Bayesian Hierarchical Clustering Implementation and Optmization

Lidia Azucena Morales Vasquez and Zhuangdie Alan Zhou

Duke University, North Carolina, United States

Abstract

Clustering algorithms are a popular technique in unsupervised learning, one of the most common used is Agglomerative Hierarchical Clustering. Due to limitations of clustering algorithms, Bayesian Clustering was proposed by Katherine Heller and Ghahramani. This is a model-based approach in clustering, this allows to have a predictive distribution of new data. This article proposes an implementation in Python, based on their article. An optimization of the algorithm was also attempted, we were able to make it three times faster. The algorithm was test it with one real and one simulated data set. The final results is the comparison with the traditional hierarchical clustering method, using the F1 score.

Keywords: Bayesian, Hierarchical Clustering, DPM

1. Background

Clustering algorithms are a popular technique in unsupervised learning. The objective is to discover similar patterns among groups of data.

A common clustering method is Hierarchical Clustering. The user does not need to know the number of clusters beforehand, by looking at the binary tree (dendrogram) one can decide where is more appropriate to cut. This hierarchy allow multiple levels of granularity. There are two types: agglomerative and divisive. The first one starts with each data point belonging to its own cluster, the two closest one merge together until there is only one cluster. The latter one begins with one cluster and then splits in two, until each data point is in its own cluster. Nevertheless, there is not a decision rule for an optimal cut or which distance metric to use.

Bayesian Hierarchical Clustering, based on the paper by Katherin A. Heller and Zoubin Ghahrmani (2005)[1], has the same approach but instead of merging data points that are close to each other according to a distance metric, it uses a probabilistic model criterion. By computing the marginal likelihood of all combinations of two clusters consistent with the binary tree, the two clusters with the highest probability are merged together. The proposed method can also be viewed as an approximation inference for Dirichlet Process Mixture, a model frequently used in non-parametric Bayesian Statistics. (reference)

This algorithm can be performed where other clustering methods are being applied. For instance, in the paper the author used to detect whether an email was spam or not, classifying the type of glass or handwritten recognition. There are other fields in which clustering is popular such as market research, customer segmentation, or in biology, for clustering an organism at the species or genus level.

Some of the advantages has already been mentioned before, such as it can be viewed as approximate inference of DPM or having a probabilistic model of the data, which can be later used to compute the predictive distribution of new data of belonging to any clusters in the tree. Other advantages are the possibility of integrating prior information to our model. The use of priors also avoid the problem of overfitting the data. The depth of the tree can be decided by bayesian hypothesis testing.

On the other hand, disadvantages of this algorithm is that uses conjugate priors to model the data in order to keep track of the marginal likelihoods, this restrict the data to a few distributions. It has a complexity $O(n^2)$ as standard Hierarchical Clustering for building the three, but the calculations are computationally more intensive. The final tree is not necessarily the global optimal, this characteristic is shared with other types of greedy algorithms.

Finally, choosing the which model to user and the hyperparameters is not an easy task, the article has a section for hyperparameters optimization. This part is not covered in our algorithm, but if implemented it would add more time and complexity to the algorithm.

The implementation of the algorithm was done by following the paper. After implementation, we tried several approaches for optimization. These methods will be discussed in next sections. Along with some examples and comparisons with common types of clusterings, such as K-means and Agglomerative Hierachical Clustering.

2. Algorithm

The goal of the algorithm is to merge data points based on a threshold formed with Bayesian logistics. There are two hypotheses when you are making the decision of whether to merge two data points or trees. First, we assume the two data points or trees are from one cluster. Second, we assume the two data points and trees are from two or more clusters. Then based on our experience we assign a probability to the occurrence of each hypothesis as the weights. Using Bayesian formula we can know the probability of merging the two data points or trees. Each times we find the two data points or trees with the highest probability of merging and merge them. In this way we form the whole tree of clustering until there is only one tree left.

The implementation of this method follows the steps below.

2.1. Cluster Initialization

For each data point in the data set $\mathcal{D} : x_1, \ldots, x_n$, initialize them by assigning them clusters from 1 to n.

