Clustering

(Part 3)

Cluster Evaluation

GOALs:

- Clustering tendency: assessment whether the data contain meaningful clusters, namely clusters that are unlikely to occur in random data.
- Unsupervised evaluation: assessment of the quality of a clustering (or the relative quality of two clusterings) without reference to external information.
- Supervised evaluation: assessment of the quality of a clustering (or the relative quality of two clusterings) with reference to external information (e.g., class labels).

Clustering tendency: Hopkins statistic

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 $t \ll N$ (typically $t < 0.1 \cdot N$) let:

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

The Hopkins Statistic is

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

Clustering tendency: Hopkins statistic (cont'd)

- $H(P) \simeq 1$: P is likely to have a clustering structure.
- $H(P) \simeq 0.5$: P is likely to be a random set.
- $H(P) \ll 0.5$: the points of P are likely to be well (i.e., regularly) spaced.

Exercise

Show that setting $t \in o(N)$, $H(P)$ can be efficiently computed in MapReduce.

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 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.

Unsupervised evaluation: Silhouette coefficient

Let $\mathcal{C}$ be a clustering (intended as a partition) of a pointset P.

For a point $p \in P$ belonging to some cluster $C \in \mathcal{C}$

- $a_p =$ 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 $\mathcal{C}$ can be assessed through the

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

C is a "good" clustering if $s_{\mathcal{C}} \simeq 1$, i.e., for most points p, $b_p \gg a_p$.

Unsupervised evaluation for big data

- Computing cohesion and separation or the average silhouette coefficient exactly, requires computing $\Theta\left(|P|^2\right)$ 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 (e.g., the centroid in Euclidean space). Then, for each $p \in C$, the value b_p is approximated with the average distance between p and the points of the *cluster* $C' \neq C$ *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 $C' \neq C$, 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 $\mathcal{C}$ 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 $C \in \mathcal{C}$ 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 (i.e., min impurity when all points of C belong to the same class), to $\log_2 L$ (i.e., max impurity when all classes are equally represented in C).

Supervised evaluation: entropy (cont'd)

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 (i.e., all points of class i concentrated in the same cluster), to $\log_2 K$ (i.e., the points of class i are evenly spread among all clusters), where K is the number of clusters.

It is defined also when points belong to multiple classes (e.g., categories of wikipedia pages).

Supervised evaluation: entropy (cont'd)

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

If $k \cdot L = o(N)$, with $N = |P|$ use the following MapReduce algorithm (assume that each point of P comes with its cluster and class labels):

- Round 1: Partition P arbitrarily in $\sqrt{N}$ subsets of equal size $P_1, P_2, \ldots, P_{\sqrt{N}}$. Within each P_j compute, for each class i and cluster C, the number $m_{C,i}(j)$ of points of class i in $C \cap P_j$ (only values $m_{C,i}(j) > 0$ need to be computed).
- Round 2: For each class i and cluster C, gather all $m_{C,i}(j)$'s and compute their sum $m_{C,i} = \sum_{j=1}^{\sqrt{N}} m_{C,i}(j)$.
- Round 3: Gather all $m_{C,i}$'s and compute the desired entropies. (Note that the m_i's and m_C's can be easily derived from the $m_{C,i}$'s.)

The algorithm (patterned after the category counting strategy seen earlier) requires $O\left(\sqrt{N} + k \cdot L\right)$ local space and linear aggregate space.

Case Study

Analysis of votes in the Italian Chamber of Deputies

Goal: Cluster Italian deputies based on their votes in Parliament

- Data source: Openparlamento (from Openpolis) `https://parlamento17.openpolis.it/`
- 630 deputies
- Restriction to key votes, i.e., high political relevance. Overall 149 votes, from beginning of term until end of 2015
- Dataset: one row ("point" for each deputy) containing
  - ID of the deputy
  - Group: 24 = Mixed Group; 71 = PD; 90 = FI; 115 = Fratelli d'Italia; 117 = M5S; 119 = Sinistra Italiana; 120 = Lega Nord; 121 = Scelta Civica; 124 = Area Popolare; 125 = Democrazia Solidale;
  - For each vote: -2 = against; -1 = abstention; 0 = missing; 2 = in favor.
- Distance: Manhattan distance

