Ανάπτυξη Λογισμικού για Δυσεπίλυτα Αλγοριθμικά Προβλήματα

Ενότητα 2: Clustering

Γιάννης Εμίρης

Τμήμα Πληροφορικής & Τηλεπικοινωνιών
Εθνικό Καποδιστριακό Πανεπιστήμιο Αθηνών

November 10, 2019

Outline

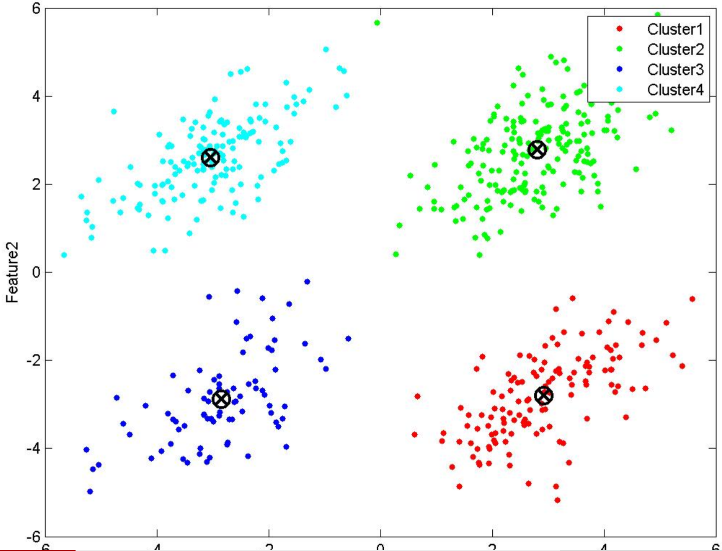
Clustering

Definition (k clusters)

Given n objects (and $k > 1$) partition the objects into k clusters so as to optimize some objective function.

- Objects in the same cluster are more "similar" (or closer) to each other than to those in other clusters.
- Possible criteria: minimizing the total distance among all cluster points, minimizing the distance of cluster points to some center, etc.
- Variations: k is unknown and computed, e.g., by the Silhouette method. Capacitated/balanced: k given, clusters of equal cardinality.
- Applications: Classification, Social Network Analysis, Recommender Systems, Market Research, Bioinformatics etc

Good Clustering, with centers



Approaches

- hierarchical (agglomerative): each point initializes a cluster, merge until stopping criterion, e.g., predetermined number of clusters, or if merging creates cluster with points too far apart.
- centroid-based (point-assignment): given some initial clusters, assign points to "best" cluster; might allow combining / splitting clusters, or unassign points. Example: k-means (our focus).

(Ullman et al:Mining Massive datasets)

Outline

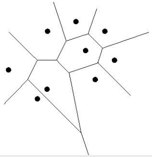
Vector spaces

Problem definition

- Clustering that minimizes objective function.
- *k* is given.
- Centroids do not have to be part of the dataset

k-means

- k-means is the most common problem: Main algorithms:
  -- Lloyd's algorithm is standard.
  -- Elkan's uses triangular inequality to accelerate updates.
- Also used to construct initial clusters for more sophisticated method.
- In Euclidean space, assignment is point location to the *k* Voronoi cells.



k-means: Objective function

Typical ambient space is $\mathbb{R}^d$ but can generalize to metric space $\mathcal{Z}$.

Minimization function

In any metric space over points/objects $\mathcal{Z}$ with distance/metric function d, let the dataset be $X = \{x_1, \ldots, x_n\} \subseteq \mathcal{X} \subseteq \mathcal{Z}$, $k > 1$. Given centroids $C \subset \mathcal{Z}$, let

$$\mathrm{d}(x_i, C) = \min_{c \in C} \mathrm{d}(x_i, c).$$

Consider vector $v(C) = (\mathrm{d}(x_1, C), \ldots, \mathrm{d}(x_n, C))$. The k-means objective is:

$$\min_{C \subseteq \mathcal{Z}, |C| = k} \|v(C)\|_2^2 = \sum_{i=1}^{n} \mathrm{d}(x_i, C)^2.$$

The k-means objective is NP-hard, but for the ℓ_2 metric, Lloyd's algorithm converges quickly to a *local* minimum.

