lorenzo.dallamico@unito.it

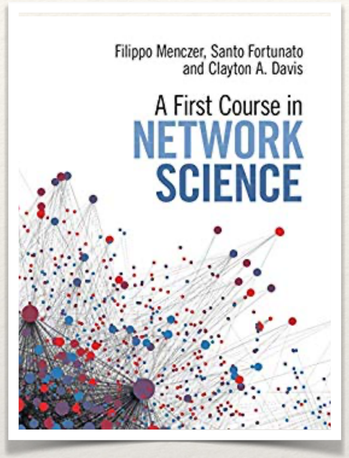
di.unito.it

Lecture 19.ns13

Communities 2

Course: **Complex Networks Analysis and Visualization**
Sub-Module: **NetSci**

References

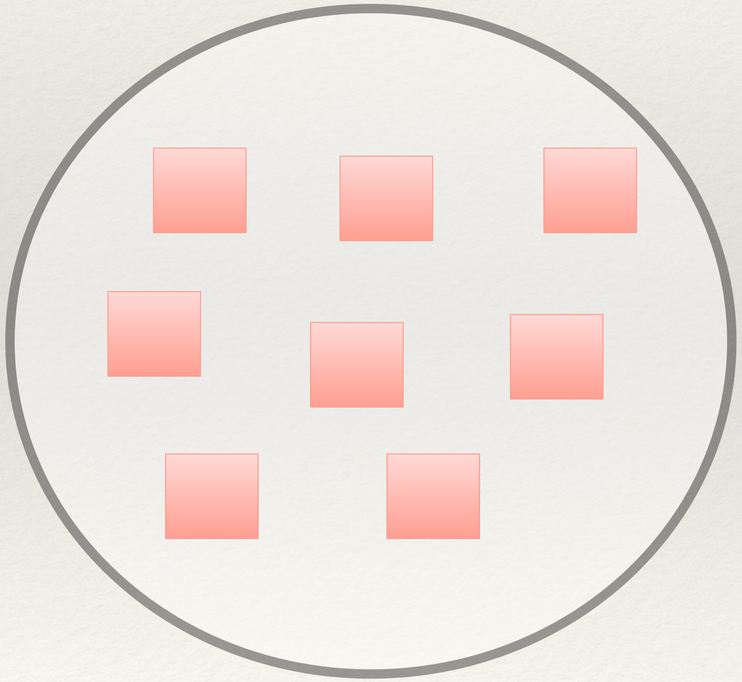


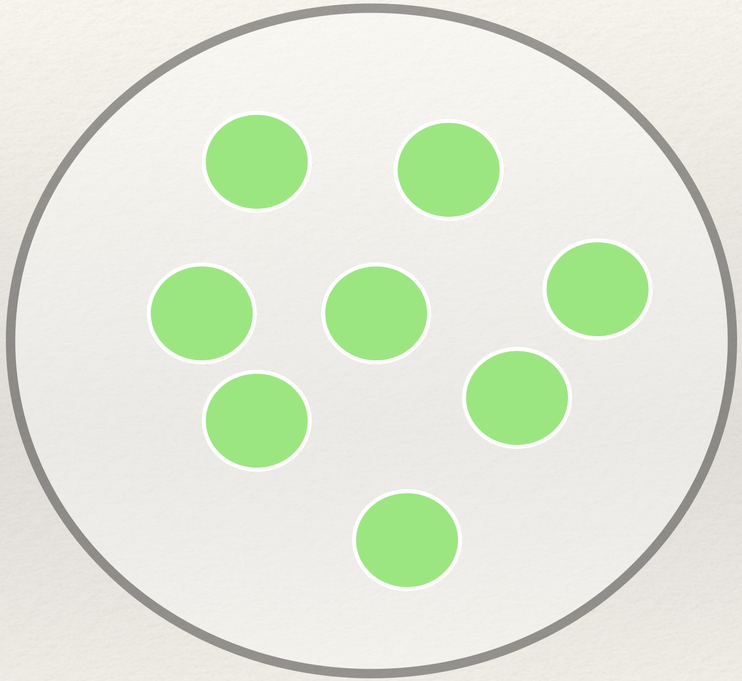
Chapter 6: Communities

A Tutorial on Spectral Clustering: https://arxiv.org/pdf/0711.0189.pdf

Community detection and clustering

Community detection consists in dividing the nodes of a graph into "affinity" groups. This falls more in general into the domain of data clustering.





Community detection and clustering

Which is the most similar one?

Community detection and clustering

Which is the most similar one?

distance

distance

Defining communities through distance

We may define a distance between the nodes of a graph in order to cluster them into similarity groups. We list three possibilities

1) I attach feature properties to the nodes (e.g. age, height, mother-tongue, interests...) and define a node similarity measure based on these properties. Interesting, but we need the metadata and we do not need the graph

2) I define a structural distance between nodes depending only on graph properties

3) I create a feature vector for each node (embedding) only based on structural properties and use it to define a distance.

Hierarchical clustering

Hierarchical clustering

- Hierarchical clustering delivers a nested set of partitions
- Main ingredient: **similarity measure**
- Examples:
  - In a social network it could indicate how close the profiles of two people are based on their interests
  - If nodes are embedded in space (i.e., they are points in a metric space), the (dis)similarity between two nodes can be expressed by their distance
  - If nodes are not embedded in space, similarity measures can be derived from the network structure

Similarity: structural equivalence

- **Concept:** nodes are similar if their neighbors are similar

Neighborhood overlap

$$S_{ij}^{SE} = \frac{\text{number of neighbors shared by } i \text{ and } j}{\text{total number of nodes neighboring only } i, \text{ only } j, \text{ or both}}$$

- **Examples:**
  - If the neighbors of i and j are (v_1, v_2, v_3) and (v_1, v_2, v_4, v_5), respectively, $S_{ij} = 2/5 = 0.4$, because there are two common neighbors (v_1 and v_2) out of five distinct neighbors in total (v_1, v_2, v_3, v_4, v_5)
  - If i and j have no neighbors in common, $S_{ij} = 0$ if (ij) is an edge, then it is a local bridge
  - If i and j have the same neighbors, $S_{ij} = 1$

Similarity of node groups

- **Question:** how can we define the similarity $S_{G_1 G_2}$ between two groups of nodes G_1 and G_2 via the similarity S between pairs of nodes?
- **Answer:** multiple approaches. The first step is to compute the pairwise similarity S_{ij} between each node i in G_1 and each node j in G_2. Then the following strategies can be adopted:
  - **Single linkage:** take the maximum pairwise similarity —> $S_{G_1 G_2} = \max_{i,j} S_{ij}$
  - **Complete linkage:** take the minimum pairwise similarity —> $S_{G_1 G_2} = \min_{i,j} S_{ij}$
  - **Average linkage:** take the average pairwise similarity —> $S_{G_1 G_2} = \langle S_{ij} \rangle_{i,j}$

