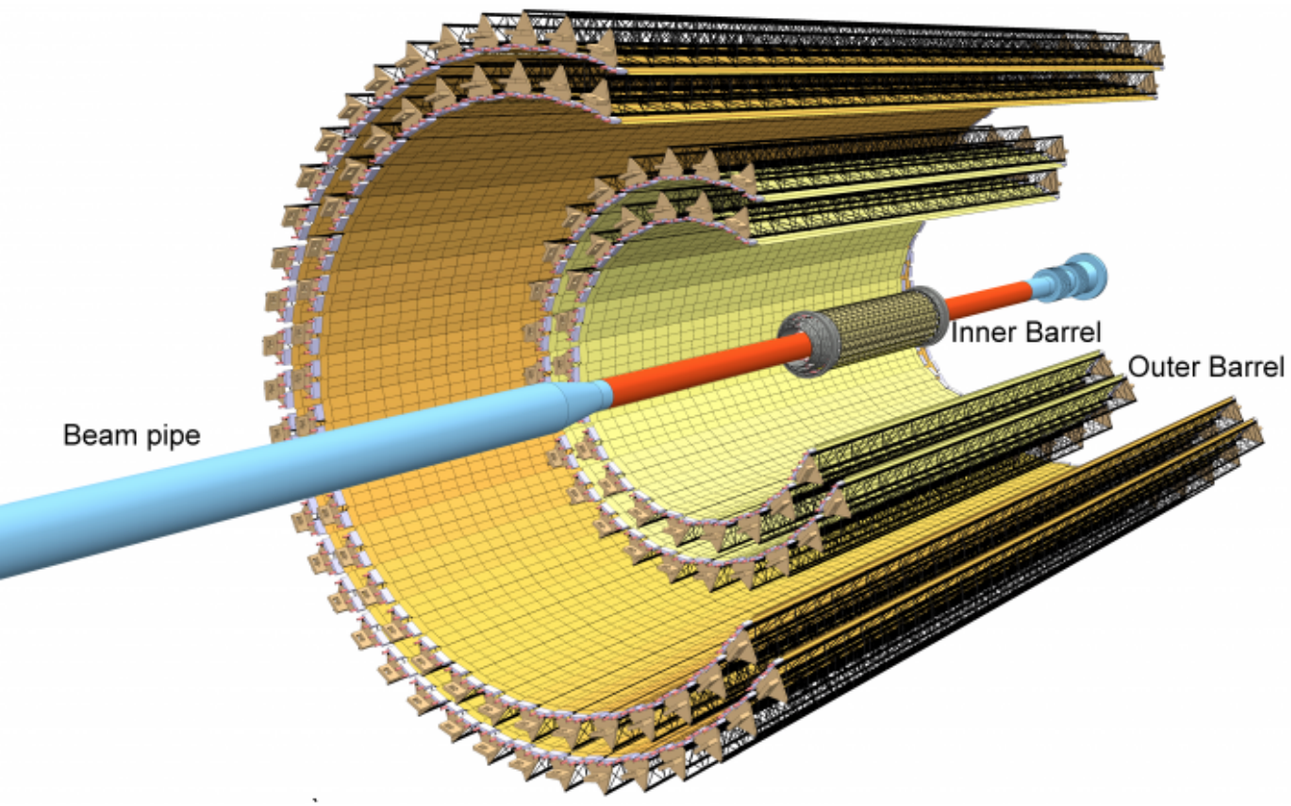
Detection of charged particles with the ITS



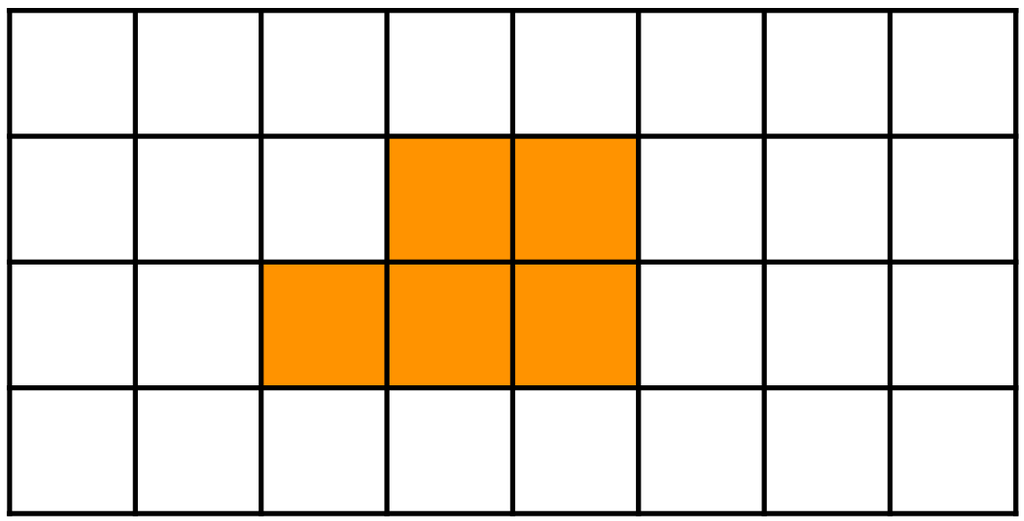
- When the charged particles pass through each layer, they fire a certain amount of pixels, depending on their energy loss



