Final Report: HDP Model

Stats Group

2023-05-27

1 Literature Review

1.1 Slice Sampler for Mixture Models (Walker, 2007; Kalli et al., 2011)

1.1.1 Slice Sampler for Mixture Models (Walker, 2007)

In the article Slice Sampler for Mixture Models (Walker, 2007), Walker introduce a new method for sampling the mixture of Dirichlet process (MDP) model, a complex task due to the countable infinitness of the discrete masses from the random distribution functions chosen from the Dirichlet process prior.

Walker first introduces the Dirichlet process model and elaborates on a sampling method known as the stick-breaking procedure. It explains that a random probability measure from DPP can be generated by using Beta-distributed variables and a prior probability measure. The paper proposes an extension of the stick-breaking procedure for sampling a more general model. It also introduces the Mapped Dirichlet Process (MDP) model, a method of constructing continuous random distribution functions, emphasizing that a random distribution function chosen from a DPP is almost surely discrete. The passage further includes the concept of a latent variable to convert the problem from an infinite sum to a finite one, allowing an efficient sampling method. This aids in the construction of a finite mixture model, enabling a simpler computational approach.

The Mixed Dirichlet Process (MDP) model is founded on the concept of creating continuous random distribution functions, as initially proposed by Lo in 1984. The random distribution function selected from a Dirichlet process is almost certain to be discrete, as posited by Blackwell in 1973. Therefore, we turn our attention to the random density function.

$$f_p(y) = \int N(y|\theta) dP(\theta)$$

$N(y|\theta)$ denotes a conditional density function, which will typically be a normal distribution with parameters θ. Therefore, in the normal case $\theta = (\mu, \sigma^2)$.

Given the form for P, we can write

$$f_{w,\theta}(y) = \sum_j w_j N(y|\theta_j)$$

Via Gibbs sampling ideas, our attempt to estimate the model is to introduce a latent variable μ such that the joint density of (y,u) given (ω, θ) is given by

$$f_{\omega,\theta}(y,u) = \sum_j \mathbf{1}(u < \omega_j) N(y|\theta_j)$$

The joint distribution exists and we can write

$$f_{\omega,\theta}(y,u) = \sum_{j=1}^{\infty} \omega_j U(u|0,\omega_j) N(y|\theta_j)$$

We can get the marginal density for u is given by

$$f_\omega(u) = \sum_{j=1}^{\infty} \omega_j U(u|0, \omega_j) = \sum_{j=1}^{\infty} \mathbf{1}(u < \omega_j)$$

Let $A_\omega(u) = \{j : \omega_j > u\}$, then we can get

$$f_{\omega,\theta}(y, u) = \sum_{j \in A_\omega(u)} N(y|\theta_j)$$

It is clear that $A_\omega(u)$ is a finite set for all u>0. The conditional density of y given u is

$$f_{\omega,\theta}(y|u) = \frac{1}{f_\omega(u)} \sum_{j \in A_\omega(u)} N(y|\theta_j)$$

Here, we move from an infinite sum to a finite sum, which makes a lot of difference when sampling involved.

Thus, given u, we have a finite mixture model with equal to $1/f_\omega(u)$.

We can introduce a further indicator latent variable which will identify the component of the mixture from which y is to be taken.

Joint density:

$$f_{\omega,\theta}(y, \delta = k, u) = N(y|\theta_k)\mathbf{1}(k \in A(u))$$

The complete data likelihood based on a sample of size n:

$$l_{\omega,\theta}(y_i, u_i, \delta_i = k_{i\,i=1}^{\,n}) = \prod_{i=1}^{n} N(y_i|\theta_{k_i})\mathbf{1}(u_i < \omega_{k_i})$$

The passage outlines an efficient method for sampling the Mapped Dirichlet Process model using latent variables to truncate the weights of Dirichlet distributions. This approach is straightforward, fast, and circumvents challenges presented by traditional sampling techniques. The method effectively handles infinite representations by converting them into finite ones for the purposes of sampling, and provides a viable solution for non-conjugate cases.

1.1.2 Slice Sampling Mixture Models (Kalli et al., 2011)

In this article, Kalli et.al (2011) proposed a modified slice sampler called slice efficient sampler for inference problems of Dirichlet process mixture models. While preserving the property of reducing infinite number of components to finite number, slice efficient sampler is able to control the updating process of auxiliary variables so that the MCMC sampling process is smooth.

The authors extended the idea of slice sampler to normalized weights priors for mixture models and performed two numerical experiments to demonstrate the effeteness of slice sampler.

Slice efficient sampler is constructed by introducing ζ_i, $i = 1, 2, 3, 4, ...$, a positive decreasing deterministic sequence and d, a latent variable indicating which finite number of components can produce the observed data y. d together with u, where u here is the auxiliary variable, reduce the infinite number of mixture of components to finite components so that MCMC algorithms are smooth. The construction is followed:

$$f_{v,\mu,\sigma^2}(y, u, d) = \zeta_d^{-1}\mathbf{1}(u < \zeta_d) w_d N(y; u_d, \sigma_d^2)$$

The joint posterior likelihood is then:

$$\prod_{i=1}^{n} \zeta_{d_i}^{-1}\mathbf{1}(u < \zeta_{d_i}) w_{d_i} N(y; u_{d_i}, \sigma_{d_i}^2)$$

Notice, here ζ_i is decreasing and is positive, so $\mathbf{1}(u < \zeta_i)$ happens with less chance, while $\frac{w_i}{\zeta_i}$ gets bigger so gives more weights to the Gaussian kernel. And the infinite numbers of components are restricted by u as only some of the components can fulfill this indicator function. Thus, the sampling algorithm is followed:

$$1.\pi\left(\mu_j, \sigma_j^2 \mid \cdots\right) \propto p_0\left(\mu_j, \sigma_j^2\right) \prod_{d_i=j} \mathrm{N}\left(y_i; \mu_j, \sigma_j^2\right)$$

$$2.\pi\left(v_j\right) \propto \mathrm{Be}\left(v_j; a_j, b_j\right), a_j = 1 + \sum_{i=1}^{n} \mathbf{1}\left(d_i = j\right), b_j = M + \sum_{i=1}^{n} \mathbf{1}\left(d_i > j\right)$$

$$3.\pi\left(u_i \mid \cdots\right) \propto \mathbf{1}\left(0 < u_i < \xi_{d_i}\right)$$

$$4.\mathrm{P}\left(d_i = k \mid \cdots\right) \propto \mathbf{1}\left(k : \xi_k > u_i\right) w_k / \xi_k \ \mathrm{N}\left(y_i; \mu_k, \sigma_k^2\right)$$

In addition to Dirichlet prior, the authors also introduced Normalized Weights Prior. The basic construction of Normalized Weights Prior is followed:

$f(y) = \sum_{j=1}^{\infty} w_j K\left(y; \phi_j\right)$

$w_j = \lambda_j / \Lambda$

$\Lambda = \sum_{j=1}^{\infty} \lambda_j$ and $\phi_1, \phi_2, \phi_3, \ldots$ are iid

$\Lambda_m = \sum_{j=m+1}^{\infty} \lambda_j$ and $\lambda_j \sim \pi_j\left(\lambda_j\right)$

π_j as the authors suggested can be gamma distribution and inverse Gaussian distribution

Moreover, Slice Sampler and Normalized Weights Prior can be combined. By assuming $\pi_m^\star\left(\Lambda_m\right)$ for all m, then the joint density:

$$f(y, v, u, d) = \exp(-v\Lambda) \mathbf{1}\left(u < \xi_d\right) \lambda_d / \xi_d K\left(y; \phi_d\right)$$

let $v_1, v_2, \ldots, v_n$ be $v = \sum_{i=1}^{n} v_i$, the likelihood is then

$$v^{n-1} \exp(-v\Lambda) \prod_{i=1}^{n} \mathbf{1}\left(u_i < \xi_{d_i}\right) \lambda_{d_i} / \xi_{d_i} K\left(y_i; \phi_{d_i}\right)$$

Then, the authors provided two algorithms to perform sampling: 1) dependent slice-efficient sampler and $\xi_j = \lambda_j$ 2) independent slice-efficient sampler and ξ_j, λ_j independent

At last, Kalli et.al pointed out that random hazard functions can also be represented as a infinite mixture and with three sets of numerical experiments, the author demonstrate the effectiveness of slice efficient sampler on infinite mixture models:

1) on density estimation, the authors compared independent and dependent slice sampler with retrospective sampler. The results conclude that retrospective sampler performs marginally better but e slice-efficient samplers are much easier to implement. The comparisons remains the same under different approaches of priors. However, in all cases, the best is independent slice-efficient samplers.
2) On inference problems for infinite Dirichlet and infinite normalized inverse-Gaussian priors on galaxy data, the authors concludes that infinite normalized inverse-Gaussian distribution can be a more easier method to construct.
3) Lastly is a inference experiment for hazard functions, and the authors found out that a 95% pointwise credible interval and a Kaplan Meier estimate for a posterior median integrated hazard function suggests that those two methods using together with this Bayesian model can reproduce a smoothed version of the Kaplan-Meier estimate

1.2 HDP Model (Teh et al., 2004)

1.2.1 Introductions

The paper addresses problems involving groups of data, where each observation within a group is generated from a mixture model, and the goal is to share mixture components between groups. The number of mixture components is unknown and needs to be inferred from the data. To handle this scenario, the authors propose using sets of Dirichlet processes, where each group has its own Dirichlet process. The Dirichlet process, known for its clustering property, provides a nonparametric prior for the number of mixture components within each group.

To tie the mixture models across different groups, the authors introduce a hierarchical model(HDP). In this model, the base measure for the child Dirichlet processes is distributed according to a Dirichlet process itself. Since the base measure is discrete, the child Dirichlet processes share atoms. Consequently, the mixture models in different groups inherently share mixture components, achieving the desired goal of component sharing.

The paper discusses the representation of hierarchical Dirichlet processes using a stick-breaking process and introduces a generalization of the Chinese restaurant process called the "Chinese restaurant franchise." These representations facilitate the understanding and implementation of hierarchical Dirichlet processes.

The authors present Markov chain Monte Carlo algorithms for performing posterior inference in hierarchical Dirichlet process mixtures. These algorithms enable the estimation of the model parameters from the data.

The paper also describes applications of the proposed hierarchical Dirichlet process mixtures to problems in information retrieval and text modeling, showcasing the practical relevance and usefulness of the approach.

In summary, the paper introduces a hierarchical Dirichlet process model that allows for sharing mixture components between groups of data. It presents representations, inference algorithms, and applications of the model to various domains, demonstrating its potential in addressing problems involving group data and mixture modeling.

1.2.2 Setting

- j index the groups and i index the observations within each group. $\mathbf{x}_j = (x_{j1}, x_{j2}, ...)$ denote all observations in group j, then $\mathbf{x}_1, \mathbf{x}_2, ...$ are exchangeable.
- θ_{ji} denote a parameter specifying the mixture component associated with the observation x_{ji}
- Let $F(\theta_{ji})$ denote the distribution of x_{ji} given the factor θ_{ji}. Let G_j denote a prior distribution for the factors $\theta_{ji} = (\theta_{j1}, \theta_{j2}, ...)$ associated with group j.
- We have model

$$\begin{aligned} \theta_{ji}|G_j &\sim G_j, \text{ for each } j \text{ and } i \\ x_{ji}|\theta_{ji} &\sim F(\theta_{ji}), \text{ for each } j \text{ and } i \end{aligned}$$

1.2.3 Dirichlet processes

Let (Θ, B) be a measurable space, with G_0 a probability measure on the space. Let α_0 be a positive real number. For any finite partition $\{A_1, ..., A_k\}$ of the sample space, consider the random vector $(G(A_1), ..., G(A_k))$ where $G()$ is a random distribution function. If

$$(G(A_1), ..., G(A_k)) \sim Dirichlet(\alpha_0, G_0(A_1), ..., G_0(A_k))$$

where G_0 is also a fixed distribution function, then $G \sim DP(\alpha_0, G_0)$ is a Dirichlet process.

We ignore the stick-breaking representation and Chinese restaurant process in Dirichlet processes, since they are discussed in the class.

Dirichlet process mixture models. It is an application of DP. We have the model

$$\begin{aligned} \theta_i|G &\sim G \\ x_i|\theta_i &\sim F(\theta_i) \end{aligned}$$

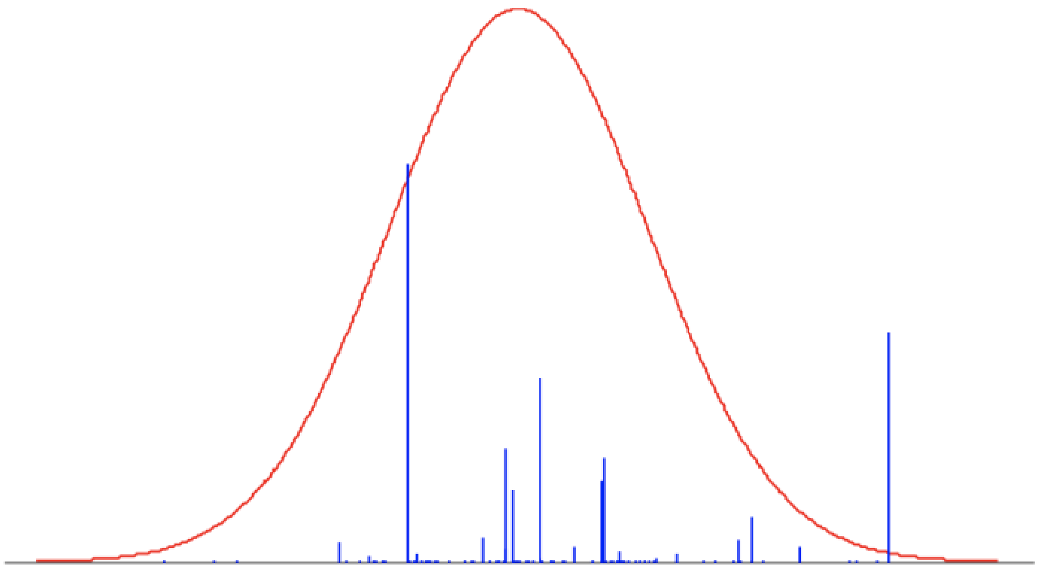


Figure 1: DP illustration

where $F(\theta_i)$ denotes the distribution of the observation x_i given θ_i. The factors θ_i are conditionally independent given G, and the observation x_i is conditionally independent of the other observations given the factor θ_i. When G is distributed according to a Dirichlet process, this model is referred to as a Dirichlet process mixture model.

1.2.4 Hierarchical Dirichlet processes (HDP)

We propose a nonparametric Bayesian approach to the modeling of grouped data, where each group is associated with a mixture model, and where we wish to link these mixture models. By analogy with Dirichlet process mixture models, we first define the appropriate nonparametric prior, which we refer to as the hierarchical Dirichlet process.

