DATA MINING 1
Clustering

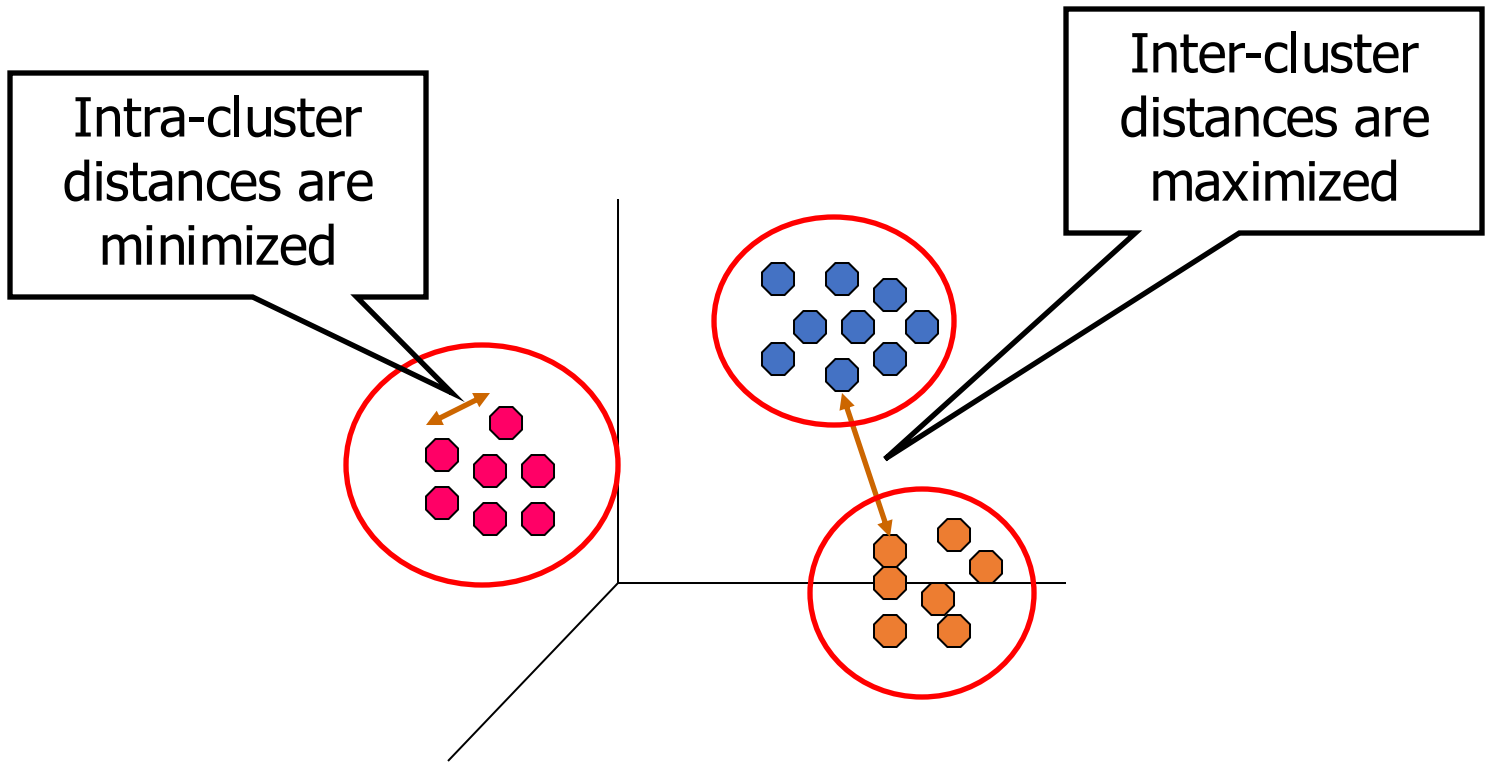
Dino Pedreschi, Riccardo Guidotti

Revisited slides from Lecture Notes for Chapter 7 “Introduction to Data Mining”, 2nd Edition by Tan, Steinbach, Karpatne, Kumar

UNIVERSITÀ DI PISA

What is Cluster Analysis?

- Finding groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects in other groups.

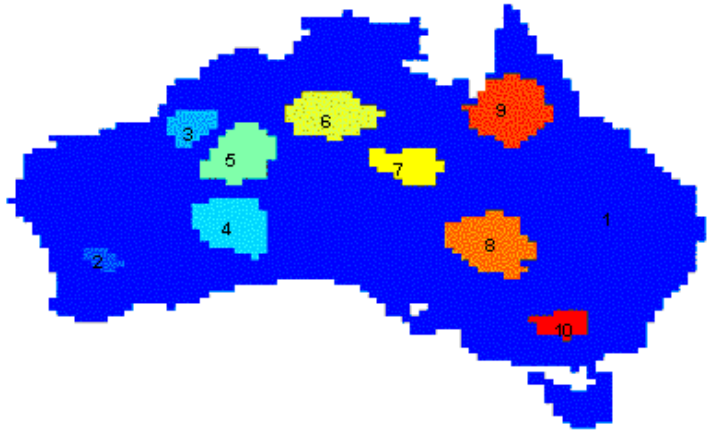


Applications of Cluster Analysis

- **Understanding**
  - Group related documents for browsing, group genes and proteins that have similar functionality, or group stocks with similar price fluctuations

- **Summarization**
  - Reduce the size of large data sets

	Discovered Clusters	*Industry Group*
1	Applied-Matl-DOWN,Bay-Network-Down,3-COM-DOWN, Cabletron-Sys-DOWN,CISCO-DOWN,HP-DOWN, DSC-Comm-DOWN,INTEL-DOWN,LSI-Logic-DOWN, Micron-Tech-DOWN,Texas-Inst-Down,Tellabs-Inc-Down, Natl-Semiconduct-DOWN,Oracl-DOWN,SGI-DOWN, Sun-DOWN	Technology1-DOWN
2	Apple-Comp-DOWN,Autodesk-DOWN,DEC-DOWN, ADV-Micro-Device-DOWN,Andrew-Corp-DOWN, Computer-Assoc-DOWN,Circuit-City-DOWN, Compaq-DOWN, EMC-Corp-DOWN, Gen-Inst-DOWN, Motorola-DOWN,Microsoft-DOWN,Scientific-Atl-DOWN	Technology2-DOWN
3	Fannie-Mae-DOWN,Fed-Home-Loan-DOWN, MBNA-Corp-DOWN,Morgan-Stanley-DOWN	Financial-DOWN
4	Baker-Hughes-UP,Dresser-Inds-UP,Halliburton-HLD-UP, Louisiana-Land-UP,Phillips-Petro-UP,Unocal-UP, Schlumberger-UP	Oil-UP



Clustering precipitation in Australia

What is not Cluster Analysis?

- Simple segmentation
  - Dividing students into different registration groups alphabetically, by last name
- Results of a query
  - Groupings are a result of an external specification
  - Clustering is a grouping of objects based on the data
- Supervised classification
  - Have class label information
- Association Analysis
  - Local vs. global connections

Notion of a Cluster can be Ambiguous

How many clusters?

Six Clusters

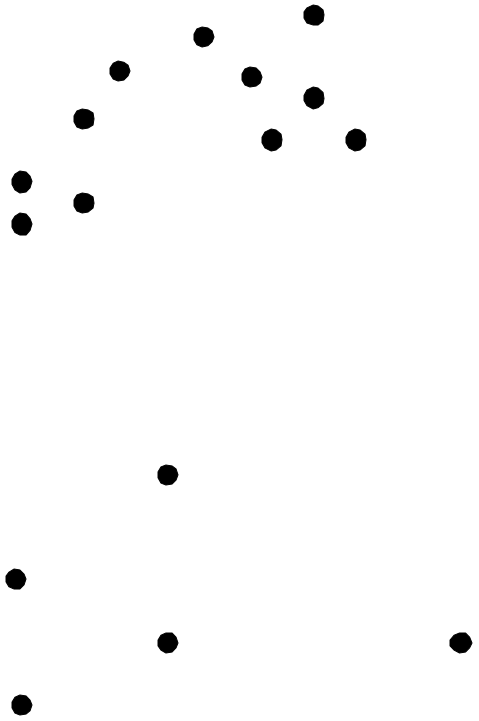
Two Clusters

Four Clusters

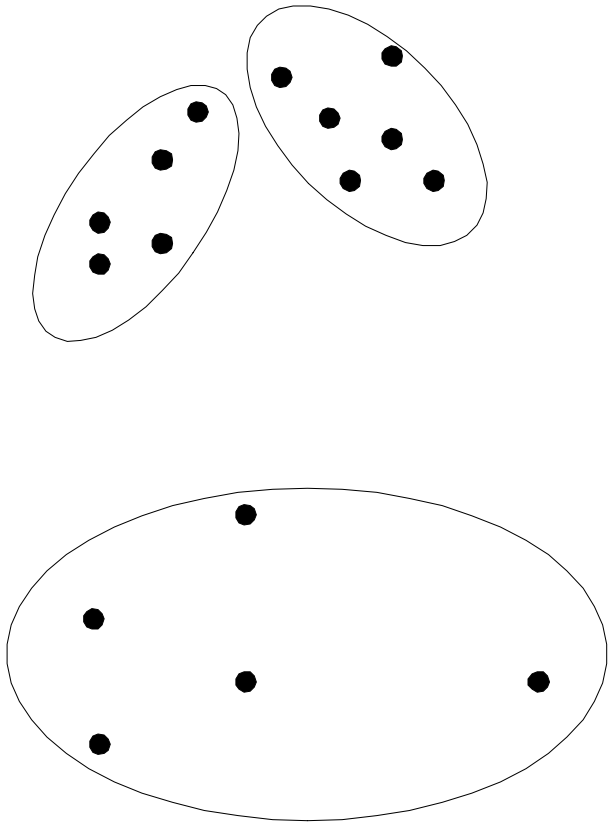
Types of Clusterings

- A clustering is a set of clusters
- Important distinction between hierarchical and partitional sets of clusters
- Partitional Clustering
  - A division of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset
- Hierarchical clustering
  - A set of nested clusters organized as a hierarchical tree

Partitional Clustering

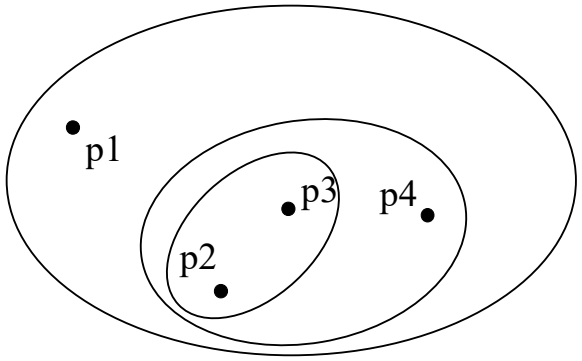


Original Points

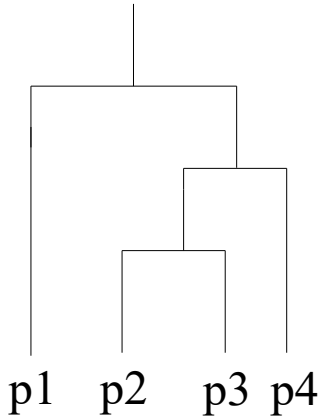


A Partitional Clustering

Hierarchical Clustering



Traditional Hierarchical Clustering



Traditional Dendrogram

Other Distinctions Between Sets of Clusters

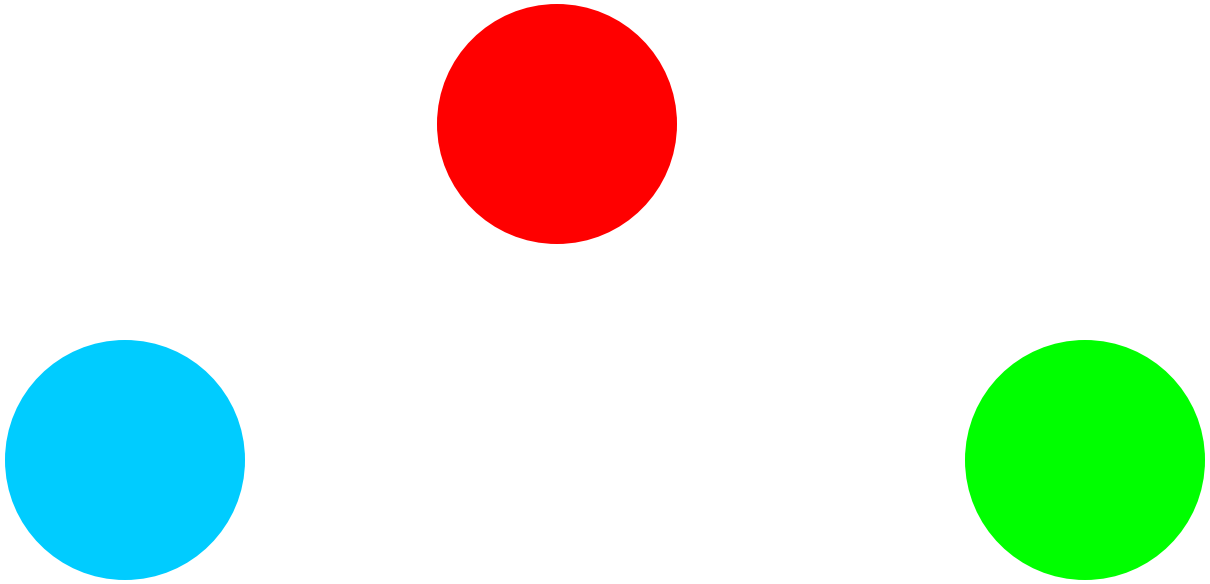
- Exclusive versus non-exclusive
  - In non-exclusive clusterings, **points may belong to multiple clusters.**
  - Can represent multiple classes or '**border**' points
- Fuzzy versus non-fuzzy
  - In fuzzy clustering, a point belongs to every cluster with some weight between 0 and 1
  - Weights must sum to 1
  - **Probabilistic** clustering has similar characteristics
- Partial versus complete
  - In some cases, we only want to cluster some of the data
- Heterogeneous versus homogeneous
  - Clusters of widely different sizes, shapes, and densities

Types of Clusters

- Well-separated clusters
- Center-based clusters
- Contiguous clusters
- Density-based clusters
- Property or Conceptual
- Described by an Objective Function

Types of Clusters: Well-Separated

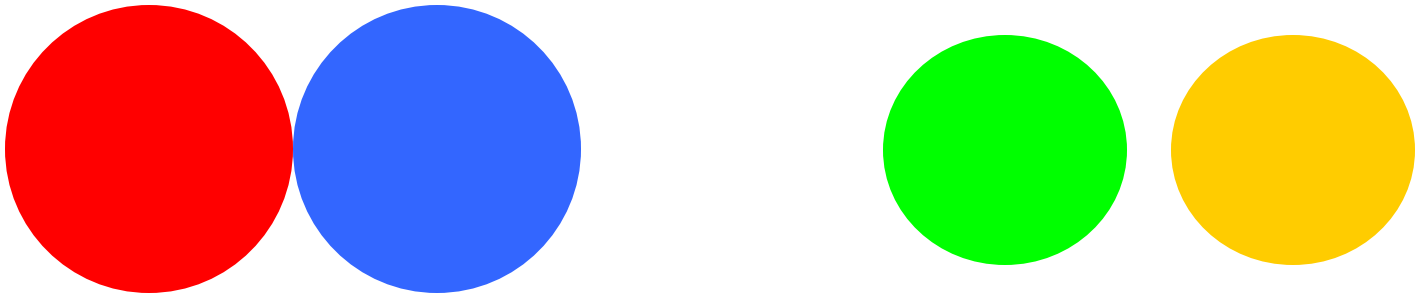
- Well-Separated Clusters:
  - A cluster is a set of points such that any point in a cluster is closer (or more similar) to every other point in the cluster than to any point not in the cluster.



3 well-separated clusters

Types of Clusters: Center-Based

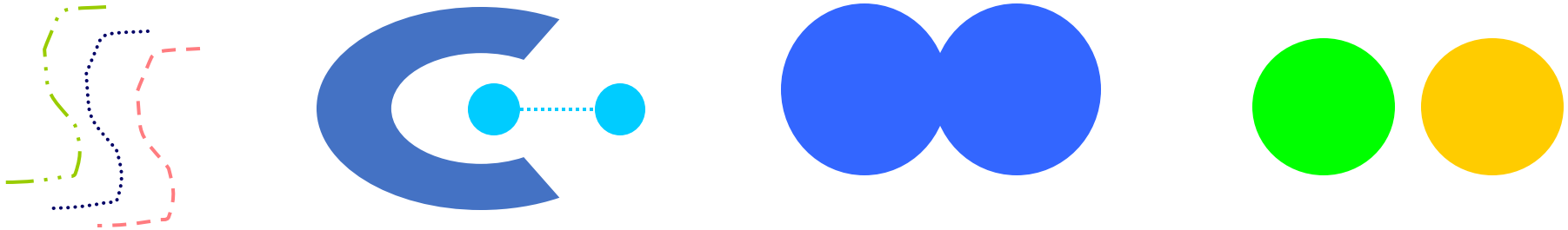
- Center-based
  - A cluster is a set of objects such that an object in a cluster is closer (more similar) to the "center" of a cluster, than to the center of any other cluster
  - The center of a cluster is often a centroid, the average of all the points in the cluster, or a medoid, the most "representative" point of a cluster



4 center-based clusters

Types of Clusters: Contiguity-Based

- Contiguous Cluster (Nearest neighbor or Transitive)
  - Each point is closer to at least one point in its cluster than to any point in another cluster.
  - Graph based clustering

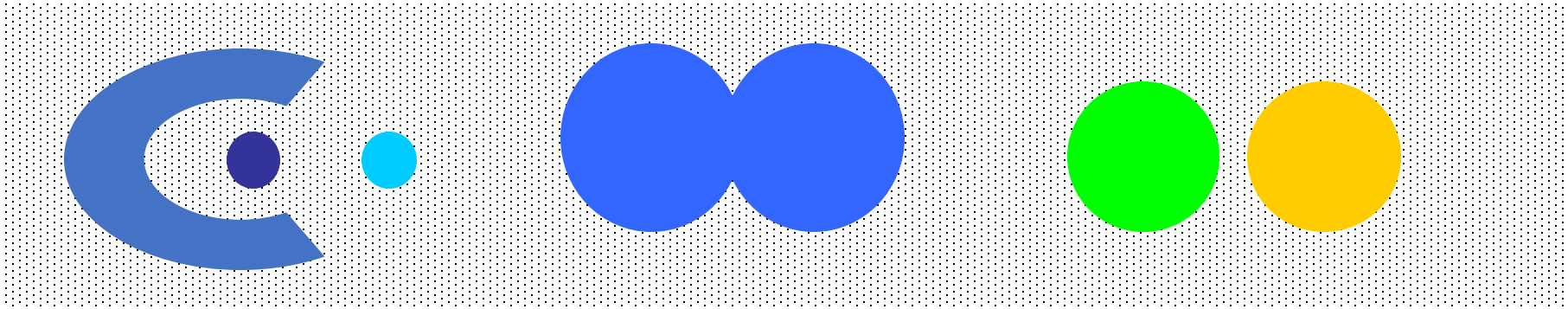


- This approach can **have trouble when noise is present** since a small bridge of points can **merge two distinct clusters**

8 contiguous clusters

Types of Clusters: Density-Based

- Density-based
  - A cluster is a dense region of points, which is separated by low-density regions, from other regions of high density.
  - Used when the clusters are irregular or intertwined, and when noise and outliers are present.



6 density-based clusters

Types of Clusters: Objective Function

- Clusters Defined by an Objective Function
  - Finds clusters that **minimize** or **maximize an objective function**.
  - Enumerate all possible ways of dividing the points into clusters and evaluate the `goodness' of each potential set of clusters by using the given objective function. (NP Hard)
  - Can have global or local objectives.
    - Hierarchical clustering algorithms typically have local objectives
    - Partitional algorithms typically have global objectives

Characteristics of the Input Data Are Important

- Type of proximity or density measure
  - Central to clustering
  - Depends on data and application
- Data characteristics that affect proximity and/or density are
  - Dimensionality and Sparseness
  - Attribute type
  - Special relationships in the data (e.g., autocorrelation)
  - Distribution of the data
- Noise and Outliers
  - Often interfere with the operation of the clustering algorithm

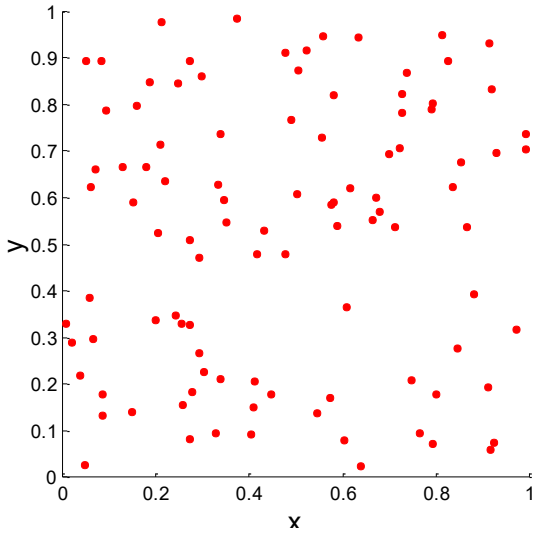
Cluster Validity

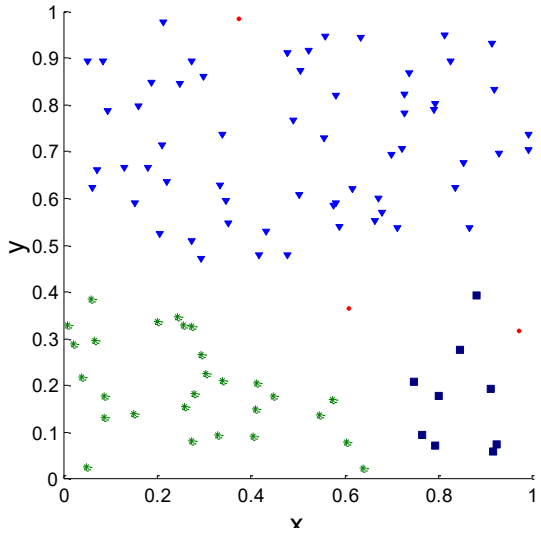
Cluster Validity

- How can we evaluate the “goodness” of the resulting clusters?
- But “clusters are in the eye of the beholder”!
- Then why do we want to evaluate them?
  - To avoid finding patterns in noise
  - To compare clustering algorithms
  - To compare two sets of clusters
  - To compare two clusters

Clusters found in Random Data

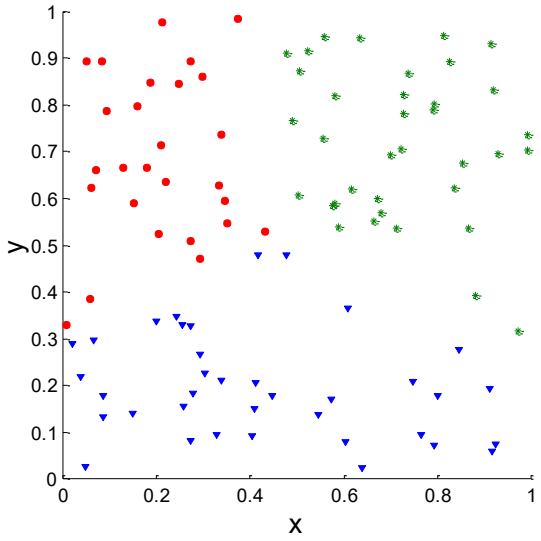
Random Points

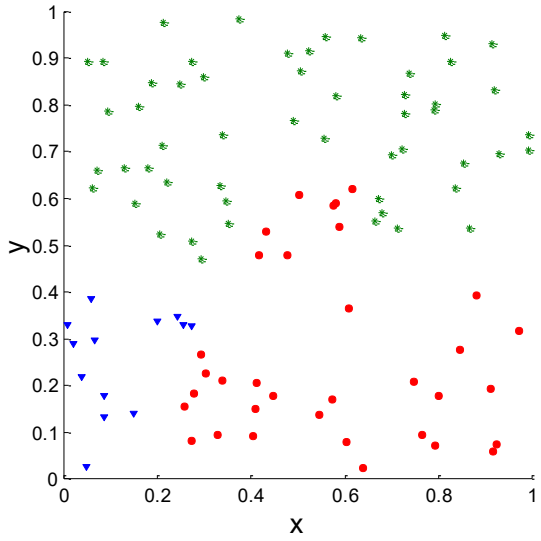




DBSCAN

K-means





Complete Link

Different Aspects of Cluster Validation

1. Determining the clustering tendency of a set of data, i.e., distinguishing whether non-random structure actually exists in the data.
2. Comparing the results of a cluster analysis to externally known results, e.g., to externally given class labels.
3. Evaluating how well the results of a cluster analysis fit the data *without* reference to external information (Use only the data).
4. Comparing the results of two different sets of cluster analyses to determine which is better.
5. Determining the 'correct' number of clusters.

   For 2, 3, and 4, we can further distinguish whether we want to evaluate the entire clustering or just individual clusters.

Measures of Cluster Validity

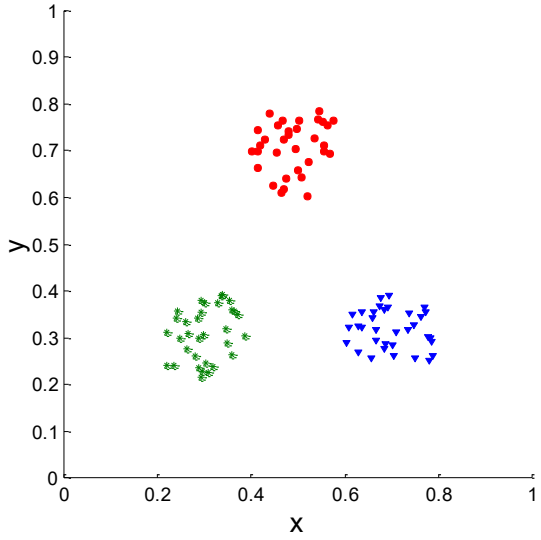
- Numerical measures that are applied to judge various aspects of cluster validity, are classified into the following three types.
  - External Index: Used to measure the extent to which cluster labels match externally supplied class labels.
    - Entropy
  - Internal Index: Used to measure the goodness of a clustering structure *without* respect to external information.
    - Sum of Squared Error (SSE)
  - Relative Index: Used to compare two different clusterings or clusters.
    - Often an external or internal index is used for this function, e.g., SSE or entropy
- Sometimes these are referred to as criteria instead of indices
  - However, sometimes criterion is the general strategy and index is the numerical measure that implements the criterion.

Measuring Cluster Validity Via Correlation

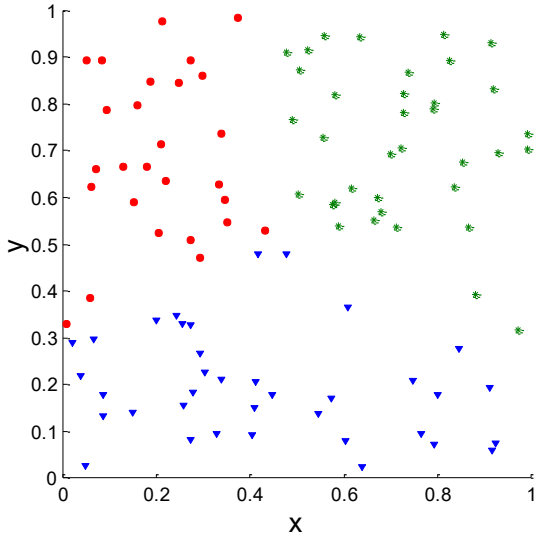
- Two matrices
  - A: Distance/Similarity Matrix
  - B: Ideal Similarity Matrix
    - One row and one column for each data point
    - An entry is 1 if the associated pair of points belong to the same cluster
    - An entry is 0 if the associated pair of points belongs to different clusters
- Compute the correlation between the two matrices A and B
  - Since the matrices are symmetric, only the correlation between n(n-1) / 2 entries needs to be calculated.
- High correlation indicates that points that belong to the same cluster are close to each other.
- Not a good measure for some density or contiguity based clusters.

Measuring Cluster Validity Via Correlation

- Correlation of ideal similarity and proximity matrices for the K-means clusterings of the following two data sets.



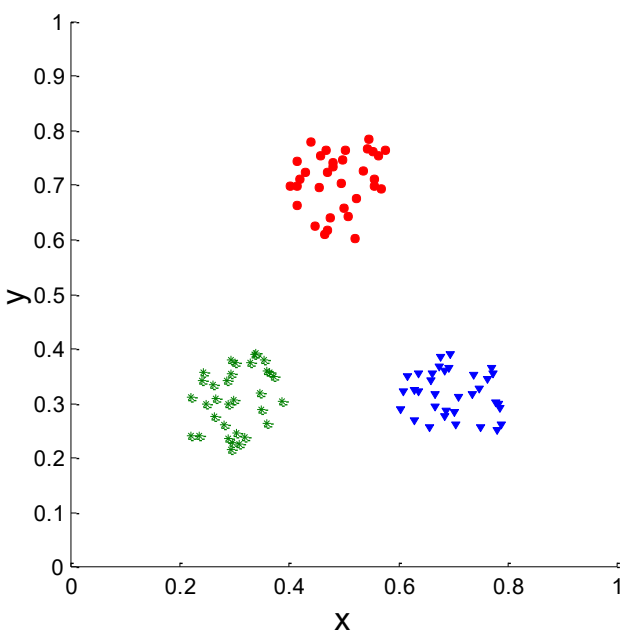
Corr = -0.9235

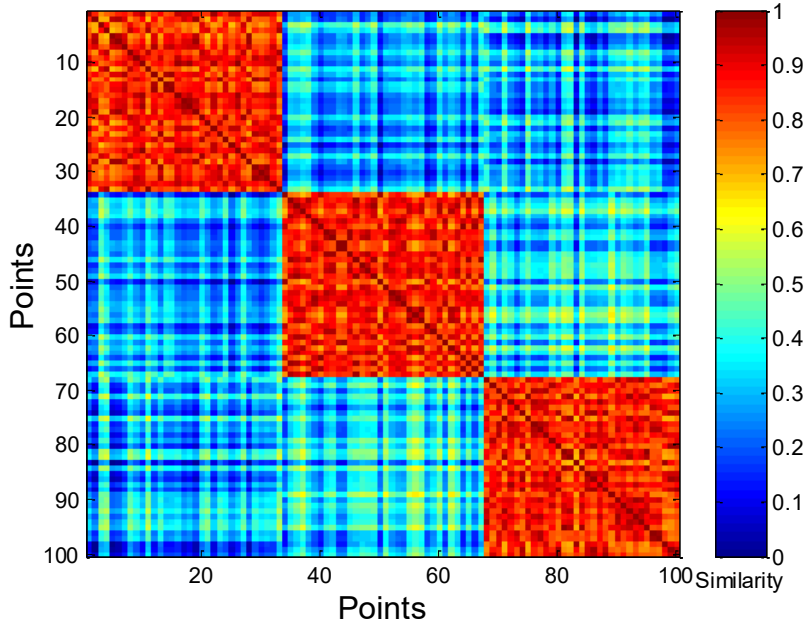


Corr = -0.5810

Using Similarity Matrix for Cluster Validation

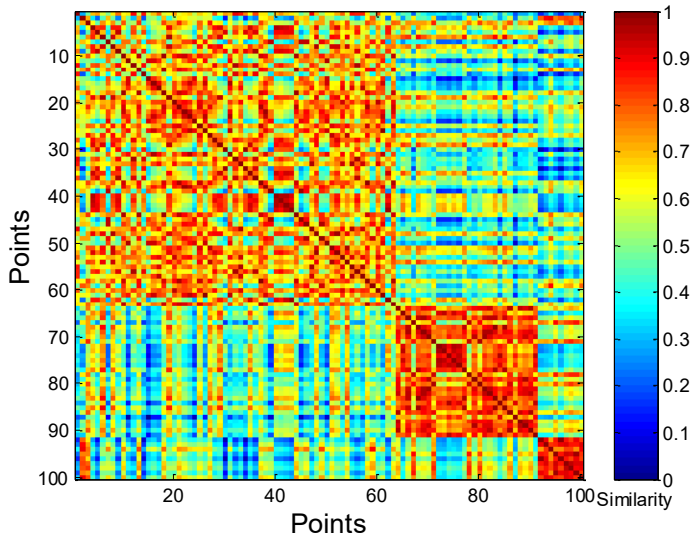
- Order the distance matrix with respect to cluster labels and inspect visually.

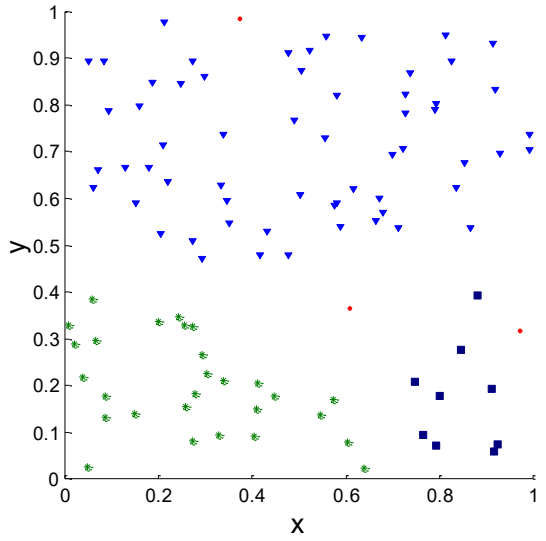




Using Similarity Matrix for Cluster Validation

- Clusters in random data are not so crisp

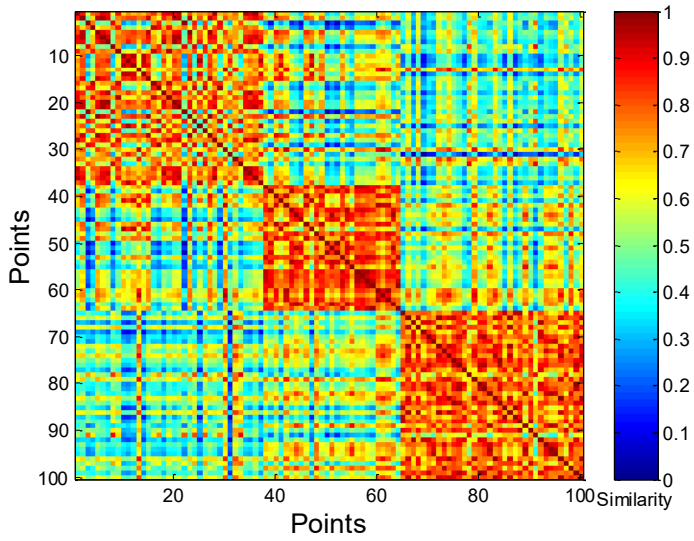


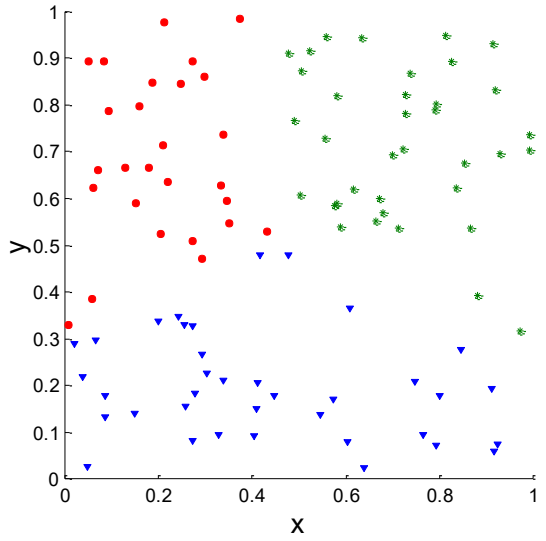


DBSCAN

Using Similarity Matrix for Cluster Validation

- Clusters in random data are not so crisp

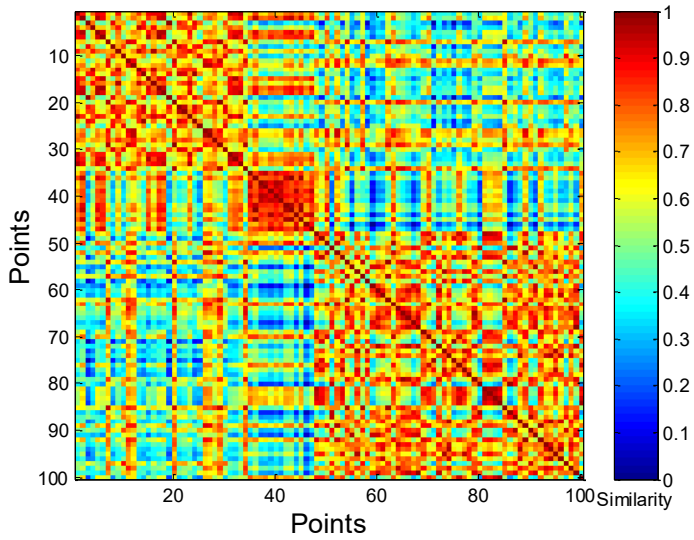


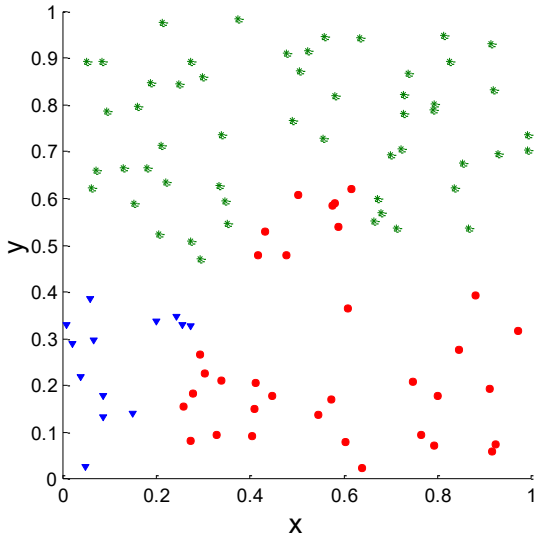


K-means

Using Similarity Matrix for Cluster Validation

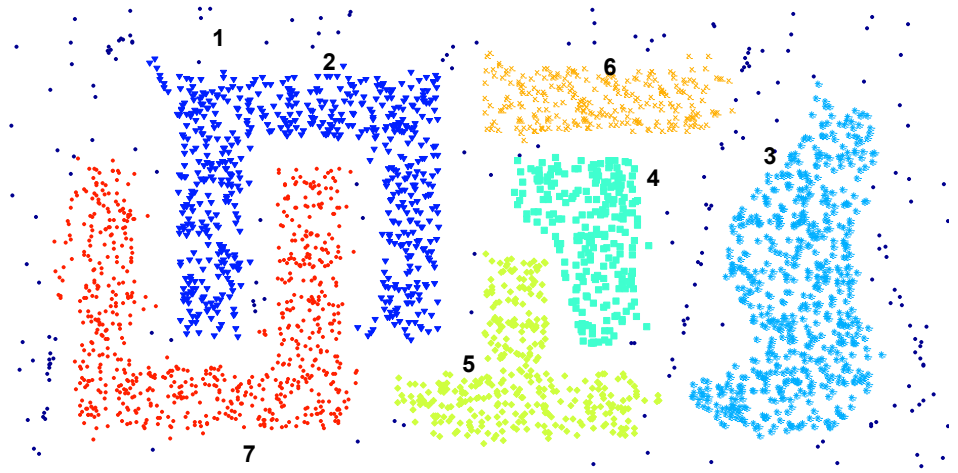
- Clusters in random data are not so crisp

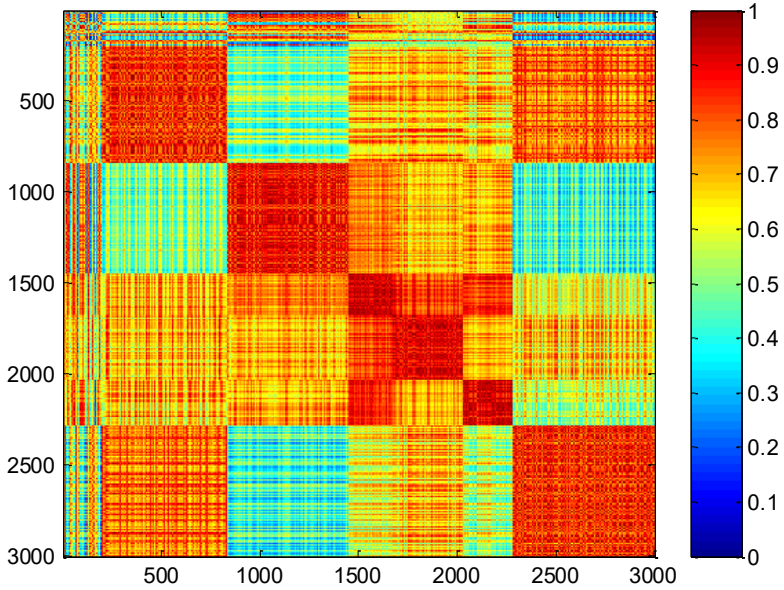




Complete Link

Using Similarity Matrix for Cluster Validation

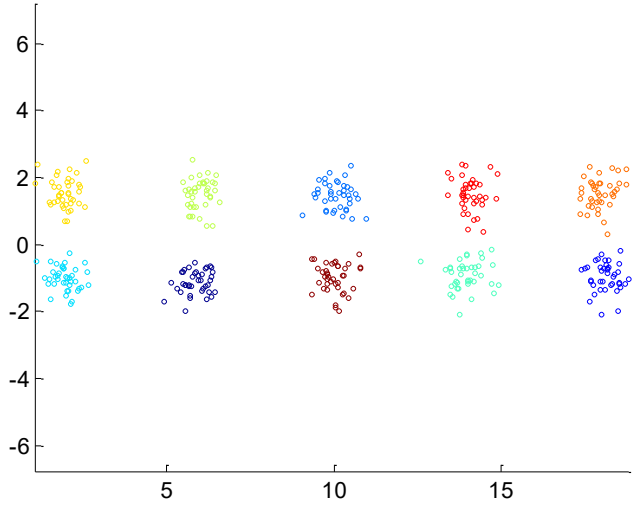


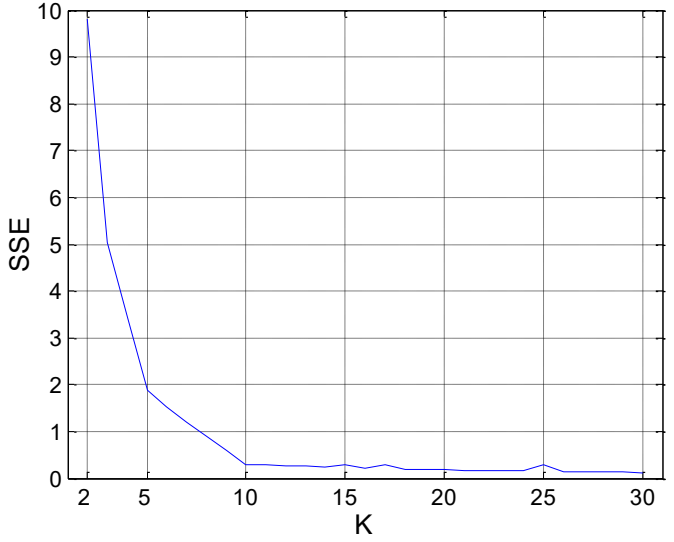


DBSCAN

Internal Measures: SSE

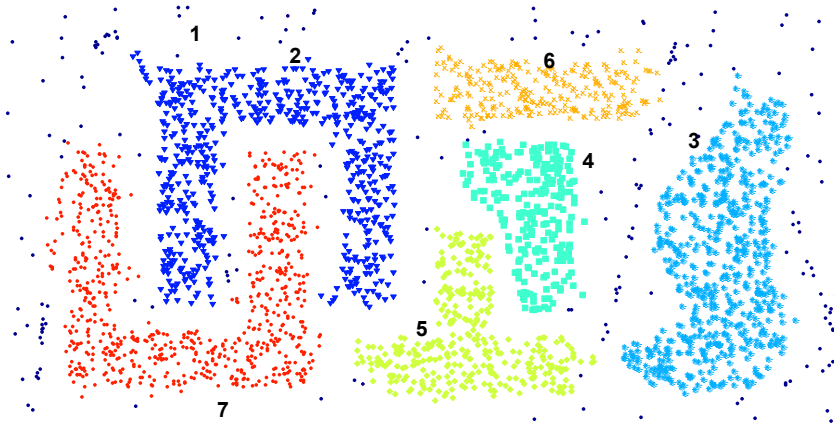
- Clusters in more complicated figures aren't well separated
- Internal Index: Used to measure the goodness of a clustering structure without respect to external information
  - SSE
- SSE is good for comparing two clusterings or two clusters (average SSE).
- Can also be used to estimate the number of clusters

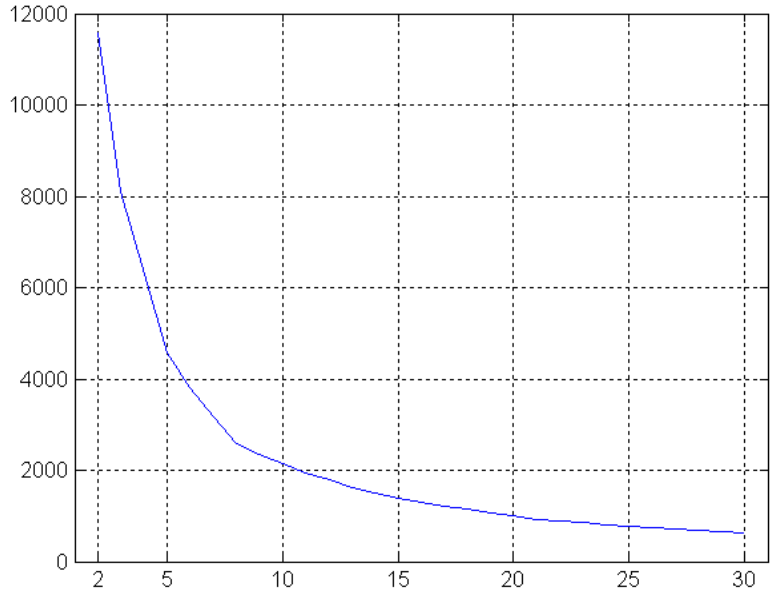




Internal Measures: SSE

- SSE curve for a more complicated data set





SSE of clusters found using K-means

Internal Measures: Cohesion and Separation

- Cluster Cohesion: Measures how closely related are objects in a cluster
  - Example: SSE
- Cluster Separation: Measure how distinct or well-separated a cluster is from other clusters
- Example: Squared Error
  - Cohesion is measured by the within cluster sum of squares (SSE)

$$SSE = WSS = \sum_i \sum_{x \in C_i} (x - m_i)^2$$

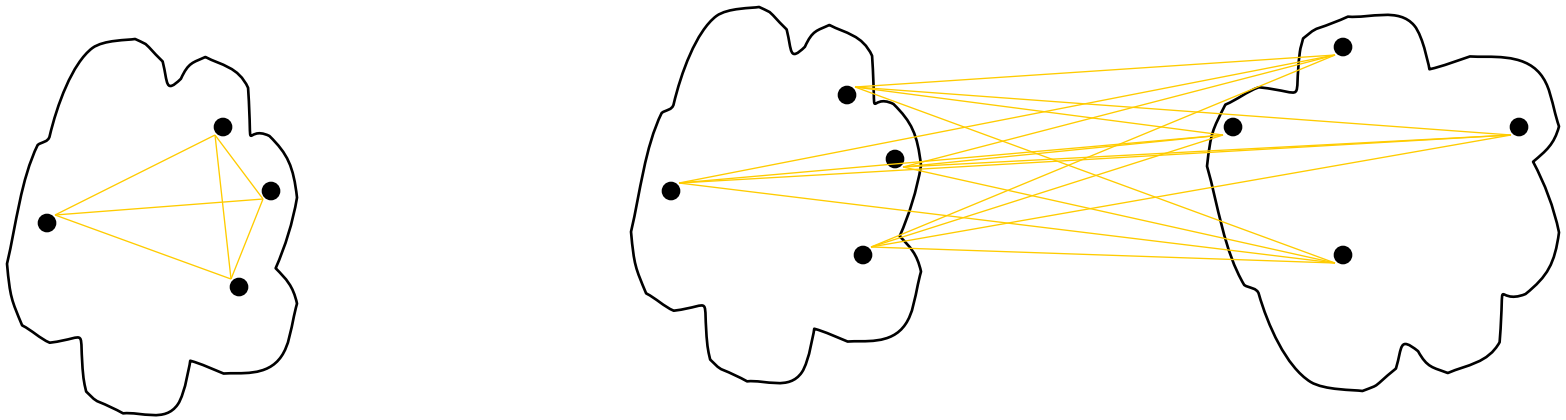
  - Separation is measured by the between cluster sum of squares

$$BSS = \sum_i |C_i| (m - m_i)^2$$

    - Where $|C_i|$ is the size of cluster i

Internal Measures: Cohesion and Separation

- A proximity graph-based approach can also be used for cohesion and separation.
  - Cluster cohesion is the sum of the weight of all links within a cluster.
  - Cluster separation is the sum of the weights between nodes in the cluster and nodes outside the cluster.

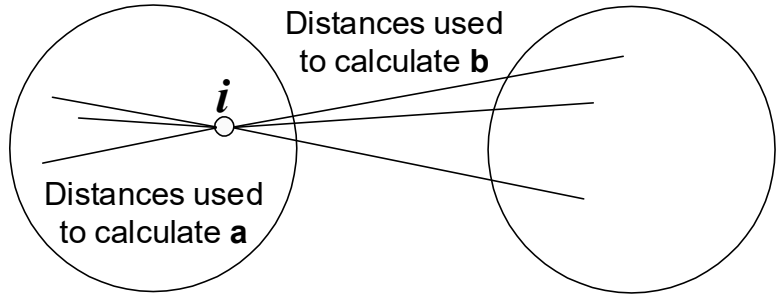


cohesion

separation

Internal Measures: Silhouette Coefficient

- Silhouette coefficient combines ideas of both cohesion and separation, but for individual points, as well as clusters and clusterings
- For an individual point, ***i***
  - Calculate ***a*** = average distance of ***i*** to the points in its cluster
  - Calculate ***b*** = min (average distance of ***i*** to points in another cluster)
  - The silhouette coefficient for a point is then given by
    s = (b – a) / max(a,b)
  - Typically between 0 and 1.
  - The closer to 1 the better.



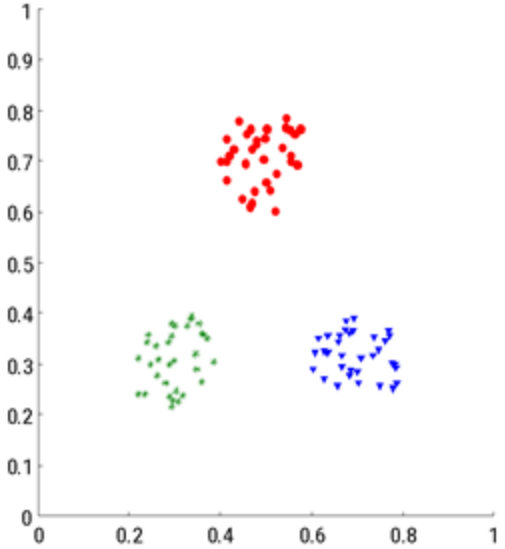
- Can calculate the average silhouette coefficient for a cluster or a clustering

Assessing the significance of Cluster Validity Measures

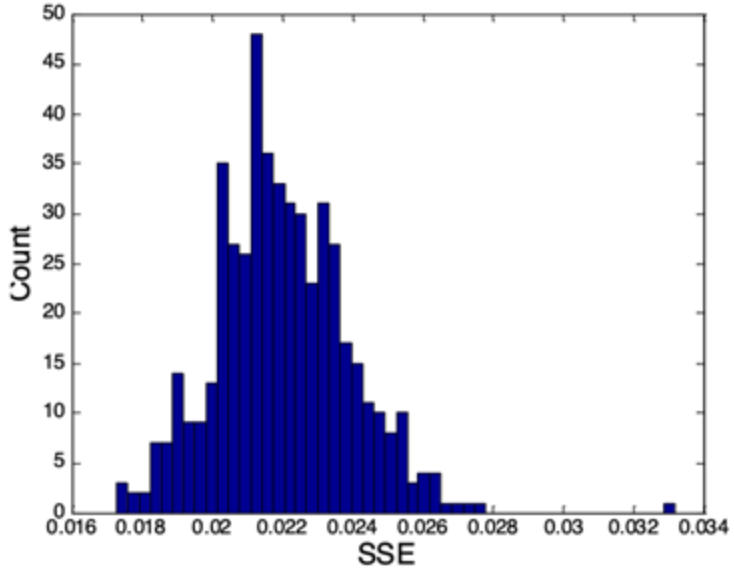
- Need a framework to interpret any measure.
  - For example, if our measure of evaluation has the value 10, is that good, fair, or poor?
- Statistics provide a framework for cluster validity
  - The more “atypical” a clustering result is, the more likely it represents valid structure in the data
  - Compare the value of an index obtained from the given data with those resulting from random data.
    - If the value of the index is unlikely, then the cluster results are valid

Statistical Framework for SSE

- Example
  - Compare SSE of three cohesive clusters against three clusters in random data



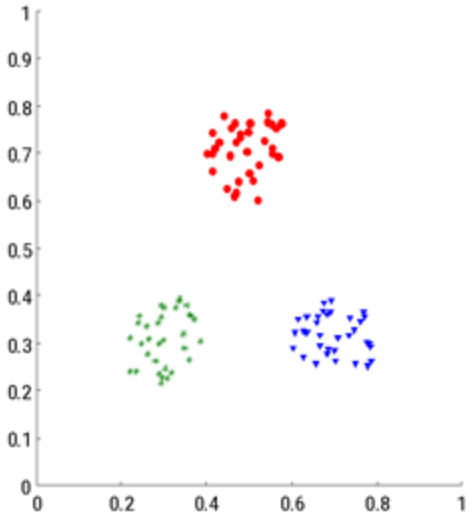
SSE = 0.005



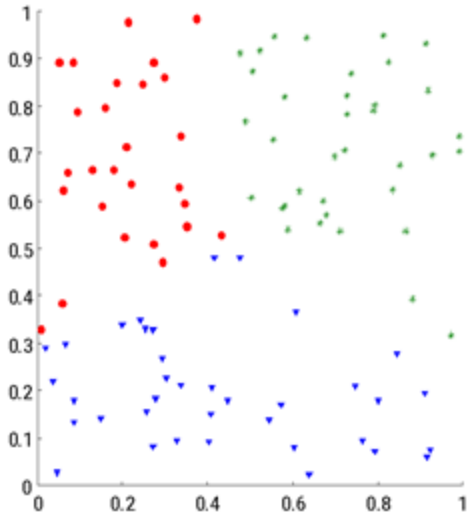
Histogram shows SSE of three clusters in 500 sets of random points of size 100 distributed over the range 0.2 - 0.8 for x and y values.

Statistical Framework for Correlation

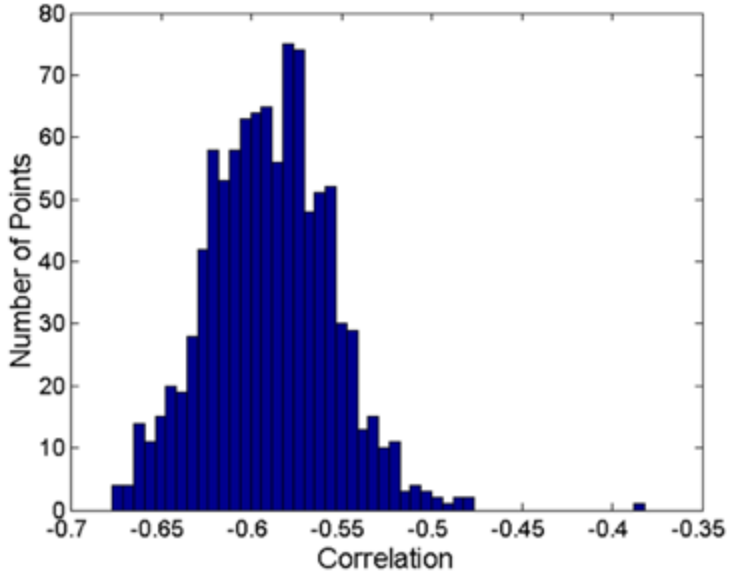
- Correlation of ideal similarity and proximity matrices for the K-means clustering of the following two datasets.



Corr = -0.9235



Corr = -0.5810



Histogram of correlation for 500 random datasets of size 100 with *x* and *y* values of points between 0.2 and 0.8.

