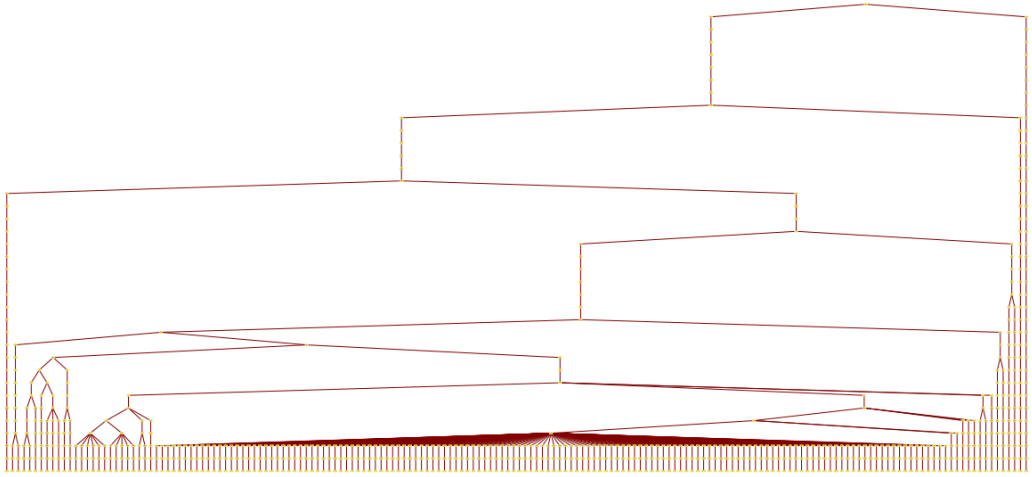
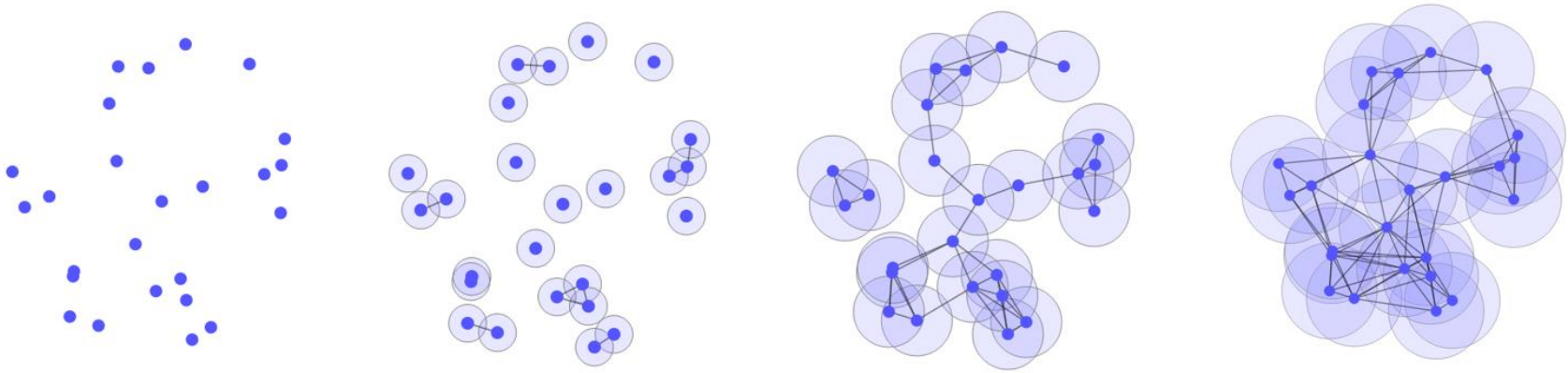
Where can we use graph algorithms we learned in ML?

Mustafa Hajij



Hierarchical Clustering



Hierarchical Clustering

Hierarchical clustering is a family of clustering algorithms that build a tree clusters. It is usually done be done by merging or splitting the clusters successively.

Hierarchical Clustering

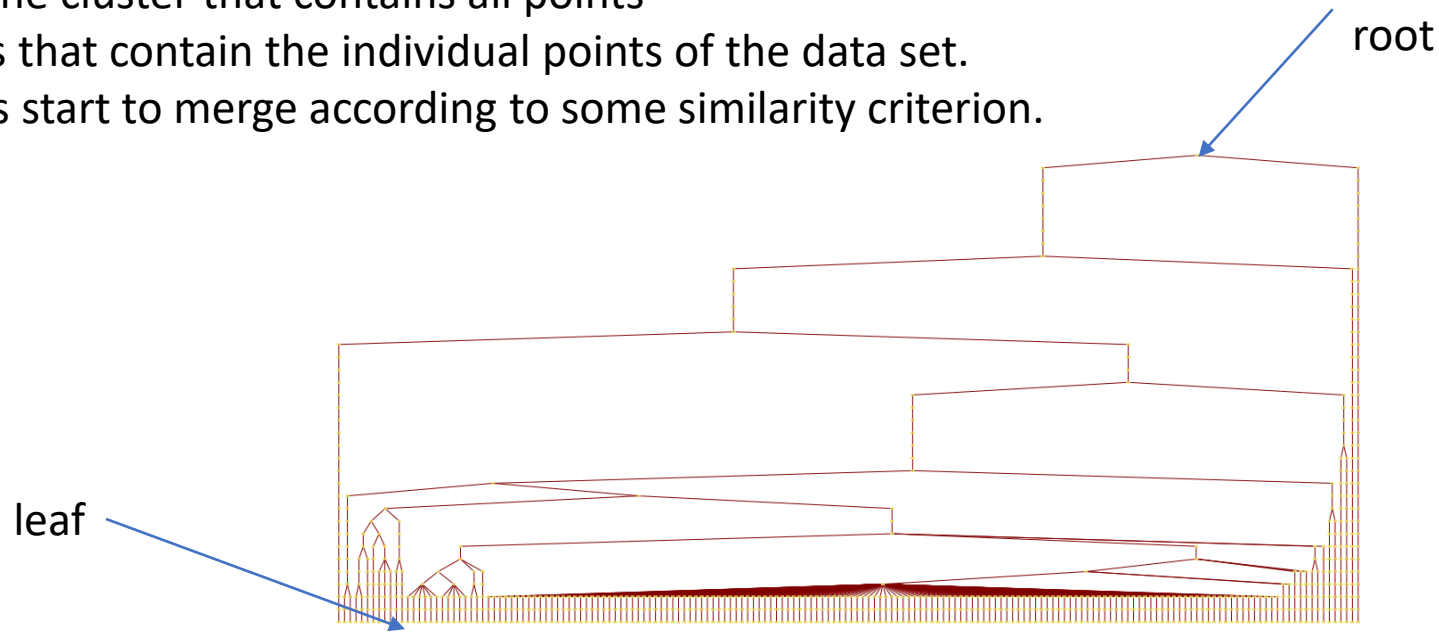
Hierarchical clustering is a family of clustering algorithms that build a tree clusters. It is usually done be done by merging or splitting the clusters successively.

Hierarchical clustering is usually represented by a tree called the dendrogram that represents the clusterings at all levels.

Each node in the tree represents a cluster

- In particular the root of the tree represents the cluster that contains all points
- The leaves of the tree represents the clusters that contain the individual points of the data set.
- As we go from the leaves to the root, clusters start to merge according to some similarity criterion.

There are two types of Hierarchical clustering
Agglomerative : bottom up (Merge).
Divisive : Top down (Split).



General Steps

General Steps in a standard hierarchical agglomerative clustering algorithm:

1. Compute the distance matrix, or the dissimilarity, between all points in the input data. Choosing the metric will impact the results largely.

General Steps

General Steps in a standard hierarchical agglomerative clustering algorithm:

1. Compute the distance matrix, or the dissimilarity, between all points in the input data. Choosing the metric will impact the results largely.
2. Initialize every point in the dataset to be its own cluster

General Steps

General Steps in a standard hierarchical agglomerative clustering algorithm:

1. Compute the distance matrix, or the dissimilarity, between all points in the input data. Choosing the metric will impact the results largely.
2. Initialize every point in the dataset to be its own cluster

3. Compute the distance between all clusters

General Steps

General Steps in a standard hierarchical agglomerative clustering algorithm:

1. Compute the distance matrix, or the dissimilarity, between all points in the input data. Choosing the metric will impact the results largely.
2. Initialize every point in the dataset to be its own cluster

3. Compute the distance between all clusters
4. Combine the closest two clusters: the two clusters $c_i and\ c_j\ with \min_{i,j} D(c_i, c_j)$

General Steps

General Steps in a standard hierarchical agglomerative clustering algorithm:

1. Compute the distance matrix, or the dissimilarity, between all points in the input data. Choosing the metric will impact the results largely.
2. Initialize every point in the dataset to be its own cluster

3. Compute the distance between all clusters
4. Combine the closest two clusters: the two clusters $c_i and\ c_j\ with \min_{i,j} D(c_i, c_j)$
5. Remove the clusters $c_i and\ c_j$ and add the cluster $c_i + c_j$.

