Clustering

General definition

Given a **set of points** belonging to some **space**, with a notion of **distance** between points, **clustering** aims at grouping the points into a number of subsets (***clusters***) such that

- points in the same cluster are “close” to one another
- points in different clusters are “distant” from one another

THe distance captures a notion of similarity: close points are similar, distant points are dissimilar

A **clustering problem** is usually defined by requiring that clusters optimize a given **objective function**, and/or satisfy certain **properties**. Numerous clustering problems have been defined, studied and employed in applications over the years.

Metric Space

Typically, the input of a clustering problem consists of a set of points from a **metric space**

Definition

A **metric space** is an ordered pair (M, d) where M is a set and d(.) is a metric on M, i.e., a function

```
d: M * M->R
```

such that for every x, y, z in M the following rules holds

- d(x, y) >= 0
- d(x, y) = 0 iff x = y
- d(x, y) = d(y, x) (symmetry)
- d(x, z)<=d(x, y) + d(y, z) (triangle inequality)

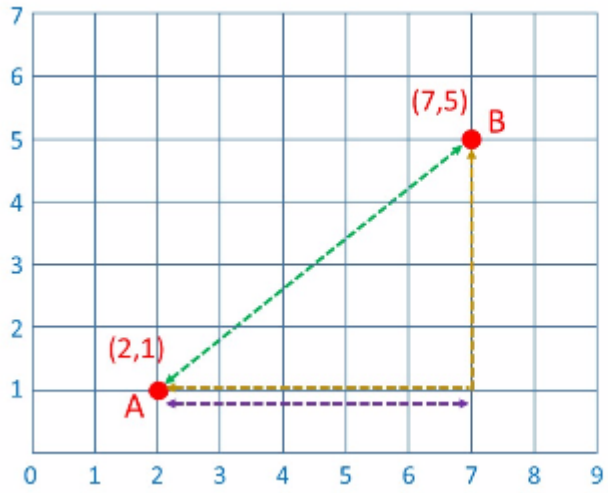
Distance functions

Euclidean distances

- Euclidean distances. Let $X, Y \in \Re^n$, with $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$. For $r \geq 1$ the $L_r -$ distance between X and Y (a.k.a. $L_r -$ norm)

$$d_{Lr}(X, Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^r \right)^{1/r}.$$

  - $r = 2$: standard distance in $\Re^n$ (also denoted by $|| \cdot ||$)
  - $r = 1$: referred to as *Manhattan distance*, it is the sum of the absolute differences of coordinates in each dimension. Used in grid-like environments
  - $r = \infty$: (limit for r that tends to ∞) it is the maximum absolute differences of coordinates, over all dimensions.



$d_{L1}(A,B) = (7-2)+(5-1) = 9$

$d_{L2}(A,B) = (5^2+4^2)^{1/2} = 6.403..$

$d_{L\infty}(A,B) = \max\{5,4\} = 5$

Jaccard distance

Used when points are sets. Let S and T be two sets over the same ground set of elements. The Jaccard distance between S and T is defined as

$$d_{\text{Jaccard}}(S, T) = 1 - \frac{|S \cap T|}{|S \cup T|}.$$

Note that the distance ranges in [0,1], and it is 0 iff S = T and 1 iff S and T are disjoint. The fraction in the Jaccard distance is referred to as the *Jaccard similarity* of the two sets

Cosine (or angular) distance

- Cosine (or angular) distance. It is used when points are *vectors* in $\Re^n$ (or, in general in an inner-product space). Given two vectors $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$ in $\Re^n$, their cosine distance is the angle between them, that is

$$d_{\text{cosine}} = \arccos\left(\frac{X \cdot Y}{||X|| \cdot ||Y||}\right) = \arccos\left(\frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n}(x_i)^2}\sqrt{\sum_{i=1}^{n}(y_i)^2}}\right)$$

  Example: For $X = (1, 2, -1)$ and $Y = (2, 1, 1)$ the cosine distance is

$$\arccos\left(\frac{3}{\sqrt{6}\sqrt{6}}\right) = \arccos(1/2) = \pi/3.$$

Remark. The cosine distance is often used in information retrieval to assess similarity documents. Consider an alphabet of *n* words. A document X over the alphabet can be represented as an *n*-vector, where *xi* is the number of occurrences in X of the *i*-th word of the alphabet

- For non-negative coordinates it takes values in [0,π/2], while for arbitrary coordinates the range is [0,π]. Dividing by π/2 (or π) the range becomes [0,1]
- In order to satisfy the second property of distance functions for metric spaces, scalar multiplesof a vector must be regarded as the same vector

Edit distance

Used for strings. Given two strings X and Y, their edit distance is the **minimum number of deletion and insertion** that must be applied to transform X in Y

It is immediate to show that in all cases the edit distance is

|X|+|Y|-2|LCS(X, Y)|

Hamming distance

Used when points are vectors over some *n*-dimensional space. Most commonly, it is used for binary vectors. The Hamming distance between two vectors is **the number of coordinates in which they differ**.

