

# DAT630 Clustering

Introduction to Data Mining, Chapter 8

16/10/2017

Dario Garigliotti | University of Stavanger

## Supervised vs. Unsupervised Learning

- Supervised learning
  - Labeled examples (with target information) are available
- Unsupervised learning
  - Examples are not labeled

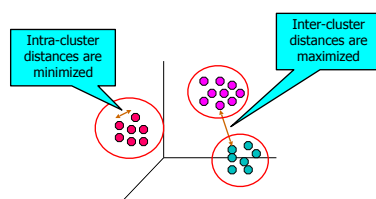
## Outline

- Clustering
- Two algorithms:
  - K-means
  - Hierarchical (Agglomerative) Clustering

## Clustering

## Clustering

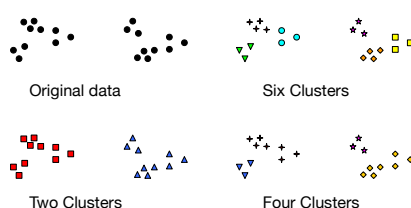
- Finding groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects in other groups



## Why?

- For **understanding**
  - E.g., biology (taxonomy of species)
  - Business (segmenting customers for additional analysis and marketing activities)
  - Web (clustering search results into subcategories)
- For **utility**
  - Some clustering techniques characterize each cluster in terms of a cluster prototype
  - These prototypes can be used as a basis for a number of data analysis and processing techniques

## How many clusters?



- The notion of a cluster can be ambiguous

## Types of Clustering

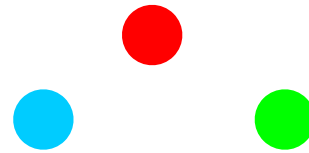
- Partitional vs. hierarchical
  - Partitional: non-overlapping clusters such that each data object is in exactly one cluster
  - Hierarchical: a set of nested clusters organized as a hierarchical tree
- Exclusive vs. non-exclusive
  - Whether points may belong to a single or multiple clusters

## Types of Clustering (2)

- Partial versus complete
  - In some cases, we only want to cluster some of the data
- Fuzzy vs. 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

## Different Types of Clusters

- 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



## Different Types of Clusters

- Center-based (or prototype-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



## Different Types of Clusters

- Shared Property or Conceptual Clusters
  - Clusters that share some common property or represent a particular concept



## Notation

- $x$  an object (data point)
- $m$  the number of points in the data set
- $K$  the number of clusters
- $C_i$  the  $i$ th cluster
- $c_i$  the centroid of cluster  $C_i$
- $m_i$  the number of points in cluster  $C_i$

## K-means Clustering

## K-means

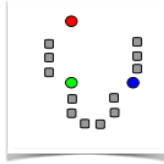
- One of the oldest and most widely used clustering techniques
- Prototype-based clustering
  - Clusters are represented by their centroids
- Finds a user-specified number of clusters ( $K$ )

## Basic K-means Algorithm

1. Select  $K$  points as initial centroids
2. **repeat**
  3. Form  $K$  clusters by assigning each point to its closest centroid
  4. Recompute the centroid of each cluster
5. **until** centroids do not change

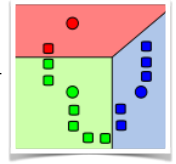
## Basic K-means Algorithm

1. Select  $K$  points as initial centroids
2. **repeat**
3. Form  $K$  clusters by assigning each point to its closest centroid
4. Recompute the centroid of each cluster
5. **until** centroids do not change



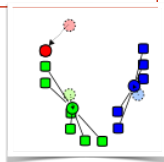
## Basic K-means Algorithm

1. Select  $K$  points as initial centroids
2. **repeat**
3. Form  $K$  clusters by assigning each point to its closest centroid
4. Recompute the centroid of each cluster
5. **until** centroids do not change



## Basic K-means Algorithm

1. Select  $K$  points as initial centroids
2. **repeat**
3. Form  $K$  clusters by assigning each point to its closest centroid
4. Recompute the centroid of each cluster
5. **until** centroids do not change



## Basic K-means Algorithm

1. Select  $K$  points as initial centroids
2. **repeat**
3. Form  $K$  clusters by assigning each point to its closest centroid
4. Recompute the centroid of each cluster
5. **until** centroids do not change



## Basic K-means Algorithm

1. Select  $K$  points as initial centroids
2. **repeat**
3. Form  $K$  clusters by assigning each point to its closest centroid
4. Recompute the centroid of each cluster
5. **until** centroids do not change



## 1. Choosing Initial Centroids

- Most commonly: select points (centroids) randomly
- They may be poor
- Possible solution: perform multiple runs, each with a different set of randomly chosen centroids