General Steps

General Steps in a standard hierarchical agglomerative clustering algorithm:

1. Compute the distance matrix, or the dissimilarity, between all points in the input data. Choosing the metric will impact the results largely.
2. Initialize every point in the dataset to be its own cluster

3. Compute the distance between all clusters
4. Combine the closest two clusters: the two clusters $c_i and\ c_j\ with \min_{i,j} D(c_i, c_j)$
5. Remove the clusters $c_i and\ c_j$ and add the cluster $c_i + c_j$.
6. Go back to 3 and repeat until we have a single cluster.

General Steps

General Steps in a standard hierarchical agglomerative clustering algorithm:

1. Compute the distance matrix, or the dissimilarity, between all points in the input data. Choosing the metric will impact the results largely.
2. Initialize every point in the dataset to be its own cluster

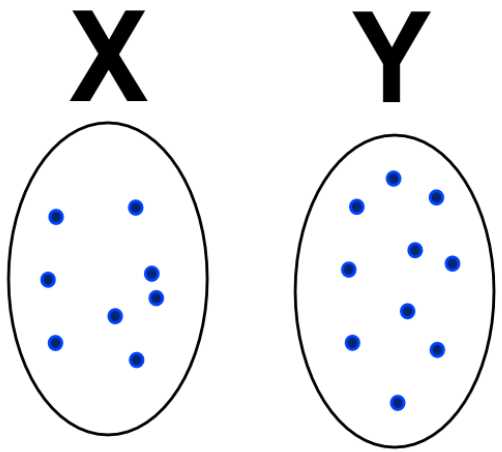
3. Compute the distance between all clusters
4. Combine the closest two clusters: the two clusters $c_i and\ c_j\ with \min_{i,j} D(c_i, c_j)$
5. Remove the clusters $c_i and\ c_j$ and add the cluster $c_i + c_j$.
6. Go back to 3 and repeat until we have a single cluster.

Question : how do we measure the distance between two clusters ?

This is important because step 3 we need to measure the distance between clusters rather than points.

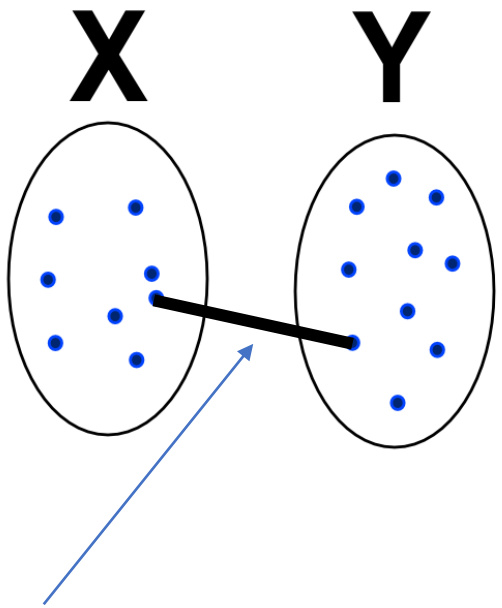
Distance between clusters

Given two clusters X and Y. How do we measure the distance between them ?



Distance between clusters

Given two clusters X and Y. How do we measure the distance between them ?

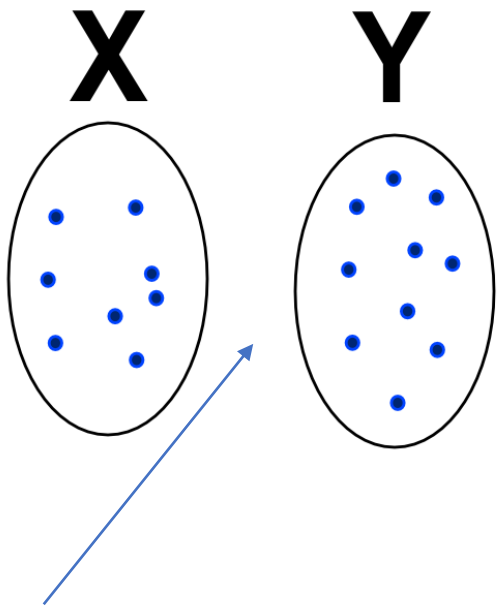


$$D(X,Y) := \min_{x \text{ in } X, y \text{ in } Y} d(x,y)$$

One way is to measure the minimal distance between all points of X and Y.
This distance induce single linkage clustering.

Distance between clusters

Given two clusters X and Y. How do we measure the distance between them ?

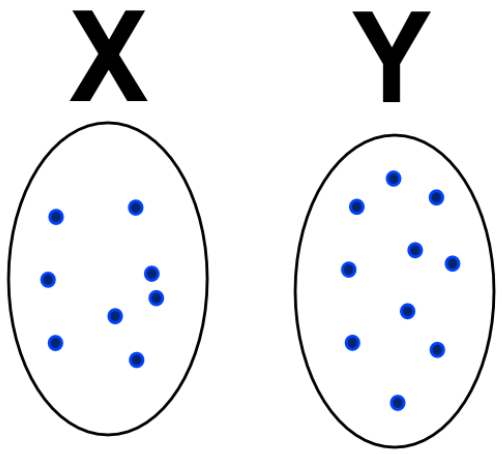


$$D(X,Y) := \frac{1}{|X||Y|} \sum_{x\ in\ X, y\ in Y}^{n} d(x,y)$$

We could also consider the mean distance between the points of the clusters

Distance between clusters

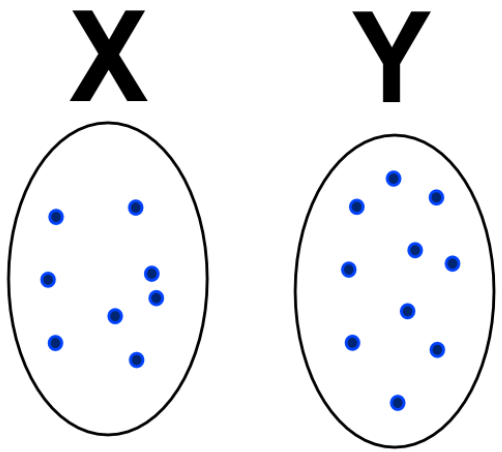
Given two clusters X and Y. How do we measure the distance between them ?



There are other measures as well : The minimal energy criterion , the distance between the centroids of the clusters

Distance between clusters

Given two clusters X and Y. How do we measure the distance between them ?



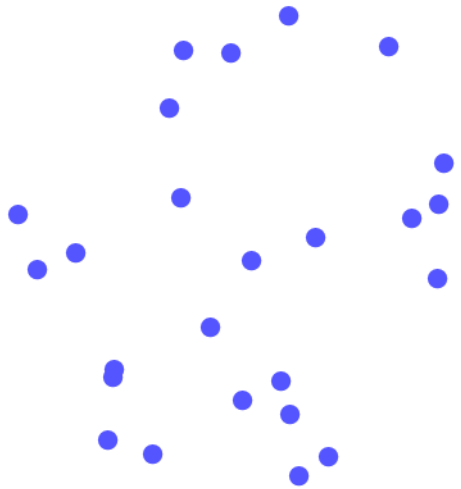
For efficient calculations we usually require the following condition:
Knowing the distance D(A,C), D(B,C)
Implies we can calculate in constant time the distance D(A+B,C)

There are other measures as well : The minimal energy criterion , the distance between the centroids of the clusters

Single Linkage Hierarchical Clustering and the ε- Neighborhood Graph

Suppose that we are given a set of points $X = \{p_1, p_2, \dots, p_n\}$ in R^d with a distance function d defined one them.

Consider the connected components of the ε-neighborhood graph as we continuously increase ε from zero to infinity.

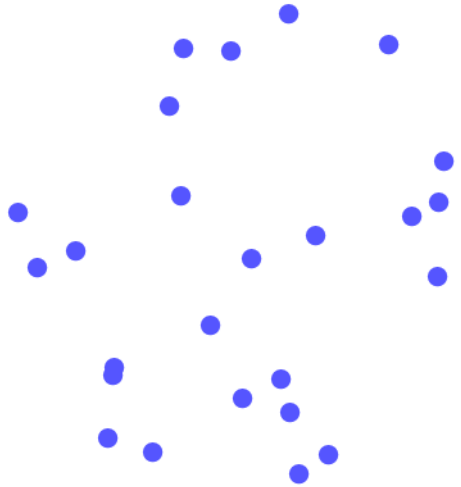


Every point is a connected component

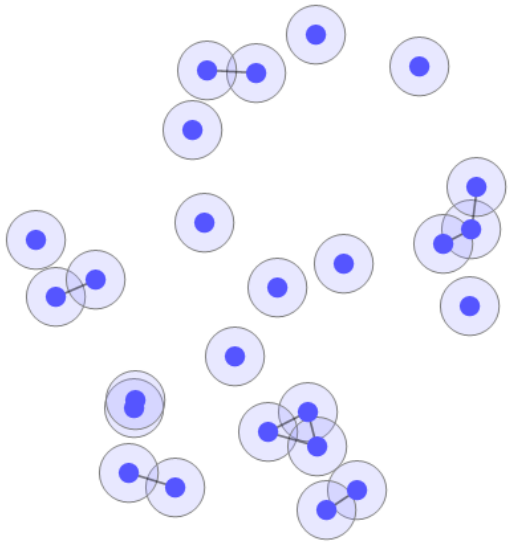
Single Linkage Hierarchical Clustering and the ε- Neighborhood Graph

Suppose that we are given a set of points $X = \{p_1, p_2, \dots, p_n\}$ in R^d with a distance function d defined one them.