Curse of dimensionality

Random points in high-dimensional metric space tend to be

- sparse
- almost equally distant from one another
- almost orthogonal (as vectors) to one another

As a consequence, in high dimensions distance functions may lose effectiveness in assessing similarity/dissimilarity. However, this holds in particular when points are random, and it might be less a problem in some real-world datasets.

As an example, consider a set S of N random points in [0,1]^t, where for each point the i-th coordinate is a random number in [0,1] drawn with uniform probability indipendently of the other coordinates and of the

other points.

Let $X = (x_1, x_2, \ldots, x_t)$ and $Y = (y_1, y_2, \ldots, y_t)$ two points of S. Their L_2 distance is

$$d_{L_2}(X, Y) = \sqrt{\sum_{i=1}^{t} (x_i - y_i)^2} \leq \sqrt{t}.$$

Theorem

If $t \geq 4 \log_2 N$, then with probability $\geq 1 - 1/N$ every two points of S are at distance $\geq \sqrt{t}/(\sqrt{2} \cdot 24) = \Omega\left(\sqrt{t}\right)$ from one another.

Types of clustering

Given a set of points in a metric space, hence a distance between points, a clustering problem often specifies and **objective function** to optimize. The objective function also allows comparing different solutions.

Objective functions can be categorized based on wether or not

- a target number *k* of clusters is given in input
- for each cluster a *center* must be identified

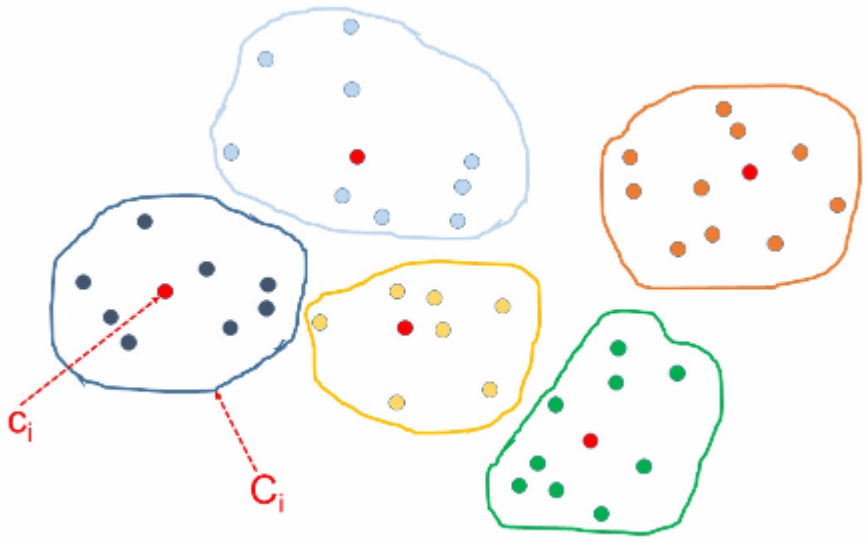
  When centers are required, the value of the objective function depends on the selected centers. Cluster centers must belong to the underlying metric space but, sometimes, they are costrained to belong to the input set.
- *Disjoint clusters* are sought

Center-based clustering

Let P be a set of N points in a metric space (M, d), and *k* be the target number of clusters, 1 <= *k* <= N. We define a ***k*-clustering of P** as a tuple C = (C1,..., C *k*;c1,..., c *k*) where

- $(C_1, C_2, \ldots, C_k)$ defines a partition of P, i.e., $P = C_1 \cup C_2 \cup \cdots \cup C_k$
- $c_1, c_2, \ldots, c_k$ are suitably selected centers for the clusters, where $c_i \in C_i$ for every $1 \leq i \leq k$.

Observe that the above definition requires the center to belong to the clusters, hence to the pointset. Whenever appropriate, we will discuss the case when the center can be chose more freely from the metric space.



Definition

Consider e metric space (M, d) and an integer $k > 0$. A k-clustering problem for (M, d) is an optimization problem whose instances are the finite pointsets $P \subseteq M$. For each instance P, the problem requires to compute a k-clustering $\mathcal{C}$ of P (feasible solution) which minimizes a suitable objective function $\Phi(\mathcal{C})$.

The following are three popular objective functions.

