Course project
"Optimization approaches to community detection"

Marina Danilova, Alexander Podkopaev, Nikita Puchkin, Igor Silin

December 15, 2016

1 Introduction to community detection

Wide range of real-life objects can be represented in terms of graphs. When working with graphs, it may be very useful to find groups of nodes, such that there are much more edges within these groups, than between groups. Such groups are called clusters or communities, and the problem of finding these groups is called community detection. A lot of algorithms for community detection were developed and huge part of them uses optimization method. In this work we want to present a number of methods, that allows to reduce the community detection problem to an optimization problem.

Let's introduce some notations. We consider undirected unweighted graphs without loops with n nodes and m edges. The nodes are enumerated as $\{1, ..., n\}$. Graph is given by its $n \times n$ adjacency matrix A. Degree of the node i is d_i.

The number of clusters is k (some algorithms require to specify this number). The clusters are denoted as $\{\mathcal{C}_1, ..., \mathcal{C}_k\}$. Cluster sizes are $|\mathcal{C}_1|, ..., |\mathcal{C}_k|$. Sometimes it's convenient to describe cluster structure in terms of labeling z: $z(i)$ is the cluster containing node i, i.e. $i \in \mathcal{C}_{z(i)}$.

2 Stochastic block model

In this section we discuss the stochastic block model, which is common widely used tool for generating graphs with special structure. Some of the algorithms that we used (natural conjugate gradients method, semidefinite relaxation method) works exactly with this model.

Again we consider a graph with n nodes. Suppose we want to create some graph with k clusters. Let the structure be given by labeling $z : \{1, ..., n\} \to \{1, ..., k\}$. Also we introduce a symmetric matrix of inter-cluster probabilities $P = ||P_{ij}||_{i,j=1}^{k} \in [0; 1]^{k \times k}$. Elements of this matrix give probability with which an edge between vertex of one cluster and vertex of another or the same cluster is generated.

Using labeling z and inter-cluster probabilities P we can construct SBM parameter matrix θ:

$$\theta_{ij} = P_{z(i)\, z(j)} \ \ \forall i \neq j, \quad \theta_{ii} = 0 \ \ \forall i.$$

Element θ_{ij} means the probability of generating an edge between i-th and j-th nodes. So a graph, or more specifically its adjacency matrix A, is generated in the following way:

$$a_{ij} \sim Bernoulli(\theta_{ij}), \ a_{ji} = a_{ij} \ \ \forall i > j, \quad a_{ii} = 0 \ \ \forall i, \tag{1}$$

where all A_{ij} are independent. In this work we consider unweighted undirected graph without self-loops.

The goal of community detection is to reconstruct labeling z, while also one can be interested in estimation of parameter matrix $\theta = [\theta_{ij}]_{i,j=1}^n$. In order to find this structure, we use only given graph.

As an example, take a look on the simplest case with two clusters ($k = 2$). Let these clusters be of equal size $n/2$. Also suppose that an edge between two nodes of the same cluster appears with probability a while an edge between two nodes of different clusters appears with probability b, where $a > b$. Described case corresponds to the following labeling z and inter-cluster probabilities P:

$$z = [\underbrace{1, ..., 1}_{n/2}, \underbrace{2, ..., 2}_{n/2}]^T, \quad P = \begin{bmatrix} a & b \\ b & a \end{bmatrix}.$$

One can easily construct SBM parameter matrix θ based on z and P. Now the graph generating process is clear.

3 Algorithms

3.1 Semidefinite relaxations

3.2 Natural conjugate gradients method

It is assumed, that latent group labels Z has a multinomial distribution with parameter π and each element of A has a Bernoulli distribution with parameter θ_{ij}:

$$\begin{aligned} &Z_i \sim \text{Poly}(\pi), \quad \pi^T = (\pi_1, \dots, \pi_k), \\ &a_{ij} \sim \text{Bernoulli}(P_{Z(i)Z(j)}), \quad i, j = \overline{1, n}, \end{aligned}$$

A Bayesian approach is used to estimate the most probable configuration of Z given an adjacency matrix A

$$Z^* = \arg\max_Z p(Z|A) = \arg\max_Z \iint p(Z, \pi, P|A) d\pi dP \tag{3.2.2}$$

