6c. Data mining - clustering

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. Usually performed for **understanding** or **summarization**

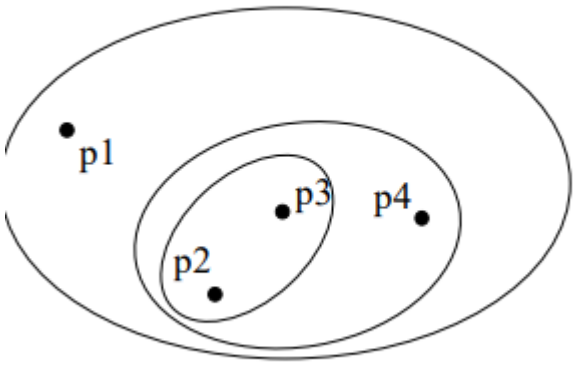
Types of Clusters

Partitional clustering

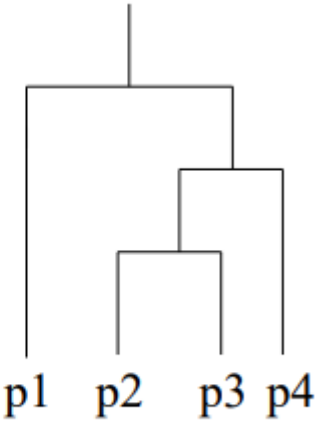
Data is divided into non overlapped subsets(clusters)

Hierarchical clustering

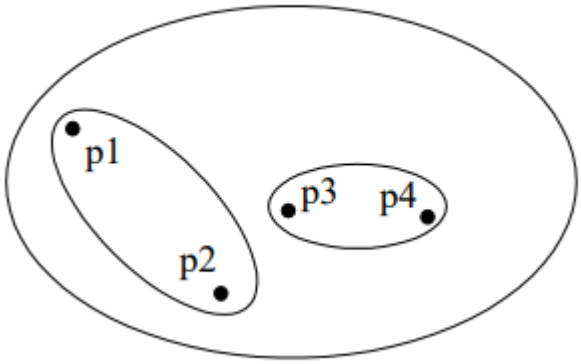
A set of nested clusters organized as a hierarchical tree.
Cluster sets can be visualized as a dendogram, a specific kind of tree.