First, we have the global random probability measure G_0,

$$G_0|\gamma, H \sim DP(\gamma, H)$$

Then, we have conditional random measure G_j follow,

$$G_j|\alpha_0, G_0 \sim DP(\alpha_0, G_0)$$

In this model we have three hyper-parameters: baseline probability measure H, concentration parameters γ, α_0

For now, we can have HDP

$$\begin{aligned} \theta_{ji}|G_j &\sim G_j \\ G_j|\alpha_0, G_0 &\sim DP(\alpha_0, G_0) \\ G_0|\gamma, H &\sim DP(\gamma, H) \\ x_{ji}|\theta_{ji} &\sim F(\theta_{ji}), \text{ for each } j \text{ and } i \end{aligned}$$

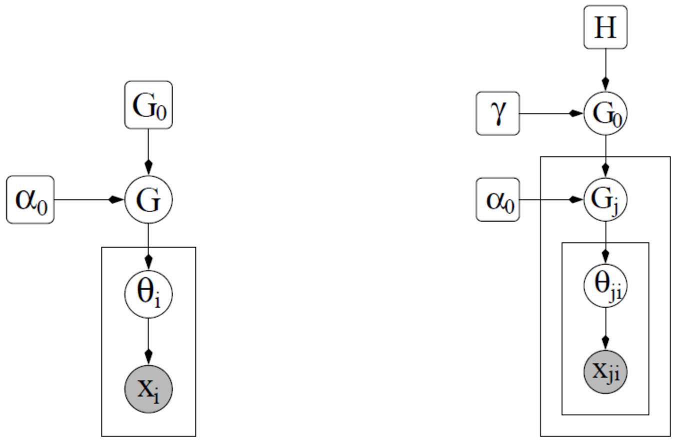


Figure 1: (Left) A representation of a Dirichlet process mixture model as a graphical model. (Right) A hierarchical Dirichlet process mixture model. In the graphical model formalism, each node in the graph is associated with a random variable, where shading denotes an observed variable. Rectangles denote replication of the model within the rectangle. Sometimes the number of replicates is given in the bottom right corner of the rectangle.

Figure 2: HDP illustration

Stick-breaking Representation. Given that the global measure G_0 is distributed as a Dirichlet process, it can be expressed using a stick-breaking representation:

$$G_0 = \sum_{k=1}^{\infty} \beta_k \delta_{\phi_k} \quad (16)$$

where $\phi_k \sim H$ independently and $\beta = (\beta_k)_{k=1}^{\infty} \sim \text{GEM}(\gamma)$ are mutually independent. Since G_0 has support at the points $\phi = (\phi_k)_{k=1}^{\infty}$, each G_j necessarily has support at these points as well, and can thus be written as:

$$G_j = \sum_{k=1}^{\infty} \pi_{jk} \delta_{\phi_k} \quad (17)$$

Let $\pi_j = (\pi_{jk})_{k=1}^{\infty}$. Note that the weights π_j are independent given β (since the G_j are independent given G_0). We now describe how the weights π_j are related to the global weights β.

Let $(A_1, ..., A_r)$ be a measurable partition, and $K_l = \{k : \phi_k \in A_l\}, l = 1, ..., r$ is a finite partition of the positive integers. We have

$$\begin{aligned} &(G_j(A_1), ..., G_j(A_r)) \sim Dir(\alpha_0 G_0(A_1), ..., \alpha_0 G_0(A_r)) \\ \Rightarrow &(\sum_{k \in K_1} \pi_{jk}, ..., \sum_{k \in K_r} \pi_{jk}) \sim Dir(\alpha_0 \sum_{k \in K_1} \beta_k, ..., \alpha_0 \sum_{k \in K_r} \beta_k) \quad (18) \end{aligned}$$

Then try to derive the explicit relationship of β, π_j.

$$\beta'_k \sim Beta(1,\gamma), \quad \beta_k = \beta'_k \prod_{l=1}^{k-1}(1-\beta'_l) \quad (20)$$

It can be showed that the following stick-breaking construction produces a random probability measure $\pi_j \sim DP(\alpha_0, \beta)$:

$$\pi'_{jk} \sim Beta(\alpha_0\beta_k, \alpha_0(1-\sum_{l=1}^{k}\beta_l)), \quad \pi_{jk} = \pi'_{jk}\prod_{l=1}^{k-1}(1-\pi'_{jl}) \quad (21)$$

To derive (21), first notice that for a partition $(\{1,...,k-1\},\{k\},\{k+1,k+2,...\})$, (18) gives

$$(\sum_{l=1}^{k-1}\pi_{jl}, \pi_{jk}, \sum_{l=k+1}^{\infty}\pi_{jl}) \sim Dir(\alpha_0\sum_{l=1}^{k-1}\beta_l, \alpha_0\beta_k, \alpha_0\sum_{l=k+1}^{\infty}\beta_l) \quad (22)$$

Removing the first element, and using standard properties of the Dirichlet distribution that support should be added to 1(Re-normalization Property). Dirichlet Distribution follows the renormalization property. Let $\pi_1,...,\pi_K \sim Dir(\alpha_1,...,\alpha_K)$,

$$\frac{(\pi_2,...,\pi_K)}{\sum_{k=2}^{K}\pi_k} \sim Dir(\alpha_2,...,\alpha_K)$$

So we have

$$\frac{1}{1-\sum_{l=1}^{k-1}\pi_{jl}}(\pi_{jk}, \sum_{l=k+1}^{\infty}\pi_{jl}) \sim Dir(\alpha_0\beta_k, \alpha_0\sum_{l=k+1}^{\infty}\beta_l) \quad (23)$$

Finally, define $\pi'_{jk} = \frac{\pi_{jk}}{1-\sum_{l=1}^{k-1}\pi_{jl}}$ and notice that $1-\sum_{l=1}^{k}\beta_l = \sum_{l=k+1}^{\infty}\beta_l$ to obtain (21)

$$(\pi'_{jk}, 1-\pi'_{jk}) \sim Dir(\alpha_0\beta_k, \alpha_0(1-\sum_{l=1}^{k}\beta_l))$$
$$\Rightarrow \pi'_{jk} \sim Beta(\alpha_0\beta_k, \alpha_0(1-\sum_{l=1}^{k}\beta_l))$$

Together with (16) and (17), this completes the description of the stick-breaking construction for hierarchical Dirichlet processes.

Chinese Restaurant Franchise (CRF) Process. HDP has a similar metaphor as the Chinese Restaurant Process to DP, which is called the Chinese restaurant franchise process. Different from original Chinese Restaurant Process, We have a restaurant franchise with a shared menu across the restaurants. At each table of each restaurant one dish is ordered from the menu by the first customer who sits there, and it is shared among all customers who sit at that table. Multiple tables in multiple restaurants can serve the same dish.

Process assumes $j = 1,...,J$ Chinese restaurants:

- with infinite tables in each restaurant;
- all share the same menu;
- 1st customer to a new table orders a plate for the table;

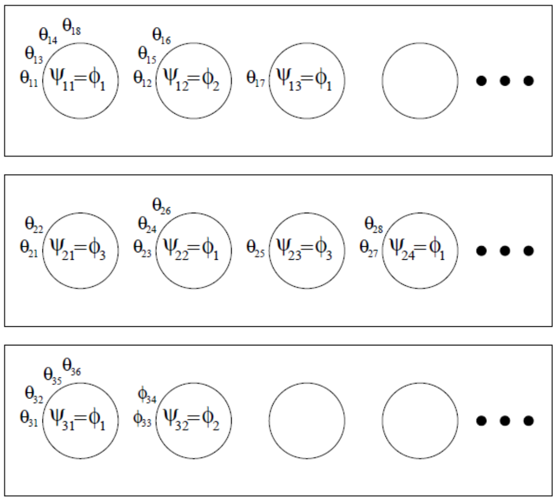

θ_{11} θ_{13} θ_{14} θ_{18} $\psi_{11}=\phi_1$
θ_{12} θ_{15} θ_{16} $\psi_{12}=\phi_2$
θ_{17} $\psi_{13}=\phi_1$
θ_{21} θ_{22} $\psi_{21}=\phi_3$
θ_{23} θ_{24} θ_{26} $\psi_{22}=\phi_1$
θ_{25} $\psi_{23}=\phi_3$
θ_{27} θ_{28} $\psi_{24}=\phi_1$
θ_{31} θ_{32} θ_{35} θ_{36} $\psi_{31}=\phi_1$
ϕ_{33} ϕ_{34} $\psi_{32}=\phi_2$

Figure 2: A depiction of a Chinese restaurant franchise. Each restaurant is represented by a rectangle. Customers (θ_{ji}'s) are seated at tables (circles) in the restaurants. At each table a dish is served. The dish is served from a global menu (ϕ_k), whereas the parameter ψ_{jt} is a table-specific indicator that serves to index items on the global menu. The customer θ_{ji} sits at the table to which it has been assigned in (24).

Figure 3: CRF

- multiple tables in multiple restaurants can serve the same plate.

Each customer i entering a restaurant j does the following:

- Chooses a table to sit proportion to the number of customers already at that table.
- If sits in a new table, then order a plate from the menu proportion to the popularity of the plate across all restaurants.

To construct the HDP, the restaurants correspond to groups and the customers correspond to the factors θ_{ji}. Let $\phi_1, ..., \phi_K$ be global menu of dishes, ψ_{jt} be the table-specific choice of dishes(served at table t in restaurant j). In the Chinese restaurant franchise metaphor, customer i in restaurant j sat at table t_{ji} while table t in restaurant j serves dish k_{jt}. We use the notation n_{jtk} to denote the number of customers in restaurant j at table t eating dish k, and m_{jk} denotes the number of tables in restaurant j serving dish k.

Based on CRF, we can compute the marginals by integrate out G_j and G_0.

Consider $\theta_{ji}|\theta_{j1}, ..., \theta_{j(i-1)}$ condition on G_0 (integrate out G_j):

$$\theta_{ji}|\theta_{j1}, ..., \theta_{j(i-1)}, \alpha_0, G_0 \sim \sum_{t=1}^{m_j} \frac{n_{jt}}{i-1+\alpha_0}\delta_{\psi_{jt}} + \frac{\alpha_0}{i-1+\alpha_0}G_0$$

Then, consider another layer $\psi_{jt}|\psi_{11}, \psi_{12}, ..., \psi_{21}, ..., \psi_{j(t-1)}$ condition on H (integrate out G_0):

$$\psi_{jt}|\psi_{11}, \psi_{12}, ..., \psi_{21}, ..., \psi_{j(t-1)}, \gamma, H \sim \sum_{k=1}^{K} \frac{m_{.k}}{m_{..}+\gamma}\delta_{\phi_k} + \frac{\gamma}{m_{..}+\gamma}H$$

where K the total number of unique plates from table ψ_{11} to table ψ_{jt}. $m_{.k}$ the total number of tables in all restaurants ordered plate t, and $m_{..}$ the total number of tables in all restaurants.

1.2.5 Experiments

The paper describes two experiments in this section to highlight the two aspects of the HDP: its nonparametric nature and its hierarchical nature.

The paper presents a third experiment highlighting the ease with which we can extend the framework to more complex models, specifically a hidden Markov model with a countably infinite state space.

Topics model. In the above mentioned metaphors, to relate to the problem of topic model we can consider restaurants, dishes as documents, topics respectively. Let H be a V-dimensional Dirichlet distribution, so a sample from H is a distribution over a vocabulary of V words.

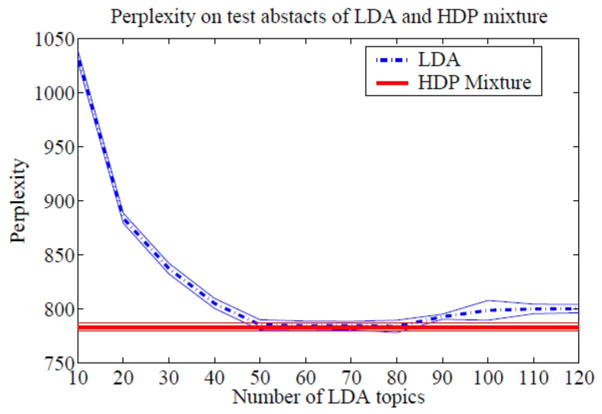
$$G_0 = \sum_{k=1}^{\infty} \pi_k \delta_{\beta_k} \sim DP(\alpha, H)$$

For each document $m = 1, ..., M$,

- Sample a distribution over topics $G_m \sim DP(\gamma, G_0)$.
- For each word $n = 1, ..., N_m$, we sample a topic $\phi_{mn} \sim \text{Discrete}(G_0)$ and a word $w_{mk} \sim \text{Discrete}(\phi_{mn})$.

In Figure (left), the perplexity is evaluated by mixture models range between 10 and 120. In Figure (right) the posterior over the number of topics is computed by the hierarchical Dirichlet Process mixture models of the optimal LDA models.

Figure: (Left) Comparison of latent Dirichlet allocation and the hierarchical Dirichlet process mixture. Results are averaged over 10 runs; the error bars are one standard error. (Right) Histogram of the number of topics for the hierarchical Dirichlet process mixture over 100 posterior samples.



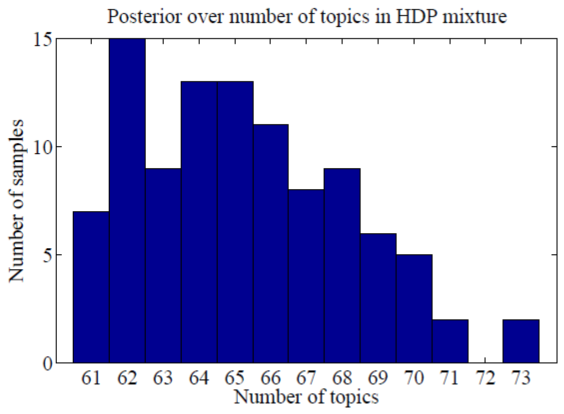


Figure 4: LDA, HDP

The rest part of the paper will be demonstrated by implementation.

