MDL population clustering

Alec Kirkley

February 14, 2022

1 Unipartite model

- Improvements on existing method (Gibbs sampling the posterior):
  - infers K automatically, whereas the other method requires separate runs of the algorithm at each K to select K
  - allows model selection via comparison of compression. can see if this model can compress the data at all, and in some cases no values of the parameters can compress the data (like a "compressibility transition")
  - first transmission step decouples the densities of the modes, so we can have modes with different densities
  - does not require sampling
  - performance improvement per move/sample from $O(NSK)$ to worst-case $O(NS)$
  - other method requires prior on mode density for good performance, but this one has no free parameters
- Let there be S undirected networks $\mathcal{D} = \{\boldsymbol{D}^{(s)}\}_{s=1}^{S}$ of N nodes each that we want to summarize with K modes $\mathcal{A} = \{\boldsymbol{A}^{(k)}\}_{k=1}^{K}$ that have associated clusters $\mathcal{C} = \{C_k\}_{k=1}^{K}$. Let $S_k = |C_k|$ be the number of networks in cluster C_k, t_k be the number of true positive edges in cluster C_k, and f_k be the number of false positive edges in C_k. Let M be the total number of edges in all networks, M^* be the total number of edges in all modes, and M_k^* be the number of edges in mode $\boldsymbol{A}^{(k)}$. We will assume that all networks in $\mathcal{D}$ and $\mathcal{A}$ have no multi-edges.
- Suppose we want to transmit to a receiver all of the networks $\mathcal{D}$. We ignore the transmission of S, N, M, M^*, M_k^*, S_k, t_k, f_k, and K, as for $K \ll S, N$ these all have comparatively negligible information content and they clutter the expressions.
- If we want to transmit all of the networks $\mathcal{D}$ to a receiver, a naive way of doing it could be to use a single fixed-length code to transmit the where all of the edges are in each network, without using representatives. This would require an information of

$$\mathcal{L}_0(\mathcal{D}) = \log \binom{S\binom{N}{2}}{M}. \tag{1}$$

  This is the baseline amount of information to which we can compare the compression of our model.
- If we want to do better, we can transmit a set of modes $\mathcal{A}$ first, then transmit which cluster C_k each network in $\mathcal{D}$ belongs to, then transmit the exact positions of the edges of all the sample networks in $\mathcal{D}$ using all the information we already transmitted. The information content of each step in the process is (with Stirling approximations)

– transmitting the modes $\mathcal{A}$ (all their edge positions):

$$\mathcal{L}(\mathcal{A}) = \sum_{k=1}^{K} \log \binom{\binom{N}{2}}{M_k^*} \tag{2}$$

$$\approx \binom{N}{2} \sum_{k=1}^{K} H\left(\left\{\frac{M_k^*}{\binom{N}{2}}, 1 - \frac{M_k^*}{\binom{N}{2}}\right\}\right), \tag{3}$$

where H is the Shannon entropy

– transmitting the cluster labels of the networks in $\mathcal{D}$:

$$\mathcal{L}(\mathcal{C}) = \log \frac{S!}{\prod_{k=1}^{K} S_k!} \tag{4}$$

$$\approx S H(\{S_k\}), \tag{5}$$

where H is the standard entropy function

– transmitting the networks $\mathcal{D}$ given the cluster assignments $\mathcal{C}$ and modes $\mathcal{A}$:

$$\mathcal{L}(\mathcal{D}|\mathcal{A},\mathcal{C}) = \sum_{k=1}^{K} \log \left[\binom{S_k M_k^*}{t_k} \binom{S_k\left(\binom{N}{2} - M_k^*\right)}{f_k}\right] \tag{6}$$

$$\approx \sum_{k=1}^{K} S_k M_k^* H\left(\left\{\frac{t_k}{S_k M_k^*}, 1 - \frac{t_k}{S_k M_k^*}\right\}\right) \tag{7}$$

$$+ \sum_{k=1}^{K} S_k \left(\binom{N}{2} - M_k^*\right) H\left(\left\{\frac{f_k}{S_k\left(\binom{N}{2} - M_k^*\right)}, 1 - \frac{f_k}{S_k\left(\binom{N}{2} - M_k^*\right)}\right\}\right)$$

