Sampling Methods for Dirichlet Process Mixture Models

Marius Alonso * Paul Bonin* Théo Communal*

January 7, 2024

Abstract

In this paper, we review and summarize algorithms from [Neal(2000)]. Dirichlet Process Mixture Models is a probabilistic unsupervised learning technique that assumes that the instances were generated from a mixture of several distributions with unknown parameters. As such, it can be understood as a more general case of Gaussian Mixture Models, as it allows to use different families of distribution. Furthermore, the number of distributions does not need to be specified beforehand. Sparsity of the clustering is enforced by the Dirichet Process "rich get richer" property given by the α parameter. We implemented these algorithms in Python and tested them on synthetic and real data (Old Faithful Dataset).

1 Dirichlet Process Mixture Models

1.1 Introduction

We have data $y_1, ..., y_n$ which we regard as independently drawn from some unknown distribution. We model the distribution from which the y_i are drawn as a mixture of distributions of the form $F(\theta_i)$, with the θ_i being drawn independently from G. We let the prior for this mixing distribution be a Dirichlet process with concentration parameter α and base distribution G_0. Hence, we have the following model:

$$\begin{aligned} y_i \mid \theta_i &\sim F(\theta_i) \\ \theta_i \mid G &\sim G \\ G &\sim \mathrm{DP}(G_0, \alpha) \end{aligned} \tag{1}$$

Here, $X \sim S$ means *X has the distribution S*. DP is a Dirichlet process (i.e. a distribution over distributions).

The goal of the following analysis is to find relevant algorithms that allow to compute the latent parameters $\theta_1, \ldots, \theta_n$ of each of the samples $y_1, \ldots, y_n$, given F, α and G_0.

1.2 Representation of Dirichlet Process

In [Neal(2000)], in order to sample $\theta_1, ..., \theta_n$ from G, two schemes are used in the various algorithms. Both contructions reflects the "rich get richer" property.

1.2.1 The Polya urn (or Blackwell-McQueen) scheme

Let's assume that G_0 is a distribution over colors, and that each θ_k represents the color of a single ball placed in the urn. Start with no balls in the urn. On step i:

- With probability proportional to α, draw $\theta_i \sim G_0$, and add a ball of that color into the urn.
- With probability proportional to $i - 1$ (i.e., the number of balls currently in the urn), pick a ball at random from the urn. Record its color as θ_i, and return the ball into the urn, along with a new one of the same color.

This process can be summarized by the following conditional distribution:

$$\theta_i \mid \theta_1, \ldots, \theta_{i-1} \sim \frac{1}{i - 1 + \alpha} \sum_{j=1}^{i-1} \delta(\theta_j) + \frac{\alpha}{i - 1 + \alpha} G_0 \tag{2}$$

Here, $\delta(\theta_j)$ is the distribution concentrated at the single point θ.

*Mines Paris, PSL University and Master MVA, ENS Paris-Saclay. Contact: {firstname.lastname}@etu.minesparis.psl.eu

1.2.2 The Chinese Restaurant Process

Imagine a restaurant that has an unlimited number of tables (that represent the clusters c, each with an associated parameter ϕ_c). The first customer (observation) sits at the first table. Customer i sits at:

- Table c with probability $\frac{n_{-i,c}}{\alpha+i-1}$, where $n_{-i,c}$ is the number of other customers at table c;
- A new table with probability $\frac{\alpha}{\alpha+i-1}$.

The Chinese restaurant process illustrate the "cluster" property of the DP, i.e., the more customers sit at a table, the higher the chance a new customer will choose to sit at this table. Therefore, only a limited number of tables will be occupied, although there is an unlimited number of tables available in the restaurant. We consider observation i to be the last, without loss of generality. This property makes it feasible for us to sample from a DP mixture, following this equation:

$$\begin{aligned} \text{If } c_j = c \text{ for some } j \neq i, P\left(c_i = c \mid c_{-i}\right) &= \frac{n_{-i,c}}{n-1+\alpha}, \\ P\left(c_i \neq c_j \text{ for all } j \neq i \mid c_{-i}\right) &= \frac{\alpha}{n-1+\alpha} \end{aligned} \tag{3}$$

1.3 Graph representation of Dirichlet Process

Figure 1 is the graphical model associated with the model introduced in (1). Figure 2 is the graphical model associated with the Chinese Restaurant Process, $\mathbf{p}$ being a mixing distribution vector of infinite length that sums to 1.

