CM122/222 Bioinformatics Algorithms

Discussion 1A
(slides prepared by Yihe Deng)

Quick Reminder

- Due Tue 4/25: HW3 chapter 8
- Due Thur 4/27: Project 1a

```
def TraverseTrie(root):
    node_list = [root]
    while len(node_list) > 0:
        current node = pop up first element of node list
        for edge in current node.edges:
            node_list.append(current node.edge2node[edge])
```

Project 1a

- Sequencing error and mutations
- Pair reads

Today: Textbook Chapter 8

Clustering algorithms

- K-Center Clustering
  - Farthest First Traversal
- K-Means Clustering
  - The Lloyd algorithm
- Hierarchical Clustering

Credit to Luke

Gene expression matrix

Gene	Expression Vector						
YLR361C	0.14	0.03	-0.06	0.07	-0.01	-0.06	-0.01
YMR290C	0.12	-0.23	-0.24	-1.16	-1.40	-2.67	-3.00
YNR065C	-0.10	-0.14	-0.03	-0.06	-0.07	-0.14	-0.04
YGR043C	-0.43	-0.73	-0.06	-0.11	-0.16	3.47	2.64
YLR258W	0.11	0.43	0.45	1.89	2.00	3.32	2.56
YPL012W	0.09	-0.28	-0.15	-1.18	-1.59	-2.96	-3.08
YNL141W	-0.16	-0.04	-0.07	-1.26	-1.20	-2.82	-3.13
YJL028W	-0.28	-0.23	-0.19	-0.19	-0.32	-0.18	-0.18
YKL026C	-0.19	-0.15	0.03	0.27	0.54	3.64	2.74
YPR055W	0.15	0.15	0.17	0.09	0.07	0.09	0.07

Textbook 8.3

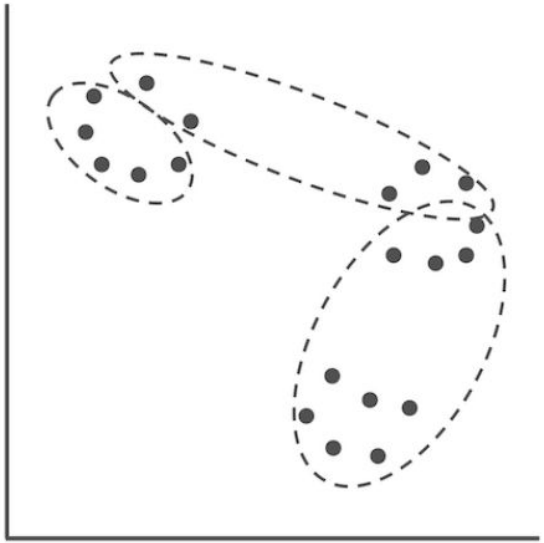
Question: can you identify groups by just looking at this matrix?

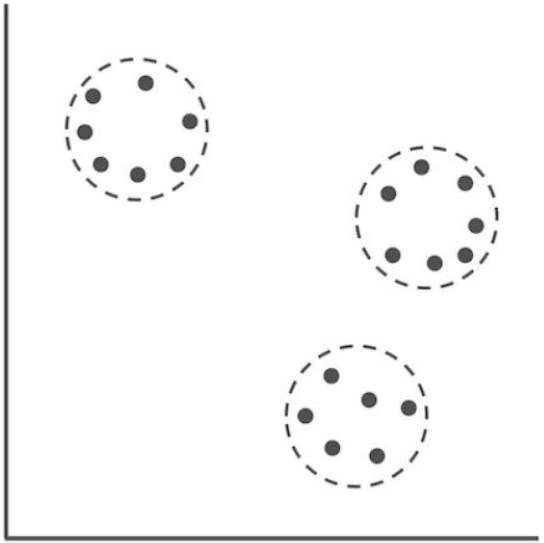
Each row is a gene and each column is an experimental condition (e.g. different time points)

The goal is to find groups of genes with similar behavioral patterns across conditions.

Good Clustering Principle

Every pair of points from the same cluster should be closer to each other than any pair of points from different clusters.





*Images for this slides all from textbook chapter 8

Clustering as an Optimization Problem

Goal: select a set of k points that will serve as centers of the clusters.

- Minimize the distance between centers and data (its cluster) over all possible choices of centers.
- **Euclidean distance** in m-dimensional space
  - Given vector v = (v_1, …, v_m) and w = (w_1, …, w_m)

$$d(v,w) = \sqrt{\sum_{i=1}^{m}(v_i - w_i)^2}.$$

Today: Textbook Chapter 8

Clustering algorithms

- K-Center Clustering
  - Farthest First Traversal
- K-Means Clustering
  - The Lloyd algorithm
- Hierarchical Clustering

K-Center Clustering

Given a set of centers and a data point, it belongs to the cluster of the **closest center**.

