Algorithms in Bioinformatics

Clustering: Part 1

Recap

- HW1 is graded
- Stay tuned for next HW
- Stay tuned for final exam or project info

Today: Clustering

Today: Clustering

Biological Motivation:

genes + their functions

Not always meaningful to look at sequence similarity.

Approach:

- analyze expressionlevels (amount of mRNA in cell) at different times
- look for patterns: if expression patterns are similar, will suspect these genes have similar or related functions.

Result:

We are given an $n \times m$ expression matrix I:

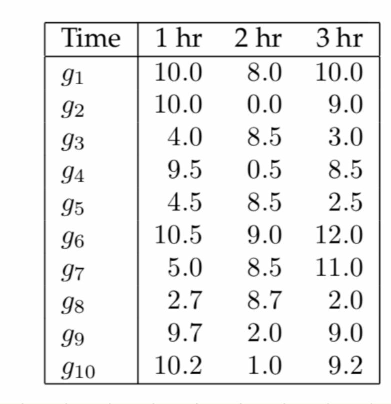
- n rows (one per gene)
- m columns (one per time pt)

The entry at position (i,j) represents the expression level of gene i at timestamp j.

Goal: Find similar rows.

Caution: - data is noisy!
- also, not a guarantee

Example 10 genes, 3 timestamps



Time	1 hr	2 hr	3 hr
g_1	10.0	8.0	10.0
g_2	10.0	0.0	9.0
g_3	4.0	8.5	3.0
g_4	9.5	0.5	8.5
g_5	4.5	8.5	2.5
g_6	10.5	9.0	12.0
g_7	5.0	8.5	11.0
g_8	2.7	8.7	2.0
g_9	9.7	2.0	9.0
g_{10}	10.2	1.0	9.2

How can we interpret this?

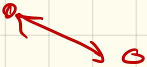
Need a way to compare entire rows.

Back to good old geometry...

We can simply interpret each gene as a point in m-dimensional space.

If we do this, automatically get a distance metric!

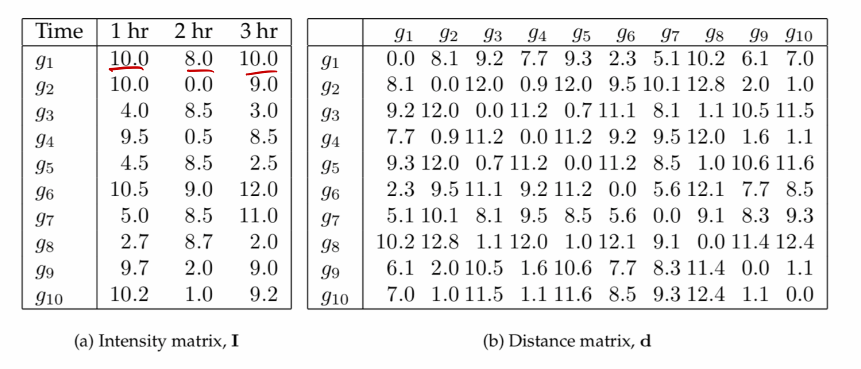
$$d(p_1, p_2) = ?$$



$$d\left((x_1, y_1, z_1), (x_2, y_2, z_2) \right)$$

$$= \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Ex again:



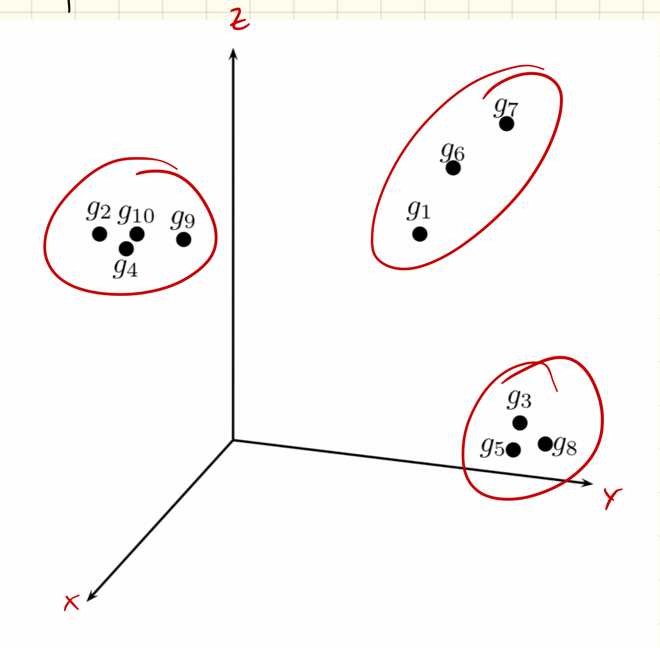
Time	1 hr	2 hr	3 hr
g_1	10.0	8.0	10.0
g_2	10.0	0.0	9.0
g_3	4.0	8.5	3.0
g_4	9.5	0.5	8.5
g_5	4.5	8.5	2.5
g_6	10.5	9.0	12.0
g_7	5.0	8.5	11.0
g_8	2.7	8.7	2.0
g_9	9.7	2.0	9.0
g_{10}	10.2	1.0	9.2

(a) Intensity matrix, **I**

	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

(b) Distance matrix, **d**

Final picture:



(c) Expression patterns as points in three-dimentsional space.

Q: How do you think these should be clustered?

Goal: Make "good" clusters:

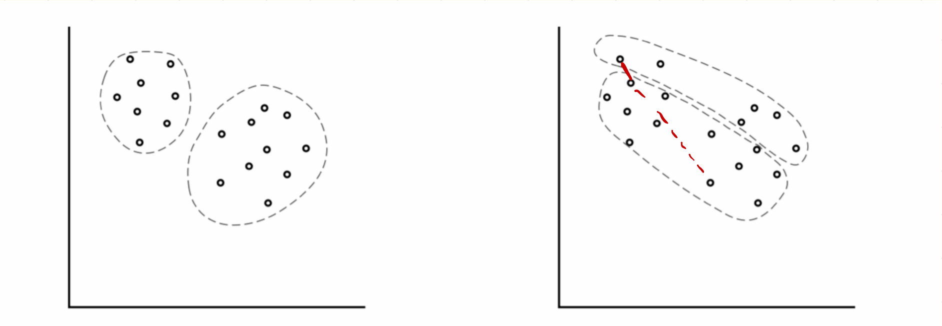
- homogeneous: things within a cluster should be similar

- Separation: genes in different clusters should be different

ie large distance

(We need to make this more precise later...)

Examples:

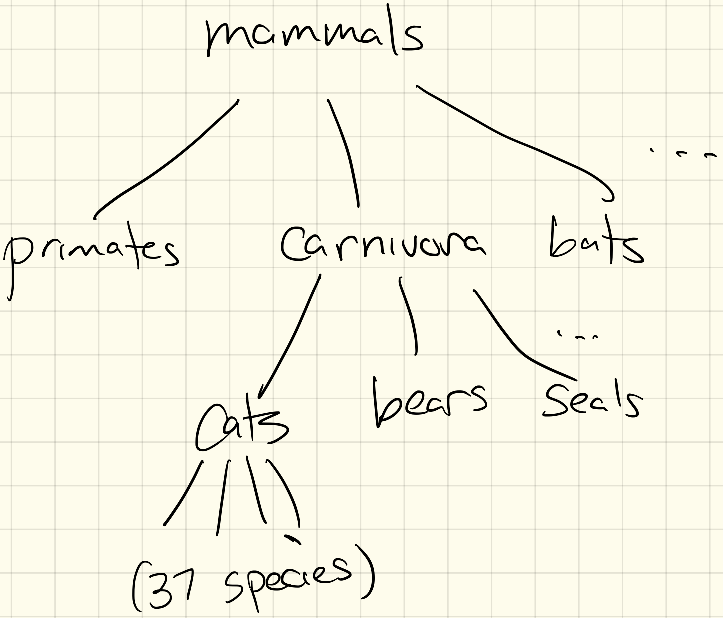


Hierarchical Clustering:

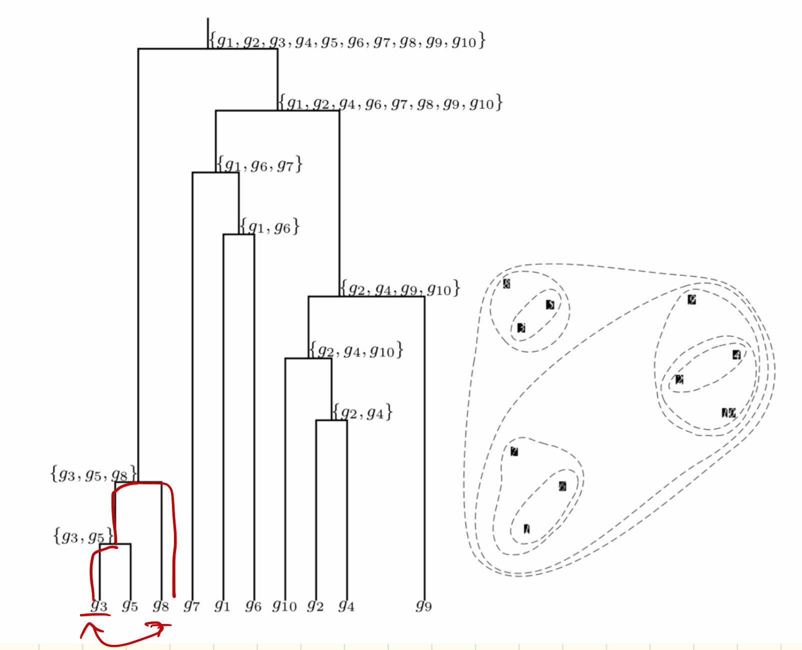
Tree based idea:
(on same data)

Often, we break data into high level clusters, & then break them down further.

Example:



Gene picture:

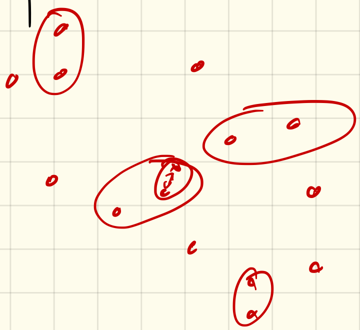

$\{g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8, g_9, g_{10}\}$
$\{g_1, g_2, g_4, g_6, g_7, g_8, g_9, g_{10}\}$
$\{g_1, g_6, g_7\}$
$\{g_1, g_6\}$
$\{g_2, g_4, g_9, g_{10}\}$
$\{g_2, g_4, g_{10}\}$
$\{g_2, g_4\}$
$\{g_3, g_5, g_8\}$
$\{g_3, g_5\}$
g_3 g_5 g_8 g_7 g_1 g_6 g_{10} g_2 g_4 g_9

Here: - genes are leaves
- edges of tree get lengths
- distance in tree somehow encodes distance b/t the genes

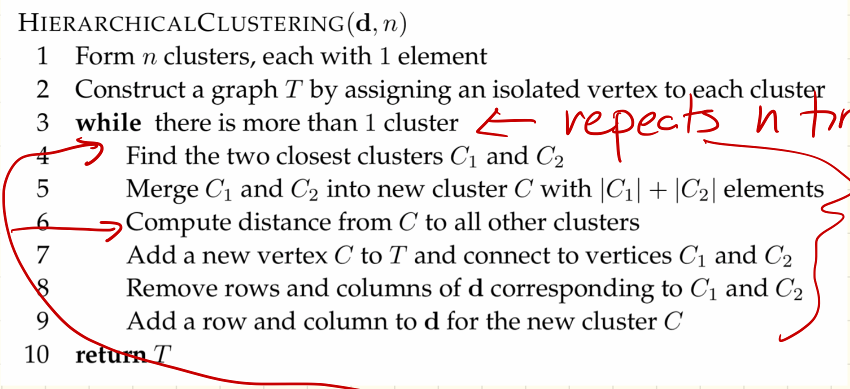
High level idea:

Be greedy!