- $\Phi_{\text{kcenter}}(\mathcal{C}) = \max_{i=1}^{k} \max_{a \in C_i} d(a, c_i)$ (k-center clustering)
- $\Phi_{\text{kmeans}}(\mathcal{C}) = \sum_{i=1}^{k} \sum_{a \in C_i} (d(a, c_i))^2$ (k-means clustering).
- $\Phi_{\text{kmedian}}(\mathcal{C}) = \sum_{i=1}^{k} \sum_{a \in C_i} d(a, c_i)$ (k-median clustering).

In other words, under the above functions, the problem requires to find the k-clustering that minimizes, respectively, the maximum distance (k-center), or the average square distance (k-means), or the average distance (k-median) of a point to its cluster center.

- All aforementioned problems (k-center, k-means, k-median) are **NP-hard**. Hence, in general it is impractical to search for optimal solutions.
- There are severa efficient approximation algorithms that in practice return good-quality solutions. However, dealing efficiently with large inputs is still a challenge.
- K-center and k-median belong to the family of **facility-location-problems**. In these problems, a set F of candidate facilities and a set C of clients are given and the objective is to find a subset of at most k candidate facilities to open, and an assignment of clients to them, so to minimize the maximum of average distance between a client and its assigned facilities. In our formulation, each input point represents both a facility and a client. Numerous variants of these problems have been studied in the literature.
- K-means objective is also referred to as **Sum of Squared Errors (SSE)**

Partitioning primitive

Let P be a pointset and S subset of P a set of *k* selected centers. For all previously defined clustering problems, the best *k*-clustering around these centers is the one where each ci belongs to a distinct cluster and each other point is assigned to the cluster of the closest ci (ties broken arbitrarily)

```
Partition(P, S)

Let S = {c_1, c_2, ..., c_k} ⊆ P
for i ← 1 to k do C_i ← {c_i}
for each p ∈ P − S do
   ℓ ← argmin_{i=1,k} {d(p, c_i)} // ties broken arbitrarily
   C_ℓ ← C_ℓ ∪ {p}
C ← (C_1, C_2, ..., C_k; c_1, c_2, ..., c_k)
return C
```

k-center clustering

Farthest-First Traversal: algorithm

Input Set P of N points from a metric space (M, d), integer $k > 1$

Output k-clustering $\mathcal{C} = (C_1, C_2, \ldots, C_k; c_1, c_2, \ldots, c_k)$ of P.

```
S ← ∅ {c_1} //c_1\in P arbitrary point
for i ← 1 to k do
   Find the point c_i ∈ P − S that maximizes d(c_i, S)
   S ← S ∪ {c_i}
return Partition(P, S)
```

- $d(c_i, S)$ denotes the minimum distance of c_i from a point of S. That is, $d(c_i, S) = \min\{c \in S : d(c_i, c)\}$
- The assignment of points to clusters can be accomplished while determining the centers in the first for-loop.

Theorem

Let $\Phi^{\text{opt}}_{\text{kcenter}}(k)$ be the minimum value of $\Phi_{\text{kcenter}}(\mathcal{C})$ over all possible k-clusterings $\mathcal{C}$ of P, and let $\mathcal{C}_{\text{alg}}$ be the k-clustering of P returned by the Farthest-First Traversal algorithm. Then:

$$\Phi_{\text{kcenter}}(\mathcal{C}_{\text{alg}}) \leq 2 \cdot \Phi^{\text{opt}}_{\text{kcenter}}(k).$$

That is, Farthest-First Traversal is a 2-approximation algorithm.

- K-center clustering provides a strong guarantee on how close each point is to the center of its cluster.
- However, gor noisy pointsets (e.g. pointsets with outliers) the clustering which optimizes the *k*-center objective may obfuscate some "natural" clustering inherent in the data
- For any fixed ε > 0 it is NP-hard to compute a *k*-clustering C_alg with

  $$\Phi_{\text{kcenter}}(C_{\text{alg}}) \leq (2-\epsilon)\Phi^{\text{opt}}_{\text{kcenter}},$$

  hence, the Farthest-First Traversal is likely to provide almost the best approximation guarantee obtainable in polynomial time

k-center clustering for big data

How can we compute a "good" k-center clustering of a pointset P that is too large for a single machine?

We employ a **coreset-based approach**:

1. Extract from P a small T (coreset) of representatives
2. Compute within T the set S of *k* cluster centers
3. Compute the final clustering around the center of S

Observations:

- The coreset-based approach is effective when step 1 and 3 can be performed efficiently, and the coreset T contains a good set of centers
- Coreset are used to confine computations which are too expensive to run on the whole input on small instances
- A coreset needs not to be a simple subset of the input

MapReduce-Farthest-First Traversal

Let P be a set of N points (N large) from a metric space (M, d), and let *k* > 1 be an integer. The following MapReduce algorithm computes a good *k*-center clustering.

