A Bayesian Alternative to Hierarchal Clustering

Authors: Thomas Yan, Mark Christopher Uy, Gregory Reid, Jose Carlos Dairo

Abstract

A common problem in many analyses is determining how to group data, which observations belong to which group and how many groups are appropriate. In these problems, often the true grouping and group memberships of each observation are unknown and the data for that is simply unavailable. This is the reason why clustering methods are required to solve these sorts of problems, by creating groups using similarities between observations. An example of such a problem is trying to determine the number of sub-species of an animal in a habitat when the sub-species can not be easily distinguished visually.

One method of clustering is using Bayesian inference to estimate the posterior probability to assign the most likely group for each observation as its designated cluster. A common method of estimating probabilities is Monte Carlo methods, however there are cases where its impossible to generate the independent and identically distributed samples required for Monte Carlo.

This report's goal is to create a Markov Chain Monte Carlo (MCMC) method, specifically using a Gibbs Sampler approach, to perform clustering on data with unknown responses by estimating a posterior distribution for the class memberships. By using MCMC instead of just Monte Carlo methods, this approach is able to work in more problems and is able to estimate posterior probabilities in situations where Monte Carlo methods would not be possible.

Introduction

When trying to group data where the true group memberships, if they exist, are unknown, then methods used to perform this task fall under the umbrella term, clustering, which itself is a subset of unsupervised machine learning (B. Zhang, 2003). There are already many well known methods of clustering which are widely used and effective in many contexts. These include, k-means clustering, mean-shift clustering, density-based clustering and heirarchal clustering. However, a prominent drawback for these methods is that they are heuristic and thus have no way of drawing formal inferences from their results (K. M. A. Patel and P. Thakral, 2016).

When we elect to use model-based clustering, it is possible to draw interpretations from a statistical point of view (Bouveyron, 2013). A common approach is the maximum likelihood method where one would estimate the model parameters by maximizing the log likelihood function using an Expectation Maximization (EM) Algorithm and then estimating cluster memberships using the maximum a posterioiri rule (A.Samé, 2009).

As dimensionality and data size increases, the computational time needed to perform EM increases immensely. Neal and Hitton introduced a way to speed up the EM algorithm by recalculating the distribution for only one unobserved variable at each expectation step as opposed to calculating for all unobserved variables (R.M. Neal and Hinton, 1998). Bradley et al. proposed an implementaion of the EM Algorithm that was scalable to large databases with limited memory buffers (Bradley et al., 1998). While these solutions focused on improving the EM algorithm for specific cases, our paper is focused on developing an approach which does not involve any EM algorithm. We propose a solution that uses an exclusively bayesian approach by implementing a Gibb's sampler to estimate the parameters of our model as well as estimate the cluster memberships of our observations. The following sections will showcase what assumptions were made to make the simplify the problem enough to be able to use of a gibb's sampler, its performance on a real data set and lastly a discussion on the merits and flaws of our proposed solution.

Methodology

The first assumption this method makes is that the observed data follows a mixture normal distribution, $x_i \sim \sum_{k=1}^{K} \rho_k N(\mu_k, \Sigma_k), i = 1, ..., N$ and are independent and identically distributed. Then following this, we take a Bayesian inference approach where the unknown, but true cluster memberships follows a multinomial distribution, $z_i \sim \text{Multinomial}(K, \rho)$ where $\rho = (\rho_1, ..., \rho_K)$ and $(N_1, ..., N_k) \sim \text{Multinomial}(N, \rho)$. However, a problem with this is that it quickly suffers from the curse of dimensionality as the dimension of x_i increases and causes this mixture-normal model to fail.

The solution to this was to instead assume that the x's belonged to a family of models $f(x_i|\theta_i)$, where the distribution of each x_i depended on its own parameter θ_i. Then instead of assuming that the x's followed a mixture-normal, we assume that the θ's follow a mixture-normal. Where $\theta_i \in \mathbb{R}^{\text{p}}$, such that p is much smaller than N and will not severely suffer from the curse of dimensionality. As in:

$$\theta_i \overset{iid}{\sim} \sum_{k=1}^{K} \rho_k N(\mu_k, \Sigma_k), i = 1, ..., N$$

$$x_i|\theta_i \overset{ind}{\sim} f(x_i|\theta_i)$$

Where θ_i and $f(x_i|\theta_i)$ are chosen based on the data and problem at hand and needs to be decided before the Gibbs sampler can be used. In this setup, θ_i is an unknown and random parameter and this assumed model is also known as a "random-effects" model. The standard fixed and unknown parameters that are used to define cluster sizes and membership probabilities are $\psi_k = (\rho_k, \mu_k, \Sigma_k)$.

However, the problem is still too difficult to deal with, and impossible to do analytically, and requires further simplifcation. Let $\hat{\theta}_i = \arg\max_\theta f(x_i|\theta)$, the maximum likelihood estimate of θ_i from our observed data, and $\hat{V}_i = -[\frac{\partial}{\partial \theta^2} \log(f(x_i|\hat{\theta}_i))]^{-1}$ denote the observed Fisher information.

