Mean Field Variational Inference for the Popularity Adjusted Block Model

CSCI-B 659 Final Project

John Koo

1 Introduction

Various probabilistic graph models have been proposed to model real-world networks, such as the Erdos-Renyi model [1], the Stochastic Block Model (SBM) [4], the Degree-Corrected Block Model (DCBM) [2], etc. Often, the main objective of analyzing graphs under such models is community detection, which assumes that each vertex of the graph has some hidden community label, and edges between vertices are drawn from a probability distribution that is conditioned on the community labels of each pair. For example, under the homogeneous SBM, edges between each pair of vertices i and j are drawn such that if labels $z_i = z_j$, then it is $Bernoulli(p)$, and otherwise it is $Bernoulli(q)$, where presumably $0 < q < p < 1$.

Because maximum likelihood estimation for community detection is NP-hard, various methods and algorithms have been proposed, such as expectation maximization or spectral clustering. Alternatively, one can set priors on the parameters of the graph model and perform some sort of Bayesian analysis such as Markov Chain Monte Carlo. An alternative approach is Mean Field Variational Inference (MFVI), which aims to find some approximate posterior $q(\vec{\theta})$ that is close to the true posterior $p(\vec{\theta}|\vec{y})$ by the Kullback-Leibler divergence, under the constraint that each θ_r is independent of the others, i.e., $q(\vec{\theta}) = \prod_r^R q(\theta_r)$. This yields $\theta_r \overset{q}{\sim} Q(f(\vec{\theta}_{-r}, \vec{y}))$, i.e., under $q(\cdot)$, each θ_r is modeled by some distribution with hyperparameters that depend on the distributions of all of the other θ_t and the data $\vec{y}$. This leads to Coordinate Ascent Variational Inference (CAVI), an iterative algorithm that updates each θ_r until convergence in the hyperparameters. Zhang and Zhou [6] derived CAVI for the SBM and proved that it achieves the minimax lower bound.

2 The Popularity Adjusted Block Model

In this project, we want to derive CAVI for the Popularity Adjusted Block Model (PABM) [5].

Definition (Popularity Adjusted Block Model). Let graph $G = (V, E)$ such that each $v_i \in V$ belongs to one of K communities. Associated with each v_i are K parameters λ_{ik} that measure v_i's popularity with community k. G follows the Popularity Adjusted Block Model if $\forall i < j$, edge $e_{ij} \overset{indep}{\sim} Bernoulli(\lambda_{i,z_j}\lambda_{j,z_i})$, and $e_{ji} = e_{ij}$.

Once G is drawn from the PABM, the edges are compiled into adjacency matrix $A \in \{0,1\}^{n\times n}$. Since G is unweighted, undirected, and without self loops, A is binary, symmetric, and hollow.

It is straightforward to write the likelihood function for the PABM:

$$p(A|z, \{\lambda_{ik}\}) = \prod_{i<j} (\lambda_{iz_j}\lambda_{jz_i})^{A_{ij}}(1 - \lambda_{iz_j}\lambda_{jz_i})^{1-A_{ij}}$$

Where A is the observed adjacency matrix, $z \in [K]^n$ is the vector of unobserved community labels, and $\{\lambda_{ik}\}$ are the $n \times K$ popularity parameters.

The λ_{iz_j} terms can be resolved by introducing K^2 indicator functions into the likelihood:

$$p(A|z,\{\lambda_{ik}\}) = \prod_{i<j}\prod_{k}^{K}\prod_{l}^{K}(\lambda_{il}\lambda_{jk})^{A_{ij}z_{ik}z_{jl}}(1-\lambda_{il}\lambda_{jk})^{(1-A_{ij})z_{ik}z_{jl}}$$

Where $z_{ik} = I(z_i = k)$.