$d(DataPoint,Centers) = \min_{\text{all points } x \text{ from } Centers} d(DataPoint, x)$.

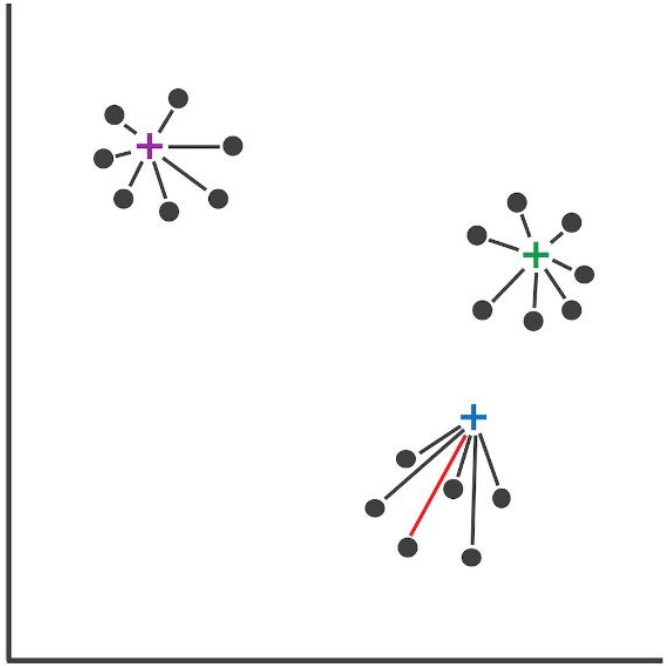
The distance between all data points Data and centers Centers is then defined as

- the **max distance** between a cluster point and its center.

$MaxDistance(Data, Centers) = \max_{\text{all points } DataPoint \text{ from } Data} d(DataPoints,Centers)$.

as shown in the red line.

K-Center Clustering: minimizes this max distance.



Today: Textbook Chapter 8

Clustering algorithms

- K-Center Clustering
  - Farthest First Traversal
- K-Means Clustering
  - The Lloyd algorithm
- Hierarchical Clustering

Heuristic Algorithm: Farthest First Traversal

Input: Data points, cluster number k

Output: Centers

Centers ← the set consisting of a single randomly chosen point from Data

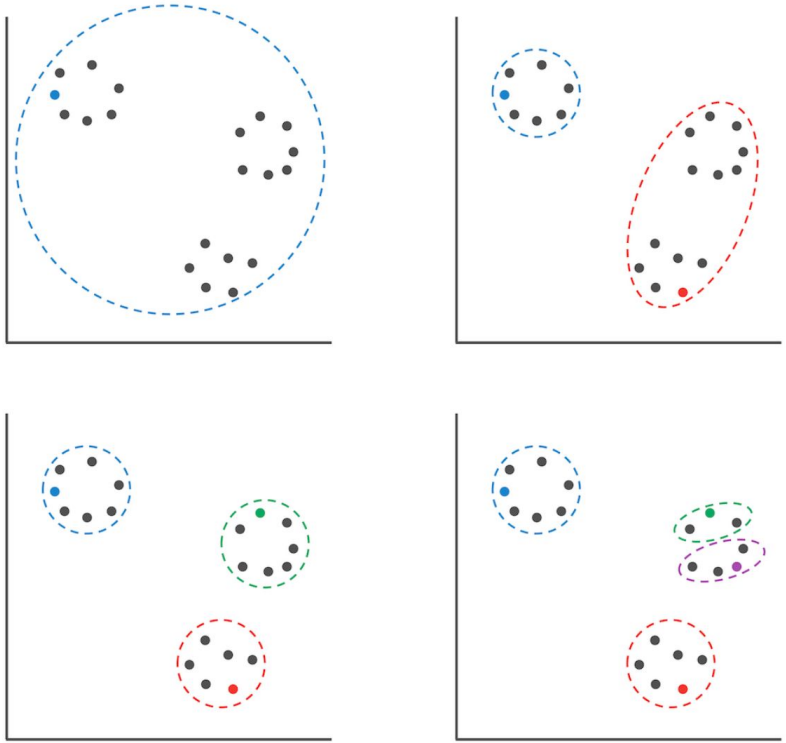
while |Centers| < k

    DataPoint ← the point in Data *maximizing* d(DataPoint, Centers)

add DataPoint to Centers

return Centers

Note. d(DataPoint, Centers) is the *minimum* distance between the data point and all centers.