It treats parameters π and θ_{ij} as random variables with following prior distributions

$$\begin{aligned} &\pi \sim \text{Dirichlet}(\alpha) \\ &P_{ii} \sim \text{Beta}(\beta), \quad i = \overline{1, k}, \end{aligned}$$

where α and β are predefined hyperparameters and P_{ij}, $i \neq j$ are set to equal to a small constant ε. Futhermore, to tackle the intractable integration in 3.2.2 a restriction on the family of factorized distributions is considered

$$\mathcal{Q} = \{q : \, q(Z, \pi, P) = q(\pi)q(P) \prod_i q(Z_i)\},$$

where

$$\begin{aligned} &q(Z_i) : Z_i \sim \text{Poly}(\tilde{\pi}) \\ &q(\pi) : \pi \sim \text{Dirichlet}(\tilde{\alpha}) \\ &q(P) : P_{ii} \sim \text{Beta}(\tilde{\beta}_i) \end{aligned}$$

The optimal distribution $q^* \in \mathcal{Q}$, that approximates the true posterior $p(Z, \pi, P|A)$ minimizes the Kullback-Leibler divergence

$$q^* = \arg\min_{q \in \mathcal{Q}} \text{KL}\left(q \| p(Z, \pi, P|X)\right) \tag{3.2.3}$$

Define

$$\mathcal{L}(q) = \sum_Z \iint q(Z, \pi, P) \log \frac{p(A, Z, \pi, P)}{q(Z, \pi, P)} d\pi dP \tag{3.2.4}$$

Since

$$\mathcal{L}(q) + \mathrm{KL}\left(q \| p(Z, \pi, P|X)\right) = \log p(A)$$

the minimization problem 3.2.3 can be equivalently solved by maximizing $\mathcal{L}(q)$

$$q^* = \arg\max_{q \in \mathcal{Q}} \mathcal{L}(q) \tag{3.2.5}$$

Denote $\tilde{\Pi} = \|\tilde{\pi}_{ij}\|$, $i = \overline{1,n}$, $j = \overline{1,k}$. Given $\tilde{\Pi}$ values of parameters, that maximize $\mathcal{L}(q)$ can be found according to formulas

$$\begin{aligned} \tilde{\alpha} &= \alpha + \tilde{\Pi}^T 1 \\ \tilde{\beta}_i &= \beta + \frac{1}{2}\left(\tilde{\Pi}_{\cdot i}^T A \tilde{\Pi}_{\cdot i}, \tilde{\Pi}_{\cdot i}^T \overline{A} \tilde{\Pi}_{\cdot i}\right)^T, \quad i = \overline{1,k}, \end{aligned} \tag{3.2.6}$$

where 1 denotes an all-ones vector, $\overline{A} = 11^T - I - A$, and $\tilde{\Pi}_{\cdot i}$ stands for the i-th column of the matrix $\tilde{\Pi}$. A corresponding value of log-likelihood is equal to

$$\mathcal{L}(\tilde{\Pi}) = \sum_{i<j} \tilde{\Pi}_{\cdot i}^T A \tilde{\Pi}_{\cdot j} \log \varepsilon + \tilde{\Pi}_{\cdot i}^T \overline{A} \tilde{\Pi}_{\cdot j} \log(1-\varepsilon) - \sum_{i,j} \tilde{\pi}_{ij} \log \tilde{\pi}_{ij} + \log \frac{\mathcal{B}(\tilde{\alpha})}{\mathcal{B}(\alpha)} + \sum_i \log \frac{\mathcal{B}(\tilde{\beta})}{\mathcal{B}(\beta)}, \tag{3.2.7}$$

where $\tilde{\alpha} = \tilde{\alpha}(\tilde{\Pi})$ and $\tilde{\beta} = \tilde{\beta}(\tilde{\Pi})$ can be found from 3.2.6, $\mathcal{B}(\alpha) \triangleq \frac{\Gamma\left(\sum\limits_i \alpha_i\right)}{\prod\limits_i \Gamma(\alpha_i)}$ and Γ is gamma-function. Now the maximization problem 3.2.5 can be reformulated as follows

$$\begin{cases} \mathcal{L}(\tilde{\Pi}) \longrightarrow \max \\ \sum\limits_j \tilde{\pi}_{ij} = 1, \quad i = \overline{1,n} \end{cases} \tag{3.2.8}$$