- Adding it all together gives

$$\mathcal{L}(\mathcal{D}) = \mathcal{L}(\mathcal{A}) + \mathcal{L}(\mathcal{C}) + \mathcal{L}(\mathcal{D}|\mathcal{A},\mathcal{C}) = \sum_{k=1}^{K} \mathcal{L}_k(\boldsymbol{A}^{(k)}, C_k), \tag{8}$$

where

$$\mathcal{L}_k(\boldsymbol{A}^{(k)}, C_k) = \log \binom{\binom{N}{2}}{M_k^*} + S_k \log\left(\frac{S}{S_k}\right) + \log\left[\binom{S_k M_k^*}{t_k} \binom{S_k\left(\binom{N}{2} - M_k^*\right)}{f_k}\right] \tag{9}$$

is the cluster-level description length

- looking at the Stirling-approximated versions of the expressions, we can see the intuitive correspondence between the entropies and the average information content of each step
- the compression ratio between the model and the baseline is

$$\eta(\mathcal{D}) = \mathcal{L}_{MDL}(\mathcal{D}) / \mathcal{L}_0(\mathcal{D}), \tag{10}$$

where $\mathcal{L}_{MDL}(\mathcal{D})$ is the minimum description length of $\mathcal{D}$ over all configurations of $\mathcal{A},\mathcal{C}$. If $\eta(\mathcal{D}) < 1$ then the representatives model is effectively compressing the data $\mathcal{D}$, and if $\eta(\mathcal{D}) > 1$ then it is not

- all clusters are decoupled in this objective function, so it can be easily optimized using a merge-split method (since we can ignore terms from all the other clusters when merging/splitting/reassigning). the input is S edge sets for the networks in $\mathcal{D}$, an initial number of clusters K_0, and a maximum number of failed moves n_{fails}. the algorithm is as follows:

1. initialize K_0 clusters $\mathcal{C}$ by placing each network in $\mathcal{D}$ in a cluster chosen uniformly at random
2. for all clusters C_k, create a dict $E_k = \{(i,j) : X_{ij}^{(k)}\}$ mapping all edges (i,j) in networks within C_k to the number of times (i,j) occurs in C_k. we do not need to keep track of the edges that never occur in a cluster.
   - total complexity for all clusters: $O(SN)$ for sparse networks
3. for all clusters C_k, compute the mode $\boldsymbol{A}^{(k)}$ using the following algorithm:
   (a) create list $\tilde{E}_k$, a list of tuples $((i,j), X_{ij}^{(k)})$, with items sorted in increasing order of frequency $X_{ij}^{(k)}$
      - complexity: $O(M_k \log M_k)$, where M_k is the number of unique edges that appear in E_k. I think M_k should scale roughly like $O(N)$, so this step would have complexity $O(N \log N)$, and is the bottleneck of this mode update
   (b) set $r = 0$, $t_k = \sum_{(i,j)} X_{ij}^{(k)}$, $f_k = 0$, $M_k^* = \left|\tilde{E}_k\right|$ $\boldsymbol{A}^{(k)} = \{(i,j) : (i,j) \in E_k\}$
   (c) while True:
      i. set $(i,j), X_{ij}^{(k)} = \tilde{E}_k[r]$
      ii. compute $\Delta\mathcal{L}_{sub} = \mathcal{L}(\boldsymbol{A}^{(k)} \setminus \{(i,j)\}, C_k) - \mathcal{L}(\boldsymbol{A}^{(k)}, C_k)$, which amounts to changing $t_k \to t_k - X_{ij}^{(k)}$, $f_k \to f_k + X_{ij}^{(k)}$, and $M_k^* \to M_k^* - 1$
      iii. if $\Delta\mathcal{L}_{add} < 0$:
         - remove (i,j) from $\boldsymbol{A}^{(k)}$
         - increment r by 1, t_k by $-X_{ij}^{(k)}$, f_k by $X_{ij}^{(k)}$, and M_k^* by -1
      iv. else return $\boldsymbol{A}^{(k)}$
   (d) total complexity for all clusters: roughly $O(KN \log N)$ for sparse networks that are sufficiently clustered
4. initialize $n_f = 0$, compute the total description length $\mathcal{L}(\mathcal{D})$ for the current cluster and mode configurations
5. while $n_f < n_{fails}$, attempt to do one of the following moves at random:
   (a) reassign a randomly chosen network $\boldsymbol{D}^{(s)}$ from its cluster k to the cluster k' that most reduces the description length
      - for all $k' \neq k$, compute

$$\begin{aligned}\Delta\mathcal{L}_1(k') = \mathcal{L}(\boldsymbol{A}^{(k')}, C_{k'} \cup \{\boldsymbol{D}^{(s)}\}) + \mathcal{L}(\boldsymbol{A}^{(k)}, C_k \setminus \{\boldsymbol{D}^{(s)}\}) \\ - \mathcal{L}(\boldsymbol{A}^{(k')}, C_{k'}) - \mathcal{L}(\boldsymbol{A}^{(k)}, C_k)\end{aligned}$$

