Nearest Neighbors-Based Clustering and Classification Methods

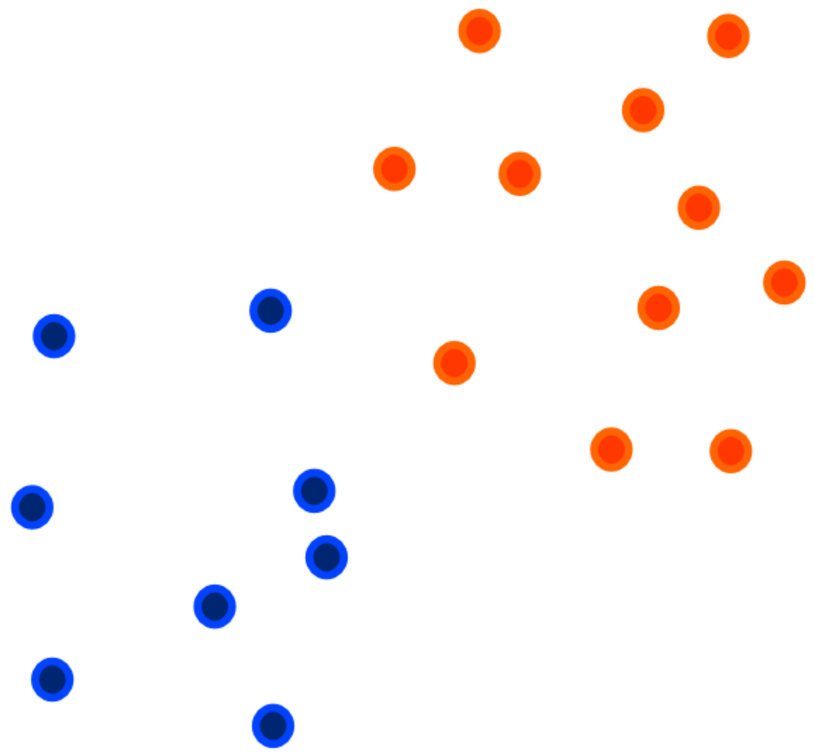
Mustafa Hajij

Application of K-nearest neighbors: $K\ Neighbors\ Classifier$

This is an supervised classifier that takes into consideration the nearest k-points to an input point to determine the class that point belongs to. The idea is to use the *majority vote* of the neighbors of the input points.

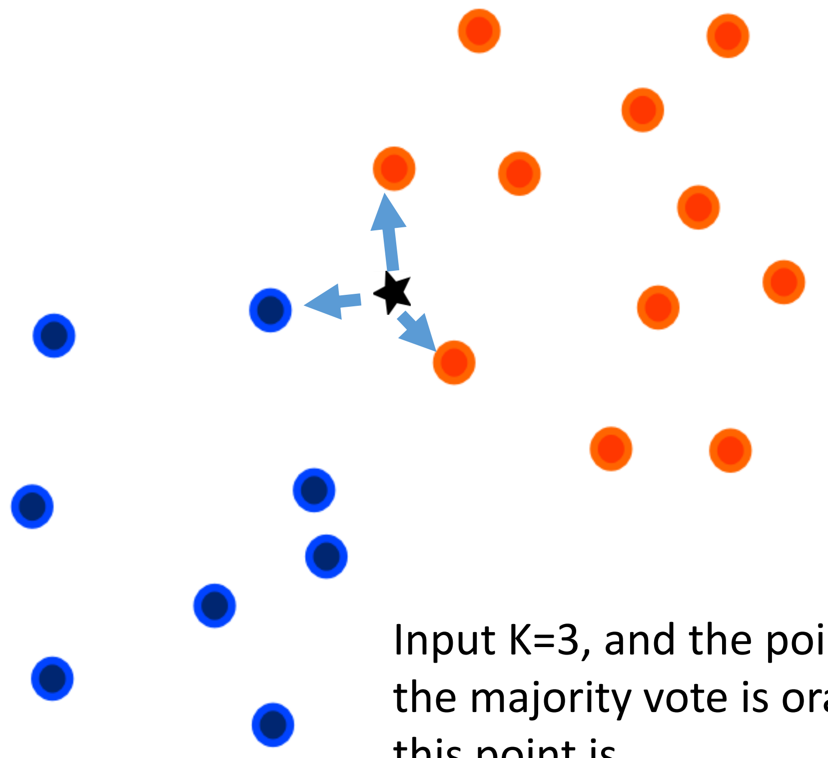
Application of K-nearest neighbors: $K\ Neighbors\ Classifier$

This is an supervised classifier that takes into consideration the nearest k-points to an input point to determine the class that point belongs to. The idea is to use the *majority vote* of the neighbors of the input points.



Application of K-nearest neighbors: *K Neighbors Classifier*

This is an supervised classifier that takes into consideration the nearest k-points to an input point to determine the class that point belongs to. The idea is to use the *majority vote* of the neighbors of the input points.



Input K=3, and the point ★
the majority vote is orange hence
this point is
classified orange

Application of K-nearest neighbors: *K Neighbors Classifier*

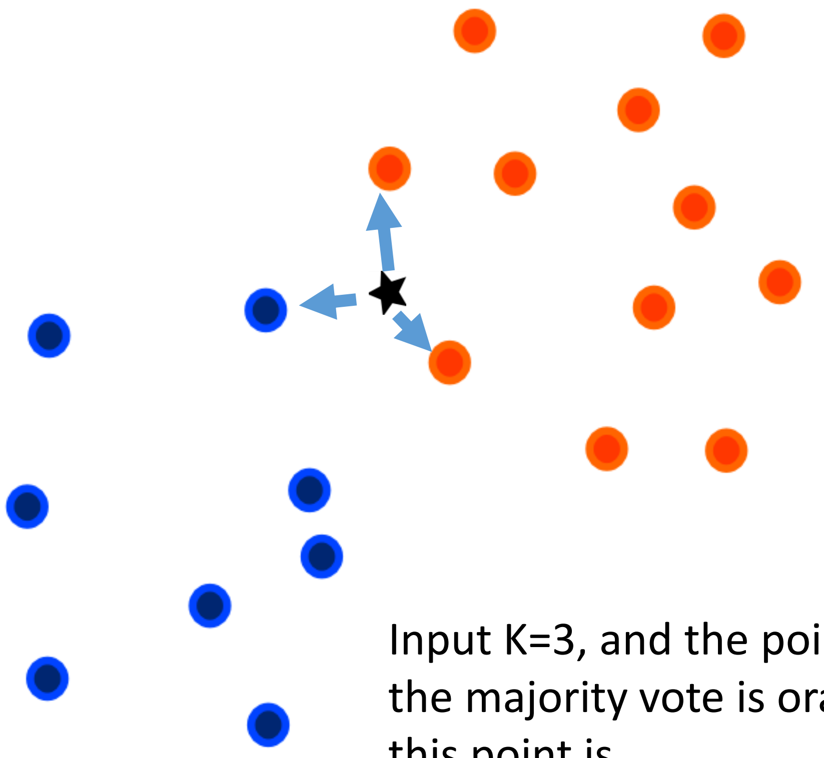
This is an supervised classifier that takes into consideration the nearest k-points to an input point to determine the class that point belongs to. The idea is to use the *majority vote* of the neighbors of the input points.

In sklearn :

```
from sklearn.neighbors import KNeighborsClassifier # import the classifier

X = [[0], [1], [2], [3]]                              # the data
y = [0, 0, 1, 1]                                       # the labels

neigh = KNeighborsClassifier(n_neighbors=3)  # define the parameters of the classifie
neigh.fit(X, y)                              # fit the data
```



Input K=3, and the point ★ the majority vote is orange hence this point is classified orange

Application of K-nearest neighbors: *K Neighbors Classifier*

This is an supervised classifier that takes into consideration the nearest k-points to an input point to determine the class that point belongs to. The idea is to use the *majority vote* of the neighbors of the input points.