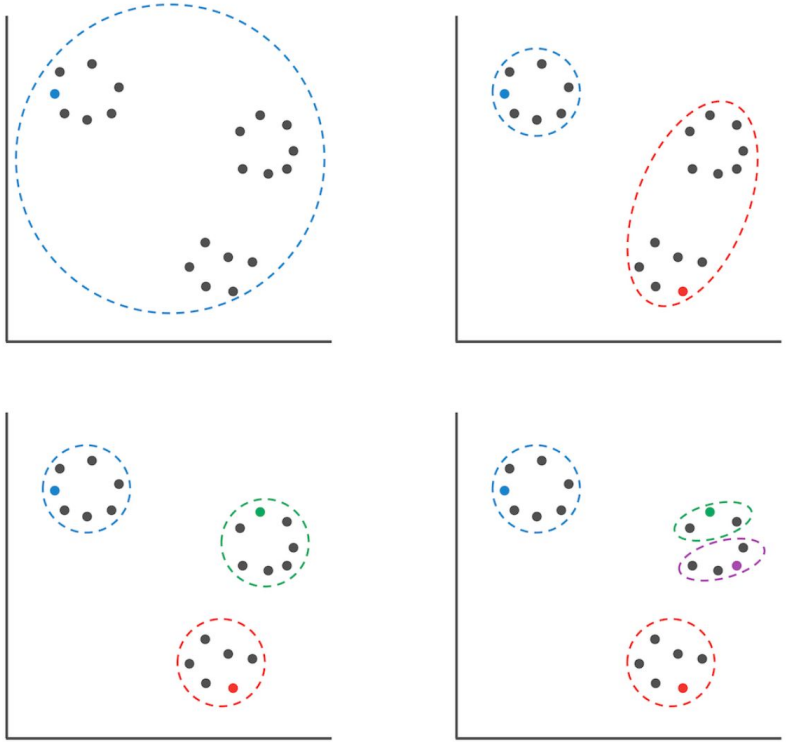


Heuristic Algorithm: Farthest First Traversal

Input: Data points, cluster number k

Output: Centers

Briefly, while we haven't reached desired number of centers, we keep adding center by finding the data point that has the largest distance to its current assigned center.



Heuristic Algorithm: Farthest First Traversal

HW3 8.6 Q2

- Input: Integers k and m; data points in m-dimensional space.
- Output: k centers.

Note: the first point from Data is chosen as the first center to initialize the algorithm.

Sample Input:

```
3 2
0.0 0.0
5.0 5.0
0.0 5.0
1.0 1.0
2.0 2.0
3.0 3.0
1.0 2.0
```

Sample Output:

```
0.0 0.0
5.0 5.0
0.0 5.0
```

```
Def farthestFirstTraversa(k, points):
    Centers = [points[0]]
    While len(centers) < k:
        Max_dist = 0
        New_center = None
        For point in points:
            Min_dist = inf
            For center in centers:
                If distance(point, center) < min_dist:
                    Min_dist = distance(point, center)
            If min_dist > max_dist:
                New_center = point
        centers.append(new_center)
```

Code Example

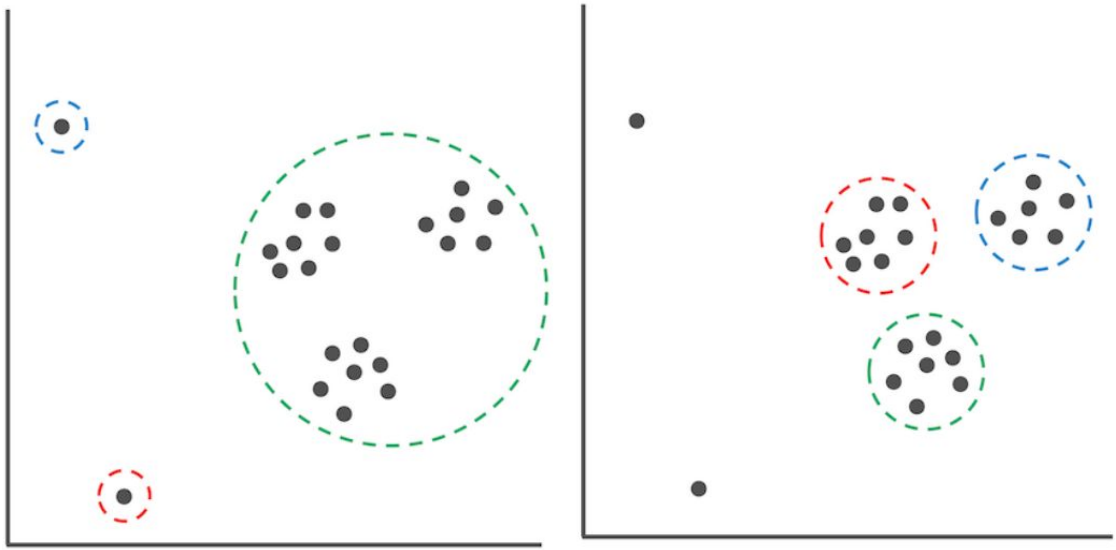
```
def distance(point1, point2):
    """Compute the Euclidean distance between two points."""

def farthest_first_traversal(Data, k):
    """Return a set of k points (centers) resulting from applying the FarthestFirstTraversal
algorithm."""
    Centers = [Data[0]] # initialize with the first point in Data
    while len(Centers) < k:
        # while we haven't found k centers, ...
        for point in Data:
            if point in Centers:
                continue
            # find the farthest point to the current Centers
        Centers.append(farthest_point)
    return Centers
```

Heuristic Algorithm: Farthest First Traversal

Advantages: fast; the solution approximates the optimal solution of the k-Center Clustering Problem.