Variations

Various minimizations

Recall $X = \{x_i\}$, $v(C) = (\mathrm{d}(x_1, C), \ldots, \mathrm{d}(x_n, C))$, where $C \subset \mathcal{Z}$ are the centroids, and the k-means objective is:

$$\min_{C \subseteq \mathcal{Z}, |C|=k} \|v(C)\|_2^2 = \sum_{i=1}^{n} \mathrm{d}(x_i, C)^2.$$

Similar objectives:
-- k-median: $\min_{C \subseteq \mathcal{Z}, |C|=k} \|v(C)\|_1$,
-- k-medoid: $\min_{C \subseteq X, |C|=k} \|v(C)\|_1$.
-- k-center: $\min_{C \subseteq X, |C|=k} \|v(C)\|_\infty$,

Lloyd's

Algorithm

Initialize k centers randomly (or using some strategy).

1. Assignment: Assign each object to its nearest center.
2. Update: Calculate mean $\frac{1}{T}\sum_{i=1}^{T}\vec{v}_i$ of each cluster, make it new center.

Repeat the two steps until there is no change in the assignments.

Properties

- Each distance calculation $= O(d)$ because vectors in $\mathbb{R}^d$.
- Assignment $= O(nkd)$, Update $= O(nd)$,
- $\#$iterations unknown, in practice $\ll n$.
- Converges to local minimum in Euclidean space (depends on initialization)

Outline

k-medoids

Goal: Handle any distance metric; k-means only if consistent with mean.

k-medoids (PAM is simplest algorithm) use centroids that **belong** to the dataset:

Definition (Medoid)

The medoid of a set is the object of the set that minimizes total dissimilarity (distance) to all other objects in the set.

Objective function (cf. above): Minimize sum of distances to point's centroid.

vs k-means

-- k-means tends to select convex spherical clusters; k-medoids less so.
-- k-means is more sensitive to noisy data and outliers.
-- k-means is faster and easier to implement.

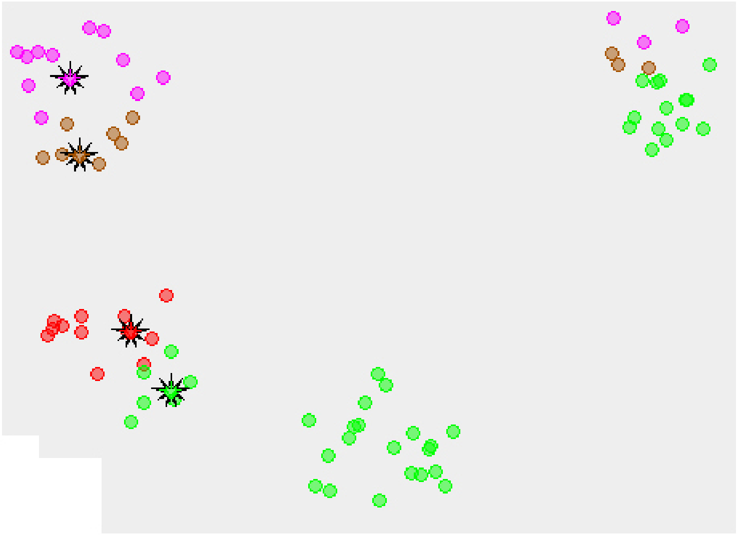
(Kaufman-Rousseeuw'87)

Partitioning Around Medoids (PAM)

