# Hierarchical clustering

School of Electrical and Computer Engineering

University of Tehran

Erfan Darzi

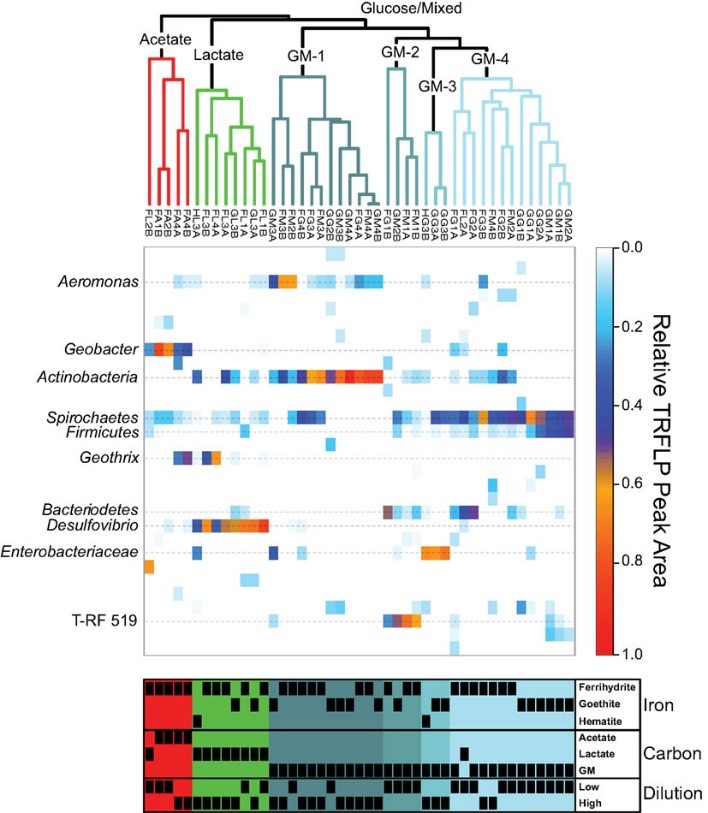
erfandarzi@ut.ac.ir

Today’s lecture

* Hierarchical clustering algorithm
  + Bottom-up: agglomerative
  + Distance between clusters
  + Complexity analysis

Hierarchical clustering

* Build a tree-based hierarchical taxonomy from a set of instances
  + Dendrogram – a useful tool to summarize



similarities

*After cutting, each*

*connected component*

*will be a cluster*

## Agglomerative hierarchical clustering

• Pairwise distance metric between instances



0

1

2

2

3

3

4

0

2

2

3

3

4

0

2

3

3

4

0

3

4

4

0

2

3

0

2

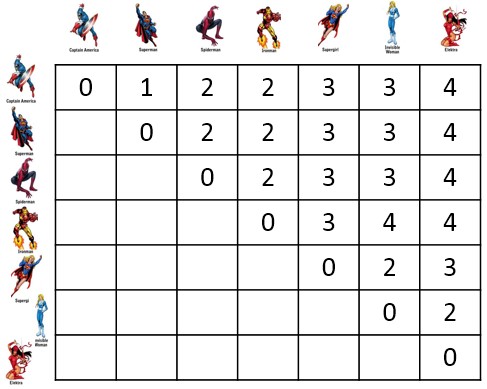
0



## Agglomerative hierarchical clustering

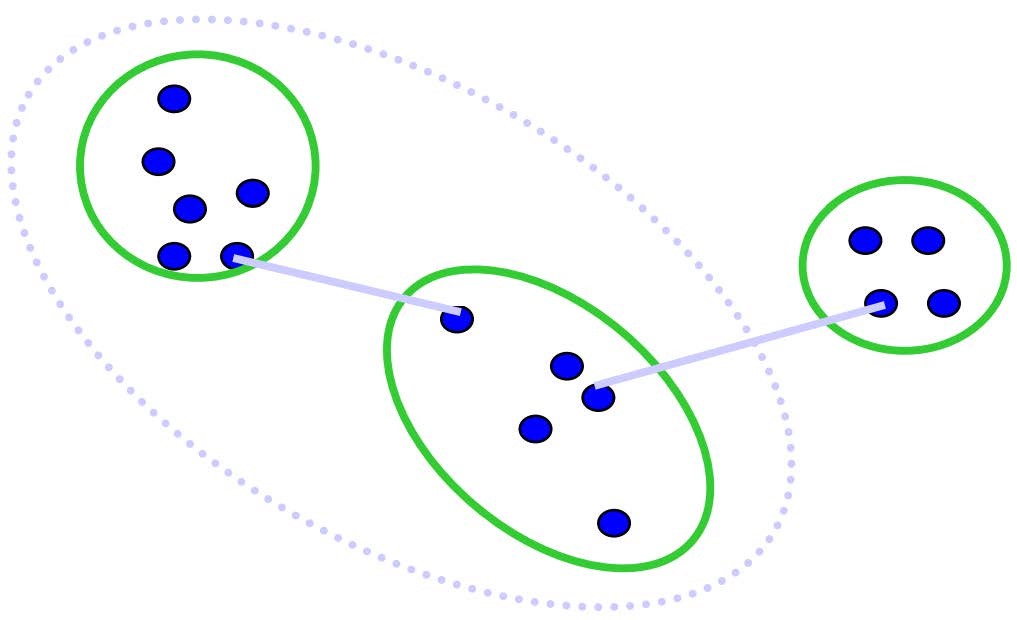
1. Every instance is in its own cluster when initialized
2. Repeat until one cluster left ***Enumerate all the possibilities!***

1. Find the best pair of clusters to merge and break the tie arbitrarily

How to compare distance between an ***?*** instance and a cluster of instances? 



