Stochastic bloc model

Christophe Ambroise

10/7/2020

Algorithme de simulation

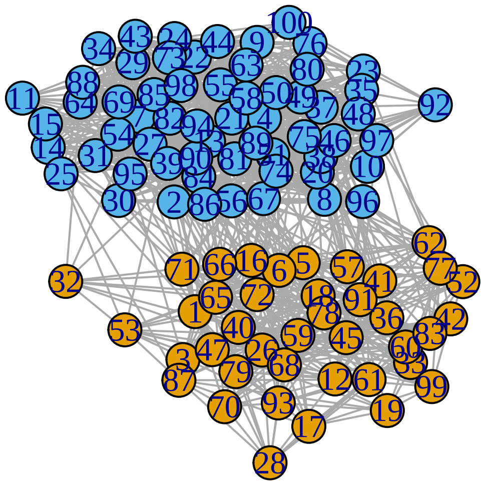
```r
class.ind<-function (cl)
{
    n <- length(cl)
    cl <- as.factor(cl)
    x <- matrix(0, n, length(levels(cl)))
    x[(1:n) + n * (unclass(cl) - 1)] <- 1
    dimnames(x) <- list(names(cl), levels(cl))
    x
}
graph.affiliation<-function(n=100,Pi=c(1/2,1/2),alpha=0.7,beta=0.05) {
      # INPUT  n: number of vertex
      #           Pi : vecteur of class proportion
      #           alpha: proba of edge given  same classe
      #           beta: proba of edge given two different classes
      # OUTPUT x: adjacency matrix
      #        cluster: class vector
      #

      X<-matrix(0,n,n);
      Q<-length(Pi);
      rmultinom(1, size=n, prob = Pi)->nq;
      Z<-class.ind(rep(1:Q,nq));
      Z<-Z[sample(1:n,n),];
      for (i in 1:n)
        for (j in i:n)
            {
            # if i and j in same class
            if (which.max(Z[i,])  == which.max(Z[j,])) p<-alpha else  p<-beta
            if ((rbinom(1,1,p))&(i != j)) {X[i,j]<-1; X[j,i]<-1}
            }
       return(list(X=X,cluster=apply(Z,1,which.max)) )
  }

mygraph<-graph.affiliation(alpha=0.35,beta=0.05)
library(igraph)
```

```
##
## Attachement du package : 'igraph'

## Les objets suivants sont masqués depuis 'package:stats':
```

```
##
##     decompose, spectrum
```

```
## L'objet suivant est masqué depuis 'package:base':
##
##     union
```

```
plot(graph_from_adjacency_matrix(mygraph$X,mode="undirected"),vertex.color=mygraph$cluster)
```



Algorithme variationel pour un modèle SBM d'affiliation

```
VEMaffiliation<-function(X,Q=2,equal.proportions=TRUE,nbstart=10,max.iter=100,epsilon=1e-6){
######################################################################
## DESPCRIPTION:
##  VEMaffiliation implements the estimation of an MixNet affiliation model via a variationnal EM
##
##
## INPUT:
##   X : a binary adjacency matrix
##   Q : number of classes
##   max.iter: maximum number of iterations
##   nbstart: number of different startpoint
##   epsilon: precision of the convergence
## OUTPUT:
##  res: a list with following items
##  parameters: pi, alpha and beta which are vector of proportions, intra and inter probabilities
##  Tau: matrix of posterior probabilities
##  J: approximated log-likelihood
## EXAMPLE:
##
## n<-300
## pi<-c(0.3,0.3,0.4)
## alpha<-0.1
## beta<-0.02
## G<- rMixNetAffiliation(n,pi,alpha,beta)
## res<-VEMaffiliation(G$M,Q=3,nbstart=10,max.iter=50)
######################################################################
```