One can use reparametrization

$$\begin{aligned} &\tilde{\pi}_{ij} = e^{\theta_{ij} - \mathcal{A}_i}, \quad i = \overline{1,n}, j = \overline{1,k-1}, \\ &\tilde{\pi}_{ik} = e^{-\mathcal{A}_i}, \quad i = \overline{1,n}, \\ &\mathcal{A}_i = \log\left(1 + \sum_{j=1}^{k-1} e^{\theta_{ij}}\right), \qquad i = \overline{1,n} \end{aligned} \tag{3.2.9}$$

and obtain a problem of unconstrained maximization

$$\mathcal{L}(\theta) \longrightarrow \max \tag{3.2.10}$$

The problem 3.2.10 can be solved via natural conjugate gradient method. Namely, given an initial value $\theta^{(0)}$, one iteratively finds optimal value of θ as follows

$$\theta^{(t+1)} = \theta^{(t)} + \lambda^{(t)} d^{(t)}, \quad t = 0, 1, 2, \ldots$$

Here $d^{(t)}$ is so called natural conjugate gradient. $d^{(t)}$ can be found according to formulas

$$d^{(t)} = \begin{cases} g^{(t)}, & t = 0 \\ g^{(t)} + \frac{\|g^{(t)}\|_\theta^2}{\|g^{(t-1)}\|_\theta^2} d^{(t-1)}, & t > 0, \end{cases} \tag{3.2.11}$$

where $g^{(t)}$ is a natural gradient of $\mathcal{L}(\theta)$. $\|\cdot\|_\theta$ stands for the norm with respect to Riemannian metrics

$$G(\theta) = \operatorname{diag}\left(\mathcal{I}(\theta_1), \ldots, \mathcal{I}(\theta_N)\right),$$

where $\theta_i = (\theta_{i1}, \ldots, \theta_{ik})^T$, $i = \overline{1,n}$ and (θ_i) is a Fischer information. This method is nothing else but a conjugate gradient method in a Riemannian space with metrics $G(\theta)$. Given θ, g and $\|g\|_\theta$ can be found as follows

$$\begin{aligned} g = \nabla_{\tilde{\Pi}}\mathcal{L}(\tilde{\Pi}) &= \begin{pmatrix} (I,-1)\nabla_{\tilde{\pi}_1}\mathcal{L}(\tilde{\Pi}) \\ \vdots \\ (I,-1)\nabla_{\tilde{\pi}_n}\mathcal{L}(\tilde{\Pi}) \end{pmatrix} \\ \|g\|_\theta^2 &= \sum_{i=1}^{n} \left(\nabla_{\tilde{\pi}_1}\mathcal{L}(\tilde{\Pi})\right)^T \left(\operatorname{diag}(\tilde{\pi}_i) - \tilde{\pi}_i^T\tilde{\pi}_i\right) \left(\nabla_{\tilde{\pi}_1}\mathcal{L}(\tilde{\Pi})\right), \end{aligned} \tag{3.2.12}$$

where $\tilde{\Pi} = \tilde{\Pi}(\theta)$ and

$$\begin{aligned} \frac{\partial\mathcal{L}(\tilde{\Pi})}{\partial\pi_{ij}} = \sum_{l\neq j} \Big(A_{i\cdot}\tilde{\Pi}_{\cdot l}\log\varepsilon + \overline{A}_{i\cdot}\tilde{\Pi}_{\cdot j}\log(1-\varepsilon)\Big) + \\ \Big(A_{i\cdot}\tilde{\Pi}_{\cdot j}, \overline{A}_{i\cdot}\tilde{\Pi}_{\cdot j}\Big)\nabla\log\mathcal{B}(\tilde{\beta}_j) + \psi(\tilde{\alpha}_j) - \log\tilde{\pi}_{ij} - 1, \end{aligned} \tag{3.2.13}$$

where $\psi(\cdot)$ is digamma function.