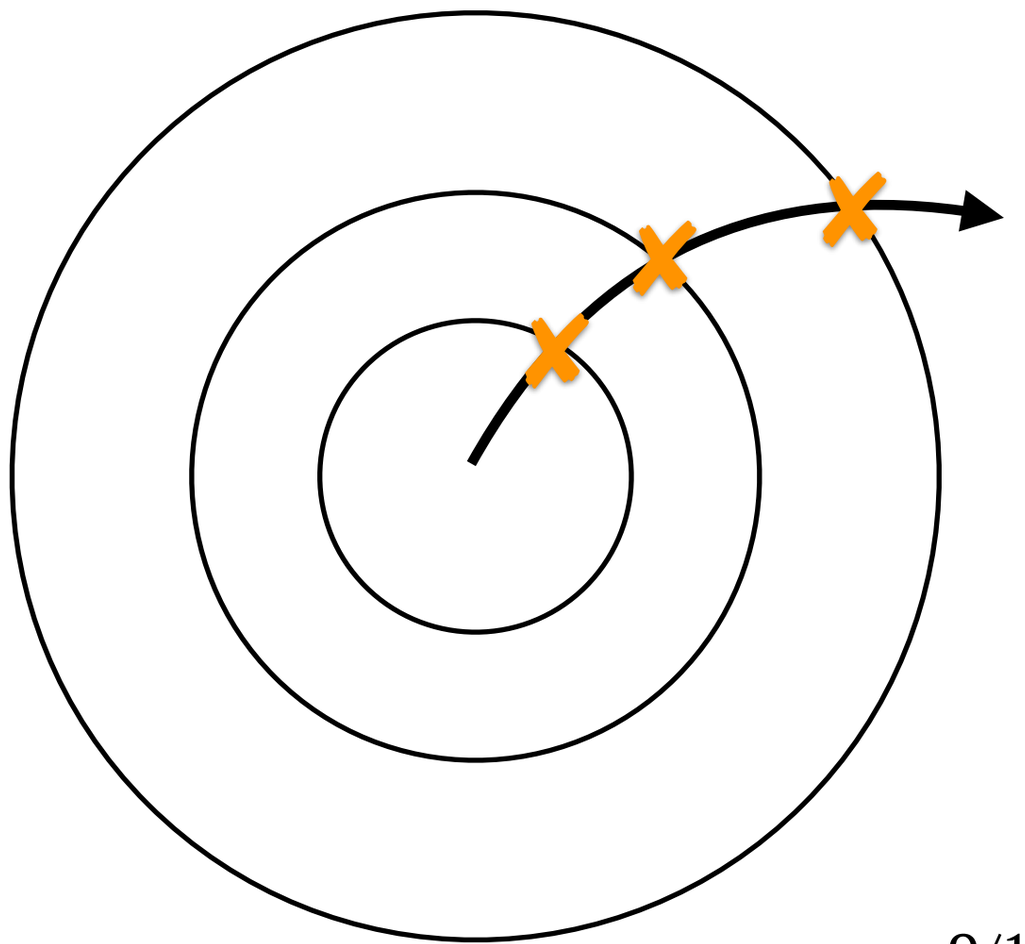
Detection of charged particles with the ITS



→ When the charged particles pass through each layer, they fire a certain amount of pixels, depending on their energy loss

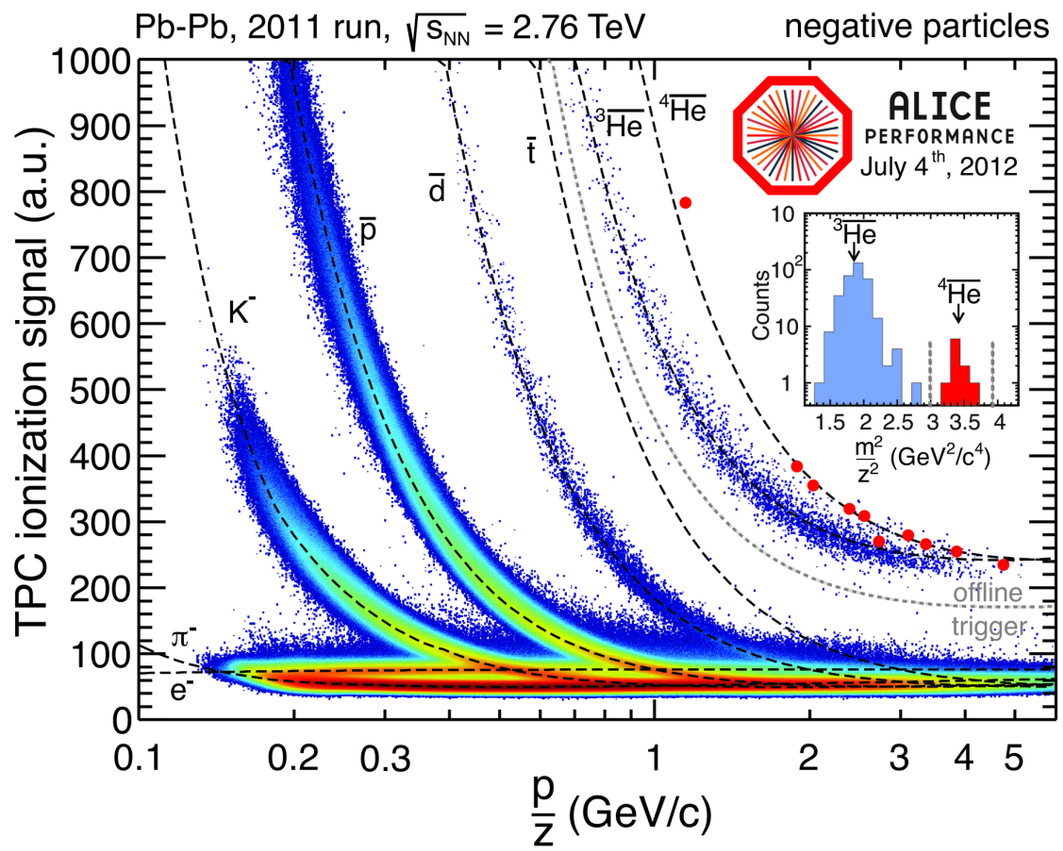


→



Data simulation

- For the simulation of the data two different particle species were considered:
  - (i) pions ($\pi^{\pm}$)
  - (ii) helium nuclei (^{3}He)



- The cluster size of pions and helium nuclei is expected to be different because of their energy loss

- Simulated transport of 1000 pions and 250 helium nuclei only through the innermost layer of the detector with a Monte Carlo (no noise included)

