Inference for Non Conjugate Dirichlet Process Mixture Models

Siddhant Tandon[1,*]

[1]Matricola : 1771650
[*]siddhant.tandon203@gmail.com

ABSTRACT

In this project I study the algorithms for carrying out inference under non conjugate Dirichlet Process Mixture Models. The implication of the algorithms is Clustering effect under a completely nonparameteric bayesian regime. In particular I implement Algorithm 7[1] of Radford Neal.

Introduction

In a typical parametric bayesian setup, we have a collection of random variables y_1,.....y_n where each of the y_i itself could be a vector of variables or data collected during the experiment. The parametric thing about this data is that it is generated from an underlying distribution G and during inference we always assume that this distribution G is given which is quite inappropriate in some scenarios. To inject uncertainty in the model and expand our belief distribution, one goes completely nonparametric. By this it means that the parameter space which was earlier fixed now becomes infinite. And the prior in such models becomes a stochastic process rather than a probability distribution as in the case of parametric models. This prior is called the Dirichlet Process.

One of the aim in Nonparametric Bayesian Data analysis is continuous density estimation. If one wants to estimate the unknown density of data y_1.....$y_n \sim f$ Dirichlet Process becomes unfit as the realizations from DP are discrete. This problem can be solved by convolving continuous kernel densities. In very simple words it means in a typical parametric bayesian setup, the prior is replaced by Dirichlet Process. Such models are called Dirichlet Process Mixture Models.

Dirichlet Process

Assume there is a space of infinite distributions and one needs to put a prior on this space of distributions. Drawing a sample from this prior essentially means drawing one distribution. Every time a sample is drawn a different distribution is obtained. Let us denote this distribution as $G \sim DP(M, G_O)$. G_O here can be thought of as a prior guess or prior belief for G and M determines how strong your belief is.

A formal definition given by Ferguson[2] is as follows: Let $M > 0$ and G_O be a probability measure defined on S. A DP with parameters M , G_O is a random probability measure G defined on S which assigns probability $G(B)$ to every (measurable) set B such that for each (measurable) finite partition $(B_1.....B_k)$ of S, the joint distribution of the vector $(G(B_1).....G(B_k))$ is a Dirichlet distribution with parameters $(MG_O(B_1).....MG_O(B_k))$.

Computational Methods of representing Dirichlet Process

A Dirichlet Process can be simulated in 3 ways namely the Stick Breaking construction, Ferguson's Definition and the Polya Urn scheme (or computationally known as Chinese Restaurant Process).