The final algorithm is given in 1.

Algorithm 1: Natural Conjugate Gradient

Input: adjacency matrix A, maximum number of clusters k, tolerance η, maximum number of iterations $t_{\max}$
Output: array of predicted labels Z
Initialize θ;
$\mathcal{L}_{old} \leftarrow -\infty$;
$\lambda \leftarrow 1$;
for *t in range ($t_{\max}$)* **do**
  Calculate $\tilde{\Pi}$ using 3.2.9;
  Calculate $\tilde{\alpha}$, $\tilde{\beta}$ using 3.2.6;
  Calculate $\mathcal{L}$ using 3.2.7;
  if $0 \leq \frac{\mathcal{L}-\mathcal{L}_{old}}{|\mathcal{L}|} < \eta$ **then**
    break
  if $\eta \geq 0$ **then**
    Update d using 3.2.11, 3.2.12, 3.2.13;
    $\theta_{old} \leftarrow \theta$
    $\theta \leftarrow \theta_{old} + \lambda d$;
    $\mathcal{L}_{old} \leftarrow \mathcal{L}$;
  else
    $\lambda \leftarrow \frac{\lambda}{2}$
    $\theta \leftarrow \theta_{old} + \lambda|\eta|d$
  end
end
$Z = \left(\arg\max_{1\leq j\leq k}\tilde{\pi}_{1j}, \ldots, \arg\max_{1\leq j\leq k}\tilde{\pi}_{nj}\right)$
return Z

Initial value of θ was generated from a standart normal distribution $\mathcal{N}(0, 1)$. Probability of occurence an inter-cluster edge ε was set to 10^{-10}. The maximal number of iterations and relative tolerance were taken equal to 100 and 10^{-6} respectively. Examples of performance of the natural conjugate gradient method can be found, for example, in [1].

3.3 Modularity-based methods

In this subsection we discuss the method described in [2]. The method is based on the concept of modularity. To introduce it, we first of all compute the fraction of edges which lie within communities:

$$\frac{1}{2m}\sum_{i,j=1}^{n} a_{ij} \cdot \mathbb{1}\{z(i) = z(j)\}.$$

We are also interested in the expected number with respect to the configuration model. The configuration model is a randomized realization of a particular network. Given a network with n nodes, where each node i has a node degree d_i, the configuration model cuts each edge into two halves, and then each half edge, called a stub, is rewired randomly with any other stub in the network even allowing self loops. Thus, even though the node degree distribution of the graph remains intact, the configuration model results in a completely random network and the expected fraction of edges which lie within communities:

$$\sum_{i,j=1}^{n} \frac{d_i}{2m}\frac{d_j}{2m} \cdot \mathbb{1}\{z(i) = z(j)\}.$$

Finally, **Modularity** is the difference between two previous fractions:

$$Q(z) = \frac{1}{2m}\sum_{i,j=1}^{n} \left(a_{ij} - \frac{d_i \cdot d_j}{2m}\right) \cdot \mathbb{1}\{z(i) = z(j)\}.$$

Modularity can take values in interval $[-1/2;\ 1)$. When modularity is large, it means that the network is far from its average state and there the density of edges within communities is higher. But finding exact maximum of modularity is NP-hard problem because we need to search through exponential number of clusterings. So, the goal of modularity-based algorithms is to approximately maximize modularity over all possible partitions of the graph.

Different ideas for maximization can be used, but we focus on simple greedy algorithm. To apply this method, it will be more convinient for us to rewrite the definition of modularity in the following form:

$$Q = \sum_{q=1}^{k} e_{\mathcal{C}_q\mathcal{C}_q} - b_{\mathcal{C}_q}^2,$$

where

$$e_{\mathcal{C}_q\mathcal{C}_p} = \frac{1}{2m}\sum_{i,j=1}^{n} a_{ij} \cdot \mathbb{1}\{i \in \mathcal{C}_q, j \in \mathcal{C}_p\}$$

and

$$b_{\mathcal{C}_q} = \sum_{p=1}^{k} e_{\mathcal{C}_q\mathcal{C}_p}.$$

One can easily check that this formulation is equivalent to the previous one.