Disadvantages: is prune to the influence of outliers representing experimental errors.

→ Due to max distance



Today: Textbook Chapter 8

Clustering algorithms

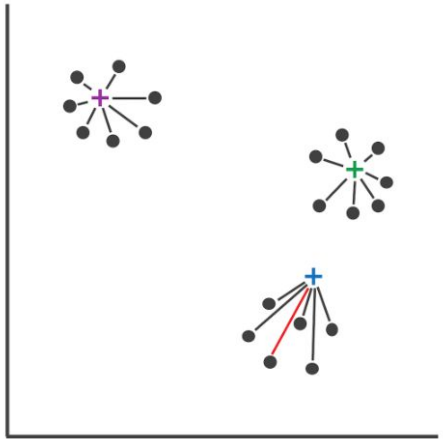
- K-Center Clustering
  - Farthest First Traversal
- K-Means Clustering
  - The Lloyd algorithm
- Hierarchical Clustering

Squared Error Distortion

Instead of max distance, let's define a new metric for finding centers.

- **Squared error distortion**: the mean squared distance from each data point to its nearest center

$$Distortion(Data, Centers) = (1/n) \sum_{all\ points\ DataPoint\ in\ Data} d(DataPoint, Centers)^2 .$$



Max distance: account for the red line

Distortion: account for all lines.

Squared Error Distortion

HW3 8.7 Q3

- Input: A set of centers and a set of points.
- Output: The squared error distortion.

Sample Input:

```
2 2
2.31 4.55
5.96 9.08
--------
3.42 6.03
6.23 8.25
4.76 1.64
4.47 4.33
3.95 7.61
8.93 2.97
9.74 4.03
1.73 1.28
9.72 5.01
7.27 3.77
```

Sample Output:

```
18.246
```

Code Example

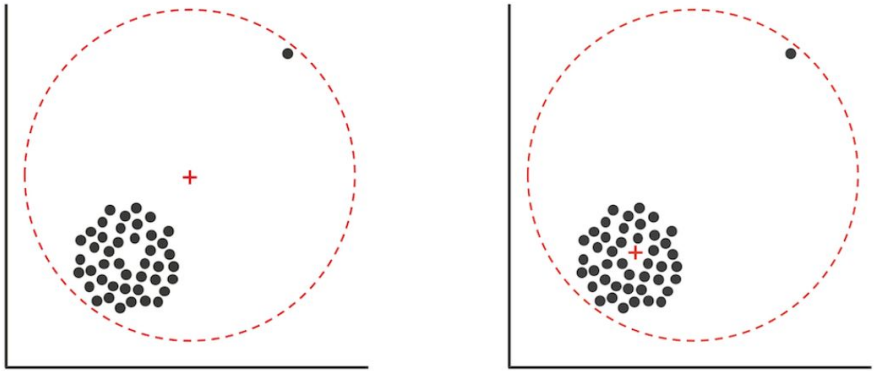
```
def euclidean_distance(point1, point2):
    """Computes the Euclidean distance between two points represented as tuples."""

def distortion(data, centers):
    """Computes the squared error distortion of the given data points and centers."""
    total_distortion = 0
    for point in data:
        # Find the nearest center to the point
        nearest_center = ...
        # Add the squared distance between the point and the nearest center to the total
distortion
        total_distortion += euclidean_distance(point, nearest_center) ** 2
    # Divide the total distortion by the number of data points to get the average distortion
    return total_distortion / len(data)
```

K-Means Clustering

Goal: given a set of data points, find k center points minimizing the **squared error distortion**.

- As compared to K-Center Clustering (Farthest First Traversal), which minimizes the **maximum distance** between the center and any point in the cluster.

Result: the position of the center is less influenced by outliers.



The Center of Gravity

How to find the centers given a cluster?

- When k = 1, the k-Means Clustering Problem amounts to finding a single center point x that minimizes the squared error distortion.

Center of gravity: the point whose i-th coordinate is the average of the i-th coordinates of all considered points.