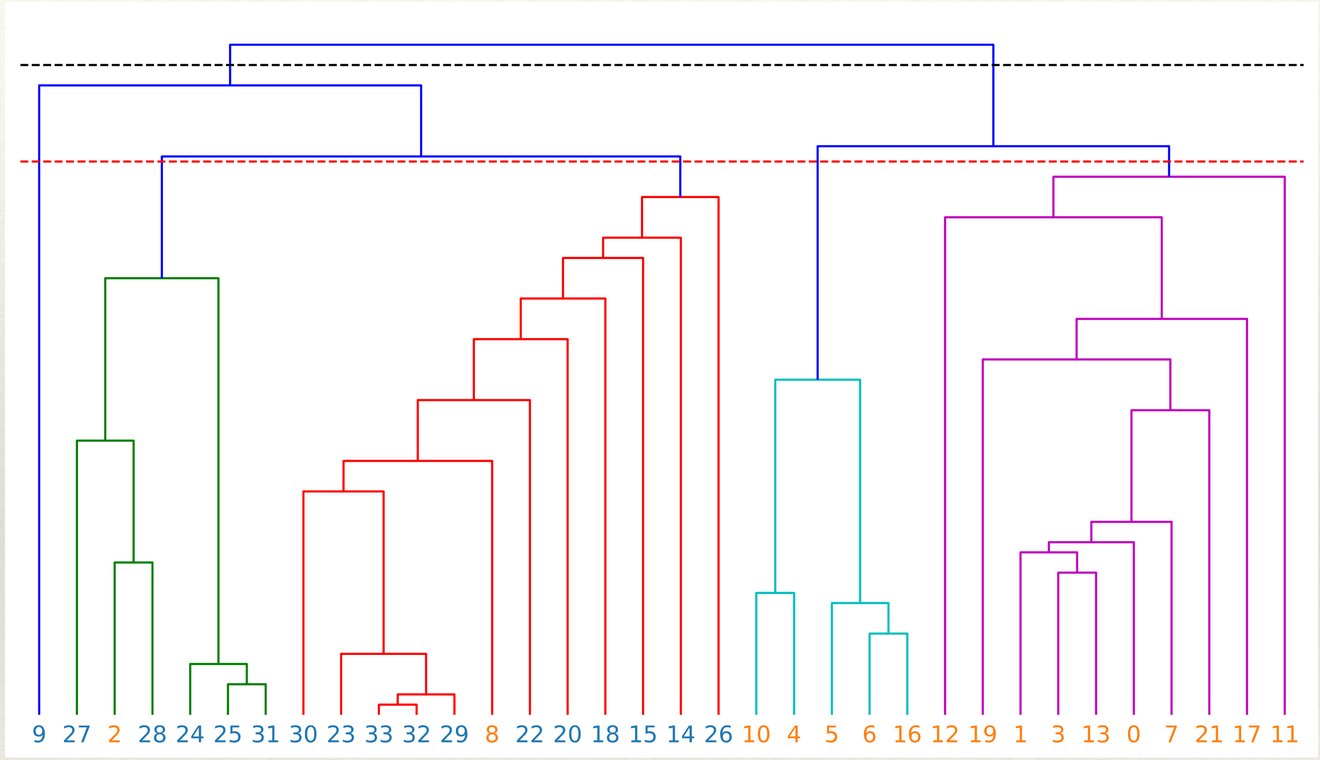
Hierarchical clustering

- **Two approaches**
  - **Agglomerative hierarchical clustering:** partitions are generated by iteratively merging groups of nodes
  - **Divisive hierarchical clustering:** partitions are generated by iteratively splitting groups of nodes

- **Agglomerative hierarchical clustering:**
  - **Start:** partition into N groups, each group consists of one node
  - At each step the pair of groups with the largest similarity are merged

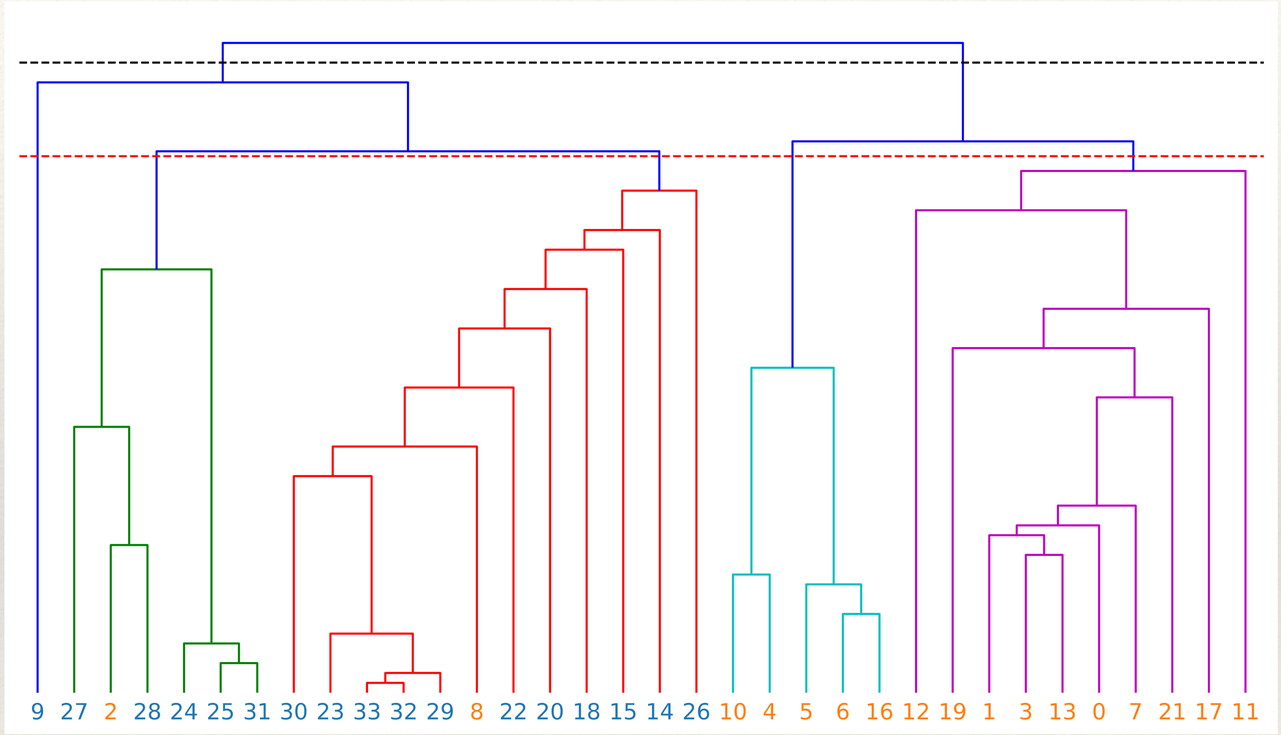
Dendrograms

- Outcome: **dendrogram (hierarchical tree)**
- A dendrogram is a compact summary of all partitions created by hierarchical clustering
- Since each merger reduces the number of groups by one, **the total number of partitions is *N***



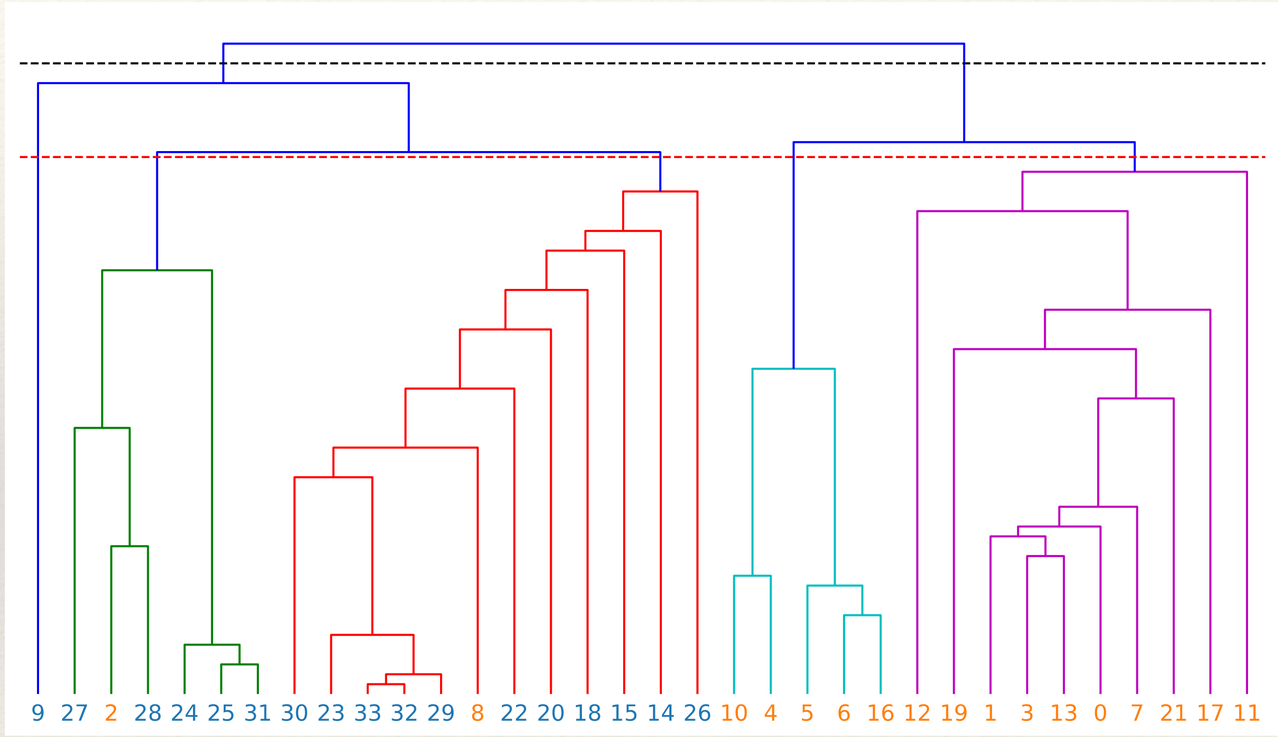
Dendrograms

- **Features:**
  - Bottom: leaves of the tree, indicated by the labels of the nodes
  - Going upwards, pairs of clusters are merged. Mergers are illustrated by horizontal lines joining two vertical lines, each representing a cluster
  - The nodes of each cluster can be identified by following the vertical line representing the cluster all the way down



