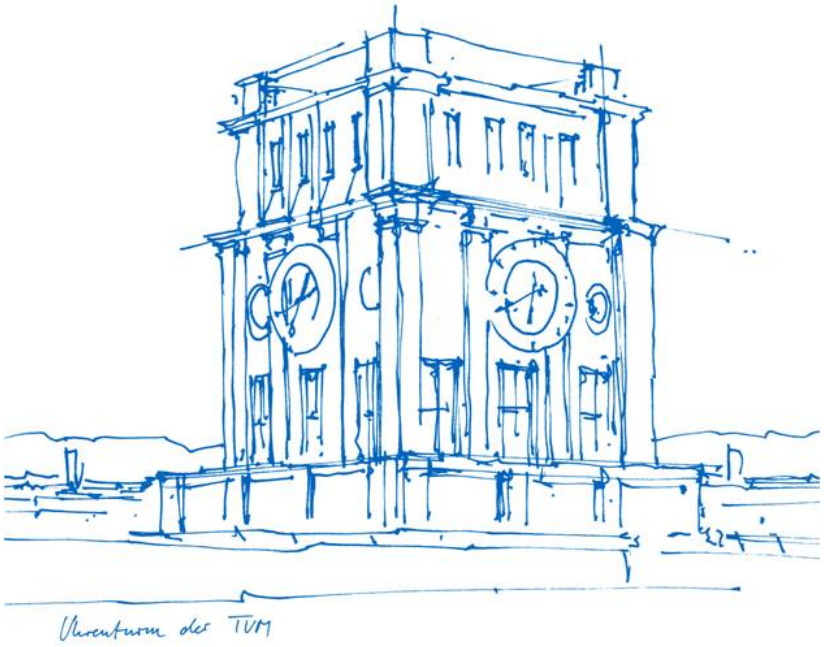
Exercises for Social Gaming and Social Computing (IN2241 + IN0040) – Introduction to Exercise 3



Exercise Content

Sheet Number	Exercise	Working Time
1	• Introduction to Python and Network Visualization	May 24-30
2	• Centrality Measures	May 31 – June 6
3	• Finding Groups & Clustering Methods	June 7 – 13
4	• Predicting Social Tie Strength with Linear Regression	June 14 - 20
5	• Natural Language Processing: Hate Speech detection + Social Context Influence	June 21 - 27
6	• Natural Language Processing: Modern Machine Learning Methods and Explainable AI	June 28 – July 4

Exercise Sheet 3: Clustering

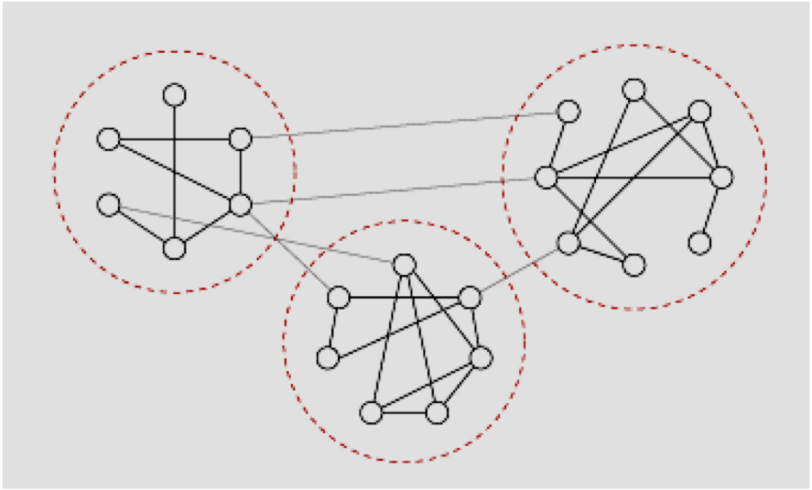
- Goals:
  - Being able to apply and judge different clustering measures for finding groups
  - calculate and apply Modularity
  - understand how the Louvain Algorithm works

- Data: no external data files required
  - simple graphs
  - dataset containing two overlapping Gaussian distributions as well as a uniform distribution

Part 1: Graph Clustering:
Implement Louvain Clustering Method

Newman Girvan Method: Centrality-based Splitting + Modularity

Modularity:



[3]

- k clusters → k x k symmetric matrix **e**: e_{ij} = |E(C_i,C_j)| / |E| : fraction of edges between communities

- $\mathrm{Tr}\,\mathbf{e} = \sum_i e_{ii}$: fraction of edges within communities

- $a_i = \sum_j e_{ij}$: fraction of edges that connect to cluster C_i

- Random network (keep a_i fixed): $e_{ij}^{\mathrm{rnd}} = a_i a_j \rightarrow e_{ii}^{\mathrm{rnd}} = a_i^2$

- f: Compare (→difference) real with rnd → $Q = \sum_i (e_{ii} - a_i^2) = \mathrm{Tr}\,\mathbf{e} - \| \mathbf{e}^2 \|$

Newman Girvan Method: Centrality-based Splitting + Modularity

How to compute Modularity with a given (weighted) adjacency matrix?