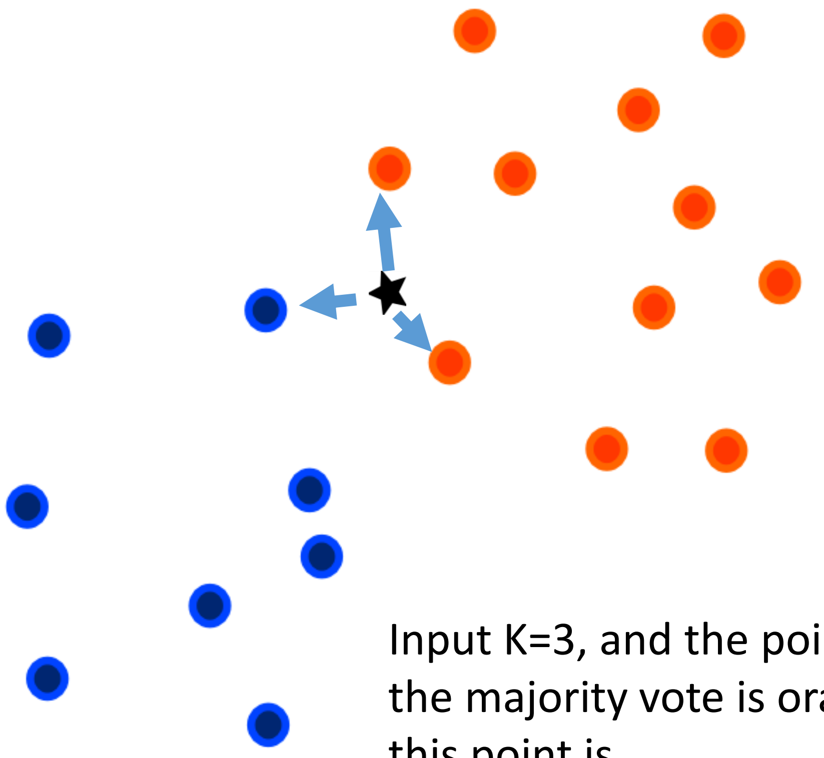
In sklearn :

```
from sklearn.neighbors import KNeighborsClassifier # import the classifier

X = [[0], [1], [2], [3]]                          # the data
y = [0, 0, 1, 1]                                   # the labels

neigh = KNeighborsClassifier(n_neighbors=3)  # define the parameters of the classifie
neigh.fit(X, y)                              # fit the data
```

If k=1, this classifier simply assigns the
The point to the class of its nearest neighbor



Input K=3, and the point ★
the majority vote is orange hence
this point is
classified orange

Application of K-nearest neighbors: *K Neighbors Classifier*

This is an supervised classifier that takes into consideration the nearest k-points to an input point to determine the class that point belongs to. The idea is to use the *majority vote* of the neighbors of the input points.

In sklearn :

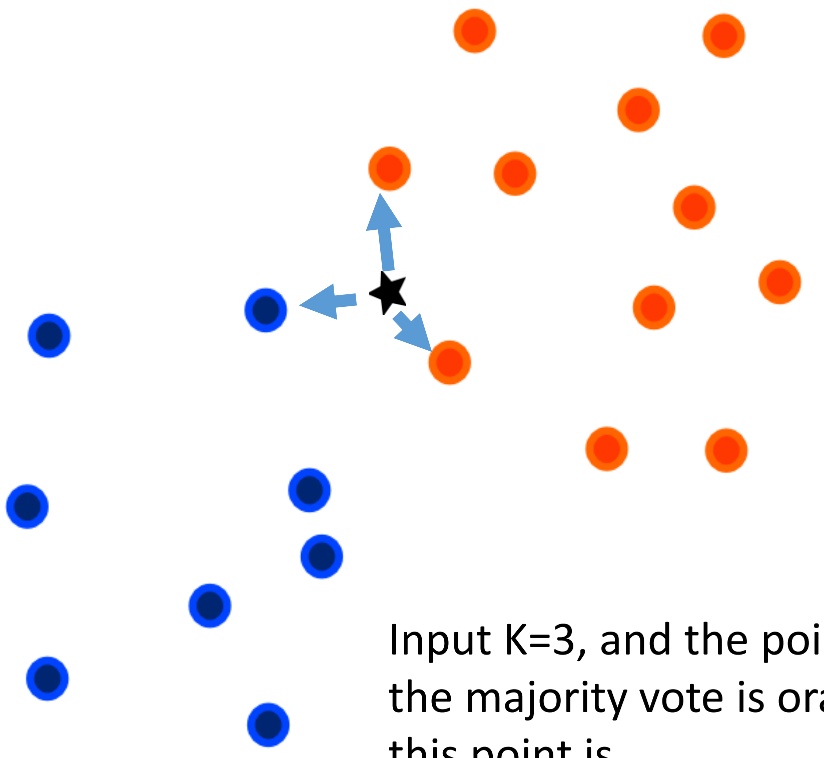
```
from sklearn.neighbors import KNeighborsClassifier # import the classifier

X = [[0], [1], [2], [3]]                          # the data
y = [0, 0, 1, 1]                                   # the labels

neigh = KNeighborsClassifier(n_neighbors=3)  # define the parameters of the classifie
neigh.fit(X, y)                              # fit the data
```

If k=1, this classifier simply assigns the
The point to the class of its nearest neighbor

What happens when k is huge ?



Input K=3, and the point ★
the majority vote is orange hence
this point is
classified orange

Application of K-nearest neighbors: *K Neighbors Classifier*

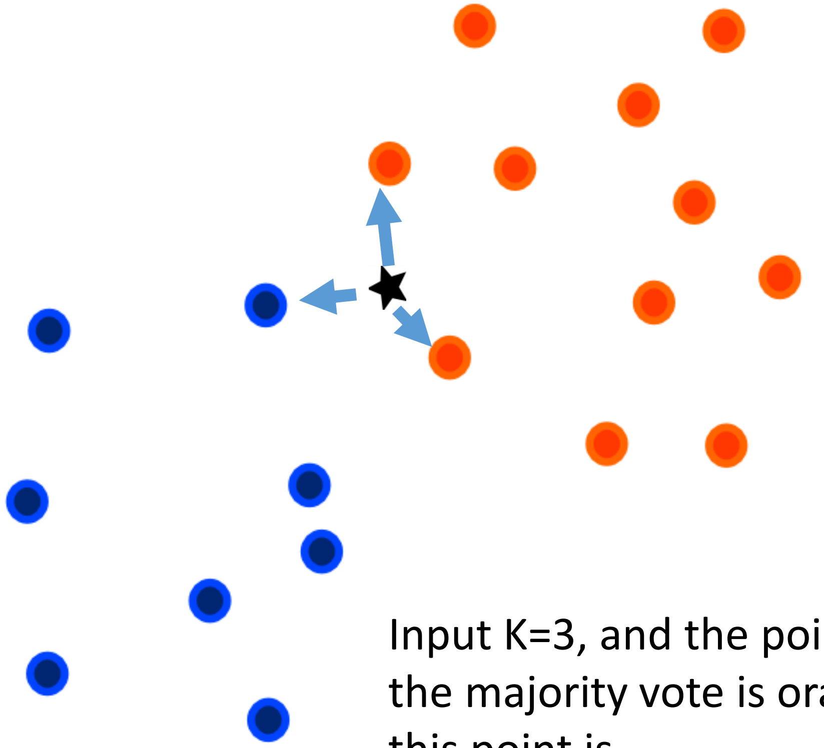
This is an supervised classifier that takes into consideration the nearest k-points to an input point to determine the class that point belongs to. The idea is to use the *majority vote* of the neighbors of the input points.

In sklearn :

```
from sklearn.neighbors import KNeighborsClassifier # import the classifier

X = [[0], [1], [2], [3]]                            # the data
y = [0, 0, 1, 1]                                     # the labels

neigh = KNeighborsClassifier(n_neighbors=3)  # define the parameters of the classifie
neigh.fit(X, y)                              # fit the data
```



If k=1, this classifier simply assigns the
The point to the class of its nearest neighbor

What happens when k is huge ?

