Nonparametric Bayesian inference - a guide to clustering algorithms

Erik-Cristian Seulean

November 24, 2021

1 Motivation

In 2020, every human being created 1.7 MB of data per second. While this leads to an enormous quantity of data created, we are becoming more and more proficient at managing the data that we create by indexing or clustering in such ways that searching or retrieving datapoints is nowadays a nonissue. Today, there are over 6 million articles on Wikipedia. Regardless of the size of this article collection, everything is grouped in ways that allows us to explore different interests with just a few clicks. The library, here in St. Andrews contains over 1 million books, yet finding a book today on a topic such as Bayesian inference is a matter of minutes. As the University increases the diversity of degrees that students can take, the number of book categories increase over time and this leads to the following question *How do we generate a mathematical model that is capable to group books that are alike into well-defined categories ?* This essay describes a way of creating clustering algorithms for continuously increasing data collections with a focus on Nonparametric Bayesian methods. The goal is to start by introducing Bayesian Nonparametrics and the problem of data clustering from a fixed and known number of cluster, and relax this limitation to enable working with an unlimited number of clusters, using the Dirichlet process and variations such as Pólya Urn or Chinese Restaurant Process. In the end, a short description of how well known algorithms can be transformed to work for topic modelling, which could potentially be applied to cluster large text files.

2 Nonparametric Bayes a brief description

There are two separate classes of Bayesian inference. One class contains inference on parameters of distributions, where the prior and posterior distributions are functions of parameters of interest with fixed dimension. In this case we can consider the following:

- $\theta = (\theta_1, \theta_2, ..., \theta_n)$ - the parameter that we are interested in
- $P(\theta)$ - prior probability
- $\pi(\theta|X)$ - posterior probability

In the above example, the parameter has a fixed determined dimension. We know what probability distributions work for the number of parameters we have. For a single unknown parameter for example we can use a Uniform, Poisson or Exponential distribution. For two parameters we could use a Normal or Beta distribution (among others) depending on what sort of data we are modelling. In this situation we can use Bayes formula to define the posterior distribution based on the prior and likelihood:

$$\pi(\theta|x_1, x_2, ..., x_n) \propto \prod_i^n f(x_i|\theta)p(\theta)$$

where $f(x_i|\theta)$ denotes the likelihood of the data given the parameter.

But what happens when you don't know beforehand how many parameters you need ? How many parameters will describe your data well ? Are your observations dependent on one parameter or 3 parameters ? Nonparametric Bayesian analysis is focused on problems where the number of parameters are not known beforehand. In other words, we can assume that there can be any number of parameters, we will examine the data and draw some conclusions on how many parameters of interest describe the data well. In more rigorous mathematical terms, so far we can define the following:

- $X_1, X_2, X_3, ...X_n$ are i.i.d observations from an *unknown* distribution F
- F is not from a finite space, we cannot index it with a finite number of parameters

Similar to the parametric case, we would like to apply Bayesian methods. This means that we need to specify a prior distribution, but because we don't know the number of parameters that we are dealing with, we cannot specify priors on parameters, we need to specify a prior on the distribution itself. What in the parametric case was $P(\theta)$ becomes now $P(F)$ where F is a distribution! More exactly, $P(F)$ is a distribution over all the possible distributions that are available. To put it in the context of the example from the beginning of the essay, if we start making an analysis on the topics of the books in the library, the more books we examine the more topics we can encounter. That doesn't mean that every book is a new topic, but the number of topics grows as we examine more books. This is equivalent to the number of parameters in a nonparametric Bayesian model where the number of parameters is unknown but growing with the data.

3 The clustering problem

Assuming we have observations $X_1, X_2, ..., X_n$ we want to assign every datapoint to a single cluster. In a general Gaussian mixture model, with the number of clusters K set to 2, we could potentially have the following setup:

$$\mu_k \sim \mathcal{N}(\mu_0, \sigma_0^2)$$

$$\rho_1 \sim Beta(a, b)$$

$$\rho_2 = 1 - \rho 1$$

$$Z_n \sim Categorical(\rho_1, \rho_2)$$

Here we're choosing ρ_1 coming from a Beta distribution as it is mathematically convenient, but in practice, it would require further analysis to figure out if this is a valid assumption. Depending on the parameters of this Beta distribution, it would put more or less mass on ρ_1 compared to ρ_2. ρ_1 and ρ_2 represent, in this case, the probabilities to be part of one of the two cluster, while μ_k is a measure for the center of the cluster.