Consider the connected components of the ε-neighborhood graph as we continuously increase ε from zero to infinity.



Every point is a connected component

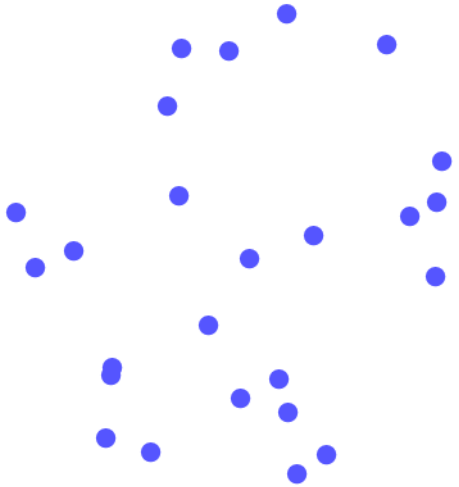


When ε is a little larger we start some clusters starts to get form

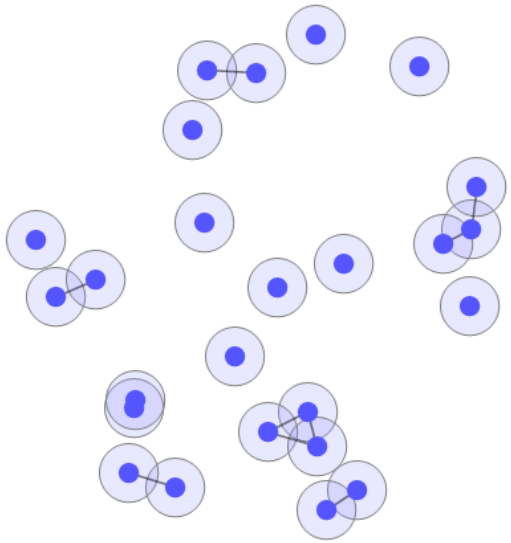
Single Linkage Hierarchical Clustering and the ε- Neighborhood Graph

Suppose that we are given a set of points $X = \{p_1, p_2, \ldots, p_n\}$ in R^d with a distance function d defined one them.

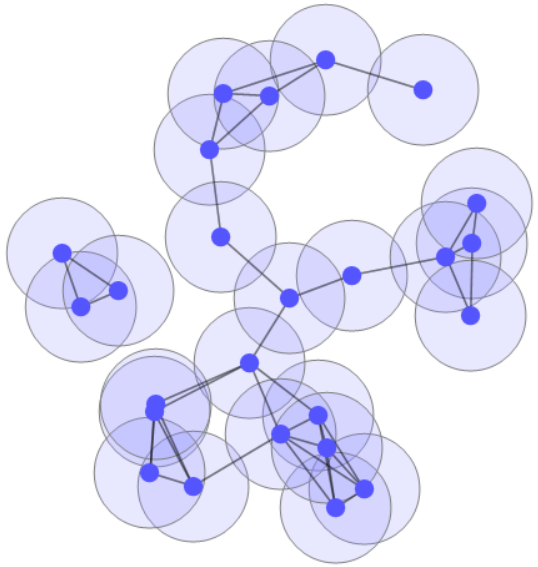
Consider the connected components of the ε-neighborhood graph as we continuously increase ε from zero to infinity.



Every point is a connected component



When ε is a little larger we start some clusters starts to get form

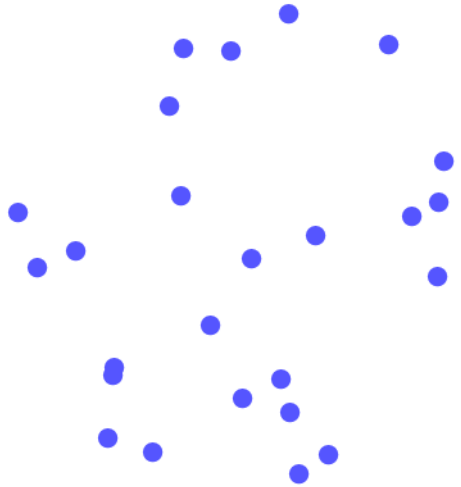


When ε is even larger we have few clusters
As the clusters get larger and larger

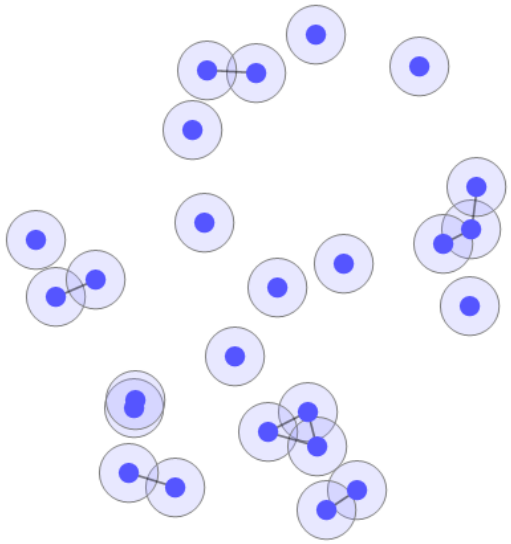
Single Linkage Hierarchical Clustering and the ε- Neighborhood Graph

Suppose that we are given a set of points $X = \{p_1, p_2, \ldots, p_n\}$ in R^d with a distance function d defined one them.

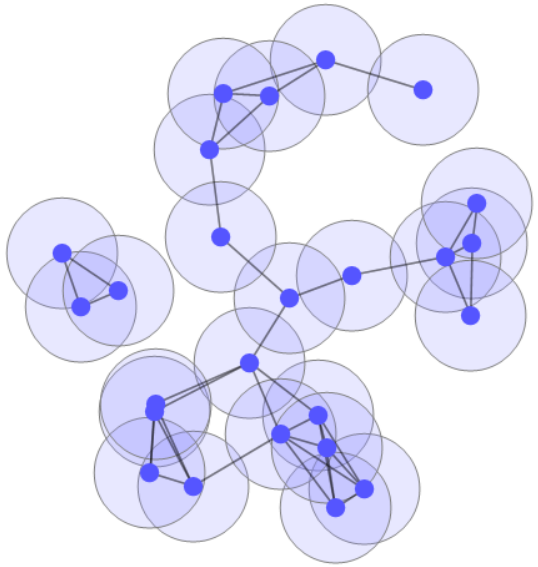
Consider the connected components of the ε-neighborhood graph as we continuously increase ε from zero to infinity.



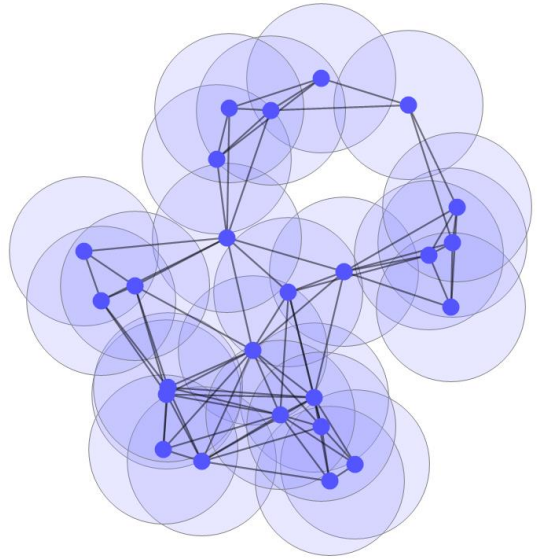
Every point is a connected component



When ε is a little larger we start some clusters starts to get form



When ε is even larger we have few clusters
As the clusters get larger and larger



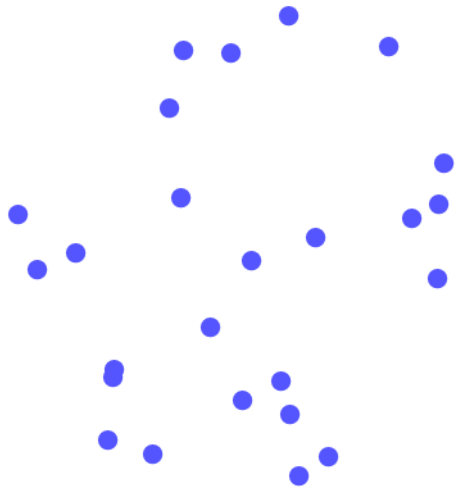
At some point all points become a par to of a single cluster

Single Linkage Hierarchical Clustering and the ε- Neighborhood Graph

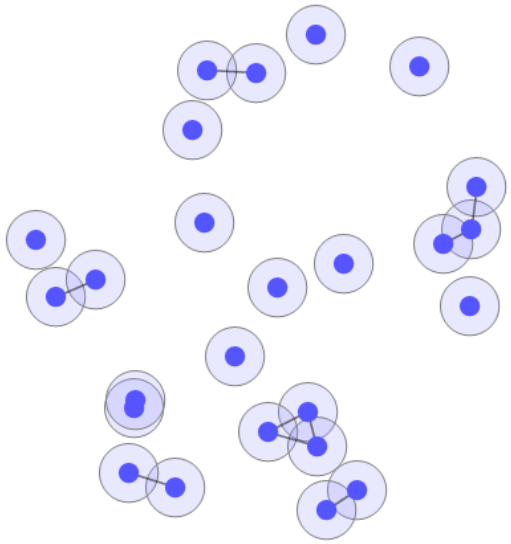
Remark : Observe that the growing of this epsilon corresponds exactly to the idea of **Hierarchical Clustering!**

Question : How can we implement this effectively ?

