Model-based Clustering using Gibbs Sampler and other algorithms

0. Abstract

Model-based clustering is a statistical method to identify and create groups based on similarities of attributes. As increased computing power enabled many machine learning algorithms to be used in practice. A couple of functions in R that are famous for handling clustering problems are 'kmeans' from built-in R package and 'mclust' from Mclust package. However, this method may not be the best choice when the dimension of dataset increases, so called cures of dimensionality. In this project, we implemented our version of Gibbs sampling and to a Gaussian mixture model and estimate each observation's cluster distribution by Bayesian inference. Its performance is compared with kmeans and mclust's performances on how well they distinguish clusters. To compare their performances, we used real-world dataset, and it turned out that our Gibbs Sampler have very weak clustering ability. We addressed briefly why Gibbs Sampler has to have poor performance in this project.

1. Introduction

Classification and clustering are performed on data to identify labels/clusters of observations based on their similarities. Classification requires part of data to have known membership information to train the model. Clustering can be considered as special case of classification with no training data. A number of algorithms have been proposed and implemented by researchers. One most popular and famous algorithm being used for clustering is called K-Means Clustering(Hartigan and Wong (1979)), where clusters are determined by the distances from the centroids. Another algorithm, which will be studied in this project, is model-based classification(McNicholas (2011)). This algorithm assumes Gaussian mixture model where parameters in the model are believed to have certain distributions. The estimates of these parameters are evaluated through Bayesian inference where prior distribution affects the value of our interests called posterior. In this project, MCMC(Andrieu et al. (2003)) method, which attemps to update parameters of interest through sufficient number of iterations. Once the MCMC is converged, we can estimate the posteriors by simply averaging it. In this project, parameter that we want to estimate is a probability of each observation to be in each cluster, and group membership can be assigned based on these probabilities. For this project, we will implement a Gibbs sampler(George and Edward (1992)). There are also published R-packages that handles clustering problem. We will use *mclust*(Scrucca et al. (2016), which also use model-based clusterig of Gaussian mixture model, and built-in R function 'stats::kmeans()'R Core Team (2013), which identify clusters based on euclidean distances from centriods, to compare performances on generated dataset.

2. Methodology

Gaussian mixture model has a structure

$$x_i \overset{\text{iid}}{\sim} \sum_{k=1}^{K} \rho_k \cdot N(\mu_k, \Sigma_k)$$

where $i = 1, \ldots, n$ and ρ_k is a probability of x_i being in cluster k.
We will use hierarchical model to explain the behaviour of θ, which is referred to "random-effect" of each

observations. This θ_i has the same distribution as x_i, hence it is a random variable. We can find $\hat{\theta}_i$ and its Fisher information matrix $\hat{V}_i$ by defining a family of models $f(x_i|\theta_i)$ and finding MLE corresponding to each observation. Then we can obtain

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

. We will use y_i to denote $\hat{\theta}_i$ and $\hat{V}_i$ to V_i to avoid confusion of notations. Our interest is to estimate posterior probability to determine each observation's cluster, and this can be done via MCMC. In this project, Gibbs sampler is chosen as a method of MCMC. The parameters that are updated through Gibbs sampler are $\Psi_k = (\theta_i, \mu_k, \Sigma_k, \rho_k)$, where θ_i = parameters for observation x_i, μ_k = mean of parameters for cluster k, Σ_k = within-group variance covariance matrix for cluster k, and ρ_k = prior probability of being in cluster k. In the next section, we will describe how our Gibbs sampler works.

2.1 Gibbs sampler algorithm

1. Initialize parameters

It is possible to manually assign initial values of parameters. Theoretically, Gibbs sampler calculates numerical approximation of desired quantities no matter which values are assigned to as initial values. Unless initial values for parameters $\Theta, \mu, \Sigma, \rho, z$ are pre-specified, we will use following to initialize parameters before the iterations begin.

Set initial values:
$z^{(0)}$: kmeans(y,k)$cluster
$\theta^{(0)} : y$
$\mu_k^{(0)}$: within-group sample mean
$\Sigma_k^{(0)}$: within-group sample variance
$\rho_k^{(0)}$: proportion of cluster k in $z^{(0)}$

Here, we used built-in R function **'kmeans()'** to assign initial group memberships. This function is using K-Means clustering methods, which is another clustering algorithm based on euclidean distance of data points from the centroids. Throughout iterative steps, dataset will be separated into pre-specified number of clusters, k, by random sampling without replacement and the centroids is avaerage of all data points that belong to each cluster. Then compute each data point's distances from all centroids and assign it to the closest cluster(centroid). This step will generate clusters with smallest within-cluster variance, but biggest between-cluster variance once it converges. However, we will use *'kmeans()'* just as a starting point, so, by default, only 10 iteration will be used.

