Stat651 Final Project - The Indian Buffet Process

Arthur Lui

9 December 2014

1 Introduction

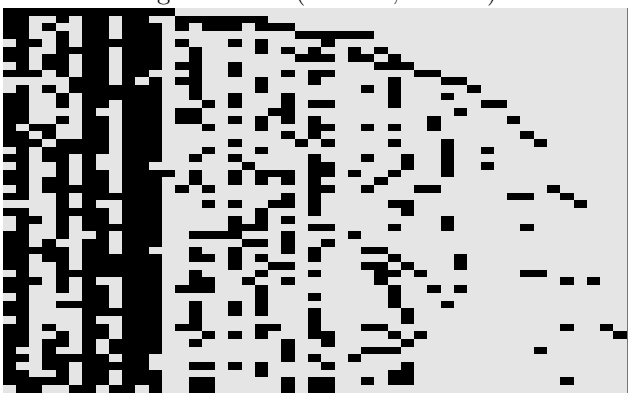
One key problem in recovering the latent structure responsible for generating observed data is determining the number of latent features. The Indian Buffet process (IBP) provides a flexible distribution for sparse binary matrices with infinite dimensions (i.e. finite number of rows, and infinite number of columns). When used as a prior distribution in a latent feature model, the IBP can learn the number of latent features generating the observations because it can draw binary matrices which have a potentially infinite number of columns. I will use the IBP as a prior distribution in a Gaussian latent feature model to recover the latent structures generating the observations.

2 Model

The IBP is a distribution for sparse binary matrices with finite number of rows and potentially infinite number of columns. The process of generating a realization from the IBP can be described by an analogy involving Indian buffet restaurants.

Let Z be an $N \times \infty$ binary matrix. Each row in Z represents a customer which enters an Indian buffet and each column represents a dish in the buffet. Customers enter the restaurant one after another. The first customer samples an $r = Poisson(\alpha)$ number of dishes, where $\alpha > 0$ is a mass parameter which influences the final number of sampled dishes. This is indicated in by setting the first r columns of the first row in Z to be 1. The other values in the row are set to 0. Each subsequent customer samples each previously sampled dish with probability proportional to its popularity. That is, the next customer samples dish k with probability $\frac{m_k}{i}$, where m_k is the number of customers that sampled dish k, and i is the current customer number (or row number in Z). Each customer also samples an additional $Poisson(\alpha/i)$ number of new dishes. Once all the N customers have gone through this process, the resulting Z matrix will be a draw from the Indian buffet process with mass parameter α. In other words, $Z \sim IBP(\alpha)$. Note that $\alpha \propto K_+$, where K_+ is the final number of sampled dishes (occupied columns). Figure1 shows a draw from an IBP(10) with 50 rows. The white squares are 1, indicating that a dish was taken; black squares are 0, indicating that a dish was not taken.

Figure 1: IBP($N = 50$, $\alpha = 10$)



The probability of any particular matrix produced from this process is

$$P(\boldsymbol{Z}) = \frac{\alpha^{K_+}}{\prod_{i=1}^{N} K_1{}^{(i)}!} exp\{-\alpha H_N\} \prod_{k=1}^{K_+} \frac{(N-m_k)!(m_k-1)!}{N!}, \tag{1}$$

where H_N is the harmonic number, $\sum_{i=1}^{N} \frac{1}{i}$, K_+ is the number of non-zero columns in $\boldsymbol{Z}$, m_k is the k^{th} column sum of $\boldsymbol{Z}$, and $K_1^{(i)}$ is the "number of new dishes" sampled by customer i.

One way to get a draw from the IBP(α) is to simulate the process according to the description above. Another way is to implement a Gibbs sampler as follows:

1. Start with an arbitrary binary matrix of N rows
2. For each row, i,
   (a) For each column, k,
   (b) if $m_{-i,k} = 0$, delete column k. Otherwise,
   (c) set z_{ik} to 0
   (d) set z_{ik} to 1 with probability $P(z_{ik} = 1|\boldsymbol{z_{-i,k}}) = \frac{m_{-i,k}}{i}$
   (e) at the end of row i, add Poisson($\frac{\alpha}{N}$) columns of 1's
3. iterate step 2 a large number of times

We can likewise incorporate this Gibbs sampler to sample from the posterior distribution P($\boldsymbol{Z}|\boldsymbol{X}$) where $\boldsymbol{Z} \sim IBP(\alpha)$ by sampling from the complete conditional

$$P(z_{ik} = 1|\boldsymbol{Z_{-(ik)}}, \boldsymbol{X}) \propto p(\boldsymbol{X}|\boldsymbol{Z})P(z_{ik} = 1|\boldsymbol{Z_{-(ik)}}). \tag{2}$$

Note that the conjugate prior for α is a Gamma distribution.

$$\begin{aligned} \boldsymbol{Z}|\alpha &\sim IBP(\alpha) \\ \alpha &\sim Gamma(a,b), \text{where } b \text{ is the scale parameter} \end{aligned}$$

$$\begin{aligned} p(\alpha|\boldsymbol{Z}) &\propto p(\boldsymbol{Z}|\alpha)p(\alpha) \\ p(\alpha|\boldsymbol{Z}) &\propto \alpha^{K_+} e^{-\alpha H_N} \alpha^{a-1} e^{-\alpha/b} \\ p(\alpha|\boldsymbol{Z}) &\propto \alpha^{a+K_+-1} e^{-\alpha(1/b+H_N)} \end{aligned}$$

$$\alpha|\boldsymbol{Z} \sim Gamma(a+K_+, (1/b+H_N)^{-1}) \tag{3}$$

3 Example: Linear-Gaussian Latent Feature Model with Binary Features

Suppose, we observe an $N \times D$ matrix $\boldsymbol{X}$, and we believe

$$\boldsymbol{X} = \boldsymbol{ZA} + \boldsymbol{E},$$

where $\boldsymbol{Z}|\alpha \sim IBP(\alpha)$, $\boldsymbol{A} \sim MVN(\boldsymbol{0}, {\sigma_A}^2\mathbf{I})$, $\boldsymbol{E} \sim MVN(\boldsymbol{0}, {\sigma_X}^2\mathbf{I})$.
α $Gamma(a, b)$,
It can be shown that