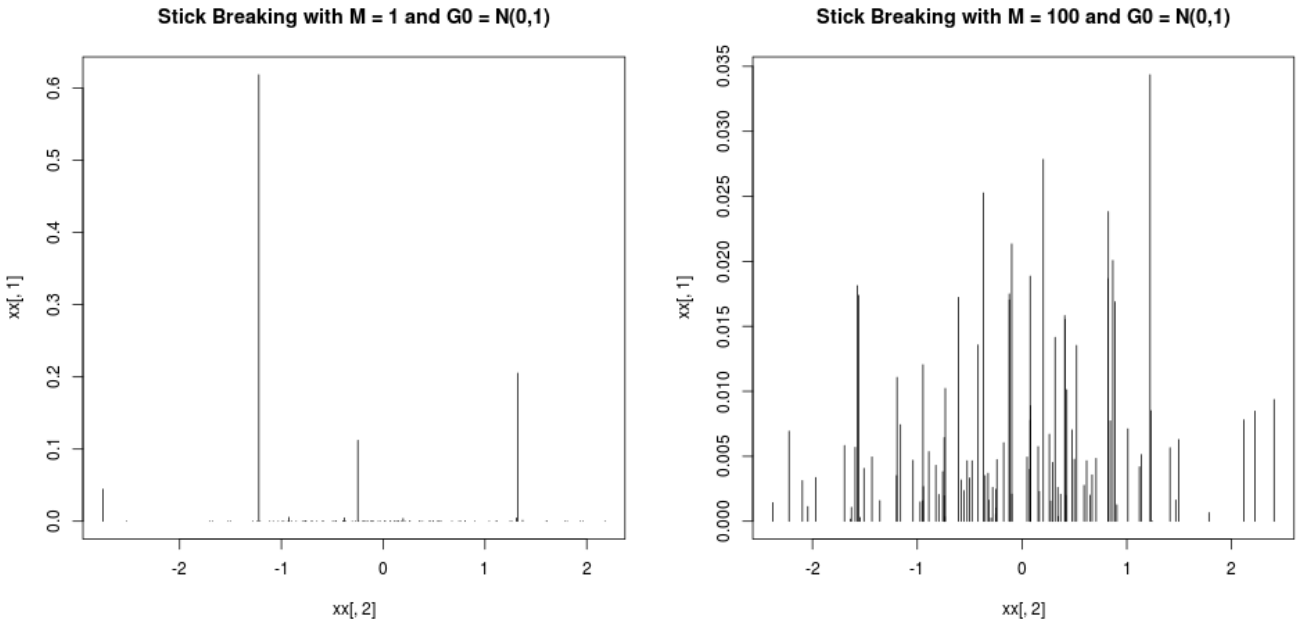
The Stick Breaking Construction

Sethuraman gave a constructive recipe[3] for simulating the Dirichlet Process which is as follows :

1. Start with a stick of unite length 1.
2. Draw a random variable $\beta_1 \sim beta(1,\alpha)$. This will be a real number between 0 and 1 with expected value $1/(1+\alpha)$. Break off the stick at length β_1 from the left to obtain a weight w_1.
3. Take the remaining part of the stick (1 - β_1) and again draw a value β_2 from $beta(1,\alpha)$. Break off the stick from the left again to obtain a weight w_2 equal to (1 - β_1)*β_2.
4. And so on.

The above vector of weights can be thought of as the weights that will be placed on measurable partitions $(B_1.....B_k)$. To obtain these measurable partitions or in other words random points on real line, draw values $\theta_1....\theta_n \sim G_O$ which is your prior guess. And then just place the above weights on these points.

Figure 1 shows the stick breaking process for a small & large value of M.



(a) Figure the stick breaking process for $N(0,1)$ as base distribution and $M = 1$. Notice that a small value of M puts high weights on only few selected points.

(b) Figure the stick breaking process for $N(0,1)$ as base distribution and $M = 100$. Notice that a large value of M disperses the weight quite fairly among the points.

Figure 1. Stick Breaking Process for $M = 1$ and $M = 100$

Polya Urn Scheme

Consider there is an urn containing balls of different colors say A, B and C. According to some probability , you draw a ball from the urn observe the color and add another ball of the same color back to urn or you can add a ball of some new color D. Mathematically this can be expressed as :

$$p(y_i|y_1.....y_n) = \sum_{h=1}^{i-1} \delta_{y_h}(y_i)/(M+i-1) + MG_O(y_i)/(M+i-1) \text{ for } i = 2,3,.... \text{ and } y_1 \sim G_O.$$

The above expression is marginal distribution of the realizations $y_1.....y_n$ from the Dirichlet Process i.e. the Dirichlet Process distribution G has been marginalized out. The above expression also tells the posterior predictive distribution for a future observation y_{n+1}.

All of the above translates into something called Chinese Restaurant Process which is almost equivalent to Polya Urn scheme and makes things easier for programmers. Imagine there is a restaurant and customers keep walking in one by one. Every customer has an option to sit on a new table or sit on a table occupied by previous customers sitting on it. Every table has different cuisines served on it. So if 10 customers had to be seated there could be a possibility of 5 customers on table A 5 on table B, or 3 on table A 3 on table B etc. The Polya Urn equation can give the probability of one such permutation of seating of customers. Similarly for any kind of partition such a probability can be found out. The algorithm for simulating the CRP is as follows:

1.The very first person sits at any table (but doesn't order the food yet).
2. The second person can sit on a new table with probability $M/(M+1)$ and with probability $1/(M+1)$ on the existing table where 1st person is seated.
3. The n^{th} person will have the option to sit on a new table with probability $M/(M+n)$ and choose existing some existing table (among previous k tables) with probability $n^k/(n+M)$ where n^k are number of people sitting at the k^{th} table already.

