Proyecto Fin de Máster
Máster en Ingeniería de Telecomunicación

Bayesian Non Parametric Machine Learning, a comprehensive approach

Autor: Pedro Morales Hernández
Tutor: Juan José Murillo Fuentes

Dep. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2016

Trabajo Fin de Máster
Máster en Ingeniería de Telecomunicación

Bayesian Non Parametric Machine Learning, a comprehensive approach

Autor:

Pedro Morales Hernández

Tutor:

Juan José Murillo Fuentes

Prof. Catedrático de Universidad

Dep. de Teoría de la Señal y Comunicaciones

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2016

Trabajo Fin de Máster: Bayesian Non Parametric Machine Learning, a comprehensive approach

Autor: Pedro Morales Hernández

Tutor: Juan José Murillo Fuentes

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2016

El Secretario del Tribunal

A mi familia

A mis profesores

Acknowledgments

At the final steps of this hard journey I think that I have to stop for a second, to look backwards and to analyse how I came up here. First of all, I want to specially thank my parents for the effort they made to help me become what I am. Also, I want to thank all my family for encouraging me to keep on pushing forward. Besides, I want to thank my girlfriend, Inmaculada, that taught me that the failure is not an option and once you have tried 100 times to succeed just 1, you learnt something from the other 99 times you tried.

This master degree shows you, not only the technical knowledgement but also a deep sense of the effort required to achieve your goals and a high capacity to overwhelm difficulties. And these also have to be thanked to all my professors and colleagues.

Finally, I want to specially thank professor Juan José Murillo for helping me to dive into the deepest topics of Machine Learning and Statistics.

Thank you all,

Pedro Morales Hernández

Estudiante de Máster de Ingeniería de Telecomunicación

Sevilla, 2016

Resumen

Cuando se trata de clasificar cosas los seres humanos lo hacemos bastante bien. Somos capaces de discernir en segundos si un objeto está caliente o frio, si una acción es peligrosa o segura, etc. Es algo que hacemos a menudo y de forma muy eficiente. Sin embargo, hacer que los ordenadores sean capaces de realizar estas tareas siempre se ha antojado bastante complicado. Instruir a un ordenador para que, sin intervención humana, tome decisiones basándose en su conocimiento previo es uno de los grandes objetivos de los algoritmos de *Machine Learning*. Durante la segunda mitad del siglo XX, con el avance de la informática, se crearon nuevos algoritmos de clasificación para dar solución a los problemas comunes de la época.

A grandes rasgos, podemos distinguir 2 tipos de problemas de clasificación: el aprendizaje supervisado, en el que tenemos pares entrada-salida del sistema y el aprendizaje no supervisado, en la que únicamente conocemos los datos de entrada. Cabe destacar que el aprendizaje no supervisado, comúnmente llamado *clustering* en los problemas de clasificación, suele ser más complejo que su contraparte supervisada. Esto se debe al desconocimiento, a priori, del número de agrupaciones o clases que existen en el sistema. Imaginemos un caso concreto, ¿Cuantos grupos podemos encontrar si deseamos clasificar gente en función de sus gustos musicales? Gente a la que le gusta el rock, el folk, el pop, gente a la que solo le gusta un determinado grupo musical, gente a la que le encanta la música de los 80, etc. Las combinaciones son infinitas.

Por ello, uno de los parámetros que debemos de establecer normalmente en los algoritmos de clusterización más tradicionales es el número de clases que deseamos. Esto crea un problema adicional, como en el ejemplo anterior, en algunos casos no es posible conocer con anterioridad el número de elementos que compone un sistema. Es por ello que, durante el siglo XXI se ha estado trabajando en modelos y algoritmos que no necesitan conocer el número de agrupaciones de antemano. Estos modelos van a ir proponiendo diferentes soluciones, con una cantidad variable de clases.

Las herramientas necesarias para poder emplear los modelos no parámetricos son complejas y difíciles de desentrañar a primera vista. Entre ellas, podemos encontrar: el proceso de Dirichlet (DP), el proceso del restaurante chino (CRP) y el muestreador de Gibbs (un tipo de muestreador basado en las cadenas de Markov). Es por ello que, durante este trabajo, realizaremos una introducción detallada de cada una de ellas, en las que explicaremos qué son y para que se van a utilizar. Posteriormente, las emplearemos conjuntamente para diseñar un algoritmo que nos permita proponer, de manera dinámica, un número de agrupaciones variable para ajustar nuestros datos.

En lugar de determinar mediante el modelo el número de agrupaciones, dejaremos que sean los datos los que determinen la cantidad de ellas que mejor los explican. El objetivo no es obtener un número de agrupaciones determinado si no, más bien, una estimación de la función masa de probabilidad de su cantidad. La cual podremos marginalizar posteriormente para obtener un resultado en conjunto.

Seguidamente, describiremos empleando UML y propondremos una implementación en Matlab® de este algoritmo tratando de sea lo más eficiente posible que finalmente pondremos en práctica con modelos de mezclas infinitas de gaussianas pero que podremos emplear con cualquier distribución de probabilidad de nuestros datos a través de una interfaz estandarizada. Por último, detallaremos algunos ejemplos para explicar el funcionamiento de este algoritmo.

Abstract

When the task is classifying things, we, humans, do it fairly well. We can determine within seconds whether an object is hot or cold, if an action is dangerous or safe, etc. We do it very often and very efficiently. But, to instruct computers in doing these tasks has always been quite complex. To make a computer make decisions based upon its previous knowledge, without human intervention, is one of the main goals of Machine Learning algorithms. During the second half of the 20th century, as computers improved, new algorithms were developed to solve ordinary problems.

At first sight, we can distinguish between 2 types of classification problems: **supervised learning**, in which input-output samples exist for our system and **unsupervised learning**, in which we only know the input data. Remark that, unsupervised learning, commonly called clustering for classification problems, is rather more complex than its supervised counterpart. This is due to the *a priori* uncertainty on the number of classes that compose the system. For example, how many groups can we find if we are classifying people according to its musical likes? There are people who like rock music, people who like pop music, people who only likes one artist, people devoted to the 80's music, etc. The possible combinations are endless.

For so, one of the parameters that we usually have to configure in the most traditional clusterization algorithms is the number of classes we are looking for. This creates an additional problem: as in the previous example, in some cases, it is not possible to know beforehand the number of elements that compose a system. Due to this limitation, during the 21st century, people have worked on models and algorithms that do not require to set the number of clusters. These models are going to propose iteratively different solutions, with a variable amount of groups.

Tools required to make use of this non-parametric models are complex and difficult to work out at a glance. Among them, we can find: The Dirichlet process (DP), the Chinese restaurant process (CRP), and the Gibbs Sampler (a type of sampler based on Markov Chains). This is the reason why, during this work, we will make a detailed introduction on each of these tools, in which we will explain briefly what they are and how are we going to use them. Later on, we will use all of them to design an algorithm that will allow us to propose, dynamically, a variable number of clusters to fit our data.

Instead of determining, throughout the model the number of classes, we will let the data define the amount of them that better fits it. The objective is not to obtain a given number of clusters but rather, to get an estimation of the probability mass function for this quantity. We could later marginalize it to obtain a joint result.

Next, using UML we will describe and we will propose an implementation of this algorithm using Matlab®, trying to make it as efficient as possible. Finally, we will put it into practice with infinite Gaussians mixture models. Although, we can use it with every probability distribution for our data through a standardized interface. At last, but not least, we will detail some examples to explain how does the algorithm work.

Table of Contents

Tables Index

Figures Index

Notation

Algebra

$\boldsymbol{A}$	Matrix (upcase, bold)
$\boldsymbol{x}$	Vector (lowercase, bold)
α	Scalar (lowecase)
$\boldsymbol{x}^T$	Tranpose
$\boldsymbol{A}^{-1}$	Inverse of $\boldsymbol{A}$
$\lvert\boldsymbol{A}\rvert$	Determinant of $\boldsymbol{A}$
$\mathbb{I}_D$	D-dimensional identity matrix

Calculus

e	Number $e = 2.7182$ ...
$\partial/\partial x$	Partial derivative with respect to x
$\ln x$	Napierian logarithm of x.

Probability and statistics

$p(x)$	Probability of x (discrete)
$f(x)$	Probability of x (continious)
$\mathbb{E}[x]$	Espected value of x
$\mathcal{N}_D(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	Multivariate D-dimensional gaussian with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$
$Dir(\boldsymbol{\alpha})$	Dirichlet Distribution with concentration vector $\boldsymbol{\alpha}$
$\mathcal{W}^{-1}(\boldsymbol{R}, \nu)$	Inverse Wishart Distribution with precision matrix $\boldsymbol{R}$ and ν degrees of freedom

Sets and Intervals

$\{a, b, ...\}$	Set
$[a, b]$	Closed interval, within a and b, inclusive
(a, b)	Open interval within a and b, exclusive

Functions

$\mathbb{I}(x)$	Indicator function
$\delta(x)$	Delta function
$\Gamma(x)$	Gamma function

Other

$\boldsymbol{x}_i$	Sample i
$x_i^{(d)}$	Value for dimension d of sample i
$\boldsymbol{x} \sim F(\boldsymbol{\theta})$	$\boldsymbol{x}$ follows a distribution F with parameters $\boldsymbol{\theta}$
$x \propto y$	x is proportional to y
■	Quod erat demonstrandum

1 THE CLASSIFICATION PROBLEM

"The real problem is not whether machines think
but whether men do"

- B.F. Skinner -

Classifying has always been one of the major commitments of ML algorithms. Teaching a computer to make decisions without human interaction based on its previous knowledge is, for sure, one of the most interesting applications of Machine Learning in real life. As a matter of fact, everybody unconsciously uses them: when we send emails and SPAM classifiers, when we use credit cards and fraud detectors or simply when we use our mobile phone, and consequently signal detection. Basically a classification problem is the one that assigns a class or label to a datapoint. Therefore, making a decision about the datapoint itself. This datapoint could contain different information. For example, a set of characteristics from a given email that helps to determine whether it is SPAM or not. Also, in the simplest signal detection systems, classification helps to choose if a given signal is either a 0 or a 1.

This chapter aims to provide a first classification outline on the different classification problem types that we can face. Then a traditional classification problem solution, through the Expectation-Maximization algorithm, is provided. This is done in order to finally state the disadvantages of this approach in certain cases.

1.1 Classification Types

To make it even more interesting; classification systems can also be classified. There have been proposed several ways of doing so. Just the most common are listed here.

1.1.1 Based upon the output

The first classification type we can introduce states the difference between the diverse output types of the system:

1. **Hard-Classification:** We assign to each datapoint a given label, usually a natural number.

$$C(\mathbb{R}^D) \to \mathbb{N}$$

2. **Soft-Classification:** We assign to each datapoint a real number, indicating likehood to belong to a given class. Usually a real positive number

$$C(\mathbb{R}^D) \to \mathbb{R}^+$$

Here $C(\cdot)$ is our classification system, that takes a D-dimensional datapoint and produces an output. Though different systems comprise different requirements, most of them can be modelled as binary decisors. Furthermore, we can model any multi-class classification problem as several smaller binary decisors. Thus, previously given models for classifiers become even simpler:

1. Hard-Classification: $C_k(\mathbb{R}^n) \to \{0,1\}$
2. Soft-Classification: $C_k(\mathbb{R}^n) \to [0,1]$

Where the index k denotes each of the classes for our system. Through this work classes, components and clusters are the same: groups of datapoints for our system.

Also, every soft-classifier can be converted into its hard counterpart just by adding a decisor after it, as observed in Figure 1. This one will assign a label to the datapoint based on the soft classification made in advance. From an initial dataset a soft-classifier is built and then a decisor is chained in order to obtain the data labels.

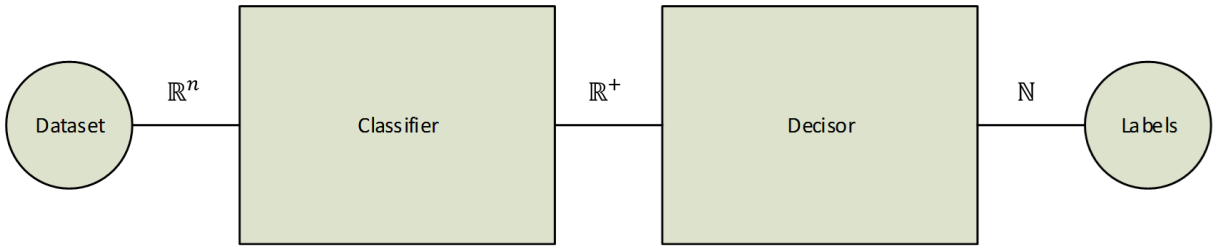


Figure 1 Decision problems block diagram

1.1.2 Based upon the available data (Clustering vs. Classification)

Another classification we can introduce is based on the existence of training samples, that is we know the output of our system for a set of inputs:

1. **Supervised Problems (Classification):** Input-Output samples exist.
2. **Unsupervised Problems (Clustering):** Only input samples exist.

The first type of problems is the one in which, for a given set of inputs, their respective output is known. In some cases, for classification problems a regressive model is built using linear or polynomial regression, along with some kind of logistic function. Other classification approaches for supervised classification problems include SVM or random forests. Although very interesting, this kind of problems is beyond the scope of this document, and will no longer be treated. Further references can be found in [1]. We will center in the unsupervised learning problems.

Besides, a brief example of this type of problems, using a logistic regressor, is the shown. Let

$$\hat{y} = \frac{1}{1 + e^{-\boldsymbol{X}\boldsymbol{w}}} \tag{1.1}$$

be a logistic function. Here $\boldsymbol{X}$ are the input samples, and $\hat{y}$ is the output of the system. We compute the set of weights $\boldsymbol{w}$ which provides the minimum MSE between $\hat{y}$ **and the known outputs,** $\boldsymbol{y}$. The result can be seen in the following pictures, where an 8th degrees polynomial function has been used.

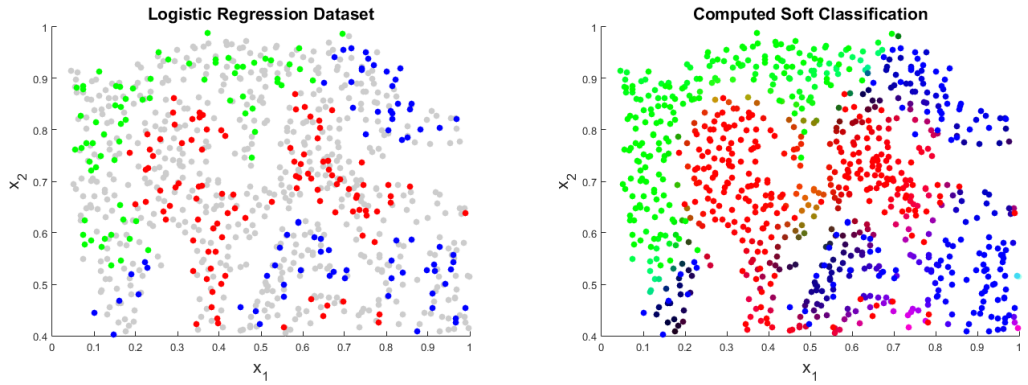


Figure 2 Logistic Regression

Figure 2 shows a typical logistic regression problem. The right picture illustrates the known input-output samples and the samples to be computed, where the colour indicates the label, and the gray samples are unknown. Then, we only take the training samples and compute a regressor. This is done in order to avoid overfitting and allow crossvalidation. Finally, using the rest of the samples, we estimate its likehood of being part of each group, which is the picture on the right.

The second approach is used when no output samples are available. The classification is made upon the underlying structure of the data; algorithms such as **k-means** or **DBSCAN** fall into this category.

Nevertheless**, k-means** or **DBSCAN** do not impose any restriction on how the samples were generated. But,

sometimes this assertion turns out to be too restrictive. Suppose that we already know that data have been generated by a statistical model, potentially very complex. Then, better clustering algorithms can be built if we can impose some restrictions taking advantage on the underlying statistical model. In these models, we can make some assumptions and therefore the model becomes more precise.

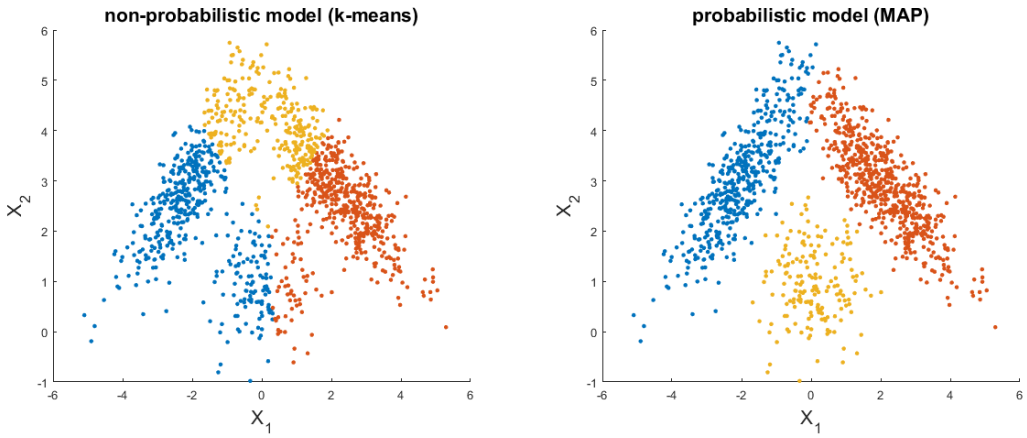


Figure 3 Different Clustering algorithms

As an example, Figure 3 shows the results of different clustering algorithms, the k-means algorithms does not suppose any underlying structure and tries to find the clusters by proximity of the datapoints, this sometimes conduces to errors, specially when clusters are wrapped by other clusters. The right picture shows the result using the underlying model to estimate the maximum *a posteriori* probability (MAP) for each cluster.

As a matter of fact, the model used to generate this data was composed by 3 clusters with the following parameters and weigths given by $\boldsymbol{\pi} = [0.35\ 0.5\ 0.15]$.

Cluster 1	Cluster 2	Cluster 3
$\boldsymbol{\mu}_1 = [-2\ 3]^T$	$\boldsymbol{\mu}_1 = [2\ 3]^T$	$\boldsymbol{\mu}_1 = [0\ 1]^T$
$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.8 & 0.48 \\ 0.48 & 0.96 \end{bmatrix}$	$\boldsymbol{\Sigma}_2 = \begin{bmatrix} 0.96 & -0.48 \\ -0.48 & 0.8 \end{bmatrix}$	$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.8 & 0 \\ 0 & 0.8 \end{bmatrix}$

Table 1 Figure 3 parameters

1.1.3 Based upon the objective

When a statistical model is used, depending on what we want to learn from it, a problem could be either:

1. **Generative:** When we want to learn the generating PDF.
2. **Discriminative:** When we want to learn conditional PDFs on the model.

1.2 GMM Definition

First, let us describe what Gaussian mixture models are. Gaussian models are highly used in telecommunications systems and will be the main subject of this article. A system, in which the output can be modelled as the sum of RV, each following a Gaussian distribution, is named Gaussian mixture models (GMM). A GMM is composed by a weighted sum of n-dimensional Gaussian PDFs each with a given mean and covariance matrix. Also, we must add an additional constraint on the weights, so the output is also a PDF.

$$\sum_k \pi_k = 1 \tag{1.2}$$

In a GMM, we can name each generating Gaussian as a class. Thus, sampling the model means sampling over the classes obtaining a latent variable z. And then, taking a sample from the chosen class.

