Bayesian Inference for the Model-Based Clustering

Chenxi Zhang Ziqing Chen Jiabo Liu

Abstract

Data Clustering is a critical problem for big date due to the assumption "Similar data-points have similar behavior". Model-based classification is a method to classify multivariate observations into different clusters. The importance of studying this problem is that it is difficult to conduct real sampling for some data sets. For example, sampling about the probability of observing different weather phenomena separately in a simple normal sampling is easy. However, if we wish to combine the probability of observing multiple weather phenomena within the same day into a normal-mixture model, such sampling is difficult to realize. Therefore we proposed a new method for this project, it is similar to the approach that is commonly known as the Gaussian Mixture Models (GMM) Clustering Algorithm; it is achieved with Bayesian Inference with Gibbs Sampling process and Dirichlet Process. For a normal-mixture model, if we arise the x-dimension of a mixture model, it will break down both computationally and conceptually. From the perspective of computational break-down, as x-dimension increases, it becomes impossible to calculate such large and complicated matrix. While from the conceptual standing point, as x-dimension increases, most of x_i is not meaningful in regards to determining the clusters. Hence, it becomes a problem that we must solve. The method to estimate the posterior distribution variables is by fitting a Bayesian non-parametric model into the pre-smoothed data. Thereby reducing the computational time while disregarding some of the insignificant information. This is extremely useful when we are using big data. Utilizing Gibbs Sampling and Markov chain Monte Carlo(MCMC) is a good way to estimate the posterior distribution latent variables.

1 Introduction

As mentioned in the abstract, Data Clustering is an essential process to model big data, since it is reasonable to assume that similar data will share similar behaviors and properties(S.Kaushik)[1]. By classifying data into some sort of logical groups, data scientists can extract value from a large set of data easier.(L.Pierson)[2]. The current research shows that there are several common data clustering methods including:

- **K-Means Clustering**: K-Means Clustering is an algorithm that classifies data into K different groups based on the distance to their group means. This algorithm is efficient and simple, but it may result in some inaccuracy due to different initialization of K. Also it may break down easily if the data set is too large and complicated.(G.Seif)[3].
- **Mean-shift Clustering**: Mean-shift Clustering attempts to find the dense area of data points. Comparing to K-Means Cluster Algorithm, this method doesn't require us to choose the value of K. Consequently it is more consistent than the K-Means Cluster Algorithm, but it may also break down easily if the data set is too large and complicated.(G.Seif)
- **Hierarchical Clustering**: Hierarchical Clustering is a similar process to the divide-and-conquer sorting algorithm, which treats each data points as a single cluster at the outset and then merges pairs of clusters until all clusters have been merged into a single cluster that contains all data points.(G.Seif)
- **Gaussian Mixture Models Clustering Algorithm**: GMM Clustering Algorithm is similar to K-Means Algorithm, but it is more flexable. GMM is a classic example of mixture models. It is usually achieved by Expectation–Maximization to optimatize the process. GMM algorithm is much more powerful, however, it still collapses easily when facing a huge dimension data matrix.(G.Seif)

Expectation-Maximization algorithm computation involves matrix inversion. Unfortunately, as the dimension of the observed data increases, the algorithm will break down easily due to the large volume of data.(M. Medvedovic)[4] In order to fit the clustering algorithm into more complicated data, one approach is to reduce the dimension of the observed data points by git rid of the not-so-meaningful observed data points. As such, one way is to introduce a family of model $f(x|\theta)$ such that reducing the observed data to a small set of features $\theta \in R^p$. As a result, the approach of Data Clustering in this project is similar to the GMM Clustering Algorithm. Instead of Expectation-Maximization, this

[1]Kaushik, Saurav, and Saurav. "Clustering Introduction & Different Methods of Clustering." Analytics Vidhya, 14 Sept. 2019

[2]Pierson, Lillian. The Importance of Clustering and Classification in Data Science. n.d.

[3]Seif, George. The 5 Clustering Algorithms Data Scientists Need to Know. Feb 5,2018.

[4]Medvedovic, M, et al. Bayesian Mixture Model Based Clustering of Replicated Microarray Data. 2003, Bayesian Mixture Model Based Clustering of Replicated Microarray Data.

project will use Bayesian Inference to approximate the mixture model and use Markov Chain Monte Carlo(MCMC) Gibbs Sampler to update and converge the parameter, this will also involve a Dirichlet Process. (Mengersen.K)[5]. Five parameters are involved in the model, which include cluster indicator z, cluster probability vector ρ, matrix mean μ, covariance matrix σ and significant family function θ. In the assumed mixture model, the cluster indicator z comes from a multinomial distribution, observed data and θ come from the multivariate normal distribution.(D. B. Dahl)[6]

In this project, we will be researching the performance of the MCMC Gibbs Sampler of the Mixture Model on a set of simulated data. The data itself will come from a set of observed stock information. To simplify the process, we will sample $N = 600$ data with observation density $p = 6$ randomly, the observed Fisher Information will be sampled by R function `rnorm()` based on p. We will implant an R package that contains the MCMC Gibbs Sampler of the mixture model with the unit test of each parameter. In the following sections of the report, we will present the method about how we achieve and finish the Gibbs Sampler by introducing each parameter's conditional log distribution update; as well as some technical challenges we face during the implanting process such as the testing procedure. In addition, we will apply our MCMC Gibbs Sampler function from the package to the data and presents the results. Lastly, we will discuss our findings and the performance of the procedure.

2 Methodology

By fitting the Bayesian Inference into the Mixture model, the hierarchical clustering model can be approximated and rewrite as:

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

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

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

where $Y = (y_1 .... y_N)$ is the observed data, and $Z = (z_1 .... z_N)$ and $\theta = (\theta_1 .... \theta_N)$ are the missing data, such that the loglikelihood for the model parameters Ψ given the complete

[5]Insha Ullah & Kerrie Mengersen. Bayesian mixture models and their Big Data implementations with application to invasive species presence-only data. 22 March 2019.