2 Repeat and Research

2.1 Research 1: HDP Model with Latent Indicator Variables

Now, the question states that the stick-breaking representation of HDP is

$$\begin{aligned} & y_{ij} \mid \theta_{ij} \sim F(\theta_{ij}) && \theta_{ij} \mid G_j \sim G_j \\ & G_j = \sum_{k=1}^{\infty} \pi_{jk} \delta_{\phi_k} && \pi_{jk} = \pi'_{jk} \prod_{l=1}^{k-1} (1 - \pi_{jl}) \\ & \pi'_{jk} \sim \text{Beta}\left(\alpha_0 \beta_k, \alpha_0 (1 - \sum_{l=1}^{k} \beta_l)\right) && \beta_k \sim \beta'_k \prod_{l=1}^{k-1} (1 - \beta'_l) \\ & \beta'_k \sim \text{Beta}(1, \gamma) \\ & \phi_k \sim H \quad (\text{Prior for location } \phi_k) \end{aligned}$$

Then, now

$$\begin{aligned} & F(\cdot) = N(\cdot, \cdot) \quad \theta_{ij} = \{\mu_{ij}, \sigma^2_{ij}\} \\ & \phi_k = \{\mu_k, \sigma^2_k\} \end{aligned}$$

Then, now we introduce latent indicator variable z_{ij} (indicating which component the observation ij belongs to) In other words,

$$\theta_{ji} = \phi_{z_{ji}}$$

Then, using the latent representation, the model can be written as

$$\begin{aligned} y_{ij} \mid z_{ij}, \{\mu_k\}_{k=1}^L, \{\sigma^2_k\}_{k=1}^L &\sim N(\mu_{z_{ij}}, \sigma_{z_{ij}}) \\ z_{ij} \mid \{\pi_{jk}\}_{k=1}^L &\sim \{\pi_{jk}\}_{k=1}^L \\ \{\pi_{jk}\}_{k=1}^L \mid \alpha_0, \{\beta_k\}_{k=1}^L &\sim \text{Dir}(\alpha_0 \{\beta_k\}_{k=1}^L) \\ \{\beta_k\}_{k=1}^L \mid \gamma &\sim \text{Dir}\left(\frac{\gamma}{L}, \dots, \frac{\gamma}{L}\right) \end{aligned}$$

and

$$\begin{cases} \mu_k \mid H_\mu \sim H_\mu & \text{(Prior for } \mu_k) \\ \sigma_k^2 \mid H_{\sigma^2} \sim H_{\sigma^2} & \text{(Prior for } \sigma_k^2) \end{cases}$$

Graphical model is as follows:

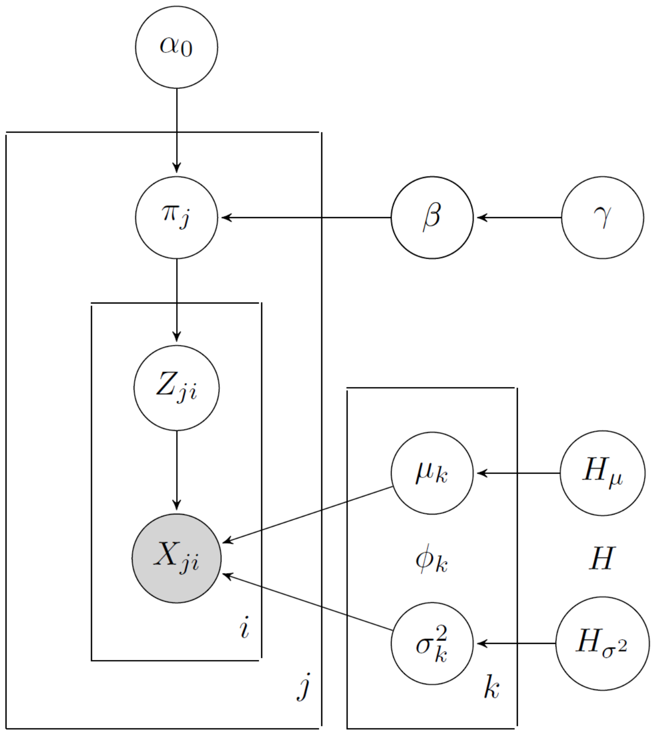


Figure 5: alt text here

2.2 Repeat: Chinese Restaurant Franchise Sampling Method (Teh et al., 2004)

2.2.1 Chinese Restrauant Process Derivation

In this part, we want to derive $ f_k^{-x_{ij}}(x_{ij})$ with general conjugate prior H.

We first derive the formula (30) in page 14; that is,

$$f_k^{-x_{ij}}(x_{ij}) = \frac{\int f(x_{ij} \mid \phi_k) \prod_{j'i' \neq ji, Z_{j'i'}=k} f(x_{i'j'} \mid \phi_k) h(\phi_k) d\phi_k}{\int \prod_{j'i' \neq ji, Z_{j'i'}=k} f(x_{i'j'} \mid \phi_k) h(\phi_k) d\phi_k}$$

and now, for $z_{ij} = k$,

$$x_{ij} \mid \underbrace{\{\mu_k, \sigma_k^2\}}_{=\phi_k} \sim \underbrace{N(\mu_k, \sigma_k^2)}_{=f(x_{ij}|\phi_k)}$$

and set the inverse gamma as a prior (because of conjugacy). I.e.,

$$(\mu_k, \sigma_k^2) \sim \underbrace{\mathcal{NIG}(m_0, V_0, \alpha_0, b_0)}_{=H} = \mathcal{N}(\mu|m_0, \sigma^2 V_0) IG(\sigma^2|\alpha_0, b_0)$$

$$= \frac{1}{\sqrt{2\pi V_0}} \frac{b_0^{\alpha_0}}{\Gamma(\alpha_0)} \frac{1}{\sigma_k} (\sigma_k^2)^{-\alpha_0 - 1} \exp\left(-\frac{1}{2\sigma_k^2} [V_0^{-1}(\mu_k - m_0)^2 + 2b_0]\right)$$

Then, because of conjugacy, if

$$\mu_{z_{ij}}, \sigma^2_{z_{ij}} \sim \mathcal{NIG}(m_0, V_0, \alpha_0, b_0)$$
$$x_{i,j} \mid \mu_{z_{ij}}, \sigma^2_{z_{ij}} \sim \mathcal{N}(\mu_{z_{ij}}, \sigma^2_{z_{ij}})$$

then, the posterior distribution is

$$\mu_{z_{ij}}, \sigma^2_{z_{ij}} \mid \{x_{i,j}\}_{i,j} \sim \mathcal{NIG}(\mu_{z_{ij}}, \sigma^2_{z_{ij}}; m_n, V_n, \alpha_n, b_n)$$

where

$$\begin{aligned}
m_n &= \frac{V_0^{-1} m_0 + n\overline{x}}{V_0^{-1} + n} \\
V_n^{-1} &= V_0^{-1} + n \\
\alpha_n &= \alpha_0 + \frac{n}{2} \\
b_n &= b_0 + \frac{1}{2}\left[m_0^2 V_0^{-1} + \sum_{i,j} x^2_{i,j} - m_n^2 V_n^{-1} \right]
\end{aligned}$$

with $n = \sum_{i'j' \neq ij} \delta(Z_{j'i'} = k)$ Then, notice that

$$\begin{aligned}
&h(\mu_{z_{i'j'}}, \sigma^2_{z_{i'j'}} \mid \{x_{i',j'}\}_{i',j' \neq i,j, Z_{i'j'}=k}) \\
&= \frac{\overbrace{h(\phi_k)}^{\text{Prior}} \overbrace{\prod_{j'i' \neq ji, Z_{j'i'}=k} f(x_{i'j'} \mid \phi_k)}^{\text{Likelihood}}}{\underbrace{\prod_{j'i' \neq ji, Z_{j'i'}=k} g(x_{i'j'})}_{\text{marginal}}}
\end{aligned}$$

but since the integral is over $\phi_k (= \{\mu_k, \sigma_k^2\})$ and both numerator and denominator have the same term, they cancel out. Hence,

$$\begin{aligned}
f_k^{-x_{ij}}(x_{ij}) &= \frac{\int f(x_{ij} \mid \phi_k) \prod_{j'i' \neq ji, Z_{j'i'}=k} f(x_{i'j'} \mid \phi_k) h(\phi_k) d\phi_k}{\int \prod_{j'i' \neq ji, Z_{j'i'}=k} f(x_{i'j'} \mid \phi_k) h(\phi_k) d\phi_k} \\
&= \frac{\frac{1}{\prod_{j'i' \neq ji, Z_{j'i'}=k} g(x_{i'j'})} \int f(x_{ij} \mid \phi_k) \prod_{j'i' \neq ji, Z_{j'i'}=k} f(x_{i'j'} \mid \phi_k) h(\phi_k) d\phi_k}{\frac{1}{\prod_{j'i' \neq ji, Z_{j'i'}=k} g(x_{i'j'})} \int \prod_{j'i' \neq ji, Z_{j'i'}=k} f(x_{i'j'} \mid \phi_k) h(\phi_k) d\phi_k} \\
&= \frac{\int \frac{f(x_{ij}|\phi_k) \prod_{j'i' \neq ji, Z_{j'i'}=k} f(x_{i'j'}|\phi_k) h(\phi_k)}{\prod_{j'i' \neq ji, Z_{j'i'}=k} g(x_{i'j'})} d\phi_k}{\int \frac{\prod_{j'i' \neq ji, Z_{j'i'}=k} f(x_{i'j'}|\phi_k) h(\phi_k)}{\prod_{j'i' \neq ji, Z_{j'i'}=k} g(x_{i'j'})} d\phi_k} \\
&= \frac{\int f(x_{ij} \mid \phi_k) h(\mu_{z_{i'j'}}, \sigma^2_{z_{i'j'}} \mid \{x_{i',j'}\}_{i',j' \neq i,j, Z_{i'j'}=k}) d\phi_k}{\underbrace{\int h(\mu_{z_{i'j'}}, \sigma^2_{z_{i'j'}} \mid \{x_{i',j'}\}_{i',j' \neq i,j, Z_{i'j'}=k}) d\phi_k}_{=1\ (\because \text{ integrating over } \mu_k, \sigma_k^2)}} \\
&= \int \underbrace{f(x_{ij} \mid \mu_k, \sigma_k^2)}_{\mathcal{N}(x_{ij}; \mu_k, \sigma_k^2)} \underbrace{h(\mu_k, \sigma_k^2 \mid \{x_{i',j'}\}_{i',j' \neq i,j, Z_{i'j'}=k})}_{\mathcal{NIG}(m_n, V_n, \alpha_n, b_n)} d\{\mu_k, \sigma_k^2\}
\end{aligned}$$

Also, notice that

$$\mathcal{N}(x_{ij}; \mu_k, \sigma_k^2) = \delta(z_{ij} = k) \mathcal{N}(x_{ij}; \mu_{z_{ij}}, \sigma^2_{z_{ij}})$$

Then, by bayes theorem,

$$\underbrace{h(\mu_k, \sigma_k^2 \mid x_{ij}, \{x_{i',j'}\}_{i',j' \neq i,j, Z_{i'j'}=k})}_{\text{(New) Posterior}} = \frac{f(x_{ij} \mid \mu_k, \sigma_k^2) h(\mu_k, \sigma_k^2 \mid \{x_{i',j'}\}_{i',j' \neq i,j, Z_{i'j'}=k})}{\underbrace{\int f(x_{ij} \mid \mu_k, \sigma_k^2) h(\mu_k, \sigma_k^2 \mid \{x_{i',j'}\}_{i',j' \neq i,j, Z_{i'j'}=k}) d\{\mu_k, \sigma_k^2\}}_{=f_k^{-x_{ij}}(x_{ij})}}$$

where the new posterior is

$$h(\mu_k, \sigma_k^2 \mid x_{ij}, \{x_{i',j'}\}_{i',j' \neq i,j, Z_{i'j'}=k}) = \mathcal{NIG}(m_s, V_s, \alpha_s, b_s)$$

where

$$\begin{aligned} m_s &= \frac{V_n^{-1} m_n + n_s \overline{x}}{V_n^{-1} + n_s} \\ V_s^{-1} &= V_n^{-1} + n_s \\ \alpha_s &= \alpha_n + \frac{n_s}{2} \\ b_s &= b_n + \frac{1}{2} \left[m_n^2 V_n^{-1} + \sum_{i,j} x_{i,j}^2 - m_s^2 V_s^{-1} \right] \end{aligned}$$

where n_s is the number of observations x_{ij} (input of the objective function).

Hence, you can calculate the quantity of interest without solving any integral. In other words,

$$\begin{aligned} f_k^{-x_{ij}}(x_{ij}) &= \frac{\overbrace{f(x_{ij} \mid \mu_k, \sigma_k^2)}^{\text{Likelihood}} \overbrace{h(\mu_k, \sigma_k^2 \mid \{x_{i',j'}\}_{i',j' \neq i,j, Z_{i'j'}=k})}^{\text{(New) Prior}}}{\underbrace{h(\mu_k, \sigma_k^2 \mid x_{ij}, \{x_{i',j'}\}_{i',j' \neq i,j, Z_{i'j'}=k})}_{\text{Posterior}}} \\ &= \frac{\delta(Z_{ij} = k) \mathcal{N}(x_{ij}; \mu_{z_{ij}}, \sigma_{z_{ij}}^2) \times \mathcal{NIG}(\mu_k, \sigma_k^2; m_n, V_n, \alpha_n, b_n)}{\mathcal{NIG}(\mu_k, \sigma_k^2; m_s, V_s, \alpha_s, b_s)} \end{aligned}$$

2.2.2 CRF Sampling

Rather than dealing with the θ_{ji}'s and ψ_{jt}'s directly, we shall sample their index variables t_{ji} and k_{jt} instead. Motivation: theθ_{ji}'s and ψ_{jt}'s can be reconstructed from these index variables and the ϕ_k's. Advantage: It makes the MCMC sampling scheme more efficient. Notice that the t_{ji} and k_{jt} inherit the exchangeability properties of the θ_{ji} and ψ_{jt}, allowing us to adapt the conditional distributions to be expressed in terms of t_{ji} and k_{jt}.

Let $F(\theta)$ have density $f(\cdot \mid \theta)$ and H have density $h(\cdot)$

Denote the conditional density of x_{ji} under mixture component k given all data items except x_{ji} as

$$f_k^{-x_{ji}}(x_{ji}) = \frac{\int f(x_{ji} \mid \phi_k) \prod_{j'i' \neq ji, z_{j'i'}=k} f(x_{j'i'|\phi_k}) h(\phi_k) d\phi_k}{\int \prod_{j'i' \neq ji, z_{j'i'}=k} f(x_{j'i'|\phi_k}) h(\phi_k) d\phi_k}$$

$$f_{k^{new}}^{-x_{ji}}(x_{ji}) = \int f(x_{ji|\phi}) h(\phi) d\phi$$

Sampling t. We Obtain the conditional posterior for t_{ji} by combining the conditional prior distribution for t_{ji} with the likelihood of generating x_{ji}.

$$p(x_{ji} = t \mid \boldsymbol{t}^{-ji}, t_{ji} = t^{new}, \boldsymbol{k}) = \sum_{k=1}^{K} \frac{m_{\cdot k}}{m_{\cdot\cdot} + \gamma} f_k^{-x_{ji}}(x_{ji}) + \frac{\gamma}{m_{\cdot\cdot} + \gamma} f_{k^{new}}^{-x_{ji}}(x_{ji})$$

$$p(t_{ji} = t \mid \boldsymbol{t}^{-ji}, \boldsymbol{k}) \propto \begin{cases} n_{jt\cdot}^{-ji} f_{k_{jt}}^{-x_{ji}}(x_{ji}) & \text{if } t \text{ previously used,} \\ \alpha_0 p(x_{ji} \mid \boldsymbol{t})^{-ji}, t_{ji} = t^{new}, \boldsymbol{k}) & \text{if } t = t^{new} \end{cases}$$

If the sampled value of t_{ji} is t^{new}, we obain a sample of $k_{jt^{new}}$ by sampling from

$$p(k_{jt^{new}} = k \mid \boldsymbol{t}, \boldsymbol{k}^{-jt^{new}}) \propto \begin{cases} m_{\cdot k} f_k^{-x_{ji}}(x_{ji}) & \text{if } k \text{ previously used,} \\ \gamma f_{k^{new}}^{-x_{ji}}(x_{ji}) & \text{if } k = k^{new} \end{cases}$$

If some table t becomes unoccupied as a result of updating t_{ji}, i.e., $n_{jt\cdot} = 0$, then the probability that this table will be reoccupied in the future will be zero. Hence, we delete the corresponding k_{jt} from the data structure.