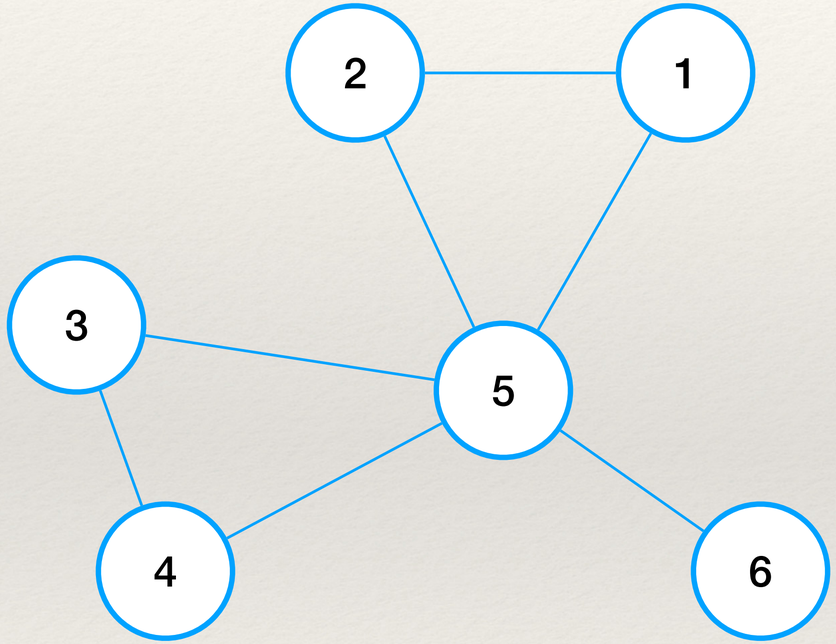
Dendrograms

- Partitions are selected via **horizontal cuts** of the dendrogram: the clusters are the ones corresponding to the vertical lines severed by the cut
- High cuts yield partitions into a few large clusters, low cuts yield partitions into many small clusters
- **Hierarchy:** each partition has clusters including clusters of all partitions lying lower in the dendrogram



Example

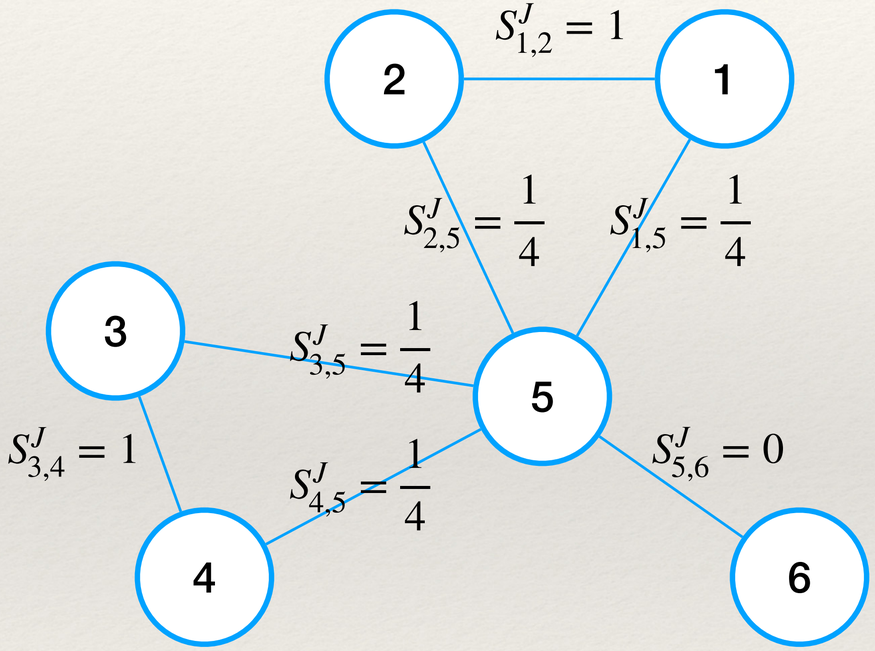
Jaccard Sim. $S^J_{i,j} = \frac{|N_i \cap N_j|}{|N_i \cup N_j|}$



1 2 3 4 5 6

Example

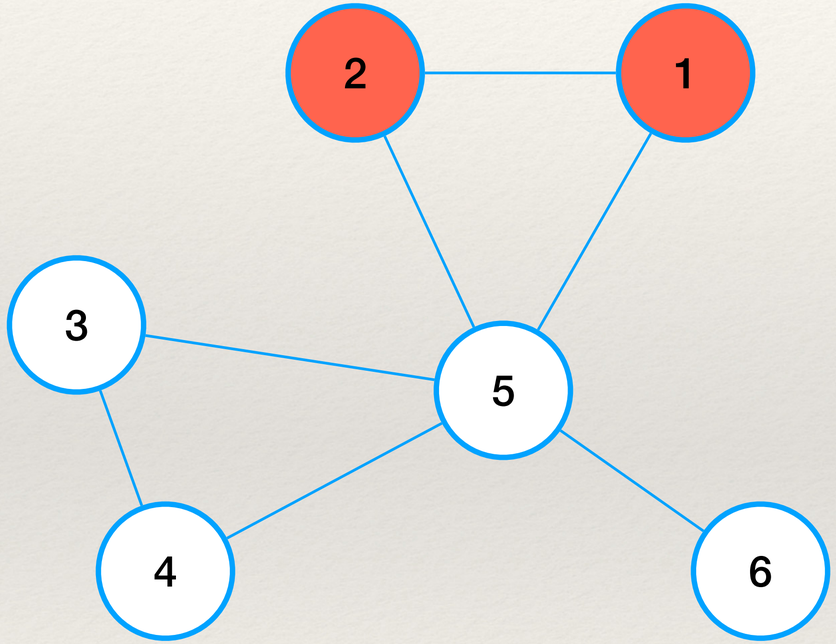
Jaccard Sim. $S^J_{i,j} = \dfrac{|N_i \cap N_j|}{|N_i \cup N_j|}$

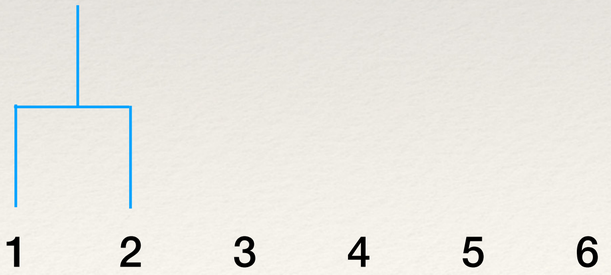


1 2 3 4 5 6

Example

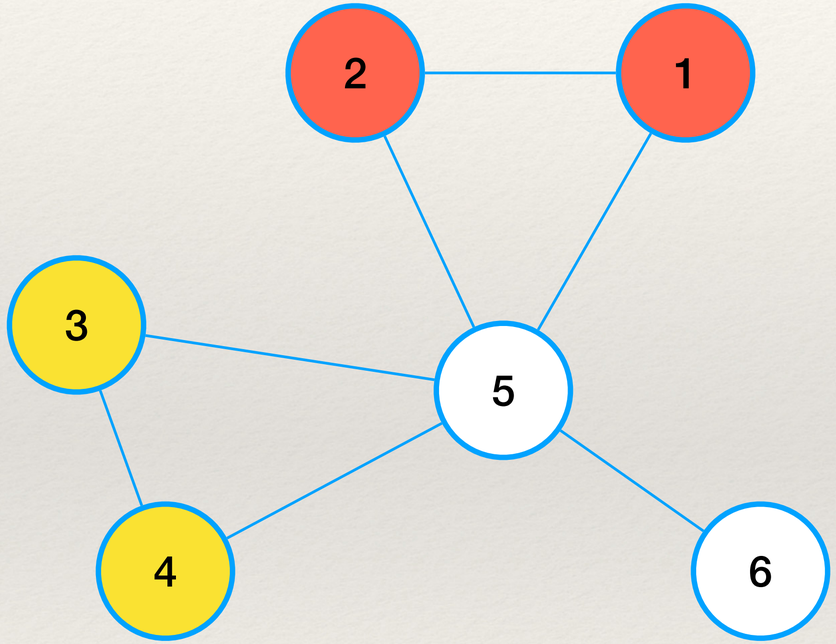
Jaccard Sim. $S^J_{i,j} = \frac{|N_i \cap N_j|}{|N_i \cup N_j|}$