There are some points to be noted about CRP and Polya Urn scheme. Both of them assign the table probabilities in the

same way however the table parameters are not drawn in the CRP unlike the Polya Scheme. One can go from CRP to Polya by first assigning the tables or indicators and then drawing parameters for the tables conditioned on the indicators.

$$\phi_1, \phi_2 ..... \phi_k \sim G_O.$$
$$c_1, c_2 ..... c_n \sim CRP(M, n).$$
where k is the k_{th} distinct group

Dirichlet Process Mixtures

Realizations from the DP are discrete making it unfit for continuous density estimation. In such cases, a continuous kernel density is put into the picture by having one layer of gaussian distribution conditioned on samples from the DP. Mathematically,

$$y_i|\theta_i \sim f_{\theta_i}$$
$$\theta_1, \theta_2 ..... \theta_n \sim G.$$
$$G \sim DP(M, G_O).$$
where f_{θ_i} can be a Gaussian centered at θ_i

When the goal is to estimate the density, the posterior distribution of G conditioned on y_i with the θ_i integrated out. This result was shown by Antoniak[4].

$$G|y_1, y_2 .... y_n \sim \int DP(MG_O + \sum_{i=1}^{n} \delta_{\theta_i}) dp(\theta_1, \theta_2 ... \theta_n | y_1 ... y_n)$$
where $\mathbf{y}$ and θ are the same quantities mentioned in the DPM model.

Clustering effect under DPM

When values are drawn from DP, we are actually drawing the locations on the real line with probability equal to their weights. Because these are discrete locations it is very much possible that a same point can be drawn multiple number of times. As a result out of n values drawn, k of them will be unique where $k <= n$. This property induces clustering effect which makes DPM very useful to cluster your data. One big advantage is that there is no assumption of number of clusters beforehand and the process itself decides the number of clusters in the end.

Let's introduce some notations. Let $\theta_j^* j = 1 ..... k$ where k are the number of unique values among $\theta_1 ..... \theta_n$. Let $s_j = \left\{ i : \theta_i = \theta_j^* \right\}$ that is the set of indicators tied to j^{th} θ. The number of members in the j^{th} cluster can be denoted by n_j.

Derivation of sampling probabilities

Let's go back to the Polya Urn equation.

$$p(\theta_i|\theta_{-i}) = \sum_{j=1}^{k^-} n_{j^-} \delta_{\theta_{j^{*-}}} + MG_O(\theta_i)$$
where θ_{-i} indicates all θ except θ_i, and k^- indicates number of unique values among θ_{-i}, n_{j-} are the number of members in the cluster j.

When the data or $y_i|\theta_i \sim f_{\theta_i}$ is injected into the picture, multiply the above equation with the sampling distribution $f_{\theta_i}(y_i)$ to obtain the posterior of θ_i.

$$p(\theta_i|\theta_{-i}, y) \propto \sum_{j=1}^{k^-} n_{j^-} f_{\theta_{j^{*-}}}(y_i) \delta_{\theta_{j^{*-}}} + M f_{\theta_i}(y_i) G_O(\theta_i)$$

The second term $f_{\theta_i}(y_i)G_O(\theta_i)$ is without the normalizing constant. Define $H_i = \frac{f(y_i|\theta_i)G_O(\theta_i)}{\int f(y_i|\theta_i)G_O(\theta_i)d\theta_i}$. The denominator that is the marginal of y_i causes most of the problems. If G_0 is not conjugate with $f(y_i, \theta_i)$, the integral cannot be found out analytically. The first three algorithms[1] in Neal's paper relies on the conjugacy of base distribution and sampling distribution. The very first solution[5] to tackle non conjugate base distribution and sampling distribution was given by Maceachern & Muller which was called "no gaps" algorithm. Their solution introduced indicators s_i into the sampling probabilities by augmenting the existing DPM model. They represented θ in terms of (ϕ, s). In terms of CRP representation it means the values drawn from DP were expressed in form of table indicators and table parameters. This algorithm is also reviewed by Neal and presented as algorithm 4[1] in his paper.

Usage of MCMC methods

As far as I understand, even though the Metropolis-Hastings algorithm was written in 1970 nobody tried to handle non conjugate priors using MCMC. This seems a bit unusual to me but anyways Radford Neal presented four algorithms which were based on MCMC methods.