What's an obvious good pair to cluster?



Algorithm:



```
HIERARCHICALCLUSTERING($\mathbf{d}$, $n$)
    Form $n$ clusters, each with 1 element
    Construct a graph $T$ by assigning an isolated vertex to each cluster
    while there is more than 1 cluster
        Find the two closest clusters $C_1$ and $C_2$
        Merge $C_1$ and $C_2$ into new cluster $C$ with $|C_1| + |C_2|$ elements
        Compute distance from $C$ to all other clusters
        Add a new vertex $C$ to $T$ and connect to vertices $C_1$ and $C_2$
        Remove rows and columns of $\mathbf{d}$ corresponding to $C_1$ and $C_2$
        Add a row and column to $\mathbf{d}$ for the new cluster $C$
    return $T$
```

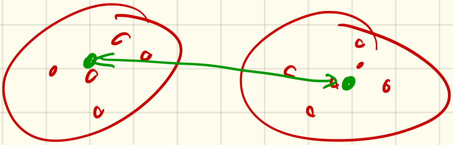
← repeats n times

?!

Ambiguous part:

Many different ways to compute distance.

What's an obvious one?



center: distance from centroid

Another?

- find closest pt in each
- average distance
  - ↳ to center
  - ↳ between all points

Run time:

Depends a bit on distances & data structures

Naive implementation:

$O(n^3)$

Improved (for some data sets):

$O(n^2 \log n)$

Pf of correctness:

It isn't!

No theoretical guarantee (even approximate)

Try all clusters: exponential

Actually, two ways to compute these:

- We saw bottom-up approach
- Also top down: decide on a split

How?

DIANA (Divisive Analysis Clustering):

- Find element with maximum average distance
  ↖ $n^2+n+n = O(n^2)$
- Group all objects with it that are more similar to it than to old cluster
  ↳centroid
- Recurse on each cluster.

(repeat until n clusters)

Note: Again, not optimal!

Also doesn't give same clustering!

Recent work:

- This is still an active area of research:

In CS: trying to prove any approximation guarantee

In Bio: trying to figure out how well it works on various date sets.

(Maybe a good future essay...)

K-means clustering (10.3)

A different variant: fix the number of desired clusters, k.

n = # pts, m = d = dim

Determine a set of k points, or centers, that minimize distance or distortion.

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

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

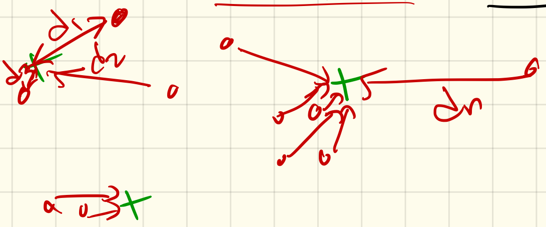
(where $d(v,x_i)$ is Euclidean dist)

Then squared error distortion for a set of points

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

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

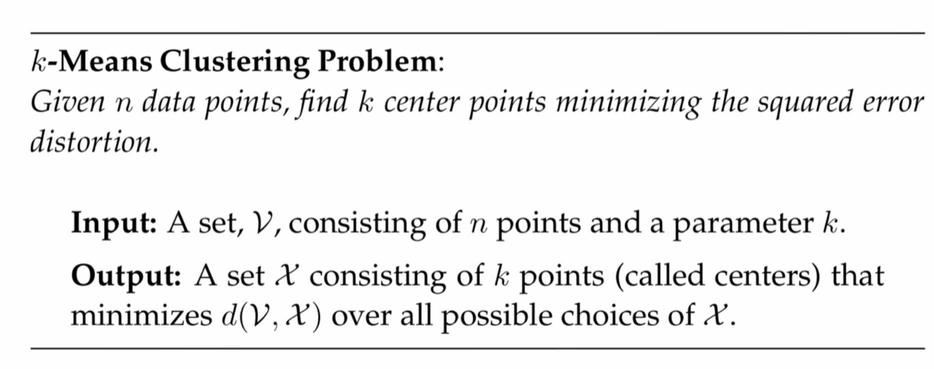
is: $$\underline{d(V,X)} = \frac{\sum_{i=1}^{n} d(v_i,X)^2}{n}$$



k centers

n points

Then:



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 $\mathcal{X}$.

