UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
PROGRAMA DE MAESTRÍA Y DOCTORADO EN CIENCIAS MATEMÁTICAS Y DE LA ESPECIALIZACIÓN EN ESTADÍSTICA APLICADA

DIRICHLET GEOMETRIC PROCESS

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIAS

PRESENTA:
CARLOS FIDEL SELVA OCHOA

DIRECTOR
RAMSÉS HUMBERTO MENA CHÁVEZ
DEPARTAMENTO DE PROBABILIDAD Y ESTADíSTICA, IIMAS, UNAM

CIUDAD DE MÉXICO, 20 DE OCTUBRE DEL AÑO 2020.

Dedication

I want to dedicate this thesis firstly to Maria Fernanda Gil-Leyva Villa and Ramsés H. Mena without whom this thesis would not have been possible. Also and in alphabetical order, Alejandro Santoyo, Ana Castillo, Bruno Martinez, Gibrán Morales, and my family for their help and support both economically and mentally. Lastly, to those who can take some utility from this, sincerely hoping that it will not be an empty set for too long.

Objective

The aim of this thesis is to introduce a new species sampling process, study its properties, and show with a practical example and a package implementation, its usefulness for density estimation and clustering. We will begin by defining a random probability measure, a species sampling process, their construction, and some useful results to prove when a species sampling process has full support in the space of target probability measures. A brief introduction to exchangeable random variables and de Finetti's theorem are given as the ground for density estimation by means of species sampling mixtures. We also introduce the Dirichlet process, the Geometric Process and the Dirichlet Geometric Process (DGP). Then we describe the steps and properties of density estimation and clustering with the DGP mixture. Finally we conclude with numerical illustrations and comparison against commonly used methods.

Contents

Chapter 1

Introduction

Density estimation and clustering have become very useful statistical tools for data science and machine learning thanks to their usefulness to make predictions, establish boundaries, segregate groups with similar traits, among others.

On its basics, performing density estimation for a population means to give a proposal for the probability density function of the population using the information already available, namely, the observed sample.

Given an observed sample $(Y_1, \dots, Y_n)$, there are many ways to make inference about a future observation Y_{n+1}, all of them with their respective biases and assumptions. A straightforward approach would be to use the empirical distribution function of the observed sample

$$F_Y(y) = n^{-1} \sum_{i=1}^{n} \mathbf{1}(Y_i \leq y)$$

as the population density from which Y_{n+1} will be sampled. Here, although it seems like we are not making any assumption at all, there is actually a very strong assumption: the discreteness of the next observation.

So what if we know that the data source draws continuous variables? We can make the empirical distribution function continuous by joining the discontinuity points with a straight line, but then we are modeling the next variable as being uniform between observations, and giving zero probability to a new minimum or maximum value.

Among other methods are taking an histogram of the data and scaling it to be a probability density function (this density will depend greatly on how we bin the data), or do a Kernel Density Estimation (KDE) in which the estimate is given by $\rho_K(x;h) = n^{-1}\sum_{i=1}^{n} K(x - Y_i; h)$. The variable h is called the bandwidth parameter and controls the density "thickness" that each observation contributes, and the function $K(x;h)$ is the kernel, a positive function that integrates to one. Examples of kernels are (Bonaccorso, 2019) the Tophat Kernel defined as $K(x;h) = \frac{1}{2h}\mathbf{1}(|x| < h)$ or the Gaussian Kernel defined as $K(x;h) = \frac{1}{\sqrt{2\pi}h}\exp\left(-\frac{x^2}{2h^2}\right)$.

This model can be enhanced in a couple of ways. To begin with, the KDE of n observations is an equal weighted mixture of n densities. This is improved with a Dirichlet Process Mixture (DPM) which is a Species Sampling Mixture where each weight is drawn from a Dirichlet Process. Other improvement is that each kernel in a KDE is homogeneous, whereas in a DPM, the parameter of the i'th kernel ξ_i is drawn from a priori distribution g. There is another density estimation method based on a Species Sampling Mixture with the Geometric Process as a driver, called the Geometric Process Mixture (GPM). It was introduced in Fuentes-García, Mena, and Walker (2009). In this article they concluded that the Geometric Process "is sufficient for mixture modeling and that the current trend of further elaborating on Dirichlet process mixture models must be compromised with an application that potentially would require more complicated weights specifications".

One of the main reasons for choosing the GPM over the DPM is that the process of training the models for a particular data set can be computationally faster with the former. But having as a disadvantage that the number of fitted groups might get greatly overestimated due to lack of a flexible weighting structure such as the

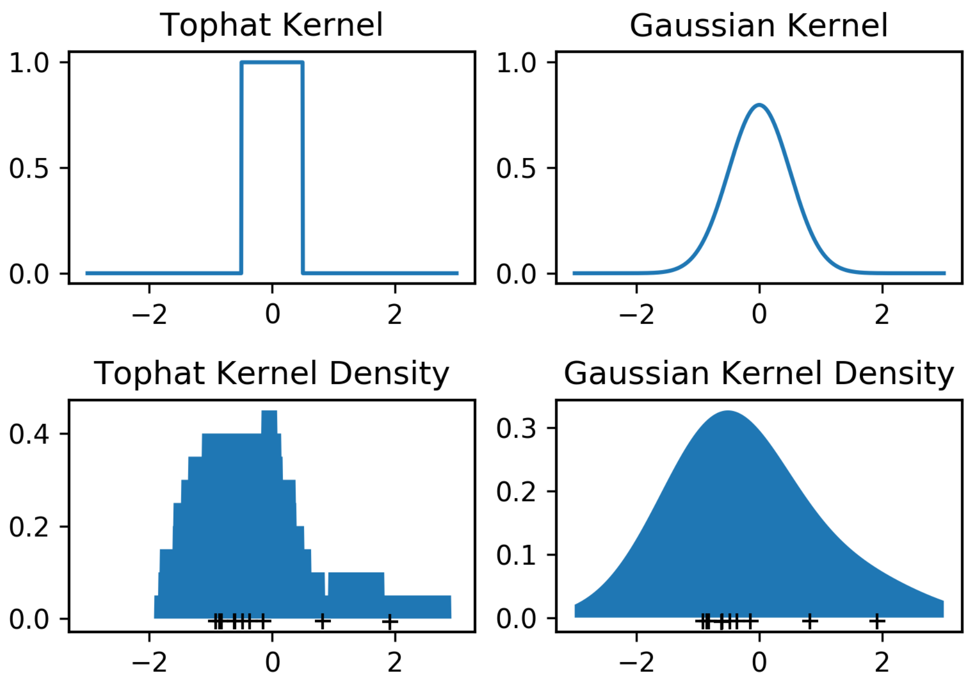

Figure 1.1: The Tophat Kernel to the top left and the Gaussian Kernel at the top right, both with bandwidth parameter $h = \frac{1}{2}$. At the bottom the respective KDE for a sample of ten data points.

one showcased in the DPM.

Following with our introduction, clustering is the process of partitioning a set of observations $(Y_1, \ldots, Y_n)$ into groups, usually so that elements in a same group share more similarities with that group than with any other.

One common method for clustering is the k-means algorithm. It is an iterative method to cluster a set of n observations into a previously known number, k, of groups via a centroid distance minimization step. The algorithm goes as follows (Han, Kamber, and Pei, 2012):

1. arbitrarily choose k distinct elements from $(Y_1, \dots, Y_n)$ as the initial cluster centers.
2. **repeat:**
   (a) (re)assign each element to the cluster with nearest center.
   (b) update the cluster centers, that is, calculate the mean value of the elements of each cluster.
3. **until** no change.

Again, there are some drawbacks. This algorithm can vary drastically the final clusters depending on the initial choice of elements, it is sensitive to outliers and heteroscedasticity between groups, furthermore the number of groups to cluster is needed as a parameter. Some of this points are illustrated in Figure 1.2 from the scikit-learn project (Pedregosa et al., 2011). "This example is meant to illustrate situations where k-means will produce unintuitive and possibly unexpected clusters. In the first three plots, the input data does not conform to some implicit assumption that k-means makes and undesirable clusters are produced as a result. In the last plot, k-means returns intuitive clusters despite unevenly sized blobs".

Clustering with the DPM improves over the k-means algorithm in all the problems illustrated in Figure 1.2 since the number of clusters needed is inferred from the data so an arbitrary number of groups can be assigned, and the distribution of each cluster is independently fitted. This improvements can be seen empirically in Figure 1.3.

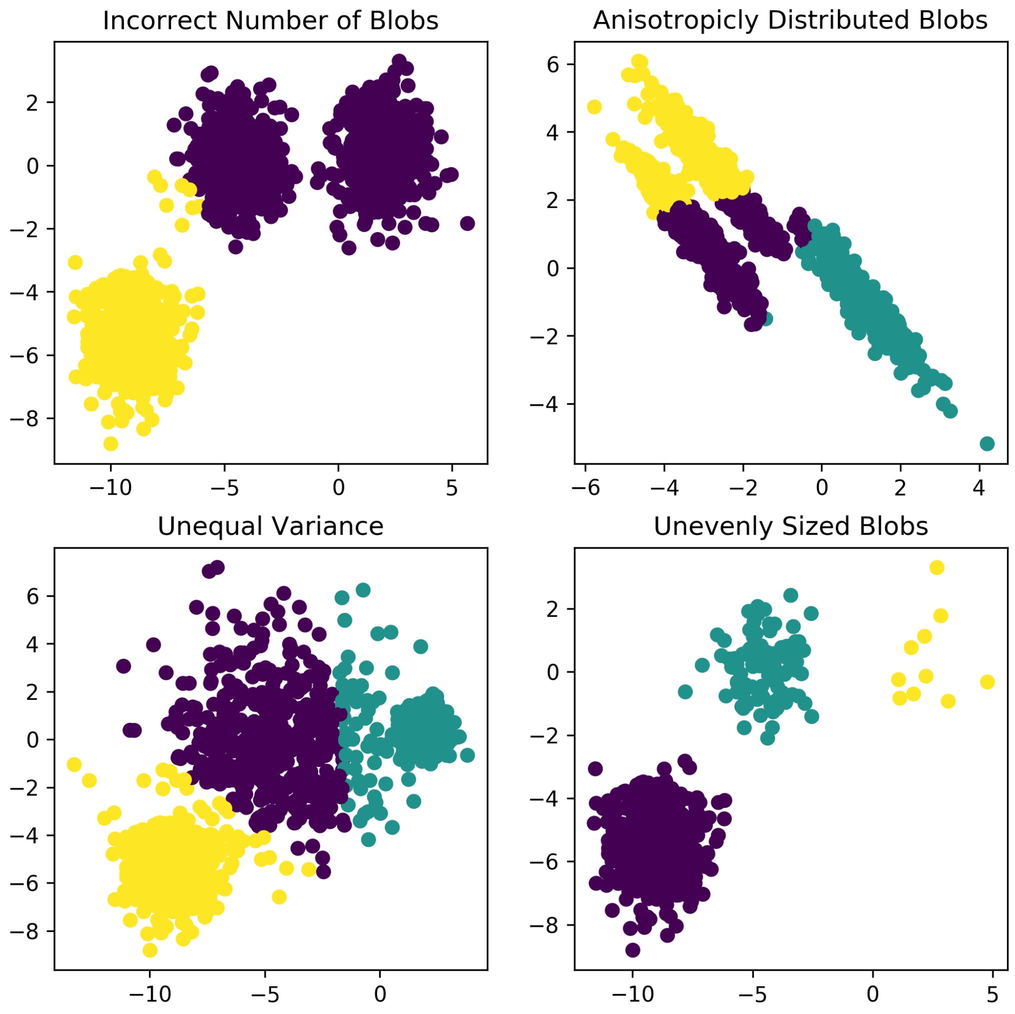

Figure 1.2: Plots of k-means examples from scikit-learn project site.

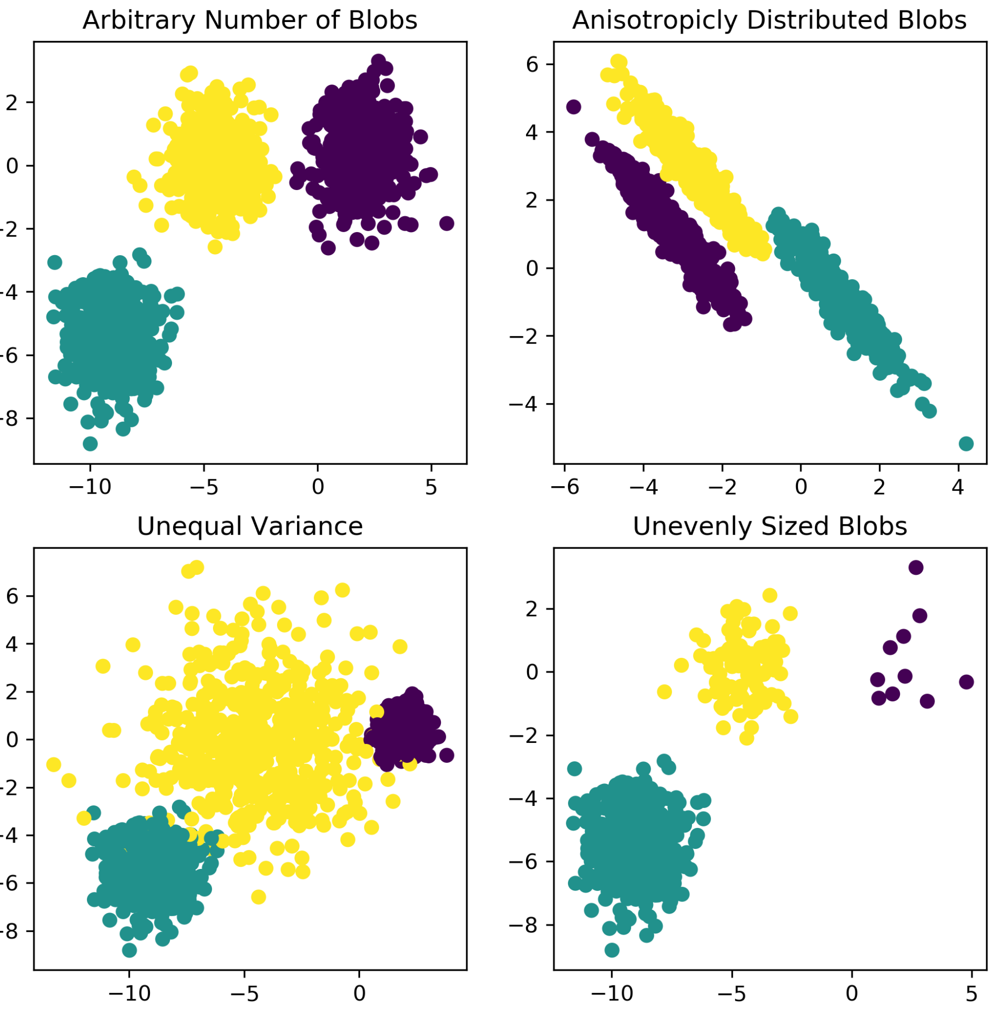

Figure 1.3: Plots of DPM examples from scikit-learn project site.

Chapter 2

Preliminaries

2.1 Random Probability Measures

A random probability measure is a random object whose values are probability measures themselves.

Definition. A random probability measure Q is a measurable mapping from a probability space $(\Omega, \mathscr{F}, \mathbb{P})$ into a measurable space $(\mathcal{P}, \mathscr{G})$, where $\mathcal{P}$ is the set of all probability measures on a common measurable space $(\mathbb{X}, \mathscr{X})$ and $\mathscr{G}$ is a σ-field on $\mathcal{P}$.

As a brief example, let the probability space be the toss of a fair coin and Q defined as a Normal$(0, 1)$ if heads and Normal$(0, 2)$ if tails.

In general the space of random probability measures can be too broad so it is customary to ask $\mathbb{X}$ to be a Polish space to make it more manageable. Then, $\mathcal{P}$ is also Polish under the induced Prokhorov metric (Appendix, Definition 11). Moreover, a sequence in $\mathcal{P}$ converges in metric if and only if it converges weakly and to the same limit. Then we can take $\mathscr{G} = \mathscr{B}(\mathcal{P})$, the Borel σ-field of $\mathcal{P}$.

An open base of neighborhoods for $\mu \in \mathcal{P}$ in the topology of weak convergence (hereafter called weak topology) is the class of sets

$$U_{\epsilon_1,\dots,\epsilon_k}(\mu, A_1, \dots, A_k) := \cap_{j=1}^{k}\{\nu \in \mathcal{P} : |\nu(A_j) - \mu(A_j)| < \epsilon_j\},$$

where k is a positive integer, $(\epsilon_1, \dots, \epsilon_k) \in (0, \infty)^k$ and $(A_1, \dots, A_k)$ is a k-tuple of μ-continuity sets in $\mathscr{X}$. Also μ belongs to the support (in the weak topology) of a random probability measure Q if and only if

$$\mathbb{P}(Q \in U_{\epsilon_1,\dots,\epsilon_k}(\mu, A_1, \dots, A_k)) > 0,$$

for every $k \leq 1$ and every k-tuple $(A_1, \dots, A_k)$ in the μ-continuity subsets of $\mathscr{X}$.