2.2. Hypotheses Under Dirichlet Process Mixture Model

- Null Hypothesis $\mathcal{H}_1^k$:

$\mathcal{D}_k$ is a sub data set of data set $\mathcal{D}$ at tree $\mathcal{T}_k$. If data $\mathcal{D}_k$ at tree $\mathcal{T}_k$ is generated from the same cluster, the probability that $\mathcal{D}_k$ is under $\mathcal{H}_1^k$ can be computed as follows:

$$p(\mathcal{D}_k|\mathcal{H}_1) = \int p(\mathcal{D}\theta)p(\theta|\beta)d\theta$$

- Alternative Hypothesis $\mathcal{H}_2^k$:

If data $\mathcal{D}_k$ have two or more clusters in it, the probability of the data under alternative hypothesis is as follows:

$$p(\mathcal{D}_k|\mathcal{H}_2^k) = p(\mathcal{D}_i|T_i)p(\mathcal{D}_j|T_j)$$

- Marginal Probability of Data:

π_k is the prior under $\mathcal{H}_1$ that all data points in $\mathcal{D}_k$ belong to one cluster. Accordingly, $(1-\pi_k)$ is the prior under $\mathcal{H}_2$ that data points in $\mathcal{D}_k$ belong to more than one cluster.

Combining the probability of the data under two hypotheses weighted by the priors we can get the marginal probability of the data in the tree $\mathcal{T}_k$ as follows:

$$p(\mathcal{D}_k|T_k) = \pi_k p(\mathcal{D}_k|\mathcal{H}_1^k) + (1-\pi_k)p(\mathcal{D}_i|T_i)p(\mathcal{D}_j|T_j)$$

2.3. Probability of Merged Hypotheses

In $\mathcal{D}$, we find the pair $\mathcal{D}_i$, $\mathcal{D}_j$ with the highest probability of the merged hypothesis and calculate the probability as below:

$$r_k = \frac{\pi_k p(\mathcal{D}_k|\mathcal{H}_1^k)}{p(\mathcal{D}_k|\mathcal{T}_k)}$$

$$= \frac{\pi_k p(\mathcal{D}_k|\mathcal{H}_1^k)}{p(\mathcal{D}_k|\mathcal{H}_1^k) + (1-\pi_k)p(\mathcal{D}_i|T_i)p(\mathcal{D}_j|T_j)}$$

After finding the pair $\mathcal{D}_i$, $\mathcal{D}_j$ that have the highest probability to merge, we merge them to a new cluster $\mathcal{D}_k$ and delete the two clusters $\mathcal{D}_i$, $\mathcal{D}_j$.

2.4. Prior Computation

π_k is the prior for $\mathcal{H}_1^k$ that all data points in $\mathcal{D}_k$ are in one cluster. To calculate π_k in each merge, we first initialize each data point or leaf i to have $d_i = \alpha$, $\pi_i = 1$. $d_{left_k}$and d_{right_k} are indices of the right and left subtrees of T_k. Then for each internal node k, we can calculate π_k by the following equations:

$$d_k = \alpha\Gamma(n_k) + d_{left_k}d_{right_k}$$

$$\pi_k = \frac{\alpha\Gamma(n_k)}{d_k}$$

2.5. Trees Cutting

After the whole clustering tree is grown, users can either choose the number of clusters to cut the tree or set a threshold for r_k to cut the tree.

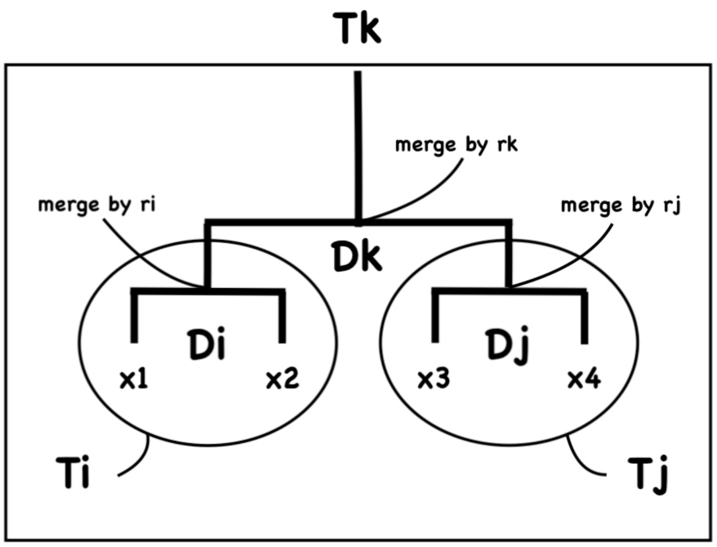


Figure 1: Schematic of how trees grows. Users can cut the tree with desired r_k.

3. Models

In this implementation we used two different marginal likelihoods coming from the exponential family, mentioned in the article and in Heller thesis [2].