Final Comment on Cluster Validity

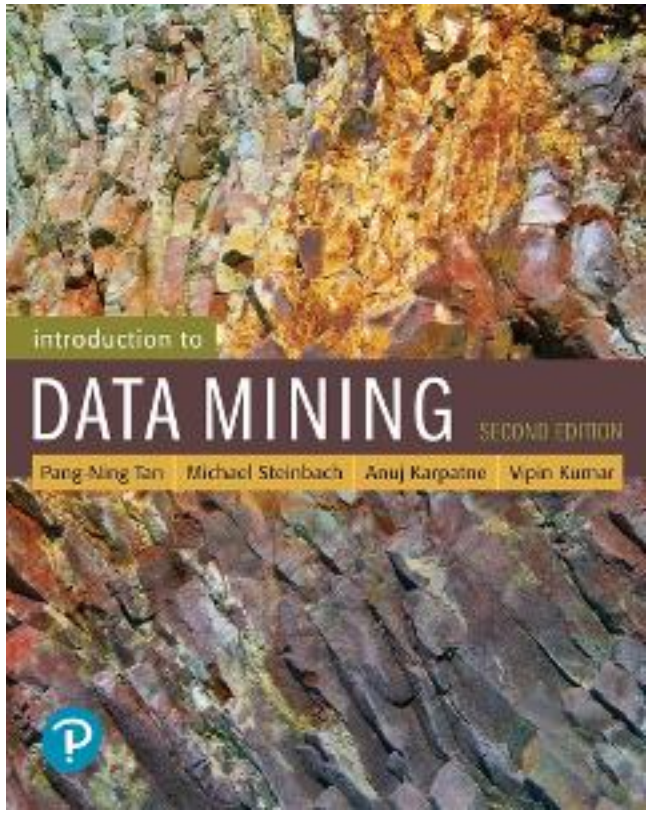
"The validation of clustering structures is the most difficult and frustrating part of cluster analysis.

Without a strong effort in this direction, cluster analysis will remain a black art accessible only to those true believers who have experience and great courage."

Algorithms for Clustering Data, Jain and Dubes

References

- Clustering. Chapter 7. Introduction to Data Mining.



DATA MINING 1
Centroid-based Clustering

Dino Pedreschi, Riccardo Guidotti

Revisited slides from Lecture Notes for Chapter 7 “Introduction to Data Mining”, 2nd Edition by Tan, Steinbach, Karpatne, Kumar

UNIVERSITÀ DI PISA

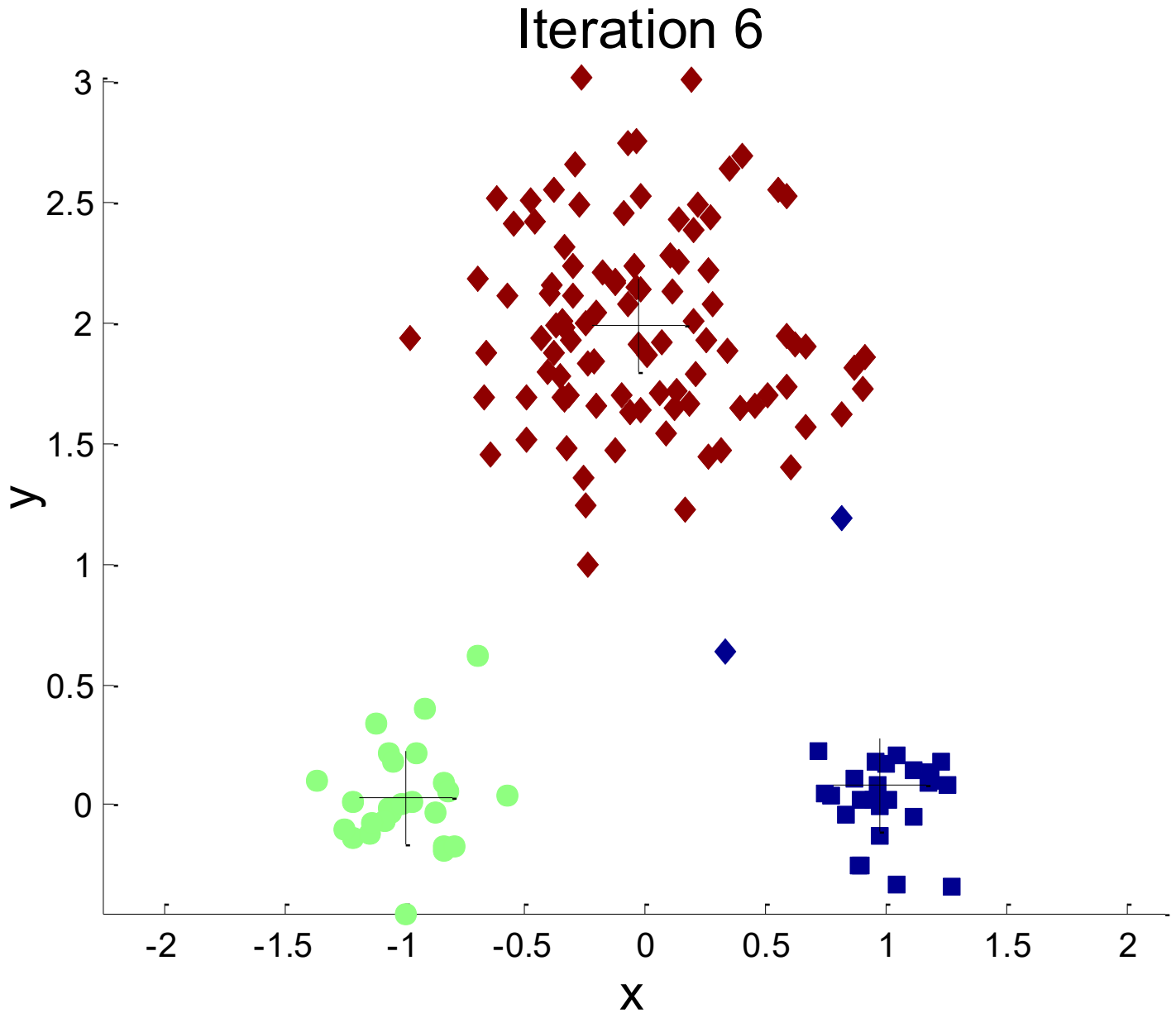
K-Means

K-Means Clustering

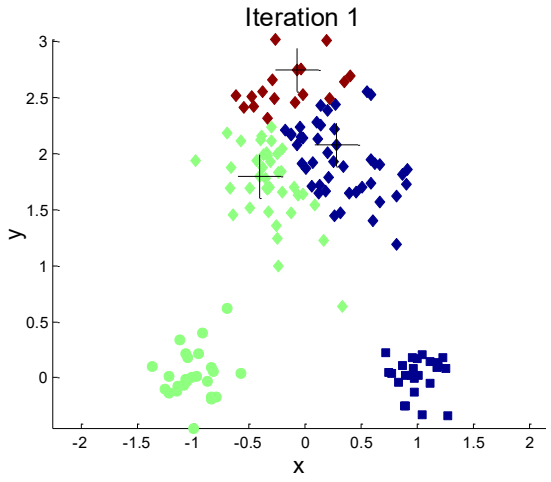
- Partitional clustering approach
- Number of clusters, K, must be specified
- Each cluster is associated with a **centroid** (center point)
- Each point is assigned to the cluster with the **closest centroid**
- The basic algorithm is very simple

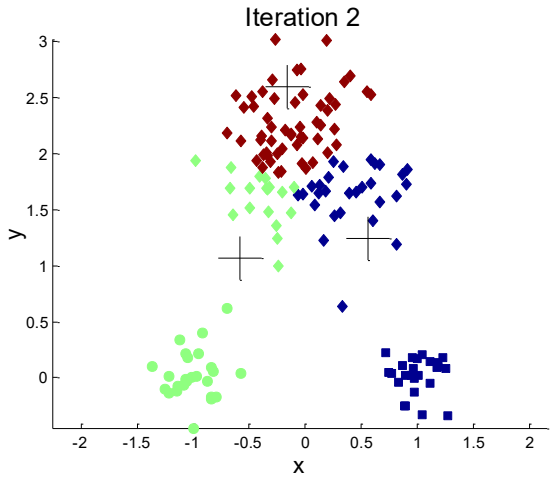
1: Select K points as the initial centroids.
2: **repeat**
3: Form K clusters by assigning all points to the closest centroid.
4: Recompute the centroid of each cluster.
5: **until** The centroids don't change

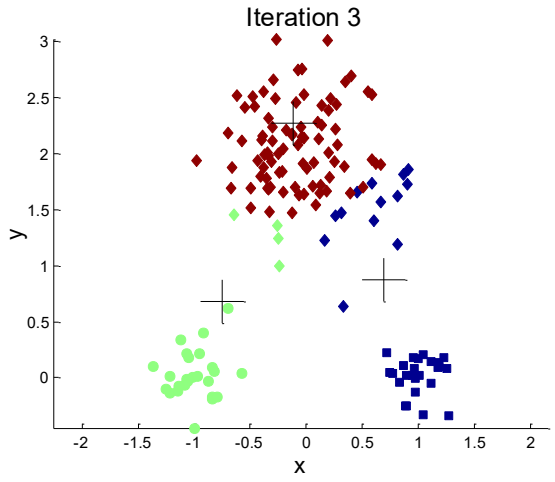
Example of K-Means Clustering

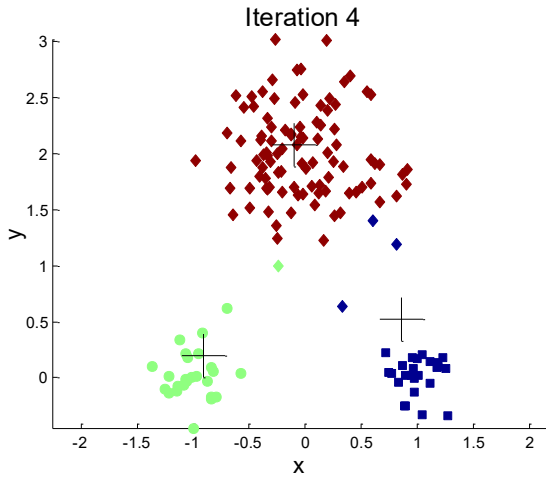


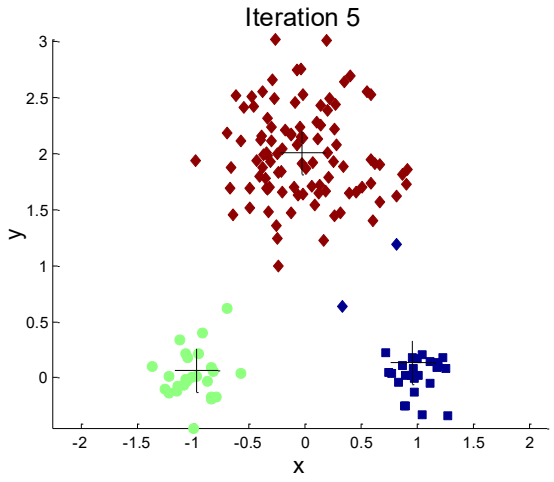
Example of K-Means Clustering

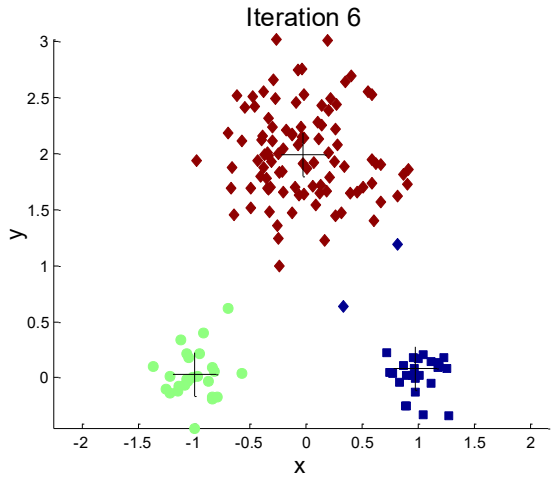








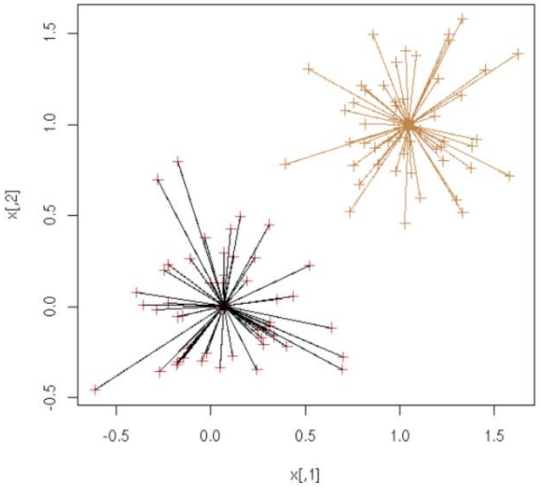




K-Means Clustering – Details

- Initial centroids are often chosen randomly.
  - Clusters produced vary from one run to another.
- The centroid is (typically) the mean of the points in the cluster.
- ‘Closeness’ is measured by Euclidean distance, cosine similarity, correlation, etc.
- K-means will converge for common similarity measures mentioned above.
- Most of the convergence happens in the first few iterations.
  - Often the stopping condition is changed to ‘Until relatively few points change clusters’
- Complexity is O(n * K * I * d)
  - n = number of points, K = number of clusters,
    I = number of iterations, d = number of attributes

Evaluating K-Means Clusters

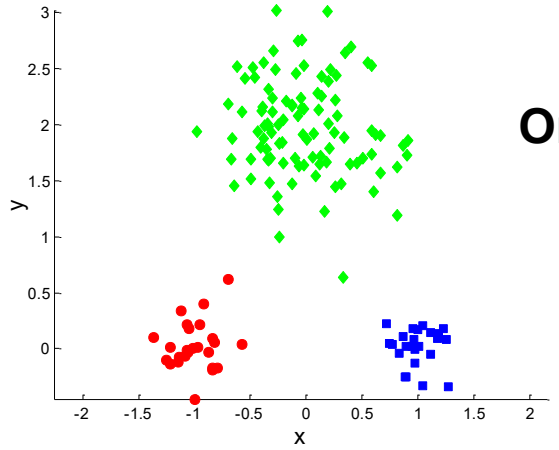


- Most common measure is Sum of Squared Error (SSE)
  - For each point, the error is the distance to the nearest cluster
  - To get SSE, we square these errors and sum them.

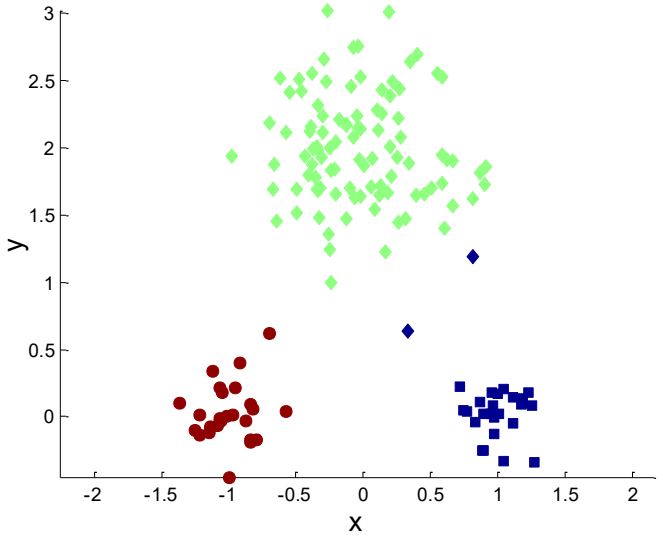
$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} dist^2(m_i, x)$$

- x is a data point in cluster C_i and m_i is the representative point for cluster Ci
  - can show that mi corresponds to the center (mean) of the cluster
- Given two sets of clusters, we prefer the one with the smallest error
- One easy way to reduce SSE is to increase K, the number of clusters
- A good clustering with smaller K can have a lower SSE than a poor clustering with higher K

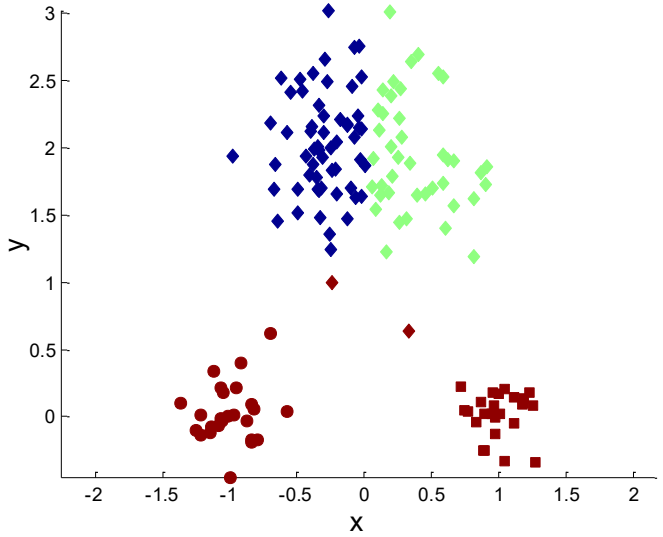
Two different K-Means Clusterings



Original Points



Optimal Clustering



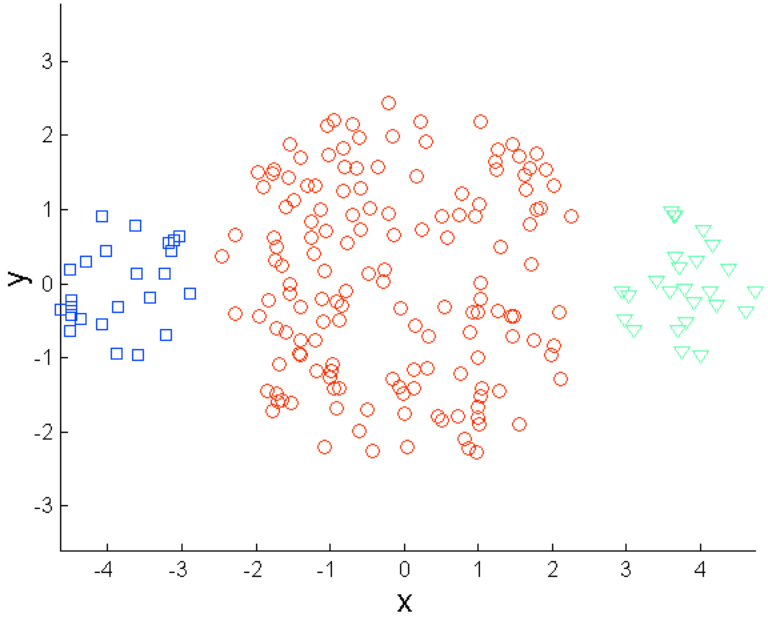
Sub-optimal Clustering

Limitations of K-Means

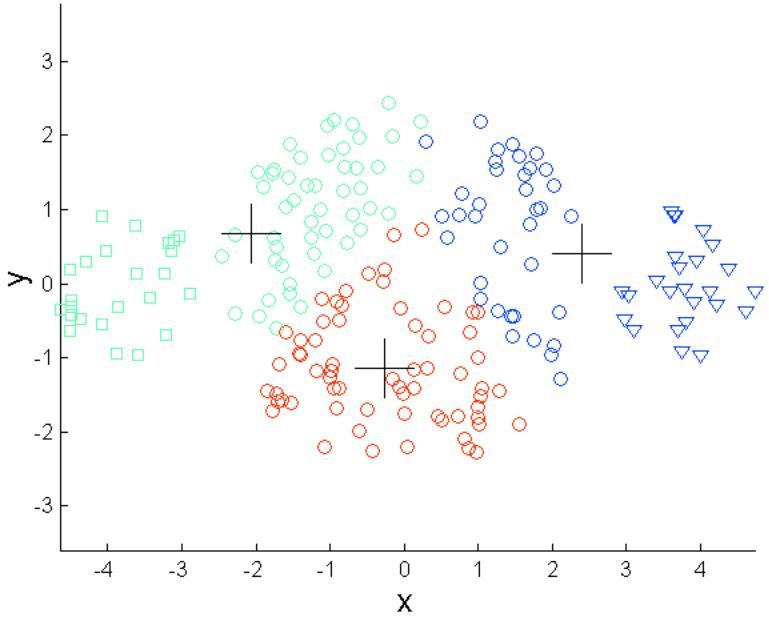
- K-Means has problems when clusters are of differing
  - Sizes
  - Densities
  - Non-globular shapes

- K-Means has problems when the data contains outliers.

Limitations of K-Means: Differing Sizes

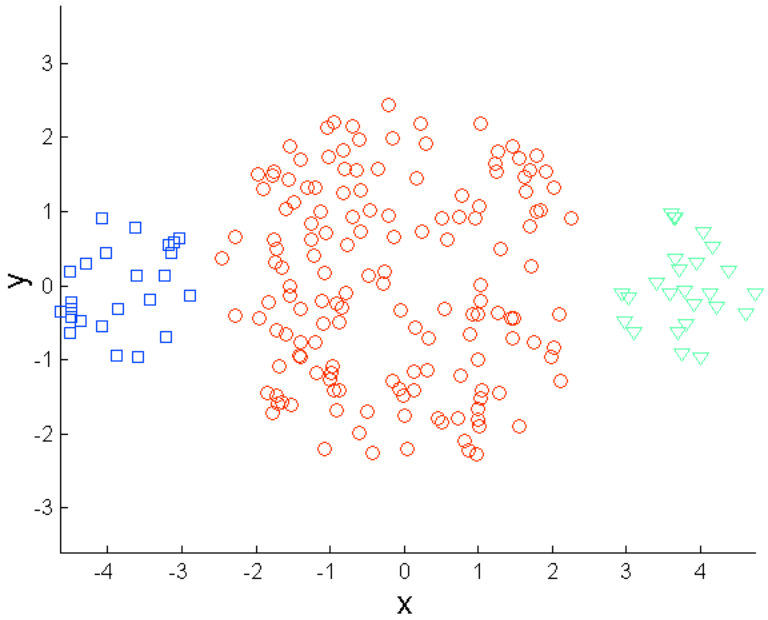


Original Points

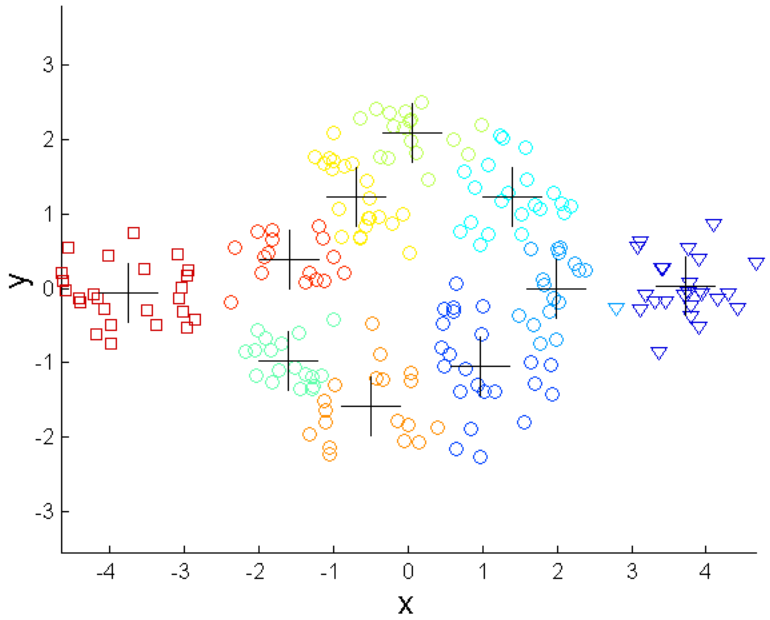


K-means (3 Clusters)

Overcoming K-Means Limitations



Original Points

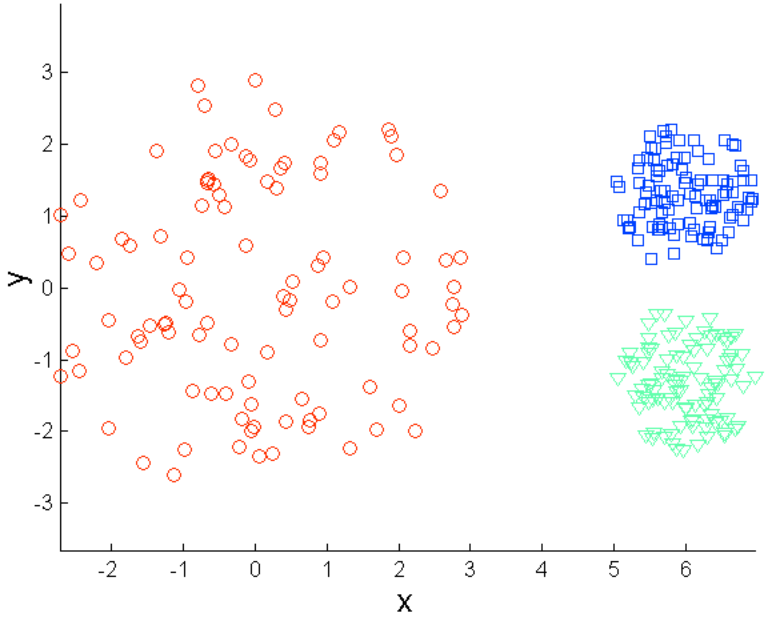


K-means Clusters

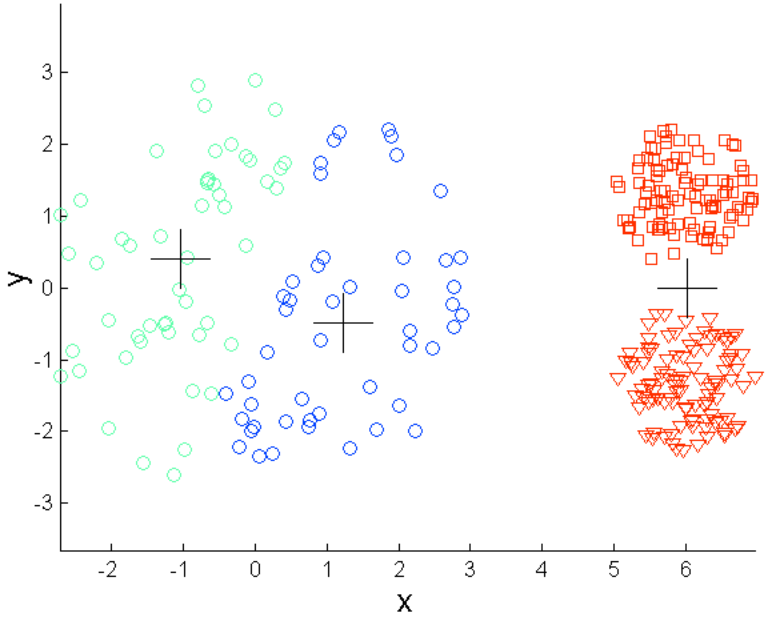
One solution is to use many clusters.

Find parts of clusters, but need to put together.

Limitations of K-Means: Differing Density

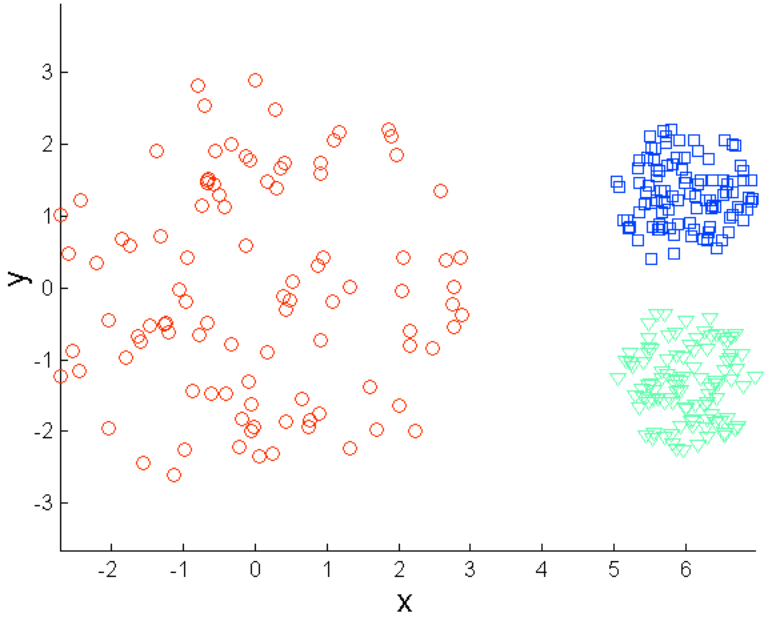


Original Points

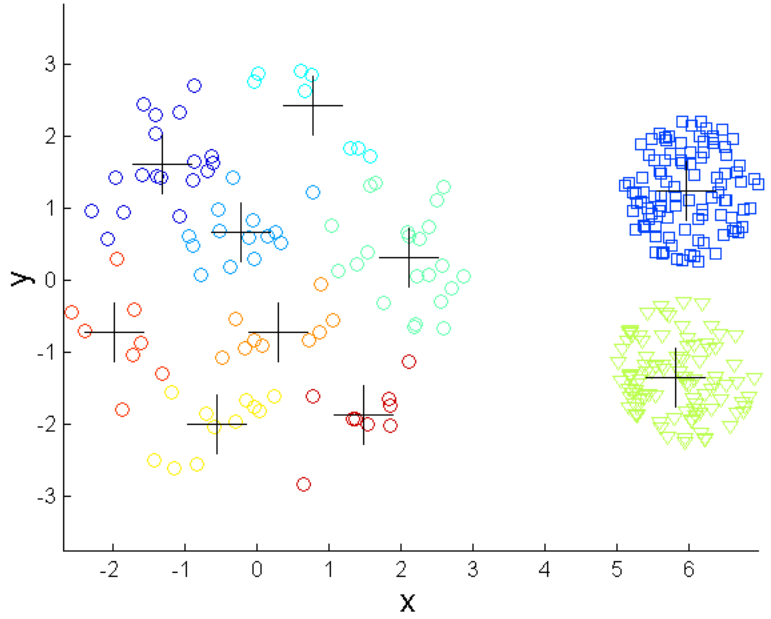


K-means (3 Clusters)

Overcoming K-Means Limitations

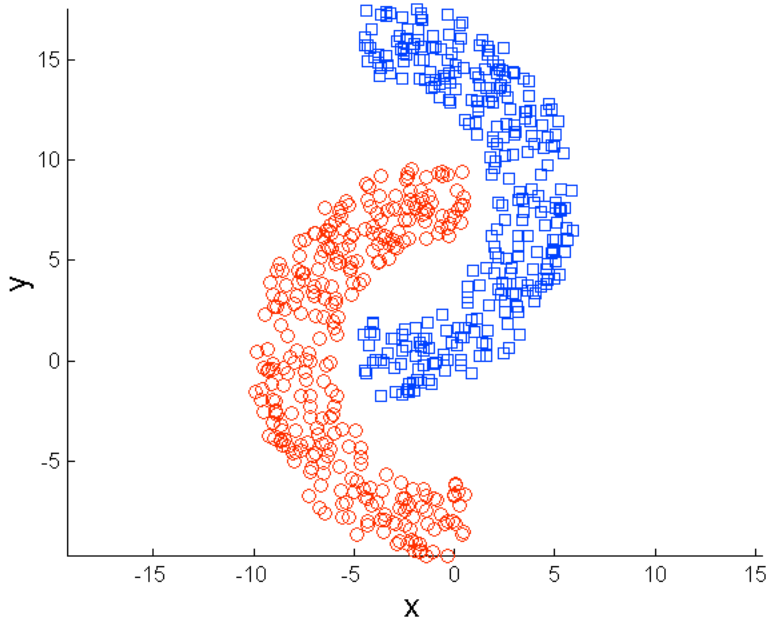


Original Points

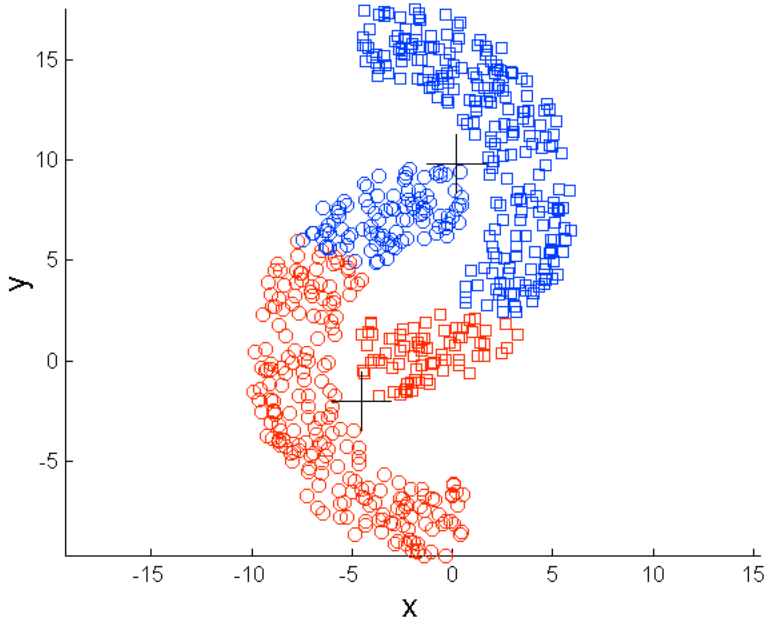


K-means Clusters

Limitations of K-Means: Non-globular Shapes

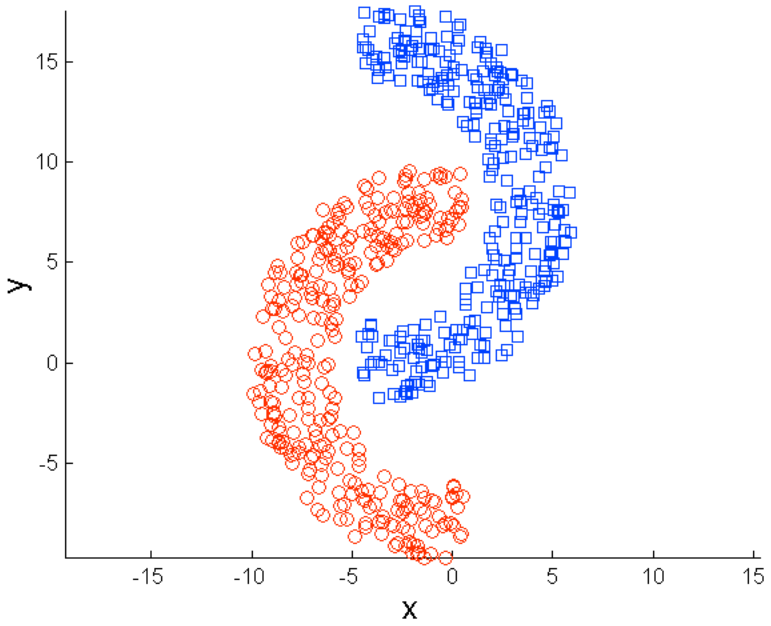


Original Points

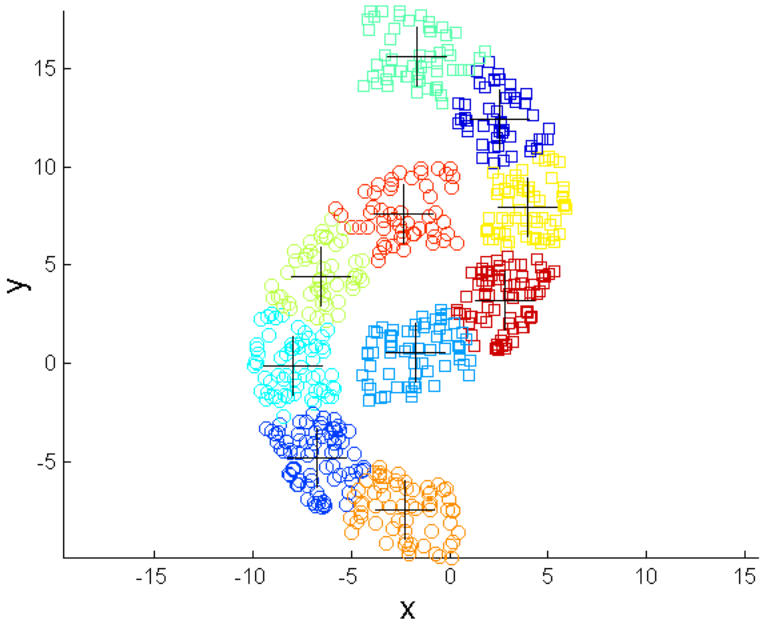


K-means (2 Clusters)

Overcoming K-Means Limitations



Original Points

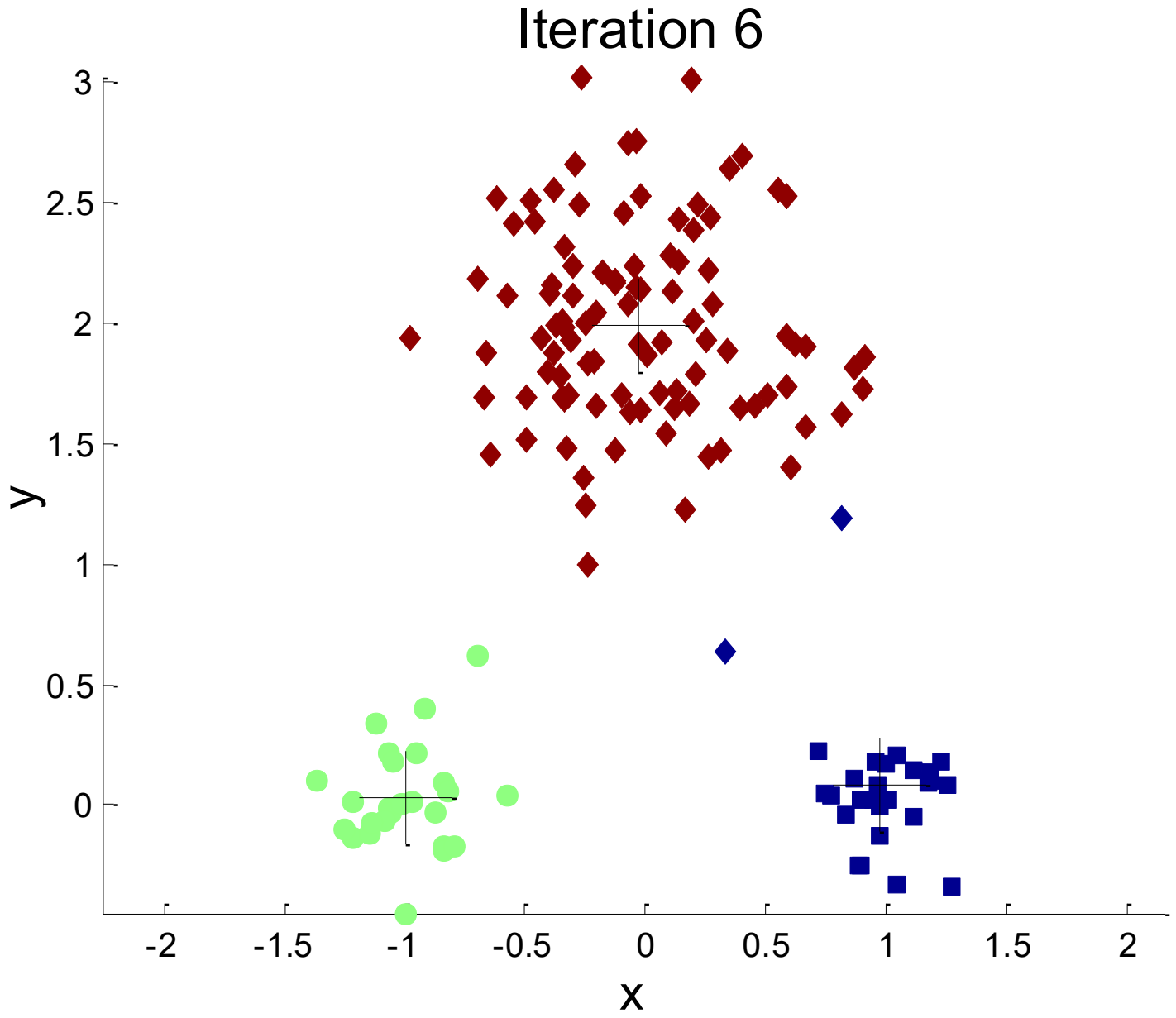


K-means Clusters

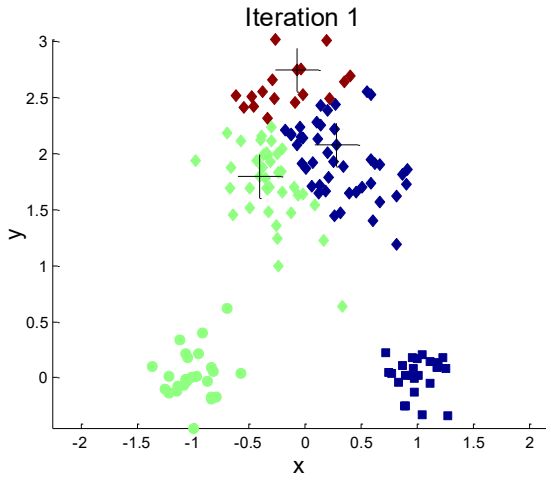
Pre-processing and Post-processing

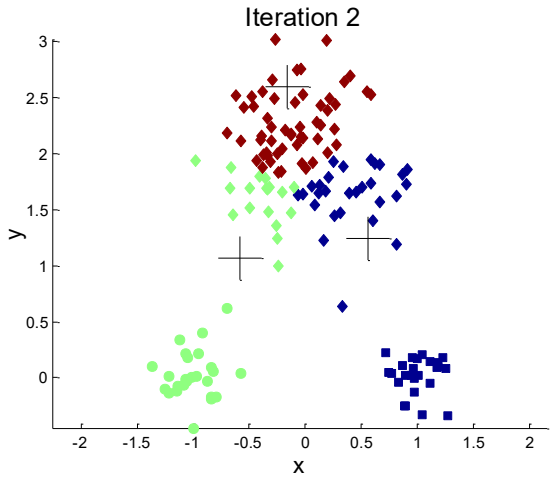
- Pre-processing
  - Normalize the data
  - Eliminate outliers
- Post-processing
  - Eliminate small clusters that may represent outliers
  - Split ‘loose’ clusters, i.e., clusters with relatively high SSE
  - Merge clusters that are ‘close’ and that have relatively low SSE
  - Can use these steps during the clustering process
    - ISODATA

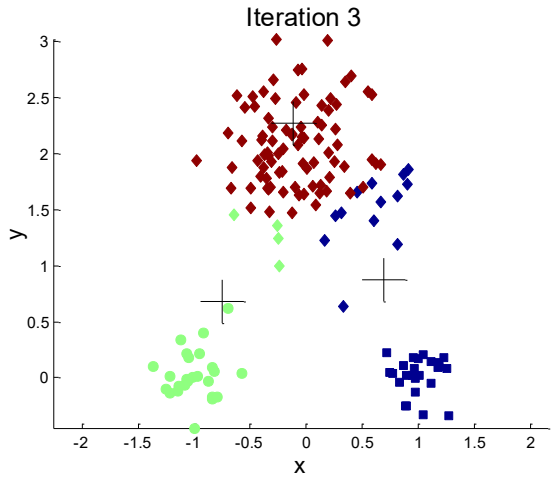
Importance of Choosing Initial Centroids

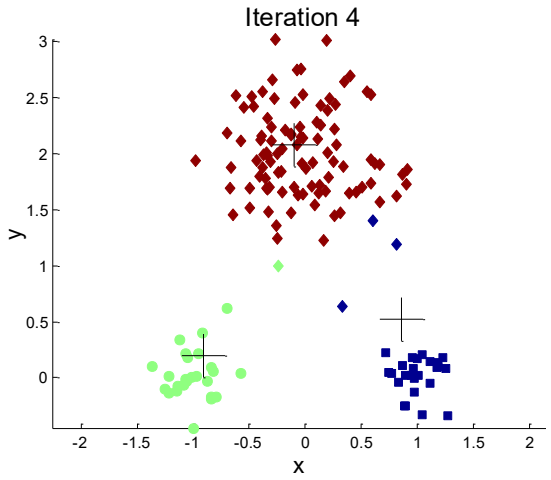


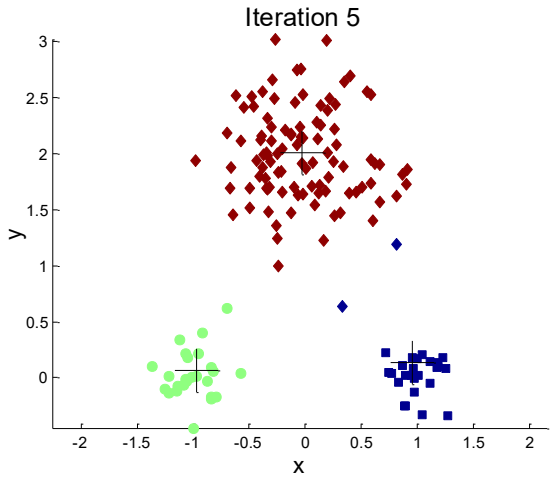
Importance of Choosing Initial Centroids

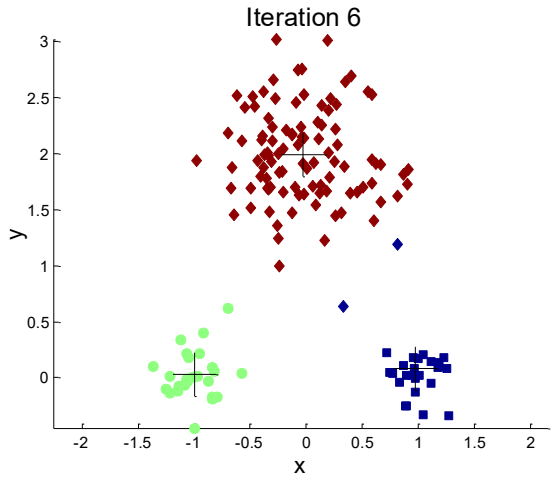




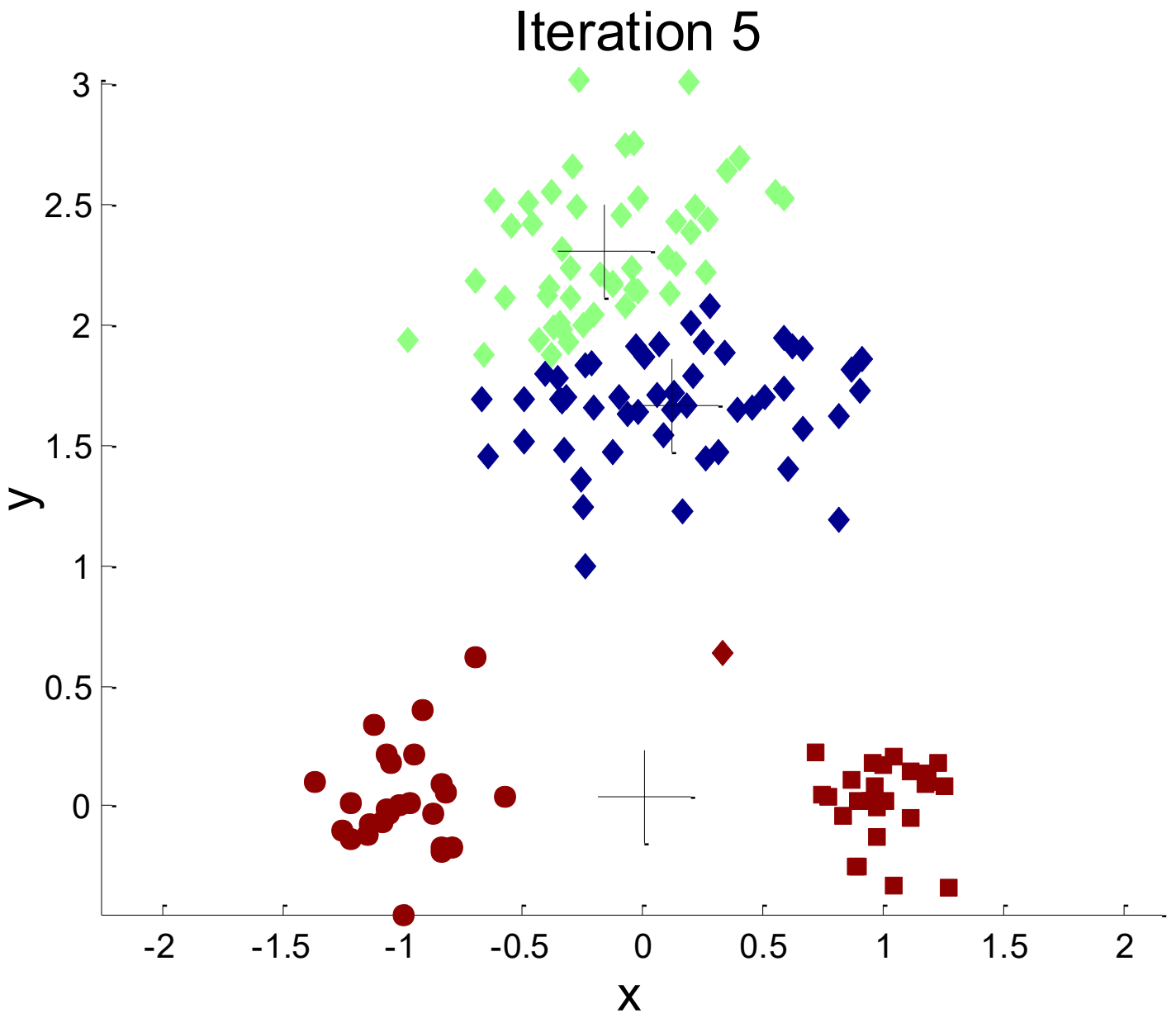




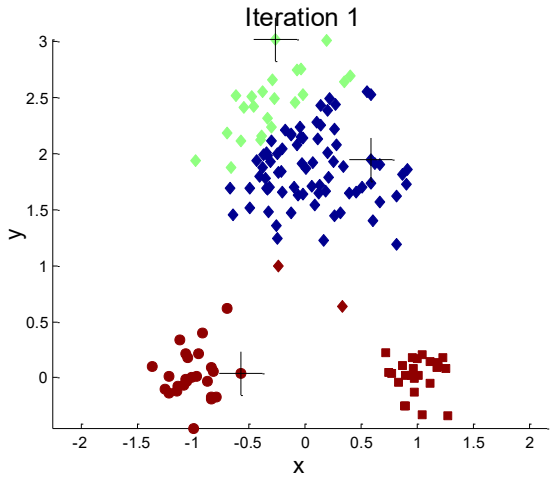


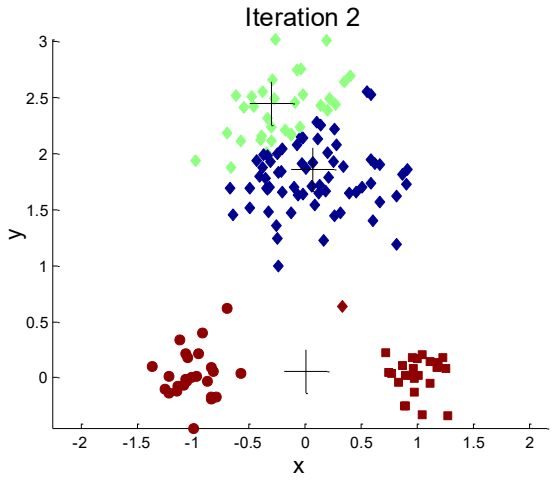


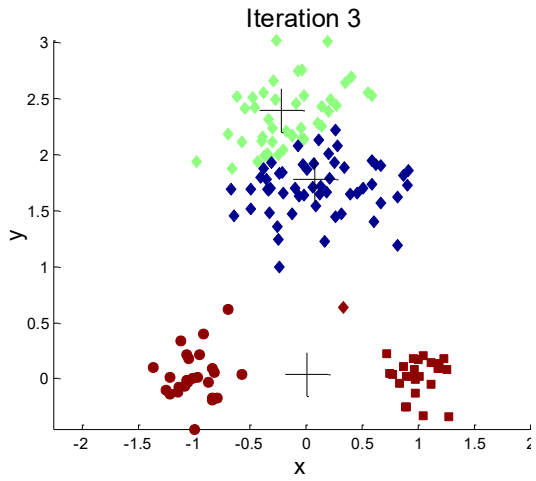
Importance of Choosing Initial Centroids …

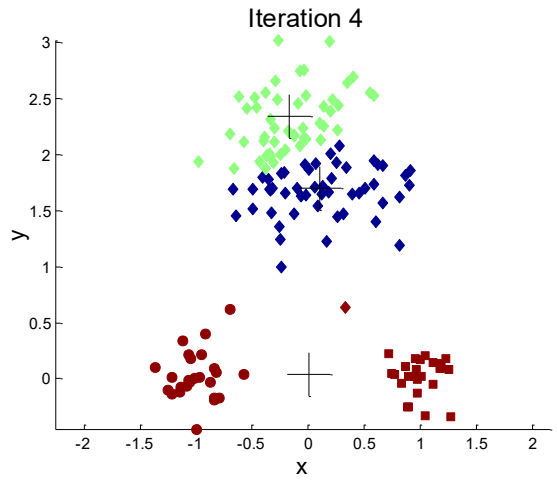


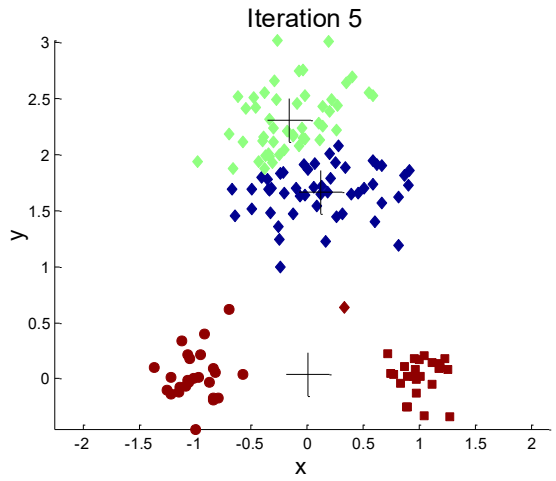
Importance of Choosing Initial Centroids …



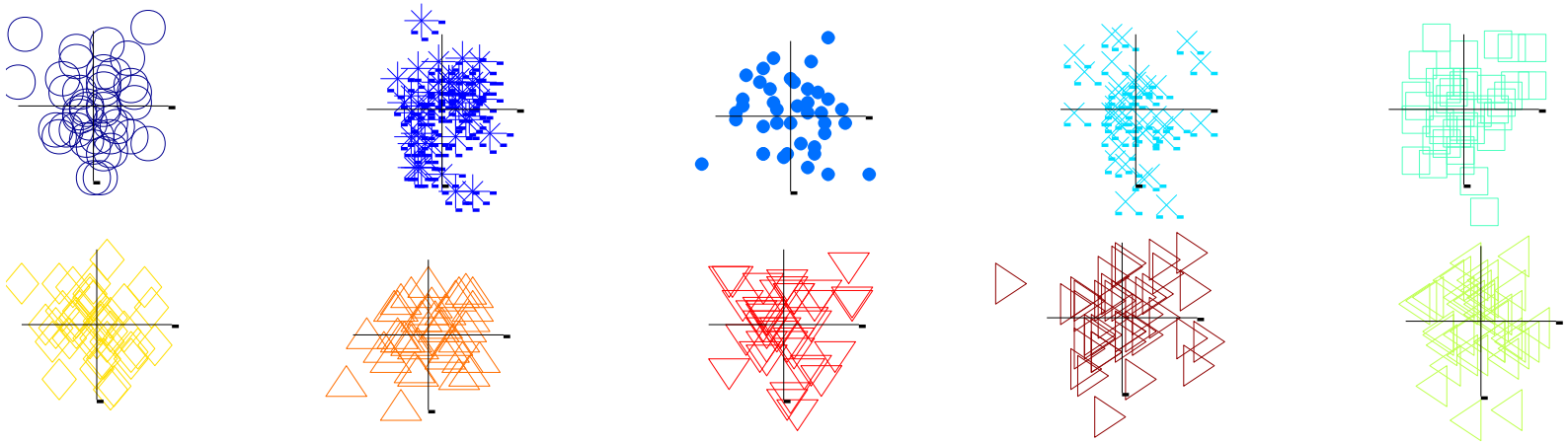






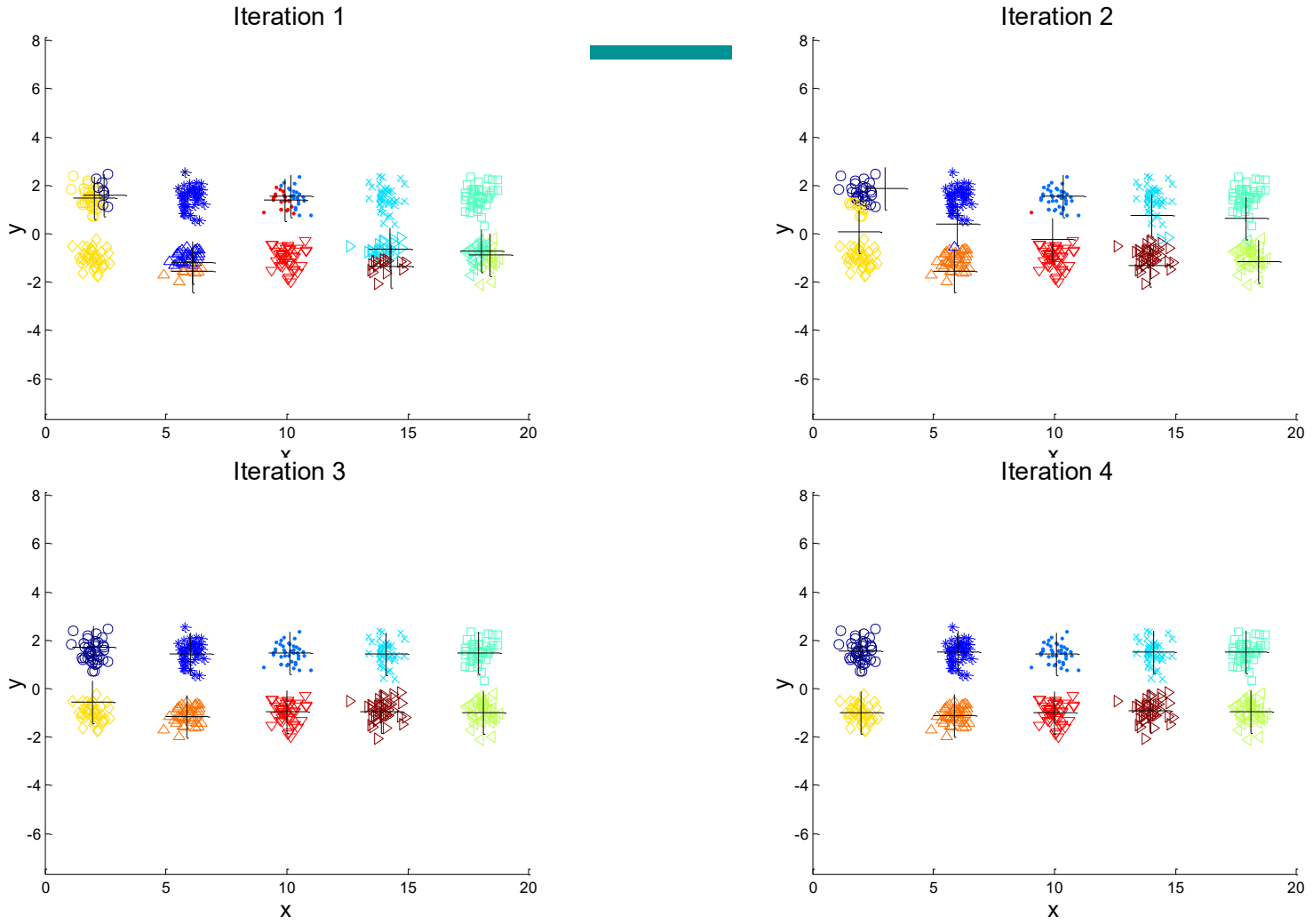


10 Clusters Example



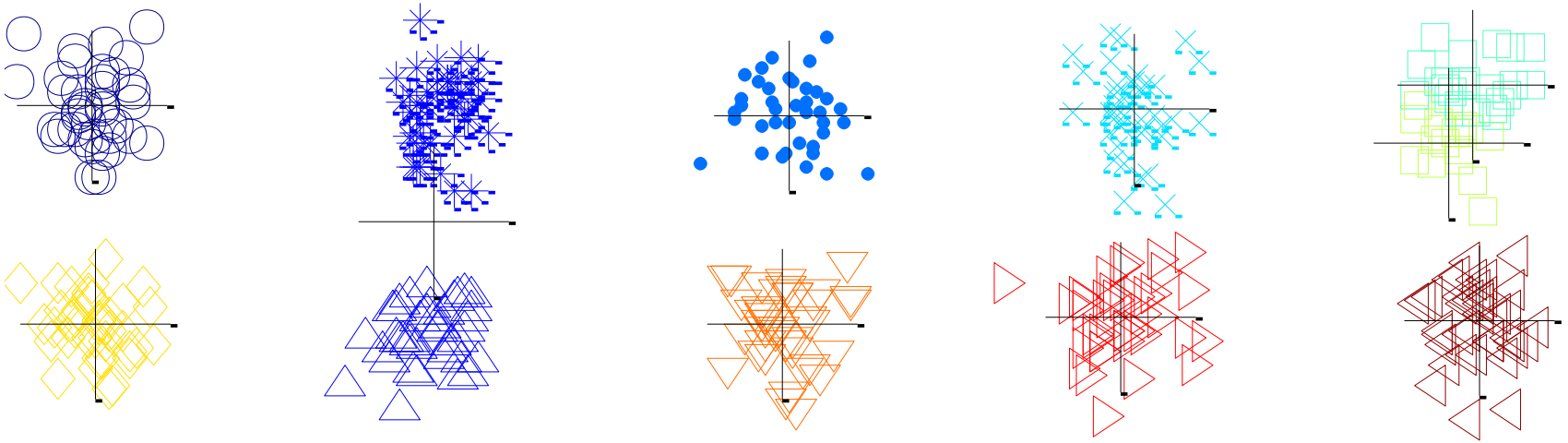
Starting with two initial centroids in one cluster of each pair of clusters

10 Clusters Example



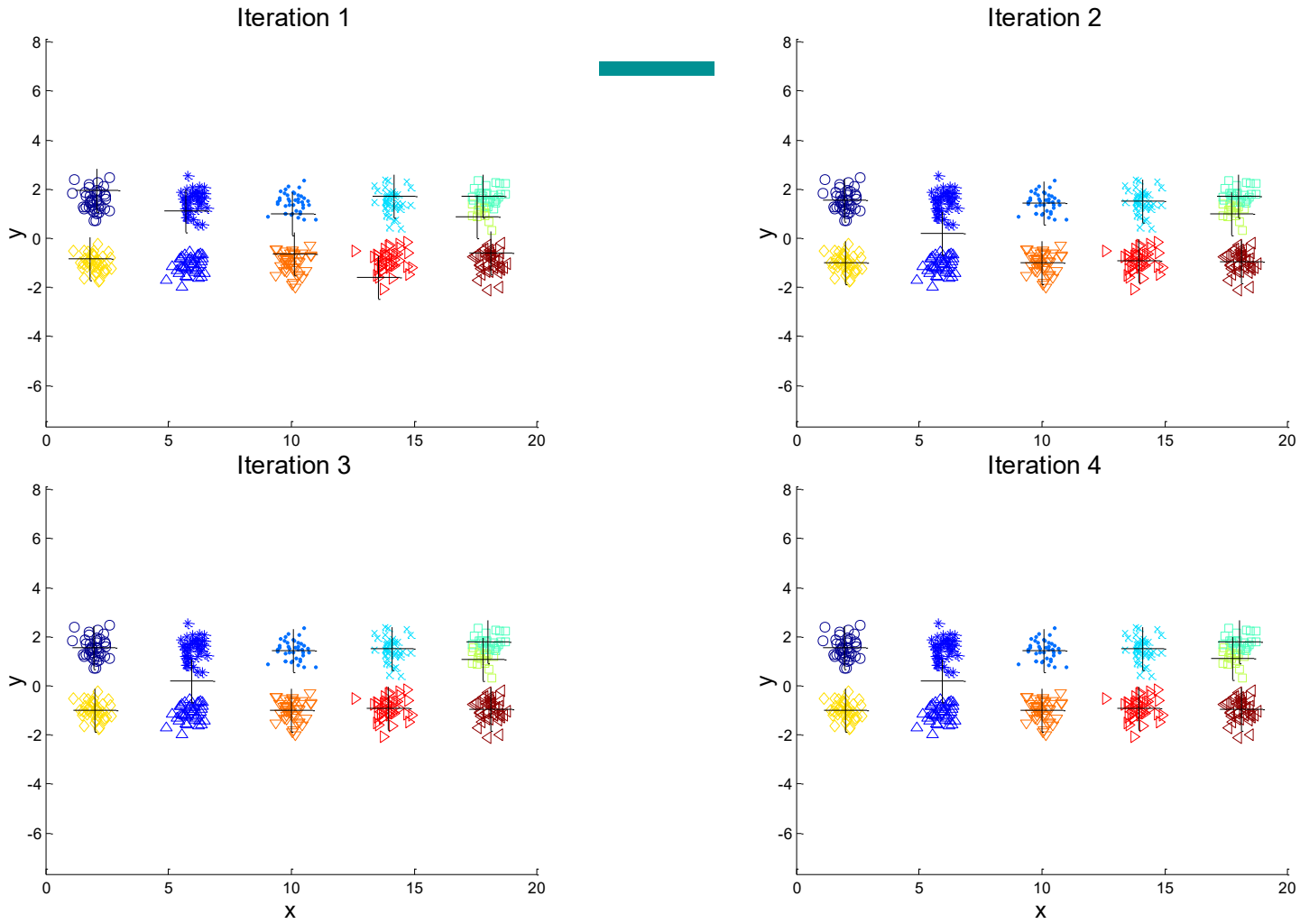
Starting with two initial centroids in one cluster of each pair of clusters

10 Clusters Example



Starting with some pairs of clusters having three initial centroids, while other have only one.