- Example: for the points (3, 8), (8, 0), and (7, 4), the center of gravity is

$$\left(\frac{3+8+7}{3}, \frac{8+0+4}{3}\right) = (6, 4).$$

Today: Textbook Chapter 8

Clustering algorithms

- K-Center Clustering
  - Farthest First Traversal
- K-Means Clustering
  - The Lloyd algorithm
- Hierarchical Clustering

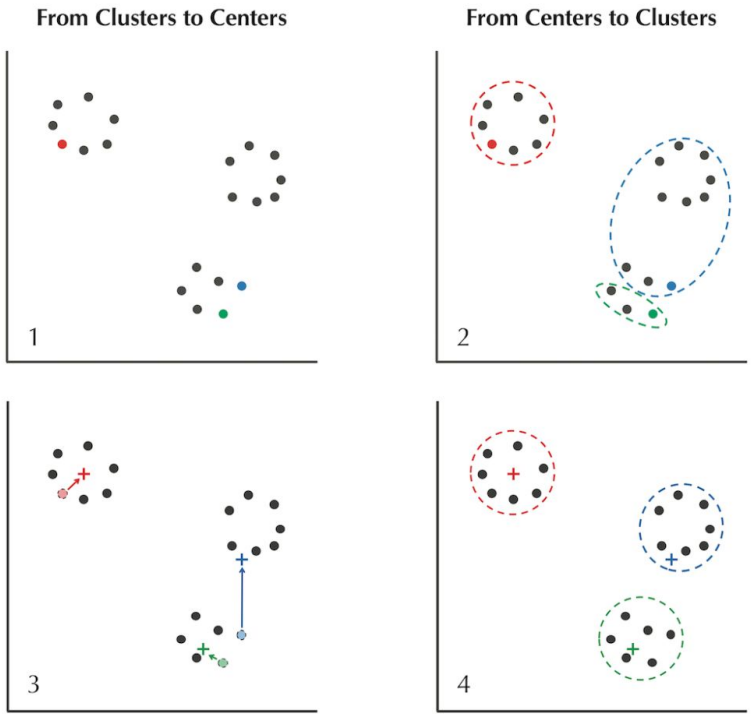
The Lloyd Algorithm

Clustering heuristics for the k-Means Clustering Problem.

First chooses k arbitrary distinct points, then iteratively perform the following:

- **Centers to Clusters**: Given selected centers, assign each data point to the cluster corresponding to its nearest center; ties are broken arbitrarily.
- **Clusters to Centers**: After data points have been assigned to clusters, assign each cluster center of gravity to be the cluster's new center.

The algorithm has converged if the centers stop changing between iterations.



The Lloyd Algorithm

HW3 8.8 Q3

- Input: Integers k and m; a set of points in m-dimensional space.
- Output: A set consisting of k points (centers) resulting from the Lloyd algorithm.

Note: the first k points from Data are selected as the first k centers.

Sample Input:

```
2 2
1.3 1.1
1.3 0.2
0.6 2.8
3.0 3.2
1.2 0.7
1.4 1.6
1.2 1.0
1.2 1.1
0.6 1.5
1.8 2.6
1.2 1.3
1.2 1.0
0.0 1.9
```

Sample Output:

```
1.800 2.867
1.060 1.140
```

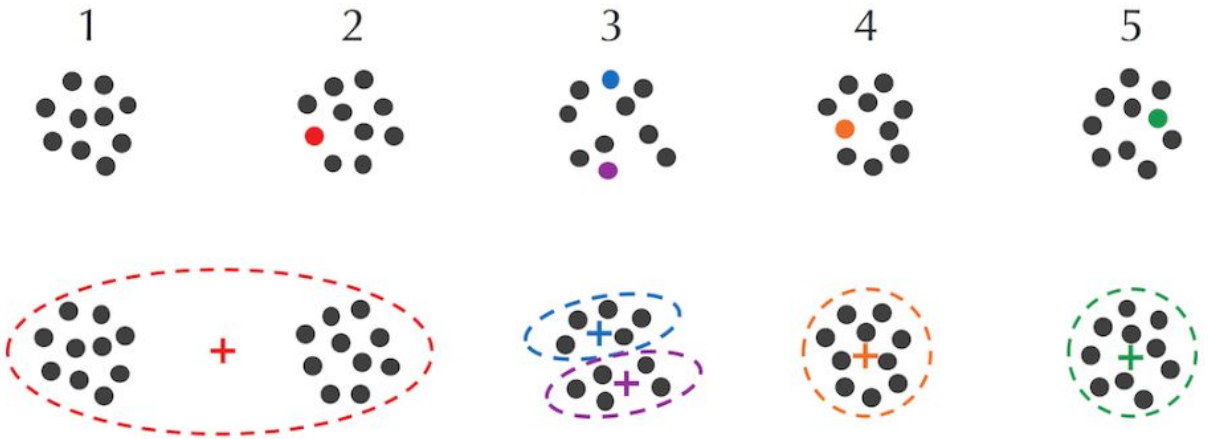
Code Example

```
def lloyd_algorithm(k, m, data):
    # Select the first k points from data as the initial centers
    centers = data[:k]

    while True:
        # Assign each point to the closest center
        labels = ...

        # Update the centers to be the means of the assigned points
        centers_new = ...

        # If the centers haven't changed, break out of the loop
        if np.allclose(centers, centers_new):
            break

        centers = centers_new
    return centers
```

The Lloyd Algorithm

The Lloyd algorithm does not always converge to the optimal solution.