Clusters:

take each point to its closest center

- a cluster is set of closest pts to any center

Unfortunately, it is NP-Hard:

- no real hope of any polynomial time solution

Heuristic approach: Lloyd's algorithm:

- Randomly select an arbitrary partition into k clusters
- Improve iteratively, by moving points between clusters

First:

- elect K points (randomly) as centers, $X = \{x_1, \dots, x_k\}$

Repeat: (until centers don't change)

- assign each point to its nearest center x_i
- compute "center of gravity" for each cluster:

$$\frac{\sum_v v}{|C|}$$

& make it the new center

Runtime: Per iteration,

$$O(\underbrace{n}_{\#\text{ pts}}\, k\, \underbrace{m}_{\text{dim}})$$

No guarantee!

- Of optimality, at least.

However, does tend to converge very quickly.

(Usually, only need about a dozen iterations.)

$\cancel{2^n}$

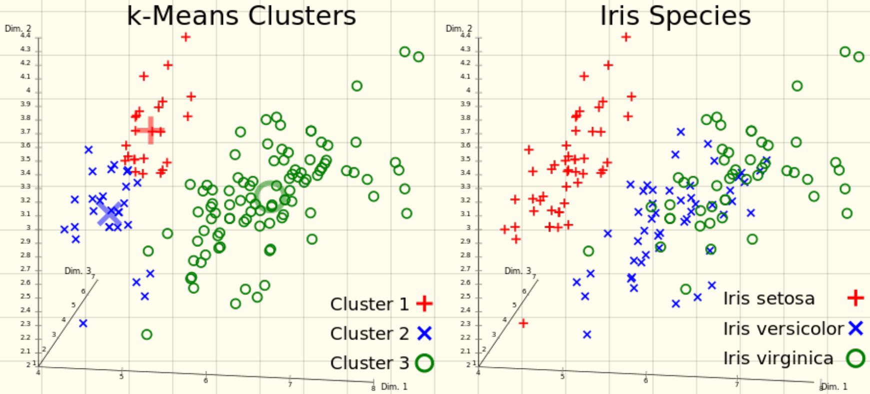
[2006]: # of iterations is $2^{\Omega(\sqrt{n})}$ "super-polynomial"

[2009]: "Perturbed" inputs are polynomial time

$\nwarrow$ n^{85}

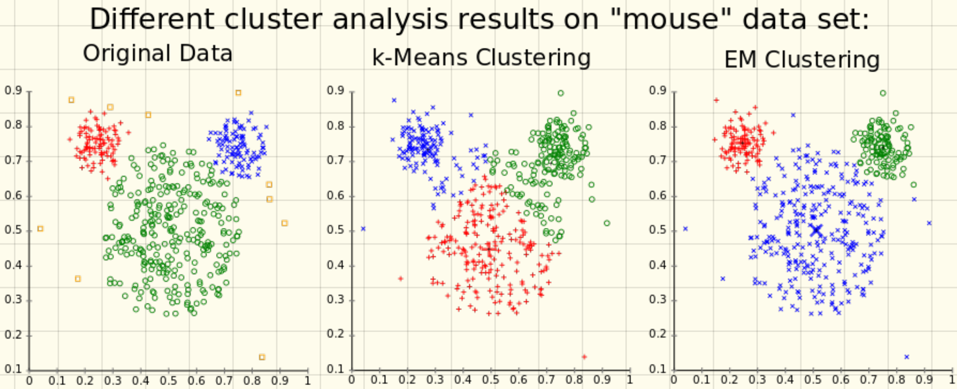
[2009]: Better on "nice" inputs.

Example (& some problems)



↑ Iris flower data set

"Mouse" data set: also issues


Different cluster analysis results on "mouse" data set:
Original Data
k-Means Clustering
EM Clustering