I will discuss the Algorithm 7 by Neal. Let me just bring back the analogy of CRP. Right now the scenario is you have people walking into the restaurant and they will eventually be clustered at their favorite cuisine tables. This is like, in the very beginning all the data points are randomly assigned to clusters. Then the i^{th} data point is unassigned from it's cluster and the closest cluster to this datapoint if found out by calculating the probability density of this point against every cluster parameter ϕ. The way these MCMC algorithms work is in such a way that clusters with lot of datapoints get richer and this is evident in the Polya Urn equation as this change is proportional to number of cluster members n_j at the j^{th} table. Also another point is that how many times the algorithm gives a chance to create a new cluster. Of course this change is proportional to α or M but the question how many times the algorithm should let this change happen.

In algorithm 7, the creation of a new component is given a chance many number of times. The algorithm creates a new component whenever a cluster id is encountered which is not a singleton. If you have a markov chain of indicators as $(1,1,3,5,5,3,2,6,3,3)$ then out of 10 times , 8 times the if condition to create a new cluster component is executed. Of course then it depends on the acceptance probability if the new component is accepted in the markov chain or not. Then if a singleton cluster id is encountered it is alloted to the richest cluster.

Implementation

Using the Chinese Restaurant Process, I generated the cluster ids or table indicators. For every table (or cluster) the parameters were sampled from a very wide gaussian $N(125,75)$. The data generated was a total of 200 observations from gaussians centered far apart from each other. There were 80 observations from $N(0,1)$, 40 from $N(70,1)$, 40 from $N(140,1)$ and 40 from $N(190,1)$. The reason I chose the base distribution with large variance because if I had to choose a very narrow gaussian, then it would have decreased the chances of accepting new cluster ids in our markov chain. What I mean is , if I have a base distribution as $N(0,1)$ most of the ϕ sampled from it will be close to 0. In the first step of the algorithm when the expression $f(y_i,\phi_i)$ will be computed for the acceptance of new cluster, the value of this expression will be zero for most of the data points. Because 160 observations come from a gaussian which is far from mean 0.

After sampling the indicators the next step is updating the cluster parameters or ϕ for every table which can be updated according to the following equation :

$$p(\phi_{j^*}|s,y) \propto G_O(\phi_{j^*}) \prod_{i \varepsilon s_j} f(y_i,\phi_{j^*}).$$

where $i\varepsilon s_j$ are set of all the indices which are members of j^{th} cluster.

In simple words this is just the multiplication of likelihood of those data points which are members of the present table, with the base distribution. For the sake of simplicity I kept the base distribution and sampling distribution both to be normal distributions so that at this step I could exploit the conjugacy property. However in the case of non conjugate model, this step could be replaced by another MCMC update.