Then we can approximate our above clustering model simply using the asymptotic properties of MLE which gives the following results:

$$\theta_i \overset{iid}{\sim} \sum_{k=1}^{K} \rho_k N(\mu_k, \Sigma_k), i = 1, ..., N$$

$$\hat{\theta}_i|\theta \overset{ind}{\sim} N(\theta_i, \hat{V}_i)$$

This gives the result: $\hat{\theta}_i|\{z_i = k\} \sim N(\mu_k, \hat{V}_i + \Sigma_k)$. Then using this and the previous assumptions, and letting $\hat{\theta}_i = y_i$ be our observed data, we have the result:

$$z_i \overset{iid}{\sim} \text{Multinomial}(K, \rho)$$

$$\theta_i|z_i \overset{ind}{\sim} N(\mu_{z_i}, \Sigma_{z_i})$$

$$y_i|\theta_i \overset{ind}{\sim} N(\theta_i, V_i)$$

Where $Y = (y_1, ..., y_N) = (\hat{\theta}_1, ..., \hat{\theta}_N)$ is the observed data, and both $z = (z_1, ..., z_N)$ and $\Theta = (\theta_1, ..., \theta_N)$ are the missing, unobserved data. Then we can construct a complete likelihood function for our parameters $\Psi = (\psi_1, ..., \psi_K), \psi_k = (\rho_k, \mu_k, \Sigma_k)$, by assuming that the missing data was observed:

$$l(\Psi|z, \Theta, Y) = \log[p(Y, \Theta, z, |\Psi)] = \log[\Pi_{i=1}^{N} p(y_i|\theta_i, \Psi) * p(\theta_i|z_i, \Psi) * p(z_i|\Psi)]$$

Where we know the distributions of each of these density functions from our previous assumptions. Now, the problem is to estimate the parameters when we have missing data, which will be done using Bayesian inference and an noninformative improper prior:

$$\pi(\Psi) \ \alpha \ \Pi_{k=1}^{K} |\Sigma_k|^{-(\nu_k+p+1)/2} \exp(-\frac{1}{2}\text{tr}(\Sigma_k^{-1}\Omega_k))$$

Where p is the size of each θ_i, Ω_k is a p by p symmetric positive definite matrix and ν_k are constants. There are certain ways to choose Ω_k and ν_k, but we will skip the details.

Then using the prior and the above log likelihood, we can construct a posterior probability and a respective likelihood, $p(\Psi, z, \Theta|Y) \ \alpha \ p(Y, \Theta, z|\Psi)\pi(\Psi)$, using Bayes. Then we can simplify this posterior likelihood by conditioning on different parameters.

If we let $\Phi = (Y, \Theta, z, \Psi)$ denote all the variables in the model and $\Phi\backslash\{\theta_i\}$ denote all variables excluding θ_i. Then by simplifying the complete posterior likelihood by assuming all given terms are constant, it can be shown that:

$$\theta_i|\Phi\backslash\{\theta_i\} \sim N(G_i(y_i - \mu_{z_i}) + \mu_{z_i}, G_i V_i) \text{ where } G_i = \Sigma_{z_i}(V_i + \Sigma_{z_i})^{-1}$$

And that when conditioned on all other variables, all θ_i's are conditionally independent, as in $\theta_i|\Phi\backslash\{\theta_i\} \overset{\text{distribution}}{=} \theta_i|\Phi\backslash\{\Theta\}$.

A similar result can be shown for the other variables where:

$$\mu_k|\Phi\backslash\{\mu_k\} \overset{ind}{\sim} N(\bar{\theta}_k, \Sigma_k/N_k) \text{ where } N_k = \sum_{i=1}^{N} \mathbb{I}(\text{z}_\text{i} = \text{k})$$

$$\Sigma_k|\Phi\backslash\{\Sigma_k\} \overset{ind}{\sim} \text{InvWish}(\Omega_k + \sum_{i:z_i=k} (\theta_i - \mu_K)(\theta_i - \mu_k)', N_k + \nu_k)$$

$$\rho_k|\Phi\backslash\{\rho_k\} \overset{ind}{\sim} \text{Dirichlet}(\alpha) \text{ where } \alpha = (N_1 + 1, ..., N_K + 1)$$

$$z_i|\Phi\backslash\{z_i\} \overset{ind}{\sim} \text{Multinomial}(K, \lambda_i) \text{ where } \lambda_{ik} = \frac{\exp(\kappa_{ik})}{\sum_{m=1}^{K} \exp(\kappa_{im})}$$

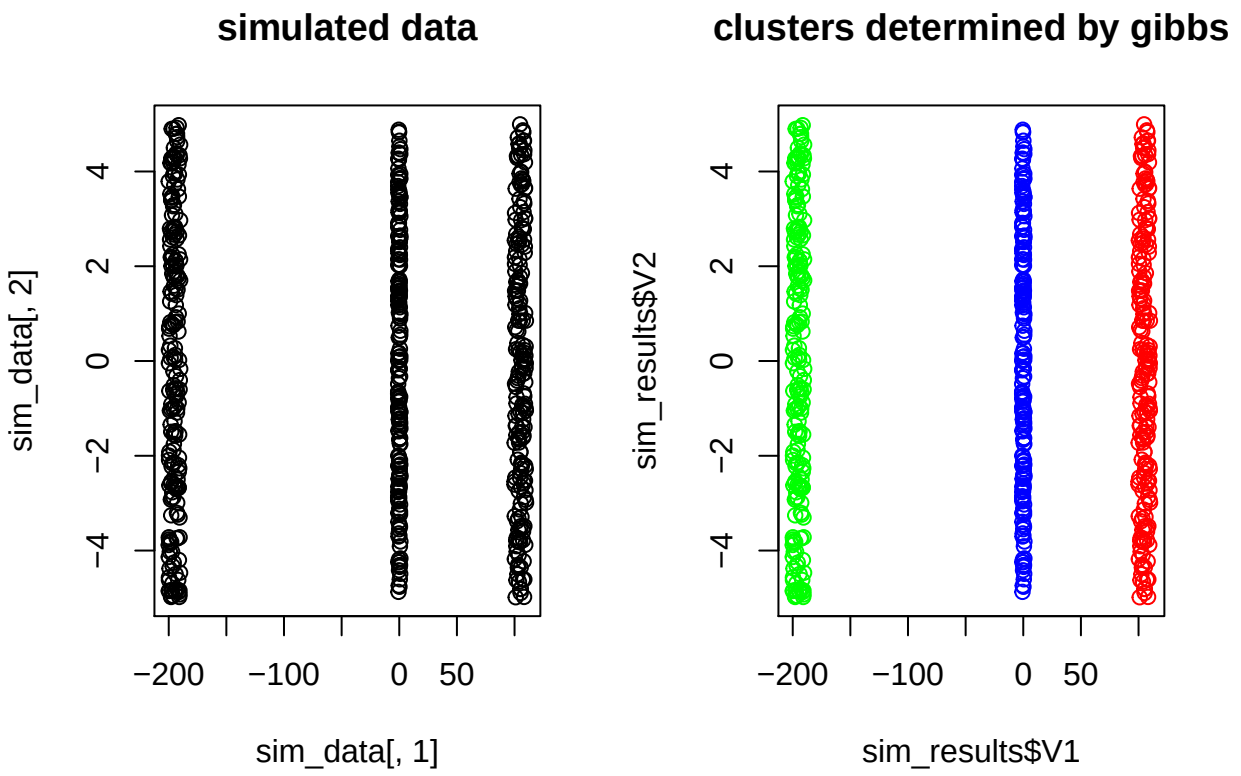
Where $\kappa_{ik} = \log(P(z_i = k|\Phi\backslash\{z_i\})) = \log(\rho_k) - \frac{1}{2}[(\theta_i - \mu_k)'\Sigma_k^{-1}(\theta_i - \mu_k)' + \log|\Sigma_k|]$