3.1. Bernoulli-Beta

The Bernoulli distribution models a binary outcome.

$$P(\theta|\mathcal{D}) = \prod_{i=1}^{N}\prod_{d=1}^{D}\theta_d^{x_d^{(i)}}(1-\theta_d)^{(1-x_d^{(i)})} \tag{1}$$

where
N: Number of data points in the dataset
D: total number of dimensions

θ_d : Probability of outcome 1 for the dth dimension of observation i.

The conjugate prior is the beta distribution:

$$P(\theta|\alpha,\beta) = \prod_{d=1}^{D} \frac{\Gamma(\alpha_d+\beta_d)}{\Gamma(\alpha_d)\Gamma(\beta_d)} \theta_d^{\alpha_d-1}(1-\theta_d)^{\beta_d-1} \tag{2}$$

with α and β are beta hyperparameters.
The marginal likelihood for this model is:

$$P(D|\alpha,\beta) = \prod_{d=1}^{D} \frac{\Gamma(\alpha_d+\beta_d)\Gamma(\alpha_d+m_d)\Gamma(\beta_d+N-m_d)}{\Gamma(\alpha_d)\Gamma(\beta_d)\Gamma(\alpha_d+\beta_d+N)} \tag{3}$$

3.2. Multinomial-Dirichlet

The Multinomial distribution is the generalization of the binomial distribution for more than two outcomes.

$$P(\theta|\mathcal{D}) = \prod_{i=1}^{N} \theta_1^{x_K^{(i)}} ... \theta_K^{x_K^{(i)}} \tag{4}$$

where
N: Number of data points in the dataset
K : Number of bins
$x_k^{(i)}$: Indicator variable that the ith data is in k class
θ_i: Probability of i class and $\sum_i \theta_i = 1$

The conjugate prior is the Dirichlet distribution, a generalization of the beta distribution.

$$P(\theta|\alpha) = \prod_{d=1}^{D} \frac{\Gamma\left(\sum_{k=1}^{K}\alpha_k\right)}{\prod_{k=1}^{K}\Gamma(\alpha_k)} \theta_1^{\alpha_{i-1}} ... \theta_K^{\alpha_K-1} \tag{5}$$

with α is a Dirchilet hyperparameter.
The marginal likelihood for this model is:

$$P(D|\alpha) = \prod_{d=1}^{D} \frac{\Gamma\left(\sum_{k=1}^{K}\alpha_k\right)\prod_{k=1}^{K}\Gamma(\alpha_k+m_k)}{\Gamma\left(M+\sum_{k=1}^{K}\alpha_k\right)\prod_{k=1}^{K}\Gamma(\alpha_k)} \tag{6}$$

4. Optimization

The algorithm is really slow compared to a standard hierarchical clustering. A big limitation was the efficiency of our algorithm, it was not possible to test it in data sets with more than 200 rows, we had to work with subsets of data sets in order to test it. One main disadvantage is that python does not provide a tree structure, and we have to build one from scratch by looking to other examples. This was also an inconvenient when trying to improve it using Numba and Cython.

4.1. First approach

For our first approach, we tried to optimize the code by vectorizing the loops in the function that calculates the probabilities. This was useful to make both, the Bernoulli-Beta and Dirichlet-Multinomial model faster than our first attempt.

Additionally, we decided to remove some attributes from the node class, to avoid carrying with information that was not relevant to the problem. Finally, we also tried to change the attribute data to indexes, but this attempt did not work because the data still needs to be provided in the functions parameters.

4.2. Numba

Our first approach was to use the jit decorator from numba, to see if this could improve the time. We faced two problems, one is that is harder to specify class instances and the second one, is that it does not handle functions such as gamma.

4.3. Cython

Since we are using more complex structures in Python, such as dictionaries and object classes, it was not possible to use Cython at is full capacity. Although the time was improved by just using Cython decorator.

4.4. Cache

Since the algorithm does some calculations more than one time, we decided to use cache function. This made the algorithm faster than the previous one.

4.5. Comparison

We compared the time of our original code with the final version in the same data set. After the modifications, we were able to make it three times faster in data set of 62 data points and 13 columns. Below are the results.