* Single link – Cluster distance = distance of two closest members between the clusters



–

𝑑𝑑

𝑐𝑐

𝑖𝑖

,

𝑐𝑐

𝑗𝑗

=

min

𝑥𝑥

𝑛𝑛

∈𝑐𝑐

𝑖𝑖

,

𝑥𝑥

𝑚𝑚

∈𝑐𝑐

𝑗𝑗

𝑑𝑑

(

𝑥𝑥

𝑛𝑛

,

𝑥𝑥

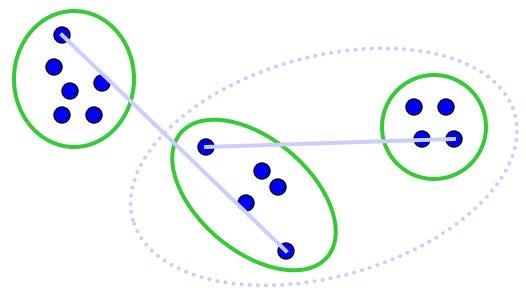
𝑚𝑚

)

Tend to generate

scattered clusters

* Complete link
  + Cluster distance = distance of two farthest members between the clusters



–

𝑑𝑑

𝑐𝑐

𝑖𝑖

,

𝑐𝑐

𝑗𝑗

=

max

𝑥𝑥

𝑛𝑛

∈𝑐𝑐

𝑖𝑖

,

𝑥𝑥

𝑚𝑚

∈𝑐𝑐

𝑗𝑗

𝑑𝑑

(

𝑥𝑥

𝑛𝑛

,

𝑥𝑥

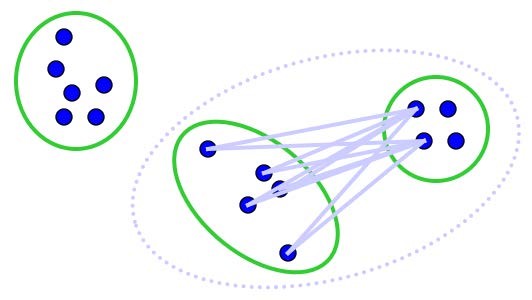
𝑚𝑚

)

Tend to generate

tight clusters

* Average link
  + Cluster distance = average distance of all pairs of members between the clusters



–

𝑑𝑑

𝑐𝑐

𝑖𝑖

,

𝑐𝑐

𝑗𝑗

=

∑

𝑥𝑥

𝑛𝑛

∈𝑐𝑐

𝑖𝑖

,

𝑥𝑥

𝑚𝑚

∈𝑐𝑐

𝑗𝑗

𝑑𝑑

(

𝑥𝑥

𝑛𝑛

,

𝑥𝑥

𝑚𝑚

)

𝑐𝑐

𝑖𝑖

×|

𝑐𝑐

𝑗𝑗

|

Mostly popularly used

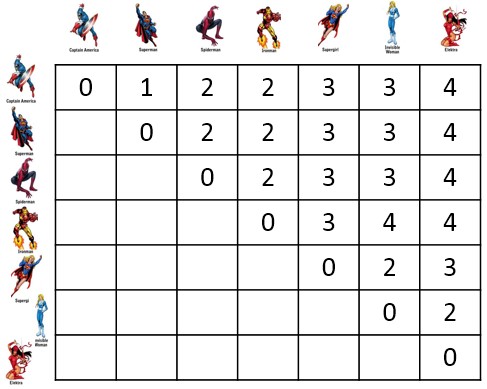
measure, robust

against noise

## Agglomerative hierarchical clustering

1. Every instance is in its own cluster when initialized
2. Repeat until one cluster left

1. Find the best pair of clusters to merge and break the tie arbitrarily



# Complexity analysis

* In step one, compute similarity between all pairs of 𝑛𝑛 individual instances - 𝑂𝑂(𝑛𝑛2)
* In the following 𝑛𝑛−2 steps

– It could be 𝑂𝑂(𝑛𝑛2 log 𝑛𝑛) or even 𝑂𝑂(𝑛𝑛3) (naïve implementation)

In *k*-means, we have 𝑂𝑂(𝑘𝑘𝑛𝑛𝑘𝑘),

a much faster algorithm

# Comparisons

* Hierarchical clustering
  + Efficiency: 𝑂𝑂(𝑛𝑛3), slow
* Assumptions – No assumption – Only need distance metric
* Output
  + Dendrogram, a tree
* *k*-means clustering
  + Efficiency: 𝑂𝑂(𝑘𝑘𝑛𝑛𝑘𝑘), fast
* Assumptions
  + Strong assumption –

centroid, latent cluster

membership

* + Need to specify 𝑘𝑘
* Output
  + 𝑘𝑘 clusters

How to get final clusters?

* If 𝑘𝑘 is specified, find a cut that generates 𝑘𝑘 clusters
  + Since every time we only merge 2 clusters, such cut must exist
* If 𝑘𝑘 is not specified, use the same strategy as in *k*-means – Cross validation with internal or external validation

What you should know

* Agglomerative hierarchical clustering
  + Three types of linkage function • Single link, complete link and average link
  + Comparison with *k*-means