Tutorial: Inference in the Stochastic Block Model

MSc in Statistics for Smart Data – Introduction to graph analysis and modeling

Julien Chiquet, November the 15th, 2018

Preliminaries

Goals.

1. Random graphs generation: Erdös-Rényi, Stochastic Block Models
2. Variational inference for the stochastic block model
3. Analysis of some real world network

Instructions. Each student *must* send an `R markdown` report generated via `R studio` to julien.chiquet@inra.fr at the end of the tutorial. This report should answer the questions by commentaries and codes generating appropriate graphical outputs. A cheat sheet of the markdown syntax can be found here.

Required packages. Check that the following packages are available on your computer:

```
library(igraph)
library(sand)
library(Matrix)
library(devtools)
library(blockmodels)
library(mixer)
library(aricode)
```

You also need `Rstudio`, LaTeX and packages for markdown:

```
library(knitr)
library(rmarkdown)
```

If(and only if!!) you have time, you can also play with

```
library(ggraph)
```

1 Background

1.1 Notations

We let $\mathcal{P} = \{1, \dots, p\}$ be a set of nodes. Presence or absence of an edge between two nodes i and j is described by the random variable $X_{ij} = \mathbf{1}_{\{i \leftrightarrow j\}}$. We assume by convention that

the nodes are not connected to themselves, that is, $X_{ii} = 0$ for all $i \in \mathcal{P}$.

1.2 Stochastic Block Model

This model has several representation. We adopt the one given by Daudin, Picard and Robin (2007), known as "mixture model for random graphs". This model spreads the nodes among a set of Q classes $\mathcal{Q} = \{1, \ldots, Q\}$ with *a priori* distribution $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_Q)$. The hidden random indicator variables $(Z_{iq})_{i \in \mathcal{P}, q \in \mathcal{Q}}$ define the classes each node belongs to. Thus

$$\alpha_q = \mathbb{P}(Z_{iq} = 1) = \mathbb{P}(i \in q), \quad \text{such that} \sum_q \alpha_q = 1. \tag{1}$$

It is straightforward to see that $\mathbf{Z}_i = (Z_{i1}, \ldots, Z_{iQ})$ has a multinomial distribution

$$\mathbf{Z}_i \sim \mathcal{M}(1, \boldsymbol{\alpha}). \tag{2}$$

Finally, let $\pi_{q\ell}$ be the probability that a node in class q connects to a node in class ℓ[1]. The probability for having edge between nodes i and j is defined *conditionally on* the classes they belong to:

$$X_{ij} | \{i \in q, j \in \ell\} \sim \mathcal{B}(\pi_{q\ell}), \quad i \neq j. \tag{3}$$

To sum up, the parameters are

- $\mathbf{X} = (X_{ij})$, the $p \times p$ adjacency matrix of the graph,
- $\boldsymbol{\pi} = (\pi_{q\ell})$ the $Q \times Q$ connectivity matrix,
- $\boldsymbol{\alpha} = (\alpha_q)$, the size-$Q$ vector of class proportions.

1.2.1 Useful quantities in the variational EM

During this practical, you will implement the variational EM (VEM) algorithm studied during this morning lecture. Here are the expressions of the key quantities that you need to compute along the algorithm.

1.2.1.1 Variational lower bound.

The variational lower bound of the loglikelihood maximized by the VEM is

$$J(\boldsymbol{\tau}, \boldsymbol{\pi}, \boldsymbol{\alpha}) = \sum_{i,q} \tau_{iq} \log \alpha_q + \sum_{i<j,q,\ell} \tau_{iq}\tau_{j\ell} \log b(X_{ij}; \pi_{q\ell}) - \sum_{i,q} \tau_{iq} \log(\tau_{iq}),$$

where $b(x; \pi) = \pi^x (1-\pi)^{1-x}$ is the probability density function of the Bernoulli distribution and τ_{iq} are the posterior probabilities for class belonging, aka the variational parameters.

1.2.1.2 M step.

For fixed values of $\hat{\tau}_{iq}$ the estimators for α_q and $\pi_{q\ell}$ by maximizing the conditional expectation are

$$\hat{\alpha}_q = \frac{1}{n} \sum_i \hat{\tau}_{iq}, \quad \hat{\pi}_{q\ell} = \frac{\sum_{i \neq j} \hat{\tau}_{iq}\hat{\tau}_{j\ell} X_{ij}}{\sum_{i \neq j} \hat{\tau}_{iq}\hat{\tau}_{j\ell}}. \tag{4}$$

[1] Since the network is undirected, the matrix $\mathbf{X}$ is symmetric and so $\pi_{q\ell} = \pi_{\ell q}$.