In case we want to generalize the model above to have K clusters, we can replace the Beta distribution above with a Dirichlet distribution. In this case, instead of drawing only one observation for ρ_1, we could draw K observation for $\rho_1, \rho_2, \rho_3, ..., \rho_K$.

$$\mu_k \sim \mathcal{N}(\mu_0, \sigma_0^2)$$

$$\rho_{1:K} \sim Dirichlet(a_1, a_2, ..., a_K)$$

$$Z_n \sim Categorical(\rho_1, \rho_2, ..., \rho_K)$$

The Dirichlet distribution used above is the natural generalization of Beta distribution, to a multiparameter case:

$$Dir(\rho_1, \rho_2, ..., \rho_n | a_1, a_2, ..., a_n) = \frac{\Gamma(\sum_{i=1}^{K} a_i)}{\prod_{i=1}^{K} \Gamma(a_i)} \prod_{i}^{K} \rho_i^{a_i - 1}$$

where $\sum_{i}^{K} \rho_i = 1, a_k > 0$ and $\rho_k \in (0, 1)$.

The *rdirichlet* function provided with the library can generate samples from a Dirichlet distribution.

So far the examples above, all work on the case where the number of clusters is fixed and known. We are capable of drawing observations from the Dirichlet distribution, because we know how many parameters we are working with. In the figure below, the first delimitation corresponds to the first cluster, second to the second cluster and so on, up to 100 clusters.

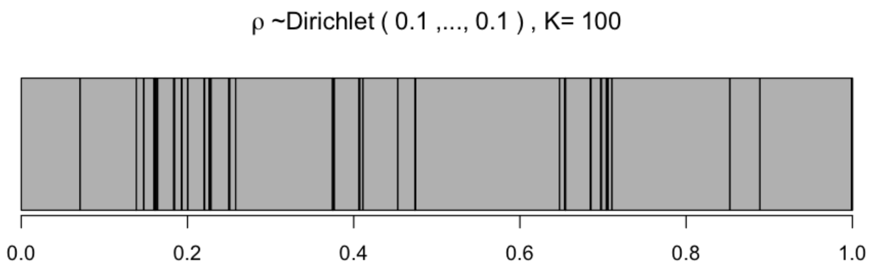

Figure 1: Clusters generated from a Dirichlet distribution.

At this point, we only considered the available clusters, but we have not sampled our datapoints $X_1, X_2, ..., X_n$ into the corresponding clusters. To put a particular datapoint into a certain cluster, we can draw an observation from a Uniform distribution and check which cluster it belongs to. We will sum up every cluster delimitation until the total sum is higher than the value sampled from the *Uniform distribution*. When we exceeded the sum, the latest ρ_k will denote the k^{th} cluster where the datapoint belongs to. We can repeat the same until all n datapoints are assigned to a cluster. The cluster generation function in the library allows to visually get an intuition of how this happens sequentially.

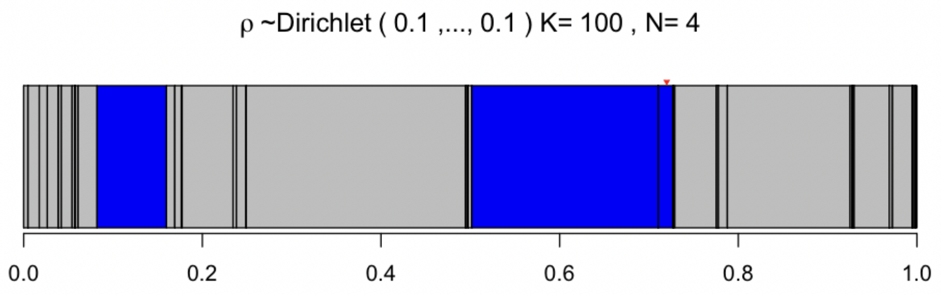

Figure 2: 4 points associated with the clusters generated

While this model looks appealing, there is always going to be an upper bound on K, that defines the number of clusters. In this scenario, there is no reliable way to choose K that would be sufficient in any general case. Even if we can come up with a value of K large enough, there are computational concerns when it comes to storing a large amount of information in memory that is associated with K.