Sampling k. The conditional probability of k_{jt} is

$$p(k_{jt} = k \mid \boldsymbol{t}, \boldsymbol{k}^{-jt}) \propto \begin{cases} m_{\cdot k}^{-jt} f_k^{-\boldsymbol{x}_{jt}}(\boldsymbol{x}_{jt}) & \text{if } k \text{ previously used,} \\ \gamma f_{k^{new}}^{-\boldsymbol{x}_{jt}}(\boldsymbol{x}_{jt}) & \text{if } k = k^{new}. \end{cases}$$

Now we implement the Markov chain Monte Carlo sampling schemes for the hierarchical Dirichlet process mixture model in the Chinese restaurant franchise. Since we can reconstruct θ_{ji} and ψ_{jt} from t_{ji}, k_{jt} and the ϕ_k 's, we could sample t_{ji} and k_{jt} instead to obtain a more efficient sampling scheme. Since our tasks required us to compare inference results obtained using the slice sampler with the results obtained here. To implement a more consistent comparison, we first choose a very simple prior and a likelihood here, which is

$$\phi_k \mid H \sim H = N(0, \frac{1}{\tau_\phi^2});$$

$$x_{ji} \mid t_{ji}, (k_{jt})_{t=1}^T, (\phi_k)_{k=1}^L \sim F(\phi_{k_{jt_{ji}}}) = N(\phi_{k_{jt_{ji}}}, \frac{1}{\tau_x^2}).$$

Though this setting may obey the original intention of The Bayesian nonparametrics, its implementation could offer us more insight for the dataset and the sampling scheme. Now we derive the conditional density of x_{ji} under mixture component k given all data except x_{ji} (i.e. $\{x_{j'i';j'\neq j,i'\neq i}\}$).

$$\int_{\mathbb{R}} \prod_{j'i'\neq ji, z_{j'i'}=k} f(x_{j'i'} \mid \phi_k) h(\phi_k) \mathrm{d}\phi_k = \int_{\mathbb{R}} \prod_{j'i'\neq ji, z_{j'i'}=k} \frac{\tau_x}{\sqrt{2\pi}} \mathrm{e}^{-\frac{\tau_x^2 (x_{j'i'}-\phi_k)^2}{2}} \frac{\tau_\phi}{\sqrt{2\pi}} \mathrm{e}^{-\frac{\tau_\phi^2 \phi_k^2}{2}} \mathrm{d}\phi_k$$

$$= (\frac{\tau_x}{\sqrt{2\pi}})^{n_{\cdot\cdot k}^{-ji}} \tau_\phi \sqrt{\frac{1}{\tau_x^2 n_{\cdot\cdot k}^{-ji} + \tau_\phi^2}} \exp\Big(-\frac{1}{2}(\tau_x^2 \sum_{j'i'\neq ji, z_{j'i'}=k} x_{j'i'}^2 - \frac{(\tau_x^2 \sum_{j'i'\neq ji, z_{j'i'}=k} x_{j'i'})^2}{\tau_x^2 n_{\cdot\cdot k}^{-ji} + \tau_\phi^2})\Big)$$

Similarly we could get the numerator

$$\int_{\mathbb{R}} f(x_{ji} \mid \phi_k) \prod_{j'i'\neq ji, z_{j'i'}=k} f(x_{j'i'} \mid \phi_k) h(\phi_k) \mathrm{d}\phi_k$$

and

$$f_{k^{\text{new}}}^{-x_{ji}}(x_{ji}) = \int_{\mathbb{R}} f(x_{ji} \mid \phi) h(\phi) \mathrm{d}\phi.$$

Thus,

$$f_k^{-x_{ji}}(x_{ji}) = \frac{\int_{\mathbb{R}} f(x_{ji} \mid \phi_k) \prod_{j'i' \neq ji, z_{j'i'}=k} f(x_{j'i'} \mid \phi_k) h(\phi_k) \mathrm{d}\phi_k}{\int_{\mathbb{R}} \prod_{j'i' \neq ji, z_{j'i'}=k} f(x_{j'i'} \mid \phi_k) h(\phi_k) \mathrm{d}\phi_k}$$
$$= \frac{\tau_x}{\sqrt{2\pi}} \sqrt{\frac{\tau_x^2 n_{\cdot\cdot k}^{-ji} + \tau_\phi^2}{\tau_x^2 (n_{\cdot\cdot k}^{-ji} + 1) + \tau_\phi^2}} \exp\left(\tau_x^2 x_{ji}^2 + \frac{(\tau_x^2 \sum_{j'i' \neq ji, z_{j'i'}=k} x_{j'i'})^2}{\tau_x^2 n_{\cdot\cdot k}^{-ji} + \tau_\phi^2} - \frac{\left(\tau_x^2 (x_{ji} + \sum_{j'i' \neq ji, z_{j'i'}=k} x_{j'i'})\right)^2}{\tau_x^2 (n_{\cdot\cdot k}^{-ji} + 1) + \tau_\phi^2}\right);$$

$$f_{k^{\mathrm{new}}}^{-x_{ji}}(x_{ji}) = \frac{\tau_x \tau_\phi}{\sqrt{2\pi}} \sqrt{\frac{1}{\tau_x^2 + \tau_\phi^2}} \exp\left(-\frac{\tau_x^2 \tau_\phi^2 x_{ji}^2}{2(\tau_x^2 + \tau_\phi^2)}\right).$$

Now we load the data and write these conditional densities of x_{ji} into functions.

```r
#### Loading the given data
raw.data<-read.csv("final_data.csv")
attach(raw.data)

## pre-processing data
X1 = logArea[group == 'immuno']
X2 = logArea[group == 'cryo']
X = logArea

all_i.1 = length(X1)
all_i.2 = length(X2)
all_i = length(X)

## set up this HDP
tau.phi = 1
tau.x = 1
alpha0 = 1
gam = 1

## compute the conditional density x_ji|x_-ji
f_k = function(i, x_list, k, k_list){
  x = x_list[i]
  k_list_cut = k_list[-i]
  x_list_cut = x_list[-i]
  n.k = sum(k_list_cut == k)
  x_ji_list = x_list_cut[k_list_cut == k]

  factor = sqrt((tau.x^2*n.k+tau.phi^2)/(tau.x^2*(n.k+1)+tau.phi^2))
  power = tau.x^2*x^2+(tau.x^2*sum(x_ji_list))^2/(tau.x^2*n.k+tau.phi^2)-
    (tau.x^2*(x+sum(x_ji_list)))^2/(tau.x^2*(n.k+1)+tau.phi^2)
  result = (tau.x/sqrt(2*pi))*factor*exp(-0.5*power)
  return(result)
}

f_k_new = function(i, x_list){
  x = x_list[i]
  factor = sqrt(1/(tau.x^2+tau.phi^2))
  power = (tau.x^2*tau.phi^2*x^2)/(tau.x^2+tau.phi^2)
  result = (tau.x*tau.phi/sqrt(2*pi))*factor*exp(-0.5*power)
  return(result)
}
```

To get the likelihood for $t_{ji} = t^{\text{new}}$, which is

$$p(x_{ji} \mid \boldsymbol{t}^{-ji}, t_{ji} = t^{\text{new}}, \boldsymbol{k}) = \sum_{k=1}^{K} \frac{m_{\cdot k}}{m_{\cdot\cdot} + \gamma} f_k^{-x_{ji}}(x_{ji}) + \frac{\gamma}{m_{\cdot\cdot} + \gamma} f_{k^{\text{new}}}^{-x_{ji}}(x_{ji}),$$

we could build up the following function.

```
P_t_new = function(i, x_list, t_list, k_list){
  K = max(k_list)
  k_list1 = k_list[1:all_i.1]
  k_list2 = k_list[(all_i.1+1):all_i]
  t_list1 = t_list[1:all_i.1]
  t_list2 = t_list[(all_i.1+1):all_i]
  m = length(unique(t_list1)) + length(unique(t_list2))
  prod = 0
  for (k in 1:K){
    rec1 = t_list1[k_list1 == k]
    val1 = length(unique(rec1))
    rec2 = t_list2[k_list2 == k]
    val2 = length(unique(rec2))
    prod = prod + (val1+val2)*f_k(i, x_list, k, k_list)
  }
  result = (prod+gam*f_k_new(i, x_list))/(m+gam)
  return(result)
}
```

After defining these functions for the categorical distribution of sampling t, we could define the sampling function.

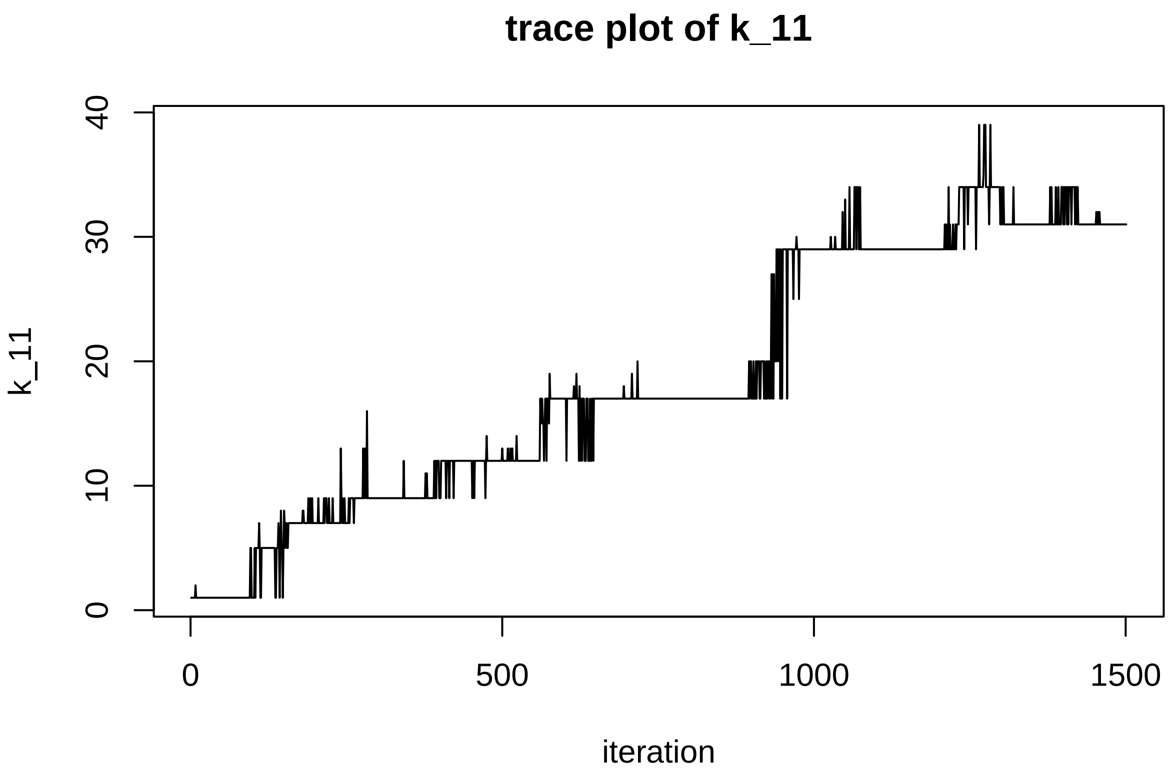
```
update_t = function(i, x_list, t_list, k_list){
  if (i <= all_i.1){
    x = x_list[i]
    k_list1 = k_list[1:all_i.1]
    k_list_cut1 = k_list1[-i]
    t_list1 = t_list[1:all_i.1]
    t_list_cut1 = t_list1[-i]
    t_unique1 = unique(t_list_cut1)
    n1 = length(t_unique1)
    T1 = max(abs(t_unique1))
    prob = rep(0, n1+1)
    for (j in 1:n1){
      n = sum(t_list_cut1 == t_unique1[j])
      k = k_list_cut1[t_list_cut1 == t_unique1[j]][1]
      f = f_k(i, x_list, k, k_list)
      prob[j] = n*f
      }
    prob[n1+1] = alpha0 * P_t_new(i, x_list, t_list, k_list)
    prob = prob/sum(prob)
    new_t = sample(c(t_unique1,T1+1), 1, replace = TRUE, prob = prob)
    if (new_t == T1+1){
      ## sample new_k
      K = max(k_list)
      k_list1 = k_list[1:all_i.1]
      k_list2 = k_list[(all_i.1+1):all_i]
      t_list1 = t_list[1:all_i.1]
      t_list2 = t_list[(all_i.1+1):all_i]
      k_unique1 = unique(k_list)
      n.k1 = length(k_unique1)
      K1 = max(k_unique1)
      prob_k = rep(0, n.k1+1)
      for (j in 1:n.k1){
        rec1 = t_list1[k_list1 == k_unique1[j]]
        val1 = length(unique(rec1))
        rec2 = t_list2[k_list2 == k_unique1[j]]
        val2 = length(unique(rec2))
        prob_k[j] = (val1+val2)*f_k(i, x_list, k_unique1[j], k_list)
      }
      prob_k[n.k1+1] = gam * f_k_new(i, x_list)
      prob_k = prob_k/sum(prob_k)
      new_k = sample(c(k_unique1,K1+1), 1, replace = TRUE, prob = prob_k)
    }
    else{
      new_k = k_list_cut1[t_list_cut1 == new_t][1]
    }
  }
  else{
    #prob = rep(0, T+1)
    x = x_list[i]
    k_list2 = k_list[(all_i.1+1):all_i]
    k_list_cut2 = k_list2[-(i-all_i.1)]
    t_list2 = t_list[(all_i.1+1):all_i]
    t_list_cut2 = t_list2[-(i-all_i.1)]
    t_unique2 = unique(t_list_cut2)
    n2 = length(t_unique2)
    T2 = max(abs(t_unique2))
    prob = rep(0, n2+1)
    for (j in 1:n2){
      n = sum(t_list_cut2 == t_unique2[j])
      k = k_list_cut2[t_list_cut2 == t_unique2[j]][1]
      f = f_k(i, x_list, k, k_list)
      prob[j] = n*f
      }
    prob[n2+1] = alpha0 * P_t_new(i, x_list, t_list, k_list)
    prob = prob/sum(prob)
    new_t = sample(c(t_unique2, -T2-1), 1, replace = TRUE, prob = prob)
    if (new_t == -T2-1){
      ## sample new_k
      K = max(k_list)
      k_list1 = k_list[1:all_i.1]
      k_list2 = k_list[(all_i.1+1):all_i]
      t_list1 = t_list[1:all_i.1]
      t_list2 = t_list[(all_i.1+1):all_i]
      k_unique2 = unique(k_list)
      n.k2 = length(k_unique2)
      K2 = max(k_unique2)
      prob_k = rep(0, n.k2+1)
      for (j in 1:n.k2){
        rec1 = t_list1[k_list1 == k_unique2[j]]
        val1 = length(unique(rec1))
        rec2 = t_list2[k_list2 == k_unique2[j]]
        val2 = length(unique(rec2))
        prob_k[j] = (val1+val2)*f_k(i, x_list, k_unique2[j], k_list)
      }
      prob_k[n.k2+1] = gam * f_k_new(i, x_list)
      prob_k = prob_k/sum(prob_k)
      new_k = sample(c(k_unique2,K2+1), 1, replace = TRUE, prob = prob_k)
    }
    else{
      new_k = k_list_cut2[t_list_cut2 == new_t][1]
    }
  }
  t_list[i] = new_t
  k_list[i] = new_k
  return(c(t_list, k_list))
}
```

Based on this, we run our MCMC sampling scheme for 1500 times.