1. Partition P arbitrarily in *l* subsets of equal size and execute the Farthest-First Traversal algorithm on each subset separately to identify a set T_i of *k* centers.
2. Gather the coreset T = union of all T_i and run using a single reducer, the Farthest-First Traversal algorithm on T to identify S = {c1,..., c *k*} of *k* centers
3. Execute Partition(P, S)

Observation:

Note that in rounds 1 and 2 the Farthest-First traversal algorithm is used to determine only centers and not complete clusterings.

Analysis of MR-Farthest-First Traversal

Assume *k*=o(N). By settings *l* = (N/*k*)^1/2, it is easy to see that the 3-round MR-Farthest-First traversal algorithm uses

- Local space $M_L = O\left(\sqrt{N \cdot k}\right) = o(N)$
- Aggregate space $M_A = O(N)$

Theorem

Let $\Phi^{\text{opt}}_{\text{kcenter}}(k)$ be the minimum value of $\Phi_{\text{kcenter}}(\mathcal{C})$ over all possible k-clusterings $\mathcal{C}$ of P, and let $\mathcal{C}_{\text{alg}}$ be the k-clustering of P returned by the MR-Farthest-First Traversal algorithm. Then:

$$\Phi_{\text{kcenter}}(\mathcal{C}_{\text{alg}}) \leq 4 \cdot \Phi^{\text{opt}}_{\text{kcenter}}(k).$$

That is, MR-Farthest-First Traversal is a 4-approximation algorithm.

Observations on MR-Farthest-First Traversal

- The sequential Farthest-First Traversal algorithm is used both to extract the coreset and to compute the final set of centers. It provides a good coreset since it ensures that any point not in the coreset be well represented by some coreset point.
- The main feature of MR-Farthest-First Traversal is that while only small subsets of the input are processed at once, and many of them in parallel, the final approximation is not too far from the best achievable one.
- By selecting $k' > k$ centers from each subset Pi in round 1, the quality of the final clustering improves. In fact, it can be shown that when P satisfy certain properties and k' is sufficiently large, MR-Farthest-First Traversal returns a (2+ε)-approximation for any constant ε > 0, still using sublinear local space and linear aggregate space.

k-means clustering

General observations on k-means clustering

- In essence, **k-means clustering aims at minimizing cluster variance**. It is typically used in Euclidean spaces and works well for discovering ball-shaped clusters.
- Because of the quadratic dependence on distances, **k-means clustering is rather sensitive to outliers**. However, the fact that the objective function sums all distances, makes it more robust than the k-center objective to noise and outliers.
- Some **established algorithms for k-means clustering perform well in practice** although only recently a rigorous assessment of their performance-accurancy tradeoff has been carried out.

Propererties of Euclidean spaces

Let $X = (X_1, X_2, \ldots, X_D)$ and $Y = (Y_1, Y_2, \ldots, Y_D)$ be two points in $\Re^D$. Recall that their Euclidean distance is

$$d(X, Y) = \sqrt{\sum_{i=1}(X_i - Y_i)^2} \triangleq ||X - Y||.$$

Definition

The centroid of a set P of N points in $\Re^D$ is

$$c(P) = \frac{1}{N} \sum_{X \in P} X,$$

where the sum is component-wise.

Observe that $c(P)$ does not necessarily belong to P

Lemma

The centroid $c(P)$ of a set $P \subset \Re^D$ is the point of $\Re^D$ which minimizes the sum of the square distances to all points of P.

Observation: The lemma implies that when seeking a *k*-clustering for points in R^D which minimizes the **kmeans** objective, the best center to select for each cluster is its centroid (assuming that centers need not necessarily belong to the input pointset)

Lloyd's algorithm

- Also Known as **k-means algorithm**
- It focuses on Euclidean spaces and does not require cluster centers to belong to the input pointset
- It relates to generalization of the Expectation-Maximization algorithm

Input Set P of N points from $\Re^D$, integer $k > 1$

Output k-clustering $\mathcal{C} = (C_1, C_2, \ldots, C_k; c_1, c_2, \ldots, c_k)$ of P which is a good approximation to the k-means problem for P. The centers need not belong to P.

