Algorithms in Bioinformatics

Clustering pt 2

Recap

- HW up
- HW1 graded

Last time: Ended w/ k-means clustering

---

k-Means Clustering Problem:
Given n data points, find k center points minimizing the squared error distortion.

Input: A set, $\mathcal{V}$, consisting of n points and a parameter k.

Output: A set $\mathcal{X}$ consisting of k points (called centers) that minimizes $d(\mathcal{V}, \mathcal{X})$ over all possible choices of $\mathcal{X}$.

More formally: centers $X = \{x_1, \dots, x_k\}$

$$d(v, X) = \min_{1 \le i \le k} d(v, x_i)$$

(where $d(v, x_i)$ is Euclidean dist)

Then squared error distortion for a set of points

$$V = \{v_1, \dots, v_n\}$$

+ centers $X = \{x_1, \dots, x_k\}$

is: $$d(V, X) = \frac{\sum_{i=1}^{n} d(v_i, X)^2}{n}$$

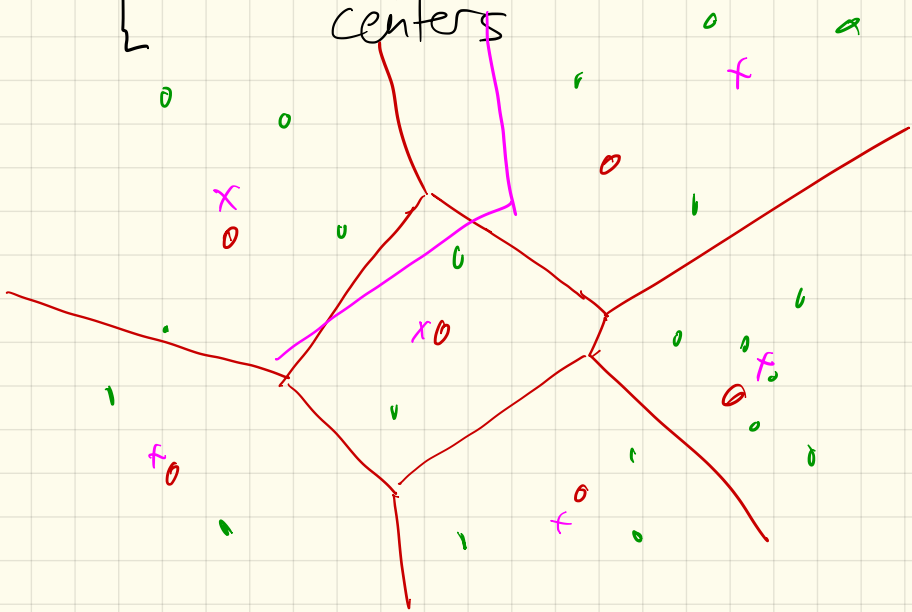
Unfortunately, NP-Hard.

<u>Lloyd's algorithm</u>:

- Begin with k arbitrary centers (typically chosen randomly)

<u>Repeat</u>: until centers don't change

- Assign each point to its closest center
- Compute center of mass of each cluster
- Elect these as k new centers



Termination:

This algorithm will always terminate.

pf:

There are only n^k clusterings possible.

There are possible centers.

Each iteration of loop improves "score".

So algorithm must terminate.

However: we might not compute OPT.

<u>Note</u>: Won't always get the best clustering!

For fixed n & k, there are examples where

$$\frac{d(V,X)}{d(V,OPT)} \text{ is } \underline{\text{unbounded}}.$$

(This can even hold with high probability.)

Interesting new work: (2007)

Instead of choosing centers at random, weight each point by squared distance from closest center.

<u>Result</u>: $O(\log n)$ approximation guarantee!