More precisely,

$$p(z_i = k|\boldsymbol{\pi}) = \pi_k \tag{1.3}$$

The following picture illustrates an example of the PMF of $p(z_i)$. Which can be described through the following equation

$$p(z_i = k|\boldsymbol{\pi}) = \sum_{k=1}^{K} \pi_k \delta(k - z_k) \tag{1.4}$$

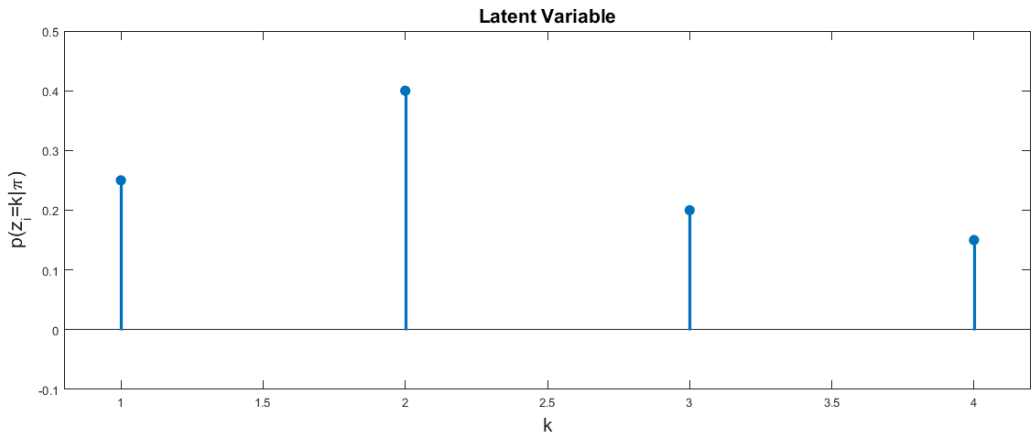


Figure 4 **PMF of the latent variable**

Then, we sample a Gaussian by the parameters of its class

$$f(\boldsymbol{x}_i|z_i = k, \boldsymbol{\theta}) = f(\boldsymbol{x}_i|\boldsymbol{\theta}_k) \tag{1.5}$$

Here $\boldsymbol{\theta}$ stands for the parameters of the whole distribution. Note that, (1.5) is valid not only for Gaussian mixtures but also for every mixture of PDFs, as we do not impose any restriction on the PDF of each class. In the case of GMM, $\boldsymbol{\theta}$ are the mean and the covariance matrix. Nevertheless, in this article we will dive into GMM but the reader is encouraged to find its own particular equations for other PDFs.

The objective when solving a GMM problem is to find the generating PDFs and the mixing coefficients. As we already know that the PDFs are Gaussian. So, we must only find the mean and the covariance matrix for each class. When done, we can compute the conditional probabilities of each datapoint of belonging to a given class, and finally a decisor based on them can be defined.

1.3 The EM Algorithm

A first approach to GMM is the Expectation-Maximization (EM) algorithm. It was first introduced by [2] on 1977. It defines an iterative process for the computation of maximum-likehood estimates for any PDF. More precisely, the algorithm tries to maximize the log-likehood, due to numerical issues. As the log function is a monotonically increasing function, both are equivalent.

We can define the PDF of any mixture model as

$$f(\boldsymbol{x}_i) = \sum_{k=1}^{K} f(\boldsymbol{x}_i|\boldsymbol{\theta}_k)p(\boldsymbol{\theta}_k) \tag{1.6}$$

A class, in a GMM, is a Gaussian involved in the mixture. Also, for each class k, its parameters $\boldsymbol{\theta}_k$ are defined.

Therefore, we can model our system using the following expressions

$$p(\boldsymbol{\theta}_k) = \pi_k \tag{1.7}$$

$$f(\boldsymbol{x}_i|\boldsymbol{\theta}_k) = \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{1.8}$$

Thus, the weights or mixing coefficients, are the prior of our data, and the marginal PDFs of each class follow a Gaussian distribution with parameters given by its class.

With EM, we try to invert the mixing process and, given $f(\boldsymbol{x}_i)$, find the following data:

- π_k: The mixing coefficients
- $\boldsymbol{\mu}_k$: The mean of each class
- $\boldsymbol{\Sigma}_k$: The covariance matrix

We have to find the class, or the probability of each datapoint of belonging to each class. This is called the datapoint responsibilities and it is the posterior of the data. So, it is computed using the Bayes theorem.

$$\gamma_k(\boldsymbol{x}_i) = p(\boldsymbol{\theta}_k|\boldsymbol{x}_i) = \frac{f(\boldsymbol{x}_i|\boldsymbol{\theta}_k)p(\boldsymbol{\theta}_k)}{f(\boldsymbol{x}_i)} \tag{1.9}$$

Substituing (1.7) and (1.8) in (1.9) we obtain

$$\gamma_k(\boldsymbol{x}_i) = p(\boldsymbol{\theta}_k|\boldsymbol{x}_i) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)} \tag{1.10}$$

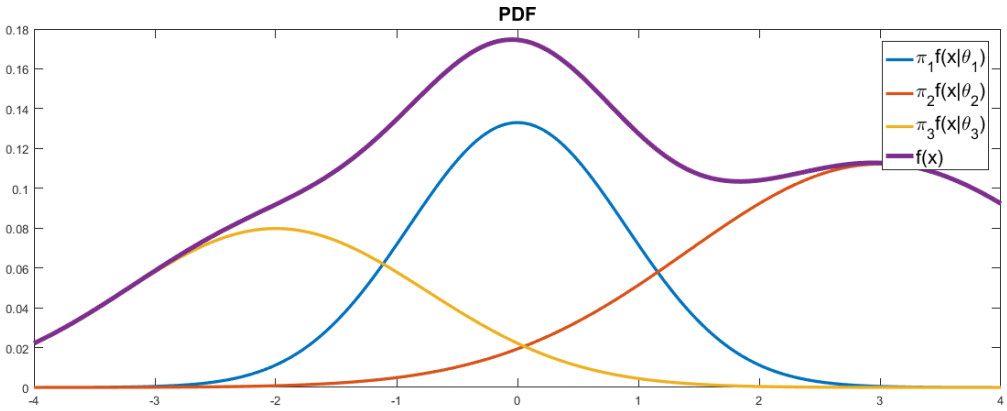


Figure 5 Marginal PDF and Posterior

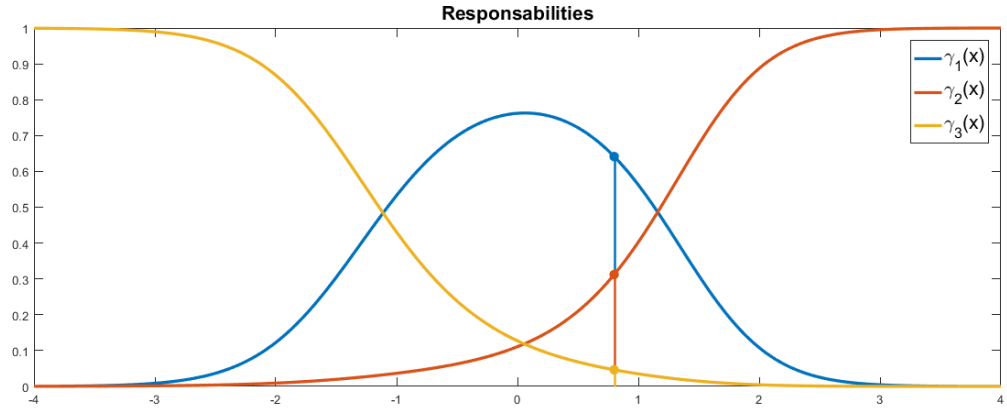


Figure 6 Responsibilities evaluation

We can finally model a decisor based on these responsibilities, for example the MAP decisor will assign a datapoint to the class with the highest responsibility or a posteriori probability.

The Expectation-maximization algorithm tries to iteratively maximize the likehood of a PDF with respect to the data provided changing the PDF parameters. In the case of GMM these are the means and covariance matrices. On each iteration, the algorithm performs 2 steps:

1. **Expectation step:** To calculate an estimation of the likehood function ($f(\boldsymbol{x}_i|\boldsymbol{\theta}_k)$). That is to say, to evaluate each datapoint responsibility which is conditionally dependant on the current parameters $\boldsymbol{\theta}$.
2. **Maximization step:** To find the parameters $\boldsymbol{\theta}$ that maximize the likehood, using previous ML results.

Given all datapoints from a single class, its likehood has the following expression, assuming that they are generated independently:

$$p(\boldsymbol{X}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \prod_{i=1}^{N} \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu},\boldsymbol{\Sigma}) \tag{1.11}$$

The former equation conduces to numerical problems, when many of the probabilities are close to 0. For so, the log-likehood is used. Because, as seen in the picture at the right, the logarithm function is monotonically increasing.

Therefore, as stated before, maximizing the likehood function is the same than maximizing the log-likehood. Due to the properties of the log function, this method eliminates the numerical issue.

$$\ln\big(p(\boldsymbol{X}|\boldsymbol{\mu},\boldsymbol{\Sigma})\big) = \sum_{i=1}^{N} \ln\big(\mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu},\boldsymbol{\Sigma})\big) \tag{1.12}$$

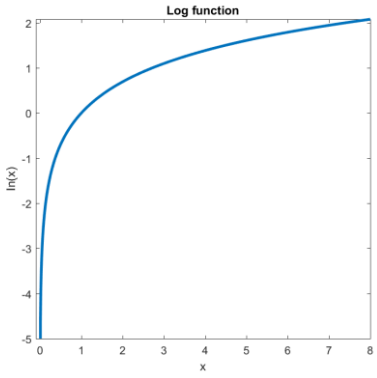


Figure 7 Logarithm function

A more detailed derivation of this function can be found in Appendix A. Where also derivatives with respect to the parameters are calculated. Further reference about this kind of approaches can be found in [3].

These algorithms are proven useful when model is well defined and the number of components K is known or can be estimated accurately. This is the reason why they are called **model-driven clustering algorithms**. In these models, the objective is to maximize the likehood or posterior as traditional maximum likehood and MAP algorithms do. However, the issue is still the same: **How many clusters are there? Can we infer its characteristics when we do not know how many of them there are?**

2 MIXTURE MODELS

"I just wondered how things were put together"

- C. C. Shannon -

Another approach to mixture models is to sample each datapoint parameters from a probability mass function, known as the **parameters probability mass function**. This new approach could be extended to a possible infinite number of componets, as we will see in this chapter. This types of algorithms are called **non-parametric clustering algorithms** and will be the main concern of this section.

Non-parametric models provide a suitable approach to clustering, when the number of components is unknown or is not well defined. Instead of sampling from a finite set of possible components, we define a non-parametric prior of the components, and then we sample a possibly infinite number of components parameters in order to fit our model.

To do so, we will use a kind of process known as the **Dirichlet Process** named after Peter Gustav Lejeune Dirichlet who first introduced them, along with the Dirichlet Distribution. This will allow our model to fit as many clusters as needed for the given data, this approach is sometimes referred as **data-driven clustering algorithms** as it only depends on the available data and makes less assumptions on the statistical data model.

In this section a non-traditional way to define mixture models will be introduced, their aim is to provide a probability mass function (PMF) from where to **sample the parameters used for each datapoint**, which in turn is the assigned class. When the PMF has always the same amount of elements we still have to infer how many of them we want. That is to say how many classes compose out system. But if we could make this model provide an infinite amount of clusters as we will do we eliminate the dependency on K. The rest of the section is destined to introduce the required tools for using these kind of algorithms.

2.1 Finite mixture models

A finite mixture model (FMM) is composed by a given number of components, usually denoted as K. In a finite mixture model we can define, for each datapoint, a hidden or latent random variable z_i which assigns each datapoint to a component of the mixture. This is the traditional approach to mixture models. But there is also another way to define these mixture models. For each datapoint, $\boldsymbol{x}_i$, a component parameter $\overline{\boldsymbol{\theta}}_i$ is sampled from a parameters PMF, $G(\boldsymbol{\theta})$, composed by all possible parameters and their weights. Having $\overline{\boldsymbol{\theta}}_i = \boldsymbol{\theta}_k$ is equivalent to say that sample i belongs to component k.

A relationship between both approaches is straightforward using the following equations, as denoted in [3].

$$p(\boldsymbol{x}_i|z_i = k, \boldsymbol{\theta}) = p(\boldsymbol{x}_i|\,\overline{\boldsymbol{\theta}}_i = \boldsymbol{\theta}_k) \tag{2.1}$$

$$p(z_i = k|\boldsymbol{\pi}) = p(\pi_k) \tag{2.2}$$

$$p(\boldsymbol{\pi}|\alpha) = \mathrm{Dir}(\boldsymbol{\pi}|(\alpha/K)\mathbf{1}_K) \tag{2.3}$$

Under this approach, (2.1) describes the relationship between the cluster assignments and the datapoint parameters. In the right hand side of the equation we omit RV z_i and directly map each sample parameter $\bar{\boldsymbol{\theta}}_i$ to its generating set of parameters. Moreover, the probability of a sample belonging to cluster k, is the same than the probability of the sample being generated by component with parameters $\boldsymbol{\theta}_k$.

The second equation is the prior over the cluster assignments. It states that the *a priori* probability of assigning a datapoint to cluster k is given by its weight in the mixture π_k.

Finally, last equation defines the probability of a given set of weights, which follows a Dirichlet Distribution [4].

In fact, (2.1) can be seen as a sample generated by a probability function with parameters $\boldsymbol{\theta}_k$

$$\boldsymbol{x}_i \sim F(\boldsymbol{\theta}_k) \tag{2.4}$$

The set of N samples, also denoted as $\boldsymbol{X}$, corresponds to the datapoints available. We can also suppose that the parameters $\boldsymbol{\theta_k}$ are randomly sampled from a prior over the cluster parameters, usually denoted as $H(\boldsymbol{\lambda})$. Although $H(\boldsymbol{\lambda})$ does not have to be discrete we take K finite number of samples from it, for a finite number of clusters. Thus

$$\boldsymbol{\theta}_k \sim H(\boldsymbol{\lambda}) \tag{2.5}$$

Where $H(\boldsymbol{\lambda})$ is chosen to be the conjugate prior[1] of $F(\boldsymbol{\theta}_k)$ so $p(\boldsymbol{x}_i)$ can be written as

$$p(\boldsymbol{x}_i) = \sum_{k=1}^{K} f(\boldsymbol{x}_i|\boldsymbol{\theta}_k) p(\boldsymbol{\theta}_k|\lambda) \tag{2.6}$$

Unfortunately, **this approach also holds the dependency on the number of components** K, which we want to avoid. Finite mixture models are always composed by K classes, which is the support for $G(\boldsymbol{\theta})$. The value of $G(\boldsymbol{\theta})$ for class k is π_k as denoted in (2.2). Therefore, we can define $G(\boldsymbol{\theta})$ as

$$G(\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \delta(\boldsymbol{\theta} - \boldsymbol{\theta}_k) \tag{2.7}$$

Where $p(\boldsymbol{\pi}|\alpha)$ follows a Dirichlet Distribution, as stated in (2.3) and $\boldsymbol{\theta}_k \sim H(\boldsymbol{\lambda})$.

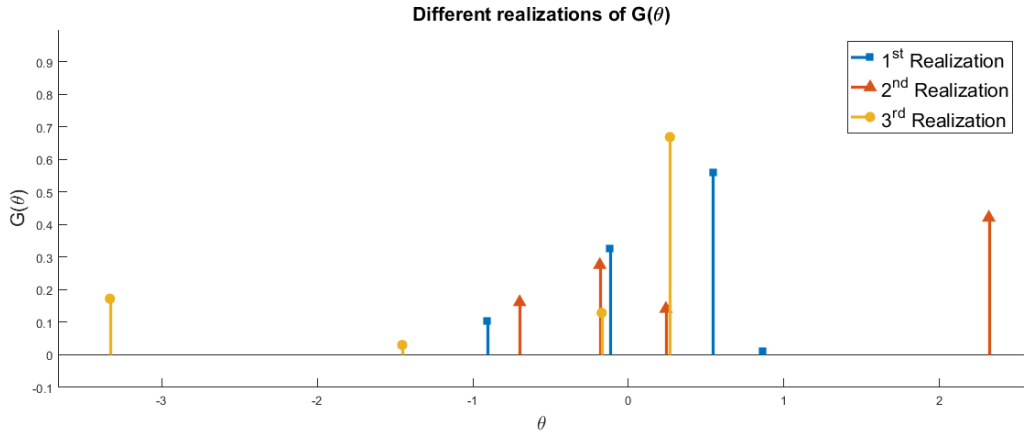


Figure 8 Realizations of $G(\boldsymbol{\theta})$

Note that all realizations of $G(\boldsymbol{\theta})$ generated **the same amount of samples**, which is the amount of components defined in our model. Then, for each datapoint we will sample $G(\boldsymbol{\theta})$ and obtain a datapoint parameter $\bar{\theta}_i$.

[1] Conjugate priors are those in which the posterior of the data are in the same family than the prior. In such cases, the prior is called to be conjugate prior of the likehood function. This is in fact an algebraic convenience, thus it provides a closed and integrable form for the posterior. Otherwise, numerical methods should be used to compute it.

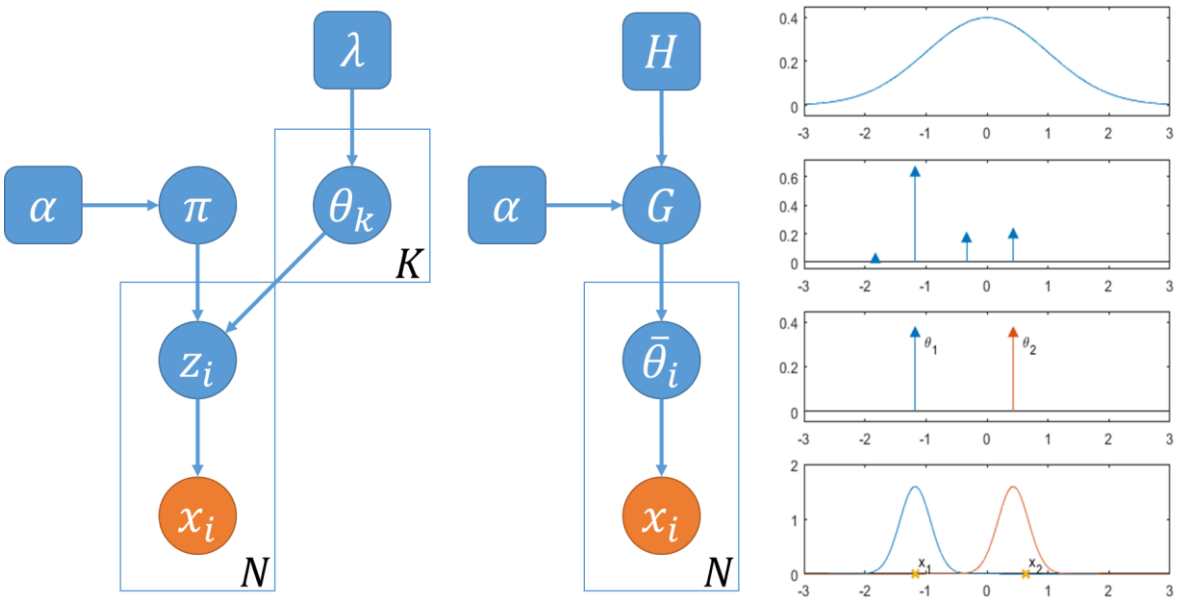


Figure 9 Representations of a Finite Mixture Model (source: [3])

The picture above shows the two different ways for modelling a finite mixture. The leftmost diagram shows how the model is built using the latent variable z_i which states the relationship between the datapoint and the component parameter θ_k through equation (2.1). The right model represents the mixture where the component parameter is sampled from the probability mass function $G(\boldsymbol{\theta})$, obtaining parameters for each datapoint $\bar{\boldsymbol{\theta}}_i$. Also remark that, $G(\boldsymbol{\theta})$ is a realization from a Dirichlet Distribution with Gaussian prior, $H(\boldsymbol{\lambda})$,and concentration parameter α. Finally, two datapoints are selected where $x_i \sim \mathcal{N}(\bar{\theta}_i, 0.25)$.