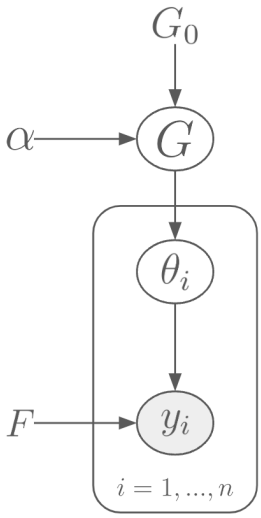

Figure 1: Hierarchical model defined by (1), directly used in **Algorithms 1, 6**

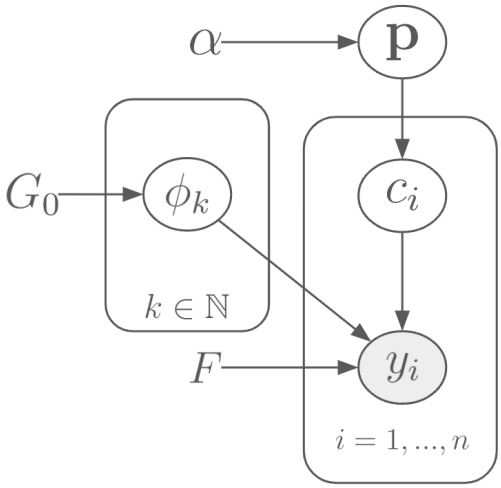

Figure 2: Chinese Restaurant Process representation of the Model, used in **Algorithms 2, 4, 5, 7, 8**

2 Gibbs Sampling and Conjuguate Priors

2.1 Conjugate Priors

2.1.1 Principle

The model is said to have conjugate priors when the following two operations are easily accessible:

- Sampling from $H_i = \theta_i \,|\, G_0, y_i$.
- Computing $\int F(y_i, \theta) dG_0(\theta)$

2.1.2 Example

For example, if the model is the following:

$$\begin{aligned} y_i \mid \theta_i &\sim \mathcal{N}(\theta_i, \tau^2) \\ \theta_i \mid G &\sim G \\ G &\sim \mathrm{DP}\left(\mathcal{N}(0, \sigma^2), \alpha\right) \end{aligned}$$

Then H_i follows a normal distribution of mean $(y_i/\tau^2)/(1/\sigma^2 + 1/\tau^2)$ and variance $(\sigma^{-2} + \tau^{-2})^{-1}$. In addition,

$$\int F(y_i, \theta) dG_0(\theta) = C \int \exp(-(y_i - \theta)^2/(2\tau^2)) \exp(-\theta^2/(2\sigma^2)) d\theta$$

is easily tractable. In addition, sampling from the *a posteriori* distribution $\phi_c \mid G_0, y_i$ for i such that $c_i = c$ is just sampling from a normal distribution of mean $\frac{\sigma^2}{\frac{\tau^2}{|Y_c|}+\sigma^2}\overline{Y_c}$ and variance $\left(\frac{1}{\sigma^2} + \frac{|Y_c|}{\tau^2}\right)^{-1}$, with $Y_c = \{y_i \mid c_i = c\}$.

2.2 Gibbs Sampling

2.2.1 A first approach

Gibbs sampling consists in sampling from the posterior distribution of each parameter, one at a time, while keeping the other parameters fixed. In our case, given the data y_i, we have to sample from the posterior distribution θ_i for $i = 1, ..., n$.

Recall the conditional prior from Equation 2

$$\theta_i \mid \theta_{-i} \sim \frac{1}{n - 1 + \alpha} \sum_{j \neq i} \delta(\theta_j) + \frac{\alpha}{n - 1 + \alpha} G_0$$

where θ_{-i} is the vector of all θ_j for $j \neq i$.