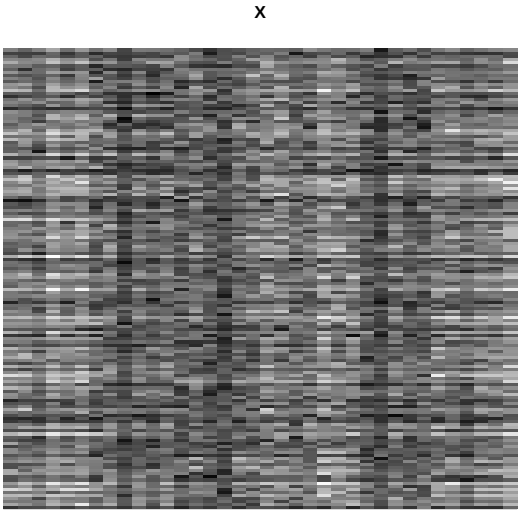
$$p(\boldsymbol{X}|\boldsymbol{Z}) = \frac{1}{(2\pi)^{ND/2}\sigma_X^{(N-K)D}\sigma_A^{KD}|\boldsymbol{Z}^T\boldsymbol{Z} + (\frac{\sigma_X}{\sigma_A})^2\mathbf{I}|^{D/2}} \exp\{-\frac{1}{2\sigma_X^2}tr(\boldsymbol{X}^T(\mathbf{I} - \boldsymbol{Z}(\boldsymbol{Z}^T\boldsymbol{Z} + (\frac{\sigma_X}{\sigma_A})^2\mathbf{I})^{-1}\boldsymbol{Z}))\boldsymbol{X}\} \tag{4}$$

Now, we can use equation (2) to implement a Gibbs sampler to draw from the posterior $\boldsymbol{Z}|\boldsymbol{X}, \alpha$.

3.1 Data & Results

Each of ten Stat666 students created a 6×6 binary image. To the image, Gaussian noise (mean=0,variance=.25) was added to each cell of the binary image. Gaussian noise was added to the same image to generate 10 6×6 images. These images were turned into 1×36 row vectors. All these vectorized images were stacked together to form one large 100×36 matrix. (Note that I did not know the design of these images before hand. The images could be letters, numbers, interesting patterns, pokemon faces, etc.) Figure2 displays the data from the ten Stat666 students.

Figure 2: Data From Ten Stat666 Students



A Gibbs sampler was implemented to retrieve posterior distributions for $\boldsymbol{Z}$, $\boldsymbol{A}$, and α. α was initially set to 1, and the posterior for α was obtained by equation (3) with prior distribution $Gamma(3, 2)$. The parameters were chosen such that α was centered at 6 and had a variance of 12, as I believed I may have many latent features, but my uncertainty was high. Equation(2) was used to retrieve the posterior distribution for $\boldsymbol{Z}$ (a collection of binary matrices). After 5000 iterations, the trace plot for the number of columns in the binary matrices drawn were plotted (See Figure 3). Diagnostics for a cell by cell trace plot could also be plotted, but may be too difficult to analyze as the dimensions of the matrices are changing, and the matrices are large. The execution time was about 4 hours.

Figure 3:

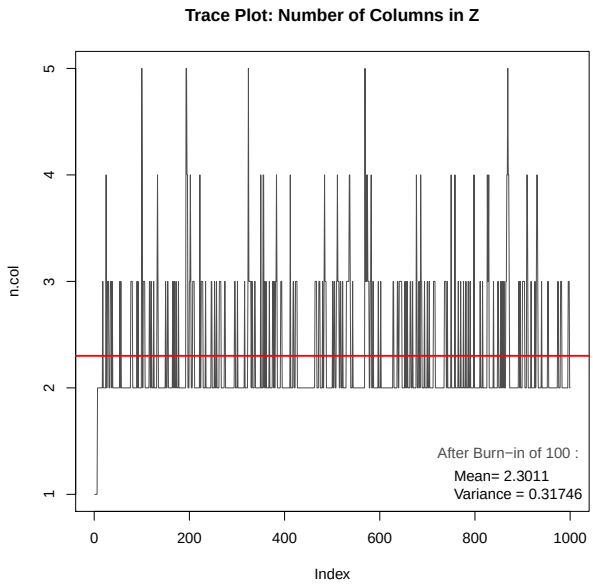
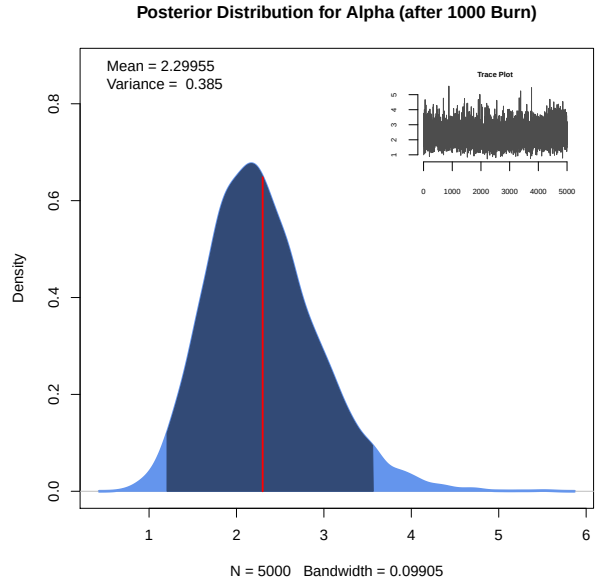


Figure 4:



The number of columns in $\boldsymbol{Z}$ appears to have converged to 9. This means that the number of latent features discovered appears to be 9. This is reasonable as the image $\boldsymbol{X}$ is comprised of 10 students' images. A burn-in of the first 1000 draws were removed. Then, the 4000 $\boldsymbol{Z}$ matrices were superimposed, summed element by element, and divided by 4000. Cells that had values $> .9$ were set to 1; and 0 otherwise. This is not necessary, but this removes the columns that were not likely to exist. To elaborate, from the trace plot (Figure3), we see that after burn-in, there is one instance where the number of columns in $\boldsymbol{Z}$ was 10. One instance out of 4000 is not significantly large enough to say that *that* latent feature is generating the observed data. So, the tenth column was removed. The resulting matrix, I will call the posterior mean for $\boldsymbol{Z}$. The trace plot for α

(Figure 4) shows that α appears to have converged, with mean = 2.08 and variance = .359. Figure 5 shows the posterior mean for $\boldsymbol{Z}$. We can interpret the matrix in the following way. All the observations are being generated by the first column (feature). The $11^{th} through 20^{th}$ observations in $\boldsymbol{X}$ are being generated by the second column, etc. The posterior mean for $\boldsymbol{A}$ calculated as $E[\boldsymbol{A}|\boldsymbol{X},\boldsymbol{Z}] = (\boldsymbol{Z}^T\boldsymbol{Z} + \frac{\sigma_X^2}{\sigma_A^2}\mathbf{I})^{-1}\boldsymbol{Z}^T\boldsymbol{X}$, and is shown in Figure 6 and Figure 7. Figure 6 shows the matrix in a 10×36 form; Figure 7 shows the matrix in 6×6 form. That is, each row in $\boldsymbol{A}$ were back-transformed into its matrix form.

Figure 5:

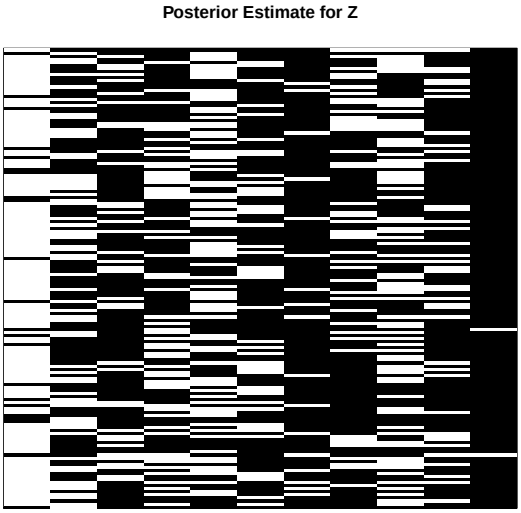


Figure 6:

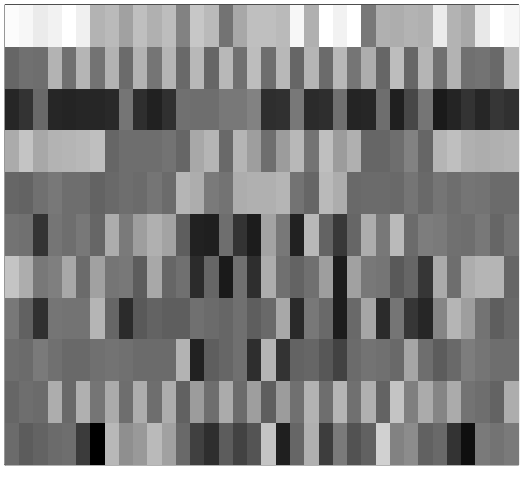
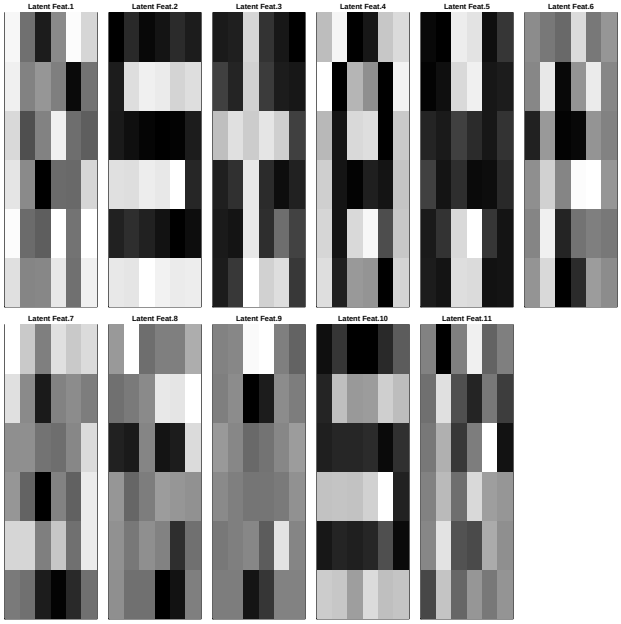


Figure 7: The latent features turned back into 6×6 images.

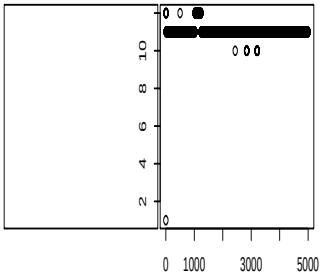


Now, we can predict the latent features generating each observation by multiplying the posterior mean of $\boldsymbol{Z}$ by the posterior mean of $\boldsymbol{A}$. We will call this matrix the posterior mean of $\boldsymbol{ZA}$. Each row in this matrix will reveal the latent structures generating the observation in the respective row of $\boldsymbol{X}$. Figure 8 shows the latent structure learned from the data. For each observation, the data and latent structures are layed side by side for a visual comparison. The features learned were similar to the images created by the ten Stat666 students.

4 Conclusions

The IBP provides a flexible distribution on sparse binary matrices. The distribution can be extremely valuable when used as a prior on binary matrices in latent feature models. Using Gibbs sampling, the number of latent features can be quickly learned and the features can be discovered. Prior distributions can be set on σ_X and σ_A to improve performance. In practice, these parameters are unknown so they should be modeled. Areas of practical application for the IBP are still being studied, and requires further investigation.

Figure 8: The latent features for each student.