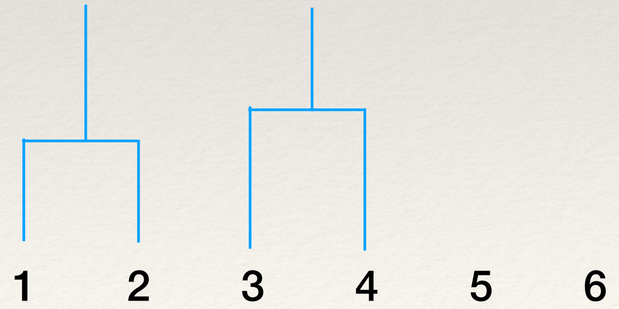




Example

Jaccard Sim. $S^J_{i,j} = \frac{|N_i \cap N_j|}{|N_i \cup N_j|}$

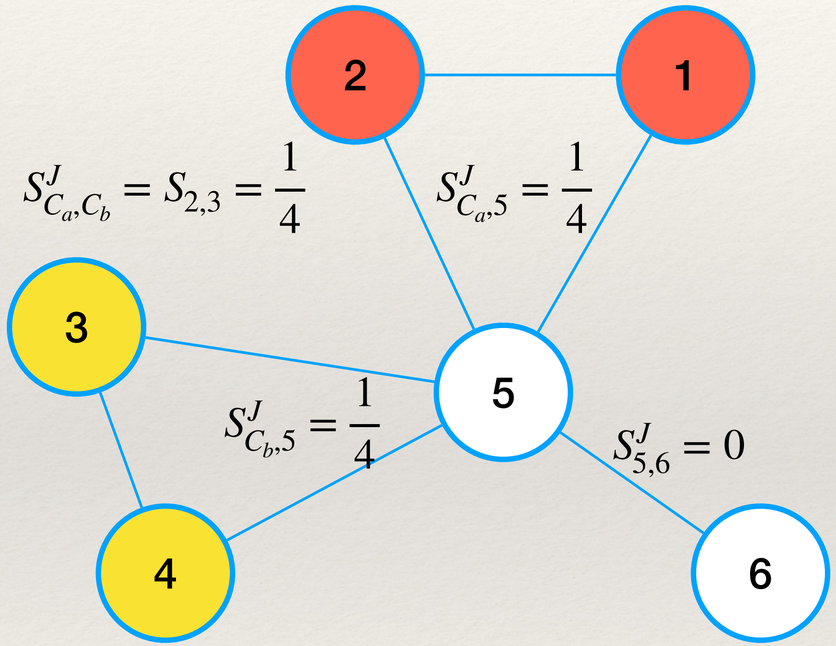


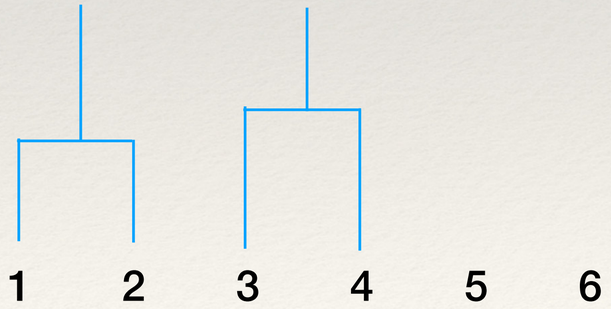


Example

Jaccard Sim. $S^J_{i,j} = \frac{|N_i \cap N_j|}{|N_i \cup N_j|}$

Single linkage $S^J_{C_a,C_b} = maxS_{i,j} : i \in C_a, j \in C_b$

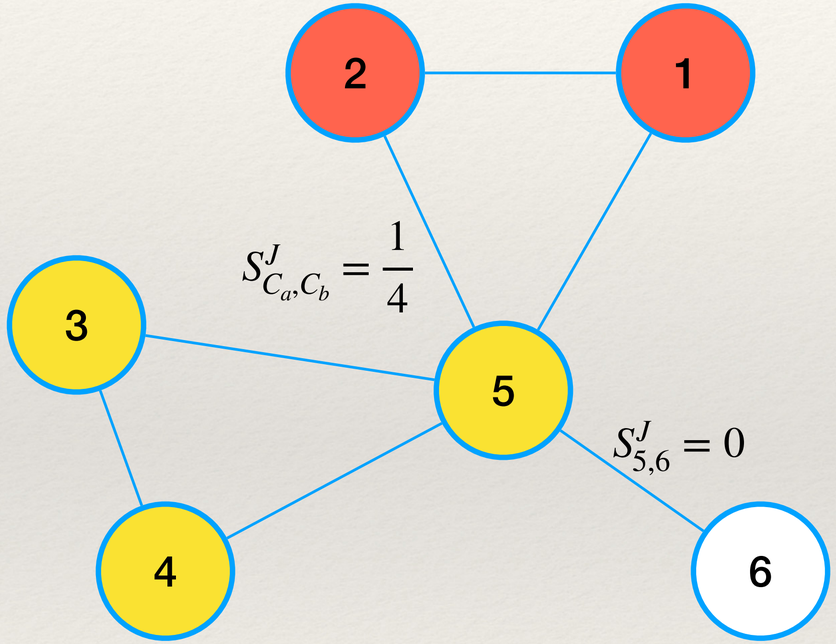


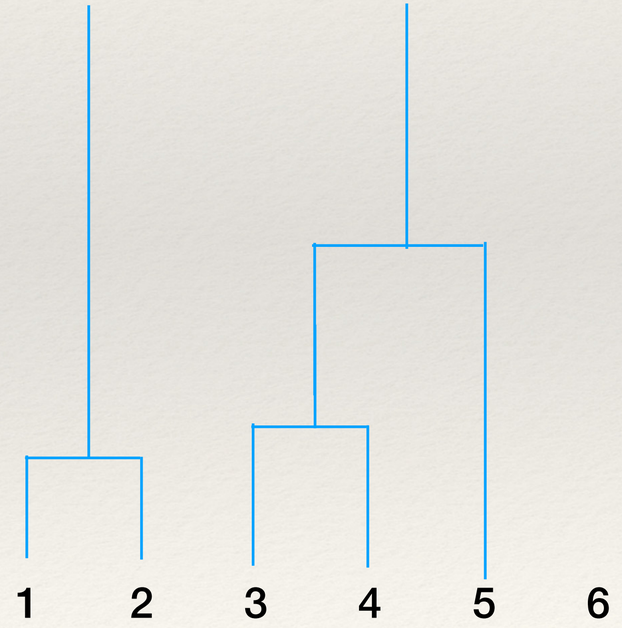


Example

Jaccard Sim. $S^J_{i,j} = \frac{|N_i \cap N_j|}{|N_i \cup N_j|}$

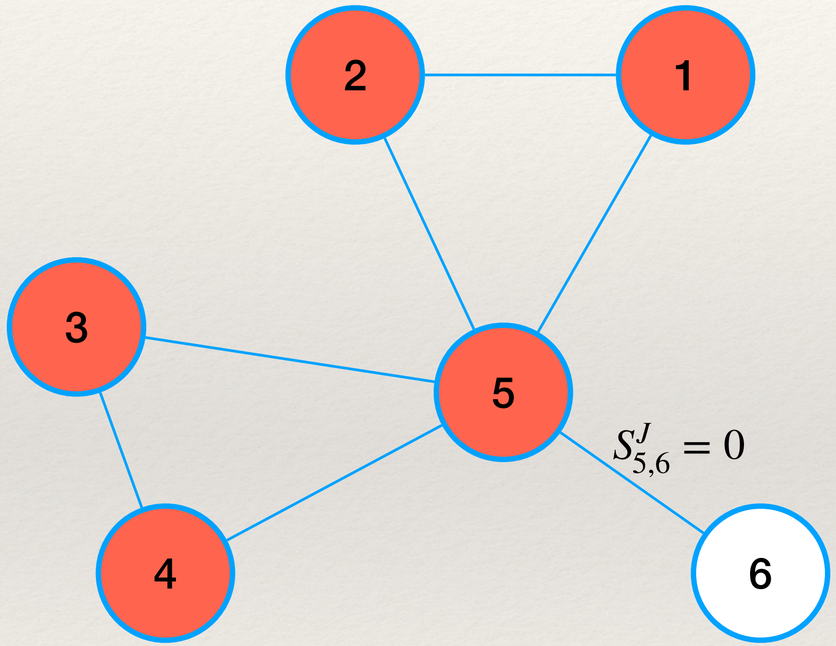
Jaccard Sim. + single linkage $S^J_{C_a,C_b} = maxS_{i,j} : i \in C_a, j \in C_b$