10 Clusters Example



Starting with some pairs of clusters having three initial centroids, while other have only one.

Solutions to Initial Centroids Problem

- Multiple runs
  - Helps, but probability is not on your side
- **Sample and use hierarchical clustering to determine initial centroids**
- Select more than k initial centroids and then select among these initial centroids
  - Select most widely separated
- Postprocessing
- Generate a larger number of clusters and then perform a hierarchical clustering
- Bisecting K-means
  - Not as susceptible to initialization issues

K-Means Extensions

Bisecting K-Means

Bisecting K-means

- Variant of K-Means that can produce a hierarchical clustering
- The number of clusters K must be specified.
- Start with a unique cluster containing all the points.

1: Initialize the list of clusters to contain the cluster containing all points.
2: **repeat**
3: Select the cluster with the highest SSE to the list of clusters
4: **for** $i = 1$ to *number_of_iterations* **do**
5: Bisect the selected cluster using basic 2-Means
6: **end for**
7: Add the two clusters from the bisection to the list of clusters.
8: **until** Until the list of clusters contains K clusters

Bisecting K-means Limitations

- The algorithm can be also exhaustive and terminating at a singleton clusters if K is not specified.
- Terminating at singleton clusters
  - Is time consuming
  - Singleton clusters are meaningless (i.e., over-splitting)
  - Intermediate clusters are more likely to correspond to real classes

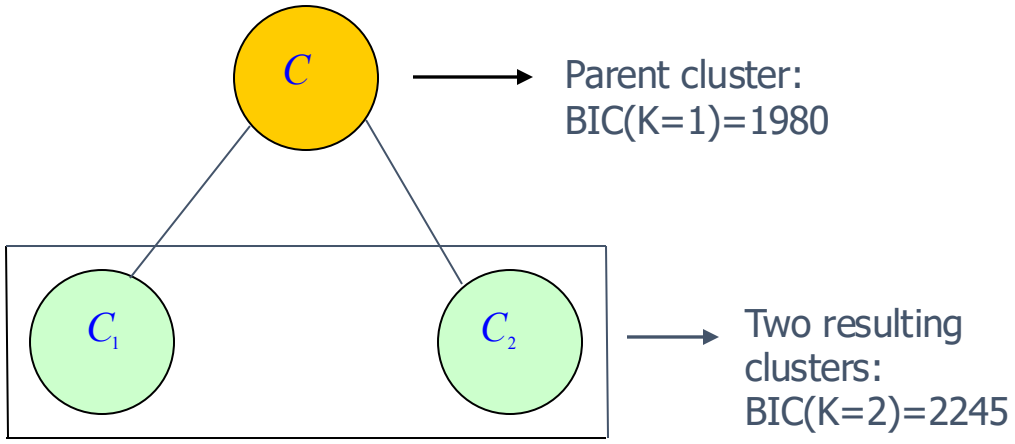
- Bisecting K-Means do not use any criterion for stopping bisections before singleton clusters are reached.

K-Means Extensions

X-Means

Bayesian Information Criterion (BIC)

- A strategy to stop the Bisecting algorithm when meaningful clusters are reached to avoid over-splitting.
- The **BIC** can be adopted as **splitting criterion** of a cluster in order to decide whether a cluster should split or no.
- BIC **measures the improvement** of the cluster structure between a cluster and its two children clusters.
- If the BIC of the parent is less than BIC of the children than we accept the bisection.



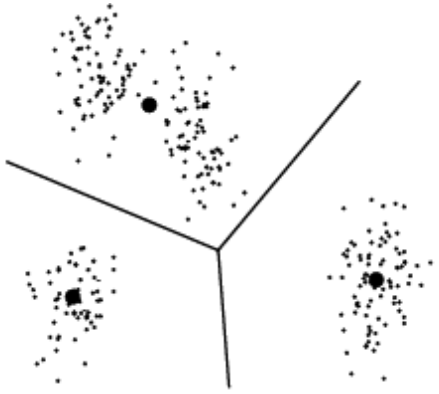
X-Means

For k in a given range $[r_1, r_{max}]$:

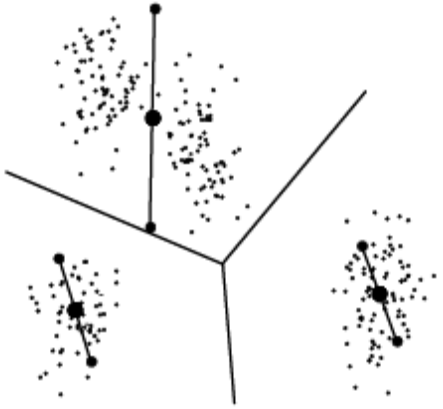
1. Improve Params: run K-Means with with the current k.
2. Improve Structure: recursively split each cluster in two (Bisecting 2-Means) and use *local BIC* to decide to keep the split. Stop if the current structure does not respect *local BIC* or the number of clusters is higher than r_{max}.
3. Store the actual configuration with a global BIC calculated on the whole configuration
4. If $k > r_{max}$ stop and return the best model w.r.t. the *global BIC*.

X-Means

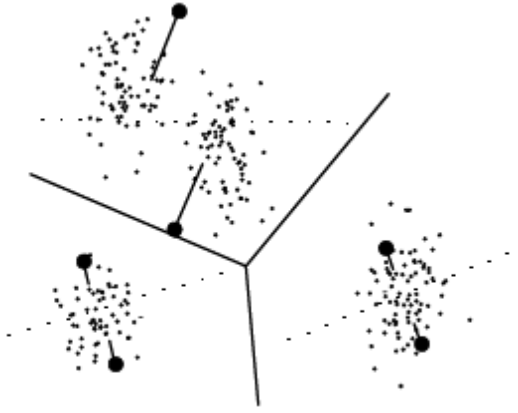
1. K-means with k=3



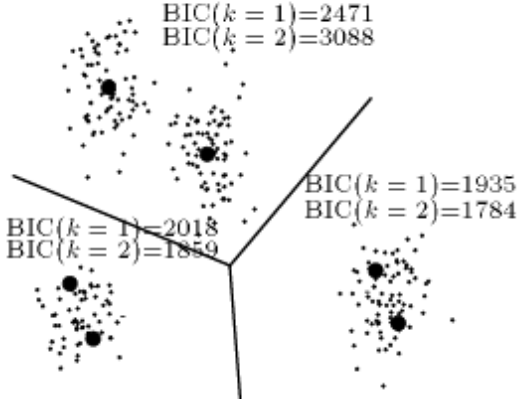
2. Split each centroid in 2 children moved a distance proportional to the region size in opposite direction (random)



3. Run 2-means in each region locally



4. Compare BIC of parent and children



4. Only centroids with higher BIC survives

BIC Formula in X-Means

- The BIC score of a data collection is defined as (Kass and Wasserman, 1995):

$$BIC(M_j) = \hat{l}_j(D) - \frac{p_j}{2}\log R$$

- $\hat{l}_j(D)$ is the log-likelihood of the dataset D
- p_j is a function of the number of independent parameters: centroids coordinates, variance estimation.
- R is the number of points of a cluster, M is the number of dimensions
- Approximate the probability that the clustering in M_j is describing the real clusters in the data

BIC Formula in X-Means

- Adjusted Log-likelihood of the model.
- **The likelihood that the data is "explained by" the clusters** according to the spherical-Gaussian assumption of K-Means

$$BIC(M_j) = \hat{l}_j(D) - \frac{p_j}{2}\log R$$

- Focusing on the set D_n of points which belong to centroid *n*

$$\begin{aligned}\hat{l}(D_n) \quad = \quad & -\frac{R_n}{2}\log(2\pi) - \frac{R_n \cdot M}{2}\log(\hat{\sigma}^2) - \frac{R_n - K}{2} \\ & + R_n \log R_n - R_n \log R\end{aligned}$$

- It estimates how closely to the centroid are the points of the cluster.

K-Means Origins

Expectation Maximization

Model-based Clustering (probabilistic)

- In order to understand our data, we will assume that there is a generative process (a model) that creates/describes the data, and we will try to find the model that **best fits** the data.
  - Models of different complexity can be defined, but we will assume that our model is a distribution from which data points are sampled
  - **Example**: the data is the height of all people in **Greece**
- In most cases, a single distribution is not good enough to describe all data points**: different parts of the data follow a different distribution**
  - **Example**: the data is the height of all people in Greece and China
  - We need a mixture model
  - Different distributions correspond to different clusters in the data.

Expectation Maximization Algorithm

- Initialize the values of the parameters in Θ to some random values
- Repeat until convergence
  - **E-Step**: Given the parameters Θ **estimate** the membership probabilities $\mathrm{P}(G_j|x_i)$
  - **M-Step:** Given the probabilities $\mathrm{P}(G_j|x_i)$, calculate the parameter values Θ that (in expectation) **maximize** the data likelihood

- **Examples**
  - **E-Step**: Assignment of points to clusters
    - K-Means: **hard** assignment, EM: soft assignment
  - **M-Step**: Parameters estimation
    - K-Means: Computation of centroids, EM: Computation of the new model parameters

EM in K-Means

centroids

- Initialize the values of the parameters in Θ to some random values (randomly select the centroids)
- Repeat until convergence
  - **E-Step**: Given the parameters Θ (given the centroids) **estimate** the membership probabilities $\mathrm{P}(G_j|x_i)$ (assign points to clusters based on distances with the centroids)
  - **M-Step:** Given the probabilities $\mathrm{P}(G_j|x_i)$ (given the membership of points to clusters, i.e., 100% probability of belonging to a cluster) calculate the parameter values Θ that (in expectation) **maximize** the data likelihood (calculate the new centroids as mean values, i.e., those that minimize the distances with the other points in the cluster)

Expectation Maximization Algorithm

Algorithm 9.2 EM algorithm.

1: Select an initial set of model parameters.
(As with K-means, this can be done randomly or in a variety of ways.)
2: **repeat**
3: **Expectation Step** For each object, calculate the probability that each object belongs to each distribution, i.e., calculate $prob(distribution\ j|\mathbf{x}_i, \Theta)$.
4: **Maximization Step** Given the probabilities from the expectation step, find the new estimates of the parameters that maximize the expected likelihood.
5: **until** The parameters do not change.
(Alternatively, stop if the change in the parameters is below a specified threshold.)

K-Means Brother

K-Modes

K-Modes

$$\text{Minimise} \quad P(W, \boldsymbol{Q}) = \sum_{l=1}^{k} \sum_{i=1}^{n} w_{i,l}\, d(X_i, Q_l)$$

$$\text{subject to} \quad \sum_{l=1}^{k} w_{i,l} = 1, \quad 1 \le i \le n$$

$$w_{i,l} \in \{0, 1\}, \quad 1 \le i \le n,\ 1 \le l \le k$$

- $X = \{ X_1, \ldots, X_n \}$ is the dataset of objects.
- $X_i = [x_1, \ldots, x_m]$ is an object i.e., a vector of m categorical attributes
- W is a matrix $n \times k$, with $w_{i,l}$ equal to 1 if X_i belongs to Cluster l, 0 otherwise.
- $Q = \{ Q_1, \ldots, Q_k \}$ is the set of representative objects (mode) for the k clusters.
- $d(X_i , Q_l)$ is a distance function for objects in the data

K-Modes: Distance

- K-Means as distance uses Euclidean distance

$$d(X,Y) = \sum_{i=1}^{m} (x_i - y_i)^2$$

- K-Modes as distance uses the number of mismatches between the attributes of two objects.

$$d_1(X, Y) = \sum_{j=1}^{m} \delta(x_j, y_j)$$

$$\delta(x_j, y_j) = \begin{cases} 0 & (x_j = y_j) \\ 1 & (x_j \neq y_j) \end{cases}$$

K-Modes: Mode

- K-Modes uses the mode as representative object of a cluster
- Given the set of objects in the cluster C_l the mode is get computing the max frequency for each attribute

$$f_r(A_j = c_{l,j} \mid X_l) = \frac{n_{c_{l,k}}}{n}$$

K-Modes: Algorithm

1. Randomly select the initial objects as modes
2. Scan of the data to assign each object to the closer cluster identified by the mode
3. Re-compute the mode of each cluster
4. Repeat the steps 2 and 3 until no object changes the assigned cluster

K-Means Brother

Mixture Gaussian Model

Gaussian Distribution

- Example: the data is the height of all people in Greece
- Experience has shown that this data follows a Gaussian (Normal) distribution

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- μ = mean, σ = standard deviation

Mixture Gaussian Model

- What is a model?
  - A Gaussian distribution is defined by the mean μ and the standard deviation σ
  - We define our model as the pair of parameters $\theta = (\mu, \sigma)$
- More generally, a model is defined as a vector of parameters θ

- We want to find the normal distribution $N(\mu, \sigma)$ that best fits our data
  - Find the best values for μ and σ
  - But what does "best fit" mean?

Maximum Likelihood Estimation (MLE)

- Suppose that we have a vector $X = \{x_1, \dots, x_n\}$ of values
- We want to fit a Gaussian model $N(\mu, \sigma)$ to the data
- Probability of observing a point x_i

$$P(x_i) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}$$

- Probability of observing all points (we assume independence)

$$P(X) = \prod_{i=1}^{n} P(x_i) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}$$

- We want to find the parameters $\theta = (\mu, \sigma)$ that maximizes the probability $P(X|\theta)$

Maximum Likelihood Estimation (MLE)

- The probability $P(X|\theta)$ as a function of θ is the **Likelihood** function

$$L(\theta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}$$

- It is usually easier to work with the **Log-Likelihood** function

$$LL(\theta) = -\sum_{i=1}^{n} \frac{(x_i-\mu)^2}{2\sigma^2} - \frac{1}{2} n \log 2\pi - n \log \sigma$$

- Thus, the Maximum Likelihood Estimation for the Gaussian Model consists in finding the parameters μ, σ that maximize $LL(\theta)$

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i = \mu_X$$

Sample Mean

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} (x_i-\mu)^2 = \sigma_X^2$$

Sample Variance

Maximum Likelihood Estimation (MLE)

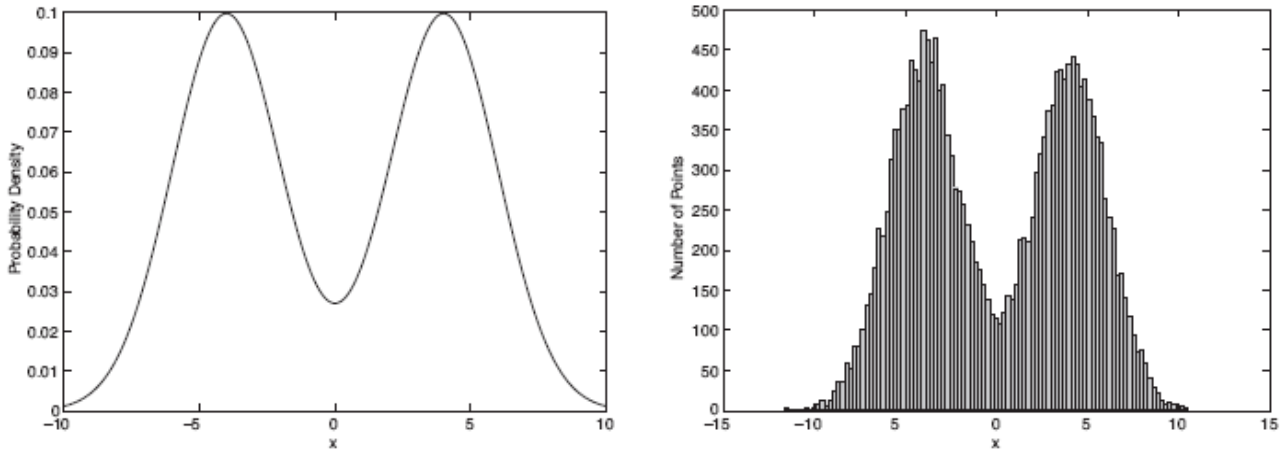
- Note: these are also the most likely parameters given the data.

$$P(\theta|X) = \frac{P(X|\theta)P(\theta)}{P(X)}$$

- If we have no prior information about θ, or X, then maximizing $P(\theta|X)$ is the same as maximizing $P(X|\theta)$.

Mixture of Gaussians

- Suppose that you have the heights of people from Greece and China and the distribution looks like the figure below (dramatization)



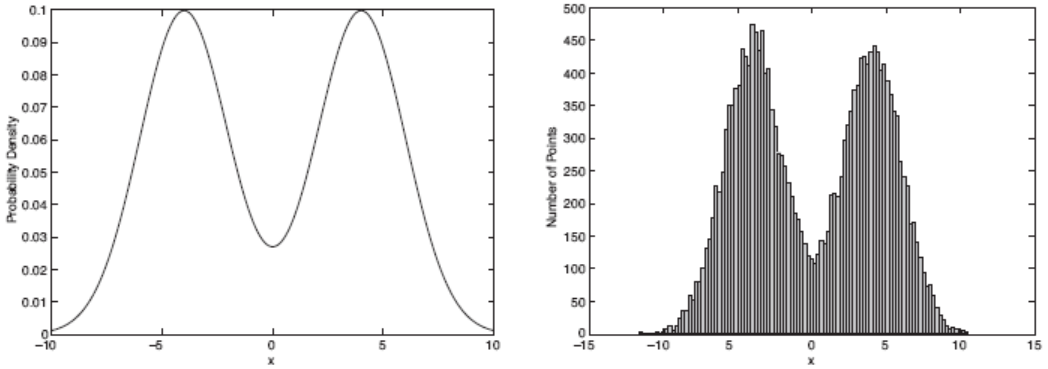
(a) Probability density function for the mixture model.

(b) 20,000 points generated from the mixture model.

Figure 9.2. Mixture model consisting of two normal distributions with means of -4 and 4, respectively. Both distributions have a standard deviation of 2.

Mixture of Gaussians

- In this case the data is the result of the **mixture** of two Gaussians
  - One for Greek people, and one for Chinese people
  - Identifying for each value which Gaussian is most likely to have generated it will give us a clustering.



(a) Probability density function for the mixture model.

(b) 20,000 points generated from the mixture model.

Figure 9.2. Mixture model consisting of two normal distributions with means of -4 and 4, respectively. Both distributions have a standard deviation of 2.

Mixture Model

- A value x_i is generated according to the following process:
  - First select the nationality
    - With probability π_G select Greek, with probability π_C select China ($\pi_G + \pi_C = 1$)
  - Given the nationality, generate the point from the corresponding Gaussian
    - $P(x_i|\theta_G) \sim N(\mu_G, \sigma_G)$ if Greece
    - $P(x_i|\theta_C) \sim N(\mu_C, \sigma_C)$ if China

Mixture Model

Assign a point to a cluster. In K-Means they are the membership: hard assignment.

Describe a cluster. In K-Means they are the centroids.

- Our model has the following parameters

$$\Theta = (\pi_G, \pi_C, \mu_G, \mu_C, \sigma_G, \sigma_C)$$

Mixture probabilities | Distribution Parameters

- For value x_i, we have:

$$P(x_i|\Theta) = \pi_G P(x_i|\theta_G) + \pi_C P(x_i|\theta_C)$$

- For all values $X = \{x_1, \ldots, x_n\}$

$$P(X|\Theta) = \prod_{i=1}^{n} P(x_i|\Theta)$$

- We want to estimate the parameters that **maximize** the Likelihood

Mixture Model

- Once we have the parameters $\theta = (\pi_G, \pi_C, \mu_G, \sigma_G, \mu_C, \sigma_C)$, we can **estimate** the **membership probabilities** $P(G|x_i)$ and $P(C|x_i)$ for each point x_i:

- This is the probability that point x_i belongs to the Greek or the Chinese population (cluster)

$$P(G|x_i) = \frac{P(x_i|G)P(G)}{P(x_i|G)P(G) + P(x_i|C)P(C)}$$
$$= \frac{P(x_i|G)\pi_G}{P(x_i|G)\pi_G + P(x_i|C)\pi_C}$$

Mixture of Gaussians as EM

- Initialize the values of the parameters in θ to some random values
- Repeat until convergence
  - **E-Step**: Given the parameters Θ **estimate** the membership probabilities $\mathrm{P}(G|x_i)$ and $\mathrm{P}(C|x_i)$.
  - **M-Step:** Calculate the parameter values Θ that (in expectation) **maximize** the data likelihood.

$$\pi_G = \frac{1}{n}\sum_{i=1}^{n} P(G|x_i) \qquad \pi_C = \frac{1}{n}\sum_{i=1}^{n} P(C|x_i)$$

Fraction of population in G,C

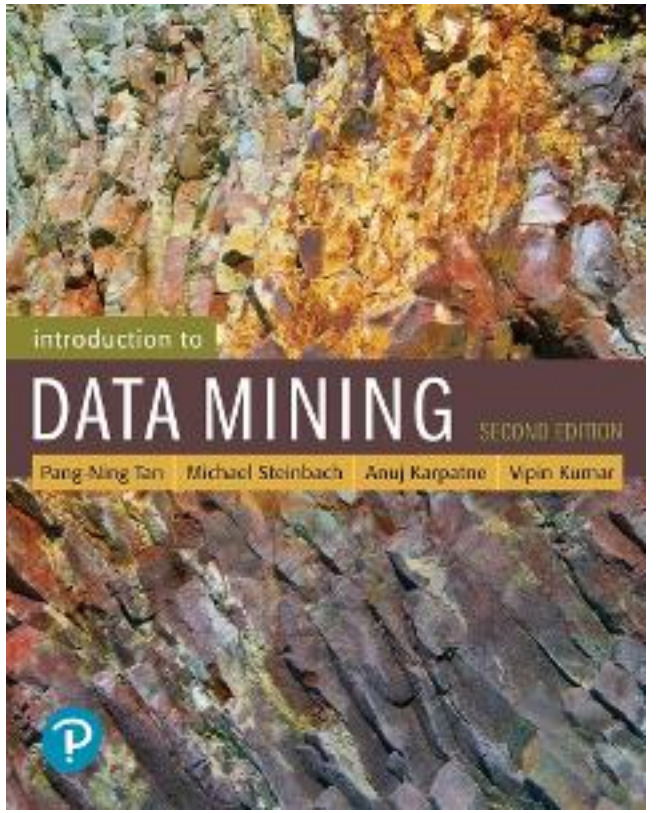
$$\mu_C = \sum_{i=1}^{n} \frac{P(C|x_i)}{n * \pi_C} x_i \qquad \mu_G = \sum_{i=1}^{n} \frac{P(G|x_i)}{n * \pi_G} x_i$$

MLE Estimates if π's were fixed

$$\sigma_C^2 = \sum_{i=1}^{n} \frac{P(C|x_i)}{n * \pi_C} (x_i - \mu_C)^2 \qquad \sigma_G^2 = \sum_{i=1}^{n} \frac{P(G|x_i)}{n * \pi_G} (x_i - \mu_G)^2$$

References

- Clustering. Chapter 7. Introduction to Data Mining.
- Pelleg, Dan, and Andrew W. Moore. "X-means: Extending k-means with efficient estimation of the number of clusters." 2000.



DATA MINING 1
Hierarchical Clustering

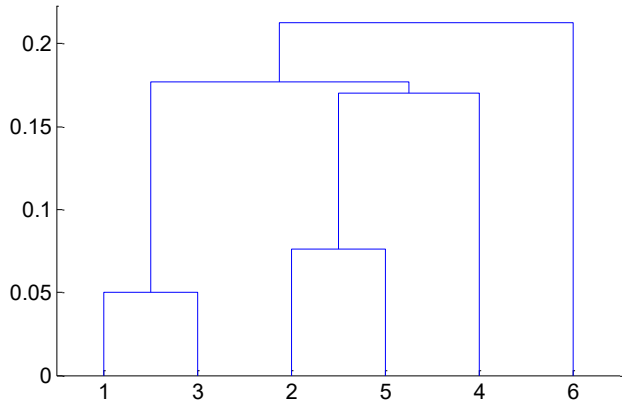
Dino Pedreschi, Riccardo Guidotti

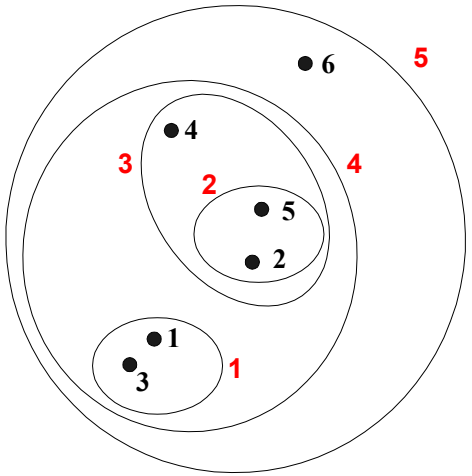
Revisited slides from Lecture Notes for Chapter 7 “Introduction to Data Mining”, 2nd Edition by Tan, Steinbach, Karpatne, Kumar

UNIVERSITÀ DI PISA

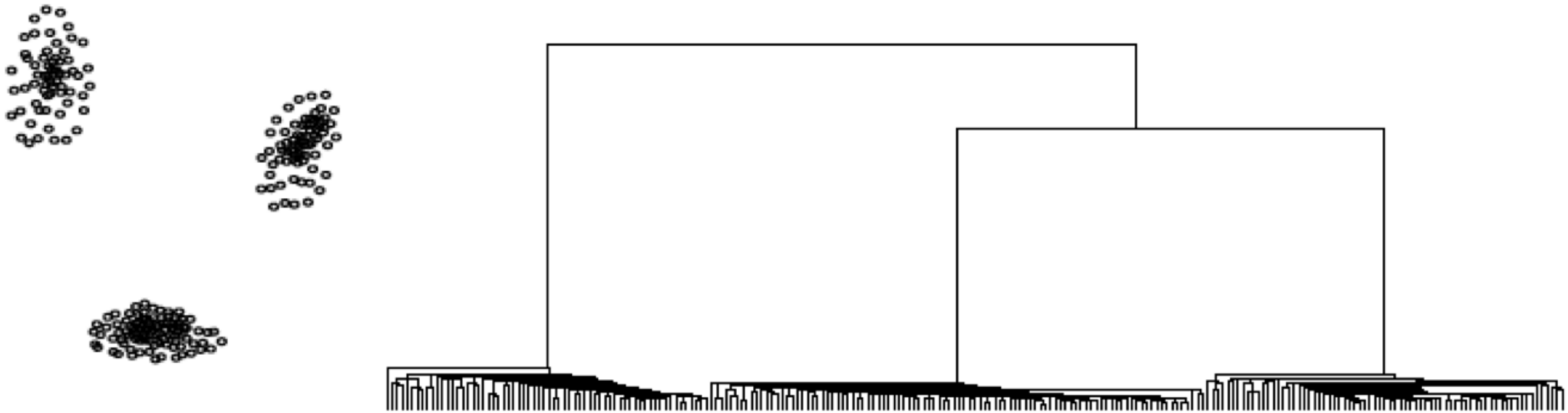
Hierarchical Clustering

- Produces a set of nested clusters organized as a hierarchical tree
- Can be visualized as a dendrogram
  - A tree like diagram that records the sequences of merges or splits

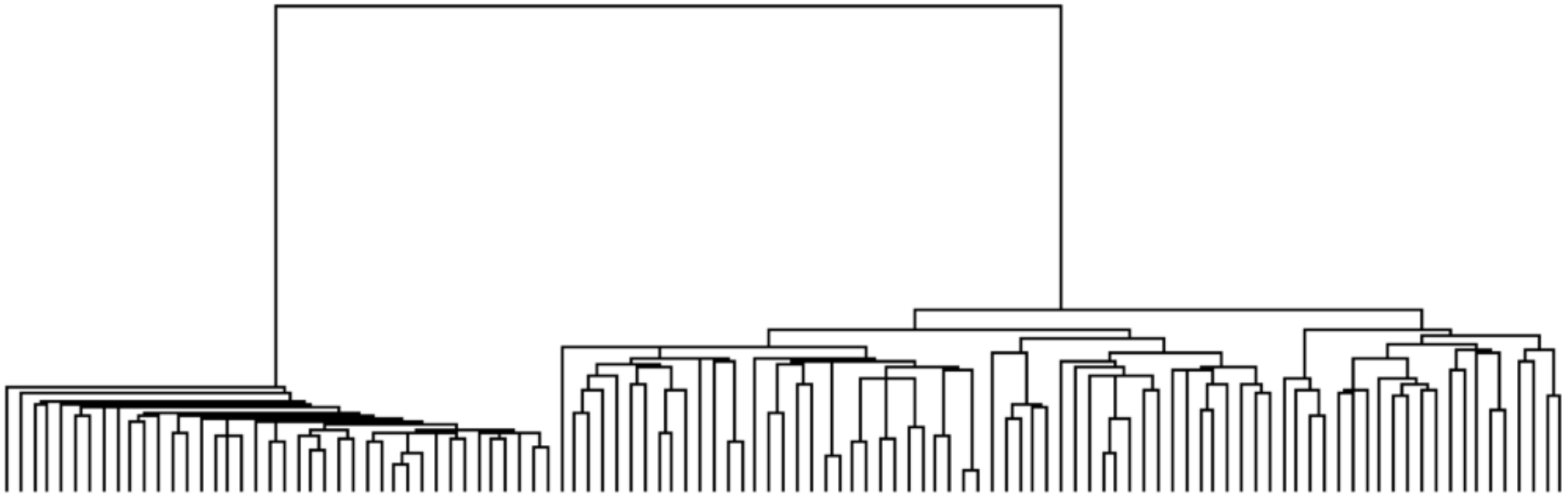




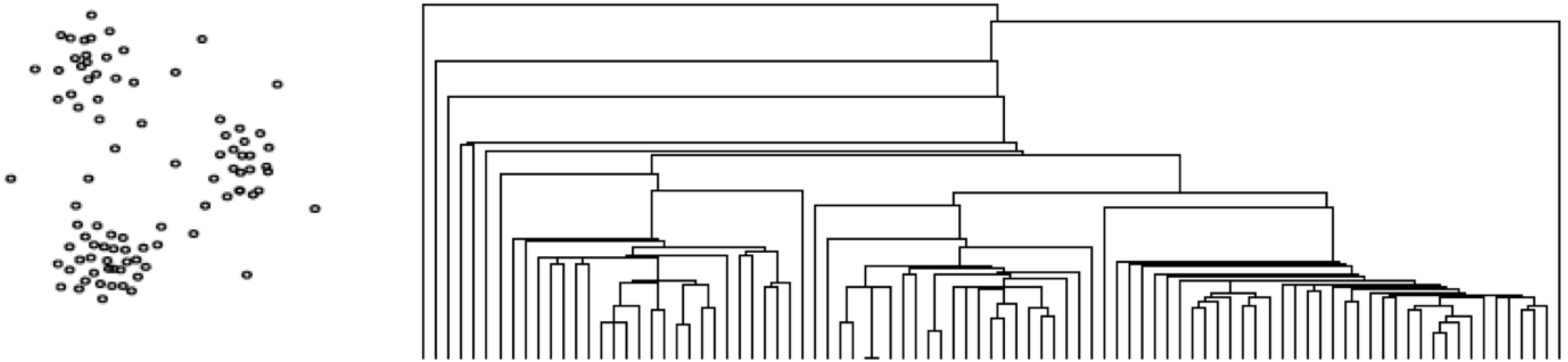
Dendrograms



Dendrograms



Dendrograms



Strengths of Hierarchical Clustering

- Do not have to assume any particular number of clusters
  - Any desired number of clusters can be obtained by ‘cutting’ the dendrogram at the proper level

- They may correspond to meaningful taxonomies
  - Example in biological sciences (e.g., animal kingdom, phylogeny reconstruction, …)

Hierarchical Clustering

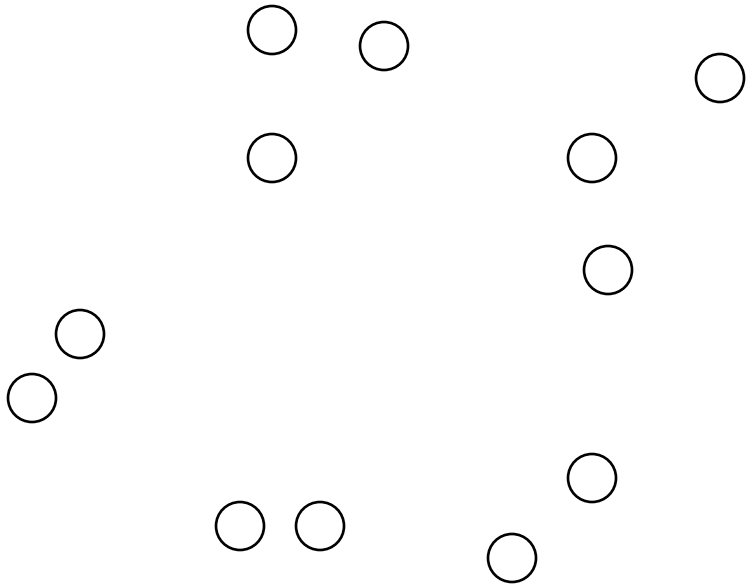
- Two main types of hierarchical clustering
  - Agglomerative:
    - Start with the points as individual clusters
    - At each step, merge the closest pair of clusters until only one cluster (or k clusters) left
  - Divisive:
    - Start with one, all-inclusive cluster
    - At each step, split a cluster until each cluster contains an individual point (or there are k clusters)
- Traditional hierarchical algorithms use a similarity or distance matrix
  - Merge or split one cluster at a time

Agglomerative Clustering Algorithm

- Most popular hierarchical clustering technique
- Basic algorithm is straightforward
  1. Compute the proximity matrix
  2. Let each data point be a cluster
  3. **Repeat**
  4. Merge the two closest clusters
  5. Update the proximity matrix
  6. **Until** only a single cluster remains
- Key operation is the computation of the proximity of two clusters
  - Different approaches to defining the distance between clusters distinguish the different algorithms

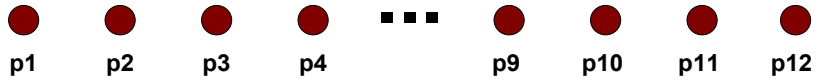
Starting Situation

- Start with clusters of individual points and a proximity matrix



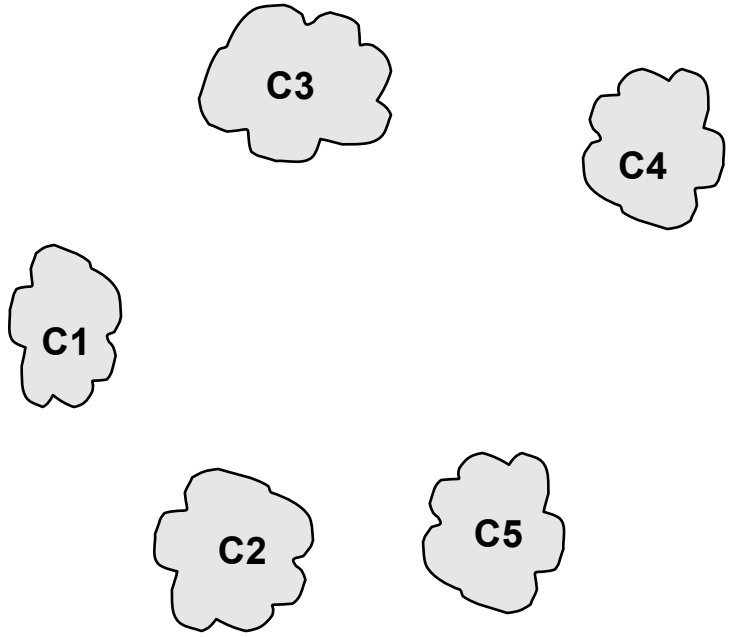
	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix



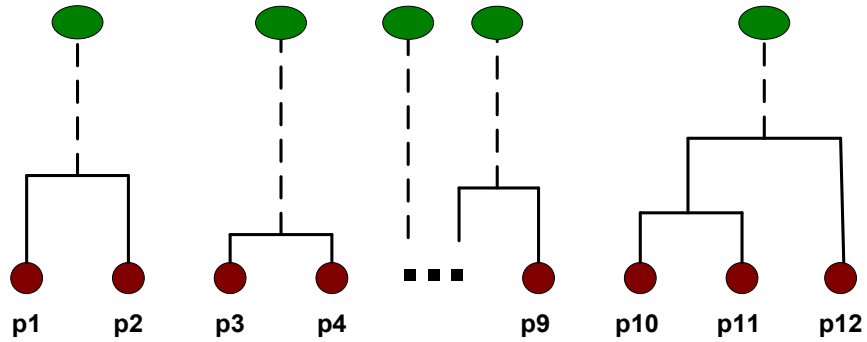
Intermediate Situation

- After some merging steps, we have some clusters



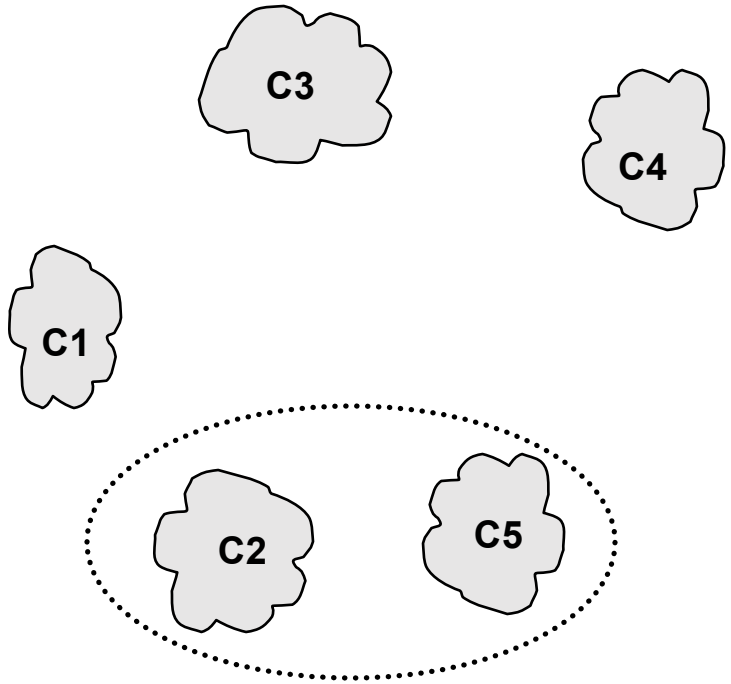
	C1	C2	C3	C4	C5
C1					
C2					
C3					
C4					
C5					

Proximity Matrix



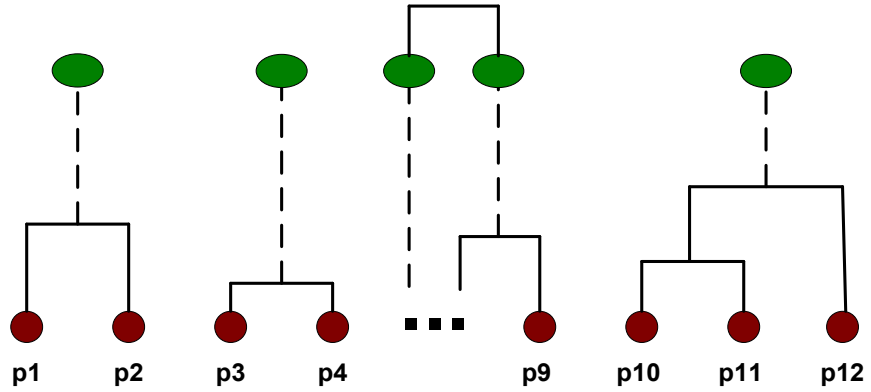
Intermediate Situation

- We want to merge the two closest clusters (C2 and C5) and update the proximity matrix.



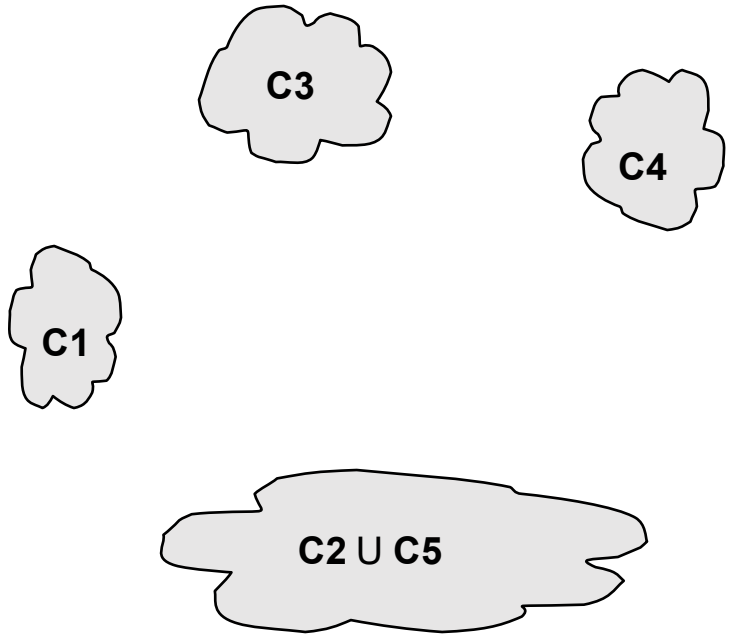
	C1	C2	C3	C4	C5
C1					
C2					
C3					
C4					
C5					

Proximity Matrix



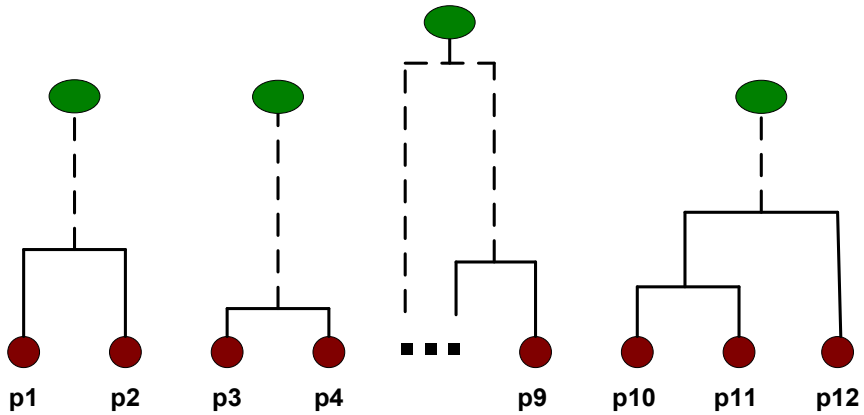
After Merging

- The question is “How do we update the proximity matrix?”

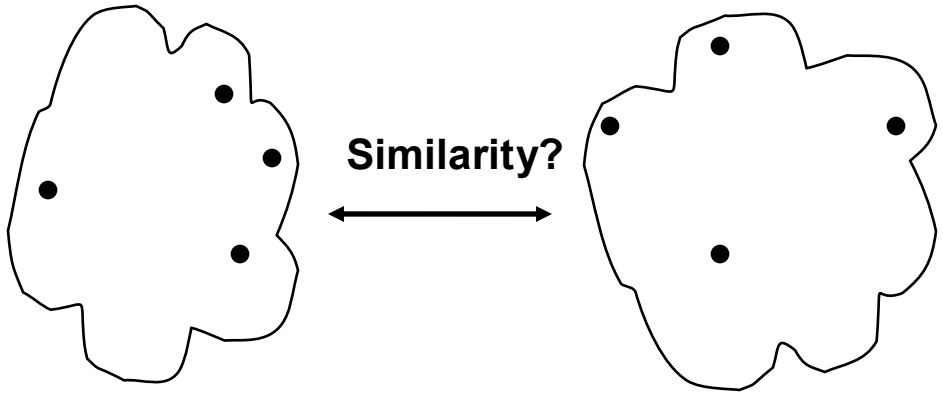


	C1	C2 U C5	C3	C4
C1		?		
C2 U C5	?	?	?	?
C3		?		
C4		?		

Proximity Matrix



How to Define Inter-Cluster Distance

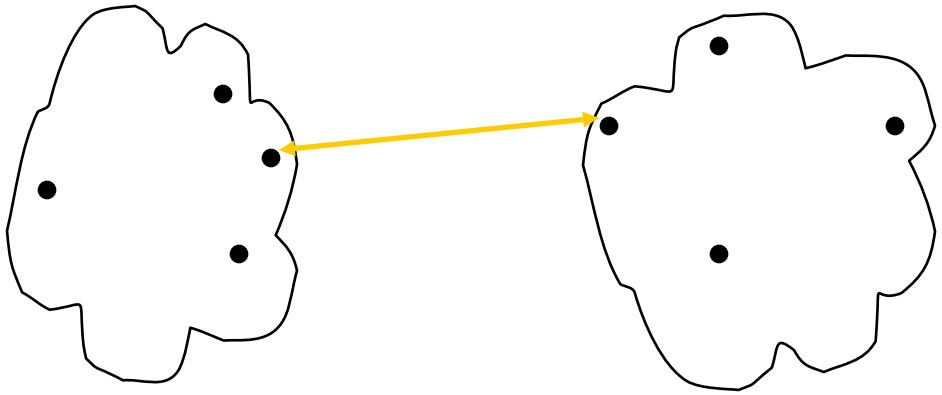


- MIN
- MAX
- Group Average
- Distance Between Centroids
- Other methods driven by an objective function
  - Ward's Method uses squared error

	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix

How to Define Inter-Cluster Similarity

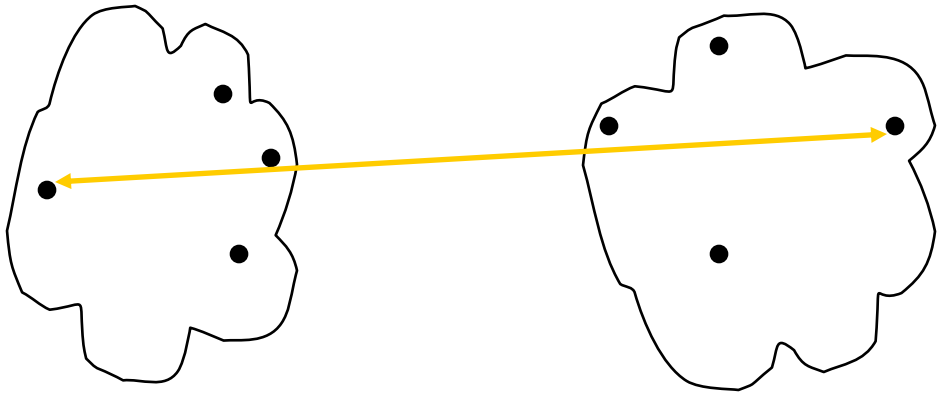


	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix

- MIN
- MAX
- Group Average
- Distance Between Centroids
- Other methods driven by an objective function
  - Ward's Method uses squared error

How to Define Inter-Cluster Similarity

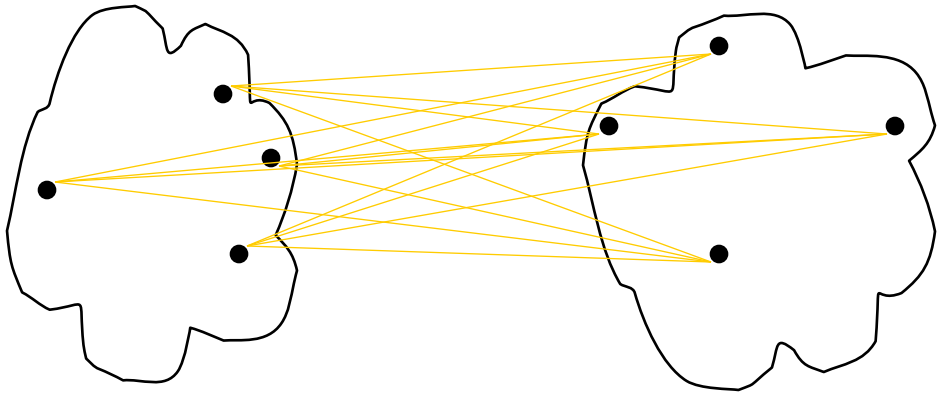


	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix

- MIN
- MAX
- Group Average
- Distance Between Centroids
- Other methods driven by an objective function
  - Ward's Method uses squared error

How to Define Inter-Cluster Similarity

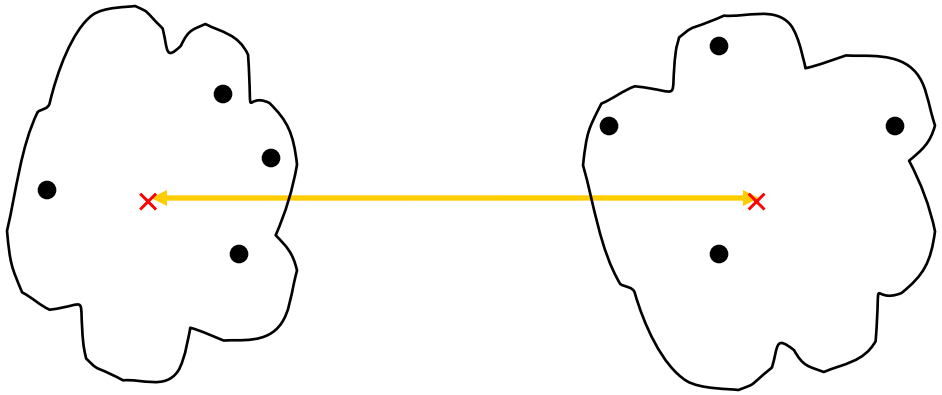


	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix

- MIN
- MAX
- Group Average
- Distance Between Centroids
- Other methods driven by an objective function
  - Ward's Method uses squared error

How to Define Inter-Cluster Similarity



	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix

- MIN
- MAX
- Group Average
- Distance Between Centroids
- Other methods driven by an objective function
  - Ward's Method uses squared error

Ward Linkage Method

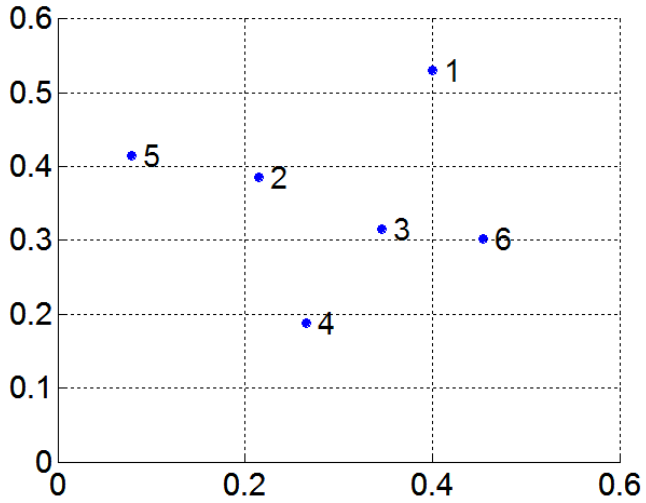
- Ward's method says that the distance between two clusters, A and B, is how much the sum of squares will increase when we merge them.

$$\Delta(A,B) = \sum_{i \in A \cup B} ||\vec{x_i} - \vec{m}_{A \cup B}||^2 - \sum_{i \in A} ||\vec{x_i} - \vec{m}_A||^2 - \sum_{i \in B} ||\vec{x_i} - \vec{m}_B||^2 = \frac{n_A n_B}{n_A + n_B} ||\vec{m}_A - \vec{m}_B||^2$$

- where m_j is the center of cluster *j*, and n_j is the number of points in it.
- Δ is called the merging cost of combining the clusters A and B.
- With hierarchical clustering, the sum of squares starts from zero (because every point is in its own cluster) and then grows as we merge clusters.