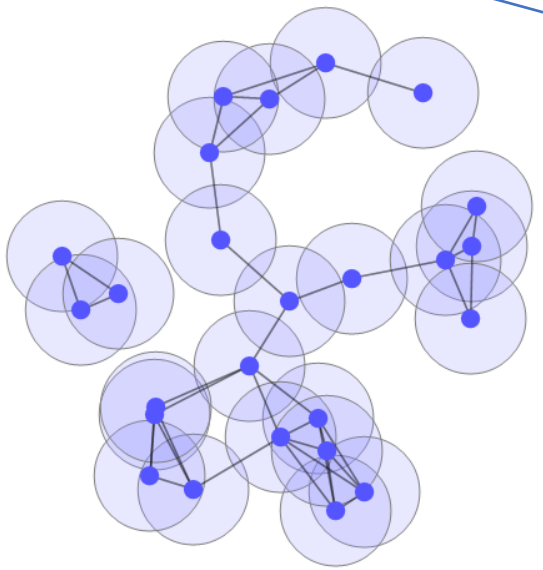
Question : Does the minimal spanning tree of this graph give any information about the clustering?



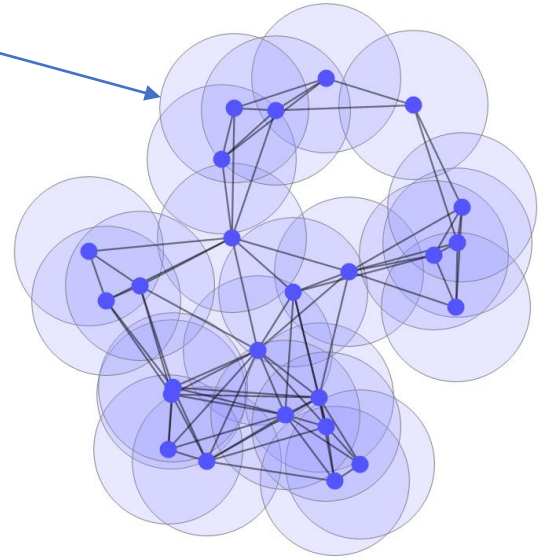
Every point is a connected component



When ε is a little larger we start some clusters starts to get form

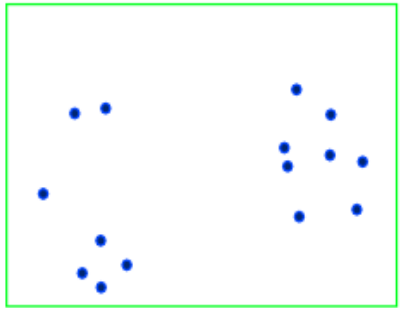


When ε is even larger we have few clusters
As the clusters get larger and larger

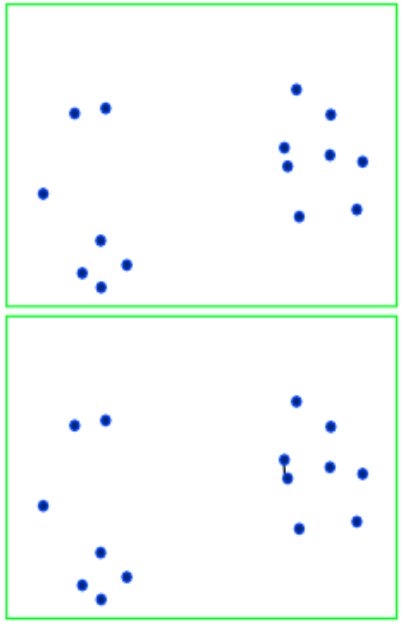


At some point all points become a par to of a single cluster

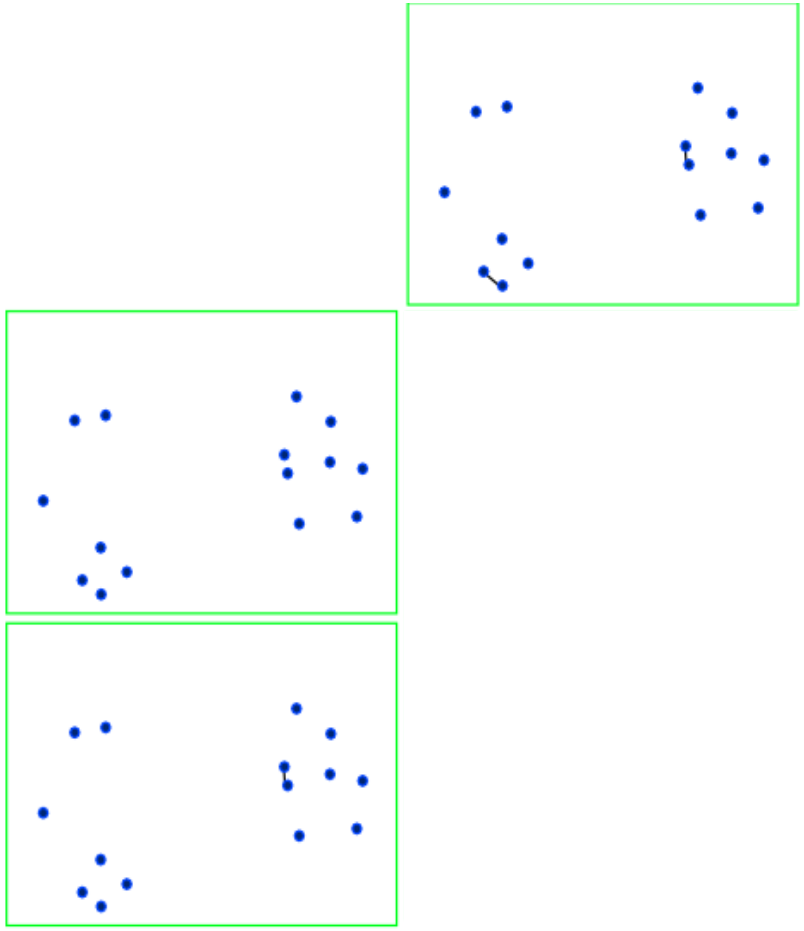
Single Linkage Hierarchical Clustering and the and Kruskal's algorithm



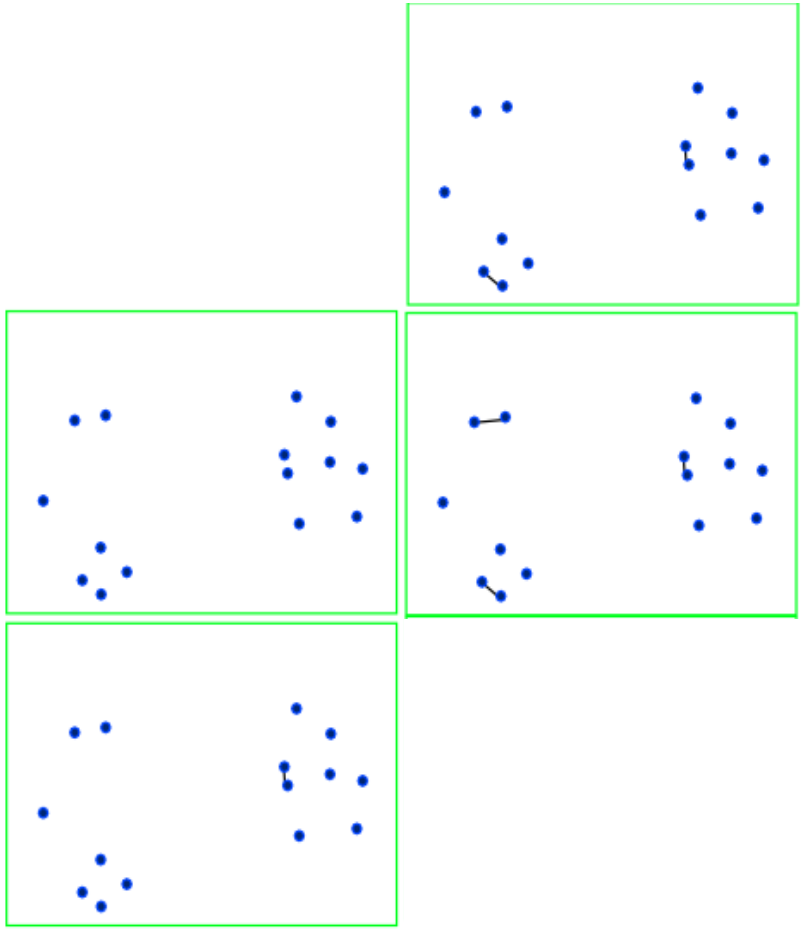
Single Linkage Hierarchical Clustering and the and Kruskal's algorithm



