Statistics and Its Interface Volume 0 (2018) 1

Supplement to "Bayesian Modeling and Uncertainty Quantification for Descriptive Social Networks" [*]

Thomas Nemmers, Anjana Narayan, and Sudipto Banerjee,

R CODE

To use MCMC to sample from the posterior distributions, the R programming language and rjags package were used. The code for model 1 is:

```
sink("model1.txt")
cat("
model
{
  for ( i in 1:N ) {
    for ( j in 1:M ) {
      y[i , j] ~ dbern(pi[i , j])
      logit(pi[i , j]) = beta0 + (beta1 * X1[i])
        + (beta2 * X2[j])

      Like[i , j] = (pi[i,j]^y[i,j])*((1-pi[i,j])^(1-y[i,j]))
    }
  }

  beta0 ~ dnorm(0 , 0.000001)
  beta1 ~ dnorm(0 , 0.000001)
  beta2 ~ dnorm(0 , 0.000001)
}
  " , fill = T)
sink()

X = c(rep(1 , 8) , rep(0 , 10))
modData1 = list(N = nrow(adj.mat) ,
  M = ncol(adj.mat) , y = adj.mat , X1 = X , X2 = X)
params1 = c("beta0" , "beta1" , "beta2" , "pi" , "Like")
inits1 = list(list(beta0 = 0 , beta1 = 0 , beta2 = 0))
nIter = 10000
model1 = jags.model("model1.txt" , data = modData1 ,
  inits = inits1 , n.chains = 1)
samples1 = jags.samples(model1 ,
  variable.names = params1 , n.iter = nIter)
```

adj.mat is a matrix object that is equal to the adjacency matrix for the network.

The code for model 2 is:

```
sink("model2.txt")
cat("
model
{
  for ( i in 1:N ) {
    for ( j in 1:M ) {
      y[i , j] ~ dbern(pi[i , j])
      logit(pi[i , j]) = inprod(beta[] , c(1 , X1[i] , X2[j]))

      Like[i , j] = (pi[i,j]^y[i,j])*((1-pi[i,j])^(1-y[i,j]))
    }
  }

  Omega ~ dwish(R , 3+1)
  Sigma = inverse(Omega)

  beta ~ dmnorm(c(0 , 0 , 0) , Omega)
}
  " , fill = T)
sink()

X = c(rep(1 , 8) , rep(0 , 10))
i3 = matrix(c(1 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 1) , nrow = 3)
modData2 = list(N = nrow(adj.mat) ,
  M = ncol(adj.mat) , y = adj.mat , X1 = X , X2 = X ,
  R = i3)
params2 = c("beta" , "Sigma" , "pi" , "Like")
inits2 = list(list(beta = rep(0 , 3) , Omega = i3))
nIter = 10000
model2 = jags.model("model2.txt" , data = modData2 ,
  inits = inits2 , n.chains = 1)
samples2 = jags.samples(model2 ,
  variable.names = params2 , n.iter = nIter)
```

For model 3:

```
sink("model3.txt")
cat("
model
{
  for ( i in 1:N ) {
    for ( j in 1:M ) {
      y[i , j] ~ dbern(pi[i , j])
      logit(pi[i , j]) = inprod(beta[] , c(1 , X1[i] , X2[j] ,
```

[*]People we want to thank

```
          X1[i]*X2[j]))

        Like[i , j] = (pi[i,j]^y[i,j])*((1-pi[i,j])^(1-y[i,j]))
      }
    }

    Omega ~ dwish(R , 4+1)
    Sigma = inverse(Omega)

    beta ~ dmnorm(c(0 , 0 , 0 , 0) , Omega)
  }
    " , fill = T)
  sink()

  i4 = matrix(c(1 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 1 , 0 , 0 ,
    0 , 0 , 1) , nrow = 4)
  modData3 = list(N = nrow(adj.mat) ,
    M = ncol(adj.mat) , y = adj.mat , X1 = X , X2 = X ,
    R = i4)
  inits3 = list(list(beta = rep(0 , 4) , Omega = i4))
  model3 = jags.model("model3.txt" , data = modData3 ,
    inits = inits3 , n.chains = 1)
  samples3 = jags.samples(model3 ,
    variable.names = params2 , n.iter = nIter)
```