Proposition 1. *Let Q be a random probability measure on $(\mathbb{X}, \mathscr{X})$, such that $g(\cdot) = E(Q(\cdot))$, for every $A \in \mathscr{X}$. If μ is a probability measure on $(\mathbb{X}, \mathscr{X})$ belonging to the support (in the weak topology) of the distribution of Q, then $S_\mu \subseteq S_g$ where S_ν stands for the support of the deterministic measure ν.*

This results can be found in Parthasarathy (2005). Even under these constrains the space of random probability measures can get too broad to be tractable for density estimation purposes. For this reason it is useful to narrow down the aforementioned space. That can be done by means of the species sampling process, which we will introduce next.

2.2 Species Sampling Process

Definition. A species sampling process is a random probability measure that can be decomposed as

$$Q = \sum_{j=1}^{\infty} w_j \delta_{\xi_j} + \left(1 - \sum_{j=1}^{\infty} w_j\right) g,$$

where g is a probability measure on $\mathscr{X}$ which is diffuse (that is $g\{x\} = 0$ for every x in $\mathbb{X}$), and $(w_j)_{j\geq 1}$ and $(\xi_j)_{j\geq 1}$ are two independent sequences of random variables called weights and locations respectively such that $w_j \geq 0$ for every $j \geq 1$ and $\sum_{j=1}^{\infty} w_j \leq 1$, $\mathbb{P}$-a.s. and $(\xi_j)_{j \geq 1}$ is an i.i.d. sequence whose common distribution is g, called the base measure of Q. If $\sum_{j=1}^{\infty} w_j = 1$, P-a.s. then Q is called a proper sampling process.

Some random probability measures have a property called *full support*. It is an attribute of interest since it give us insight about which measures are attainable by the random probability measure. Roughly speaking and taking into account Proposition 1, a species sampling process has *full support* if the support of the random probability measure is as large as possible.

Definition 2. Let Q be a random probability measure with prior distribution Π and define the measure $g(\cdot) := E(Q(\cdot))$. Then Π (or equivalently Q) is said to have full support if and only if the support (in the weak topology) of Π is the set of all measures $\mu \in \mathcal{P}$ such that $S_\mu \subseteq S_g$.

We can easily see if any given species sampling process has full support thanks to the following proposition.

Proposition 3. *Consider the distribution Π for Q on $\mathcal{P}$ to be the distribution of a species sampling process. The following statements are equivalent:*

1. *The prior distribution Π has full support.*
2. *For every $\epsilon > 0$, $P\left(\max_{j \geq 1} \overline{w}_j < \epsilon, \sum_{l \geq 1} w_l > 1 - \epsilon\right) > 0$, where*
$$\overline{w}_j = w_j / \sum_{l \geq 1} w_l.$$
3. *For every $\epsilon > 0$ there is an integer $m \geq 1$ such that*
$$P\left(\{w_j < \epsilon : j = 1, \dots, m\}, \sum_{l=1}^{m} w_l > 1 - \epsilon\right) > 0.$$

The proof of this proposition can be found in Bissiri and Ongaro (2014).

2.3 Species Sampling Mixture

In a classical arrangement, a mixture is a probabilistic combination of two or more probability distributions. This combination can be done by choosing randomly, with distribution P, the parameter, ξ, from the domain of parameters of a parametric distribution K_ξ, and then computing the expected distribution $Q = \int K_\xi dP$. The case is analogous if P is a random probability measure, and when P is a proper species sampling process, we have a species sampling mixture.

Definition. A Species Sampling Mixture is a random probability measure that can be decomposed as

$$Q = \sum_{j=1}^{\infty} w_j K_{\xi_j},$$

where K_ξ is a member of a parametric family on $\mathscr{X}$, and $(w_j)_{j\geq 1}$ and $(\xi_j)_{j\geq 1}$ are the weights and location sequences of a proper species sampling process.

2.4 Stick-Breaking

Since the weights and atoms in a species sampling process are mutually independent, we can focus in ways to generate a collection of random variables in $[0,1]$ whose sum is less than one. That is, in the space

$$\Delta_\infty := \left\{ \mathbf{s} = (s_1, s_2, \dots) : s_j \geq 0, j \in \mathbb{N}, \sum_{j=1}^{\infty} s_j \leq 1 \right\}.$$

One method is via the "stick-breaking" construction, described in Ishwaran and James (2001), where we picture a stick of length 1 and start breaking the stick randomly at a point according to the realization of a random variable $0 \leq v_1 \leq 1$ and assign this stick length v_1 to the first weight w_1. We then break the remaining stick $1 - v_1$ according to the realization of another random variable $0 \leq v_2 \leq 1$, and assign this length to the following weight $w_2 = (1 - v_1)v_2$. Continuing in this way, the broken stick length after j steps is

$$w_j = [\prod_{l=1}^{j-1}(1 - v_l)]v_j,$$

and the leftover length is

$$1 - \sum_{l=1}^{j} w_j = \prod_{l=1}^{j}(1 - v_j).$$

We will call $\mathbf{V} = (v_j)_{j\geq 1}$ the sequence of sticks and $\mathbf{W} = (w_j)_{j\geq 1}$ the sequence of weights. We will also call the function acting on $\mathbf{V}$ the “stick-breaking function” defined by $\mathcal{SB}(\mathbf{V}) = (v_1, (1 - v_1)v_2, \dots) = \mathbf{W}$.

2.4.1 Dirichlet Process

Now, we will give the definition of a Dirichlet Process (DP) and its construction via stick-breaking which is called the Sethuraman representation.

Definition. A random measure Q on $(\mathcal{P}, \mathscr{G})$ is said to follow a Dirichlet Process distribution $\mathcal{D}_\alpha$ with base measure α if for every finite measurable partition $A_1, \dots, A_k$ of $\mathscr{X}$,

$$(Q(A_1), \dots, Q(A_n)) \sim \text{Dirichlet}(k; \alpha(A_1), \dots, \alpha(A_k)),$$

where $\alpha(\cdot)$ is a finite positive Borel measure on $\mathscr{X}$.

Given the base measure of a Dirichlet Process, $\alpha(\cdot)$, we can construct a species sampling process by doing the following steps.

- Sample each element of $\Theta = (\xi_j)_{j\geq 1}$ independently from $g(\cdot)$, the normalized base measure. This is possible since α is finite so $g(\cdot) = \alpha(\cdot)/\theta$, where $\theta = \alpha(\mathscr{X})$ is called the scale or total mass.

- Sample each element of $\mathbf{V}$ independently from the distribution $Beta(1, \theta)$ and transform it via the "stick-breaking process" to $\mathbf{W} = \mathcal{SB}(\mathbf{V})$.

Obtaining with this a species sampling process.

Sethuraman proved that this is a proper species sampling process and a Dirichlet Process with base measure α (Sethuraman, 1994). Thus we can give a representation of the Dirichlet Process via a proper species sampling process by giving the distribution g, and the scale $\theta > 0$, and denote it by $\mathrm{DP}(g, \theta)$.

The Dirichlet Process has the following characteristics demonstrated in Ghosal and van der Vaart (2017),

Proposition 4. *If $Q \sim \mathrm{DP}(g, \theta)$, then for any measurable sets A and B,*

$$\begin{aligned} E(Q(A)) &= g(A), \\ \mathrm{Var}(Q(A)) &= \frac{g(A)g(A^c)}{1+\theta}, \\ \mathrm{Cov}(Q(A), Q(B)) &= \frac{g\,(A \cap B) - g\,(A)\,g\,(B)}{1+\theta}, \\ E(Q(A)Q(B)) &= \frac{g\,(A \cap B) + g\,(A)\,g\,(B)\,\theta}{1+\theta}. \end{aligned}$$

Another useful characteristic of the Dirichlet Process is the conjugacy also demonstrated in Ghosal and van der Vaart (2017).

Theorem 5. *If $X_1, \ldots, X_n | Q \overset{iid}{\sim} Q$ and $Q \sim \mathrm{DP}(g, \theta)$, then the posterior distribution of Q is given by $Q|X_1, \ldots, X_n \sim DP(g^*, \theta + n)$ where $g^* = (\theta g + \sum_{i=1}^n \delta_{X_i})/(\theta + n)$.*

The species sampling representation of the DP shows that it is discrete almost surely.

One of the simplest ways to draw samples from a $\mathrm{DP}(g, \theta)$, is through the construction of a Polya urn scheme, proven and discussed in Ghosal and van der Vaart (2017) which goes as follows. To sample $\{X_1, \ldots, X_n\}$ from a $\mathrm{DP}(g, \theta)$, first draw $X_1 \sim g$ and recursively draw $X_{n+1} | X_1, \ldots, X_n \sim g_n := (\theta g + \sum_{i=1}^n \delta_{X_i})/(\theta + n)$.

2.4.2 Geometric Process

The Geometric Process is another species sampling process introduced by Fuentes-García, Mena, and Walker (2009) and is very useful for density estimation since it is easier to implement than the DP.

Definition. A $GP(g, \alpha, \beta)$ (Geometric Process) is a species sampling process characterized by an i.i.d. sequence of locations $(\xi_j)_{j\geq 1}$ from a distribution g, and a sequence of weights $(w_j)_{j\geq 1}$ via stick-breaking from the sequence $(v_j)_{j\geq 1}$ where all $v_j = p$ and $p \sim \text{Beta}(\alpha, \beta)$.

It has been shown by Bissiri and Ongaro (2014) that both the Dirichlet Process and the Geometric Process have full support, introduced in Definition 2.

2.5 Introduction to de Finetti's Theorem

It is often easier to handle i.i.d. observations than it is to handle observations from an infinite exchangeable sequence. A common approach to alleviate the numerical problems arising from exchangeable sequences, or even avoiding them completely, is applying de Finetti's theorem.

Theorem (de Finetti)**.** *For any infinite random sequence $\mathbf{Y} = (y_1, y_2, \dots)$ in a Borel space S, the following conditions are equivalent:*

1. *$\mathbf{Y}$ is exchangeable.*
2. *$P[\mathbf{Y} \in \cdot | Q] = Q^\infty$ a.s. for some random distribution Q on S. Where Q^∞ is the infinite product measure defined over S^∞.*

Moreover, Q is a.s. unique and given by

$$Q(A) = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}_A(y_i).$$

The proof of this theorem can be found in Kallenberg (2002). This is a strong result that links exchangeable sequences of random variables to conditionally independent sequences, made independent by conditioning to Q, which is called the directing random measure. The distribution of this random probability measure is called the "de Finetti's measure".

As a brief example lets think of an exchangeable sequence of $\{0,1\}$-valued variables $\{B_i\}_{i\geq 1}$ so that the directing random measure η is defined over the set $\{0,1\}$. If we name $p = \eta(\{1\}) = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{\{1\}}(B_i)$, then η is completely characterized by p and even more, the random variables $\{B_i|p\}_{i\geq 1}$ are an i.i.d. sequence of Bernoulli random variables.

2.6 Density Estimation

An interesting application of a species sampling mixture in statistics is density estimation. One can estimate the underlying density of a finite collection of observations $(Y_1,\ldots,Y_n)$ if its assumed that they come from an infinite sequence of exchangeable variables. Indeed we can loosen the usual hypothesis of pairwise independent observations thanks to de Finetti's theorem and try to estimate a realization of Q, the common random distribution from which the observations are independent.
Thus when we have the observations $(Y_1,\ldots,Y_n)$ drawn from a random probability measure Q, our goal is to compute

$$f_{Y_{n+1}|Y_1,\ldots,Y_n}(\cdot) = E[Q(\cdot)|Y_1,\ldots,Y_n],$$

which for an arbitrary random probability measure is a very hard endeavor. However, one can make things easier by modeling Q as a species sampling mixture. We will give an algorithm for density estimation using a particular species sampling mixture in the third chapter.

2.7 Clustering with Species Sampling Mixtures

Following the same idea as above, if the task at hand is to cluster a finite collection of observations $(Y_1, \ldots, Y_n)$ and a species sampling mixture model is assumed for the underlying common random distribution Q, then we can cluster the observations by finding both $\mathbf{W} = (w_j)_{j\geq 1}$ and $\boldsymbol{\xi} = (\xi_j)_{j\geq 1}$ for which $E[Q(Y_1, \ldots, Y_n)|\mathbf{W}, \boldsymbol{\xi}]$ is maximized. Then Y_i will belong to the same cluster as Y_k if it shares the same index j for which $w_j K_{\xi_j}(Y_i)$ is maximized, it is

$$Y_i \in [k] \iff \text{argmax}_j\{w_j K_{\xi_j}(Y_i)\} = k,$$

where $[k]$ is an equivalence class and $\text{argmax}_j\{f_j\}$ is the index for which f_j is maximized. In order to remove ambiguity and have a real equivalence relation, if two indexes j, j' hold the maximum property for f_j, the smallest will be drawn.

In the next chapter a brief introduction to sampling algorithms needed for the implementation of the numerical illustrations will be given.

2.8 Sampling Algorithms

In the numerical illustrations chapter we will need to make use of the following sampling algorithms.

2.8.1 Inverse Transform Sampling

The inverse transform is a sampling method to sample a random observation from an arbitrary continuous probability distribution function on $\mathbb{R}$.

Theorem. *Let F be a continuous distribution function on $\mathbb{R}$ with inverse F^{-1} defined by*

$$F^{-1}(u) = \inf\{x : F(x) = u, 0 < u < 1\}.$$

If U is a Unif$(0,1)$ *random variable, then $F^{-1}(U)$ has distribution function F. Also, if X has distribution function F, then $F(X)$ is uniformly distributed on* $[0,1]$.

The following proof can be found in Devroye (2006) as Theorem 2.1.

Proof. The first statement follows after noting that for all $x \in \mathbb{R}$,

$$\begin{aligned} P(F^{-1}(U) \leq x) &= P(\inf\{y : F(y) = U\} = x) \\ &= P(U \leq F(x)) = F(x). \end{aligned}$$

The second statement follows from the fact that for all $0 < u < 1$,

$$\begin{aligned} P(F(X) \leq u) &= P(X \leq F^{-1}(u)) \\ &= F(F^{-1}(u)) = u. \end{aligned}$$

□

2.8.2 Gumbel-Max Trick

The Gumbel-Max trick is an algorithm that allows to sample from a finite categorical distribution over classes $i \in \{1, \ldots, n\}$ with probability proportional to $\exp(f_i)$.

Theorem (Gumbel-Max trick)**.** *Given a finite unnormalized categorical distribution $\{\exp(f_i)\}_{i=1}^n$, if $\{G_i\}_{i=1}^n$ are i.i.d.* Gumbel(0) *then,*

$$\text{argmax}_i\{G_i + f_i\} \sim \frac{\exp(f_i)}{\sum_i \exp(f_i)}$$

and for any $A \subseteq \{1, \ldots, n\}$,

$$\max_{i \in A}\{G_i + f_i\} \sim \text{Gumbel}\left(\log \sum_{i \in A} \exp(f_i)\right)$$

$$\operatorname*{argmax}_{i \in A}\{G_i + f_i\} \sim \frac{\exp(f_i)}{\sum_{i \in A} \exp(f_i)}$$

A proof can be found in the appendix of Maddison, Tarlow, and Minka (2014). We can see that the inverse transform sampling is already paying off since now it suffices to have $U \sim \text{Unif}(0,1)$ and transform it to get $-\log(-\log(U)) \sim \text{Gumbel}(0)$.

2.8.3 Gibbs Sampling

Gibbs sampling is a Monte Carlo simulation tool for obtaining samples from multidimensional random variables. The algorithm goes as follows (Robert and Casella, 2013). If for $n > 1$, the random variable $\boldsymbol{X} \in \mathbb{X}$ can be written as $\boldsymbol{X} = (X_1, \dots, X_n)$, where the X_i's are either uni- or multidimensional; and we can simulate from the corresponding *full conditional* densities $f_1, \dots, f_n$ given by

$$X_i | x_1, x_2 \dots, x_{i-1}, x_{i+1}, \dots, x_n \sim f_i(x_i | x_1, x_2, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$$

for $i = 1, 2, \dots, n$; then the associated *Gibbs sampling* algorithm is the iterative transition from $\boldsymbol{X}^{(t)}$ to $\boldsymbol{X}^{(t+1)}$ given by:

1. simulate $X_1^{(t+1)}$ from $f_1(x_1^{(t+1)} | x_2^{(t)}, \dots, x_n^{(t)})$,
2. simulate $X_2^{(t+1)}$ from $f_2(x_2^{(t+1)} | x_1^{(t+1)}, x_3^{(t)}, \dots, x_n^{(t)})$,

$\vdots$

n. simulate $X_n^{(t+1)}$ from $f_n(x_n^{(t+1)} | x_1^{(t+1)}, \dots, x_{n-1}^{(t+1)})$.

In this way, sampling the multi-dimensional vector $\boldsymbol{X}$ can been replaced by sampling the lower dimensional components X_i for $i = 1, 2, \dots, n$.

The idea behind the *Gibbs sampler* is to have a way to draw samples from $\boldsymbol{X}$ when there is no direct way of sampling or when finding the marginal densities is too expensive computationally. Since samples $\boldsymbol{X}^{(t)}$ approximate the distribution of $\boldsymbol{X}$ but have dependency to the starting point, it is customary to have a *burn-in* period of samples to be discarded.

Chapter 3

Dirichlet Geometric Process

The main focus of this chapter is to introduce the Dirichlet Geometric Process.