Case Study (cont'd)

- All deputies except for Mixed Group (630-62=568 deputies)
- Algorithm: k-means with $k = 12$
- Results:

	Cluster											
Group	0	1	2	3	4	5	6	7	8	9	10	11
71	174	26	2	2	12	26	13	32	10	3	0	0
90	1	1	0	17	0	0	0	0	0	0	36	0
115	0	0	0	7	0	0	0	0	0	1	0	0
117	0	0	0	2	0	0	0	0	0	89	0	0
119	0	0	0	1	4	0	1	0	0	0	0	25
120	0	0	0	2	1	0	0	0	0	10	3	0
121	4	2	10	1	0	1	0	0	3	1	0	1
124	3	11	1	4	0	0	0	4	7	0	1	0
125	1	3	1	1	1	0	1	4	1	0	0	0
Entropy	0.37	1.54	1.29	2.42	1.34	0.23	..	..	..	..	..	0.24

- Entropy of PD (G.71): 2.12; Entropy of M5S (G.117): 0.15

Case Study (cont'd)

- Deputies from PD (G.71), FI (G.90), M5S (G.117) who never changed group (422 deputies out of 446)
- Algorithm: k-means with $k = 5$
- Results:

	Cluster				
Group	0	1	2	3	4
71	186	43	0	0	51
90	0	3	48	0	0
117	0	0	0	91	0
Entropy	0	0.35	0	0	0

- Unlike FI and M5S, the PD group is split into 3 subgroups probably reflecting a political subdivision (e.g., Orfini and Guerini in Cluster 0; Bersani in Cluster 1; Cuperlo in Cluster 4)