For Bayesian inference, a natural choice of priors might be:

- $z_i \overset{iid}{\sim} Categorical(\pi_1, ..., \pi_K)$
- $\lambda_{ik} \overset{indep}{\sim} Beta(a_{ik}, b_{ik})$

This yields the full joint distribution:

$$\begin{aligned} p(A,z,\{\lambda_{ik}\}) \propto & \prod_{i<j}\prod_{k}\prod_{l}(\lambda_{il}\lambda_{jk})^{A_{ij}z_{ik}z_{jl}}(1-\lambda_{il}\lambda_{jk})^{(1-A_{ij})z_{ik}z_{jl}} \\ & \times \prod_{k}\pi_k^{\sum_i z_{ik}} \\ & \times \prod_{i}\prod_{k}(\lambda_{ik})^{a_{ik}-1}(1-\lambda_{ik})^{b_{ik}-1} \end{aligned}$$

And its logarithm:

$$\begin{aligned} \log p = & \, constant \\ & + \sum_{i<j}\sum_{k}\sum_{l} z_{ik}z_{jl}(A_{ij}\log\lambda_{il}\lambda_{jk} + (1-A_{ij})\log(1-\lambda_{il}\lambda_{jk})) \\ & + \sum_{k}\sum_{i} z_{ik}\log\pi_k \\ & + \sum_{i}\sum_{k}(a_{ik}-1)\log\lambda_{ik} + (b_{ik}-1)\log(1-\lambda_{ik}) \end{aligned}$$

3 Mean Field Variational Inference and Implementation Details

The Mean Field Approximation for the PABM is:

$$p(A,z,\{\lambda_{ik}\}) \propto p(z,\{\lambda_{ik}\}|A) \approx q(z,\{\lambda_{ik}\}) = \Big(\prod_i q(z_i)\Big)\Big(\prod_{i,k} q(\lambda_{ik})\Big)$$

We can take for granted that $q(z_i)$ is Categorical, and the Mean Field Approximation comes out to:

$$P_q(z_i = k) = \pi'_{ik} \propto \exp\Big(\log\pi_k + \sum_{j\neq i}\sum_{l}\pi'_{jl}(A_{ij}(E[\log\lambda_{il}\lambda_{jk}]) + (1-A_{ij})E[\log(1-\lambda_{il}\lambda_{jk})])\Big) \quad (1)$$

```
#' @title Update community probability P(z_i = k)
#' @param i, k (numeric) i is the example, k is the community label
#' @param A (numeric) Adjacency matrix
#' @param community.probs (numeric) Current estimated posterior community probs
#' @param prior.pi (numeric) Prior probability for ith example and kth community
#' @param a.params, b.params (numeric) Matrix of current estimated a and b
```

```r
#'   parameters
#' @param constant (numeric) Since this tends to output very negative numbers,
#'  and exp(x) where x is very negative is close to 0, we add some number to x,
#'  and since this is only up to proportion, it will cancel out when normalized
update.comm.probs <- function(i, k,
                              A,
                              community.probs,
                              prior.pi,
                              a.params,
                              b.params,
                              constant = 2 ** 5) {
  # start at the prior
  pi.new <- prior.pi

  # then for each j != i and l = 1, 2, ..., K
  for (j in seq(n)[-i]) {
    for (l in seq(2)) {
      # update according to eq (1)
      pi.new <- pi.new +
        community.probs[j, l] *
        (A[i, j] * (ex.log.beta(a.params[i, l], b.params[i, l]) +
                      ex.log.beta(a.params[j, k], b.params[j, k])) +
           (1 - A[i, j]) * ex.log.1.minus.beta.2(a.params[i, l],
                                                 b.params[i, l],
                                                 a.params[j, k],
                                                 b.params[j, k]))
    }
  }

  # add a constant for numerical stability
  return(exp(pi.new + constant))
}
```

If we say $q(\lambda_{ik}) = Beta(\lambda_{ik}|a'_{ik}, b'_{ik})$, then it is simple enough to compute:

$$\begin{aligned} E[\log \lambda_{il}\lambda_{jk}] &= E[\log \lambda_{il}] + E[\log \lambda_{jk}] \\ &= \psi(a'_{il}) - \psi(a'_{il} + b'_{il}) + \psi(a'_{jk}) - \psi(a'_{jk} + b'_{jk}) \end{aligned}$$

```r
ex.beta <- function(a, b) {
  a / (a + b)
}

ex.log.beta <- function(a, b) {
  digamma(a) - digamma(a + b)
}
```