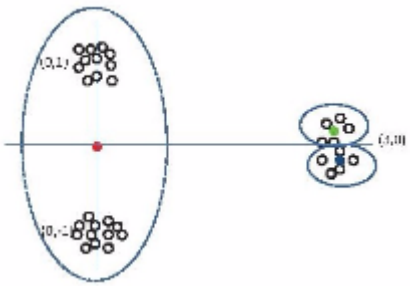
$S \leftarrow$ arbitrary set of k points in $\Re^D$
$\Phi \leftarrow \infty$; stopping-condition $\leftarrow$ false
while (!stopping-condition) **do**
  $(C_1, C_2, \ldots, C_k; S) \leftarrow$ Partition(P, S)
  for $i \leftarrow 1$ **to** k **do** $c_i' \leftarrow$ centroid of C_i
  $S' \leftarrow \{c_1', c_2', \ldots, c_k'\}$
  Let $\mathcal{C} = (C_1, C_2, \ldots, C_k; S')$
  if $\Phi_{\text{kmeans}}(\mathcal{C}) < \Phi$ **then**
    $\Phi \leftarrow \Phi_{\text{kmeans}}(\mathcal{C})$
    $S \leftarrow S'$
  else stopping-condition $\leftarrow$ true
return $\mathcal{C}$

Theorem

The Lloyd's algorithm always terminates

Observations on Lloyd's algorithms

- Lloyd's algorithm may be trapped into a local optimum whose value of the objective function can be much larger than the optimal value. Hence, no guarantee can be proved on its accuracy. Consider the following example and suppose that $k = 3$.



  If initially one center falls among the points on the left side, and two centers fall among the points on the right side, it is impossible that one center moves from right to left, hence the two obvious clusters on the left will be considered as just one cluster.

- While the algorithm surely terminates, the number of iterations can be exponential in the input size
- Besides the trivial k^N **upper bound** on the number of iterations, more sophisticated studies proved an **O(N^(kD)) upper bound** which is improved upone the trivial one, in scenarios where k and D are small
- Some recent studies proved also a 2^(Ω(sqrt(N))) **lower bound** on the number of iterations in the worst case
- Despite the not so promising theoretical results, empirical studies show that, in practice, **the algorithm requires much less than *N* iterations**, and, **if properly seeded it features guaranteed accurancy**
- In order to improve performance, without sacrificing the quality of the solution too much, one could stop the algorithm earlier, e.g., when the value of the objective function decreases by a small additive factor.

Effective initialization

The quality of the solution and the speed of convergence of Lloyd's algorithm depend considerably from the choice of the initial set of centers

k-means++

Let *P* be a set of *N* points in R^D, and let k>1 be an integer

__Algorithm k-means++ computes a initial set *S* of *k* centers for *P* using the following procedure (note that *S* subset of *P*):

$c_1 \leftarrow$ random point chosen from P with uniform probability
$S \leftarrow \{c_1\}$
for $i \leftarrow 2$ **to** k **do**
  for each $p \in P - S$ **do** $d_p \leftarrow \min_{c \in S} d(c, p)$
  $c_i \leftarrow$ random point chosen from $P - S$, where a point P is chosen with probability $(d_p)^2 / (\sum_{q \in P-S} (d_q)^2)$
  $S \leftarrow S \cup \{c_i\}$
return S

Observation:

Although k-means++ already provides a good set of centers for the k-means problem it can be used to provide the initialization for Lloyd's algorithm, whose iterations can only improve the initial solution

k-means clustering in MapReduce

- In practice, one could initialize the centers using k-means++ and then run a few rounds of Lloyd's algorithm, until the quality of the clustering shows little improvement
- In order to reduce the number of iterations of k-means++, hence the number of MapReduce rounds, a parallel variant of k-means++ has been proposed: in this variant a coreset of > k centers is selected in a few iterations and then k centers are extracted from the coreset using a weighted version of k-means++ or of any other algorithm for k-means clustering.

k-median clustering

Partitioning Around Medoids (PAM) algorithm

- Based on the local search optimization strategy
- Unlike the k-means algorithm, cluster centers belong to the input pointset and are referred as **medoids**

Input Set P of N points from a metric space (M, d), integer $k > 1$

Output k-clustering $\mathcal{C} = (C_1, C_2, \ldots, C_k; c_1, c_2, \ldots, c_k)$ of P
which is a good approximation to the k-median problem for P.
The centers must belong to P.

$S \leftarrow \{c_1, c_2, \ldots, c_k\}$ (arbitrary set of k points of P)
$\mathcal{C} \leftarrow$ Partition(P, S)
stopping-condition $\leftarrow$ false
while (!stopping-condition) **do**
  stopping-condition $\leftarrow$ true
  for each $p \in P - S$ **do**
    for each $c \in S$ **do** $S' \leftarrow (S - \{c\}) \cup \{p\}$
      $\mathcal{C}' \leftarrow$ Partition(P, S')
      if $\Phi_{\text{kmedian}}(\mathcal{C}') < \Phi_{\text{kmedian}}(\mathcal{C})$ **then**
        stopping-condition $\leftarrow$ false
        $\mathcal{C} \leftarrow \mathcal{C}'$; $S \leftarrow S'$
        exit both for-each loops
return $\mathcal{C}$