Hierarchical clustering

- Produces a hierarchy of nested clusterings of decreasing cardinalities.
- No need to fix the number of clusters apriori or to choose cluster centers.
- Two alternative high-level strategies:
  - Agglomerative: Starts with each input point in a separate cluster and progressively merges suitably selected pairs of clusters. It is the most common strategy and we focus on it.
  - 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
```

Observations:

- In order to instantiate the algorithm one needs to decide when to stop (*stopping condition*) and which pair of clusters to merge at each iteration (*closest clusters*)
- 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)

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

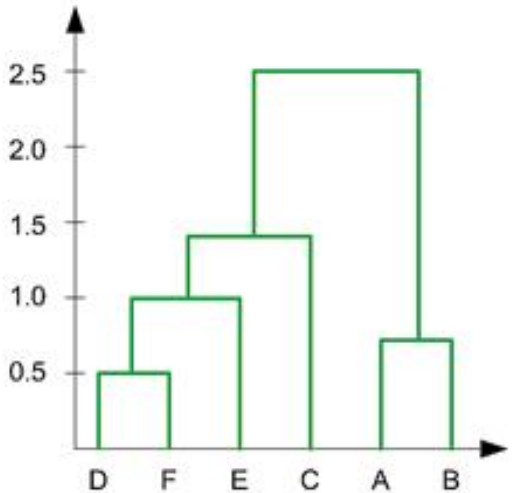
We will focus on single linkage

Example

- Distance matrix (6 points: A,B,C,D,E,F):

Dist	A	B	C	D	E	F
A	0.00	0.71	5.66	3.61	4.24	3.20
B	0.71	0.00	4.95	2.92	3.54	2.50
C	5.66	4.95	0.00	2.24	1.41	2.50
D	3.61	2.92	2.24	0.00	1.00	0.50
E	4.24	3.54	1.41	1.00	0.00	1.12
F	3.20	2.50	2.50	0.50	1.12	0.00

- Dendogram (stopping condition: 1 cluster left):



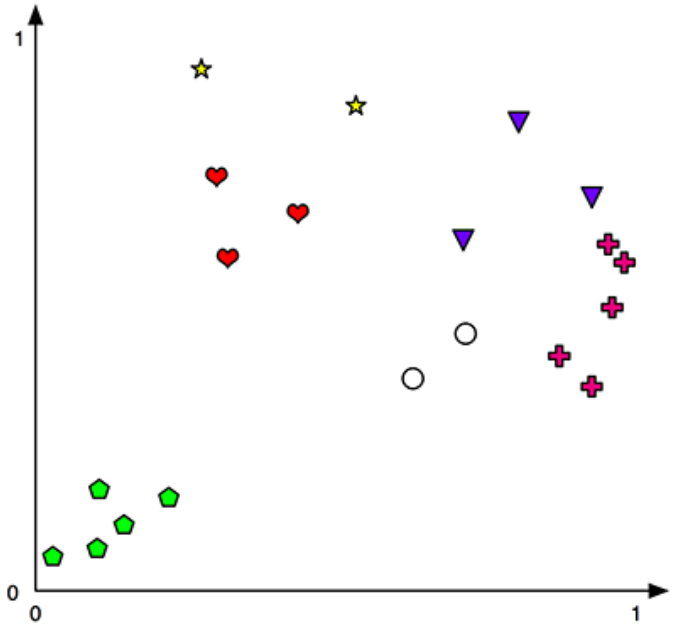
- It shows sequence of mergings: D-F; A-B; DF-E; DFE-C; DFEC-AB
- y-axis: inter-cluster distances. E.g., root of subtree for DFEC has $y = 1.41$, which is the distance between cluster DFE and cluster C

Stopping condition

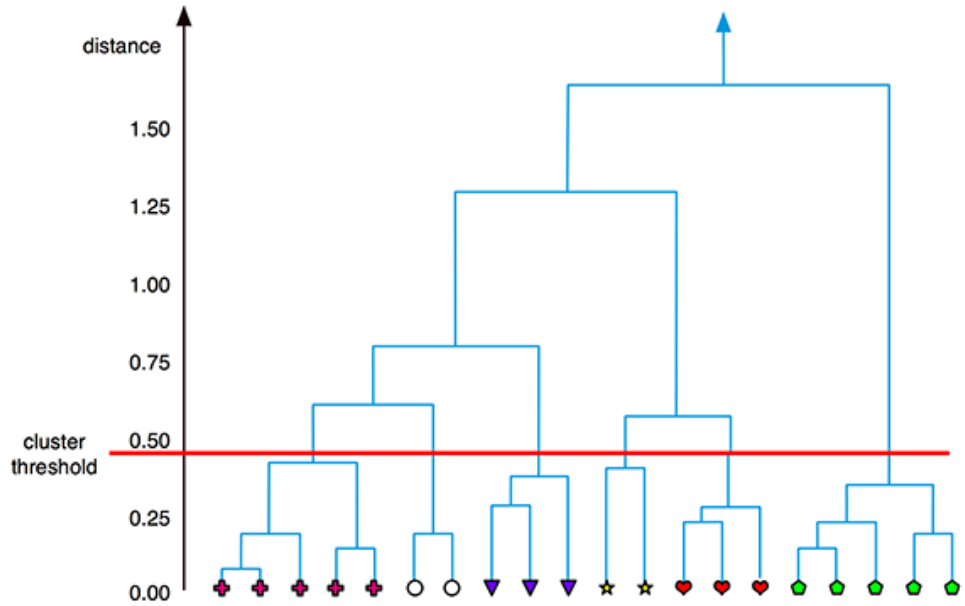
Depending on the application, the merging process can be stopped using one of the following conditions

- A desired number K of clusters is obtained (i.e., after $N - K$ iterations)
- 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 (e.g., threshold on maximum distance of a point from the centroid of its cluster in Euclidean space)

Example



sample dataset



hierachical clustering of the sample dataset

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 ($\Theta(N)$ space).

Straightforward implementation:

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

Time Complexity (cont'd)

Improved implementation:

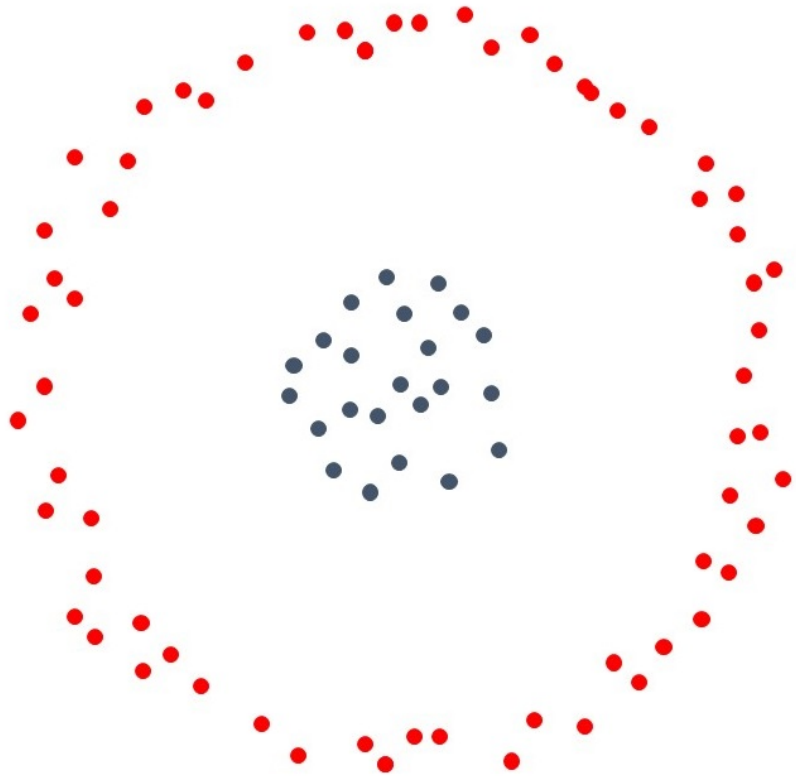
- Precompute all $\Theta\left(N^2\right)$ distances and store each pair of points (x, y) into a min-heap H, using $d(x, y)$ as a key.
- Extract, one after the other, the pairs of points from H in increasing order of distance. After extracting pair (x, y), if the two points belong to different clusters,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

Remarks

- More efficient implementations for both single and complete linkage ($O\left(N^2\right)$ time and $O(N)$ space) exist: see [MC12] for details.