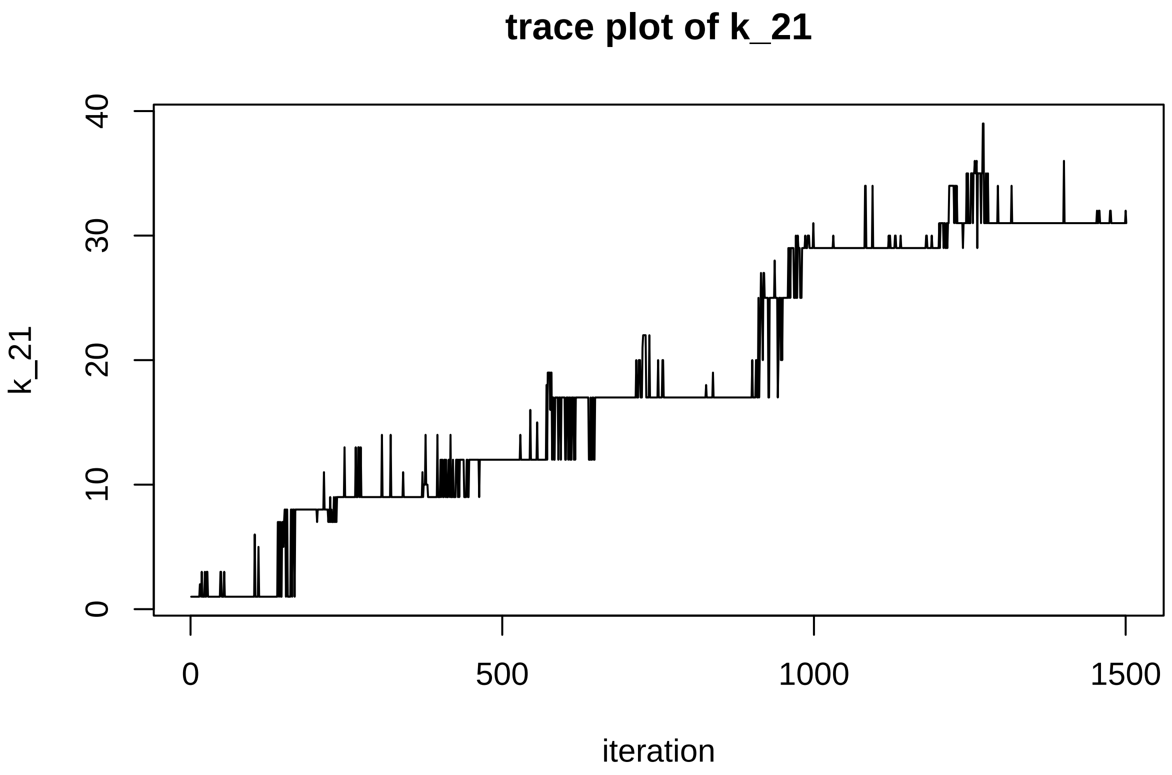
```
set.seed(123)
## Initialize
t_list = rep(1, all_i)
t_list[(all_i.1+1):all_i] = rep(-1, all_i.2)
k_list = rep(1, all_i)
x_list = X

## save the sampling result
t_trace = t_list
k_trace = k_list

N = 1500
for (n in 1:N){
  for (i in 1:all_i){
  tk_list = update_t(i, x_list, t_list, k_list)
  t_list = tk_list[1:all_i]
  k_list = tk_list[(all_i+1):(2*all_i)]
  }
  t_trace = cbind(t_trace, t_list)
  k_trace = cbind(k_trace, k_list)
}
```

Now we check our sampling results of $k_{jt_{ji}}$.

```
par(mfrow=c(1,1))
plot(k_trace[1,], type='l', main = "trace plot of k_11",
     xlab='iteration', ylab='k_11')
```



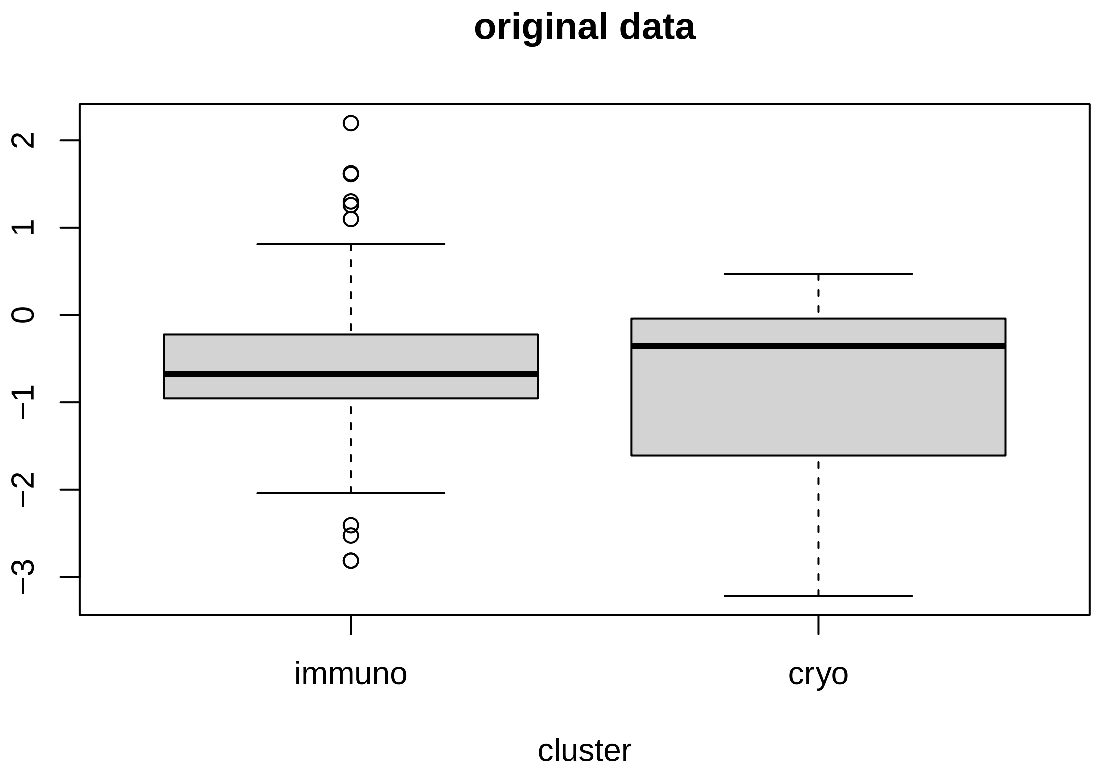
```
par(mfrow=c(1,1))
plot(k_trace[(all_i.1+1),], type='l',main = "trace plot of k_21",
     xlab="iteration", ylab='k_21')
```



From the trace plots of k_{11} and k_{21}, we could see that the k tends to converge.

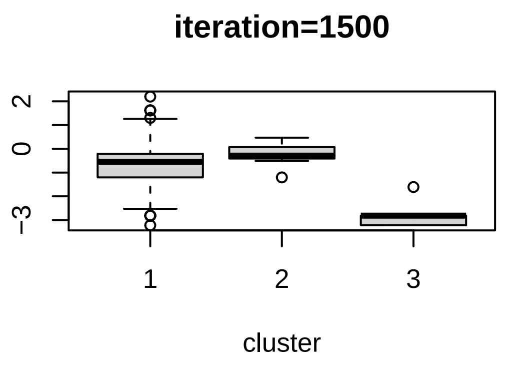
Then we use boxplots to check the clustering results. The original grouped data is distributed like the following group.

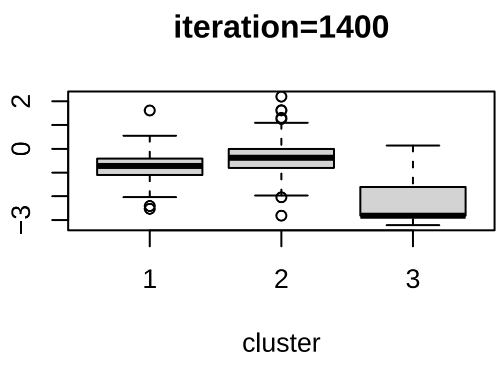
```r
par(mfrow=c(1,1))
boxplot(X1,X2, main="original data",xlab="cluster",xaxt="n")
axis(1, at = c(1,2),label=c("immuno","cryo"), las=1)
```

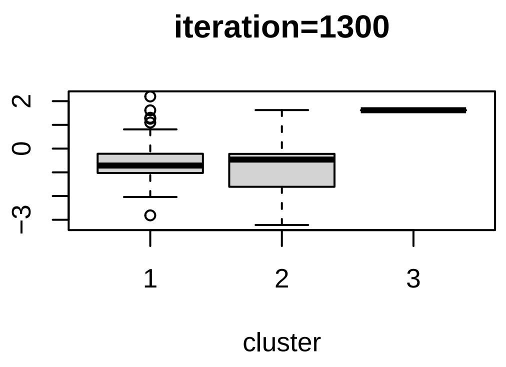


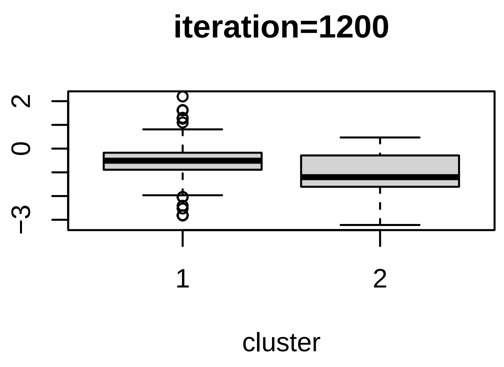
Then for the sampling iteration times 1500, 1400, 1300, 1200, we can plot their cluster distribution in the following plots.

```r
par(mfrow=c(2,2))
col_num = 1501
boxplot(X[k_trace[,col_num] == 31], X[k_trace[,col_num] == 32], X[k_trace[,col_num] == 33],
        main="iteration=1500",xlab="cluster",xaxt="n")
axis(1, at = c(1,2,3), las=1)

col_num = 1401
boxplot(X[k_trace[,col_num] == 34], X[k_trace[,col_num] == 31], X[k_trace[,col_num] == 36],
        main="iteration=1400",xlab="cluster",xaxt="n")
axis(1, at = c(1,2,3), las=1)

col_num = 1301
boxplot(X[k_trace[,col_num] == 34], X[k_trace[,col_num] == 31], X[k_trace[,col_num] == 35],
        main="iteration=1300",xlab="cluster",xaxt="n")
axis(1, at = c(1,2,3), las=1)

col_num = 1201
boxplot(X[k_trace[,col_num] == 29], X[k_trace[,col_num] == 31],main="iteration=1200",
        xlab="cluster",xaxt="n")
axis(1, at = c(1,2), las=1)
```





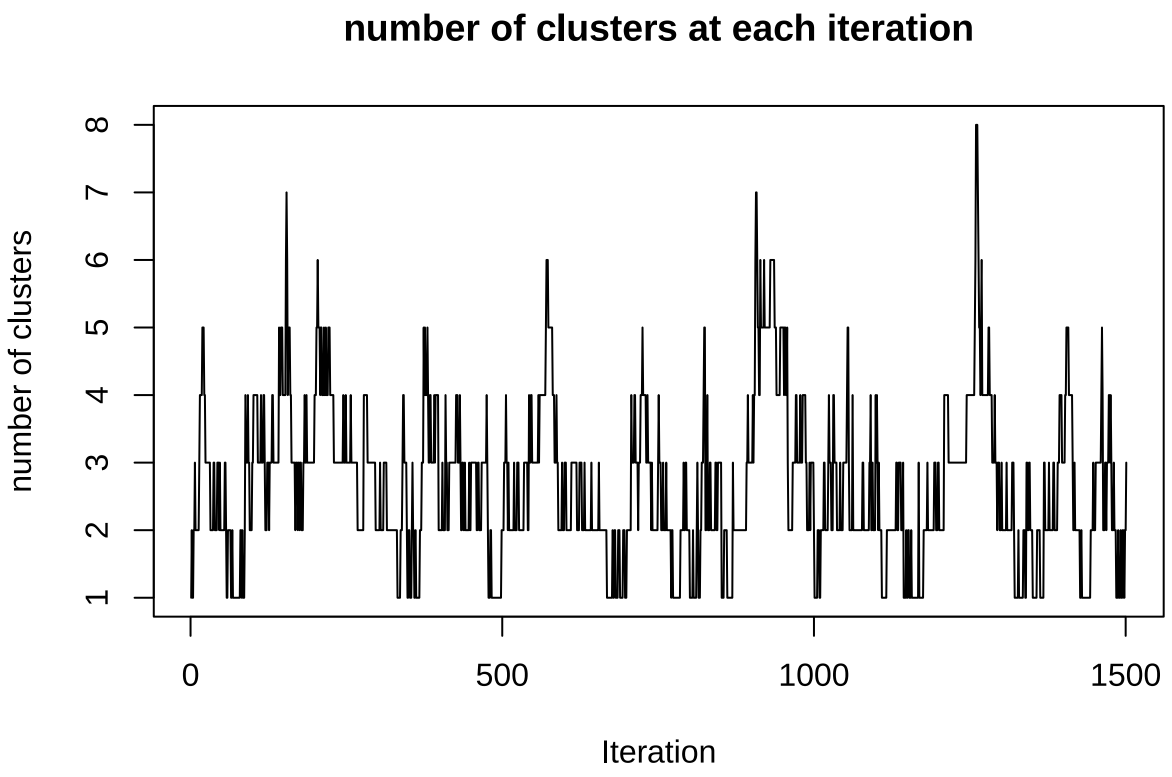




We can see the clustering results are not very stable. The number of clusters are changing, and the distribution are varying. There seems to always exist a cluster having a non-skewed distribution and having several outliers in both direction.

We could also check the number of clusters at each iteration.