2.2 Infinite mixture models

Although the previous approach might seem suitable for a variety of problems; it has the same limitations when concerning to the uncertainty of K as the traditional way, which is the fact that the number of classes is fixed. Unlike this, **infinite mixture models** (IMM) do not impose any restriction on the number of components of the mixture. In fact, they provide more flexible model, which will be used when the number of components involved in the mixture is unknown. When using IMM the number of components is dynamically changed to best fit the data. This does not mean that IMM detects the number of components involved in the mixture but rather it provides an estimation on their amount through a PMF.

Infinite mixture models can be seen as a generalization of the second approach to FMM described in section 2.1. Probability mass function $G(\boldsymbol{\theta})$ is replaced by a random probability measure, known as **Dirichlet process** (DP). The DP is a generalization of the **Dirichlet Distribution** to infinite dimensions, which will be explained next.

Instead of using a finite mixture function for generating the PMFs of the components parameter, we will make use of the DP which produces a non-deterministic and possibly infinite number of components. This approach will make the model more suitable for problems in which the number of components is unknown.

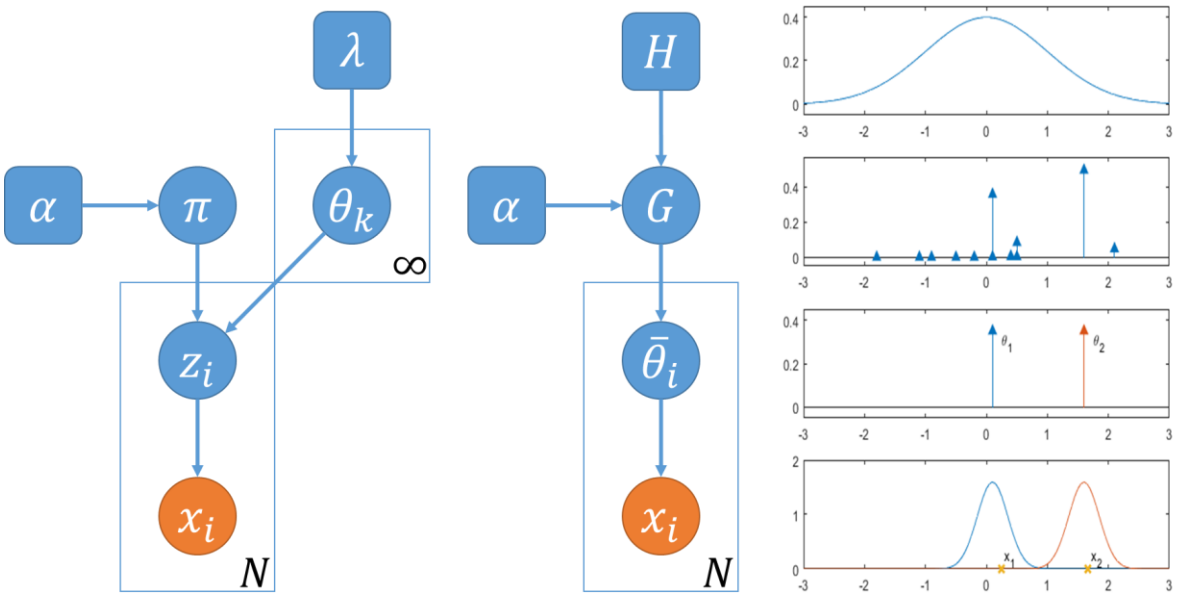


Figure 10 Representation of an infinite mixture model (source: [3])

Figure 10 shows how IMM are essentially equal to FMM, the only difference strives in the infinite amount of possible values of $\boldsymbol{\theta}_k$ that can be used and the changes required in $G(\boldsymbol{\theta})$ in order to be able to sample upto infinite clusters. Although this can be misleading, this approach does not impose an infinite number of clusters but it rather has the ability to provide as many clusters as needed to fit the data.

2.3 The Dirichlet Process

As defined below, the probability mass function $G(\boldsymbol{\theta})$, with prior $H(\boldsymbol{\lambda})$ and concentration parameter α, from which we will sample the values of $\boldsymbol{\theta}_k$ will be sampled from a Dirichlet Process. This is denoted as $G(\boldsymbol{\theta}) \sim DP(\alpha, H)$. Using the DP, we will be able to sample a PMF. A DP is a kind of stochastic process which produces probability mass functions with a different amount of elements. But before going into further details, it is better to know what is a Dirichlet Distribution.

2.3.1 Dirichlet Distribution

In the previous description of the finite mixture models it was mentioned that $G(\boldsymbol{\theta})$ values, $\boldsymbol{\pi}$, is the result of a realization of a Dirichlet Distribution with concentration parameter $\boldsymbol{\alpha}$. A PMF is just a function that assigns a number to a discrete RV (DRV) value, which in turn is the probability of the event liaised to the value of the DRV to happen. That is to say, given a finite set of possible events $S \in \{S_1, S_2, \dots, S_n\}$ first a n-dimensional real value is assigned to each event. Secondly, the probability of the event is the value of the PMF at this point.

Any discrete function can be a PMF, as long as the following two constrains hold

$$\sum_{\Omega} p(\boldsymbol{X} = \boldsymbol{\omega}) = 1 \tag{2.8}$$

$$p(\boldsymbol{X} = \boldsymbol{\omega}) \geq 0 \; \forall \boldsymbol{\omega} \tag{2.9}$$

(2.8) states the requirement that, for the whose sample space, Ω, the sum of the values of the PMF equals to 1. Also, as stated in (2.9) all the values must be positive or 0[2].

[2] Only the most purists would assign a probability of 0 for an event that will never happen, it would be simpler to just do not take into account an event that will never happen.

As the PMF models a random set of events, a Dirichlet Distribution is a PMF whose events are PMFs.

An example of a case where we can use a Dirichlet Distribution would be the following

---

We can model the result of rolling a dice (with values between 1 and 6) and non-uniform probabilities (as we can say that no dice is perfect) as a PMF.

We can then put a bunch of dices in a bag, each representing a different PMF, then extract a dice from the bag. This gives us a PFM. Therefore, the extraction of the dice can be modelled with a Dirichlet Distribution.

More formally, as described in [5], we say that a probability mass function with k components lies on the (k-1)-dimensional probability simplex, which is a surface in $\mathbb{R}^k$ where (2.8) and (2.9) holds. This is denoted as Δ_k.

$$\Delta_k = \{q \in \mathbb{R}^k | \sum_{i=1}^{k} q_i = 1, q_i \geq 0 \; for \; i = 1,2,\dots,k\} \tag{2.10}$$

Dirichlet distribution: *Let* $Q = [Q_1, Q_2, \dots, Q_k]$ *be a random pmf, that is* $Q_i \geq 0$ *for* $i = 1, 2, \dots, k$ *and* $\sum_{i=1}^{k} Q_i = 1$. *In addition, suppose that* $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \dots, \alpha_k]$, *with* $\alpha_i > 0$ *for each* i, *and let* $\alpha_0 = \sum_{i=1}^{k} \alpha_i$. *Then,* Q *is said to have a Dirichlet distribution with parameter* $\boldsymbol{\alpha}$, *which we denote by* $Q \sim Dir(\boldsymbol{\alpha})$, *if it has* $f(q; \boldsymbol{\alpha}) = 0$ *if* q *is not a PMF, and if* q *is a PFM then*

$$f(q; \boldsymbol{\alpha}) = \frac{\Gamma(\alpha_0)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \prod_{i=1}^{k} q_i^{\alpha_i - 1} \tag{2.11}$$

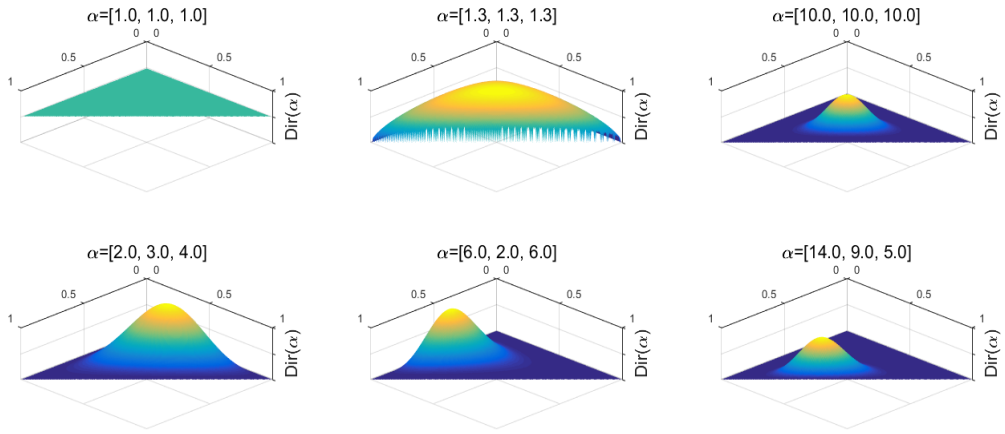


Figure 11 Examples of a Dirichlet Distribution over the 2D probability simplex for different values of α

2.3.2 Dirichlet Process Definition

As mentioned before, a Dirichlet Process is a type of stochastic process whose realizations are PMF but with no limit on the potential number of components. That is the reason why it is used in IMM [6]. It is an extension of the Dirichlet Distribution over a possibly infinite set of events.

As the Dirichlet Distribution models the extraction of a dice (along with its corresponding PMF) from a bag filled with 6-sided dices. A Dirichlet Process can be seen as the extraction of a dice from a bag where there could be dices with an upto infinite number of sides.

It is better to see the DP as a model, for example, for querying people about their favourite colour, as there is an infinite number of possible colours each person can choose, but and individual will choose from a limited set of them.

In a more mathematical way, a Dirichlet Process is a type of stochastic process whose realizations are distributions over an arbitrary sample space [5]. The Dirichlet Process is specified by 2 components:

1. **A base distribution**, which in fact is the expected value of the process. Recall that it returns PMFs.
2. **A concentration parameter $\boldsymbol{\alpha}$**, which specifies how discretized the process is. In practice, how many points are generated by the process.

Dirichlet process [7]: *Let H be a distribution over Θ and α be a positive real number. Then for any finite measurable partition $A_1, \dots, A_r$ of Θ, the vector $\big(G(A_1), \dots, G(A_r)\big)$ is random since G is random. We say G is a Dirichlet Process distributed with base distribution H and concentration parameter α, written $G \sim DP(\alpha, H)$, if*

$$\big(G(A_1), \dots, G(A_r)\big) \sim Dir\big(\alpha H(A_1), \dots, \alpha H(A_r)\big) \tag{2.12}$$

For every finite measurable partition $A_1, \dots, A_r$ of Θ.

Note that, here $G(\cdot)$ indicates a Dirichlet Process and r is the number of partitions of the space Θ. Furthermore, the concentration parameter α is a measure of the discretization of this space.

As the normal distribution draws real numbers close to the mean value, **the DP generates PMFs around the base distribution**. As the concentration parameter increases the process returns more and more values and when $\alpha \to \infty$ the returned PMFs become the continiuos base distribution.

Figure 12 illustrates how, as more samples are generated the histograms tends to fit the base distribution, this is due to the fact that $\mathbb{E}[DP(\alpha, H)] = H$.

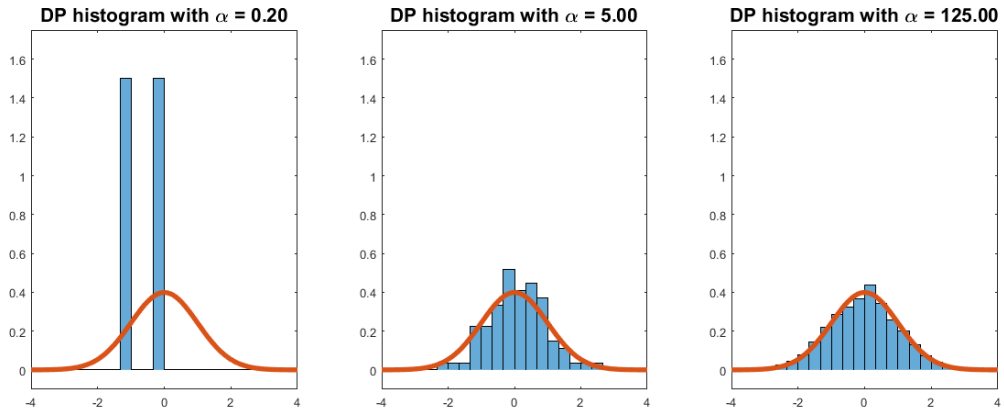


Figure 12 Histogram of $G \sim DP(\alpha, H)$ with respect to α.

2.3.3 Predictive Distribution

One result that will be used in next sections requires the definition of a predictive distribution for the DP. What we want to model is the fact that, as we get more and more samples from the DP, our knowledge about G increases. Thus, the predictive distribution takes the form

$$p(\boldsymbol{\theta}_{n+1} | \boldsymbol{\theta}_n, \dots, \boldsymbol{\theta}_1, \alpha, H) \tag{2.13}$$

We will start by assuming that N samples have been already generated from a DP, named $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_N$. Let $A_1, \dots, A_r$ be a finite measurable and arbitrary partition of Θ into R parts, as stated in the definition of the DP. And let $n_k = |\{i : \theta_i \in A_k\}|$ be the number of samples within partition A_k. The likehood function is multinomial because partitions are finite. By conjugacy of Dirichlet and multinomial, as stated in [4], we have

$$\big(G(A_1), \dots, G(A_r)\big) | \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_n \sim Dir(\alpha H(A_1) + n_1, \dots, \alpha H(A_r) + n_r) \tag{2.14}$$

Therefore, since this is valid for any finite partition of Θ, the posterior of the DP over $G(\boldsymbol{\theta})$ is also a DP. Note that, all the parameters have a constant sum, which does not depend on the partition scheme:

$$\sum_{i=1}^{r}(\alpha H(A_i)+n_i)=\alpha+N \tag{2.15}$$

Proof

$$\sum_{i=1}^{R}\alpha H(A_i)+\sum_{i=1}^{R}n_i=\alpha+N \tag{2.16}$$

$$\alpha\sum_{i=1}^{R}H(A_i)+\sum_{i=1}^{R}n_i=\alpha+N \tag{2.17}$$

Then, the first term is the summation of $H(A_i)$ for the whole sample space, which is always equals to 1, as H is a PDF. Also, the second sum equals the total number of samples, N, as we are also summing up the whole sample space.

We can try to find a new base distribution H' such as, for every partition

$$\alpha H(A_i)+n_i=\alpha' H'(A_i)\ \forall i=1,\dots r \tag{2.18}$$

If it exists, it follows that the predictive distribution also is a DP as stated before, it is easy to see that

$$\alpha'=\alpha+N \tag{2.19}$$

$$H'(\boldsymbol{\theta})=\frac{\alpha H(\boldsymbol{\theta})+\sum_{i=1}^{n}\delta(\boldsymbol{\theta}-\boldsymbol{\theta}_i)}{\alpha+N} \tag{2.20}$$

And therefore

$$G(\boldsymbol{\theta})|\boldsymbol{\theta}_1,\dots,\boldsymbol{\theta}_n\sim DP\left(\alpha+N,\frac{\alpha H(\boldsymbol{\theta})+\sum_{i=1}^{n}\delta(\boldsymbol{\theta}-\boldsymbol{\theta}_i)}{\alpha+N}\right) \tag{2.21}$$

The proof for this expression can be found in Appendix B. Then, for a new sample $\boldsymbol{\theta}_{n+1}$, its probability to belong to partition A is

$$p(\boldsymbol{\theta}_{n+1}\in A|\boldsymbol{\theta}_1,\dots,\boldsymbol{\theta}_n,\alpha,H)=\mathbb{E}[G(A)|\boldsymbol{\theta}_1,\dots,\boldsymbol{\theta}_n]=\frac{\alpha H(A)+\sum_{i=1}^{n}\delta(A-\boldsymbol{\theta}_i)}{\alpha+N} \tag{2.22}$$

$\delta(A-\boldsymbol{\theta}_i)$ is a generalization of the δ function, that takes value 1 if $\boldsymbol{\theta}_i\in A$, and 0 otherwise. Provided that, the expected value of the DP is the base distribution H. And taking $A=\boldsymbol{\theta}$ we reach the most general expression for the predictive distribution

$$p(\boldsymbol{\theta}|\boldsymbol{\theta}_1,\dots,\boldsymbol{\theta}_n,\alpha,H)\sim\frac{\alpha H(\boldsymbol{\theta})+\sum_{i=1}^{n}\delta(\boldsymbol{\theta}-\boldsymbol{\theta}_i)}{\alpha+N} \tag{2.23}$$

2.4 Sampling the DP

Although describing the Dirichlet Distribution and the Dirichlet Process is helpful, when using IMM we are more interested in how to sample the DP, that is, how can we generate random PMFs from a DP. In this section we will cover up two different approaches, the Stick-Breaking Construction (SBC) and the Chinese Restaurant Process (CRP) which is also known as the Pólya Urn Scheme.