For model 4:

```
  sink("model4.txt")
  cat("
  model
  {
    for ( i in 1:N ) {
      pi[i , 1:N] ~ ddirch(alpha)

      for ( j in 1:M ) {
        y[i , j] ~ dbern(pi[i , j])

        Like[i , j] = (pi[i,j]^y[i,j])*((1-pi[i,j])^(1-y[i,j]))
      }
    }

    for ( i in 1:N ) {
      mu[i] = inprod(beta[] , X[i ,])
      numerator[i] = ifelse(i == 1 , 1 , exp(mu[i]))

      alpha[i] = numerator[i] / sum(numerator[])
    }

    beta ~ dmnorm(c(0 , 0), Omega)
  }   " , fill = T)
  sink()

  i2 = matrix(c(1 , 0 , 0 , 1) , nrow = 2)
  i2 = 0.0001 * i2
  mod4designMatrix = cbind(rep(1 , 18) , X)
  modData4 = list(N = nrow(adj.mat) ,
    M = ncol(adj.mat) , y = adj.mat ,
    X = mod4designMatrix , Omega = i2)
  params4 = c("beta" , "pi" , "Like")
  inits4 = list(list(beta = rep(0 , 2)))

  model4 = jags.model("model4.txt" , data = modData4 ,
    inits = inits4 , n.chains = 1)
  samples4 = jags.samples(model4 ,
    variable.names = params4 , n.iter = nIter)
```

For model 5:

```
  sink("model5.txt")
  cat("
  model
  {
    for ( i in 1:N ) {
      pi[i , 1:N] ~ ddirch(alpha[i , 1:N])

      for ( j in 1:M ) {
        y[i , j] ~ dbern(pi[i , j])

        Like[i , j] = (pi[i,j]^y[i,j])*((1-pi[i,j])^(1-y[i,j]))
      }

      for ( j in 1:M ) {
        mu[i , j] = inprod(beta[] , c(1 , X1[i] , X2[j] ,
          X1[i]*X2[j]))
        numerator[i , j] = ifelse(i == 1 , 1 , exp(mu[i , j]))

        alpha[i , j] = numerator[i , j] /
          sum(numerator[i , 1:M])
      }
    }

    beta ~ dmnorm(c(0 , 0 , 0 , 0) , Omega)
  }
    " , fill = T)
  sink()

  i45 = 0.0001 * i4

  modData5 = list(N = nrow(adj.mat) ,
    M = ncol(adj.mat) , y = adj.mat , X1 = X , X2 = X ,
    Omega = i45)
  inits5 = list(list(beta = rep(0 , 4)))

  model5 = jags.model("model5.txt" , data = modData5 ,
    inits = inits5 , n.chains = 1)
  samples5 = jags.samples(model5 ,
    variable.names = params4 , n.iter = nIter)
```

For assessing the WAIC of the models, the package loo was loaded and the following code was used (model 1 had a burn-in of 200, all of the other models had a burn-in of 2,000):

```
nBurn = 200
nBurn2 = 2000

Like1 = as.mcmc.list(samples1$Like)[[1]][-(1:nBurn) ,]
loglik1 = as.matrix(log(Like1))
waic(loglik1)

Like2 = as.mcmc.list(samples2$Like)[[1]][-(1:nBurn2) ,]
loglik2 = as.matrix(log(Like2))
waic(loglik2)
```

And similar code was used for evaluation of the WAIC of models 3, 4, and 5 (using nBurn2 and not nBurn).

For all network analysis measures except the PageRank, the sna package was loaded. From that package, the following functions were used: degree, evcent, betweenness, closeness, graphcent, structure.statistics, centralization, connectedness, efficiency, hierarchy, and lubness. These functions are seen in the later code for the generation of the CI's of the network analysis measures.