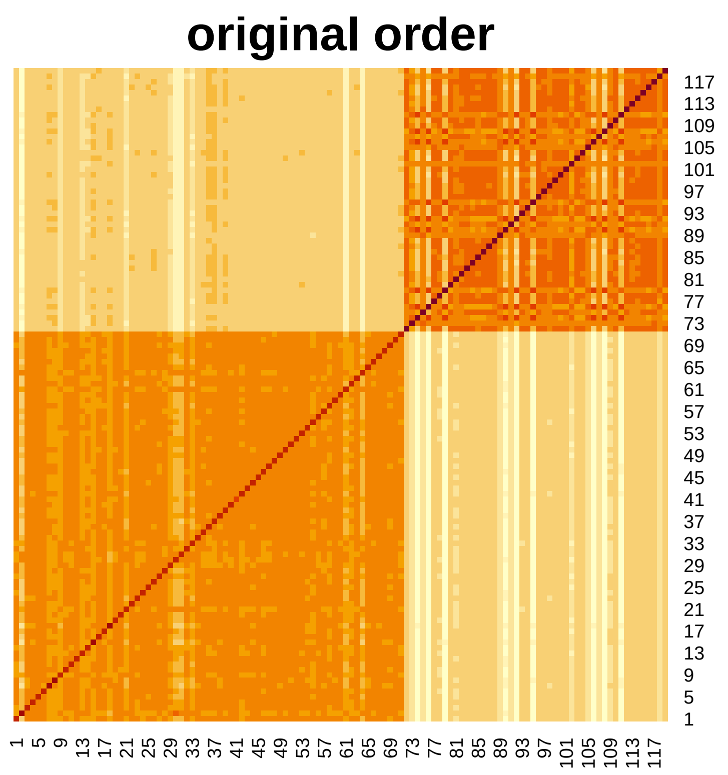
```
par(mfrow=c(1,1))
l = ncol(k_trace)
num_list = rep(0, l)
for (i in 1:l){
  num_list[i] = length(unique(k_trace[,i]))
}
plot(1:l, num_list, 'l', main="number of clusters at each iteration",
     xlab='Iteration', ylab='number of clusters')
```



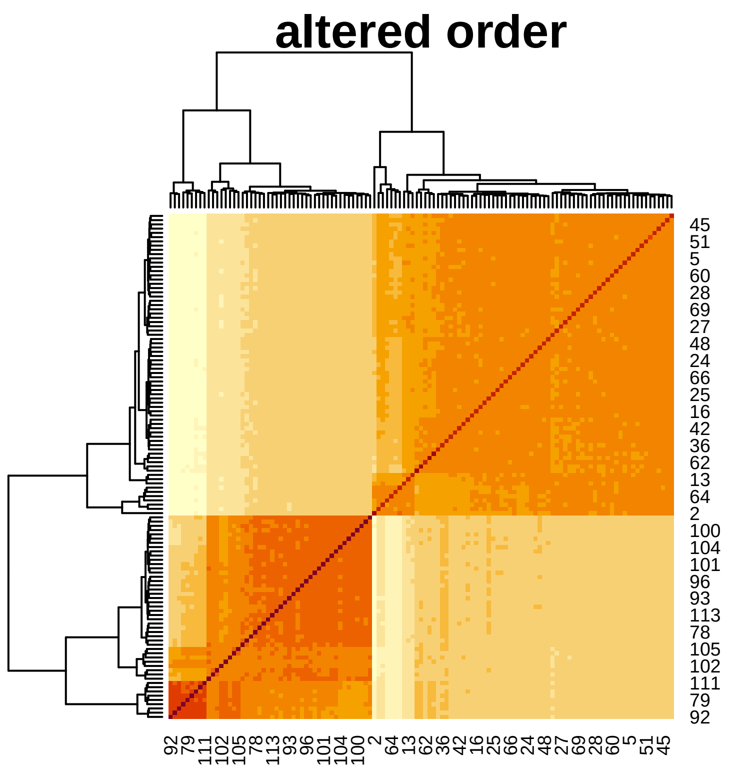
We can see after the 1000-th iteration, there are 2 or 3 clusters most of the time.

To get a better glimpse of the index j and i in each cluster, we can use heatmap to visualize the clustering result. For the following plots, points $1, \dots 71$ are data x_{1i} $(i = 1, \dots, 71)$ in immuno group, points $72, \dots 119$ are data x_{2i} $(i = 1, \dots, 48)$ in cryo group.

```
simu_mat = matrix(0, all_i, all_i)
for (i in 1:all_i){
  for (j in 1:all_i){
    simu_mat[i,j] = sum(k_trace[i,] == k_trace[j,]) / l
  }
}
par(mfrow=c(1,2))
heatmap(simu_mat, Rowv = NA, Colv = NA, main="original order")
```



```
heatmap(simu_mat, main="altered order")
```



In these two plots, the darker color indicates its row index and column index are clustered into the same cluster more often. From the first plot, we can see that the data from the same original group ("immuno" or "cryo") are still more likely to fall into the same clusters. The block structure is obvious. From the second plot, we can see there may be more cluster structure needs to be discovered. Along the diagonal in the plot, there are at least 3 apparent block structures. The left two of them have data points mostly from "cryo" group.

2.3 Research 2: Slice Sampler for HDP Model with Latent Indicator Variables

In this part, we will show the result of using slice sampler for HDP model with latent indicator variables.

2.3.1 HDP with Only Labels

To identify the cluster membership, we only need to consider the labels of HDP. Here we use the general version of HDP. In practice, we can choose different kernels, but the logic should be the same.

Consider the following HDP model.

$$\begin{aligned} \beta|\gamma_0 &\sim \text{GEM}(\gamma_0) \\ \pi_j|\alpha_0,\beta &\sim \text{DP}(\alpha_0,\beta). \\ z_{ij}|\pi_j &\sim \pi_j \end{aligned}$$

As π_j is also a Dirichlet Process, we can also use the stick-breaking expression on π_j. Then we can get

$$\begin{aligned} \beta|\gamma_0 &\sim \text{GEM}(\gamma_0) \\ \gamma_j|\alpha_0 &\sim \text{GEM}(\alpha_0), \quad \gamma_j = (\gamma_{jt})_{t=1}^{\infty} \\ k_{jt}|\beta &\sim \beta \\ \pi_j &= \sum_{t=1}^{\infty} \gamma_{jt}\delta_{k_{jt}} \\ z_{ij}|\pi_j &\sim \pi_j \end{aligned} .$$

As this set of expressions is still in teh form of infinite summation, we rewrite it by introducing t_{ij}.

$$\begin{aligned} \beta|\gamma_0 &\sim \text{GEM}(\gamma_0) \\ \gamma_j|\alpha_0 &\sim \text{GEM}(\alpha_0), \quad \gamma_j = (\gamma_{jt})_{t=1}^{\infty} \\ k_{jt}|\beta &\sim \beta \\ t_{ij}|\gamma_j &\sim \gamma_j \\ z_{ij}|t_j,k_j &= k_{j,t_{ij}} \end{aligned} .$$

Note that z_{ij} is totally decided by β, γ_j, k_j and t_j, thus, we only need to consider sampling these four parameters. We have the following items,

$$\begin{aligned} [F(x)]_1 := x_1, &\quad [F(x)]_j := x_j \prod_{l=1}^{j-1}(1-x_l) \\ \gamma'_{jt} \sim \text{Beta}(1,\alpha_0), &\quad \gamma_j = F(\gamma'_j) \\ \beta'_k \sim \text{Beta}(1,\gamma_0), &\quad \beta = F(\beta') \end{aligned} .$$

Then, we can get that

$$p(t,k,\gamma',\beta') = \prod_{j=1}^{J}\prod_{n=1}^{n_j}[F(\gamma'_j)]_{t_{ij}} \prod_{t=1}^{\infty} b_{\alpha_0}(\gamma'_{jt}) \prod_{t=1}^{\infty}[F(\beta')]_{k_{jt}} \prod_{k=1}^{\infty} b_{\gamma_0}(\beta'_k)$$

where b_ζ denotes Beta$(1,\zeta)$.

2.3.2 HDP with Labels and Mixture Part

Based on HDP with only labels which we consider above, we now add the mixture part. Consider

$$\begin{aligned} f_k|\mathcal{F} &\sim \mathcal{F}, f = (f_k) \\ y_{ij}|z_{ij},f &\sim f_{z_{ij}}(y_{ij}) \end{aligned} ,$$

we can get

$$\begin{aligned}p(y, f, t, k, \gamma', \beta') &= p(y|t, k, f)p(t, k, \gamma', \beta')p(f) \\ &= \prod_{j=1}^{J}\prod_{i=1}^{n_j} f_{k_{j,t_{ij}}}(y_{ij})\gamma_{j,t_{ij}} \prod_{t=1}^{\infty} b_{\alpha_0}(\gamma'_{jt}) \prod_{t=1}^{\infty} \beta_{k_{jt}} \prod_{k=1}^{\infty} b_{\gamma_0}(\beta'_k)\mathcal{F}(f_k)^{.}\end{aligned}$$

This expression can be used to do slice sampling.

2.3.3 Slice Sampling for HDP

In the joint density we get above, there are two infinite parameters, β and γ. Thus, we introduct u and v for slice sampling to solve this the problem of infinity. Now we have

$$p(y, f, t, k, \gamma', u, \beta', v) = \prod_{j=1}^{J}\prod_{i=1}^{n_j} f_{k_{j,t_{ij}}}(y_{ij})\mathbf{1}(u_{ij} \le \gamma_{j,t_{ij}}) \prod_{t=1}^{\infty} b_{\alpha_0}(\gamma'_{jt}) \prod_{t=1}^{\infty} \mathbf{1}(v_{jt} \le \beta_{k_{jt}}) \prod_{k=1}^{\infty} b_{\gamma_0}(\beta'_k)\mathcal{F}(f_k).$$

Based on this, we can have the following sampling procedure.

Sampling u. This can be done by sampling $(u|\gamma', t, k, \beta', v)$ based on that

$$u_{ij}|\gamma', t, k, \beta', v \sim \text{Unif}(0, \gamma_{j,t_{ij}}).$$

Sampling γ'. We have the sampling distribution

$$\gamma'_{jt}|\gamma'_{-jt}, t, k, \beta' \sim \text{Beta}(n_t(t_j) + 1, n_{>t}(t_j) + \alpha_0)$$

where $n_t(t_j) = |\{i : t_{ij} = t\}|$ and $n_{>t}(t_j) = |\{i : t_{ij} > t\}|$.

Sampling v. We have the sampling distribution

$$v_{jt}|\beta', k, t, \gamma', u \sim \text{Unif}(0, \beta_{k_{jt}}).$$

Sampling β'. We have the sampling distribution

$$\beta'_k|\beta'_{-k}, \gamma', t, k \sim \text{Beta}(n_k(k) + 1, n_{>k}(k) + \gamma_0).$$

Sampling t. We have the sampling distribution

$$p(t_{ij} = t|t_{-ij}, k, \gamma', u, \beta', v) \propto f_{k_{jt}}(y_{ij})\mathbf{1}(u_{ij} < \gamma_{jt}).$$

Then let $T_{ij} := T(\gamma_k; u_{ij}) := \sup\{t : u_{ij} \le \gamma_{jt}\}$. Then we can further get that

$$t_{ij}| \cdots \sim (f_{k_{jt}}(y_{ij}))_{t\in[T_{ij}]}.$$

Sampling k. Similar to sampling t, we can get that

$$p(k_{jt} = k| \cdots) \propto \mathbf{1}(v_{jt} \le \beta_k) \prod_{i:t_{ij}=t} f_k(y_{ij}).$$

Then let $K_{jt} := K(\beta; v_{jt}) := \sup\{k : v_{jt} \le \beta_k\}$, and we can further get that

$$k_{jt}| \cdots \sim (\prod_{t:t_{ij}=t} f_k(y_{ij}))_{k\in[K_{jt}]}.$$

Sampling f. Finally, we sample f according to

$$p(f_k|\cdots) \propto \mathcal{F}(f_k) \prod_{(i,j):z_{ij}=k} f_k(y_{ij}).$$

Based on the logic above, here is our code for slice sampling of HDP.

```r
slice_sampler_HDP <- function(y, beta0=1, alpha0=1, iter_max=500, Kcap=20, Tcap=20,
                              extension_factor=1.5) {
  # input of the function:
  ## y: observations, a list of matrices, each matrix is a group
  ## beta0: the parameter of global DP, beta~GEM(beta0)
  ## alpha0: the parameter of group level DP, gamma~GEM(alpha0)
  ## iter_max: maximum iterations
  ## Kcap: initial length to store k, might be updated if the length isn't enough
  ## Tcap: initial length to store t, might be updated if the length isn't enough
  ## extension_factor: if the length is not enough, extend the length of k/t to
  ## be length * extension_factor

  calculate_prod <- function(prod_list){
    # this is a function to calculate product with normalization
    log_sums <- sapply(prod_list, function(x) sum(log(x)))
    sapply(log_sums, function(x) exp(x-max(log_sums)))
  }

  find_turning_pos <- function(beta, threshold) {
    # if the saved length is not enough, we return the position
    # this is used to calculate the position we need to get to
    tmp <- which(c(cumsum(beta),1) > 1-threshold)[1]
    tmp
  }

  sample_from_pmf <- function(n, pmf){
    # this function is used to sample from a given pmf
    sample(x = seq(1,length(pmf)), n, replace = T, prob=pmf)
  }

  stick_break_func <- function(x) {
    # this is used to calculate the stick breaking expression
    temp <- c(1,cumprod(1-x))
    temp[1:length(x)] * x
  }

  count_beta <- function(z, K) {
    # output[1, j]: how many labels == j are in z
    # output[2, j]: how many labels > j are in z
    zcounts <- tabulate(z, K)
    rbind(zcounts, c(rev(cumsum(rev(zcounts)))[-1], 0))
  }

  update_phi <- function(y, z, K, prec2_y = 1, prec2_phi = 1) {
    # this is a function to update phi
    z_flat <- unlist(z)
    y_flat <- do.call(rbind,y)
    z_freq <- tabulate(z_flat,K)
    yz_df <- data.frame(y_flat, z=z_flat)
    temp <- which(z_freq == 0)

    yz_df <- rbind(data.frame(x1=0, z=temp), yz_df)

    phi_mean <- aggregate(.~z, yz_df,
                          function(x) sum(x)*(prec2_y/(prec2_phi
                                                       + length(x)*prec2_y)) )
    phi_mean <- as.matrix(phi_mean[, -1])
    prec2_post <- prec2_phi + z_freq*prec2_y

    phi_new <- do.call(rbind, lapply(1:K,
                                     function(k)
                                       rnorm(1, mean = phi_mean[k,],
                                             sd = 1/sqrt(prec2_post[k])) ) )

    list(phi=phi_new, prec2=prec2_post)
  }

  Ker <- function(y, phi, prec2_y=1) {
    # kernel function, we use a Normal kernel, N(phi, 1/prec2_y)
    dnorm(y, mean=phi, sd=1/sqrt(prec2_y))
  }

  # number of groups: J
  J <- length(y)
  # how many observations in each group, n
  n <- sapply(y, function(x) dim(x)[1])
  # the vector which stores the maximum length of t of group j
  Tjcap <- rep(Tcap,J)

  # t, k, z, u, v
  tb <- lapply(1:J, function(j) rep(1,n[j]))
  kb <- lapply(1:J, function(j) rep(1,Tjcap[j]))
  z  <- lapply(n,   function(nj) rep(1,nj))
  u <- lapply(1:J, function(j) runif(n[j]))
  v <- lapply(1:J, function(j) runif(Tjcap[j]))

  # store the old k, t, u, v, z
  kb_old <- kb
  tb_old <- tb
  u_old <- u
  v_old <- v
  z_old <- z

  # lists used to store all sampled z, t, k
  z_hist <- list()
  t_hist <- list()
  k_hist <- list()
  T_hist <- list()
  K_hist <- list()

  # indicator of iteration
```

```r
itr <- 1

while (itr < iter_max) {
  gamp <- list()
  gam <- list()
  T_all <- list()
  Tv <- rep(0,J)
  Tj_overflow <- F

  for (j in 1:J) {
    # 1. update gamma
    g_counts <- count_beta( tb[[j]], Tjcap[j] )
    gamp[[j]] <- rbeta( Tjcap[j], 1 + g_counts[1,], alpha0 + g_counts[2,] )
    gam[[j]] <- stick_break_func(gamp[[j]])
    T_all[[j]] <- sapply( 1:n[j], function(i) find_turning_pos(gam[[j]], u[[j]][i]) )
    Tv[j] <- max(T_all[[j]])

    if ( Tv[j] > Tjcap[j] ) {
      Tj_overflow <- T
      Tjcap_old <- Tjcap[j]
      Tjcap[j] <- round( extension_factor*Tjcap[j] )
      v_old[[j]] <- c( v_old[[j]], runif(Tjcap[j]-Tjcap_old) )
      break
    }
  }
  if (Tj_overflow)  {
    kb <- kb_old
    tb <- tb_old
    u <- u_old
    v <- v_old
    z <- z_old
    next
  }


  k_counts <- count_beta( unlist(kb), Kcap )
  betap <- rbeta(Kcap, 1 + k_counts[1,], beta0 + k_counts[2,])
  beta <- stick_break_func( betap )
  K_all <- list()
  Kv <- rep(0,J)
  for (j in 1:J) {
    K_all[[j]] <- sapply( 1:Tjcap[j], function(t) find_turning_pos(beta, v[[j]][t]) )
    Kv[j] <- max( K_all[[j]] )
  }
  K <- max(Kv)
  if (K > Kcap)  {
    Kcap <- round(extension_factor*Kcap)
    kb <- kb_old
    tb <- tb_old
    u <- u_old
    v <- v_old
    z <- z_old
    next
  }

  # restore the current state as the old one
  kb_old <- kb
  tb_old <- tb
  u_old <- u
  v_old <- v
  z_old <- z

  phi_vec <- update_phi(y, z, Kcap)$phi
  f_vec <- sapply(1:Kcap, function(k) { function(y) Ker(y, phi_vec[k,]) } )

  for (j in 1:J) {

    # 2. update k
    for (t in 1:Tjcap[j]) {
      prod_list <- lapply( 1:K_all[[j]][t],
                           function(k) f_vec[[k]]( y[[j]][tb[[j]] == t, ]))
      prob_vec <- calculate_prod(prod_list)
      kb[[j]][t] <- sample_from_pmf( 1, prob_vec)
    }

    # 3. update t
    for (i in 1:n[j]){
      prob_vec <- sapply(1:T_all[[j]][i],
                         function(t) f_vec[[  kb[[j]][t] ]](y[[j]][i,]))
      tb[[j]][i] <- sample_from_pmf( 1, prob_vec)
    }

    # 4. update u
    u_upper <- sapply(seq(1,n[j]), function(i) gam[[j]][ tb[[j]][i] ])
    u[[j]] <- runif(n[j], 0, u_upper)

    # 5. update v
    v_upper <- sapply(seq(1,Tjcap[j]), function(t) beta[ kb[[j]][t] ])
    v[[j]] <- runif(Tjcap[j], 0, v_upper)

    # 6. update z
    z[[j]] <-  sapply(1:n[j], function(i) kb[[j]][ tb[[j]][i] ] )

  }

  itr <- itr + 1
  if (itr %% 300 == 0) {
    cat(sprintf("%6d: ",itr),'Finish\n')
  }

  z_hist[[itr]] <- z
  t_hist[[itr]] <- tb
  k_hist[[itr]] <- kb
  T_hist[[itr]] <- T_all
  K_hist[[itr]] <- K_all
  }

  # return the stored z, t, k
  list(z=z_hist, t=t_hist, k=k_hist, T_all=T_hist, K_all=K_hist)
}
```