The conditional posterior is given by:

$$\theta_i \mid \theta_{-i}, y_i \sim q_{i,j} \sum_{j \neq i} \delta(\theta_j) + r_i H_i \tag{4}$$

where $q_{i,j}, r_i$ are numerical values and H_i is the posterior distribution of θ_i given y_i.

When the priors are conjugated, then we can compute the $q_{i,j}$ and the r_i, but also sample from H_i. In that case, we can deduce the following algorithm:

Algorithm 1. Let the state of the Markov Chain consist of $\theta = (\theta_1, \ldots, \theta_n)$. Repeatedly sample as follows:

- For $i = 1, \ldots, n$: Draw a new value from $\theta_i \mid \theta_{-i}, y_i$ as defined by Equation 4.

2.2.2 Sampling classes and class parameters separately

A rather baffling inefficiency with performing Gibbs Sampling on the $\theta_1, \ldots, \theta_n$ is that the algorithm cannot change the θ for more than one observation simultaneously. In the case where a large number of observations are naturally grouped and associated with one θ value, the algorithm will struggle to fine-tune this value. Indeed, it needs to pass through a low probability intermediate state, where observations in the group do not all have the same θ value.

In order to cope with that, Gibbs sampling can be performed on the class level instead. We resample successively the c_i, $i = 1, \ldots, n$, according to the other c_j, ϕ_{c_j}, $j \neq i$, and the data point y_i.

Here, ϕ_c is the parameter of the class c. This parameter is also resampled separately to adjust best to the data points y_i associated with class c.

All steps combined, we have the following algorithm:

Algorithm 2. Let the state of the Markov Chain consist of $\mathbf{c} = (c_1, \ldots, c_n)$ and $(\phi_c : c \in \{c_1, \ldots, c_n\})$. Repeatedly sample as follows:

- For $i = 1, \ldots, n$: Draw a new value from $c_i \mid c_{-i}, y_i, \phi$ as defined by Equation 4. If the new class value c_i^* is not associated with any other observation, draw a value for ϕ_{c_i} from H_i.
- For $c \in \{c_1, \ldots, c_n\}$, draw a new value from ϕ_c given all the data points y_i belonging to c, and the distribution G_0.

3 When Priors are not Conjugated

The question remains of what to do when priors are not conjugated. We explore hereafter various techniques to handle models with non conjugated priors, notably those relying on Metropolis-Hastings sampling.

3.1 Metropolis-Hastings

3.1.1 Direct implementation

The principle of the Metropolis-Hastings algorithm is to simulate a potential Markov chain step with the prior distribution, and accept or not the transition by computing an acceptance ratio:

$$a(x^*, x) = \min\left[1, \frac{g(x \mid x^*)\pi(x^*)}{g(x^* \mid x)\pi(x)}\right] \tag{5}$$

$\pi(x)$ is the distribution probability, g is the transition kernel, and x^* is a candidate state.

By choosing $g(c|c*) = p(c_i = c* |c_{-i})$, and as $\pi(c) = p(c_i = c|c_{-i}, y_i, \phi) = F(y_i, \phi_c)g(c|c*)$, g cancel each others and we obtain :

$$a(x^*, x) = \min\left[1, \frac{F(y_i, \phi_{c_i^*})}{F(y_i, \phi_{c_i})}\right] \tag{6}$$

We can now write down the new algorithm:

Algorithm 5. Let the state of the Markov Chain consist of $\mathbf{c} = (c_1, \ldots, c_n)$ and $(\phi_c : c \in \{c_1, \ldots, c_n\})$. Repeatedly sample as follows:

- For $i = 1, \ldots, n$: Draw a new value for c_i from the conditional prior given by Equation 3. Accept the transition with the ratio given in Equation 6.
- For $c \in \{c_1, \ldots, c_n\}$, draw a new value from ϕ_c given all the data points y_i belonging to c, and the distribution G_0.

Note that in the original paper [Neal(2000)], there is the possibility to repeat the attempts of resampling c_i R-times.