MIN or Single Link

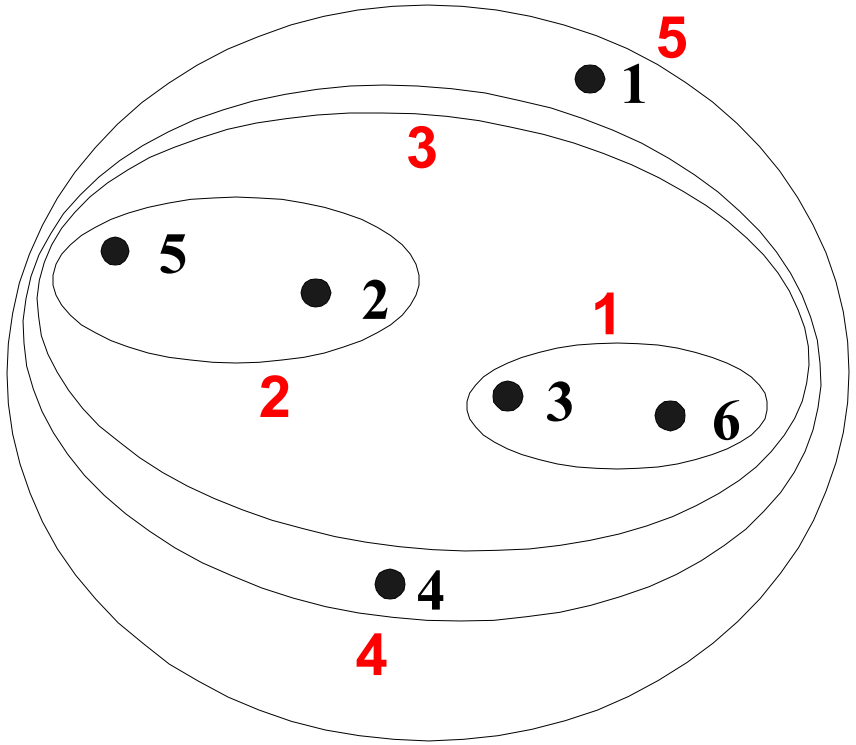
- Proximity of two clusters is based on the two closest points in the different clusters
  - Determined by one pair of points, i.e., by one link in the proximity graph
- Example:



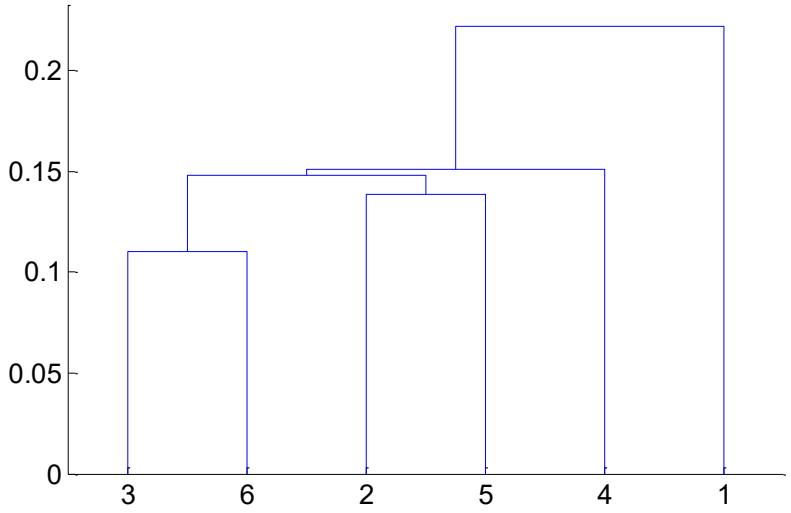
Distance Matrix:

	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

Hierarchical Clustering: MIN

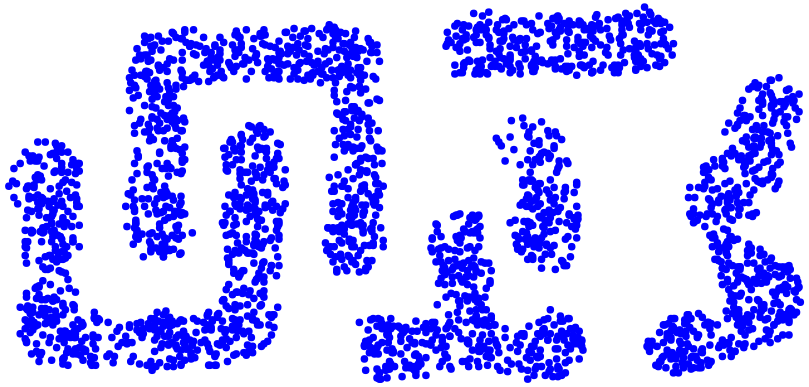


Nested Clusters

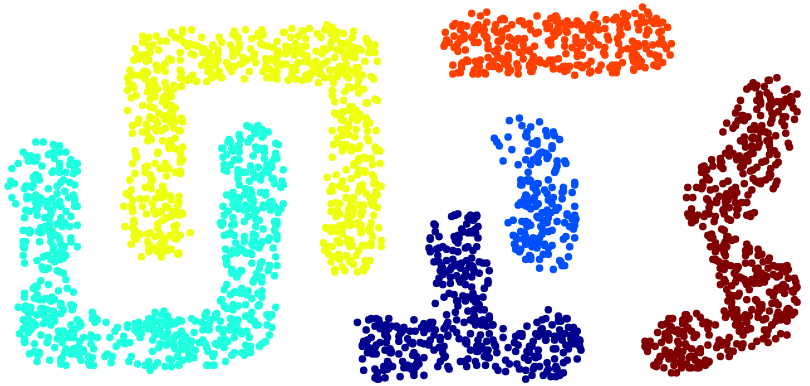


Dendrogram

Strength of MIN



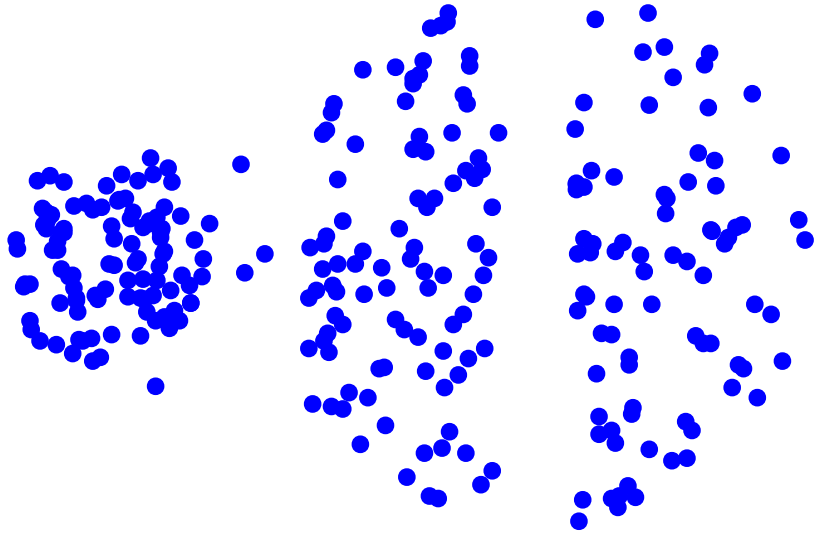
Original Points



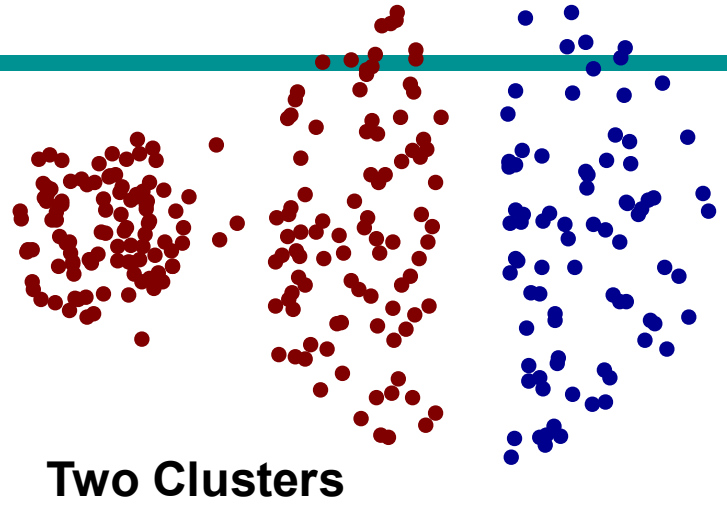
Six Clusters

- Can handle non-elliptical shapes

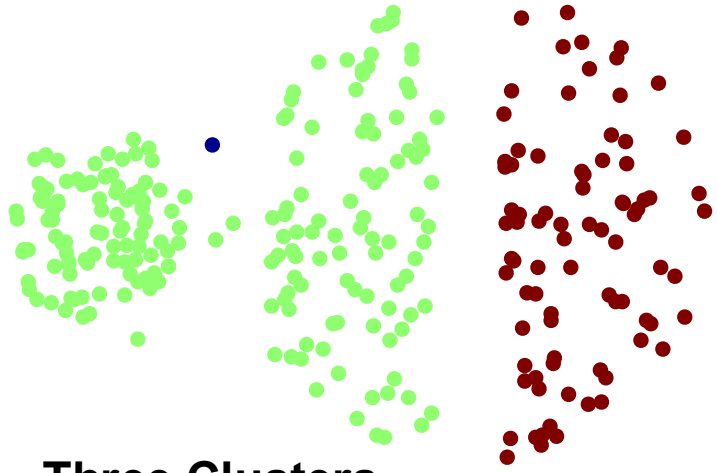
Limitations of MIN



Original Points



Two Clusters

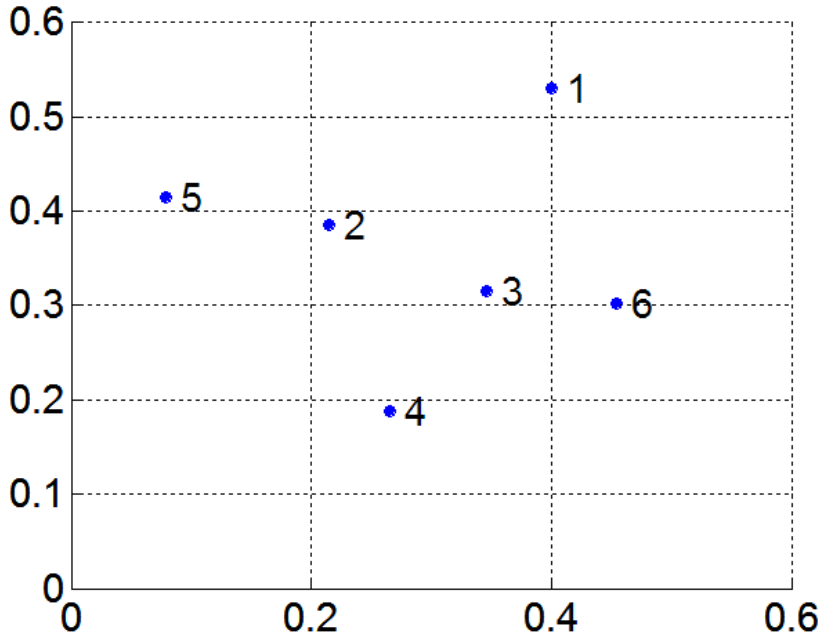


Three Clusters

- Sensitive to noise and outliers

MAX or Complete Linkage

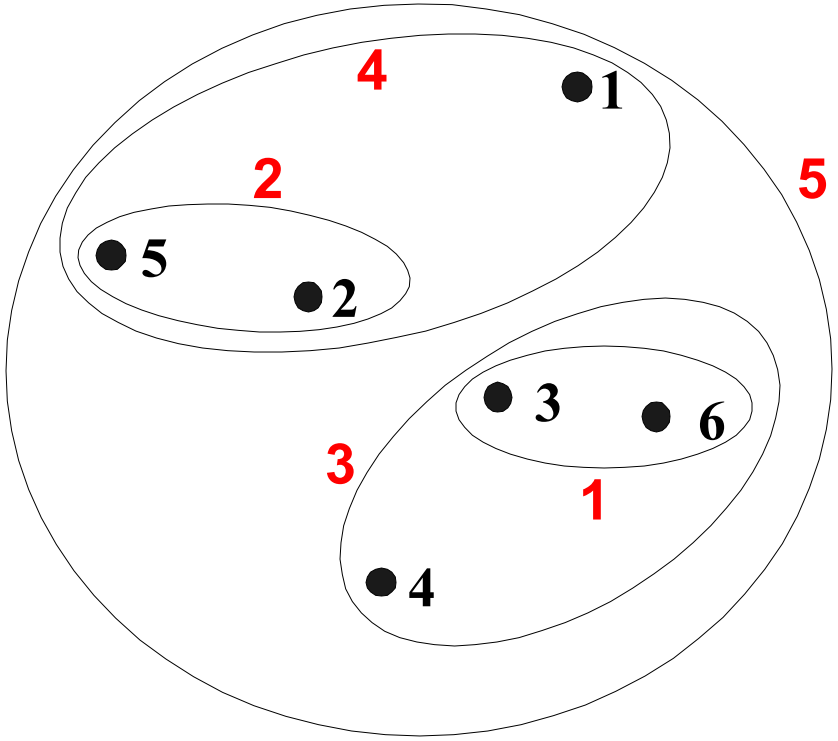
- Proximity of two clusters is based on the two most distant points in the different clusters
  - Determined by all pairs of points in the two clusters



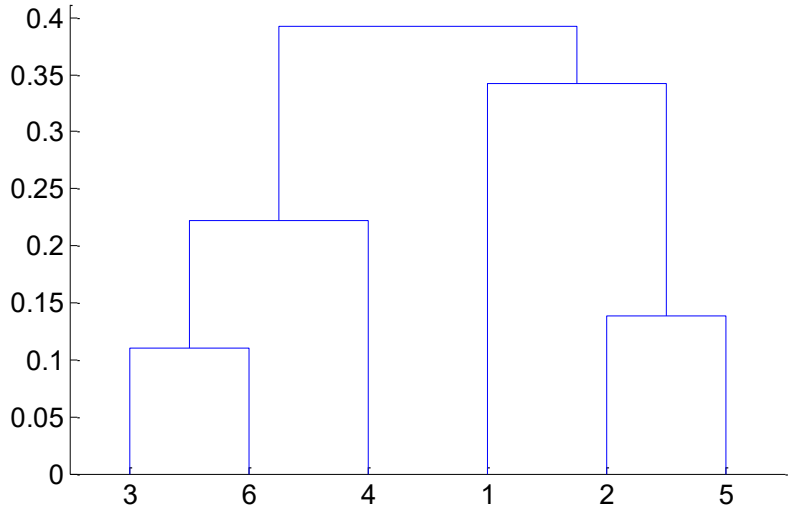
Distance Matrix:

	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

Hierarchical Clustering: MAX

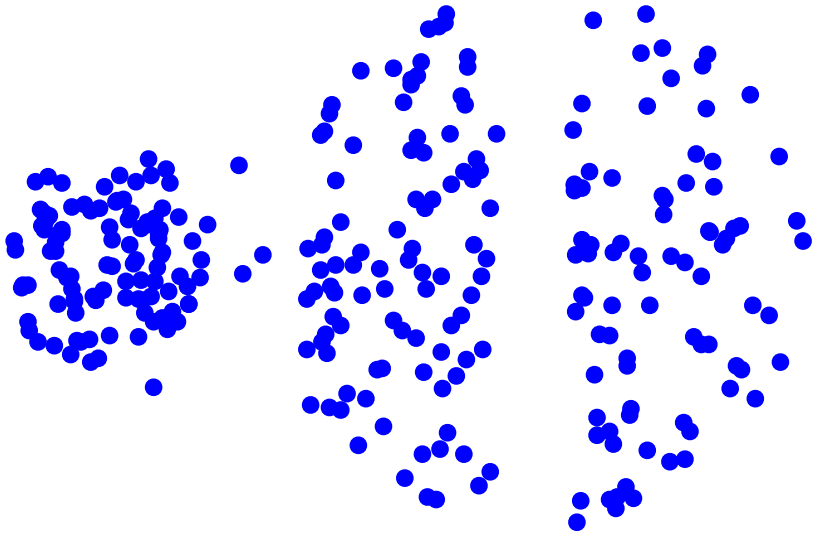


Nested Clusters

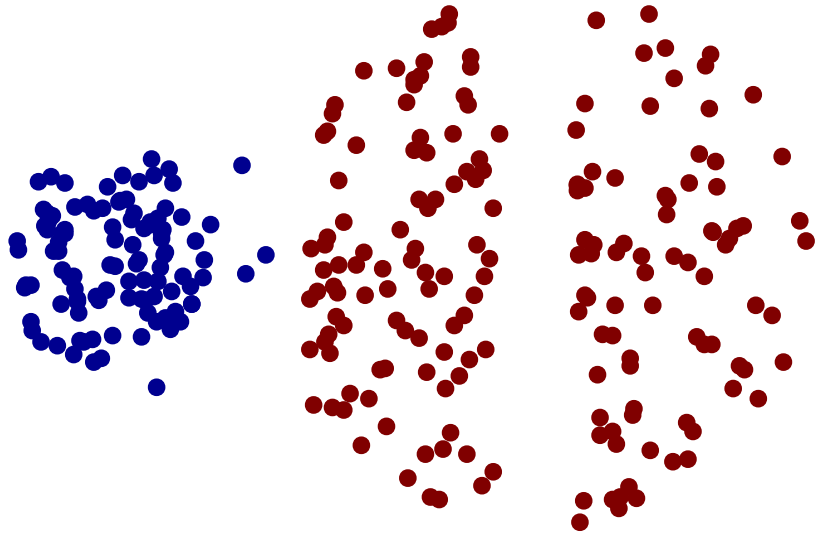


Dendrogram

Strength of MAX



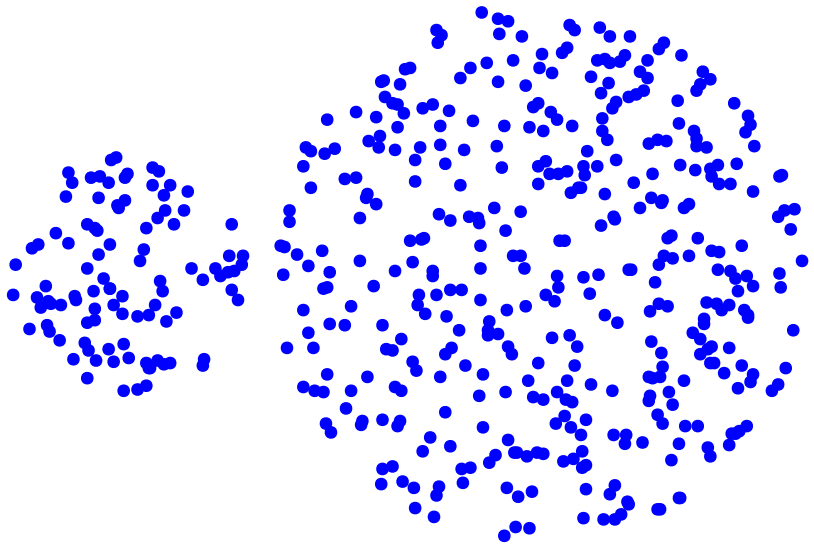
Original Points



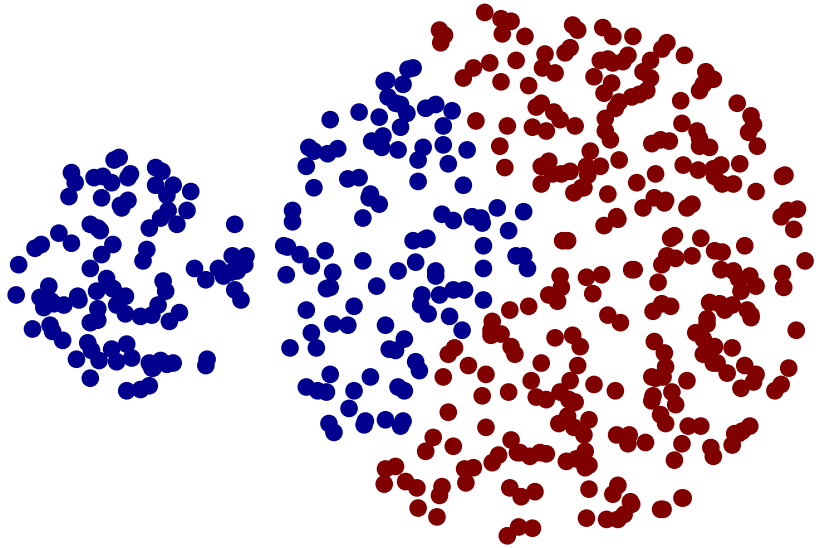
Two Clusters

- Less susceptible to noise and outliers

Limitations of MAX



Original Points



Two Clusters

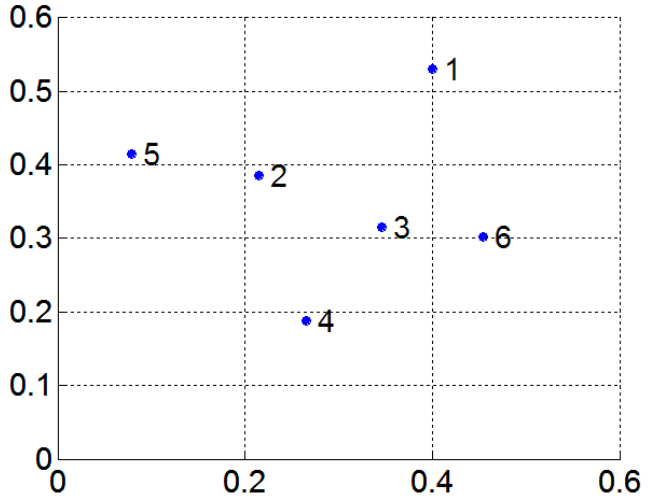
- Tends to break large clusters
- Biased towards globular clusters

Group Average

- Proximity of two clusters is the average of pairwise proximity between points in the two clusters.

$$\text{proximity}(\text{Cluster}_i, \text{Cluster}_j) = \frac{\sum_{\substack{p_i \in \text{Cluster}_i \\ p_j \in \text{Cluster}_j}} \text{proximity}(p_i, p_j)}{|\text{Cluster}_i| \times |\text{Cluster}_j|}$$

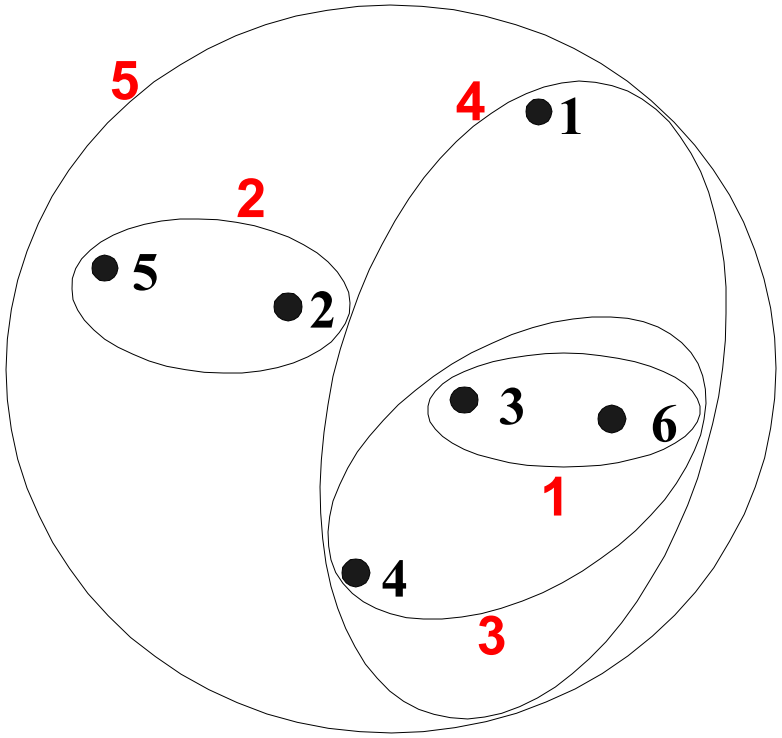
- Need to use average connectivity for scalability since total proximity favors large clusters



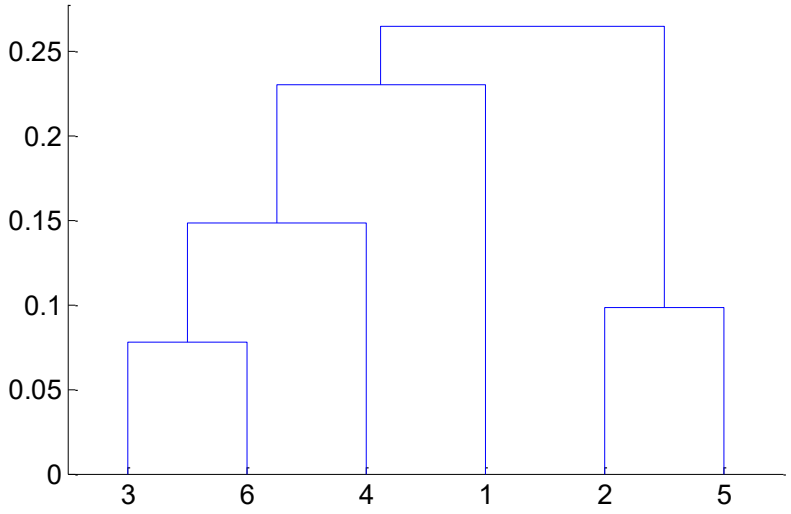
Distance Matrix:

	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

Hierarchical Clustering: Group Average



Nested Clusters



Dendrogram

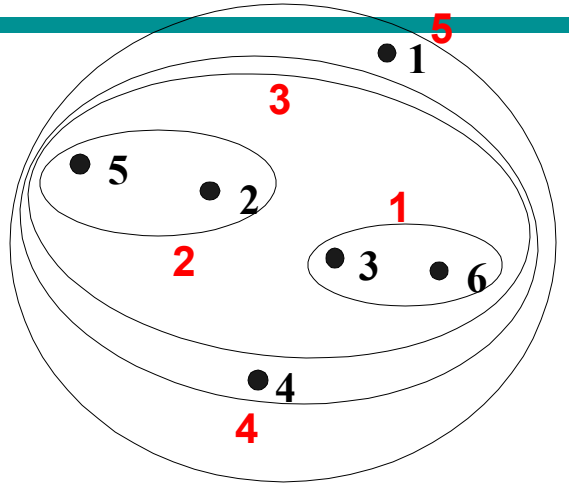
Hierarchical Clustering: Group Average

- Compromise between Single and Complete Link
- Strengths
  - Less susceptible to noise and outliers
- Limitations
  - Biased towards globular clusters

Cluster Similarity: Ward's Method

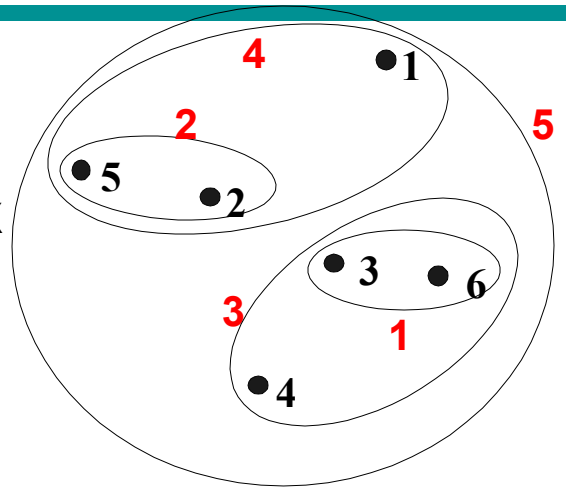
- Similarity of two clusters is based on the increase in squared error when two clusters are merged
  - Similar to group average if distance between points is distance squared
- Less susceptible to noise and outliers
- Biased towards globular clusters
- Hierarchical analogue of K-means
  - Can be used to initialize K-means

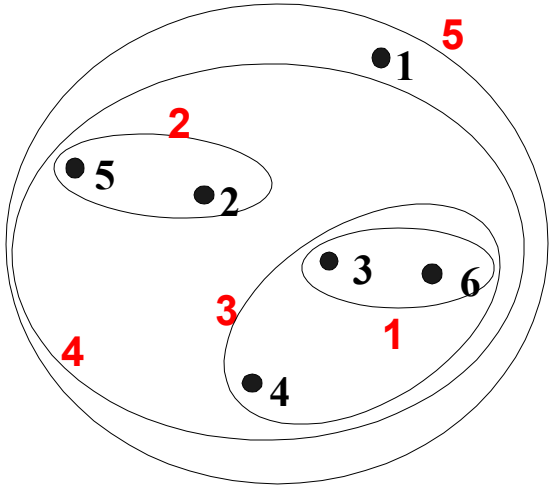
Hierarchical Clustering: Comparison



MIN

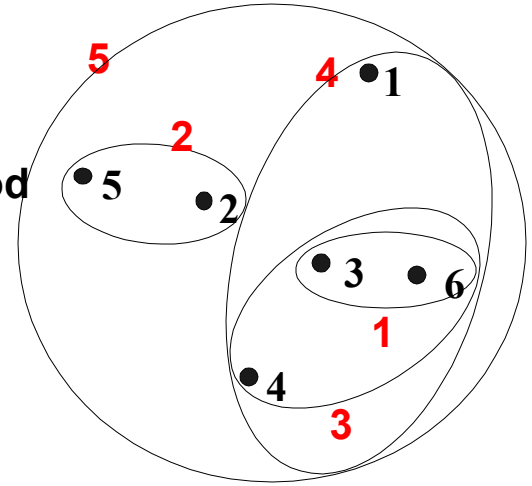
MAX





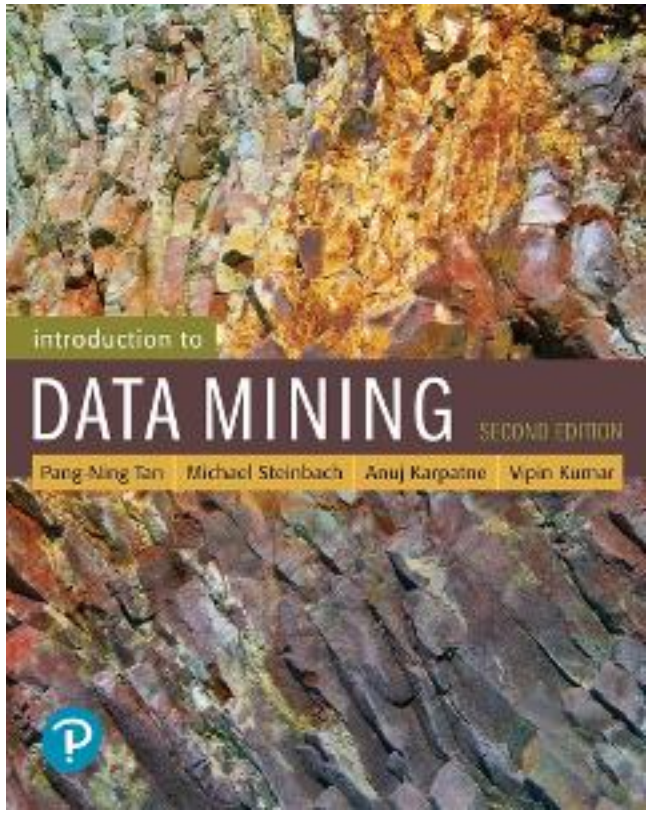
Group Average

Ward's Method



References

- Clustering. Chapter 7. Introduction to Data Mining.



DATA MINING 1
Density-based Clustering

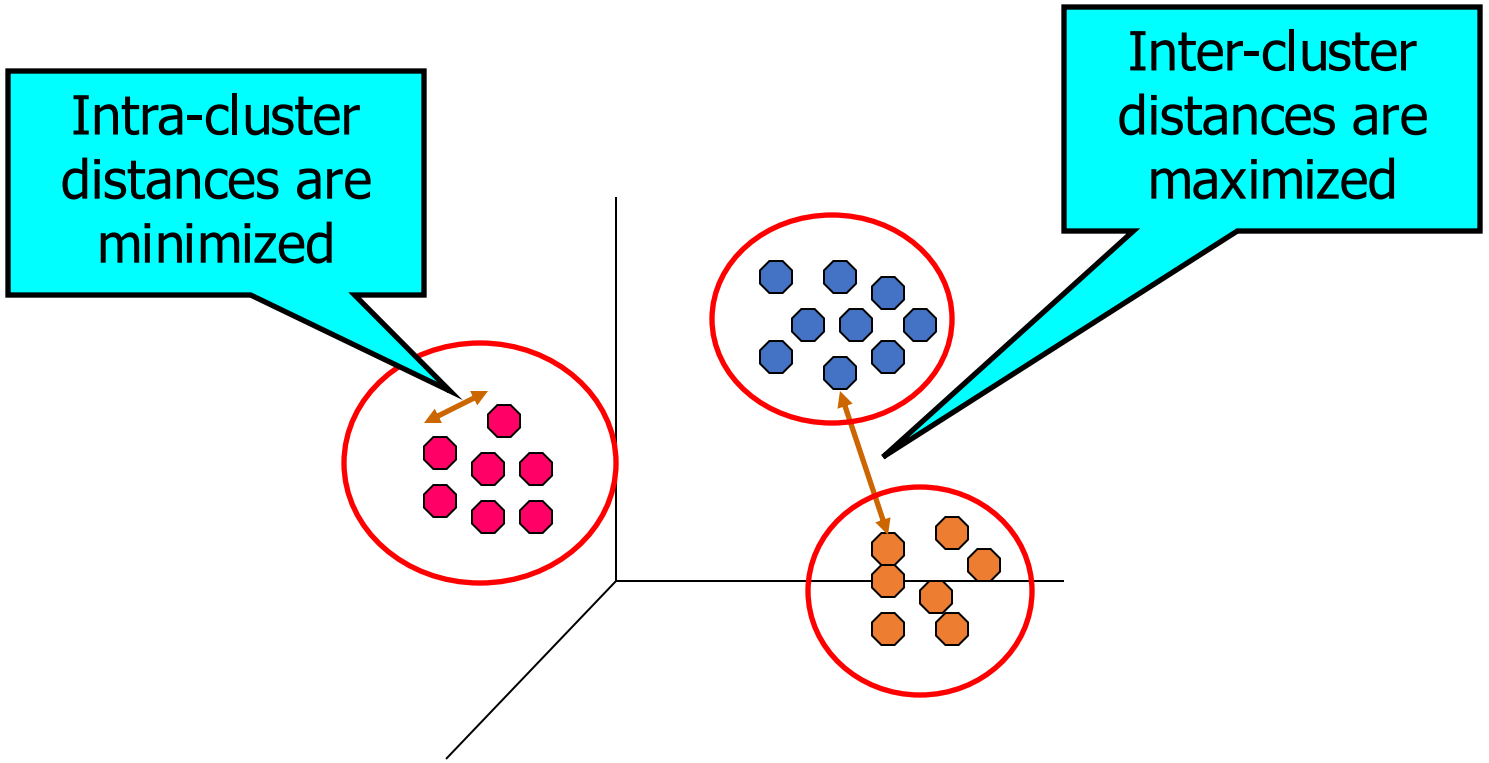
Riccardo Guidotti

Revisited slides from Lecture Notes for Chapter 7 “Introduction to Data Mining”, 2nd Edition by Tan, Steinbach, Karpatne, Kumar

UNIVERSITÀ DI PISA

What is Cluster Analysis?

- Finding groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects in other groups

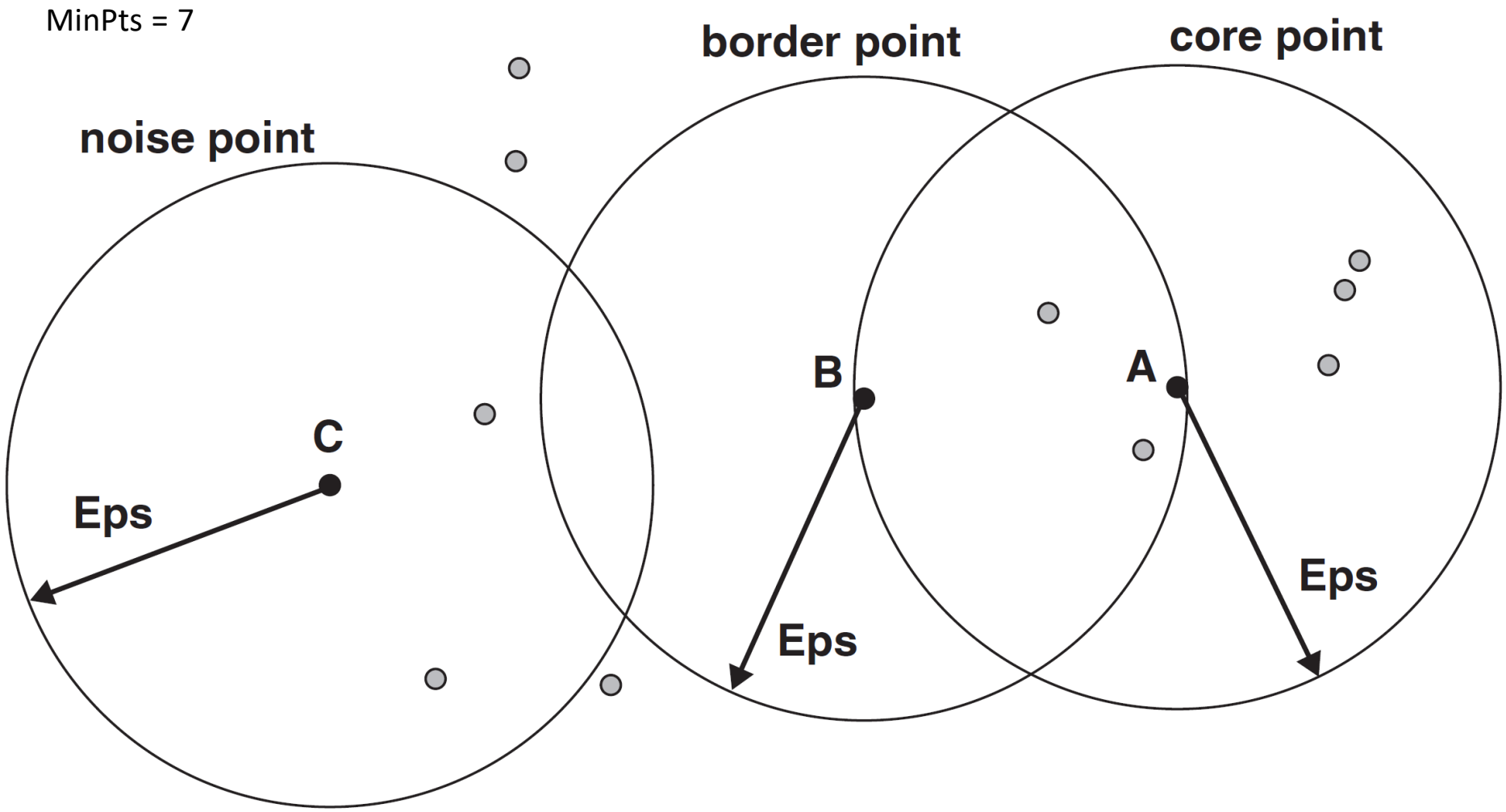


DBSCAN

DBSCAN

- DBSCAN is a density-based algorithm.
  - Density = number of points within a specified radius (Eps)
  - A point is a core point if it has at least a specified number of points (MinPts) within Eps
    - These are points that are at the interior of a cluster
    - Counts the point itself
  - A border point is not a core point, but is in the neighborhood of a core point
  - A noise point is any point that is not a core point or a border point

DBSCAN: Core, Border, and Noise Points



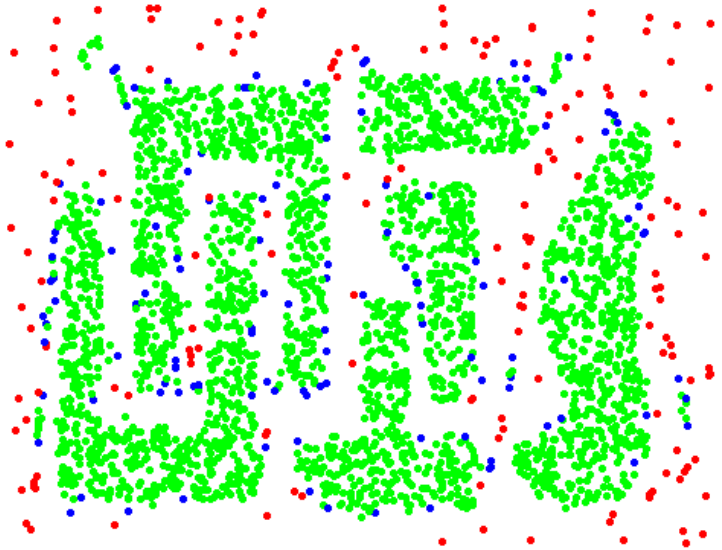
DBSCAN Algorithm

- Eliminate noise points
- Perform clustering on the remaining points

current_cluster_label $\leftarrow 1$
for all core points **do**
  if the core point has no cluster label **then**
    current_cluster_label $\leftarrow$ *current_cluster_label* $+ 1$
    Label the current core point with cluster label *current_cluster_label*
  end if
  for all points in the *Eps*-neighborhood, except i^{th} the point itself **do**
    if the point does not have a cluster label **then**
      Label the point with cluster label *current_cluster_label*
    end if
  end for
end for

DBSCAN: Core, Border and Noise Points

Original Points

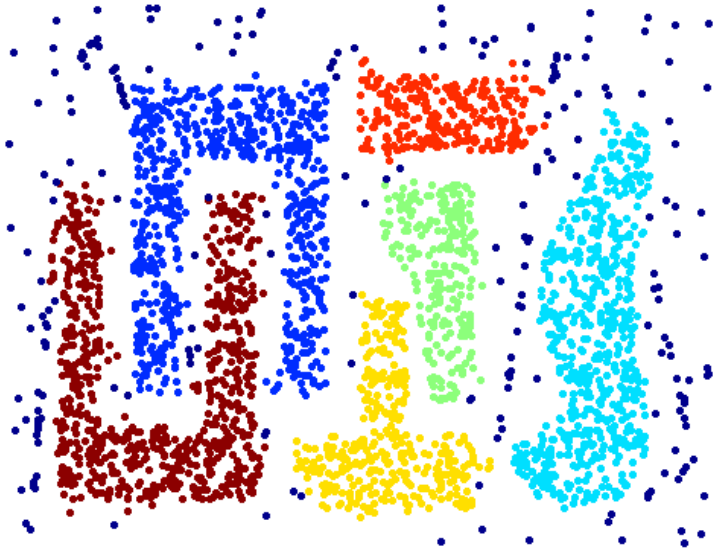


Point types: core, border and noise

Eps = 10, MinPts = 4

When DBSCAN Works Well

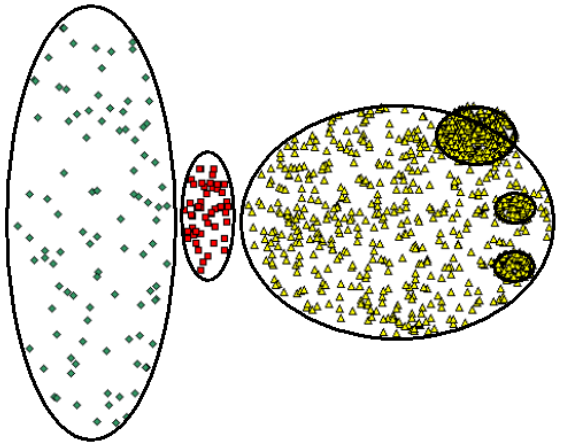
Original Points



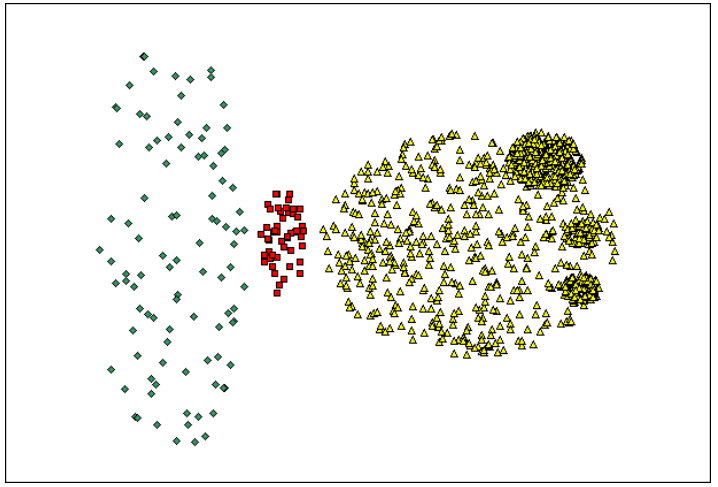
Clusters

- Resistant to Noise
- Can handle clusters of different shapes and sizes

When DBSCAN Does NOT Work Well

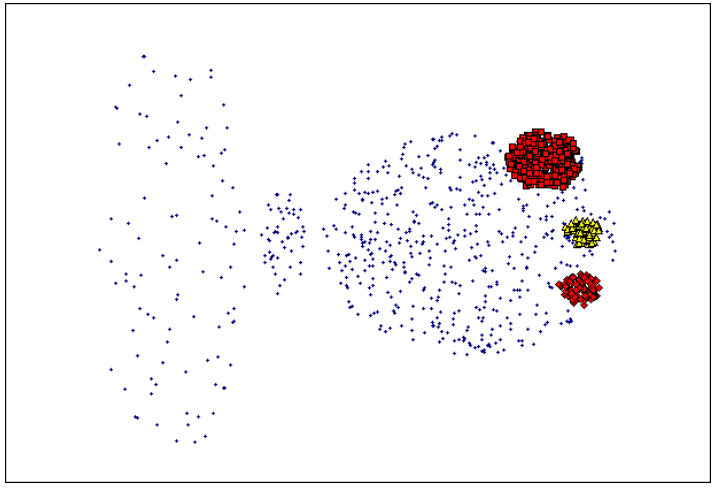


Original Points



(MinPts=4, Eps=9.75).

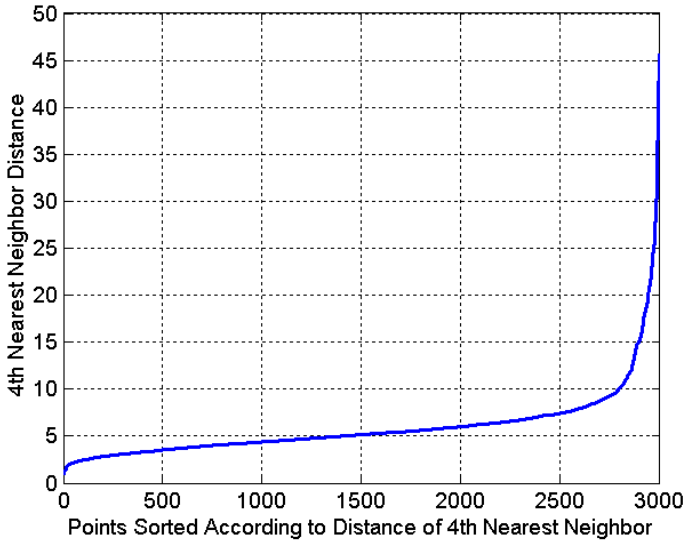
- **Varying densities**
- **High-dimensional data**



(MinPts=4, Eps=9.92)

DBSCAN: Determining EPS and MinPts

- Idea is that for points in a cluster, their k^{th} nearest neighbors are at roughly the same distance
- Noise points have the k^{th} nearest neighbor at farther distance
- So, plot sorted distance of every point to its k^{th} nearest neighbor

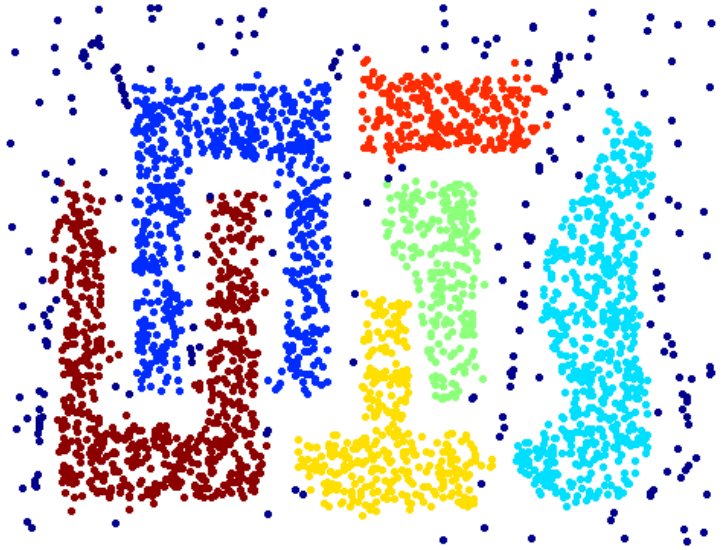


DBSCAN Evolution

OPTICS

When DBSCAN Works Well

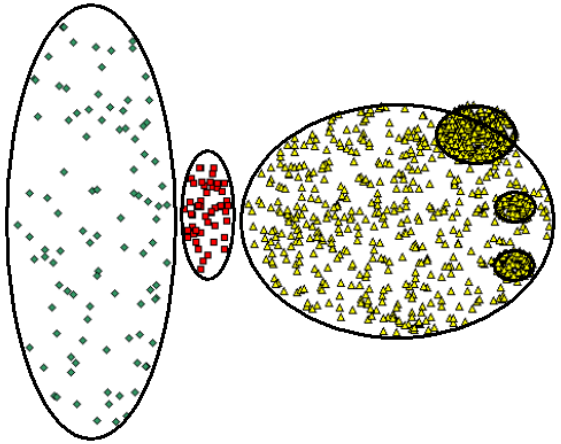
Original Points



Clusters

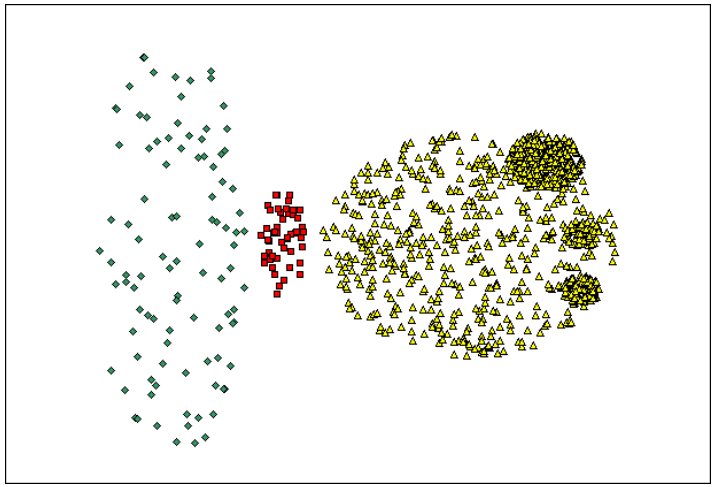
- **Resistant to Noise**
- **Can handle clusters of different shapes and sizes**

When DBSCAN Does NOT Work Well

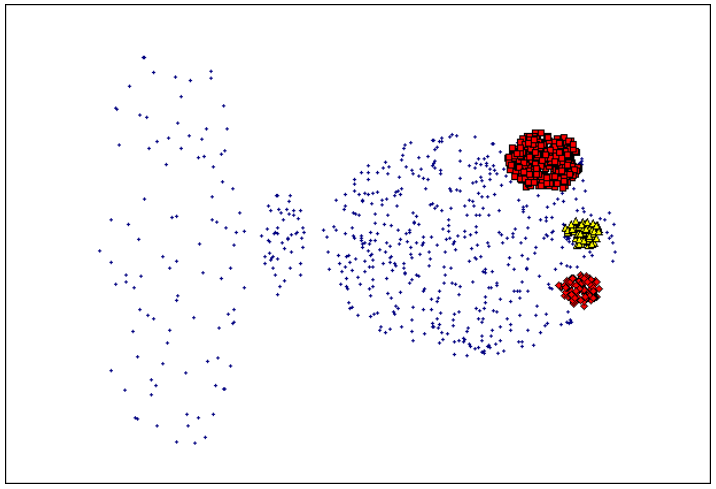


Original Points

- **Varying densities**
- **High-dimensional data**



(MinPts=4, Eps=9.92).



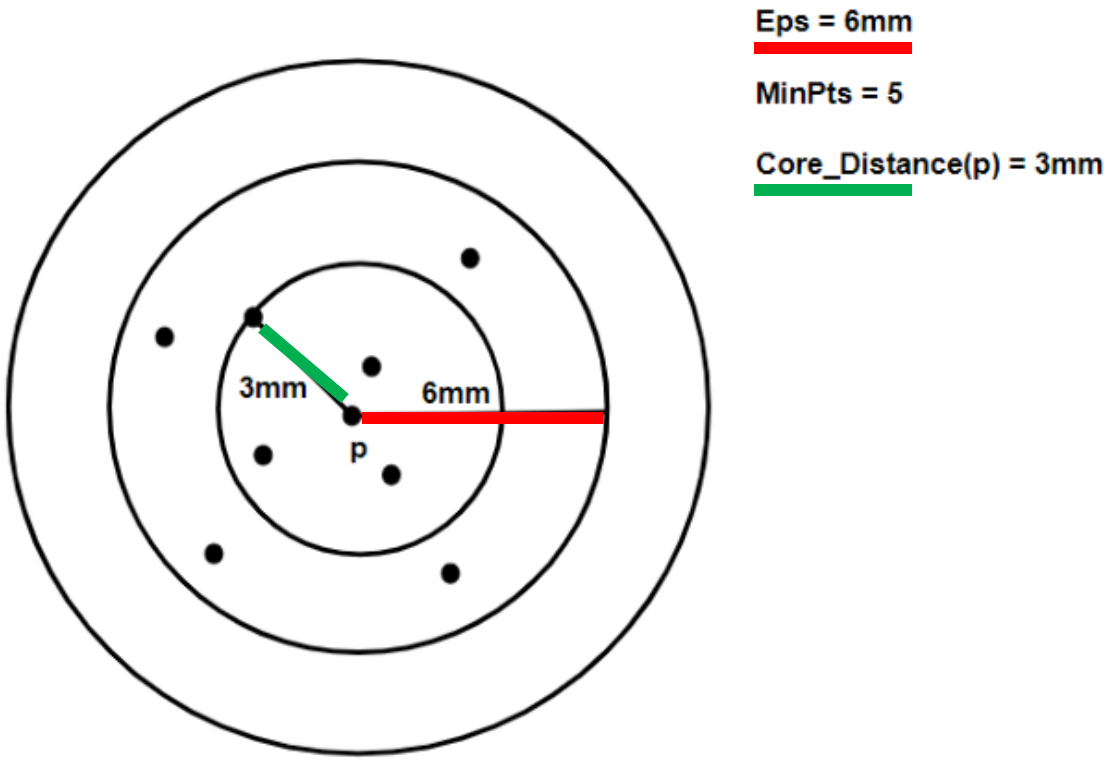
(MinPts=4, Eps=9.75)

OPTICS

- OPTICS: Ordering Points To Identify the Clustering Structure
  - Produces a special order of the dataset wrt its density-based clustering structure.
  - This cluster-ordering contains info equivalent to the density-based clusterings corresponding to a broad range of parameter settings.
  - Good for both automatic and interactive cluster analysis, including finding intrinsic clustering structure.
  - Can be represented graphically or using visualization techniques.