2. Update parameters

- For iteration $m \geq 1$;

Update parameters:

$$\mu_k^{(m)} \overset{\text{ind}}{\sim} N(\bar{\theta}_i^{(m-1)}, \Sigma_k^{(m-1)}/N_k) \qquad N_k = \sum_{i=1}^{N} \mathbb{I}(z^{(m-1)} = k)$$

$$\Sigma_k^{(m)} \overset{\text{ind}}{\sim} \text{InvWish}\left(\Omega_k + \sum_{i:\mathbb{I}(z_i^{(m-1)}=k)} (\theta_i^{(m-1)} - \mu_k^{(m)})(\theta_i^{(m-1)} - \mu_k^{(m)})', N_k + v_k\right)$$

$$\theta_i^{(m)} \overset{\text{ind}}{\sim} N(G_i^{(m)}(y_i - \mu_{z_i^{(m-1)}}^{(m)}) + \mu_{z_i^{(m-1)}}^{(m)}, G_i^{(m)} V_i) \qquad G_i^{(m)} = \Sigma_{z_i^{(m)}} V_i$$

$$\rho_k^{(m)} \sim \text{Dirichlet}(\alpha) \qquad \alpha = N_k + 1$$

$$z_i^{(m)} \sim \text{Multinomial}(K, \lambda_i)$$

Notice that Σ_k has Inverse Wishart distribution with unspecified parameters Ω_k and v_k. Ω_k is a within-group variance-covariance matrix for cluster k and v_k is an arbitrary real number with constraint $v_k > p-1$. In this project, we will let $\Omega_k = \text{var}(Y)$, sample variance of the data, and $v_k = p+2$ to make a priori $E[\Sigma_k] \equiv \text{var}(Y)$. Also we have used κ and λ when updating z. These parameters can be computed as follows;

$$\begin{aligned} \kappa_{ik} &= \log\rho_k - \tfrac{1}{2}\Big((\theta_i - \mu_k)'\Sigma_k^{-1}(\theta_i - \mu_k) + \log|\Sigma_k|\Big) \\ \lambda_{ik} &= \frac{\exp(\kappa_{ik})}{\sum_{j=1}^{K}\exp(\kappa_{ij})} \end{aligned}$$

Hence, λ_{ik} is a probability that observation y_i to be in cluster k.

3. Estimate Posteriors

Out interest is to estimate $\Lambda = (\lambda_1, \ldots, \lambda_n)$, and we can get the estimate of this matrix by

$$\hat{\Lambda} = \frac{1}{M}\sum_{m=1}^{M}\Lambda^{(m)}$$

One thing we should notice here is that our Λ in earlier iteration would not so meaningful as their initial parameter values might be off by huge from the true values. Hence, to avoid having biased $\hat{\Lambda}$, we will consider 10% of iterations as burn-in period and discard Λ's produced from this period.
Once we obtain $\hat{\Lambda}$, we can estimate data's group membership z in couple ways. One way is to sample from $z_i \sim \text{Multinomial}(K, \hat{\lambda}_i)$. Another way is to pick MAP(maximum a posteriori), i.e. $z_i = \underset{k}{\text{argmax}}\lambda_{ik}$.
Along with this Λ, we can also make inference on other parameters since they theoretically will approach their true values after sufficient number iterations.

3. Results

As the Gibbs Sampler has been briefly explained in the section above, we shall see how well it performs on the dataset.

3.1 Apply to real-world data

- Dataset specification

We prepared a HBE dataset(Hill et al., 2014). This dataset contains $N = 668$ trajectories of microparticles diffusing in human bronchial epithelial(HBE) mucosal fluid. We have reduced this dataset to have $p = 2$ and $k = 2$. The original dataset before reduced has $p = 6$ and $k = 6$. The clusters are originally distributed as the table below and the dataset has ordered observations in this order, i.e. observation $x_{1:63} \in$ '1p5', $x_{64:135} \in$ '2', $\cdots$. For the reduced dataset, as we don't know how to cluster them into 2 distinguishable groups, we simply assume cluster 1 = {'1p5','2','2p5','3'} and cluster 2 = {'4','5'} so that both clusters have almost equal sizes.