## 3. Assigning Points to the Closest Centroid

- We need a proximity measure that quantifies the notion of "closest"
- Usually chosen to be simple
  - Has to be calculated repeatedly
- See distance functions from Lecture 1
  - E.g., Euclidean distance

## 4. Recomputing Centroids

- Objective function is selected
  - I.e., what is it that we want minimize/maximize
- Once the objective function and the proximity measure are defined, we can define mathematically the centroid we should choose
  - E.g., minimize the squared distance of each point to its closest centroid

## Sum of Squared Error (SSE)

- Measures the quality of clustering in the Euclidean space
- Calculate the error of each data point (its Euclidean distance to the closest centroid), and then compute the total sum of the squared errors

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

- A clustering with lower SSE is better

## Minimizing SSE

- It can be shown that the centroid that minimizes the SSE of the cluster is the mean
- The centroid of the  $i$ th cluster

$$c_i = \frac{1}{m_i} \sum_{x \in C_i} x$$

## Example Centroid computation

- What is the centroid of a cluster containing three 2-dimensional points: **(1,1), (2,3), (6,2)**?
- Centroid:  
 $((1+2+6)/3, (1+3+2)/3) = \mathbf{(3,2)}$

## 5. Stopping Condition

- Most of the convergence occurs in the early steps
- "Centroids do not change" is often replaced with a weaker condition
  - E.g., repeat until only 1% of the points change

## Exercise

## Note

- There are other choices for proximity, centroids, and objective functions, e.g.,
- Proximity function: **Manhattan** (L1)  
Centroid: median  
Objective function: minimize sum of L1 distance of an object to its cluster centroid
- Proximity function: **cosine**  
Centroid: mean  
Objective function: maximize sum of cosine sim. of an object to its cluster centroid

## What is the complexity?

- $m$  number of points,  $n$  number of attributes,  $K$  number of clusters
- Space requirements:  $O(?)$
- Time requirements:  $O(?)$

## Complexity

- $m$  number of points,  $n$  number of attributes,  $K$  number of clusters
- Space requirements:  $O((m+K)*n)$ 
  - Modest, as only the data points and centroids are stored
- Time requirements:  $O(l*K*m*n)$ 
  - $l$  is the number of iterations required for convergence
  - Modest, linear in the number of data points

## K-means Issues

- Depending on the initial (random) selection of centroids different clustering can be produced
- Steps 3 and 4 are only guaranteed to find a local optimum
- Empty clusters may be obtained
  - replacement of centroid by (i) farthest point to any other centroid, or (ii) chosen among those in the cluster with highest SSE

## K-means Issues (2)

- Presence of outliers must sometimes be kept
  - E.g. all points must be clustered in data compression
- In general outliers may be addressed by eliminating them to improve clustering
  - Before: by outlier detection techniques
  - After: eliminating (i) points whose SSE is high, or (ii) directly small clusters as likely outliers

## K-means Issues (3)

- Postprocessing for reducing SSE
  - and ideally not introducing more clusters
  - How? Alternating splitting and merging steps
- Decrease SSE by more clusters
  - Splitting (e.g., cluster with highest SSE)
  - Introducing new centroid
- Less clusters by trying not to increase SSE
  - Dispersing (a cluster e.g. with lowest SSE)
  - Merging 2 clusters (e.g. with closest centroids)

## Bisecting K-means

- Straightforward extension of the basic K-means algorithm
- Idea:
  - Split the set of data points to two clusters
  - Select one of these clusters to split
  - Repeat until  $K$  clusters have been produced
- The resulting clusters are often used as the initial centroids for the basic K-means algorithm

## Bisecting K-means Alg.

- 
1. Initial cluster contains all data points
  2. **repeat**
  3. Select a cluster to split
  4. **for** a number of trials
    5. Bisect the selected cluster using basic K-means
  6. **end for**
  7. Select the clusters from the bisection with the lowest total SSE
  8. **until** we have  $K$  clusters
- 

## Selecting a Cluster to Split

- Number of possible ways
  - Largest cluster
  - Cluster with the largest SSE
  - Combine size and SSE
- Different choices result in different clusters

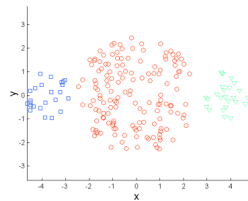
## Hierarchical Clustering

- By recording the sequence of clusterings produced, bisecting K-means can also produce a hierarchical clustering

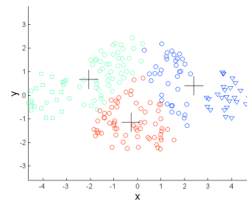
## Limitations

- K-means has difficulties detecting clusters when they have
  - differing sizes
  - differing densities
  - non-spherical shapes
- K-means has problems when the data contains outliers