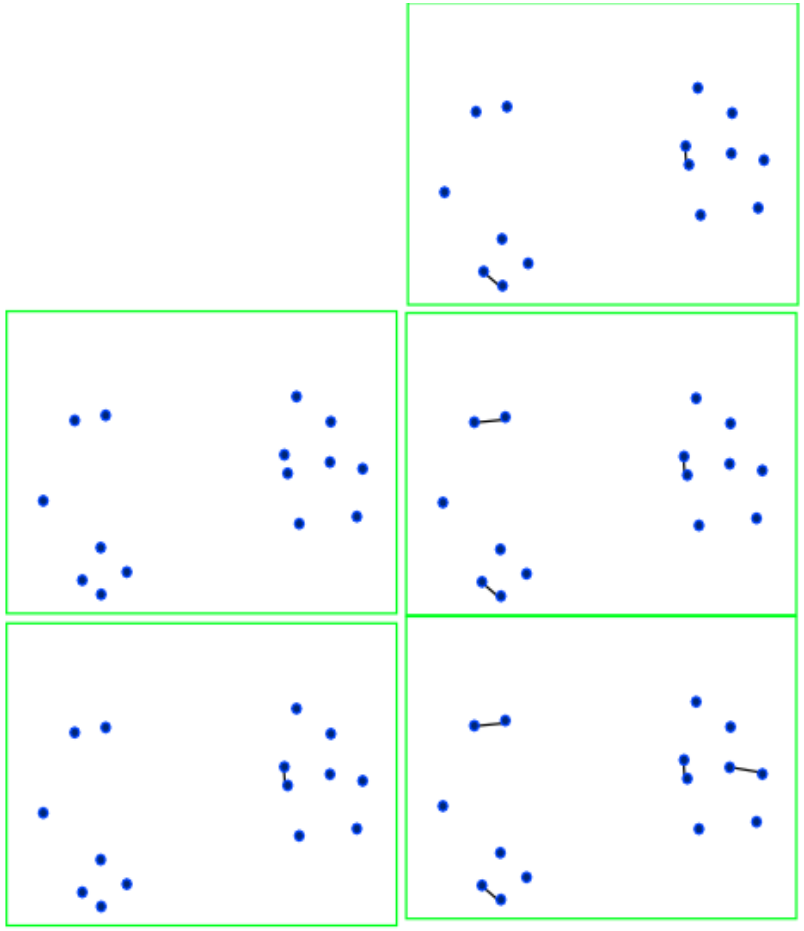
Single Linkage Hierarchical Clustering and the and Kruskal's algorithm



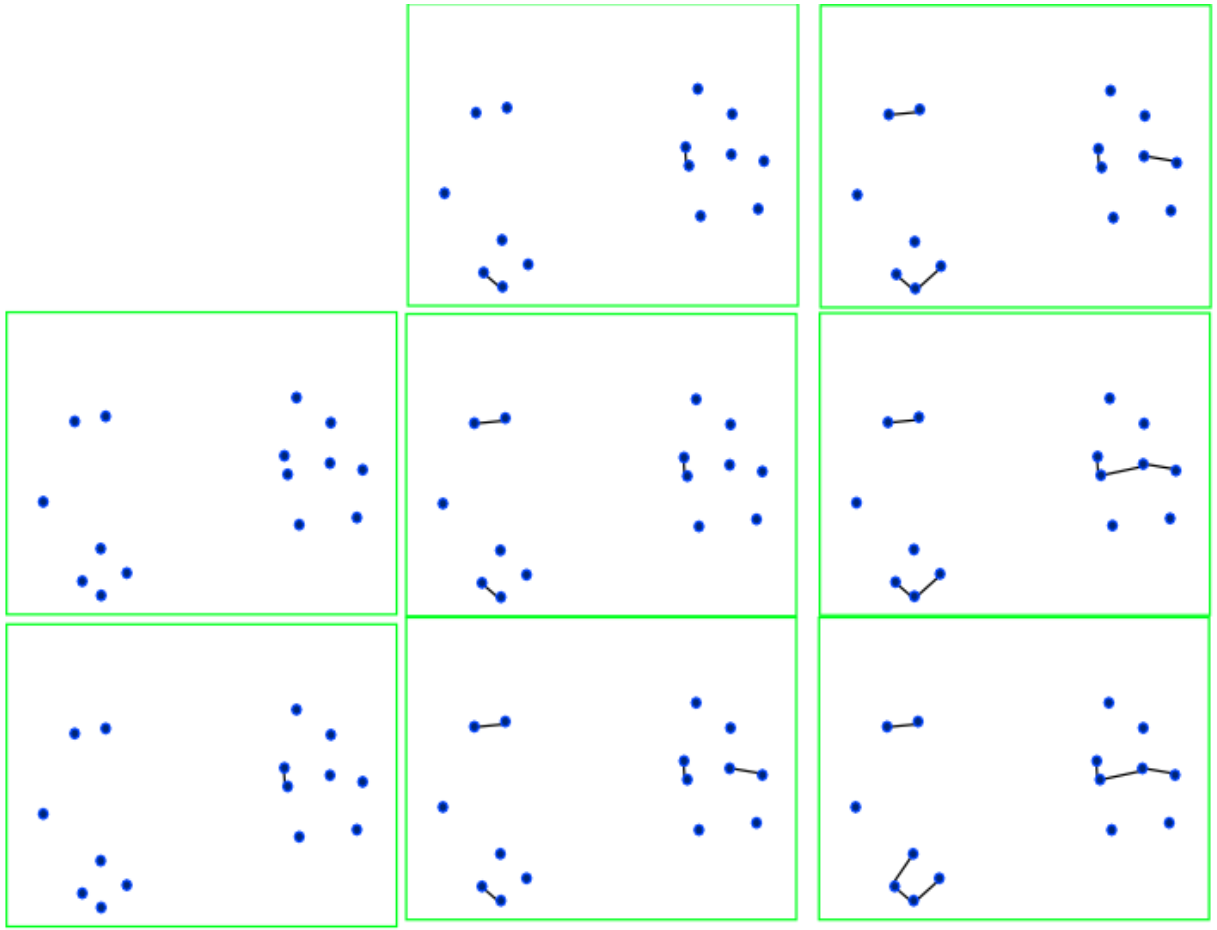
Single Linkage Hierarchical Clustering and the and Kruskal's algorithm