[6]D. B. Dahl (2006),Model-Based Clustering for Expression Data via a Dirichlet Process Mixture Model,in Bayesian Inference for Gene Expression and Proteomics, Kim-Anh Do, Peter Müller,Marina Vannucci (Eds.),Cambridge University Press.

data z, θ, Y. The loglikelihood for the model parameters Ψ given the complete data thus becomes

$$\begin{aligned} l(\Psi|z,\Theta,Y) &= logp(Y,\Theta,z|\Psi) \\ &= \textstyle\sum_{i=1}^{N} -\frac{1}{2}(y_i-\theta_i)'V_i^{-1}(y_i-\theta_i) \\ &\quad -\tfrac{1}{2}\textstyle\sum_{i=1}^{N}\sum_{k=1}^{K} 1[z_i=k]\cdot\{(\theta_i-\mu_k)'\Sigma_k^{-1}(\theta_i-\mu_k)+log|\Sigma_k|\} \\ &\quad +\textstyle\sum_{i=1}^{N}\sum_{k=1}^{K} 1[z_i=k]\cdot log\rho_k \end{aligned}$$

Instead of implementing an Expectation-Maximization algorithm for this formula, it is faster and easier to build a Gibbs sampler within a Bayesian approach with a posterior distribution in the form of

$$p(\Psi,z,\Theta|Y)=\frac{p(Y,\Theta,z|\Psi)\cdot\pi(\Psi)}{p(Y)}\propto p(Y,\Theta,z|\Psi)\cdot\pi(\Psi)$$

such that the log-posterior distribution is given by

$$logp(\Psi,z,\Theta|Y)=\ell(\Psi|z,\Theta,Y)-\frac{1}{2}\sum_{k=1}^{K}(v_k+p+1)log|\Sigma_k|+tr(\Sigma_k^{-1}\Omega_k)$$

The prior function is the PDF of a Inverse-Wishart distribution, by comparing the formulas, it is easy to see that the prior function is $InvWish(\Omega,\nu)$ given σ where ω is a $p\times p$ symmetric positive-definite matrix and ν is a scalar. Both loglikelihood for the model parameters and the log-posterior distribution are implanted and updated in the `nnm-function.r` file in the package, the prior function is implanted by using the `diwish()` function from the `mniw` library. Let $A=(Y,\theta,z,\Psi)$ denote all the variables in the model, by discarding the parameters in the log-posterior distribution that doesnot involved, we have the conditional update for each parameter as follow:

Conditional Update for θ :

The conditional log-pdf of θ by discarding all other parameters is

$$logp(\theta_i|A\setminus\{\theta_i\})=-\frac{1}{2}(y_i-\theta_i)'V_i^{-1}(y_i-\theta_i)-\frac{1}{2}(\theta_i-\mu_{z_i})'\Sigma_{z_i}(\theta_i-\mu_{z_i})$$

By comparing formulas, we can see that this distrubution is the sum of two Multivariate normal distributions with parameters θ, V and μ, Σ respectively, it is implanted with function `dmNorm()` from the `mniw` library in R. Notice that the conditional distribution of θ_i does not depend on any of the other θ_j, which means that the θ_i are in fact conditionally independent. Thus we have the conditional draw for θ is

$$\theta_i|A \setminus \{\theta_i\} \sim N(G_i(y_i - \mu_{z_i}) + \mu_{z_i}, G_i V_i), G_i = \Sigma_{z_i}(V_i + \Sigma_{z_i})^{-1}$$

In oredr to do this sampling process effciently without involve a large number of matrix inversion, this ramdom sampling process in implanted by using the function `rRXNorm()` with variable Y, V, σ and μ from the `mniw` library in R, which will accepts all inputs simultaneously and will perform the required calculations with random effects thousands of times faster directly in `C++`.

Conditional Update for μ :

The conditional log-pdf of μ by discarding all other parameters is

$$logp(\mu_k|A \setminus \{\mu_k\}) = -\frac{1}{2} \sum_{i=1[z_i=k]} (\theta_i - \mu_k)'\Sigma_k^{-1}(\theta_i - \mu_k)$$

By comparing formulas, we can see that this distrubution is the a Multivariate normal distributions with parameters μ, Σ for θ_i where for every i such that $z_i = k$. Similarily, this distrubution is implanted with function `dmNorm()` from the `mniw` library in R. From that we get that the conditional draw for μ is

$$\mu_k|A \setminus \{\mu_1, ..., \mu_k\} \overset{ind}{\sim} N(\bar{\theta}_k, \Sigma_k/N_k), N_k = \sum_{i=1}^{N} 1[z_i = k]$$

In order to sample the entire μ all at once efficiently, the functon `rmNorm()` from the `mniw` library is used in R with corresponding variables.

Conditional Update for σ :

The conditional log-pdf of σ by discarding all other parameters is

$$\begin{aligned} logp(\Sigma_k|A \setminus \{\Sigma_k\}) &= -\tfrac{1}{2}tr\{\Sigma_k^{-1}[\Omega_k + \textstyle\sum_{i:1[z_i=k]}(\theta_i - \mu_k)(\theta_i - \mu_k)']\} \\ &\quad -\tfrac{(N_k+v_k+p+1)}{2} log|\Sigma_k| \end{aligned}$$

By comparing formulas, we can see that this distrubution is an unnormalized log-PDF of an Inverse-Wishart distribution, thus we can conclude that

$$\Sigma_k | A \setminus \{\Sigma_1, ..., \Sigma_k\} \overset{ind}{\sim} InvWish(\Omega_k + \sum_{i:1[z_i=k]} (\theta_i - \mu_k)(\theta_i - \mu_k)', N_k + v_k)$$

for θ_i where for every i such that $z_i = k$. The ν_k need to be greater than $p - 1$ in order to deifne a valid Inverse-Wishart distribution, thus we define $\Omega_k = var(Y)$ and $\nu_k = p + 2$. The functon `diwish()` from the `mniw` library is used in R for the conditional log-pdf of σ with corresponding variables, and the functon `riwish()` from the `mniw` library is used in R to sample the entire σ all at once efficiently.