Following are functions which implement the CRP and the algorithm 7.

```
#function to simulate crp
crp = function(num_customers, alpha){
  table = c(1)
  for(i in 2:num_customers){
    size.table = length(table)
    existing_table = table[1:length(table)]/(alpha + i)
    new_table = alpha/(alpha+i)
    prob = c(existing_table,new_table)
    val = decide(table,prob)
    if (val > length(table)){
      table[val] = 1
    }
```

```
    else{
      table[val] = table[val] + 1
    }
  }
  return(list(val,prob,table))
}
decide = function(table,prob_vector){
  return(sample(1:(length(table)+1),prob = prob_vector,size = 1,replace = T))    #0 existing
}

#function to sample indicators and table parameters according to Algorithm 7 Neal
myfunc = function(vec,y,phi_vect,M){

  phi_G0 = c()
  new_ind = c()
  for (i in 1:length(vec)) {

    fr = sum(vec[i]==vec)
    if (fr > 1) {                    #not a singleton
      phi.star = rnorm(1,mean = 125,sd = 100)
      nmr = dnorm(y[i],phi.star,1,log = T)
      dnr = dnorm(y[i],phi_vect[i],1,log = T)
      expr = (M/(length(y)-1))*exp(nmr-dnr)
      if(expr > runif(1)){     # acceptance probability
        vec[i] = max(vec) + 1 #new table
        phi_vect[i] = phi.star
        new_ind = c(new_ind,vec[i]) #new table indicators
        phi_G0 = c(phi_G0,phi.star) #all phi drawn from G0
      }
    }

    if (fr ==1) {                  # its a singleton

      draw_from = sort(unique(vec[-i]))
      n_ic = table(vec[-i])        #frequency of clusters
      pr = n_ic/(length(y)-1)
      ci.star = sample(draw_from,1,prob = pr)
      idx = which(ci.star==vec)[1]
      phi.star = phi_vect[idx]
      nmr = dnorm(y[i],phi.star,1,log = T)
      dnr = dnorm(y[i],phi_vect[i],1,log = T)
      expr = ((length(y)-1)/M)*exp(nmr-dnr)
      if(expr > runif(1)){      #acceptance probability
        vec[i]=ci.star
        phi_vect[i]=phi.star
      }
    }

  }

  freq = table(vec)
  sing_idx = which(freq==1)
  ci = sort(unique(vec))
  idx = which(vec%in%ci[-sing_idx]) #all ids which are not singleton
  for (i in idx) {
```

```
    n_ic = table(vec[-i])
    draw_from = sort(unique(vec[-i]))
    p = n_ic / (length(y)-1)
    #get corresponding phi_ci
    phi_ci = c()
    for (id in draw_from) {
      phi_id = which(id==vec)[1]
      phi_ci=c(phi_ci,phi_vect[phi_id])
    }
    p = p*dnorm(y[i],mean = phi_ci,1)
    dr = sample(draw_from,1,prob = p)
    vec[i] = dr
    phi_vect[i]=phi_ci[which(dr==draw_from)[1]]

  }
  #update phi or sample table parameters

  freq <- table(vec)
  vec.c <- sort(unique(vec))
  for(i in 1:length(vec.c)){
    idx <- which(vec==vec.c[i])
    y.tmp <- y[idx]
    post_u =  (sum(y.tmp)+(125/(100*2))) / ( (1/(100*2)) + length(y.tmp)) #updated posterior
    post_sd = 1 /((1/(100*2)) + lengths(y.tmp))
    #A <- length(y.tmp)/1
    #b <- sum(y.tmp)
    phi_vect[idx] <- rnorm(1,post_u,post_sd)

  }
  return(list(vec,phi_vect))

}
```

Results

At the very beginning the indicators were randomly assigned. This is the markov chain of indicators at 0^{th} iteration :

```
> table(mcmc_ci[,1])

 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
23  9 14 10  9 30  2 10  4  2  7  4  2  4 14  5  2  2 13  4  4  7  2  2  4  1  3  1  1  1  2  1  1
> table(mcmc_phi[,1])

  0  10  20  30  40  50  60  70  80  90 100 110 120 130 140 150 160 170 180 190 200 210 220 230
 23   9  14  10   9  30   2  10   4   2   7   4   2   4  14   5   2   2  13   4   4   7   2   2
240 250 260 270 280 290 300 310 320
  4   1   3   1   1   1   2   1   1
>
```

(a) Figure showing the state of markov chain of indicators and parameters at 0th iteration.

```
> table(mcmc_ci[,100])

 58  68 112 116 129 134 135 139 140 141 142 143 144
 30   7   5  14  11  15  28   7  43   2  26   7   5
> table(mcmc_phi[,100])

 -1.05981592783578 -0.732721174037542  -0.49417781549312  -0.34460801394229   69.3725189045867
                30                  2                 43                  5                 26
  69.6441497519456    140.56802584242   141.200097662828   141.24019953346   141.866668689183
                14                 15                  7                  7                 11
  190.492576117108   191.679943417255   191.684217185807
                28                  5                  7
>
```

(b) Figure showing the state of markov chain of indicators and parameters at 100th iteration.