2.4.1 The Stick-Breaking Construction

Let $Beta(\alpha, \beta)$ be the continuous beta distribution. From a given base distribution H and a concentration parameter α. We will define $\boldsymbol{\pi} = \{\pi_k\}_{k=1}^{\infty}$, that is, an infinite sequence of mixture weights derived from the following schema [4]:

$$\beta_k \sim Beta(1, \alpha) \tag{2.24}$$

$$\pi_k \sim \beta_k \prod_{l=1}^{k-1}(1-\beta_l) = \beta_k \left(1 - \sum_{l=1}^{k-1} \pi_l\right) \tag{2.25}$$

Then, once the weights are obtained, we assign a value for the PMF, sampling the base distribution $H(\boldsymbol{\theta})$

$$\boldsymbol{\theta}_k \sim H(\boldsymbol{\theta}) \tag{2.26}$$

$$p(\boldsymbol{\theta}) = \sum_{k=1}^{\infty} \pi_k \delta(\boldsymbol{\theta} - \boldsymbol{\theta}_k) \tag{2.27}$$

The algorithm for generating samples is as follows, if ε is taken as a small number:

ALGORITHM 1: STICK-BREAKING CONSTRUCTION

Set $\pi_0 = 0, k = 1, l = 1$
while $l > \varepsilon$
    Sample β_k from $Beta(1, \alpha)$
    Compute $\pi_k = \beta_k \times l$
    Sample θ_k from $H(\lambda)$
    Decrement l by π_k
    Increment k
end

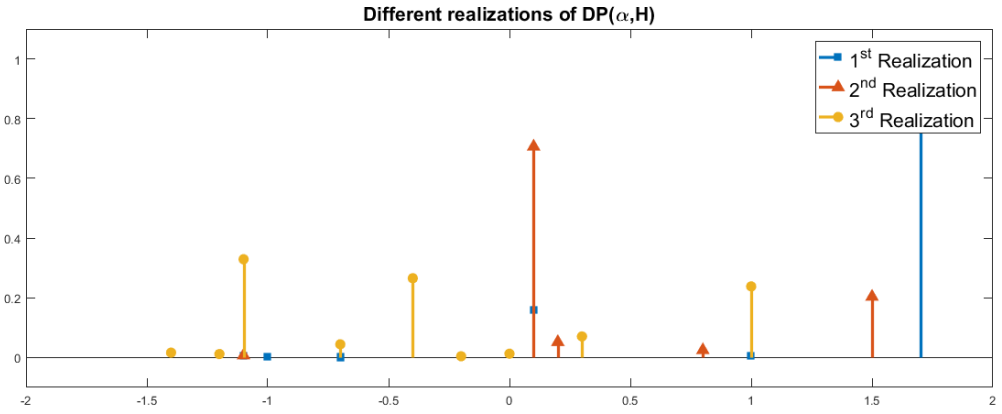


Figure 13 Different realizations of $DP(\alpha, H)$ where $H(\lambda) = \mathcal{N}(0, 1)$, $\alpha = 0.6$

Figure 13 shows different realizations of the Dirichlet Process using the SBC. Note that, the number of samples generated in each turn is different, in the previous example the DP produced 5, 5 and 9 samples respectively.

Due to its construction scheme, the SBC produces a PMF with $\sum_{i=1}^{\infty} \pi_k = 1$ and $\boldsymbol{\theta}_k$ following the base distribution $H(\lambda)$.

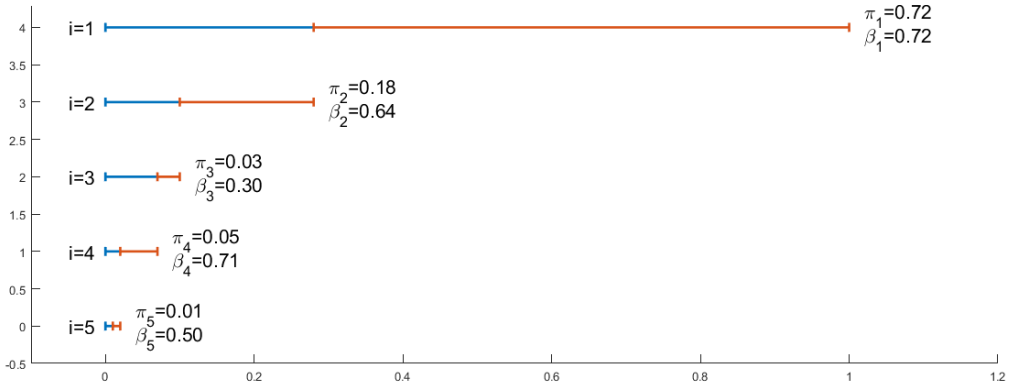


Figure 14 Stick Breaking Construction of the DP

The further we increment concentration parameter α, the more samples are generated and the PDF estimation (histogram) is closer to the base distribution, this is coherent with the predictive properties defined above.

An interesting result, which will be used in the next section, is that **this construction produces discrete PMFs with probability equals to 1** [4]. In fact, if we take samples from $G(\boldsymbol{\theta})$ we will get more and more repetitions of previously seen values, although due to the infiniteness of the SBC there is still a small odd of generating a new sample not seen before.

2.4.2 The Chinese Restaurant Process

Another approach when sampling a DP, which avoids the problem of infinitely breaking a stick into smaller chunks, takes advantage of the discreteness of $G(\boldsymbol{\theta})$ and builds the predictive model from it. From (2.23), taken all previously generated samples of $G(\boldsymbol{\theta})$, we can model the predictive probability of a new sample as

$$p(\boldsymbol{\theta}_{N+1}|\boldsymbol{\theta}_1, \dots, \theta_N, \alpha, H) = \frac{1}{\alpha + N}\left(\alpha H(\boldsymbol{\theta}) + \sum_{k=1}^{K} N_k \delta(\boldsymbol{\theta} - \boldsymbol{\theta}_k)\right) \tag{2.28}$$

Where N_k is the number of previous samples equal to $\boldsymbol{\theta}_k$. This equation could be easier understood using hidden variable z_i shown before, which assigns a sample and its parameter to a set of parameters. We redefine the variable in the following form

$$z_i = k \overset{iff}{\leftrightarrow} \boldsymbol{x}_i \; was \; generated \; with \; parameter \; \boldsymbol{\theta}_k \tag{2.29}$$

Doing so, it allows us to rewrite (2.28) as

$$p(z_{i+1} = q|z_1, \dots, z_i, \alpha) = \frac{1}{\alpha + N}\left(\alpha \mathbb{I}(q = k^*) + \sum_{k=1}^{K} N_k \mathbb{I}(q = k)\right) \tag{2.30}$$

Where k^* indicates a new cluster not seen before and $\mathbb{I}(\cdot)$ is the indicator function, it returns 1 when the inner condition holds and 0 otherwise. In other words

$$p(z_{i+1} = q|z_i, .., z_1, \alpha) = \begin{cases} \dfrac{N_k}{\alpha + N} & if \; k \; has \; been \; seen \; before \\ \dfrac{\alpha}{\alpha + N} & if \; k \; is \; a \; new \; cluster \end{cases} \tag{2.31}$$

This method is called the **Chinese Restaurant Process (CRP)** due to the model proposed by Aldous in [8] in 1983. It is described with an analogy of a Chinese restaurant with an infinite supply of tables. The first customer sits on the first table. Whenever a new customer arrives, he sits in an existing table with a probability proportional to the number people already sat on that table. However, there is always a small probability of sitting in a new table, which is given by the concentration parameter α, as seen in (2.31).

As more clusters appear, the probability of having a new cluster decreases, although it remains bigger than zero. Depending on parameter α, this probability decreases faster or slower. So controlling α allows us to approximately infer the number of cluster that will define our model.

Using the CRP, we will be able to sample the DP defining the predictive PMF over the latent variable z, that is

$$p(z_i = k|\boldsymbol{z}_{-i}, \alpha) \tag{2.32}$$

Where $\boldsymbol{z}_{-i} \overset{\text{def}}{=} \{z_1, \dots, z_i\}$. Which is the same as the predictive PMF over the parameters, as stated in (2.29).

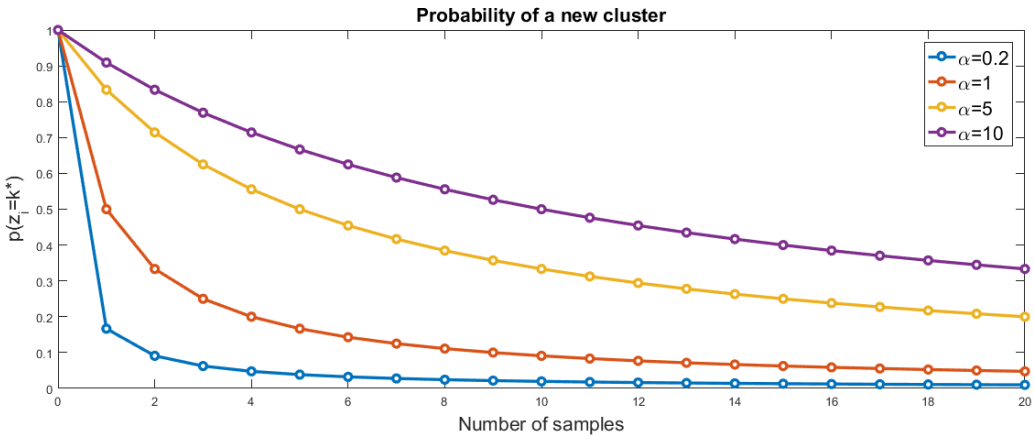


Figure 15 Probability of a new cluster with respect to α

Figure 15 shows how the probability of a new clustes quickly decreases when α is small. It also shows how, as α increases the probability tends to 0 slower.

2.5 Collapsed Gibbs Sampling

Another required tool when using IMM is the Gibbs Sampler. **The Gibbs Sampling is an algorithm for efficiently obtaining samples from a multivariate probability function**. It is used when direct sampling the PDF is difficult or computationally too expensive. The Gibbs Sampler belongs to the family of the MCMC algorithms as it tries to build a random walk of a Markov Chain for generating the samples.

2.5.1 Markov Chains

A Markov Chain is a type of stochastic process which complies with the Markov property, also known as the "memoryless property". This means that each step/sample only depends on the previous one. It is defined by a set of states, in a state space in which the Markov Chain can move. More precisely, if we define $\boldsymbol{X}_1, \boldsymbol{X}_2, \dots$ as the possible states of the Markov Chain, then the Markov property says that

$$p(\boldsymbol{X}_i = \boldsymbol{x}|\boldsymbol{X}_1, \dots, \boldsymbol{X}_{i-1}) = p(\boldsymbol{X}_i = \boldsymbol{x}|\boldsymbol{X}_{i-1}) \tag{2.33}$$

Markov Chains that are stationary, also require that

$$p(\boldsymbol{X}_i = \boldsymbol{x}|\boldsymbol{X}_{i-1}) = p\big(\boldsymbol{X}_j = \boldsymbol{x}\big|\boldsymbol{X}_{j-1}\big)\ \forall i, j \tag{2.34}$$

That is to say that transition probabilities do not change with time. If these properties hold, then a Markov Chain can be defined by a set of transition probabilities p_{ij} which determines the probability of moving from state i to state j.

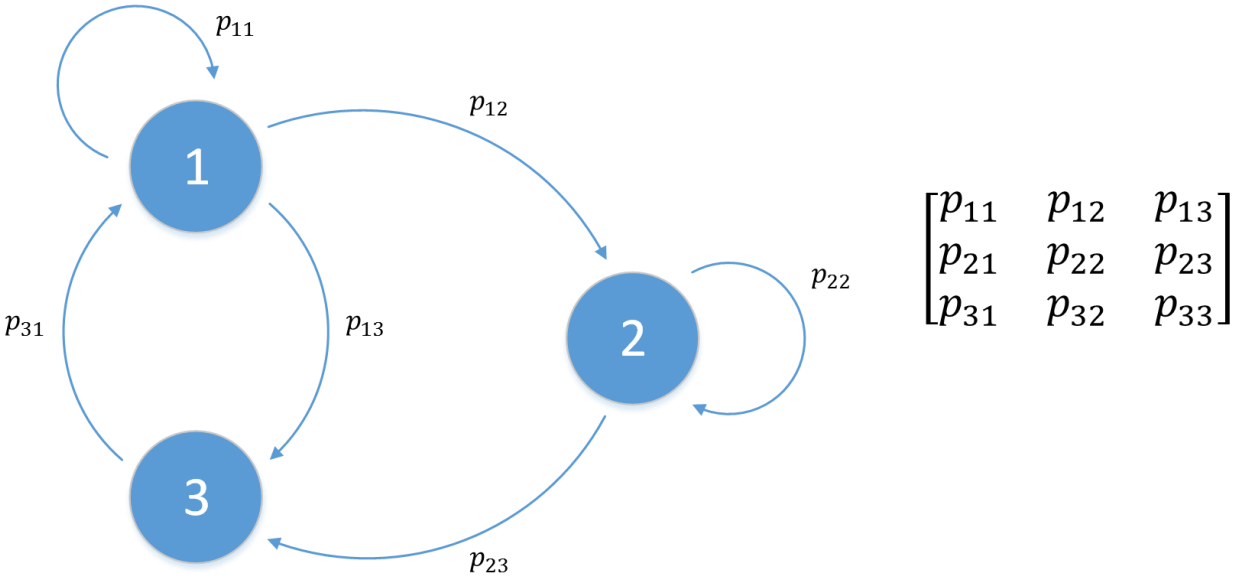


Figure 16 Graph of a Markov Chain with 3 states

Recall that, in a Markov Chain, not all transitions are necessarily possible and that the number of states does not have to be finite [9]. One property of the stationary Markov Chains is that they have a **stationary distribution** that we can approximated by considering the visited states as samples of this stationary PDF. Figure 17 shows how the Markov Chain samples can also be considered as samples of the stationary distribution. Where the upper picture shows the Markov Chain process and the bottom picture shows the histogram of the samples and the target stationary distribution. That is the reason why MCMC is a widespread family of algorithms for sampling complex PDF, the only requirement is that the underlying Markov Chain has the same stationary distribution as the target PDF.

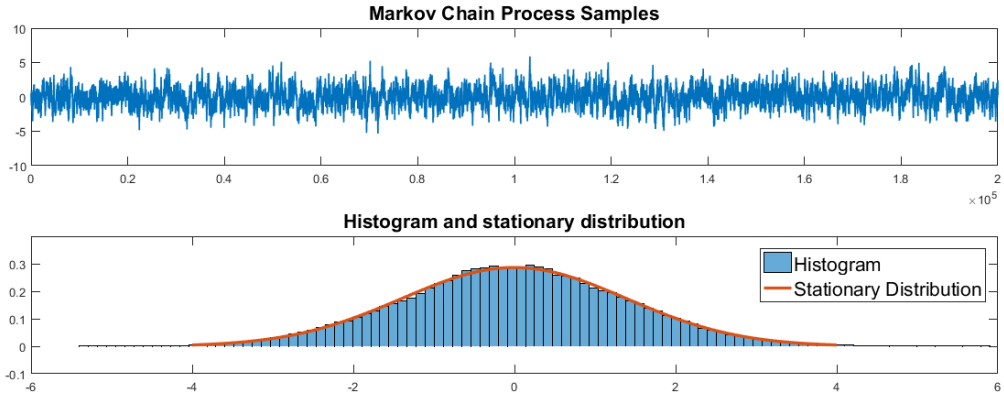


Figure 17 Markov Chain and its stationary distribution

But there is a small drawback in this process. What happens when we start our random walk at a very uncommon point? As the Markov Chains obtain the current sample from the previous one, a convergence time has to be considered. That is the reason why a burnin period, where samples are discarded, is required when using MCMC algorithms. Although it is not the smartest solution, it is simple. Using a burnin period, we just delay the beginning of our Markov Chain by n steps, but a new question arises. Which is a good starting point for our Markov Chain? As stated in [9]

Any point you do not mind having as a sample is a good starting point.

To show this issue, a simple Markov Chain has been built considering the following process [9]:

$$x_{n+1} = \rho x_n + y_n \tag{2.35}$$

Where y_n are i.i.d samples following a $N(0, \tau^2)$ distribution and $x_1 \sim N(0, \sigma^2)$. It is a Markov Chain, because the probability of sample x_{n+1} only depends on the previous state $[x_n, y_n]$.

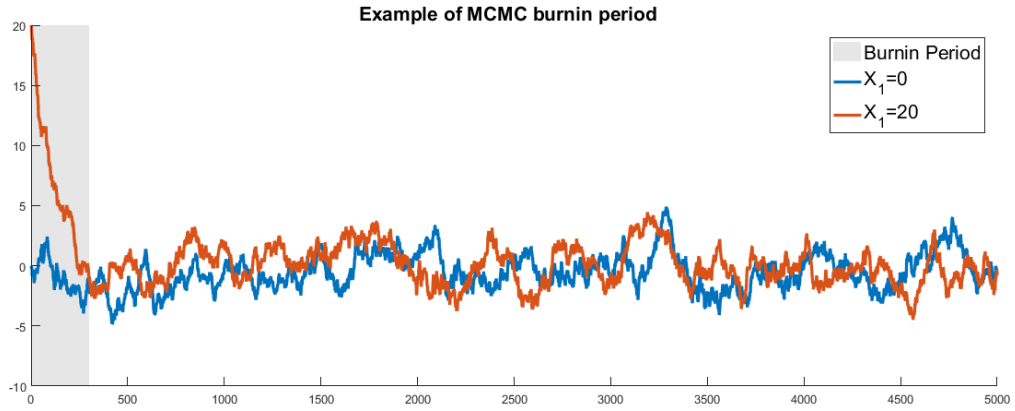


Figure 18 Example of MCMC burnin period

Figure 18 shows two different runs of the previous Markov Chain from and for different starting points. One with a high probability of happening and the other with a very small one. Note that they only differ in the initial phase with the grey background. This is the burnin period we are considering, during this time we will not take into account the samples generated by the process. After this period the Markov Chain reaches the equilibrium and samples can be considered an accurate result of the stationary PDF.

2.5.2 Gibbs Sampling

When using a Gibbs Sampler, we are trying to obtain a sample from a multivariate PDF, as stated before. Instead of sampling all the dimensions at the same time, we build a Markov Chain with the conditional probability of each variable with respect to the previous one.

Therefore, if we have a sample, $\boldsymbol{x}_i$, and we are trying to obtain the next sample, $\boldsymbol{x}_{i+1}$, and we have $D = 3$ dimensions to sample. We make use of the following conditional probabilities.

$$x_{i+1}^{(1)} \sim f\left(x^{(1)} \middle| x_i^{(2)}, x_i^{(3)}\right) \tag{2.36}$$

$$x_{i+1}^{(2)} \sim f\left(x^{(2)} \middle| x_i^{(3)}, x_{i+1}^{(1)}\right) \tag{2.37}$$

$$x_{i+1}^{(3)} \sim f\left(x^{(3)} \middle| x_{i+1}^{(1)}, x_{i+1}^{(2)}\right) \tag{2.38}$$

It easy to extend the previous formulae to the most general D-dimensional case. What we have to define is

$$p(x_i|\boldsymbol{x}_{-i}) \tag{2.39}$$

Which is called the **full conditional probability** for dimension i. And where $\boldsymbol{x}_{-i}$ stands for the samples before x_i.

An interesting case is the Collapsed Gibbs Sampler. It is a modification over the traditional Gibbs Sampler where some of the dimensions have been marginalized out, turning the algorithm much more efficient [4]. In such case, if we can integrate out one or more dimensions of the sample space, the marginalized variables do not belong to the Markov Chain and can be removed.