Before optmization

```
%timeit clus,a=bhc(Xs_r1,500,beta=None)
```

```
32.1 s ± 2.38 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

After optmization

```
%timeit clus,a=bhc(Xs_r1,200,beta=None)
```

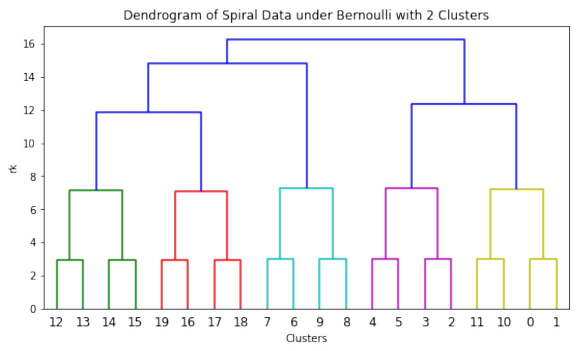
```
9.99 s ± 1.51 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

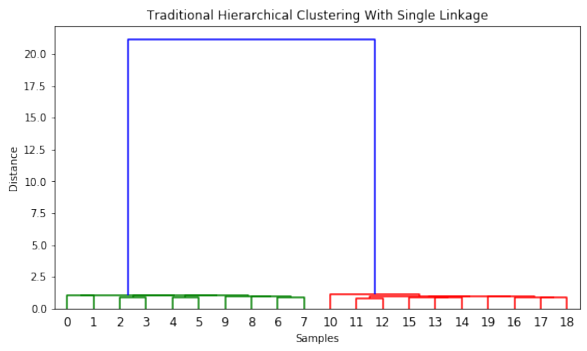
5. Applications to Simulated Data Sets

5.1. Bernoulli Prior

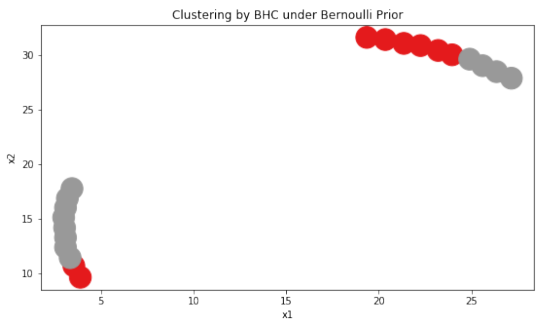
First we use a simulated Spiral data set under Bernoulli Prior using BHC clustering.

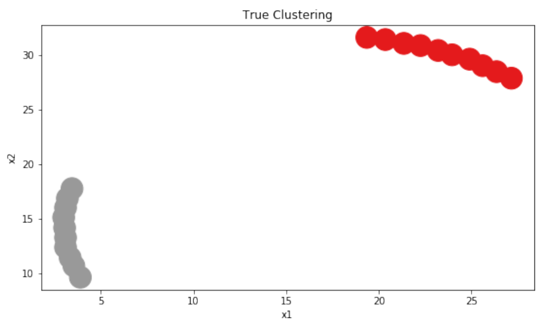
We compare the dendrogram with traditional hierarchical clustering with single linkage as below.





Also, clustering by the algorithm and true clustering are shown below.

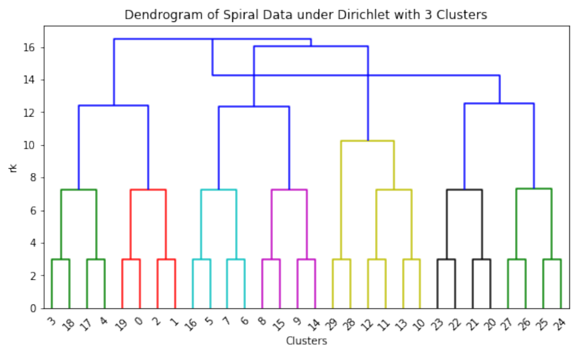


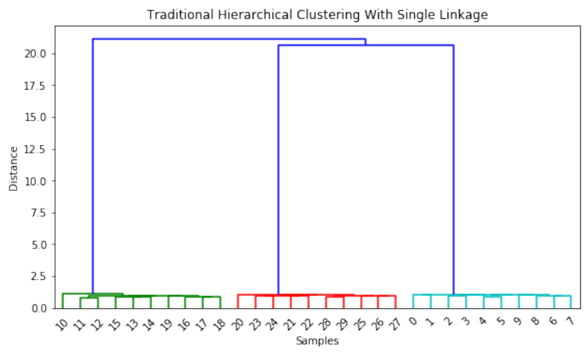


5.2. Dirichlet Prior

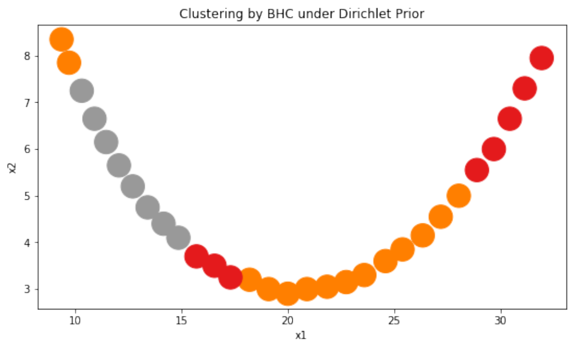
As large parts of this article are talking about Dirichlet, we use a simulated Spiral data set under Dirichlet Prior using BHC clustering.