3.1 Definition and Properties

Definition (DGP)**.** The $\mathrm{DGP}(g, x, \theta, \alpha, \beta)$ or Dirichlet Geometric Process is a species sampling process where the sequence of locations are sampled independently from a common distribution g, and the sequence of weights are drawn via stick breaking the sequence $(v_j^{(x)})_{j\geq 1}$, where $v_j^{(x)}|p \sim \mathrm{Beta}\left(1 + \frac{x}{1-x}p, \theta + \frac{x}{1-x}(1-p)\right)$ for $x \in [0, 1)$, they are mutually independent conditioned on p, and for $x = 1$, $v_j^{(1)} = p$ for all $j \geq 1$, where $p \sim \mathrm{Beta}(\alpha, \beta)$.

The location parameter x allows us to construct a continuous transition from a DP to a GP as seen in the following two propositions.

Proposition 6. *The sticks* $\mathbf{V}^{(x)} = (v_j^{(x)})_{j\geq 1}$ *from a* $\mathrm{DGP}(g, x, \theta, \alpha, \beta)$ *are continuous in distribution with respect to* x *in the whole interval* $[0, 1]$*. It is,*

$$\lim_{x\to x_0} \left(v_j^{(x)}\right)_{j\geq 1} \overset{d}{=} \left(v_j^{(x_0)}\right)_{j\geq 1} \qquad \textit{for } x_0 \in [0, 1].$$

Proof. For $x_0 \in [0,1)$, the continuity follows directly from the definition. For the limiting case $x_0 = 1$, we have that if $x < 1$,

$$E\left(v_j^{(x)}\middle| p\right) = \frac{1+\frac{x}{1-x}p}{1+\theta+\frac{x}{1-x}},$$
$$\operatorname{Var}\left(v_j^{(x)}\middle| p\right) = \frac{(1+\frac{x}{1-x}p)(\theta+\frac{x}{1-x}(1-p))}{\left(1+\theta+\frac{x}{1-x}\right)^2\left(2+\theta+\frac{x}{1-x}\right)}.$$

Taking the limit on both sides of each equation we get,

$$\lim_{x\to1} E\left(v_j^{(x)}\middle| p\right) = p,$$
$$\lim_{x\to1} \operatorname{Var}\left(v_j^{(x)}\middle| p\right) = 0.$$

We also have that

$$\begin{aligned}\lim_{x\to1}\operatorname{Var}\left(v_j^{(x)}-p\right) &= \lim_{x\to1}\operatorname{Var}\left(E\left(v_j^{(x)}-p\middle| p\right)\right)\\ &\quad+\lim_{x\to1}E\left(\operatorname{Var}\left(v_j^{(x)}-p\middle| p\right)\right)\\ &=\operatorname{Var}\left(\lim_{x\to1}E\left(v_j^{(x)}-p\middle| p\right)\right)\\ &\quad+E\left(\lim_{x\to1}\operatorname{Var}\left(v_j^{(x)}-p\middle| p\right)\right)\\ &=0.\end{aligned}$$

Here we can swap the limits thanks to the bounded and dominated convergence theorems respectively since $|E(v_j^{(x)}-p|p)| \leq 1$ and $\operatorname{Var}(v_j^{(x)}-p|p)$ is a positive and decreasing function with respect to x. This implies that $\lim_{x\to1} v_j^{(x)}-p=0$ a.s., and also defines the limiting finite joint distribution $\lim_{x\to1}(v_1^{(x)},\ldots,v_j^{(x)}) = (p\ldots,p)$ a.s.. Finally since the finite-dimensional distributions of $\lim_{x\to1}(v_j^{(x)})_{j\geq1}$ and $(v_j^{(1)})_{j\geq1}$ are the same, using the finite-dimensional distributions proposition, also their full distribution is the same. □

To get some intuition, as x tends to one, the density of v_j tends to the Dirac's delta at p, it is, $f_{v_j^{(x)}}(v) \to \delta_p(v)$ as $x\to1$ as can be seen in Figure 3.1.

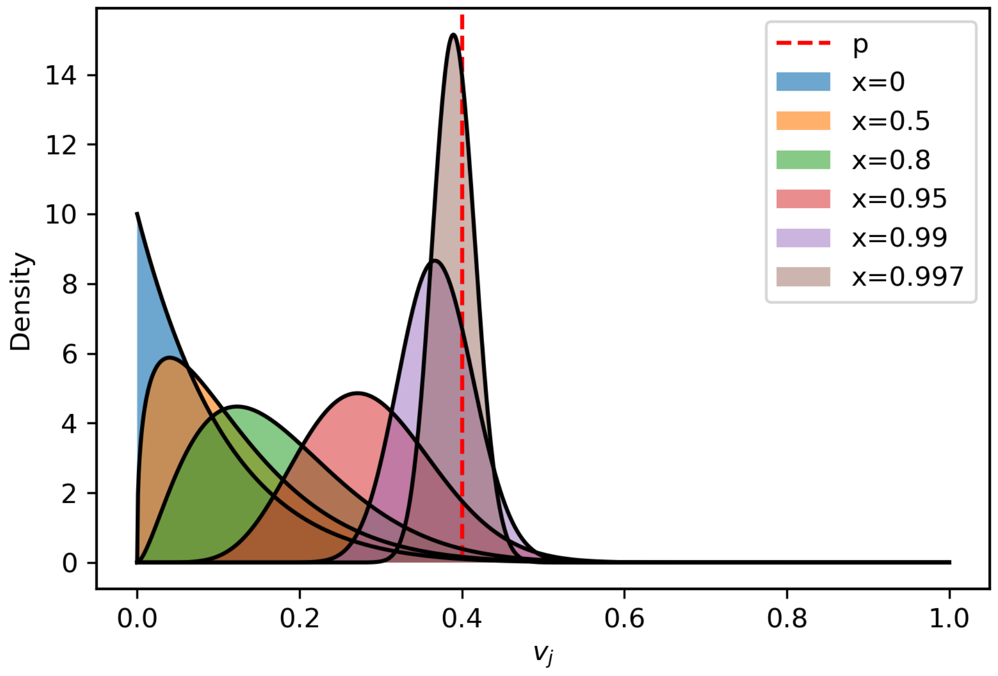

Figure 3.1: Conditional densities of v_j given $p = 0.4$ for distinct values of x. The value for θ is fixed to 10.

Proposition 7. *For a* $\mathrm{DGP}(g, x, \theta, \alpha, \beta)$*, the following properties hold:*

1. *A* $\mathrm{DGP}(g, 0, \theta, \alpha, \beta)$ *is a* $\mathrm{DP}(g, \theta)$*.*

2. *A* $\mathrm{DGP}(g, 1, \theta, \alpha, \beta)$ *is a* $\mathrm{GP}(g, \alpha, \beta)$*.*

3. *A* $\mathrm{DGP}(g, x, \theta, \alpha, \beta)$ *is continuous in distribution with respect to x in the whole interval* $[0, 1]$*. It is,*

$$\lim_{x \to x_0} \mathrm{DGP}(g, x, \theta, \alpha, \beta) \overset{d}{=} \mathrm{DGP}(g, x_0, \theta, \alpha, \beta) \qquad \textit{for } x_0 \in [0, 1].$$

Proof. Properties one and two are only a restatement of the DGP process definition for $x = 0$ and $x = 1$ respectively. For the third property, since the stick-breaking function $\mathcal{SB} : (x_1, x_2, \dots) \to (x_1, (1-x_1)x_2, \dots)$ is continuous, we have that $\lim_{x\to x_0} \mathbf{W}^{(x)} \overset{d}{=} \mathbf{W}^{(x_0)}$ and then $\lim_{x\to x_0} (\mathbf{W}^{(x)}, \boldsymbol{\xi}) \overset{d}{=} (\mathbf{W}^{(x_0)}, \boldsymbol{\xi})$ where $\mathbf{W}^{(x)} = \mathcal{SB}(\mathbf{V}^{(x)})$ is the random sequence of weights from stick-breaking $\mathbf{V}^{(x)}$, and $\boldsymbol{\xi}$ is the random sequence of locations.

We are only left to show that the function $(\mathbf{W}, \boldsymbol{\mu}) \to \sum_{j\geq 1} w_j \mu_j$ is continuous since it follows that the composition is also continuous.

Let $(\mathbf{W}_n)_{n\geq 1}, \mathbf{W}$ be elements of Δ_∞ and $(\boldsymbol{\mu}_n)_{n\geq 1}, \boldsymbol{\mu}$ be elements of $\mathcal{P}(\mathbb{X})^\infty$, such that $w_{n,j} \to w_j$ and $\mu_{n,j} \overset{w}{\to} \mu_j$ as $n \to \infty$ for every $j \geq 1$. Fix a continuous and bounded function $f : \mathbb{X} \to \mathbb{R}$. Then for $j \geq 1$, $w_{n,j}\mu_{n,j}(f) \to w_n \mu_n(f)$. Since f is bounded, there exists M such that $|f| \leq M$, hence $|w_{n,j}\mu_{n,j}(f)| \leq w_{n,j}\mu_{n,j}(|f|) \leq w_{j,n}M$, for every $n \geq 1$ and $j \geq 1$. Evidently, $w_{n,j}M \to w_j M$, and $\sum_{j\geq 1} w_{n,j}M = M = \sum_{j\geq 1} w_j M$. Hence by general Lebesgue dominated convergence theorem, we obtain

$$\sum_{j\geq 1} w_{n,j}\mu_{n,j}(f) \to \sum_{j\geq 1} w_j \mu_j(f).$$

Finally, using Portmanteau theorem it follows that the mapping is continuous. □

This leads us to the next corollary regarding ordering.

Corolary 8. *Let $\mathbf{W}^{(x)}$ be the sequence of weights from a* DGP$(g, x, \theta, \alpha, \beta)$. *Then*

1. *The sequence $\mathbf{W}^{(0)}$ is invariant under size-biased permutations.*

2. *For every $j \geq 1$,*
$$\lim_{x\to 1} P\left(w_{j+1}^{(x)} < w_j^{(x)}\right) = 1.$$

Proof. For the first point, the weights $\mathbf{W}^{(0)}$ arise from stick-breaking the sequence $\mathbf{V}^{(0)}$ of independent variables, each one distributed Beta$(1, \theta)$, which is proven to be invariant under size-biased permutations in Theorem 1 of Pitman (1996). For the second point, using the Portmanteau theorem, since $\mathbf{W}^{(x)}$ is continuous in distribution and $P(p \in \{0,1\}) = 0$, we have that

$$\begin{aligned}\lim_{x\to 1} P\left(w_{j+1}^{(x)} < w_j^{(x)}\right) &= P\left(w_{j+1}^{(1)} < w_j^{(1)}\right)\\ &= P\left(p(1-p)^j < p(1-p)^{j-1}\right)\\ &= P\left(0 < p < 1\right)\\ &= 1.\end{aligned}$$

$\square$

We can notice in Figure 3.2 that the variance of v_j decreases as x approaches one.

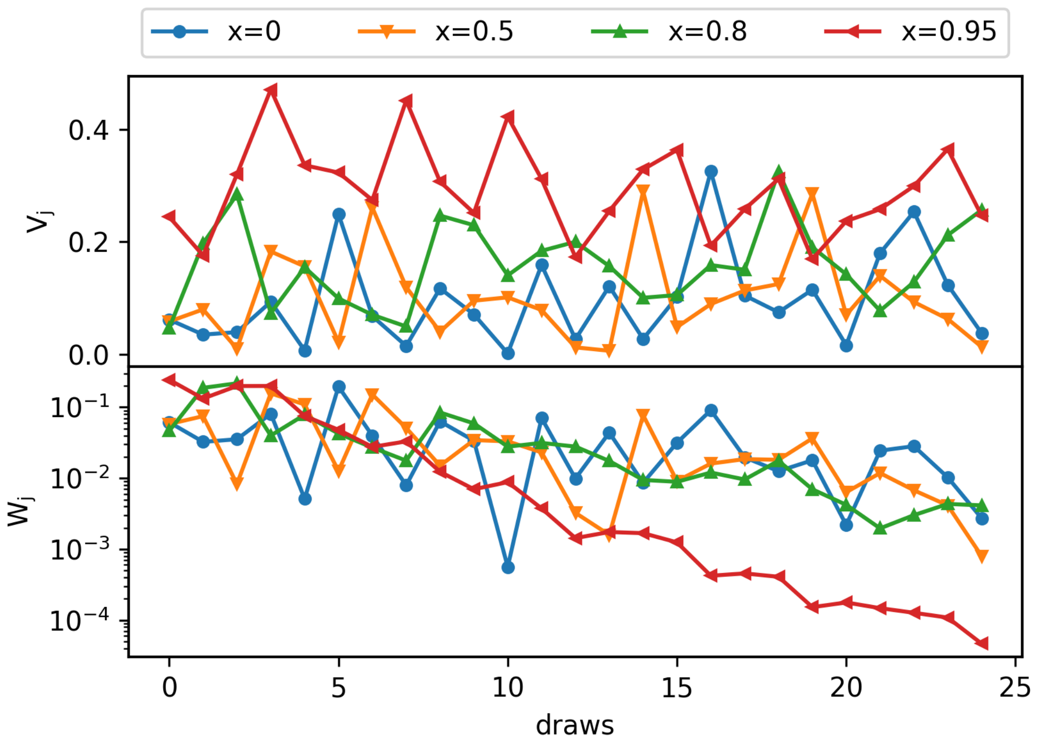

Figure 3.2: Simulations of $(v_j)_{j=1}^{25}$ and their respective $(w_j)_{j=1}^{25}$ where $p = 0.4$ and $\theta = 10$. Note: the scale for w_j is logarithmic.

In order to consider that this model is feasible for density estimation we need to prove two things. That it is a proper species sampling process so that we can model the analogous species sampling mixture, and that any distribution for the parameters in $\mathcal{P}$ can be drawn. It is, that a DGP has full support.

Proposition 9. *The* DGP$(g, x, \theta, \alpha, \beta)$ *is a proper species sampling process.*

Proof. In order to show that the DGP is a proper species sampling process we need to show that its weights sequence $\mathbf{W}^{(x)}$ satisfies $\sum_{j=1}^{\infty} w_j^{(x)} = 1$ a.s. which is equivalent to proving that $\prod_{j=1}^{k}(1 - v_j^{(x)}) \overset{a.s.}{\to} 0$ as $j \to \infty$, and since each of these r.v.'s are non-negative and bounded by 1 it suffices to show that

$$\lim_{k\to\infty} E\left(\prod_{j=1}^{k}\left(1 - v_j^{(x)}\right)\right) = 0.$$

We have that

$$\begin{aligned} E\left(\prod_{j=1}^{k}\left(1 - v_j^{(x)}\right)\right) &= E\left(E\left(\prod_{j=1}^{k}\left(1 - v_j^{(x)}\right)\middle| p\right)\right) \\ &= E\left(\prod_{j=1}^{k} E\left(1 - v_j^{(x)}\middle| p\right)\right) \\ &= E\left(E\left(1 - v_1^{(x)}\middle| p\right)^k\right), \end{aligned}$$

which tends to zero as $k \to \infty$ since $E\left(1 - v_1^{(x)}\middle| p\right) < 1$ a.s.. □

Proposition 10. *The* DGP$(g, x, \theta, \alpha, \beta)$ *has full support.*

Proof. For $x \in \{0, 1\}$, since it is either a DP or a GP, it has full support. For $x \in (0, 1)$, using Theorem 3 and following the same line as in Bissiri and Ongaro

(2014)[Proposition 7], let $\epsilon > 0$ and $\epsilon > \delta > 0$, we have that

$$\begin{aligned}
0 &< E\left[\textstyle\prod_{l=1}^{m} P\left(\delta < v_l < \epsilon | \, p\right)\right] \\
&= E[P\left(\{\delta < v_j < \epsilon : j = 1, \ldots, m\} | p\right)] \\
&= P\left(\{\delta < v_j < \epsilon : j = 1, \ldots, m\}\right) \\
&\leq P(w_1 < \epsilon, \ldots, w_m < \epsilon(1-\delta)^{m-1}, \textstyle\sum_{l=1}^{m} w_l > 1 - (1-\delta)^m) \\
&\leq P(w_1 < \epsilon, \ldots, w_m < \epsilon, \textstyle\sum_{l=1}^{m} w_l > 1 - (1-\delta)^m),
\end{aligned}$$

for every integer $m \geq 1$. In particular if $m > \log(\epsilon)/\log(1-\delta)$, then Theorem 3 condition 3 is satisfied. $\square$

Other topic of interest is the distribution of the number of different values drawn from n samples of a realization of the DGP. We will denote the number of unique values in $(Y_1, Y_2, \ldots)$ (or number of clusters) up to the n-th draw by K_n. It is possible to compute the distribution of K_n for some species sampling processes (Pitman, 2006), but not in general. In particular, the distribution of K_n for the DP (Watterson, 1974) and $E(K_n|p)$ for the GP (Mena and Walker, S. G., 2012) can be computed. This means that, for $x \in \{0, 1\}$, the expected number of clusters for the DGP$(g, x, \theta, \alpha, \beta)$ is computable.

For $x \in (0, 1)$ we can obtain samples of K_n via pseudoinverse transform sampling (Devroye, 2006), by generating n independent Unif$(0, 1)$, $\{u_i\}_{i=1}^{n}$ and for each u_i, sample w_j progressively until the first integer d_i such that $\sum_{j=1}^{d_i} w_j > u_i$. Then K_n equals the number of distinct values in $\{d_1, \ldots, d_n\}$. We can see a histogram over K_n values for different parameters of the GDP in Figure 3.3.