## Example: differing sizes

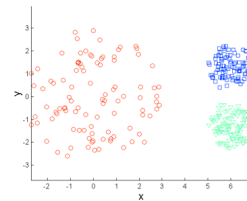


Original points

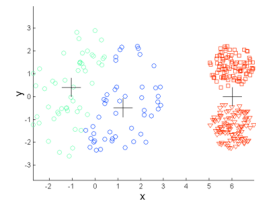


K-means (3 clusters)

## Example: differing density

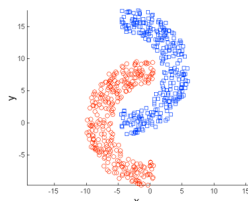


Original points

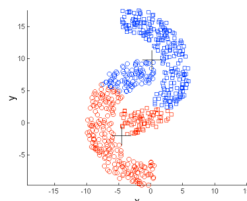


K-means (3 clusters)

## Example: non-spherical shapes



Original points

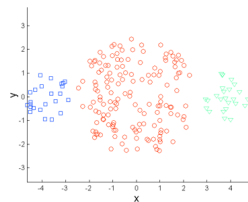


K-means (2 clusters)

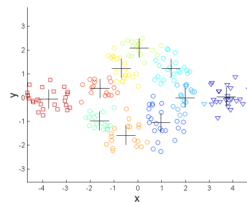
## Overcoming Limitations

- Use larger K values
- Natural clusters will be broken into a number of sub-clusters

## Example: differing sizes

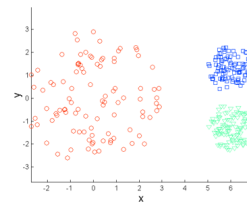


Original points

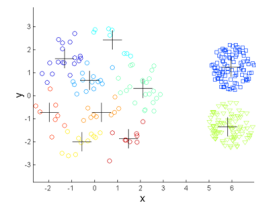


K-means clusters

## Example: differing density

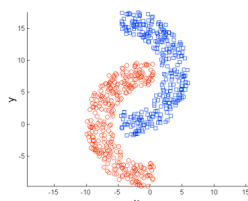


Original points

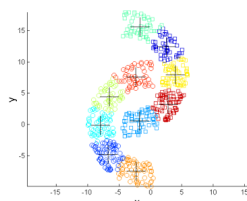


K-means clusters

## Example: non-spherical shapes



Original points



K-means clusters

## Summary

- Efficient and simple
  - Provided that K is relatively small ( $K \ll m$ )
- Bisecting variant is even more efficient and less susceptible to initialization problems
- Cannot handle certain types of clusters
  - Problems can be overcome by generating more (sub)clusters
- Has trouble with data that contains outliers
  - Outlier detection and removal can help

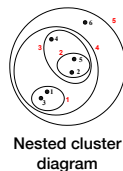
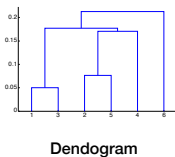
# Hierarchical Clustering

# Hierarchical Clustering

- Two general approaches
- ➔ - Agglomerative
  - Start with the points as individual clusters
  - At each step, merge the closest pair of clusters
  - Requires a notion of *cluster proximity*
- Divisive
  - Start with a single, all-inclusive cluster
  - At each step, split a cluster, until only singleton clusters of individual points remain

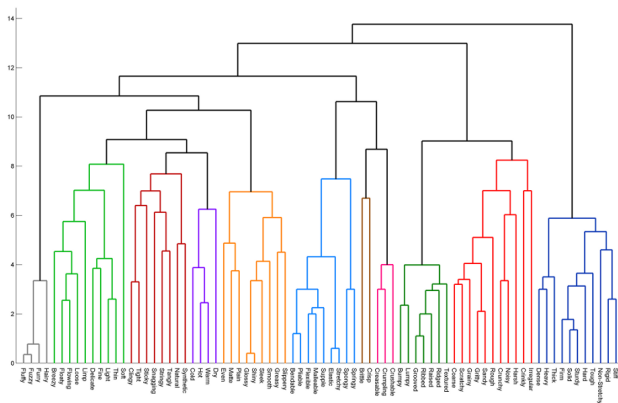
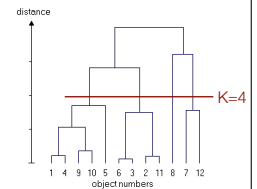
## Agglomerative Hierarchical Clustering

- Produces a set of nested clusters organized as a hierarchical tree
- Can be visualized
  - Dendrogram
  - Nested cluster diagram (only for 2D points)



## Strengths

- 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
  - E.g., in biological sciences