Table 1: Frequency table of HBE dataset

Group	Frequency
1p5	63
2	72
2p5	76
3	99
4	180
5	178

- Performance of Gibbs Sampler

To check if our Gibbs Sampler had enough iterations until convergence, we can use plots of how parameter values get updated throughout iterations.

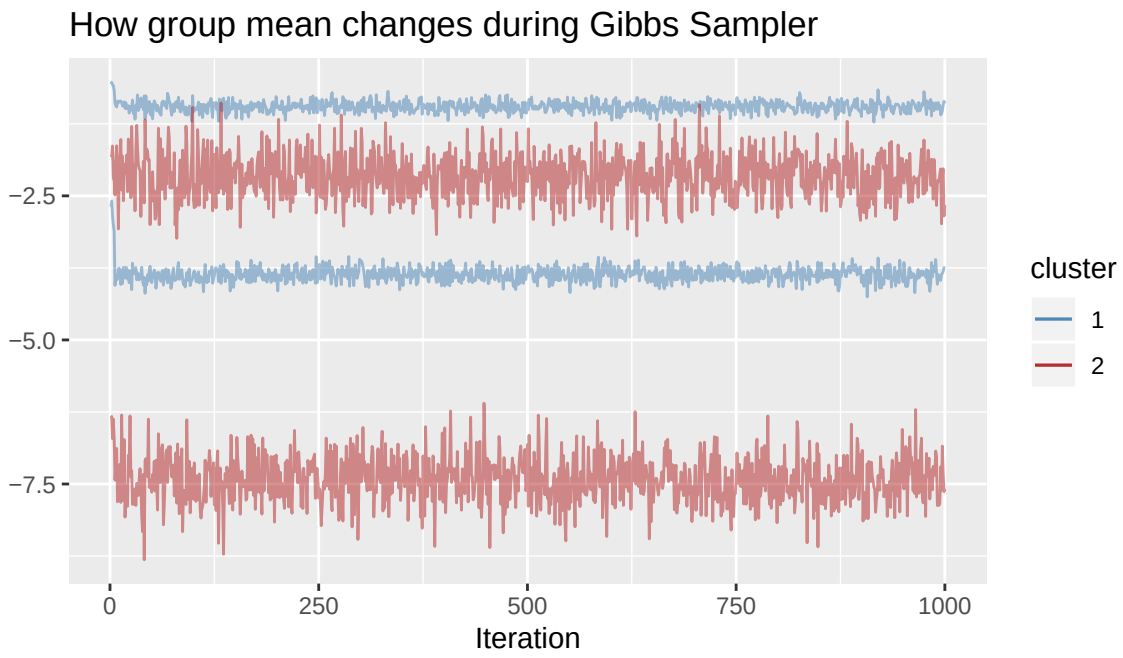


Figure 1: Behaviour of mu during Gibbs Sampler

From this plot, we can clearly see that lines are lying around horizontal lines and the clusters are well separated. However, cluster 2 has very high between group variances. Remind that $\mu_k \sim N(\hat{\theta}_i, \Sigma_k/N_k)$, and we can suspect μ_2 having too high variance compared to that of cluster 1 is because either Σ_2 being big or N_2 being small.(or possibly both). Whichever the real reason behind this issue is, both of the parameters(Σ_k and N_2) are influenced by the assigned group membership, and computing relative probability for observation x_i to be in cluster k, λ_{ik} requires reliable updates on Σ_k since their log posterior pdf contains Σ_k^{-1} and $\log(|\Sigma_k|)$ in their calculation. Hence, we should suspect our Gibbs Sampler doesn't work when updating Σ_k very well.