- Real graph: Fraction of edges within clusters:

|E(C_i)| / |E| = $\frac{\sum_{ij} A_{ij}\delta(c_i,c_j)}{\sum_{ij} A_{ij}} = \frac{1}{2m}\sum_{ij} A_{ij}\delta(c_i,c_j)$ $\qquad m = \frac{1}{2}\sum_{ij} A_{ij}$

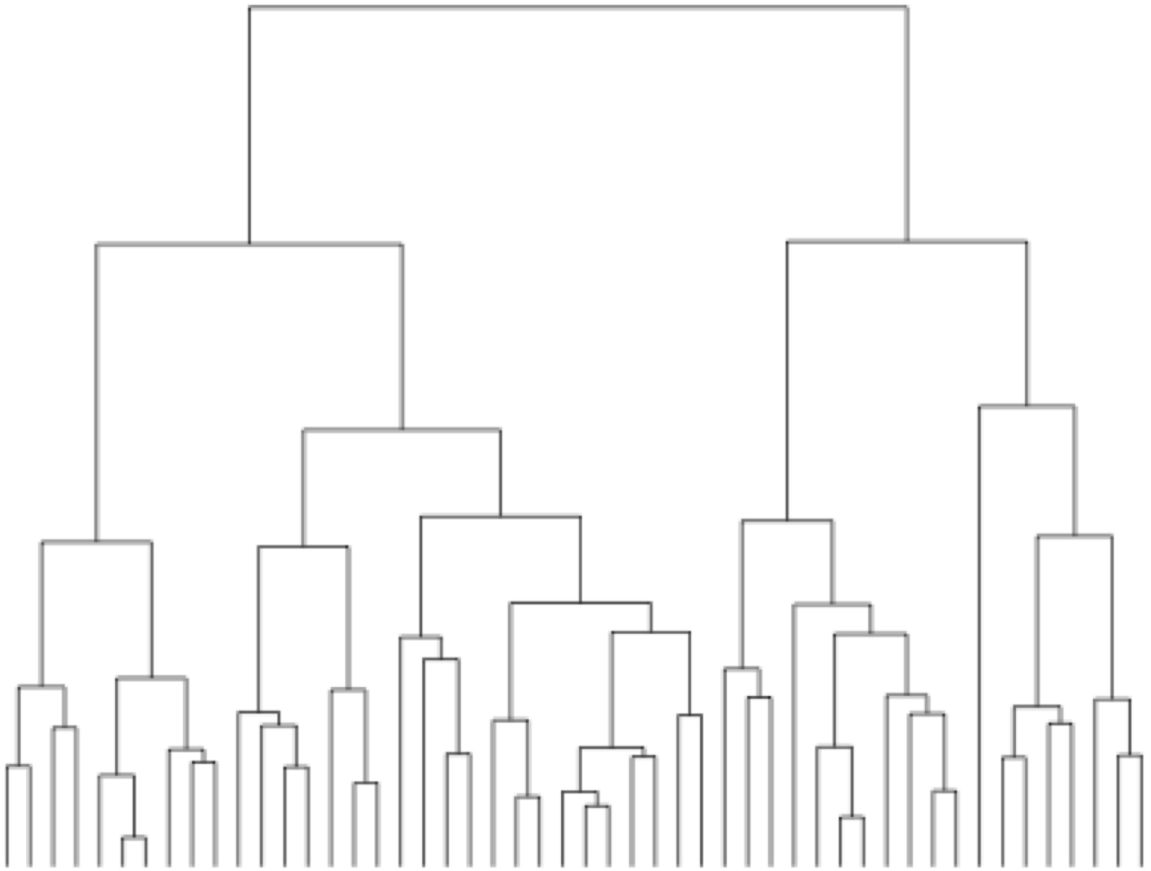
- Random graph (keep degrees k_i of vertices fixed): prob of edge between vertices i and j is $k_i k_j/(2m)^2$

$$k_v = \sum_w A_{vw}.$$

- → modularity: $Q = \frac{1}{2m}\sum_{ij}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i,c_j)$