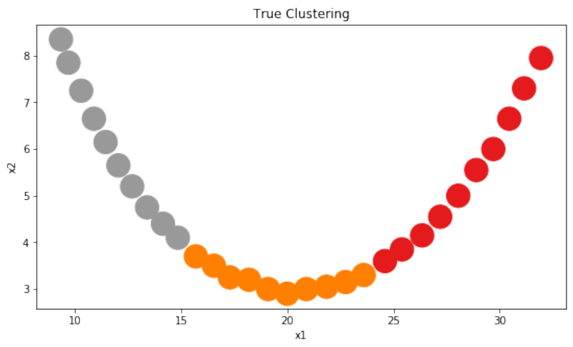
We compare the dendrogram with traditional hierarchical clustering with single linkage as shown below.





Also, clustering by the algorithm and true clustering are shown below.



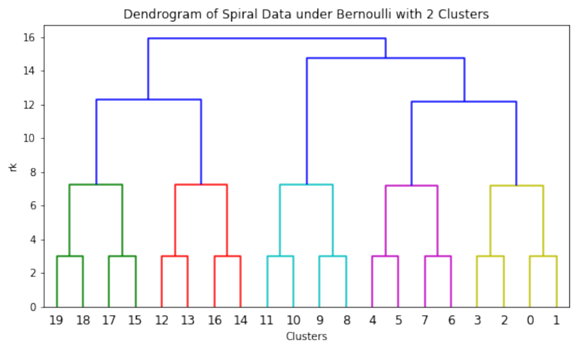


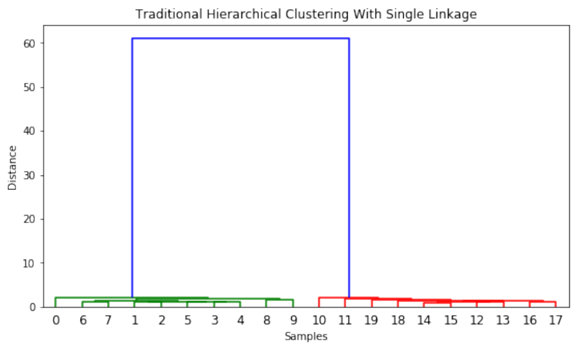
6. Applications to Real Data Sets

6.1. Bernoulli Prior

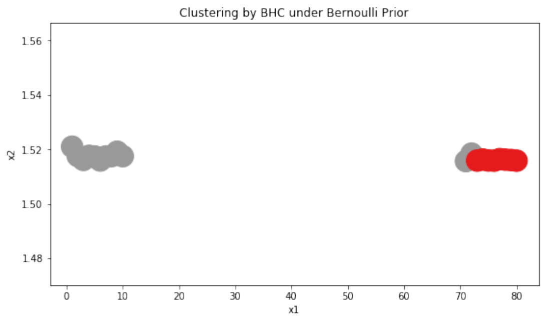
First we use a real data set glass from UCI under Bernoulli Prior using BHC clustering.

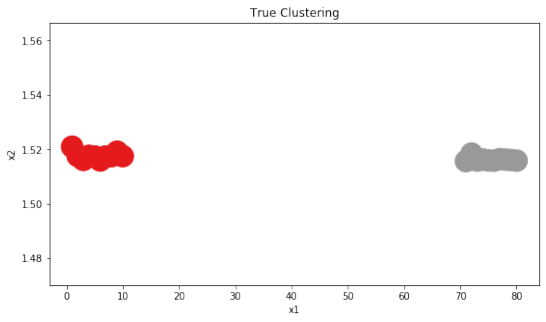
We compare the dendrogram with traditional hierarchical clustering with single linkage as shown below.





Also, clustering by the algorithm and true clustering are shown below.