Conditional Update for ρ :

The conditional log-pdf of ρ by discarding all other parameters is

$$logp(\rho | A \setminus \{\rho\}) = \sum_{k=1}^{K} N_k log\rho_k$$

This is a Dirichlet distribution such that

$$\rho | A \setminus \{\rho\} \sim Dirichlet(\alpha)$$

The conditional log-pdf of ρ is implanted by using function `ddirichlet()` from the `nnm-function.r`. The sampling process invloves the Dirichlet process where $\alpha = (N_1 + 1.....N_K + 1)$. It is implanted by using the function `rdirichlet()` from the `nnm-function.r` to sample the entire ρ all at once efficiently.

Conditional Update for z :

The conditional log-pdf of z by discarding all other parameters is

$$logPr(z_i = k | A \setminus \{z_i\}) = log\rho_k - \frac{1}{2}\{(\theta_i - \mu_k)'\Sigma_k^{-1}(\theta_i - \mu_k) + log|\Sigma_k|\} = k_{ik}$$

such that the conditional distribution for z is

$$z_i | A \setminus \{z\} \sim Multinomial(K, \lambda_i), \lambda_{ik} = \frac{exp(k_{ik})}{\sum_{m=1}^{K} exp(k_{im})}$$

The λ_{ik} is implanted directly with nested $sapply()$ function in R, where function $rcategorical()$ from the `nnm-function.r` is used to sample the entire z all at once efficiently.

MCMC Gibbs Sampler :

With all the parameters' conditional updates, we are able to build the MCMC Gibbs Sampler. The Gibbs Sampler requires initialization of each parameter, and consumes the following variables:

- nsamples: An integer represents the number of samples the Gibbs Sampler will perform
- burn: An integer represents the burn-in sampler of the Gibbs Sampler
- K: An integer represents the number of mixture components.
- Y: The observed data points
- V: The Fisher Information
- theta_init: The initialization of θ
- mu_init: The initialization of μ
- Sigma_init: The initialization of σ
- Z_init: The initialization of z
- rho_init: The initialization of ρ
- return_Z: An True/False parameter indicates whether to store z during every sample process or not since this typically requires an enormous amount of memory, default is FALSE
- return_theta: An True/False parameter indicates whether to store θ during every sample process or not since this typically requires an enormous amount of memory, default is FALSE

The Gibbs Sampler will perform sample process burn + nsamples times. During each time of the sampling process, Gibbs Sampler will use the conditional update of each parameter to generate a new value of each parameter based on the value of each parameter and from the previous sampling process. After the entire process, the Gibbs Sampler will return a list that contains all the latest value of each parameter (θ and z depends on the value of return_Z and return_theta)

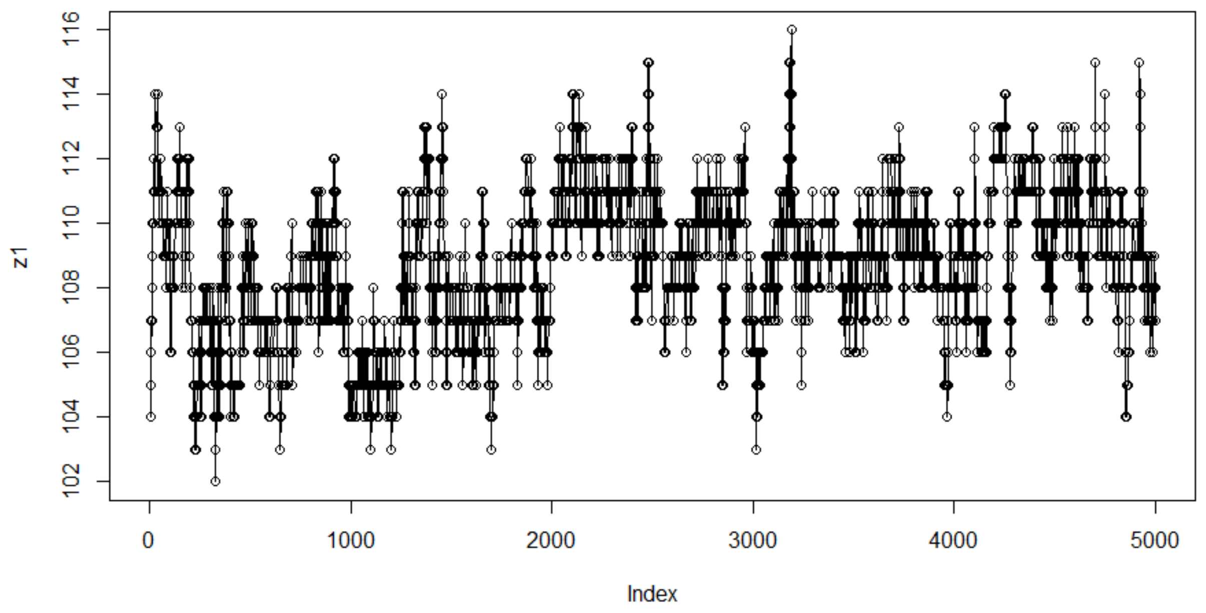
The unit test of the package for the parameters of each conditional distributions is implanted by comparing the normalized conditional log-PDF of each parameter to the unnormalized complete data log-posterior. Theoretically, they should be off by the same constant for any value of the argument with everything we condition on remaining fixed. Thus, we simulate out data and each parameter and plug into each parameter's condition log-PDF which we already implanted in each parameter's conditional update. By comparing the normalized conditional log-PDF of each parameter to the unnormalized complete data log-posterior, the results are indeed off by the same constant for any value of the argument with everything we condition on remaining fixed.