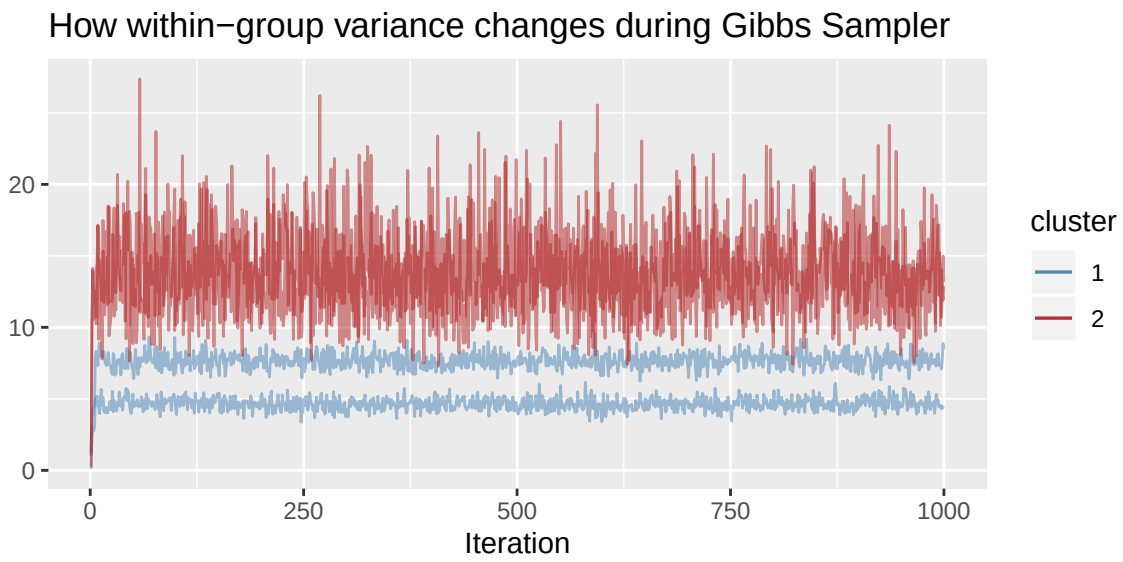


Figure 2: Behaviour of Sigma during Gibbs Sampler

As stated above, Σ_2 shows very variable behaviours. $\Sigma_{1,1,2}$ and $\Sigma_{2,2,2}$, i.e. var(p_1) and var(p_2) for observation $x_i, i \in [i|z_i = 2]$, seemed perfectly overlapped to each other.