For $E[\log(1 - \lambda_{il}\lambda_{jk})]$, we can use the Taylor approximation $\log(1 - xy) = -\sum_k \frac{x^k y^k}{k}$ along with Beta moments $E[X^K] = \prod_{k=0}^{K} \frac{a+k}{a+b+k}$ to obtain:

$$E[\log(1 - \lambda_{il}\lambda_{jk})] = -\sum_r \frac{1}{r} \prod_{s=0}^{r-1} \frac{a'_{il} + s}{a'_{il} + b'_{il} + s} \frac{a'_{jk} + s}{a'_{jk} + b'_{jk} + s}$$

```r
ex.log.1.minus.beta <- function(a, b, order = 5) {
  # E[log(1 - x)] using taylor approximation
  moments <- rep(NA, order)
  for (k in seq(order)) {
    if (k > 1) {
      moments[k] <- ex.beta(a + k, b) * moments[k - 1]
    } else {
      moments[k] <- ex.beta(a, b)
    }
  }
  return(-sum(moments / seq_along(moments)))
}

ex.log.1.minus.beta.2 <- function(a1, b1, a2, b2, order = 5) {
  # E[log(1 - xy)] where x, y are are independent beta distributed
  moments <- rep(NA, order)
  for (k in seq(order)) {
    if (k > 1) {
      moments[k] <- ex.beta(a1 + k, b1) * ex.beta(a2 + k, b2) * moments[k - 1]
    } else {
      moments[k] <- ex.beta(a1, b1) * ex.beta(a2, b2)
    }
  }
  return(-sum(moments / seq_along(moments)))
}
```

For $q(\lambda_{ik})$, the joint pdf looks *almost* Beta. We can split up $\log \lambda_{il}\lambda_{jk} = \log \lambda_{il} + \log \lambda_{jk}$, which is consistent with what we would expect if $q(\lambda_{ik})$ are Beta pdfs, but the $\log(1 - \lambda_{il}\lambda_{jk})$ term isn't separable. One approximation is to use the first-order Taylor approximation $\log(1 - xy) \approx -xy \approx y \log(1 - x)$ to force $\log p(\cdot) = c + (a'_{ik} - 1) \log \lambda_{ik} + (b'_{ik} - 1) \log(1 - \lambda_{ik})$. This yields the following updates for λ_{ik}:

$$\lambda_{ik} \overset{q}{\sim} Beta(a'_{ik}, b'_{ik})$$

$$a'_{ik} = a_{ik} + \sum_{j \neq i} \pi'_{jk} A_{ij} \tag{2}$$

$$b'_{ik} = b_{ik} + \sum_{j \neq l} \sum_{l} \pi'_{il} \pi'_{jk} (1 - A_{ij}) \Big(\frac{a'_{jl}}{a'_{jl} + b'_{jl}} \Big) \tag{3}$$

```r
update.a <- function(i, k, prior.a, A, community.probs) {
  prior.a + sum(A[i, -i] * community.probs[-i, k])
}

update.b <- function(i, k, prior.b, A, community.probs, a.params, b.params) {
  n <- nrow(A)
  b.new <- prior.b
  for (j in seq(n)[-i]) {
    for (l in seq(2)) {
      b.new <- b.new +
        (1 - A[i, j]) *
        community.probs[i, l] *
        community.probs[j, k] *
```

```
        ex.beta(a.params[j, l], b.params[j, l])
    }
  }
  return(b.new)
}
```

This leads to the following CAVI algorithm:

Algorithm 1: Approximate CAVI for the PABM

Data: Adjacency matrix A, number of communities K, priors $\{a_{ik}, b_{ik}\}, \vec{\pi}$
Result: Estimated posterior community probabilities $\{\pi'_{ik}\}$

```
while ∑_{i,k} |(π'_ik)^(s) − (π'_ik)^(s+1)| > ϵ do
    for i = 1, ..., n do
        for k = 1, ..., K do
            Update π'_ik by Eq. (1).
            Update a'_ik by Eq. (2).
            Update b'_ik by Eq. (3).
        end
    end
end
```