3.2 Mixture Conditional Distributions

Having the conditional distributions for the Dirichlet Geometric Mixture (DGM) will enable us to use a variety of numerical methods for computing $f_{Y_{n+1}|Y_1,\ldots,Y_n}(\cdot) =$

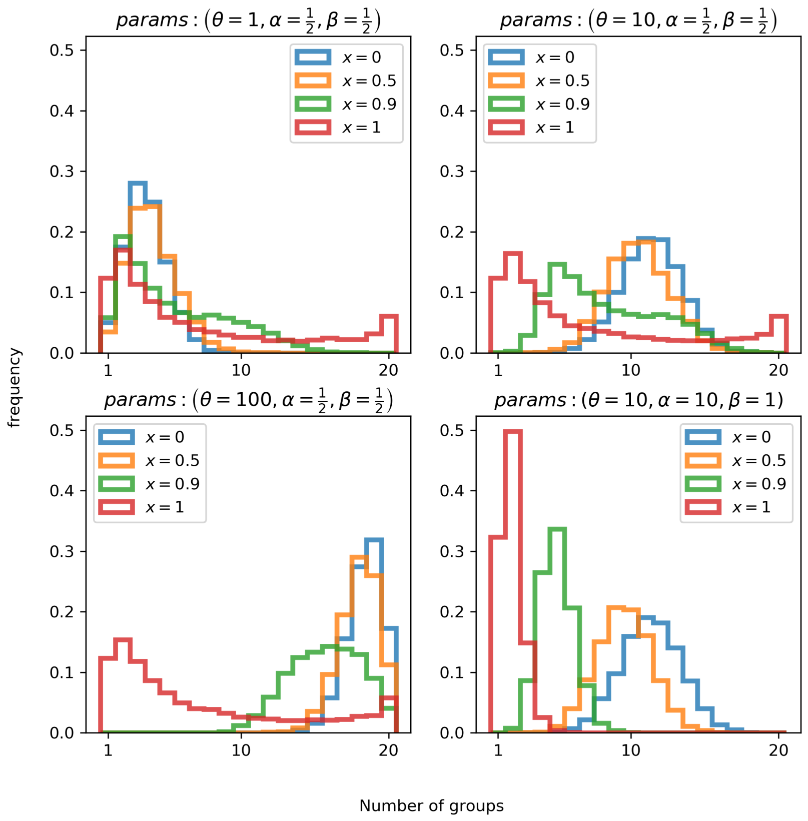

Figure 3.3: Histograms over 10,000 draws from K_n with different parameters.

$E[Q(\cdot)|Y_1, \ldots, Y_n]$. In particular we will explore the Gibbs sampler algorithm. Lets recall the representation for a species sampling mixture $Q = \sum_{j\geq 1} w_j K_{\xi_j}$, where the weights and parameters are random, so, using the notation $f(y) = f_Y(y)$ and $f(y|x) = f_{Y|X=x}(y)$ we have our first conditional distribution

$$f(y|\mathbf{W}, \boldsymbol{\xi}) = \sum_{j\geq 1} w_j K_{\xi_j}(y),$$

sampling from this mixture can be intimidating since there are infinitely many terms, but it is relatively simple if we follow the method from Kalli, M., Griffin, J. E., and

Walker, S. G. (2009) based on Gibbs sampling ideas. We introduce a latent variable u such that the joint density of (y, u) given $(\mathbf{W}, \boldsymbol{\xi})$ is

$$f(y, u|\mathbf{W}, \boldsymbol{\xi}) = \sum_{j\geq 1} \mathbf{1}(u < w_j) K_{\xi_j}(y).$$

And if we let

$$A(\mathbf{W}, u) = \{j : w_j > u\},$$

then we can write the previous joint density as

$$f(y, u|\mathbf{W}, \boldsymbol{\xi}) = \sum_{j\in A(\mathbf{W},u)} K_{\xi_j}(y),$$

with the added benefit that, since $A(\mathbf{W}, u)$ is a finite set for all $u > 0$, then $f(y|\mathbf{W}, \boldsymbol{\xi}, u)$ is also a finite mixture given by

$$f(y|\mathbf{W}, \boldsymbol{\xi}, u) = \frac{1}{|A(\mathbf{W}, u)|} \sum_{j\in A(\mathbf{W},u)} K_{\xi_j}(y).$$

Furthermore, this finite mixture is equally weighted. A simple approach to sample an equally weighted finite mixture is to introduce another latent variable d with uniform distribution in $A(\mathbf{W}, u)$ which will be the index of the distribution from which to sample Y, so that

$$f(d|\mathbf{W}, \boldsymbol{\xi}, u) = \frac{1}{|A(\mathbf{W}, u)|}\mathbf{1}(d \in A(\mathbf{W}, u)),$$

and

$$\begin{aligned} f(y, d, u|\mathbf{W}, \boldsymbol{\xi}) &= K_{\xi_d}(y)\mathbf{1}\left(d \in A(\mathbf{W}, u)\right) \\ &= K_{\xi_d}(y)\mathbf{1}\left(u < w_d\right), \end{aligned}$$

since the events $\{d \in A(\mathbf{W}, u)\}$ and $\{u < w_d\}$ are the same. We have that the conditioned and full likelihood functions based on a sample of size n, and with

$m \geq \max\{d_i | i \in \{1..n\}\}$, are

$$\ell\left(\{y_i, d_i, u_i\}_{i=1}^n | \mathbf{W}, \boldsymbol{\xi}\right) = \prod_{i=1}^n K_{\xi_{d_i}}(y_i) \mathbf{1}(u_i < w_{d_i}),$$

$$\begin{aligned}\ell\left(\{y_i, d_i, u_i\}_{i=1}^n, \{v_j, \xi_j\}_{j=1}^m, p\right) = &\prod_{i=1}^n K_{\xi_{d_i}}(y_i) \mathbf{1}(u_i < w_{d_i}) \\ &\times \prod_{j=1}^m \text{Beta}\left(v_j; 1 + \frac{x}{1-x}p, \theta + \frac{x}{1-x}(1-p)\right) \\ &\times \prod_{j=1}^m g(\xi_j) \times \text{Beta}(p; \alpha, \beta). \qquad (3.1)\end{aligned}$$

3.3 Posterior Inference

With the previous likelihood functions we can calculate the full conditional distributions and make posterior inference via a Gibbs sampler algorithm. The conditional distributions are in the following list which can also be seen as the sampling steps.

A. Update $\{u_i\}_{i=1}^n$. We have that

$$f(u_i | \dots) \propto \mathbf{1}(u_i < w_{d_i}),$$

so we only need to sample uniform distributions in $(0, w_{d_i})$.

B. Update $\mathbf{W}$. Lets first notice that we only need to sample a finite number of elements m such that $\sum_{j=1}^m w_j \geq \max_k(1 - u_k)$. If this condition is met, when updating d_i none will be greater than m. The conditional density of v_j is

$$\begin{aligned}f(v_j | v_{-j}, \dots) \propto &\text{Beta}\left(v_j; 1 + \frac{x}{1-x}p, \theta + \frac{x}{1-x}(1-p)\right) \\ &\times \prod_{i=1}^n \mathbf{1}\left(u_i < v_{d_i} \prod_{l<d_i}(1 - v_l)\right)\end{aligned}$$

with $v_{-j} = \{v_1, \dots, v_{j-1}, v_{j+1}, \dots, v_m\}$. Merging the product of indicator functions, the conditional density is proportional to

$$f(v_j | v_{-j}, \dots) \propto \text{Beta}\left(v_j; 1 + \frac{x}{1-x}p, \theta + \frac{x}{1-x}(1-p)\right) \mathbf{1}\left(\alpha_j < v_j < \beta_j\right)$$

where

$$\alpha_j = \max_{d_i = j} \left\{ \frac{u_i}{\prod_{l<j}(1-v_l)} \right\}$$

and

$$\beta_j = 1 - \max_{d_i > j} \left\{ \frac{u_i(1-v_j)}{v_{d_i} \prod_{l<d_i}(1-v_l)} \right\},$$

so we only need to sample from a truncated beta.

C. Update $\boldsymbol{\xi}$. Here it suffices to sample as many elements as we sampled in the previous step with conditional density function

$$f(\xi_j | \dots) \propto g(\xi_j) \prod_{d_i = j} K_{\xi_j}(y_i).$$

This can be done easily if g and K form a conjugate pair, for example, if K is a member of the normal family and g is the normal-gamma distribution.

D. Update $\{d_i\}_{i=1}^n$. This are finite discrete random variables with full conditional

$$f(d_i | \dots) \propto K_{\xi_{d_i}}(y_i) \mathbf{1}(u_i < w_{d_i}),$$

for numerical reasons it may be better to work with the log-probabilities and sample with the Gumbel-Max trick.

E. Update p. The density function is

$$\begin{aligned} f(p | \dots) \propto & \left(\prod_{j=1}^m \text{Beta}\left(v_j; 1 + \frac{x}{1-x}p, \theta + \frac{x}{1-x}(1-p)\right) \right) \text{Beta}(p; \alpha, \beta) \\ \propto & \Gamma\left(1 + \frac{x}{1-x}p\right)^m \Gamma\left(c + \frac{x}{1-x}(1-p)\right)^m \\ & \times \exp\left(xp \sum_{j=1}^m \log\left(\frac{v_j}{1-v_j}\right)\right) p^{\alpha-1}(1-p)^{\beta-1} \mathbf{1}(0 < p < 1), \end{aligned}$$

from which we can sample using inverse transform sampling by numerically normalizing and inverting, or by numerically getting the maximum and doing an acceptance rejection sampling.

3.3.1 Posterior Distribution Analysis

We can estimate the posterior distribution of the data with the Expected A Posteriori (EAP) of the random distribution $\{Q|\{y_i\}_{i=1}^n\}$, by running T iterations of the Gibbs sampler to get the samples $\{\boldsymbol{\xi}_t, \mathbf{W}_t, \{u_{t,i}\}_{i=1}^n\}_{t=1}^T$ and then calculate

$$\begin{aligned} E[Q(\cdot)|\{y_i\}_{i=1}^n] \approx & \frac{1}{T}\sum_{t=1}^T E[Q(\cdot)|\boldsymbol{\xi}_t, \mathbf{W}_t, \{u_{t,i}\}_{i=1}^n] \\ = & \frac{1}{T}\sum_{t=1}^T \frac{1}{n}\sum_{i=1}^n E[Q(\cdot)|\boldsymbol{\xi}_t, \mathbf{W}_t, u_{t,i}] \\ = & \frac{1}{T}\sum_{t=1}^T \frac{1}{n}\sum_{i=1}^n \frac{1}{|A(\mathbf{W}_t, u_{t,i})|} \sum_{j\in A(\mathbf{W}_t, u_{t,i})} K_{\xi_{t,j}}(\cdot), \end{aligned}$$

where $A(\mathbf{W}_t, u_{t,i}) = \{j : w_{t,j} > u_{t,i}\}$, and the dependence to $\{y_i\}_{i=1}^n$ is implicit from the variables $\{\boldsymbol{\xi}_t, \mathbf{W}_t, \{u_{t,i}\}_{i=1}^n\}_{t=1}^T$.

3.3.2 Posterior Cluster Analysis

To cluster the data points $\{y_i\}_{i=1}^n$, we can store the likelihood value for each step of the Gibbs sampler and find t^*, the step with maximum likelihood (3.1) within the sample,

$$t^* = \operatorname{argmax}_t \left\{ \ell\left(\{y_i, d_{t,i}, u_{t,i}\}_{i=1}^n, \{v_{t,j}, \xi_{t,j}\}_{j=1}^{m_t}, p_t\right)\right\},$$

to approximate the Maximum A Posteriori (MAP) estimate of the clusters via the mixture components

$$\left\{ \frac{1}{n}\sum_{i=1}^n \frac{\mathbf{1}_{j\in A(\mathbf{W}_{t^*}, u_{t^*,i})}}{|A(\mathbf{W}_{t^*}, u_{t^*,i})|} K_{\xi_{t,j}}(\cdot) \right\}_{j=1}^{m_{t^*}},$$

so that a data point y will belong to the cluster index k_y given by

$$k_y = \operatorname{argmax}_j \left\{ \left| \frac{1}{n} \sum_{i=1}^{n} \frac{\mathbf{1}_{j \in A(\mathbf{W}_{t^*}, u_{t^*,i})}}{|A(\mathbf{W}_{t^*}, u_{t^*,i})|} K_{\xi_{t,j}}(y) \right| \right\}.$$

Then, two observations $\{y_1, y_2\}$ belong to the same cluster if their respective cluster index is the same, $k_{y_1} = k_{y_2}$, where the common index value is unimportant, it is, the index does not denotes ordering of the groups nor any other property.

We can also estimate the posterior distribution of the number of groups $\{K_n | \{y_i\}_{i=1}^n\}$ with the EAP:

$$P[K_n | \{y_i\}_{i=1}^n] \approx \frac{1}{T} \sum_{t=1}^{T} \mathbf{1}_{\{m\}}(K_n^t).$$

where K_n^t is the number of distinct elements of $\{d_{t,i}\}_{i=1}^n$.

Chapter 4

Numerical Illustrations

In this chapter we will do some analysis on a couple of practical examples regarding density estimation and clustering with the DGP and do a comparison against other methods. For both sections lets assume that $\xi = (\mu, \tau)$, $K_\xi(y) = \text{Normal}(y; \mu, 1/\tau)$ and $g(\xi) = \text{NormalGamma}(\xi; \text{m}, \lambda, \text{a}, \text{b})$. This choice of distributions form a conjugate pair and so we have that

$$f(\xi_j|\dots) = \text{NormalGamma}\left(\xi_j; \frac{\lambda \text{m} + n_j \text{m}_j}{\lambda + n_j}, \lambda + n_j, \right. \\ \left. \text{a} + \frac{n_j}{2}, \text{b} + \frac{1}{2}\left(s_j + \frac{\lambda n_j (\text{m}_j - \text{m})^2}{2}\right)\right),$$

where

$$n_j := \sum_{d_i = j} 1, \qquad \text{m}_j := n_j^{-1} \sum_{d_i = j} y_i, \qquad s_j := \sum_{d_i = j} (y_i - \text{m}_j)^2,$$

and for the multidimensional case which will be introduced in the clustering section lets suppose that $\xi = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $K_\xi(\mathbf{y}) = \text{Normal}(\mathbf{y}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ and the prior for ξ is the NIW (Normal-inverse-Wishart) distribution, it is $g(\xi) = \text{NIW}(\xi; \boldsymbol{\mu}, \lambda, \boldsymbol{\Psi}, \nu)$. This two distributions also form a conjugate pair and the posterior distribution is given by

$$f(\xi_j|\dots) = \text{NIW}\left(\xi; \boldsymbol{\mu}_j, \lambda_j, \boldsymbol{\Psi}_j, \nu_j\right)$$

where

$$
\begin{aligned}
n_j &:= \sum_{d_i=j} 1, \\
\mathbf{m}_j &:= n_j^{-1} \sum_{d_i=j} \mathbf{y}_i, \\
\mathbf{S}_j &:= \sum_{d_i=j} (\mathbf{y}_i - \mathbf{m}_j)(\mathbf{y}_j - \mathbf{m}_j)^T, \\
\boldsymbol{\mu}_j &:= \frac{\lambda \boldsymbol{\mu} + n_j \mathbf{m}_j}{\lambda + n_j}, \\
\lambda_j &:= \lambda + n_j, \\
\nu_j &:= \nu + n_j, \\
\boldsymbol{\Psi}_j &:= \boldsymbol{\Psi} + \mathbf{S} + \frac{\lambda n_j}{\lambda + n_j} (\mathbf{m}_j - \boldsymbol{\mu})(\mathbf{m}_j - \boldsymbol{\mu})^T.
\end{aligned}
$$

4.1 Density Estimation with the DGP

We will begin with a straightforward example generating 200 independent data points $\{y_i\}_{i=1}^{200}$ from a Normal$(0, 1)$ distribution and fitting the density with the DGM, KDE, DPM and GPM models. The hyperparameters for the DGM were $(x, \theta, \alpha, \beta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(\frac{1}{2}, 1, 1, 1, 0, 1, \frac{1}{2}, \frac{1}{2}\right)$, for the KDE the bandwidth was $h = 1$, for the DPM $(\theta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(1, 0, 1, \frac{1}{2}, \frac{1}{2}\right)$ and for the GPM $(\alpha, \beta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(1, 1, 0, 1, \frac{1}{2}, \frac{1}{2}\right)$. We used $10,000$ steps of the Gibbs sampler as the burn-in period and $40,000$ iterations for the estimates.

The fitted densities are shown in Figure 4.1, where we can see that the DGM, DPM and GPM did find close estimates to the true density, while KDE overestimated the variance, tough this behavior can be tuned with the bandwidth parameter.

Figure 4.2 shows the evolution over time of the rolling mean number of groups $K_{t,n}$, in which we can see that it stabilizes over time.