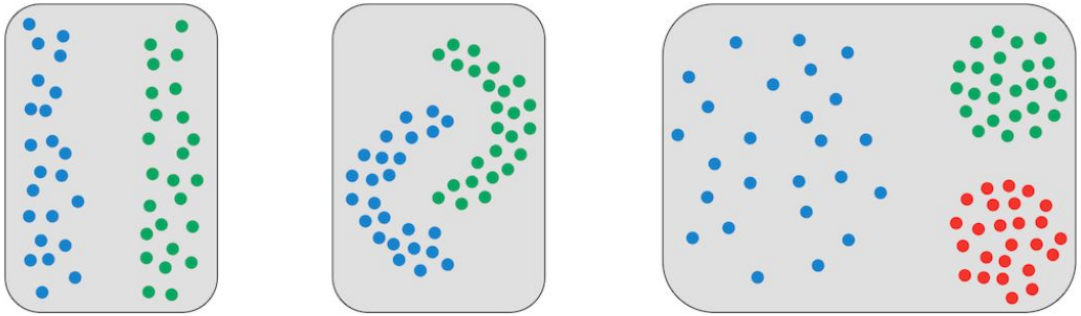
- Different initialization gives different results.



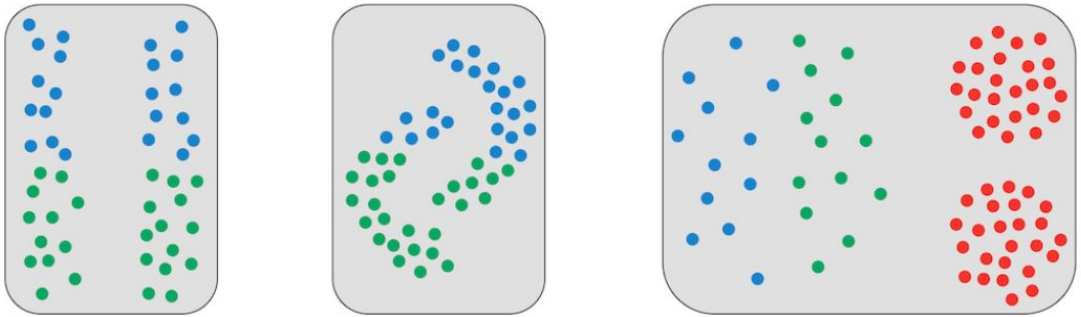
Limitations of K-Means Clustering

In the case of challenging clustering problems, the Lloyd algorithm sometimes fails to identify what may seem like obvious clusters:

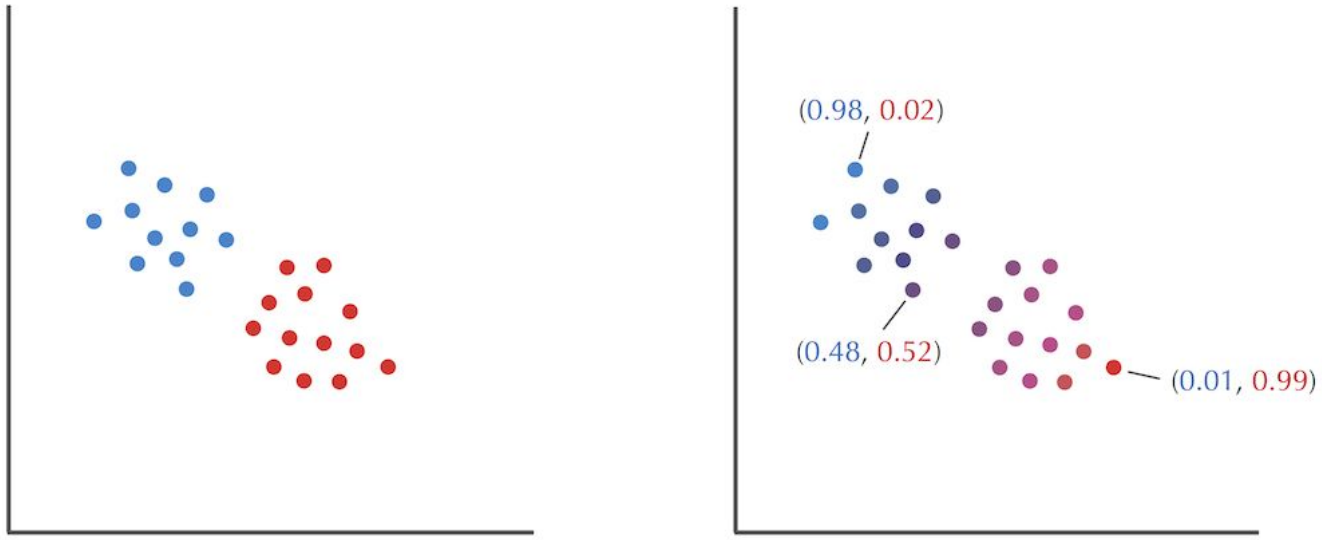
Human eye:



Lloyd algorithm:



Soft clustering



Textbook 8.10

Sometimes you want soft assignments (probability of each point belonging to each cluster)

Credit to Luke

Today: Textbook Chapter 8

Clustering algorithms

- K-Center Clustering
  - Farthest First Traversal
- K-Means Clustering
  - The Lloyd algorithm
- Hierarchical Clustering

Hierarchical Clustering

Distance matrix: each entry at (i,j) indicates the distance between the expression vectors for genes i and j.

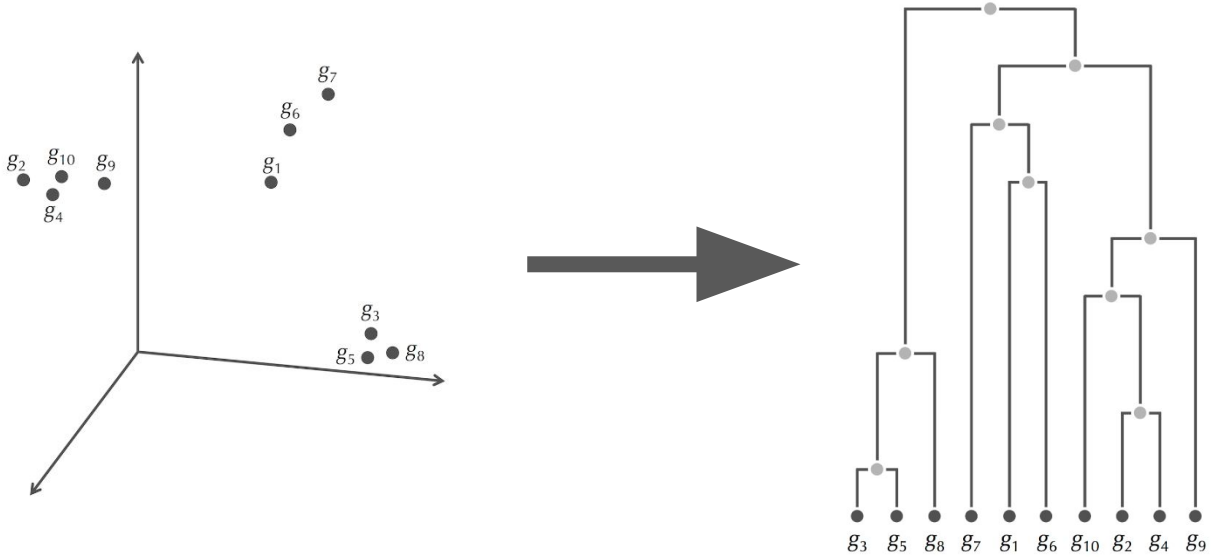
- Symmetric
- The diagonal will be 0

	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

Hierarchical Clustering

Here, we do not assume a fixed value k for the number of clusters.

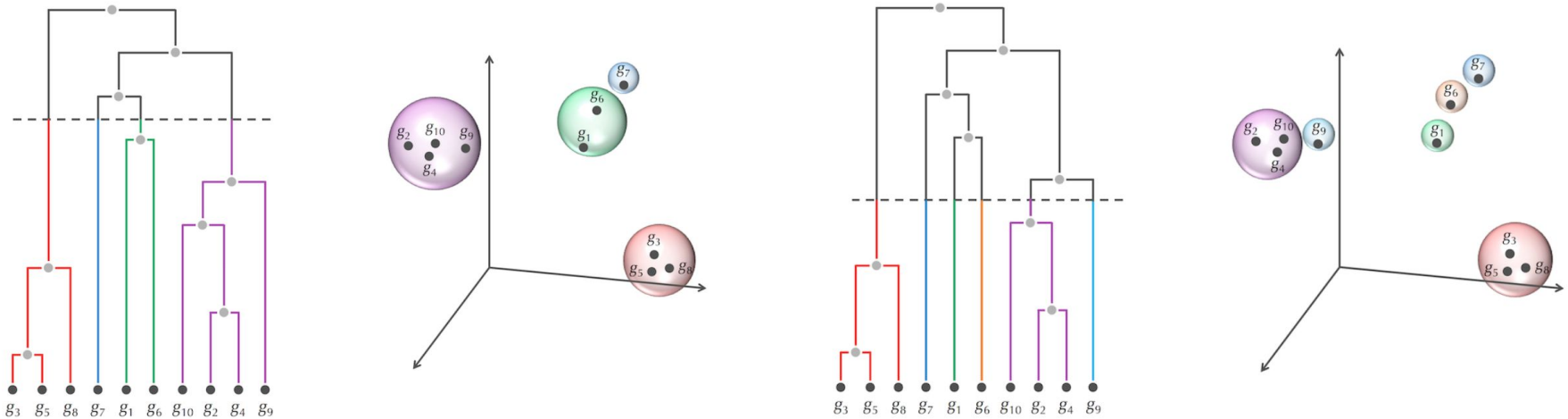
Hierarchical clustering algorithm: uses an n × n distance matrix D to organize the n data points into a tree.