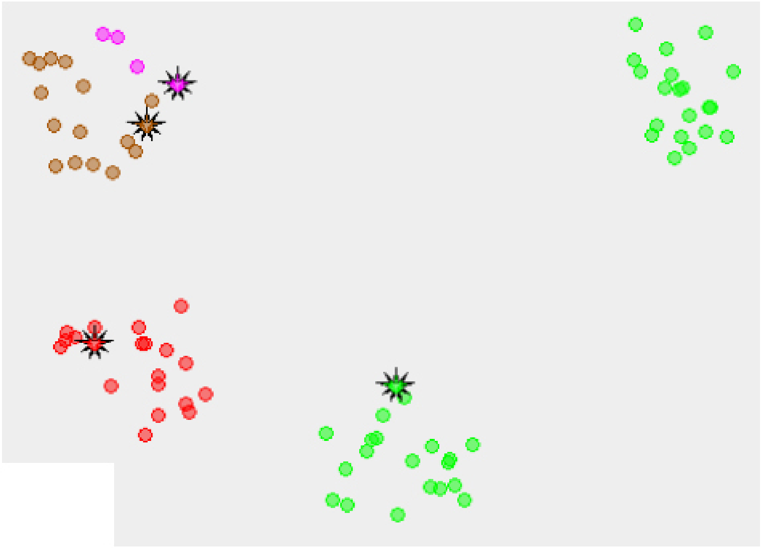
```
Initialize $k$ centroids randomly.
  1 Assignment: Assign each object to nearest centroid; compute objective
  2 Update:
      for each centroid $m$ do
          for each non-centroid $t$ do
              Swap $m$ and $t$, compute new objective function value.
          end for
      end for
      Keep configuration (centroids) with min objective value.
Repeat steps 1 and 2 until there is no change of configuration (centroids).
```

Distance calculation $= O(d')$. Objective update $= O((n-k)d')$, testing only 2nd best centroid (assumed known). Update $= O((n-k)^2kd') \sim n^2$ (costly)