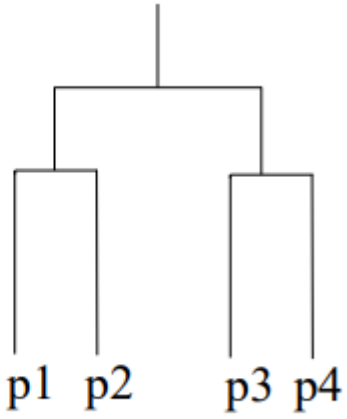
Traditional Hierarchical Clustering



Traditional Dendrogram



Non-traditional Hierarchical Clustering



Non-traditional Dendrogram

Others type

Exclusive versus non-exclusive

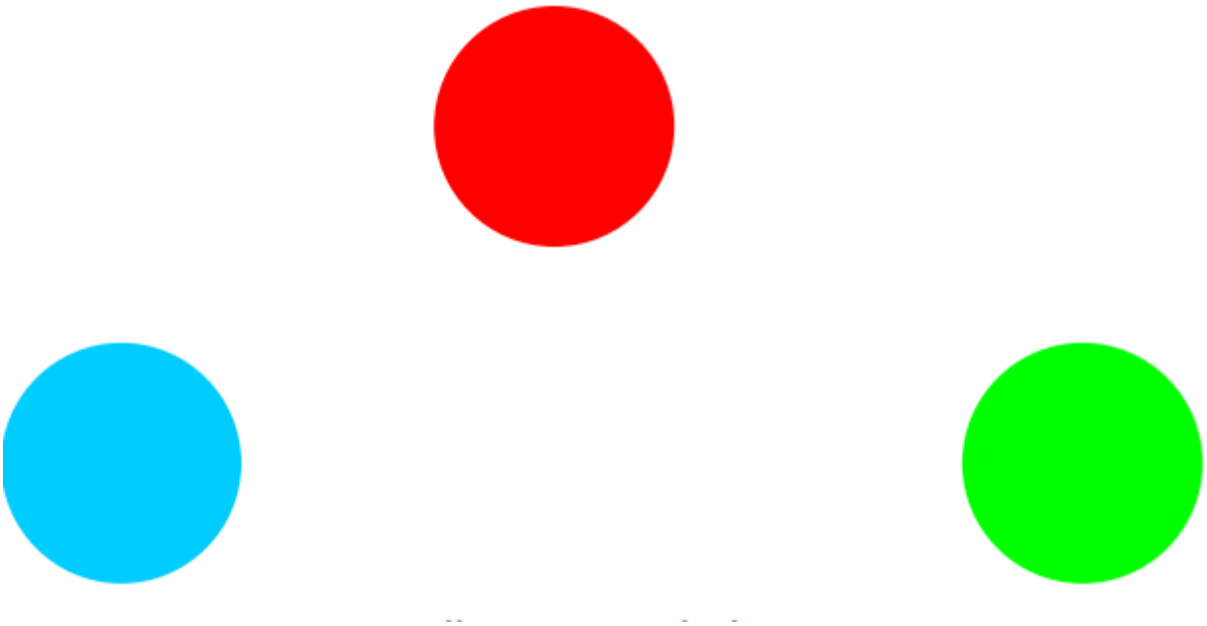
In non-exclusive, points belong to multiple clusters.

Fuzzy versus non fuzzy

In fuzzy clustering, a point belongs to every cluster with a weight from 0 to 1.
Weights must sum to 1.

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.



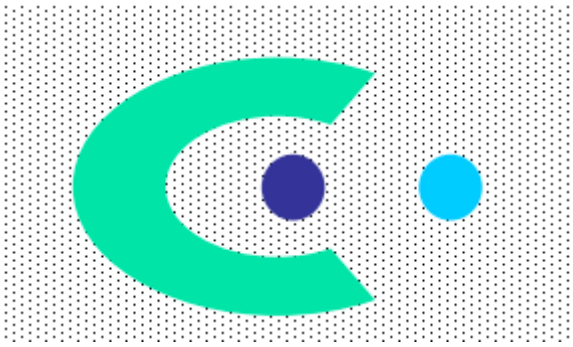
well-separated cluster

Center-based clusters

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 can be either a **centroid**(average of all points) or a **medoid**(most representative point)

Density-based clusters

In this case, a cluster is a dense area of objecsts separated by a low-area density.



Clustering Algorithm

K-means clustering

It uses a partitional clustering approach.
Each cluster is associated with a *centeroid*
Each point is assigned to the cluster with the closest *centroid*
At the beginning, a number of cluster **K** must be specified.

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

Step 1 is often performed randomly.
It converges after few iterations. It can also stops until a certain condition is met.

Evaluating a K-means cluster set

The most common measure is *SSE- Sum of Squared Errors:*

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

where m_i is the *centroid*(or representative point) of the cluster and x is a point in the C_i cluster.
The higher the SSE, the lower the cohesion inside a cluster

Solutions to initial centroid

Multiple runs

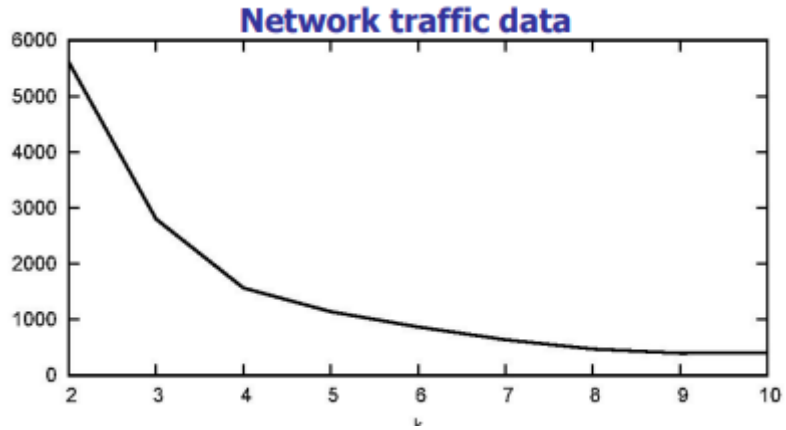
The clustering algorithm is run multiple times. A bruteforce approach basically

Using hierarchical clustering

We can sample centroids from a hierarchical cluster sets.