Like the CAVI algorithm for the SBM developed by Zhang and Zhou, the CAVI algorithm for the PABM lends itself to batch/parallel computation.

```
#' @title CAVI for the PABM with 2 communities.
#' @param A (numeric) The adjacency matrix
#' @param prior.a (numeric) The matrix of prior a params
#' @param prior.b (numeric) The matrix of prior b params
#' @param prior.pi (numeric) The matrix of prior pi params
#' @param eps (numeric) The stopping criterion for change in community probs
#' @param maxit (numeric) The maximum number of iterations
#' @return (list) A list containing the estimated posterior community
#'   probabilities, a parameters, and b parameters
cavi.pabm.2 <- function(A,
                        prior.a = matrix(1, nrow = nrow(A), ncol = 2),
                        prior.b = matrix(1, nrow = nrow(A), ncol = 2),
                        prior.pi = matrix(.5, nrow = nrow(A), ncol = 2),
                        eps = 1e-3,
                        maxit = 100) {

  # initialize
  community.probs <- matrix(.5, nrow = n, ncol = 2)
  a.params <- matrix(1, nrow = n, ncol = 2)
  b.params <- matrix(1, nrow = n, ncol = 2)

  # set change in community probabilities to something large
  change.in.community.probs <- 1 / eps

  # keep track of iterations
  iter <- 0

  # define this to store unnormalized community probabilities
```

```
  unnorm.community.probs <- community.probs

  # run loop until convergence in the community probabilities
  while (change.in.community.probs > eps) {
    prior.community.probs <- community.probs
    for (i in seq(n)) {
      for (k in seq(2)) {
        a.params[i, k] <- update.a(i, k, prior.a[i, k], A, community.probs)
        b.params[i, k] <- update.b(i, k, prior.b[i, k], A, community.probs,
                                   a.params, b.params)
        unnorm.community.probs[i, k] <- update.comm.probs(i, k, A,
                                                          community.probs,
                                                          prior.pi[i, k],
                                                          a.params, b.params)
      }
    }

    # normalize the community probabilities
    community.probs <- sweep(unnorm.community.probs, 1,
                             rowSums(unnorm.community.probs), `/`)

    # compute change in community probabilities
    change.in.community.probs <-
      sum(abs(community.probs - prior.community.probs))

    # update iter count
    iter <- iter + 1
    if (iter >= maxit) {
      warning('failed to converge')
      break
    }
  }

  return(list(community.probs = community.probs,
              a.params = a.params,
              b.params = b.params,
              iter = iter))
}
```

4 An Example

Thus far, we've let the number of communities K be arbitrary, and none of the analysis depends on constricting K, only that it is known beforehand. For the sake of simplifying the code as well as keeping runtimes low, we will limit the example to one instance of $K = 2$.

First, we need to initialize the PABM. This can be done by initializing an edge probability matrix $P \in [0, 1]^{n \times n}$ that follows the PABM structure. The method used here will follow Theorem 2 of Koo et al. [3]:

```
generate.P.beta <- function(n, K = 2, a1 = 2, b1 = 1, a2 = 1, b2 = 2,
                            unbalanced = FALSE) {
  if (unbalanced) {
    clustering <- rmultinom(n, 1, seq(K) ** -1) %>%
      apply(2, function(x) which(x == 1))
  } else {
```

```
    clustering <- sample(seq(K), n, replace = TRUE)
  }
  clustering <- sort(clustering)
  n.vector <- sapply(seq(K), function(k) sum(clustering == k))

  P <- matrix(NA, n, n)
  for (k in seq(K)) {
    n.k <- n.vector[k]
    low.ind.k <- ifelse(k == 1, 1, sum(n.vector[seq(k - 1)]) + 1)
    high.ind.k <- sum(n.vector[seq(k)])
    for (l in seq(k)) {
      n.l <- n.vector[l]
      if (k == l) {
        lambda <- rbeta(n.k, a1, b1)
        P.kk <- lambda %*% t(lambda)
        P[low.ind.k:high.ind.k, low.ind.k:high.ind.k] <- P.kk
      } else {
        low.ind.l <- ifelse(l == 1, 1, sum(n.vector[seq(l - 1)]) + 1)
        high.ind.l <- sum(n.vector[seq(l)])
        lambda.kl <- rbeta(n.k, a2, b2)
        lambda.lk <- rbeta(n.l, a2, b2)
        P.kl <- lambda.kl %*% t(lambda.lk)
        P.lk <- lambda.lk %*% t(lambda.kl)
        P[low.ind.k:high.ind.k, low.ind.l:high.ind.l] <- P.kl
        P[low.ind.l:high.ind.l, low.ind.k:high.ind.k] <- P.lk
      }
    }
  }
  return(list(P = P, clustering = clustering))
}
```