1.2.1.3 E step.

The variational parameters τ_{iq} verify the following fixed point relation:

$$\hat{\tau}_{iq} \propto \alpha_q \prod_j \prod_\ell b(X_{ij}, \pi_{q\ell})^{\hat{\tau}_{j\ell}} \tag{5}$$

1.2.1.4 Integrated complete likelihood criterion (ICL).

The variational ICL used to compare models with different numbers of clusters is

$$\text{vICL}(Q) = \sum_{i,q} \hat{Z}_{iq} \log \hat{\alpha}_q + \sum_{i<j,q,\ell} \hat{Z}_{iq}\hat{Z}_{j\ell} b(X_{ij}; \hat{\pi}_{q\ell}) - \frac{1}{2}\left(\frac{Q(Q+1)}{2} \log \frac{n(n-1)}{2} + (Q-1)\log(n)\right)$$

where $\hat{Z}_{iq}$ is the maximum a posteriori associated to the estimated probability $\hat{\tau}_{iq}$.

2 Simulations

2.1 Random graph generation.

Write a function which takes the parameters p, α, π for arguments and draw a random graph encoded as an igraph object. The vertices must own an attribute for their class membership. You may rely on the function `igraph::sample_sbm` which does almost all the job. Here is a skeleton for your own function:

```
rNetwork <- function(n, pi, alpha = c(1)) {
### fill this up
}
```

2.2 Examples

Draw some graphs with the following topologies, by choosing :

- Erdös-Rényi random networks,
- Affiliation (or community) networks, i.e. with clusters of highly connected nodes,
- Star networks, i.e. networks where a few nodes are highly connected to the others, the later being unconnected with each other.

These networks can be represented via an image of the corresponding adjacency matrix (try the `image` method of the **Matrix** package), or thanks to the `R`-package **igraph**.

3 Variational Inference in the Stochastic block model

Let us now write our own algorithm to fit an SBM on a `igraph` object. To do so, I give you the main algorithm here after. You just need to fill the functions `E_step`, `M_step`, `get_J` and compute the variational ICL and BIC.

```
VEM_SBM <- function(G, Q, cl.init, nstart=5, eps=1e-5, maxIter=50){
## Initialization
stopifnot(is.igraph(G))
n <- gorder(G)
J <- vector("numeric", maxIter)
zero <- .Machine$double.eps
### variational lower bound
get_J <- function(theta, tau){
## fill up below
## ....
## fill up above
J
}
### M step: update theta (pi and alpha)
M_step <- function(tau){
## fill up below
## ....
## fill up above
alpha[alpha < zero] <- zero
pi[pi < zero] <- zero
pi[pi > 1- zero] <- 1- zero
list(pi = pi, alpha = alpha)
}
### E step: update the clustering parameters (tau)
E_step <- function(theta, tau){
## fill this up
tau
}
VEM <- function(tau) {
cond <- FALSE; iter <- 0
while(!cond){
4iter <- iter +1
theta <- M_step(tau)
# E step
tau <- E_step(theta, tau) # M step
J[iter] <- get_J(theta, tau) # assess convergence
if (iter > 1)
cond <- (iter > maxIter) | (abs((J[iter] - J[iter-1])) < eps)
}
return(list(theta = theta, tau = tau, J = J[1:iter]))
}
if (missing(cl.init)) {
out <- replicate(nstart, {
cl0 <- sample(1:Q, n, replace = TRUE)
tau <- matrix(0,n,Q); tau[cbind(1:n, cl0)] <- 1
VEM(tau)
})
```

```
best <- out[,which.max(sapply(out['J',], max))]
} else {
tau <- matrix(0,n,Q); tau[cbind(1:n, cl.init)] <- 1
best <- VEM(tau)
}
## FILL THIS UPS FOR THE BEST MODEL
# vBIC <-
# vICL <-
return(list(theta = best$theta,
tau = best$tau, membership = apply(best$tau, 1, which.max),
J = best$J, vICL = vICL, vBIC= vBIC))
}
```

4 Numerical experiments

4.1 Simple tests

Use the `rNetwork` function to generate some SBM with various structures. Show that you can recover the original parameter thanks to the **mixer** R function. Also train yourself on the corresponding `getModel` and `plot` functions. Now check that your own VEM function is also working. You can initialize the clusters with the spectral clustering function developed during the first tutorial.

4.2 Numerical study: real worl network analysis

Consider a network of your own or one found in the **sand** package, for instance the French political blogosphere. Symmetrize the network and remove the isolated nodes. Then, apply the variational EM. Chose the number of classes with the vICL. Comment.

4.3 Simulations 1: assessing parameter estimation.

Consider for instance an affiliation network with 3 classes. Do some simulations showing that the mean square error of $\hat{\pi}_{q\ell}$ decrease when n increases. To do so, assume that you know the correct number of classes in your graph.

4.4 Simulations 2: assessing clustering efficiency.

Again, consider an affiliation network with 3 classes. Do some simulations showing that the adjusted rand index between the classification recovered by the VEM, when the number of classes is chosen so as to maximize the ICL, increases when n increases. You may use the function `ARI` from the **aricode** package to compare two classifications.