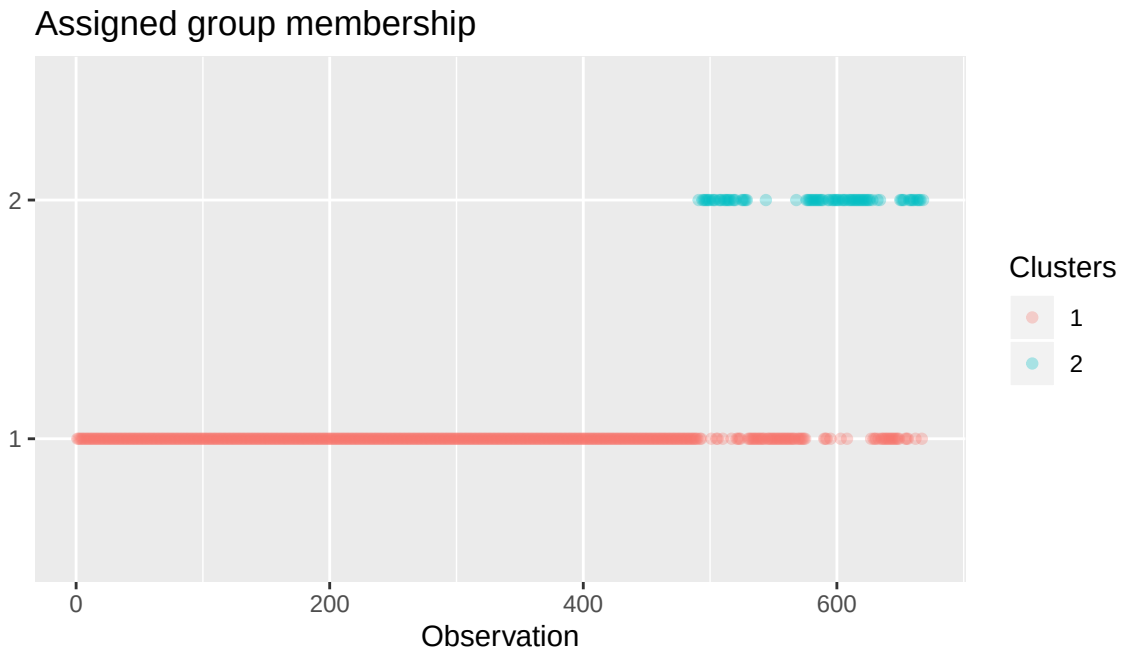


Figure 3: Gibbs Sampler estimate of clusters

Table 2: Gibbs Sampler cluster frequencies

Cluster	Frequency
1	576
2	92

Only 92 observations are distinguished as to be in cluster 2. However, good news is that all cluster 2's are appeared in observations which originally was assigned to be in group '4' or '5'. The number of estimated clusters that has been clustered wrong is $N_2^{(0)} - N_2^{(Gibbs)} = 358 - 92 = 266$. Hence the error rate is $266/668 \approx 0.398$, or approximately 40%. Since this only a binary problem, we obtained higher percentage being right than wrong. However, if it was multiclass problem, with $k > 2$, we will see much more clustering being wrong. This gives evidence that our Gibbs Sampler is yet it too weak to handle clustering problem. Fitting our Gibbs Sampler to original dataset with $p = 6$ and $k = 6$, it will highly likely to show very poor performance and yields meaningless reults. Hence, we will skip fitting the original data in this project and leave it to future research to improve its performance.

3.2 Comparing other algorithms' performances

As mentioned above, we will compare the results with other popular clustering algorithms, 'mclust()' function from *Mclust* package and R's built-in 'stats::kmeans()' function. These two are very popular and verified functions to solve clustering problems, hence, we will also fit the original dataset along with the reduced one.

- Performances on reduced dataset

First, we want to see how well these functions handles very simple problem.

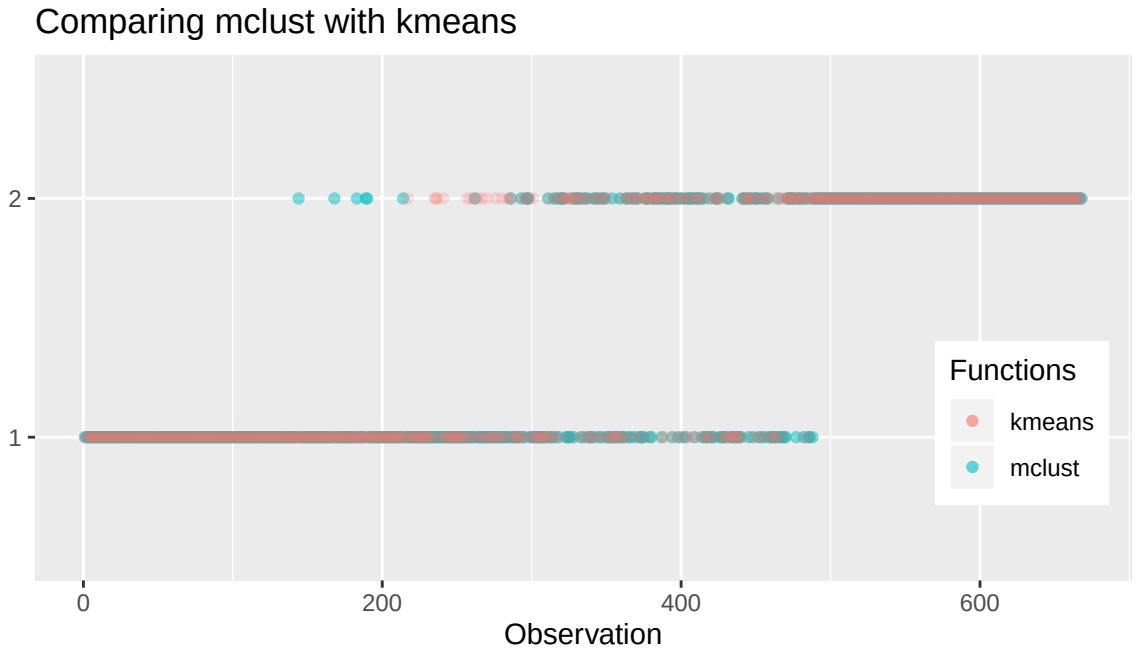


Figure 4: Comparing clustered results between 'mclust' and 'kmeans

Both functions seem to have same clustering results for majority of data, but we can still observe some mismatching result around observation=200.

Now let's fit the original dataset to these models and conclude which performs better.