To calculate the PageRanks, the following function was developed and used:

```
pageRank = function(x , d=0.85) {
  one = rep(1 , nrow(x))
  counts = x%*%one

  P = matrix(NA , nrow = nrow(x) , ncol = ncol(x))
  for ( i in 1:nrow(x) ) {
    for ( j in 1:ncol(x) ) {
      if ( counts[i] > 0 ) {
        if ( x[i , j] == 0 ) P[i , j] = 0
        else if ( x [i , j] == 1 )
          P[i , j] = 1 / counts[i]
      } else if ( counts[i] == 0 ) P[i , j] = 1 / (nrow(x) -
        1)
    }
  }

  J = matrix(1 , nrow = nrow(adj.mat) ,
    ncol = ncol(adj.mat))
  J = J / nrow(adj.mat)
  GPR = (d*P) + ((1-d)*J)

  GPRt = t(GPR)
  ranks = Re(eigen(GPRt , symmetric = F)$vectors[, 1])

  return(ranks / ((ranks%*%one)[1,1]))
}
```

To draw from the posterior predictive distribution for the matrices, the following function was used:

```
ppSamp = function(postMat , X , X2) {
  #our array that we will return
  out = array(0 , c(length(X) , length(X2) ,
    nrow(postMat)))

  for ( i in 1:nrow(postMat) ) {
    for ( j in 1:length(X) ) {
      for ( k in 1:length(X2) ) {
        #extract the covariates for this specific entry of
        #    our matrix
        covar = c(1 , X[j] , X2[k] , X[j]*X2[k])
        l = postMat[i ,]%*%covar
        #p is the probability that that entry is 1
        p = exp(l) / (1 + exp(l))

        #for the i-th array of dimensions length(X) by
        #    length(X2), generate a random Bernoulli with
        #    our probability p of being 1
        out[j , k , i] = rbinom(1 , size = 1 , prob = p)
      }
    }
  } #end the triple for-loop

  return(out)
}

#post3Beta is a matrix of posterior draws; it is nIter by 4,
#    each row is a different pull from the posterior and
#    each column is a different parameter
postPred = ppSamp(post3Beta , X = X , X2 = X)
```