5 Appendix - Code & Data

5.1 generateData.R

5.2 ibp.R

```
#rm(list=ls())
#library(expm)
library(MASS)
source("rfunctions.R")

tr <- function(m) sum(diag(m))

dmatnorm <- function(X,M,U,V){
  n <- nrow(X)
  p <- ncol(X)

  exp(-.5 * tr(solve(V) %*% t(X-M) %*% solve(U) %*% (X-M))) /
  ( (2*pi)^(n*p) * det(V)^n * det(U)^p )^(.5)
}

ldmatnorm <- function(X,M,U,V){
  n <- nrow(X)
  p <- ncol(X)

  (-.5 * tr(solve(V) %*% t(X-M) %*% solve(U) %*% (X-M))) -
  .5 * log((2*pi)^(n*p) * det(V)^n * det(U)^p)
}

#rmatnorm1 <- function(M,U,V){ # This Works! Not as fast.
#  n <- nrow(U)
#  p <- nrow(V)
#  Z <- matrix(rnorm(n*p),n,p)
#  X <- sqrtm(U) %*% Z %*% sqrtm(V) + M
#  X
#}

rmatnorm <- function(M,U,V){ # Works. This is FASTER!
  # If M is nxp, then
  #    U is nxn
  #    V is pxp.

  vec.m <- cbind(as.vector(M))
  #v.x <- mvrnorm(1,vec(M),U %x% V)
  v.x <- vec.m+ t(chol(U %x% V))%*%rnorm(length(vec.m)) #MVN draw
  X <- matrix(v.x,nrow(U),ncol(V))
  X
}

#M <- matrix(1:20,4,5)
#U <- diag(4); V <- diag(5)
#N <- 10000
```

```
#X <- foreach(i=1:N) %dopar% rmatnorm(M,U,V)
#(EX <- Reduce('+',X) / N)

r.X.given.Z <- function(Z,d=2,su=2,sv=2){ # X is n by d
  n <- nrow(Z)
  k <- ncol(Z)
  J <- matrix(1,k,d)
  Iu <- diag(n)
  Iv <- diag(d)

  rmatnorm(Z%*%J,su^2*Iu,sv^2*Iv)
}

r.X.given.Z.given.a <- function(n=2,a=10,d=2,su=2,sv=2){ # X is n by d
  Z <- rIBP(n,a,F)
  k <- ncol(Z)
  #cat("\n"); print(paste("k = ", k))
  J <- matrix(1,k,d)
  Iu <- diag(n)
  Iv <- diag(d)

  rmatnorm(Z%*%J,su^2*Iu,sv^2*Iv)
}

count <- function(m,Zs){ # counts the appearance of the matrix m in a list
                         # of matrices Zs
  cnt <- 0
  for (i in 1:length(Zs)){
    if (all(dim(Zs[[i]]) == dim(m))){
      if (all(Zs[[i]] == m)) {
        cnt <- cnt + 1
      }
    }
  }
  cnt
}


normalize <- function(x) {
  sum <- 0
  for (i in 1:length(x)){
    sum <- sum + x[i]
  }
  x / sum
}

find.k1 <- function(z){ # The number of new dishes sampled by each customer
  z <- as.matrix(z)
  k1 <- NULL
  k1[1] <- sum(z[1,])
  n <- nrow(z)
```

```
  max.col <- k1[1]
  for (i in 2:n){
     k1[i] <- sum(z[i,-c(1:max.col)]) # The number of new dishes sampled by cust. i
     if (k1[i]>0) max.col <- max.col+k1[i]
  }
  k1
}

find.kh <- function(z){

  # After finding kh,
  # foreach unique vector,
  # compute the factorial of the counts,
  # then multiply the quantities together.
  # So the constant in 14 is the product of the factorial of the counts of unique

  #n <- nrow(z)
  z <- as.matrix(z)
  k <- ncol(z)
  kh <- NULL
  z.col   <- list()
  for (i in 1:k) z.col[[i]] <- z[,i]
  uniq.zk <- unique(z.col)
  for (i in 1:length(uniq.zk)) kh[i] <- count(uniq.zk[[i]],z.col)
  kh
}

dIBP <- function(z,a=1,log=F,exchangeable=T,const=T){
  z <- as.matrix(z[,which(apply(z,2,sum)>0)])
  n <- nrow(z)
  k <- ncol(z) # This is K+
  Hn <- sum(1/(1:n))

  if (k==0) {
    if (!log) {
      return(exp(-a*Hn))
    } else {
      return(-a*Hn)
    }
  }

  mk <- apply(z,2,sum)
  kk <- NULL

  if (exchangeable) {
    kk <- find.kh(z) # exchangeable IBP uses kh
  } else {
    kk <- find.k1(z) # nonexchangeable uses k1
  }

  A <- 0
```

```
  if (!log) {
    A <- ifelse(const,prod(gamma(kk+1)),1)
    density <- a^k / A * exp(-a*Hn) * prod(gamma(mk)*gamma(n-mk+1)/gamma(n+1))
  } else {
    A <- ifelse(const,sum(lgamma(kk+1)),0)
    density <- k*log(a) -A -a*Hn + sum(lgamma(mk) + lgamma(n-mk+1) - lgamma(n+1))
  }

  density
}


rIBP <- function(ppl=50,a=10,plotting=F,...){

  last <- rpois(1,a) # I assume that the first customer may sample NO dishes

  M <- matrix(0,ppl,last)
  M[1,0:last] <- 1 #V1

  for (n in 2:ppl){
    for (k in 0:last){
      mk <- sum(M[,k])
      M[n,k] <- sample(0:1,1,prob=c(1-mk/n,mk/n))
    }
    newLast <- last+rpois(1,a/n)
    col0 <- matrix(0,ppl,newLast-last)
    if (ncol(col0) > 0){ # added & newLast > 0
      M <- cbind(M,col0)
      M[n,(last+1):newLast] <- 1
      last <- newLast
    }
  }

  if (plotting) {
    a.image(M,...)
    #pheatmap(M,cluster_row=F,cluster_col=F,display_numbers=T,number_format="%.0f")
  }

  M
}


#Z <- matrix(c(1,0,0,1,1,0,0,0,1),3,3)
#dIBP(Z,const=T)

#M <- matrix(1:12,3,4)
#U <- diag(3); V <- diag(4)
#N <- 10000
#
#X <- foreach(i=1:N) %dopar% rmatnorm(M,U,V)
#(EX <- Reduce('+',X) / N)
```

```
# Some assignment:
#  r.X.given.Z(n=5,a=12,d=3,su=1,sv=1); cat("\n\n")
#  plotGrid(10,1,plotting=F)

############################## Next Task: ###################################
#
# Changing Alpha:
#   1) E[X|Z=z]
#   2) E[X] = E[E[X|Z]] # all cells tend to a
#
# Metropolis:
#   1) X|Z ~ N(ZJ, s2_uI, s2I), where X(nxd), Z(nxk), J(kxd), I(dxd), I(kxk)
#   2) Z ~ IBP(a)
#
#   Q) What does X look like for various a, s2?
#   Q) What is Z|X?
#
#############################################################################
```