For the second example we simulated 240 independent data points $\{y_i\}_{i=1}^{240}$ from a normal mean-variance mixture of six normal distributions with weights $(0.17, 0.08, 0.125, 0.29, 0.125, 0.21)$ and mean-variance parameters given by $(-18, 2)$, $(-5, 1)$, $(0, 1)$, $(6, 1)$, $(14, 1)$, and $(23, 1.25)$. The hyperparameters for the DGM were $(x, \theta, \alpha, \beta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(\frac{1}{2}, 1, 1, 1, 0, 0.01, \frac{1}{2}, \frac{1}{2}\right)$, for the KDE the bandwidth was $h = 1$, for the DPM $(\theta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(1, 0, 0.01, \frac{1}{2}, \frac{1}{2}\right)$ and for the GPM $(\alpha, \beta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(1, 1, 0, 0.01, \frac{1}{2}, \frac{1}{2}\right)$. We used $10,000$ steps of the Gibbs sampler as the burn-in period and $40,000$ iterations for the estimates.

Figure 4.3 shows the fitted densities for the second example. Here we can see that KDE did a better job than in the previous example but the variance in all but the first mixture component is still overestimated. In this example, decreasing the bandwidth to fit the variance of other components also reduces the variance of the first one so there is no right solution with KDE. The DGM, DPM and GPM models can estimate the variance for each component separately. Figure 4.4 shows the evolution over time of the rolling mean number of groups $K_{t,n}$, after which we can see that it has stabilized. This time the GPM mean number of groups is greater than that of the DGM and DPM.

For the second example we also show the rolling mean number of weights $w_{t,j}$ that are greater than 0.5 in Figure 4.5, and the evolution of the density estimates over several runs of the Gibbs sampler in Figure 4.6.

As a third and last example of density estimation we use the “galaxy” data set: the observed velocities in km/s of 82 galaxies in the Corona Borealis region. For this database we used the following hyperparameters. For the DGM $(x, \theta, \alpha, \beta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(\frac{1}{2}, 1, 1, 1, 0, 0.01, \frac{1}{2}, \frac{1}{2}\right)$, for the KDE the bandwidth was $h = 1$, for the DPM $(\theta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(1, 0, 0.01, \frac{1}{2}, \frac{1}{2}\right)$ and for the GPM $(\alpha, \beta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(1, 1, 0, 0.01, \frac{1}{2}, \frac{1}{2}\right)$. We used $10,000$ steps of the Gibbs sampler as the burn-in period and $40,000$ iterations for the estimates.

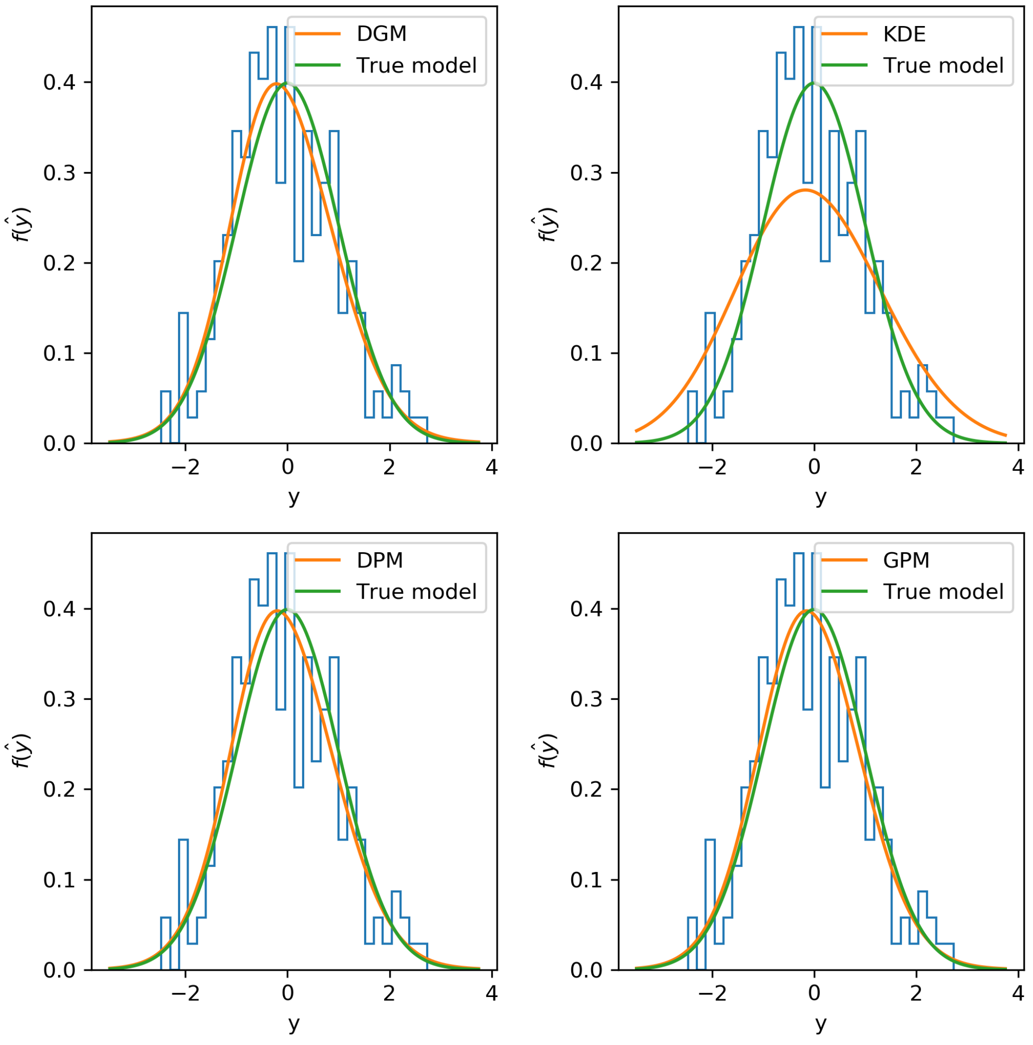

Figure 4.1: Density estimation of 200 i.i.d observations from a Normal$(0, 1)$ distribution using DGM, KDE, DPM and GPM models with a burn-in period of 10,000 iterations and 40,000 iterations for the estimates in the Gibbs sampler. The hyperparameters for the DGM were $(x, \theta, \alpha, \beta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(\frac{1}{2}, 1, 1, 1, 0, 1, \frac{1}{2}, \frac{1}{2}\right)$, for the KDE the bandwidth was $h = 1$, for the DPM $(\theta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(1, 0, 1, \frac{1}{2}, \frac{1}{2}\right)$ and for the GPM $(\alpha, \beta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(1, 1, 0, 1, \frac{1}{2}, \frac{1}{2}\right)$.

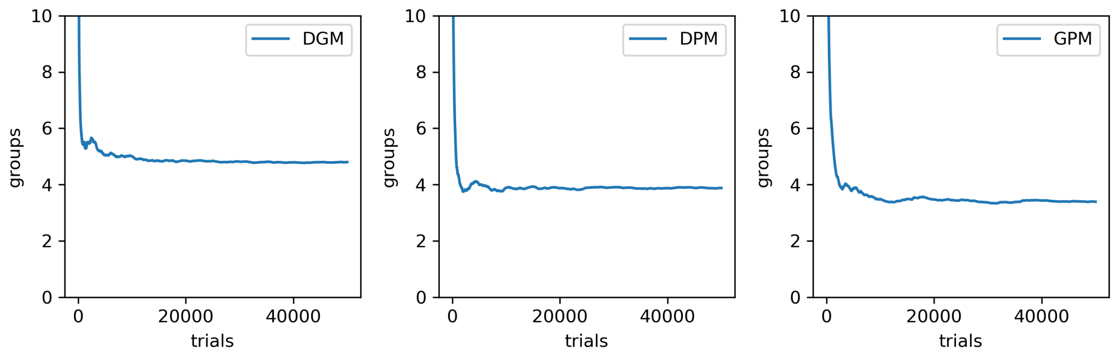

Figure 4.2: Cumulative mean number of groups $(K_{t,n})$ in a database of 200 i.i.d. observations from a Normal $(0,1)$ distribution after t iterations of the Gibbs sampler.

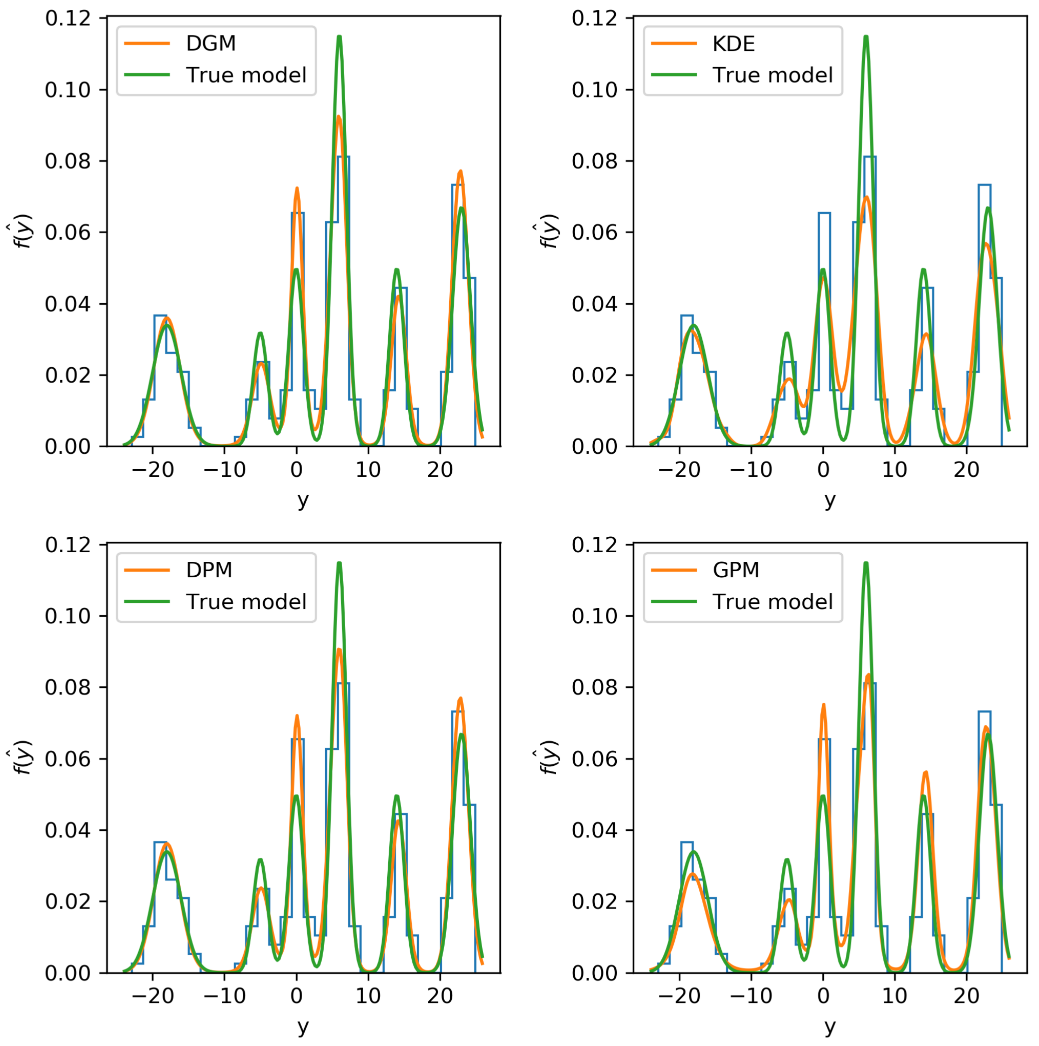

Figure 4.3: Density estimation of 240 independent data points from a normal mean-variance mixture of six normal distributions using DGM, KDE, DPM and GPM models with a burn-in period of 10,000 iterations and 40,000 iterations for the estimates in the Gibbs sampler. The hyperparameters for the DGM were $(x, \theta, \alpha, \beta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(\frac{1}{2}, 1, 1, 1, 0, 0.01, \frac{1}{2}, \frac{1}{2}\right)$, for the KDE the bandwidth was $h = 1$, for the DPM $(\theta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(1, 0, 0.01, \frac{1}{2}, \frac{1}{2}\right)$ and for the GPM $(\alpha, \beta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(1, 1, 0, 0.01, \frac{1}{2}, \frac{1}{2}\right)$.

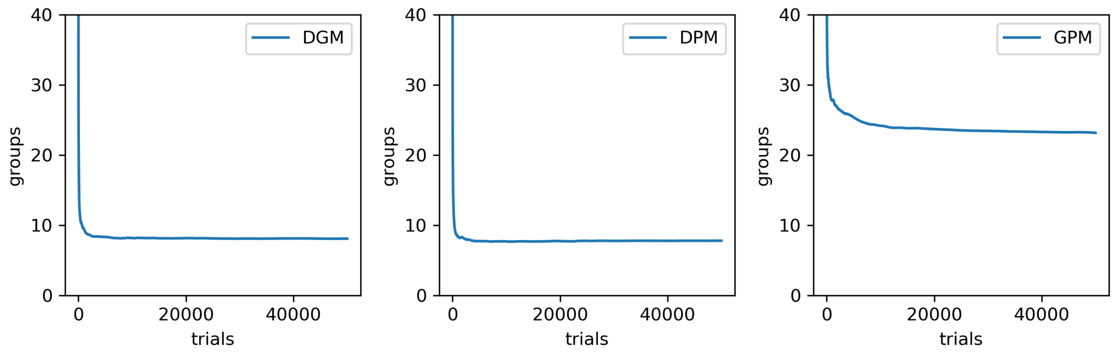

Figure 4.4: Cumulative mean number of groups ($K_{t,n}$) in a database of 240 independent data points from a normal mean-variance mixture of six normal distributions after t iterations of the Gibbs sampler.

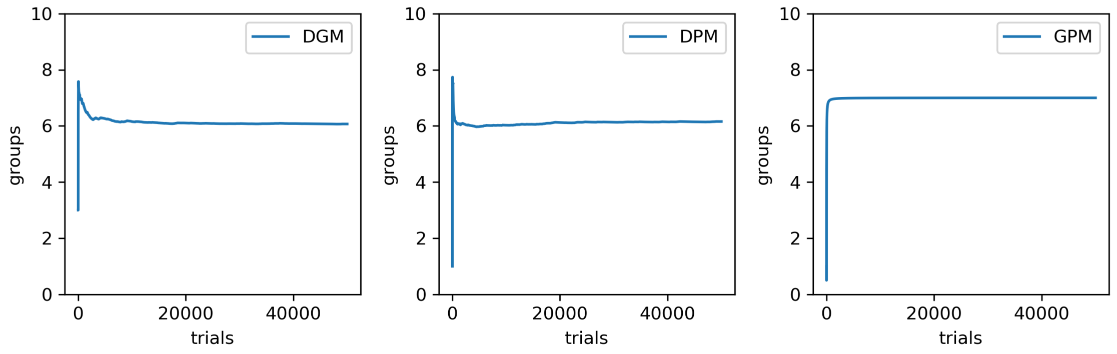

Figure 4.5: Cumulative mean number of weights $w_{t,j}$ greater than 0.05 in a database of 240 independent data points from a normal mean-variance mixture of six normal distributions after t iterations of the Gibbs sampler.

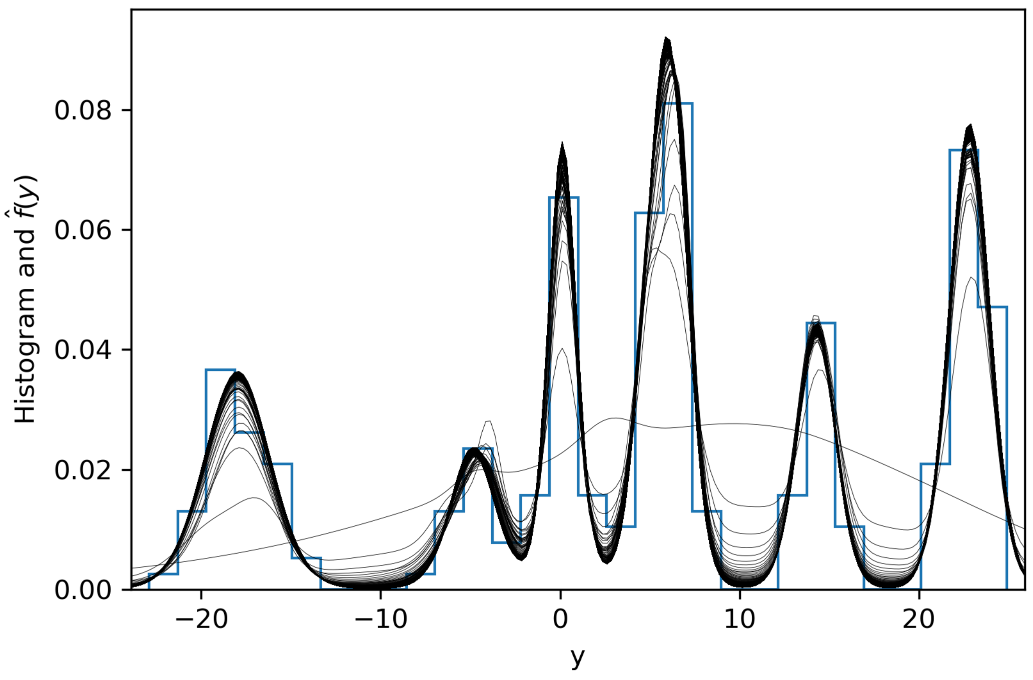

Figure 4.6: Evolution of the density estimates of 240 independent data points from a normal mean-variance mixture of six normal distributions using the DGM model with up to 50,000 iterations of the Gibbs sampler. The hyperparameters for the DGM were $(x, \theta, \alpha, \beta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(\frac{1}{2}, 1, 1, 1, 0, 0.01, \frac{1}{2}, \frac{1}{2}\right)$, for the KDE the bandwidth was $h = 1$, for the DPM $(\theta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(1, 0, 0.01, \frac{1}{2}, \frac{1}{2}\right)$ and for the GPM $(\alpha, \beta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(1, 1, 0, 0.01, \frac{1}{2}, \frac{1}{2}\right)$.