4 Setting K=∞

A solution to the above limitation is to set K to be arbitrary large that would not limit us from having a very large number of clusters. In this case we need a different procedure to generate $\rho_1, \rho_2, ..., \rho_K$ from a Dirichlet distribution. What we want to do is take the interval $(0, 1)$ and break it into an infinite number of subintervals that sum up to 1. In other words, we first break a proportion of the interval $(0, 1)$ and denote that being ρ_1. Out of the remaining part $(\rho_1, 1)$ we break another proportion and denote that to be ρ_2. We can repeat this as many times as we want, and we can generate an infinite number of proportions, that sum up to 1, the length of the initial interval. This algorithm is called the *"stick-breaking process"* and can be summarized as follows:

- Generate $P_1 \sim Beta(a_1, b_1)$. Set ρ_1 to this value.
- Generate $P_2 \sim Beta(a_2, b_2)$. Set $\rho_2 = (1 - P_1)P_2$
- Generate $P_3 \sim Beta(a_3, b_3)$. Set $\rho_3 = (1 - P_1)(1 - P_2)P_3$
- Set $\rho_K = 1 - (\rho_1 + \rho_2 + ... + \rho_{K-1})$

The choice of parameters a and b for the Beta distribution are researched by Ishwaran and James, 2001 [1], and we will simply use 1 and α with α given for all a_k and b_k. When $a_k = 1$ and $b_k = \alpha$, $\alpha > 0$ the process is called *Dirichlet process stick-breaking.* The distribution over all $\rho = (\rho_1, \rho_2, ...)$ is called Griffin-Engen-McCloskey (GEM).

$$(\rho_1, \rho_2, ...) \sim GEM(\alpha)$$

5 Dirichlet process mixture model

The Gaussian mixture model specified above, for K fixed clusters can be transformed to work with an arbitrary number of clusters by using the GEM distribution instead of Dirichlet.

$$(\rho_1, \rho_2, ...) \sim GEM(\alpha)$$

$$\mu_k \sim \mathcal{N}(\mu_0, \sigma^2)$$

$$G = \sum_{k}^{\infty} \rho_k \delta_{\mu k} \sim DPM(\alpha, \mathcal{N}(\mu_0, \sigma^2))$$

The above will put weight ρ_k at $\delta_{\mu k}$ where $\delta_{\mu k}$ denotes the k^{th} cluster. The first parameter is α, parameter needed for the GEM distribution, the second one is the distribution used to draw $\delta_{\mu k}$, the locations of the clusters. In practice, sampling from this distribution would mean that with probability ρ_k will draw an observation from cluster with center $\delta_{\mu k}$.

The problem that arises with this model is that we cannot draw an infinite number of probabilities ρ_k, but we can draw them on demand, as we sample. The example above, where we draw points into the cluster, by sampling from a Uniform distribution and check which clusters the points belong to can give an indication of what we can do here. If we first draw a point $u \sim Uniform(0, 1)$ we can then sample enough ρ_k values that would make the sum of all ρ_k sampled be greater than u. The complete step by step sampling algorithm is given by the following:

1. Draw an observation u from a $Uniform(0, 1)$ distribution.
2. Generate $P_k \sim Beta(1, \alpha)$ and find ρ_k using the *stick breaking algorithm.* Repeat until $\sum_k \rho_k > u$.
3. For every ρ_k sampled, sample the center for the cluster associated with ρ_k from $\mathcal{N}(\mu_0, \sigma_0^2)$
4. Find the cluster the point belongs to such that $\rho_{k-1} < u < \rho_k$.
5. Sample an observation from $\mathcal{N}(\mu_k, \sigma^2)$ where μ_k is the center of the cluster that was selected.
6. Repeat until all n points are sampled.

The algorithm above can be observed step by step using *rDPMM_visual* in the package provided, while *rDPMM* returns a sample.

The *Dirichlet process* as presented above, is a distribution of distributions over a sample space. If we consider $A_1, A_2, ..., A_n$ a partition of the sample space Ω, we can say that the *Dirichlet process* assigns a particular mass, to each of this partitions of the sample space. The total mass distributed to each partition is distributed following a *Dirichlet distribution.*

$$(P(A_1), P(A_2), ..., P(A_k)) \sim Dir(\alpha G(A_1), \alpha G(A_2), ..., \alpha G(A_k))$$

Given that there is a connection between the *Dirichlet process* and the *Dirichlet distribution*, we can derive the posterior distribution when we have Dirichlet-Categorical or Dirichlet-Multinomial prior and likelihood pairs:

$$\rho_1, ..., \rho_k \sim Dir(\alpha_1, ..., \alpha_k)$$

$$x_1, ..., x_k \sim Categorical(\rho_1, ...\rho_k)$$

$$\pi(\rho|x) \propto f(x|\rho)\pi(\rho) \propto \prod_i^k f(x_i|\rho)\pi(\rho)$$

$$\propto \prod_i^k \prod_j^k \rho_j^{I(x_i=j)} \prod_i^k \rho_i^{\alpha_i - 1} \propto \prod_j^k \rho_j^{\sum_i I(x_i=j)} \prod_i^k \rho_j^{\alpha_j - 1}$$