We can choose a different distance function for this classifier
to obtain different results –depending on the data this might be desirable.

Input K=3, and the point ★
the majority vote is orange hence
this point is
classified orange

Application of K-nearest neighbors: *K Neighbors Classifier*

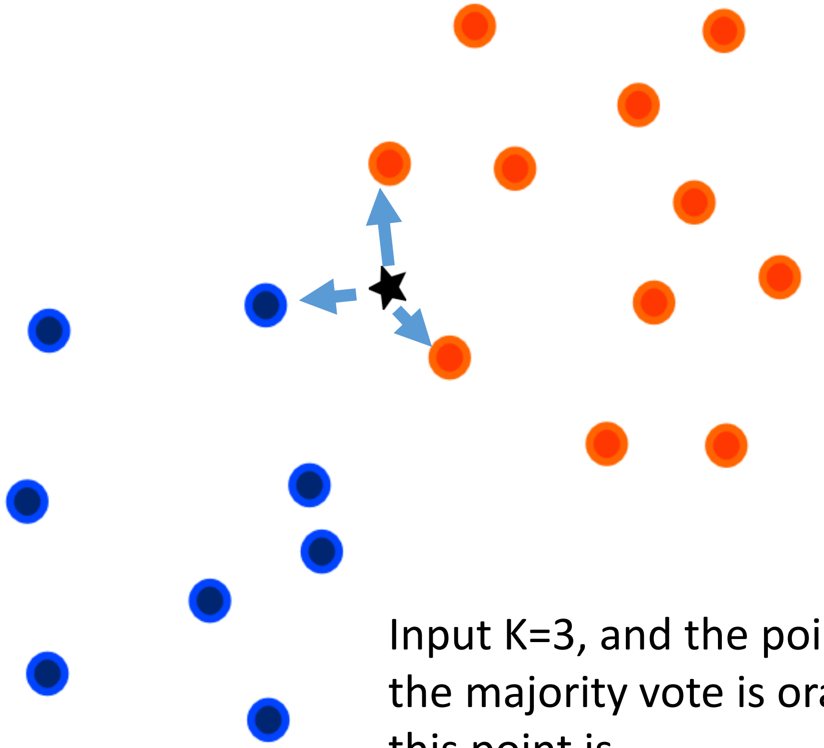
This is an supervised classifier that takes into consideration the nearest k-points to an input point to determine the class that point belongs to. The idea is to use the *majority vote* of the neighbors of the input points.

In sklearn :

```
from sklearn.neighbors import KNeighborsClassifier # import the classifier

X = [[0], [1], [2], [3]]                           # the data
y = [0, 0, 1, 1]                                    # the labels

neigh = KNeighborsClassifier(n_neighbors=3)  # define the parameters of the classifie
neigh.fit(X, y)                              # fit the data
```



If k=1, this classifier simply assigns the
The point to the class of its nearest neighbor

What happens when k is huge ?

We can choose a different distance function for this classifier
to obtain different results –depending on the data this might be desirable.

Input K=3, and the point ★
the majority vote is orange hence
this point is
classified orange

See this sklearn example

Application of ε-nearest neighbors: *Radius Neighbors Classifier*

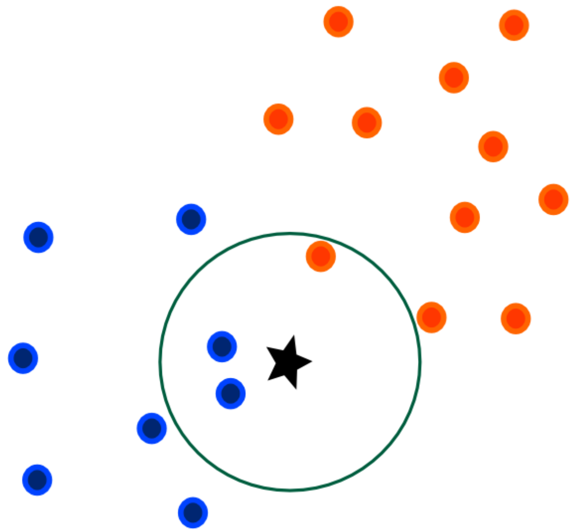
The idea is really similar to before, here instead of consider the closest k-nearest neighbor to determine the class, we Consider all instances within radius ε to determine the class of the input point

In sklearn :

```
from sklearn.neighbors import RadiusNeighborsClassifier

X = [[0], [1], [2], [3]]
y = [0, 0, 1, 1]

neigh = RadiusNeighborsClassifier(radius=1.0)
neigh.fit(X, y)                              # fit the data
```



Sklearn [example]

Neighborhood graphs and their relatives (review)

K-Nearest Neighbor Graph (KNN 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.

Sklearn implementation

K-Nearest Neighbor Graph (KNN 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.

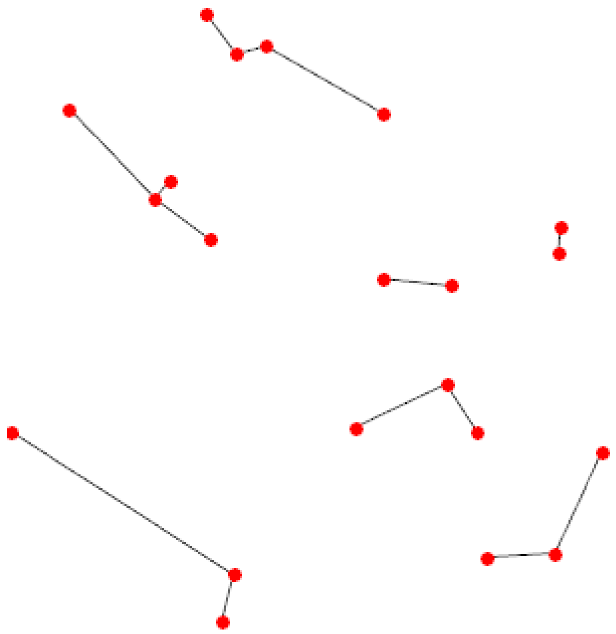
For a fixed integer k, connect the points x, y in X if either $d(x,y) \leq d(x, x_k)$ or $d(x,y) \leq d(y, y_k)$ where x_k, y_k are the k^{th} nearest neighbors of x, y respectively. Doing do for all points we obtain a graph $G(X, E)$ where E is the set of edges connect the points in X as described above.

Sklearn implementation

K-Nearest Neighbor Graph (KNN 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.

For a fixed integer k, connect the points x, y in X if either $d(x,y) \le d(x, x_k)$ or $d(x,y) \le d(y, y_k)$ where x_k, y_k are the k^{th} nearest neighbors of x, y respectively. Doing do for all points we obtain a graph $G(X,E)$ where E is the set of edges connect the points in X as described above.



Example of 1-NN graph

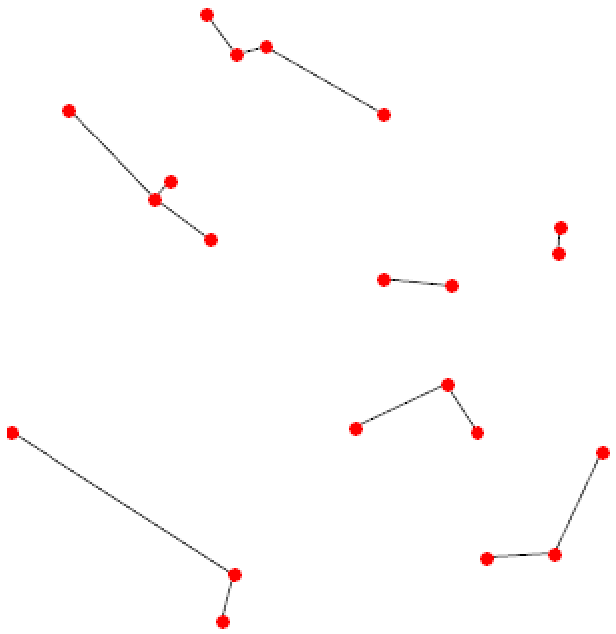
Sklearn implementation

The graph that we obtain is called the K-nearest neighbor graph or K-NN graph.