Variations

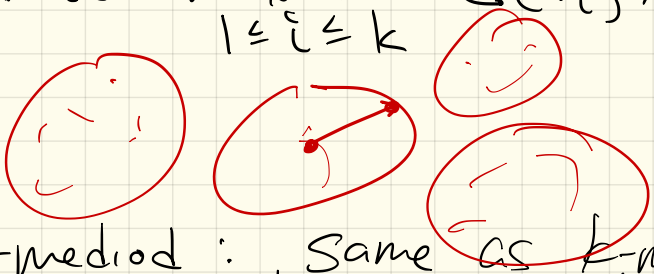
K-Median:

minimize $\sum_{i=1}^{n} d(v_i, X)$

not divided by n

K-center:

minimize $\underset{1 \le i \le k}{\text{maximum}}\ d(v_i, X)$



K-mediod:

Same as k-means, but must choose input point as a center.

(local search)

Problems with all:

These emphasize the homogeneity condition (things w/in a cluster are similar), but ignore separation.

Also:

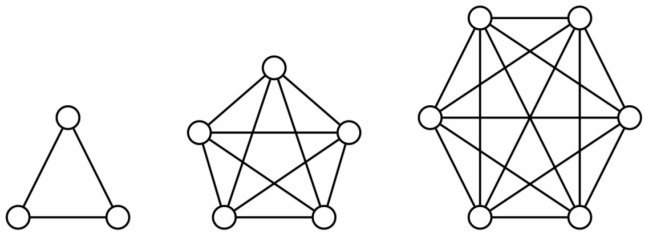
- Weak on varying size cluster (for k-means)
- sensative to outliers (for k-center)

Sec 10.4: Clustering - Corrupted Cliques

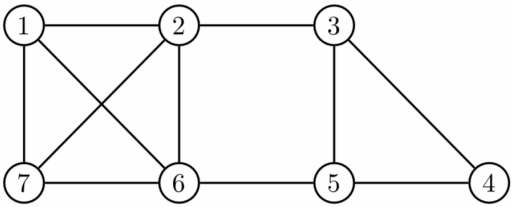
Dfn: K_n: complete graph

k-clique: subgraph of k vertices that forms K_k

clique graph: graph made of cliques



(a)



(b)

Gene application:

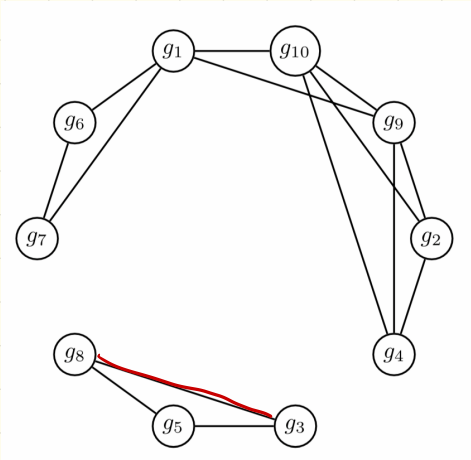
- Pick an interesting threshold θ.
- Declare genes are "close" if $d(g_i, g_j) < \theta$

	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}
g_1	0.0	8.1	9.2	7.7	9.3	**2.3**	**5.1**	10.2	**6.1**	**7.0**
g_2	8.1	0.0	12.0	**0.9**	12.0	9.5	10.1	12.8	**2.0**	**1.0**
g_3	9.2	12.0	0.0	11.2	**0.7**	11.1	8.1	**1.1**	10.5	11.5
g_4	7.7	**0.9**	11.2	0.0	11.2	9.2	9.5	12.0	**1.6**	**1.1**
g_5	9.3	12.0	**0.7**	11.2	0.0	11.2	8.5	**1.0**	10.6	11.6
g_6	**2.3**	9.5	11.1	9.2	11.2	0.0	**5.6**	12.1	7.7	8.5
g_7	**5.1**	10.1	8.1	9.5	8.5	**5.6**	0.0	9.1	8.3	9.3
g_8	10.2	12.8	**1.1**	12.0	**1.0**	12.1	9.1	0.0	11.4	12.4
g_9	**6.1**	**2.0**	10.5	**1.6**	10.6	7.7	8.3	11.4	0.0	**1.1**
g_{10}	**7.0**	**1.0**	11.5	**1.1**	11.6	8.5	9.3	12.4	**1.1**	0.0

(a) Distance matrix, **d** (distances shorter than 7 are shown in bold).