$$\propto \prod_j^k \rho_j^{c_j + \alpha j - 1}$$

$c_j = \sum_i I(x_i == j)$

Similarly, if the observations are from a *Multinomial distribution*:

$$x_1, ..., x_k \sim Multinomial(\rho_1, ...\rho_k)$$

$$\pi(\rho|x) \propto f(x|\rho)\pi(\rho) \propto \prod_i^k f(x_i|\rho)\pi(\rho)$$

$$\propto \prod_i^k \rho_i^{c_i} \prod_i^k \rho_i^{\alpha - 1} \propto \prod_i^k \rho_i^{c_i + \alpha_i - 1}$$

In both cases, we can notice that the posterior is following a *Dirichlet distribution* so the Dirichlet distribution is the Conjugate prior for both the

Categorical and the Multinomial distributions. The posterior distribution in these cases is $\pi(\rho|x) \sim Dir(\alpha_1 + c_1, ..., \alpha_k + c_k)$. In terms of the clusters (or partition of the sample space), if we observe a new observation in cluster i, the posterior then is updated with a new observation in cluster i:

$$(P(A_1), P(A_2), ..., P(A_k)) \sim Dir(\alpha G(A_1), \alpha G(A_2), ..., G(A_i) + 1, ..., \alpha G(A_k))$$

which in fact leads to $G|x_1 \sim DP(\alpha + 1, \frac{\alpha \mathcal{N} + \delta_x}{\alpha + 1})$.

6 Pólya Urn

In the section above, drawing observations from the *Dirichlet process* required to sample values for ρ_k from the *GEM distribution*. If we could find a way to avoid this step, we could remove the ρ_k variables altogether and the model would be simpler. The motivation behind the Pólya urn is that in general, when we deal with clustering algorithms, we are rarely interested in the proportion of different clusters, we are mostly interested in the cluster assignments. It turns out that there is a way of doing this by integrating out ρ_k. As before, we could start from the same setup as in the Gaussian mixture model:

$$\rho_1 \sim Beta(a, b)$$

$$Z_n \sim Categorical(\rho_1, \rho_2)$$

$$
\begin{aligned}
P(Z_n = 1|Z_1, Z_2, ..., Z_{n-1}) &= \int P(Z_n = 1, \rho_1|Z_1, Z_2, ..., Z_{n-1})\, d\rho_1 \\
&= \int P(Z_n = 1|\rho_1)P(\rho_1|Z_1, Z_2, ..., Z_{n-1})d\rho_1 \\
&= \int \rho_1 Beta(a + c_1, b + c_2)d\rho_1 \\
&= \int \rho_1 \frac{\Gamma(a + c_1 + b + c_2)}{\Gamma(a + c_1)\Gamma(b + c_2)} \rho_1^{a+c_1}(1 - \rho_1)^{b+c_2} d\rho_1 \\
&= \frac{\Gamma(a + c_1 + b + c_2)}{\Gamma(a + c_1)\Gamma(b + c_2)} \frac{\Gamma(a + c_1 + 1)\Gamma(b + c_2)}{\Gamma(a + c_1 + b + c_2 + 1)} \\
&= \frac{a + c_1}{a + c_1 + b + c_2}
\end{aligned}
$$

where $c_j = \sum_i^{n-1} I(Z_i = j)$

In this case we are independent on ρ_k, and we can put an observation Z_k into the cluster one with probability derived above. Intuitively, this can be interpreted in the following way. If we start with an urn, containing initially a number of red balls and b blue balls and draw from a *Bernoulli distribution* with probability derived above, we can keep adding red or blue balls to the urn, based on the value sampled. As the number of balls in the urn goes to infinity, the proportion of red balls in the urn is ρ_1. An alternative intuition is that using the initial setup, we can draw a ball from the urn inspect the color and return to the urn two balls of that same color. As the number of balls grows to infinity the proportion of red balls goes to ρ_1. Not just the proportion of red to blue balls is ρ_1 as the number of balls goes to infinity, the distribution of the proportion of red balls is also *Beta(a, b)*. The distribution of red balls in the urn is noted as being *PólyaUrn(a, b)*.

The Pólya urn for two types of balls can be generalized to K different colors, but instead of using Beta we can use $\rho_1, ..., \rho_k \sim Dirichlet(\alpha_1, ..., \alpha_k)$. The probability of n^{th} observation to go into cluster k given the previous *n-1* observations is the same: $P(Z_n = k|Z_1, ..., Z_{n-1}) = \frac{\alpha_k + c_k}{\sum_i^n (\alpha_i + c_i)}$. If we keep adding balls to the urn, in the same procedure as above, the proportion of balls in the urn will follow the *Dirichlet distribution* that we started with.