- For a Spark implementation (single linkage), see [J+15]

Example

The pointset in the following example exhibits two natural clusters



A hierarchical clustering run with single linkage and a suitable cluster threshold would capture the two clusters accurately, while a center-based clustering would not.

Observations on Hierarchical Clustering

Pros

- Useful when a *hierarchical taxonomy* is sought and/or a precise number of clusters cannot be established apriori
- Can capture clusters of non-elliptical shapes (especially single linkage): see example in the previous slide.

Cons

- Does not optimize any specific objective function
- Sensitive to noise (especially single linkage): *A few points bridging the gap between two different clusters can cause it to do the wrong thing* [BHK18].
- Computationally expensive. The development of efficient hierarchical clustering (approximate) algorithms for very large pointsets is still an open research problem

Examples of theory questions on clustering

- Define the Manhattan distance for points in $\Re^d$, and show that it satisfies the triangle inequality
- Briefly describe the 4-approximation MapReduce algorithm for the k-center clustering problem, analyzing its performance for $k = o(N)$, where N is the number of input points.
- Suppose that you want to open some hospitals to serve N cities so that each city is at distance at most t from a hospital and the costs (proportional to the number of opened hospitals) are minimized. What would you do?
- Show that the k-means algorithm always terminates.
- Describe what the *unsupervised evaluation* of a clustering is, and define a measure that can be used to this aim.

References

LRU14 J. Leskovec, A. Rajaraman and J. Ullman. Mining Massive Datasets. Cambridge University Press, 2014. Sections 3.1.1 and 3.5, and Chapter 7

BHK18 A. Blum, J. Hopcroft, and R. Kannan. Foundations of Data Science. Manuscript, June 2018. Chapter 7

TSK06 P.N.Tan, M.Steinbach, V.Kumar. Introduction to Data Mining. Addison Wesley, 2006. Chapter 8.

MC12 F. Murtagh, P. Contreras. Algorithms for hierarchical clustering: an overview. Wiley Interdisc. Reviews: Data Mining and Knowledge Discovery 2(1):86-97, 2012.

J+15 C. Jin et al. A Scalable Hierarchical Clustering Algorithm Using Spark. BigDataService 2015: 418-426.

Errata

Changes w.r.t. first draft:

- Slide 7: in the last bullet point a sentence regarding the approximation of a_p has been added.
- Slide 23: modified observation about sensitivity to noise.