- Coordinates of the fired pixels are the input data for the DBSCAN algorithm

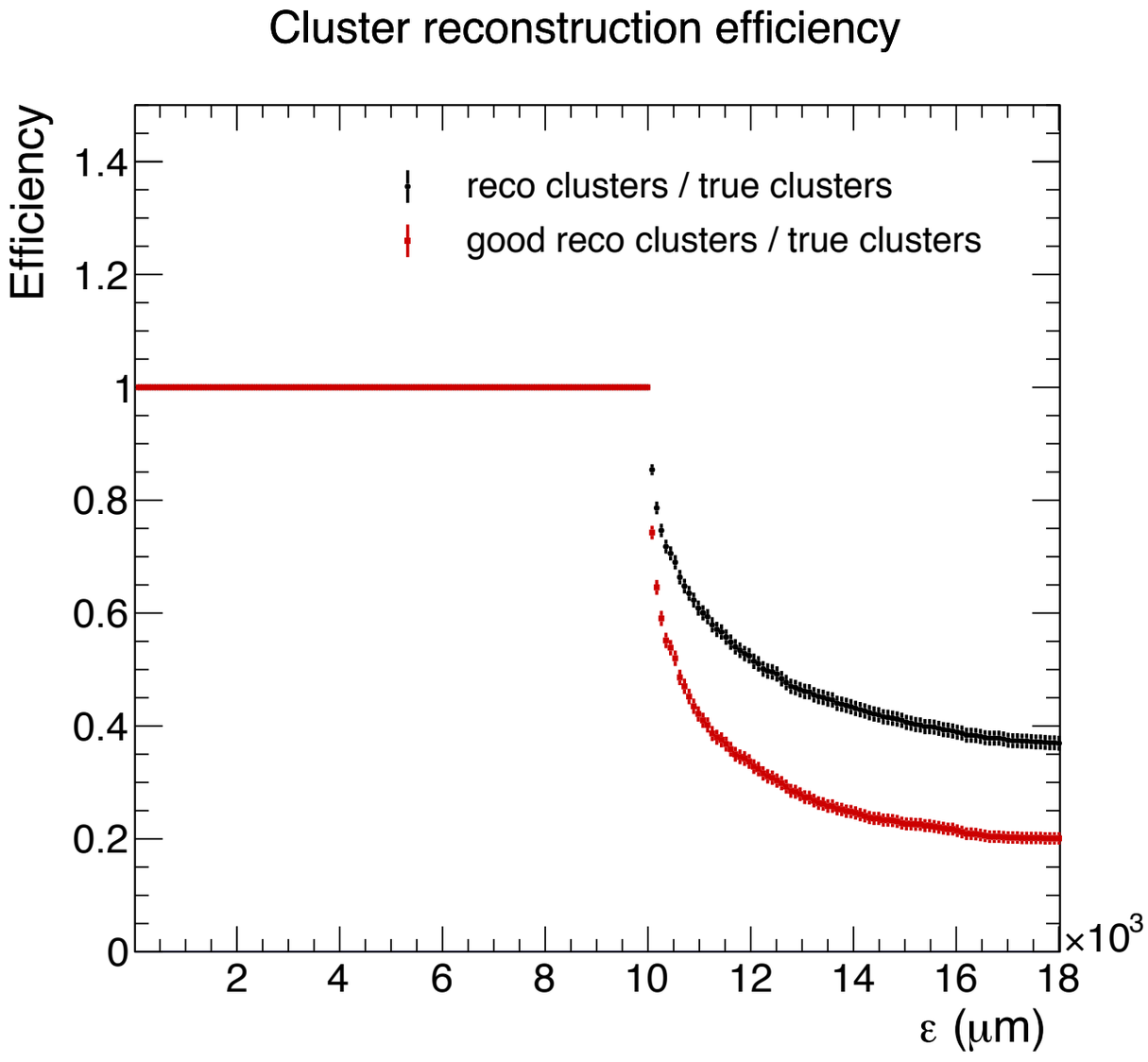
Application of the DBSCAN algorithm

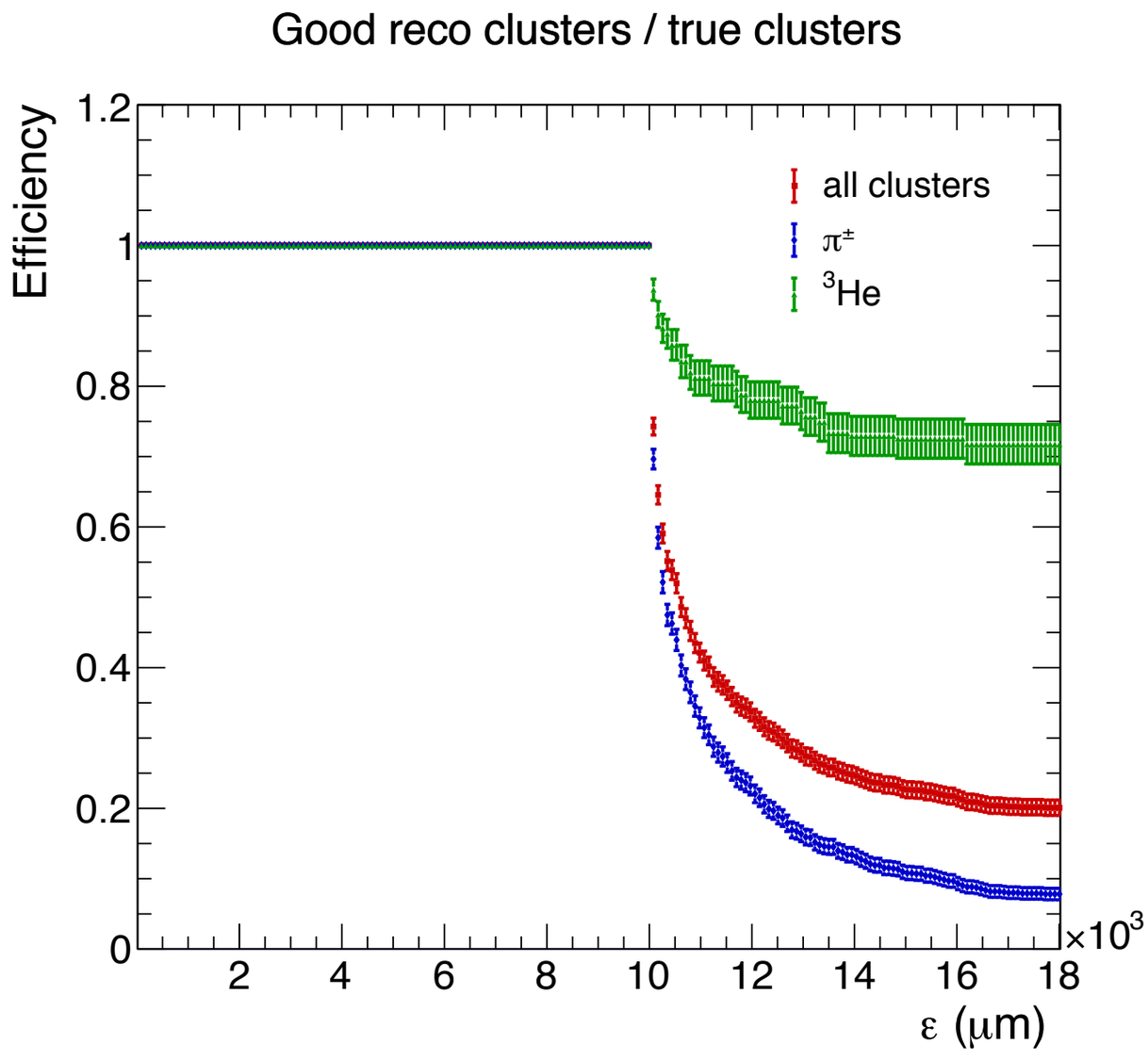
- Python implementation with `sklearn` module (https://pypi.python.org/pypi/scikit-learn/)
- Analysis with `ROOT` framework (https://root.cern.ch/)
- Study of the cluster reconstruction efficiency as a function of the parameters ε and *minPts*
- Quantities of interest:
  (i) number of reconstructed clusters
  (ii) number of good reconstructed clusters → clusters formed by pixels fired by the same particle