We also need to sample a binary A from P. This can be done simply by $A_{ij} \overset{indep}{\sim} Bernoulli(P_{ij})$ for $i < j$, and then setting $A_{ji} = A_{ij}$ and $A_{ii} = 0$.

```
draw.graph <- function(P) {
  # determine number of nodes
  n <- nrow(P)
  # initialize adjacency matrix with 0s
  A <- matrix(0, nrow = n, ncol = n)
  # draw from bernoulli distribution for lower triangle of P
  A[lower.tri(A)] <- rbinom(n * (n - 1) / 2, 1, P[lower.tri(P)])
  # symmetrize
  A <- A + t(A)
  return(A)
}
```

We will also compare the performance of CAVI against Orthogonalized Spectral Clustering and Sparse Subspace Clustering [3].

Now we can finally draw A, after setting some hyperparameters. Here, we will set:

- $\pi_k = 1/K \ \forall k$
- $a_{ik} = \begin{cases} 2 & z_i = k \\ 1 & z_i \neq k \end{cases}$

- $b_{ik} = \begin{cases} 1 & z_i = k \\ 2 & z_i \neq k \end{cases}$

```
set.seed(659)

n <- 128

Pz <- generate.P.beta(n, 2, 2, 1, 1, 2)
P <- Pz$P
z <- Pz$clustering

A <- draw.graph(P)
```

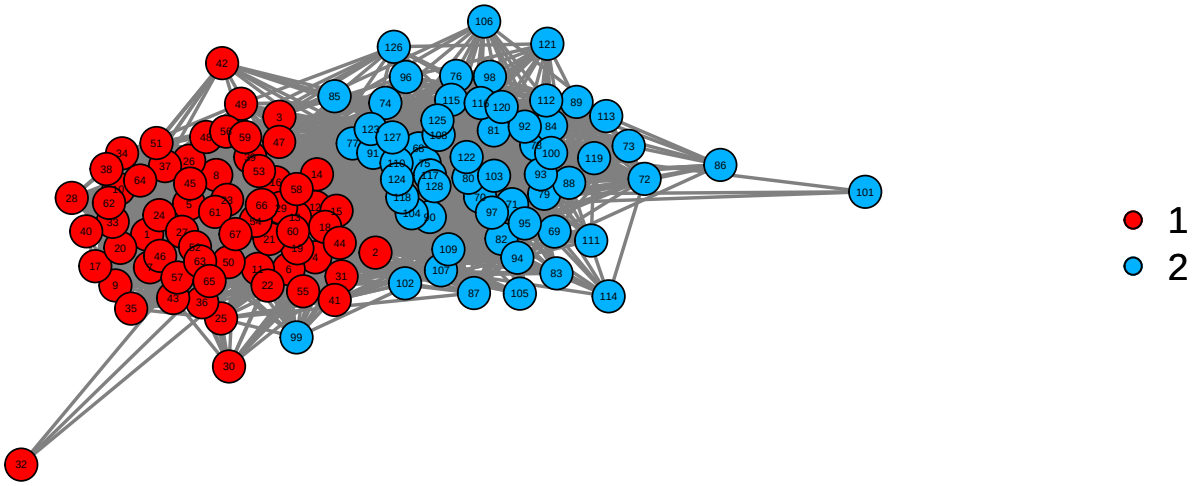


Figure 1: Graph drawn from a PABM.