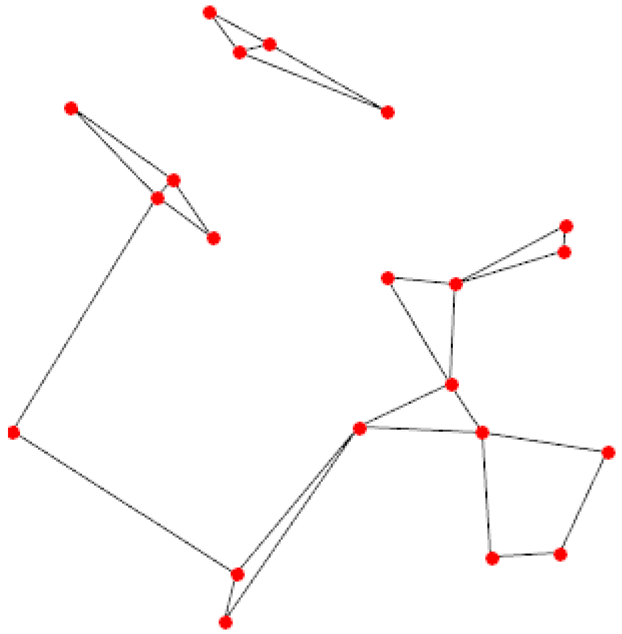
K-Nearest Neighbor Graph (KNN Graph)

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.

For a fixed integer k, connect the points x, y in X if either $d(x, y) \le d(x, x_k)$ or $d(x, y) \le d(y, y_k)$ where x_k, y_k are the k^{th} nearest neighbors of x, y respectively. Doing do for all points we obtain a graph $G(X, E)$ where E is the set of edges connect the points in X as described above.



Example of 1-NN graph



Example of 2-NN graph

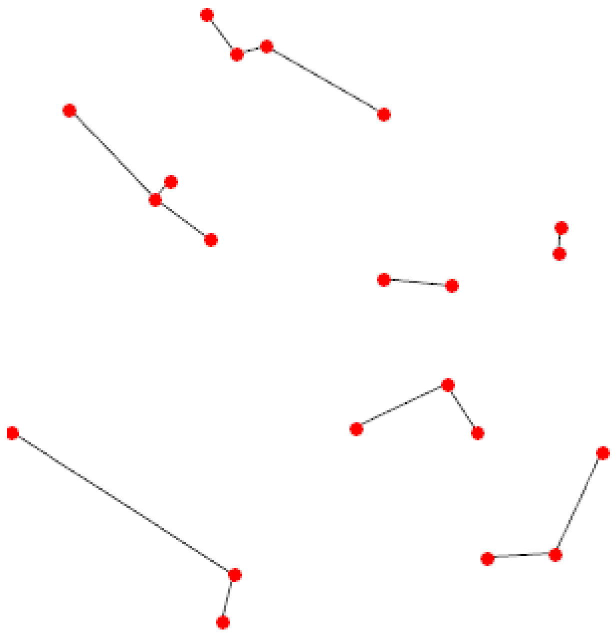
Sklearn implementation

The graph that we obtain is called the K-nearest neighbor graph or K-NN graph.

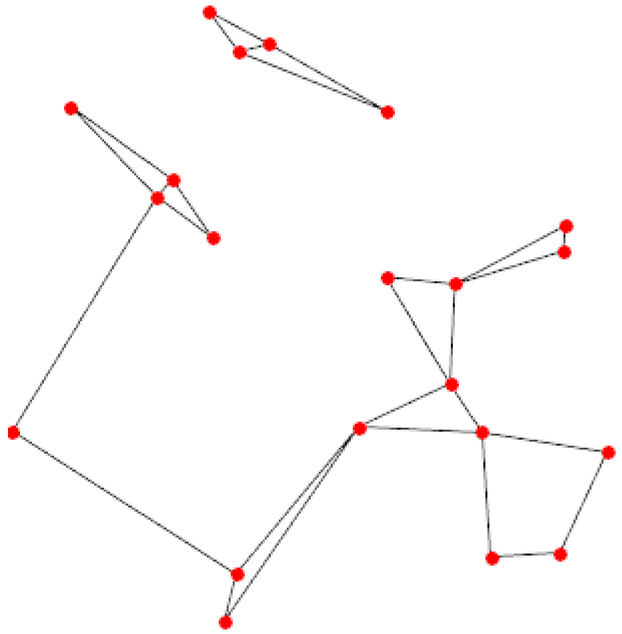
K-Nearest Neighbor Graph (KNN Graph)

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.

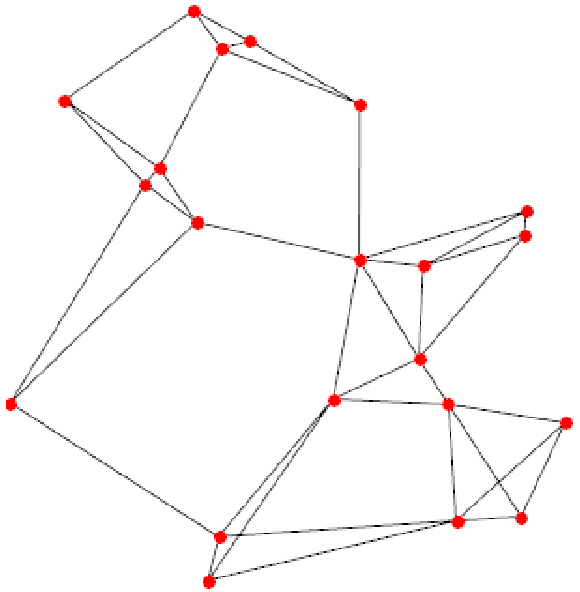
For a fixed integer k, connect the points x, y in X if either $d(x, y) \leq d(x, x_k)$ or $d(x, y) \leq d(y, y_k)$ where x_k, y_k are the k^{th} nearest neighbors of x, y respectively. Doing do for all points we obtain a graph $G(X, E)$ where E is the set of edges connect the points in X as described above.



Example of 1-NN graph



Example of 2-NN graph



Example of 3-NN graph

Sklearn implementation

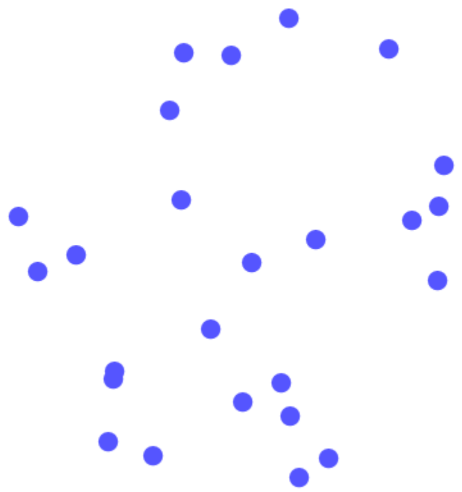
The graph that we obtain is called the K-nearest neighbor graph or K-NN graph.

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.

For a fixed ε, connect the points x, y if $d(x, y) \leq$ ε.

Doing do for all points we obtain a graph $G(X, E)$ where E is the set of edges connect the points in X as described above.