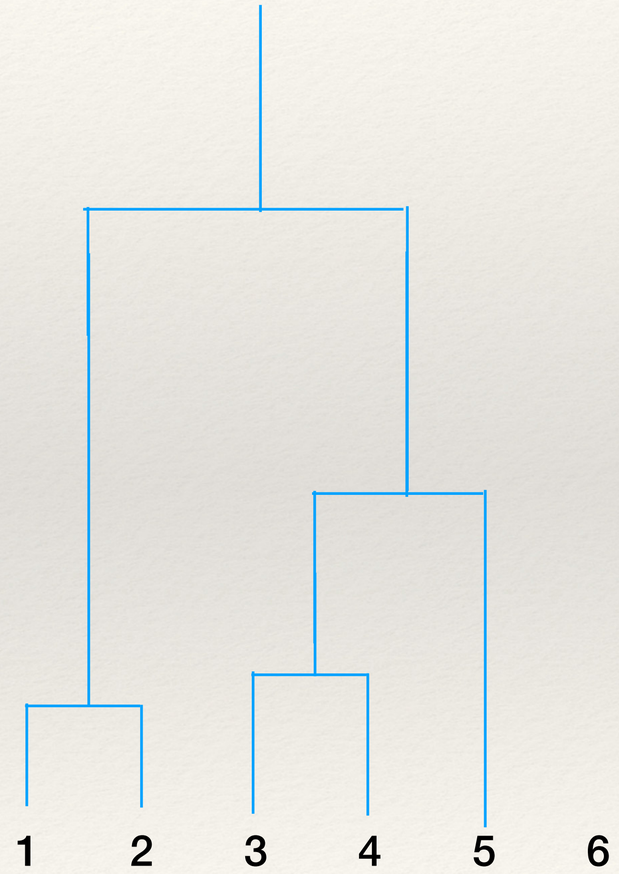


Example

Jaccard Sim. $S^J_{i,j} = \frac{|N_i \cap N_j|}{|N_i \cup N_j|}$

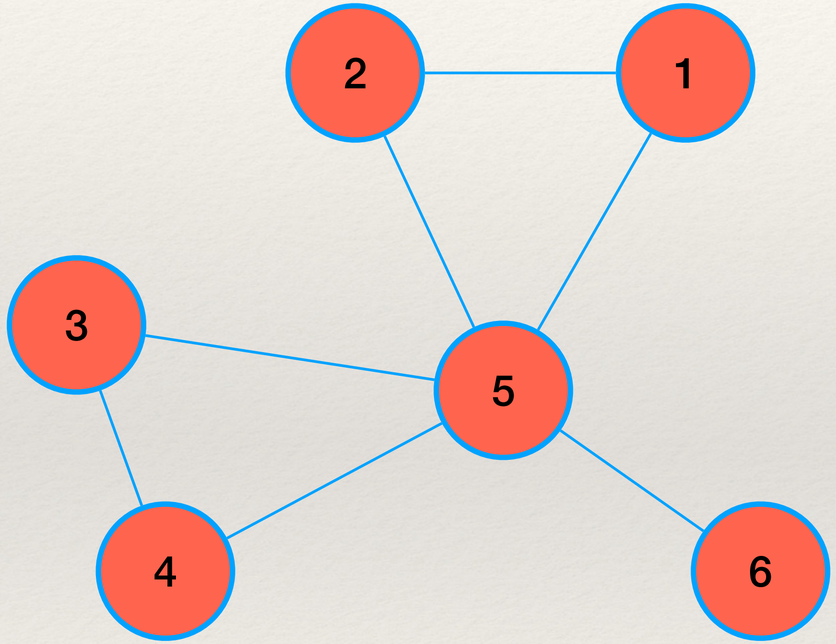


Jaccard Sim. + single linkage $S^J_{C_a,C_b} = maxS_{i,j} : i \in C_a, j \in C_b$

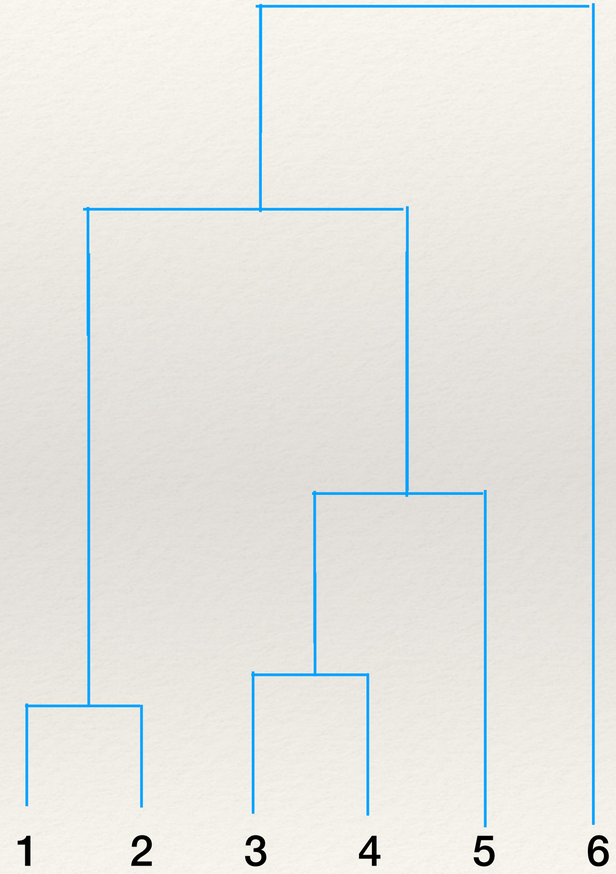


Example

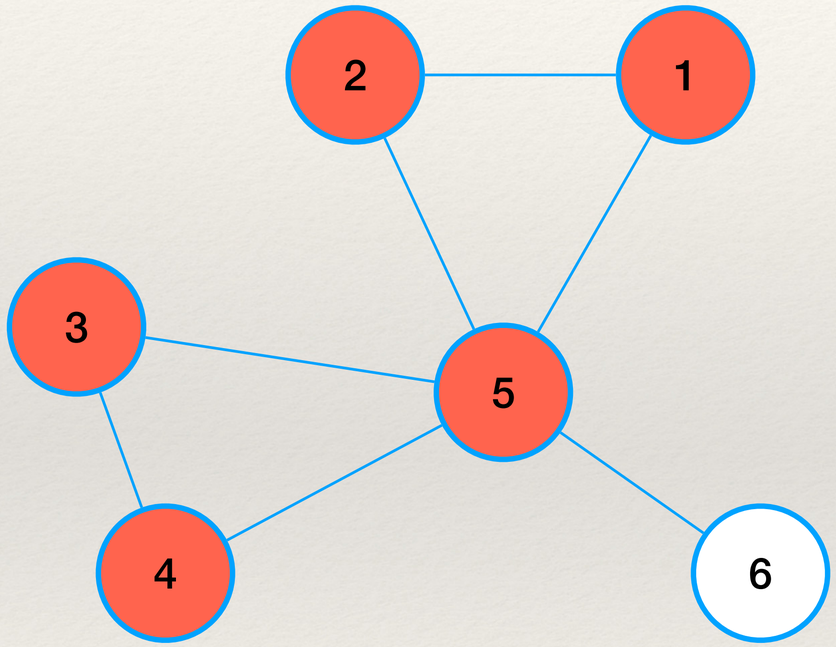
Jaccard Sim. $S^J_{i,j} = \frac{|N_i \cap N_j|}{|N_i \cup N_j|}$

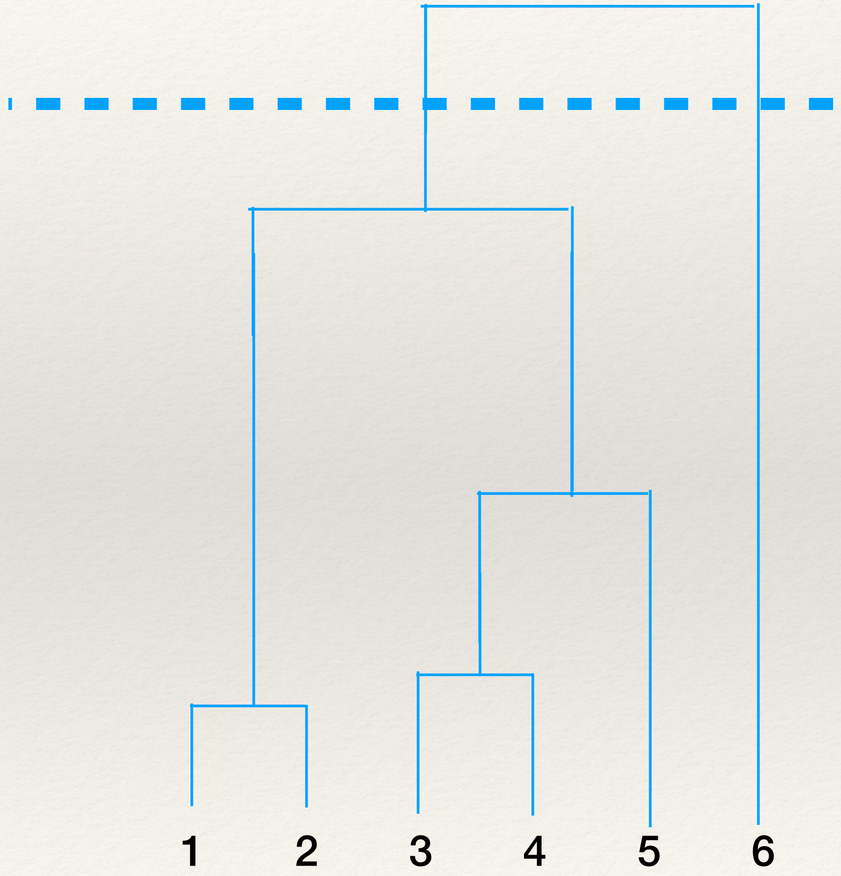


Jaccard Sim. + single linkage $S^J_{C_a,C_b} = maxS_{i,j} : i \in C_a, j \in C_b$