As before, we built the model up starting with a *Beta distribution* for two clusters, then we generalized to K clusters using a *Dirichlet distribution.* The next logical step is to set $K = \infty$. To be able to handle an arbitrary number of clusters, we will use a different type of urn called *Hoppe urn* or *Blackwell-MacQueen urn.* The urn will initially contain only one magic ball. When the magic ball is extracted from the urn, we put it back together with a ball of a new color, that does not exist in the urn. If we extract a ball of a particular color, we will return it to the urn together with another ball of the same color. The distribution of balls of any color is *PólyaUrn(1, α).* The proportion of balls of the first color is also $Beta(1, \alpha)$, and this takes us back to the *Stick breaking process.* If we consider ρ_k the proportion of balls of the k^{th} color we have the following:

$$V_k \sim Beta(1, \alpha)$$

$$\rho_1 = V_1$$

$$\rho_2 = (1 - V_1)V_2$$

$$\rho_k = \prod_{i=1}^{k-1}(1 - V_i)V_k$$

The urn is identical to the Dirichlet process, but we don't have to deal with the *GEM distribution* anymore as the ρ_k were integrated out.

7 The chinese restaurant process

The *chinese restaurant process* works in the same way as the *Pólya urn.* A customer walks into a restaurant and either sits down at a new table or sits at a table that is already occupied, proportional to the number of people that already sit at the table. This process is identical to the Blackwell-MacQueen urn, we can either draw the magic ball (sit at a new table) or draw a different colored ball (sit at an already occupied table). Each table in this case will have a color, and as in the previous example we have infinitely many tables. A librarian working at the library in St. Andrews, can either assign a new book to an existing category or create a new category for that book.

It comes naturally to ask what is the difference between the *Dirichlet process* and the *Chinese Restaurant process.* While the *Dirichlet process* describes both the cluster assignments and the proportion of the clusters the *Chinese Restaurant process* describes only the cluster assignments. In many applications, knowing the cluster proportions is not of much use, so applications generally rely on implementations of the CRP algorithm for practical purposes. I was curious to find out where the name for the algorithm comes from, and I found out that the Chinese restaurants in San Francisco, in the nineties were trying to accommodate as many customers as possible. Customers were generally interested in the popular tables (with many other customers) but now and then, a customer would come in and sit at a new table.

Given that we are only considering the cluster arrangements and not the proportions for each cluster, we are interested in modelling the partitions of the datapoints. For example, one partition of 10 datapoints can be the following $\{\{1,5,7\},\{2,3,4,6\},\{8,9\}\}$. That denotes that the first datapoint, the 5^{th} and the 7^{th} are assigned in cluster one, and so forth. The order of the clusters doesn't matter, as the clusters are interchangeable, the datapoints will be associated with the same cluster regardless. It comes naturally to ask what is the probability of a cluster arrangement. The probability for the cluster arangement above is the following:

$$\frac{\alpha}{\alpha}\frac{\alpha}{\alpha+1}\frac{1}{\alpha+2}\frac{2}{\alpha+3}\frac{1}{\alpha+5}\frac{3}{\alpha+6}\frac{2}{\alpha+7}\frac{\alpha}{\alpha+8}\frac{1}{\alpha+9}$$

From here we can easily generalize to N datapoints. Considering K clusters and n_k datapoints in cluster k we have the following:

$$\frac{\alpha^K \prod_k (n_k - 1)!}{\alpha(\alpha+1)...(\alpha+N-1)} = P(\prod_N = \pi_N)$$

We notice here that the order in which datapoints are assigned to clusters doesn't change the probability of a particular arrangement. If we observe the partition of the 10 datapoints as above, the probability for the partition is the same if we permute the datapoints and consider them in a different order, as long as the datapoints are assigned to the same clusters as initially. This gives a property called exchangeability that comes handly in Gibbs sampling, which will be discussed in the later pages. This allows us to consider any datapoint, as being always the last one:

$$P(\prod_N | \prod_{N,-n}) = \begin{cases} \dfrac{\#C}{\alpha+N-2} & \text{if joins an existing cluster} \\ \dfrac{\alpha}{\alpha+N-2} & \text{if starts a new cluster} \end{cases}$$

Where $\#C$ denotes the number of datapoints in that cluster until this point.