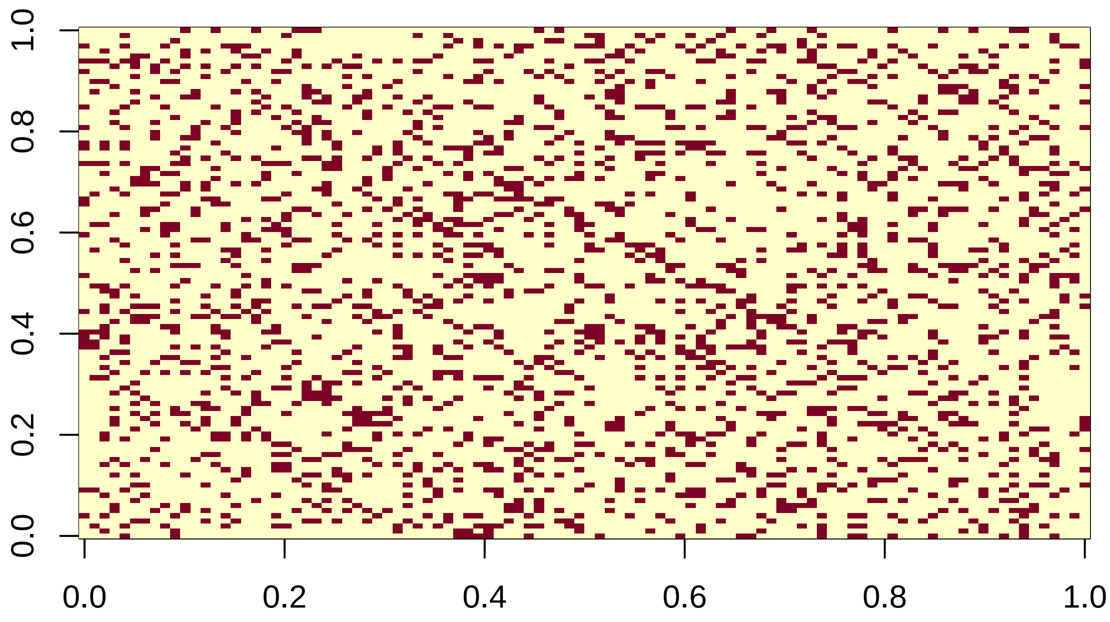
```
# Initiatisation of the parameters
parameters<-list(alpha=0.1,beta=0.1,pi=rep(1/Q,Q))

bestJ <- -Inf
for (run in 1:nbstart){
# Initialisation of the partition
  initialisation<-kmeans(X,Q,nstart=5)
  Tau<-class.ind(initialisation$cluster)

  J <- -Inf
  for (it in 1:max.iter){
    J.old<-J

    resM<- Mstep(X,Tau,parameters,equal.proportions=equal.proportions)
    parameters<-resM$parameters

    resE<-  VEstep(X,Tau,parameters)
    Tau<-resE$Tau

    J<-resE$J
     if (abs(J-J.old)< epsilon) break
    }
    if (J > bestJ) {
      bestJ<-J
      bestParameters <- resM$parameters
      bestTau <- resE$Tau}
}
return(list(Tau=bestTau,parameters=bestParameters,J=bestJ))
}




Mstep <- function(X, Tau,equal.proportions=TRUE, parameters=NULL) {
  ## INTIALIZE
  eps<-1e-6
  Q       <- ncol(Tau) # num classes
  n       <- nrow(Tau) # num nodes
  nql   <- matrix(0,Q,Q)
  theta.ijql <- matrix(0,n,n)
  u.trig     <- upper.tri(X)
  v.trig     <- upper.tri(nql)

  nodiag     <- upper.tri(X) | lower.tri(X)

   for (q in 1:Q) {

    for (l in q:Q) {

        ##  intermediary calculation
        theta.ijql <- Tau[,q] %*% t(Tau[,l])
        diag( theta.ijql)<-0
        ## nql is the number of edges between class q and class l
        nql[q,l] <-   sum(theta.ijql*X)
    } # next l

  } # next q
  nq<-colSums(Tau)

  if (equal.proportions==TRUE)
    pi<- rep(1/Q,Q)
  else {
    pi<-nq/n
  }
  nqnl<- cbind(nq) %*% rbind(nq)
  beta<- max(sum(nql[v.trig]) / sum(nqnl[v.trig]),eps)
  alpha <- max(sum(diag(nql))/sum(nq*(nq-1)),eps)
  J <- JRx(Tau, X, parameters)
return(list(parameters=list(pi=pi,alpha=alpha,beta=beta),J=J))
}



 VEstep <- function(    X,
           Tau,
           parameters) {

 # Variationnal E step of the EM algorithm for Bernoulli MixNet Model
 # INPUT
 #  X :  adjacency matrix
 #  Tau : Matrix of conditionnal probabilities
 #
 # OUTPUT
 #  Tau

FP.maxIt    = 50
eps     = 1e-6
verbose     = FALSE


  if (verbose==TRUE) {
    cat("\n     - Fixed point resolution... ")
  }

  ## =====================================
  ##        INITIALIZING
  n <- nrow(Tau) # num nodes
  Q <- ncol(Tau) # num classes

  pi<-parameters$pi
  logpi  <- pmax(log(pi),log(.Machine$double.xmin))

  beta <- parameters$beta
  alpha  <- parameters$alpha
```

```
alpha.ql<-matrix(beta,Q,Q)
diag(alpha.ql)<- alpha

ones        <- cbind(rep(1, n))
class.names <- colnames(Tau)

## =====================================
##   SOLVING TAU WITH FIXED POINT ALGO

## convergence setup
J        <- -Inf # JRx criterium
E.Delta  <- Inf # convergence deltas
E.Deltas <- c()
E.It     <- 0 # iterations
convergence <- list( JRx.pb=FALSE, Iteration.pb=FALSE, Converged=FALSE, Pb=FALSE)

## convergence loop
if (verbose) { cat(" iterate: ") }