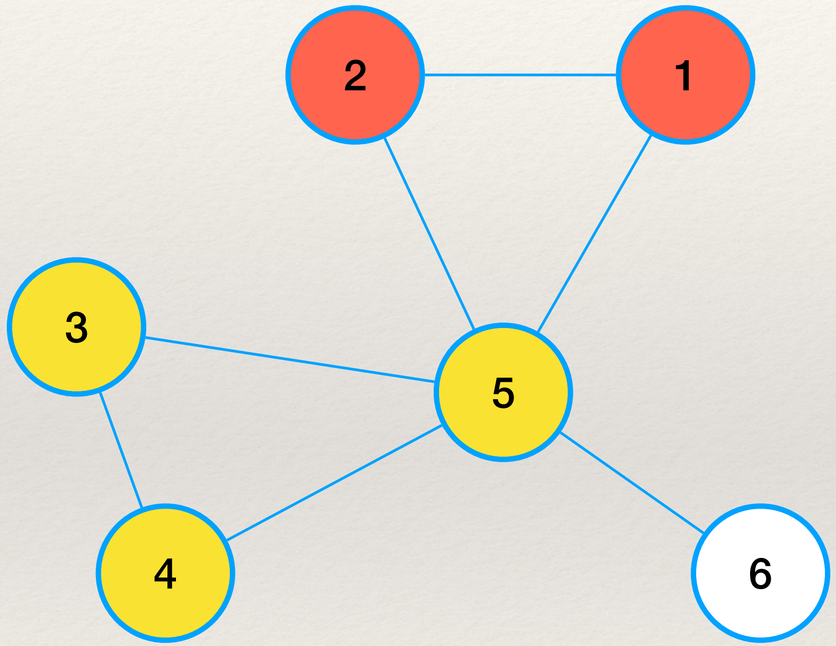


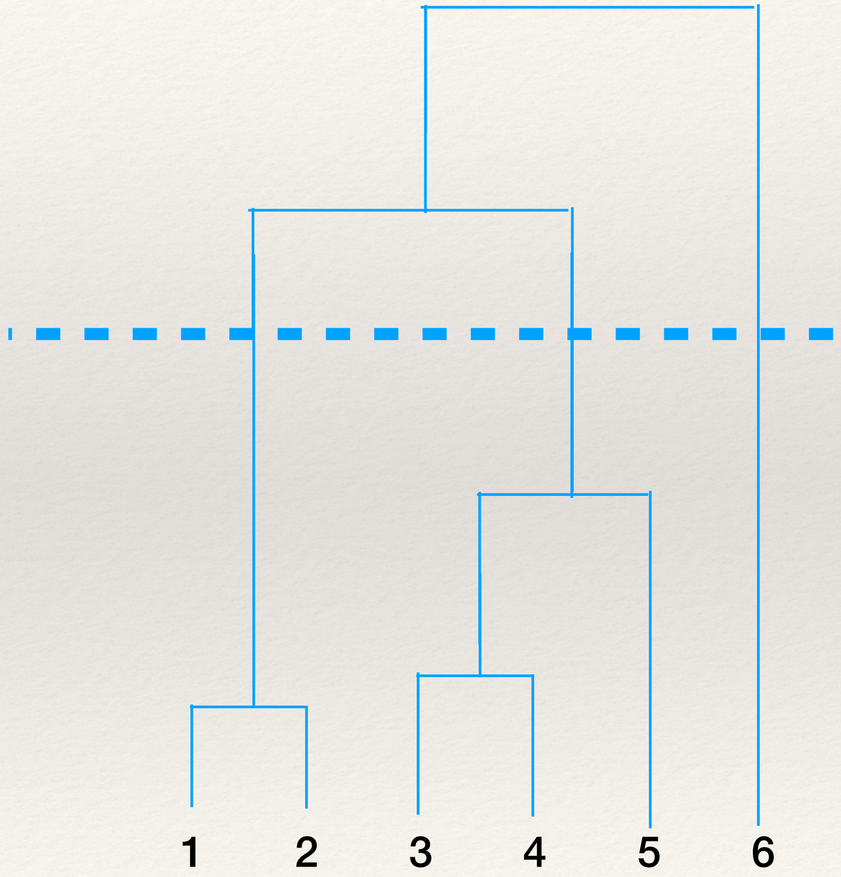
Example



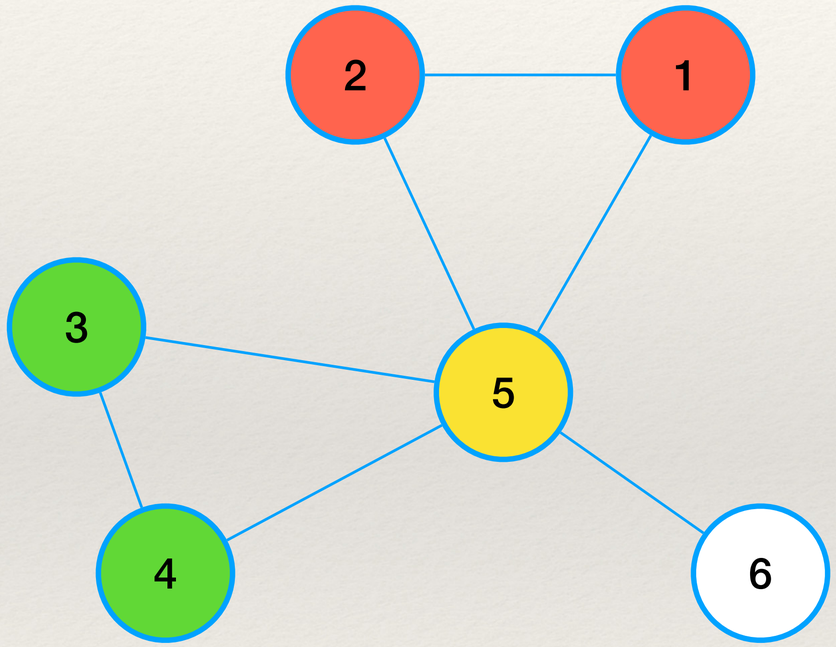


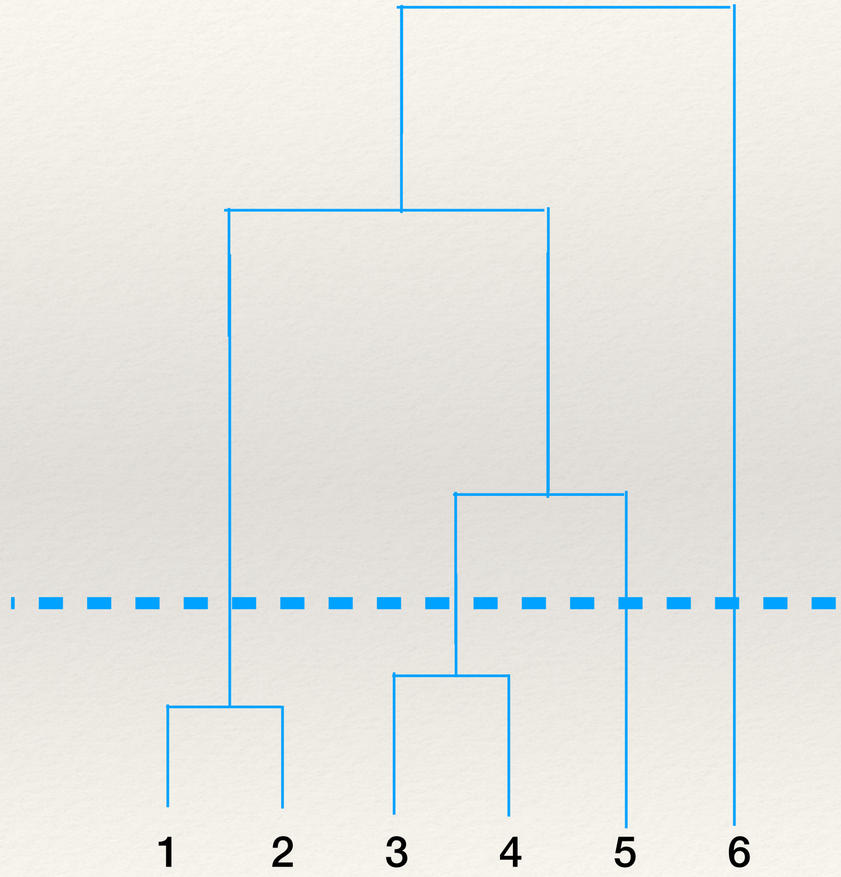
Example



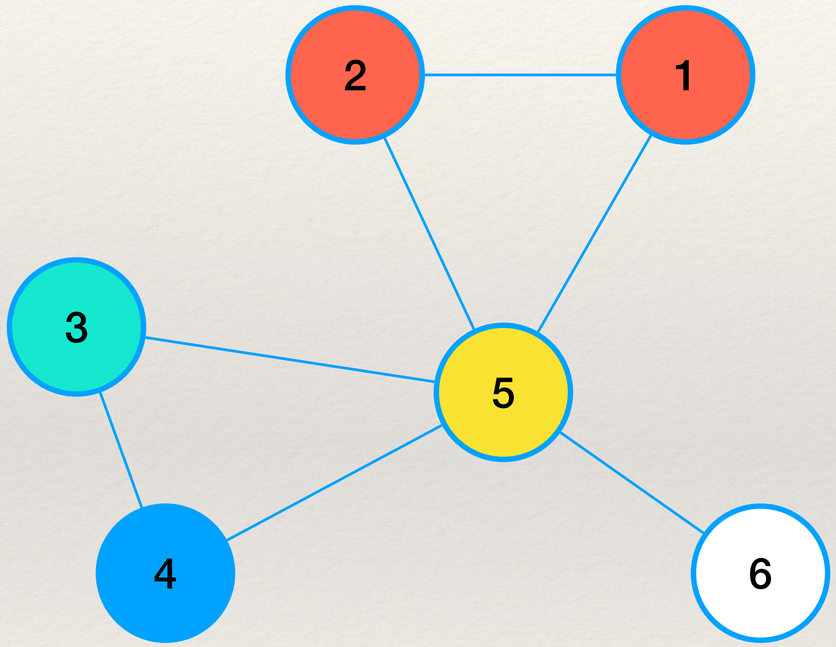


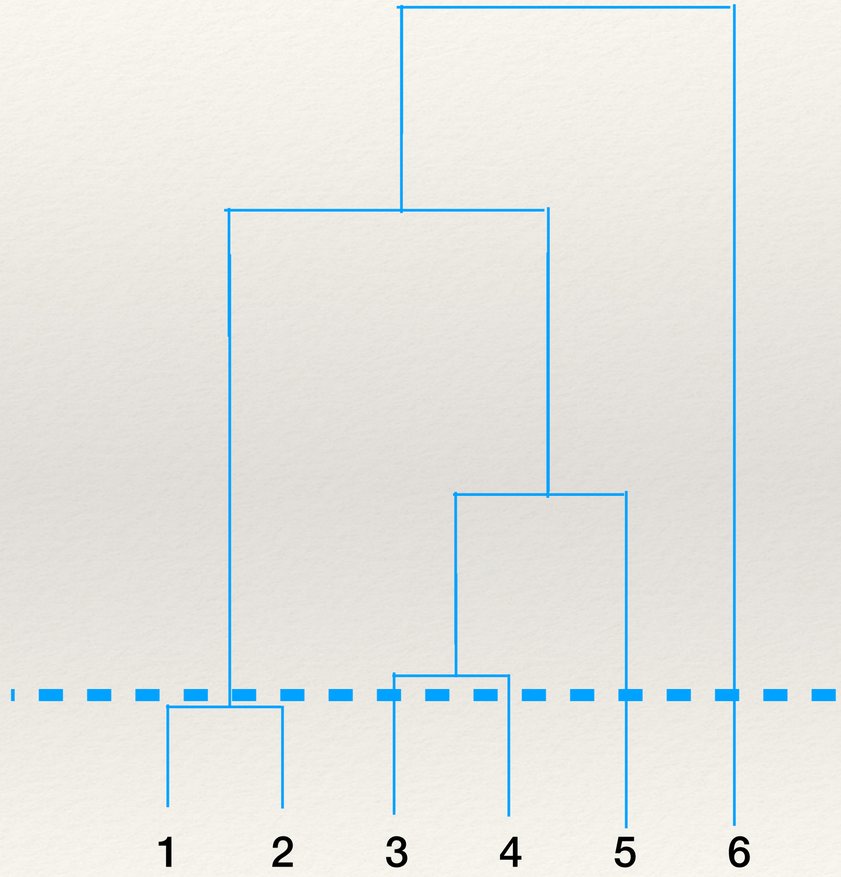
Example



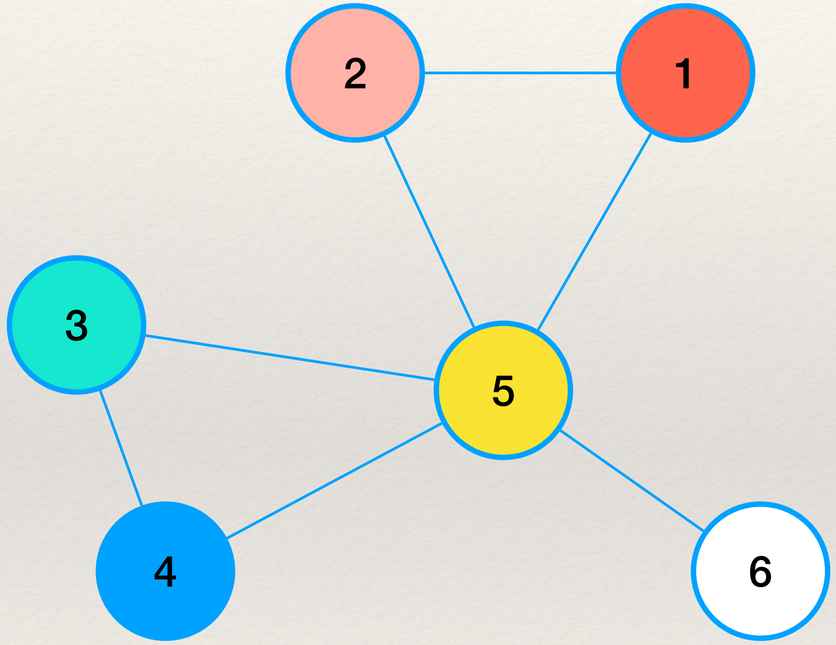


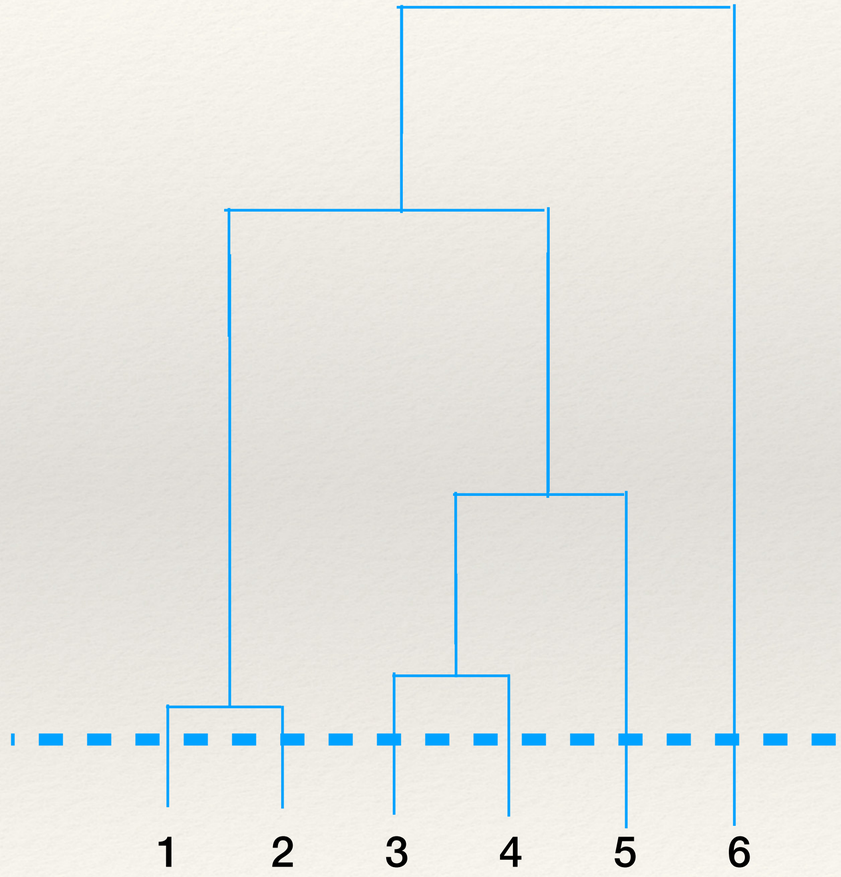
Example





Example





Hierarchical clustering summary

Pros:

- Real graph are typically structured into hierarchical communities and this approach allows one to unveil this structure
- This concept of distance leverages on the relation between weak ties and local bridges

Cons:

- Where should the dendrogram be cut? When going too deep into the dendrogram, relations are simply irrelevant. However, there is not a simple rule to choose where to cut.
- This class of algorithm is slow and only small graphs can be approached
- The result may heavily depend on the distance definition

Node embeddings

Node embeddings

Why is community detection so hard? Because we do not have a simple way to represent nodes if not with the local structure of its connections.