2.5.3 Collapsed Gibbs Sampling in FMM

As stated before, a mixture is defined by a set of PDF parameters, $\boldsymbol{\theta}_k$, one for each component in the mixture and additionally $\boldsymbol{z}$ and $\boldsymbol{\pi}$, which are the latent cluster-assignment variable and the weights of the mixture. If we try to build a Gibbs Sampler for the latent variable $\boldsymbol{z}$, what we are trying to compute the following equation

$$p(z_i = k|\boldsymbol{z}_{-i}, \boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\lambda}) \tag{2.40}$$

Where λ and α and are, respectively, the hyperparameters for the data and the class-conditional densities. That

is, $\boldsymbol{\alpha}$ is only relevant when computing $\boldsymbol{z}$ and $\boldsymbol{\lambda}$ is only relevant when computing $\boldsymbol{x}$. Then, if we split previous equation using the Bayes theorem in the following way

$$p(z_i = k|\boldsymbol{z}_{-i}, \boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\lambda}) \propto p(z_i = k|\boldsymbol{z}_{-i}, \boldsymbol{\alpha}, \boldsymbol{\lambda}) p(\boldsymbol{x}|z_i = k, \boldsymbol{z}_{-i}, \boldsymbol{\alpha}, \boldsymbol{\lambda}) \tag{2.41}$$

As $p(z_i = k| \dots)$ does not depend on $\boldsymbol{\lambda}$ and $p(\boldsymbol{x}| \dots)$ does not depend on $\boldsymbol{\alpha}$ we can remove them from the previous expression.

$$p(z_i = k|\boldsymbol{z}_{-i}, \boldsymbol{x}, \alpha, \lambda) \propto p(z_i = k|\boldsymbol{z}_{-i}, \boldsymbol{\alpha}, \boldsymbol{\lambda}) p(\boldsymbol{x}|z_i = k, \boldsymbol{z}_{-i}, \boldsymbol{\lambda}) \tag{2.42}$$

Furthermore, we can rewrite the second part as

$$p(\boldsymbol{x}|z_i = k, \boldsymbol{z}_{-i}, \boldsymbol{\lambda}) = p(x_i = x|\boldsymbol{x}_{-i}, z_i = k, \boldsymbol{z}_{-i}, \boldsymbol{\lambda}) p(\boldsymbol{x}_{-\boldsymbol{i}}|z_i = k, \boldsymbol{z}_{-i}, \boldsymbol{\lambda}) \tag{2.43}$$

Where x_i corresponds to the current sample and $\boldsymbol{x}_{-i}$ is a vector containing all samples except i. Then, as $\boldsymbol{x}_{-i}$ does not depend on the current assignment, z_i it acts as a constant for the equation, thus

$$p(\boldsymbol{x}|z_i = k, \boldsymbol{z}_{-i}, \boldsymbol{\lambda}) \propto p(x_i = x|\boldsymbol{x}_{-i}, z_i = k, \boldsymbol{z}_{-i}, \boldsymbol{\lambda}) \tag{2.44}$$

And finally, we reach the final desired expression

$$p(z_i = k|\boldsymbol{z}_{-i}, \boldsymbol{x}, \alpha, \lambda) \propto p(z_i = k|\boldsymbol{z}_{-i}, \boldsymbol{\alpha}) p(x_i = x|\boldsymbol{x}_{-i}, z_i = k, \boldsymbol{z}_{-i}, \boldsymbol{\lambda}) \tag{2.45}$$

The Collapsed Gibbs Sampling algorithm is then described as follows,

ALGORITHM 2: COLLAPSED GIBBS SAMPLER (SOURCE: [3])

1	**for each** $i \in 1{:}N$ **do**	
2	Remove $\boldsymbol{x}_i$ from old cluster z_i	
3	**for each** $k \in 1{:}K$ **do**	
4	Compute $p(z_i = k|\boldsymbol{z}_{-i}, \alpha) = \frac{N_{k,-i}}{\alpha + N - 1}$	First term of (2.45)
5	Compute $p_k(\boldsymbol{x}_i) = p(\boldsymbol{x}_i|\boldsymbol{x}_{-i}(k))$	(2.44). Second term of (2.45)
6	**end**	
7	Normalize $p(z_i|\boldsymbol{z}_{-i}, \boldsymbol{x}, \alpha)$	(2.45)
8	Sample $z_i \sim p(z_i|\boldsymbol{z}_{-i}, \boldsymbol{x}, \alpha)$	
9	Add $\boldsymbol{x}_i$ to new cluster z_i	
10	**end**	

2.6 Collapsed Gibbs Sampling for IMM

Once all the required tools have been introduced, in this section we will describe how to fit an IMM to some datapoints. For so, we will make use of the Collapsed Gibbs Sampler explained above and introduce some modifications. Let

$$p(z_i = k|\boldsymbol{z}_{-i}, \boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{\lambda}) \propto p(z_i = k|\boldsymbol{z}_{-i}, \boldsymbol{\alpha}) p(\boldsymbol{x}_i|\boldsymbol{x}_{-i}, z_i = k, \boldsymbol{z}_{-i}, \boldsymbol{\lambda}) \tag{2.46}$$

Where $\boldsymbol{x}_{-i}$ and $\boldsymbol{z}_{-i}$ represent, respectively, the previous samples and the previous cluster assignments. As seen in the former section, equation (2.30) gives the probability of a new cluster

$$p(z_i = z|\boldsymbol{z}_{-i}, \boldsymbol{\alpha}) = \frac{1}{\alpha + N - 1} \left(\alpha \mathbb{I}(z = k^*) + \sum_{k=1}^{K} N_{k,-i} \mathbb{I}(z_i = k) \right) \tag{2.47}$$[3]

Where $\boldsymbol{\alpha} = \alpha$. A better way to write this equation is to use (2.31)

[3] The -1 added to the divisor is due to the fact that, in this case we are taking sample $\boldsymbol{x}_i$ and not sample $\boldsymbol{x}_{i+1}$.

$$p(z_i = z|\boldsymbol{z}_{-i}, \alpha) = \begin{cases} \dfrac{N_{k,-i}}{\alpha + N - 1} & if\ k\ has\ been\ seen\ before \\ \dfrac{\alpha}{\alpha + N - 1} & if\ k\ is\ a\ new\ cluster \end{cases} \tag{2.48}$$

The second term is more complex to calculate. First, we will divide datapoints based on the current cluster assignments. Let $\boldsymbol{x}_{-i,c} \stackrel{\text{def}}{=} \{x_j : z_j = c, j \neq i\}$ be the datapoints already assigned to the cluster c, except datapoint x_i. Then x_i is conditionally independent to all datapoints not in its current cluster [4], put in other words when computing probability of belonging to cluster k we only take into consideration samples already in k.

$$p(x_i|\boldsymbol{x}_{-i}, z_i = k, \boldsymbol{z}_{-i}, \boldsymbol{\lambda}) = p\left(x_i\middle|\boldsymbol{x}_{-i,k}, \boldsymbol{\lambda}\right) = \frac{p(x_i, \boldsymbol{x}_{-i,k}|\boldsymbol{\lambda})}{p(\boldsymbol{x}_{-i,k}, \boldsymbol{\lambda})} \tag{2.49}$$

This is achieved making use of the Bayes theorem, then

$$p\left(x_i, \boldsymbol{x}_{-i,k}\middle|\boldsymbol{\lambda}\right) = \int p(x_i|\boldsymbol{\theta}_k)\left[\prod_{j \neq i : z_j = k} p\left(x_j\middle|\boldsymbol{\theta}_k\right)\right] H(\boldsymbol{\theta}_k|\lambda) d\boldsymbol{\theta}_k \tag{2.50}$$

It is the marginal likehood of all data assigned to cluster k, including i. It is the numerator of (2.49). Also if datapoint i is excluded, it becomes the expression for the denominator.

If no previous samples are assigned to cluster k, which means that $z_i = k^*$, the expression becomes simpler

$$p(\boldsymbol{x}_i|\boldsymbol{x}_{-i}, z_i = k^*, \boldsymbol{z}_{-i}, \boldsymbol{\lambda}) = p(\boldsymbol{x}_i|\lambda) = \int p(\boldsymbol{x}_i|\boldsymbol{\theta}) H(\boldsymbol{\theta}|\lambda) d\boldsymbol{\theta} \tag{2.51}$$

This is very similar to Collapsed Gibbs Sampler algorithm except for the additional case $z_i = k^*$.

ALGORITHM 3: MODIFIED COLLAPSED GIBBS SAMPLER FOR IMM (SOURCE: [3])

1	**for each** $i \in 1{:}N$ **do**	
2	Remove $\boldsymbol{x}_i$ from old cluster z_i	
3	**for each** $k \in 1{:}K$ **do**	
4	Compute $p(z_i = k|\boldsymbol{z}_{-i}, \alpha) = \frac{N_{k,-i}}{\alpha+N-1}$	First case of (2.48)
5	Set $N_{k,-i} = \left\|\boldsymbol{x}_{-i,k}\right\|$	
6	Compute $p_k(\boldsymbol{x}_i) = p\left(\boldsymbol{x}_i\middle|\boldsymbol{x}_{-i,k}\right)$	(2.49)
7	**end**	
8	Compute $p(z_i = *\,|\boldsymbol{z}_{-i}, \alpha) = \frac{\alpha}{\alpha+N-1}$	Second case of (2.48)
9	Compute $p_*(x_i) = p(x_i|\lambda)$	(2.51)
10	Normalize $p(z_i|\boldsymbol{z}_{-i}, \boldsymbol{x}, \alpha)$	(2.46)
11	Sample $z_i \sim p(z_i|\boldsymbol{z}_{-i}, \boldsymbol{x}, \alpha)$	
12	Add $\boldsymbol{x}_i$ to new cluster z_i	
13	If any cluster is empty, remove it and decrease K	
14	**end**	

2.6.1 Fitting a GMM

The following section is a particularization of the equations shown above, computed for the Gaussian Mixture Model. Now, we will suppose that samples were generated by a Gaussian Mixture as described in section 1.2. The only equation that depends on the mixture model is the predictive distribution over the data. In this case, as the mean and covariance matrix are unknown, we will use a **InverseWishart-Gaussian Distribution** in order to compute the predictive PDF, as described in (2.49). Because the samples were generated using a DP, the values $\boldsymbol{\theta}_k$ are discrete, and then, what we have to compute is

$$p\left(x_i\middle|\boldsymbol{x}_{-i,k}, \boldsymbol{\lambda}\right) = \frac{p(x_i, \boldsymbol{x}_{-i,k}|\boldsymbol{\lambda})}{p(\boldsymbol{x}_{-i,k}, \boldsymbol{\lambda})} \tag{2.52}$$

Then, for each existing cluster, using function $Z(\cdot)$ described in Appendix C, we compute (2.52) using the following expression

$$p(x_i|\boldsymbol{x}_{-i,k}, \boldsymbol{\lambda}) = \frac{Z(d, n+1, r+1, \nu+1, S_{x_i, \boldsymbol{x}_{-i,k}})}{Z(d, n, r, \nu, S_{x_{-i,k}})} \tag{2.53}$$

If we want to obtain the mean and covariance matrix of a given cluster, once we have some samples in the cluster, we can use InverseWishart-Gaussian distribution to sample them using the following expressions

$$\boldsymbol{\Sigma}_k \sim \mathcal{W}^{-1}(\boldsymbol{R}, \nu) \tag{2.54}$$

$$\boldsymbol{\mu}_k|\boldsymbol{\Sigma}_k \sim \mathcal{N}\left(\frac{\sum_{i=1}^{N} x_i}{r}, \frac{\boldsymbol{\Sigma}_k}{r}\right) \tag{2.55}$$

Where $\boldsymbol{R}$ is the precision matrix of the data, ν is the degrees of freedom of the distribution and r is the relative precision of μ with respect to the data. That is, the precision of μ is $r\boldsymbol{R}$. Further reference can be found in [10].

We also have to compute $p(z_i = k|\boldsymbol{z}_{-i}, \alpha)$, which is always computed using the Chinese Restaurant Process results shown in (2.48)

$$p(z_i = k|\boldsymbol{z}_{-i}, \alpha) = \frac{N_{k,-i}}{\alpha + N - 1} \tag{2.56}$$

We are now ready to compute (2.46), which is the product of the previous results, (2.53) and (2.56). But the probability of the sample being in new unseen cluster has to be calculated too, which is the same as the probability of a new cluster. It can be efficiently calculated as the predictive distribution for the Gaussian-Wishart when no samples are provided, as can be seen in Appendix C.

$$p(x_i|\lambda) = \frac{Z(d, 1, r+1, \nu+1, S_x)}{Z(d, 0, r, \nu, S)} \tag{2.57}$$

And also, compute $p(z_i = * |\boldsymbol{z}_{-i}, \alpha)$

$$p(z_i = * |\boldsymbol{z}_{-i}, \alpha) = \frac{\alpha}{\alpha + N - 1} \tag{2.58}$$

Finally, we normalize the probabilities obtained. Recall that we are obtaining proportional values of the real probability, as described in (2.46). The next step is to sample the new cluster assignment z_i for the sample $\boldsymbol{x}_i$ using the probabilities considered before and add $\boldsymbol{x}_i$ to its new cluster.

As this process is done using the Collapsed Gibbs Sampler method explained above, as in other MCMC based methods, the initial samples have to be discarded.

3 ALGORITHM IMPLEMENTATION

"The best big idea is only going to be as good as its implementation"

- J. Samit -

Although the mathematical aspects of this article are important, a further work has been done to "make it happen". An implementation of the modified Collapsed Gibbs Sampler for IMM has been developed using Matlab®, the details of this implementation will be the main topic of this chapter. This algorithm is based on the implementation made by Yee Whye Teh [11].

This section will cover up implementation details for the algorithm 3 proposed above. The first part of this chapter includes an introduction on OOP, in order to introduce some necessary concepts. Also UML, a standard for programming design, will be introduced. The second part comprises the explanation of the program architecture and behaviour described using UML.

Also, a detailed description of the interface requirements is explained. This allow users to change the data probability function without having to change the algorithm itself. Under this assumption, a user can concentrate only on designing the required functions and does not have to care on how the algorithm is programmed.

3.1 General concepts

3.1.1 OOP

Object oriented programming (OOP) is a programming paradigm that encapsulates software behavior into objects. Objects are entities that represent either real things or logical aspects of the software being developed. All objects are composed by:

- **Attributes:** Properties of the object
- **Methods:** Operations that the object can perform.

For example, a `Vehicle` could be an object in a system. Some of its attributes could be the number of wheels or the current speed. Some methods could be to accelerate, to break or to turn, etc. Objects are also called classes.

Methods in an object are called by other objects, therefore **objects talk among them**. The calling class knows nothing about how the method is internally implemented. This property is called **encapsulation**, and is one of the most interesting properties of OOP because it allows different developers to work easily on the same project without having to know the implementation details for every class in the system.

OOP also allows objects to be "children" for other objects. For example, a car or a motorbike could be "children objects" or subclasses of the `Vehicle` object shown above, which is called **superclass**. They inherit the properties and methods of their "parent". This is known as **object inheritance**. It is a representation of an "is a" relation. Therefore, a car "is a" vehicle and also a motorbike "is a" vehicle, so class `Car` and class `Motorbike` are subclasses of `Vehicle`.

Also, most objects could be **instantiated**. An instance is a particularization of an object to a given set of properties. Each object instantiation or instance, has its own values for the properties described on its

corresponding class, that may change or not. For example, your own car could be an instance of the `Car` class described above and your colleague's car could be another instance.

Some other objects cannot be directly instantiated, because either an instance for the object does not make logical sense or instances are made through particular subclasses. For example, in a given system, an instance of an abstract vehicle is not useful. But, it possible to instantiate **Car X** from class `Car`, which in fact "is a" vehicle. This means that it is also an instance, through inheritance of class `Vehicle`. These type of objects are also referred as abstract objects and are very useful when defining interfaces.

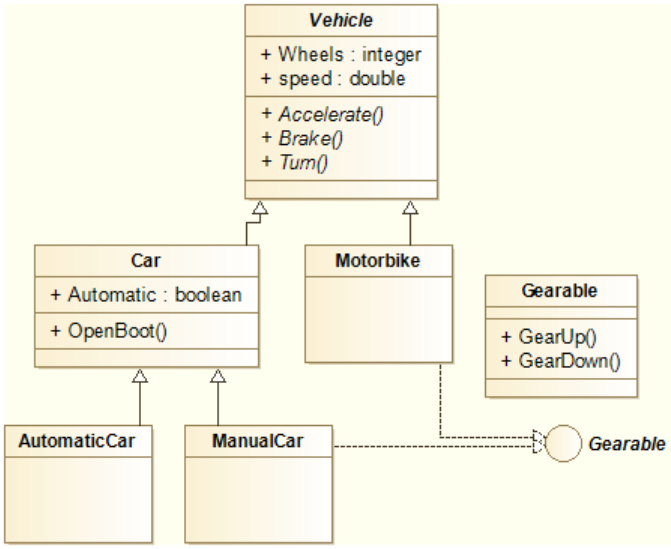


Figure 19 UML example of classes and interfaces

An **interface** is a set of methods which an implementing class MUST provide to other classes. Consequently, an interface guarantees that the implementing objects will respond as defined to the set of methods described in the interface. An interface does not impose any restriction on how these methods are finally implemented but rather on how they are called and what they do return.

Interfaces allow to create modular code, that can call methods from different classes, provided that they implement the corresponding interfaces. Interfaces are sometimes referred as *-able* classes, thus `Driveable` or `Turnable` could be interfaces implemented by the `Vehicle` abstract class, and thereby by its subclasses.

In OOP the concept of interfaces and abstract classes are confusingly similar, but the latest is preferred for methods inherently belonging to an entity in opposition to added funtionality. An example of these abstract classes is class `Vehicle` which has methods such as: `turn`, `accelerate` and `brake` inheretely belonging to a vehicle. Interfaces are often used when the functionality provided is an addition to the behavior of a class. For example, implementing interface `Gearable` makes a class provide methods such as `GearUp` and `GearDown`. However, these operations are not exclusively referred to a particular class but they are an additional functionality to a `ManualCar` or a `Motorbike`. An example of the shared behavior of abstract classes and interfaces is the fact that **in Matlab®, interfaces are always defined by abstract classes**.

3.1.2 UML

The Unified Modelling Language (UML) is a common language for software modelling, it is language-agnostic and allows many developers to implement better software which is simpler, more reliable and easier to maintain. As a conclusion, when programming software, the design process should be taken carefully into consideration and UML diagrams help in this process as they describe software architecture and behavior.

3.2 Algorithm description

3.2.1 Class Diagrams

The class diagrams are used for describing software architecture. When developing using OOP, class diagrams provide a general view of the system and the relationships among the objects that compose it.

In a class diagram, inheritance is represented by a solid arrow with an empty pointer which goes from the subclass to the superclass. Interface realizations are represented by dashed arrows with an empty pointer, which goes from the implementing class to the interface definition.

Methods are represented by their signature, which in turn is a composition of: the calling parameters, the returned value type and its visibility. Also, attributes are represented by their name, their type and their visibility. The visibility of a method, class or attribute declares which classes can access them. In Matlab® 3 types of visibility are declared [11]:

- **Public (+):** Unrestricted access
- **Protected (#):** Access from class or subclasses
- **Private (-):** Access by class members only (no subclasses)

As Matlab® provides support for OOP, and due to the advantages of using OOP (reusability, inheritance, encapsulation, …) in software development, the implementation of the algorithm 3, explained in chapter 2, has been made using this programming paradigm. Although the system is composed by many classes, just the most relevant for the correct implementation of the algorithm will be described.

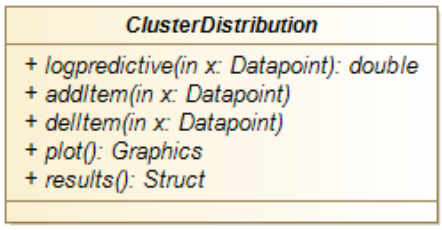


Figure 20 Cluster Distribution Abstract Class

3.3 Implementation of the Modified Collapsed Gibbs Sampler Algorithm

Algorithm 3 is the final result for the modified Collapsed Gibbs Sampler algorithm, thoroughly described in Chapter 2.

As we mentioned before, the only model-dependent aspects of the algorithm are those concerning the internal calculations for the predictive distributions, (2.49). So, we will define the `ClusterDistribution` abstract class in order to support modularization.