while ( (convergence$Pb==FALSE) && (convergence$Converged==FALSE) ) {

  E.It    <- E.It+1
  Tau.old <- Tau
  J.old   <- J

  if (verbose) { cat(" ",E.It) }

  ## prep current estimate of log(Tau)
  logTau <- matrix(0,n,Q)

  for (q in 1:Q) {
    for (l in 1:Q) {

      ## normal Lagrangians
      Beta.ijql <-  X * log(alpha.ql[q,l]) + (1-X) * log(1-alpha.ql[q,l])
      diag(Beta.ijql) <- 0
      logTau[,q] <- logTau[,q] + apply(((ones %*% Tau[,l]) * Beta.ijql), 1, sum )

    } # next l
  } # next q

  ## Normalizing in the log space to avoid numerical problems
  logTau <- logTau - apply(logTau,1,max)

  ## Now going back to exponential with the same normalization
  Tau <- (matrix(1,n,1) %*% pi) * exp(logTau)
  Tau <- pmin(Tau,.Machine$double.xmax)
  Tau <- pmax(Tau,.Machine$double.xmin)
  Tau <- Tau / (rowSums(Tau) %*% matrix(1,1,Q))
  Tau <- pmin(Tau,1-.Machine$double.xmin)
  Tau <- pmax(Tau,.Machine$double.xmin)
```

```r
    ## JRx criterium
    J <- JRx(Tau, X, parameters)

     # convergence pb : JRx decreases
    convergence$JRx.pb <- (J < J.old)
    if (convergence$JRx.pb==TRUE) {
      Tau <- Tau.old
    }

    ## Delta criterium
    E.Delta <- J - J.old
    E.Deltas[E.It] <- E.Delta

    ## convergence ?
    convergence$Iteration.pb  <- (E.It      > FP.maxIt) # have we hit iter.max ?
    convergence$Converged     <- (abs(E.Delta)  < eps    ) # has the algo converged ?
    convergence$Pb <- ( convergence$Iteration.pb  || convergence$JRx.pb)

  } # repeat till convergence or itMax

  ## probabilistic class estimation
  colnames(Tau) <- class.names

  ## check non-convergence
  if ( convergence$Pb==TRUE ) {
    if (verbose) {
      cat("   can't enhance the criteria anymore...\n" )
    }
  }

  return(list(Tau=Tau,J=J))
}


JRx <- function (Tau, X, parameters) {


beta   <- parameters$beta
alpha  <- parameters$alpha
pi     <-parameters$pi
Q<-length(pi)
alpha.ql      <- matrix(beta, Q,Q)
diag(alpha.ql)<- alpha


  if (is.null(parameters)) {
    cat("\n JRx : WARNING : null parameters argument, returning NULL !!!\n")
    return(NULL)
  }

  n      <- dim(Tau)[1]
  Q      <- dim(Tau)[2]
  u.tri  <- upper.tri(X)

    ## Doit augmenter au cours des it??rations

    logTau <- pmax(log(Tau),log(.Machine$double.xmin))
    logpi  <- pmax(log(pi),log(.Machine$double.xmin))

    ## Les 2 premiers termes de J...
    J <- - sum( Tau * logTau ) + sum ( Tau %*% logpi )

    eps <- .Machine$double.xmin
    ## ... et le 3??me
    for (q in 1:Q) {

      for (l in 1:Q) {

          lnf.ijql <-   X * log(alpha.ql[q,l]) + (1-X) * log(1-alpha.ql[q,l])
          tau.ijql <- Tau[,q] %*% t(Tau[,l])

          J <- J + sum( tau.ijql[u.tri] * lnf.ijql[u.tri] )

      } # next l
    } # next q


  return(J)
}
```