8 Gibbs sampling for the Chinese restaurant process

Gibbs sampling for the Chinese restaurant process allows us to perform inference about our data. For the given datapoints, we can run the sampler and try to recover the clusters in the data. We will consider that the cluster means are drawn from a *Normal distribution* with known mean and variance μ_0 and Σ_0. In practice, this model might be simplistic, and we would probably set a prior on μ_0 and Σ_0 as well that would perform better on real datasets. Again, for simplicity purposes, we consider that the datapoints

$X_1, X_2, ..., X_n$ are drawn from a *Normal distribution* with known Σ and unknown mean. This can be sumarised as follows:

$$\mu_c \sim \mathcal{N}(\mu_0, \Sigma_0)$$

$$X_k|\mu_c \sim \mathcal{N}(\mu_c, \Sigma)$$

This setup allows us to take advantage of the fact that the Normal distribution is the conjugate prior for the Normal distribution. In other words, the posterior distribution will be a *Normal distribution* as well, with parameters updated based on the likelihood and prior.

The posterior distribution in this case is $P(\prod_N |X_1, X_2, ..., X_n)$. This posterior gives us the assignment of datapoints to clusters, given the existing datapoints. To be able to compute this posterior we will consider that we have the cluster assignments for $n-1$ datapoints, and we are only interested in assigning the current datapoint to a cluster.

$$P(\prod_N | \prod_{N,-n}, X_1, X_2, ..., X_n)$$

The n^{th} datapoint has two possible options, either join an existing cluster or start a new cluster itself. In that case we can write the probability as follows:

$$P(\prod_N | \prod_{N,-n}, X_1, ..., X_n) = \begin{cases} \dfrac{\#C}{\alpha + N - 2} P(C \cup \{X_n\}|C) & \text{if joins a cluster} \\ \dfrac{\alpha}{\alpha + N - 2} P(\{X_n\}) & \text{if starts a new cluster} \end{cases}$$

where $\#C = \sum_i^n \mathcal{I}(X_i \in C)$

$P(C \cup \{X_n\}|C)$ is the posterior distribution derived from the likelihood and the prior by completing the circle:

$$P(C \cup \{X_n\}|C) \sim N(\mu, \Sigma_x + \Sigma_0)$$

$$\Sigma_x^{-1} = \Sigma_0^{-1} + (\#C)\Sigma^{-1}$$

$$\mu = \Sigma(\Sigma_x^{-1} \sum_{k \in C} X_k + \Sigma_0^{-1}\mu_0)$$

where Σ^{-1} is the precision.

We can see that as the number of datapoints in the cluster grows, $\Sigma_0^{-1} + (\#C)\Sigma^{-1}$ will grow so Σ_x will go to 0. Similarly, μ will go towards the empirical mean of the cluster.

The implementation of the Gibbs algorithm can be found in the *cluster_datapoints* function provided with the library and is as follows:

1. Initialise all the datapoints in one single cluster.
2. Iterate from 1 to *maxIterations* where *maxIterations* denotes how many iterations the Gibbs sampling should perform.
3. Iterate through all the datapoints one by one. For every datapoint remove it from the cluster is currently assigned.
4. Calculate the log probabilities for the point belonging to the currently existing clusters and the log probability of starting a new cluster.
5. Normalize and transform the probabilities.
6. Sample the cluster from a Categorical distribution with parameters set to the probabilities derived above.
7. Assign the point to that cluster and continue to the next point.
8. When all the existing points are checked a new iteration of the Gibbs sampler is executed.

As mentioned earlier the implementation is simplistic, considering only one prior for the mean, with variance considered known. Alternatives to this algorithm include hierarchical models where we can set priors on both variance and mean and increase the performance of the clustering algorithm. There are also cases when the *Normal distribution* is not a good representation of the center of a cluster, so a better performance can be obtained if we choose a different distribution. An alternative to initialise every point belonging to a cluster is to randomly assign each datapoint to a random cluster and start the sampler with this configuration. We would expect that the sampler would adjust the number of cluster over multiple iterations and converge to the number of clusters presented by the data.

9 Results

In order to assess the performance of the clustering algorithm I tried the library on different datasets and I considered two options to assess how well it performs. First alternative was to consider the average number of iterations a datapoint spent in a particular cluster. This ended up being problematic because over a high number of iterations, the datapoints might switch clusters. Intuitively, consider that after n iterations, all points are correctly assigned to two clusters A and B. If I run the sampler for a high number of iterations, the points associated with cluster A might switch to cluster B and vice-versa. An alternative solution was to stop the clustering algorithm after a number of iterations and inspect the latest cluster association. This solution (at least on the datasets that I considered) seems to be giving relatively good results (at least visually) but in practice, the first solution might be more accurate if the cluster switching can be avoided.