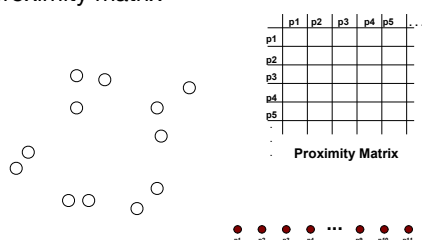


## Basic Agglomerative Hierarchical Clustering Alg.

1. Compute the proximity matrix
2. **repeat**
  3. Merge the closest two clusters
  4. Update the proximity matrix
5. **until** only one cluster remains

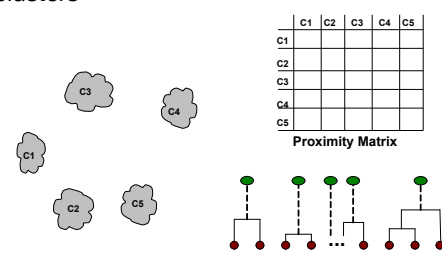
## Example Starting situation

- Start with clusters of individual points and a proximity matrix



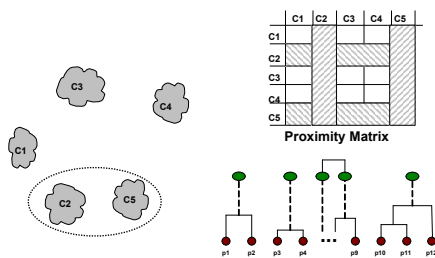
## Example Intermediate situation

- After some merging steps, we have some clusters



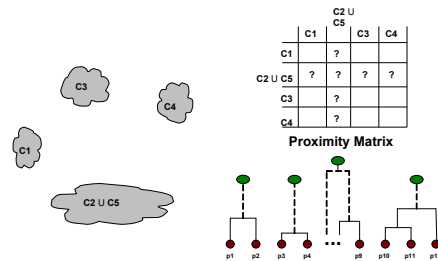
## Example Intermediate situation

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



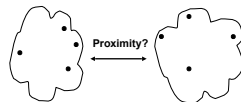
## Example After merging

- How do we update the proximity matrix?



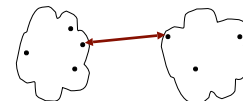
## Defining the Proximity between Clusters

- MIN (single link)
- MAX (complete link)
- Group average
- Distance between centroids



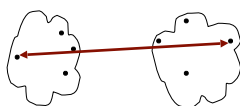
## Single link (MIN)

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



## Complete link (MAX)

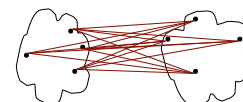
- Proximity of two clusters is based on the two least similar (most distant) points in the different clusters
- Determined by all pairs of points in the two clusters



## Group average

- Proximity of two clusters is the average of pairwise proximity between points in the two clusters
- Need to use average connectivity for scalability since total proximity favors large clusters

$$proximity(C_i, C_j) = \frac{\sum_{x \in C_i, y \in C_j} proximity(x, y)}{m_i \cdot m_j}$$

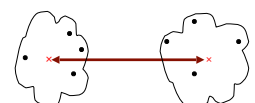


## Strengths and Weaknesses

- Single link (MIN)
  - Strength: can handle non-elliptical shapes
  - Weakness: sensitive to noise and outliers
- Complete link (MAX)
  - Strength: less susceptible to noise and outliers
  - Weakness: tends to break large clusters
- Group average
  - Strength: less susceptible to noise and outliers
  - Weakness: biased towards globular clusters

## Prototype-based methods

- Represent clusters by their centroids
  - Calculate the proximity based on the distance between the centroids of the clusters
- Ward's method
  - Similarity of two clusters is based on the increase in SSE when two clusters are merged
  - Very similar to group average if distance between points is distance squared





## Exercise

## Key Characteristics

- No global objective function that is directly optimized
- No problems with choosing initial points or running into local minima
- Merging decisions are final
  - Once a decision is made to combine two clusters, it cannot be undone

## What is the complexity?

- $m$  is the number of points
- Space complexity  $O(?)$
- Time complexity  $O(?)$

## Complexity

- Space complexity  $O(m^2)$ 
  - Proximity matrix requires the storage of  $m^2/2$  proximities (it's symmetric)
  - Space to keep track of clusters is proportional to the number of clusters ( $m-1$ , excluding singleton clusters)
- Time complexity  $O(m^3)$ 
  - Computing the proximity matrix  $O(m^2)$
  - $m-1$  iterations (Steps 3 and 4)
  - It's possible to reduce the total cost to  $O(m^2 \log m)$  by keeping data in a sorted list (or heap)

## Summary

- Typically used when the underlying application requires a hierarchy
- Generally good clustering performance
- Expensive in terms of computation and storage