5.3 gibbs.R

```
source("rfunctions.R")

gibbs.post <- function(X=Y,siga=1,sigx=.5,a=1,B=1000,burn=B*.1,showProgress=T,
                       a.a=1,a.b=1,plotProgress=F) {
  B <- ceiling(B/50)*50

  D <- ncol(X)
  N <- nrow(X)

  Z <- rIBP(N,.5)
  alpha <- NULL
  alpha[1] <- a

  Hn <- sum(1/(1:N))

  p.x.z <- function(Z,log=T,with.norm.const=T) { # p.x.z = Likelihood
    K <- ncol(Z)
    H <- Z %*% solve(t(Z)%*%Z+(sigx/siga)^2 * diag(K)) %*% t(Z)
    I <- diag(nrow(H))

    if (!log) {
      norm.const <- ifelse(with.norm.const,
                           (2*pi)^(N*D/2) * sigx^((N-K)*D) * siga^(K*D),1)
      out <-
      ( norm.const * det(t(Z)%*%Z + (sigx/siga)^2 * diag(K))^(D/2) )^(-1) *
      exp(
        -1/(2*sigx^2) * tr(t(X) %*% (I-H) %*% X)
      )
```

```
  } else {
    norm.const <- ifelse(with.norm.const,
                         (N*D/2)*log(2*pi)+((N-K)*D)*log(sigx)+(K*D)*log(siga),0)
    out <-
    -( norm.const + (D/2)*log(det(t(Z)%*%Z + (sigx/siga)^2 * diag(K))) ) +
    -1/(2*sigx^2) * tr(t(X) %*% (I-H) %*% X)
  }
  #out + dIBP(Z,a,log=T,exch=F,const=F)
  out
} # p.x.z

#p.x.z(z) # Remove this line!!!
p.zik.x <- function(z,i,k,log=T) { # P[z_{ik}=1|z_{-i,k}].  # exact

  zz <- z
  zz[i,k] <- 1
  a <- p.x.z(zz,log=T,with.norm.const=F)
  zz[i,k] <- 0
  b <- p.x.z(zz,log=T,with.norm.const=F)
  mk = sum(z[-i,k])
  p1 <- mk/N # p = Prior
  p0 <- 1 - p1

  p0 <- b+log(p0)#7Nov,2014
  p1 <- a+log(p1)#7Nov,2014

  if (mk==0){
    out <- 0 # essentially doing nothing. because mk=0 => z[i,k] = 0.
  } else if (!log) { # Not Logged
    #out <- a * p / (a*p + b*(1-p))
    out <- 1 / (1+exp(p0-p1))
  } else { # Logged
    #print(paste("a:",a,"b:",b))
    #out <- a + log(p) - log(taylor.e(a)*p + taylor.e(b)*(1-p)) # HERE IS MY PROBLEM! 31 OCT
    #out <- a + log(p) - log(exp(a)*p + exp(b)*(1-p)) # HERE IS MY PROBLEM! 31 OCT
    out <- -log(1+exp(p0-p1))
  }

  out
}

# DOES THIS MAKE SENSE???
# STOPPED HERE 8 NOVEMBER. START AGAIN on 10 NOVEMBER.

# Need a way to sample alpha. Draw posterior alpha.
#sampNewAlpha <- function(alpha=a,z,s=5) {
#  prior <- function(x) dgamma(x,1,1,log=T)
#  lp <- prior(seq(0,5,len=100))
#  ll <- p.x.z(z)
#
#}
```

```
sampNewCols <- function(z,i,s=9,a=alpha[1]) { #s is the max num of columns. Avoid mh.
  prior <- function(l) dpois(l,a/N,log=T)

  lp <- prior(0:s) # log prior
  ll <- apply(matrix(0:s),1,function(x) {
                              col1 <- matrix(0,N,x)
                              col1[i,] <- 1
                              p.x.z(cbind(z,col1))
                              }) # log like
  lg <- lp+ll

  lpi <- apply(matrix(0:s),1,function(i) -log(sum(exp(lg-lg[i+1]))))
  Pi <- exp(lpi)

  #print(Pi)
  sample(0:s,1,prob=Pi)
}

Zs <- as.list(1:B)
Zs[[1]] <- matrix(1,N)

for (b in 2:B) { # B = num of iterations in Gibbs
  old.time <- Sys.time()
  z <- Zs[[b-1]]
  alpha[b] <- alpha[b-1]

  for (i in 1:N) { # iterate through all rows of Z
    K <- ncol(z)

    k <- 1
    while(k<=K) { # iterate through all columns of Z
      if (K>0) {
        #p <- p.zik.x(z,i,k,log=T) # pzik=1|x. Here
        p <- p.zik.x(z,i,k,log=F) # pzik=1|x.
        #cat(paste("\r Num of Z columns:",K))
        #Sys.sleep(.01)
        #u <- log(runif(1))
        u <- runif(1)
        if (p<=0) { # HERE!!!!!!!!!!!!!!!!!!!!!!
          #if (p==0) print("Removed a column")
          z <- as.matrix(z[,-k])
          k <- k-1
        } else if (p>u){
          z[i,k] <- 1
        } else {
          z[i,k] <- 0
        }
      }

      k <- k + 1
```

```
      K <- ncol(z)
    } # end of its through all columns of Z

    # Drawing new dishes should be prior times likelihood
    #new <- rpois(1,a/N)
    new <- sampNewCols(z,i,a=alpha[b])

    if (new>0) {
      #print(paste("a/N = ", a/N ,". New Columns:",new,"ncol(z) =",ncol(z)))
      col1 <- matrix(0,N,new)
      col1[i,] <- 1
      z <- as.matrix(cbind(z,col1))
    }
  } # end of its through all rows of Z

  # Z|alpha propto alpha^{a-1} exp{-alpha Hn}
  #   alpha propto alpha^{a-1} exp{-alpha / b}
  # if a=1, b=1
  # a|Z ~ Gamma(a+K,(1/b+Hn)^(-1))
  #a.a <- a.a+ncol(z)
  #a.b <- (1/a.b+Hn)^(-1)
  alpha[b] <- rgamma(1,a.a+ncol(z),scale=1/(1/a.b+Hn))
  Zs[[b]] <- z

  if (b %% 50 == 0) {
    gp <- b%/%50
    sink("out/Z.post.results",append=T)
      Zs[ ((gp-1)*50+1) : (gp*50) ]
    sink()
  } else if (b==1) {
    sink("out/Z.post.results")
    sink()
  }

  if (plotProgress) {
    n.col <- unlist(lapply(Zs[1:b],ncol))
    plot(n.col,xlab="Iteration",ylab="K+",
         main=paste0("Columns of Z ","(",b,")"),col="pink",lwd=3,type="b",pch=20)
    abline(h=mean(n.col),col="blue",lwd=3)
    minor <- function() { plot(alpha,type="l",col="gray30") }
    plot.in.plot(minor,"bottomright")
  }

  if (showProgress) count.down(old.time,b,B)#pb(b,B)
 } # end of gibbs

 #out <- Zs[(burn+1):B]
 out <- list("Zs"=Zs,"alpha"=alpha)
 out
}
```

```
#elapsed.time <- system.time(out <- gibbs.post(Y,a=1,B=1000,burn=0,showProgress=T,
#                                              plotProgress=T))
```