We can also simply update the $\theta = (\theta_1, \ldots, \theta_n)$ directly, as performed in Algorithm 1, which leads to the following algorithm:

Algorithm 6. Let the state of the Markov Chain consist of $\theta = (\theta_1, \ldots, \theta_n)$. Repeatedly sample as follows:

- For $i = 1, \ldots, n$: Draw a new value for c_i according to the prior defined by Equation 2.

  Compute the acceptance probability:

  $$a(\theta_i^*, \theta_i) = \min\left[1, \frac{F(y_i, \theta_{i^*})}{F(y_i, \theta_i)}\right]$$

  Set the new value of θ_i to θ_i^* with this probability; otherwise let the new value of θ_i be the same as the old value.

The limitations with sampling directly the θ mentioned in part 2.2.2 must not be forgotten however.

3.1.2 Exploring more classes

One might wonder whether an algorithm that creates new classes more often could be more efficient. Hence comes the **Algorithm 7**, which attempts in following such a goal.

Algorithm 7. Let the state of the Markov Chain consist of $\mathbf{c} = (c_1, \ldots, c_n)$ and $(\phi_c : c \in \{c_1, \ldots, c_n\})$. Repeatedly sample as follows:

- For $i = 1, \ldots, n$, update c_i as follows:
  - If c_i is not a singleton, set c_i to c_i^*, a newly created class, associated to a ϕ_i^* drawn from G_0, with acceptance probability:

    $$a(c_i^*, c_i) = \min\left[1, \frac{\alpha}{n-1}\frac{F(y_i, \phi_{c_i}^*)}{F(y_i, \phi_{ci})}\right]$$

– Else, draw c_i^* from c_{-i}, choosing $c_i^* = c$ with probability $n_{-i,c}/(n-1)$. Accept the transition with probability:

$$a(c_i^*, c_i) = \min\left[1, \frac{n-1}{\alpha}\frac{F(y_i, \phi_{c_i^*})}{F(y_i, \phi_{ci})}\right]$$

- For $i = 1, \ldots, n$, if c_i is not a singleton, choose a new value for c_i according to the un-normalized measure:

$$\frac{n_{-i,c}}{n-1}F(y_i, \phi_c)$$

- For $c \in \{c_1, \ldots, c_n\}$, draw a new value from ϕ_c given all the data points y_i belonging to c, and the distribution G_0.

3.2 Gibbs sampling with auxiliary parameters

In this part, when c_i is updated, we introduce m temporary variables that represent possible values for the parameters of components that are not associated with any other observations. We then update c_i by Gibbs sampling with respect to the distribution that includes these auxiliary parameters.

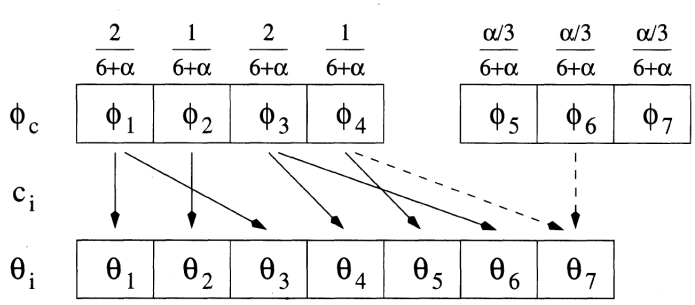

Figure 3: Example with $n = 4$ and $m = 3$. The component for the new observation is chosen from among the four components observations plus three possible new components, with parameters, ϕ_5, ϕ_6, ϕ_7 drawn independently from G_0. The probabilities used for this choice are shown at the top.

Algorithm 8. Let the state of the Markov Chain consist of $\mathbf{c} = (c_1, \ldots, c_n)$ and $(\phi_c : c \in \{c_1, \ldots, c_n\})$. Repeatedly sample as follows:

- For $i = 1, \ldots, n$:
  - Let k be the number of distinct classes c_j for $j \neq i$. Suppose all c_j are in $\{1, ..., k\}$.
  - If $c_i = c_j$ for some $j \neq i$, independently draw values from G_0 for θ_c with $k < c \leq k + m$.
  - If $c_i \neq c_j$ for all $j \neq i$, independently draw values from G_0 for θ_c with $k + 1 < c \leq k + m$.
  - Draw a new value for c_i from $\{1, ..., k + m\}$ using the following probabilities:

$$P\left(c_i = c \mid c_{-i}, y_i, \phi_1, \ldots, \phi_{k+m}\right) = \begin{cases} b\frac{n_{-i,c}}{n-1+\alpha}F\left(y_i, \phi_c\right) & \text{for } 1 \leq c \leq k \\ b\frac{\alpha/m}{n-1+\alpha}F\left(y_i, \phi_c\right) & \text{for } k < c \leq k+m \end{cases},$$

  where $n_{-i,c}$ is the number of c_j for $j \neq i$ that are equal to c, and b is the appropriate normalizing constant.

- For all $c \in \{c_1, \ldots, c_n\}$: Draw a new value from $\phi_c \mid y_i$ such that $c_i = c$.

4 Our implementation

4.1 Choices of implementation

We implemented in Python the algorithms 1, 2, 4, 5, 6, 7, 8 of [Neal(2000)]. Our implementation handles uni-dimensional data and components distribution of the form $y_i|\theta_i \sim F(\theta) = N(\theta, \tau^2)$ where τ^2 is fixed beforehand. The prior on θ_i, G_0 follows a normal distribution. We chose to have G_0 and F to follow conjugate laws to have closed form expressions of the quantities used in the algorithms. However, we also use the algorithms 4 to 8, that can also be applied for non-conjugate priors.

4.2 Data

We first tested the algorithms on a synthetic dataset, constructed through a DP sampler that we implemented. The following parameters were used : $G_0 = N(0, 1)$, $\alpha = 0.5$ and $F(\theta) = N(\theta, \tau = 0.1)$. This dataset is shown on Figure 4 and was used to assess the performance of algorithms.

We also tested the algorithms a real dataset: the "Old Faithful Dataset"[1]. This dataset contains the waiting time between the eruptions of the Old Faithful geyser in Yellowstone National Park, Wyoming, USA. We observe that the observations seem to come from a mixture of two Gaussians. We will try to infer the means of each of those Gaussians as well as which cluster each observation belongs to. Regarding the variance, we tested different parameters and $\tau = 5.8$ for both Gaussian seems to be a good fit. We changed accordingly F, G_0 and performed parameter sweeping on the value of α.

4.3 Results

4.3.1 Performance on a synthetic dataset

	time per it. (ms)	Auto-correlation time n. of clusters	Auto-correlation time θ or $\phi[c]$	Rand Index w.r.t. synthetic classes
Algorithm 1	4.1	8.4 ±6.4	9.0 ±5.7	0.97 ±0.03
Algorithm 2	8.9	3.3 ±1.3	1.1 ±0.1	0.98 ±0.02
Algorithm 4	10.2	57.2 ±8.3	1.7 ±0.3	0.87 ±0.06
Algorithm 5	6.9	35.9 ±18.6	22.9 ±8.0	0.94 ±0.04
Algorithm 6	1.2	25.4 ±10.6	31.8 ±12.4	0.93 ±0.05
Algorithm 7	12.2	1.0 ±0.0	1.4 ±1.0	0.93 ±0.02
Algorithm 8	49.6	2.9 ±0.7	1.3 ±0.3	0.92 ±0.02

Table 1: Evaluation of the algorithms on a synthetic dataset, with n = 60 (number of points), N = 200 (number of iterations per run), and 20 runs per algorithm.

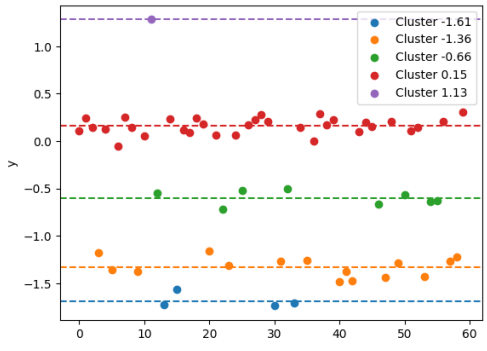

Figure 4: Synthetically generated data. Axis x has no meaning in itself, and is only used for visualization. Axis y is the value of observed data points. The class assignments depends on the values of the θ_i used for each class for data generation.