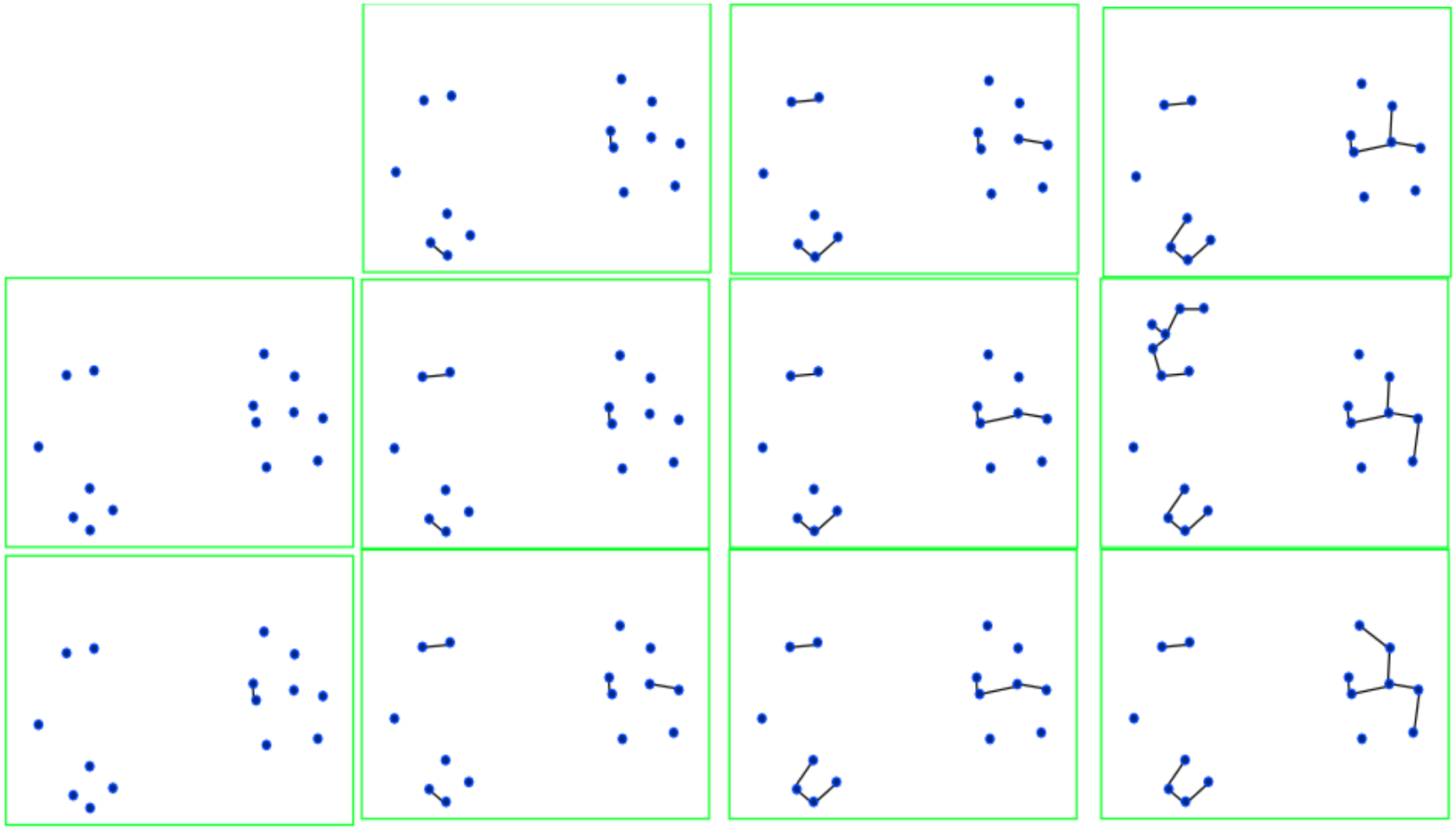
Single Linkage Hierarchical Clustering and the and Kruskal's algorithm



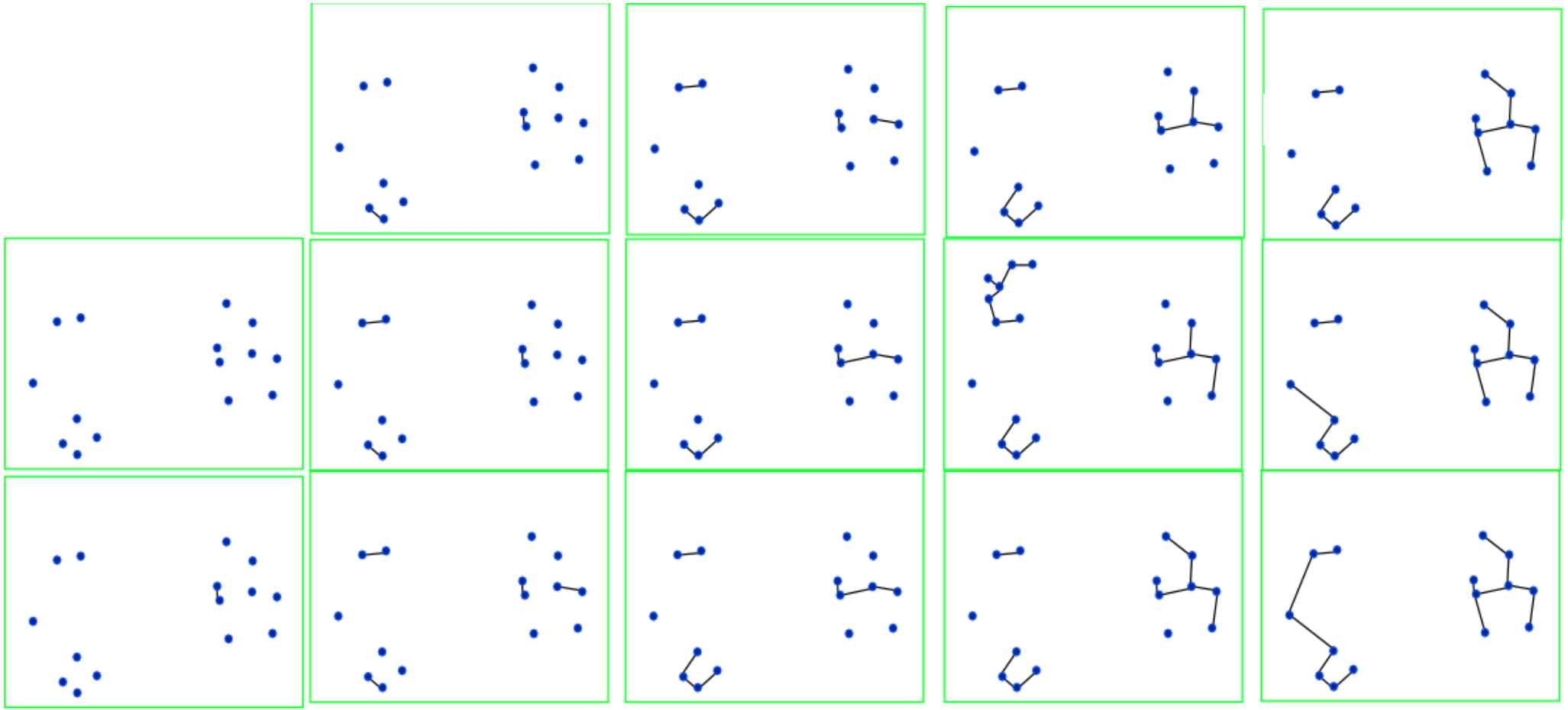
Single Linkage Hierarchical Clustering and the and Kruskal's algorithm



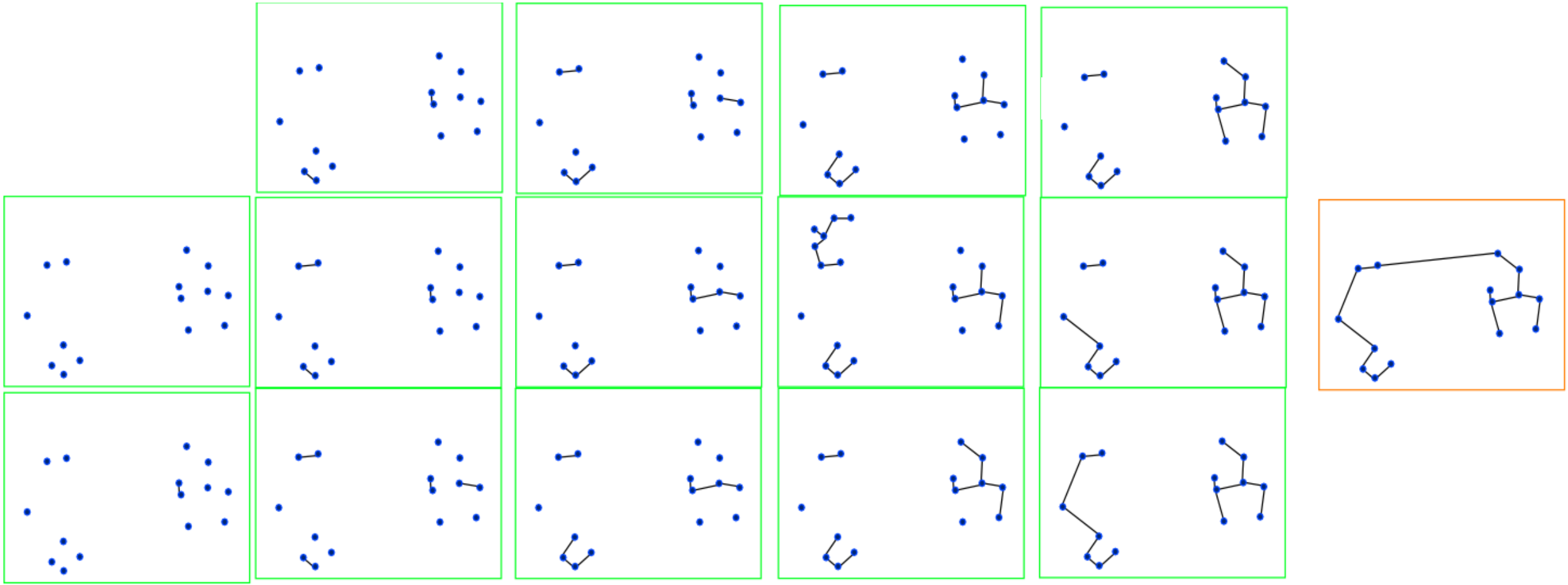
Single Linkage Hierarchical Clustering and the and Kruskal's algorithm



Single Linkage Hierarchical Clustering and the and Kruskal's algorithm



Single Linkage Hierarchical Clustering and the and Kruskal's algorithm



The minimal spanning tree obtained in this way provides exactly
the Single linkage hierarchical clustering that we talked about earlier

An Introduction to Multidimensional Scaling and ISOMAP

MDS

Let D=[d_{ij}] be an $N \times N$ dissimalrity matrix.