5.4 rfunctions.R

```
a.image <- function(Q,color=paste0("gray",90:0),...) {
  image(t(apply(Q,2,rev)),yaxt="n",xaxt="n",col=color,...)
}

count.down <- function(old.time,i,B) {
  prog <- round(100*i/B,4)
  new.time <- Sys.time()
  time.diff <- as.numeric(new.time-old.time)
  time.remain <- time.diff * (B-i)
  if (time.remain < 60) {
    secs <- round(time.remain)
    time.remain <- paste0(secs,"s     ")
  } else if (time.remain<3600) {
    mins <- round(time.remain%/%60)
    secs <- round(time.remain%%60)
    time.remain <- paste0(mins,"m ",secs,"s        ")
  } else {
    hrs <- round(time.remain%/%3600)
    mins <- round((time.remain%%3600) %/% 60)
    time.remain <- paste0(hrs,"h ",mins,"m         ")
  }
  cat(paste0("\rProgress: ",prog,"%. Time Remaining: ",time.remain," "))
  if (i==B) cat("100%\n")
}

# Example
#B <- 5000
#for(i in 1:B) {
#  old.time <- Sys.time()
#  Sys.sleep(1) # your function here
#  count.down(old.time)
#}


sum.matrices <- function(Ms,return.matrices=F) {
# Ms is a list of matrices of different lengths
# return.matrices is a boolean. If FALSE, function returns the sum of the matrices.
# If TRUE, function returns a list of the matrices also.

  l <- length(Ms)
  max.c <- max(unlist(lapply(Ms,ncol)))
  max.r <- max(unlist(lapply(Ms,nrow)))

  for (i in 1:l) {
    M <- Ms[[i]]

    ncol0 <- max.c - ncol(M)
```

```
    nrow0 <- max.r - nrow(M)

    if (ncol0>0) {
      col0 <- matrix(0,nrow(M),ncol0)
      M <- Ms[[i]] <- cbind(Ms[[i]],col0)
    }

    if (nrow0>0) {
      row0 <- matrix(0,nrow0,ncol(M))
      M <- Ms[[i]] <- rbind(Ms[[i]],row0)
    }
  }

  out <- Reduce("+",Ms)

  if (return.matrices) out <- list("sum"=out,"matrices"=Ms)

  out
}


# EXAMPLE: ############################################################3
#A <- matrix(1:10,nrow=2)
#B <- matrix(1:6,nrow=3)
#C <- matrix(1:6,nrow=1)
#D <- matrix(1:4)
#
#Ms <- list(A,B,C,D)
#
#
#sum.matrices(Ms)
#sum.matrices(Ms,T)
color.den <- function(den,from,to,col.den="black",col.area="red",add=F,...) {
  # Colors area under a density within an interval
  # den has to be a density object
  if (add) {
    #lines(den,col=col.den,...)
  } else {
    plot(den,col=col.den,...)
  }
  polygon(c(from, den$x[den$x>=from & den$x<=to], to),
          c(0, den$y[den$x>=from & den$x<=to], 0),
          col=col.area,border=col.den)
}

bound <- function(x, dens, return.x=TRUE){
  # Mickey Warner:
  # https://github.com/mickwar/r-sandbox/blob/master/mcmc/bayes_functions.R
  # returns the x-value in dens that is closest
  # to the given x
  if (return.x)
```

```
      return(dens$x[which.min(abs(dens$x-x))])

  # returns the y-value in dens at the closest x
  return(dens$y[which.min(abs(dens$x-x))])
}

col.mult = function(col1 = 0x000000, col2 = "gray50"){
  # Mickey Warner:
  # https://github.com/mickwar/r-sandbox/blob/master/mcmc/bayes_functions.R
  # returns the x-value in dens that is closest
  # to the given x
  if (is.character(col1))
      val1 = t(col2rgb(col1) / 255)
  if (is.numeric(col1))
      val1 = t(int2rgb(col1) / 255)
  if (is.character(col2))
      val2 = t(col2rgb(col2) / 255)
  if (is.numeric(col2))
      val2 = t(int2rgb(col2) / 255)
  rgb(val1 * val2)
}

int2rgb = function(x){
# int2rgb()
# convert an integer between 0 and 16777215 = 256^3 - 1,
# or between 0 and 0xFFFFFF
# this function is depended upon by col.mult
  # Mickey Warner:
  # https://github.com/mickwar/r-sandbox/blob/master/mcmc/bayes_functions.R
  # returns the x-value in dens that is closest
  # to the given x
  hex = as.character(as.hexmode(x))