And each of these variables are conditionally independent, which allows us to sample everything in 5 steps: simultaneously sample all $\theta_i, i = 1, ..., n$, then simultaneously sample all $mu_k, k = 1, ..., K$, and so on and so forth for each of the variables. Where we repeat these 5 steps every iteration of the Gibbs sampler, for M desired iterations.

Then we can estimate the posterior probability by $P(z_i = k|Y) \approx \hat{\lambda}_{ik}$ where $\hat{\lambda}_{ik} = [\hat{\Lambda}]_{ik}, \hat{\Lambda} = \frac{1}{M}\sum_{m=1}^{M} \Lambda^{(m)}$. Where $\Lambda^{(m)}$ is the Λ matrix created in the mth iteration used to sample z.

Results

To demonstrate that the gibbs sampler can detect distinguishable clusters, we simulated numerical data such that the clusters are clear separated from each other.



Now we use the proposed Gibbs Sampler to determine the three clusters in this dataset.

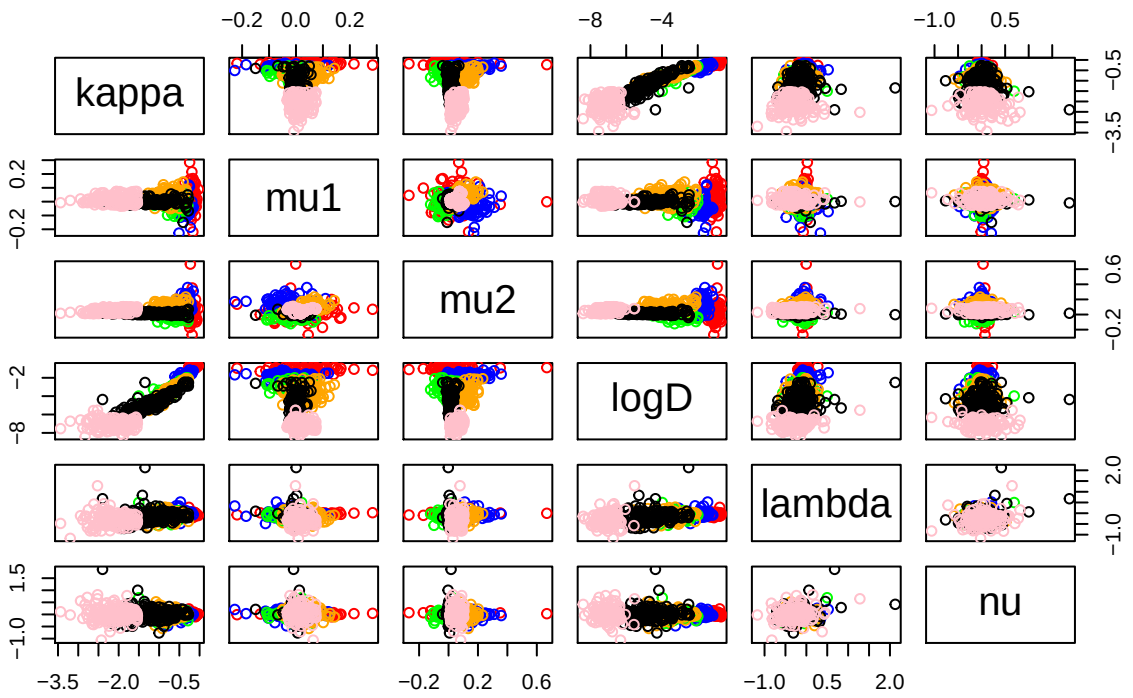
In reality, it is often difficult to determine if there are distinct clusters in the dataset. If the data has many dimensions, it is very difficult to visually determine if there are distinct clusters, as was the case with the data simulated earlier.

A real dataset that we will use in this case is dataset used by researchers trying to model microparticle trajectories with a linear drift fractional Brownian motion model (Hill et al., 2014).