To generate the CI's:

```
ppOutdegree = matrix(0 , nrow = nrow(adj.mat) ,
  ncol = nIter)
for ( i in 1:ncol(ppOutdegree) ) {
  ppOutdegree[, i] = degree(postPred[, , i] , g = 1 ,
    gmode = "digraph" , cmode = "outdegree")
}

ppIndegree = matrix(0 , nrow = nrow(adj.mat) ,
  ncol = nIter)
for ( i in 1:ncol(ppIndegree) ) {
  ppIndegree[, i] = degree(postPred[, , i] , g = 1 ,
    gmode = "digraph" , cmode = "indegree")
}

ppTotal = matrix(0 , nrow = nrow(adj.mat) ,
  ncol = nIter)
for ( i in 1:ncol(ppTotal) ) {
  ppTotal[, i] = degree(postPred[, , i] , g = 1 ,
    gmode = "digraph" , cmode = "freeman")
}

ppEVCenter = matrix(0 , nrow = nrow(adj.mat) ,
  ncol = nIter)
for ( i in 1:ncol(ppEVCenter) ) {
  ppEVCenter[, i] = evcent(postPred[, , i])
}

ppBetweenness = matrix(0 , nrow = nrow(adj.mat) ,
   ncol = nIter)
for ( i in 1:ncol(ppBetweenness) ) {
   ppBetweenness[, i] = betweenness(postPred[, , i])
}

ppCloseness = matrix(0 , nrow = nrow(adj.mat) ,
   ncol = nIter)
for ( i in 1:ncol(ppCloseness) ) {
   ppCloseness[, i] = closeness(postPred[, , i])
}

ppHarary = matrix(0 , nrow = nrow(adj.mat) ,
   ncol = nIter)
for ( i in 1:ncol(ppHarary) ) {
   ppHarary[, i] = graphcent(postPred[, , i])
}

ppSS = matrix(0 , nrow = nrow(adj.mat) ,
   ncol = nIter)
for ( i in 1:ncol(ppSS) ) {
   ppSS[, i] = structure.statistics(postPred[, , i])
}

IndegreeCentral = NULL
for ( i in 1:(nIter - nBurn2) ) {
   IndegreeCentral = c(IndegreeCentral ,
      centralization(postPred[, , i] , g=1 , FUN=degree ,
      cmode="indegree"))
}

OutdegreeCentral = NULL
for ( i in 1:(nIter - nBurn2) ) {
   OutdegreeCentral = c(OutdegreeCentral ,
      centralization(postPred[, , i] , g=1 , FUN=degree ,
      cmode="outdegree"))
}

FreemanCentral = NULL
for ( i in 1:(nIter - nBurn2) ) {
   FreemanCentral = c(FreemanCentral ,
      centralization(postPred[, , i] , g=1 , FUN=degree))
}

BetweennessCentral = NULL
for ( i in 1:(nIter - nBurn2) ) {
   BetweennessCentral = c(BetweennessCentral ,
      centralization(postPred[, , i] , g=1 ,
      FUN=betweenness))
}

ClosenessCentral = NULL
for ( i in 1:(nIter - nBurn2) ) {
   ClosenessCentral = c(ClosenessCentral ,
      centralization(postPred[, , i] , g=1 , FUN=closeness))
}

EVCentCentral = NULL
for ( i in 1:(nIter - nBurn2) ) {
   EVCentCentral = c(EVCentCentral ,
      centralization(postPred[, , i] , g=1 , FUN=evcent))
}

HararyCentral = NULL
for ( i in 1:(nIter - nBurn2) ) {
   HararyCentral = c(HararyCentral ,
      centralization(postPred[, , i] , g=1 , FUN=graphcent))
}

ppPageRank = matrix(0 , nrow = nrow(adj.mat) ,
   ncol = nIter - nBurn2)
for ( i in 1:ncol(ppPageRank) )
   ppPageRank[, i] = pageRank(postPred[, , i])

kConnect = NULL
for ( i in 1:(nIter - nBurn2) ) {
   kConnect = c(kConnect , connectedness(postPred[, , i]))
}

kEfficiency = NULL
for ( i in 1:(nIter - nBurn2) ) {
   kEfficiency = c(kEfficiency , efficiency(postPred[, , i]))
}

kHierarchy = NULL
for ( i in 1:(nIter - nBurn2) ) {
   kHierarchy = c(kHierarchy , hierarchy(postPred[, , i] ,
      measure = c("krackhardt")))
}

kLubness = NULL
for ( i in 1:(nIter - nBurn2) ) {
   kLubness = c(kLubness , lubness(postPred[, , i]))
}
```

To calculate the Dirichlet-1-Ranks, the following code was used, after getting an MCMC sample from the posterior of model 4:

```
post4Pi = as.mcmc.list(samples4$pi)[[1]][-(1:nBurn2) ,]
post4PiAsMat = array(0 , c(nIter - nBurn2 ,
   nrow(adj.mat) , ncol(adj.mat)))
for ( i in 1:(nIter - nBurn2) ) {
   post4PiAsMat[i , ,] = matrix(post4Pi[i ,] ,
      nrow = nrow(adj.mat))
}

d1Rank = matrix(0 , nrow = nrow(adj.mat) ,
   ncol = nIter - nBurn2)
```

```
  for ( i in 1:(nIter - nBurn2) ) {
```

Thomas Nemmers
UCLA Department of Biostatistics
650 Charles E. Young Dr. South
Los Angeles, CA 90095
E-mail address: thomasnemmers@ucla.edu

Anjana Narayan
Department of Psychology and Sociology
California State Polytechnic University
3801 West Temple Avenue, Pomona, CA 91768
E-mail address: anarayan@cpp.edu

Sudipto Banerjee
UCLA Department of Biostatistics
650 Charles E. Young Dr. South
Los Angeles, CA 90095 E-mail address: sudipto@ucla.edu

```
    tran = t(post4PiAsMat)[i , ,])

    ev = Re(eigen(tran , symmetric = F)$vectors[, 1])

    ones = rep(1 , nrow(adj.mat))



    d1Rank[, i] = ev / (ev%*%ones)[1,1]

  }
```

Similar code was used to calculate the Dirichlet-2-Ranks after obtaining an MCMC sample from the posterior of model 5.