      - if $\Delta\mathcal{L}_1(k') < 0$ for some $k' \neq k$, then move s to the cluster k' that minimizes $\Delta\mathcal{L}_1(k')$, set n_f to 0, and update E_k, $E_{k'}$, C_k, $C_{k'}$, $\mathcal{L}(\mathcal{D})$ accordingly
      - else, increment n_f by 1 and do nothing
      - complexity: $O(KN)$ for sparse networks, as computing the change in the description length for the move to k' requires comparing the edge list of $\boldsymbol{D}^{(s)}$ and $\boldsymbol{A}^{(k')}$
   (b) merge a random pairs of clusters $C_{k'}$ and $C_{k''}$ to form a new cluster $C_k = C_{k'} \cup C_{k''}$
      - compute E_k by merging $E_{k'}$ and $E_{k''}$

- update $\boldsymbol{A}^{(k)}$ using the method outlined in step 3 of the overall algorithm
- compute $\Delta\mathcal{L}_2 = \mathcal{L}(\boldsymbol{A}^{(k)}, C_k) - \mathcal{L}(\boldsymbol{A}^{(k')}, C_{k'}) - \mathcal{L}(\boldsymbol{A}^{(k'')}, C_{k''})$
- if $\Delta\mathcal{L}_2 < 0$, then add C_k to $\mathcal{C}$, remove $C_{k'}$ and $C_{k''}$ from $\mathcal{C}$, keep E_k, remove $E_{k'}$ and $E_{k''}$, keep $\boldsymbol{A}^{(k)}$, remove $\boldsymbol{A}^{(k')}$ and $\boldsymbol{A}^{(k'')}$, update $\mathcal{L}(\mathcal{D})$
- else, increment n_f by 1 and do nothing
- complexity: roughly $O(N \log N)$, with the bottleneck being the computation of the mode $\boldsymbol{A}^{(k)}$

(c) split a random cluster C_k to form new clusters $C_{k'}$ and $C_{k''}$

- initialize two clusters and their edge dictionaries using the method in steps 1 and 2 of the overall algorithm
- identify the modes of $C_{k'}$ and $C_{k''}$ using the method in step 3
- run K-means style algorithm (with $K = 2$) on $C_{k'}$ and $C_{k''}$, which proceeds as follows while networks are still reassigned:
  i. for all networks $\boldsymbol{D}^{(s)}$ in both clusters, compute $\Delta\mathcal{L}_1(k')$ and $\Delta\mathcal{L}_1(k'')$, and add $\boldsymbol{D}^{(s)}$ to the cluster with the lower of the two values
  ii. update $E_{k'}$ and $E_{k''}$ using the method in step 2
  iii. update the modes of $C_{k'}$ and $C_{k''}$ using the method in step 3
- compute $\Delta\mathcal{L}_3 = \mathcal{L}(\boldsymbol{A}^{(k')}, C_{k'}) + \mathcal{L}(\boldsymbol{A}^{(k'')}, C_{k''}) - \mathcal{L}(\boldsymbol{A}^{(k)}, C_k)$
- if $\Delta\mathcal{L}_3 < 0$, then add $C_{k'}$ and $C_{k''}$ to $\mathcal{C}$, remove C_k from $\mathcal{C}$, keep $E_{k'}$ and $E_{k''}$, remove E_k, keep $\boldsymbol{A}^{(k')}$ and $\boldsymbol{A}^{(k'')}$, remove $\boldsymbol{A}^{(k)}$, update $\mathcal{L}(\mathcal{D})$
- else, increment n_f by 1 and do nothing
- complexity: $O(NS/K)$ for equally sized clusters, with the bottlenecks the initialization step and the reassignment step in the local K-means algorithm

(d) pick two clusters at random to merge then immediately split following steps (b) and (c)