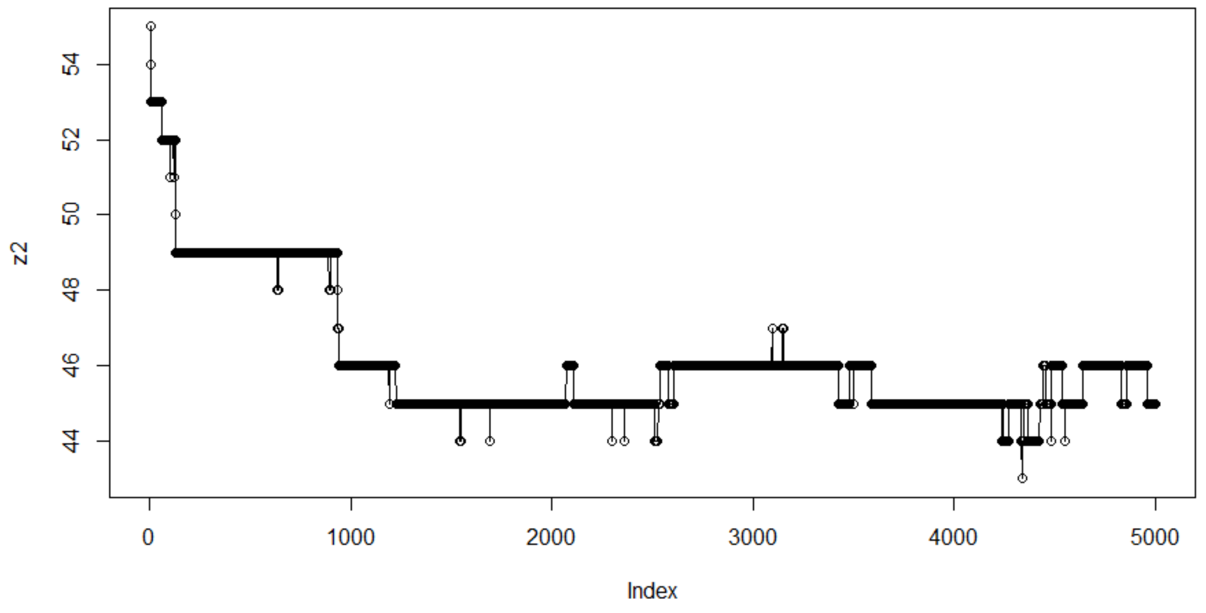
3 Results

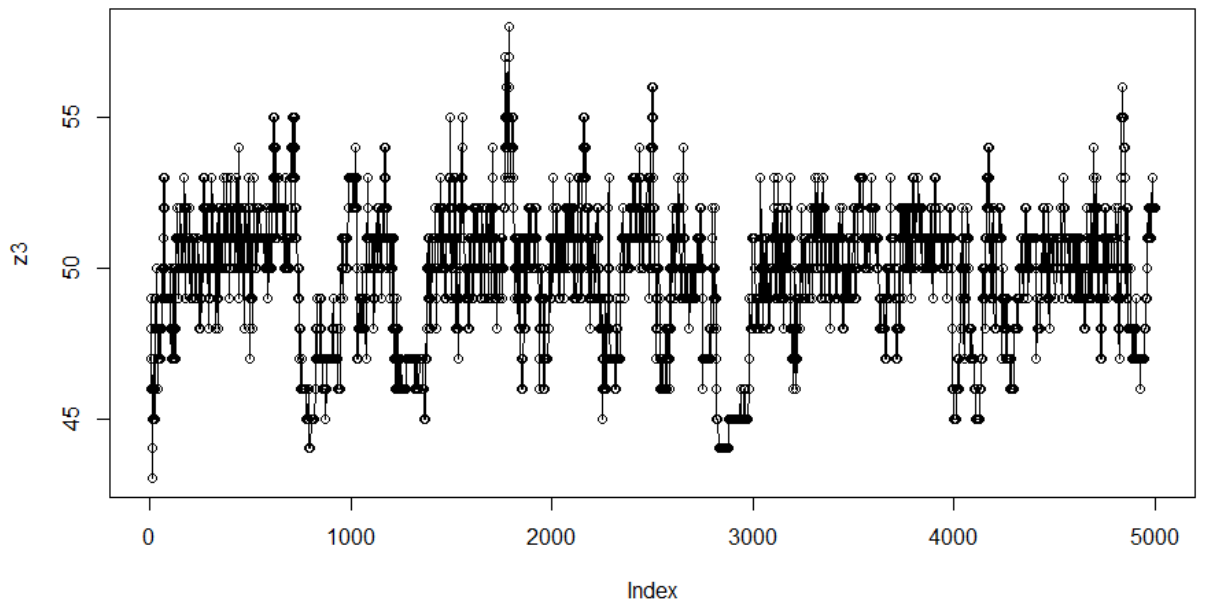
We will apply the MCMC Gibbs Sampler into the data. As mentioned in the Section Introduction 1, we will read the data from a set of observered stock information that can be found on course website. After that, we will sample $N = 600$ data with observation density $p = 6$ randomly, the observed Fished Information will be sampled by R function `rnorm()` based on p. Then we will estimate the cluster allocation vector z, and we will initialize each parameter as follow:

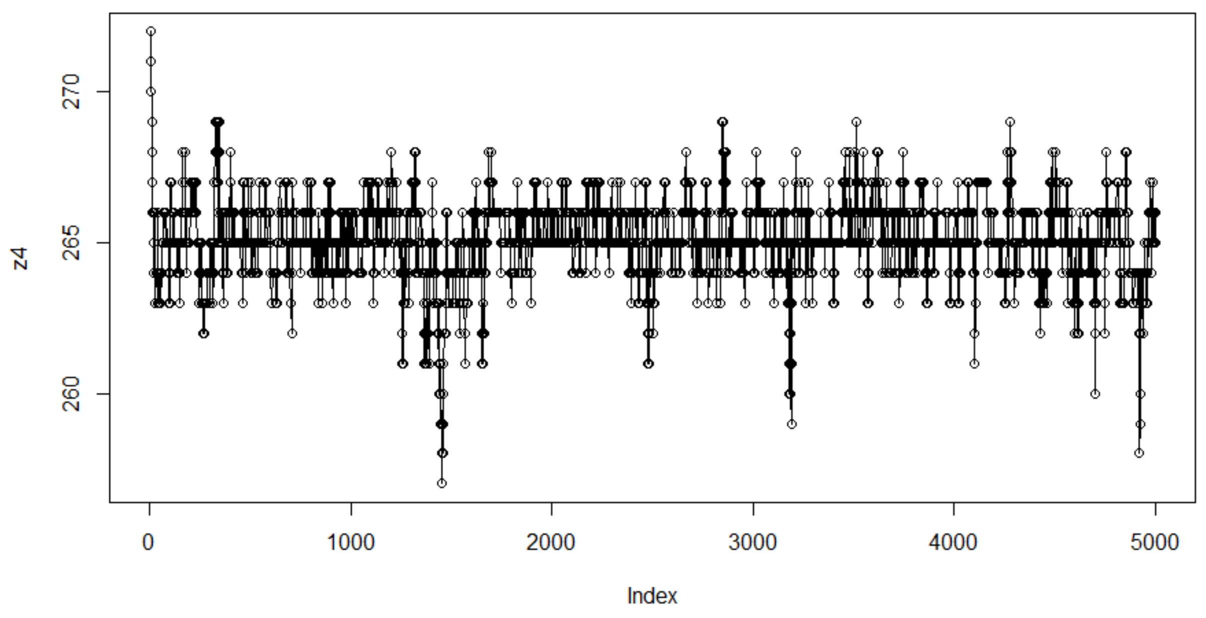
- ρ: let N_k denote the number of elements in $G_k\{i : z_i = k\}$, then $\rho_k = N_k/N$
- $\theta = Y$
- μ: $\mu_k = mean(y_i)$ such that $i \in G_k$; if $N_k = 0$, $\mu_k = mean(y_i)$
- Σ: $\Sigma_k = variance(y_i)$ such that $i \in G_k$; if $N_k < p$, $\Sigma_k = variance(y_i)$
- z: use funtion `init_z()` from the `nnm-function.r` to initialize by using **kmeans++** algorithm

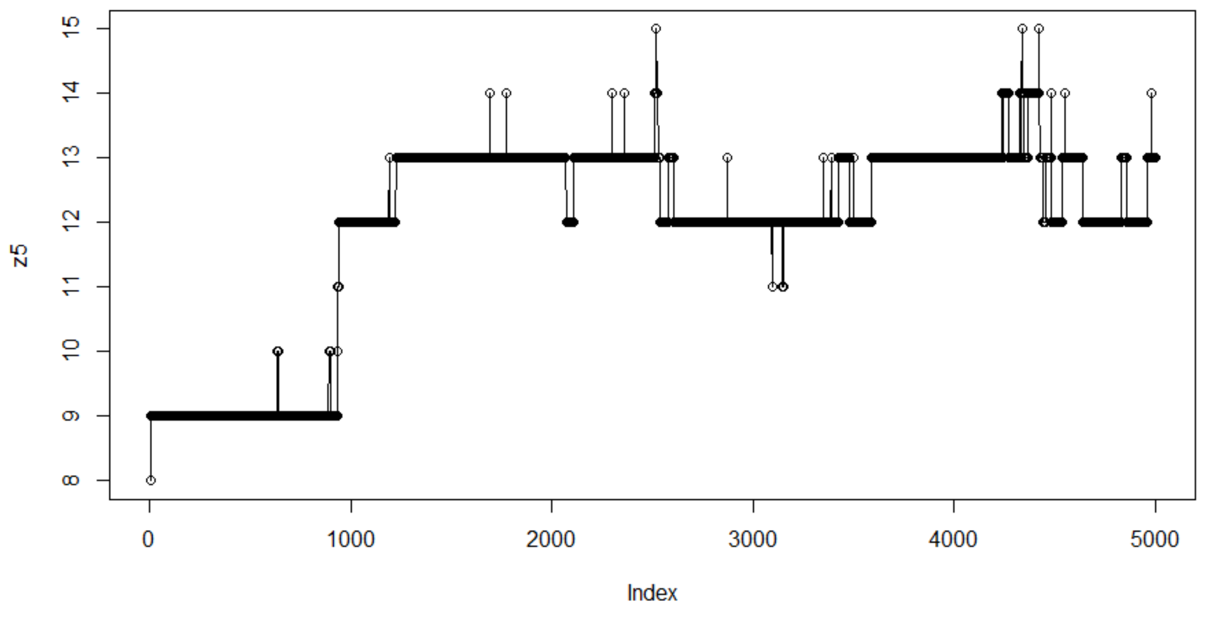
We will analyze the distrubution of the appearance of of each cluster in the final Z, the R code fo analyzing will be presented in the Section Appendix Group 6.1. The result figures are presented below.