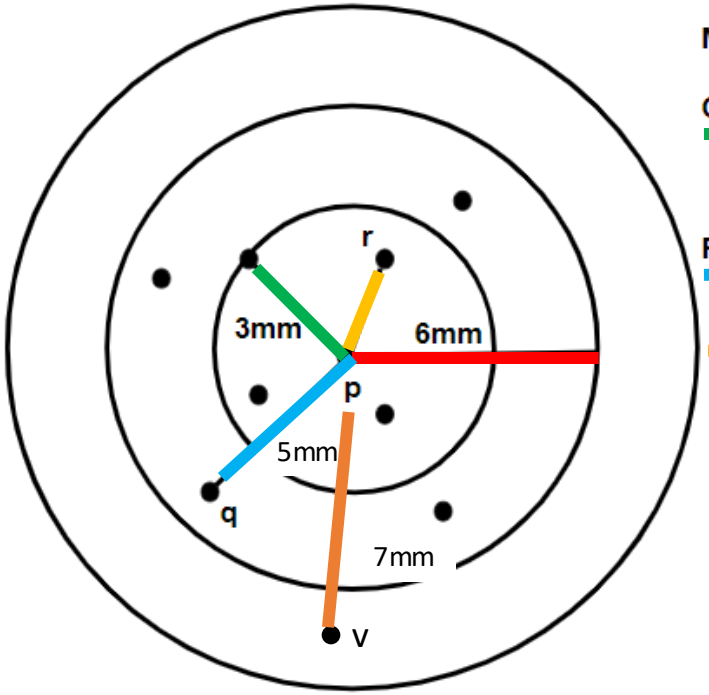
OPTICS: Extension from DBSCAN

- OPTICS requires two **parameters**:
  - ε, which describes the maximum distance (radius) to consider,
  - MinPts, describing the number of points required to form a cluster
- **Core point**. A point *p* is a core point if at least MinPts points are found within its ε-neighborhood.
- **Core Distance**. It is the **minimum** value of radius required to classify a given point as a core point. If the given point is not a Core point, then it's Core Distance is undefined.



OPTICS: Extension from DBSCAN

- **Reachability Distance**. The reachability distance between a point *p* and *q* is the **maximum** of the Core Distance of *p* and the Distance between p and q.
- The Reachability Distance is not defined if *q* is not a Core point. Below is the example of the Reachability Distance.
- In other words, if *q* is within the core distance of *p* then use the core distance, otherwise the real distance.



Eps = 6mm

MinPts = 5

Core_Distance(p) = 3mm

Reachability_Distance(q,p) = 5mm

Reachability_Distance(r,p) = 3mm

Reachability_Distance(v,p) = 7mm

OPTICS Pseudo-Code

- For each point *p* in the dataset
  - Initialize the reachability distance of *p* as undefined
- For each unprocessed point *p* in the dataset
  - Get the neighbors *N* of p
  - Mark *p* as processed and output to the *ordered list*
  - If *p* is a core point
    - Initialize a priority queue *Q* to get the closest point to *p* in terms of reachability
    - Call the function *update(N, p, Q)*
    - For each point *q* in *Q*
      - Get the neighbors *N'* of *q*
      - Mark q as processed and output to the *ordered list*
        - If *q* is a core point Call the function update(N', q, Q)

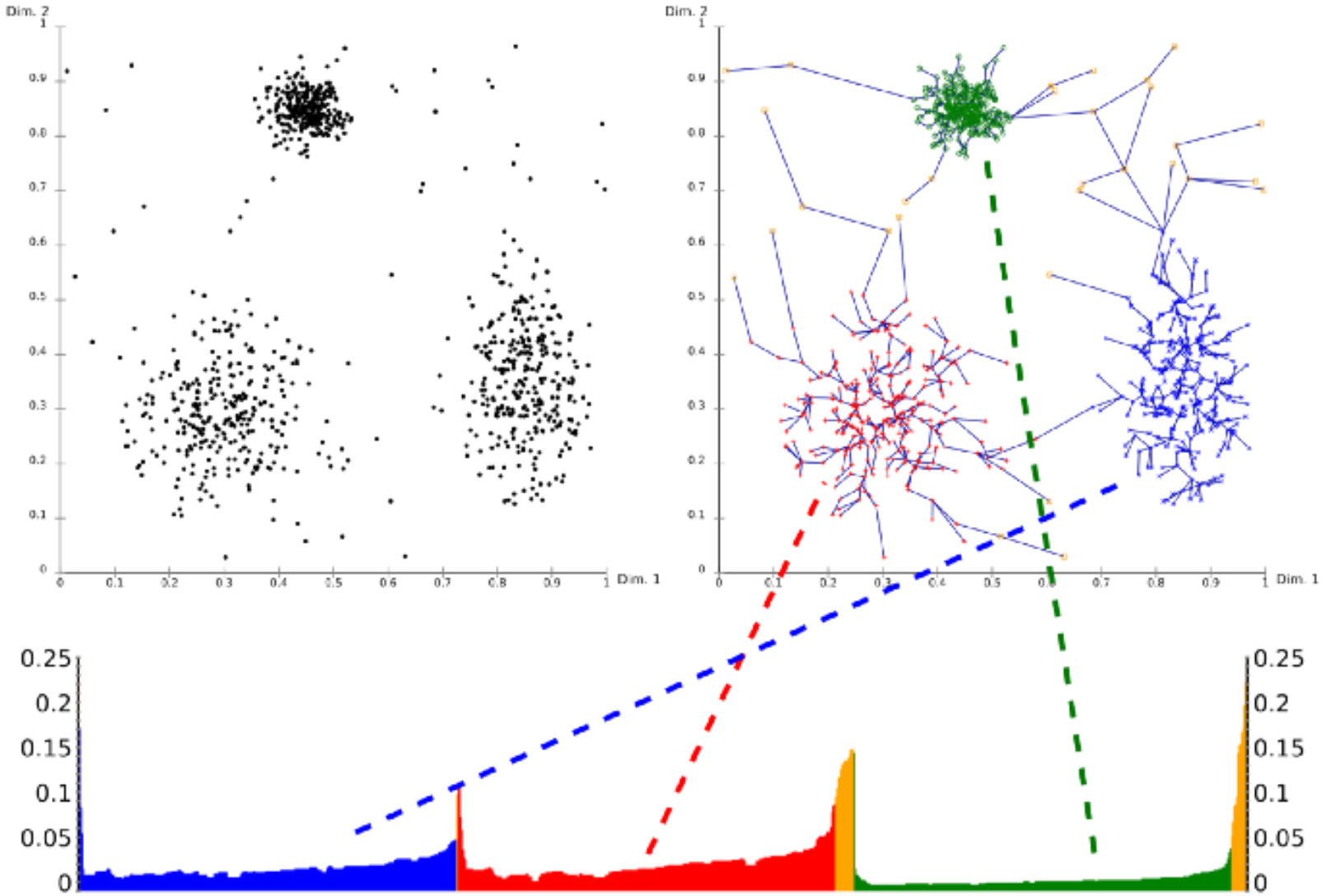
OPTICS Pseudo-Code

- Function *update(N, p, Q)*
  - Calculate the core distance for *p*
  - For each neighbor *q in N* (update the reachability)
    - If *q* is not processed
      - *new_rd* = reachability distance between *p* and *q*
    - If *q is not in Q*
      - *Q.insert(q, new_rd)*
    - Else
      - *If new_rd < q.rd*
        - *Q.move_up(q, new_rd)*

OPTICS Output

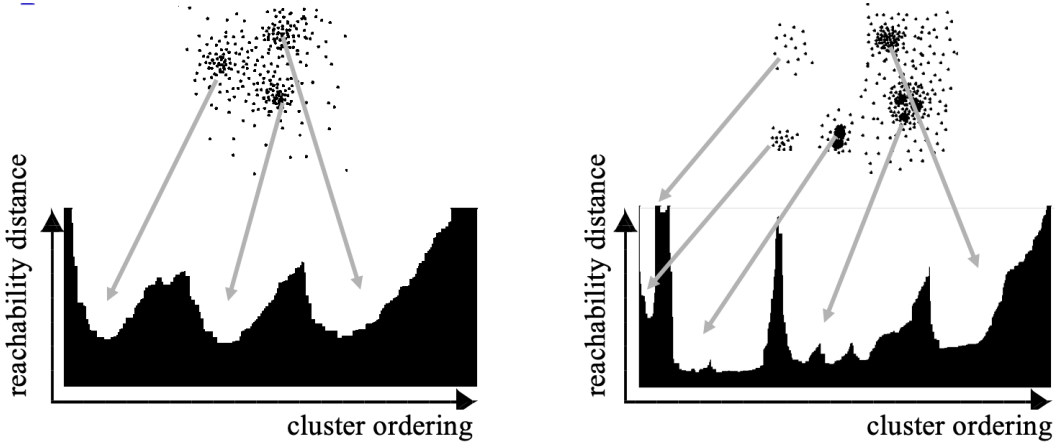
- OPTICS outputs the points in a particular ordering, annotated with their smallest reachability distance.
- A reachability-plot (a special kind of dendrogram), the hierarchical structure of the clusters can be obtained easily.
- x-axis: the ordering of the points as processed by OPTICS
- y-axis: the reachability distance
- Points belonging to a cluster have a low reachability distance to their nearest neighbor, the clusters show up as valleys in the reachability plot. The deeper the valley, the denser the cluster.

OPTICS Output

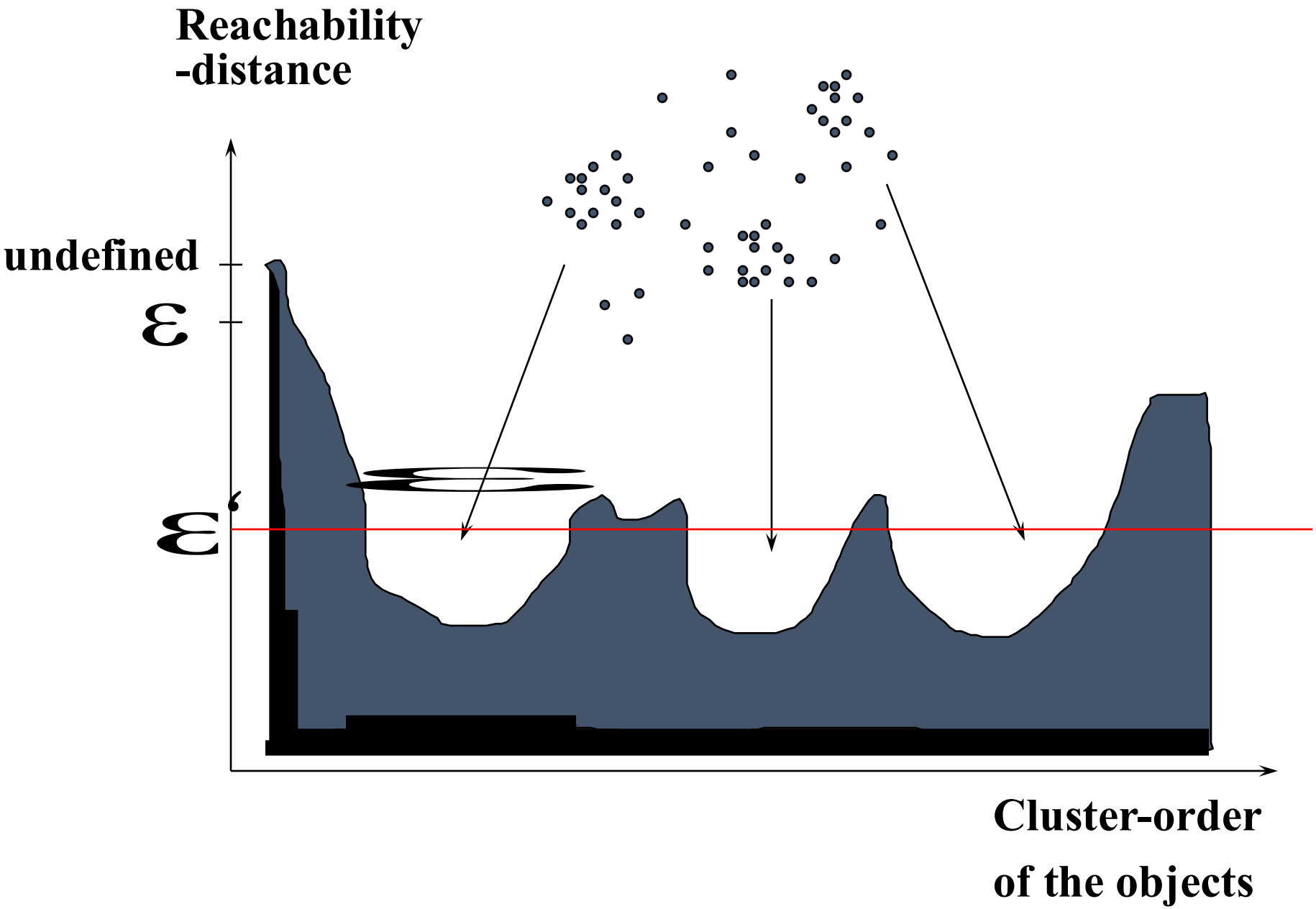


OPTICS Output

- Clusters are extracted
  1. by selecting a range on the x-axis after visual inspection,
  2. by selecting a threshold on the y-axis
  3. by different algorithms that try to detect the valleys by steepness, knee detection, or local maxima. Clustering obtained this way usually are hierarchical, and cannot be achieved by a single DBSCAN run.



https://scikit-learn.org/stable/auto_examples/cluster/plot_optics.html#sphx-glr-auto-examples-cluster-plot-optics-py


Reachability
-distance
undefined
ε
ε'
Cluster-order
of the objects

OPTICS: The Radius Parameter

- Both core-distance and reachability-distance are undefined if no sufficiently dense cluster (w.r.t. ε) is available.
- Given a sufficiently large ε, this never happens, but then every ε-neighborhood query returns the entire database.
- Hence, the ε parameter is required to cut off the density of clusters that are no longer interesting, and to speed up the algorithm.
- The parameter ε is, strictly speaking, not necessary.
- It can simply be set to the maximum possible value.
- When a spatial index is available, however, it does play a practical role with regards to complexity.
- OPTICS abstracts from DBSCAN by removing this parameter, at least to the extent of only having to give the maximum value.

DBSCAN Evolution

HDBSCAN

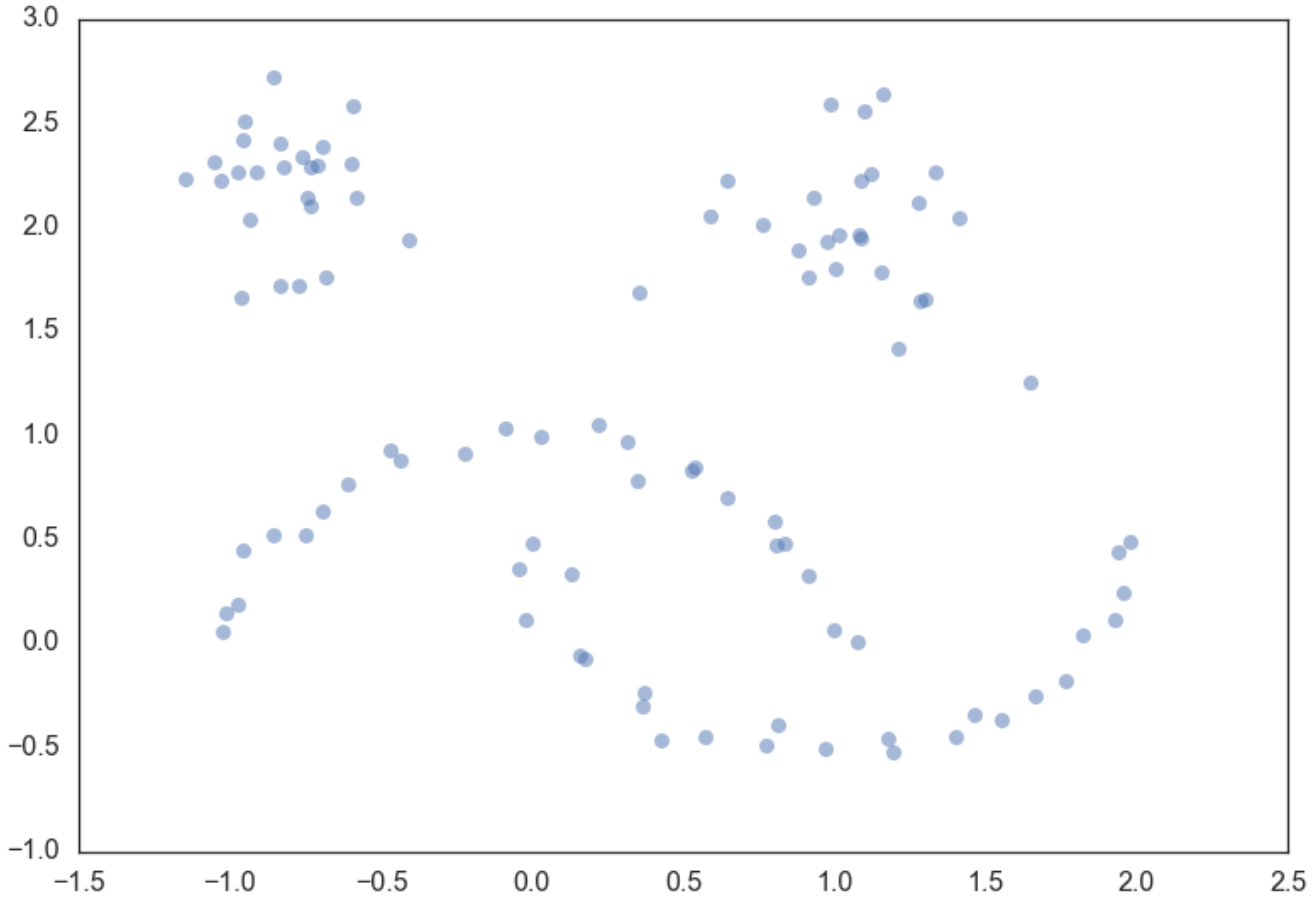
HDBSCAN

- HDBSCAN extends DBSCAN by converting it into a hierarchical clustering algorithm, but it also extracts a flat clustering.
- HDBSCAN bypass the choice of the Eps parameter
- HDBSCAN scans all possible solution with all values of Eps

HDBSCAN Main Steps

1. Transform the space according to the density/sparsity
2. Build the minimum spanning tree of the distance weighted graph
3. Construct a cluster hierarchy of connected components.
4. Condense the cluster hierarchy based on minimum cluster size.
5. Extract the stable clusters from the condensed tree.

How HDBSCAN Works



Step 1: Transform The Space

- **Goal**: Prepare the data for a single linkage clustering (real data is noisy and single linkage is not robust!)
- **Idea**: Push sparse points away from the rest of the data before clustering
- How do we evaluate density ?
  - Need an inexpensive density estimate ⇒ k-NN is the simplest
  - Call it the **core distance** for parameters k and point x_i, $core_k(x_i)$

Recall

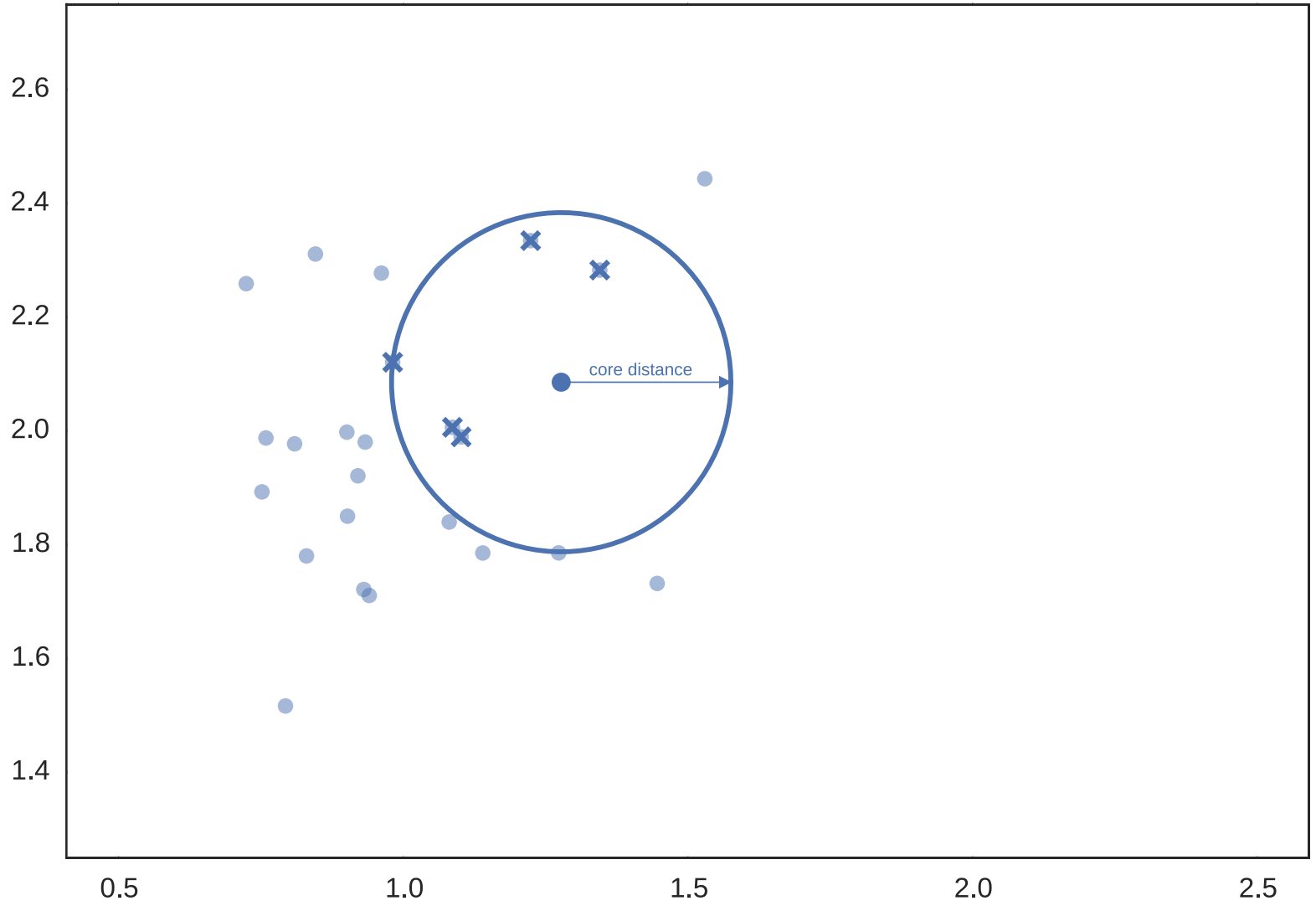
Core point. A point *p* is a core point if at least MinPts points are found within its ε-neighborhood.

Core Distance. It is the **minimum** value of radius required to classify a given point as a core point.
If the given point is not a Core point, then it's Core Distance is undefined.

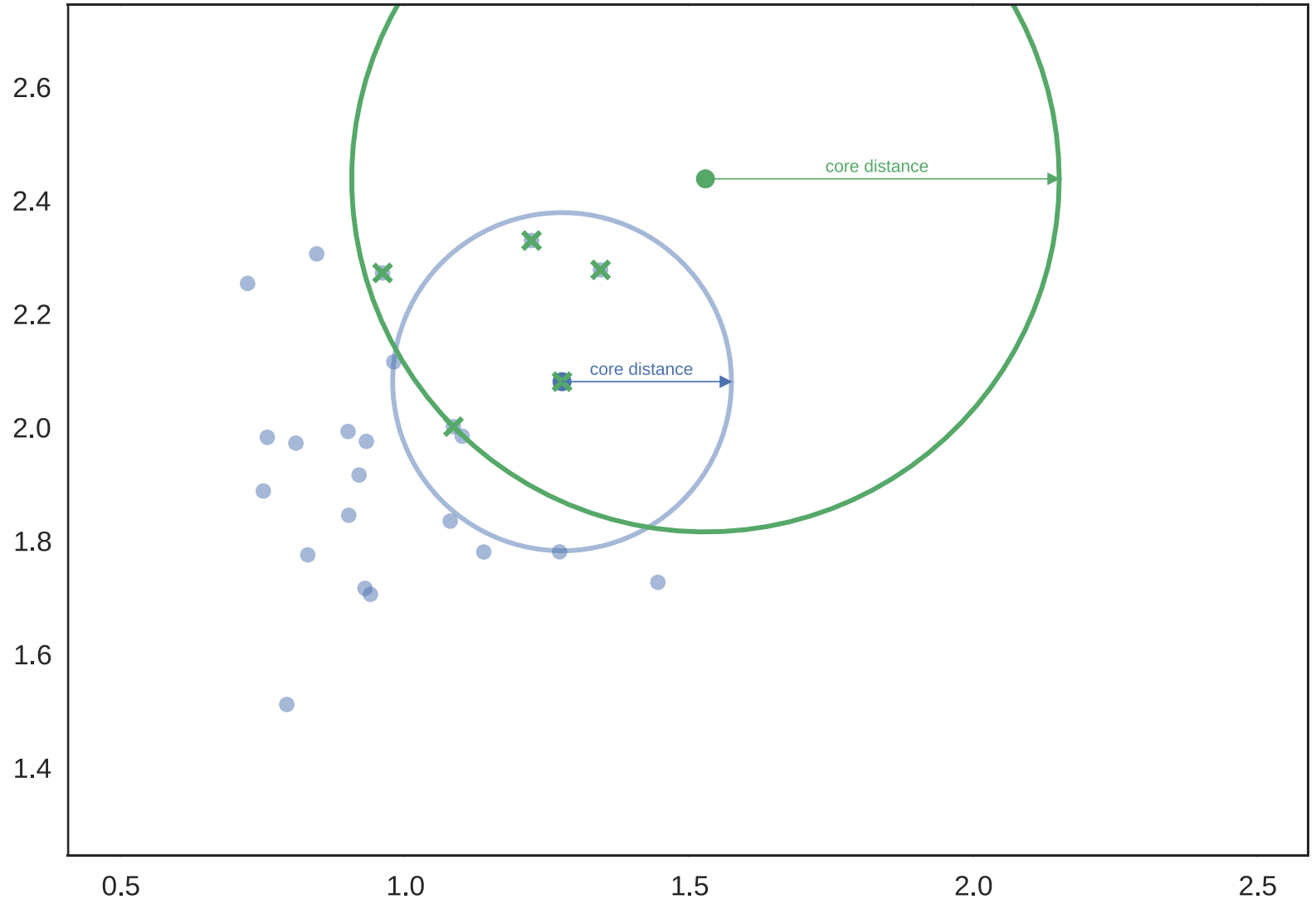
Step 1: Mutual Reachability Distance

- **Goal**: We need a way to spread apart points with low density (correspondingly high core distance).
- $d_{mreach\text{-}k}(x_i, x_j) = max\{core_k(x_i), core_k(x_j), d(x_i, x_j)\}$
- **Objectives:** connect points that are
  - close enough to each other : $d(x_i, x_j)$
  - in a dense enough region : $core_k(x_i)$
- With this metric dense points (with low core distance) remain the same distance from each other, but sparser points are pushed away to be at least their core distance away from any other point.

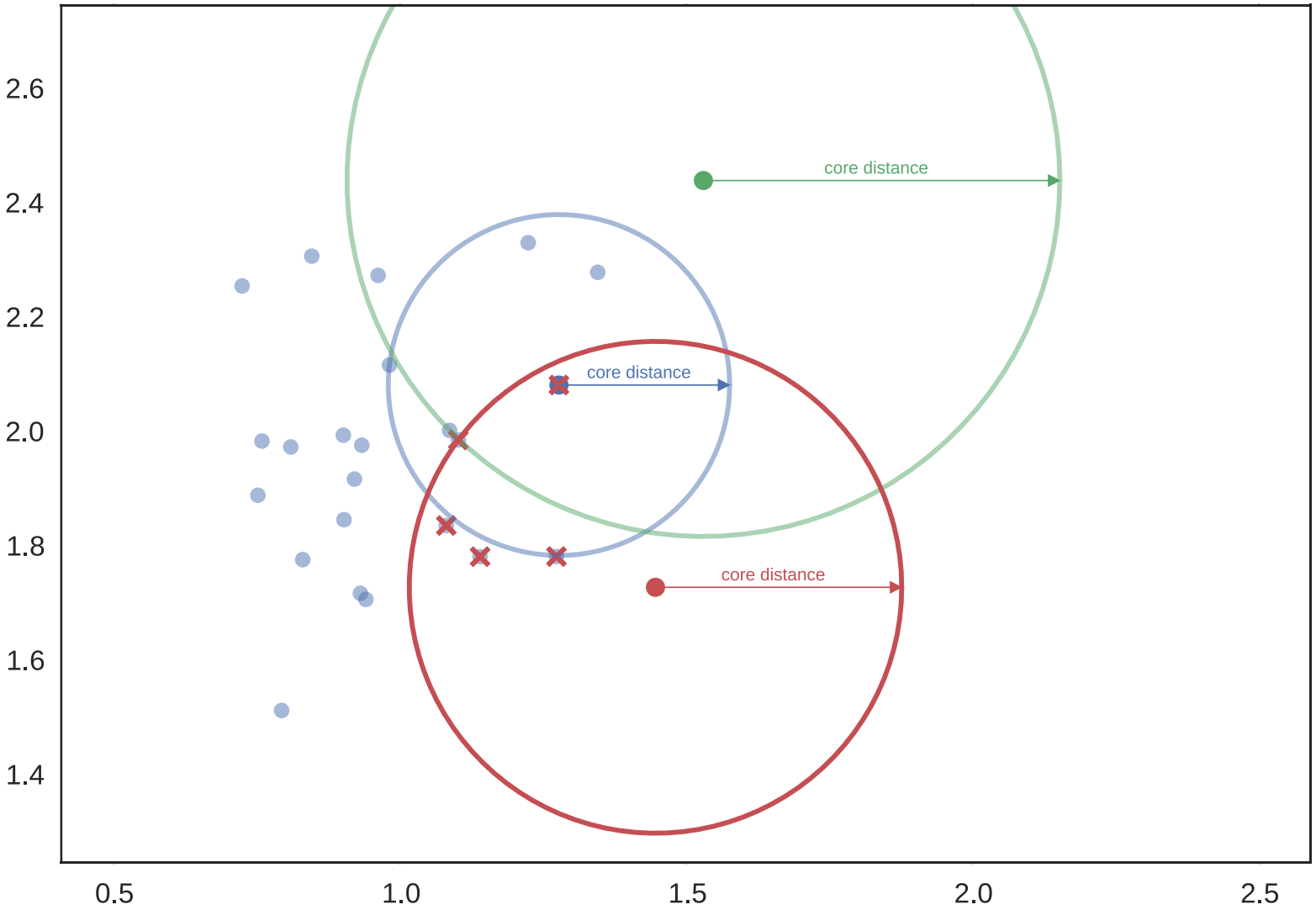
Step 1: Mutual Reachability Distance



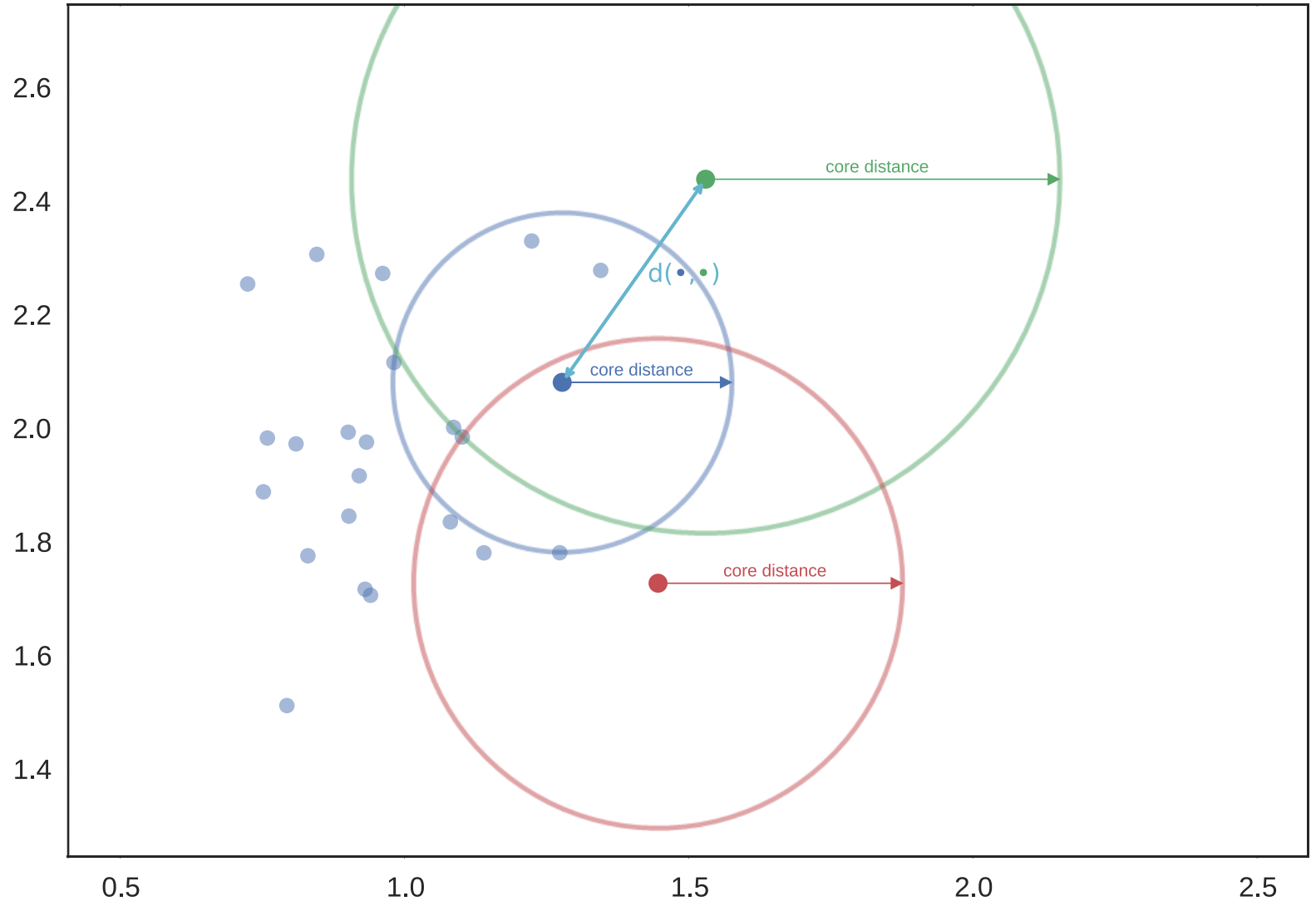
Step 1: Mutual Reachability Distance



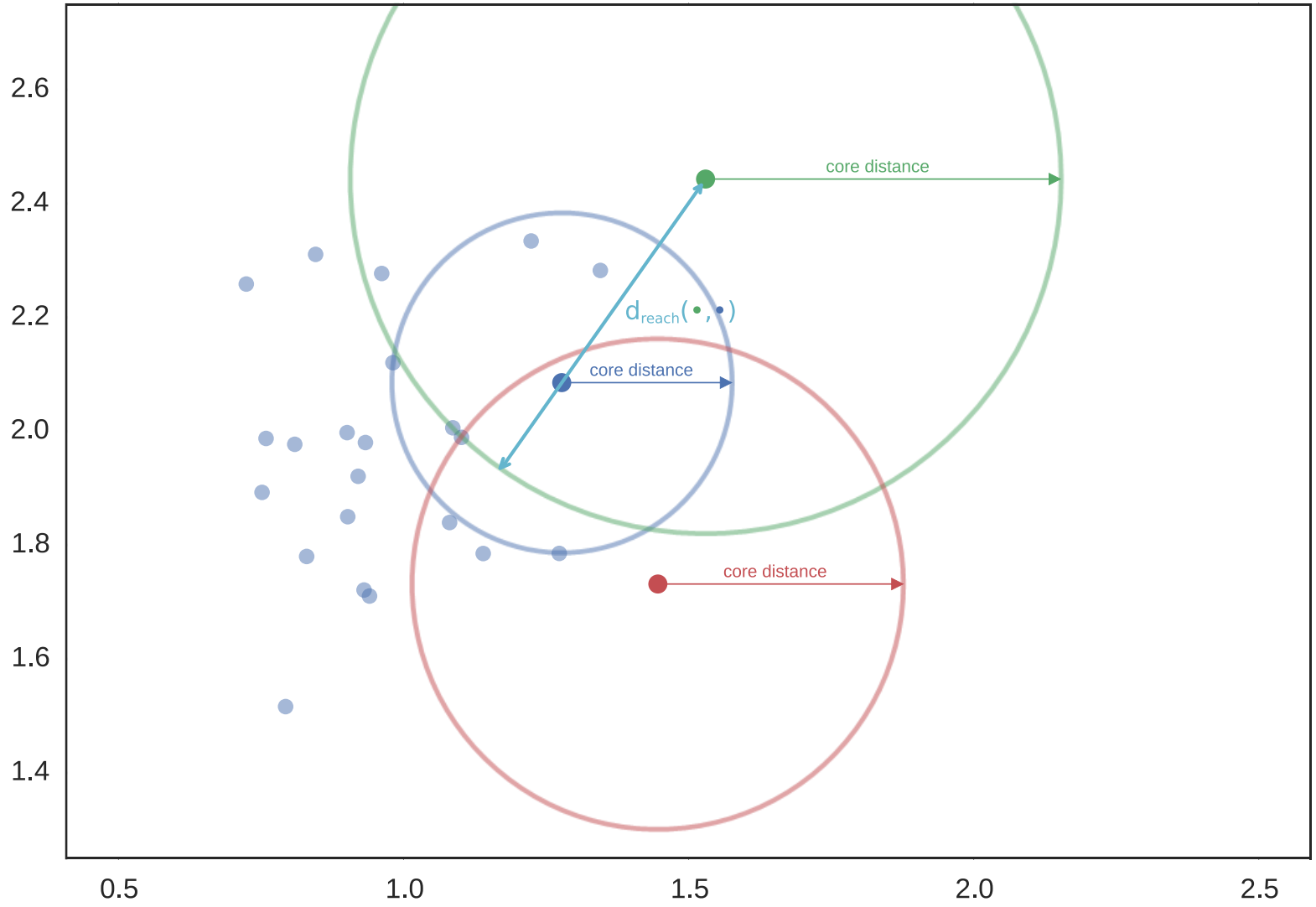
Step 1: Mutual Reachability Distance



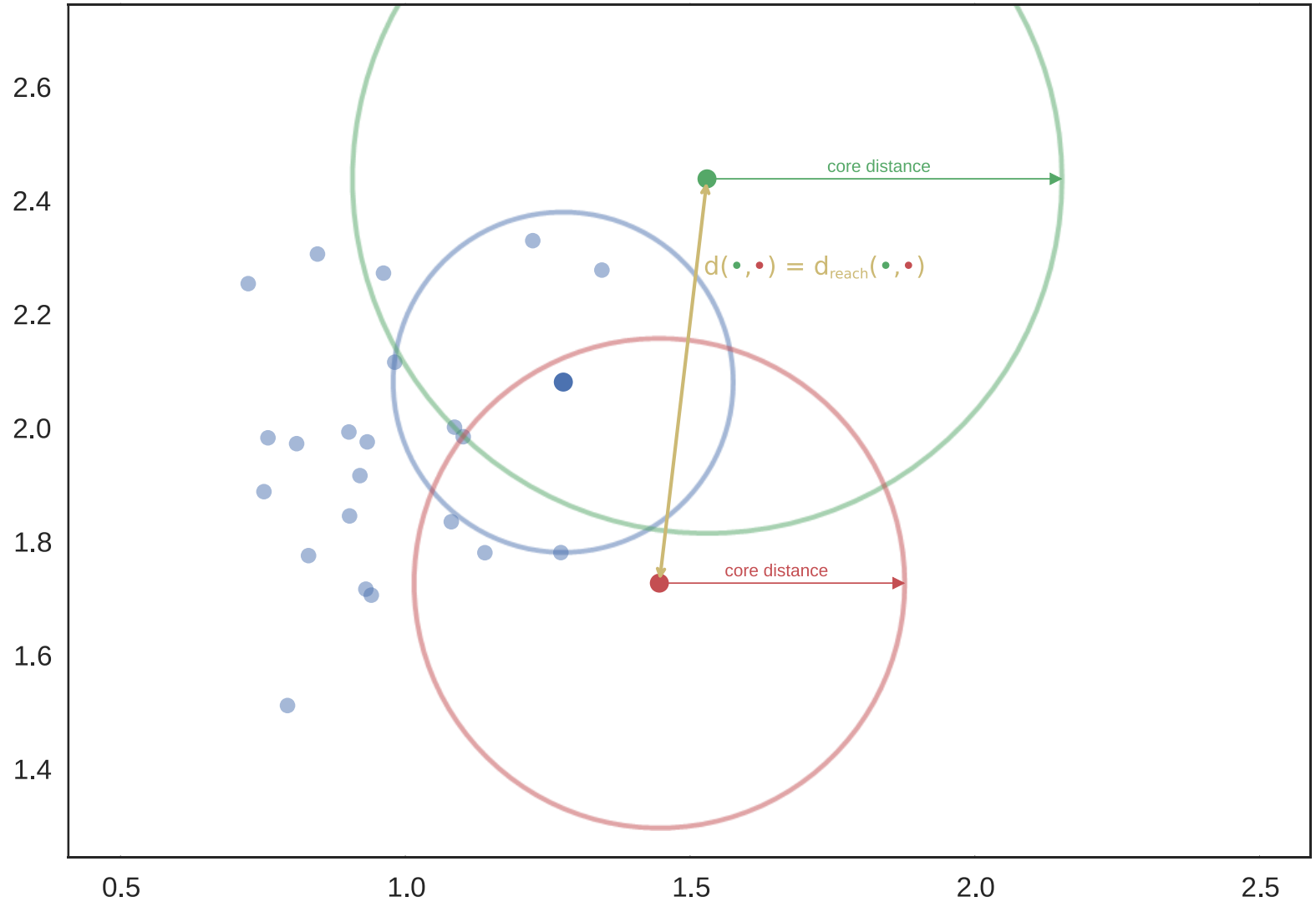
Step 1: Mutual Reachability Distance



Step 1: Mutual Reachability Distance



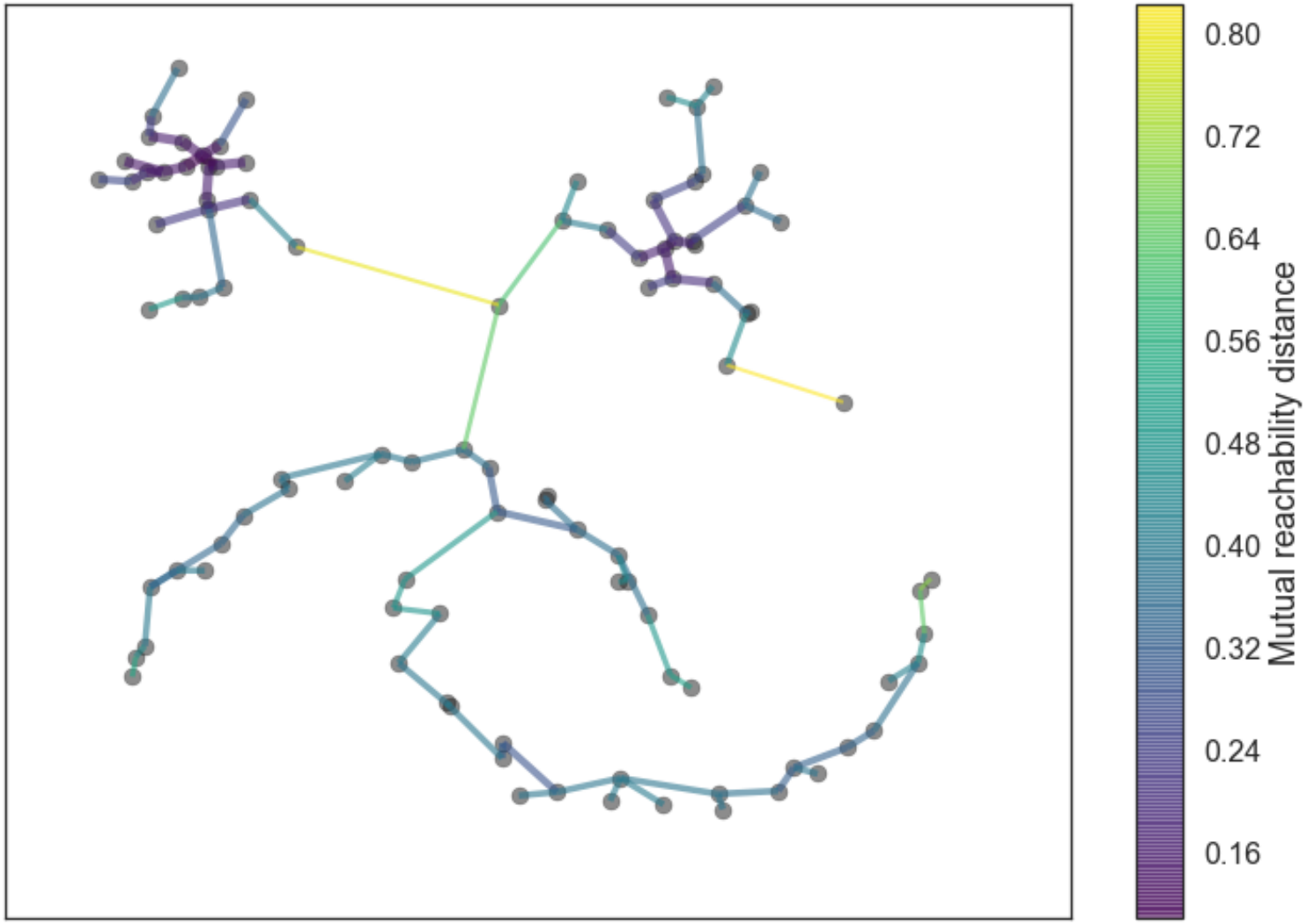
Step 1: Mutual Reachability Distance



Step 2 : The Minimum Spanning Tree

- **Goal**: Prepare the data for clustering using d_{mreach}
- Ideas:
  - Construct a graph that connects all points
  - Points are the vertices and the edges are weighted by d_{mreach}
  - Start disconnecting points by lowering a threshold
  - Among n^2 possible edges identify with the **minimum spanning tree** a threshold such that there is no lower weight edge that could connect the components
  - **Prim's algorithm:** build the tree one edge at a time, always adding the lowest weight edge that connects the current tree to a vertex not yet in the tree

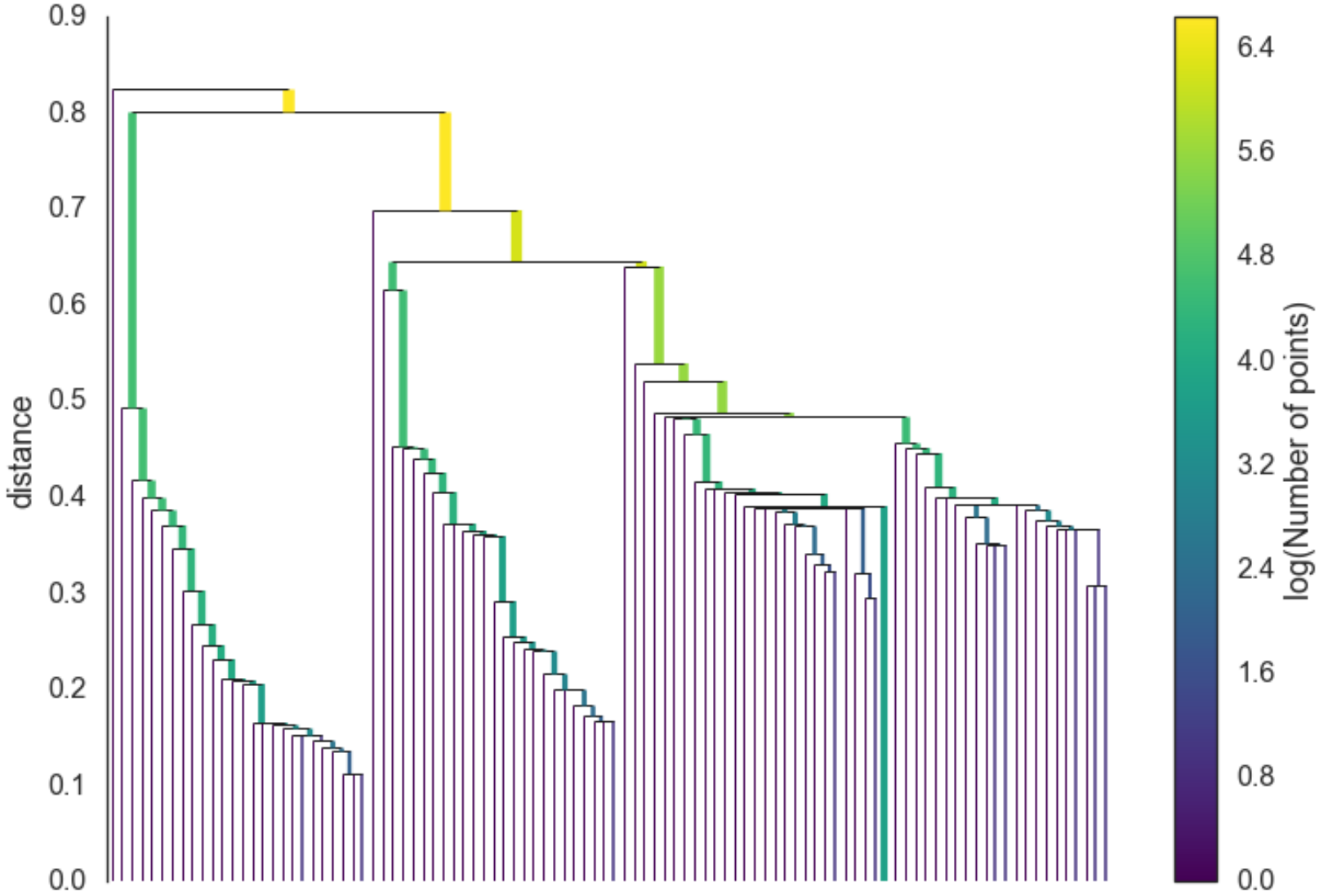
Step 2 : The Minimum Spanning Tree



Step 3: Build the Cluster Hierarchy

- **Goal**: Given the minimal spanning tree, the next step is to convert that into the hierarchy of connected components.
- **Idea**: Sort the edges of the tree by distance (in increasing order) and then iterate through, creating a new merged cluster for each edge, i.e., run single linkage hierarchical clustering algorithm

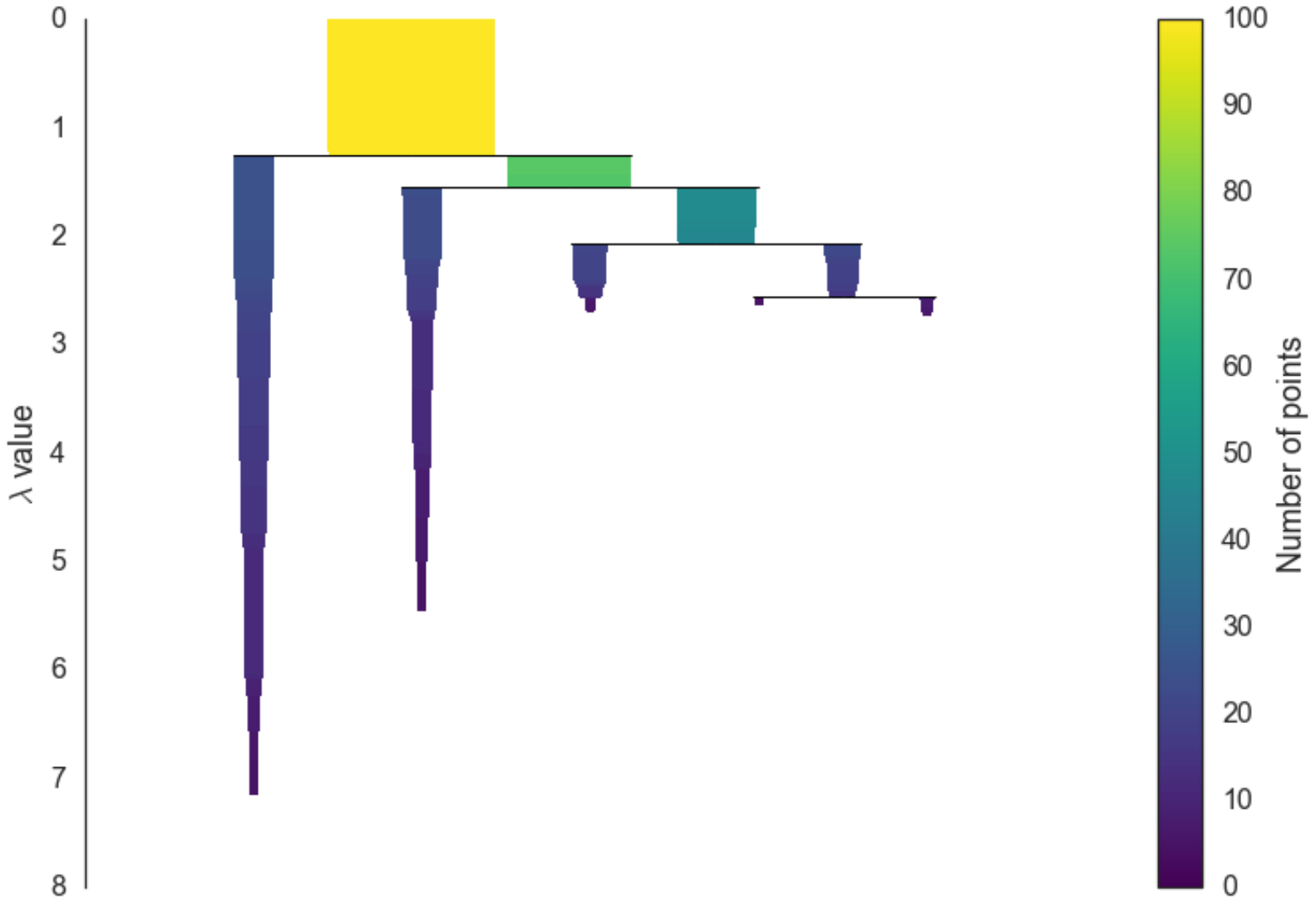
Step 3: Build the Cluster Hierarchy



Step 4 : Condense the Cluster Tree

- **Goal**: Condensing the large and complicated cluster hierarchy into a smaller tree with a little more data attached to each node.
- **Idea**: Use a notion of **minimum cluster size** which we take as a parameter to HDBSCAN.
  - Walk through the hierarchy top down and at each split ask if one of the new clusters created by the split has fewer points than the minimum cluster size.
  - If it is the case, declare it to be ‘points falling out of a cluster’ and have the larger cluster retain the cluster identity of the parent, marking down which points ‘fell out of the cluster’, i.e., noise points, and at what distance value that happened.
  - On the other hand, if the split is into two clusters each at least as large as the minimum cluster size then we consider that a true cluster split and let that split persist in the tree.