In MDS, we want to find n vectors $x_1, \dots, x_N$ in R^d such that $||x_i - x_j|| \approx d_{ij}$

MDS

Let D=$[d_{ij}]$ be an $N \times N$ dissimalrity matrix.

In MDS, we want to find n vectors $x_1, \dots, x_N$ in R^d such that $||x_i - x_j|| \approx d_{ij}$

- Usually if we choose d to be large enough, we can construct the *vectors $x_1, \dots, x_N$* with exact solutions : $||x_i - x_j|| = d_{ij}$.

MDS

Let D=$[d_{ij}]$ be an $N \times N$ dissimalrity matrix.

In MDS, we want to find n vectors $x_1, \dots, x_N$ in R^d such that $||x_i - x_j|| \approx d_{ij}$

- Usually if we choose d to be large enough, we can construct the *vectors $x_1, \dots, x_N$* with exact solutions : $||x_i - x_j|| = d_{ij}$.
- In this case the distance d above is the usual Euclidean distance.

MDS

Let D=$[d_{ij}]$ be an $N \times N$ dissimalrity matrix.

In MDS, we want to find n vectors $x_1, \dots, x_N$ in R^d such that $||x_i - x_j|| \approx d_{ij}$

- Usually if we choose d to be large enough, we can construct the *vectors* $x_1, \dots, x_N$ with exact solutions : $||x_i - x_j|| = d_{ij}$.
- In this case the distance d above is the usual Euclidean distance.
- There are cases where the matrix D is valid distance matrix, but still there exists no set of vectors $x_1, \dots, x_N$ in any R^d with perfect $||x_i - x_j|| = d_{ij}$. Such a distance is called non-Euclidean distance.

Stress Majorization

In the classical MDS algorithm the cost function that we are trying to optimize is called the stress function and it is given by :

$$Stress_D(x_1, x_2, \ldots, x_N) = \left(\sum_{i \neq j = 1, \ldots, N} \left(d_{ij} - \|x_i - x_j\| \right)^2 \right)^{1/2}$$

In this function we try to find $x_1, \ldots, x_N$ in a certain dimension d such that Stress($x_1, \ldots, x_N$) is as small as possible

Stress Majorization

The stress function has a more general form as :

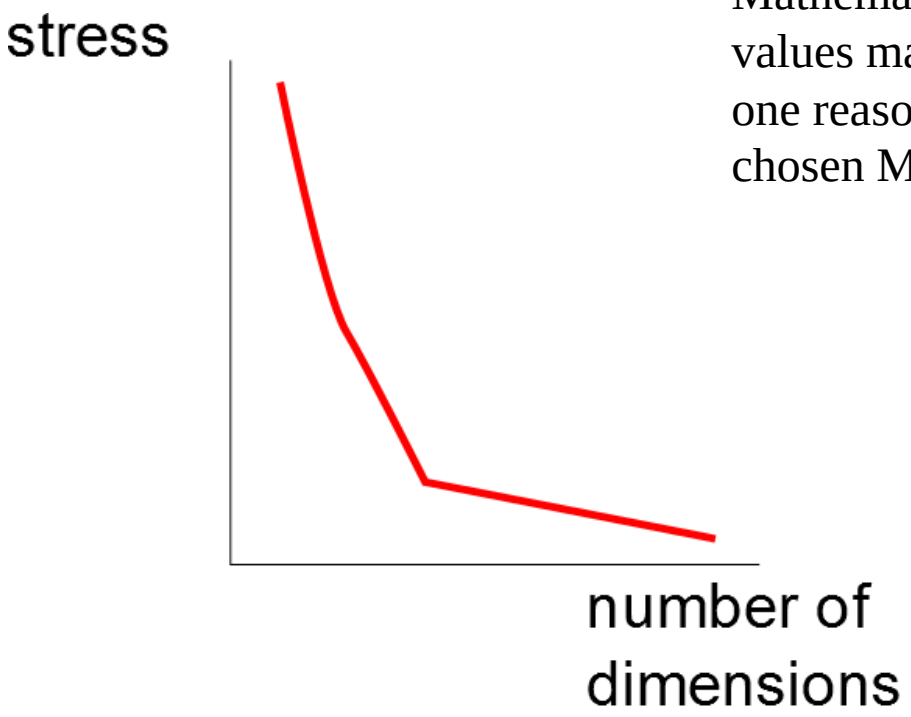
$$\sigma(X) = \sum_{i<j\leq n} w_{ij}(d_{ij}(X) - \delta_{ij})^2$$

Here wij weight between a pair of points (i,j) that represents the confidence in in the similarity between points (i,j) .

δ_{ij} the given distance between the points i,j

Stress Majorization

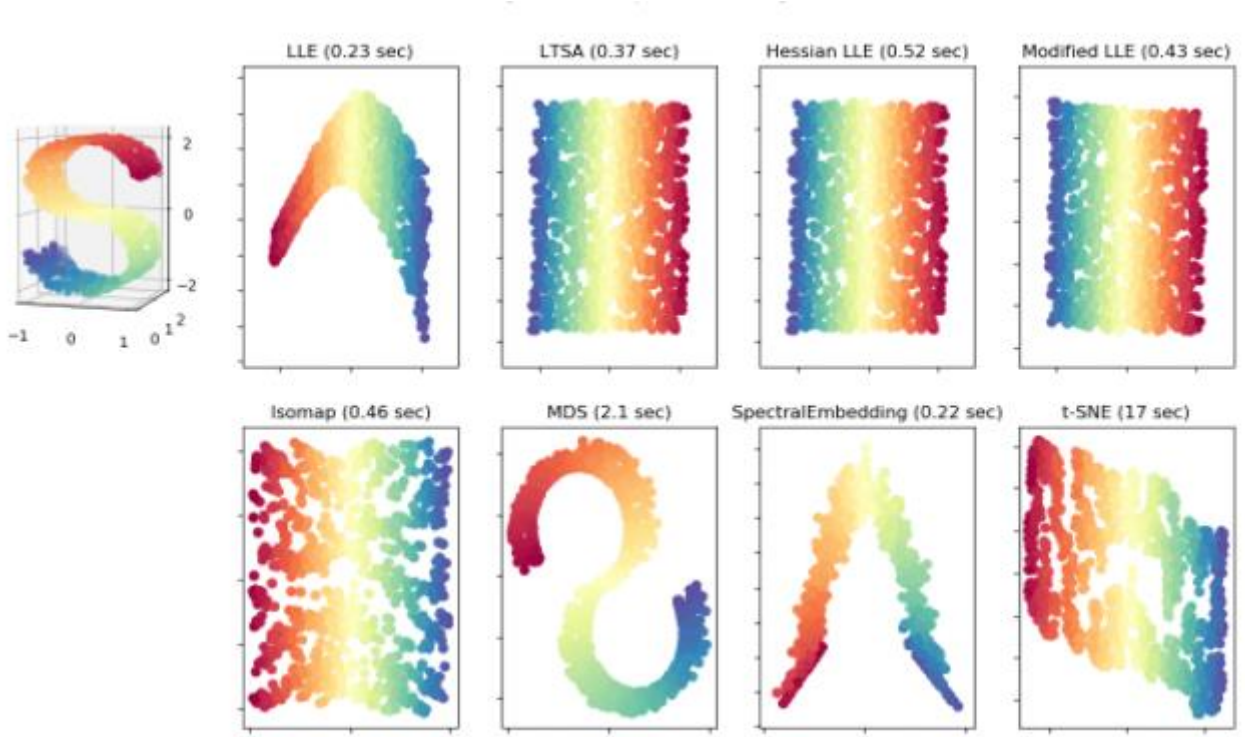
“Pressing” the data into 2 dimensions enables us to visualize the data. However, that comes with a price : high stress function value (which correlates wit distorted representation)



Mathematically non-zero stress values may occur only for only one reason: dimensionality of the chosen MDS projection is too low.