Sklearn implementation

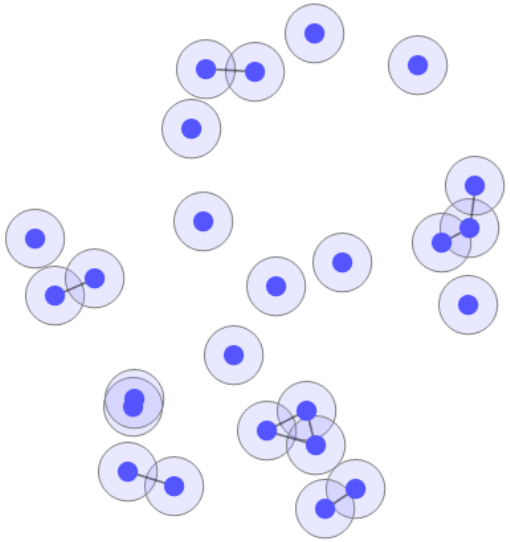
ε- neighborhood graph

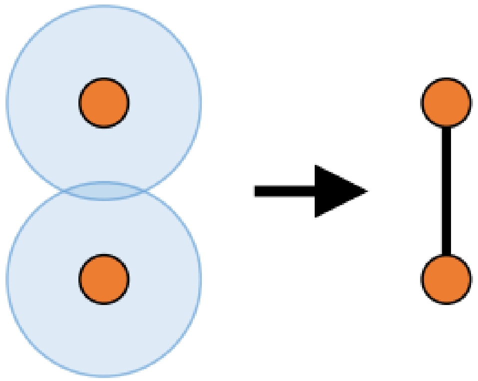
Suppose that we are given a set of points $X = \{p_1, p_2, \ldots, p_n\}$ in R^d with a distance function d defined one them.

For a fixed ε, connect the points x, y if $d(x, y) \leq$ ε.

Doing do for all points we obtain a graph $G(X, E)$ where E is the set of edges connect the points in X as described above.

This is *equivalent* to :
Insert an edge if the disks around the points intersect each other.





Note that $d(x, y) \leq \frac{\varepsilon}{2}$ if and only if the two balls surrounding x and y intersect

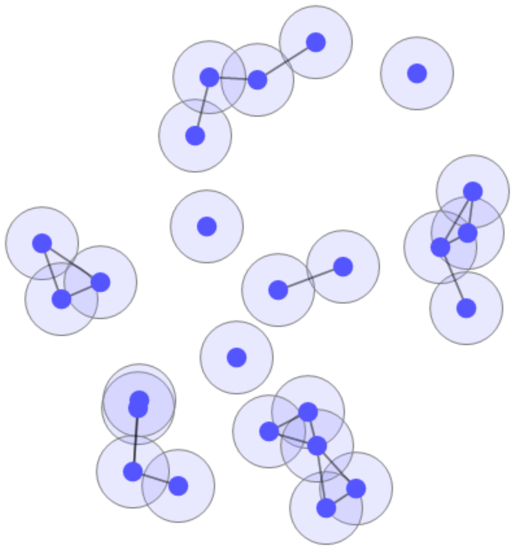
Sklearn implementation

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.

For a fixed ε, connect the points x, y if $d(x, y) \leq$ ε.

Doing do for all points we obtain a graph $G(X, E)$ where E is the set of edges connect the points in X as described above.



Trying different ε

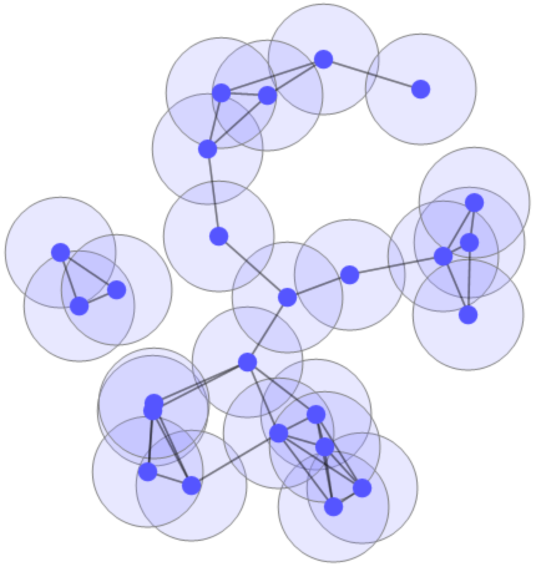
Sklearn implementation

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.

For a fixed ε, connect the points x, y if $d(x, y) \leq$ ε.

Doing do for all points we obtain a graph $G(X, E)$ where E is the set of edges connect the points in X as described above.



Trying different ε

Sklearn implementation

Euclidian Minimal Spanning Tree (EMST)

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.

The EMST of X is a minimal spanning tree where the weight of the edge between each pair of points is the distance between those two points.

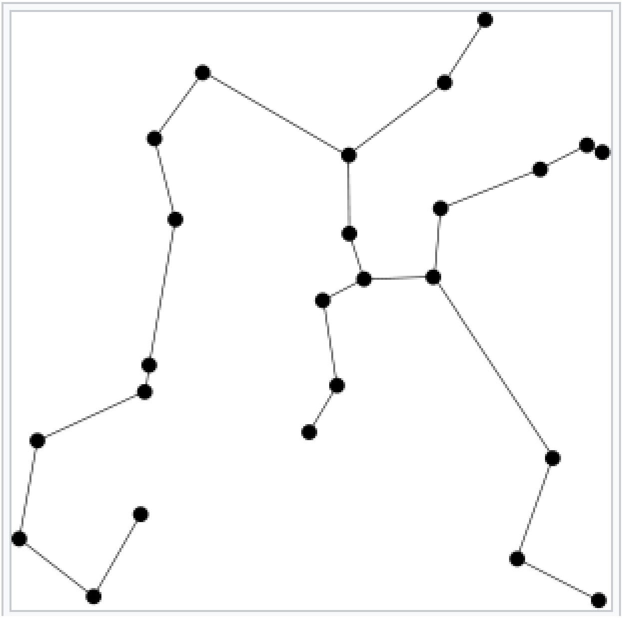


Image source-Wikipedia article

Graph Based Clustering Algorithms : Zahn's algorithm

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 it.

1. Construct the EMST of X.
2. Remove the *inconsistent edges* to obtain a collection of connected components (clusters).
3. Repeat step (2) as long as the termination condition is not satisfied.

Graph Based Clustering Algorithms : Zahn's algorithm

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 it.

1. Construct the EMST of X.
2. Remove the *inconsistent edges* to obtain a collection of connected components (clusters).
3. Repeat step (2) as long as the termination condition is not satisfied.

In this case, an edge in the tree is called inconsistent if it has a length more than a certain given length L

Graph Based Clustering Algorithms : Zahn's algorithm

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 it.

1. Construct the EMST of X.
2. Remove the *inconsistent edges* to obtain a collection of connected components (clusters).
3. Repeat step (2) as long as the termination condition is not satisfied.

In this case, an edge in the tree is called inconsistent if it has a length more than a certain given length L

Graph Based Clustering Algorithms : Zahn's algorithm

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 it.

1. Construct the EMST of X.
2. Remove the *inconsistent edges* to obtain a collection of connected components (clusters).
3. Repeat step (2) as long as the termination condition is not satisfied.

In this case, an edge in the tree is called inconsistent if it has a length more than a certain given length L

Alternatively, we could simply consider the number of desired clusters *k* as an input

Graph Based Clustering Algorithms : Zahn's algorithm

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 it.

1. Construct the EMST of X.
2. Remove the *inconsistent edges* to obtain a collection of connected components (clusters).
3. Repeat step (2) as long as the termination condition is not satisfied.

In this case, an edge in the tree is called inconsistent if it has a length more than a certain given length L

Alternatively, we could simply consider the number of desired clusters k as an input

Note that deleting *k* edges from the spanning tree results in *k+1* connected components. In particular when *k=1*, we obtain 2 subtrees

Graph Based Clustering Algorithms : Zahn's algorithm

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 it.

1. Construct the EMST of X.
2. Remove the *inconsistent edges* to obtain a collection of connected components (clusters).
3. Repeat step (2) as long as the termination condition is not satisfied.