This dataset contains the parameter estimates to the trajectories of 668 microparticles, their weight percentage of human bronchial epithelial (hbe) mucosal fluid, and their observed fisher information matrices.

In this particular dataset, all observations had weight percentage of hbe as 1.5%, 2%, 2.5%, 3%, 4%, 5%.

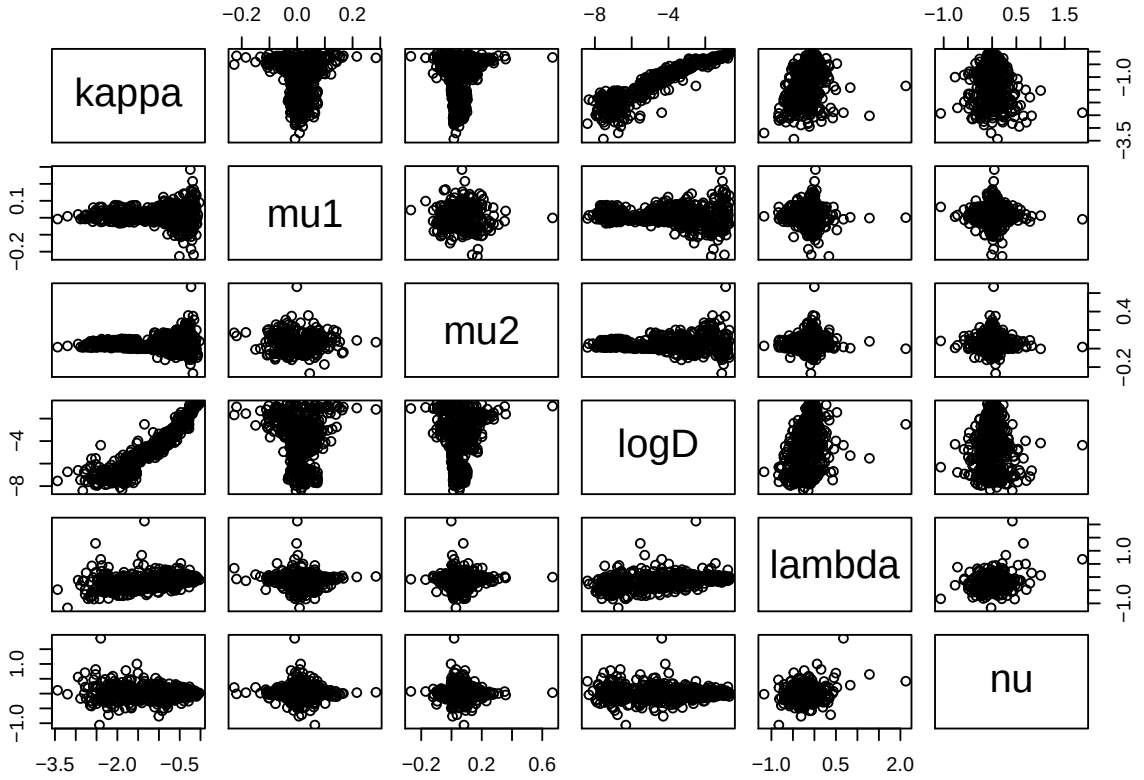
We will see the gibbs sampler can the group the observations into 6 clusters and see if each cluster corresponds to a particular weight percentage of hbe.



Based on the pairs plot of the data, coloured based on the true weight clustering. It looks as though there is little promise to clustering based on the provided data. For mu1, mu2, lambda, and nu, the pairs plots of each against weight shows that they all overlap, that there is little promise to clustering using these variables. The within cluster ranges for the predictor values seems to heavily overlap across weight clusters which will likely make it very difficult to do any proper clustering.

For kappa, there looks to be potential to do some basic clustering, where kappa seems to have some potential to split the data into cluster 6 vs not cluster 6. Similarly, logD seems like it could potentially make clusters of cluster 1, cluster 6 or the rest.

However, these possibilities are assuming that the model knows the true cluster memberships. The Gibbs sampler we created is for a clustering method, which means that the true cluster label is unknown. Thus, ignoring the first column and first row of the pairs plot and removing the colouring indicating the true clusters, we can look at a new pairs plot.



This pairs plot, without coloring, showcases the clustering problem where there does not seem to be any unique pairwise relationship between the predictors. Many of the pairwise plots simply look like single homogenous entities which will likely make it difficult to identify very distinct clusters.

Despite this, we will see what clusters are determined by the gibbs sampler, and if they match the clusters determined by hbe weight percentage.

As a comparison, we also include an RStan based implementation of the above model, and kmeans plus plus (km++) algorithm to see if any of these 2 other methods perform better.

A brief overview of of the km++ algorithm is as follows:

Step 1: Choose one center uniformly at random from among the data points.

Step 2: For each data point x, $d(x)$, the distance between x and the nearest center that has already been chosen.