Pam algorithm: analysis

Theorem (Arya et al.'04)

Let $\Phi^{\text{opt}}_{\text{kmedian}}(k)$ be the minimum value of $\Phi_{\text{kmedian}}(\mathcal{C})$ over all possible k-clusterings $\mathcal{C}$ of P, and let $\mathcal{C}_{\text{alg}}$ be the k-clustering of P returned by the PAM algorithm. Then:

$$\Phi_{\text{kmedian}}(\mathcal{C}_{\text{alg}}) \leq 5 \cdot \Phi^{\text{opt}}_{\text{kmedian}}(k).$$

Remark: by imposing that in each iteration the new clustering $\mathcal{C}'$ decreases the objective function by at least $1 - \epsilon/(Nk)$, for some constant $\epsilon \in (0, 1)$, stopping the algorithm if this is not possibile, it can be proved that the algorithm executes a number of iterations polynomial in N and $\Phi_{\text{kmedian}}(\mathcal{C}_{\text{alg}})$ is a factor $f(\epsilon) \in O(1)$ away from the optimal value.

Observations on k-median clustering

- k-median is less sensitive to outliers than k-center and k-means since all distances are taken into account
- the PAM algorithm works for any metric space and features provable performance and approximation guarantees. However, it is very slow in practive since in each iteration up to (*N*-*k*)***k* swaps may need to be checked, and for each swap a new clustering must be computed.
- a faster alternative is an **adaption of the k-means algorithm**: in each iteration of the main while loop, and for each current cluster C_i the new center (**medoid**) will be the point of C_i which minimizes the sum of the distances to all other points, instead of the centroid used by k-means. This algorithm appears faster and still very accurate in practice

How to pick the right value for k

- Sometimes, the application provides a target value for *k*
- If such a target value of *k* is known:
  - Find a *k*-clusterings with geometrically larger values of *k* and stop at *k* = *x* if the value of the objective function for *k* = *x* does not improve "too much" with respect to *k* = *x/2*
  - Refine search of *k* in [x/2,x]

Weighted k-median clustering

One can define the following, more general, **weighted variant of the k-median clustering problem**.

Let *P* be a set of *N* points from a metric space (M,d), and let *k* > 1 be an integer. Suppose that for each p in *P* an integer weight w(p) > 0 is also given. We want to compute the *k*-clustering C = (C_1,C_2,...,C_k;c_1,c_2,...,c_k) of *P* which minimize the following objective function

$$\Phi^{w}_{\text{kmedian}}(C) = \sum_{i=1}^{k} \sum_{p \in C_i} w(p) \cdot d(p, c_i),$$

that is, the average weighted distance of a point from its cluster center.

By considering each point of *P* as representing w(p) identical copies of p, one can easily devise a **weighted version of the PAM algorithm** to solve Weighted k-median clustering with the same approximation guarantees.

k-median clustering in MapReduce: MR-PAM

The PAM algorithm is impractical for large datasets.

The following coreset-based MapReduce algorithm (MR-PAM) is more suitable for the big data scenario.

- **Round 1**: Partition *P* is arbitrarily in *l* subsets of equal size P_1,P_2,...,P_l and execute the PAM algorithm independently on each P_i. For 1<= i <= l let T_i be the set of *k* centers of the clustering computed on P_i and, for each p in T_i, let w(p) be the number of points in the cluster centered at p
- **Round 2**: Gather the coreset T = Union of all T_i of *l*k* points, together with their weights. Using a simple reducer, run the weighted version of PAM on T to identify a set S = {c_1,c_2,...,c_k} of *k* centers
- **Round 3**: Run Partition(P,S) to return final clustering

MR-PAM: analysis

Assume *k = o(N)*. By setting *l = sqrt(N/k)*, it is easy to see that the 3-round MR-PAM algorithm requires

- **Local Space**: *M_L = O(sqrt(N*k)) = o(N)*
- **Aggregate Space**: *M_A = O(N)*

MR-PAM algorithm is claimed to be a **15-approximation algorithm**. In fact, by using an α-approximation algorithm instead of (weighted) PAM in Rounds 1 and 2, they claim that the final approximation factor is 3α.

In order to ensure polynomial time for the reduce functions, in Round 1 and 2 one should run the version of PAM that stops whenever the improvement in the objective function is below a fixed threshold.

Cluster Evaluation

Goals

- **Clustering tendency**: assessment wether the data contain meaningful clusters, namely clusters that are unlikely to occur in a random data.
- **Unsupervided evaluation**: assessment of the quality of a clustering *without reference to external information*
- **Supervised evaluation**: assessment of the quality of a clustering *with reference to external information*

Clustering tendency: Hopkins statistics

Let *P* be a dataset of *N* points in some metric space (M,d).