$\theta = 7$

Build a graph:



Goal: If we find a good Θ maybe the clusters will form into cliques!

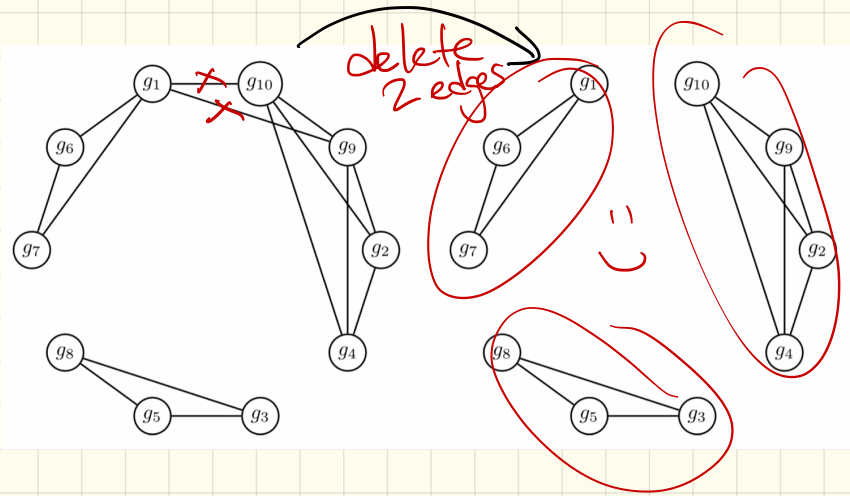
Problem:

- Errors in measurement
- No perfect Θ

Corrupted Cliques Problem:
Determine the smallest number of edges that need to be added or removed to transform a graph into a clique graph.

Input: A graph G.

Output: The smallest number of additions and removals of edges that will transform G into a clique graph.



Sadly (again), NP-Hard

Heuristics:

1. PCC: Parallel Classification w/ Cores
2. CAST: Cluster Affinity Search Technique

Trade-off:

- PCC is more theoretically sound, in some ways
- But CAST is faster

Again, same issue if clusters aren't uniform - one Θ just won't do it!

PCC Overview:

Consider brute forcing a solution for a subset $S' \subset S$ (where S is all the genes).

Let $\{C_1, \dots, C_k\}$ be S''s optimal clustering.