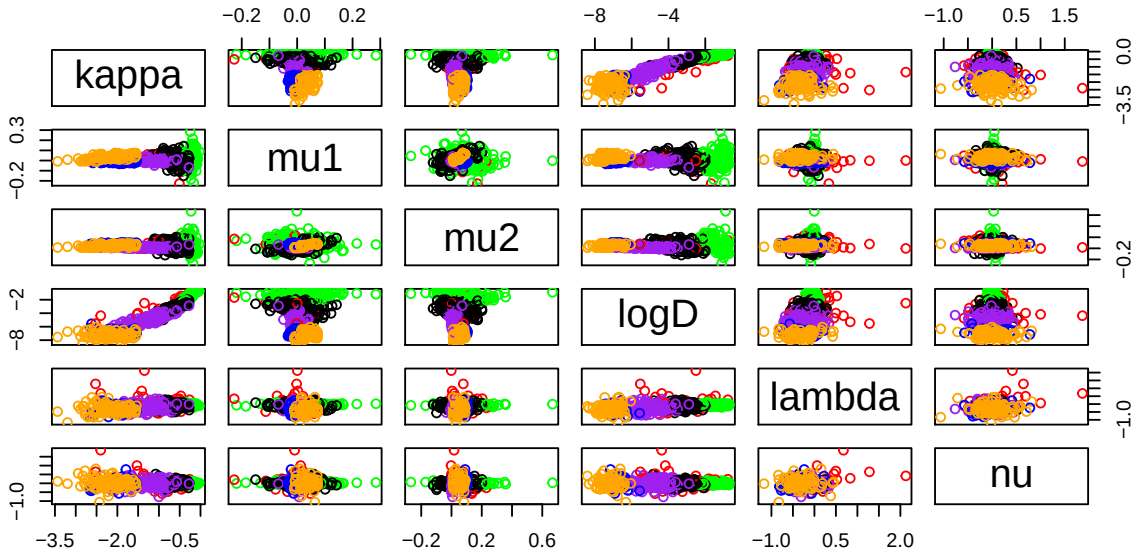
Step 3: Choose one new data point at random as a new center, using a weighted probability distribution where a point x is chosen with probability proportional to $d(x)^2$.

Step 4: Repeat Steps 2 and 3 until k centers have been chosen.

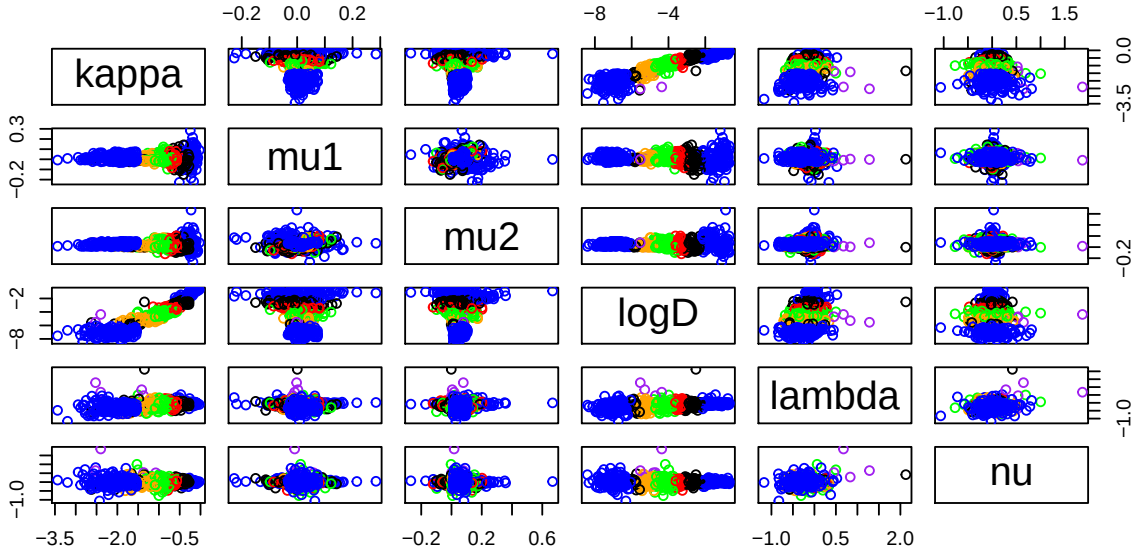
Step 5: With the initial centers have been chosen, continue using the regular k-means clustering.

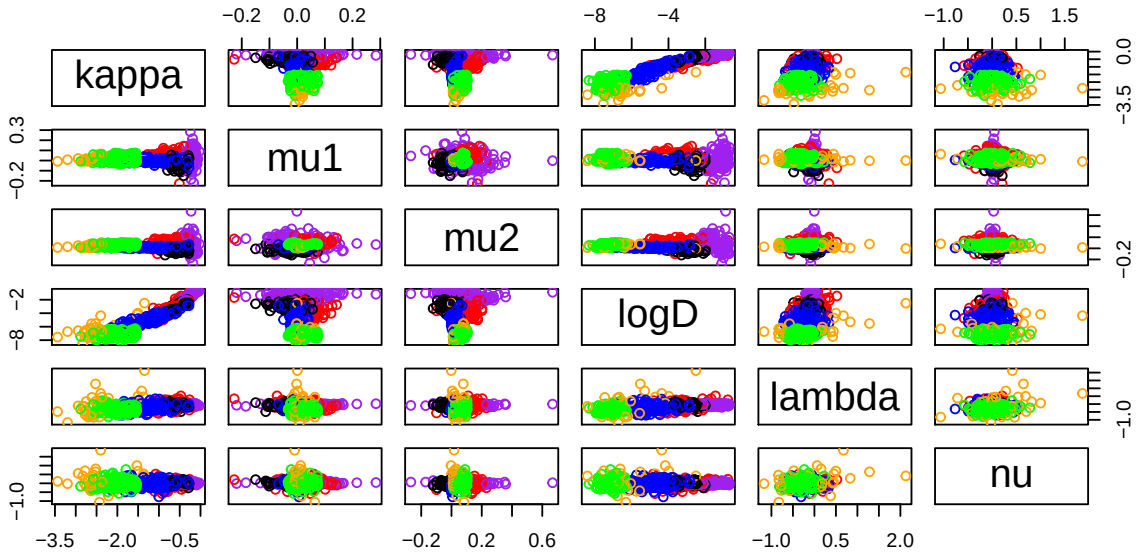
(Arthur D., 2007)

Clusters determined by gibbs



clusters determined by kmpp





Distribution of cluster memberships determined by Gibbs Sampler:

GibbsSampler.cluster	GibbsSampler.Freq
1	24
2	49
3	126
4	127
5	170
6	172

Distribution of cluster memberships determined by K-means++:

KMeansPlusPlus.cluster	KMeansPlusPlus.Freq
1	60
2	320
3	115
4	77
5	90
6	6

Distribution of cluster memberships determined by RStan Hamiltonian Based Model:

RStan.cluster	RStan.Freq
1	88
2	183
3	157
4	27
5	91
6	122

As expected, the clusters determined by all 3 algorithms overlap each other heavily.