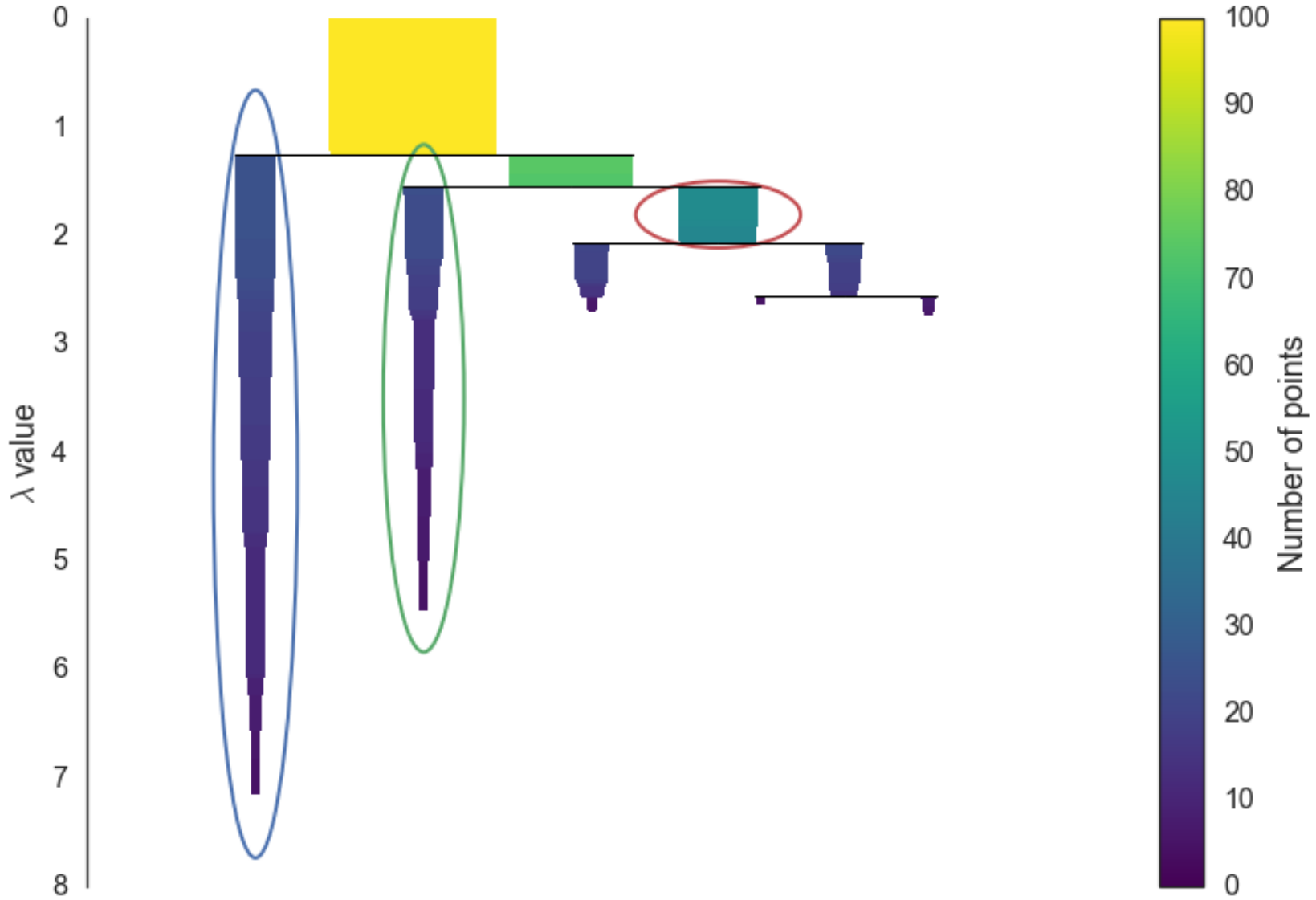
Step 4 : Condense the Cluster Tree



Step 5: Extract the Stable Clusters

- **Goal**: Choose clusters that persist and have a longer lifetime (short clusters are probably artifacts of single linkage)
- **Idea**: Looking at the previous plot we could say that we want to choose those clusters that have the greatest area of ink in the plot
- **Requirement**: if you select a cluster, then you cannot select any cluster that is a descendant of it

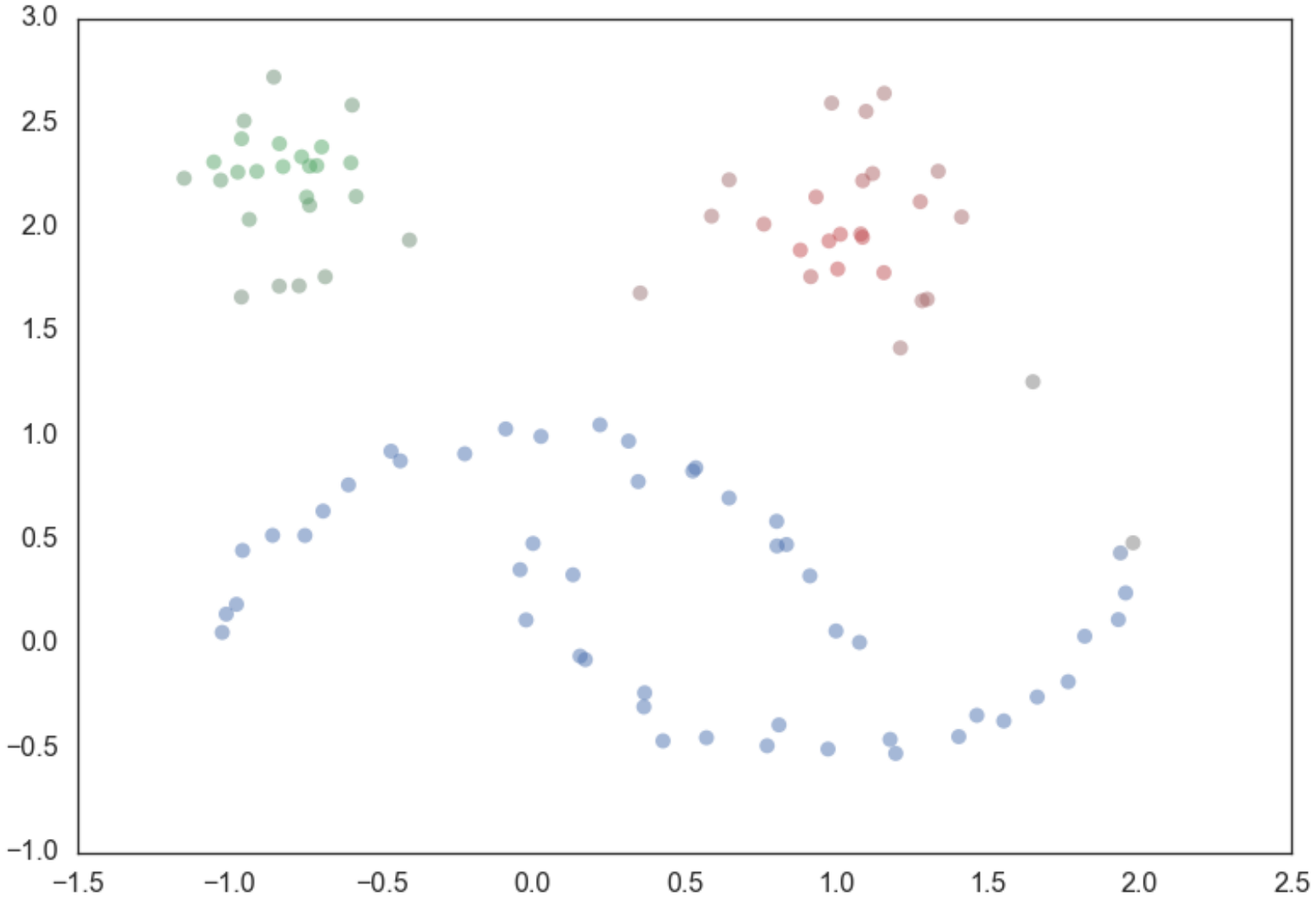
Step 5: Extract the Stable Clusters



Step 5: Extract the Stable Clusters (details)

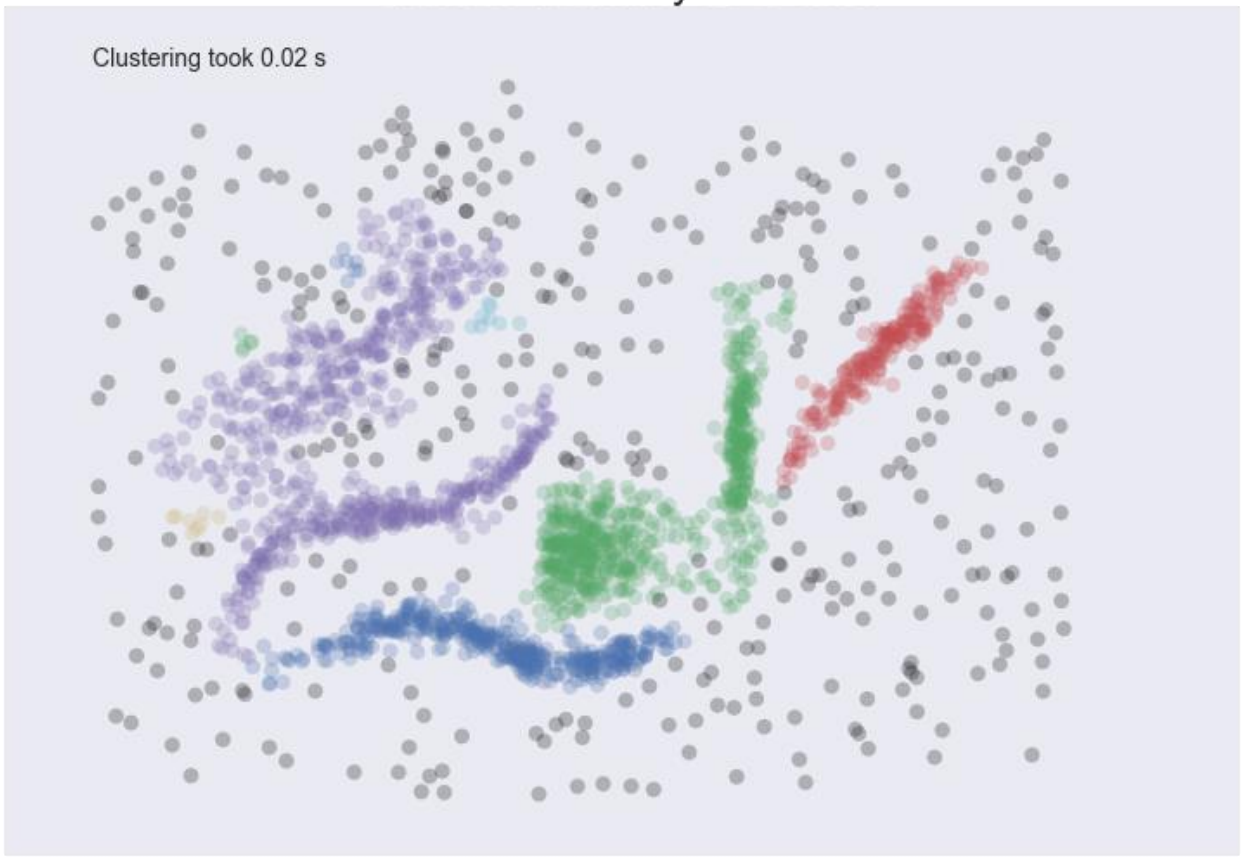
- Use $\lambda = 1/\text{distance}$
- For a given cluster we have λ_{start}, λ_{end} to identify the cluster boundaries
- For a given cluster, the value $\lambda_p \in [\lambda_{start}, \lambda_{end}]$ defines at which value a point "fell out of the cluster"
- Compute the **stability** of a cluster as $\sum_{p \in C} (\lambda_p - \lambda_{start})$
- Declare all leaf nodes to be clusters.
- Now work bottom up through the tree.
  - If the sum of the stabilities of the child clusters is greater than the stability of the cluster, then set the cluster stability to be the sum of the child stabilities.
  - If, on the other hand, the cluster's stability is greater than the sum of its children then we declare the cluster to be a selected cluster and unselect all its descendants.
  - Once at the root node all the set of selected clusters is the flat clustering returned.

HDBSCAN Result

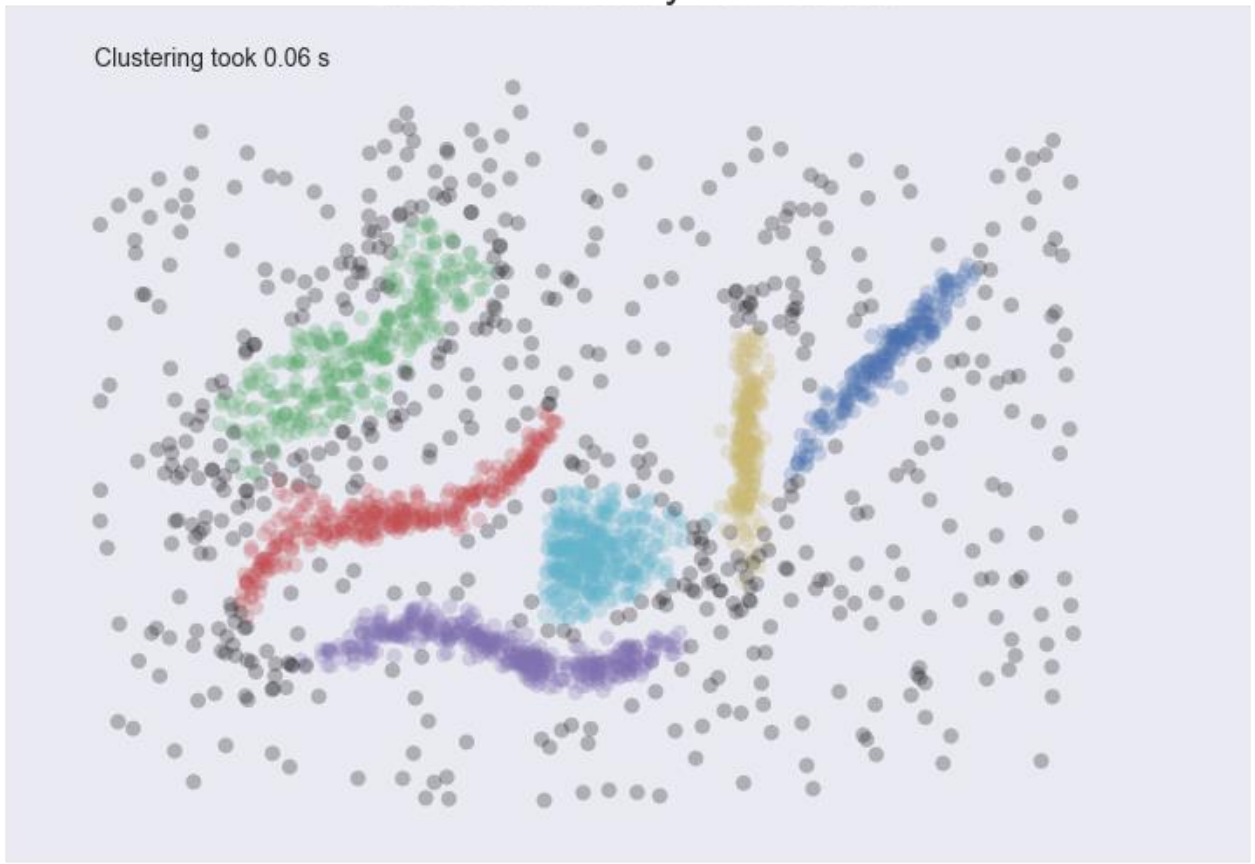


DBSCAN vs HDBSCAN

Clusters found by DBSCAN

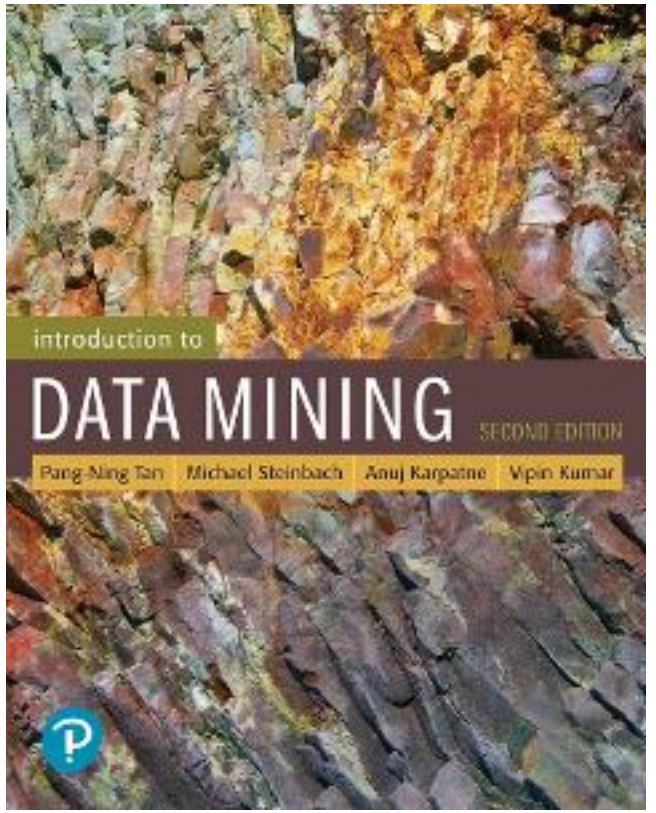


Clusters found by HDBSCAN



References

- Clustering. Chapter 7. Introduction to Data Mining.
- Mihael Ankerst; Markus M. Breunig; Hans-Peter Kriegel; Jörg Sander (1999). OPTICS: Ordering Points To Identify the Clustering Structure.



clustering_iris

November 11, 2025

[173]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[174]:
```
#scaling, normalization
from sklearn.preprocessing import StandardScaler, MinMaxScaler

#kmeans, dbscan, hierarchical (sklearn)
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
#evaluation
from sklearn.metrics import silhouette_score
#import dataset
from sklearn.datasets import load_iris

#distance matrix (dbscan elbow, hierarchical)
from scipy.spatial.distance import pdist, squareform

# hierarchical (scipy)
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
```

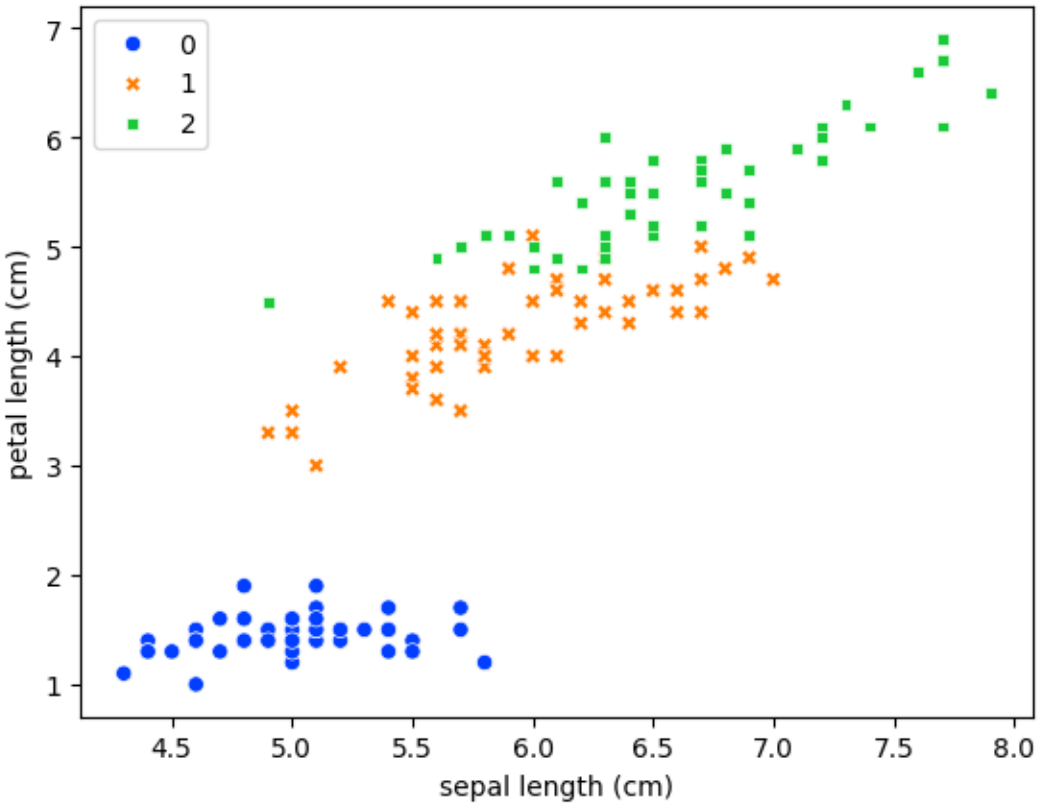
[175]:
```
frame = load_iris(as_frame=True)
df = frame['data']
X = df.values
y = np.array(frame['target'])

y_map = {0: "setosa", 1: "versicolor", 2: "virginica"}
y_mapped = pd.DataFrame(y).iloc[:,0].map(y_map)

df.head()
```

[175]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

[175]:

[176]:
```
sns.scatterplot(data=df,
                x="sepal length (cm)",
                y="petal length (cm)",
                hue=y,
                style=y,
                palette="bright")
plt.show()
```



0.1 Sklearn routine

- Initialize method
- fit
- transfrom/predict

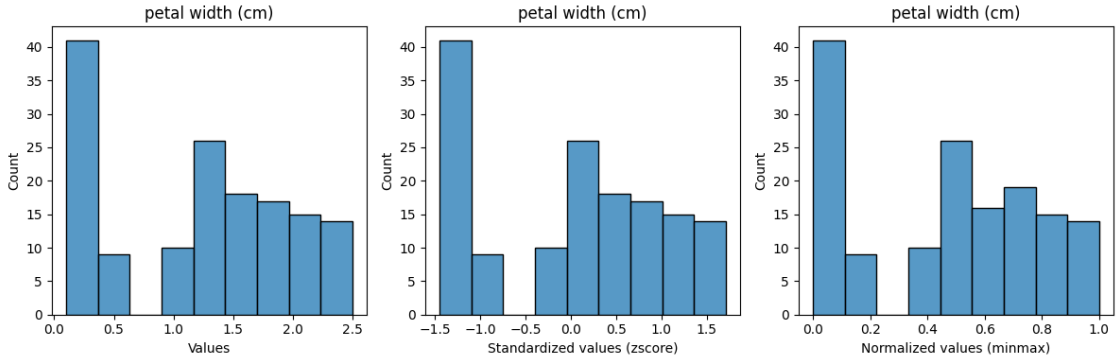
0.1.1 Normalizations

[177]:
```
# z-score, fit and then transform

#Standardize features by removing the mean and scaling to unit variance.
```

```
# z = (x - u) / s

# u is the mean and s the standard deviation

scaler = StandardScaler()
scaler.fit(X)
X_scal = scaler.transform(X)
```

[178]:
```
# min-max, fit and trasform directly
scaler = MinMaxScaler()
X_minmax = scaler.fit_transform(X)
```

[179]:
```
i = 3 # column index
fig, axs = plt.subplots(1,3, figsize=(12, 4)) # 1 row, 3 columns

sns.histplot(X[:,i], ax=axs[0]).set(title=df.columns[i])
axs[0].set(xlabel='Values')

sns.histplot(X_scal[:,i], ax=axs[1]).set(title=df.columns[i])
axs[1].set(xlabel='Standardized values (zscore)')

sns.histplot(X_minmax[:,i], ax=axs[2]).set(title=df.columns[i])
axs[2].set(xlabel='Normalized values (minmax)')

plt.tight_layout() # Adjust the padding between and around subplots
```



0.1.2 Kmeans

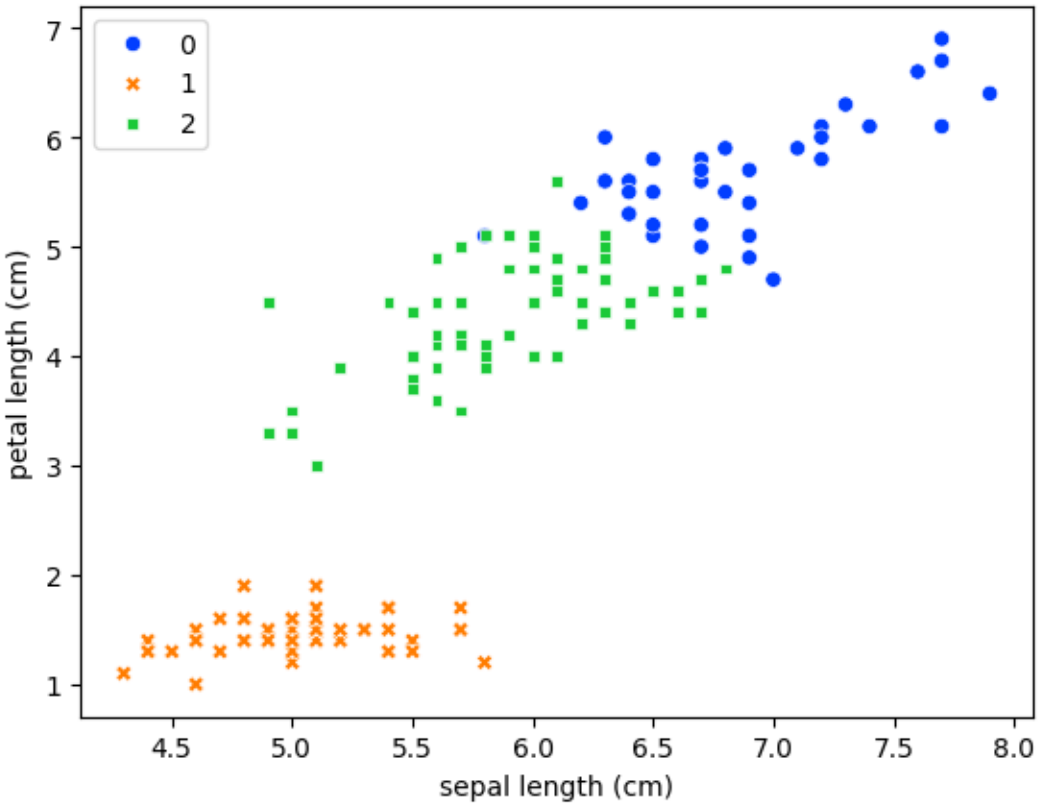
[180]:
```
kmeans = KMeans(n_clusters=3, n_init=10, max_iter=100, random_state=94)
kmeans.fit(X_minmax)
```

[180]:
```
KMeans(max_iter=100, n_clusters=3, n_init=10, random_state=94)
```

[181]:
```
kmeans.labels_
```

[181]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0,
       0, 0, 0, 2, 0, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 0, 0,
       0, 0, 2, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2], dtype=int32)
```

[182]:
```
sns.scatterplot(data=df,
                x="sepal length (cm)",
                y="petal length (cm)",
                hue=kmeans.labels_,
                style=kmeans.labels_,
                palette="bright")
plt.show()
```



[ ]:
```
# ndarray of shape (n_clusters, n_features)
kmeans.cluster_centers_
```

[ ]:
```
array([[0.70726496, 0.4508547 , 0.79704476, 0.82478632],
       [0.19611111, 0.595     , 0.07830508, 0.06083333],
```

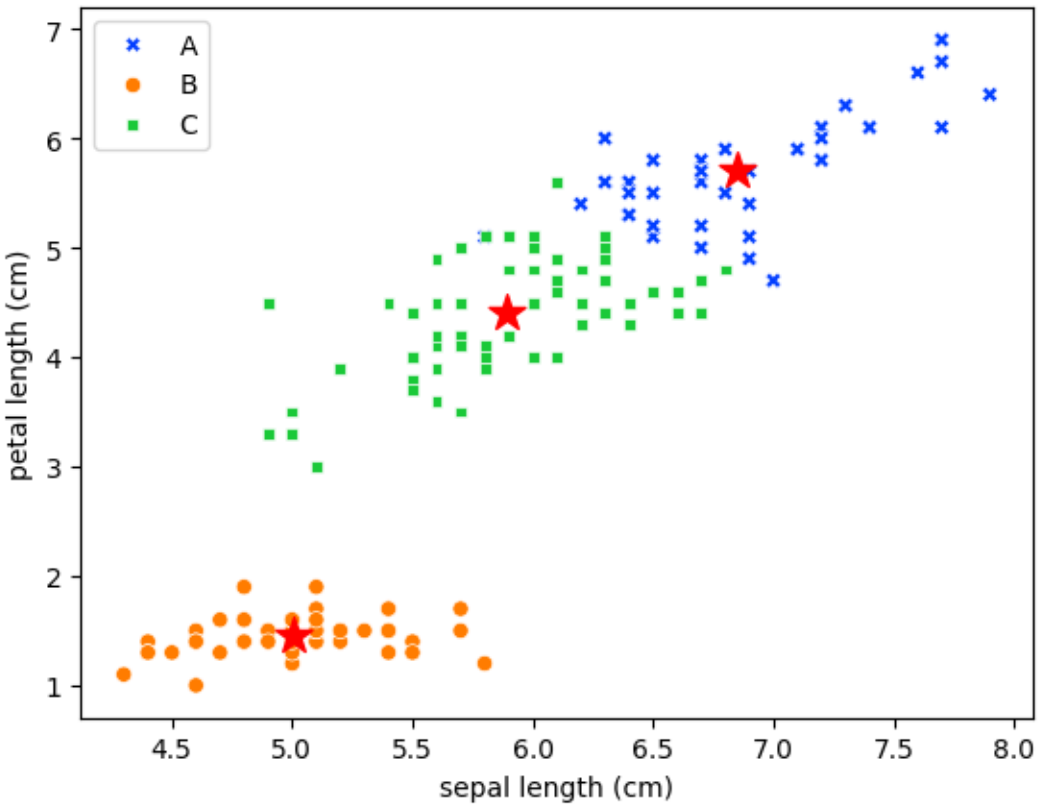
```
       [0.44125683, 0.30737705, 0.57571548, 0.54918033]])
```

[ ]:
```
centers = scaler.inverse_transform(kmeans.cluster_centers_)
centers
```

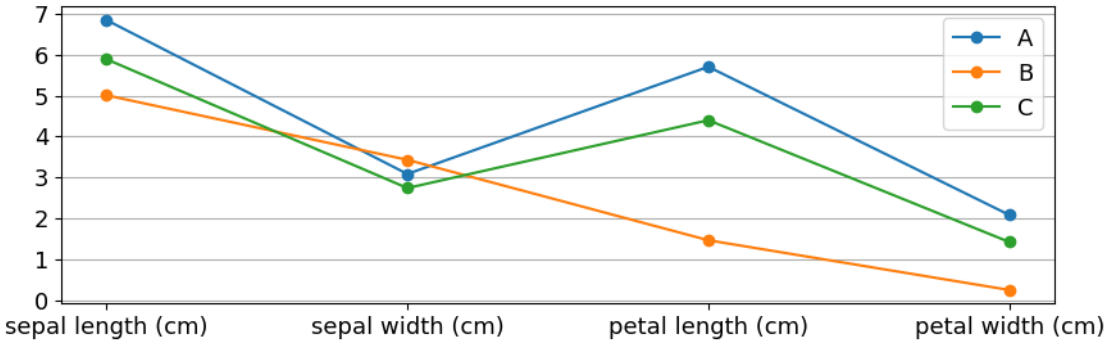
[ ]:
```
array([[6.84615385, 3.08205128, 5.7025641 , 2.07948718],
       [5.006     , 3.428     , 1.462     , 0.246     ],
       [5.88852459, 2.73770492, 4.39672131, 1.41803279]])
```

[ ]:
```
df['kmeans_labels'] = kmeans.labels_
df['kmeans_labels'] = df['kmeans_labels'].map({0:"A", 1: "B", 2: "C"})
```

[ ]:
```
sns.scatterplot(data=df,
                x="sepal length (cm)",
                y="petal length (cm)",
                hue='kmeans_labels',
                style='kmeans_labels',
                palette="bright",
                hue_order=["A", "B", "C"]
               )
plt.legend()
plt.scatter(centers[:,0], centers[:,2], c='red', marker='*', s=200)
plt.show()
```

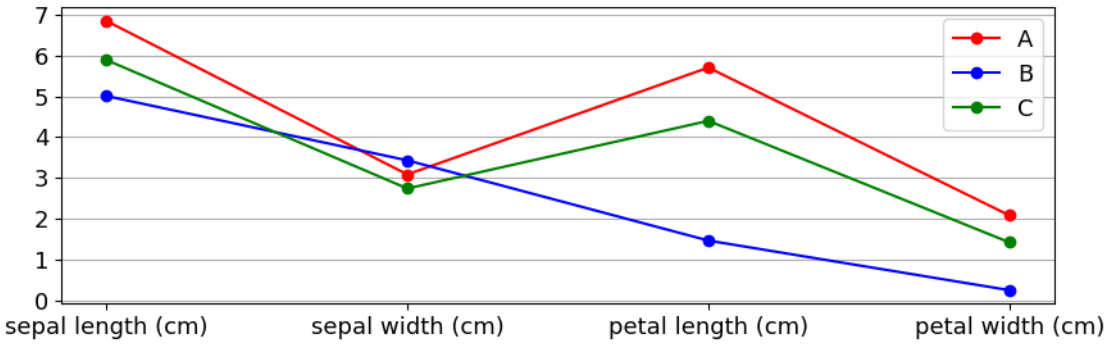


[ ]:
```
plt.figure(figsize=(10, 3))

clust_name = ['A', 'B', 'C']

for i in range(len(centers)):
    plt.plot(centers[i], marker='o', label=clust_name[i])

plt.xticks(range(0,len(df.columns) - 1), df.columns[:-1], fontsize=13)
plt.yticks(fontsize=13)

plt.legend(fontsize=13, loc='best')
plt.grid(axis='y')
```

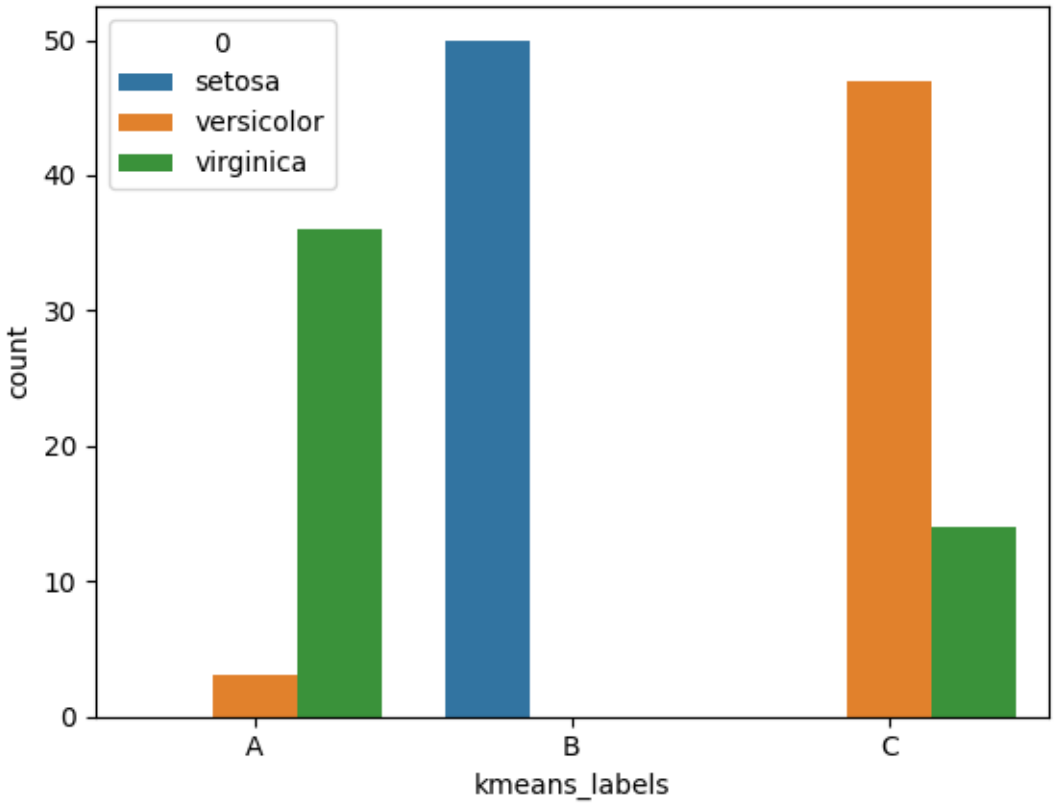


[ ]:
```
plt.figure(figsize=(10, 3))

clust_name = ['A', 'B', 'C']

# color customization (fixed to 3 colors, it would not work if the clusters␣
 ↪would be 5)
colors = {
    'A': 'r',
    'B': 'b',
    'C': 'g',
}

for i in range(len(centers)):
    plt.plot(centers[i], marker='o', label=clust_name[i],␣
 ↪color=colors[clust_name[i]])

plt.xticks(range(0,len(df.columns) - 1), df.columns[:-1], fontsize=13)
plt.yticks(fontsize=13)
```

```
plt.legend(fontsize=13, loc='best')
plt.grid(axis='y')
```



[ ]:
```
sns.countplot(data=df, x='kmeans_labels', hue=y_mapped, order=["A", "B", "C"])
plt.show()
```



[ ]:
```
y_mapped
```

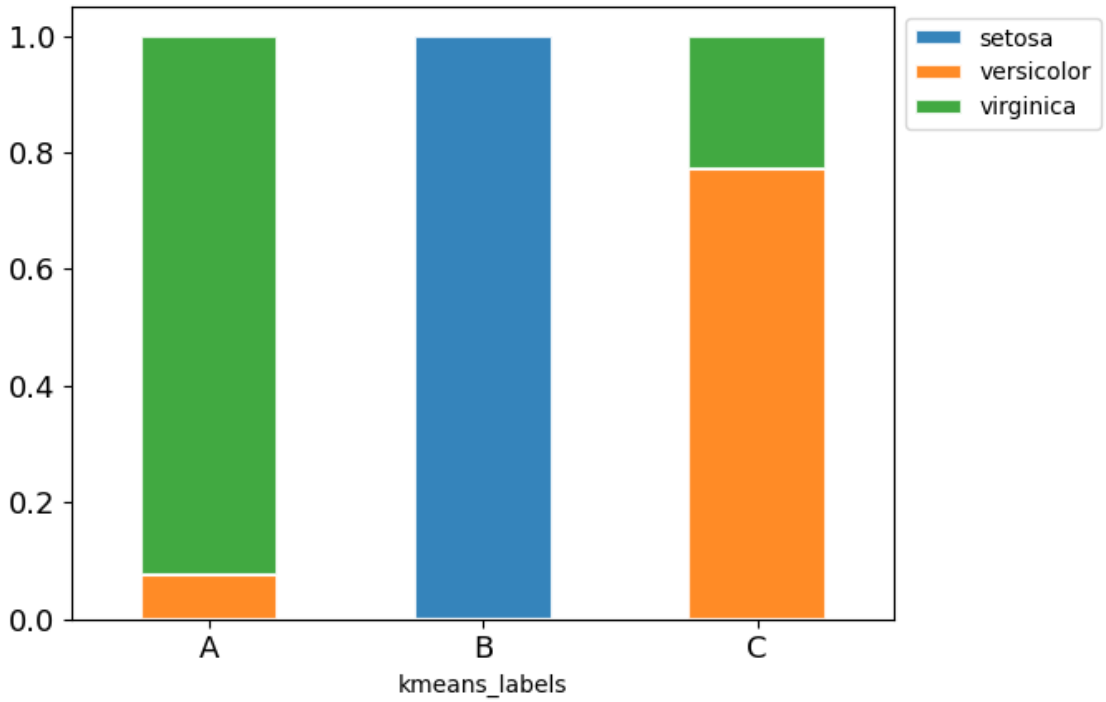
[ ]:
```
0         setosa
1         setosa
2         setosa
3         setosa
4         setosa
          …
145    virginica
146    virginica
147    virginica
148    virginica
149    virginica
Name: 0, Length: 150, dtype: object
```

[ ]:
```
y_mapped.groupby(df['kmeans_labels']).value_counts(normalize=True)
```

[ ]:
```
kmeans_labels  0
A              virginica     0.923077
               versicolor    0.076923
B              setosa        1.000000
C              versicolor    0.770492
               virginica     0.229508
Name: proportion, dtype: float64
```

[ ]:
```

```

[ ]:
```
bar_pl = y_mapped.groupby(df['kmeans_labels']).value_counts(normalize=True).
↪unstack(1)
bar_pl.plot(kind='bar', stacked=True, alpha=0.9, edgecolor='white', linewidth=1.
↪5)

plt.xticks(range(0, len(clust_name)), clust_name, fontsize=13, rotation=0)
plt.yticks(fontsize=13)
plt.legend(bbox_to_anchor=(1,1))

plt.show()
```



[ ]:
```
np.unique(kmeans.labels_, return_counts=True)
```

[ ]:
```
(array([0, 1, 2], dtype=int32), array([39, 50, 61]))
```

[ ]:
```
print('SSE', kmeans.inertia_)
print('Silhouette', silhouette_score(X_minmax, kmeans.labels_))
```

```
SSE 6.982216473785234
Silhouette 0.5047687565398589
```

[ ]:
```
# how do we select the number of clusters k?
```

[ ]:
```
%%time
sse_list = []

for k in range(2, 51):
    kmeans = KMeans(n_clusters=k, n_init=10, max_iter=100, random_state = 42)
    kmeans.fit(X_minmax)
    sse_list.append(kmeans.inertia_)
```

```
CPU times: user 4.39 s, sys: 19 ms, total: 4.41 s
Wall time: 6.84 s
```

[ ]:
```
sse_list
```

```
[ ]: [12.12779075053819,
      6.982216473785234,
      5.516933472040375,
      4.5809486401172945,
      3.97642424820984,
      3.473622320733247,
      3.1456415905967785,
      2.814016644149028,
      2.5608613580326587,
      2.2987533401137896,
      2.164318065645078,
      2.0022593438158807,
      1.9136218419482227,
      1.845344088641389,
      1.7250142667163755,
      1.6428239677141323,
      1.5736603125876003,
      1.4746274728453737,
      1.3854324519788492,
      1.2910389408577807,
      1.2772755130523463,
      1.1940938985455984,
      1.1577934403006787,
      1.1071461465584955,
      1.0704311954625618,
      1.0267934761861963,
      0.9752628007631279,
      0.9344469529963271,
      0.9098453553268242,
      0.898175535380677,
      0.8211419791349482,
      0.808951770629316,
      0.7788104386486097,
      0.7628687044231548,
      0.7704024908892751,
      0.715901256957508,
      0.6921080400722284,
      0.6728572465814967,
      0.632164669967409,
      0.6184770195918264,
      0.6067667847598058,
      0.5760920060753081,
      0.5591060283729715,
      0.518060741783396,
      0.5403486678437852,
      0.5265759718954294,
      0.48127059027582497,
```

```
 0.4707083077433833,
 0.4707451756404255]
```

[ ]:
```
%%time
sil_list = []

for k in range(2, 51):
    kmeans = KMeans(n_clusters=k, n_init=10, max_iter=100,random_state = 42)
    kmeans.fit(X_minmax)
    sil_list.append(silhouette_score(X_minmax, kmeans.labels_))
```

```
CPU times: user 3.71 s, sys: 16.6 ms, total: 3.72 s
Wall time: 3.81 s
```

[ ]:
```
pdist(X_minmax).shape
```

[ ]:
```
(11175,)
```

[ ]:
```
squareform(pdist(X_minmax))
```

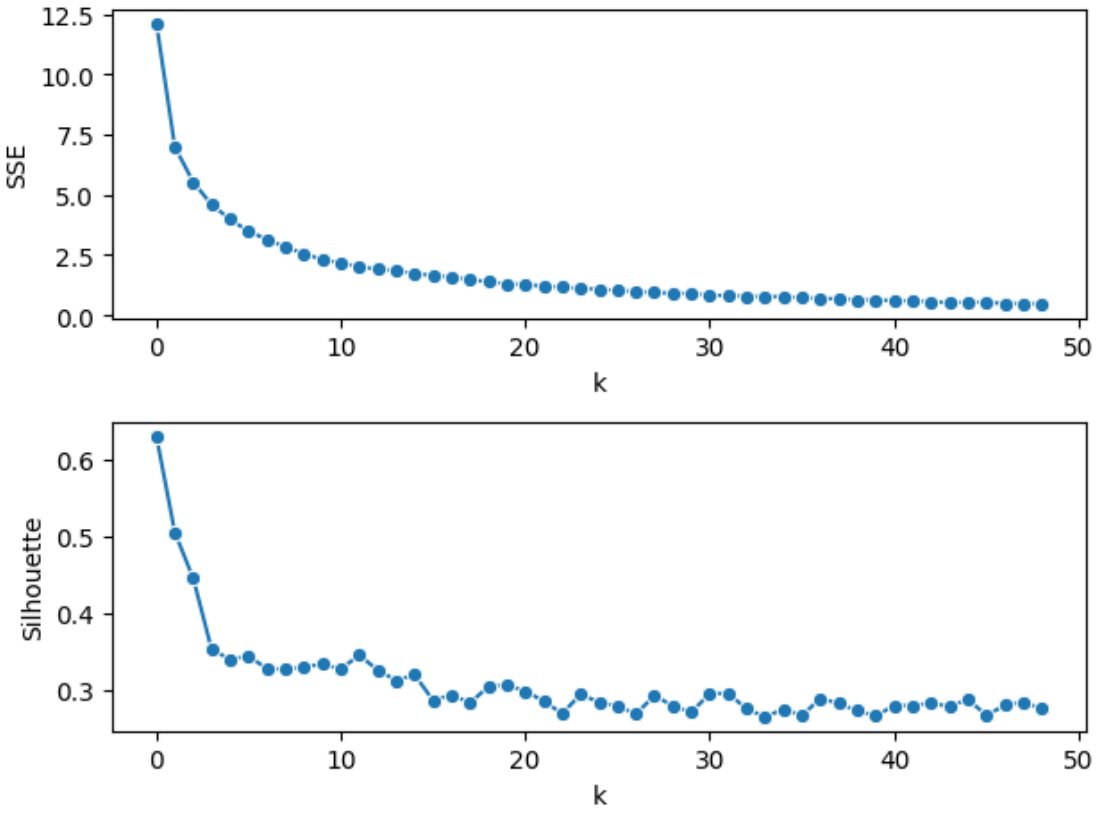
[ ]:
```
array([[0.        , 0.21561354, 0.16810102, …, 1.08257132, 1.14907064,
        0.96462829],
       [0.21561354, 0.        , 0.10157824, …, 1.08390691, 1.17619813,
        0.95649502],
       [0.16810102, 0.10157824, 0.        , …, 1.12088708, 1.19544459,
        0.98859665],
       …,
       [1.08257132, 1.08390691, 1.12088708, …, 0.        , 0.226928  ,
        0.18710825],
       [1.14907064, 1.17619813, 1.19544459, …, 0.226928  , 0.        ,
        0.28409587],
       [0.96462829, 0.95649502, 0.98859665, …, 0.18710825, 0.28409587,
        0.        ]])
```

[ ]:
```
%%time
sil_list = []
dist = squareform(pdist(X_minmax)) # using a precomputed distance matrix

for k in range(2, 51):
    kmeans = KMeans(n_clusters=k, n_init=10, max_iter=100)
    kmeans.fit(X_minmax)
    sil_list.append(silhouette_score(dist, kmeans.labels_,␣
 ↪metric='precomputed'))
```

```
CPU times: user 2.85 s, sys: 15.3 ms, total: 2.86 s
Wall time: 2.88 s
```

[ ]:

[ ]:
```
fig, axs = plt.subplots(2)

sns.lineplot(x=range(len(sse_list)), y=sse_list, marker='o', ax=axs[0])
axs[0].set(xlabel='k', ylabel='SSE')

sns.lineplot(x=range(len(sil_list)), y=sil_list, marker='o', ax=axs[1])
axs[1].set(xlabel='k', ylabel='Silhouette')

plt.tight_layout() # Adjust the padding between and around subplots
```



0.1.3 Bisecting K-means

[ ]:
```
from sklearn.cluster import BisectingKMeans
```

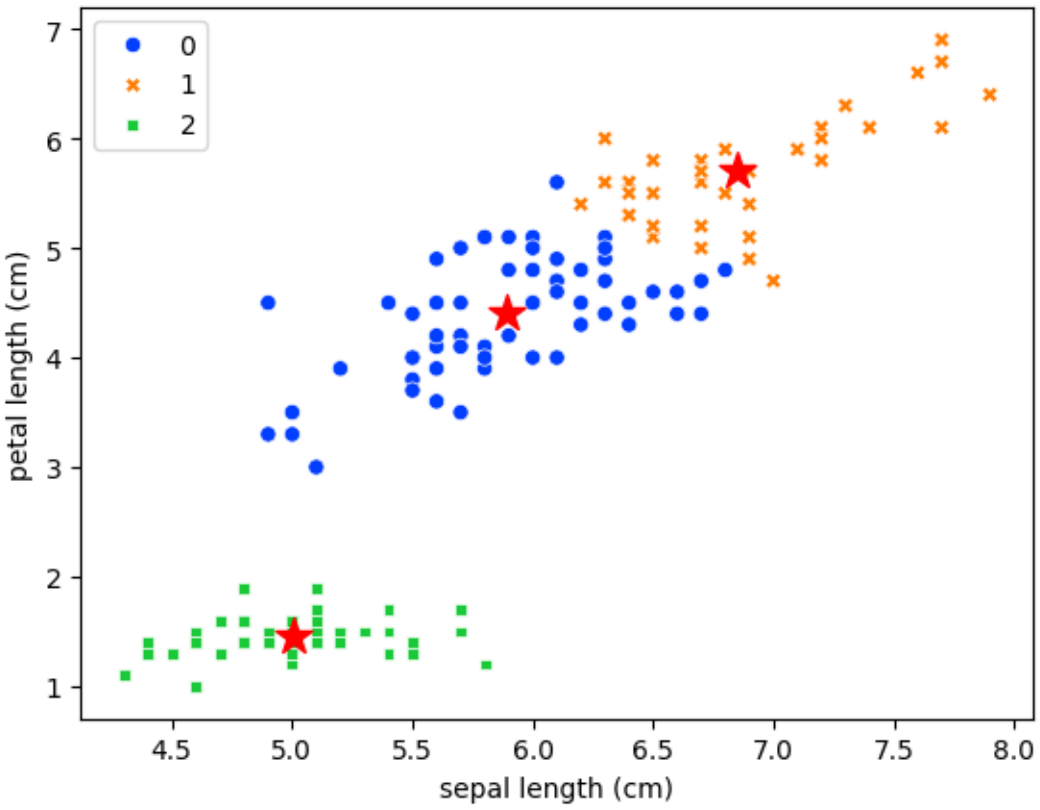
[ ]:
```
bkmeans = BisectingKMeans(n_clusters=3, n_init=10, max_iter=100,␣
↪random_state=94)
bkmeans.fit(X_minmax)
```

[ ]:
```
BisectingKMeans(max_iter=100, n_clusters=3, n_init=10, random_state=94)
```

[ ]:
```
centers = scaler.inverse_transform(bkmeans.cluster_centers_)
centers
```

[ ]:
```
array([[5.88852459, 2.73770492, 4.39672131, 1.41803279],
       [6.84615385, 3.08205128, 5.7025641 , 2.07948718],
       [5.006     , 3.428     , 1.462     , 0.246     ]])
```

[ ]:
```
sns.scatterplot(data=df,
                x="sepal length (cm)",
                y="petal length (cm)",
                hue=bkmeans.labels_,
                style=bkmeans.labels_,
                palette="bright",
               )
plt.legend()
plt.scatter(centers[:,0], centers[:,2], c='red', marker='*', s=200)
plt.show()
```



[ ]:
```
print('SSE', bkmeans.inertia_)
print('Silhouette', silhouette_score(X_minmax, bkmeans.labels_))
```

```
SSE 6.982216473785234
```

```
Silhouette 0.5047687565398589
```

[ ]:
```
# distance matrix to speed up sil score
```

0.1.4 k medoids

[ ]:
```
#!pip install pyclustering
```

[ ]:
```
# standard installation might result in error due to numpy warnings (numpy > 1.
↪24.0)

#after installing pyclustering, also install this warning fix below

#!pip install https://github.com/KulikDM/pyclustering/archive/Warning-Fix.zip
```

[183]:
```
from pyclustering.cluster import kmedoids
```

[184]:
```
# Set random initial medoids.
initial_medoids = [1, 50]

kmedoids_instance = kmedoids.kmedoids(X, initial_medoids)
kmedoids_instance.process()
```

[184]:
```
<pyclustering.cluster.kmedoids.kmedoids at 0x7dd7f437f9e0>
```

[185]:
```
clusters = kmedoids_instance.get_clusters()
```

[186]:
```
clusters = [np.array(x) for x in clusters]
```

[187]:
```
clusters3
```

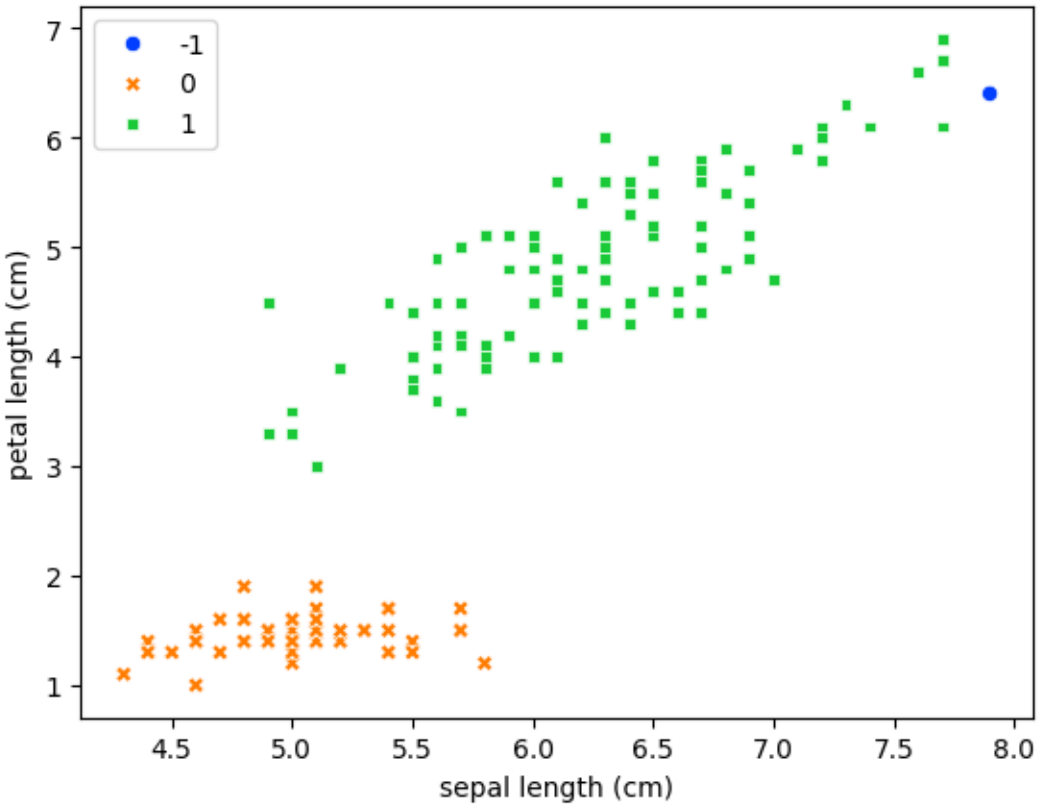
[187]:
```
[array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 57,
        98]),
 array([ 50,  51,  52,  53,  54,  55,  56,  58,  59,  60,  61,  62,  63,
         64,  65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,
         77,  78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,
         90,  91,  92,  93,  94,  95,  96,  97,  99, 100, 101, 102, 103,
        104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
        117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
        130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
        143, 144, 145, 146, 147, 148, 149])]
```

0.1.5 DBScan

[ ]:
```
dbscan = DBSCAN(eps=0.3, min_samples=5)
dbscan.fit(X_minmax)
```

[ ]:
```
DBSCAN(eps=0.3)
```

[ ]:
```
sns.scatterplot(data=df,
                x="sepal length (cm)",
                y="petal length (cm)",
                hue=dbscan.labels_,
                style=dbscan.labels_,
                palette="bright")
plt.show()
```



[ ]:
```
np.unique(dbscan.labels_, return_counts=True)
```

[ ]:
```
(array([-1,  0,  1]), array([ 1, 50, 99]))
```

[ ]:
```
dbscan.labels_
```

[ ]:
```
array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1])
```

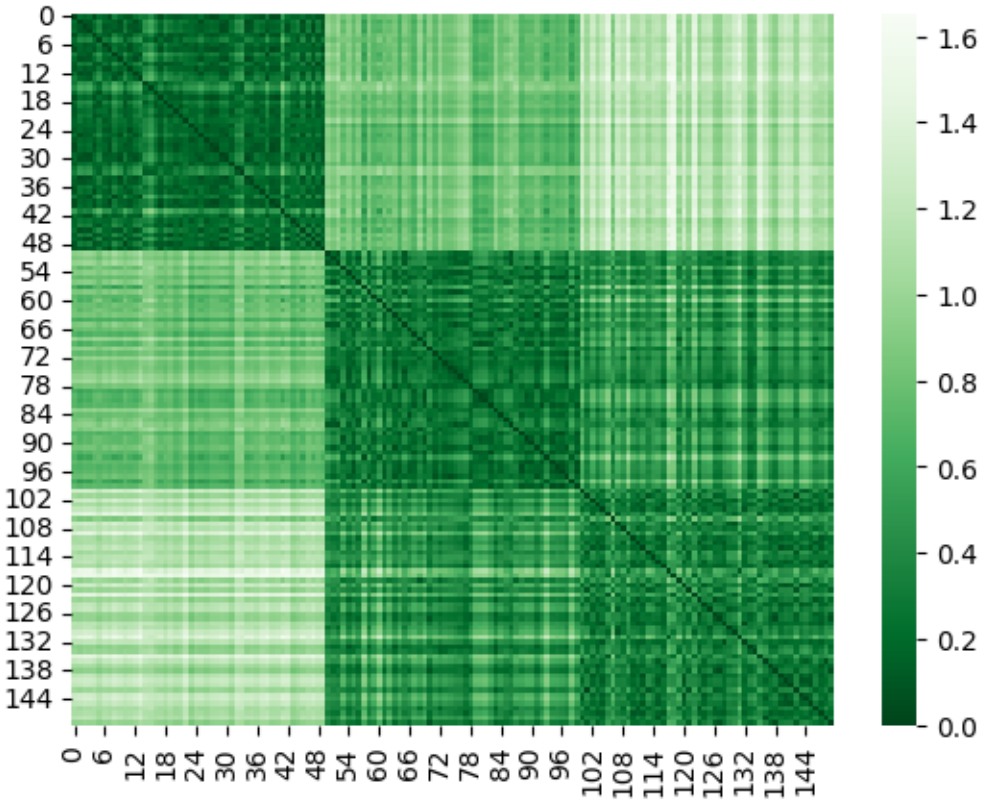
[ ]:
```
print('Silhouette', silhouette_score(X_minmax, dbscan.labels_)) # counting␣
↪silhouette also w.r.t to noise cluster (-1)
print('Silhouette', silhouette_score(X_minmax[dbscan.labels_ != -1], dbscan.
↪labels_[dbscan.labels_ != -1]))
```