However, incorporating this prior proves problematic because the hyperparameters depend on community memberships. We could add this information to the algorithm, but this is not possible outside of simulations because it means we know the community memberships *a priori*. In addition, even if we did know this information, since community labels are equivalent up to permutation, it may not end up being "correct" if the algorithm ends up switching the labels. For our example, we will just set the priors a_{ik} and b_{ik} to 1.

```
# OSC and SSC results
osc.clust <- cluster.pabm(A, 2)
ssc.clust <- ssc(A, 2, lambda = .03)

# CAVI outputs
cavi.out <- cavi.pabm.2(A)
# find a hard clustering from the cluster probabilities
cavi.clust <- round(cavi.out$community.probs[, 1]) + 1
```

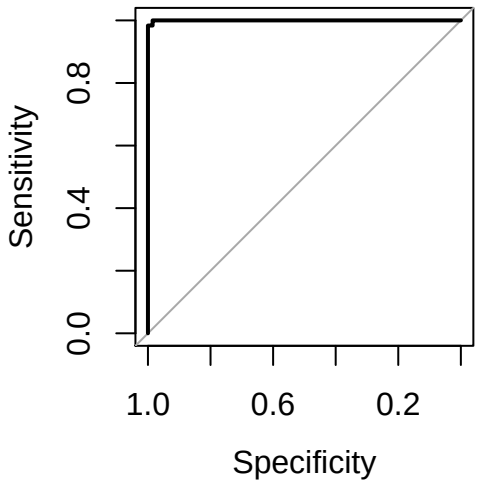
Table 1: Community detection accuracy rates.

Method	Accuracy
OSC	0.984
SSC	0.914
CAVI	0.992

Among the three algorithms, CAVI performs the best, with OSC having very similar performance and SSC performing the worst. Prior experiments suggest that SSC does not perform well for the PABM when $K = 2$, and this is consistent with those observations.

An advantage of a Bayesian analysis is it provides community label probabilities rather than just label predictions. This allows us to compute a ROC curve:

```
roc <- pROC::roc(z, cavi.out$community.probs[, 1])
plot(roc)
```




```
print(roc)
```

```
Call:
roc.default(response = z, predictor = cavi.out$community.probs[,     1])

Data: cavi.out$community.probs[, 1] in 67 controls (z 1) < 61 cases (z 2).
Area under the curve: 0.9998
```

5 Discussion

We derived an approximate CAVI algorithm for the PABM, which shows comparable performance to OSC and SSC, for which there is guaranteed perfect community detection for large n, although we did not show any such theoretical results for CAVI. Our implementation also required some approximation to make $q(\lambda_{ik})$ be a Beta density function. An alternative approach may be to use a Generalized Beta distribution for exact computation. We also used a Taylor approximation for $E[\log(1 - \lambda_{il}\lambda_{jk})]$, and it may be possible to derive an exact expression for this value. We also did not explore the outputs for a'_{ik} and b'_{ik}, and a cursory look at the outputs suggests that they failed to find the true prior values. Finally, CAVI proved to be very slow in comparison to OSC and SSC, and while some of that may be due to an unoptimized implementation, it still requires looping over $n \times K$ variables at each iteration, although the algorithm is very parallelizable, which may alleviate some of the time costs. Nevertheless, at least for a subset of PABM graphs, CAVI seems to output clusters that are comparable to other algorithms.

References

[1] P. Erdös and A. Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290, 1959.

[2] Brian Karrer and M. E. J. Newman. Stochastic blockmodels and community structure in networks. *Physical Review E*, 83(1), Jan 2011. ISSN 1550-2376. doi: 10.1103/physreve.83.016107. URL http://dx.doi.org/10.1103/PhysRevE.83.016107.

[3] John Koo, Minh Tang, and Michael Trosset. Connecting the popularity adjusted block model to the generalized random dot product graph for community detection and parameter estimation. https://github.com/johneverettkoo/pabm-grdpg/blob/master/summary.pdf, 2021.

[4] François Lorrain and Harrison C. White. Structural equivalence of individuals in social networks. *The Journal of Mathematical Sociology*, 1(1):49–80, 1971. doi: 10.1080/0022250X.1971.9989788. URL https://doi.org/10.1080/0022250X.1971.9989788.

[5] Srijan Sengupta and Yuguo Chen. A block model for node popularity in networks with community structure. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 80(2):365–386, March 2018. ISSN 1369-7412. doi: 10.1111/rssb.12245.

[6] Anderson Y. Zhang and Harrison H. Zhou. Theoretical and computational guarantees of mean field variational inference for community detection, 2017.