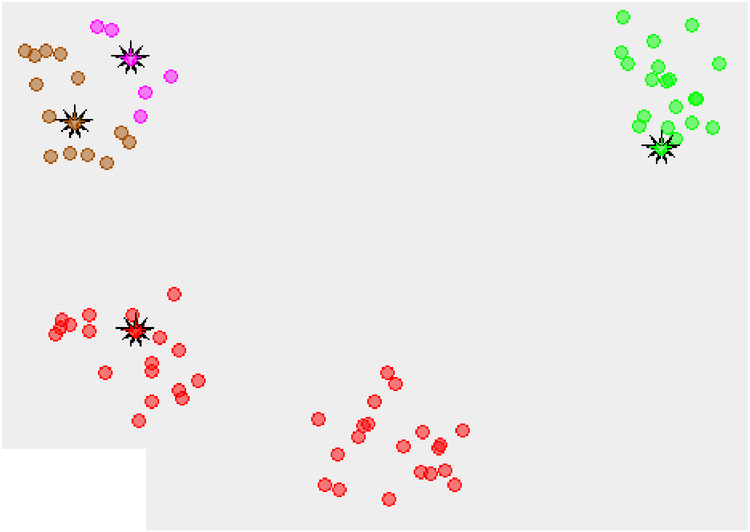
Initial



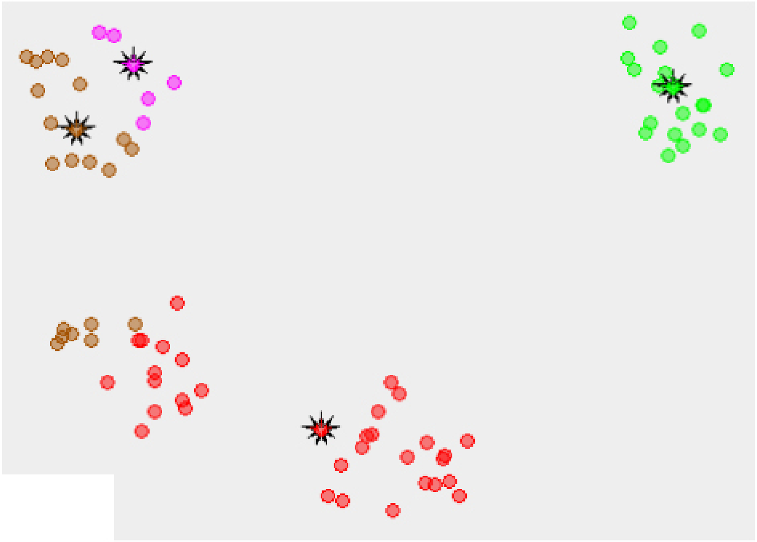
PAM 1



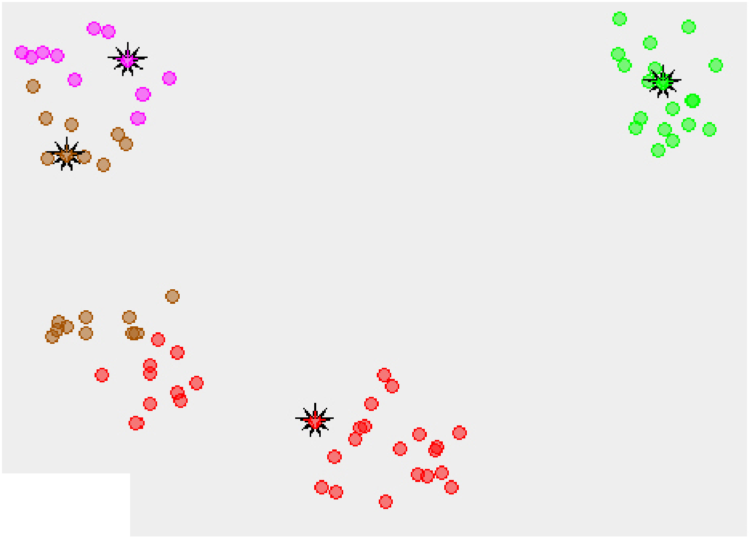
PAM 2



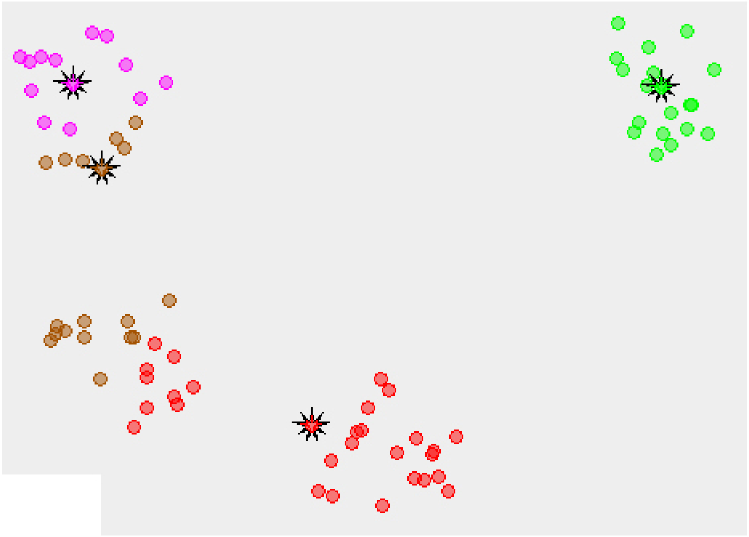
PAM 3



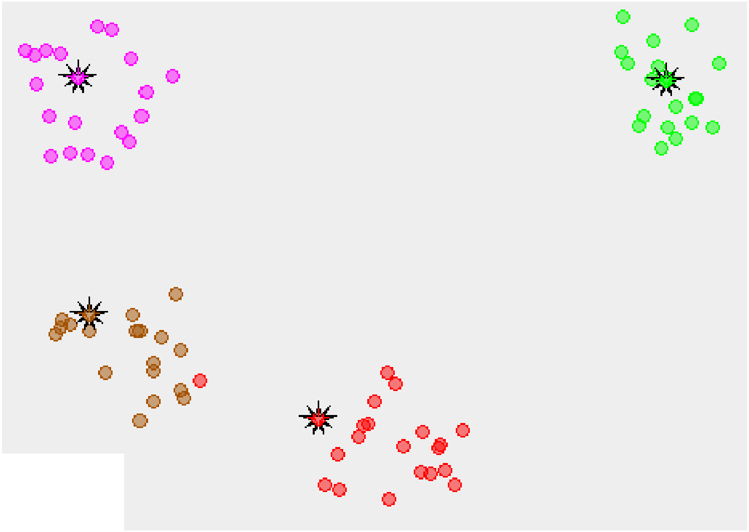
PAM 4



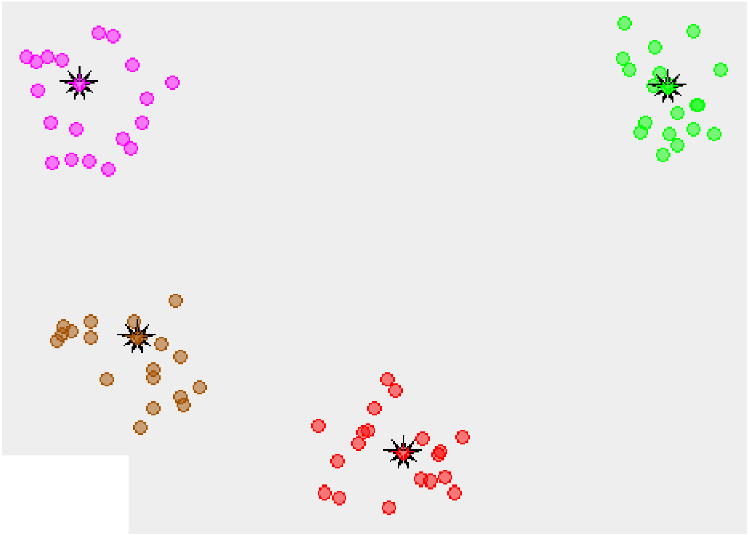
PAM 5



PAM 6



Final



Update of Objective cost function

Objective function: $J = \sum_{i=1}^{n-k} \text{dist}(i, c(i)),\ c(i) =$ centroid of i's cluster.

For each i store 2nd best centroid c'. Centroid m replaced by non-centroid t:

- For $i : c(i) = m$,

$$\Delta J = \begin{cases} \text{dist}(i,t) - \text{dist}(i,m), & \text{if } \text{dist}(i,t) \leq \text{dist}(i,c') : \text{ do nothing} \\ \text{dist}(i,c') - \text{dist}(i,m), & \text{if } \text{dist}(i,t) > \text{dist}(i,c') : \text{ assign } i \text{ to } c', \\ & \text{update } i\text{'s 2nd best centroid} \end{cases}$$

- For $i : c(i) \neq m$,

$$\Delta J = \begin{cases} 0, & \text{if } \text{dist}(i,t) \geq \text{dist}(i,c(i)) : \text{ do nothing} \\ \text{dist}(i,t) - \text{dist}(i,c(i)), & \text{if } \text{dist}(i,t) < \text{dist}(i,c(i)) : \text{ assign } i \text{ to } t \end{cases}$$

Outline

Recall Polygonal curves

Definition (Traversal)