As long as the classes used for modelling a particular probability distribution inherit from `ClusterDistribution` class, the algorithm will work. This allows to separate the algorithm itself from the data model.

The general model is as follows

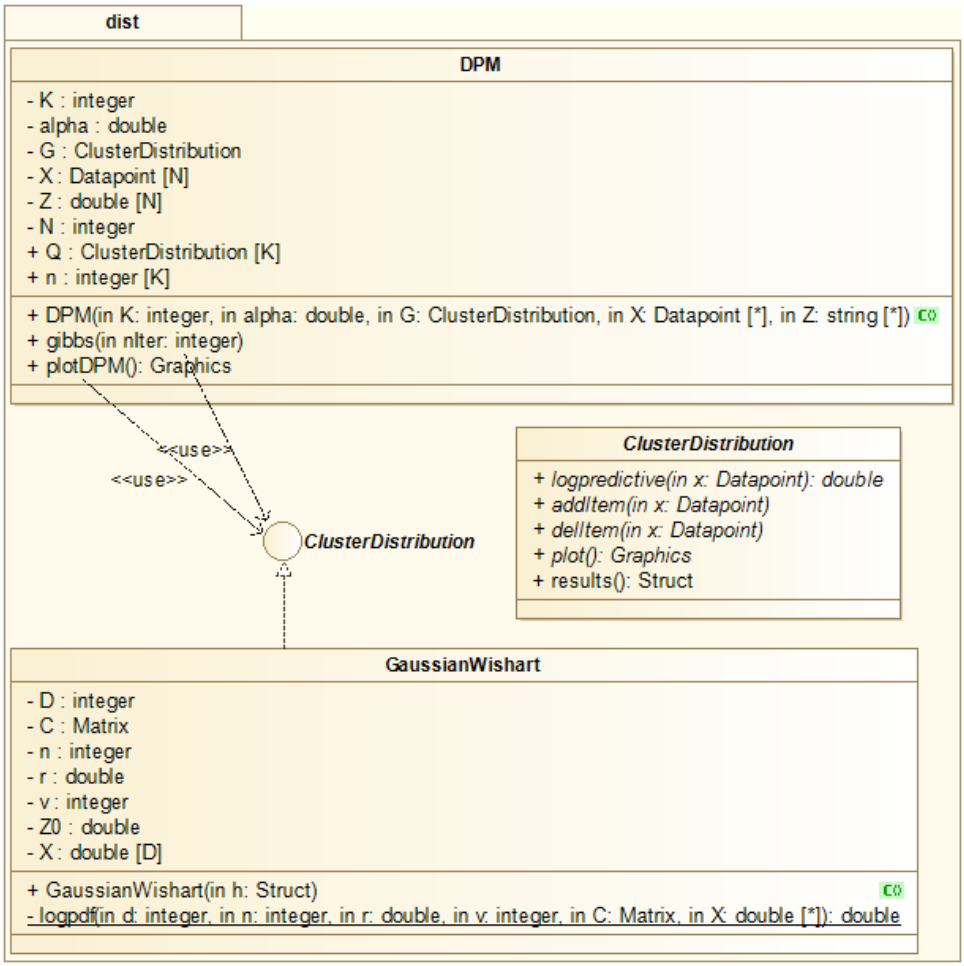


Figure 21 General Class Diagram

Package `dist` contains the `DPM` class and the `GaussianWishart` class which is a subclass of `ClusterDistribution`, and thereby it can be used from class `DPM`, which implements the algorithm itself. Methods `gibbs(nIter: integer)` and `plotDPM()` make use of the `ClusterDistribution` declared methods without taking into consideration implementation specific details for each probability function. In fact, other classes, implementing different probability models, can be created and used, as long as they implement abstract class `ClusterDistribution`. Moreover, although uncommon, different data models can be used for different clusters in the same program.

3.3.1 Methods description

Next, a detailed explanation of all methods is described. Note that all methods are publicly visible.

3.3.1.1 `+logpredictive(x: Datapoint): double`

This method computes the logarithm of equation (2.46) for a given cluster. As said previously, the logarithm calculation allows to avoid numerical errors.

3.3.1.2 `+addItem(x: Datapoint): void`

This method adds a datapoint to the cluster, which is added to the $\boldsymbol{x}_{-i}$ data vector. Therefore, it will be used for future calculations. Diverse distributions can operate in a different way when a new datapoint is added.

3.3.1.3 `+delItem(x: Datapoint): void`

This method removes a datapoint from a cluster. The datapoint is therefore removed from vector $\boldsymbol{x}_{-i}$ and will no longer be used for future calculations.

3.3.1.4 `+plot(): Graphics`

This method plots the cluster. It is a completely implementation-dependent method; as diverse distributions can be plotted differently.

3.3.1.5 `+results(): Struct`

This method returns a set with the selected results for each run. There are not any constrains in the returned values. Further processing is beyond the scope of the algorithm itself because it is conditioned by the implementation specific details.

3.3.2 Activity Diagram

Activity diagrams are behaviour diagrams that describe the flow of the system from a certain point. Activity diagrams are composed by both: activity nodes, in which some operations are made, and decision nodes, in which, upon a given condition, the flow is directed to a different path.

In the first activity diagram, right, the upper level of the algorithm is described.

First, a set of hyperparameters are initialized and computed. Also, the maximum number of iterations for the Gibbs sampler and some random seeds are set. Defining the random seeds allow to make every run of the algorithm determinant.

The next step is to create some synthetic data to run the algorithm, this is made by using the hyperparameters defined previously.

Then, for each defined seed the algorithm is run. The DPMM is initialized and until the maximum number of iterations is reached, a Gibbs iteration is run and the results of it are stored.

Finally, the results are plot. Recall that, in here we do not define the burnin step which has to be made in order to avoid using the first non-precise samples.

We have not defined yet the algorithm itself, which is run inside the `gibbs()` function. The next step is to describe the inner activity diagram.

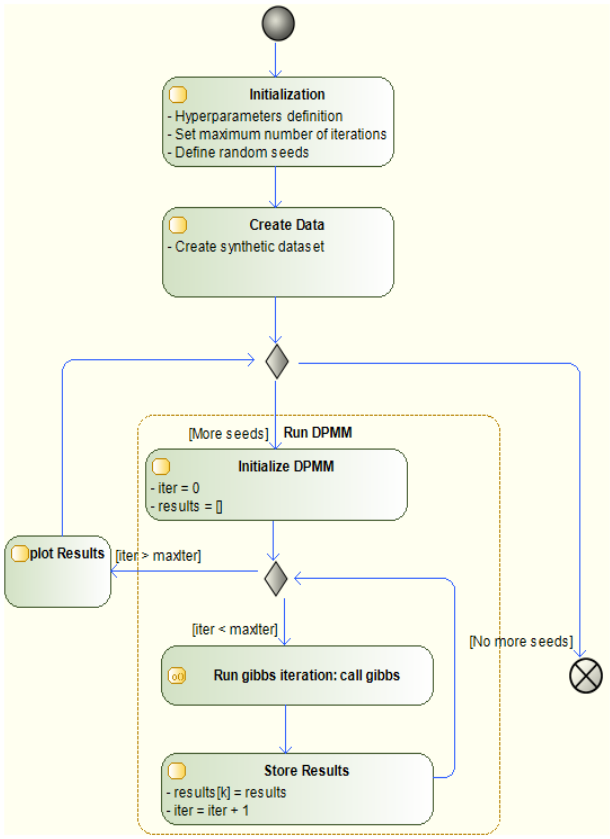


Figure 22 General Activity Diagram

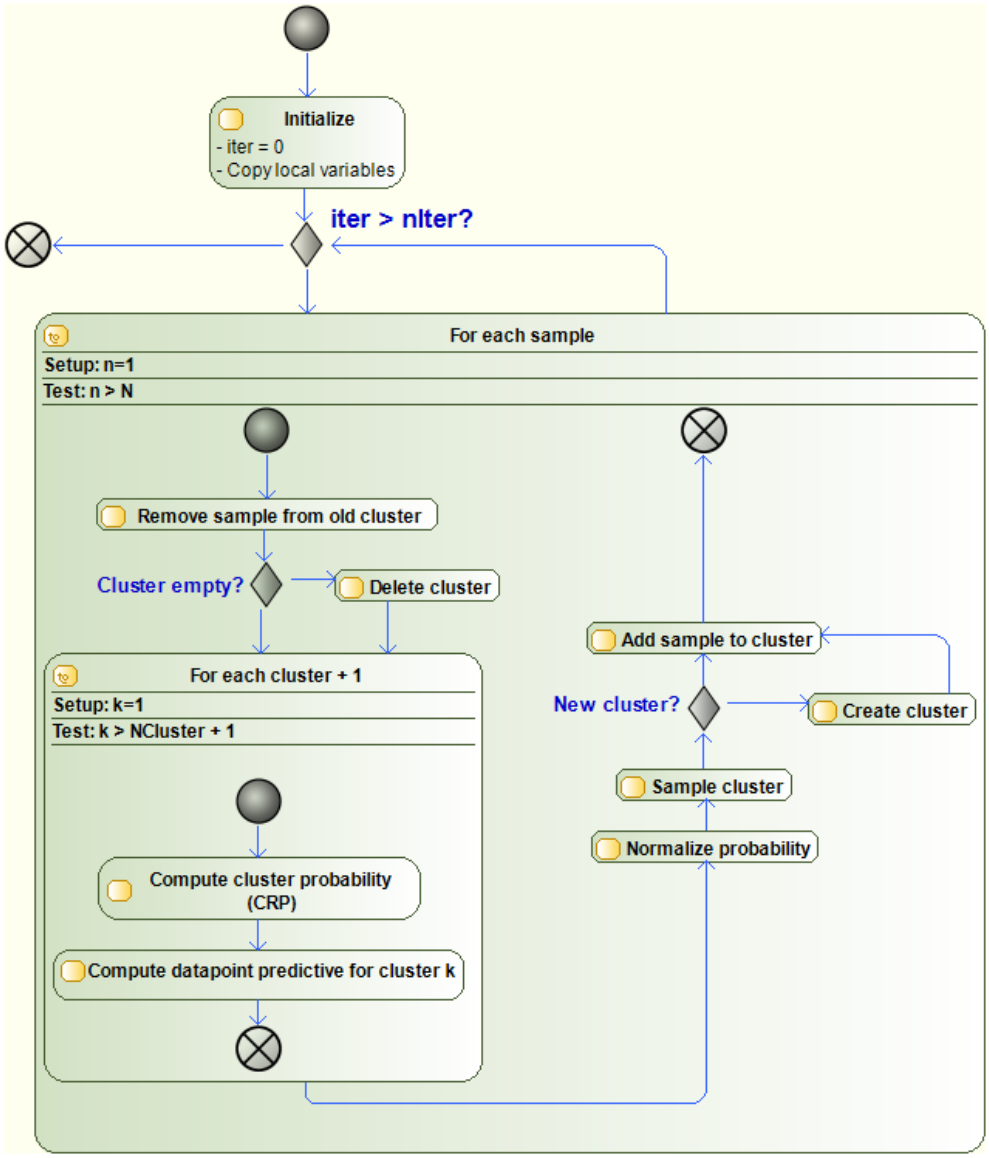


Figure 23 Activity Diagram for DPMM

First step is to initialize some variables, such as the number of current components (K), the number of datapoints (N), the current iteration, a vector with the current number of datapoints in each cluster ($\boldsymbol{n}$), etc.

Then, until the maximum number of iterations is reached, for each sample we do the following.

First, we remove the sample from the current cluster, at the beginning there is only one cluster in which all the samples are. Thus, we remove the sample using the `deItem(x: Datapoint)` function and decrement the number of samples in the component.

If the number of components is 0, that is, the cluster is empty, we remove it from the list. Then, for each remaining cluster, we compute (2.46) making use of the `logpredictive(x: Datapoint)` function. Finally, we normalize the probabilities obtained and sample the new cluster assignment from it. If the sampled cluster is new, we create it and finally we add the sample to the selected cluster using the `addItem(x: Datapoint)` function.

Note that, we have not made any assumption on how the data is distributed, as long as the called classes inherits from `ClusterDistribution` and implements the required methods the algorithm will work. This allows to split the algorithm itself from the data distribution details and use different distribution classes without changing the algorithm.

4 EXAMPLES

"Setting an example is not the main means of influencing others, it is the only means"

- A. Einstein -

Some examples have been made in order to prove how does the algorithm behave with different data. This data was generated following a mixture of Gaussian PDFs as explained above. Thus, equations particularized from Appendix C were used. On each example, the hyperparameters used are described.

4.1 Example 1: First Run

In this example we have set up $K = 5$ clusters easily separable at a glance. Thus, we have configured the following parameters:

- Cluster separability $\left(\sigma_H^2/\sigma_d^2\right)$: 50
- Data variance $\left(\sigma_d^2\right)$: 1
- GaussianWishart degrees of freedom: 5
- maxIter: 400
- Concentration parameter (α): 1
- Number of samples (N): 100

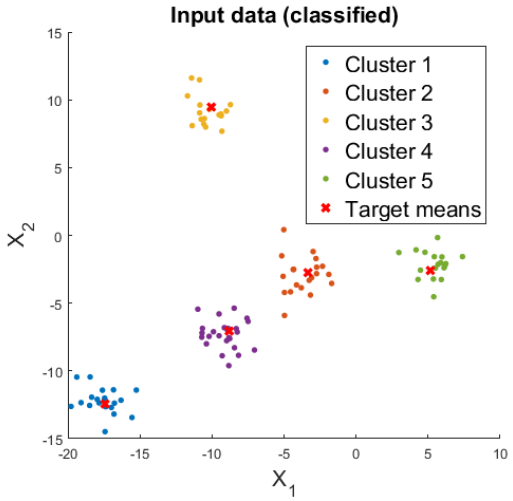


Figure 24 Example 1: Input data

We run the algorithm for a set of 2-D samples generated randomly from a mixture of Gaussians with random means. Parameters are sampled from a Gaussian Prior with $\boldsymbol{\mu}_\mu = \mathbf{0}$ and $\Sigma_{\boldsymbol{\mu}} = \sigma_H^2 \cdot \mathbb{I}_D$, the D-dimensional identity matrix. Mixture coefficients are uniform $\boldsymbol{\pi} = [0.2\ 0.2\ 0.2\ 0.2\ 0.2]^T$. For the target run we obtain the data at the right.

The target means we are looking for, in this example, are

$$\boldsymbol{\theta} = \begin{bmatrix} -17.4239 & -3.3386 & -10.0413 & -8.8018 & 5.1565 \\ -12.4305 & -2.7492 & 9.464 & -7.0588 & -2.5941 \end{bmatrix}^T$$

Then, we run the algorithm for 450 iterations, and discard the first 50 as burnin period. The following table indicates the times the algorithm has proposed each number of clusters (K^*) and its relative frequency.

K^*	f_{abs}	f_{rel}
1	0	0
2	0	0
3	0	0
4	0	0
5	164	0.4090
6	182	0.4539
7	42	0.1047
8	12	0.0299
9	1	0.0025
Total	**401**	**1**

Table 2 Histogram data for K

This results are coherent with the data generated, although the algorithm tends to propose more clusters than the existing amount. This characteristic helps the algorithm escape from poor local optima [3]. In fact, the algorithm estimates the probability of each cluster cardinality.

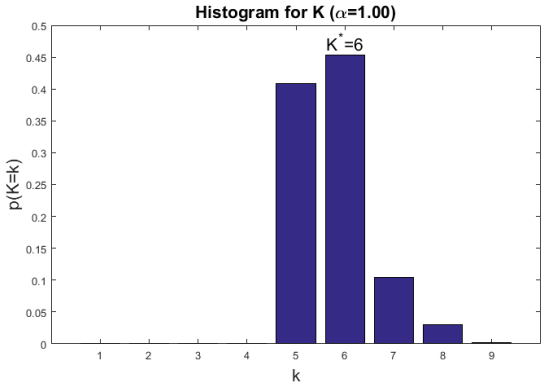
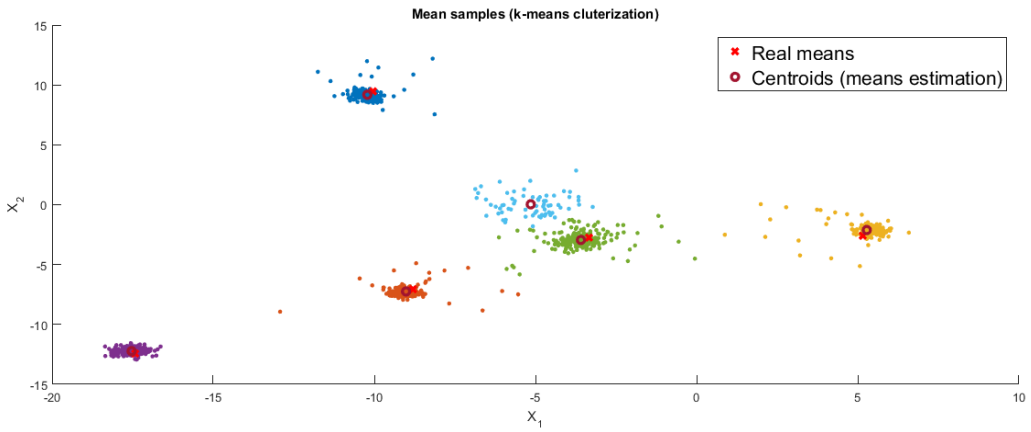


Figure 25 Histogram for K

Once we have chosen the number of clusters we have we can run parametric clustering algorithm to infer the data means. As a matter of fact, k-means algorithm has been run for the mean samples generated before.

Note that, although the k-means algorithm provides an approximate value for the mean, the number of clusters selected might differ from their real amount, making k-means estimation dependant on the selection of the real cluster measure.



It is far more interesting to infer the PDF for the hyperparameters, conditioned to the data. In this example we are more interesting in the mean. Once we had chosen a value for K^* we can compute $p(\boldsymbol{\theta}_K|\boldsymbol{X}, K = 6)$. To do this we will use a Normal kernel smoothing function to approximate the PDF from the samples that we have obtained from Collapsed Gibbs Sampler.