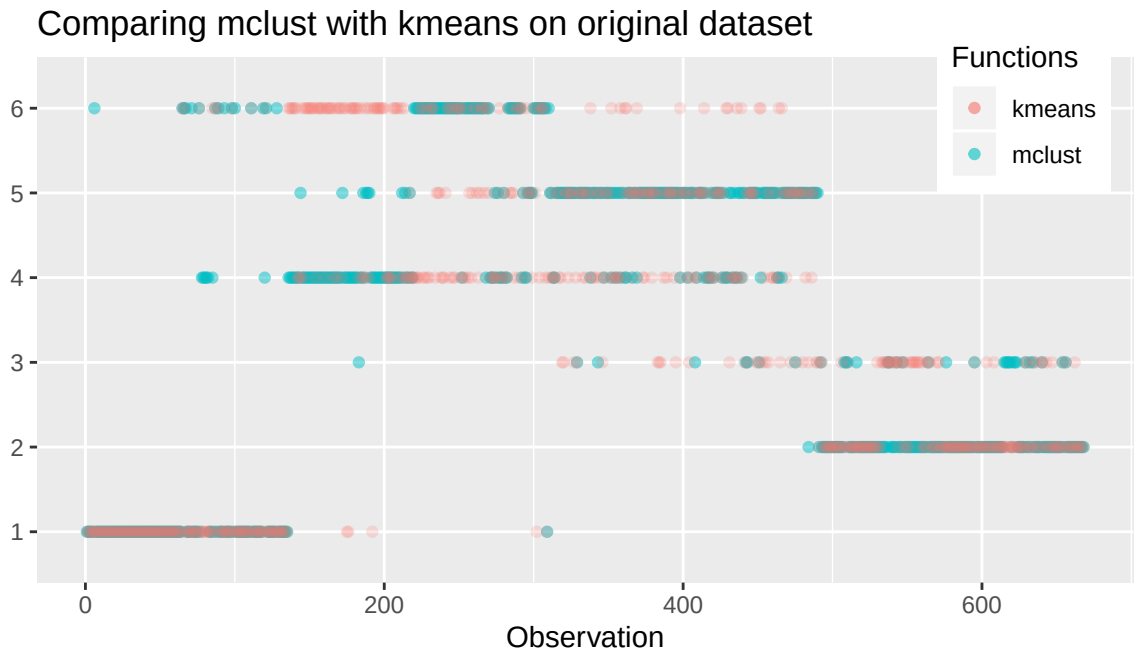


Figure 5: mclust() and kmeans() on original dataset

Except for the observations on the both ends, we can see many points not overlapping. We shall compute the error rate to compare their performances.

Table 3: Accuracy of each functions on datasets

	k=2	k=6
mclust	0.855	0.358
kmeans	0.861	0.439

In both datasets, 'kmeans()' has slightly better performance than mclust. As both algorithms are using randomness in their iterative steps, we cannot strongly argue that 'kmeans()' is a better algorithm to use when we handle clustering problems. However, at least in this dataset, we can argue that using 'kmeans()' over 'mclust' is more clever choice to do clustering.