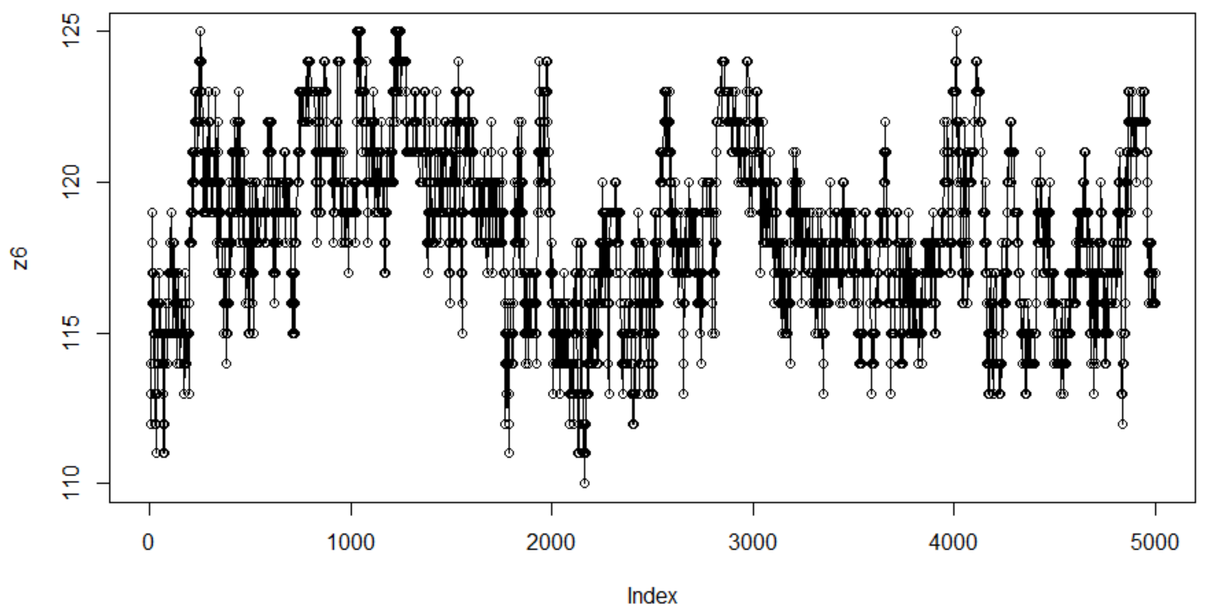



z2
54
52
50
48
46
44
0
1000
2000
3000
4000
5000
Index


z3
55
50
45
0
1000
2000
3000
4000
5000
Index

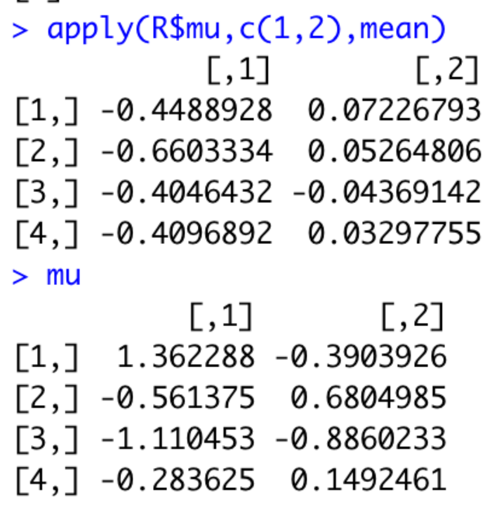

z4
260
265
270
0
1000
2000
3000
4000
5000
Index


z5
8
9
10
11
12
13
14
15
0
1000
2000
3000
4000
5000
Index



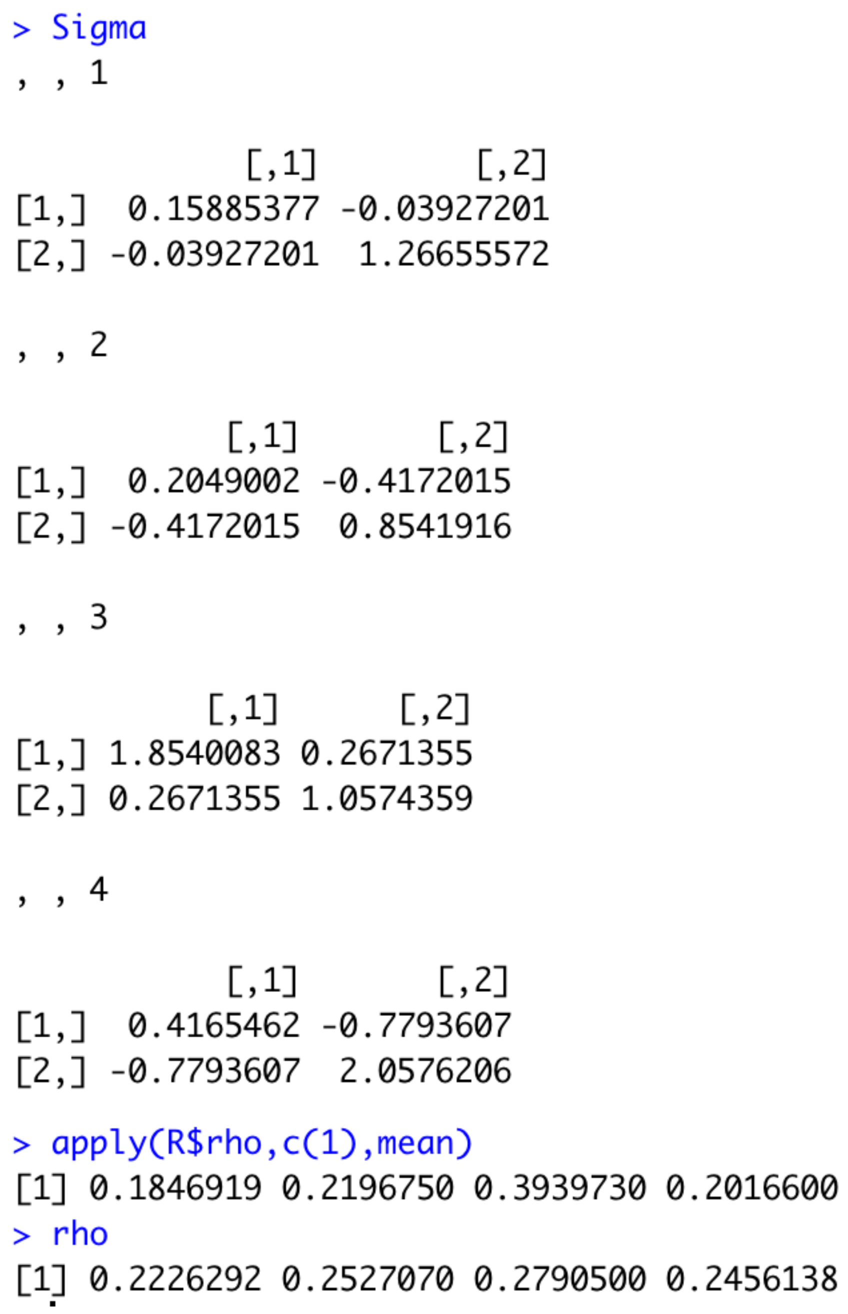
Based on the plots, we can see that the points are located relatively random in each plot. Combine them together, we can see that the frequency of cluster 4 appears is the highest, and the frequency of cluster 5 appears is the lowest, with an order $f(4) > f(1) > f(6) > f(3) > f(2) > f(5)$.