Due to this, it is difficult to assess if the gibbs sampler, RStan sampler or the km++ fit the data well, because there is no true cluster affiliation to evaluate against. Additionally, the data with the given variables cannot clearly be subdivided into distinguishable groups.

However, it is rather clear that km++ performs the worst since their pairs plot is heavily dominated by the 2nd class (Blue Color) even to the extent that 2 disjoints parts of the data are labelled as part of the same class. In contrast our Gibbs sampler, and the RStan stampler show much more distinct group clusters, which more closely matches the ground truth groups. Despite the similar performance between the 2 samplers, the sampling time required by the gibbs sampler was an order of magnitude less than the time needed by the RStan sampler. This is one clear advantage of our new proposed implementation.

Overall, the results from all 3 clustering methods suggest that given the existing parameters, observations with particular hbe weight percentages are not very distinguishable from other observations with different hbe weight percentages.

This does not rule out the possibility that hbe weight percentages or other variables can be used to group up microparticle trajectories, but for this to occur, latent variables would need be made known.

Discussion

This paper proposes an alternative to model-based hierarchal clustering that does not utilize any EM algorithms, which become increasingly expensive, but rather opts for a Bayesian approach.

By making some assumptions, we managed to reduce the complex problem of model-base hierarchal clustering into a simpler problem solved by a Gibbs Sampler.

The main benefit to this is that our Gibbs Sampler method is computationally cheaper as we avoid inverting many large matrices yet still performs the same task of hierarchal clustering.

However, as with all Bayesian analysis, our method requires arbitrarily selected prior distributions to begin. This makes any analysis done with this approach subjective.

To showcase our method, we applied it onto the hbe dataset and compared it against k-means plus plus algorithm and also an RStan Implementation of the same. The goal of this application is to see if the data can be clustered in such a way that each cluster corresponds to observations with a particular weight percentage. Through the results of both methods, we discovered that observations in the data could not be easily grouped into clusters distinguished by a particular weight percentage variable. This application does not assert the effectiveness of our proposed method or lack thereof.

With that said, we outline one potential direction for research. One particular research direction would be construct similar clustering methods with different gibbs samplers that have different prior distributions and compare the performances against each other. The goal of this research would not be to determine which gibbs sampler method works best, but rather to discover what would be the most appropriate gibbs sampler for particular situations and document them.