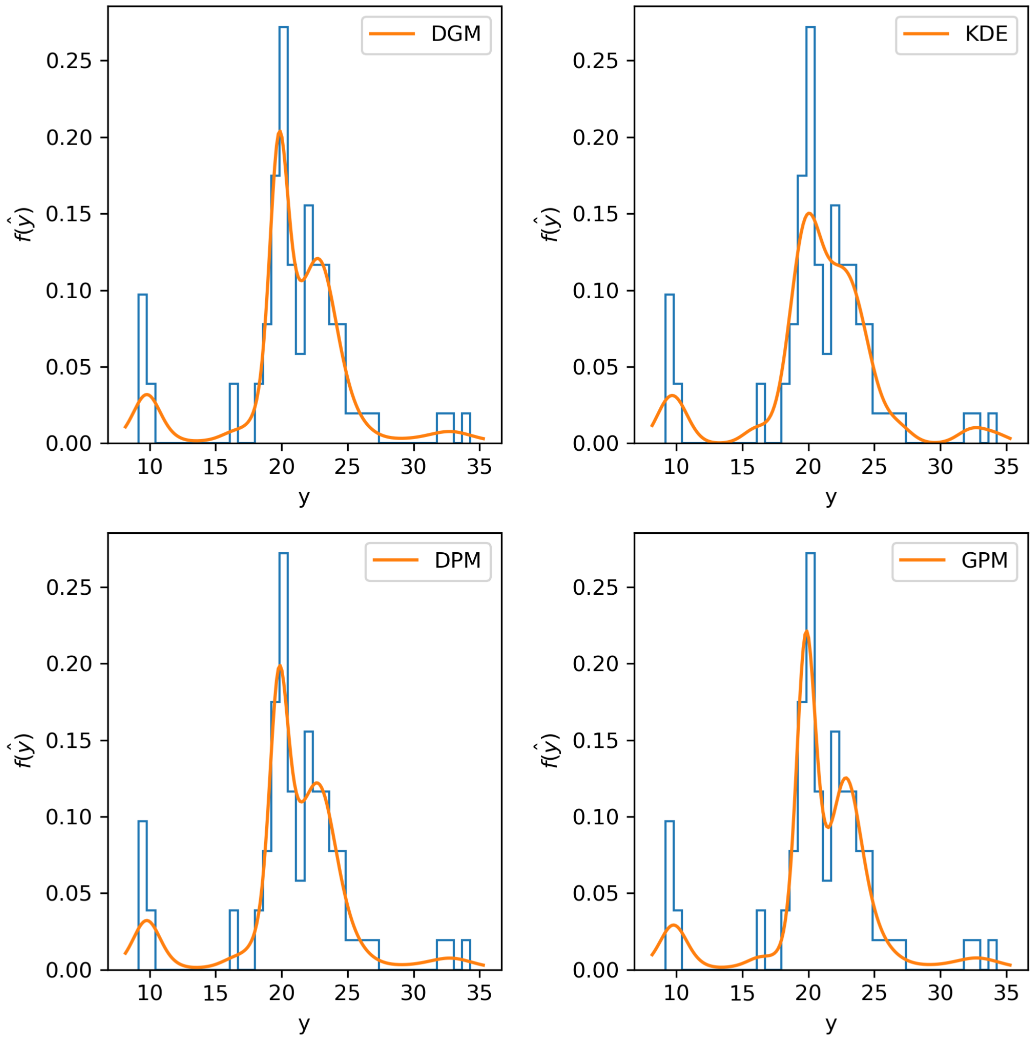

Figure 4.7: Density estimation for the velocity of galaxies in the Corona Borealis region using DGM, KDE, DPM and GPM models with a burn-in period of 10,000 iterations and 40,000 iterations for the estimates in the Gibbs sampler. The hyper-parameters for the DGM were $(x, \theta, \alpha, \beta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(\frac{1}{2}, 1, 1, 1, 0, 0.01, \frac{1}{2}, \frac{1}{2}\right)$, for the KDE the bandwidth was $h = 1$, for the DPM $(\theta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(1, 0, 0.01, \frac{1}{2}, \frac{1}{2}\right)$ and for the GPM $(\alpha, \beta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(1, 1, 0, 0.01, \frac{1}{2}, \frac{1}{2}\right)$.

4.2 Clustering with the DGP

In this section we will use and compare several methods for clustering three data sets.

As our first example we use the same database from the previous section given by 240 independent data points $\{y_i\}_{i=1}^{240}$ from a normal mean-variance mixture of six normal distributions with weights $(0.17, 0.08, 0.125, 0.29, 0.125, 0.21)$ and mean-variance parameters given by $(-18, 2)$, $(-5, 1)$, $(0, 1)$, $(6, 1)$, $(14, 1)$, and $(23, 1.25)$. The hyperparameters for the DGM were $(x, \theta, \alpha, \beta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(\frac{1}{2}, 1, 1, 1, 0, 0.01, \frac{1}{2}, \frac{1}{2}\right)$, for the DPM $(\theta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(1, 0, 0.01, \frac{1}{2}, \frac{1}{2}\right)$ and for the GPM $(\alpha, \beta, \mathrm{m}, \lambda, \mathrm{a}, \mathrm{b}) = \left(1, 1, 0, 0.01, \frac{1}{2}, \frac{1}{2}\right)$. We used $10,000$ steps of the Gibbs sampler as the burn-in period and $40,000$ iterations for the estimates.

Figure 4.8 shows an histogram of our database, the scaled densities of each cluster and the data points. We can appreciate visually the way a GPM approximates densities without the need of a flexible weighting structure: by assigning many components with similar parameters as can be seen in the left- and rightmost components. The DGM and DPM get to very similar clusters thanks to a more flexible weighting structure.

In Figure 4.9 we show the histogram over K_n, the number of groups for this example. The DGM and DPM show a similar distribution in the number of clusters, while in the GPM case more groups are needed.

For our second example we simulated 300 independent data points $\{y_i\}_{i=1}^{240}$ from a multivariate normal mean-variance mixture of three multivariate normal distributions with weights $(0.4, 0.3, 0.3)$, means $((0, -4), (3, 3), (-3, 2))$, and variance-covariance matrices $\left(\left(\begin{smallmatrix} 1 & 0.7 \\ 0.7 & 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 & -0.6 \\ -0.6 & 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)\right)$.

The hyperparameters used for the DGM were $(x, \theta, \alpha, \beta, \boldsymbol{\mu}, \lambda, \boldsymbol{\Psi}, \nu) = (\frac{1}{2}, 1, 1, 1, (0, 0), 1, \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right), 2)$, for the DPM $(x, \theta, \boldsymbol{\mu}, \lambda, \boldsymbol{\Psi}, \nu) = \left(\frac{1}{2}, 1, 1, 1, (0, 0), 1, \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right), 2\right)$ and for the GPM $(x, \alpha, \beta, \boldsymbol{\mu}, \lambda, \boldsymbol{\Psi}, \nu) = \left(\frac{1}{2}, 1, 1, 1, (0, 0), 1, \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right), 2\right)$. We used $50,000$

steps of the Gibbs sampler.

Figure 4.10 shows the fitted clusters, and Figure 4.11 the histogram over the number of groups for this example. Both the DGM and DPM agree on the number of clusters while the GPM assigns a different group to distant points, and, even though k-means seems to have done a better job in finding the original mixture components, the number of clusters was given as an input.

As our last example we chose to use the "faithful" data set, which registers 272 observations of the waiting time between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA.

The hyperparameters used for the DGM were $(x, \theta, \alpha, \beta, \boldsymbol{\mu}, \lambda, \boldsymbol{\Psi}, \nu) = (\frac{1}{2}, 1, 1, 1, \bar{y}, 1, \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right), 2)$, for the DPM $(x, \theta, \boldsymbol{\mu}, \lambda, \boldsymbol{\Psi}, \nu) = \big(\frac{1}{2}, 1, 1, 1, \bar{y}, 1, \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right), 2\big)$ and for the GPM $(x, \alpha, \beta, \boldsymbol{\mu}, \lambda, \boldsymbol{\Psi}, \nu) = \big(\frac{1}{2}, 1, 1, 1, \bar{y}, 1, \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right), 2\big)$; where $\bar{y} = n^{-1} \sum_{i=1}^{n} y_i$. We used $50,000$ steps of the Gibbs sampler.

Figure 4.12 shows the fitted clusters and Figure 4.13 the histogram over the number of groups. In this example the DPM and k-means agree on the number of clusters, but the groups in k-means are skewed in an unintuitive aggregation. The DGM differs from the DPM for the group at the center due to a less flexible weighting structure, meanwhile the GPM struggles with its more rigid weighting structure and ends finding elongated groups in the upper right region to accommodate a bigger number of components.

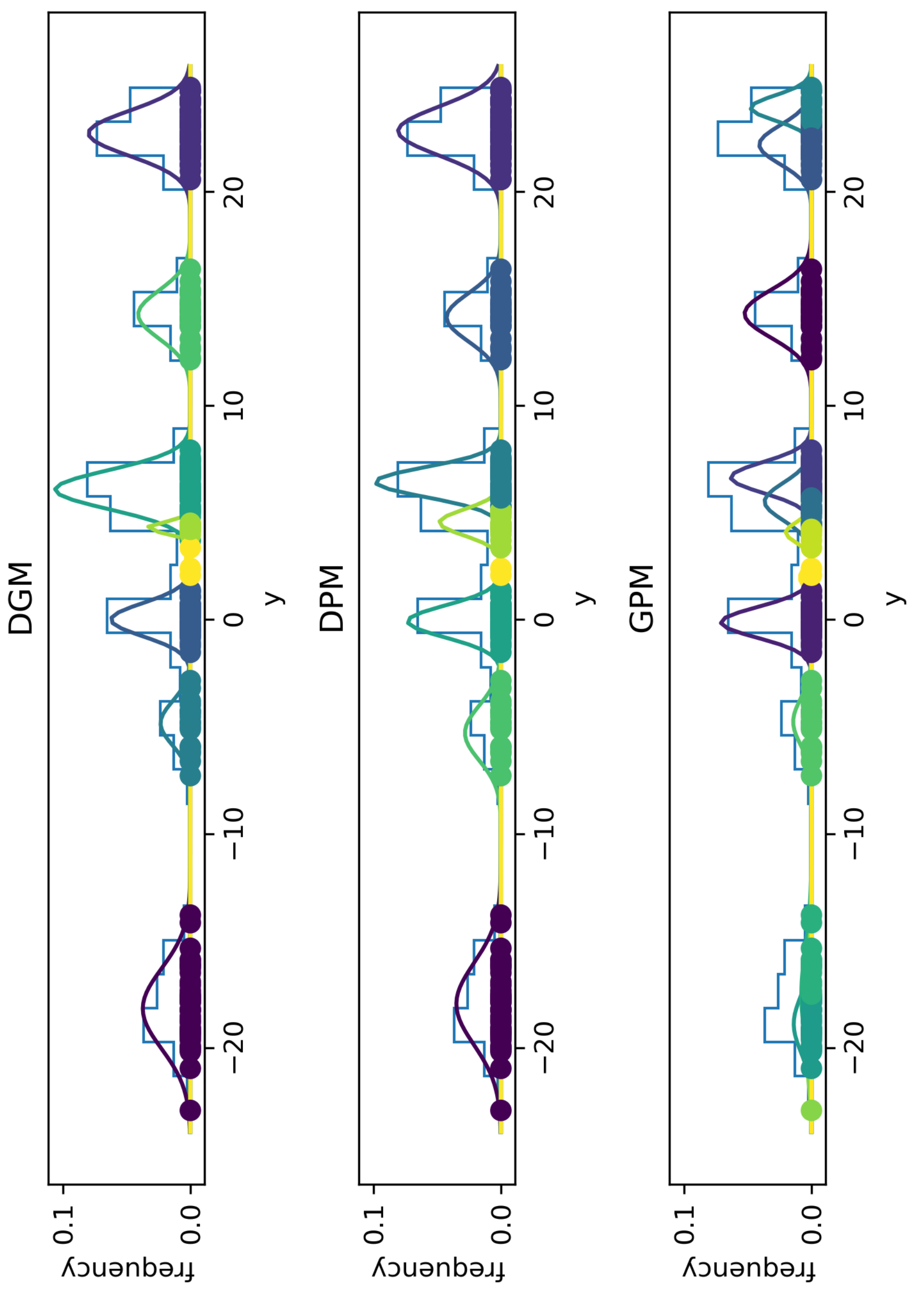

Figure 4.8: Histogram, density of the fitted components to scale, and arising clusters, of a database given by 240 independent data points from a normal mean-variance mixture of six normal distributions using DGM, DPM and GPM models.

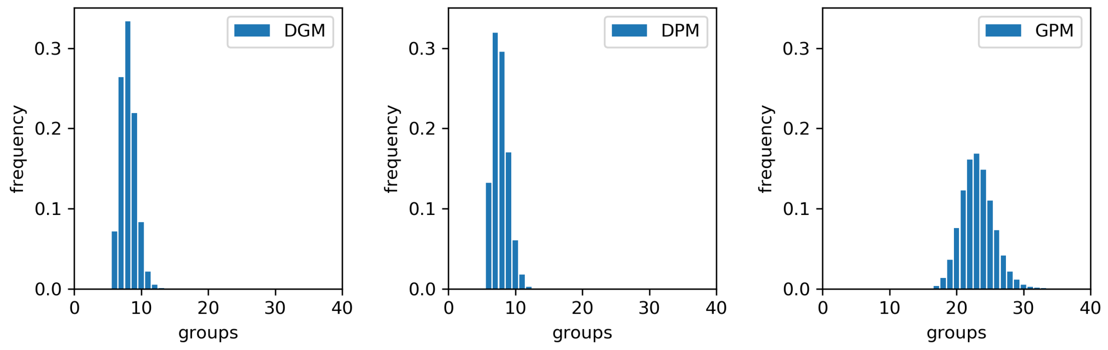

Figure 4.9: Histogram on the number of groups K_n of a database given by 240 independent data points from a normal mean-variance mixture of six normal distributions using DGM, DPM and GPM models.

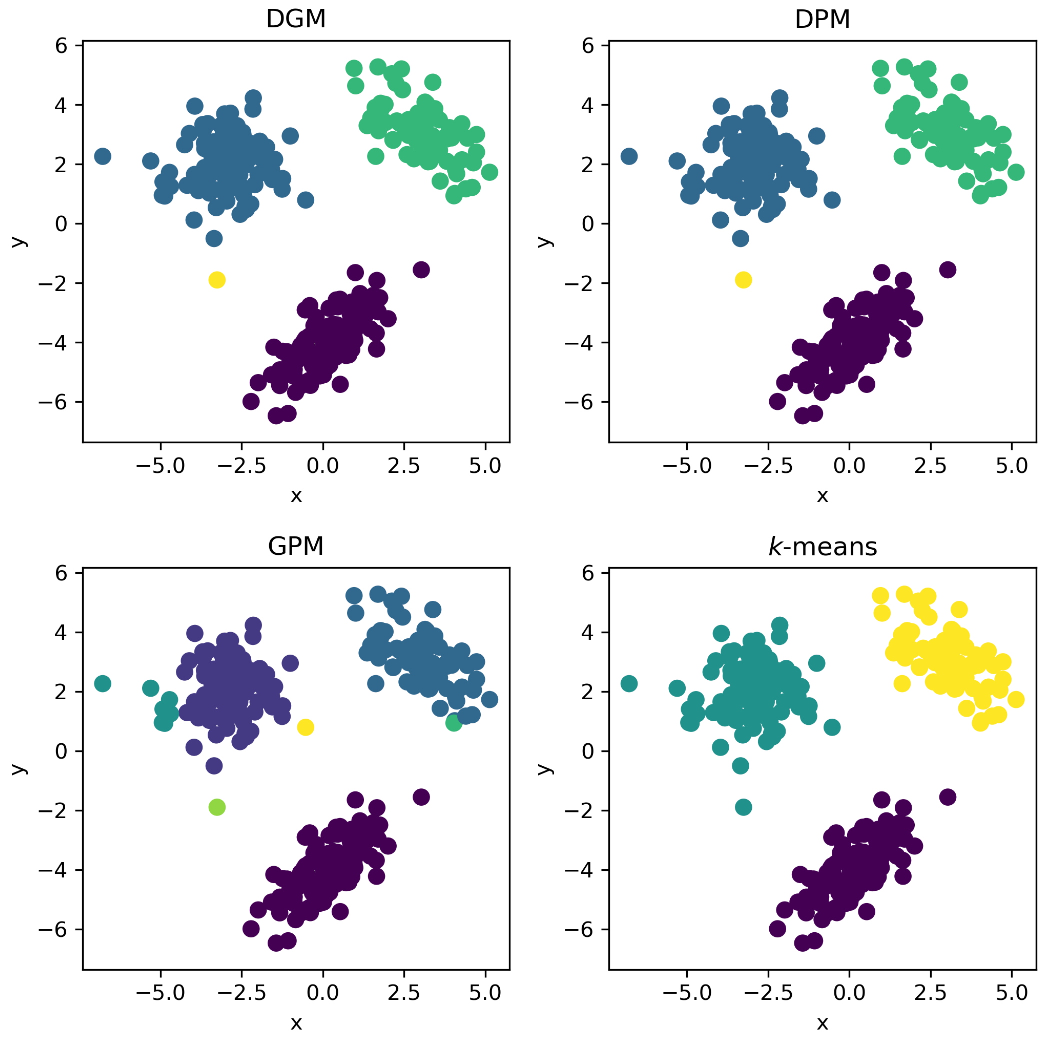

Figure 4.10: Fitted clusters in a database given by 300 independent data points from a multivariate normal mean-variance mixture of three multivariate normal distributions using DGM, DPM and GPM models after 50,000 steps of the Gibbs sampler. The hyperparameters used for the DGM were $(x, \theta, \alpha, \beta, \boldsymbol{\mu}, \lambda, \boldsymbol{\Psi}, \nu) = (\frac{1}{2}, 1, 1, 1, (0,0), 1, \left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right), 2)$, for the DPM $(x, \theta, \boldsymbol{\mu}, \lambda, \boldsymbol{\Psi}, \nu) = \left(\frac{1}{2}, 1, 1, 1, (0,0), 1, \left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right), 2\right)$ and for the GPM $(x, \alpha, \beta, \boldsymbol{\mu}, \lambda, \boldsymbol{\Psi}, \nu) = \left(\frac{1}{2}, 1, 1, 1, (0,0), 1, \left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right), 2\right)$.