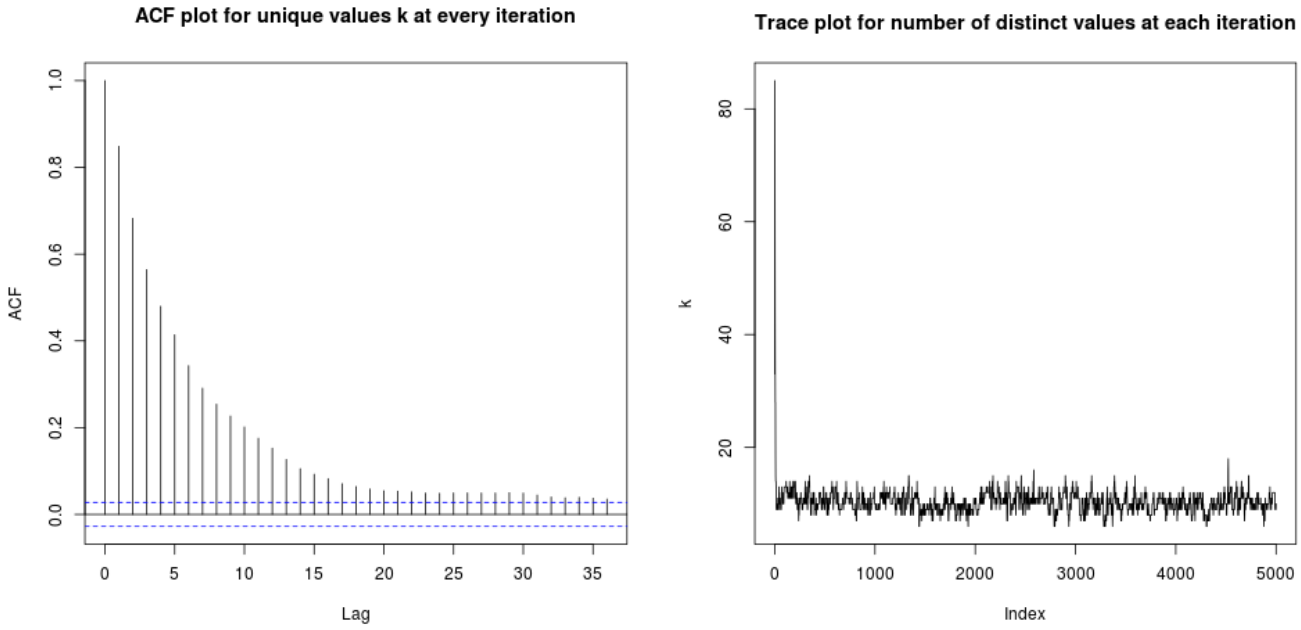
```
> table(mcmc_ci[,5000])

1539 1574 1578 1579 1583 1584 1585 1586 1587 1588
  34   13   52   28    3   21   16   25    6    2
> table(mcmc_phi[,5000])

 -1.61154760276548 -0.807874753617295   67.4009860657842   68.6418700221293   69.5578658594627
                28                  52                  3                 16                 21
  139.691625040919   140.313977580896   189.195633011361   189.603633119379   189.834515794012
                 6                  34                 13                 25                  2
>
```

(c) Figure showing the state of markov chain of indicators and parameters at 5000th iteration.

Figure 2. State of Markoc chains at 1st, 100th, and 5000th iteration.

From the above observations clustering effect is very much noticeable. In the very beginning number of unique groups which are randomly assigned are more and at the very end most of the data points are assigned to the nearest clusters. However one thing which is not very clear, even at the 5000th iteration, the data points belonging to table parameters which are very close to each other do not get assigned into the same cluster. The algorithm should definitely cluster together the parameters -1.6 and -0.8 into the same cluster to give a total of 80 observations, but somehow that doesn't happen.

Following plots show the diagnostic plots of the markov chains. Since it does not make sense to plot a markov chain of indicators, as indicator values don't hold a numerical significance. They are just merely a numbering convention. So I compute the number of unique values of these indicators at every iteration of the mcmc sampler and plot the diagnostics for this measure. Although in the plots it can be seen that the autocorrelation is decreasing. But the trace plot is not so convincing. Following figure 3 shows the plots.