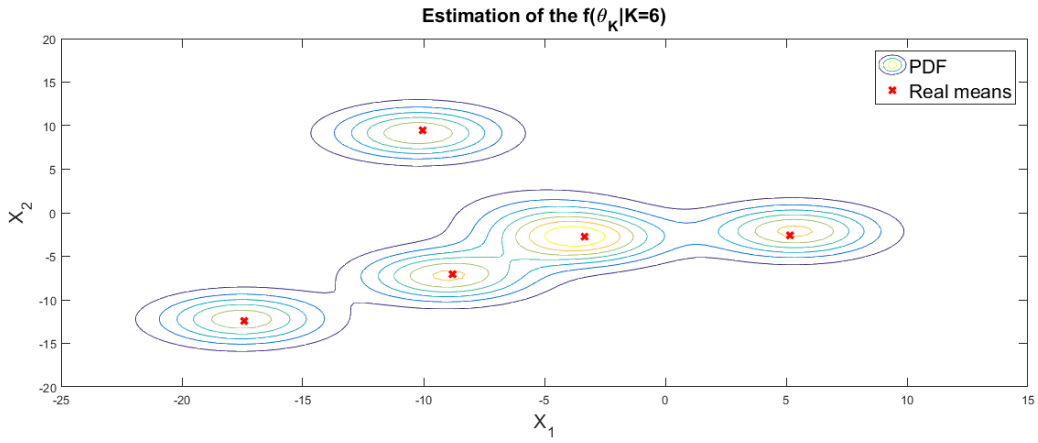


Figure 26 Example 1: Estimation of $f(\boldsymbol{\theta}_K|\boldsymbol{K}^* = \mathbf{6})$

We can also try to infer the marginal PDF as

$$f(\boldsymbol{\theta}_K, \boldsymbol{X}) = \sum_{k=1}^{\max(K^*)} p(K^* = k) f(\boldsymbol{\theta}_K|\boldsymbol{X}, K^* = k) \tag{4.1}$$

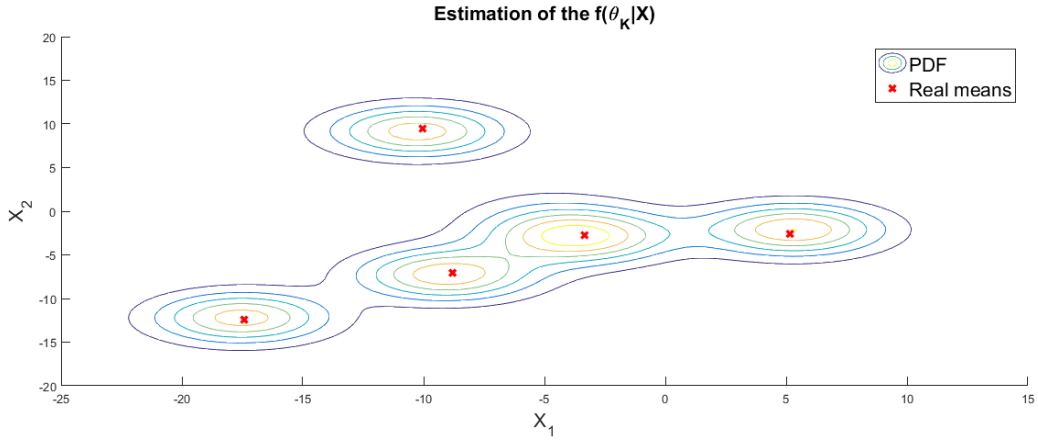


Figure 27 Example 1: Estimation of $f(\boldsymbol{\theta}_K)$

4.2 Example 2: Changes in α

The following example will address the different results we may expect when changing the value of the concentration parameter α. To do so, we will compare Example 1, run for $\alpha = 1$, with a new run where we will only change the value of α to 0.25 and 5 respectively.

We can see that the main change comes from the PMF of K^*. With a low value of α, the PMF becomes sharper and estimations tend to concentrate on 1 value. When a high value for α has been used, the PMF becomes smoother and estimations on the clusters quantity tend to change within a range.

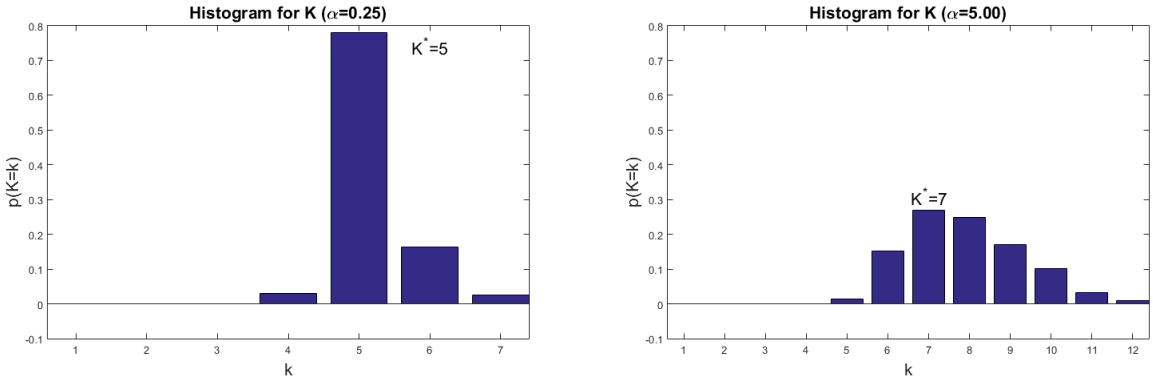


Figure 28 PMF changes with α. Left, $\alpha = 0.25$. Right, $\alpha = 5$

As a conclusion, we can use low values of α when our certainty about the real number of clusters is high, and we can use high values of α in other case. Note that, as the algorithm tends to propose more clusters that its real quantity, when using a high value of α the assessment is often overestimated.

Also, when using a low value of α, as the estimation of K is more precise, the k-means algorithm for finding the means, conditioned to the maximum value of K provides a better approximation to the real means. Furthermore, when using a low value of α, the samples from the mean tend to be more concentrated.

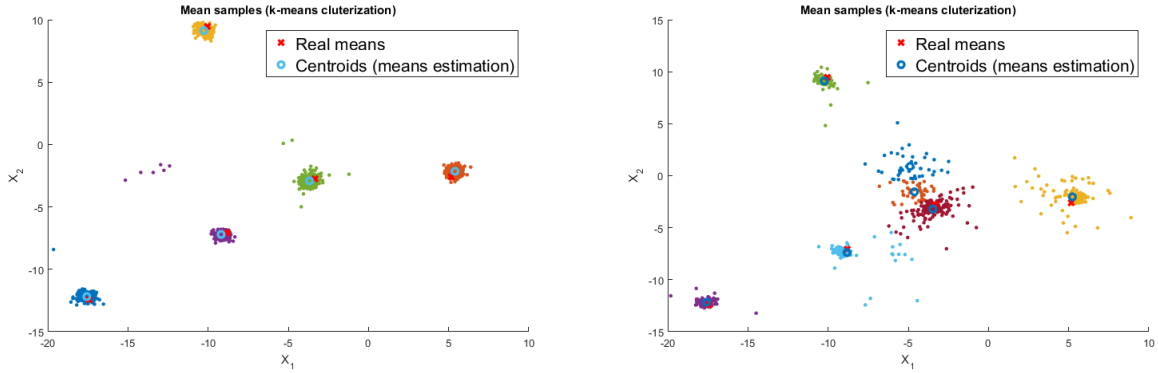


Figure 29 Samples from the mean. Left, $\alpha = 0.25$. Right, $\alpha = 5$

Changes on the marginal PDF and conditional PDF with respect to K^* are also notable but not as much as the changes in the PMF of K^*.

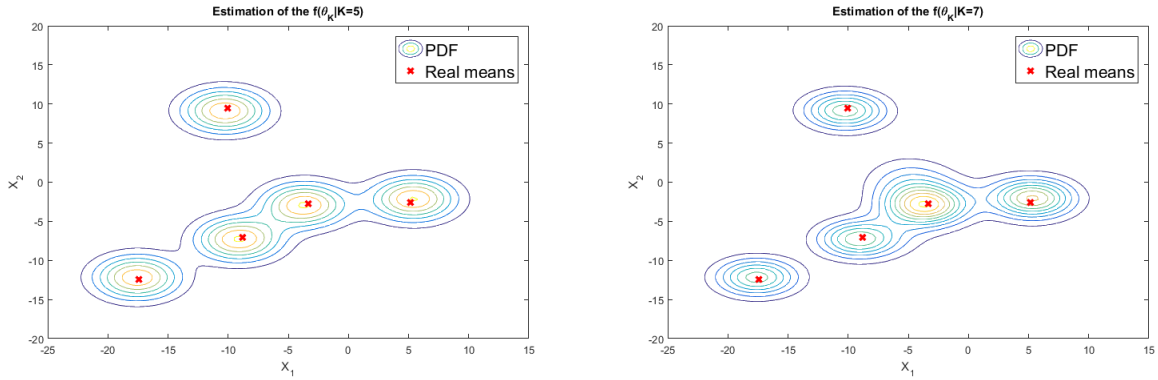


Figure 30 Estimations of the conditional PDF. Left, $\alpha = 0.25$. Right, $\alpha = 5$

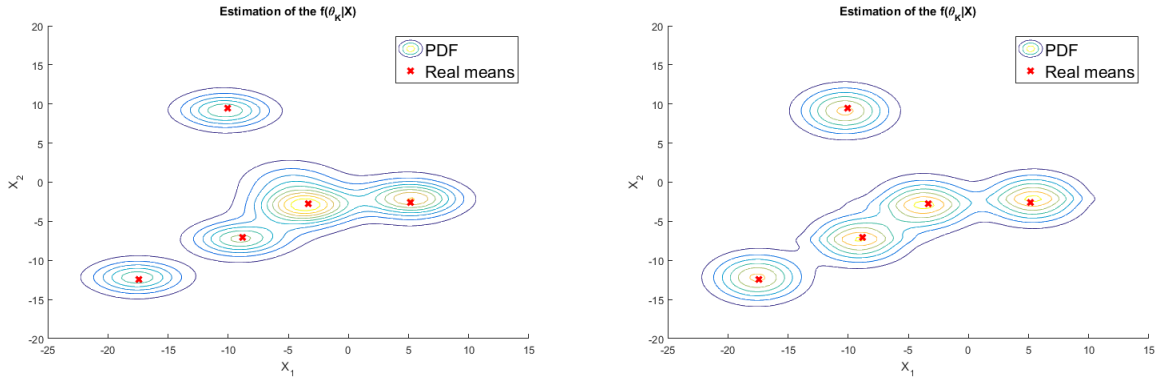


Figure 31 Estimations of the marginal PDF. Left, $\alpha = 0.25$. Right, $\alpha = 5$

4.3 Example 3: Changes in the number of samples

Good algorithms also have to take into consideration the complexity. Algorithmic complexity is defined as the amount of resources needed to run the algorithm. In our case we are mostly interested in the time to run a set of iterations for the Modified Collapsed Gibbs Sampler algorithm introduced in Chapter 2. A simple way of determining algorithm complexity includes order estimation. This method provides an approximate estimation of the time required to run a given program with respect to the number of samples.

Picture at the right shows different complexity types, note that there are 3 great groups of algorithms. The best ones are those who are either constant or logarithmic but not all problems allow these type of solutions.

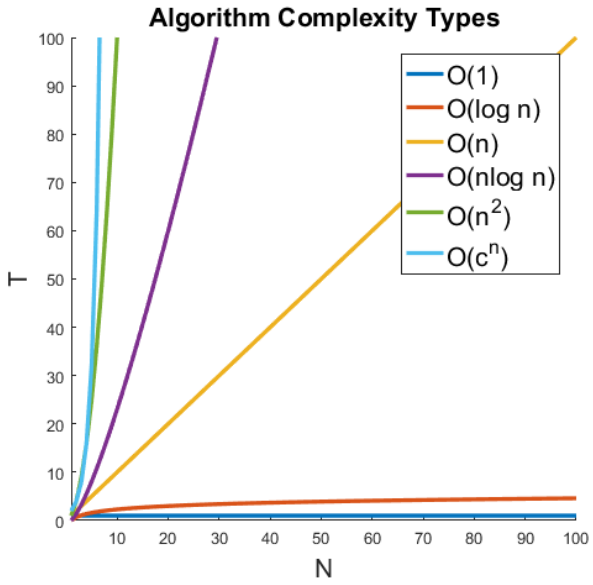


Figure 32 Complexities Types (source: Wikipedia)

Worst cases include loglinear, quadratic and exponential complexities, these algorithms require extra effort to make input size small enough to run in reasonable time. Exponential algorithms are wanted when working with cryptography problems. Linear algorithms are not good nor bad, but if the input size grows too much execution time can become excessive.

Typical algorithm complexities include

Complexity	Notation	Example
Constant	$\mathcal{O}(1)$	Determine whether a is divisible by b.
Logarithmic	$\mathcal{O}(\log n)$	Binary search
Linear	$\mathcal{O}(n)$	Unsorted list search
LogLinear	$\mathcal{O}(n \log n)$	FFT
Quadratic	$\mathcal{O}(n^2)$	Direct Convolution
Exponential	$\mathcal{O}(c^n), c > 1$	Password brute force guessing

Table 3 Complexities types (source: Wikipedia)

In order to analyse complexity, the algorithm exposed in Chapter 2 was run for different input sizes. Also, to avoid error in the measure the process was run for 5 times and the results averaged.

N	Run 1 (s)	Run 2 (s)	Run 3 (s)	Run 4 (s)	Run 5 (s)	Mean (s)
100	19.1049	23.9021	20.8796	25.5997	20.4665	**21.9906**
200	35.8320	40.3250	41.7959	51.5123	37.8293	**41.4589**
300	54.1714	69.6148	56.4610	67.0042	59.2365	**61.2976**
400	71.8522	78.3678	83.7282	90.6732	78.0339	**80.5311**
500	95.2873	103.2718	101.6166	99.3961	101.1522	**100.1448**
600	102.2152	119.3735	120.4352	130.7740	113.2382	**117.2072**
700	141.0298	138.1776	154.8046	159.3398	143.6061	**147.3916**
800	150.8409	150.9645	158.9645	181.5060	144.5494	**157.3650**
900	177.9492	169.5415	162.7490	174.7029	151.6819	**167.3249**
1000	191.0107	194.2392	213.8180	193.1364	214.4593	**201.3327**

Table 4 Execution Time results

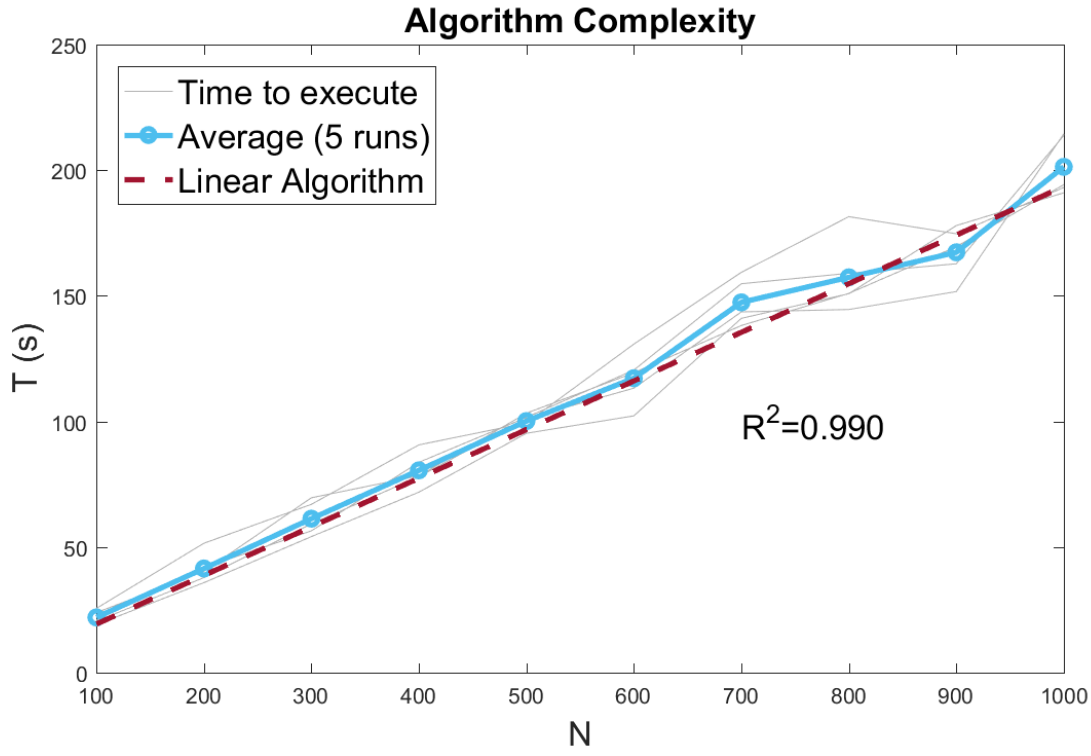


Figure 33 Modified Collapsed Gibbs Sampler Algorithm complexity

Results show that the algorithm runs in linear time.

5 CONCLUSIONS

"All in all, it is just another brick in the wall"

- Pink Floyd-

In conclusion, once the main aspects concerning ML clustering algorithms have been detailed, it is clearly visible that, sometimes, when the number of componets involved is unknown, the traditional models fall back and new ways for solving clustering problems must be proposed.

ML algorithms applied to clustering problems, when no training samples exist, always lack some of the flexibility that classification algorithms have. This is due to the lack of *a priori* knowledge about how many components are involved in the mixture.

Then, a new way for describing mixture models using the Dirichlet Distribution was introduced. But there still was the same issue than before, the number of classes had to be known or estimated beforehand. Instead of that, we squeezed the previous model a bit and introduced the Dirichlet Process, an infinite dimensional extrapolation of the Dirichlet Distribution, which allowed us to propose clusters dynamically, finding the amount that better fits our data.

Whilst working with infinite-dimensional density functions might prove to be difficult, the predictive properties described later allowed us to easily use the Dirichlet Distribution for our befenit. The Chinese Restaurant Process allowed us to sample the Dirichlet Process. But a last tool was necessary, the Collapsed Gibbs Sampler allowed to fit our data.

Finally, not only the mathmathics behind the scenes are described but also a flexible implementation is provided. This allows you, the readed, to find the equations describing your system. After that, the implemented software would allow you to use this algorithm for your own benefits, provided that you comply with the required interface.

This work is not intended to be self-contained but rather to be a good starting point for people facing similar problems as me, the uncertainty in the number of components involved.

Lack of time prevented us for using this algorithm for "real life problems", mathematics concerning this issues is always harsh and time required to completely understand them is much. You are highly encouraged to use this algorithm for real situations. Without having to know exactly everything about the mathematics behind it and while concentrating in your own problem, rely on this work and use it, while understanding what you are doing.

References

[1] D. W. Hosmer, S. Lemeshow and R. X. Sturdivant, Applied Logistic Regression, Hoboken, NJ: John Wiley & Sons, Inc, 2005.

[2] A. P. Dempster, N. M. Laird and D. B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," vol. 39, no. 1, 1977.

[3] S. Roweis and Z. Ghahramani, "Department of Statistics: Columbia University," Octubre 1998. [Online]. Available: http://www.stat.columbia.edu/~liam/teaching/neurostat-fall15/papers/hmm/roweis-ghahramani-lds.pdf.

[4] K. P. Murphy, Machine Learning - A Probabilistic Approach, Londres: The MIT Press, 2012.

[5] B. A. Frigyik, A. Kapila y M. R. Gupta, «Introduction to the Dirichlet Distribution and Related,» Diciembre 2010. [En línea]. Available: https://www.ee.washington.edu/techsite/papers/documents/UWEETR-2010-0006.pdf. [Último acceso: 23 Julio 2016].

[6] T. Yee Whye, *Dirichlet Process,* Londres: University College London.

[7] T. S. Ferguson, "A Bayesian Analysis of Some Nonparametric Problems," vol. 1, no. 2, 1973.

[8] D. J. Aldous, I. A. Ibragimov and J. Jacod, École d'Été de Probabilités de Saint-Flour XIII, Berlin: P. L. Hennequin, 1983.

[9] C. J. Geyer, *Introduction to Markov Chain Monte Carlo.*

[10] T. Yee Whye, "Exponential Families: Gaussian, Gaussian-Gamma, Gaussian-Wishart, Multinomial," University College London, 13 Agosto 2007. [Online]. Available: http://www.stats.ox.ac.uk/~teh/research/notes/GaussianInverseWishart.pdf. [Accessed 20 Agosto 2016].