The **Hopkins statistic** measures to what extent the points of *P* can be regarded as taken randomly from *M*. For some fixed *t << N* (typically *t < 0.1*N*) let:

$$
\begin{aligned}
X &= \{x_1, x_2, \ldots, x_t\} \text{ random sample from } P \\
Y &= \{y_1, y_2, \ldots, y_t\} \text{ random set of points from } M \\
u_i &= \min_{z \in P,\, z \neq y_i} d(y_i, z) \quad \text{for } 1 \leq i \leq t \\
w_i &= \min_{z \in P,\, z \neq x_i} d(x_i, z) \quad \text{for } 1 \leq i \leq t
\end{aligned}
$$

$$
H(P) = \frac{\sum_{i=1}^{t} u_i}{\sum_{i=1}^{t} u_i + \sum_{i=1}^{t} w_i}
$$

- **H(P) ~ 1: P** is likely to have a clustering structure
- **H(P) ~ 0.5: P** is likely to be a random set
- **H(P) << 0.5:** the points of **P** are likely to be well spaced.

Unsupervised evaluation

- In the case of k-center, k-means, and k-median, the value of the objective function can be employed to assess the quality of a clustering or the relative quality of two clusterings.
- For a clustering of a more general type, one could compare the **cohesion** within clusters against the **separation** between clusters:
    - **Cohesion**: average distance between two points in the same clusters, where the average is taken over all such pairs of points
    - **Separation**: average distance between two points in different clusters, where the average is taken over all such pairs of points

  The larger the gap between cohesion and separation, the better the quality of the clustering

Silhouette coefficient

Let C be a clustering of a pointset P.

For a point p in P belonging to some cluster C in C

- a_p = average distance between p and the other points in C.
- b_p = minimum, over all clusters $C' \neq C$ of the average distance between p and the other points in C'.
- The silhouette coefficient for p is

$$
s_p = \frac{b_p - a_p}{\max\{a_p, b_p\}},
$$

  which is a value between -1 (i.e., $b_p = 0$) and 1 (i.e., $a_p = 0$).

The quality of C can be assessed through the

$$
\text{average silhouette coefficient:} \quad s_C = \frac{1}{|P|} \sum_{p \in P} s_p.
$$

_C is a "good" clustering if **s_C~1**

Unsupervised evaluation for big data

- Computing cohesion and separation or the average silhouette coefficient exactly requires computing Θ(|*P*|^2) inter-point distances, which becomes prohibitive for very large inputs
- Cohesion and separation can be approximated by sampling pairs of intra-cluster and inter-cluster points, respectively
- The **average silhouette coefficient** can be approximated in several ways:
    - First a suitable center is computed for each cluster C. Then, for each p in C, the value b_p is approximated with the average distance between p and the points of the cluster C'/C whose center is closest to p
    - For each p in a cluster C, the value b_p is approximated with the minimum, over all clusters C'!=C, of the average distance between p and a random sample of the points in C'. Also, a_p can be approximated with the average distance between p and a random sample of the points in C

Supervised evaluation: entropy

Consider a clustering *C* of a pointset *P*. Suppose that each point *p in P* is associated with a class label out of a domain of *L* class labels.

For each cluster C in *C* and class *i* let

$$\begin{aligned} m_C &= \#\text{points in cluster } C \\ m_i &= \#\text{points of class } i \\ m_{C,i} &= \#\text{points of class } i \text{ in cluster } C \end{aligned}$$

Entropy of a cluster *C*:

$$-\sum_{i=1}^{L} \frac{m_{C,i}}{m_C} \log_2 \frac{m_{C,i}}{m_C} \quad \text{(N.B. } 0 \log_2 0 = 0\text{)}$$

It measures the impurity of C, ranging from 0 to log_2 *L*

Entropy of a class *i*:

$$-\sum_{C \in \mathcal{C}} \frac{m_{C,i}}{m_i} \log_2 \frac{m_{C,i}}{m_i}$$

It measures how evenly the points of class *i* are spread among clusters, ranging from 0 to log_2 *K*, where *K* is the number of clusters.

It is defined also when points belong to multiple classes

When the number *k* of clusters and the number *L* of labels are small, entropies van be determined efficiently even for very large pointsets *P*.

If *k*L = o(N)*, with *N* = |*P*| use the following MapReduce algorithm:

1. **Round 1**: Partition *P* arbitrarily in sqrt(*N*) subsets of equal size *P_1, P_2,...,P_sqrt(N)*. Within each P_j compute, for each class *i* and cluster C, the number m_{C,i}(j) of points of class *i* in C intersected P_j

2. **Round 2**: For each class i and cluster C, gather all m_{C,i}(j)'s and compute their sum
3. **Round 3**: Gather all m_{C,i}'s and compute the desired entropies.