First dataset that I tried the clustering algorithm on was a dataset that I generated. I tried to keep the cluster centers non-overlapping and relatively separated to have an initial assessment if it works to a certain degree. This dataset is available in the library as *split_data*.

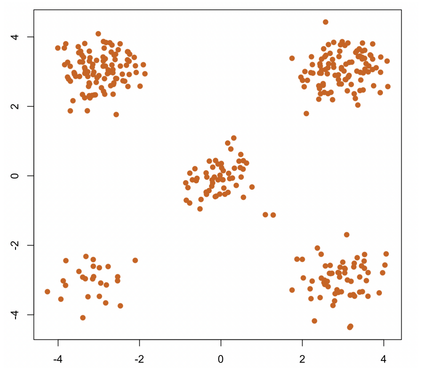

Figure 3: Before clustering

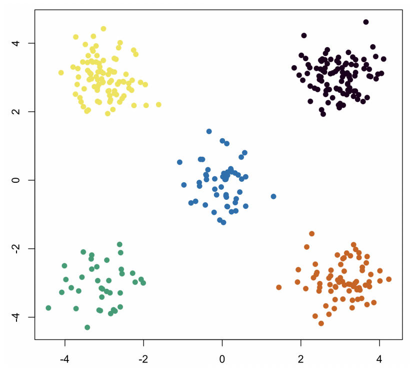

Figure 4: After 1000 iterations

Surprisingly (or not) this worked perfectly. The cluster separation is correct and as expected. At this point, I started getting excited, so I decided to test the clustering algorithm on some datasets that I did not generate. My first thought was to try the clustering algorithm on the iris dataset. This dataset was introduced by Ronald Fisher and contains 150 samples from 3 Iris species and is available in R - $data(iris)$. The dataset contains 4 features sepal length, sepal width, petal length, petal width, and the idea was to use two of these features at a time and differentiate between the flowers. Admittedly, while there are 3 different Iris species in the dataset, I was expecting to see two clusters as 2 of the different Iris species are very similar in features and I wouldn't expect to be able to distinguish between the 3 species without a hierarchical model.

Iris Petal Width vs Length

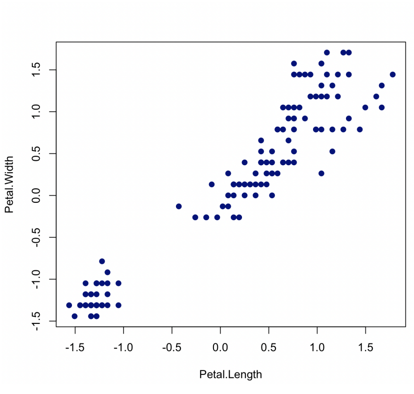

Figure 5: Before clustering

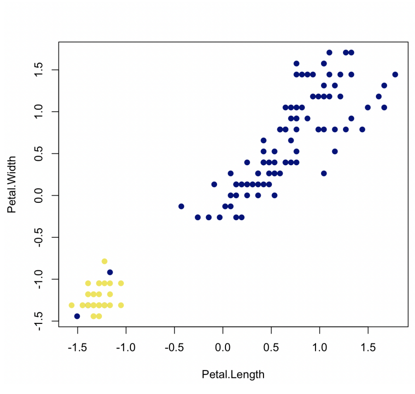

Figure 6: After 1000 iterations

Iris Sepal Width vs Petal Width

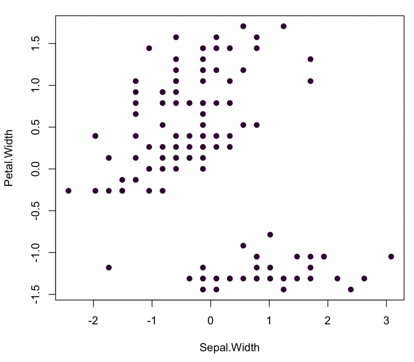

Figure 7: Before clustering

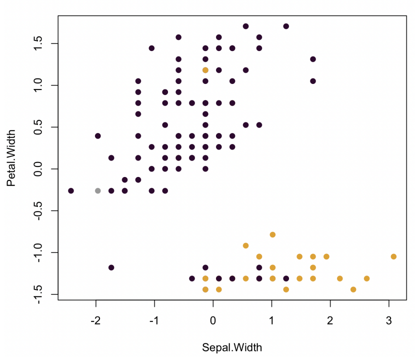

Figure 8: After 1000 iterations

The cluster correctly finds the number of clusters, however the separation of points into clusters is not perfect. I tried running the chain for a longer time but the points on the edge of the clusters sometimes get associated with the wrong cluster. Nevertheless, the algorithm got over 90% accuracy, calculated by considering how many of the 150 points were correctly associated with the correct cluster. It's worth mentioning that these results are experimental and convergence was not checked using metrics that could tell if the stability distribution was reached.