Results - *minPts* fixed

The efficiency of the algorithm was tested varying ε, keeping *minPts* fixed

$minPts = 1$

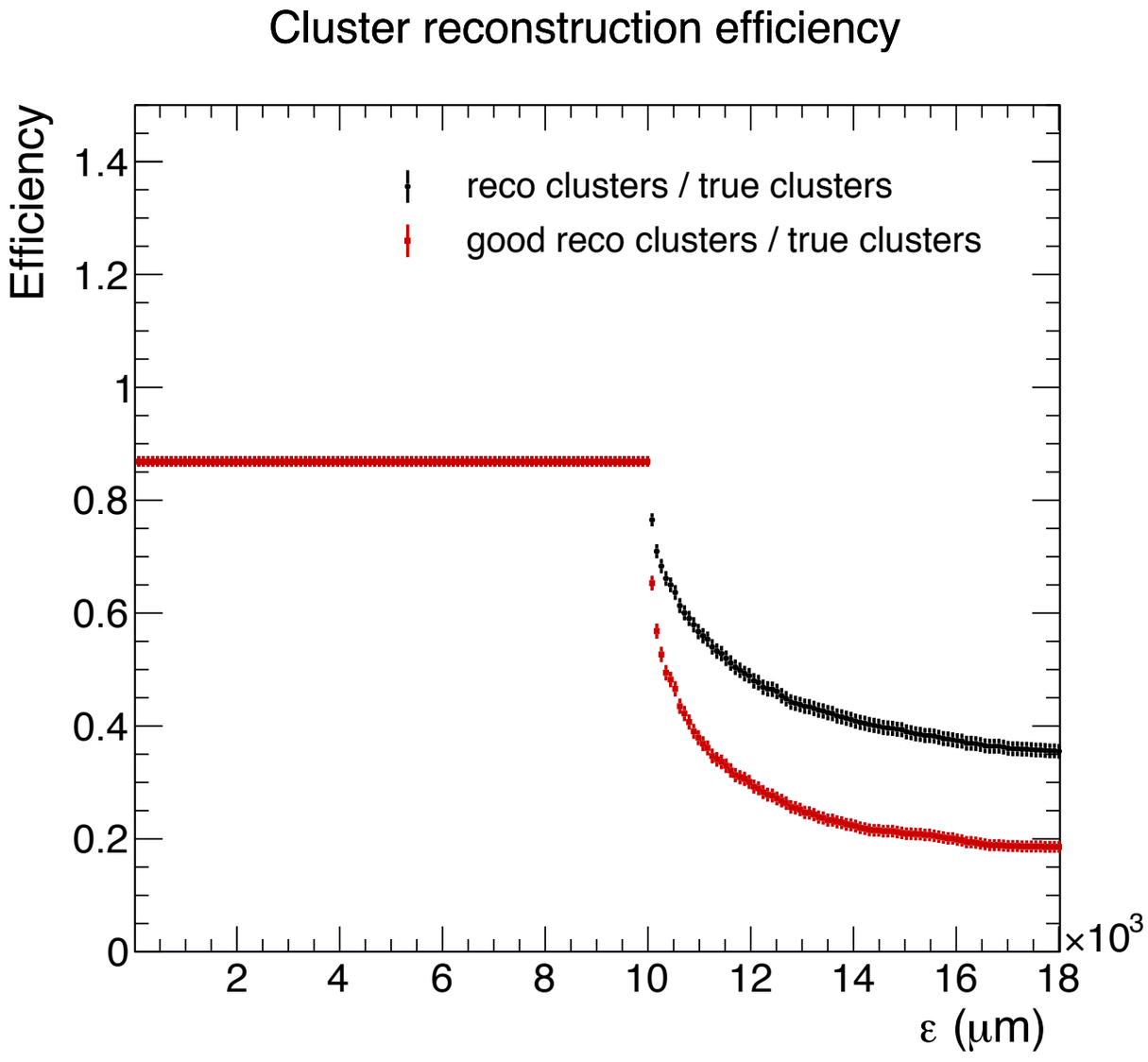


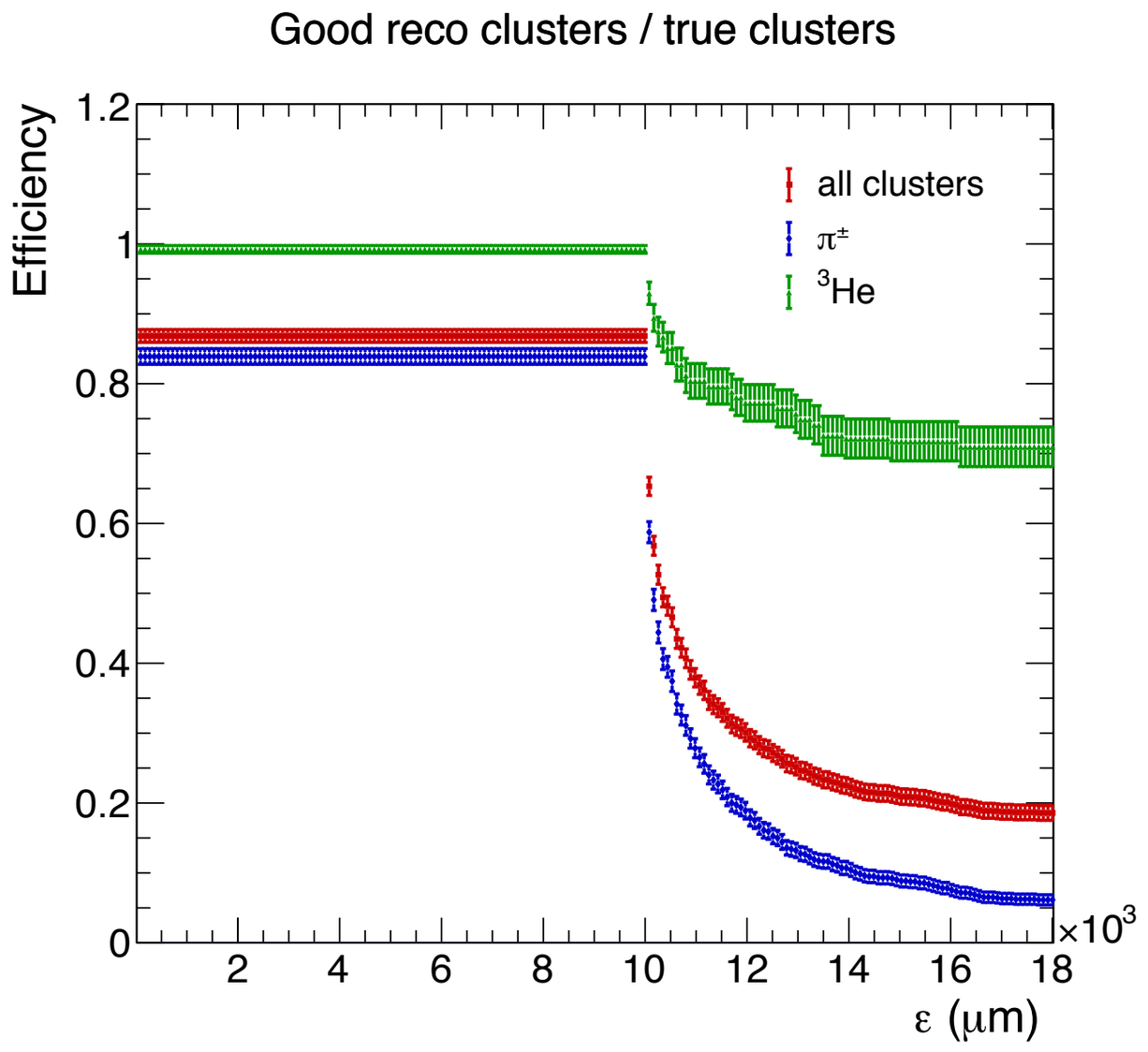


Results - *minPts* fixed

The efficiency of the algorithm was tested varying ε, keeping *minPts* fixed

$minPts = 2$