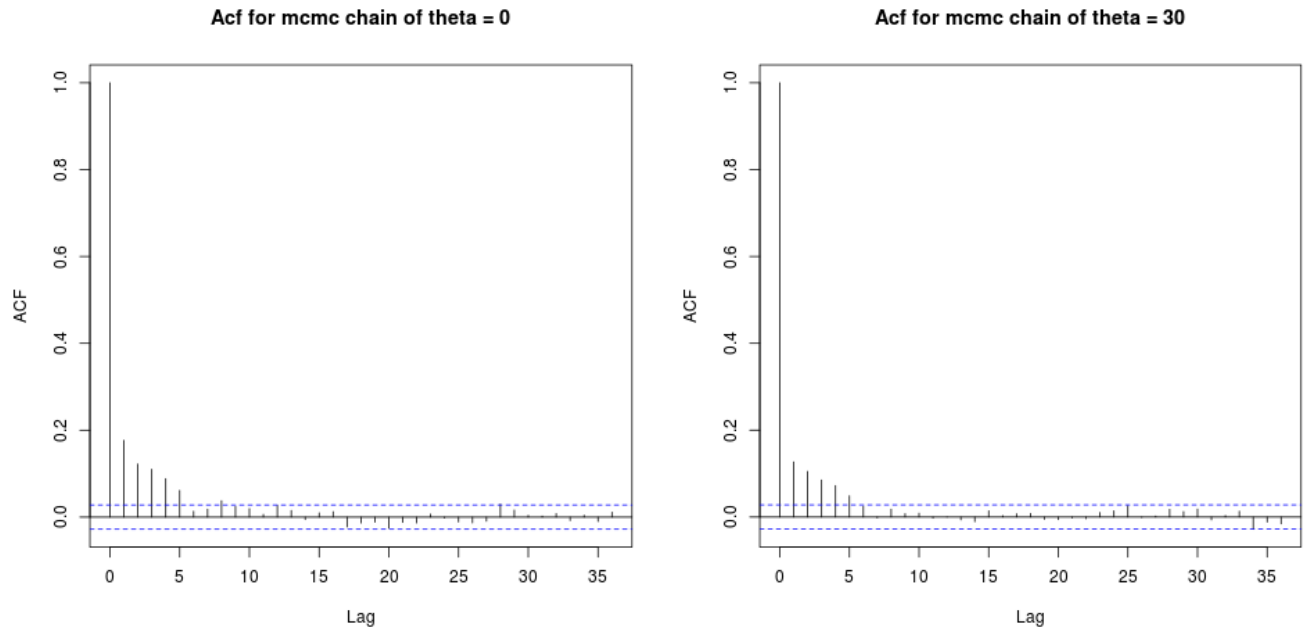


(a) Figure showing autocorrelation plot for number of unique values k at every iteration

(b) Figure showing trace plot for number of unique values k at every iteration.

Figure 3. Diagnostic plots for number of unique values k.

The next figures4 show the plots for cluster parameters in the final sampled table parameters.



(a) Figure showing autocorrelation plot $\theta = 0$

(b) Figure showing autocorrelation plot $\theta = 30$

Figure 4. Diagnostic plots for the original table parameters.

I also plotted the running mean, trace plots, for the original parameters θ=0,30,70,140 and from the graphical plots I can say that the chain has converged. But as I stated the problem earlier, that cluster parameters which were very close to each other tend to form different groups instead of clustering into one group. So, somehow this result says the chain has not converged. Apart from this I also performed *heidel.diag* and *geweke* test. Both the tests said the chain converged for some parameters and not for the others. Anyways my objective was to see the Clustering effect under non conjugate DPM model and that is very much evident.

References

1. Neal, R. M. Markov chain sampling methods for dirichlet process mixture models. *J. Comput. Graph. Stat.* **9**, 249–265 (2000). DOI 10.1080/10618600.2000.10474879.