- → with the new formulation we can also compute modularity for weighted graphs

All Hierarchical Bottom-Up Clustering Approaches: Dendrogram



Similar Method also Using Modularity: Louvain-Method

Modularity (as before):

$$Q = \frac{1}{2m} \sum_{i,j \in V} [A_{ij} - \frac{k_i k_j}{2m}] \delta(c_i, c_j)$$

where $m = 1/2 \sum_{i,j \in V} A_{ij}$ and c_n is the id of the cluster that node n is in and $\delta(x, y) = 1$ if $x = y$ and $\delta(x, y) = 0$ else.

We may as well use the following simplified formula [1]:

$$Q = \sum_{c=1}^{n} [\frac{L_c}{m} - (\frac{k_c}{2m})^2]$$

with L_c being the number of intra-community links for community c, and k_c being the sum of degrees of the nodes in community c.

Louvain method:

- (1) For each node i: For each neighbour j: Calculate modularity values for possible merges of i with the cluster containing j. Add i to the cluster with highest resulting modularity gain (if no gain is possible, i stays as it is)
- (2) Merge each cluster into a new node representing the cluster.
- Repeat (1) + (2) until no improvement can be made

Tasks

a) Modularity

In order to compare different clusterings later, the first thing to do is **implementing a function for the modularity calculation** by means of the formula given above.

In addition to that we want to calculate the modularity gain when analyzing possible merges for a node i into a cluster c. We get the following expression:

$$\Delta Q_{c,i} = \frac{k_{i,c}}{m} - \frac{2\Sigma_{tot}\ k_i}{2m}$$

where $k_{i,in}$ denotes the sum of weights of edges from node i to nodes in cluster c and Σ_{tot} the sum of weights of edges incident to nodes in c.

b) Community Aggregation

After finishing the first step, all nodes belonging to the same community are merged into a single node. This step also generates self-loops which are the sum of all links inside a given community, before being collapsed into one node.

Tasks (cont.)

c) Louvain Loop

The next step is to implement the core functionality. Determine the best partition based on modularity in the current graph and use your function `mergeCommunities` to merge accordingly. This is repeated until no improvement can be made. The final output is the graph with the best clustering and the corresponding modularity value.

Notes:

- It may be helpful to look at the lecture's slides on the topic again for understanding modularity and clustering algorithms.
- Use this pseudo-code [3] as help, the algorithm and pseudo-code is explained on this website [2], that we have already linked above.
- You are free to implement everything on your own (as long as the result is correct), but the given code construct should give you a good starting point.
- Our test graph is taken from this website [4] which once again explains the algorithm and you can lookup the correct partition of said graph there.
- For testing purposes you might want to use more than one graph. Remember to assign weights to your graphs edges as we did below on the Krackhardt Kite Graph.

Part 2: Metric Clustering: Compare K-Means and GMM

Task 3.2: Clustering Comparison

Now you have implemented a clustering algorithm yourself, it is time to compare performances of different clustering approaches. It is your task to compare algorithms "K-means" and "Gaussian Mixture Models" and evaluate their results.

a) The dataset

First we create a dataset. The dataset contains samples from two overlapping Gaussian distributions as well as a uniform distribution. You can change the distributions to different values for testing and answering questions below but besides that you do not have to edit this code block.

b) Prediction

With a dataset in your hands, you can now define a K-means and GMM model that compute clusterings. Use the same number of max iterations for both, so they are comparable to each other.

First you have to initialize the models and then compute clusterings on our our data.

Notes:

- You do not have to code anything yourself, just use the pre-defined functions.
- Think about an optimal number of clusters for each case.

Tasks (cont.)

c) Plotting and evaluation

Finally you can plot the results. You can change different parameters in the code blocks above to answer the following questions:

- What kind of effect does the amount of datapoints have?
- Describe the kind of error (shape) K-means/GMM produce!
- Compare the results of K-means and GMM with different parameter settings

TODO: Write your observations here

Submitting Your Solution

- work by expanding the .ipynb iPython notebook for the exercise that you downloaded from Moodle.
- save your expanded .ipynb iPython notebook in your working directory.
  Submit your .ipynb iPython notebook via Moodle (nothing else)
- remember: working in groups is not permitted.
  Each student must submit her own ipynb notebook!
- we check for plagiarism. Each detected case will have the consequence of 5.0 for the whole exercise grade.
- deadline: please check Moodle