The algorithm requires *O(sqrt(N)+k*L)* local space and linear aggregate space.

Hierarchical clustering

- Produces a hierarchy of nested clusterings of decreasing cardinalities
- No need to fix the number of clusters a priori or to choose cluster centers.
- Two alternative high-level strategies:
    - Agglomerate: Starts with each input point in a separate cluster and progressively merges suitably selected pairs of clusters. It is the most common strategy.
    - Divisive: Starts with one cluster containing all points and progressively splits a cluster into two.

General Agglomerative strategy

Let *P* be a set of *N* points in a metric space (M,d).

```
Make each point as a distinct singleton cluster
while (!stopping-condition) do
  merge the two closest clusters
return the current set of clusters
```

Observation:

- In order to instantiate the algorithm one needs to decide when to stop and which pair of clusters to merge at each iteration
- The number of clusters decreases by 1 at each iteration
- Instead of returning the clustering resulting after the last iteration, one may return the dendogram, i.e., the tree/forest defined by all clusters created at the various iterations

Merging criterion

How do we measure the distance between two clusters C_1,C_2 so to be able to identify the "two closest clusters" at each iteration?

- Single linkage: $\text{dist}(C_1, C_2) = \min_{x \in C_1, y \in C_2} d(x, y)$
- Complete linkage: $\text{dist}(C_1, C_2) = \max_{x \in C_1, y \in C_2} d(x, y)$
- Average linkage: (average distance between C_1 and C_2)

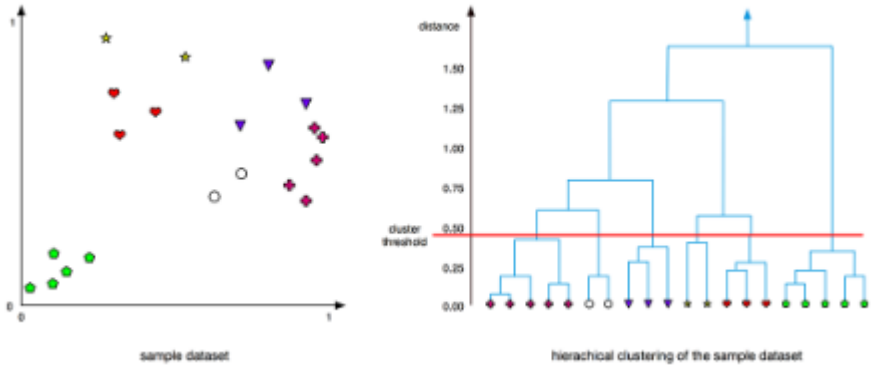
$$\text{dist}(C_1, C_2) = \frac{1}{|\{(x, y) \ : \ x \in C_1, y \in C_2\}|} \sum_{x \in C_1, y \in C_2} d(x, y)$$

Stopping condition

Depending on the application, the merging process can be stopped using one of the following conditions

- A desired number *K* of clusters obtained

- The distance between the next pair of clusters to be merged exceeds a fixed cluster threshold t
- The clustering resulting after the next merge would violate some specific condition on the density or cohesion of the clusters



Time complexity

Consider the execution of the hierarchical clustering strategy for a set P of N points.

At each iteration, maintain with each point the ID of the cluster it belongs to ($\Theta(N)$)

Straightforward implementation

- In each iteration, search for the pair of closest clusters by computing the distances between all pairs of points
- $\Theta(N^2)$ time per iteration, hence $\Theta(N^3)$ overall time, if full hierarchy is sought
- $\Theta(N)$ space, if distances are computed on-the-fly, while $\Theta(N^2)$ space if all distances are precomputed

Improved implementation

- Precompute all $\Theta(N^2)$ distances and store each pair of points (x,y) into a min-heap H, using d(x,y) as a key
- Extract, one after the other, the pairs of points from H in increasing order of distance. After extracting pair(x,y), if the two points belong to different clusters then merge the two clusters
- The initialization requires $\Theta(N^2)$ time and space. Each extraction from H takes $O(\log N)$ time, while each merge takes $O(N)$ time.

  Thus, the implementation requires

  $O(N^2 \log N)$ overall running time and $\Theta(N^2)$ space.

Remark

- More efficient implementations for both single and complete linkage exists

Observation on Hierarchical Clustering

Pros

- Useful when a hierarchical taxonomy is shought and/or a precise number of clusters cannot be established apriori
- Can capture clusters of non-elliptical shapes

Cons

- Does not optimize any specific objective function
- Sensitive to noise
- Computationally expesive. The development of efficient hierarchical clustering algorithms for very large pointsets is still an open research problem