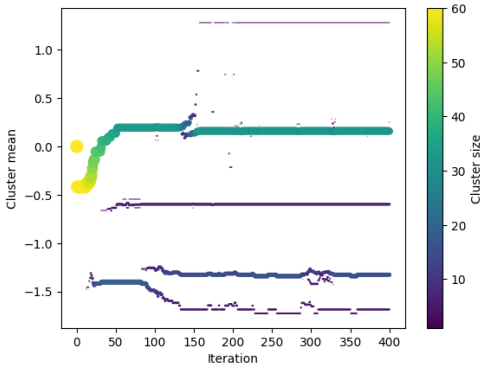

Figure 5: Evolution of class means during the iterations, for **Algorithm 5**

The metric **Rand Index** is computed by comparing the class assignment obtained at the end of the execution to the *a priori* class assignment (either the implicit assignment of the vector θ used to generate synthetic data y, or an external user-defined asignment). Note that this assignment is not necessarily the most likely *a posteriori* assignment.

The metric **Autocorrelation Time (ACT)** is defined as $1 + 2\sum_{i=1,\dots,T_0} \gamma(i)$ where γ is the autocorrelation function of the signal of interest (here the θ_i, $\phi[c_i]$ or number of distinct clusters) and $T_0 = \min(i|\gamma(i) \leq 0)$. This

[1] https://www.stat.cmu.edu/~larry/all-of-statistics/=data/faithful.dat

metric gives quantifies how closely the samplers generates i.i.d. variables. In that case, ACT = 1. However, this metric do not give any information on whether the sampler generates pertinent clusters (in the sense of the likelihood).

4.3.2 Performance on the Old Faithful Dataset - Discussion on the value of α

The scaling parameter α can be interpreted as an inverse variance of the DP because it specifies how the realizations of the DP are shared among singular values / clusters: when $\alpha = 0$, the realizations are all concentrated in a unique cluster, while in the limit of $\alpha \to \infty$ the realizations become scattered in many clusters. Between the two extremes the realizations are discrete distributions with more and more diffusion as α increases.

To understand the effect of α, we ran the **Algorithm 5** 10 times with 400 iterations on the Old Faithful Dataset for different values of α chosen through parameter sweeping.

We observe that when α is too small (e.g. $\alpha = 0.001$), the DP is very very unlikely to explore different values of θ and thus fails to cluster data. Increasing α favorizes exploration : when $\alpha = 0.01$, convergence to the two clusters occurs in 200 iterations on average, reduced to circa 50 when $\alpha = 0.05$. This range of α seems to be appropriate for this problem as exploration of new θ is not too frequent, especially once great clusters have been found. On the contrary, for higher values of α (e.g. 0.1), great clusters are found quickly (in c. 20 iterations) but exploratory phenomena occurs, leading to creation of sub-optimal clusters and likelihood drop. These phenomena disappear after some iterations. Eventually, when $\alpha \geq 0.5$, exploratory phenomena occur very frequently and disappear less often, leading to creation of unnecessary clusters and loss of performance.

For the Old Faithful Dataset, an appropriate value of α seems to be 0.1, a trade-off between computational efficiency and convergence of the likelihood.