Given polygonal curves $P = [p_1, \dots, p_{m1}]$, $Q = [q_1, \dots, q_{m2}]$, a traversal $T = (i_1, j_1), \dots, (i_t, j_t)$ is a sequence of pairs of indices referring to a pairing of vertices from the two curves such that:

1. $i_1, j_1 = 1$, $i_t = m_1$, $j_t = m_2$,
2. $\forall (i_k, j_k) \in T : i_{k+1} - i_k \in \{0, 1\}$ *and* $j_{k+1} - j_k \in \{0, 1\}$,
3. $\forall (i_k, j_j) \in T$: $(i_{k+1} - i_k) + (j_{k+1} - j_k) \geq 1$.

Definition (DTW distance)

Let $\mathcal{T}$ be the set of possible traversals for curves P, Q in $\mathbb{R}^d$. If $\| \cdot \|$ is the Euclidean distance in $\mathbb{R}^d$, their DTW distance $d_F(P, Q)$ is:

$$d_W(P, Q) = \min_{T \in \mathcal{T}} \sum_{(i_k, j_k) \in T} \| p_{i_k} - q_{j_k} \|.$$

Optimal traversal

All traversals start with pair $(1, 1)$ and finish with pair (m_1, m_2).

Length

The optimal traversal with regards to DTW, for curves of length m_1, m_2, has length t:

$$\max\{m_1, m_2\} \leq t \leq m_1 + m_2.$$

Computation

One (few) optimal traversal is computed in linear time by back-propagation from the filled table of Dynamic Programming computing DTW.

There may be several (exponential number) optimal traversals, of different lengths. The shortest one is hard to obtain.