- the greedy mode update should work because when choosing among possible edges to remove from the mode, the best one is the one that produces the least decrease to t_k and increase to f_k, assuming $t_k > S_k M_k^*/2$ and $M_k^* \ll \binom{N}{2}$ (i.e. not the mirrored mode regime). this is because the change to M_k^* will be the same regardless of which edge we choose. the optimal choice of edge is then the one with the lowest $X_{ij}^{(k)}$, since $X_{ij}^{(k)} = -\Delta t_k = \Delta f_k$. also, at some point, $\Delta\mathcal{L}_{add}$ will rise above 0, because the benefit to M_k^* of removing the extra edge will be outweighed by the cost of decreasing t_k. for sparse modes ($M_k^* \ll \binom{N}{2}$), it is never worth having an edge that is not present in $\tilde{E}_k$. this is because the only term that could possible decrease the description length if we add such an edge is the last one, but the addition of an edge to the mode that is not present will have a negligible effect on the numerator and not change the denominator.
- the complexity of the original Gibbs sampling algorithm was $O(NKS)$ per sample (updating the Y's), whereas the complexity of this algorithm is at worst roughly $O(NS)$ per merge-split step (the worst case being a split move for $K = 1$)
- can keep track of attempted merges and splits to reject these in the future if they are proposed multiple times
- We can also derive the MDL objective presented here by taking the heterogeneous network population model with the same group label prior but a separate Bernoulli prior for each mode density. Profiling out the model parameters leaves us with (minus) the description length

we are trying to minimize here. This unifies the Bayesian and MDL formulations of network population clustering.

2 Modifications for different network data

- bipartite networks: just replace $\binom{N}{2}$ with $N_1 N_2$, the product of the number of nodes in each group. this is because the number of edge slots is reduced for a bipartite graph
- frequency-weighted networks: change combinatorics to work for multigraphs. a naive approach could be to just transform $\binom{N}{2} \to M_k^* \binom{N}{2}$, and set M_k^* to now be the sum of the edge weights in mode k
- temporally contiguous clusters: start with all networks s in their own cluster, then repeatedly check all pairs of adjacent clusters (of which there are order $O(K)$ with K clusters currently in the algorithm), merging the pair of adjacent clusters that most reduces the description length. this has complexity $O(SN \log N)$ (for sparse graphs) for the first move, and is significantly faster for subsequent moves ($O(N \log N)$ for sparse graphs) because we can reuse the proposed description length changes for all possible cluster merges except those adjacent to the newly merged cluster

3 Dynamic programming for contiguous clustering

- temporally contiguous clustering can also be done exactly using dynamic programming. letting $\mathcal{L}_{MDL}^{(i)}$ be the minimum description length cluster configuration of the first i (starting with index 0) networks in $\mathcal{D}$, we have that

$$\mathcal{L}_{MDL}^{(j)} = \min_{i \in [0,j]} \left\{ \mathcal{L}_{MDL}^{(i-1)} + \mathcal{L}([i,j]) \right\}, \tag{11}$$

where $\mathcal{L}([i,j])$ is the description length of the cluster of networks with indices $\{i, ..., j\}$ and we use the conventions $\mathcal{L}_{MDL}^{(-1)} = 0$ and $\mathcal{L}_{MDL}^{(0)} = \mathcal{L}([0])$. it is important that we keep S as the same value in the term $S_k \log(S/S_k)$ for the whole algorithm

- we can prove this by induction on S. for the base case, let $S = 1$. we have that

$$\begin{aligned} \mathcal{L}_{MDL}^{(1)} = \min \{ \mathcal{L}([0]) + \mathcal{L}([1]), \mathcal{L}([0,1]) \} &= \min \left\{ \mathcal{L}_{MDL}^{(-1)} + \mathcal{L}([0,1]), \mathcal{L}_{MDL}^{(0)} + \mathcal{L}([1,1]) \right\} \\ &= \min_{i \in [0,1]} \left\{ \mathcal{L}_{MDL}^{(i-1)} + \mathcal{L}([i,1]) \right\}, \end{aligned} \tag{12}$$

and the base case is proven.

- now, suppose the induction hypothesis is true for $1, 2, ..., j$. if $[l, j+1]$ is the rightmost cluster in the optimal configuration that gives rise to $\mathcal{L}_{MDL}^{(j+1)}$, then we have

$$\mathcal{L}_{MDL}^{(j+1)} = \mathcal{L}_{MDL}^{(l-1)} + \mathcal{L}([l, j+1]), \tag{13}$$

and $l \in [0, j+1]$, so we're done.

- in practice, for each j we can iterate over the i values backwards, so then when we compute each new $\mathcal{L}([i,j])$ the dictionary E_k will only need to be updated by iterating over a single new edge set. then each of the $\mathcal{L}([i,j])$ will take $O(N \log N)$ steps to compute (the bottleneck being sorting $\tilde{E}$).
- we then have a complexity of $O(jN \log N)$ for finding $\mathcal{L}_{MDL}^{(j)}$ given all of the previous known values, and since we have to compute this for $j = 1, ..., S$ the total complexity of the whole algorithm is $O(S^2 N \log N)$. this is slower than the greedy merge algorithm, but provably optimal