Now let's describe the idea of the greedy maximization of the modularity for given graph. Initially we put each node in its own cluster, i.e. $\forall i \to \mathcal{C}_i^{(0)} = \{i\}$. Then we start our iterations. On t-th iteration we look at current clusters $\mathcal{C}_q^{(t-1)}$ and look for a pair of clusters, union of which gives us the maximal gain ΔQ of modularity. Namely, if we join clusters $\mathcal{C}_q^{(t-1)}$ and $\mathcal{C}_p^{(t-1)}$ together and form new cluster $\mathcal{C}_{(qp)}^{(t)}$ and leave all other clusters as they are, i.e. $\mathcal{C}_s^{(t)} = \mathcal{C}_s^{(t-1)}\ \forall s \neq q, p$, then e and b can be recalculated as follows:

$$
\begin{aligned}
&e_{\mathcal{C}_{(qp)}^{(t)}\mathcal{C}_s^{(t)}} = e_{\mathcal{C}_q^{(t-1)}\mathcal{C}_s^{(t-1)}} + e_{\mathcal{C}_p^{(t-1)}\mathcal{C}_s^{(t-1)}} \quad \forall s \neq q, p,\\
&e_{\mathcal{C}_{(qp)}^{(t)}\mathcal{C}_{(qp)}^{(t)}} = e_{\mathcal{C}_q^{(t-1)}\mathcal{C}_q^{(t-1)}} + e_{\mathcal{C}_p^{(t-1)}\mathcal{C}_p^{(t-1)}} + e_{\mathcal{C}_q^{(t-1)}\mathcal{C}_p^{(t-1)}} + e_{\mathcal{C}_p^{(t-1)}\mathcal{C}_q^{(t-1)}},\\
&b_{\mathcal{C}_{(qp)}^{(t)}} = b_{\mathcal{C}_q^{(t-1)}} + b_{\mathcal{C}_p^{(t-1)}}.
\end{aligned}
$$

All other elements don't change. Hence, according to our formula for modularity we have:

$$
\begin{aligned}
Q^{(t)} &= Q^{(t-1)} + \Delta Q,\\
\Delta Q &= e_{\mathcal{C}_q^{(t-1)}\mathcal{C}_p^{(t-1)}} + e_{\mathcal{C}_p^{(t-1)}\mathcal{C}_q^{(t-1)}} - 2b_{\mathcal{C}_q^{(t-1)}} b_{\mathcal{C}_p^{(t-1)}}.
\end{aligned}
$$

If we keep and recalculate all e and b during the iterations, we can determine pair of current clusters that gives maximal ΔQ effectively, since for any possible pair calculating ΔQ takes constant time (actually, more advanced approaches for more effective search of such pair were developed, e.g. one can use heap structure). So, we choose two clusters that we want to unite and unite them. We also have the value of modularity for current clustering. We also need to recalculate e and b as shown above.

One can consider this sequential uniting of clusters as the process of building a dendrogarm, which is a tree, that shows in which order cluster were joined.

After we did $(n-1)$ iterations, we have one cluster that contains all nodes. So as a final clustering we need to choose clustering from iteration t with the maximal modularity $Q^{(t)}$.

For technical details of implementation see code in "Lib/modularity.py".

Advantages of this procedure are that it gives some partition in any case and doesn't require to specify the number of clusters that we are looking for. It's good, because in real-world problems we rarely know the exact number of communities that we want to detect. At the same time, there are no theoretical results that guarantee some good performance of the method.

3.4 Spectral method

4 Data

In our experiments we consider standard real-world graphs that are often used for testing community detection algorithms, such as "Zachary's karate club", "American college football", "Books about US politics". For this graphs we know true clusters. The description of these graphs can be found in internet.

We have generated several artificial graphs using stochastic block model as well.

5 Experimental results

We launch all described algorithms on our graphs and compare their performance using different metrics, such as: NMI, F1-score, Ratio cut, Modularity.

6 Work split

References

[1] A fast inference algorithm for stochastic blockmodel, Zhiqiang Xu, Yiping Ke, Yi Wang, 2014.

[2] Fast algorithm for detecting community structure in networks, M.E.J.Newman, 2003.