Mean of two curves

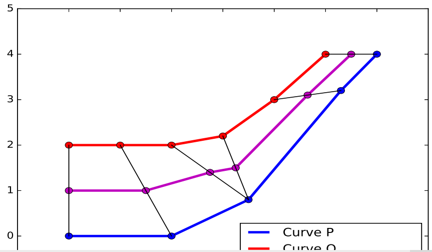
Definition (Mean DTW curve)

Given curves P, Q, let $T = (i_1, j_1), \ldots, (i_t, j_t)$ denote any optimal traversal for DTW, with

$$d_W(P,Q) = \sum_{(i_k, j_k) \in T} \|p_{i_k} - q_{j_k}\|.$$

The Mean DTW Curve is defined (not uniquely) as:

$$MC(P,Q) = [(p_{i_1} + q_{j_1})/2, \ldots, (p_{i_t} + q_{j_t})/2].$$

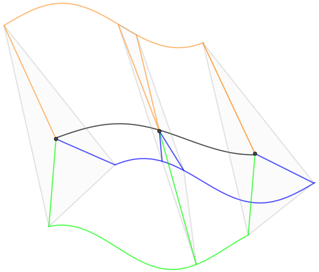


DTW Barycenter Averaging (DBA)

Compute a DTW-centroid of n sequences $S_1, \ldots, S_n \subset \mathbb{R}^d$, i.e. sequence $C \subset \mathbb{R}^d$ minimizing

$$\sum_i d^2_{DTW}(C, S_i), \quad i = 1, \ldots, n,$$

among sequences of length (number of points) λ, for given λ.

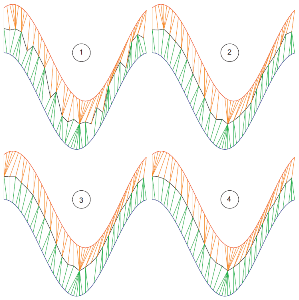


Temporary centroid (black) of length 3, given 3 sequences

DBA iteration

1. Compute a best traversal (for DTW) between C and each S_i;
2. each point $C[j]$ associated to matched points of all sequences S_i;
3. update $C[j]$, $j = 1, \ldots, \lambda$, as mean of these points.

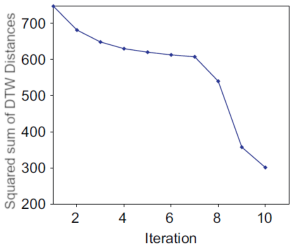
(Petitjean,Ketterlin,Ganharski:Global averaging for DTW,2011)



Centroid at step 3, confirmed at step 4 (same best traversal)

Convergence

DBA converges to global optimum (distance decreases, finite number of traversals), typically in $\leq \lambda$ iterations or much faster:



Iteration stops when current centroid C does not change from previous one C', or changes little: $d_{DTW}(C, C') < \epsilon$, for fixed ϵ.

Initialization

First, determine the length λ of target sequence C. Heuristically:

$$\lambda = mean\{len(S_1), \ldots, len(S_n)\} = \frac{1}{n}\sum_{i=1}^{n} len(S_i).$$

Then initialize C:

1: Pick random sequence S_0 among those of length $\geq \lambda$.

2. Take a length-λ random subsequence from S_0.

Suboptimal: build sequence with random points taken from all S_i (with normal distribution).

DBA Algorithm

Input: sequences $S_1, \dots, S_n$, initial sequence C.
Output: centroid sequence C of the S_i.

```
int $\lambda \leftarrow len(C)$
repeat
    seq $C' \leftarrow C$
    array A $\leftarrow [\emptyset, \dots, \emptyset]$                    array of $\lambda$ pointsets
    for $i = 1, \dots, n$ do
        set IPairs $\leftarrow$ index-pairs of best-traversal$(C, S_i)$
        for $p \in$ IPairs do                             $p = (u, v) \in (\mathbb{N}^*)^2,\ u \leq \lambda$
            $A[p_1] \leftarrow A[p_1] \cup \{S_i[p_2]\}$          pair $= (C[p_1], S_i[p_2]) \in (\mathbb{R}^d)^2$
    for $j = 1, \dots, \lambda$ do
        $C[j] \leftarrow$ mean$(A[j])$
until $C' == C$                                         or $d_{DTW}(C, C')$ small
return $C$
```

Mean of n curves

Definition

The Mean DTW/Fréchet Curve of a set of n curves is defined as the curve that minimizes the sum of the respective distances to all of them.