4. Discussion

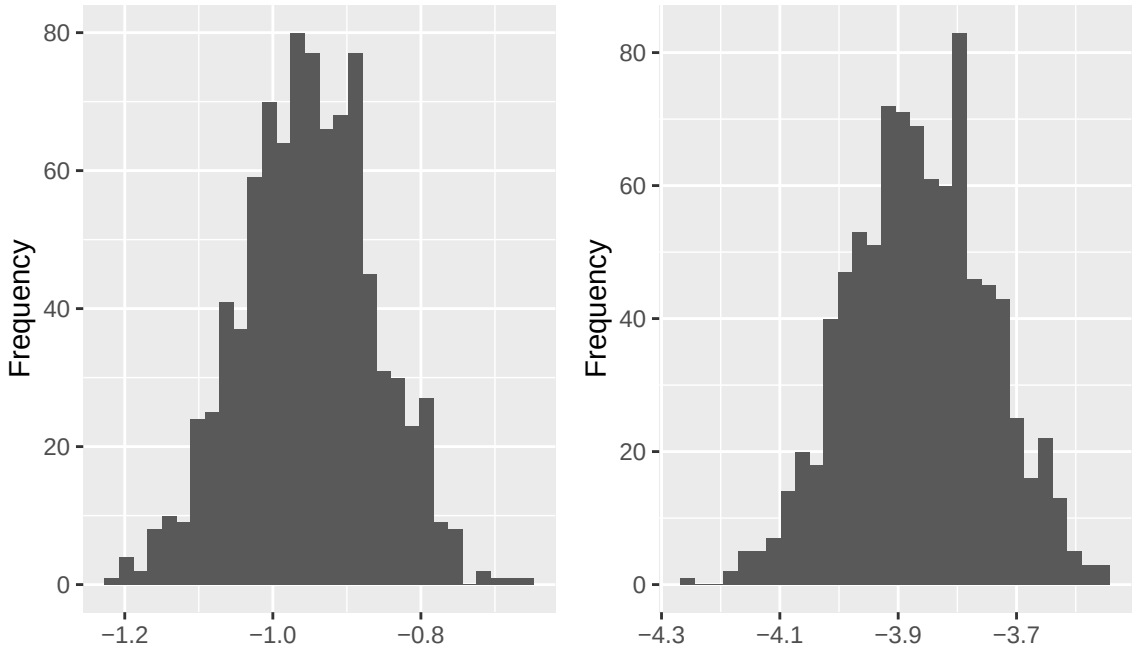
We have implemented our version of Gibbs Sampler for this project, and it seems to have very low performance compared to other popular methods to solve clustering problems. Major difference between Gibbs Sampler and mclust/kmeans functions is they are using different algorithms. In other words, Gibbs Sampler is one method of MCMC(Markov Chain Monte Carlo), which updates one parameter at a time by sampling from conditional distributions and it eventually will estimate the posterior of our interests. As it only care about updating one parameter at a time, Gibbs sampler theortically has no problem when dataset has large number of covariates. mclust/kmeans, however, are functions that are based on EM algorithm. EM algorithm is an optimzation problem which tries to maximize likelihood of target function through iterative steps. Hence, this method is in very high danger of failure to convergence.

One major defect that our version of Gibbs Sampler had was that updating $Sigma_k$ was not reliable. As we simply assigned $\Omega_k = \text{var}(y)$ and $\nu_k = p + 2$, for $k = 1, \ldots, K$, and this affected sampled Σ_k. We observed that when degrees of freedom of Inverse Wishart distribution get smaller, produced matrices have very high absolute values in their entries. Thus, we obtained very variable values for certain parameters and lead to wrong decision of choosing group memberships. We should address and fix problem arose from this updating Σ_k process more deeply in future research. Along with this, in future research, simulational studies by generating datasets with multiple different settings(e.g. sizes of n, p, k, correlation between covariates, etc.) is recommended so that we can find relationships between those conditions.