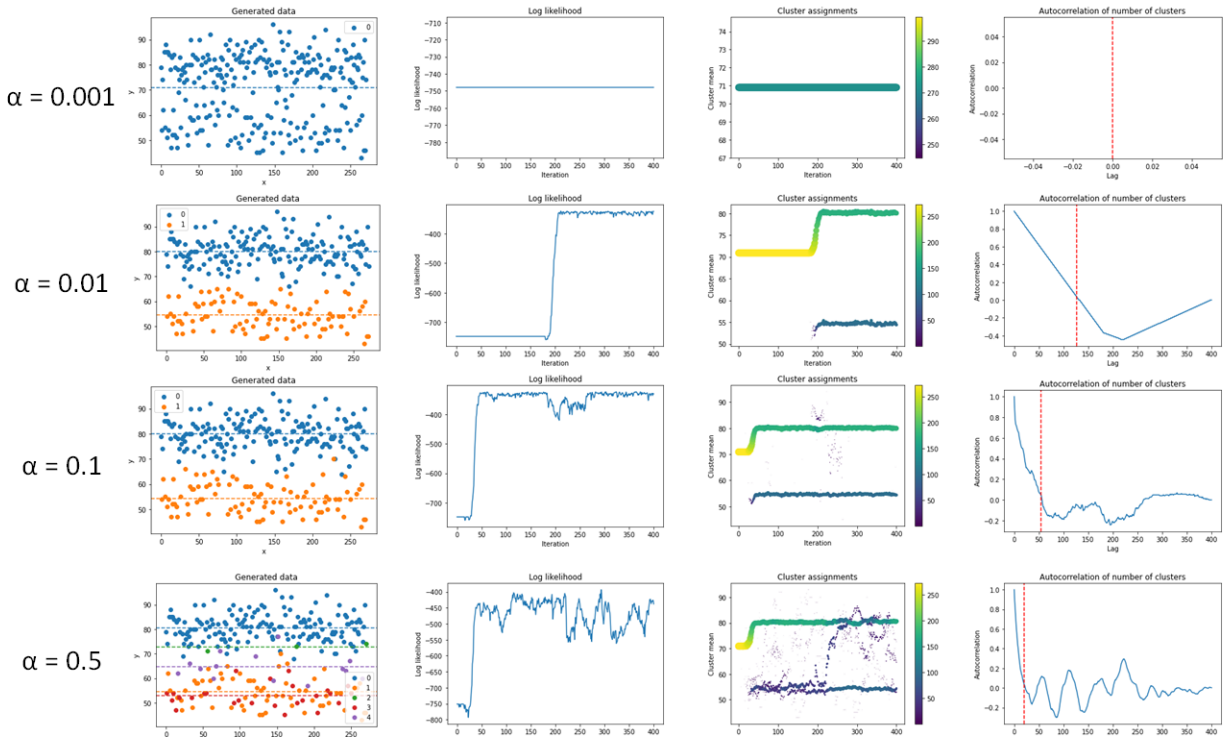

Figure 6: Clustering of the Old Faithful dataset, Loglikelihood, clusters evolution and auto-correlation time for different values of α. Only one realization is shown but 10 were run - we paid attention that shown results are truthful to results obtained on average.

4.4 Remarks

- Coding algorithms was not straightforward, since Neal gives somewhat unclear instructions (example: "if" loop with no clear "endif") or uses inconsistent notations along the paper.
- Two choices of initialization have been explored : initializing with $\theta_i = \mathbb{E}[G_0]$ / $c_i = 0$ and $\phi_0 = \mathbb{E}[G_0]$, initializing with $\theta_i = y_i$ / $c_i = i$ and $\phi_i = y_i$.
- Algorithms 5 and 6 are rather slow in terms of iteration to converge (see *auto-correlation times*), but on the other hand achieve good time performance per iteration.
- Algorithm 4 is the so-called "no gaps" algorithm from [MacEachern et al.(1998)], also reproduced in [Neal(2000)]. It seems rather exploratory by nature, creating and deleting many classes along each iteration, but does not seem to converge systematically to a *proper* class assignment.
- Comparing "Time per iteration" between algorithms and with the values obtained by [Neal(2000)] can be unfair, as we didn't search for optimal execution time in a systematical way, and (2) [Neal(2000)]'s implementation were done more than two decades ago on a different programming language (R).

References

[MacEachern et al.(1998)] Steven N. MacEachern, Peter Müller, and Peter Muller. 1998. Estimating Mixture of Dirichlet Process Models. *Journal of Computational and Graphical Statistics* 7, 2 (June 1998), 223. https://doi.org/10.2307/1390815

[Neal(2000)] Radford M. Neal. 2000. Markov Chain Sampling Methods for Dirichlet Process Mixture Models. *Journal of Computational and Graphical Statistics* 9, 2 (2000), 249–265. http://www.jstor.org/stable/1390653