MDS in Sklearn

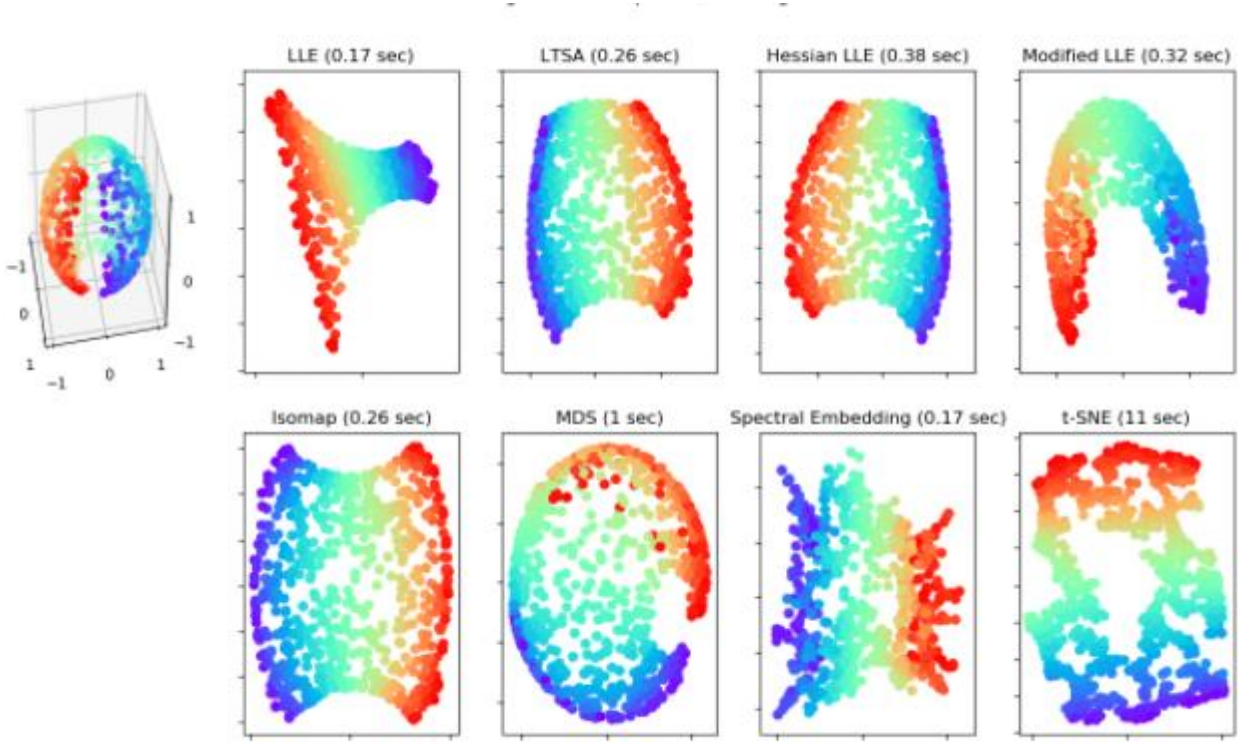
MDS is implemented in [Sklearn]



[Example]

MDS in Sklearn

MDS is implemented in [Sklearn]

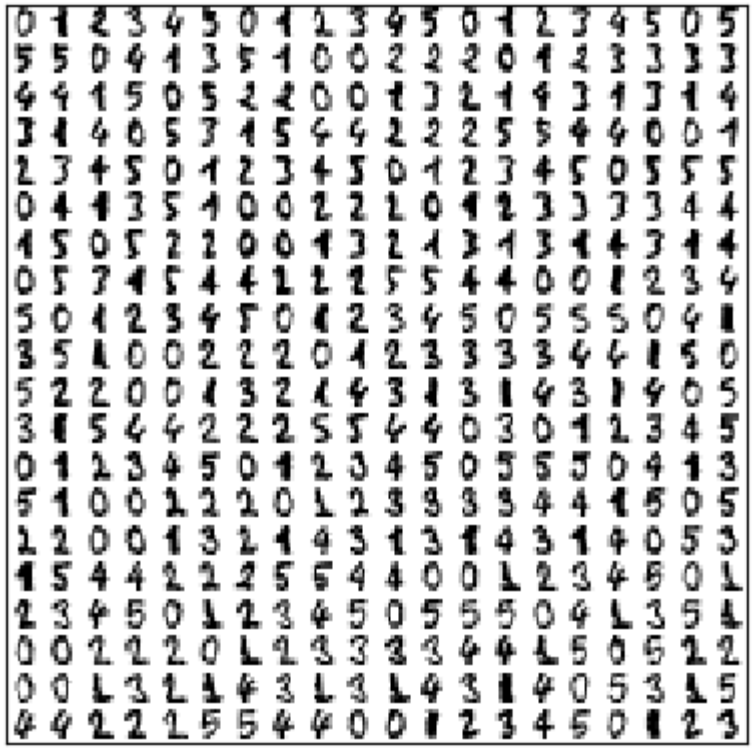


[Example]

MDS in Sklearn

MDS is implemented in [Sklearn]

A selection from the 64-dimensional digits dataset



[Example]

ISOMAP

- Isomap extends MDS by utilizing geodesic distances induced by some neighborhood graph.
- There is no essential difference between MDS and ISOMAP algorithm once we find the distance matrix induced by the neighborhood graph.

ISOMAP

- Isomap extends MDS by utilizing geodesic distances induced by some neighborhood graph.
- There is no essential difference between MDS and ISOMAP algorithm once we find the distance matrix induced by the neighborhood graph.

So the steps for ISOMAP on a given data :

1- Construct the neighborhood graph of the data X using one of the neighborhood graphs we studied earlier in the course

ISOMAP

- Isomap extends MDS by utilizing geodesic distances induced by some neighborhood graph.
- There is no essential difference between MDS and ISOMAP algorithm once we find the distance matrix induced by the neighborhood graph.

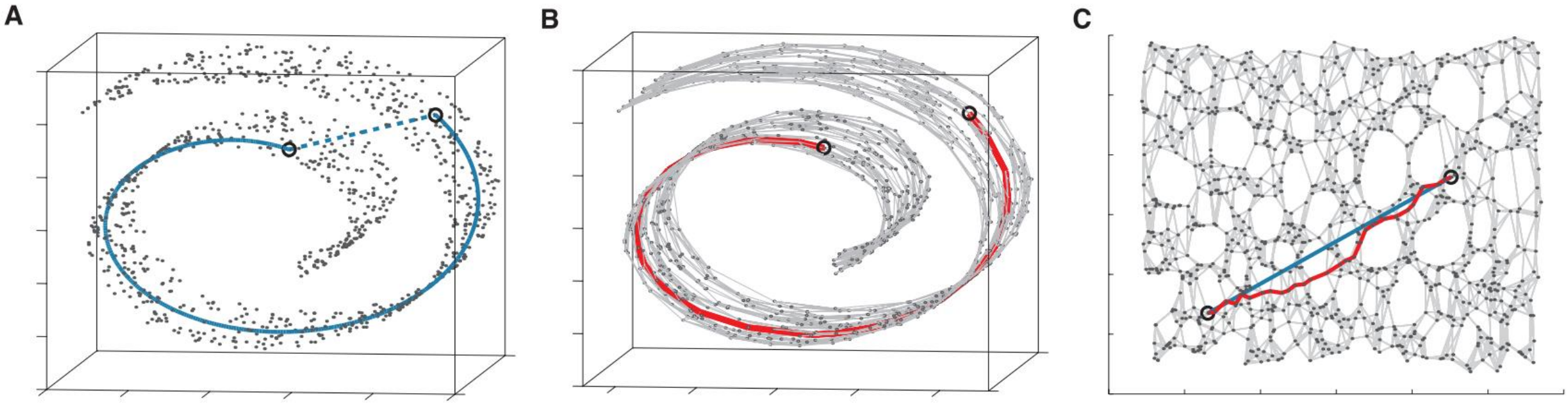
So the steps for ISOMAP on a given data :
1- Construct the neighborhood graph of the data X using one of the neighborhood graphs we studied earlier in the course
2- Use the Dijekstra algorithm or the Floyd–Warshall algorithm to find the distance between nodes on the graph

ISOMAP

- Isomap extends MDS by utilizing geodesic distances induced by some neighborhood graph.
- There is no essential difference between MDS and ISOMAP algorithm once we find the distance matrix induced by the neighborhood graph.

So the steps for ISOMAP on a given data :
1- Construct the neighborhood graph of the data X using one of the neighborhood graphs we studied earlier in the course
2- Use the Dijekstra algorithm or the Floyd–Warshall algorithm to find the distance between nodes on the graph
3- Apply MDS on the distance matrix above and extract the coordinates with the desired dimension

ISOMAP



A- Euclidian distance might not represent the actual distance between the points in the data.
B- We can construct the neighborhood graph of the data and then compute the geodesic distance between the points of the graph
C- Embedding the space we obtained in B into the plane.

Figure reference

Appendix : Floyd–Warshall algorithm

```
let dist be a |V| × |V| array of minimum distances
initialized to infinity

for each edge (u,v)
 dist[u][v] ← w(u,v)  // the weight of the edge (u,v)
for each vertex v
 dist[v][v] ← 0
for k from 1 to |V|
 for i from 1 to |V|
   for j from 1 to |V|
          if dist[i][j] > dist[i][k] + dist[k][j]
                     dist[i][j] ← dist[i][k] + dist[k][j]
           end if
```

Floyd algorithm is good to use when we want to compute the distance matrix on a dense graph.
When the graph G is sparse, Dijekstra algorithm is a better choice.