In addition, we also apply the MCMC Gibbs Sampler a simulated data followed in the model as mentioned in the Section Methodology 2 above, the R code fo analyzing will be presented in the Section Appendix Group 6.2. By applying our model the methodology into the simulated data, we have the following result:



```
> apply(R$mu,c(1,2),mean)
           [,1]        [,2]
[1,] -0.4488928  0.07226793
[2,] -0.6603334  0.05264806
[3,] -0.4046432 -0.04369142
[4,] -0.4096892  0.03297755
> mu
          [,1]       [,2]
[1,]  1.362288 -0.3903926
[2,] -0.561375  0.6804985
[3,] -1.110453 -0.8860233
[4,] -0.283625  0.1492461
```

```
> apply(R$Sigma,c(1,2,3),mean)
, , 1

          [,1]      [,2]
[1,]  1.829729 -0.515867
[2,] -0.515867  1.680081

, , 2

           [,1]       [,2]
[1,]  1.6731211 -0.4404297
[2,] -0.4404297  1.6339513

, , 3

           [,1]       [,2]
[1,]  1.3517589 -0.4571621
[2,] -0.4571621  1.2487130

, , 4

           [,1]       [,2]
[1,]  1.6564585 -0.4424783
[2,] -0.4424783  1.6945907
```



```
> Sigma
, , 1

            [,1]        [,2]
[1,]  0.15885377 -0.03927201
[2,] -0.03927201  1.26655572

, , 2

           [,1]       [,2]
[1,]  0.2049002 -0.4172015
[2,] -0.4172015  0.8541916

, , 3

          [,1]      [,2]
[1,] 1.8540083 0.2671355
[2,] 0.2671355 1.0574359

, , 4

           [,1]       [,2]
[1,]  0.4165462 -0.7793607
[2,] -0.7793607  2.0576206

> apply(R$rho,c(1),mean)
[1] 0.1846919 0.2196750 0.3939730 0.2016600
> rho
[1] 0.2226292 0.2527070 0.2790500 0.2456138
```

According to the results of Gibbs Sampling, we obtained the results of parameter estimation and found that several groups of μ can be separated, proving that the classification effect is acceptable. However, the estimated effect of μ is not good. It is speculated that the setting of Σ may cause a large variance and a large noise. Then we found that the ρ's estimate is still close to the true value; thus, the estimation effect is overall great.

4 Discussion

In this project, we proposed a method of Data clustering, by using Bayesian Inference to Approximate the Normal-Mixture Model and applying MCMC Gibbs Sampling Procedure for each parameter. After applying the Sampler to the simulated data set, we noticed that the entire procedure takes more than ten minutes to complete, which is not as efficient as we expected it to be, but fortunately, the program did not break down due to large volume and complexity of the data matrix. Compare to other research projects, most of the GMM Sampling process is done by the EM algorithm and shows a good performance result. The reasons that cause this result may have the following:

- Even though the program did not break down during the process, the data set and each parameter matrixes are still too large and complicated that may require a longer time to process, and larger memory to store, even in the lower end of the system.
- Our implantation of each parameter's conditional update and the MCMC Gibbs Sampler still not efficient enough. In the implantation of each function, we constantly use the function `sapply()` and the technique of `for loop`, which may decrease the efficiency of the sampling process, especially when dealing with parameter Σ.
- This mixture model still has its drawbacks, especially based on our simulated parameters. The way we simulate each variable, such as V, may cause a random effect on the process, which may complicate the sampling process.

To sum up, we believe there are still flaws and things we haven't consider in this project, such as the choice of the data set, the initialization of each parameter, etc. On further research, it is a good choice of topic to see how different initialization can affect the cluster results.