Exact DTW / Discrete-Fréchet Mean Curve of n curves computed in exp-time but approximated in $O(nm^2)$, where $m = \max$ curve length.

Recall: A complete binary tree is completely filled, with possible exception of the bottom level, which is filled from left to right.
Idea: Construct such a tree of height h, where the n curves correspond to leaves, $h = \lfloor \lg n \rfloor$. At each internal node, compute Mean Curve of 2 children.
Final mean curve corresponds to root of tree:

```
1: Define a Complete Binary Tree of height $h$ and root $\texttt{r}$
2: Randomly scatter the curves to the leaves of the tree
3: PostOrderTraversal($\texttt{r}$)
```

Post-order tree traversal

```
function PostOrderTraversal(node)
    if node.isLeaf() then
        return node.curve
    else
        leftCurve = PostOrderTraversal(node.leftChild)
        if node.rightChild $\neq$ NULL then
            rightCurve = PostOrderTraversal(node.rightChild)
        else
            rightCurve = NullCurve
        end if
        return MeanCurve(leftCurve, rightCurve)
    end if
end function
```

Outline

Outline

Accelerating update for k-medoids

Faster updates may however lose accuracy compared to PAM (swaps old medoid with every point); after every swap, compute J in $O((n-k)d')$.

1. Update within cluster

Swap old medoid m only with every non-medoid in same cluster as m.

Complexity: $n-k$ iterations instead of $k(n-k)$, so update $= O((n-k)^2 d')$.

2. Update à la Lloyd's

For every cluster: compute new medoid t, and swap it with old medoid m.

Medoid t minimizes $\sum_{i \in C} d(i,t)$ over all objects t in cluster C. Computed in $O(a^2 d')$, assuming clusters have $a \simeq n/k$ items.

Total Complexity $= O((ka^2 + k(n-k))d') = O((n^2/k + nk)d') = O(n^2 d')$ (Park-Jun'09).

Outline

Clustering Large Applications (CLARA)

General Idea: run entire algorithm with sample of size $n' \ll n$. Use s samples drawn independently, return best clustering.

Overall algorithm:

```
**for** $i = 1, \ldots, s$ **do**
    apply PAM on a random (uniform) sample of size $n'$
    assign $n$ points to $k$ computed centroids
    calculate the total cost of the partitioning
**end for**
return best partitioning
```

Experimental results recommend: $s = 5$, $n' = 40 + 2k$.

CLARA based on RANdomised Search (CLARANS)

- Update: swap m's with t's, for some randomly selected (m, t)'s only.
- Picking random $Q \subset \{1, \ldots, k\} \times \{1, \ldots, n-k\}$, s times.

```
Select $k$ centroids by some initialization method.
**for** $i = 1, \ldots, s$ **do**
    Cluster $n - k$ points to $k$ centroids by some assignment method.
    Randomly select set $Q$ of $|Q|$ pairs $(m, t)$, $|Q| < k(n-k)$.
    **for** $(m, t) \in Q$ **do**
        Swap $m$ with $t$; compute new objective value.
    **end for**
    Keep centroids with minimum objective value over $|Q|$ choices.
**end for**
Output centroids yielding minimum objective value over $s$ candidates.
```

Experiments recommend: $s = 2$, $|Q| = \max\{0.12 \cdot k(n-k), 250\}$.

(Ng-Han:IEEE Tran.Know.Data Eng'02, Theodoridis et al.:Patt.Recogn.,ch.14)

Outline

Improve Initialisation 1: Spread-out

initialization++ : k-means++ / k-medoids++:

(1) Choose a centroid uniformly at random; $t \leftarrow 1$.

(2) $\forall$ non-centroid point $i = 1, \ldots, n - t$, let $D(i) \leftarrow \min$ distance to some centroid, among t chosen centroids.

(3) Choose new centroid: r chosen with probability proportional to $D(r)^2$:

$$\text{prob[choose } r] = D(r)^2 / \sum_{i=1}^{n-t} D(i)^2.$$

Let $t \leftarrow t + 1$.