  hex = paste0("#", paste0(rep("0", 6-nchar(hex)), collapse=""), hex)
  col2rgb(hex)
}

plot.post <- function(x,main=NULL,hpd=T,color="cornflowerblue") {
  mn.x <- round(mean(x),5)
  v.x <- round(var(x),3)
  den <- density(x)
  rng <- c(min(den$y),max(den$y))

  diff <- rng[2]-rng[1]
  main <- ifelse(is.null(main),"Posterior Distribution",
                         paste("Posterior Distribution for",main))
  plot(density(x),col=color,ylim=c(rng[1],rng[2]+diff*.3),lwd=3,
       main=main)
  legend("topleft",legend=c(paste("Mean =",mn.x),
                            paste("Variance = ",v.x)),bty="n")
  rng.x <- range(den$x)
  x.diff <- rng.x[2] - rng.x[1]
```

```
  opts <- par(no.readonly=T)
    left <- rng.x[1] + x.diff*2/3
    right <- rng.x[2]
    par(fig = c(grconvertX(c(left,right),from="user",to="ndc"),
                grconvertY(c(rng[2],rng[2]+diff*.3),from="user",to="ndc")),
        mar = c(.1,.1,1,.1), new = TRUE)
    plot(x,type="l",col="gray30",cex.main=.5,axes=F,main="Trace Plot")
    axis(1,cex.axis=.5)
    axis(2,cex.axis=.5)
  par(opts)

  color.den(den,rng.x[1],rng.x[2],col.den=color,col.area=color,add=T)
  if (hpd) {
    hpd <- get.hpd(x)
    color.den(den,hpd[1],hpd[2],col.den=col.mult(color),
              col.area=col.mult(color),add=T)
  }

  lines(c(mn.x,mn.x),c(0,bound(mn.x,den,ret=F)),lwd=2,col="red")
  #abline(v=mn.x,col="red",lwd=2)
}

get.hpd <- function(x,a=.05,len=1e3) {
  V <- matrix(seq(0,a,length=len))
  quants <- t(apply(V,1,function(v) quantile(x,c(v,v+1-a))))
  diff <- quants[,2]-quants[,1]
  min.d <- V[which.min(diff)]
  hpd <- quantile(x,c(min.d,min.d+1-a))
  hpd
}


plot.in.plot <- function(minor.plot,coords="topright",scale=1/3) {
  # coords = x1,y1,x2,y2
  # minor.plot is a function with no parameters that plots the smaller plot
  mar <- x1 <- x2 <- y1 <- y2 <- NULL
  s <- par("usr")
  if (is.numeric(coords)) {
    x1 <- coords[1]; x2 <- coords[2]
    y1 <- coords[3]; y2 <- coords[4]
  } else if (coords=="topright"){
    x1 <- s[1] + (s[2]-s[1]) * (1-scale)
    x2 <- s[2] - (s[2]-s[1]) * .01
    y1 <- s[3] + (s[4]-s[3]) * (1-scale)
    y2 <- s[4]
    mar <- c(.1,.1,1,.1)
  } else if (coords=="bottomright") {
    x1 <- s[1] + (s[2]-s[1]) * (1-scale)
    x2 <- s[2] - (s[2]-s[1]) * .01
    y1 <- s[3] + (s[4]-s[3]) *.05
```

```
    y2 <- y1 + (s[4]-s[3]) * (scale)
    mar <- c(1,.1,1,.1)
  } else if (coords=="topleft") {
    x1 <- s[1] + (s[2]-s[1]) * .05
    x2 <- x1 + (s[2]-s[1]) * (scale)
    y1 <- s[3] + (s[4]-s[3]) * (1-scale)
    y2 <- s[4]
    mar <- c(.1,1,1,.1)
  }else if (coords=="bottomleft") {
    x1 <- s[1] + (s[2]-s[1]) * .05
    x2 <- x1 + (s[2]-s[1]) * (scale)
    y1 <- s[3] + (s[4]-s[3]) *.05
    y2 <- y1 + (s[4]-s[3]) * (scale)
    mar <- c(1,1,1,.1)
  }
  opts <- par(no.readonly=T)
    par(fig = c(grconvertX(c(x1,x2),from="user",to="ndc"),
                grconvertY(c(y1,y2),from="user",to="ndc")),
        mar = mar, new = TRUE)
    minor.plot()
    #axis(1,cex.axis=.5)
    #axis(2,cex.axis=.5)
  par(opts)
}

#x <- rnorm(10000)
#plot(density(x),ylim=c(0,.5))
#
#minor <- function() {
#  plot(x,type="l",axes=F,main="Trace",cex.main=.8)
#  axis(1,cex.axis=.5)
#  axis(2,cex.axis=.5)
#}
#
##plotinplot(minor,c(1,4,.4,.5))
#plot.in.plot(minor,"topright")
#plot.in.plot(minor,"bottomright")
```