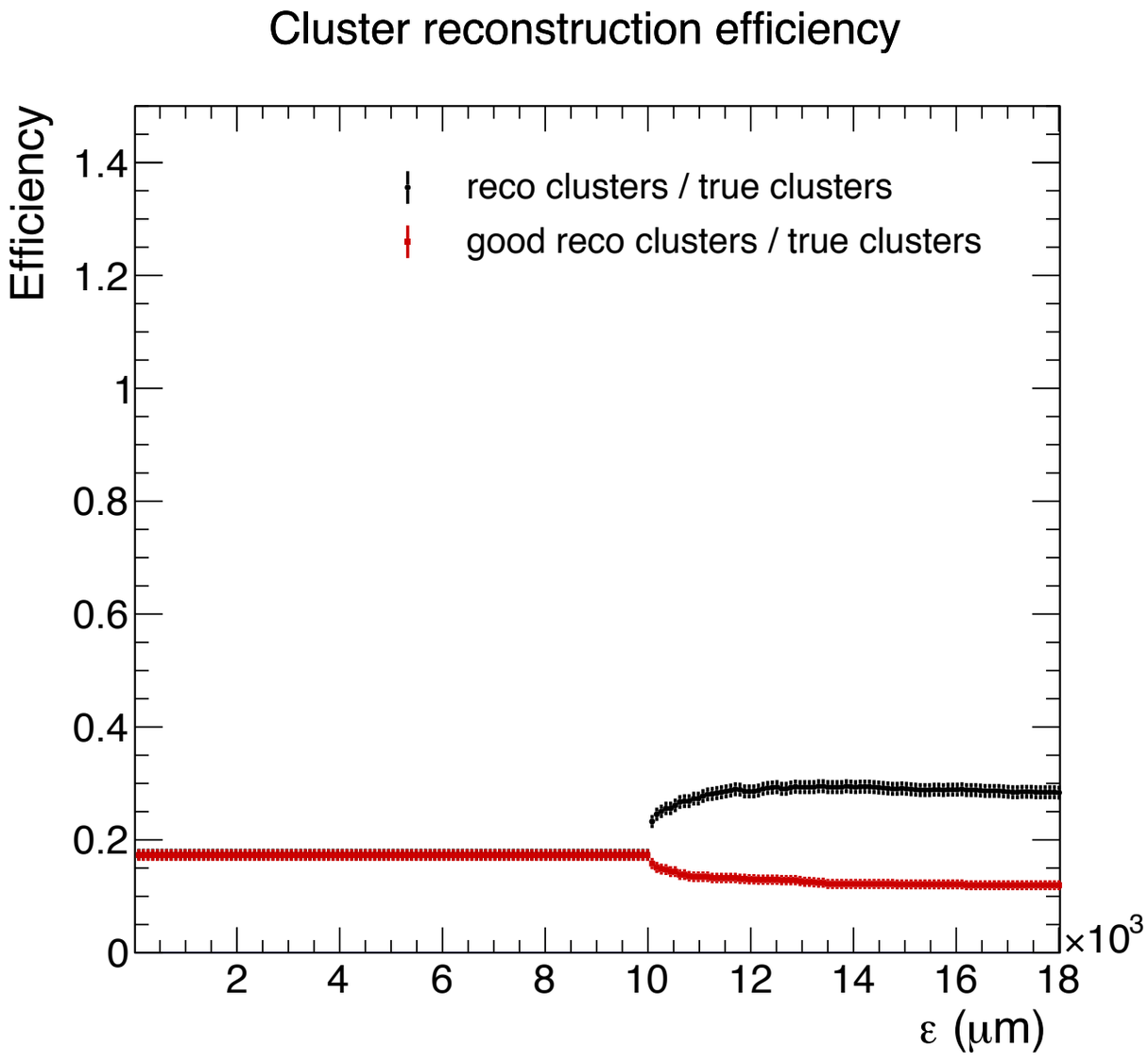


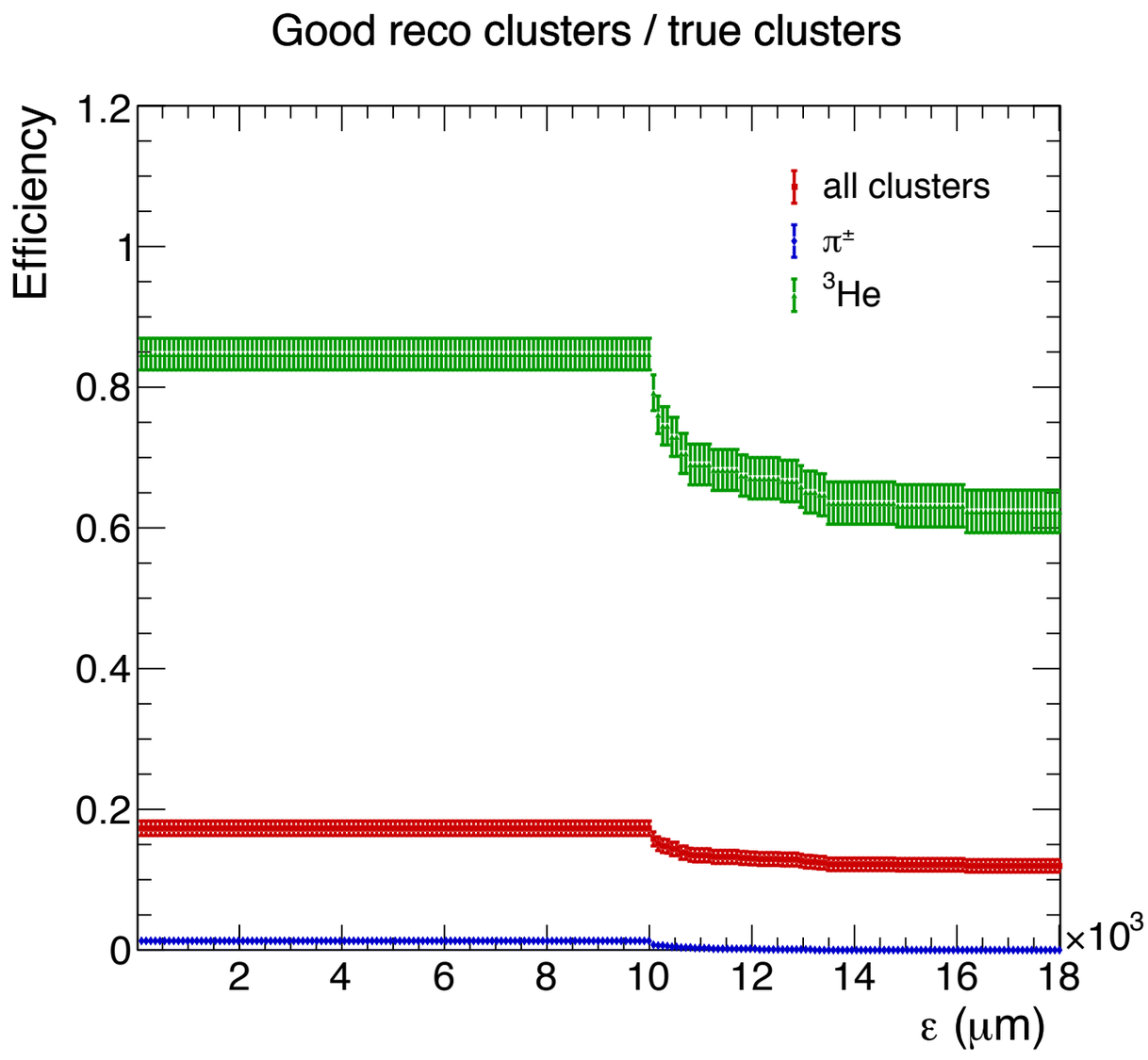


Results - *minPts* fixed

- The efficiency of the algorithm was tested varying ε, keeping *minPts* fixed

$minPts = 5$

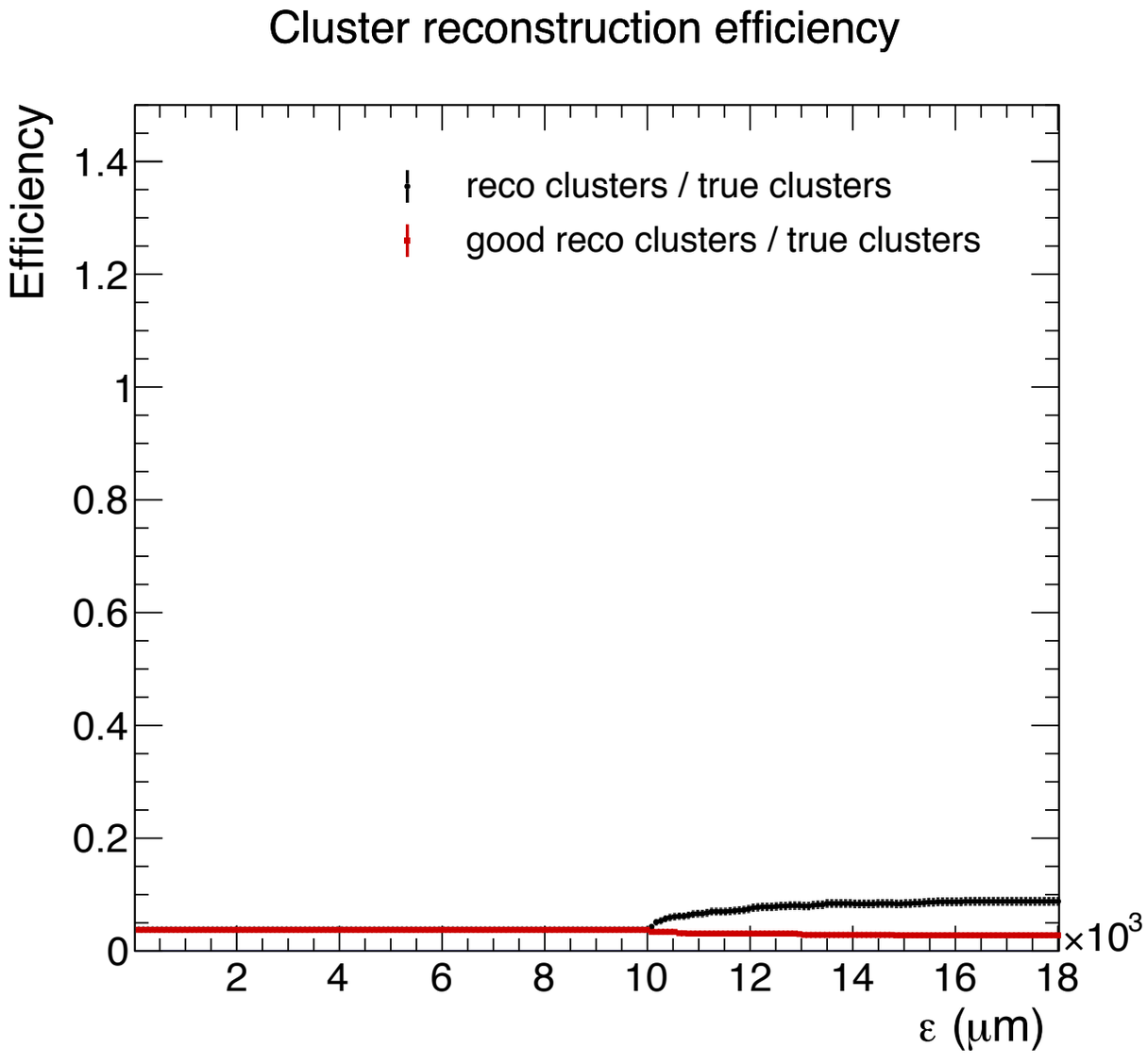


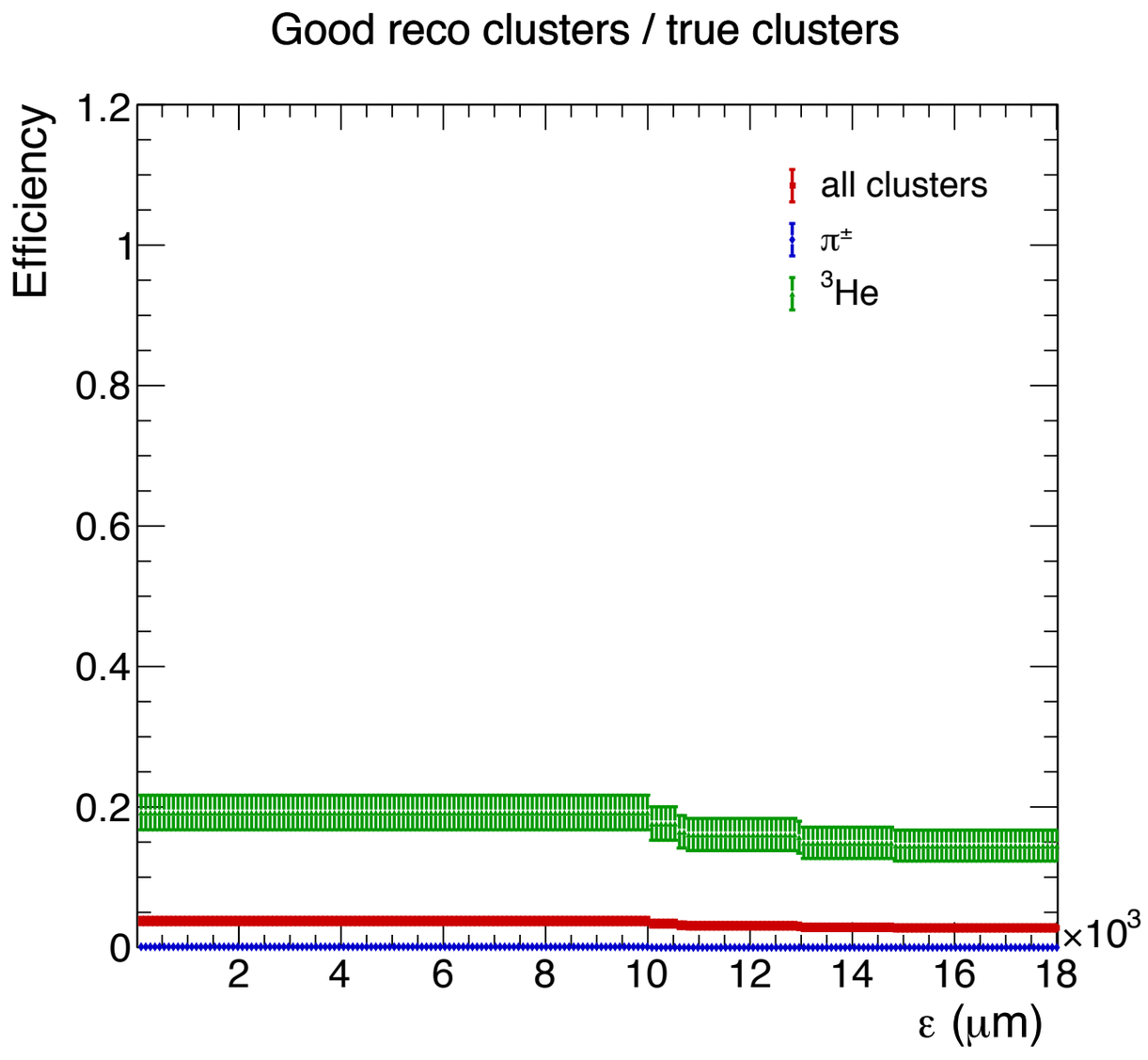


Results - *minPts* fixed