(4) Go to (2) until $t = k =$ given $\#$centroids.

Expected approximation ratio $= O(\log k)$ (Arthur-Vassilvitskii:SODA'07)

Similar algo for 2-approx of k-center (NP-hard prob)

Implement initialization++

Given $D(i) > 0$, $i = 1, \ldots, n-t$, compute $n-t$ (`float`) partial sums

$$P(r) = \sum_{i=1}^{r} D(i)^2, \qquad r = 1, \ldots, n-t,$$

and store them in an array (or binary tree) P. To avoid the $P(r)$'s being very large, we may normalize all $D(i)$'s by dividing them by $\max_i D(i)$.

Pick a uniformly distributed `float` $x \in [0, P(n-t)]$ and return

$$r \in \{1, 2, \ldots, n-t\} \; : \; P(r-1) < x \leq P(r),$$

where $P(0) = 0$: r chosen with probability proportional to $P(r) - P(r-1) \sim D(i)^2$. Can find r by binary search in array P.

Improve Initialisation 2: Concentrate

Select centroids close to dataset's center of mass (and to each other) as follows.

(1) Calculate symmetric $n \times n$ distance matrix of all objects, i.e. all distances d_{ij} from every object $i = 1, \ldots, n$ to every other object $j = 1, \ldots, n, i \neq j$.
(2) For object i compute

$$v_i = \sum_{j=1}^{n} \frac{d_{ij}}{\sum_{t=1}^{n} d_{jt}}, \qquad i = 1, \ldots, n.$$

(3) Return the k objects with k smallest v_i values.

Algorithm proposed in (Park-Jun'09).

Outline

Assignment by direct method

Exact approach for small data

At each iteration:

1. For every point, compute distance to every centroid.
2. Return (exact) nearest centroid.

Approximate approach for big data

At each iteration:

1. Index k centroids into data-structure, e.g. LSH hashtables.
2. For every non-centroid point, run ANN to find nearest centroid.
3. Return (approximate) nearest centroid.

This is the standard approach in almost all big data implementations today.

Assignment by Range search

Reverse approach (ANN)

- Index n points into L hashtables: once for entire algorithm.
- LSH/DBH TableSize $\leq n/8$: avoid buckets with very few items.
- At each iteration, for each centroid c, range/ball queries centered at c.
- Mark assigned points: either move them at end of LSH buckets (and insert "barrier", or mark them using "flag" field).
- Multiply radii by 2, start with min(dist between centers)/2, until most balls get no new point (centers mapped to buckets once)
- For a given radius, if a point lies in ≥ 2 balls, compare its distances to the respective centroids, assign to closest centroid.
- At end: for every unassigned point, compare its distances to all centroids

Outline

Internal evaluation

Evaluate clustering without reference to objective function. Try to capture meaning of clustering.

- Let k be the number of computed clusters.
- Internal evaluation considers the given pointset and the clusters, produces quality coefficient for each partition; k may be a parameter.
- External evaluation: use known class labels and benchmarks; often created by humans.

In the sequel we present an internal evaluation, mainly Silhouette.

Silhouette

-- For $1 \leq i \leq n$, $a(i) =$ average distance of i to other objects in same cluster.
-- Let $b(i) =$ average distance of i to objects in *next best* (neighbor) cluster, i.e. cluster of 2nd closest centroid.

Silhouette of Object *i*

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} = \left\{ \begin{array}{ll} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i) \end{array} \right\} \in [-1, 1].$$

Interpret silhouette

$s(i) \rightarrow 1$: *i* seems correctly assigned to its cluster;
$s(i) \simeq 0$: borderline assignment (but not worth to change);
$s(i) \rightarrow -1$: *i* would be better if assigned to next best cluster.

Silhouette: Cluster and clustering

Specific clusters

-- Evaluate a cluster: Compute average $s(i)$ over all i in some cluster.
-- If k is too large or too small, some clusters shall display much smaller silhouettes than the rest.
-- Silhouette plots are used to improve k: try different k's and see if clusters have roughly equal silhouettes.

Overall Clustering

Overall Silhouette coefficient = average $s(i)$ over $i = 1, \ldots, n$.

High if well clustered, low may indicate bad k (or existance of outlier points).