Simulation suivie d'une estimation de la structure cachée (Z_{est})

```
mygraph<-graph.affiliation(alpha=0.35,beta=0.05)
X=as.matrix(mygraph$X)
image(X)
```



```
res.VEM<-VEMaffiliation(X,Q=2,equal.proportions=TRUE,nbstart=10,max.iter=100,epsilon=1e-6)
print(res.VEM$parameters)
```

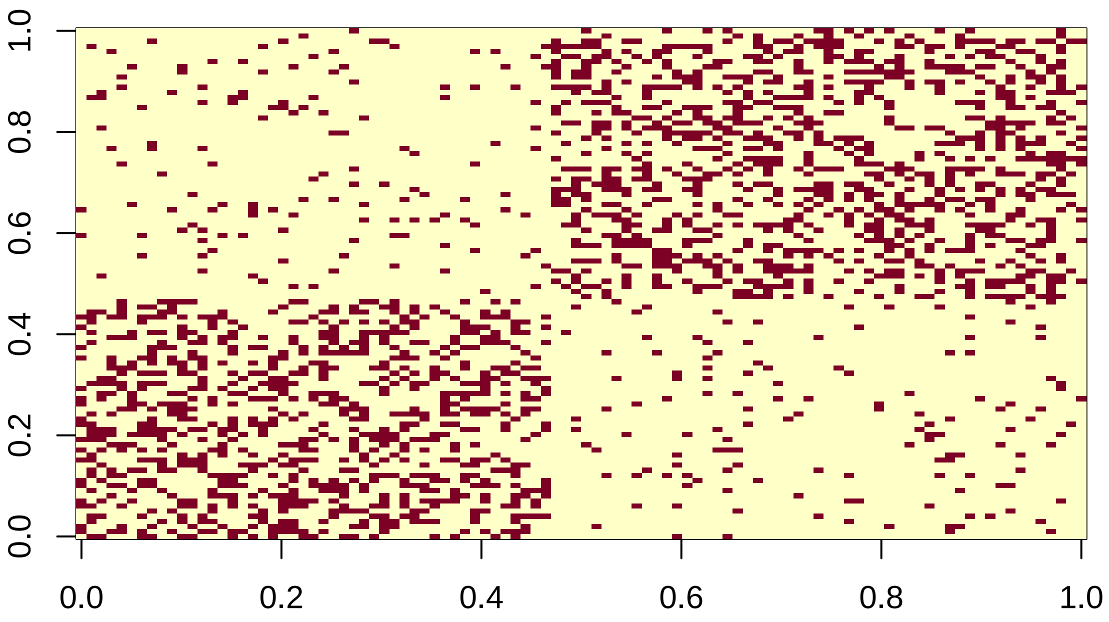
```
## $pi
## [1] 0.5 0.5
##
## $alpha
## [1] 0.3546157
##
## $beta
## [1] 0.054998
```

La structure cachée estimée est comparée à la structure cachée réelle.

```
Zest<-apply(res.VEM$Tau,1,which.max)
table(mygraph$cluster,Zest)
```

```
##    Zest
##      1  2
##   1  0 53
##   2 47  0
```

```
image(X[order(Zest),order(Zest)])
```


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0