Appendix

```
init_z <- function(y, K, max_iter = 10, nstart = 10) {
  # init++
  N <- nrow(y)
  p <- ncol(y)
  x <- t(y) # easier to use columns
  centers <- matrix(NA, p, K) # centers
  icenters <- rep(NA, K-1) # indices of centers
  minD <- rep(Inf, N) # current minimum distance vector
  # initialize
  icenters[1] <- sample(N,1)
  centers[,1] <- x[,icenters[1]]
  for(ii in 1:(K-1)) {
    D <- colSums((x - centers[,ii])^2) # new distance
    minD <- pmin(minD, D)
    icenters[ii+1] <- sample(N, 1, prob = minD)
    centers[,ii+1] <- x[,icenters[ii+1]]
  }
  centers <- t(centers)
  colnames(centers) <- rownames(x)
  # run kmeans with these centers
  km <- kmeans(x = y, centers = centers, nstart = nstart, iter.max = max_iter)
  km$cluster
}
```

```
set.seed(440)
source("gibbs-wrapper.R")
require(rstan)
# install.packages("plot3D") # sometimes require() does not install the plot3D package, if so, run this
```

```
#-----------------------------------------
# generate simulated data
# ----------------------------------------
n <- 600
A <- runif(n = 200,min = -1, max = 1)
B <- runif(n = 200, min = 100, max = 110)
C <- runif(n = 200, min = -200, max = -190 )
D <- runif(n = n, min = -5, max = 5)
# note that the data generated and stored in variables A,B,C are clearly in 3 distinct clusters
sim_data <- matrix(nrow = n,ncol = 2)
sim_data[,1] <- c(A,B,C)
sim_data[,2] <- D
var_sim1 <- var(sim_data[,1])
var_sim2 <- var(sim_data[,2])
cov_sim <- cov(sim_data[,1],sim_data[,2])
# inverse of fisher obvs is the asymptotic covariance matrix,
# thus  we will use the inverse of the observed covariance matrix as an estimate for the fisher obvs
sim_obs_info <- solve(matrix(data = c(var_sim1,cov_sim,
                                cov_sim,var_sim2),nrow = 2,ncol=2,byrow=TRUE))
```

```r
# fit gibbs sampler
gibbs_fit2 <- gibbsSampler(data=sim_data, V=sim_obs_info, burnin_period=1e2, numIter=1e3, K=3, Theta.ou
# get estimated cluster affiliation from gibbs sampler
fitted_clusters4 <- numeric(length=n)
for (i in 1:600){
  fitted_clusters4[i] <- which.max(gibbs_fit2$Lambda[i, ])
}
# indices
index1 <- 1:n
# error rate
sim_results <- as.data.frame(sim_data)
sim_results$col <- fitted_clusters4
sim_results$col[sim_results$col == 1] <- "red"
sim_results$col[sim_results$col == 2] <- "green"
sim_results$col[sim_results$col == 3] <- "blue"

par(mfrow=c(1,2))
plot(x=sim_data[,1],y=sim_data[,2], main = "simulated data")
plot(x=sim_results$V1,y = sim_results$V2,col = sim_results$col, main="clusters determined by gibbs")
par(mfrow=c(1,1))
```

```r
hbe_fit <- readRDS("hbe-fbm_fit_v1.rds")
N <- length(hbe_fit)
p <- length(hbe_fit[[1]]$coef)
# Extract data
wts <- vector(length=N)
col_names <- c("kappa","mu1","mu2","logD","lambda","nu") # set column names
theta_hat <- matrix(nrow=N, ncol=p)
colnames(theta_hat) <- col_names
obs_info <- array(dim=c(p, p, N))
for (i in 1:N){
  theta_hat[i, ] <- hbe_fit[[i]]$coef
  wts[i] <- hbe_fit[[i]]$wt
  obs_info[, , i] <- hbe_fit[[i]]$vcov
}
# Transform the wts into numeric labels
lab <- unique(wts)
K <- length(lab)
cluster <- numeric(length=N)
for (i in 1:N){
  if (wts[i] == lab[1]){
    cluster[i] <- 1
  }
  else if (wts[i] == lab[2]){
    cluster[i] <- 2
  }
  else if (wts[i] == lab[3]){
    cluster[i] <- 3
  }
  else if (wts[i] == lab[4]){
    cluster[i] <- 4
  }
  else if (wts[i] == lab[5]){
```

```
    cluster[i] <- 5
  }
  else{
    cluster[i] <- 6
  }
}
# getting true cluster affiliations
real_N_k <- numeric(length=K)
real_cols <- rep(NA,N) # colors for true cluster affiliations
for (k in 1:K){
  real_N_k[k] <- sum(cluster == k)
}
# assigning colours to each cluster
real_cols[cluster == 1] = "red"
real_cols[cluster == 2] = "blue"
real_cols[cluster == 3] = "green"
real_cols[cluster == 4] = "orange"
real_cols[cluster == 5] = "black"
real_cols[cluster == 6] = "pink"
pairs(theta_hat,col=real_cols, main="pairs plot clustered by wt") # pairs plot
```

```
# fit gibbs sampler
gibbs_fit <- gibbsSampler(data=theta_hat, V=obs_info, burnin_period=1e3, numIter=1e4, K=6, Theta.out=FA

# get estimated cluster affiliation from gibbs sampler
fitted_clusters <- numeric(length=N)
for (i in 1:N){
  fitted_clusters[i] <- which.max(gibbs_fit$Lambda[i, ])
}


# clustering real data with kmeans++ algorithm
# note since kmeans starts with random centroids, individual kmeans can have different results.
# We will conduct many kmeans and aggregate the results
#' Initial cluster allocation.
#'
#' Initializes the clusters using the kmeans++ algorithm of Arthur & Vassilvitskii (2007).
#'
#' @param y An `N x p` matrix of observations, each of which is a column.
#' @param K Number of clusters (positive integer).
#' @param max_iter The maximum number of steps in the [stats::kmeans()] algorithm.
#' @param nstart The number of random starts in the [stats::kmeans()] algorithm.
#'
#' @return A vector of length `N` consisting of integers between `1` and `K` specifying an initial clus
#' @author Martin Lysy
#' @references Arthur, D., Vassilvitskii, S. "k-means++: the advantages of careful seeding" *Proceeding
init_z <- function(y, K, max_iter = 10, nstart = 10) {
  # init++
  N <- nrow(y)
  p <- ncol(y)
  x <- t(y) # easier to use columns
  centers <- matrix(NA, p, K) # centers
  icenters <- rep(NA, K-1) # indices of centers
```

```
  minD <- rep(Inf, N) # current minimum distance vector
  # initialize
  icenters[1] <- sample(N,1)
  centers[,1] <- x[,icenters[1]]
  for(ii in 1:(K-1)) {
    D <- colSums((x - centers[,ii])^2) # new distance
    minD <- pmin(minD, D)
    icenters[ii+1] <- sample(N, 1, prob = minD)
    centers[,ii+1] <- x[,icenters[ii+1]]
  }
  centers <- t(centers)
  colnames(centers) <- rownames(x)
  # run kmeans with these centers
  km <- kmeans(x = y, centers = centers, nstart = nstart, iter.max = max_iter)
  km$cluster
}
max_iters <- 10000  # set number of kmeans to run
kmeans_results <- matrix(nrow = N,ncol = max_iters) # pre-allocate matrix
# simple function to calculate the mode in vector of numbers
Mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}
# get results from max_iters kmeans
for (iter in 1:max_iters){
  kmeans_results[,iter] <- init_z(y = theta_hat,K = 6)
}
fitted_clusters1 <- matrix(nrow = N,ncol = 1) # pre-allocate matrix to store estimated clusters
# the mode cluster membership for each observation becomes the estimate cluster membership
for (row in 1:N){
  fitted_clusters1[row,] <- Mode(kmeans_results[row,])
}

################################################################
#                      Stan

finMixData_fit<- readRDS("fitMixDataStan.rds")

extract_rho<- extract(finMixData_fit)$rho
extract_theta<-  extract(finMixData_fit)$theta
extract_mu<-  extract(finMixData_fit)$mu
extract_sigma<-  extract(finMixData_fit)$sigma

numIter<- 24000
Lambda<-matrix(0, nrow =N, ncol=K)

for ( i in 1:numIter){
  cur_rho<- extract_rho[i,]
  cur_mu<- extract_mu[i, ,]
  cur_theta<- extract_theta[i, ,]
  cur_sigma<- extract_sigma[i, , ,]
  K<-length(cur_rho)
  Kappa<-matrix(0, nrow=N, ncol=K)
```

```
  inv_sigma <- vector("list", length=K)

  for (k in 1:K){
    inv_sigma[[k]] <- solveV(cur_sigma[k,,], ldV=TRUE)
  }

  log_det <- sapply(inv_sigma, FUN=function(z){ z$ldV })
  log_rho <- log(cur_rho)

  for (j in 1:N){
    for (k in 1:K){
      temp <- as.matrix(cur_theta[j, ] - cur_mu[k, ])
      Kappa[j, k] <- log_rho[k] - (t(temp) %*% inv_sigma[[k]]$y %*% temp + log_det[k]) / 2
    }
  }

  cur_Lambda<- exp(Kappa)

  if ( sum(is.nan(cur_Lambda)) > 0){
    print("NaN found.")
    stop()
  }

  cur_Lambda <- cur_Lambda / rowSums(cur_Lambda)

  Lambda <- Lambda + cur_Lambda

  #if(mod(i,120) == 0) print(i)
}

Lambda <- Lambda/ numIter

fitted_clustersRSTAN <- numeric(length=N)
for (i in 1:N){
  fitted_clustersRSTAN[i] <- which.max(Lambda[i, ])
}

###############################################################################
real_rstan_cols <- rep(NA,N)
real_gibbs_cols <- rep(NA,N)
real_kmpp_cols <- rep(NA,N)
# assigning colours to each cluster
real_gibbs_cols[fitted_clusters == 1] = "red"
real_gibbs_cols[fitted_clusters == 2] = "blue"
real_gibbs_cols[fitted_clusters == 3] = "green"
real_gibbs_cols[fitted_clusters == 4] = "orange"
real_gibbs_cols[fitted_clusters == 5] = "black"
real_gibbs_cols[fitted_clusters == 6] = "purple"

real_kmpp_cols[fitted_clusters1 == 1] = "red"
real_kmpp_cols[fitted_clusters1 == 2] = "blue"
real_kmpp_cols[fitted_clusters1 == 3] = "green"
real_kmpp_cols[fitted_clusters1 == 4] = "orange"
```

```
real_kmpp_cols[fitted_clusters1 == 5] = "black"
real_kmpp_cols[fitted_clusters1 == 6] = "purple"

real_rstan_cols[fitted_clustersRSTAN == 1] = "red"
real_rstan_cols[fitted_clustersRSTAN == 2] = "blue"
real_rstan_cols[fitted_clustersRSTAN == 3] = "green"
real_rstan_cols[fitted_clustersRSTAN == 4] = "orange"
real_rstan_cols[fitted_clustersRSTAN == 5] = "black"
real_rstan_cols[fitted_clustersRSTAN == 6] = "purple"

# plot side-by-side pairs plots
par(mfrow=c(3,1))
pairs(x = theta_hat,col = real_gibbs_cols, main = "Clusters determined by gibbs")
pairs(x = theta_hat,col = real_kmpp_cols, main = "clusters determined by kmpp")
pairs(x = theta_hat,col = real_rstan_cols, main = "clusters determined by RStan")
par(mfrow=c(1,1))

print("Distribution of cluster memberships determined by Gibbs Sampler")
table(fitted_clusters)
print("Distribution of cluster memberships determined by K-means++")
table(fitted_clusters1)
print("Distribution of cluster memberships determined by RStan Hamiltonian Based Model")
table(fitted_clustersRSTAN)
```