10 Completing the circle - clustering Books

I started this paper by setting up a scene to cluster books, or large text documents, but the reality is that I haven't explained how the clustering algorithm, in the form of the CRP can be applied to cluster large text files. Let's consider that every book in the collection can take a limited number of topics K. In order to cluster books, we would consider books that have the same topic as part of the same cluster. Each book is represented by a number of words $B = (w_0, w_1, ..., w_n)$ (after tokenization, removal of stopwords and lemmatization). We can consider a multi-CRP, where multiple restaurants serve the same number of dishes (topics) and each table will contain only one dish. Every restaurant will be linked to only one book, and the words in a book are "sitting" at only one table (representing the topic they are associated with). For every word in the book, it can either sit at a table with a particular topic or can start a table with a new topic. After all the words are assigned to tables, we have a probability distribution on the topics for this particular book. The word frequencies would represent how many people are sitting at a particular table at a given point. If we apply the same for all the books, we could associate books together using the probability distributions over the topics. In order to be able to do this, we would require prior probabilities on the topic given a particular word and priors on the topics as well. Prior probabilities for the topics and words given topics can be either learned as well or can be done via prior elicitation, for example, by considering well known books and categories as a priori knowledge from different experiments. This would build a hierarchical model that could cluster books of the same topics together.

In the latest years, such models are applied to cluster together Twitter tweets or social media posts by topics, or documents and the paradigm is called Latent Dirichlet Allocation (LDA). Each document is considered a mixture of topics and each topic is a mixture of words. [2]

References

[1] Herman Ishwaran, Lancelot F. James *Gibbs sampling methods for stick-breaking process*, 2001

[2] Blei, NG, Jordan *Latent Dirichlet Allocation*, 2003 `https://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf`

[3] Tamara Broderick *Nonparametric Bayesian Statistics* `tamarabroderick.com/files/broderick_mlss_cadiz_part_i.pdf`

[4] Gordon J. Ross, Dean Markwick *dirichletprocess: An R Package for Fitting Complex Bayesian Nonparametric Models*

[5] Junyang Chen, Zhiguo Gong, Weiwen Liu *A Dirichlet process biterm-based mixture model for short text stream clustering*

[6] Radford M. Neal *Markov Chain Sampling Methods for Dirichlet Process Mixture Models*

[7] Sivakumar Murugiah *Bayesian Nonparametric Clustering based on Dirichlet Processes*

[8] Peter Müller, Ritten Mitra *Bayesian Nonparametric Inference - Why and How*

[9] Michael Jordan, Tamara Broderick *Nonparametric Bayesian Methods: Models, Algorithms, and Applications I* `https://simons.berkeley.edu/talks/tamara-broderick-michael-jordan-01-25-2017-1`

A Code notes

I would suggest reading the paper and following the code simultaneously. The package provided has comments explaining how to use every function:

```
?generate_split_data
?rdirichlet
?cluster_datapoints
?rDPM_visual
?generate_dirichlet_clusters
?generate_dirichlet_clusters_with_sampled_points
```

I would recommend checking the functions before reading the paper, and calling the functions depending on the section being read. I left some notes in the essay specifying that a function is available for that part, hopefully, the function that I was referring to is fairly obvious from the context.

To load the code, proceed in the following way:

```
tar -xf nonparametric.bayes_0.0.1.tar.gz
cd nonparametric.bayes
R # this will open a new R shell
require(devtools)
devtools::load_all(".")
```

I don't use RStudio, but feel free to use your preferred way to load a library from the disk.

* An alternative that I tried is to upload the library to CRAN. I submitted a new request, but since this takes roughly 10 days (being manual), it might not be available by the time this essay is checked. It is worth trying the following:

```
  install.packages("nonparametric.bayes")
```

If the above doesn't work, please load the code locally as mentioned from the tar.gz provided.

In terms of platforms, this was developed on MacOS, hence the plots and graphics were checked on MacOS. I suspect this will work OK on any Linux distribution, however I have no guarantees on Windows, CRAN however, does some cross-platform compatibility checks which have passed. On MacOS the graphics might depend on Quartz, but I suspect this is installed by default on newer distributions.

If there are issues when loading the code please get in touch!