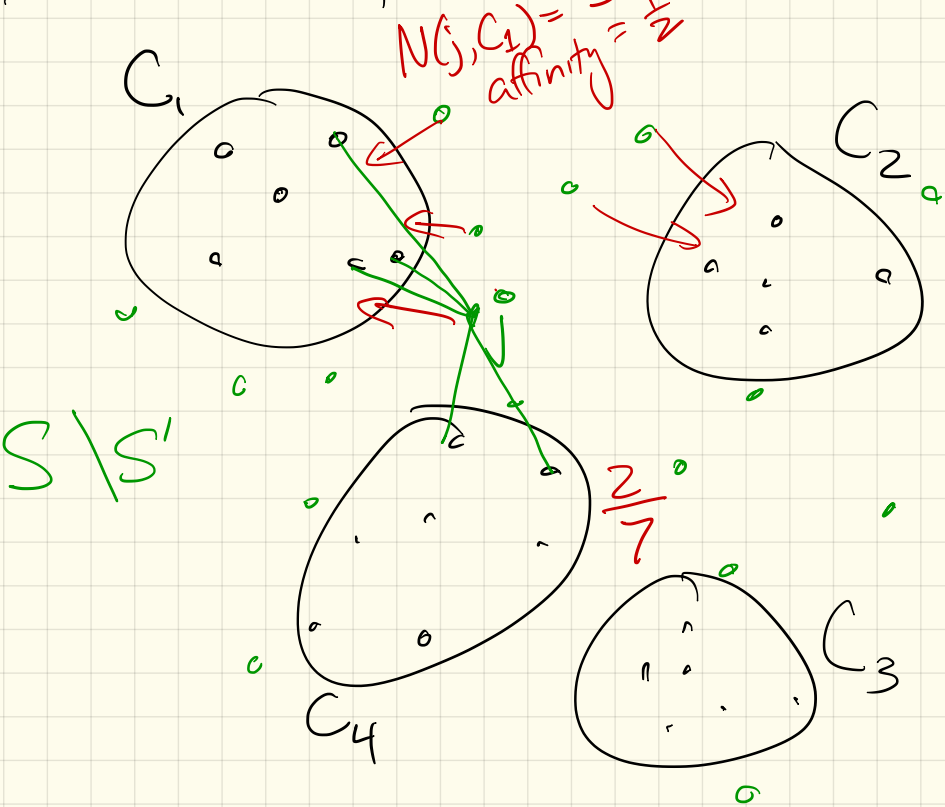
↳ brute force

Idea:

- let $S \setminus S'$ be rest of the genes
- let $N(j, C_i)$ = # of edges between gene $j \in S \setminus S'$ and cluster C_i
- affinity of gene j to cluster i:

$$\frac{N(j, C_i)}{|C_i|}$$

PCC cont: picture:



Maximal affinity: assign j to cluster with highest affinity.

Ben-Dor et al 1999:

Prove that if S' is large enough (random enough) resulting entire cluster is fairly good.

Downside: Computing S''s optimal cluster is NP-Hard!

(So, actual implementation makes S' quite small.)

```
PCC(G, k)
S ← set of vertices in the distance graph G
n ← number of elements in S
bestScore ← ∞
Randomly select a "very small" set S' ⊂ S, where |S'| = log log n
Randomly select a "small" set S'' ⊂ (S \ S'), where |S''| = log n.
for every partition P' of S' into k clusters
    Extend P' into a partition P'' of S''
    Extend P'' into a partition P of S
    if score(P) < bestScore
        bestScore ← score(P)
        bestPartition ← P
return bestPartition
```

exponential in $|S'|$!

$O(n^2)$

Note: • 2-stage!

• score(P) := score from prior page

Runtime:

Number of iterations =

\# of partitions of S'

$= k^{|S'|}$

Choose $|S'| = \log \log n$

$$k^{\log \log n} = (\log n)^{\log k}$$

$$\approx \underline{(\log n)^c}$$

(if k is small)

In the loop: $O(n^2)$

$O(n^2 \log^c n)$ (# time for dist calculation)

(still fairly slow)

CAST:

- define $d(i, C) = \dfrac{\sum_{j \in C} d(i,j)}{|C|}$

- again, i is close to C if $< \theta$ away.

- CAST iteratively builds a partition P of S


```
CAST(G, θ)
  S ← set of vertices in the distance graph G
  P ← ∅
  while S ≠ ∅
      v ← vertex of maximal degree in the distance graph G.
      C ← {v}
      while there exists a close gene i ∉ C or distant gene i ∈ C
          Find the nearest close gene i ∉ C and add it to C.
          Find the farthest distant gene i ∈ C and remove it from C.
      Add cluster C to the partition P
      S ← S \ C
      Remove vertices of cluster C from the distance graph G
  return P
```

doesn't necessarily result in uniform clusters

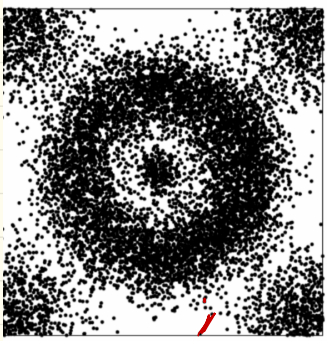
Issues:

- No guarantee
- However, works well (& is fast) for some data

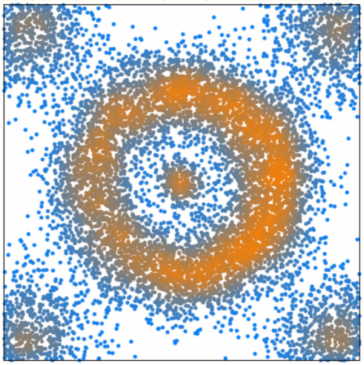
Density estimation:

Let's remove Θ, or even k, & try to "infer" the centers from our data.