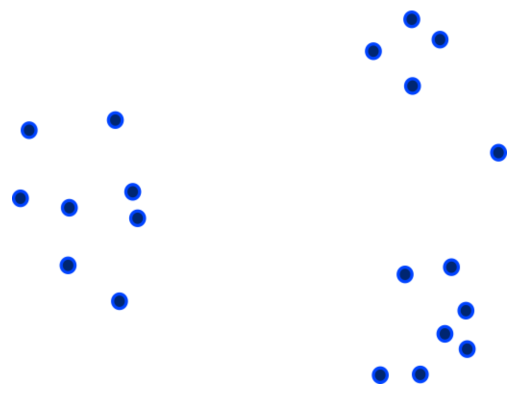
In this case, an edge in the tree is called inconsistent if it has a length more than a certain given length L

Alternatively, we could simply consider the number of desired clusters *k* as an input

Note that deleting *k* edges from the spanning tree results in *k+1* connected components. In particular when *k=1*, we obtain 2 subtrees

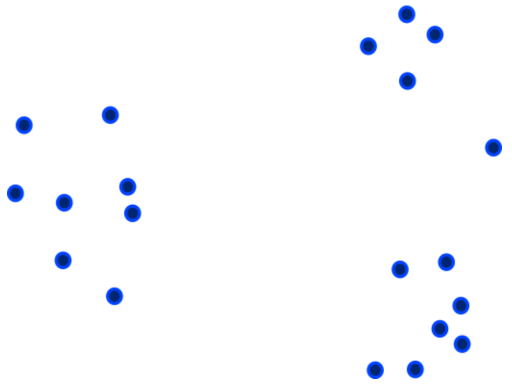
This definition of consistent is not always ideal!. See here

Graph Based Clustering Algorithms : Zahn’s algorithm

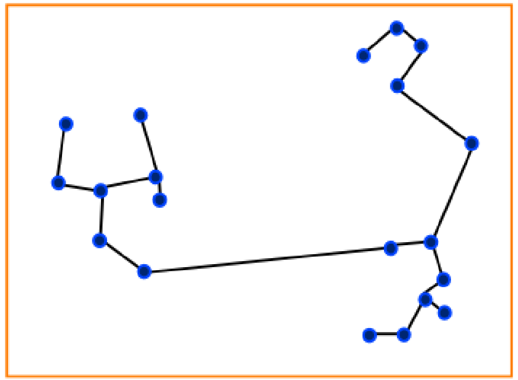


Input : point cloud and
Number of connected
components k

Graph Based Clustering Algorithms : Zahn’s algorithm

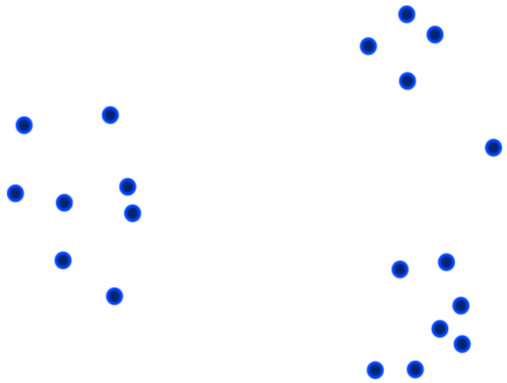


Input : point cloud and Number of connected components k

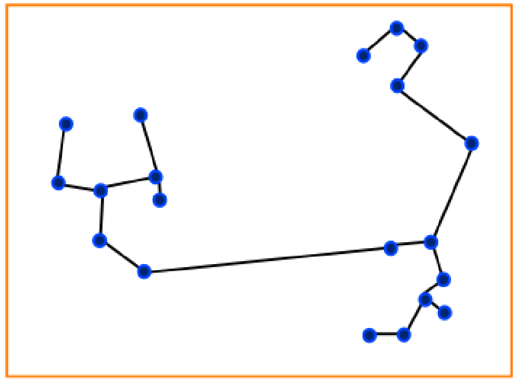


Construct EMST

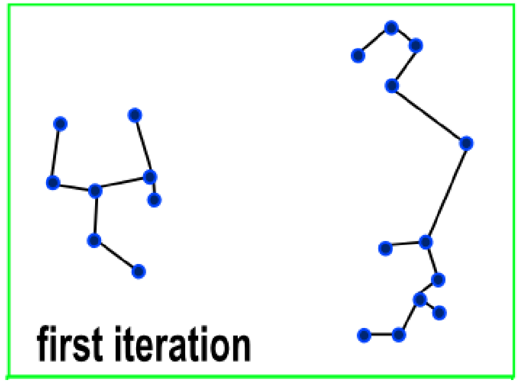
Graph Based Clustering Algorithms : Zahn's algorithm



Input : point cloud and Number of connected components k

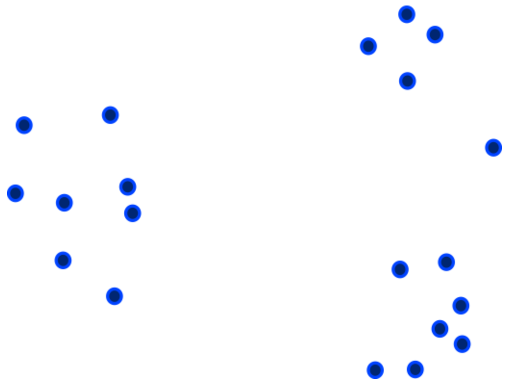


Construct EMST

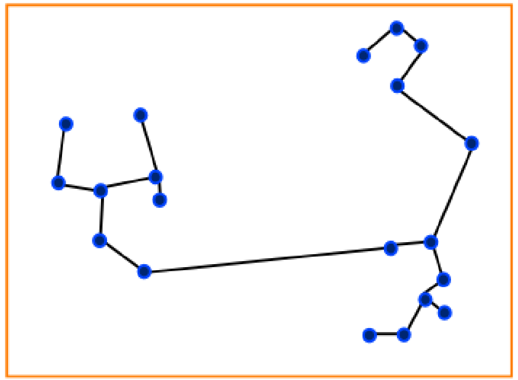


Delete an edge

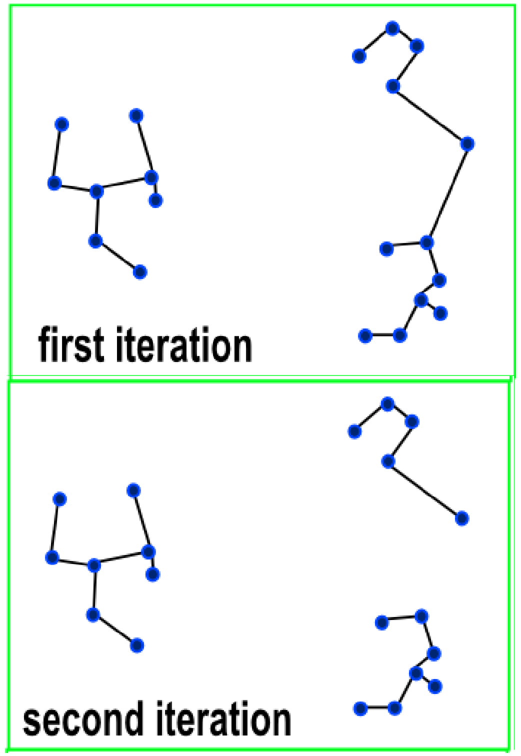
Graph Based Clustering Algorithms : Zahn's algorithm