[11] T. Yee Whye, "DPmixturesTeh: Dirichlet process mixture models," [Online]. Available: https://github.com/probml/pmtksupport/tree/master/dpmixturesTeh07.

[12] The MathWorks, Inc, "Matlab Documentation," [Online]. Available: http://es.mathworks.com/help/matlab/matlab_oop/property-attributes.html.

[13] K. P. Murphy, "Gaussians," 24 Noviembre 2006. [Online]. Available: http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/murphy-gaussians.pdf.

Glossary

Acrónimos

CRP: Chinese Restaurant Process 13
DRV: Discrete Random Variable 10
EM: Expectation-Maximization 4
FMM: Finite Mixture Model 7
GMM: Gaussian Mixture Model 3
IMM: Infinite Mixture Models 9
MCMC: Markov Chain Monte Carlo 21
ML: Machine Learning 1
OOP: Object Oriented Programming 22
PDF: Probability Density Function 3
RV: Random Variable 3
SBC: Stick-Breaking Construction 13

Appendix A - Normal Log-Likehood

A.1 Normal Log-Likehood

Derivation of the normal, log-likehood function where $\boldsymbol{X}^{(k)}$ are the samples in cluster k and $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ are the mean and covariance matrix for this cluster.

$$p\left(\boldsymbol{X}^{(\boldsymbol{k})}\middle|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)=\prod_{i=1}^{N_k}\mathcal{N}(\boldsymbol{x}_i^{(k)}|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)$$

Taking $ln(\cdot)$ in both sides

$$\ln\left(p\left(\boldsymbol{X}^{(\boldsymbol{k})}\middle|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)\right)=\ln\left(\prod_{i=1}^{N_k}\mathcal{N}\left(\boldsymbol{x}_i^{(k)}\middle|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)\right)$$

Given that the logarithm of the product is the sum of the product of the logarithms

$$\ln\left(p\left(\boldsymbol{X}^{(\boldsymbol{k})}\middle|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)\right)=\sum_{i=1}^{N_k}\ln\left(\mathcal{N}\left(\boldsymbol{x}_i^{(k)}\middle|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)\right)$$

Substituting $\mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu},\boldsymbol{\Sigma})$ by its expression, where K denotes the number of dimensions

$$\ln\left(p\left(\boldsymbol{X}^{(\boldsymbol{k})}\middle|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)\right)=\sum_{i=1}^{N_k}\ln\left(\frac{1}{\sqrt{(2\pi)^K|\boldsymbol{\Sigma}_k|}}e^{-\frac{1}{2}\left(\boldsymbol{x}_i^{(k)}-\boldsymbol{\mu}_k\right)^T\boldsymbol{\Sigma}_k^{-1}\left(\boldsymbol{x}_i^{(k)}-\boldsymbol{\mu}_k\right)}\right)$$

Applying logarithm to the inner expression we obtain

$$\ln\left(p\left(\boldsymbol{X}^{(\boldsymbol{k})}\middle|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)\right)=\sum_{i=1}^{N_k}-\frac{1}{2}\ln\left|\boldsymbol{\Sigma}_k^{-1}\right|-\frac{K}{2}\ln 2\pi-\frac{1}{2}\left(\boldsymbol{x}_i^{(k)}-\boldsymbol{\mu}_k\right)^T\boldsymbol{\Sigma}_k^{-1}\left(\boldsymbol{x}_i^{(k)}-\boldsymbol{\mu}_k\right)$$

Taking into account that the first 2 terms don't depend on the index I, we can extract then from the inner sum.

$$\ln\left(p\left(\boldsymbol{X}^{(\boldsymbol{k})}\middle|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)\right)=-\frac{N_k}{2}\ln\left|\boldsymbol{\Sigma}_k^{-1}\right|-\frac{N_kK}{2}\ln 2\pi-\frac{1}{2}\sum_{i=1}^{N_k}\left(\boldsymbol{x}_i^{(k)}-\boldsymbol{\mu}_k\right)^T\boldsymbol{\Sigma}_k^{-1}\left(\boldsymbol{x}_i^{(k)}-\boldsymbol{\mu}_k\right)$$

From here, we can compute the derivatives with respect the parameters

A.2 Mean derivative

From the log-likehood calculation, we can take the first derivative with respect to the mean

$$\frac{\partial}{\partial\boldsymbol{\mu}_k}\ln\left(p\left(\boldsymbol{X}^{(\boldsymbol{k})}\middle|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)\right)$$

Taking into account that, the first 2 terms do not depend on the mean

$$\frac{\partial}{\partial \boldsymbol{\mu}_k} \ln\left(p\left(\boldsymbol{X}^{(\boldsymbol{k})} \middle| \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)\right) = -\frac{\partial}{\partial \boldsymbol{\mu}_k} \frac{1}{2} \sum_{i=1}^{N_k} \left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)^T \boldsymbol{\Sigma}_k^{-1} \left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)$$

Then, deriving

$$\frac{\partial}{\partial \boldsymbol{\mu}_k} \ln\left(p\left(\boldsymbol{X}^{(\boldsymbol{k})} \middle| \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)\right) = -\sum_{i=1}^{N_k} \boldsymbol{\Sigma}_k^{-1} \left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)$$

If we then, make the derivative equals to 0, to find the minimum, it follows

$$\frac{\partial}{\partial \boldsymbol{\mu}_k} \ln\left(p\left(\boldsymbol{X}^{(\boldsymbol{k})} \middle| \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)\right) = -\sum_{i=1}^{N_k} \boldsymbol{\Sigma}_k^{-1} \left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right) = 0$$

$$\frac{\partial}{\partial \boldsymbol{\mu}_k} \ln\left(p\left(\boldsymbol{X}^{(\boldsymbol{k})} \middle| \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)\right) = \sum_{i=1}^{N_k} \left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right) = 0$$

$$\widehat{\boldsymbol{\mu}_k} = \frac{1}{N_k} \sum_{i=1}^{N_k} \boldsymbol{x}_i^{(k)}$$

Where N_k is the amount of samples in cluster k. This is, the mean for class k is computed only with the datapoints assigned to the class k, as the arithmetic mean of the subset.

A.3 Covariance matrix derivative

Now we are interested in the calculation of the covariance matrix derivative

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_k} \ln\left(p\left(\boldsymbol{X}^{(\boldsymbol{k})} \middle| \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)\right)$$

Taking the closed form of $\ln\left(p\left(\boldsymbol{X}^{(\boldsymbol{k})} \middle| \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)\right)$, and as that the second term does not depend on $\boldsymbol{\Sigma}_k$. We reach the following expression

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_k} \ln\left(p\left(\boldsymbol{X}^{(\boldsymbol{k})} \middle| \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)\right) = \frac{\partial}{\partial \boldsymbol{\Sigma}_k} \left(-\frac{N_k}{2} \ln\left|\boldsymbol{\Sigma}_k^{-1}\right| - \frac{1}{2} \sum_{i=1}^{N_k} \left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)^T \boldsymbol{\Sigma}_k^{-1} \left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right) \right)$$

We can define the **matrix trace** as

$$tr[\boldsymbol{A}] = \sum_i A_{ii}$$

This operation satisfies the cyclic permutation property [12]

$$tr[\boldsymbol{ABC}] = tr[\boldsymbol{BCA}] = tr[\boldsymbol{CAB}]$$

We can make use of the **trace trick** which states the following[4]

$$\boldsymbol{x^T A x} = tr[\boldsymbol{x^T A x}] = tr[\boldsymbol{x^T x A}]$$

Therefore, we can reorder $\ln\left(p\left(\boldsymbol{X}^{(\boldsymbol{k})} \middle| \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)\right)$

$$\ln\left(p\left(\boldsymbol{X}^{(\boldsymbol{k})} \middle| \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)\right) = -\frac{N_k}{2} \ln\left|\boldsymbol{\Sigma}_k^{-1}\right| - \frac{1}{2} \sum_{i=1}^{N_k} tr\left[\left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)^T \boldsymbol{\Sigma}_k^{-1} \left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)\right]$$

[4] $\boldsymbol{x^T A x} = tr[\boldsymbol{x^T A x}]$ as the product is in $\mathbb{R}_{1x1}$ and therefore it is equal to its trace.

$$\ln\left(p\left(\boldsymbol{X}^{(k)}|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k\right)\right) = -\frac{N_k}{2}\ln|\boldsymbol{\Sigma}_k^{-1}| - \frac{1}{2}\sum_{i=1}^{N_k} tr\left[\left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)\left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)^T \boldsymbol{\Sigma}_k^{-1}\right]$$

Taking into account that

$$\frac{\partial}{\partial \boldsymbol{X}}(tr[\boldsymbol{A}\boldsymbol{X}]) = \boldsymbol{A}^T$$

$$\frac{\partial}{\partial \boldsymbol{X}}\ln|\boldsymbol{X}| = (\boldsymbol{X}^T)^{-1} = (\boldsymbol{X}^{-1})^T$$

We can compute easily the result applying the chain rule, taking $\boldsymbol{R}_k = \boldsymbol{\Sigma}_k^{-1}$.

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_k} f(\boldsymbol{R}_k) = \frac{\partial \boldsymbol{R}_k}{\partial \boldsymbol{\Sigma}_k}\frac{\partial}{\partial \boldsymbol{R}_k} f(\boldsymbol{R}_k)$$

We can take derivatives with respect to $\boldsymbol{R}_k$

$$\frac{\partial \boldsymbol{R}_k}{\partial \boldsymbol{\Sigma}_k}\frac{\partial}{\partial \boldsymbol{R}_k}\left(-\frac{N_k}{2}\ln|\boldsymbol{R}_k| - \frac{1}{2}\sum_{i=1}^{N_k} tr\left[\left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)\left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)^T \boldsymbol{R}_k\right]\right)$$

$$\frac{\partial \boldsymbol{R}_k}{\partial \boldsymbol{\Sigma}_k}\left(-\frac{N_k}{2}\left(\boldsymbol{R}_k^{-1}\right)^T - \frac{1}{2}\sum_{i=1}^{N_k}\left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)\left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)^T\right)$$

But, as the result has to be equal to 0. We can simplify

$$\frac{\partial \boldsymbol{R}_k}{\partial \boldsymbol{\Sigma}_k}\left(-\frac{N_k}{2}\left(\boldsymbol{R}_k^{-1}\right)^T - \frac{1}{2}\sum_{i=1}^{N_k}\left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)\left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)^T\right) = 0$$

$$\left(-\frac{N_k}{2}\left(\boldsymbol{R}_k^{-1}\right)^T - \frac{1}{2}\sum_{i=1}^{N_k}\left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)\left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)^T\right) = 0$$

Changing back $\boldsymbol{R}_k^{-1} = \boldsymbol{\Sigma}_k$. And due to the fact that $\boldsymbol{\Sigma}_k$ is symmetric, and therefore $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}_k^T$. The previous expression yields

$$\left(-\frac{N_k}{2}\boldsymbol{\Sigma}_k - \frac{1}{2}\sum_{i=1}^{N_k}\left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)\left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)^T\right) = 0$$

$$\widehat{\boldsymbol{\Sigma}_k} = \frac{1}{N_k}\sum_{i=1}^{N_k}\left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)\left(\boldsymbol{x}_i^{(k)} - \boldsymbol{\mu}_k\right)^T$$

Appendix B - Proof of Dirichlet Process Posterior

Let $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_n$ be samples generated following a DP with concentration parameter α and base distribution $H(\boldsymbol{\theta})$, that is $G(\boldsymbol{\theta}) \sim DP(\alpha, H)$. Let $A_1, \dots, A_r$ be a finite measurable partition of sample space Θ. Then, due to (2.14)

$$\big(G(A_1), \dots, G(A_r)\big) | \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_n \sim Dir(\alpha H(A_1) + n_1, \dots, \alpha H(A_r) + n_r)$$

We will prove that this posterior also follows a DP with parameters

$$\alpha' = \alpha + N$$

$$H'(\boldsymbol{\theta}) = \frac{\alpha H(\boldsymbol{\theta}) + \sum_{i=1}^{N} \delta(\boldsymbol{\theta} - \boldsymbol{\theta}_i)}{\alpha + N}$$

The problem is reduced to prove that

$$\alpha H(A_i) + n_i = \alpha' H'(A_i)$$

Start by assuming that $\alpha' = \alpha + N$, where N is the total number of samples

$$\alpha H(A_i) + n_i = (\alpha + N) H'(A_i)$$

Then, isolate $H'(A_i)$

$$H'(A_i) = \frac{\alpha H(A_i) + n_i}{\alpha + N}$$

We need a way to define n_i, as defined previously $n_i = |\{k: \boldsymbol{\theta}_k \in A_i\}|$, if we take $A_i = \boldsymbol{\theta}$, to get the more general expression $H'(\boldsymbol{\theta})$, we can define $n_i = |\{i: \boldsymbol{\theta}_k = \boldsymbol{\theta}\}|$, making use of the δ function

$$n_i = \sum_{k=1}^{N} \delta(\boldsymbol{\theta} - \boldsymbol{\theta}_k)$$

Then,

$$H'(\boldsymbol{\theta}) = \frac{\alpha H(\boldsymbol{\theta}) + \sum_{k=1}^{N} \delta(\boldsymbol{\theta} - \boldsymbol{\theta}_k)}{\alpha + N}$$

Finally, we can conclude

$$G(\boldsymbol{\theta}) | \boldsymbol{\theta}_i, \dots, \boldsymbol{\theta}_n \sim DP\left(\alpha + N, \frac{\alpha H(\boldsymbol{\theta}) + \sum_{i=1}^{N} \delta(\boldsymbol{\theta} - \boldsymbol{\theta}_i)}{\alpha + N}\right) \blacksquare$$

Appendix C - Expressions for Gaussian-Wishart Distribution

The following Gaussian-Wishart Distribution expressions are considered [10].

Posterior Hyperparameters

$$\boldsymbol{r'} = \boldsymbol{r} + \boldsymbol{n}$$
$$\boldsymbol{\nu'} = \boldsymbol{\nu} + \boldsymbol{n}$$
$$\boldsymbol{m'} = \frac{\boldsymbol{rm} + \sum \boldsymbol{x_i}}{\boldsymbol{r} + \boldsymbol{n}}$$
$$\boldsymbol{S_x} = \boldsymbol{S} + \sum \boldsymbol{x_i x_i^T} + \boldsymbol{rmm^T} - \boldsymbol{r'm'm'^T}$$

Marginal Probability

$$p(\boldsymbol{x}) = \pi^{-nd/2} \frac{r^{d/2}|S|^{\nu/2}}{r'^{d/2}|S_{\boldsymbol{x}}|^{\nu'/2}} \frac{\prod_{i=1}^{d} \Gamma\left(\frac{\nu' + 1 - i}{2}\right)}{\prod_{i=1}^{d} \Gamma\left(\frac{\nu + 1 - i}{2}\right)}$$

Predictive Probability

We are interested in efficiently computing the predictive distribution for the Gaussian-Wishart Distribution, given a new sample y. Precisely, we want to compute

$$p(y|\boldsymbol{x}) = \frac{p(\boldsymbol{x}, y)}{p(\boldsymbol{x})}$$

Thus, using the marginal probability formula we reach the following expression, where n is the number of elements

$$p(y|\boldsymbol{x}) = \frac{\pi^{-(n+1)d/2} \frac{r^{d/2}|S|^{\nu/2}}{(r+n+1)^{d/2}|S_{x,y}|^{(\nu+n+1)/2}} \frac{\prod_{i=1}^{d} \Gamma\left(\frac{\nu + n + 2 - i}{2}\right)}{\prod_{i=1}^{d} \Gamma\left(\frac{\nu + 1 - i}{2}\right)}}{\pi^{-nd/2} \frac{r^{d/2}|S|^{\nu/2}}{(r+n)^{d/2}|S_{\boldsymbol{x}}|^{(\nu+n)/2}} \frac{\prod_{i=1}^{d} \Gamma\left(\frac{\nu + n + 1 - i}{2}\right)}{\prod_{i=1}^{d} \Gamma\left(\frac{\nu + 1 - i}{2}\right)}}$$

Simplifying the previous equations

$$p(y|\boldsymbol{x}) = \frac{\pi^{-(n+1)d/2} \frac{\prod_{i=1}^{d} \Gamma\left(\frac{\nu + n + 2 - i}{2}\right)}{(r+n+1)^{d/2}|S'_{x,y}|^{(\nu+n+1)/2}}}{\pi^{-nd/2} \frac{\prod_{i=1}^{d} \Gamma\left(\frac{\nu + n + 1 - i}{2}\right)}{(r+n)^{d/2}|S'_{\boldsymbol{x}}|^{(\nu+n)/2}}}$$

Then, we can define a function $Z(d, n, r, \nu, S)$ in the following form

$$Z(d,n,r,\nu,S) = \pi^{-nd/2}\frac{\prod_{i=1}^{d}\Gamma\left(\frac{\nu+n+1-i}{2}\right)}{(r+n)^{d/2}|S|^{(\nu+n)/2}}$$

And, therefore rewrite the predictive probability as

$$p(y|\boldsymbol{x}) = \frac{Z(d,n+1,r+1,\nu+1,S_{x,y})}{Z(d,n,r,\nu,S_{\boldsymbol{x}})}$$

Another interesting result is that, we can define the marginal probability as

$$p(\boldsymbol{x}) = \frac{Z(d,1,r+n,\nu+n,S_{\boldsymbol{x}})}{Z(d,0,r,\nu,S)}$$

Which can be explained as the predictive probability when no previous samples exist.

Proof

Let

$$Z(d,n,r,\nu,S) = \pi^{-nd/2}\frac{\prod_{i=1}^{d}\Gamma\left(\frac{\nu+n+1-i}{2}\right)}{(r+n)^{d/2}|S|^{(\nu+n)/2}}$$

Then, $Z(d,0,r,\nu,S)$ equals to

$$Z(d,0,r,\nu,S) = \frac{\prod_{i=1}^{d}\Gamma\left(\frac{\nu+1-i}{2}\right)}{(r)^{d/2}|S|^{\nu/2}}$$

Therefore, as $\nu' = \nu + n$ and $r' = r + n$

$$p(\boldsymbol{x}) = \frac{Z(d,1,r+n,\nu+n,S_{\boldsymbol{x}})}{Z(d,0,r,\nu,S)} = \frac{\pi^{-nd/2}\frac{\prod_{i=1}^{d}\Gamma\left(\frac{\nu'+1-i}{2}\right)}{r'^{d/2}|S_{\boldsymbol{x}}|^{\nu'/2}}}{\frac{\prod_{i=1}^{d}\Gamma\left(\frac{\nu+1-i}{2}\right)}{r^{d/2}|S|^{\nu/2}}}$$

Finally, rewriting

$$p(\boldsymbol{x}) = \frac{Z(d,1,r+n,\nu+n,S_{\boldsymbol{x}})}{Z(d,0,r,\nu,S)} = \pi^{-nd/2}\frac{r^{d/2}|S|^{\nu/2}}{r'^{d/2}|S_{\boldsymbol{x}}|^{\nu'/2}}\frac{\prod_{i=1}^{d}\Gamma\left(\frac{\nu'+1-i}{2}\right)}{\prod_{i=1}^{d}\Gamma\left(\frac{\nu+1-i}{2}\right)}\blacksquare$$