Consider data points in $\mathbb{R}^m$:



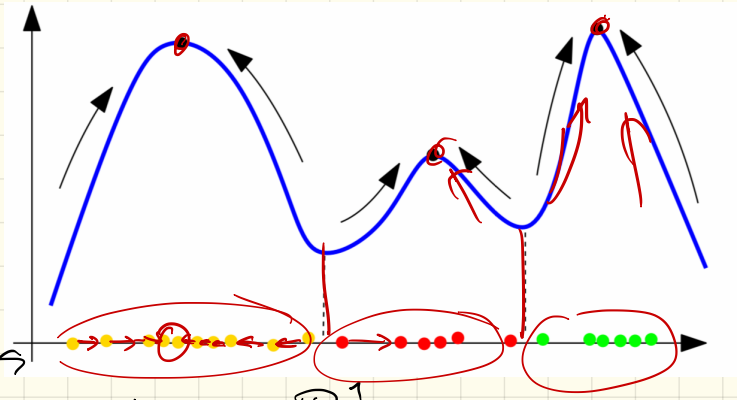
Now, give each a "weight" — say, distance to closest next point:



Idea:

Group points to nearby maxima

1-dim picture first:



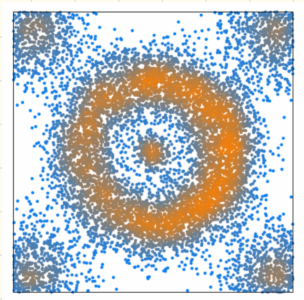
← function value.

points in $\mathbb{R}^1$

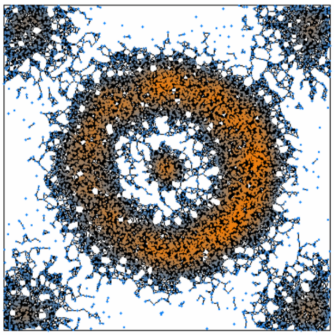
We'll consider a graph based m-dim version:

[Koontz, Narendra & Fukunaga 1976]

Build a neighborhood graph:



→



Then, apply something called gradient method:

- For each vertex, select an edge to its highest value neighbor.

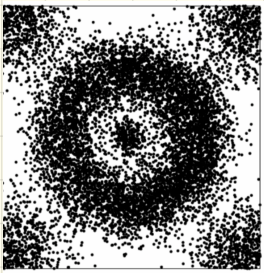
(Think of this as a flow to maxima)

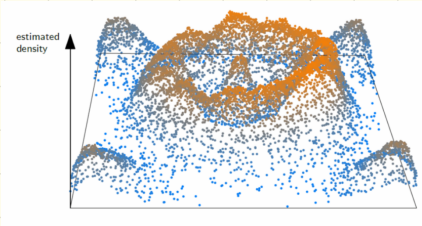
Form clusters simply by keeping these edges as the cluster.

Issues:

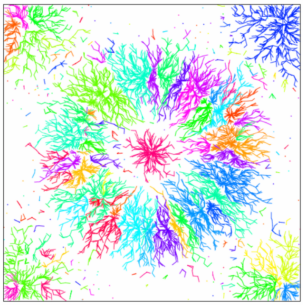
- Very sensitive to noise
- Slight change in neighborhood graph can drastically change clusters.

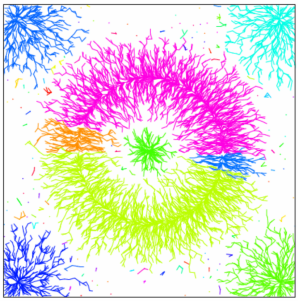
Ex:





Results:





Next time:

How to deal with this...

persistent homology