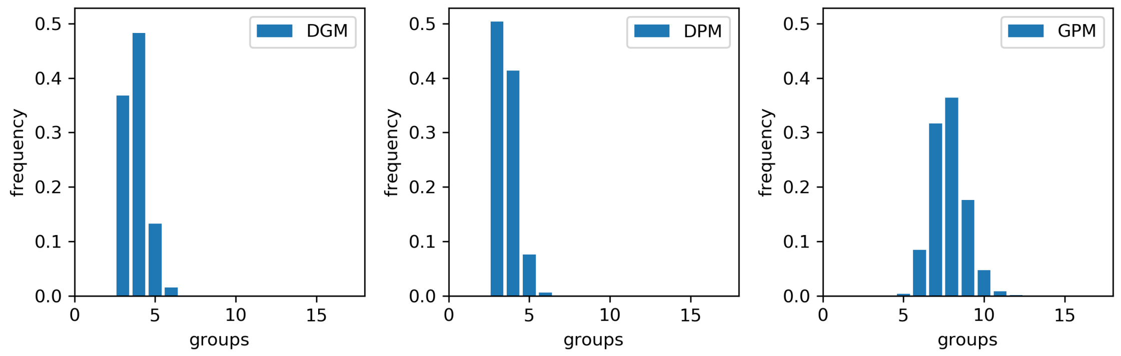

Figure 4.11: Histogram on the number of groups K_n of a database given by 300 independent data points from a multivariate normal mean-variance mixture of three multivariate normal distributions using DGM, DPM and GPM models.

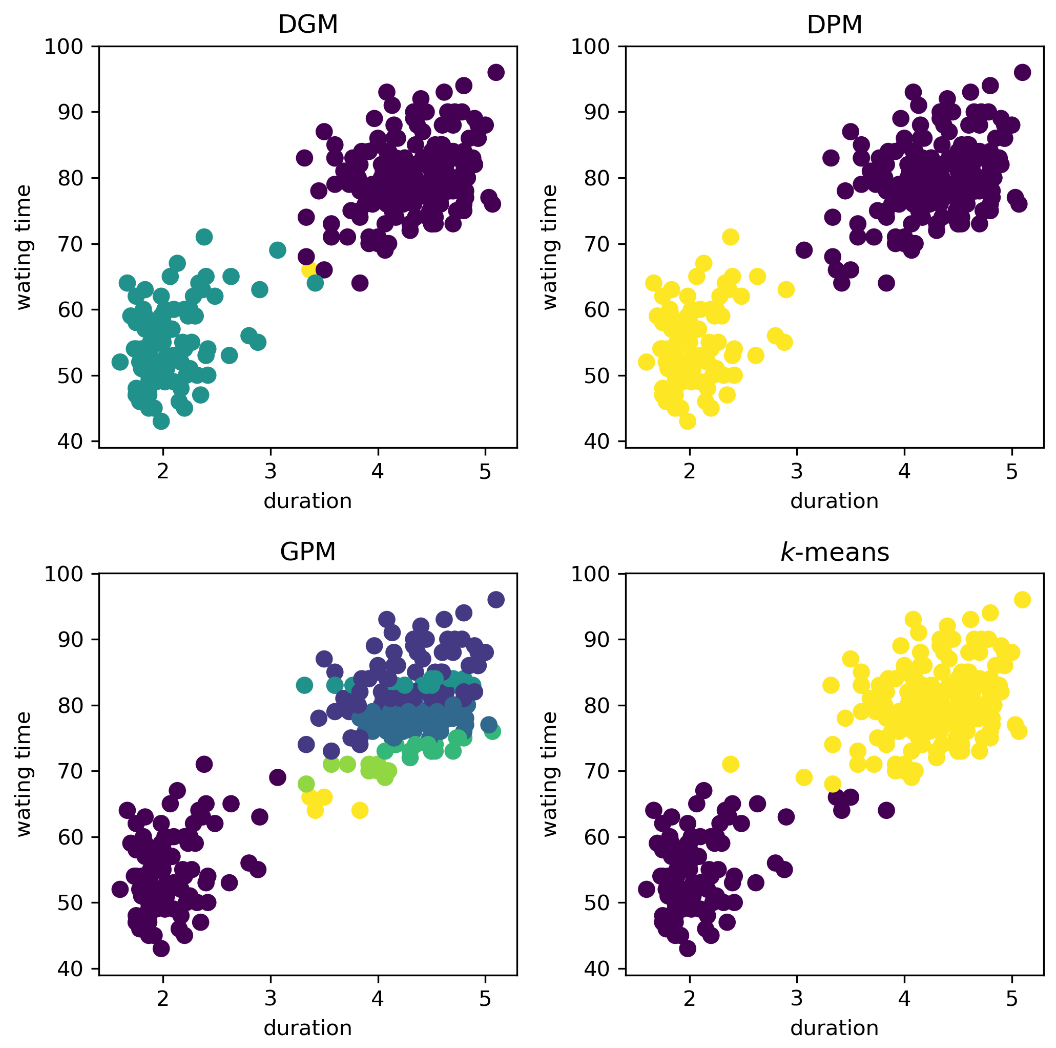

Figure 4.12: Fitted clusters for the “faithful” data set using DGM, DPM and GPM models after $50,000$ steps of the Gibbs sampler. The hyperparameters used for the DGM were $(x, \theta, \alpha, \beta, \boldsymbol{\mu}, \lambda, \boldsymbol{\Psi}, \nu) = \left(\frac{1}{2}, 1, 1, 1, (0,0), 1, \left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right), 2\right)$, for the DPM $(x, \theta, \boldsymbol{\mu}, \lambda, \boldsymbol{\Psi}, \nu) = \left(\frac{1}{2}, 1, 1, 1, (0,0), 1, \left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right), 2\right)$ and for the GPM $(x, \alpha, \beta, \boldsymbol{\mu}, \lambda, \boldsymbol{\Psi}, \nu) = \left(\frac{1}{2}, 1, 1, 1, (0,0), 1, \left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right), 2\right)$.

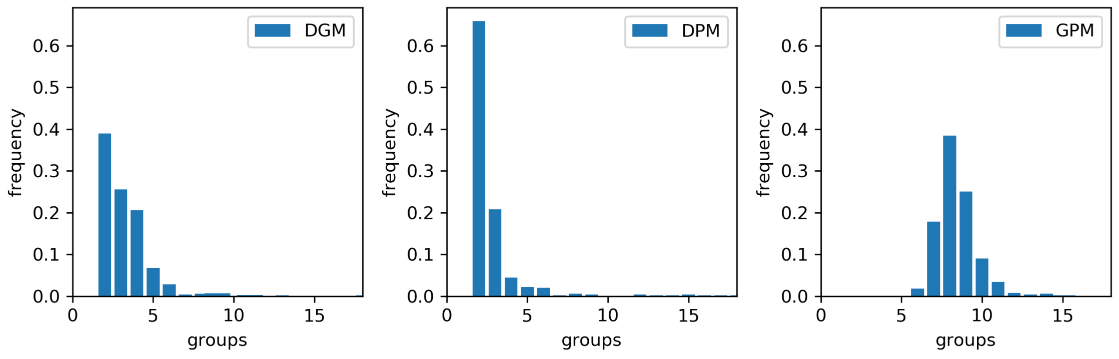

Figure 4.13: Histogram on the number of groups K_n in the “faithful” data set using DGM, DPM and GPM models.

4.3 Discussion

In this chapter we showed that the Dirichlet Geometric Mixture is a useful tool for doing both density estimation and clustering.

Empirically the Dirichlet Geometric Mixture showed a slightly faster stabilization than the competitors on the number of clusters for all the examples.

One of the main advantages of the Geometric Process is that fitting a density mixture is faster compared to the Dirichlet and, as shown in Fuentes-García, Mena, and Walker (2009), it can arbitrarily approximate the Dirichlet model. A direct consequence is that the number of groups that the Geometric Process Mixture needs to estimate a density is greater that what the Dirichlet Process Mixture needs. This translate to poor cluster fitting from the GPM as shown by Figure 4.12.

A disadvantage of the DGM against DPM and GPM is that, since there are additional calculations in the Gibbs sampler, the fitting is slower.

We implemented a free source library (Selva, 2020) in Python under the MIT license to do density estimation and clustering with the DGP using Dirichlet Geometric Mixture models via a Gibbs sampler.

The most computing intensive task was fitting the second example in section 4.1, which averaged 86 iterations per second for the DGM, 120 iterations per second for the DPM and 263 iterations per second for the GPM on an Intel(R) Core(TM) i5-9300H CPU.

Chapter 5

Conclusions

We found a process that can interpolate continuously the Dirichlet and Geometric Processes, namely, the Dirichlet Geometric Process. Also, since the weights in the weighting structure are conditionally independent, they are (without conditioning) exchangeable.

This process can be useful in applications that require an insight in how to transition continuously from a result yielded by the Dirichlet Process to one given by the Geometric Process.

Some topics open to further research are the sensitivity of the posteriori model to the model parameters, an analysis of the assignation probabilities in order to extract information about the possible overestimation of groups for values of x close to 1, and methods to enable this process to fit streaming data.

Appendix

Definition 11 (Lévy-Prokhorov metric)**.** Let $\mathcal{P}$ be the space of probability measures on the Borel σ-field $\mathcal{S}$ of a metric space s. The Prokhorov metric $\pi(P,Q)$ between elements P and Q of $\mathcal{P}$ is defined as the infimum of those positive ϵ for which the two inequalities

$$P(A) \leq Q(A^{\epsilon}) + \epsilon, \quad Q(A) \leq P(A^{\epsilon}) + \epsilon$$

hold for all Borel sets A; where the set $A^{\epsilon} = \{s \in S : d(s,A) < \epsilon\}$ and the distance $d(s,A) = \inf\{d(s,a) : a \in A\}$. Billingsley, 2013

Definition 12 (Borel measure)**.** A Borel measure is a measure defined on the Borel sets of a topological space. Aliprantis, 1999

Definition 13 (Support of a measure)**.** The topological support of a finite Borel measure μ on a completely separable space is defined as the intersection of all closed sets F such that $\mu(F^c) = 0$. Bissiri and Ongaro, 2014

Definition 14 (Exchangeable sequences)**.** A finite or infinite random sequence $\mathbf{Y} = (Y_1, Y_2, \dots)$ with index set I is said to be exchangeable if

$$(Y_{k_1}, Y_{k_2}, \dots) \overset{d}{=} (Y_1, Y_2, \dots)$$

for any finite permutation $(k_1, k_2, \dots)$ of I.

Proposition 15 (finite-dimensional distributions). *Fix a measurable space $(S, \mathcal{S})$, an index set T, and a subset $U \subset \mathcal{S}^T$, and let X and Y be processes on T with paths in U. Then $X \overset{d}{=} Y$ iff*

$$(X_{t_1}, \dots, X_{t_n}) \overset{d}{=} (Y_{t_1}, \dots, Y_{t_n}), \qquad t_1, \dots, t_n \in T,\ n \in \mathbb{N}$$

The demonstration of this can be found as Proposition 3.2 in Kallenberg, 2002.

Theorem 16 (Portmanteau). *For any random elements $\xi, \xi_1, \xi_2, \dots$ in a metric space S, these conditions are equivalent:*

1. $\xi_n \xrightarrow{d} \xi$.
2. $\liminf_n P(\xi_n \in G) \geq P(\xi \in G)$ *for any open set* $G \subset S$.
3. $\limsup_n P(\xi_n \in F) \geq P(\xi \in F)$ *for any closed set* $F \subset S$.
4. $P(\xi_n \in B) \to P(\xi \in B)$ *for any* $B \in \mathscr{B}(S)$ *with* $\xi \notin \partial B$ *almost surely*