5 References

1. Kaushik, Saurav, and Saurav. "Clustering Introduction & Different Methods of Clustering." Analytics Vidhya, 14 Sept. 2019
2. Pierson, Lillian. The Importance of Clustering and Classification in Data Science. n.d.
3. Seif, George. The 5 Clustering Algorithms Data Scientists Need to Know. Feb 5,2018.
4. Medvedovic, M, et al. Bayesian Mixture Model Based Clustering of Replicated

Microarray Data. 2003, Bayesian Mixture Model Based Clustering of Replicated Microarray Data.

5. Insha Ullah & Kerrie Mengersen. Bayesian mixture models and their Big Data implementations with application to invasive species presence-only data. 22 March 2019.
6. D. B. Dahl (2006),Model-Based Clustering for Expression Data via a Dirichlet Process Mixture Model,in Bayesian Inference for Gene Expression and Proteomics, Kim-Anh Do, Peter Müller,Marina Vannucci (Eds.),Cambridge University Press.
7. Blei, David M. "Bayesian Mixture Models and the Gibbs Sampler." November 1, 2016
8. G´omez-Rubio, Virgilio. Mixture model fitting using conditional models and modal Gibbs sampling. December 29, 2017
9. Rasmussen, Carl Edward. "The Infinite Gaussian Mixture Model." n.d.
10. VanderPlas, Jake. Python Data Science Handbook. O'Reilly Media, November 2016.

6 Appendix

6.1 Analyzing Code (1)

```
# Read Stock Data
rdata <- read.csv("snp500-adj_close_2004-2018.csv")
rdata[, 1] <- NA

# Simulate variables
N <- 600
K <- 6
p <- 6

# Random sample indexes
rindex <- sort(sample(1:nrow(rdata), N))
cindex <- sort(sample(1:ncol(rdata), p))
data <- rdata[rindex, cindex]
```

```
# Initialize parameters
Y <- data
V<- replicate(N, crossprod(matrix(rnorm(p ^ 2), p, p)))
# Initial guess of z
rho <- runif(K)
rho <- rho / sum(rho) # 1 * 4
Z <- replicate(n = N,expr = rcategorical(matrix(rho)))
# Rho
Nk <- c()
G <- list()
for(kk in 1:K) {
  idk <- 0
  Gk <- c()
  for(i in 1:N) {
    if(Z[i] == kk) {
      Gk <- c(Gk, i)
      idk <- idk + 1
    }
  }
  Nk <- c(Nk, idk)
  G[[kk]] <- Gk
}
rho <- c()
# Theta
for(kk in 1:K) {
  rho_k <- Nk[kk] / N
  rho <- c(rho, rho_k)
}
theta <- Y
# mu
mu <- array(NA, dim = c(K, p))
for(kk in 1:K) {
  idx <- G[[kk]]
  mu_k <- c()
  if(Nk[kk] == 0) {
    for(pp in 1:p){
```

```r
      mu_kp <- mean(Y[, pp])
      mu_k <- c(mu_k, mu_kp)
    }
  } else {
    for(pp in 1:p){
      mu_kp <- mean(Y[idx, pp])
      mu_k <- c(mu_k, mu_kp)
    }
  }
  mu[kk, ] <- mu_k
}
# Sigma
Sigma <- array(NA, dim = c(p, p, K))
for(kk in 1:K) {
  idx <- G[[kk]]
  s_k <- 0
  if(Nk[kk] < p) {
    s_k <- s_k + var(Y)
  } else {
    s_k <- s_k + var(Y[idx, ])
  }
  Sigma[, , kk] <- s_k
}
# Initial Z
Z <- init_z(Y, K)

# Gibbs Sampling
nsamples <- 5000
burn <- 0
r <- gibbs_sampler(nsamples, burn, K, Y, V,
                   theta, mu, Sigma, Z, rho, return_Z = TRUE)

# Record the frequency of the appearance of Z
z_out <- (r$z)
z1 <- c()
z2 <- c()
```

```r
z3 <- c()
z4 <- c()
z5 <- c()
z6 <- c()
for(i in 1:nsamples) {
  a <- z_out[,i]
  f1 <- sum(a == 1)
  f2 <- sum(a == 2)
  f3 <- sum(a == 3)
  f4 <- sum(a == 4)
  f5 <- sum(a == 5)
  f6 <- sum(a == 6)
  z1 <- c(z1, f1)
  z2 <- c(z2, f2)
  z3 <- c(z3, f3)
  z4 <- c(z4, f4)
  z5 <- c(z5, f5)
  z6 <- c(z6, f6)
}

# Plot Z
plot(z1, type = "o")
plot(z2, type = "o")
plot(z3, type = "o")
plot(z4, type = "o")
plot(z5, type = "o")
plot(z6, type = "o")
```

6.2 Analyzing Code (2)

```r
# Simulate Data
p <- 2; N <- 100 ;K <- 4

# Simulate Parameters
rho <- rdirichlet(1,rep(N/K,K))
```

```r
z <- replicate(n = N,expr = rcategorical(matrix(rho)))
mu <- matrix(rnorm(K*p),K)
Sigma <- replicate(K, crossprod(matrix(rnorm(p^2), p, p)))
theta <- rmNorm(n = N, mu = mu[z,], Sigma = Sigma[,,z])
V <- replicate(N, crossprod(matrix(rnorm(p^2), p, p)))
y <- rmNorm(n = N, mu = theta, Sigma = V)

# Gibbs Sampling
nsamples = 1e4
R <- gibbs_sampler(nsamples, burn=100, K, Y=y, V,
                   theta_init=theta, mu_init=mu,
                   Sigma_init=Sigma, Z_init=z, rho_init=rho,
                   return_Z = FALSE, return_theta = FALSE)

# Show Results
apply(R$mu,c(1,2),mean)
mu

apply(R$Sigma,c(1,2,3),mean)
Sigma

apply(R$rho,c(1),mean)
rho
```