```
Silhouette 0.4681853590627469
Silhouette 0.6331315097272541
```

0.1.6 kth neighbor distance

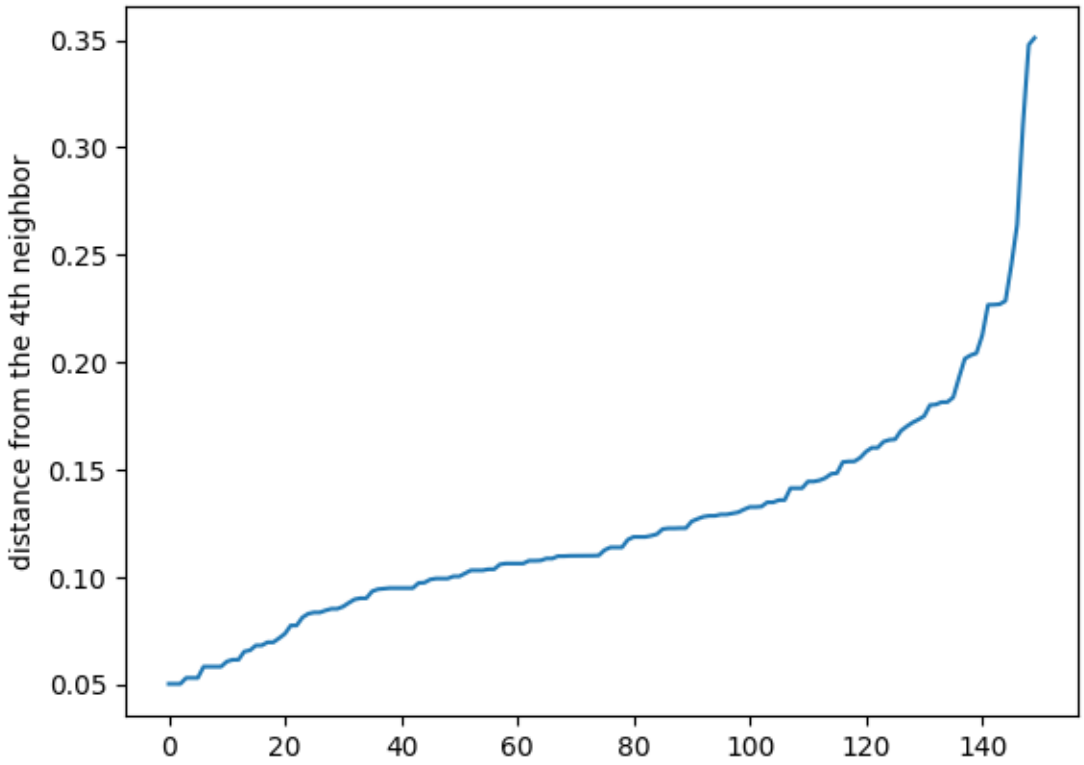
[ ]:
```
dist = pdist(X_minmax, 'euclidean')
dist = squareform(dist)
```

[ ]:
```
sns.heatmap(dist, cmap="Greens_r", annot=False)
plt.show()
```



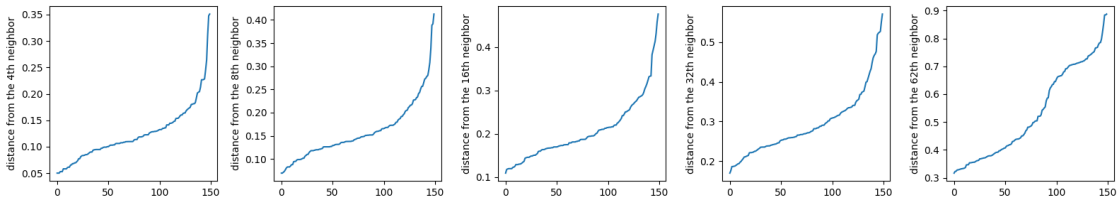
[ ]:
```
k=4
kth_distances = []

for d in dist:  # d is a vector containing distances between the ith record and all the others
    index_kth_distance = np.argsort(d)[k]  # take the index of the kth nearest neighbor
    kth_distances.append(d[index_kth_distance])  # store the distance in a list
```

[ ]:
```
plt.plot(range(0, len(kth_distances)), sorted(kth_distances))
plt.ylabel('distance from the {}th neighbor'.format(k))

plt.show()
```



[ ]:
```
ks = [4, 8, 16, 32, 62]

fig = plt.figure(figsize=(16, 3)) # dimensions of the overall plot
fig_dims = (1, len(ks))

for i in range(len(ks)):
    k = ks[i]
    kth_distances = []

    for d in dist:  # d is a vector containing distances between the ith record␣
 ↪and all the others
        index_kth_distance = np.argsort(d)[k]  # take the index of the kth␣
 ↪nearest neighbor
        kth_distances.append(d[index_kth_distance])  # store the distance in a␣
 ↪list

    plt.subplot2grid(fig_dims, (0, i))
    plt.plot(range(0, len(kth_distances)), sorted(kth_distances))
    plt.ylabel('distance from the {}th neighbor'.format(k))

plt.tight_layout()
```

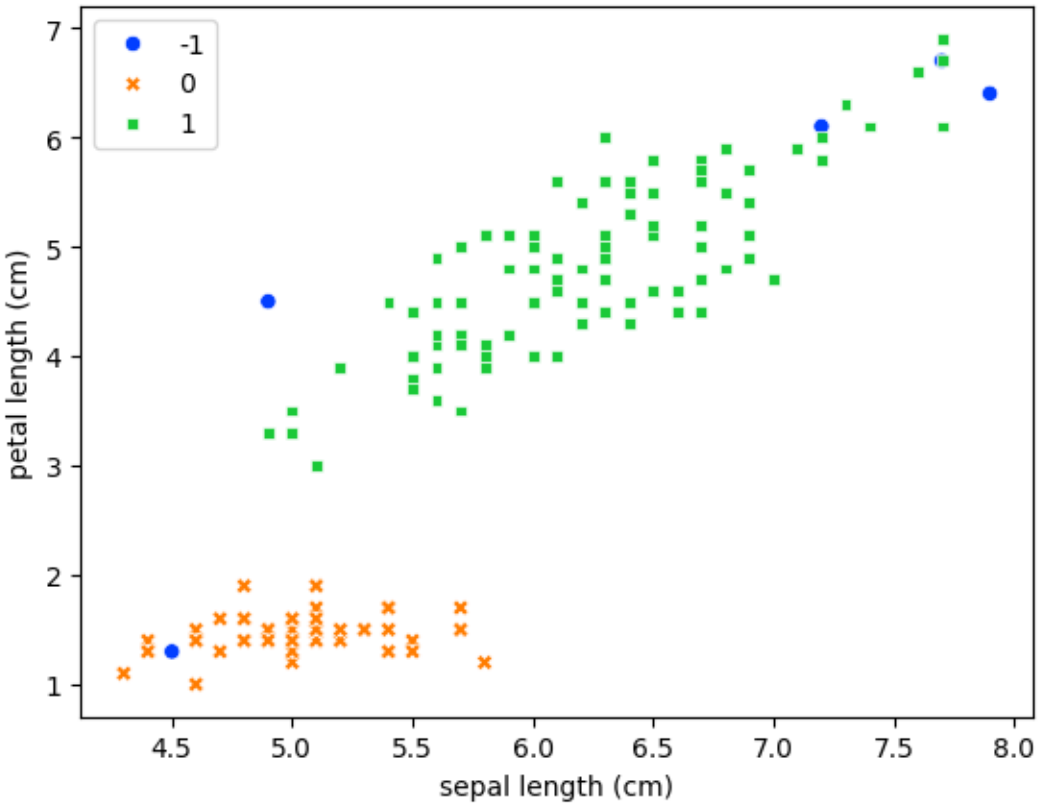
```
plt.show()
```



[ ]:
```
dbscan = DBSCAN(eps=0.19, min_samples=4, metric='precomputed')
dbscan.fit(dist)
```

[ ]:
```
DBSCAN(eps=0.19, metric='precomputed', min_samples=4)
```

[ ]:
```
sns.scatterplot(data=df,
                x="sepal length (cm)",
                y="petal length (cm)",
                hue=dbscan.labels_,
                style=dbscan.labels_,
                palette="bright")
plt.show()
```



[ ]:
```
np.unique(dbscan.labels_, return_counts=True)
```

[ ]:
```
(array([-1,  0,  1]), array([ 5, 49, 96]))
```

[ ]:
```
df['dblabels'] = dbscan.labels_
df.head()
```

[ ]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

  kmeans_labels  dblabels
0             B         0
1             B         0
2             B         0
3             B         0
4             B         0
```

[ ]:
```
# dbscan centroids
df[['sepal length (cm)', 'sepal width (cm)', 'petal width (cm)','petal length␣
↪(cm)', 'dblabels']].groupby('dblabels').mean()
```

[ ]:
```
           sepal length (cm)  sepal width (cm)  petal width (cm)  \
dblabels
-1                  6.440000          3.200000          1.740000
 0                  5.016327          3.451020          0.244898
 1                  6.234375          2.848958          1.658333

           petal length (cm)
dblabels
-1                  5.000000
 0                  1.465306
 1                  4.863542
```

[ ]:
```
print('Silhouette', silhouette_score(X_minmax[dbscan.labels_ != -1], dbscan.
↪labels_[dbscan.labels_ != -1]))
```

```
Silhouette 0.6432450335271236
```

OPTICS

[ ]:
```
from sklearn.cluster import OPTICS, cluster_optics_dbscan
```

[ ]:
```
optics = OPTICS(min_samples = 4, max_eps = np.inf, min_cluster_size=40)
optics.fit(X_minmax)
```

[ ]:
```
OPTICS(min_cluster_size=40, min_samples=4)
```

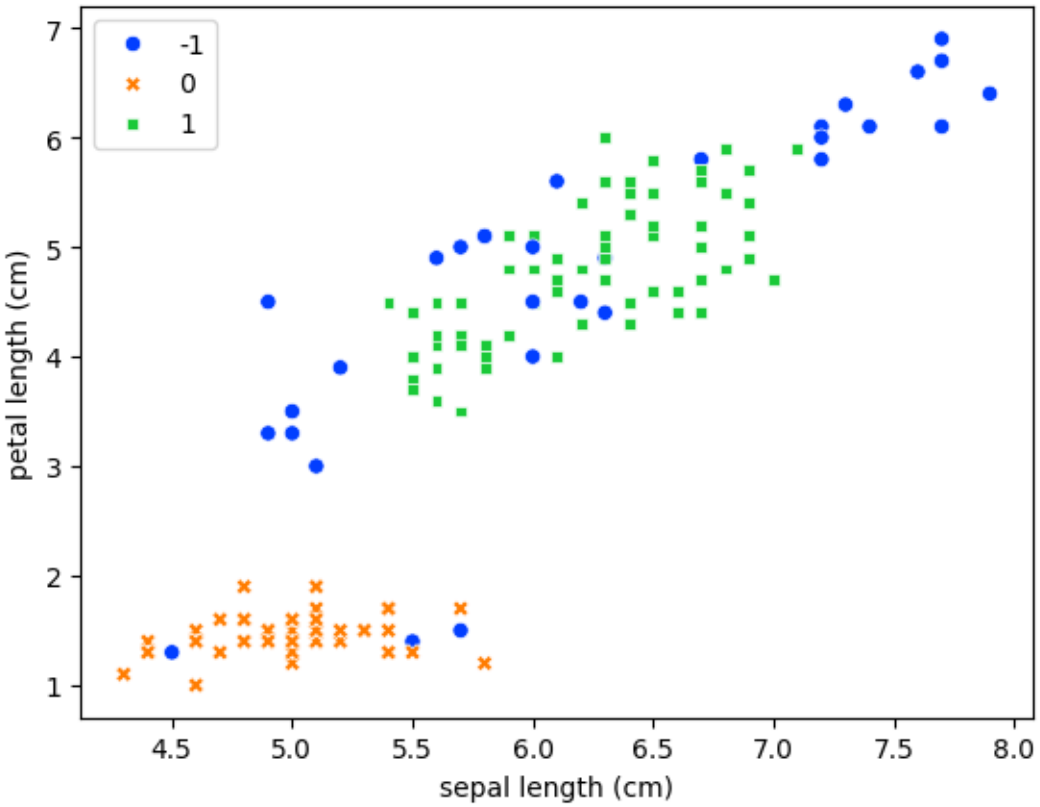
[ ]:
```
print("Silhouette", silhouette_score(X_minmax[optics.labels_ != -1], optics.
↪labels_[optics.labels_ != -1]))
```

```
Silhouette 0.6901722966689927
```

[ ]:
```
np.unique(optics.labels_, return_counts=True)
```

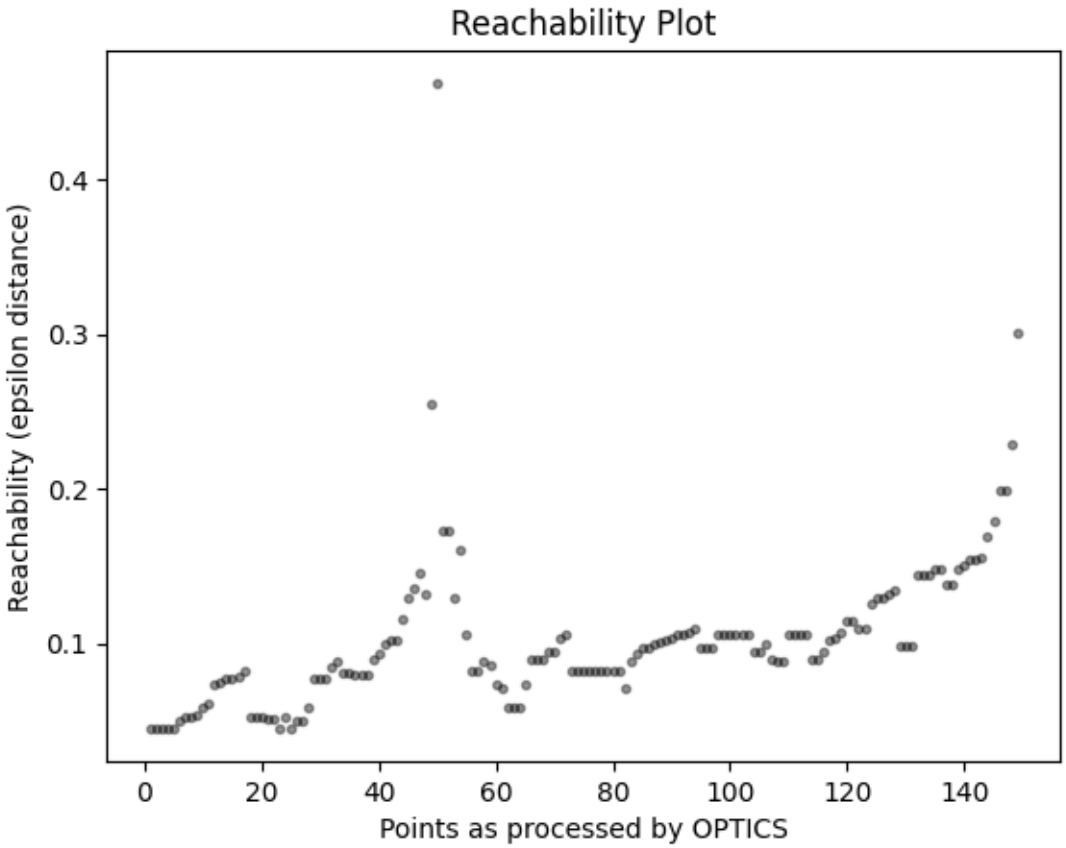
[ ]:
```
(array([-1,  0,  1]), array([33, 47, 70]))
```

[ ]:
```
sns.scatterplot(data=df,
                x="sepal length (cm)",
                y="petal length (cm)",
                hue=optics.labels_,
                style=optics.labels_,
                palette="bright")
plt.show()
```



[ ]:
```
# Valleys in the plot correspond to the clusters
space = np.arange(len(X_minmax))
reachability = optics.reachability_[optics.ordering_]
labels = optics.labels_[optics.ordering_]

# Reachability plot
plt.plot(space, reachability, "k.", alpha=0.4)
plt.xlabel("Points as processed by OPTICS")
plt.ylabel("Reachability (epsilon distance)")
# plt.axhline(0.3, c="red")
plt.title("Reachability Plot")
```

[ ]:
```
Text(0.5, 1.0, 'Reachability Plot')
```



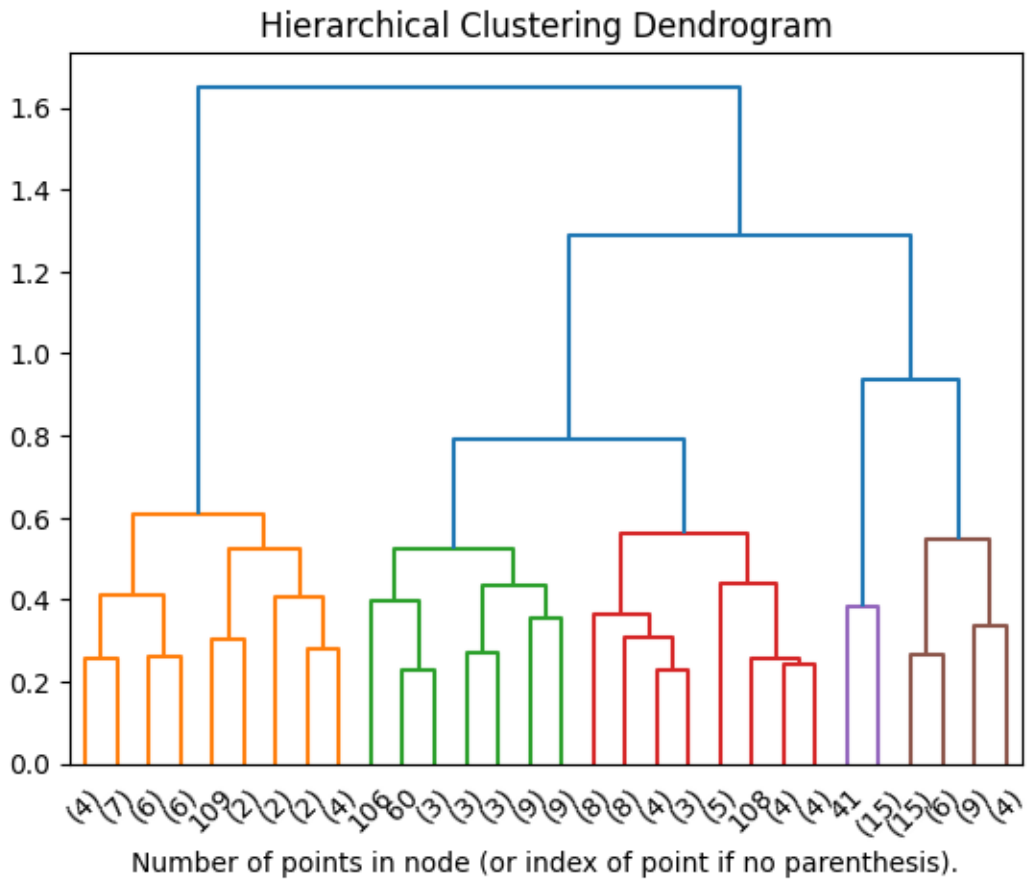
0.1.7 Hierarchical

[ ]:
```
def get_linkage_matrix(model):
    # Create linkage matrix

    # create the counts of samples under each node
```

```
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack(
        [model.children_, model.distances_, counts]
    ).astype(float)

    return linkage_matrix

def plot_dendrogram(model, **kwargs):
    linkage_matrix = get_linkage_matrix(model)
    dendrogram(linkage_matrix, **kwargs)
```

[ ]:
```
# setting distance_threshold=0 ensures we compute the full tree.
# it is the linkage distance threshold above which clusters will not be merged
model = AgglomerativeClustering(distance_threshold=0, n_clusters=None,
                                metric='euclidean', linkage='complete')
model = model.fit(X_minmax)
```

[ ]:
```
plt.title("Hierarchical Clustering Dendrogram")
plot_dendrogram(model, truncate_mode='lastp', color_threshold=0.7)
plt.xlabel("Number of points in node (or index of point if no parenthesis).")
plt.show()
```



[ ]:
```
# get the labels according to a specific threshold value cut
Z = get_linkage_matrix(model)
labels = fcluster(Z, t=0.7, criterion='distance')
```

[ ]:
```
labels
```

[ ]:
```
array([5, 4, 4, 4, 5, 5, 5, 5, 4, 4, 5, 5, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 4, 5, 5, 5, 4, 4, 5, 5, 5, 4, 5, 5, 5, 4, 5, 5, 4, 4, 5,
       5, 4, 5, 4, 5, 5, 3, 3, 3, 2, 3, 2, 3, 2, 3, 2, 2, 3, 2, 3, 2, 3,
       2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 1, 3, 2, 2, 2, 2, 3, 2, 3, 3, 2,
       2, 2, 2, 3, 2, 2, 2, 2, 2, 3, 2, 2, 1, 3, 1, 1, 1, 1, 2, 1, 3, 1,
       1, 3, 1, 3, 3, 1, 1, 1, 1, 2, 1, 3, 1, 3, 1, 1, 3, 3, 1, 1, 1, 1,
       1, 3, 3, 1, 1, 1, 3, 1, 1, 1, 3, 1, 1, 1, 3, 1, 1, 3], dtype=int32)
```

[ ]:
```
print('Silhouette', silhouette_score(X_minmax, labels))
```

```
Silhouette 0.3364082243047556
```

Choosing the number of clusters

[ ]:
```
hier = AgglomerativeClustering(n_clusters=3, metric='euclidean',␣
↪linkage='complete')
hier.fit(X_minmax)
```

[ ]:
```
AgglomerativeClustering(linkage='complete', n_clusters=3)
```

[ ]:
```
hier.labels_
```

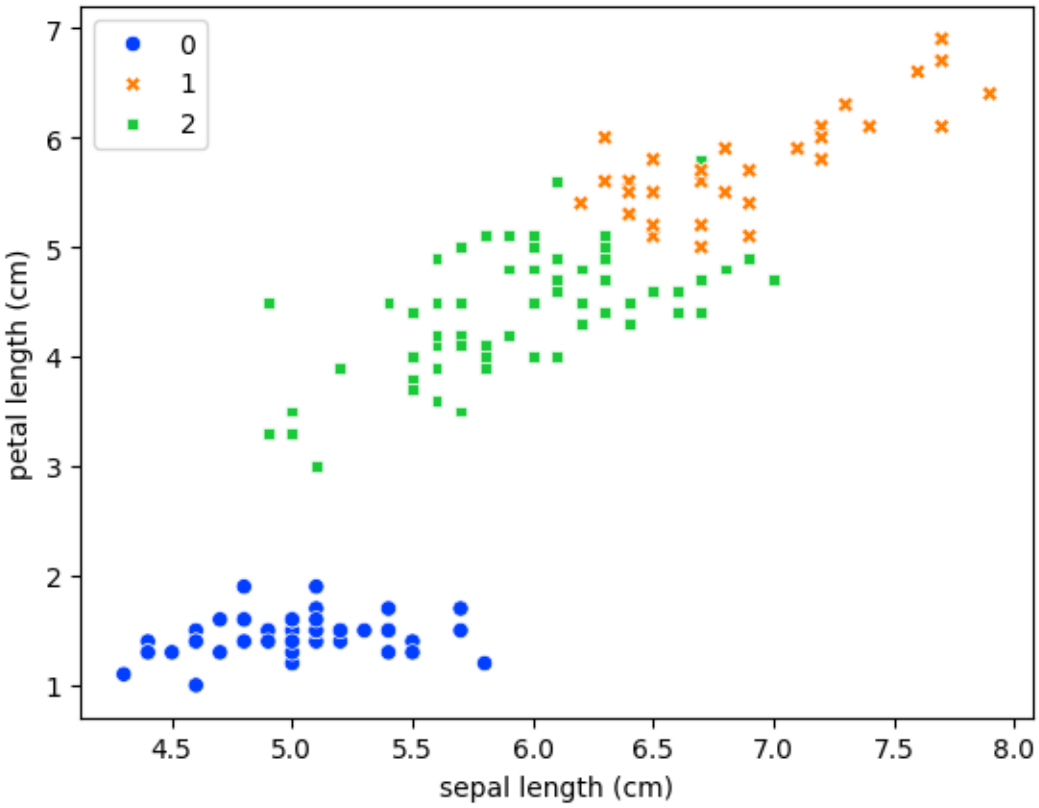
[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1,
       1, 2, 1, 2, 2, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 2, 2, 1, 1, 1, 1,
       1, 2, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2])
```

Precomputed distance matrix

[ ]:
```
hier = AgglomerativeClustering(n_clusters=3, metric='precomputed',␣
↪linkage='complete')
hier.fit(dist)
```

[ ]:
```
AgglomerativeClustering(linkage='complete', metric='precomputed', n_clusters=3)
```

[ ]:
```
sns.scatterplot(data=df,
                x="sepal length (cm)",
                y="petal length (cm)",
                hue=hier.labels_,
                style=hier.labels_,
                palette="bright")
plt.show()
```



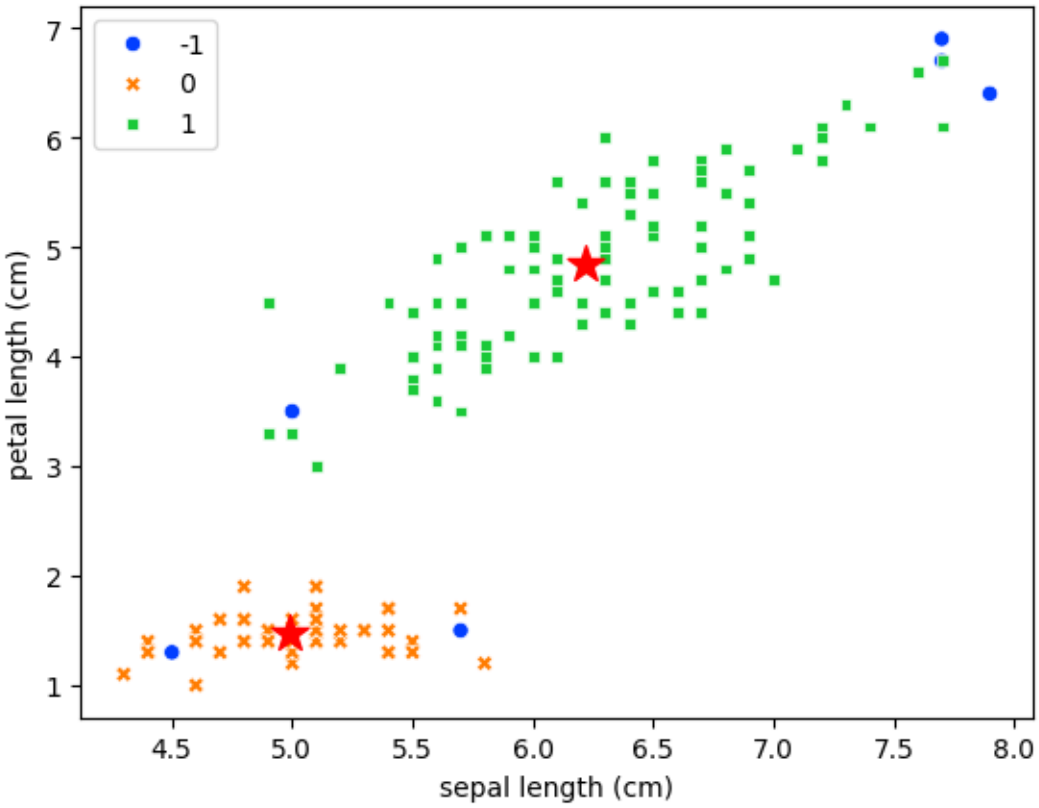
0.1.8 HDBScan

[ ]:
```
from sklearn.cluster import HDBSCAN
```

[ ]:
```
hdb = HDBSCAN(cluster_selection_epsilon=0.3, min_samples=30,
              min_cluster_size=5, max_cluster_size=None,
              store_centers="centroid")
hdb.fit(X_minmax)
```

[ ]:
```
HDBSCAN(cluster_selection_epsilon=0.3, min_samples=30, store_centers='centroid')
```

[ ]:
```
sns.scatterplot(data=df,
                x="sepal length (cm)",
                y="petal length (cm)",
                hue=hdb.labels_,
                style=hdb.labels_,
                palette="bright")

plt.scatter(scaler.inverse_transform(hdb.centroids_)[:,0], scaler.
 ↪inverse_transform(hdb.centroids_)[:,2], c='red', marker='*', s=200)
plt.show()
```



[ ]:
```
np.unique(hdb.labels_, return_counts=True)
```

[ ]:
```
(array([-1,  0,  1]), array([ 6, 48, 96]))
```

[ ]:
```
hdb.centroids_
```

[ ]:
```
array([[0.19207892, 0.58858965, 0.0796667 , 0.05915125],
       [0.53191836, 0.36183505, 0.65110017, 0.64597702]])
```

[ ]:
```
print("Silhouette", silhouette_score(X_minmax[hdb.labels_ != -1], hdb.
↪labels_[hdb.labels_ != -1]))
```

```
Silhouette 0.6480221541804821
```

[ ]:
```

```

0.1.9 Similarity Matrix for Cluster Validation

[ ]:
```
dist = pdist(X_minmax, 'euclidean')
dist = squareform(dist)
```

[ ]:
```
hier = AgglomerativeClustering(n_clusters=3, metric='precomputed',␣
 ↪linkage='complete')
hier.fit(dist)
```

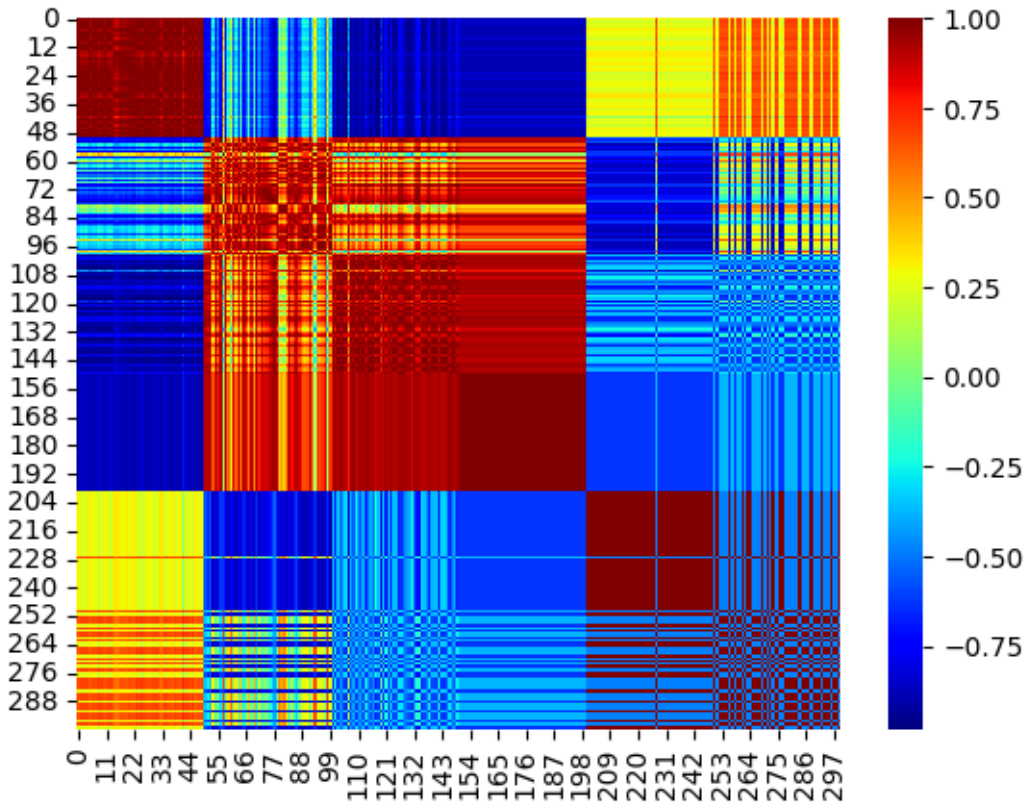
[ ]:
```
AgglomerativeClustering(linkage='complete', metric='precomputed', n_clusters=3)
```

[ ]:
```
ideal_sim
```

[ ]:
```
array([[1., 1., 1., …, 0., 0., 0.],
       [1., 1., 1., …, 0., 0., 0.],
       [1., 1., 1., …, 0., 0., 0.],
       …,
       [0., 0., 0., …, 1., 1., 0.],
       [0., 0., 0., …, 1., 1., 0.],
       [0., 0., 0., …, 0., 0., 1.]])
```

[ ]:
```
shape = len(X_minmax)
ideal_sim = np.eye(shape)

for i in range(shape):
    for j in range(0, i+1):
        ideal_sim[j][i] = 1 if hier.labels_[i] == hier.labels_[j] else 0

ideal_sim = ideal_sim + ideal_sim.T - np.diag(np.diag(ideal_sim)) # copying␣
 ↪upper triangle in lower triangle
```

[ ]:
```
sim = np.corrcoef(dist, ideal_sim)
sns.heatmap(sim, cmap="jet", annot=False)
plt.show()
```



SSE Statistical evaluation

[ ]:
```
import random
```

[ ]:
```
kmeans = KMeans(n_clusters=3, n_init=10, max_iter=100, random_state=94)
kmeans.fit(X_minmax)
```

[ ]:
```
KMeans(max_iter=100, n_clusters=3, n_init=10, random_state=94)
```

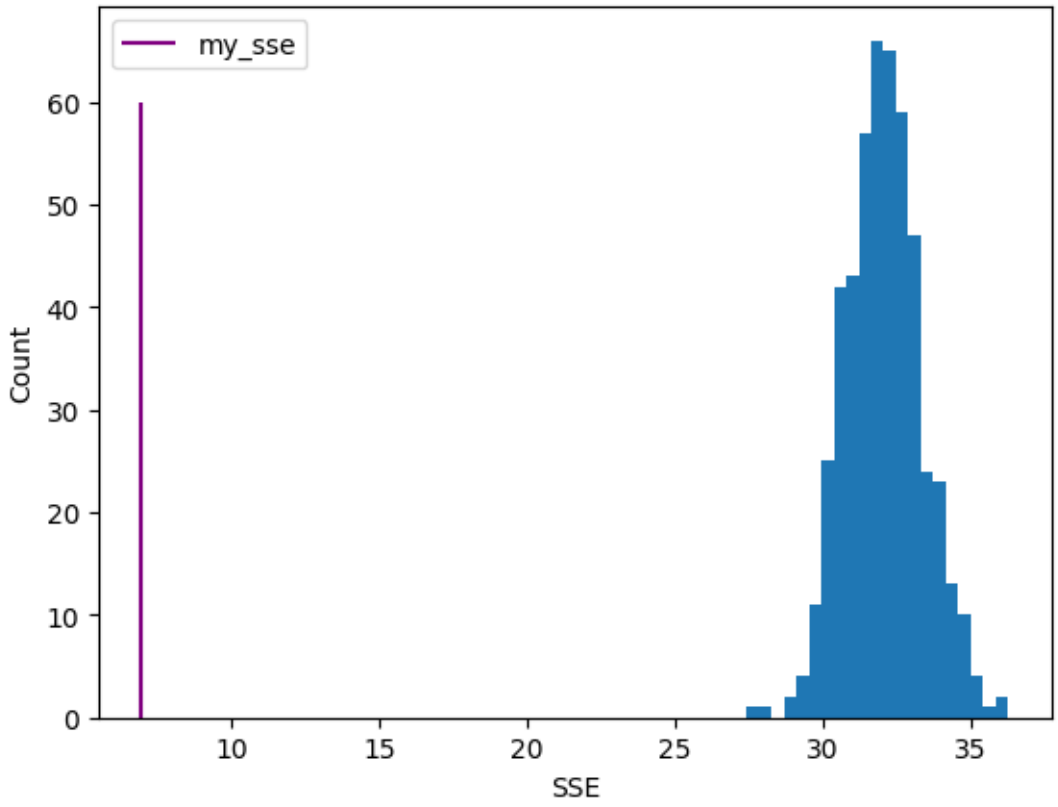
[ ]:
```
my_sse = kmeans.inertia_
my_sse
```

[ ]:
```
6.982216473785234
```

[ ]:
```
N = 500
sse_stats = []

for _ in range(N):
    Xi = np.zeros(shape=X_minmax.shape)

    for cidx in range(Xi.shape[1]):
        col = X_minmax[:, cidx]
```

```
        min, max, nor = np.min(col), np.max(col), len(col)
        Xi[:, cidx] = np.random.uniform(min, max, (1, nor))

    kmeans = KMeans(n_clusters=3, n_init=10, max_iter=100, random_state=94)
    kmeans.fit(Xi)
    sse_stats.append(kmeans.inertia_)
```

[ ]:
```
plt.hist(sse_stats, bins='auto')
plt.vlines(x = my_sse, ymin = 0, ymax = 60, colors = 'purple', label = 'my_sse')
plt.xlabel('SSE')
plt.ylabel('Count')
plt.legend()
plt.show()
```



[ ]:
```

```

clustering_titanic

November 11, 2025

[41]:
```
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
import seaborn as sns
from scipy.spatial.distance import pdist, squareform
```

[42]:
```
df = pd.read_csv("data/titanic.csv")
df.head()
```

[42]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th…	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

0.0.1 Preprocessing

[43]:
```
#from google.colab import drive
#drive.mount('/content/drive')
```

Fill missing values and convert to numerical (where possible)

[44]:
```
sexes = sorted(df['Sex'].unique())
sexes_mapping = dict(zip(sexes, range(0, len(sexes) + 1)))
df['Sex_Val'] = df['Sex'].map(sexes_mapping).astype(int)
```

[45]:
```
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()[0])
embarked_locs = sorted(df['Embarked'].unique())
embarked_locs_mapping = dict(zip(embarked_locs, range(0, len(embarked_locs) +␣
↪1)))
df['Embarked_Val'] = df['Embarked'].map(embarked_locs_mapping).astype(int)
```

[46]:
```
df['AgeFill'] = df.groupby(['Sex', 'Pclass'])['Age'].transform(lambda x: x.
↪fillna(x.median()))
```

[47]:
```
df['FamilySize'] = df['SibSp'] + df['Parch']
```

[48]:
```
df.head()
```

[48]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Sex_Val	Embarked_Val	AgeFill	FamilySize
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	1	2	22.0	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th…	female	38.0	1	0	PC 17599	71.2833	C85	C	0	0	38.0	1
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	0	2		
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	0	2		
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S	1	2		

```
2     26.0           0
3     35.0           1
4     35.0           0
```

Check categorical variables

[49]:
```
df.dtypes[df.dtypes.map(lambda x: x == 'object')]
```

[49]:
```
Name        object
Sex         object
Ticket      object
Cabin       object
Embarked    object
dtype: object
```

[50]:
```
df_train = df.drop(['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'], axis=1)
df_train.drop(['Survived', 'Age', 'SibSp', 'Parch', 'PassengerId',␣
 ↪'Embarked_Val', 'Sex_Val'], axis=1, inplace=True)
df_train.dtypes
```

[50]:
```
Pclass          int64
Fare          float64
AgeFill       float64
FamilySize      int64
dtype: object
```

[51]:
```
df_train.head()
```

[51]:

	Pclass	Fare	AgeFill	FamilySize
0	3	7.2500	22.0	1
1	1	71.2833	38.0	1
2	3	7.9250	26.0	0
3	1	53.1000	35.0	1
4	3	8.0500	35.0	0

[52]:
```
scaler = MinMaxScaler()
train_data = scaler.fit_transform(df_train)
```

0.0.2 Clustering

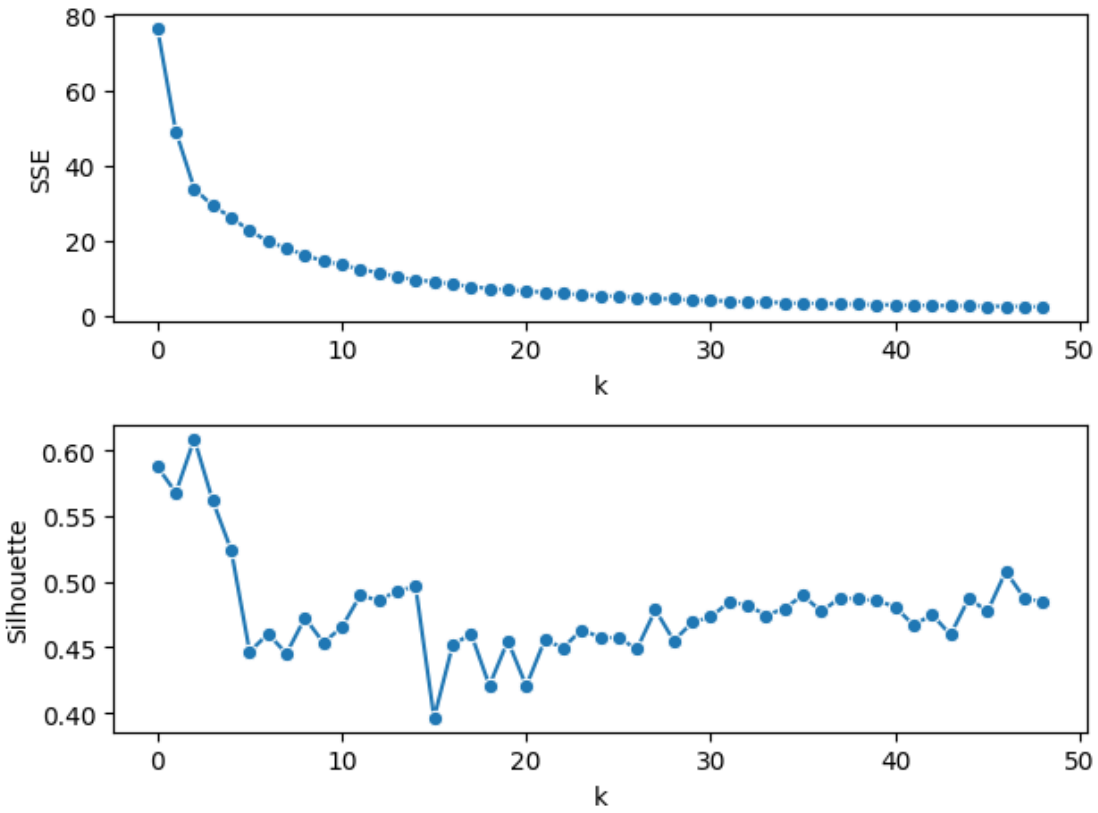
[53]:
```
from sklearn.metrics import *
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.neighbors import kneighbors_graph
```

[17]:
```
%%time
sse_list = []
sil_list = []
```

```
for k in range(2, 51):
    kmeans = KMeans(init='k-means++', n_clusters=k, n_init=10, max_iter=100)
    kmeans.fit(train_data)
    sse_list.append(kmeans.inertia_)
    sil_list.append(silhouette_score(train_data, kmeans.labels_))
```

```
CPU times: user 9.45 s, sys: 44.5 ms, total: 9.5 s
Wall time: 5.31 s
```

[18]:
```
fig, axs = plt.subplots(2) # 1 row, 2 columns

sns.lineplot(x=range(len(sse_list)), y=sse_list, marker='o', ax=axs[0])
axs[0].set(xlabel='k', ylabel='SSE')
sns.lineplot(x=range(len(sil_list)), y=sil_list, marker='o', ax=axs[1])
axs[1].set(xlabel='k', ylabel='Silhouette')

plt.tight_layout() # Adjust the padding between and around subplots
```



[22]:
```
kmeans = KMeans(init='k-means++', n_clusters=5, n_init=10, max_iter=100)
kmeans.fit(train_data)
```

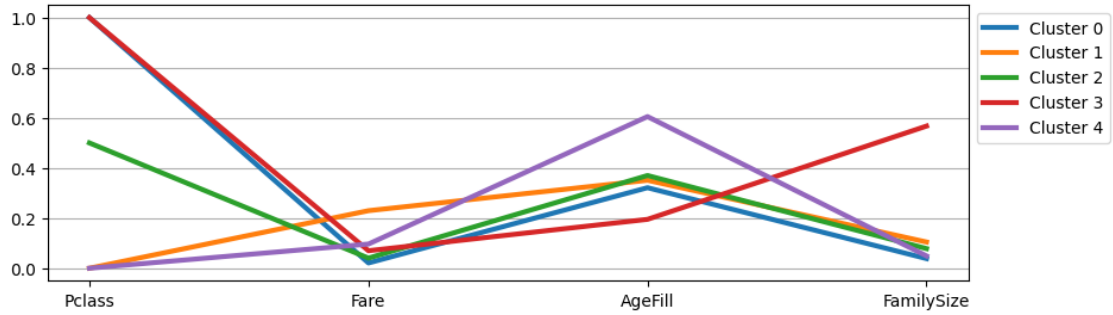
[22]:
```
KMeans(max_iter=100, n_clusters=5, n_init=10)
```

[23]:
```
print('labels', np.unique(kmeans.labels_, return_counts=True))
print('sse', kmeans.inertia_)
print('silhouette', silhouette_score(train_data, kmeans.labels_))
```

```
labels (array([0, 1, 2, 3, 4], dtype=int32), array([433, 109, 184,  58, 107]))
sse 29.169946054563958
silhouette 0.5614640538880703
```

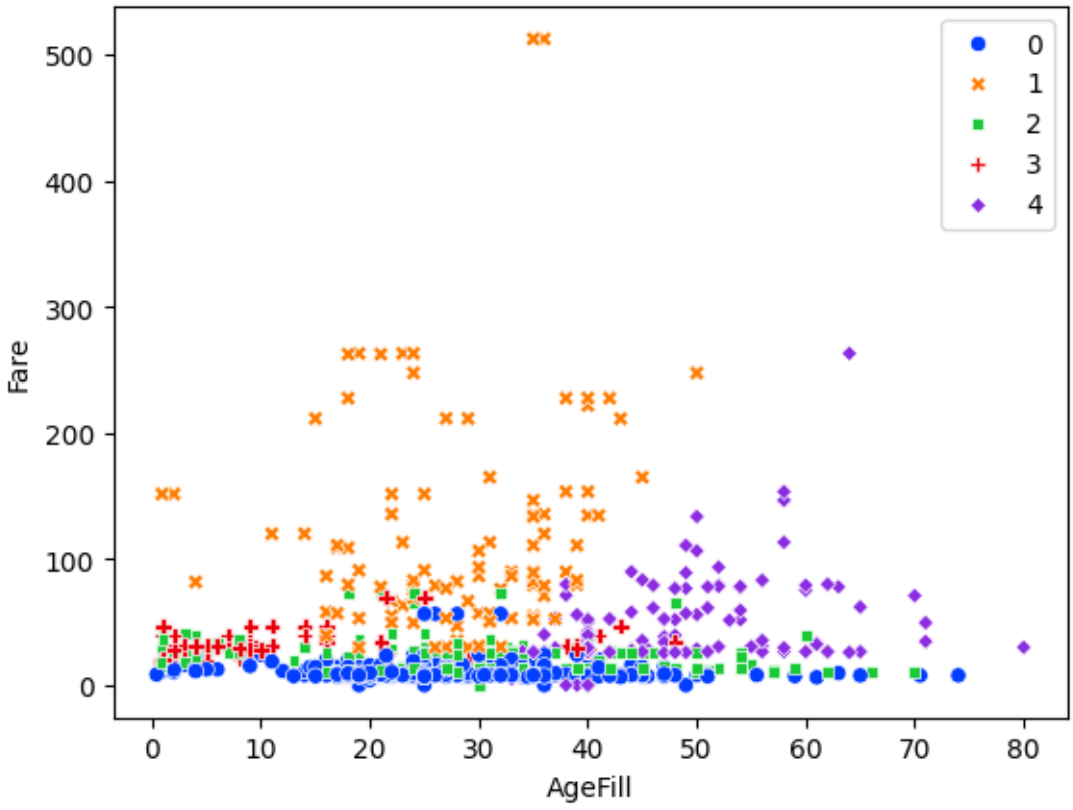
[23]:
```

```

[24]:
```
plt.figure(figsize=(10, 3))

for i in range(len(kmeans.cluster_centers_)):
    plt.plot(range(0, 4), kmeans.cluster_centers_[i], label='Cluster %s' % i,␣
 ↪linewidth=3)
plt.xticks(range(0, 4), list(df_train.columns))
plt.legend(bbox_to_anchor=(1,1))
plt.grid(axis='y')
plt.show()
```



[25]:
```
df_clusters = df_train.copy()
df_clusters['Labels'] = kmeans.labels_
```

[26]:
```
sns.scatterplot(data=df_clusters,
                x="AgeFill",
                y="Fare",
                hue=kmeans.labels_,
                style=kmeans.labels_,
                palette="bright")
plt.show()
```



[27]:
```
df_clusters.head()
```

[27]:

	Pclass	Fare	AgeFill	FamilySize	Labels
0	3	7.2500	22.0	1	0
1	1	71.2833	38.0	1	4
2	3	7.9250	26.0	0	0
3	1	53.1000	35.0	1	1
4	3	8.0500	35.0	0	0

[28]:
```
pclass_xt = pd.crosstab(df_clusters['Pclass'], df_clusters['Labels'])
pclass_xt
```

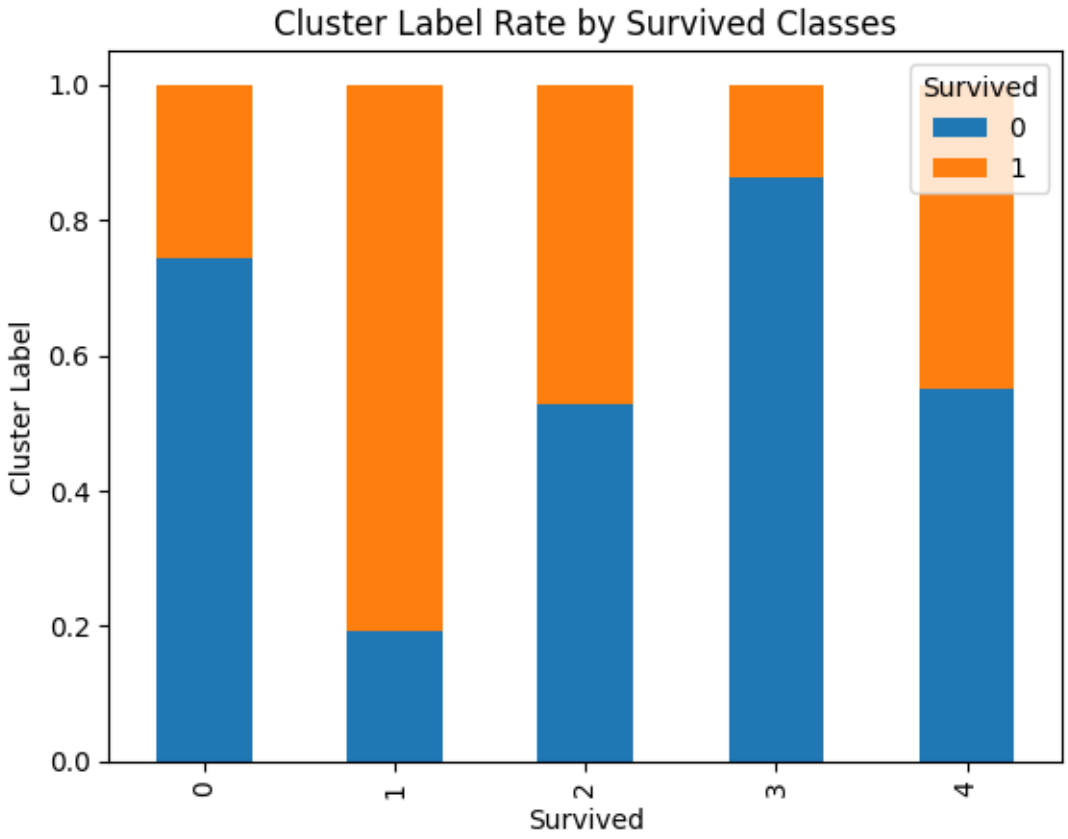
[28]:

Labels	0	1	2	3	4
Pclass					
1	0	109	0	0	107
2	0	0	184	0	0
3	433	0	0	58	0

[29]:
```
psurv_xt = pd.crosstab(df_clusters['Labels'], df['Survived'])
psurv_xt
```

[29]:
Survived	0	1
Labels		
0	322	111
1	21	88
2	97	87
3	50	8
4	59	48

[30]:
```
psurv_xt_pct = psurv_xt.div(psurv_xt.sum(1).astype(float), axis=0)
psurv_xt_pct.plot(kind='bar', stacked=True, title='Cluster Label Rate by␣
 ↪Survived Classes')
plt.xlabel('Survived')
plt.ylabel('Cluster Label')
plt.show()
```

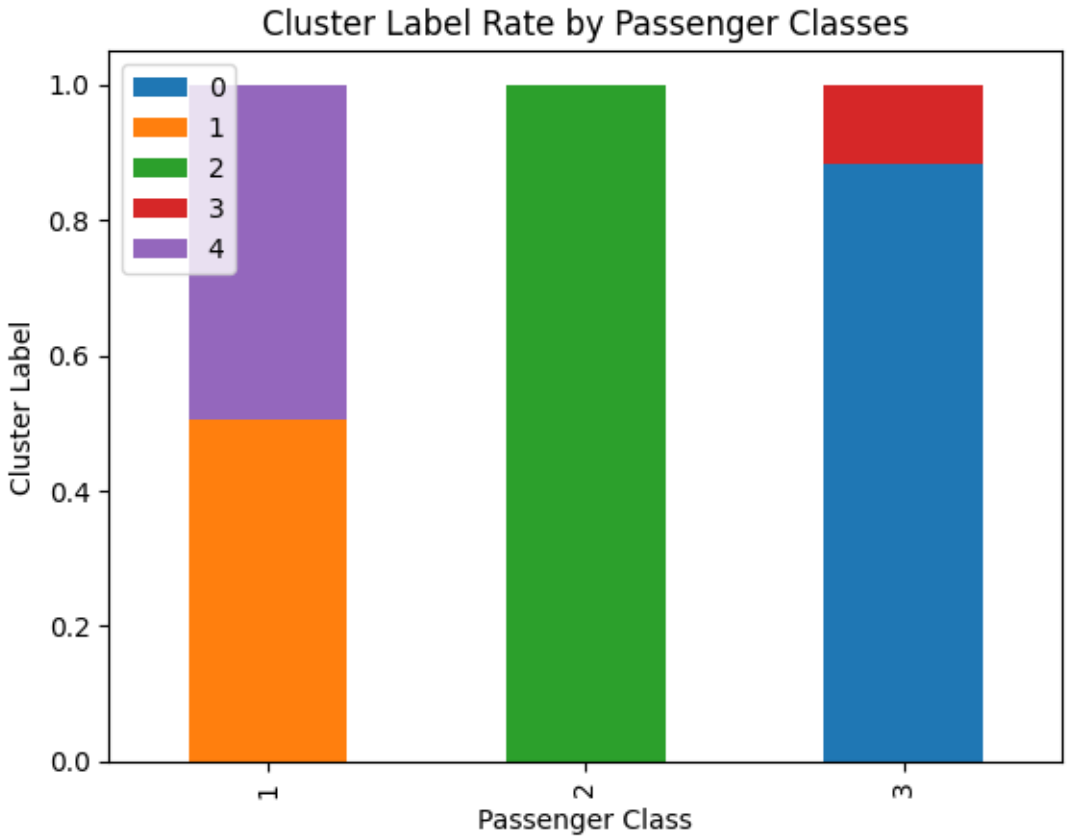


[31]:
```
df_clusters[df_clusters['Labels']==1].describe()
```

[31]:
	Pclass	Fare	AgeFill	FamilySize	Labels
count	109.0	109.000000	109.000000	109.000000	109.0
mean	1.0	118.044533	28.256147	1.045872	1.0

std	0.0	92.682510	9.034489	1.173672	0.0
min	1.0	26.283300	0.920000	0.000000	1.0
25%	1.0	56.929200	22.000000	0.000000	1.0
50%	1.0	86.500000	29.000000	1.000000	1.0
75%	1.0	146.520800	35.000000	1.000000	1.0
max	1.0	512.329200	50.000000	5.000000	1.0

[32]:
```
pclass_xt_pct = pclass_xt.div(pclass_xt.sum(1).astype(float), axis=0)
pclass_xt_pct.plot(kind='bar', stacked=True, title='Cluster Label Rate by␣
↪Passenger Classes')
plt.xlabel('Passenger Class')
plt.ylabel('Cluster Label')
plt.legend(loc='best')
plt.show()
```

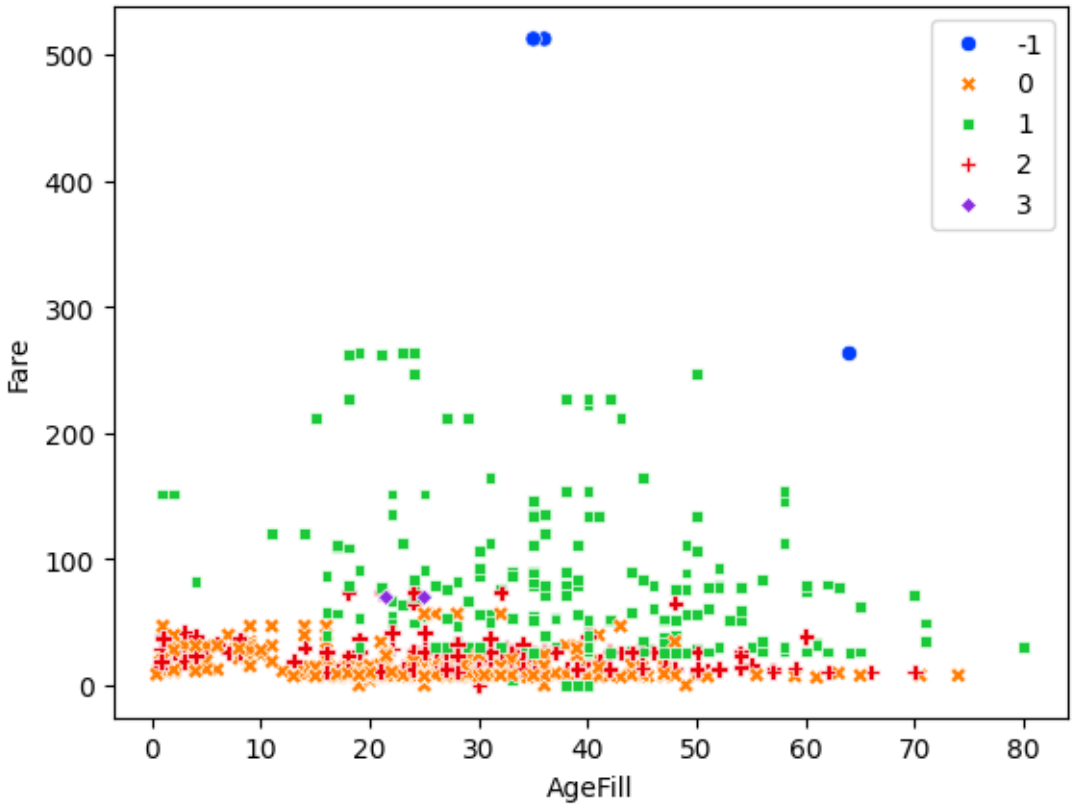


0.0.3 DBScan

[34]:
```
# density based clustering
print('dbscan')

dbscan = DBSCAN(eps=0.3, min_samples=5, metric='euclidean')
dbscan.fit(train_data)

hist, bins = np.histogram(dbscan.labels_, bins=range(-1, len(set(dbscan.
 ↪labels_)) + 1))
print('labels', dict(zip(bins, hist)))
print('silhouette', silhouette_score(train_data[dbscan.labels_ != -1], dbscan.
 ↪labels_[dbscan.labels_ != -1]))
```