We use the provided data to test the performance.

```
data <- read.csv('final_data.csv')

y <- list(immuno=matrix(data[data$group=='immuno', 'logArea']), cryo=matrix(data[data$group=='cryo', 'l
colnames(y[[1]]) <- 'x1'
colnames(y[[2]]) <- 'x1'

iter_max <- 1500
result <- slice_sampler_HDP(y, iter_max=iter_max)
```

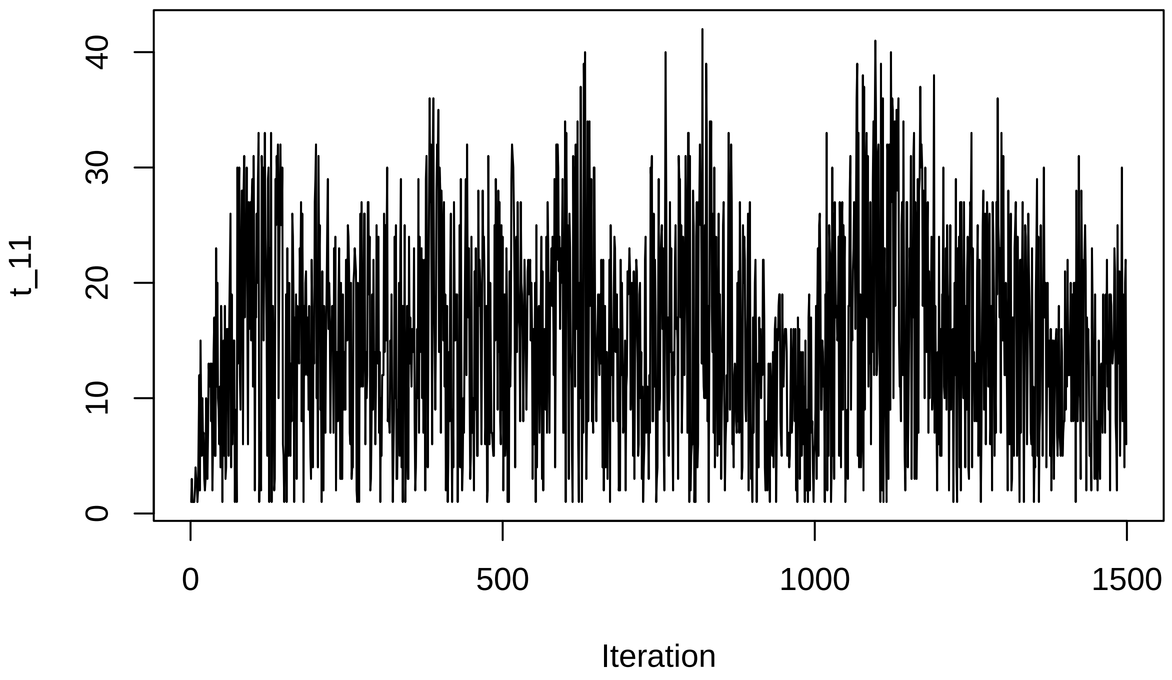
```
##    300:  Finish
##    600:  Finish
##    900:  Finish
##   1200:  Finish
##   1500:  Finish
```

We run 1500‘ iterations. To test the convergence, we provide some trace plots.

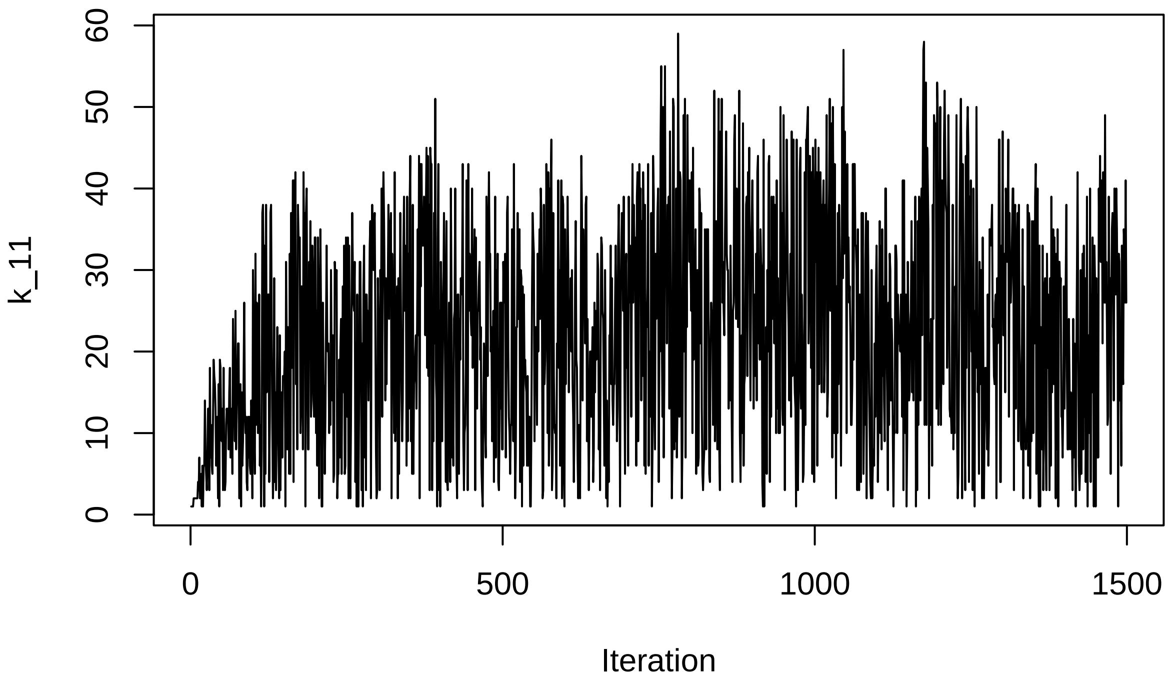
```
t_trace <- c()
for (i in 2:(iter_max)) {
  t_trace <- append(t_trace, result$t[[i]][[1]][1])
}

k_trace <- c()
for (i in 2:(iter_max)) {
  k_trace <- append(k_trace, result$k[[i]][[1]][1])
}

z_trace <- c()
for (i in 2:(iter_max)) {
  z_trace <- append(z_trace, result$z[[i]][[1]][1])
}

T_trace <- c()
for (i in 2:(iter_max)) {
  T_trace <- append(T_trace, result$T_all[[i]][[1]][1])
}

K_trace <- c()
for (i in 2:(iter_max)) {
  K_trace <- append(K_trace, result$K_all[[i]][[1]][1])
}

cluster_number <- c()
for (i in 2:iter_max) {
  tmp <- unique(c(result$z[[i]][[1]], result$z[[i]][[2]]))
  cluster_number <- append(cluster_number, length(tmp))
}
```

The first three trace plots are the samples of t_{11}, k_{11} and z_{11}.

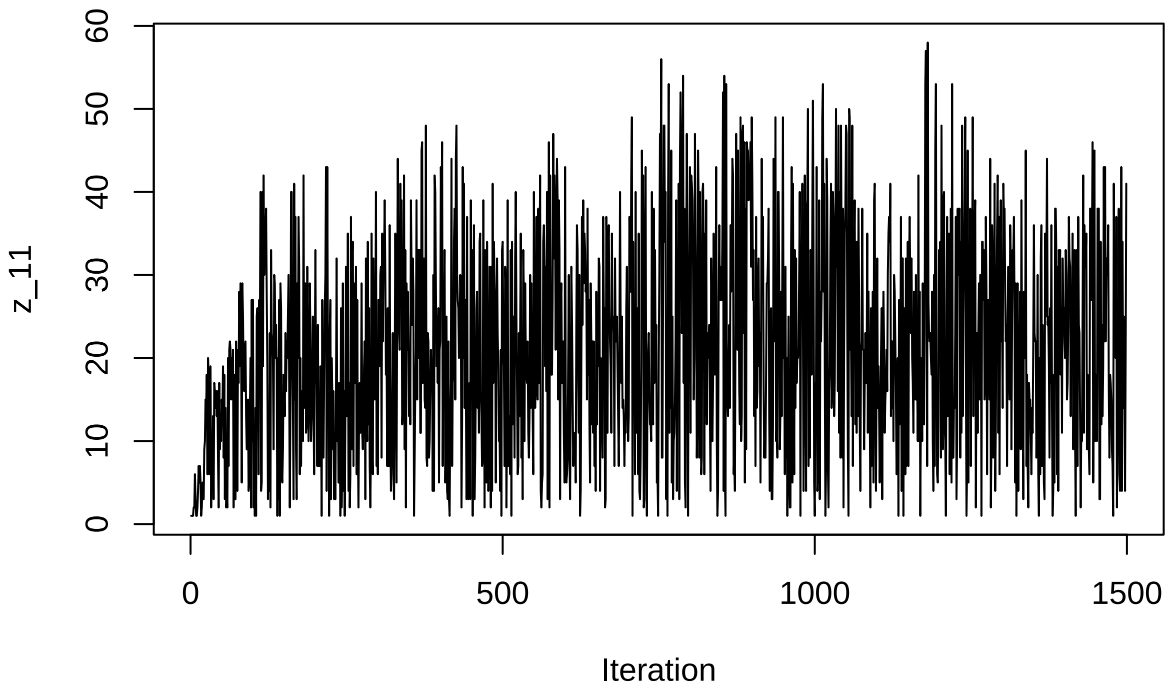
```
plot(1:(iter_max-1), t_trace, 'l', xlab='Iteration', ylab='t_11')
```



```
plot(1:(iter_max-1), k_trace, 'l', xlab='Iteration', ylab='k_11')
```

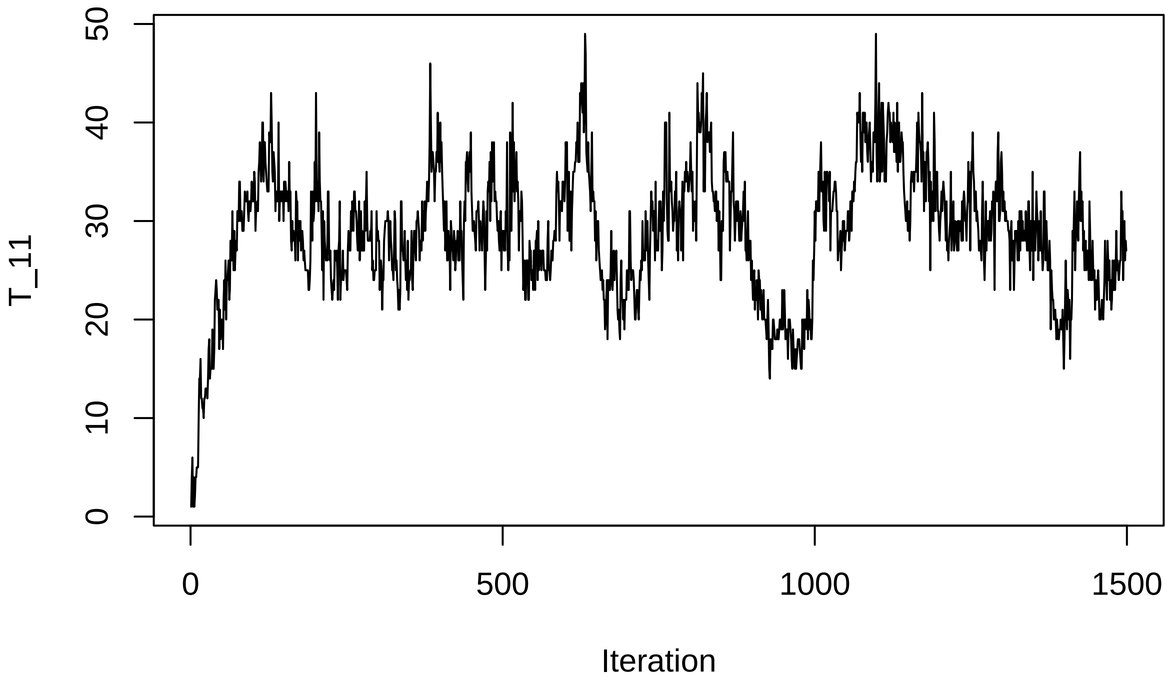


```
plot(1:(iter_max-1), z_trace, 'l', xlab='Iteration', ylab='z_11')
```