Input : point cloud and Number of connected components k

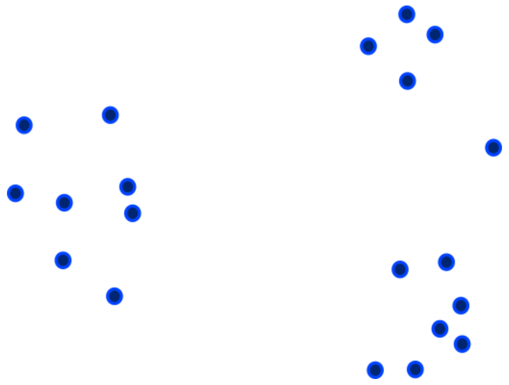


Construct EMST

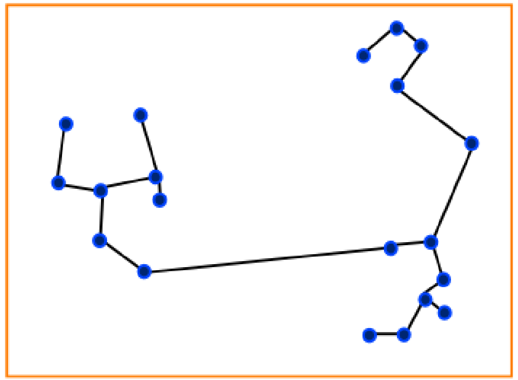


Delete an edge

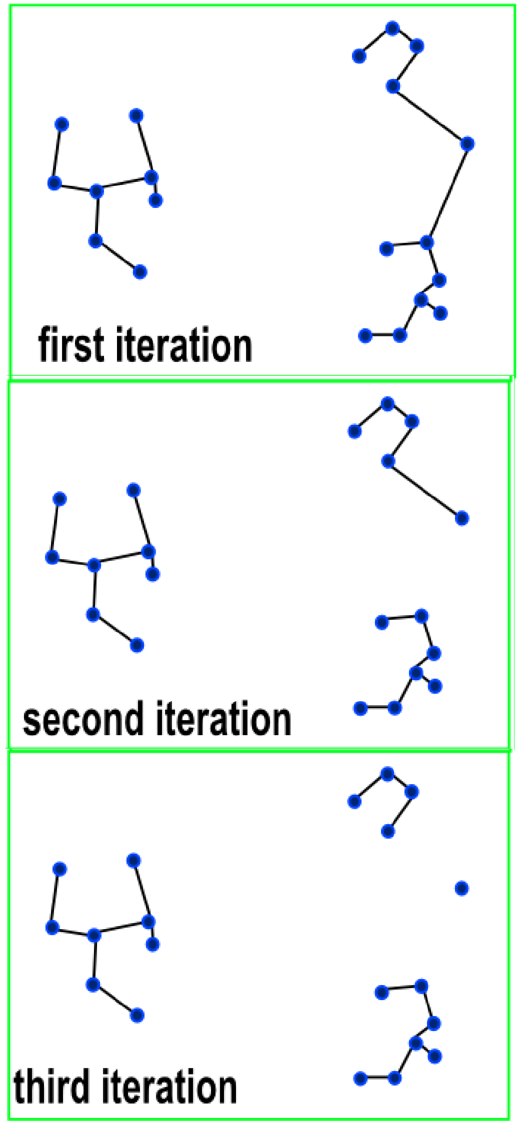
Graph Based Clustering Algorithms : Zahn's algorithm



Input : point cloud and Number of connected components k

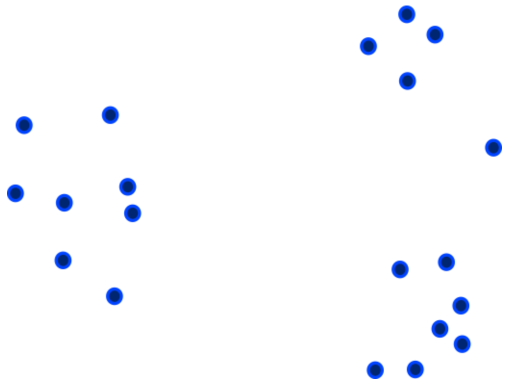


Construct EMST

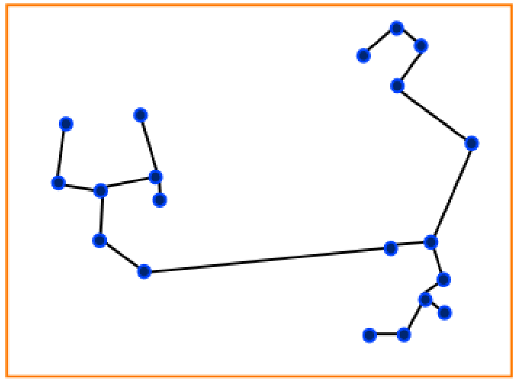


Delete an edge

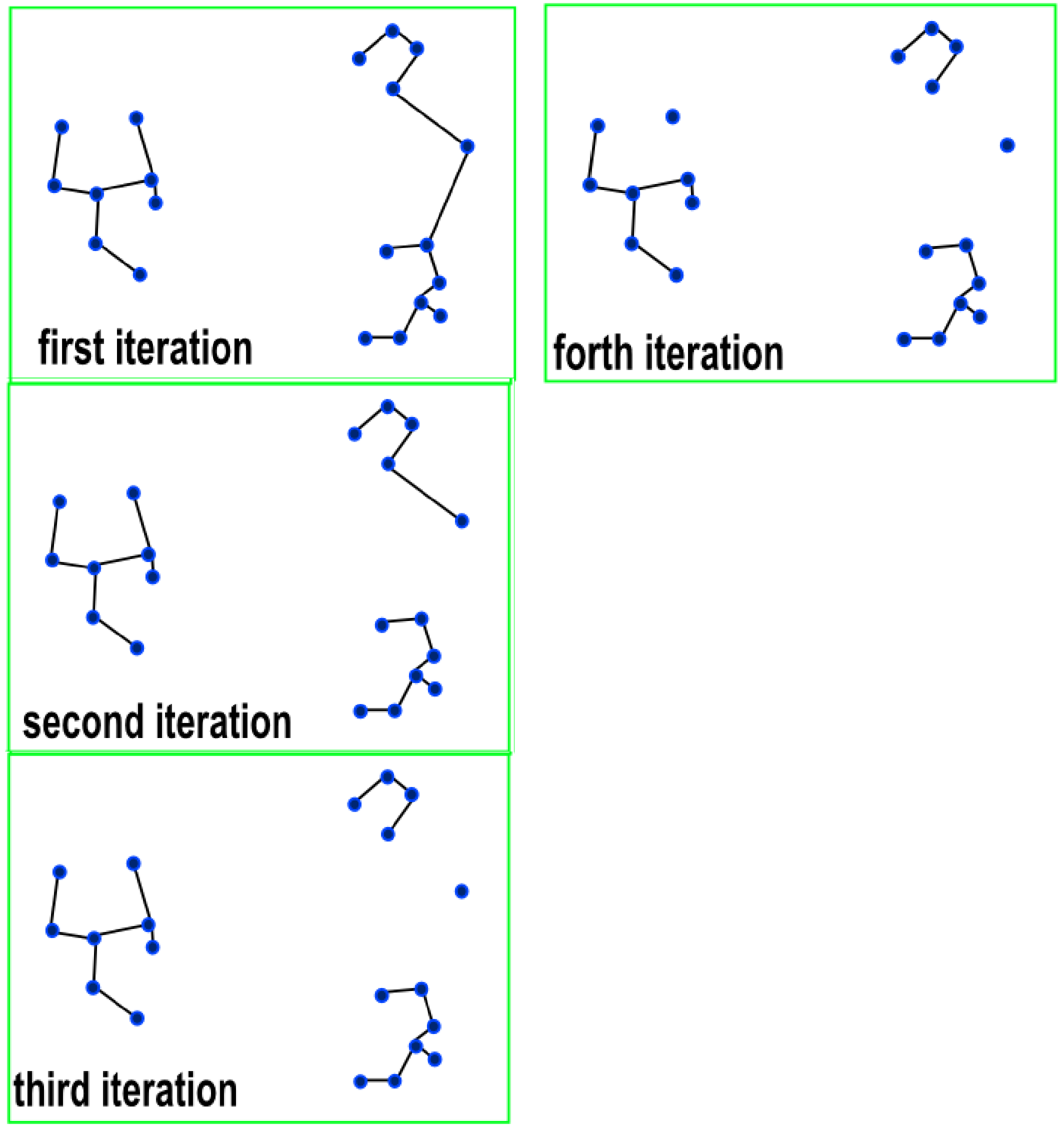
Graph Based Clustering Algorithms : Zahn’s algorithm