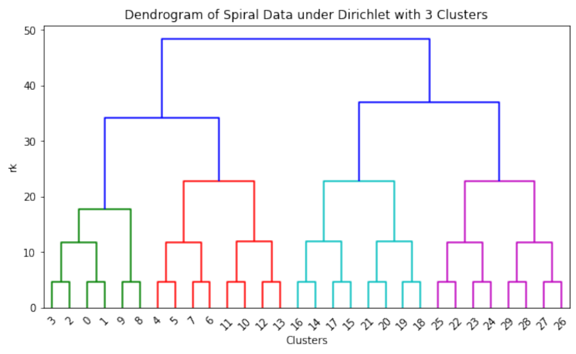


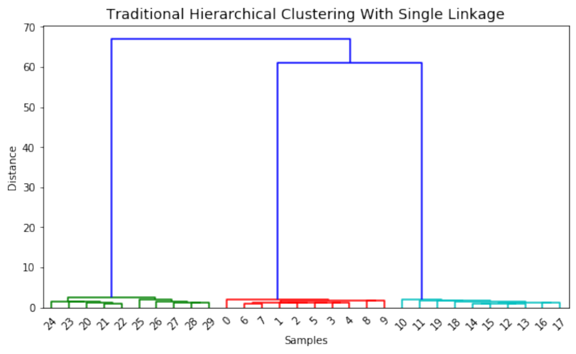


6.2. *Dirichlet Prior*

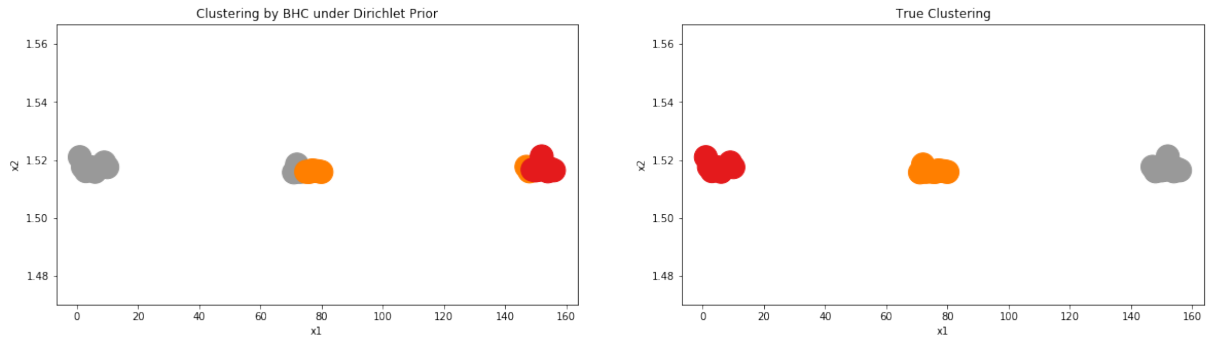
As large parts of this article are talking about Dirichlet, we use the glass data set under Dirichlet Prior using BHC clustering.

We compare the dendrogram with traditional hierarchical clustering with single linkage as shown below.





Also, clustering by the algorithm and true clustering are shown below.



Prior	Spiral Data	Glass Data
Bernoulli	0.667	0.909
Dirichlet 2	0.676	0.676

Table 1: F1 Score

From the F1 Score table we can see that Glass Data on Bernoulli prior performs best and Spiral Data on Bernoulli the worst if we take F1 score as our metric.

7. Conclusion

It was a challenge to implement this algorithm. We had to review the paper several times and other implementations, we also made a few assumptions over the priors. Although the algorithm seems to perform reasonable, for the metric that we have chosen is performing poorly compared with the traditional hierarchical clustering. This can be improved by updating the priors, but this will add more complexity to the model.

The main problem with this algorithm is its complexity, it is really slow compared to other popular clustering techniques. If the data set has above 200 rows the algorithm will take minutes to finish, even with the modifications. Another problem this algorithm has is interpreting the dendrogram, since the height of the lines does not have any meaning. In hierarchical clustering, it was clear were to cut based in the distance.

As mentioned before, this algorithm also is limited to the distributions that have a conjugate prior. It does not give a global optimal tree, and the tree is fixed. The main advantage we see in this method is that it gives you the possibility of calculating the predictive distribution of new data of belonging to any cluster in the tree, in that sense it fulfill its purpose.

References

[1] K. A. Heller, Z. Ghahramani, Bayesian Hierarchical Clustering., ICML 21 (2005).

[2] K. A. Heller, Efficient Bayesian Methods for Clustering, PhD thesis, Gatsby Unit, UCL (2008).