Node embeddings allow one to represent each node as a vector in a (high dimensional) space in which a clear meaning of proximity can be defined

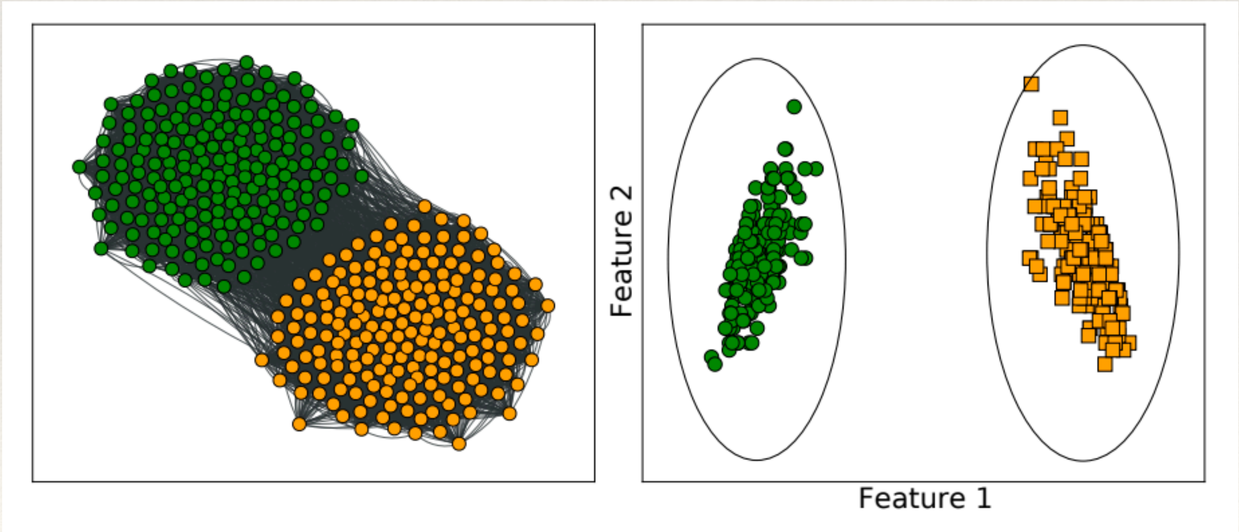
This technique falls into the domain of representation learning: how should represent complex data so that we are able to extract significant information from it?

The vectorial representation of the nodes depends on the graph structure alone. Similar nodes should be mapped into nearby points.

Node embeddings

Typical community detection algorithm with node embeddings:

1. Represent each node of the graph as a vector in d dimensions $i \rightarrow x_i \in \mathbb{R}^d$
2. Cluster the points in the d dimensional space with (for instance) k-means



The k-means algorithm

This algorithm divides a set of points in a high dimensional space into q communities. The algorithm finds the class partition solving the following (NP hard) optimization problem

$$\hat{\ell} = \arg\min_{\ell} \sum_{a=1}^{q} \sum_{i \in N_a} \|x_i - \mu_a\|^2$$

$$\mu_a = \frac{1}{|N_a|} \sum_{i \in N_a} x_i$$

In words, we compute the center of each cluster and assign i to the closest center

The Lloyd algorithm

The Lloyd algorithm is an approximate and fast method to find a (local) minimum of this cost function. It consists of the following steps

1. Initialize the label assignment at random
2. Iterate until convergence
    A. Compute the center of each cluster
    B. Compute the distance of each node from each cluster center

This algorithm runs in O(nqd) operations and it is rather fast

K-means summary

The Lloyd algorithm is the approximation of an NP hard problem and it comes with the usual problems associated with this approach.

- It is likely that the algorithm will fall into a local minimum of the cost function
- The algorithm will find a partition no matter what is the input data
- The number of classes should be given as an input, even if some heuristics (elbow method) exist.
- More accurate methods exist (such as k-medoid or expectation maximization) but they have a higher computational complexity.

To cope with most of the problems above, however, the solution is to provide a meaningful node embedding and to know already whether or not there are communities.

K-means in python

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 2).fit(X)

kmeans.labels_
```

X is a matrix of size (n, d) containing all the embeddings in its rows. Given that the output of the algorithm depends on the (random) initial condition, one might want to iterate this step multiple times and take the best realization.

Spectral clustering

Spectral clustering

This methods consists in creating a node embedding using the eigenvectors of appropriate graph matrix representations

Spectral clustering

1. Build M, a graph matrix representation (a function of the adjacency matrix)
2. Compute q eigenvectors (largest or smallest according to M) of M and store them in a matrix X of size (n, q). Each row corresponds to a node and is a vector of q dimensions
3. Use k-means on X

Spectral clustering with the graph Laplacian

We let A be the graph adjacency matrix of a connected graph and D the diagonal degree matrix. We define the graph Laplacian matrix L as

$$L = D - A$$

Property: consider an arbitrary vector y, then

$$y^T L y = \frac{1}{2} \sum_{i,j} A_{ij} (y_i - y_j)^2$$

The projection of y over L tells us how fast y changes on the graph, hence if y is the community assignment it should change slowly

Spectral clustering with the graph Laplacian

Proof:

$$
\begin{aligned}
y^T L y &= \sum_{i,j} y_i L_{ij} y_j \\
&= \sum_{i,j} y_i (k_i \delta_{ij} - A_{ij}) y_j \\
&= \sum_i y_i^2 k_i - \sum_{i,j} y_i A_{ij} y_j \\
&= \sum_{i,j} y_i^2 A_{ij} - \sum_{i,j} y_i A_{ij} y_j \\
&= \sum_{i,j} A_{ij} (y_i^2 - y_i y_j) \\
&= \tfrac{1}{2} \sum_{i,j} A_{ij} (y_i^2 + y_j^2 - 2 y_i y_j)
\end{aligned}
$$

Spectral clustering with the graph Laplacian

What is the relation with community detection? Recall the RatioCut problem

$$Rcut(N_1, \ldots, N_q) = \frac{1}{2} \sum_{a=1}^{q} \frac{1}{|N_a|} \sum_{i \in N_a} \sum_{b \neq a} \sum_{j \in N_b} A_{ij}$$

For each node we count how many edges it has towards nodes in another community. We average this quantity over all nodes in the same community and we sum over all communities
This problem is NP hard but it can be approximated with spectral clustering

Spectral clustering with the graph Laplacian

Consider the case of q = 2 communities. Let us denote the two sets of nodes as A and B. We define the vector y as

$$y_i = \begin{cases} \sqrt{\frac{|B|}{|A|}} = \sqrt{\frac{N_B}{N_A}} \text{ if } i \in A \\ -\sqrt{\frac{|A|}{|B|}} = -\sqrt{\frac{N_A}{N_B}} \text{ if } i \in B \end{cases}$$

Let us compute yLy for this choice of y

Spectral clustering with the graph Laplacian

$$= \sum_{i \in A} \sum_{j \in B} A_{ij} \left(\frac{N_A}{N_B} + \frac{N_B}{N_A} + 2 \right)$$

$$= \sum_{i \in A} \sum_{j \in B} A_{ij} \left(\frac{N_A + N_B}{N_B} + \frac{N_B + N_A}{N_A} \right)$$

$$= N \left[\frac{1}{N_A} \sum_{i \in N_A} \sum_{j \in N_B} A_{ij} + \frac{1}{N_B} \sum_{i \in N_A} \sum_{j \in N_B} A_{ij} \right]$$

$$= N \cdot \mathrm{RatioCut}(A, B)$$

Spectral clustering with the graph Laplacian

So, if we constrain the vector y to the form we saw, the (NP hard) community detection optimizing the ratio cut problem can be reformulated equivalently as follows

$$\ell = \text{argmin } y^T L y$$

Now we *relax* the optimization problem and let y take any value (and not only the ones introduced some slides ago). The result is the y solving problem is the eigenvector associated to the second smallest eigenvalue of L and it constitutes an approximate solution of RatioCut.

This can be generalized to q > 2 communities.

Spectral clustering with the graph Laplacian

Algorithm:

Build the matrix L from the graph
Compute the q eigenvectors associated to the q smallest eigenvalues of L
Store them in a matrix of size (n,q)
Perform k-means on the rows of X
Return the labels

The algorithm runs in O(Lq^2) operations

The random walk Laplacian

Let us take another approach to introduce a popular spectral clustering algorithm. Suppose that v_i is a value associated to node i. At each iteration all nodes update their value v_i with a value proportional to the average of their neighbors and iterate until convergence.
Mathematically this can be written as

$$v_i^{(t+1)} = \frac{\lambda}{k_i} \sum_{j \in N} A_{ij} v_j^{(t)} = \lambda \left(D^{-1} A v^{(t)} \right)_i$$

The convergence imposes $v_i^{(t+1)} = v_i^{(t)}$ and so that v is an eigenvector of the random walk Laplacian

Spectral clustering with the random walk Laplacian

Algorithm:

Build the matrix L_rw from the graph
Compute the q eigenvectors associated to the q largest eigenvalues of L
Store them in a matrix of size (n,q)
Perform k-means on the rows of X
Return the labels

The algorithm runs in O(Lq^2) operations