Input : point cloud and
Number of connected
components k



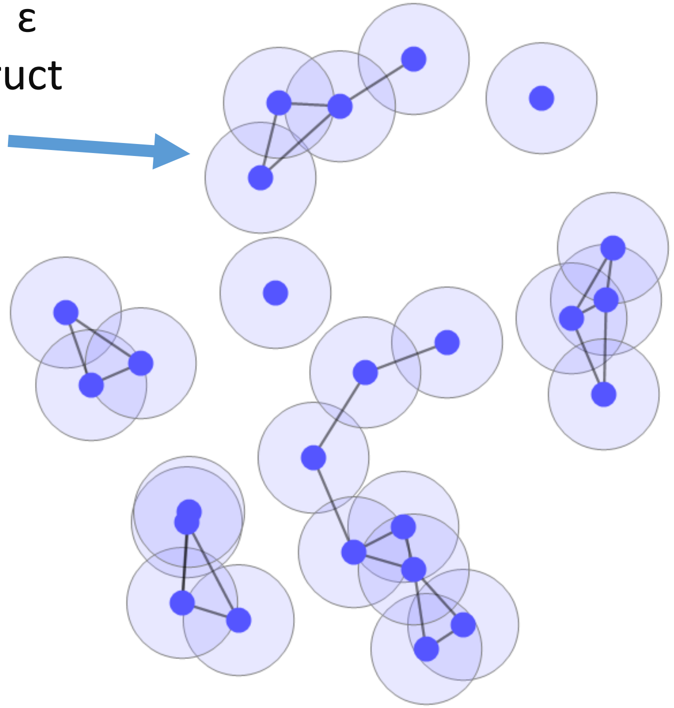
Construct EMST



Until the termination criterion is satisfied

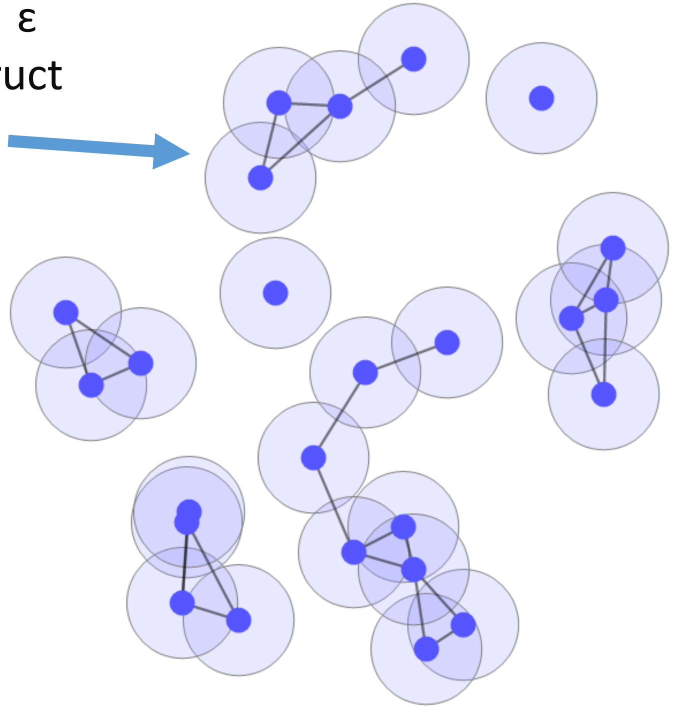
Graph Based Clustering Algorithms : ε- neighborhood graph

Choose an ε
And construct
the graph

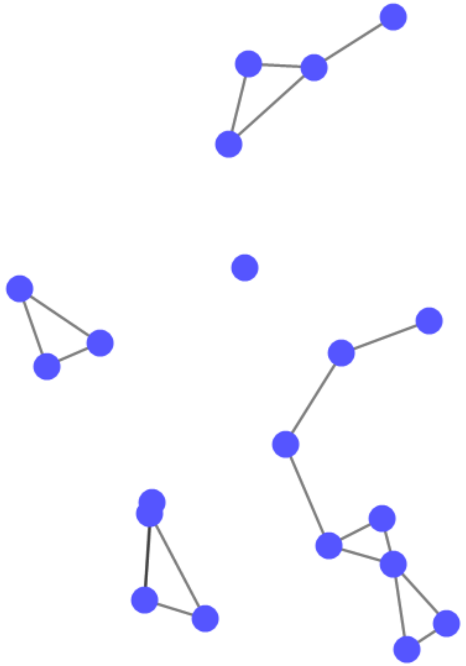


Graph Based Clustering Algorithms : ε- neighborhood graph

Choose an ε
And construct
the graph



Consider the
connected
components



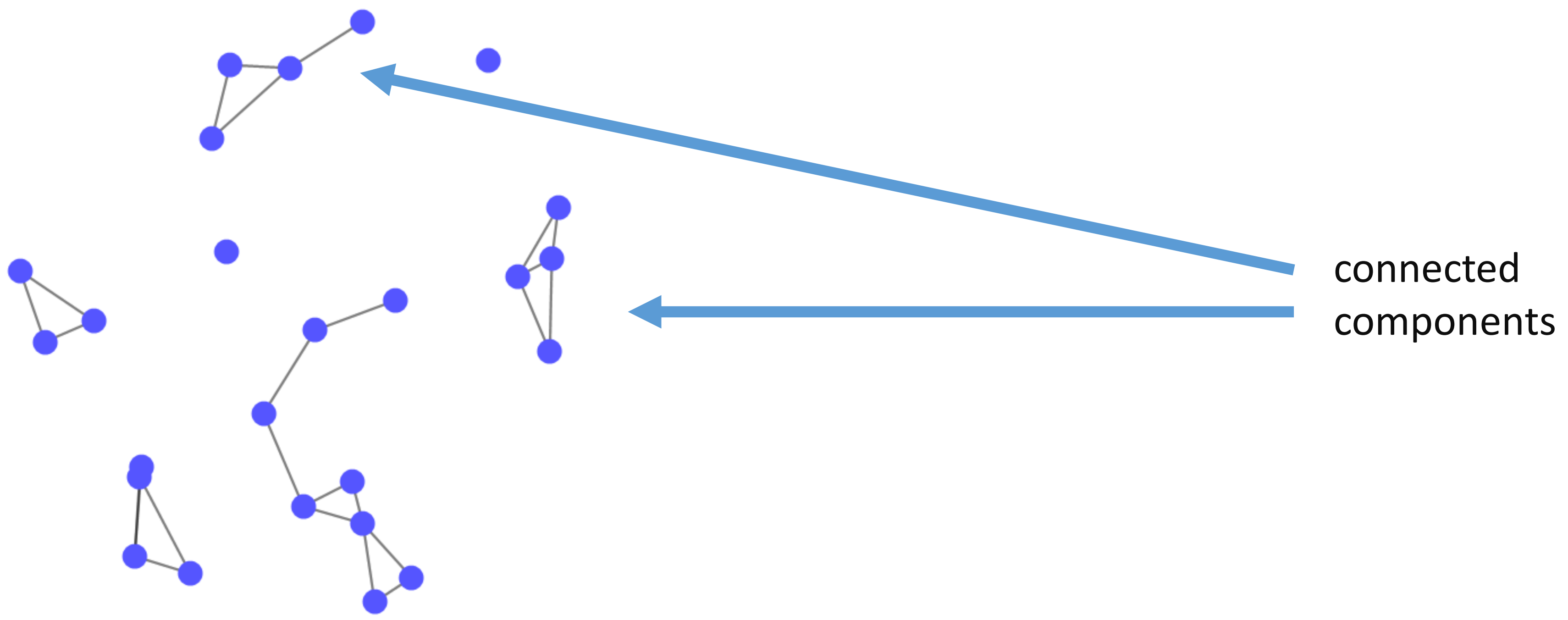
the points of
each connected
component form
a cluster

Recall : connected components of a graph

What is a connected component of a graph ?

How can we find the connected components of a graph ?

See this video for a review



Networkx

Introduction on networkx and how to use its basic functions.

```
import networkx as nx
G=nx.Graph()
G.add_edge(1,2)  # default edge data=1
G.add_edge(2,3,weight=0.9) # specify edge data
```

Finding connected components of a graph using networkx. See here

```
import networkx as nx
G = nx.path_graph(4)
G.add_path([10, 11, 12])
sorted(nx.connected_components(G), key = len, reverse=True)
```

Drawing a graph using networkx

```
import matplotlib.pyplot as plt
import networkx as nx
G=nx.path_graph(3)
nx.draw(G)  # networkx draw()
plt.draw()  # pyplot draw()
```