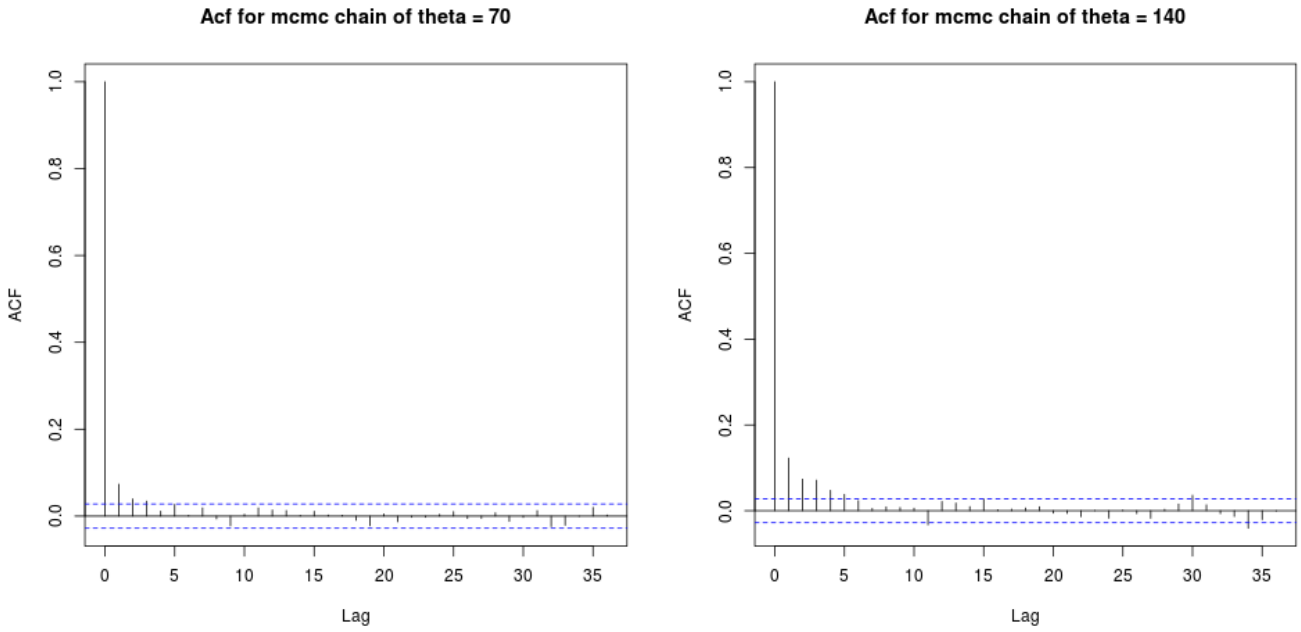
2. Ferguson, T. S. A bayesian analysis of some nonparametric problems. *The Annals Stat.* **1**, 209–230 (1973).

3. Ishwaran, H. & James, L. F. Gibbs sampling methods for stick-breaking priors. *J. Am. Stat. Assoc.* **96**, 161–173 (2001). DOI 10.1198/016214501750332758.

4. Antoniak, C. E. Mixtures of dirichlet processes with applications to bayesian nonparametric problems. *The Annals Stat.* **2**, 1152–1174 (1974).

5. Maceachern, S. N. & Müller, P. Estimating mixture of dirichlet process models. *J. Comput. Graph. Stat.* **7**, 223–238 (1998). DOI 10.1080/10618600.1998.10474772.

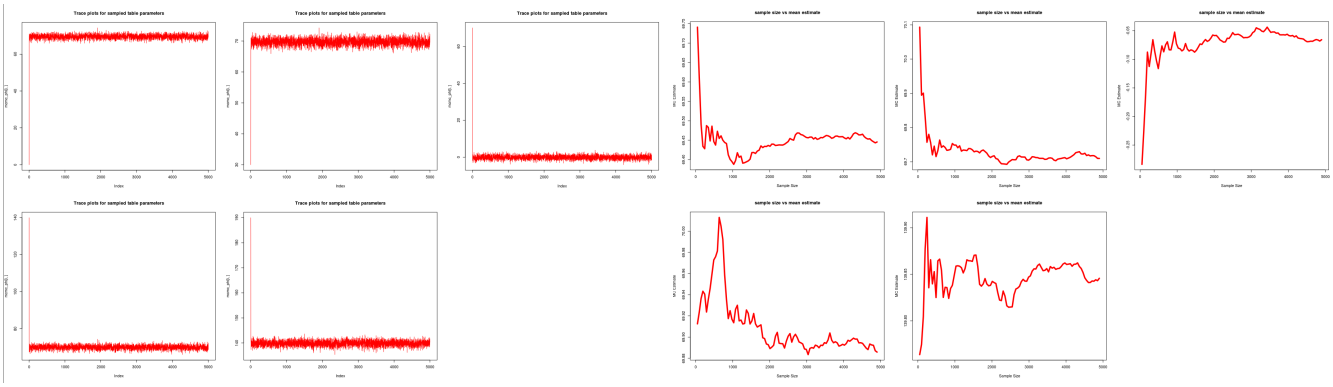
Additional figures



(a) Figure showing autocorrelation plot $\theta = 70$

(b) Figure showing autocorrelation plot $\theta = 140$

Figure 5. Diagnostic plots for the table parameters $\theta = 70, 140$.



(a) Figure showing trace plots $\theta = 0, 30, 70, 140$

(b) Figure showing running mean plot for $\theta = 0, 30, 70, 140$

Figure 6. Diagnostic plots for the table parameters $\theta = 0, 30, 70, 140$.