The efficiency of the algorithm was tested varying ε, keeping *minPts* fixed

$minPts = 10$

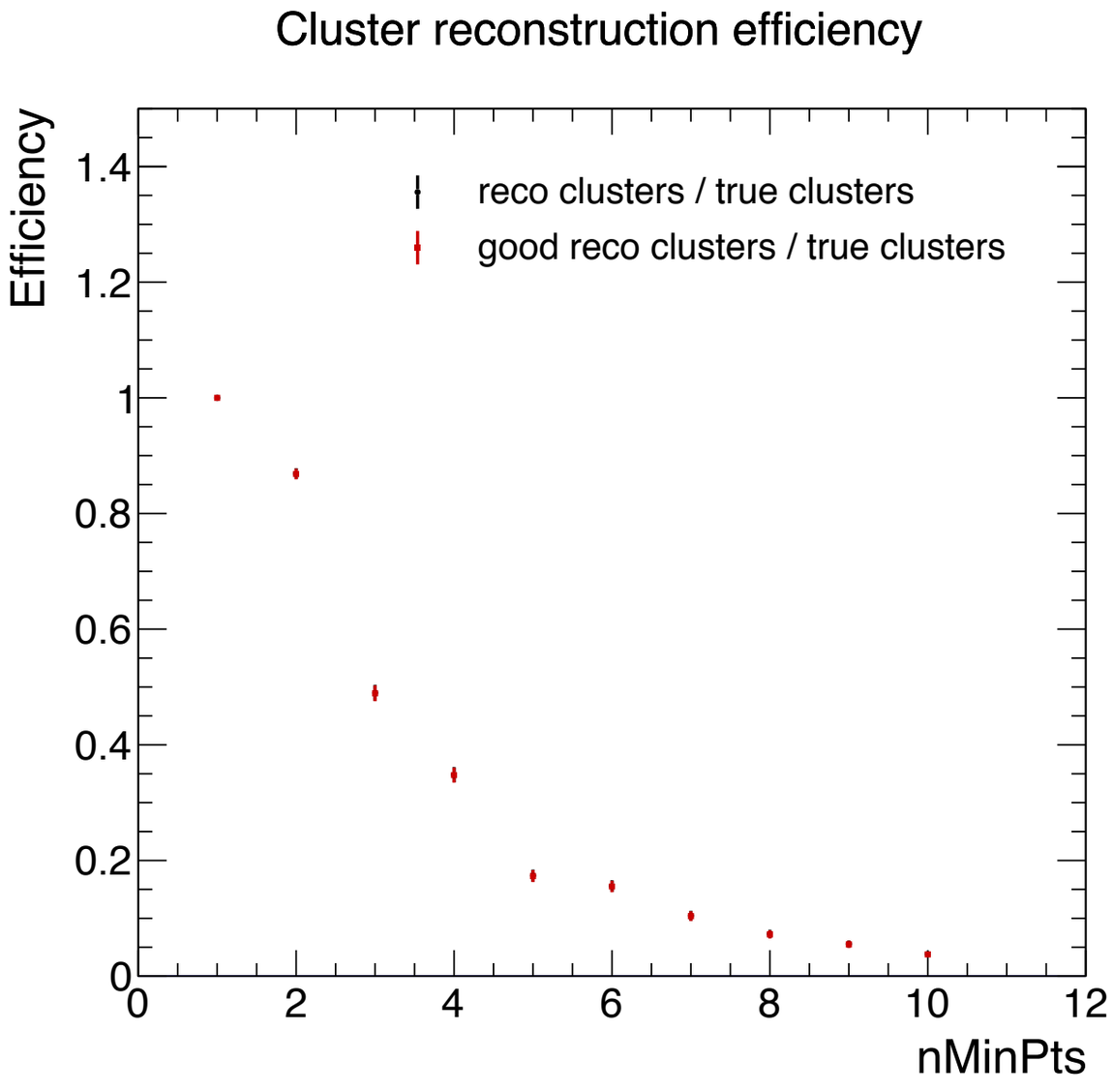


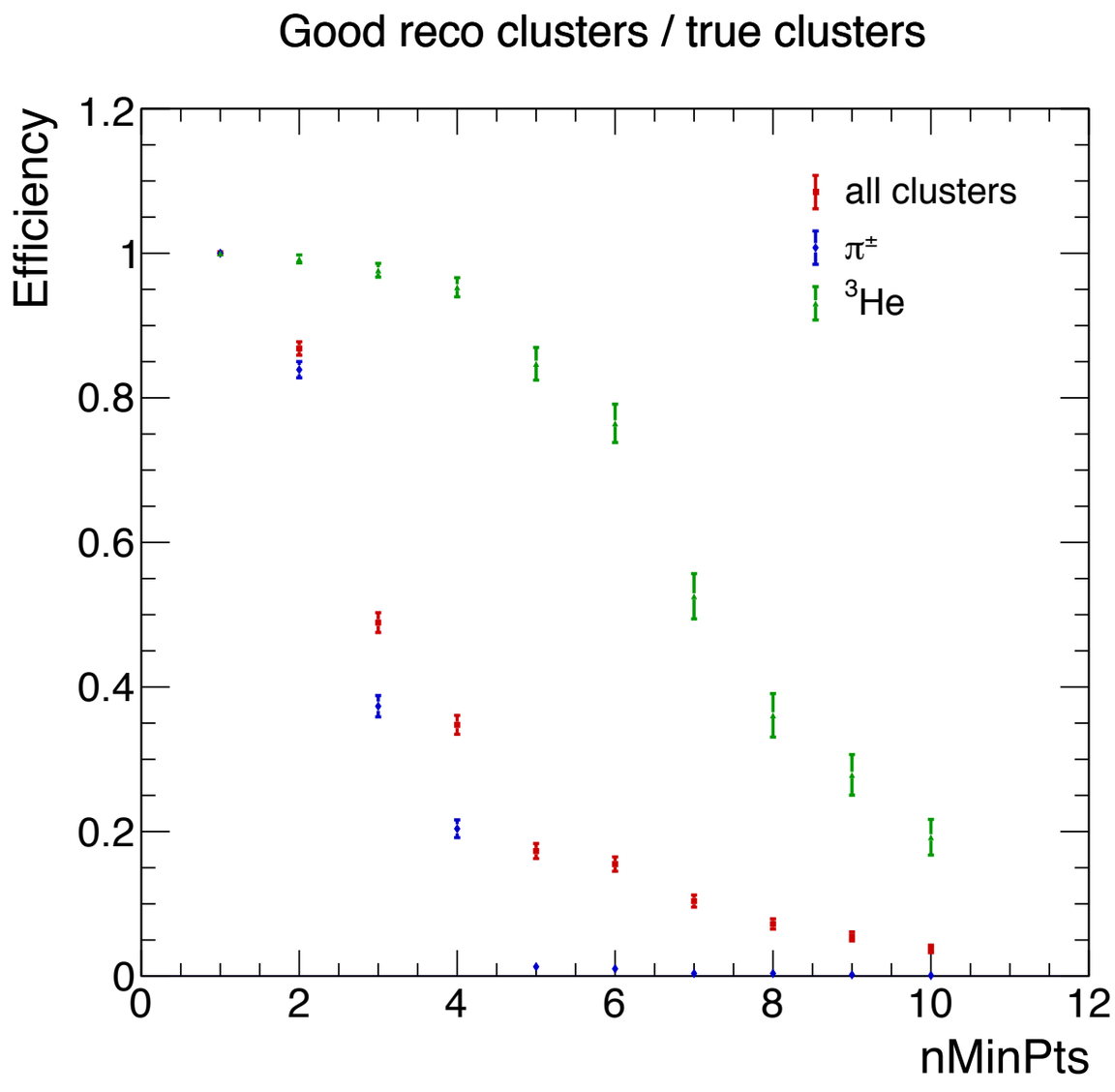


Results - ε fixed

- The efficiency of the algorithm was tested varying *minPts*, keeping ε fixed

$\varepsilon = 90\ \mu\text{m}$





Summary

- GitHub repository: git@github.com:mconcas/unsupervisedNN.git
- DBSCAN is a valuable algorithm both to find clusters and to separate clusters generated by different particle species
- Optimisation of the parameters depends on pixel density (particle multiplicity) and kind of particles searched (energy deposition
  - cluster size)
- For real life use cases, the procedure is much more complex because the information has to be matched with other sub-detectors, and there are many particle species, dead pixels and noise, that should be rejected (not included in the example presented)
  - The noise suppression is usually performed by combining the results from different layers (reconstruction of full trajectories)