Knee approach

The idea behind this approach is to find a *knee* in an Elbow graph.
A knee is a location in the graph where the tend changes.
That location is where the best point of **k** resides.



In this case, around 5-6 we can notice a knee. Thus optimal K should be around that range

Post-processing

We can post-process a solution produced by an algorithm in an earlier run.
We can split *loose* clusters, clusters that have high **SSE** or merge clusters that have low **SSE**.
We can also eliminate small clusters that represents outliers.

Bisecting K-means

A variant of the basic K-means algorithm where each time we focus only a specific portion of the data collection.
Each time, we run k-means with K=2 on the collection with the highest **SSE.**
Bisecting K-means can also be used to produced **centroids** for a *basic K-means* algorithm.

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

Pseudocode of Bisecting K-means

Limitations of K-means

They are susceptible to outliers and have problems with clusters with different shapes/size.
To overcome this, we may increase K to obtain more clusters and then merge them together.

Hierarchical Clustering

Unlike the algorithms described above, here no assumptions are made on the number of the expected clusters.
Two different approaches:

- **top-down:** from a single cluster containing all data to multiple clusters. It's a divise approach.
- **bottom-up:** from clusters of single points up to a single cluster. It's an **agglomerative** approach and it's the most used

Agglomerative clustering algorithm

Firstly, a **proximity matrix** is built containing all the distances/similarities between each possible pair of clusters.
Then it merges the two closest clusters and lastly the matrix is updated.
This last step is performed until a single cluster remains.

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

Pseudocode

The key operation is the computation of the **proximity matrix.**
Thus, it's also crucial the definition of the inter-cluster similarity.
Multiple options to choose from:

- **MIN(or single-link):** minimum distance between the two closest points of the two clusters. It's sensitive to outliers and noise. Can handle non-elliptical shapes
- **MAX(or complete-linkage):** maximum distance between the two most distant points of the two clusters
  Less susceptible to noise and outliers. Tends to break large clusters and is biased towards globular clusters.
- **Group average:** average distance between two clusters

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

Less susceptible to noise and outliers.

- **Distance between centroids:** the name is self-explanatory
- Other **objective functions** like **SSE(Ward's method)**: In case of **SSE**, we can use this algorithm to initialize **K-means**

Time and space complexity

Space complexity of $O(N^2)$ since it uses the proximity matrix
Time complexity of $O(N^3)$ in many cases due to N steps and each step updates a matrix(N^2)

Density-based clustering - DBSCAN

Definitions

The density is specified by a number of points withing a specified radius(*Eps*).
A point is said to be **core point** if it has more than a specified number of points(MinPts) withing *Eps*.
A point is said to be a **border point** if it has fewer than MinPts withing *Eps* but it's in the neighbourhood of a **core point**.
Lastly, a point is said to be a **noise point** if it's neither a border or a core point.

Algorithm

The core point is used to initialize a cluster.
This area contains border points and other core points.
The area gets expanded further based on other core points.
Firstly, we eliminate noise points, then

```
current_cluster_label ← 1
for all core points do
  if the core point has no cluster label then
    current_cluster_label ← 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
```

The identification of core points,border points and noise points is performed as the very first step of the DBSCAN algorithm.

