Weekly notes on Online Social Network Polarisation Project

Adrián Fernández Cid

August 6, 2021

Link to the GitHub.

1 Week 1 (05-11/07)

The objectives for this first week were:

1. Find some open source agglomerative clustering code
2. Use the found code to reproduce Emanuele's procedure with some toy model (projecting a bipartire network, obtaining the two-point phi correlations, transforming those to Euclidean distances, applying some agglomerative clustering based on those initial distances and measuring the polarisation at a given level of the dendogram –eg 70% progress–).
3. Modify the code to use the "polarisation distance" for clustering

Progress was:

1. Found scipy method, scipy.cluster.hierarchy.linkage. I later found sklearn's AgglomerativeClustering class, but it seems a bit more complex (it is a whole class) and doesn't implement clustering by centroid distance, so I decided to use scipy's method for now.

The scipy method takes a condensed distance matrix and outputs a "linkage matrix" Z, with 4 columns and $n-1$ (number of steps, where n is the number of points/obsevations) rows: the first 2 columns are the clusters merged at a given step (row), the third is their distance and the fourth is the size of the resulting cluster.

The method includes several different criteria for clustering: simple, complete, average, centroid, median, Ward... some of which use different algorithms. Ward works with nearest-neighbour chain clustering, centroids with other method: since Emanuele's notes specify they've been using Ward, I looked at the NN chain algorithm. The actual routine is simple, as shown in algorithm 1.

The scipy implementation is optimised with `cython` and encapsulated functions: that's why we decided to follow Oriol's suggestion for Week 2 to just

Algorithm 1 Nearest-Neighbour Chain Clustering

procedure NN_CHAIN(D, *method*) ▷ D: distance matrix, *method*: criterion
  Initialise linkage matrix Z, size list P, cluster list C and cluster *chain* (empty)
  for $k \in range(n-1)$ **do** ▷ n: number of initial clusters
    if *chain* empty **then** APPEND(*chain*, c) for 1 $c \in C$
    while TRUE **do**
      $x \leftarrow chain[-1]$ ▷ Python indexing
      if *chain length* > 1 **then** ▷ To avoid going in cycles
        $y \leftarrow chain[-2]$
        $current\_min \leftarrow D_{xy}$
      else $current\_min \leftarrow \infty$
      for every $i \in C / i \neq x$ **do**
        if $D_{xi} < current\_min$ **then** $y \leftarrow i$
      if $y = chain[-2]$ **then** BREAK ▷ Break if found 2 mutual NN
      else APPEND(*chain*, y)
    REMOVE(*chain*, $chain[-2:]$) ▷ Pop out 2 top clusters
    $Z_{k0} \leftarrow x$ ▷ Update Z (except for labelling of new clusters)
    $Z_{k1} \leftarrow y$
    $Z_{k2} \leftarrow current\_min$
    $Z_{k3} \leftarrow P_x + P_y$
    for every $c \in C / c \neq xy$ **do** ▷ Update D (xy: new cluster)
      $D_{xy,c} \leftarrow method(xy, c)$
  SORT(Z, by=$Z_{:,2}$) ▷ Sort Z rows by ascending distance
  LABEL(Z) ▷ Correctly label new clusters in Z
  return Z

redo the code from scratch in python (copy-pasting whatever useful), to avoid unrelated software issues and get a deeper feel of the algorithm.

2. Did that for Ward and the Southern Women dataset (code in `SoutherWomen.ipynb`).

The Southern Women dataset is a bipartite network of 18 women linked to 14 different social events only if they attended the given event. It is a popular dataset for testing clustering methods in bipartite networks, as there are two distinct communities of women, each preferring one of two classes of social event. Data from http://casos.cs.cmu.edu/computational_tools/datasets/external/davis/index2.html.

I only got up to the dendogram, though: I don't how to smartly get all cluster sizes and distances for a given step from the Z in order to get the polarisation. I intended to modify the code to compute polarisation on the fly, but Emanuele pointed out one could get all relevant distances and sizes from the linkage matrix: I did not quite understand how, though.

Emanuele also pointed out that Ward is ok but if one uses it then one has to recover the centroid distance from the Ward distance by multiplying by some factor, because the polarisation formula only guarantees the imposed theoretical axioms when applied with centroid distance.

3. Couldn't get to this in time. Would do it next week with code from scratch (easier).

For Week 2 then I had to redo the agglomerative clustering code in `python` and check its results with those of the original `scipy` method.

2 Week 2 (12-18/07)

The objectives for the week were:

1. Remake the agglomerative clustering code in `python`
2. Check it gives the same as the `scipy` method

1. Basically copy-pasted and decythonised everything necessary (see `utils/clustering.py`). Since I had been studying the NN-chain algorithm, I used that one, and the routine is the one explained in algorithm 1.

2. Did so for Southern Women dataset (see `SouthernWomen.ipynb`), and the Z are the same for any clustering criterion using the NN-chain algorithm in the `scipy` version: since that is not the case for centroids, results are a bit different for them, but it makes sense due to `scipy`'s using a different algorithm in that case. In any case, the validity of the code w.r.t. `scipy` is proven by the checks with Ward and other methods using the same algorithm.

I also implemented the polarisation distance method,

$$d_{u,v} = d^c_{u,v}(\pi_u^{1+\alpha}\pi_v + \pi_v^{1+\alpha}\pi_u) \tag{1}$$

where $d_{u,v}$ is the polarisation distance (the one used for clustering) between u and v, $\alpha \in (0, 1.6]$, π_i is the size of cluster i and $d^c_{u,v}$ is the centroid distance

between u and v. For this, I simply had to keep updating another distance matrix with centroid distances at the same time as the polarisation one. Results seem ok for now.

However, I still don't see how to get all the distances and sizes from the final Z. I think I'll just make the code compute the polarisation at each step and append it to a 5th column in the Z (which should be easy).

3 Week 3 (19-25/07)

Objectives for the week:

1. Finish updating past weekly notes
2. Polish code (clustering in `utils/clustering.py` + `SouthernWomen.ipynb`)
3. Add the computation of the polarisation at every step to the clustering code

Progress:

1. Done

2. Done

3. I'm on it.

4 Week 4 (26/07-01/08)

Finished adding the computation of the polarisation at every step in the clustering code (updated `utils/clustering.py` and `SouthernWomen.ipynb`). The code now also returns the global polarisation P at every step (except when there is only one cluster, because the polarisation of such a system is null). The formula is:

$$P = K \sum_{i=1}^{n} \sum_{j=1}^{n} d_{i,j}^{c} \pi_i^{1+\alpha} \pi_j \tag{2}$$

where $\alpha \in (0, 1.6]$, π_i is the size of cluster i and $d_{i,j}^{c}$ is the centroid distance between i and j.

Although the computation of the global polarisation is implemented for every clustering method available, for now only "polarisation" and "centroid" use the right distance for the above formula (the centroid distance): I still have to correct the distance for the other cases.

As we might expect, polarisation seems to increase more or less monotonically when clustering by polarisation distance, while fluctuating more when using centroid distance.

For now I've only implemented the above for $\alpha = K = 1$: it will be nice to play with the available range of these parameters and see what happens.

Finally, I've also corrected the uni-partite projection of the network: as Emanuele suggested, the analysis in `SouthernWomen.ipynb` is now on the more interesting women network, rather than on the events one.