References

Charles Bouveyron, Camille Brunet. Model-Based Clustering of High-Dimensional Data: A review. Computational Statistics and Data Analysis, Elsevier, 2013, 71, pp.52-78. ff10.1016/j.csda.2012.12.008ff. ffhal-00750909f

K. M. A. Patel and P. Thakral, "The best clustering algorithms in data mining," 2016 International Conference on Communication and Signal Processing (ICCSP), Melmaruvathur, 2016, pp. 2042-2046.

A. Samé, "Grouped data clustering using a fast mixture-model-based algorithm," 2009 IEEE International Conference on Systems, Man and Cybernetics, San Antonio, TX, 2009, pp. 2276-2280.

R.M. Neal and Hinton, A view of the EM algorithm that justifies incremental, sparse and other variants. In M.I. Jordan, editor, Learning in Graphical Models, 355-368, Kluwer, Dordrecht, 1998

P.S. Bradley, M.U. Fayyad and C.A. Reina, Scaling EM (Expectation Maximization) clustering to large databases. Technical Repport MSRTR-98-35, Microsoft Research, 1998.

B. Zhang, "Comparison of the Performance of Center-Based Clustering Algorithms", Advances in Knowledge Discovery and Data Mining, Lecture Notes in Computer Science, Volume 2637/2003, pp. 569, 2003.

Hill, D.B., Vasquez, P.A., Mellnik, J., McKinley, S.A., Vose, A., Mu, F., Henderson, A.G., Donaldson, S.H., Alexis, N.E., Boucher, R.C., and Forest, M.G. (2014). "A biophysical basis for mucus solids concentration as a candidate biomarker for airways disease." PLoS ONE , 9(2): e87681

Arthur, D. & Vassilvitskii, S. (2007). k-means++: the advantages of careful seeding.. In N. Bansal, K. Pruhs & C. Stein (eds.), SODA (p./pp. 1027-1035), : SIAM. ISBN: 978-0-898716-24-5