Performances

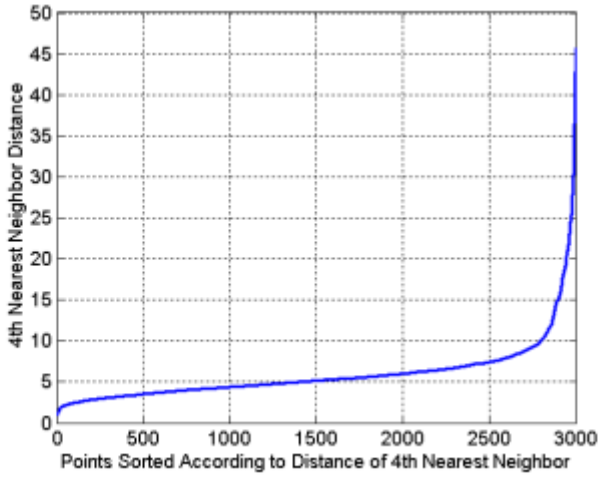
It works well and it's robust to noises as it can handle clusters of differnt shapes/sizes. However, in a high-dimensional data setting with varying densities, DBSCAN doesn't perform too well

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



On the Y-axis we have the radius(EPS) and on the X-axis the number of elements inside said radius.
Let's say we have k=minPts=4(as in the figure), we have that with radius of 10, we have about 2750 points.
As the EPS increases from 10 onwards, the trend goes up exponentially as it includes outliers.

Cluster Validation

Several aspects to take into consideration:

1. Determining **clustering tendency**: that is, distinguishing whether a non-random structure exist
2. Comparing the result of a cluster analysis to *externally* known results(e.g. given class labels)
3. Comparing two different clusters to see which is better
4. Evaluating how well the results of a cluster analysis fit the data without reference to external information.

Measures

We can use different indexes, based on the aspect we are trying to measure.

Internal measures

Cluster cohesion

Used to evaluate if objects in same cluster are similar
Computed through **SSE:**

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

Cluster separation

Used to evaluate if objects placed in different clusters are different.
Computed by measuring the cluster sum of squares:

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

Both indexes are computed through the means of a representative point: the **centroid**.
These measures can also be computed by using a *proximity* graph

Silhouette index

It's computed for each points in the data collection and considers both *cohesion* and *separation*.
The following formula computes said index:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

where:

- **a(i)** is the average *dissimilarity* for a given object *i* with all other objects in the same cluster
- **b(i)** is the minimun of all the average dissimilarities for a given object **i** to any other cluster's objects.
  It gives a value in a range of $[-1, 1]$, the closer to 1, the better.
  The average of $s(i)$ over all data of a cluster measures how tightly grouped all the data in the cluster are.
  The average of $s(i)$ over all data of the dataset measures how appropriately the data has been clustered.

External Measures

Entropy

entropy For each cluster, the class distribution of the data is calculated first, i.e., for cluster j we compute p_{ij}, the 'probability' that a member of cluster j belongs to class i as follows: $p_{ij} = m_{ij}/m_j$, where m_j is the number of values in cluster j and m_{ij} is the number of values of class i in cluster j. Then using this class distribution, the entropy of each cluster j is calculated using the standard formula $e_j = \sum_{i=1}^{L} p_{ij} \log_2 p_{ij}$, where the L is the number of classes. The total entropy for a set of clusters is calculated as the sum of the entropies of each cluster weighted by the size of each cluster, i.e., $e = \sum_{i=1}^{K} \frac{m_i}{m} e_j$, where m_j is the size of cluster j, K is the number of clusters, and m is the total number of data points.

Purity

purity Using the terminology derived for entropy, the purity of cluster j, is given by $purity_j = \max p_{ij}$ and the overall purity of a clustering by $purity = \sum_{i=1}^{K} \frac{m_i}{m} purity_j$.

Relative measures

Rand index

The idea behind this is that two objects belonging to the same cluster should also belong to the same class.

Given

- f_{00} = number of pairs of objects having a different class and a different cluster
- f_{01} = number of pairs of objects having a different class and the same cluster
- f_{10} = number of pairs of objects having the same class and a different cluster
- f_{11} = number of pairs of objects having the same class and the same cluster

Rand Index

$$Rand\ Index = \frac{f_{00} + f_{11}}{f_{00} + f_{01} + f_{10} + f_{11}}$$