5.5 patterns.R

```
source("rfunctions.R")
source("ibp.R")
source("gibbs.R")

#Y <- as.matrix(read.table("phil3.dat"))
source("genDat.R")
elapsed.time <- system.time(out <- gibbs.post(Y,a=1,B=5000,burn=0,showProgress=T,
                                              plotProgress=T,a.a=3,a.b=2,
                                              siga=1,sigx=.5))

M <- out$Zs
```

```
alpha <- out$alpha
burn <- 1000#round(length(M) * .1)

n.col <- unlist(lapply(M,ncol))

pdf("out/traceplot.pdf")
  plot(n.col,type="l",main="Trace Plot: Number of Columns in Z",lwd=1,cex=.1,
       col="gray30",pch=20)
  mean.col <- round(mean(n.col[-(1:burn)]),4)
  var.col <-  round(var(n.col[-(1:burn)]),5)
  abline(h=mean.col,lwd=2,col="red")
  legend("bottomright",legend=c(paste("Mean=",mean.col),
                                paste("Variance =",var.col)),title.col="gray30",
                                title=paste("After Burn-in of",burn,":"),bty="n")
dev.off()

pdf("out/tracealpha.pdf")
  plot(alpha,type="l",main="Trace Plot: Alpha",lwd=1,cex=.1,
       col="gray30",pch=20)
  mean.a <- round(mean(alpha[-(1:burn)]),4)
  var.a <-  round(var(alpha[-(1:burn)]),5)
  abline(h=mean.a,lwd=2,col="red")
  legend("topleft",legend=c(paste("Mean=",mean.a),
                                paste("Variance =",var.a)),title.col="gray30",
                                title=paste("After Burn-in of",burn,":"),bty="n")
dev.off()

EAXZ <- function(X,Z,siga=1,sigx=sigX) {
  k <- ncol(Z)
  Ik <- diag(k)
  ZT <- t(Z)
  out <- solve(ZT%*%Z +(sigx/siga)^2 * Ik,ZT%*%X)
  #out <- solve(ZT%*%Z, ZT%*%X)
  out
}

Z.post <- M[-(1:burn)] # Burn in about 100
Z.post.mean <- sum.matrices(Z.post) / length(Z.post)
#Z.post.mean <- ifelse(Z.post.mean>runif(length(Z.post.mean)),1,0)
Z.post.mean <- ifelse(Z.post.mean>.5,1,0)
col0.ind <- which(apply(Z.post.mean,2,function(x) sum(x)==0))
if (length(col0.ind)>0) Z.post.mean <- Z.post.mean[,-col0.ind]
a.image(Z.post.mean)


one.A <- EAXZ(Y,Z.post.mean,siga=1,sigx=.5)
d2 <- 2
d1 <- ceiling(nrow(one.A)/d2)


pdf("out/postA.pdf")
```

```
  a.image(one.A,main="Posterior Mean for A")
dev.off()

plot.post.As <- function(one.A) {
  par(mfrow=c(d2,d1),mar=c(.5,.5,1,.5))
  for (i in 1:nrow(one.A)) {
    one.Ai <- matrix(one.A[i,],6,6) # matrix(Y[n,],6,6) = X[[n]]
    a.image(one.Ai,main=paste0("Latent Feat.",i),cex.main=.8)
    #a.image(one.Ai,main=paste0("Posterior Mean A",i),col=BLUE)
  }
  par(mfrow=c(1,1))
}

pdf("out/postA66.pdf")
  plot.post.As(one.A)
dev.off()

pdf("out/Y.pdf")
  a.image(Y,main="X")
dev.off()

pdf("out/postZ.pdf")
  a.image(Z.post.mean,main="Posterior Estimate for Z")
dev.off()

pdf("out/postAlpha.pdf")
  plot.post(alpha,"Alpha (after 1000 Burn)")
  #plot(density(alpha[-(1:burn)]),main="Posterior for Alpha",col="cornflowerblue",
  #     lwd=3)
dev.off()


post.ZA <- Z.post.mean %*% one.A

plot.post.ZA <- function(n) {
  par(mfrow=c(1,2))
    a.image(matrix(Y[n,],6,6),main=paste0("n=",n,": ",label[n]))
    a.image(matrix(post.ZA[n,],6,6),
            main=paste0("n=",n,":  ",toString(Z.post.mean[n,])))
  par(mfrow=c(1,1))
}

plot.post.each <- function() {
  opts <- par(no.readonly=T)
  par(mfrow=c(5,4),mar=c(.1,.1,1,.1))
    for (n in 15*(1:(nrow(Y)/15))) {
      a.image(matrix(Y[n,],6,6),main=paste0("n=",n,": ",toString(Z[n,])),cex.main=.8)
      a.image(matrix(post.ZA[n,],6,6),
              main=paste0("n=",n,":  ",toString(Z.post.mean[n,])),cex.main=.8)
     }
  par(opts)
```

```
}

pdf("out/postFriends.pdf")
  plot.post.each()
dev.off()


#a.image(matrix(apply(y2,2,mean),6,6))
a.image(Z.post.mean)
plot.post.As(one.A)
a.image(matrix(post.ZA[70,],6,6))


pdf("out/Z11postpred.pdf")
  X.post.pred <- lapply(as.list((burn+1):length(Z.post)),
                        function(i) Z.post[[i]]%*%EAXZ(Y,Z.post[[i]])+
                                    rnorm(prod(dim(Y)),0,.5))
  X.post.pred.11 <- unlist(lapply(X.post.pred,function(z) z[1,1]))
  plot.post(X.post.pred.11,main="Posterior Predictive for Z[1,1]")
dev.off()

pdf("out/qq.pdf")
  qq <- apply(as.matrix(X.post.pred.11),1,function(x) mean(x<X.post.pred.11))
  plot(density(qq,from=0,to=1),col=col.mult("cornflowerblue","grey80"),lwd=5,
       main="Density of Posterior Predictive Quantiles for Z[1,1]")
  ks.test(qq,"punif")
  legend("bottomleft",legend="P-val for KS-stat = 1",bty="n")
dev.off()
```

5.6 Data

The data can be downloaded at:
https://github.com/luiarthur/Fall2014/blob/master/Stat666/Final/code/Y.dat
Arthur! MAKE SURE THIS IS CURRENT!