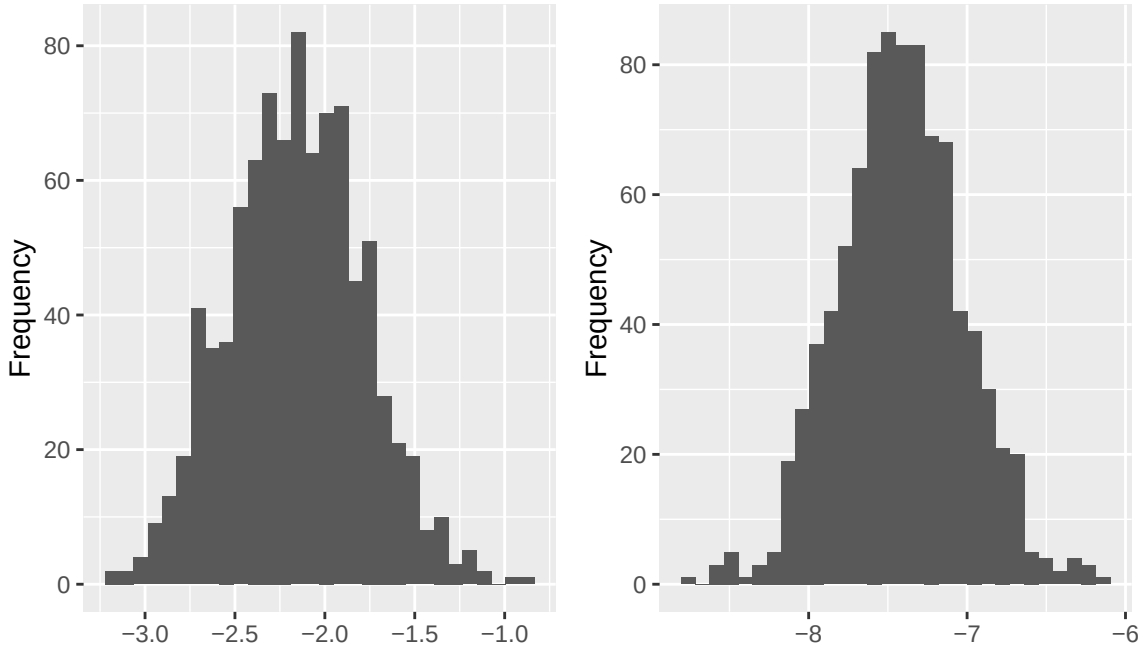
5. Appendix

- Fitting Gibbs Sampler to reduced dataset
- Visual representation

```
library(gridExtra)
#  visual representation to observe whether parameters are
#normally distributed
ranges <- (gbs$burn+1):gbs$m_iter
mu_means <- matrix(0,k,p)
mu_means[1,] <- colSums(y[clust_2==1,])/length(which(clust_2==1))
mu_means[2,] <- colSums(y[clust_2==2,])/length(which(clust_2==2))

# First cluster
mu11 <- ggplot() +
  geom_histogram(aes(x=gbs$mu[1,1,ranges])) +
  labs(x=NULL,y='Frequency')
mu12 <- ggplot() +
  geom_histogram(aes(x=gbs$mu[1,2,ranges])) +
  labs(x=NULL,y='Frequency')

# Second cluster
mu21 <- ggplot() +
  geom_histogram(aes(x=gbs$mu[2,1,ranges])) +
  labs(x=NULL,y='Frequency')
mu22 <- ggplot() +
  geom_histogram(aes(x=gbs$mu[2,2,ranges])) +
  labs(x=NULL,y='Frequency')

grid.arrange(mu11,mu12,ncol=2)
```



```
grid.arrange(mu21,mu22,ncol=2)
```




```
## they all have normal shapes!!
```

- Mclust & kmeans

```
require(mclust)
set.seed(1)
mclust <- Mclust(y, 2, verbose=F)
m_result <- mclust$classification
kmean <- kmeans(y, 2)
k_result <- kmean$cluster

m_2 <- length(which(clust_2==m_result))/nrow(y)
k_2 <- length(which(clust_2==k_result))/nrow(y)
```

- Mclust & kmeans on original dataset

```
n <- length(hbe_fit)
p <- 6
k <- 6
y_full <- matrix(0,n,p)
V_full <- array(0, dim=c(p,p,n))
for(i in 1:n) {
  y_full[i,] <- hbe_fit[[i]]$coef
  V_full[,,i] <- hbe_fit[[i]]$vcov
}

#matching clusters
clust_full <- ifelse(cluster=='1p5','1',
```

```
                    ifelse(cluster=='2','4',
                           ifelse(cluster=='2p5','6',
                                  ifelse(cluster=='3','5',
                                         ifelse(cluster=='4','3','2')))))

clust_full <- as.numeric(clust_full)
m_f_result <- as.numeric(m_f_result)

# Accuracy rate of mclust and kmeans
m_f <- length(which(clust_full==m_f_result))/nrow(y)
k_f <- length(which(clust_full==k_f_result))/nrow(y)
```

6. Reference

Hartigan, J., & Wong, M. (1979). Algorithm AS 136: A K-Means Clustering Algorithm. Journal of the Royal Statistical Society. Series C (Applied Statistics), 28(1), 100-108. doi:10.2307/2346830

McNicholas, P.D. (2011). On Model-Based Clustering, Classification, and Discriminant Analysis.

Andrieu, C., De Freitas, N., Doucet, A., & Jordan, M. I. (2003). An introduction to MCMC for machine learning. Machine learning, 50(1-2), 5-43.

George Casella & Edward I. George (1992) Explaining the Gibbs Sampler, The American Statistician, 46:3, 167-174, DOI: 10.1080/00031305.1992.10475878

Scrucca L., Fop M., Murphy T. B. and Raftery A. E. (2016) mclust 5: clustering, classification and density estimation using Gaussian finite mixture models The R Journal 8/1, pp. 205-233

R Core Team (2013). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/

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., & Forest, M. G. (2014). A biophysical basis for mucus solids concentration as a candidate biomarker for airways disease. PloS one, 9(2), e87681. https://doi.org/10.1371/journal.pone.0087681