A proof of this can be found in Kallenberg, 2002 as Theorem 4.25

Appendix II

Python code

Code of the library

```python
import mpmath
import scipy.stats
import numpy as np
from tqdm import trange
from itertools import repeat
from collections import namedtuple

np.seterr(divide='ignore')

normal_invw_params = namedtuple("normal_invw_params",
                                ['mu', 'lam', 'psi', 'nu'])
normal_params = namedtuple("normal_invw_params",
                           ['mu', 'Sigma'])

class dgp_mixture:
    def __init__(self, y, c, xp, a, b, mu0, lam0, psi0, nu0,
    ↪ fit_var=True, *,
                 p_method=0, max_iter=10, rng = None):
        if rng is None:
            self.rng = np.random.default_rng()
        else:
            self.rng = rng

        self.y = y
        self.fit_var = fit_var
        self.p_method = p_method

        self.c = c
        self.xp = xp
        self.a = a
        self.b = b

        self.init_params = normal_invw_params(mu0, lam0, psi0, nu0)

        self.max_iter = max_iter

        self.sim_params = []
        self.n_groups = []
        self.n_theta = []

        self.p = self.rng.beta(self.a, self.b)
        self.u = self.rng.uniform(0, 1, len(self.y))

        if self.xp==1:
            self.v = np.array([self.p])
        else:
            self.v = self.rng.beta(1 +  self.xp / (1 - self.xp) *
   ↪ self.p,
                                    self.c +  self.xp / (1 -
   ↪ self.xp) * (1 - self.p), 1)
        self.w = self.v
        while (sum(self.w) < 1 - min(self.u)):
            if self.xp==1:
                self.v = np.concatenate((self.v, np.array([self.p])))
            else:
                self.v = np.concatenate((self.v,
                                         self.rng.beta(1 + self.xp / (1 - self.xp) * self.p,
                                                       self.c + self.xp / (1 - self.xp) * (1 - self.p), 1)))
            self.w = self.v * np.cumprod(np.concatenate(([1], 1 - self.v[:-1])))

        self.k = len(self.w)

        self.mu, self.Sigma = random_normal_invw(*self.init_params)
        self.mu = np.array([self.mu])
        self.Sigma = np.array([self.Sigma])
        self.complete_theta()

        self.d = self.rng.integers(len(self.y)/5, size=len(self.y))

    def gibbs_step(self):
        self.update_theta()
        self.update_v_w_u()
        self.complete_theta()

        self.update_p()
        self.update_d()

        self.sim_params.append((self.w, self.mu, self.Sigma, self.u, self.d, self.p))
        self.n_groups.append(len(np.unique(self.d)))
        self.n_theta.append(len(self.mu))

    def train(self, n_iter):
        for i in trange(n_iter):
            self.gibbs_step()

    def density(self, x, periods=None):
        y_sim = []
        if periods==None:
            for ip in self.sim_params:
                y_sim.append(mixture_density(x, ip[0], ip[1],
↪ ip[2], ip[3]))
        else:
            periods = min(periods, len(self.sim_params))
            for ip in self.sim_params[-periods:]:
                y_sim.append(mixture_density(x, ip[0], ip[1],
↪ ip[2], ip[3]))
        return np.array(y_sim).mean(axis=0)

    def density_ix(self, x, ix):
        return mixture_density(x,
                               self.sim_params[ix][0],
                               self.sim_params[ix][1],
                               self.sim_params[ix][2],
                               self.sim_params[ix][3])

    def likelikood(self, x, periods=None):
        ret_likelihood = []
        if periods==None:
            for ip in self.sim_params:
                ret_likelihood.append(full_log_likelihood(x, ip[0],
↪ ip[1], ip[2], ip[3]))
        else:
            periods = min(periods, len(self.sim_params))
            for ip in self.sim_params[-periods:]:
                ret_likelihood.append(full_log_likelihood(x, ip[0], ip[1], ip[2], ip[3]))
        return np.array(ret_likelihood)

    def update_d(self):
        logproba = np.log([scipy.stats.multivariate_normal.pdf(self.y,
                                                               self.mu[j],
                                                               self.Sigma[j],
                                                               1)*(self.w[j] > self.u)
                           for j in range(self.k)])

        samp = sample(logproba, rng = self.rng)
        self.d = samp

    def update_theta(self):
        assert len(self.mu)==len(self.Sigma)
        self.d = np.unique(self.d, return_inverse=True)[1]
        self.mu = []
        self.Sigma = []
        for j in range(max(self.d)+1):
            inj = (self.d == j).nonzero()[0]

            posteriori_params = posterior_norm_invw_params(self.y[inj],
                                                           *self.init_params)
            temp_mu, temp_Sigma = random_normal_invw(*posteriori_params, rand = self.rng)
            self.mu.append(temp_mu)
            self.Sigma.append(temp_Sigma)
```

```python

        self.mu = np.array(self.mu)
        self.Sigma = np.array(self.Sigma)


    def complete_theta(self):
        missing_len = self.k-len(self.mu)
        for _ in range(missing_len):
            temp_mu, temp_Sigma = random_normal_invw(*self.init_params)
            self.mu  = np.concatenate((self.mu, [temp_mu]))
            self.Sigma = np.concatenate((self.Sigma, [temp_Sigma]))


    def update_v_w_u(self):
        if self.xp == 1:
            self.v = np.repeat(self.p, max(self.d)+1)
            self.w = self.v * np.cumprod(np.concatenate(([1], 1 - self.v[:-1])))
            self.u = self.rng.uniform(0, self.w[self.d])
            n_p    = int(np.log(min(self.u))/np.log(1-self.p))+1
            self.v = np.repeat(self.p, n_p)
            self.w = self.v * np.cumprod(np.concatenate(([1], 1 - self.v[:-1])))
            self.k = len(self.v)
            return

        a_c = np.bincount(self.d)
        b_c = np.concatenate((np.cumsum(a_c[::-1])[::-1][1:], [0]))

        self.v = self.rng.beta(1 + self.xp/(1-self.xp)*self.p + a_c,
                               self.c + self.xp/(1-self.xp)*self.p + b_c)
        self.w = self.v * np.cumprod(np.concatenate(([1], 1 - self.v[:-1])))
        self.u = self.rng.uniform(0, self.w[self.d])
        w_sum = sum(self.w)
        while (w_sum < 1 - min(self.u)):
            self.v = np.concatenate((self.v, [self.p] if self.xp == 1 else self.rng.beta(1 + self.xp / (1 - self.xp) * self.p,
                                                        self.c + self.xp / (1 - self.xp) * self.p, 1)))
            self.w = np.concatenate((self.w, [(1 - sum(self.w)) * self.v[-1]]))
            w_sum += self.w[-1]
        self.k = len(self.v)



    def update_p(self):
        if self.xp==0:
            return
        if self.xp==1:
            self.p = self.rng.beta(self.a+len(self.d), self.b+self.d.sum())
            return
        if self.p_method==0:
            prev_logp = l_x(self.xp, self.p, self.c, self.v, self.a, self.b)
            curr_iter = 0
            pass_condition = 0
            while curr_iter<self.max_iter:
                pass_var = self.rng.uniform()
                temp_p = self.rng.uniform()
```

```python
                temp_logp = l_x(self.xp, temp_p, self.c, self.v, self.a, self.b)
                pass_condition += np.exp(temp_logp-prev_logp) > pass_var
                curr_iter += 1
                if pass_condition>3:
                    break
            self.p = temp_p if pass_condition else self.p
        elif self.p_method==1:
            max_param = scipy.optimize.minimize(lambda p: -l_x(self.xp, p,self.c,self.v, self.a, self.b), self.p,
                                                bounds=[(0,1)],
                                                options={'maxiter':self.max_iter })
            if -max_param.fun[0] == np.inf:
                self.p = rejection_sample(lambda p: -l_x(self.xp, p,self.c,self.v, self.a, self.b),
                                          1e2)
            else:
                self.p = rejection_sample(lambda p: -l_x(self.xp, p,self.c,self.v, self.a, self.b),
                                          -max_param.fun[0])
        else:
            max_param = scipy.optimize.minimize(lambda p: -l_x(self.xp, p,self.c,self.v, self.a, self.b), self.p,
                                                bounds=[(0,1)],
                                                options={'maxiter':self.max_iter })
            if max_param.success:
                self.p = max_param.x[0]
            else:
                return

```

```
    def get_n_groups(self):
        return self.n_groups

    def get_n_theta(self):
        return self.n_theta

    def get_sim_params(self):
        return self.sim_params


def sample(logp, size=None, *, rng = None):
    if size is None:
        ret = np.argmax(logp -
    ↪ np.log(-np.log(rng.uniform(size=logp.shape))),axis=0)
    else:
        ret = []
        for i in range(size):
            ret.append(np.argmax(logp -
    ↪ np.log(-np.log(rng.uniform(size=len(logp))))))
        ret = np.array(ret)
    return ret

def rejection_sample(f, max_y, a=0, b=1, size=None, *, rng = None):
    if size is None:
        x = rng.uniform(a, b)
        y = rng.uniform(0, max_y)
        while y > f(x):
            x = rng.uniform(a, b)
            y = rng.uniform(0, max_y)
        return x
    else:
        x = rng.uniform(a, b, size)
        y = rng.uniform(0, max_y, size)
        while (y > f(x)).any():
            x[y > f(x)] = rng.uniform(a, b, (y > f(x)).sum())
            y[y > f(x)] = rng.uniform(0, max_y, (y > f(x)).sum())
        return x

def l_x(x, p, c, v, a, b):
    s = 0
    for vi in v:
        s += np.log(scipy.stats.beta.pdf(vi, 1 + x / (1 - x) * p, c
    ↪ + x / (1 - x) * (1 - p)))
    s += np.log(scipy.stats.beta.pdf(p, a, b))
    return s

def full_log_likelihood(y, w, mu, lam, u):
    return np.log(mixture_density(y, w, mu, lam, u)).sum()

def mixture_density(x, w, mu, Sigma, u):
    k = len(w)

    ret = []
    for j in range(k):
        ret.append(scipy.stats.multivariate_normal.pdf(x,
                                                        mu[j],
                                                        Sigma[j],
                                                        1))

    ret = np.array(ret).T
    mask = (np.array(list(repeat(u, k))) <
            np.array(list(repeat(w, len(u))))).transpose()

    ret = ret.dot(mask/mask.sum(0)).mean(1)
    return ret

```

```
def mixture_density_pre(x, w, mu, lam):
    k = len(w)
    ret = scipy.stats.norm.pdf(np.array(list(repeat(x, k))).T,
    ↪ loc=list(repeat(mu[:k], len(x))),
                               scale=list(repeat(np.sqrt(1 /
    ↪ lam[:k]), len(x))))
    ret = (w * ret).sum(1)
    ret /= sum(w)

    return ret

def cluster(x, w, mu, Sigma, u):
    k = len(w)
    ret = []
    for j in range(k):
        ret.append(scipy.stats.multivariate_normal.pdf(x,
                                                        mu[j],
                                                        Sigma[j],
                                                        1))
    ret = np.array(ret).T
    weights = (np.array(list(repeat(u, k))) <
            np.array(list(repeat(w, len(u)))).transpose())

    weights = (weights/weights.sum(0)).sum(1)/len(u)
    ret = ret*weights
    grp = np.argmax(ret, axis=1)
    u_grp, ret = np.unique(grp, return_inverse=True)
    return (ret, weights[u_grp], mu[u_grp], Sigma[u_grp])

def random_normal_invw(mu, lam, psi, nu, rand=None):
    ret_Sigma = scipy.stats.invwishart.rvs(nu, psi,
                                          random_state=rand)
    ret_mu = scipy.stats.multivariate_normal.rvs(mu, ret_Sigma/lam,
                                                                       random_state=rand)
    return normal_params(ret_mu, ret_Sigma)

def posterior_norm_invw_params(y, mu, lam, psi, nu):
    n = y.shape[0]
    ret_mu = (lam*mu+n*y.mean(axis=0))/(lam+n)
    ret_lam = lam+n
    ret_psi = psi + n*np.cov(y.T, bias=True) +
    ↪ (lam*n)/(lam+n)*((y.mean(axis=0)-mu)@(y.mean(axis=0)-mu))
    ret_nu = nu+n
    return normal_invw_params(ret_mu, ret_lam, ret_psi, ret_nu)
```

Code used to fit the data and generate the density plots in Figure 4.7.

```
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from dgp_multidim import dgp_mixture as dgm_mixture

# Load dataset
galaxy = sm.datasets.get_rdataset("galaxies", "MASS")
y = galaxy.data["dat"].to_numpy()/1000

#Set seed
seed=0
rand = np.random.default_rng(seed)

#Set parameters
n_burn = int(1e3)
n_train = int(4e3)
n_dim = 1
mu0 = np.repeat(0, n_dim)
lam0 = 1
psi0 = np.identity(n_dim)
```

```
nu0 = n_dim

#Fit the data
my_dgm0 =  dgm_mixture(y, 1, 0,   1, 1, mu0, lam0, psi0, nu0, rng =
    ↪ rand)
my_dgm05 = dgm_mixture(y, 1, 0.5, 1, 1, mu0, lam0, psi0, nu0,
    ↪ p_method=0, rng = rand)
my_dgm1 =  dgm_mixture(y, 1, 1,   1, 1, mu0, lam0, psi0, nu0, rng =
    ↪ rand)
my_dgm0.train(n_burn+n_train)
my_dgm05.train(n_burn+n_train)
my_dgm1.train(n_burn+n_train)

#Get fitted density
x = np.linspace(min(y)-1, max(y)+1, 200)
dgm0_y = my_dgm0.density(x,n_train)
dgm05_y = my_dgm05.density(x,n_train)
dgm1_y = my_dgm1.density(x,n_train)

#Plot
from sklearn.neighbors import KernelDensity
fig = plt.figure(figsize=(7,7))
n_bins = 40

ax = fig.add_subplot(2,2,1)
ax.hist(y, n_bins, density=True, histtype='step')
ax.plot(x, dgm05_y, label="DGM")
ax.set_xlabel('y')
ax.set_ylabel('$\hat{f(y)}$')
ax.legend()

ax = fig.add_subplot(2,2,2,sharey=ax)
kde = KernelDensity(kernel='gaussian', bandwidth=1)
```

```
kde.fit(y.reshape(-1, 1))
ax.hist(y, n_bins, density=True, histtype='step')
ax.plot(x,np.exp(kde.score_samples(x.reshape(-1,1))),label='KDE')
ax.set_xlabel('y')
ax.set_ylabel('$\hat{f(y)}$')
ax.legend()

ax = fig.add_subplot(2,2,3,sharey=ax)
ax.hist(y, n_bins, density=True, histtype='step')
ax.plot(x, dgm0_y, label="DPM")
ax.set_xlabel('y')
ax.set_ylabel('$\hat{f(y)}$')
ax.legend()

ax = fig.add_subplot(2,2,4,sharey=ax)
ax.hist(y, n_bins, density=True, histtype='step')
ax.plot(x, dgm1_y, label="GPM")
ax.set_xlabel('y')
ax.set_ylabel('$\hat{f(y)}$')
ax.legend()

plt.tight_layout()
plt.savefig('../images/galaxy_fit.png', dpi=300)
```

Code used to fit the data and generate the cluster plots in Figure 4.12

```
import scipy.stats
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from dgp_multidim import cluster
from sklearn.cluster import KMeans
import dgp_multidim

```

```python
#Load dataset
y = sm.datasets.get_rdataset("faithful").data.to_numpy()

#Set seed
seed = 0
rand = np.random.default_rng(seed)

#Set parameters
n_burn = int(1e4)
n_train = int(4e4)
n_dim = 2
mu0 = y.mean(0)
lam0 = 0.1
psi0 = np.identity(2)
nu0 = n_dim

#Fit the data
my_dgm0 =  dgp_multidim.dgp_mixture(y, 1, 0,   1, 1, mu0, lam0,
    ↪ psi0, nu0, rng = rand)
my_dgm05 = dgp_multidim.dgp_mixture(y, 1, 0.5, 1, 1, mu0, lam0,
    ↪ psi0, nu0, p_method=0, rng = rand)
my_dgm1 =  dgp_multidim.dgp_mixture(y, 1, 1,   1, 1, mu0, lam0,
    ↪ psi0, nu0, rng = rand)
my_dgm0.train(n_burn+n_train)
my_dgm05.train(n_burn+n_train)
my_dgm1.train(n_burn+n_train)

#Plot
fig = plt.figure(figsize=(7,7))

ax = fig.add_subplot(2,2,1)
ret, w, mu, lam = cluster(y,*(dgm05_max_params[0:4]))
col = plt.cm.get_cmap('viridis', max(ret)+1)
```

```
ax.scatter(y[:,0], y[:,1], c=ret, s=50, zorder=1)
ax.set_xlabel('duration')
ax.set_ylabel('wating time')
ax.set_title('DGM')

ax = fig.add_subplot(2,2,2,sharey=ax)
ret, w, mu, lam = cluster(y,*(dgm0_max_params[0:4]))
col = plt.cm.get_cmap('viridis', max(ret)+1)
ax.scatter(y[:,0], y[:,1], c=ret, s=50, zorder=1)
ax.set_xlabel('duration')
ax.set_ylabel('wating time')
ax.set_title('DPM')

ax = fig.add_subplot(2,2,3,sharey=ax)
ret, w, mu, lam = cluster(y,*(dgm1_max_params[0:4]))
col = plt.cm.get_cmap('viridis', max(ret)+1)
ax.scatter(y[:,0], y[:,1], c=ret, s=50, zorder=1)
ax.set_xlabel('duration')
ax.set_ylabel('wating time')
ax.set_title('GPM')

ax = fig.add_subplot(2,2,4,sharey=ax)
kmeans = KMeans(n_clusters=2, random_state=0).fit(y)
ret, w, mu, lam = cluster(y,*(dgm1_max_params[0:4]))
col = plt.cm.get_cmap('viridis', max(ret)+1)
ax.scatter(y[:,0], y[:,1], c=kmeans.labels_, s=50, zorder=1)
ax.set_xlabel('duration')
ax.set_ylabel('wating time')
ax.set_title('$k$-means')

plt.savefig('../images/faithful_cluster.png', dpi=300)
```

Bibliography

Aliprantis, Charalambos (1999). *Infinite dimensional analysis : a hitchhiker's guide*. Berlin New York: Springer. ISBN: 9783540658542.

Billingsley, Patrick (2013). *Convergence of probability measures*. John Wiley & Sons.

Bissiri, Pier Giovanni and Andrea Ongaro (2014). "On the topological support of species sampling priors". In: *Electronic Journal of Statistics* 8.1, pp. 861–882.

Bonaccorso, Giuseppe (2019). *Hands-On Unsupervised Learning with Python: Implement machine learning and deep learning models using Scikit-Learn, TensorFlow, and more*. Packt Publishing Ltd.

Devroye, Luc (2006). "Nonuniform random variate generation". In: *Handbooks in operations research and management science* 13, pp. 83–121.

Fuentes-García, Ruth, Ramsés H Mena, and Stephen G Walker (2009). "A nonparametric dependent process for Bayesian regression". In: *Statistics & probability letters* 79.8, pp. 1112–1119.

Ghosal, Subhashis and Aad van der Vaart (2017). *Fundamentals of nonparametric Bayesian inference*. Vol. 44. Cambridge University Press.

Han, Jiawei, Micheline Kamber, and Jian Pei (2012). "Cluster Analysis". In: *Data Mining*. Elsevier, pp. 443–495. DOI: 10.1016/b978-0-12-381479-1.00010-1.

Ishwaran, Hemant and Lancelot F James (2001). "Gibbs sampling methods for stick-breaking priors". In: *Journal of the American Statistical Association* 96.453, pp. 161–173.

Kallenberg, Olav (2002). *Foundations of Modern Probability.* Springer New York. DOI: `10.1007/978-1-4757-4015-8`.

Kalli, M., Griffin, J. E., and Walker, S. G. (Sept. 2009). "Slice sampling mixture models". In: *Statistics and Computing* 21.1, pp. 93–105. DOI: `10.1007/s11222-009-9150-y`.

Maddison, Chris J, Daniel Tarlow, and Tom Minka (2014). "A* sampling". In: *Advances in Neural Information Processing Systems*, pp. 3086–3094.

Mena, Ramsés H and Walker, S. G. (2012). "An EPPF from independent sequences of geometric random variables". In: *Statistics & Probability Letters* 82.6, pp. 1059–1066.

Parthasarathy, Kalyanapuram R (2005). *Probability measures on metric spaces.* Vol. 352. American Mathematical Soc.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Pitman, Jim (1996). "Random discrete distributions invariant under size-biased permutation". In: *Advances in Applied Probability* 28.2, pp. 525–539.

— (2006). "Combinatorial stochastic processes-Saint-Flour Summer School of Probabilities XXXII-2002". In: *Combinatorial stochastic processes* 1875, pp. 1–+.

Robert, Christian and George Casella (2013). *Monte Carlo statistical methods.* Springer Science & Business Media.

Selva, Fidel (2020). *Dirichlet Geometric Process.* `https://github.com/cabo40/Dirichlet-Geometric-Process/tree/master`.

Sethuraman, Jayaram (1994). "A constructive definition of Dirichlet priors". In: *Statistica sinica*, pp. 639–650.

Watterson, G. A. (Sept. 1974). “The sampling theory of selectively neutral alleles”. In: *Advances in Applied Probability* 6.3, pp. 463–488. DOI: `10.2307/1426228`.