Hierarchical Clustering

How to use the tree?

- A horizontal line crossing the tree in i places divides the n genes into i clusters.



Hierarchical Clustering

Input: Distance matrix D; number of data n.

Output: The graph/tree T.

Briefly, the tree construction is done in a bottom-up manner.

- Start from n clusters where each cluster contain 1 data point -> leaf nodes.
- Merge the clusters to construct the parent node.

```
Clusters ← n single-element clusters labeled 1, ..., n
        construct a graph T with n isolated nodes
while there is more than one cluster
        find the two closest clusters Ci and Cj
        merge Ci and Cj into a new cluster C_new
        add a new node labeled by cluster C_new to T
        connect node C_new to Ci and Cj by directed edges
        remove the rows and columns of D corresponding to Ci
and Cj
        remove Ci and Cj from Clusters
        for each C in Clusters
                compute D(C_new, C)
                add a row/column to D for C_new
        add C_new to Clusters
Assign root in T as a node with no incoming edges
return T
```

```
Def hierarchicalClustering(n):
    Clusters = [(1), (2), …., (10)]
    Dist_mat = n
    While len(clusters) > 1:
        Row, col = mindist(dist_mat)
        New_cluster = merge(clusters[row], clusters[col])
        clusters.remove(clusters[row])
        clusters.remove(clusters[col])
        Dist_mat = compute_mat(clusters, n)
```

```
Def compute_mat(clusters, n):
    New_mat <- shape of (len(clusters), len(clusters):
    For i from 0, len(clusters): ← (0, 1, 2)
        For j from i+1, len(clusters): ← ( 4, 5)
            Total_dist = 0
            For p1 in clusters[i]: 0
                For p2 in clusters[j]: 4
                    Total_dist += n[p1, p2]
            New_mat[i, j] = total_dist / total num of combination
    Return new_mat
```

```
Def min_dist(mat):
    Min_dist = inf
    Min_row, min_col = None, None
    Row, col = mat.shape
    For row_id from 0 to row:
        For col_id from 0 to col:
            If mat[row_id, col_id] < min_dist:
                Min_dist = mat[row_id, col_id]
                Min_row = row_id
                Mind_col = col_id
    Return min_row, min_col
```

Hierarchical Clustering

Distance metric D(C1,C2):

- The smallest distance between any pair of elements from C1 and C2.

$$D_{\text{min}}(C_1,C_2) = \min_{\text{all points } i \text{ in cluster } C_1,\ \text{all points } j \text{ in cluster } C_2} D_{i,j} \ .$$

- The average distance between elements in C1 and C2.

$$D_{\text{avg}}(C_1, C_2) = \frac{\sum_{\text{all points } i \text{ in cluster } C_1} \sum_{\text{all points } j \text{ in cluster } C_2} D_{i,j}}{|C_1| \cdot |C_2|}$$

Hierarchical Clustering

HW3 8.14 Q7

- Input: An integer n, followed by an n x n distance matrix.
- Output: The result of applying Hierarchical Clustering to D, with each newly created cluster listed on each line.

Note: we use the average distance **D_avg** as the distance metric.

Sample Input:

```
7
0.00 0.74 0.85 0.54 0.83 0.92 0.89
0.74 0.00 1.59 1.35 1.20 1.48 1.55
0.85 1.59 0.00 0.63 1.13 0.69 0.73
0.54 1.35 0.63 0.00 0.66 0.43 0.88
0.83 1.20 1.13 0.66 0.00 0.72 0.55
0.92 1.48 0.69 0.43 0.72 0.00 0.80
0.89 1.55 0.73 0.88 0.55 0.80 0.00
```

Sample Output:

```
4 6
5 7
3 4 6
1 2
5 7 3 4 6
1 2 5 7 3 4 6
```

Questions?

Have a great weekend!