```
dbscan
labels {np.int64(-1): np.int64(4), np.int64(0): np.int64(484), np.int64(1):
np.int64(212), np.int64(2): np.int64(184), np.int64(3): np.int64(7),
np.int64(4): np.int64(0)}
silhouette 0.5728928528267885
```

[35]:
```
sns.scatterplot(data=df_clusters,
                x="AgeFill",
                y="Fare",
                hue=dbscan.labels_,
                style=dbscan.labels_,
                palette="bright")
plt.show()
```



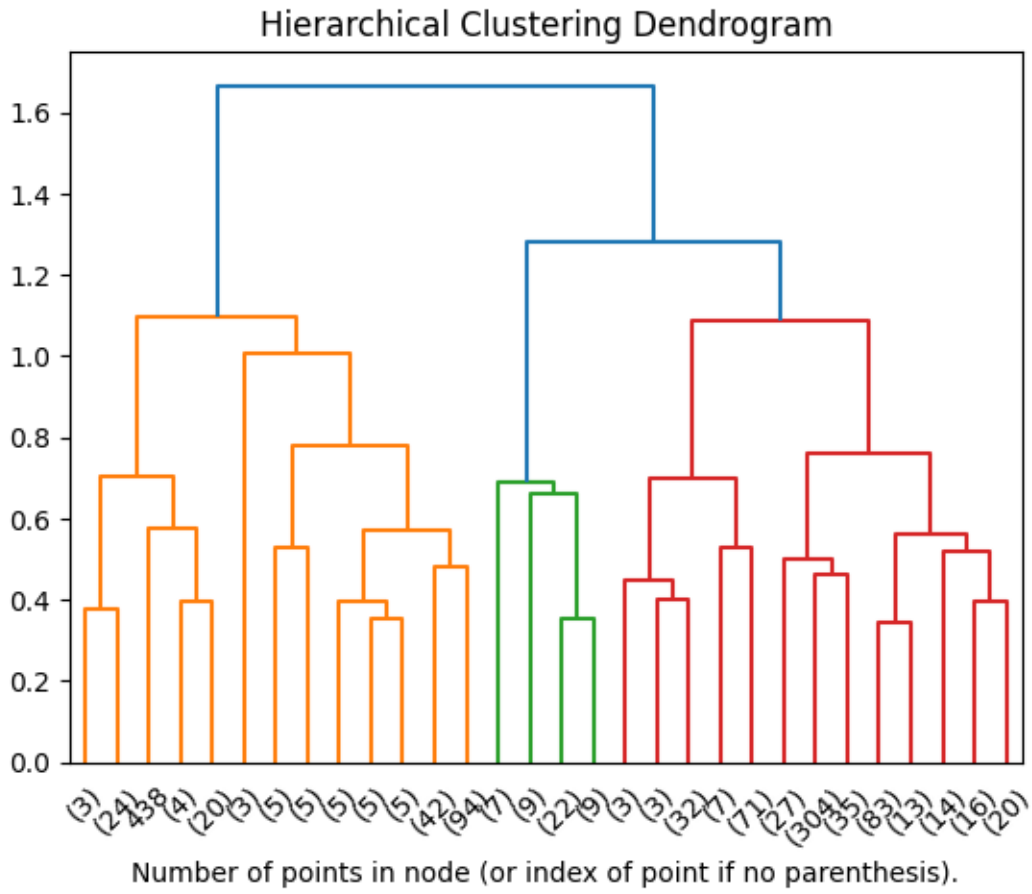
0.0.4 Hierarchical

[36]:
```
def get_linkage_matrix(model):
    # Create linkage matrix

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack(
        [model.children_, model.distances_, counts]
    ).astype(float)
```

```
    return linkage_matrix

def plot_dendrogram(model, **kwargs):
    linkage_matrix = get_linkage_matrix(model)
    dendrogram(linkage_matrix, **kwargs)
```

[37]:
```
# setting distance_threshold=0 ensures we compute the full tree.
model = AgglomerativeClustering(distance_threshold=0, n_clusters=None,␣
 ↪metric='euclidean', linkage='complete')
model = model.fit(train_data)
```

[38]:
```
plt.title("Hierarchical Clustering Dendrogram")
plot_dendrogram(model, truncate_mode="lastp", color_threshold=1.2)
plt.xlabel("Number of points in node (or index of point if no parenthesis).")
plt.show()
```



[39]:
```
# get the labels according to a specific threshold value cut
Z = get_linkage_matrix(model)
```

```
labels = fcluster(Z, t=1.2, criterion='distance')
```

[40]:
```
labels
```

[40]:
```
array([3, 1, 3, 1, 3, 3, 1, 2, 3, 3, 3, 1, 3, 2, 3, 3, 2, 3, 3, 3, 3, 3,
       3, 1, 2, 2, 3, 1, 3, 3, 1, 1, 3, 3, 1, 1, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 2, 3, 1, 3, 1, 1, 3, 3, 3, 2, 3, 1, 1, 2, 1, 3,
       3, 3, 2, 3, 3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3,
       1, 3, 3, 3, 1, 3, 3, 3, 1, 1, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3,
       1, 3, 3, 3, 3, 3, 3, 3, 1, 2, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 1, 1, 3, 1, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 1, 3, 3,
       3, 1, 3, 3, 3, 2, 3, 3, 3, 3, 2, 3, 1, 2, 1, 3, 1, 2, 3, 3, 1, 3,
       3, 1, 3, 3, 2, 3, 2, 3, 3, 1, 3, 1, 3, 3, 3, 3, 3, 3, 1, 1, 3, 3,
       3, 3, 3, 2, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 1, 3, 3, 1, 3,
       3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 1, 3, 3, 2, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 1, 3, 3, 1, 3, 3, 3, 1, 3, 3, 3, 1, 1, 1, 3, 3, 2, 1, 1,
       3, 3, 2, 3, 1, 1, 1, 3, 3, 1, 3, 1, 3, 3, 2, 3, 3, 3, 3, 3, 1, 3,
       3, 3, 3, 3, 1, 1, 3, 3, 3, 1, 3, 1, 1, 1, 3, 3, 3, 3, 3, 1, 1, 1,
       3, 1, 1, 1, 3, 3, 3, 3, 3, 3, 1, 1, 3, 3, 3, 3, 2, 1, 3, 3, 3, 1,
       3, 1, 1, 3, 1, 3, 1, 1, 3, 1, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1,
       3, 3, 3, 3, 1, 3, 3, 3, 2, 3, 3, 3, 3, 3, 1, 3, 3, 1, 1, 3, 3, 1,
       2, 1, 3, 1, 3, 3, 1, 3, 3, 1, 3, 3, 2, 3, 3, 3, 1, 3, 3, 1, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 1, 1, 3, 3, 1, 3,
       3, 3, 3, 3, 3, 1, 3, 1, 3, 1, 3, 3, 1, 1, 3, 3, 1, 1, 3, 3, 1, 3,
       1, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 2, 3, 3, 3,
       1, 3, 1, 1, 3, 3, 3, 3, 1, 1, 3, 3, 1, 3, 1, 3, 3, 3, 3, 3, 1, 1,
       3, 1, 3, 3, 3, 3, 1, 1, 3, 1, 3, 3, 3, 3, 1, 3, 3, 1, 3, 3, 3, 1,
       3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 3, 1, 1, 2, 2, 3, 1, 1, 3, 3, 3, 3,
       1, 3, 3, 3, 3, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1,
       1, 3, 3, 3, 3, 1, 3, 3, 3, 1, 3, 1, 3, 1, 3, 1, 3, 3, 3, 1, 3, 3,
       3, 3, 3, 3, 3, 1, 3, 3, 1, 3, 1, 3, 3, 1, 3, 1, 2, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 1, 3, 3, 3, 1, 3, 1, 3, 3, 1, 3, 1, 1, 2, 3, 3, 3,
       2, 3, 3, 1, 2, 3, 3, 1, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1,
       1, 3, 1, 3, 3, 3, 3, 3, 3, 1, 3, 1, 3, 3, 3, 3, 3, 3, 2, 1, 3, 1,
       3, 2, 3, 3, 2, 3, 3, 1, 1, 3, 3, 3, 1, 3, 3, 3, 1, 3, 1, 1, 3, 3,
       3, 3, 3, 1, 1, 3, 1, 1, 1, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3,
       3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 2, 1, 3, 3, 1, 1, 1, 3, 3, 1, 3, 3,
       1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 1, 3, 1, 1, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 1, 1, 3, 3, 3, 3, 2, 3, 1, 3, 3,
       2, 1, 3, 3, 1, 3, 3, 3, 3, 3, 1, 3, 3, 3, 1, 3, 3, 1, 3, 3, 3, 2,
       3, 1, 3, 3, 3, 2, 1, 3, 1, 3, 2, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 1,
       3, 3, 3, 1, 3, 3, 1, 3, 3, 3, 2, 3, 3, 1, 2, 3, 3, 1, 3, 3, 1, 1,
       3, 3, 3, 3, 1, 2, 3, 3, 3, 1, 3, 3, 3, 1, 1, 3, 3, 3, 3, 3, 3, 1,
       3, 3, 3, 3, 3, 2, 3, 1, 3, 1, 3], dtype=int32)
```

[ ]:
```
print('Silhouette', silhouette_score(train_data, labels))
```

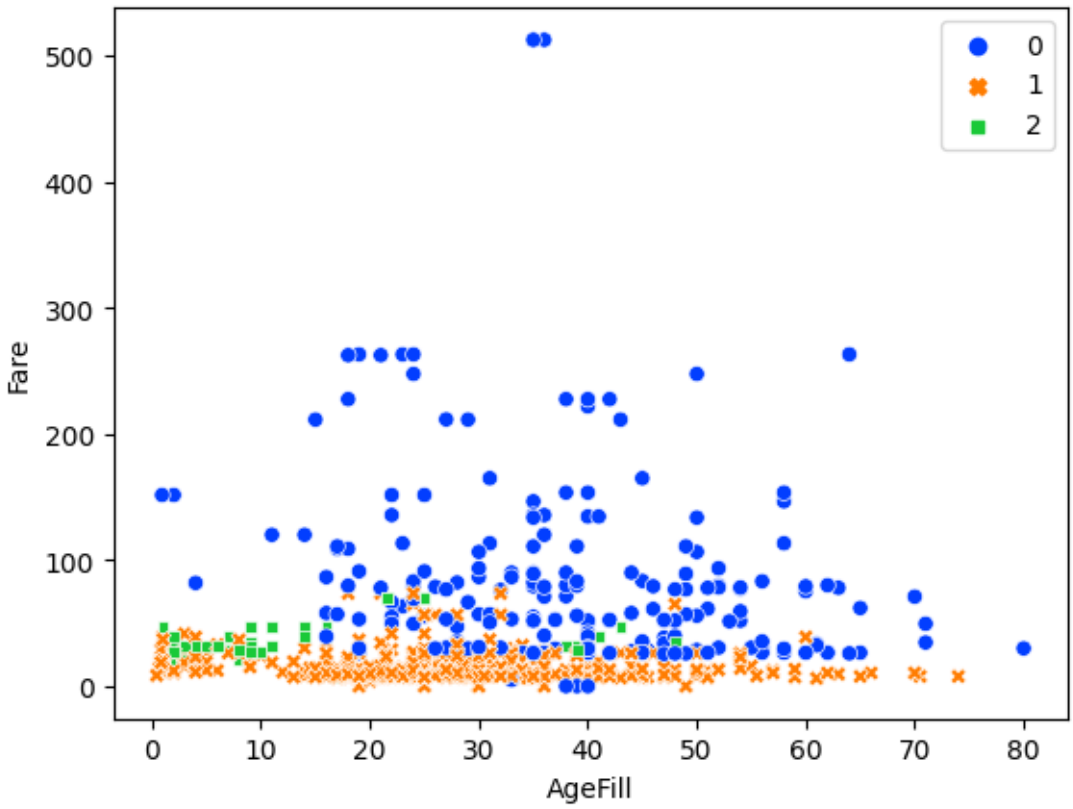
```
Silhouette 0.5026555448852588
```

0.0.5 Choosing the number of clusters

[ ]:
```
hier = AgglomerativeClustering(n_clusters=3, metric='euclidean',␣
↪linkage='complete')
hier.fit(train_data)
```

[ ]:
```
AgglomerativeClustering(linkage='complete', n_clusters=3)
```

[ ]:
```
sns.scatterplot(data=df_clusters,
                x="AgeFill",
                y="Fare",
                hue=hier.labels_,
                style=hier.labels_,
                palette="bright")
plt.show()
```



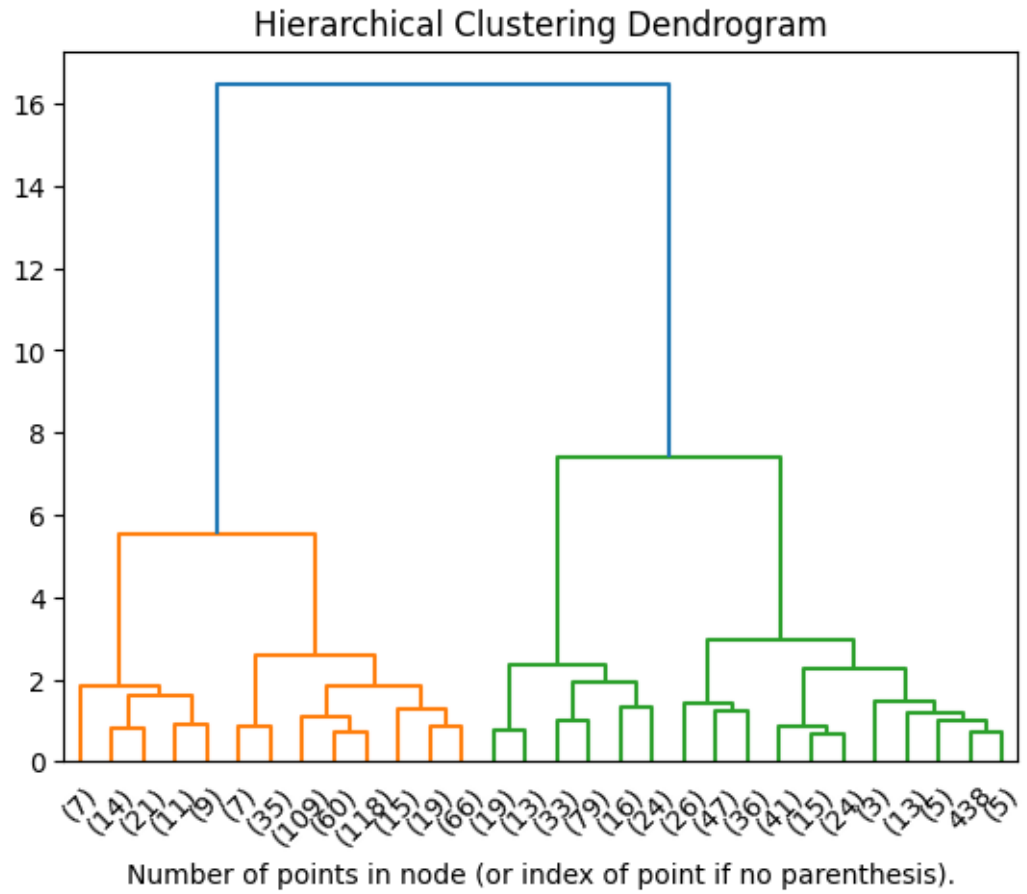
connectivity constraint

[ ]:
```
# hierarchical clustering
# Compute the (weighted) graph of k-Neighbors for points in X
```

```
connectivity = kneighbors_graph(train_data, n_neighbors=100, include_self=False)
```

[ ]:
```
# setting distance_threshold=0 ensures we compute the full tree.
model = AgglomerativeClustering(distance_threshold=0,
                                n_clusters=None,
                                metric='euclidean',
                                linkage='ward',
                                connectivity=connectivity)

# connecet: Defines for each sample the neighboring
# samples following a given structure of the data.

model = model.fit(train_data)
```

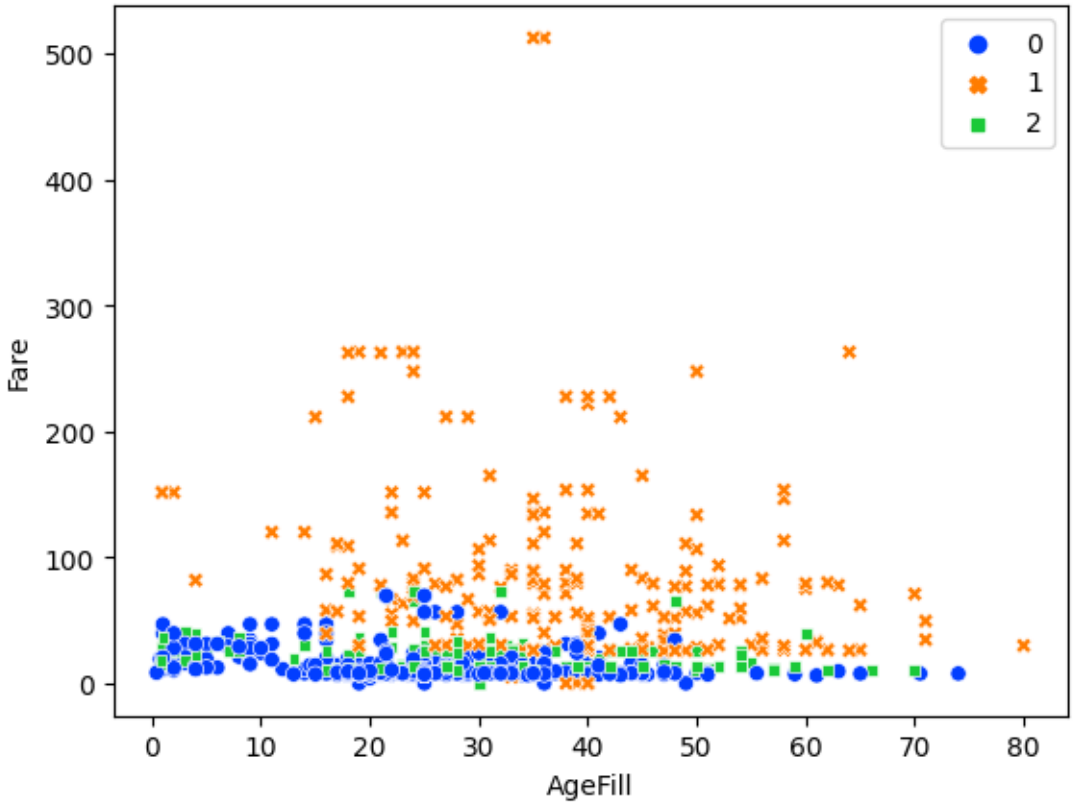
[ ]:
```
plt.title("Hierarchical Clustering Dendrogram")
plot_dendrogram(model, truncate_mode="lastp")
plt.xlabel("Number of points in node (or index of point if no parenthesis).")
plt.show()
```



[ ]:
```
ward = AgglomerativeClustering(n_clusters=3,
                               linkage='ward',
                               metric='euclidean',
                               connectivity=connectivity)
ward.fit(train_data)

hist, bins = np.histogram(ward.labels_, bins=range(0, len(set(ward.labels_)) +␣
 ↪1))
print('labels', dict(zip(bins, hist)))
print('silhouette', silhouette_score(train_data, ward.labels_))
```

```
labels {0: 491, 1: 216, 2: 184}
silhouette 0.5676262875725462
```

[ ]:
```
sns.scatterplot(data=df_clusters,
                x="AgeFill",
                y="Fare",
                hue=ward.labels_,
                style=ward.labels_,
                palette="bright")
plt.show()
```



[ ]:
```
print('average linkage')
average_linkage = AgglomerativeClustering(n_clusters=3, linkage='average',␣
 ↪metric='manhattan',  connectivity=connectivity)
average_linkage.fit(train_data)

hist, bins = np.histogram(average_linkage.labels_,
                          bins=range(0, len(set(average_linkage.labels_)) + 1))

print('labels', dict(zip(bins, hist)))
print('silhouette', silhouette_score(train_data, average_linkage.labels_))
```

```
average linkage
labels {0: 881, 1: 7, 2: 3}
silhouette 0.37132365914471405
```

[ ]:
```
print('complete linkage')
complete_linkage = AgglomerativeClustering(n_clusters=3, linkage='complete',␣
 ↪metric='l1', connectivity=connectivity)
complete_linkage.fit(train_data)

hist, bins = np.histogram(complete_linkage.labels_,
                          bins=range(0, len(set(complete_linkage.labels_)) + 1))

print('labels', dict(zip(bins, hist)))
print('silhouette', silhouette_score(train_data, complete_linkage.labels_))
```

```
complete linkage
labels {0: 882, 1: 3, 2: 6}
silhouette 0.35741232675404144
```

[ ]:
```

```

Categorical & Mixed distances

[54]:
```
cols2drop = ['PassengerId', 'Name', 'Cabin', 'Ticket', 'FamilySize', 'Sex_Val',␣
 ↪'Embarked_Val', 'Age']
df_xm = df.drop(cols2drop, axis=1)
df_xm['Pclass'] = df_xm['Pclass'].map({1: '1st', 2: '2nd', 3: '3rd'})
df_xm.head()
```

[54]:

	Survived	Pclass	Sex	SibSp	Parch	Fare	Embarked	AgeFill
0	0	3rd	male	1	0	7.2500	S	22.0
1	1	1st	female	1	0	71.2833	C	38.0
2	1	3rd	female	0	0	7.9250	S	26.0
3	1	1st	female	1	0	53.1000	S	35.0
4	0	3rd	male	0	0	8.0500	S	35.0

[55]:
```
df_xm2 = pd.get_dummies(df_xm[[c for c in df_xm.columns if c != 'Survived']],␣
↪prefix_sep='=')
df_xm2
```

[55]:

	SibSp	Parch	Fare	AgeFill	Pclass=1st	Pclass=2nd	Pclass=3rd	\
0	1	0	7.2500	22.0	False	False	True	
1	1	0	71.2833	38.0	True	False	False	
2	0	0	7.9250	26.0	False	False	True	
3	1	0	53.1000	35.0	True	False	False	
4	0	0	8.0500	35.0	False	False	True	
..	…	…	…	…	…	…	…	
886	0	0	13.0000	27.0	False	True	False	
887	0	0	30.0000	19.0	True	False	False	
888	1	2	23.4500	21.5	False	False	True	
889	0	0	30.0000	26.0	True	False	False	
890	0	0	7.7500	32.0	False	False	True	

	Sex=female	Sex=male	Embarked=C	Embarked=Q	Embarked=S
0	False	True	False	False	True
1	True	False	True	False	False
2	True	False	False	False	True
3	True	False	False	False	True
4	False	True	False	False	True
..	…	…	…	…	…
886	False	True	False	False	True
887	True	False	False	False	True
888	True	False	False	False	True
889	False	True	True	False	False
890	False	True	False	True	False

[891 rows x 12 columns]

[56]:
```
X = df_xm2[['Pclass=1st', 'Pclass=2nd',        'Pclass=3rd',␣
↪'Sex=female',        'Sex=male',        'Embarked=C',␣
↪'Embarked=Q',        'Embarked=S']].values
```

[57]:
```
X[:5]
```

[57]:
```
array([[False, False,  True, False,  True, False, False,  True],
       [ True, False, False,  True, False,  True, False, False],
       [False, False,  True,  True, False, False, False,  True],
       [ True, False, False,  True, False, False, False,  True],
       [False, False,  True, False,  True, False, False,  True]])
```

[58]:
```
D = pdist(X, 'jaccard')
D = squareform(D)
```

[59]:
```
D
```

[59]:
```
array([[0. , 1. , 0.5, …, 0.5, 0.8, 0.5],
       [1. , 0. , 0.8, …, 0.8, 0.5, 1. ],
       [0.5, 0.8, 0. , …, 0. , 1. , 0.8],
       …,
       [0.5, 0.8, 0. , …, 0. , 1. , 0.8],
       [0.8, 0.5, 1. , …, 1. , 0. , 0.8],
       [0.5, 1. , 0.8, …, 0.8, 0.8, 0. ]])
```

[59]:
```

```

[60]:
```
# Mixed custom distance
```

[61]:
```
from scipy.spatial.distance import seuclidean, jaccard
```

[ ]:
```
def mixed(a, b):
    index = 4
    d_con = seuclidean(a[:index], b[:index], V=np.ones(index))
    w_con = index/len(a)
    d_cat = jaccard(a[index:], b[index:])
    w_cat = (len(a)-index)/len(a)
    d = w_con * d_con + w_cat * d_cat
    return d
```

[ ]:
```
df_xm2.head()
```

[ ]:
```
   SibSp  Parch     Fare  AgeFill  Pclass=1st  Pclass=2nd  Pclass=3rd  \
0      1      0   7.2500     22.0       False       False        True
1      1      0  71.2833     38.0        True       False       False
2      0      0   7.9250     26.0       False       False        True
3      1      0  53.1000     35.0        True       False       False
4      0      0   8.0500     35.0       False       False        True

   Sex=female  Sex=male  Embarked=C  Embarked=Q  Embarked=S
0       False      True       False       False        True
1        True     False        True       False       False
2        True     False       False       False        True
3        True     False       False       False        True
4       False      True       False       False        True
```

[ ]:
```
X = df_xm2.values
```

[ ]:
```
X[:5]
```

[ ]:
```
array([[1, 0, 7.25, 22.0, False, False, True, False, True, False, False,
        True],
       [1, 0, 71.2833, 38.0, True, False, False, True, False, True,
```

```
 False, False],
[0, 0, 7.925, 26.0, False, False, True, True, False, False, False,
 True],
[1, 0, 53.1, 35.0, True, False, False, True, False, False, False,
 True],
[0, 0, 8.05, 35.0, False, False, True, False, True, False, False,
 True]], dtype=object)
```

[ ]:
```
mixed(X[0], X[10])
```

[ ]:
```
7.118140707983781
```

[ ]:
```
D = pdist(X, mixed)
D = squareform(D)
```

[ ]:
```
D
```

[ ]:
```
array([[ 0.        , 22.66733208,  1.72599765, …,  5.77688201,
         8.24020256,  3.6874353 ],
       [22.66733208,  0.        , 22.03081168, …, 17.41288898,
        14.6678715 , 21.94127388],
       [ 1.72599765, 22.03081168,  0.        , …,  5.43931802,
         8.025     ,  2.53418385],
       …,
       [ 5.77688201, 17.41288898,  5.43931802, …,  0.        ,
         3.41848425,  6.87315457],
       [ 8.24020256, 14.6678715 ,  8.025     , …,  3.41848425,
         0.        ,  8.21493111],
       [ 3.6874353 , 21.94127388,  2.53418385, …,  6.87315457,
         8.21493111,  0.        ]])
```

[ ]:
```

```

0.0.6 K-Mode

https://github.com/nicodv/kmodes

[62]:
```
!pip install kmodes
```

```
Collecting kmodes
  Downloading kmodes-0.12.2-py2.py3-none-any.whl.metadata (8.1 kB)
Requirement already satisfied: numpy>=1.10.4 in /usr/local/lib/python3.12/dist-
packages (from kmodes) (2.0.2)
Requirement already satisfied: scikit-learn>=0.22.0 in
/usr/local/lib/python3.12/dist-packages (from kmodes) (1.6.1)
Requirement already satisfied: scipy>=0.13.3 in /usr/local/lib/python3.12/dist-
packages (from kmodes) (1.16.3)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.12/dist-
packages (from kmodes) (1.5.2)
```

```
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn>=0.22.0->kmodes)
(3.6.0)
Downloading kmodes-0.12.2-py2.py3-none-any.whl (20 kB)
Installing collected packages: kmodes
Successfully installed kmodes-0.12.2
```

[ ]:
```
from kmodes.kmodes import KModes
```

[ ]:
```
X = df[['Pclass', 'Sex', 'Embarked']].values
```

[ ]:
```
X[:5]
```

[ ]:
```
array([[3, 'male', 'S'],
       [1, 'female', 'C'],
       [3, 'female', 'S'],
       [1, 'female', 'S'],
       [3, 'male', 'S']], dtype=object)
```

[ ]:
```
km = KModes(n_clusters=4, init='Huang', n_init=5, verbose=0)
clusters = km.fit_predict(X)
```

[ ]:
```
km.cluster_centroids_
```

[ ]:
```
array([['3', 'female', 'S'],
       ['1', 'male', 'C'],
       ['3', 'male', 'S'],
       ['2', 'male', 'S']], dtype='<U21')
```

[ ]:
```
km.labels_
```

[ ]:
```
array([2, 1, 0, 0, 2, 2, 1, 2, 0, 0, 0, 0, 2, 2, 0, 0, 2, 3, 0, 0, 3, 3,
       0, 1, 0, 0, 1, 1, 0, 2, 1, 1, 0, 3, 1, 1, 1, 2, 0, 0, 0, 0, 1, 0,
       0, 2, 2, 0, 1, 0, 2, 2, 1, 0, 1, 1, 0, 1, 0, 2, 1, 0, 1, 2, 1, 1,
       0, 2, 0, 2, 3, 0, 3, 1, 2, 2, 2, 2, 3, 0, 2, 2, 0, 1, 0, 0, 2, 2,
       0, 2, 2, 2, 1, 2, 2, 2, 1, 1, 0, 3, 0, 2, 1, 2, 2, 2, 0, 2, 2, 0,
       1, 0, 2, 0, 0, 2, 2, 3, 1, 0, 3, 2, 1, 0, 1, 1, 2, 2, 0, 2, 1, 2,
       0, 0, 3, 1, 0, 1, 2, 1, 0, 0, 0, 2, 3, 3, 2, 0, 3, 3, 3, 0, 2, 2,
       2, 1, 0, 2, 2, 2, 2, 0, 2, 2, 2, 2, 0, 0, 1, 2, 1, 2, 0, 2, 1, 2,
       2, 1, 3, 2, 0, 1, 2, 3, 0, 1, 0, 1, 2, 2, 0, 3, 0, 3, 1, 1, 2, 2,
       0, 0, 2, 2, 2, 1, 2, 0, 2, 1, 0, 1, 2, 0, 2, 3, 2, 1, 0, 3, 1, 3,
       2, 3, 2, 2, 1, 2, 3, 2, 3, 0, 0, 2, 3, 0, 3, 0, 3, 0, 3, 3, 0, 0,
       3, 2, 1, 1, 0, 0, 1, 3, 2, 0, 1, 2, 0, 0, 1, 0, 1, 0, 2, 2, 1, 1,
       0, 3, 2, 2, 0, 0, 1, 2, 0, 1, 0, 0, 0, 3, 2, 0, 2, 2, 2, 2, 1, 1,
       2, 2, 3, 0, 0, 1, 1, 0, 2, 1, 1, 0, 1, 1, 0, 2, 2, 0, 2, 1, 1, 1,
       1, 1, 1, 1, 0, 2, 3, 0, 0, 3, 0, 1, 2, 2, 0, 0, 2, 1, 2, 0, 0, 1,
       0, 1, 1, 2, 0, 2, 1, 1, 2, 1, 3, 0, 3, 3, 3, 0, 0, 0, 2, 2, 2, 1,
       1, 2, 1, 2, 0, 0, 0, 0, 2, 1, 0, 2, 2, 2, 1, 0, 0, 1, 1, 2, 2, 1,
```

```
0, 1, 0, 1, 1, 2, 1, 0, 2, 0, 2, 3, 2, 0, 2, 0, 1, 2, 2, 1, 0, 2,
0, 3, 3, 0, 2, 2, 0, 2, 0, 3, 2, 3, 2, 0, 2, 2, 0, 3, 2, 0, 0, 0,
3, 0, 1, 2, 2, 0, 2, 2, 0, 0, 2, 2, 1, 0, 0, 2, 1, 0, 0, 0, 1, 3,
0, 2, 2, 0, 2, 1, 0, 1, 0, 1, 3, 2, 1, 1, 2, 1, 1, 0, 0, 2, 1, 2,
1, 3, 2, 2, 3, 1, 2, 0, 2, 2, 0, 0, 0, 1, 3, 2, 2, 0, 2, 3, 2, 0,
1, 0, 0, 1, 2, 2, 2, 2, 1, 1, 2, 1, 1, 2, 0, 2, 2, 0, 0, 0, 0, 1,
0, 1, 2, 2, 2, 2, 1, 1, 2, 1, 0, 2, 0, 2, 0, 2, 1, 1, 1, 2, 0, 1,
2, 3, 0, 1, 1, 0, 0, 0, 1, 1, 2, 1, 0, 0, 0, 3, 1, 1, 0, 1, 2, 3,
1, 3, 2, 1, 0, 1, 1, 1, 0, 0, 2, 2, 3, 2, 0, 2, 2, 0, 1, 2, 3, 0,
1, 0, 2, 2, 0, 0, 0, 2, 0, 1, 3, 1, 1, 0, 3, 1, 2, 2, 2, 1, 2, 0,
3, 2, 0, 2, 1, 1, 0, 2, 1, 2, 1, 2, 2, 1, 0, 0, 0, 2, 0, 2, 2, 0,
2, 0, 0, 3, 1, 1, 1, 2, 2, 1, 3, 0, 2, 2, 1, 2, 1, 1, 0, 0, 2, 3,
0, 2, 2, 1, 0, 2, 0, 1, 2, 1, 2, 0, 2, 0, 2, 0, 0, 3, 2, 0, 3, 1,
1, 1, 1, 2, 2, 3, 3, 2, 2, 0, 0, 1, 3, 3, 3, 2, 2, 0, 0, 1, 0, 1,
2, 2, 3, 1, 2, 2, 2, 0, 1, 0, 2, 1, 1, 3, 2, 0, 1, 2, 1, 1, 0, 2,
2, 3, 0, 1, 0, 1, 1, 1, 1, 2, 3, 2, 1, 0, 2, 2, 0, 2, 3, 3, 1, 2,
0, 0, 3, 0, 0, 1, 3, 3, 3, 2, 0, 1, 2, 2, 1, 1, 1, 2, 2, 1, 2, 0,
1, 2, 0, 2, 2, 2, 0, 3, 2, 3, 2, 0, 2, 2, 1, 0, 2, 0, 1, 0, 2, 2,
2, 2, 0, 1, 0, 2, 2, 0, 2, 0, 0, 0, 1, 2, 2, 2, 0, 2, 2, 1, 2, 3,
0, 1, 2, 3, 0, 0, 1, 0, 3, 0, 1, 1, 2, 2, 1, 0, 3, 0, 2, 2, 3, 0,
2, 1, 0, 1, 2, 2, 0, 2, 1, 0, 2, 2, 2, 1, 2, 0, 0, 3, 1, 2, 2, 1,
2, 2, 2, 1, 2, 3, 1, 1, 2, 2, 2, 1, 3, 1, 2, 2, 0, 0, 0, 0, 0, 1,
0, 1, 2, 3, 0, 0, 3, 0, 0, 1, 2, 2, 2, 0, 1, 2, 0, 0, 2, 2, 2, 1,
0, 2, 0, 3, 2, 0, 3, 0, 0, 1, 2], dtype=uint16)
```

[ ]:

clustering_others

November 11, 2025

Author: Riccardo Guidotti
Python version: 3.x

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from collections import defaultdict
```

[2]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import silhouette_score
```

1 Data Preparation

[3]:
```
from sklearn.datasets import load_breast_cancer

frame = load_breast_cancer(as_frame=True)
df = frame['data']
X = df.values
y = np.array(frame['target'])

df.head()
```

[3]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	\
0	17.99	10.38	122.80	1001.0	0.11840	
1	20.57	17.77	132.90	1326.0	0.08474	
2	19.69	21.25	130.00	1203.0	0.10960	
3	11.42	20.38	77.58	386.1	0.14250	
4	20.29	14.34	135.10	1297.0	0.10030	

	mean compactness	mean concavity	mean concave points	mean symmetry	\
0	0.27760	0.3001	0.14710	0.2419	
1	0.07864	0.0869	0.07017	0.1812	
2	0.15990	0.1974	0.12790	0.2069	
3	0.28390	0.2414	0.10520	0.2597	
4	0.13280	0.1980	0.10430	0.1809	

```
   mean fractal dimension  …  worst radius  worst texture  worst perimeter  \
0                 0.07871  …         25.38          17.33           184.60
1                 0.05667  …         24.99          23.41           158.80
2                 0.05999  …         23.57          25.53           152.50
3                 0.09744  …         14.91          26.50            98.87
4                 0.05883  …         22.54          16.67           152.20

   worst area  worst smoothness  worst compactness  worst concavity  \
0      2019.0            0.1622             0.6656           0.7119
1      1956.0            0.1238             0.1866           0.2416
2      1709.0            0.1444             0.4245           0.4504
3       567.7            0.2098             0.8663           0.6869
4      1575.0            0.1374             0.2050           0.4000

   worst concave points  worst symmetry  worst fractal dimension
0                0.2654          0.4601                  0.11890
1                0.1860          0.2750                  0.08902
2                0.2430          0.3613                  0.08758
3                0.2575          0.6638                  0.17300
4                0.1625          0.2364                  0.07678

[5 rows x 30 columns]
```

[4]:
```
scaler = MinMaxScaler()
```

[5]:
```
X = scaler.fit_transform(X)
```

2 X-Means

https://github.com/annoviko/pyclustering/

[2]:
```
#!pip install pyclustering
```

[5]:
```
# standard installation might result in error due to numpy warnings (numpy > 1.
↪24.0)

#after installing pyclustering, also install this warning fix below

#!pip install https://github.com/KulikDM/pyclustering/archive/Warning-Fix.zip
```

[4]:
```
from pyclustering.cluster import xmeans
```

[7]:
```
xm = xmeans.xmeans(X)
xm.process()
```

[7]:
```
<pyclustering.cluster.xmeans.xmeans at 0x7ee062164710>
```

[8]:
```
clusters = xm.get_clusters()
```

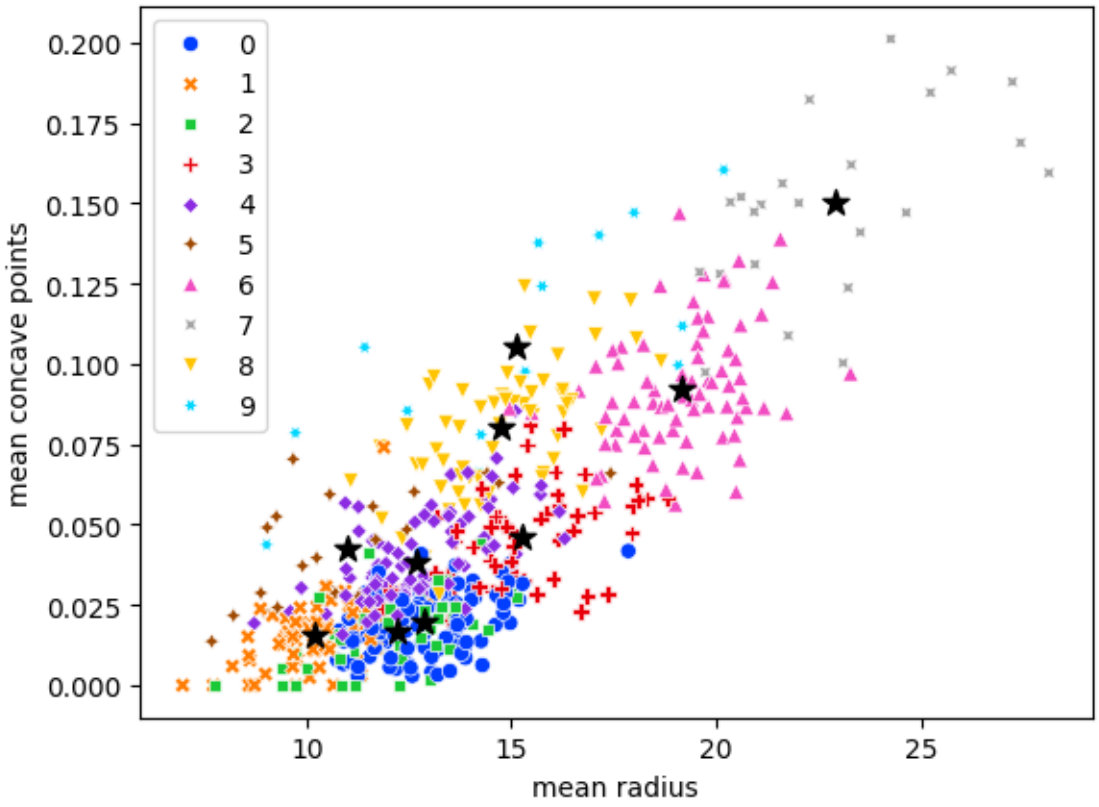
[9]:
```
def clusters_to_labels(clusters):
    labels = np.empty(shape=(len(np.concatenate(clusters))), dtype=int)
    for i in range(len(clusters)):
        for idx in clusters[i]:
            labels[idx] = i
    return labels
```

[10]:
```
labels = clusters_to_labels(clusters)
```

[11]:
```
centers = np.array(xm.get_centers())
centers_unscaled = scaler.inverse_transform(centers)
```

[12]:
```
sns.scatterplot(data=df, x="mean radius", y="mean concave points", hue=labels,␣
↪palette="bright", style=labels)
plt.scatter(centers_unscaled[:, 0], centers_unscaled[:,7], color="black",␣
↪marker="*", s=100)
```

[12]:
```
<matplotlib.collections.PathCollection at 0x7ee03d6abda0>
```



3 BISECTING K-MEANS

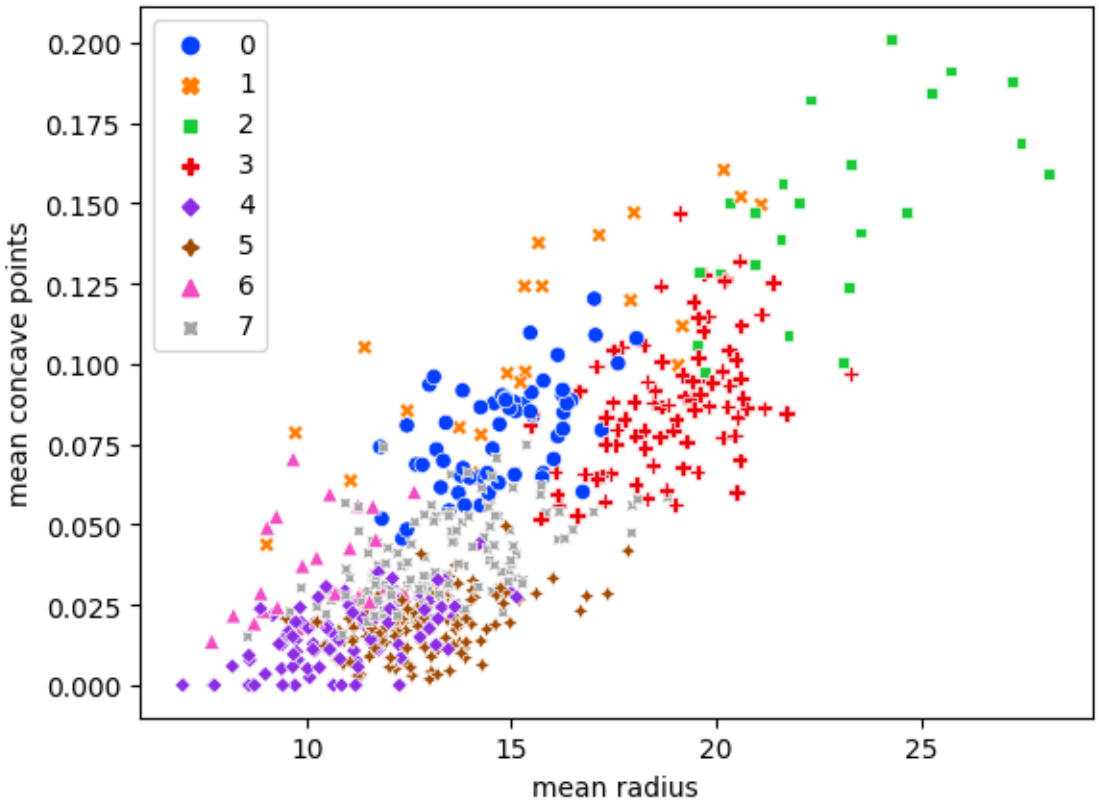
[ ]:
```
from sklearn.cluster import BisectingKMeans
```

[ ]:
```
bkmeans = BisectingKMeans(n_clusters=8)
bkmeans.fit(X)
```

[ ]:
```
BisectingKMeans()
```

[ ]:
```
sns.scatterplot(data=df, x="mean radius", y="mean concave points", hue=bkmeans.
↪labels_,
                palette="bright", style=bkmeans.labels_)
```

[ ]:
```
<Axes: xlabel='mean radius', ylabel='mean concave points'>
```

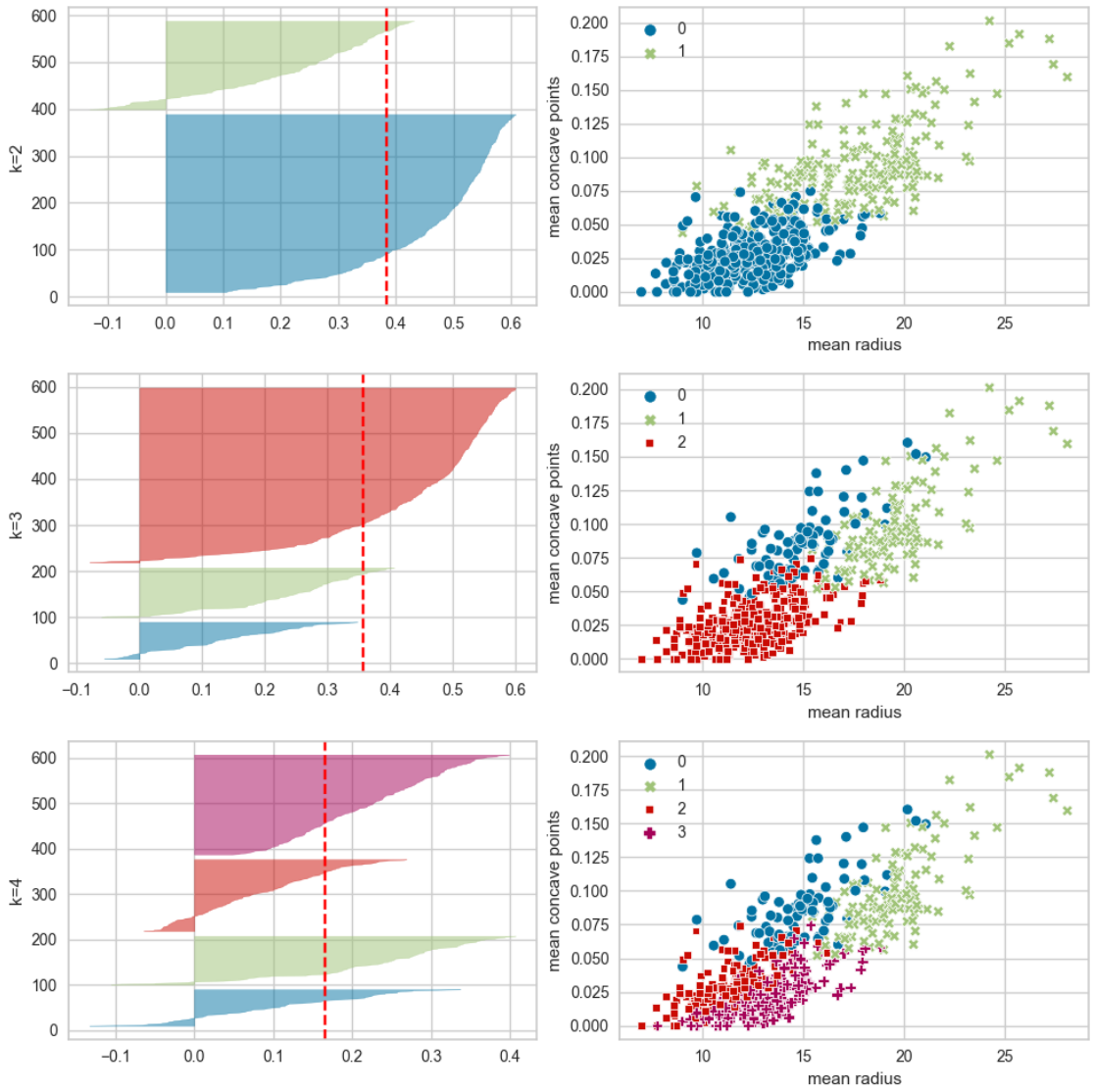


Silhouette plot

[ ]:
```
# !pip install yellowbrick
```

[ ]:
```
from yellowbrick.cluster import SilhouetteVisualizer
```

```
[ ]: colors=['C0', 'C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9']
n_clust = 5
fig, axs = plt.subplots(n_clust-2, 2, figsize=(10,10))
for i in range(2, n_clust):
    bkmeans = BisectingKMeans(n_clusters=i)
    visualizer = SilhouetteVisualizer(bkmeans, colors=colors, ax=axs[i-2][0])
    axs[i-2][0].set_ylabel("k=" + str(i))
    visualizer.fit(X)
    sns.scatterplot(data=df, x="mean radius", y="mean concave points",␣
 ↪hue=bkmeans.labels_,
                    palette=sns.color_palette(colors[:i]), style=bkmeans.
 ↪labels_, ax=axs[i-2][1])
plt.tight_layout()
```



[ ]:
```
silhouette_score(X, bkmeans.labels_)
```

[ ]:
```
0.16557736812746163
```

4 OPTICS

[ ]:
```
from sklearn.cluster import OPTICS
```

[ ]:
```
optics = OPTICS(min_samples=5, max_eps=np.inf)
optics.fit(X)
```

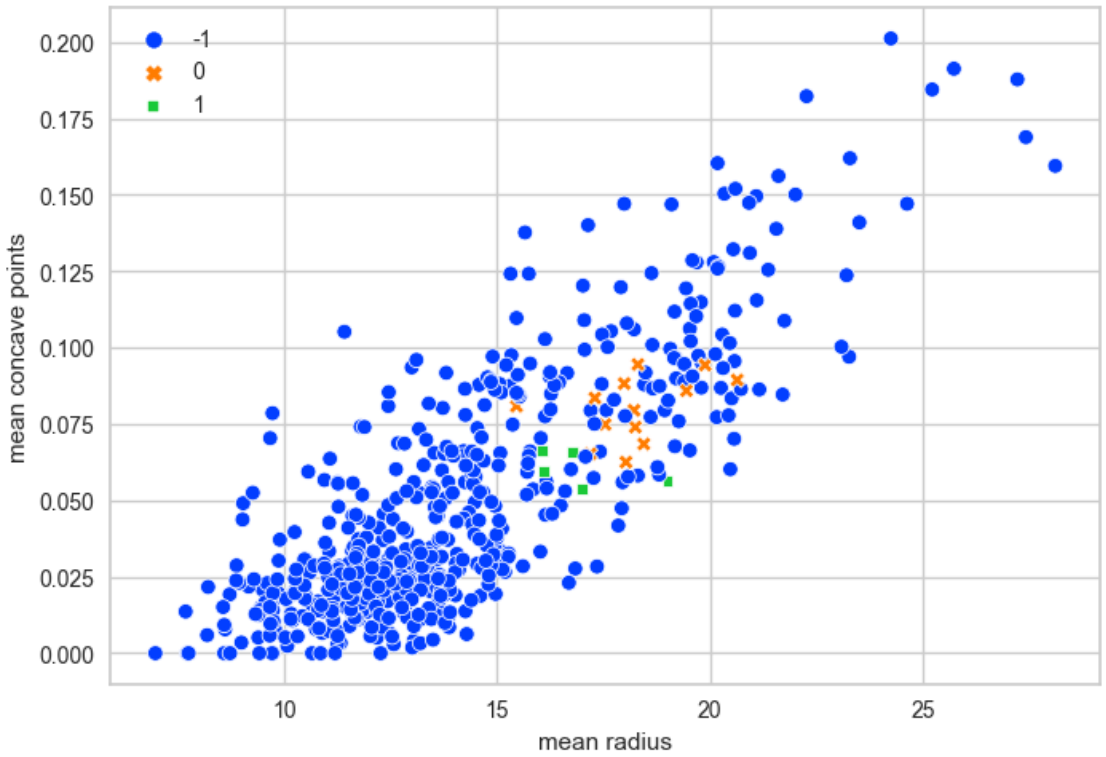
[ ]:
```
OPTICS()
```

[ ]:
```
silhouette_score(X[optics.labels_ != -1], optics.labels_[optics.labels_ != -1])
```

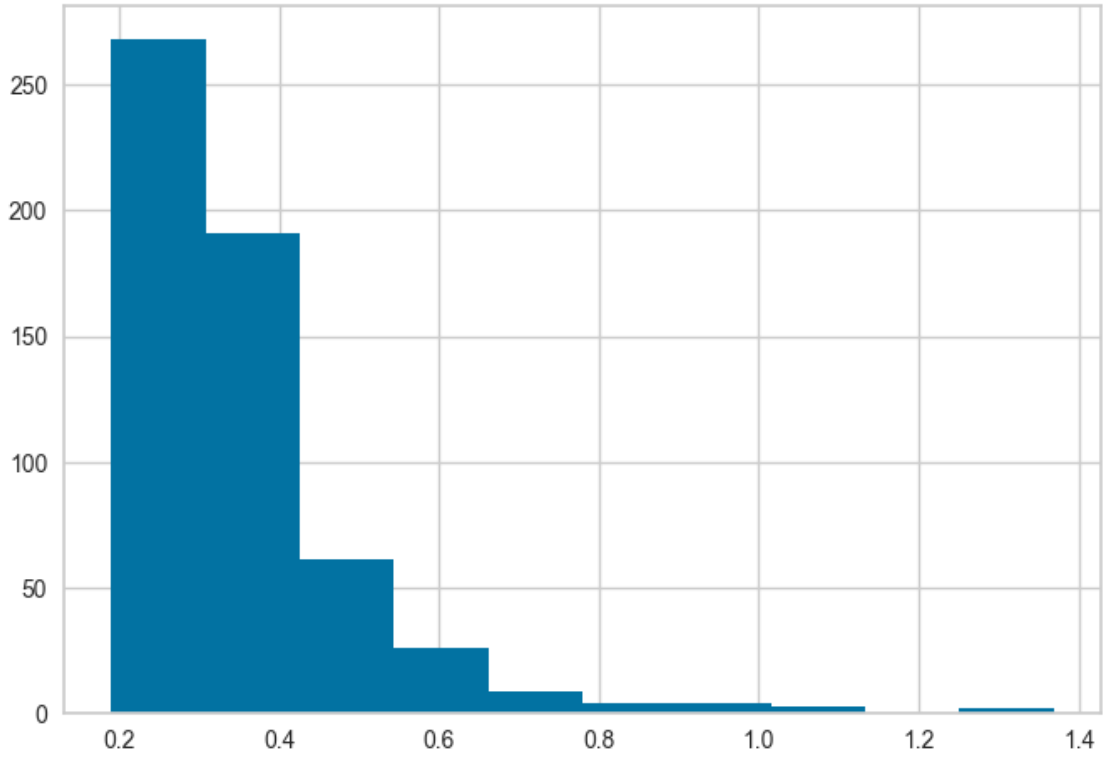
[ ]:
```
0.29091187971598037
```

[ ]:
```
sns.scatterplot(data=df, x="mean radius", y="mean concave points", hue=optics.
↪labels_,
                palette="bright", style=optics.labels_)
```

[ ]:
```
<Axes: xlabel='mean radius', ylabel='mean concave points'>
```



[ ]:
```
plt.hist(optics.reachability_[1:])
plt.show()
```



[ ]:
```
optics = OPTICS(min_samples=5, max_eps=np.inf, cluster_method='dbscan', eps=0.3)
optics.fit(X)
```

[ ]:
```
OPTICS(cluster_method='dbscan', eps=0.3)
```

[ ]:
```
silhouette_score(X[optics.labels_ != -1], optics.labels_[optics.labels_ != -1])
```

[ ]:
```
0.07553499719530653
```

[ ]:
```
np.unique(optics.labels_)
```

[ ]:
```
array([-1,  0,  1,  2,  3])
```

[ ]:
```
np.unique(optics.labels_, return_counts=True)
```

[ ]:
```
(array([-1,  0,  1,  2,  3]), array([316,  13, 231,   4,   5]))
```

[ ]:
```
# https://scikit-learn.org/stable/auto_examples/cluster/plot_optics.html
```