Lecture 23

Clustering - a case study

- Agglomerative clustering
- Min radius clustering
  - approximate best cluster number (for given rad
  - approximate best radius (for given cluster number)
Clustering

Can we find $K$ students to hold candy bowls so that each 6,046 student is within $r$ seats from a candy bowl?

This is an example of a clustering problem. Clustering is one of the most widely used tools for analyzing data sets:

- websites, search results, click stream data
- customer behavioral patterns, community discovery in social networks, photographs, music, movies, genes & proteins, word behaviors, biological species ... (lots more examples!)
The scenario

Given many objects

The goal

Divide into groups so that

- Objects in same group are close
- Objects in different groups are far apart

Need to define:

- The objects
- Notions of distance (What is "close" + "far"?)

Many formalizations

How would you cluster these?
radius
aggregate
purple - might be good for computer vision to find connected objects

green - all elements in same group are pretty close

blue - small number of groups

What should a good clustering definition look like?
(there are millions of formulations!)

One possibility:

Hierarchical Agglomerative Clustering

A "spacing" of a $k$-clustering is the minimum distance between any pair of points lying in different clusters.

Find $k$-clustering with maximum possible spacing

(assume (1) $d(p_i, p_i) = 0$
(2) $d(p_i, p_j) = 0$ for $i \neq j$
(3) $d(p_i, p_j) = d(p_j, p_i)$ )
Hierarchical Agglomerative Clustering: Single link

Idea: "grow" a graph on nodes (objects)

st. connected components in clusters,

+ bring nearby points into same cluster quickly

(don't want nearby pts to be in different but nearby clusters)

⇒ seems like a good idea to consider:

pairs in order of distance

Algorithm: Initially: each object is own cluster

H is graph with node for each object, no edges

1. sort pairs of objects by distance

i.e. \((u_1v_1), (u_2v_2), \ldots, (u_mv_m)\) s.t. \(d(u_i,v_i) \leq d(u_{im},v_{im})\)

2. For \(i = 1 \) to \(m\),

   - if \(u_i \neq v_i\) not already in same cluster,
     - merge their clusters
     - insert edge \((u_iv_i)\) in \(H\)

   - stop when only \(k\) clusters left

Comments:

* \(O(n^2 \log n)\)

* \(H\) is union of trees (never add edge to nodes in same cluster)

* at end, \(H\) is union of \(k\) clusters,

* exactly same as Kruskal's MST algorithm when \(k=1\) (general \(k\), MST minus \(k-1\) costliest edges)
Claim: Output is \( k \)-clustering of max spacing

Proof:

Fact: Output \( C = \langle c_1 \ldots c_k \rangle \) has spacing \( d^* \), which is distance of first pair not considered by algorithm.

Note: All edges in \( H \) have distance \( \leq d^* \)

Consider \( C' = \langle c'_1 \ldots c'_k \rangle \)

Can \( C' \) have better spacing?

\[ C + C' \supseteq \exists \ C_r \text{ s.t. not subset of any } C_s \text{ in } C' \]

\[ \Rightarrow \exists \ p_i, p_j \in C_r \text{ s.t. } p_i \in C'_s \text{ and } p_j \notin C'_s \]

No! will show that spacing of \( C' \leq d^* \)

here we use that \( C' \) is also \( a \) \( k \)-clustering.
since \( p_i, p_j \) in \( C_r \),

there is a path of edges, each of length \( \leq d^* \)

joining them

Since \( p_i \in C'_s \) \& \( p_j \in C'_s \), at some point the

path leaves \( C'_s \) \& goes to \( C'_t \)

(say between \( p_i \) & \( p_j \))

\[ \Rightarrow \text{distance between } C'_s + C'_t \leq d^* \]

\[ \Rightarrow \text{spacing of } C' \leq d^* \]

\[ \therefore \text{no other k-clustering has larger spacing} \]

But is this the kind of clustering you want?

often used in practice.....
Another Alternative:

\[ \text{Min radius clustering: Given } \mathbf{K}, R \]

Given a set of points \( P \)

\[ s.t. \quad P \subseteq \mathbb{R}^n \quad (\text{distances satisfy } \Delta 
\]

is there a subset of points \( C \subseteq P \)

such that

1) \( |C| = k \)
2) each point in \( P \) is distance \( \leq R \)

from some point in \( C \) ?

\((k, r)\) - radius clustetable

\[
F[C] = \max_{p \in P} \min_{c \in C} d(p, c)
\]

Cost of clustering according to \( C \)
An idea for an algorithm:

For every $v \in P$, let

$$S_v \leftarrow \text{all points in } P \text{ within distance } R \text{ of } v$$

which gives a collection of sets $S_1, S_2, \ldots, S_m$.

New question:
Is there a subcollection of size $k$

$$S_{i_1}, S_{i_2}, \ldots, S_{i_k}$$

such that $\bigcup_{j=1}^k S_{i_j} = P$?

What does this remind you of?

Set Cover

- Input: $k, S_1, \ldots, S_n$ s.t. $S_i \subseteq P$
- Output: Is there $S_{i_1}, \ldots, S_{i_k}$ such that $\bigcup_{j=1}^k S_{i_j} = P$?

But Set-Cover is NP-complete!

(careful - this reduction is in the wrong direction!
but min radius clustering is still NP-complete!)
Still - we can use $O(\log n)$-approximation algorithm for set-cover to get:

An approximation algorithm for clustering:

**Guarantee:**

if points are $(k, r)$-radius clusterable

then algorithm outputs a $(k, \log n, r)$-radius clustering

**Algorithm**

use $O(\log n)$ approx for set-cover on $S_x$'s

Note: above gives right radius $r$

but approximates $k$, number of centers.

What if you care about $k$ and not $r$?
Another approximation algorithm for clustering:
(this time, fix number of clusters & allow worse radius)

**New Guarantee:**
If points are \((k, r)\)-radius clusterable then output is \((k, 2r)\)-radius clustering
(recall that points are in \(\mathbb{R}^n\))

**Algorithm:**
\(i \leftarrow 0\)
\(S \leftarrow \text{all points}\) \(\quad (S\text{ is "remaining unclustered point)}\)

While \(S\) not empty
\(i \leftarrow i+1\)
pick arbitrary \(v \in S\)
let new cluster \(C_i \leftarrow \text{all points within distance} \leq 2r\) of \(v\)
\(S \leftarrow S \setminus C_i\)

Output \(C_1, \ldots, C_i\)
Claim If points are $(k,r)$-r.c.
then after $k$ loops, $S$ will be empty.

Corollary
$O(nk)$ runtime
output satisfies guarantee

Proof of Claim

Let $C^* = <C_1^*, \ldots, C_k^*>$ be a $(k,r)$-r.c.
with centers $p_1, \ldots, p_k$ respectively.

For each $C_i^*$
pick any pair $u, v \in C_i^*$
\[ d(u,v) \leq d(u, p_i) + d(p_i, v) \]
\[ \leq 2r \]

So, once algorithm picks any $v \in C_i^*$
it will put all other points in $C_i^*$ into the same cluster!

So, at each round, get rid of anew $C_i^*$ from $S$, so after $k$ rounds, no points are left!
Other clustering measures:

- diameter
- average distance to center
  - not requiring point in $P$ to be center
  - not requiring all points to be clustered
- graph based - min/sparse cuts.