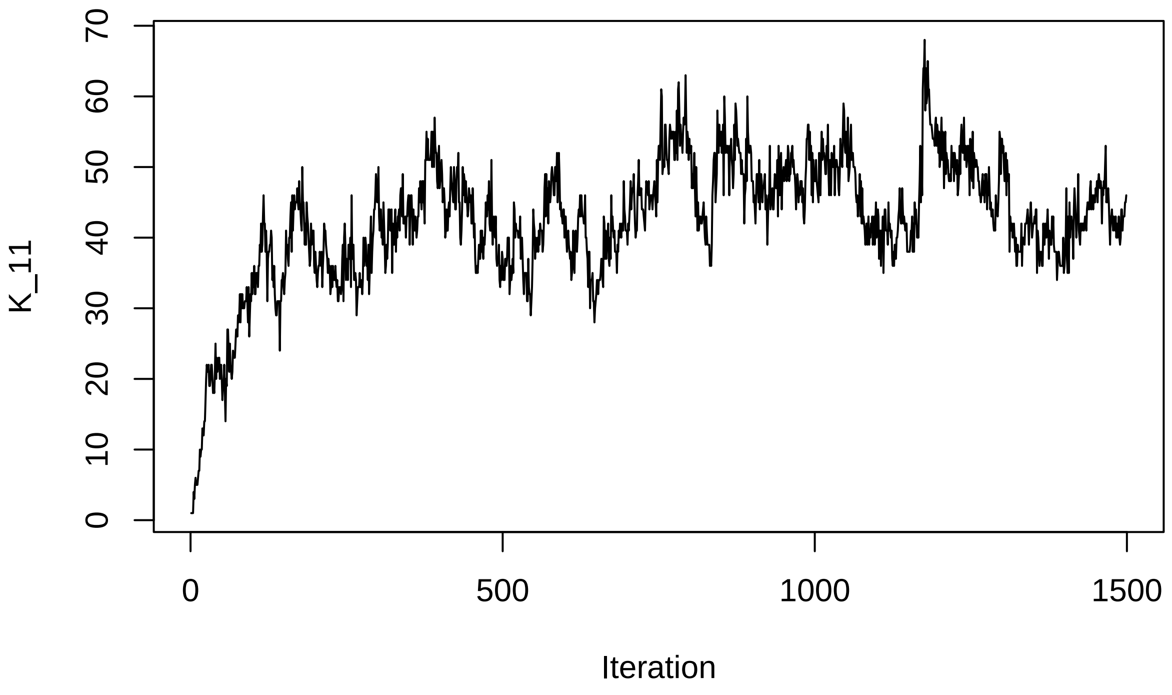


The coming two trace plots are T_{11} and K_{11}.

```
plot(1:(iter_max-1), T_trace, 'l', xlab='Iteration', ylab='T_11')
```

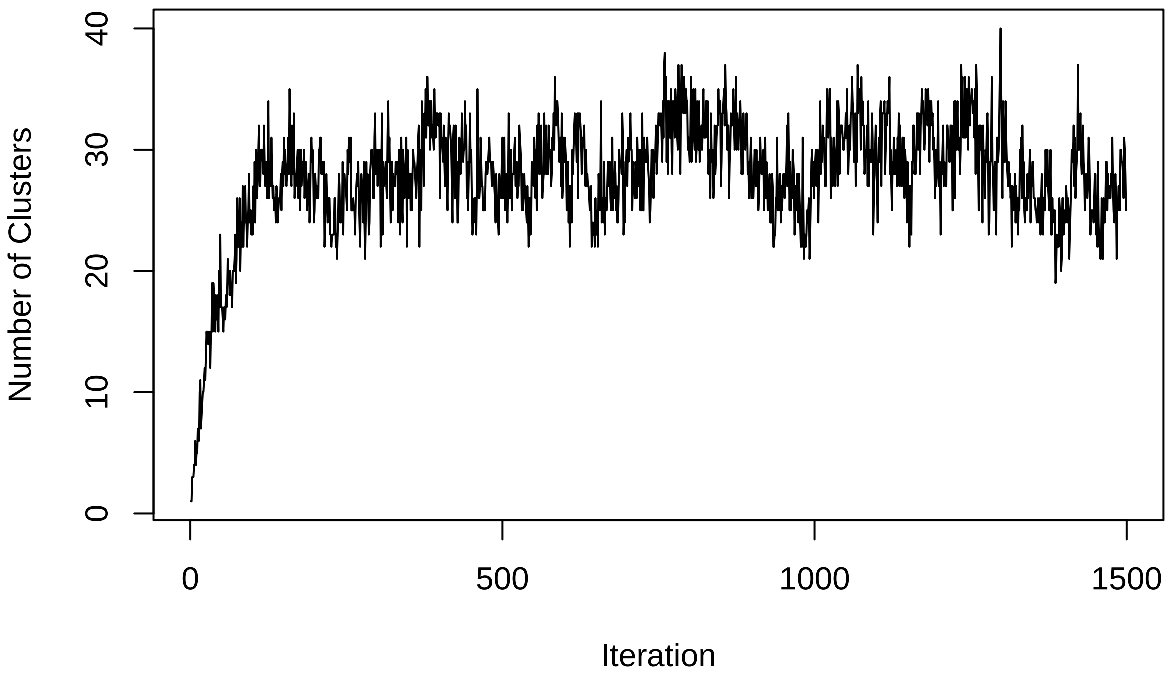


```
plot(1:(iter_max-1), K_trace, 'l', xlab='Iteration', ylab='K_11')
```



The last trace plot is the number of clusters.

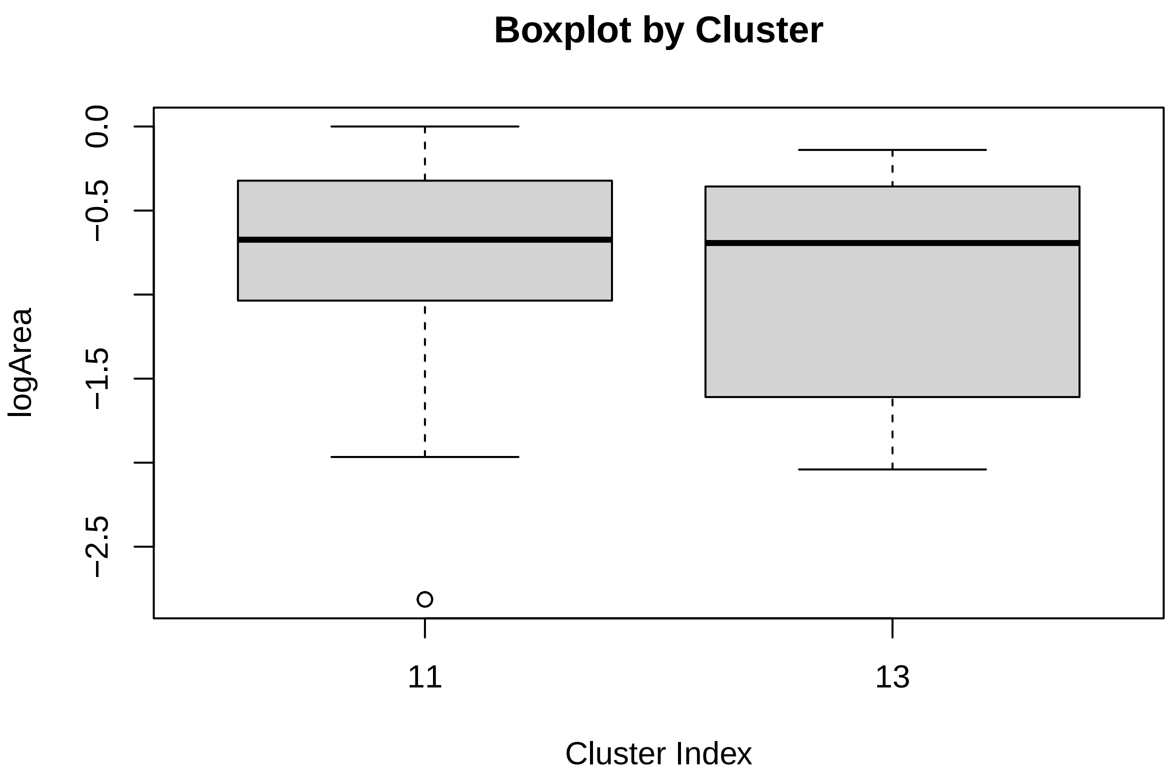
```r
plot(1:(iter_max-1), cluster_number, 'l', xlab='Iteration',
     ylab='Number of Clusters')
```



From the trace plots, it seems that the convergence tends to be reached. The number of clusters tend to be in the range of $[25, 35]$.

Here are some examples of points from several clusters using the samples from the last iteration. We use the box-plots here, as it can help us to have a clear look at the distribution of these points.

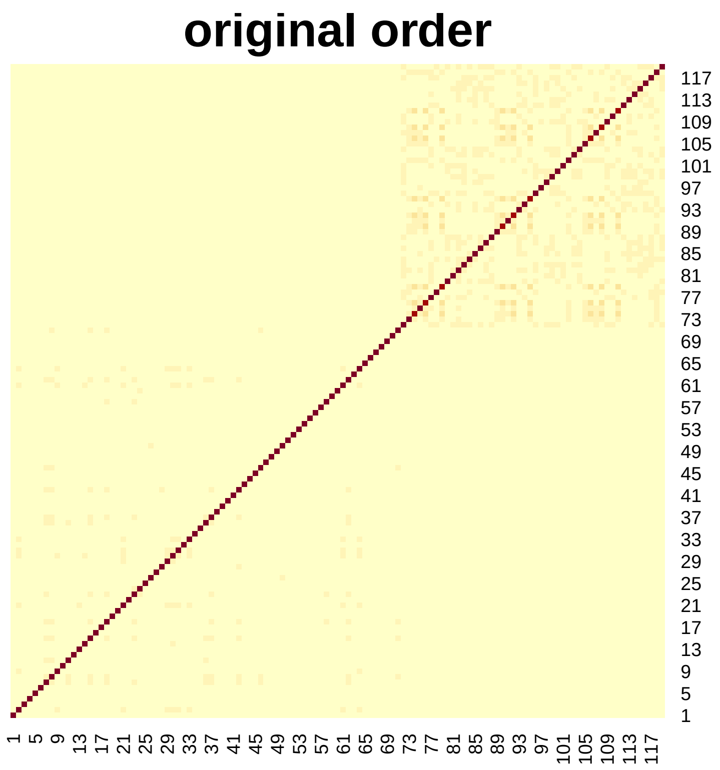
```r
# sample distribution: last iteration
data$index <- 0
data[data$group == 'immuno', 'index'] <- result$z[[iter_max]][[1]]
data[data$group == 'cryo', 'index'] <- result$z[[iter_max]][[2]]
unique_index <- unique(data$index)

group_data <- split(data$logArea, data$index)
names(group_data) <- c(1:length(unique_index))
group_data <- group_data[lengths(group_data) >= 8]
boxplot(group_data,
        xlab = "Cluster Index", ylab = "logArea",
        main = "Boxplot by Cluster")
```



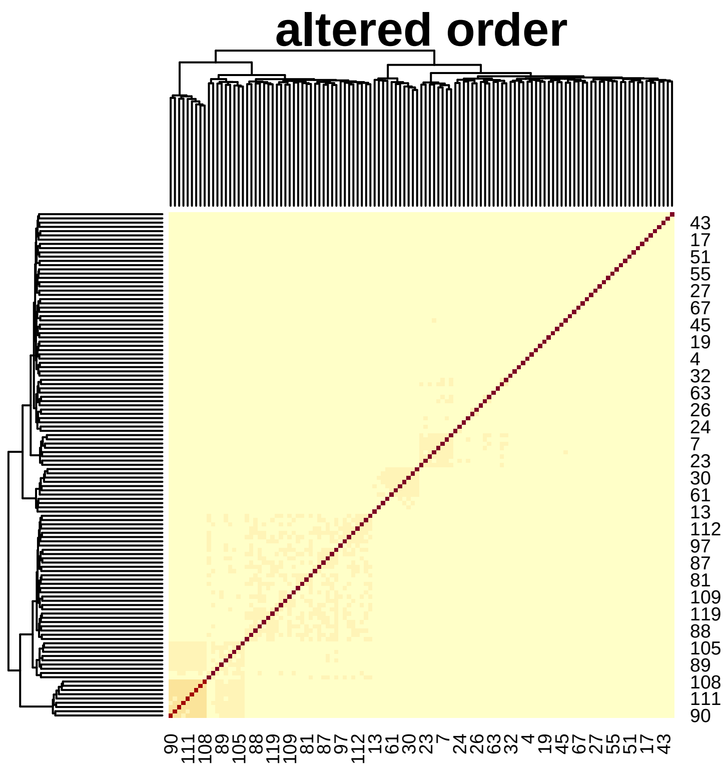
From the box-plots, we can see that there tend to be some differences between clusters.

Finally, we provide some heat maps to view the cluster structure.

```r
all_i.1 <- sum(data$group == "immuno")
all_i.2 <- sum(data$group == "cryo")
all_i = dim(data)[1]

simu_mat <- matrix(0, all_i, all_i)

for (i in 2:iter_max) {
  z <- c(result$z[[i]][[1]], result$z[[i]][[2]])
  for (j in 1:all_i) {
    for (k in 1:all_i){
      simu_mat[j, k] = simu_mat[j, k] + (z[j] == z[k])
    }
  }
}
simu_mat <- simu_mat / iter_max

heatmap(simu_mat, Rowv = NA, Colv = NA, main="original order")
```



```
heatmap(simu_mat, main="altered order")
```



2.4 Research 3: Slice Sampler vs Chinese Restaurant Franchise Sampling Method

The differences between slice sampler and CRF sampling are obvious. Although there might be some minor mistakes with our coding procedure which we fail to find due to limited time, the much difference between patterns can at least show some information.

Firstly, slice sampler converges greatly faster than CRF. From the trace plots, we can know that slice sampler tend to converge after only 300 iterations, but CRF tend to converge after about 1000 iterations, and the later pattern is also not as stable as slice sampler. This shows that slice sampler is more efficient than CRF considering computation cost.

Secondly, slice sampler tends to assign individuals in greatly more clusters than CRF. In slice sampler, when the algorithm converge, we find about 25 to 35 clusters. However, CRF tends to find less than 10 clusters. Too many clusters mean that the cluster membership of each individual is not very constant. From the heat maps of slice sampler, although we observe some block with darker color on the diagonal, they are quite scarce. For CRF, although it finds greatly less clusters than slice sampler, the clustering structure is somehow more stable. Individuals who belong to the same cluster in one iteration also tends to belong to the same cluster in the next iteration.

Besides the differences, both sampling methods can at least identify some heterogeneity between different clusters shown by box plots. These clusters have different distributions, and always with at least one which tends to be evenly distributed, and one with some outliers.