Bayesian Hierarchical Clustering

Muxin Diao, Chenyang Wang
Course: STA 663 - Statistical Computation

Abstract— **This paper will focus on Bayesian Hierarchical Clustering (BHC), proposed by K. A. Heller and Z. Ghahramani (2005)[1]. Compared with traditional agglomerative clustering model using distance metrics, BHC uses a probability model to merge existing clusters. We implement the BHC algorithm described in the original paper using plain python with critical functions rewritten by c++ wrapper and test it under simulated data and real-world data. We also compare the performance of BHC with traditional agglomerative clustering models.**

Keywords— **Bayesian, hierarchical clustering, posterior probability, marginal probability, Dirichlet process mixture model**

I. Background

Unsupervised learning[2], or clustering is an important branch of machine learning, where agglomerative hierarchical clustering is one of the popular clustering methods. Agglomerative hierarchical clustering uses bottom-up approach to form clusters, where it starts from assigning each data point as its own cluster, then merging two clusters at a time, until there is only one cluster. The advantage of this approach is that unlike another popular clustering methods, e.g. K-means, Gausian Mixture Model (GMM), it does not need to specify the number of clusters in the beginning. Also, the algorithm of this method is simpler than training K-means or GMM where EM algorithm is involved, as it only needs to evaluate the distance between clusters. However, the traditional agglomerative hierarchical clustering has trouble with choosing a proper distance metric, i.e. linkage. There are several linakge methods used including

1) 'single' linkage which uses the distance between nearest points
2) 'average' linkage which uses the average distance of all pairs of points
3) 'complete' linkage which uses the farthest distance of two clusters
4) 'centroid' linkage which uses the distance between the centroids of two clusters

In reality, choosing different distance metrics often produce different results. Also, in the framework of traditional hierarchical clustering, adding a new data point requires rebuild the whole tree, which is time consuming.

Bayesian Hierarchical Clustering (BHC) is a novel type of bottom-up hierarchical clustering approach. Instead of merging by the linkage, it uses posterior probability to decide which two clusters to be merged. Hence, the natural advantage of BHC is the consistency, where it specifies a single probability model to describe the data and gets rid of choosing proper linkage. Also, when new data points coming in, BHC can make probabilistic predictions using $\mathcal{O}(n)$ time complexity.

As mentioned by Heller and Ghahramani (2005)[1], who jointly proposed this method, BHC can provide competitive clustering for real-world datasets. The application of BHC is as wide as other clustering methods, for example, automatic spam email identification, customer grouping. Our paper mainly focuses on efficient implementation of the BHC algorithm proposed by Heller and Ghahramani (2005), and compare its performance with traditional agglomerative clustering methods.

II. Description of Algorithm

This part will describe the BHC algorithm. We keep the notations consistent with the original paper. Let $\mathcal{D} = \{x^{(1)}, ..., x^{(n)}\}$ denote the whole dataset, $\mathcal{D}_i$ is the set of data points in subtree $\mathcal{T}_i$. The algorithm uses Dirichlet process mixture (DPM) model as the generative model and focuses

on the marginal probability of the subtree $\mathcal{T}_i$, given by

$$P(\mathcal{D}_k|\mathcal{T}_k) = \pi_k P(\mathcal{D}_k|\mathcal{H}_1^k)+(1-\pi_k)P(\mathcal{D}_k|\mathcal{T}_i)P(\mathcal{D}_k|\mathcal{T}_j)$$

We denote $\mathcal{H}_1^k$ as the hypothesis that all the data points in $\mathcal{D}_k$ are i.i.d. generated from the same probabilistic model $P(x|\theta)$ with unknown parameters θ with some priors. The likelihood $P(\mathcal{D}_k|\mathcal{H}_1^k)$ can be from any prior-likelihood family, e.g. Beta-Bernoulli, Normal-Inverse Wishart distributions. In addition, $\mathcal{D}_k = \mathcal{D}_i \bigcup \mathcal{D}_j$, and $\pi_k := P(\mathcal{H}_1^k)$ which is calculated bottom-up in DPM. Then, the posterior probability we take the most interest in is given by

$$\begin{aligned} r_k &:= P(\mathcal{H}_1^k|\mathcal{D}_k) \\ &= \frac{\pi_k P(\mathcal{D}_k|\mathcal{H}_1^k)}{\pi_k P(\mathcal{D}_k|\mathcal{H}_1^k) + (1-\pi_k)P(\mathcal{D}_k|\mathcal{T}_i)P(\mathcal{D}_k|\mathcal{T}_j)} \end{aligned}$$

The posterior r_k is used to decide which two clusters to merge. The algorithm is described below.

Algorithm 1 Bayesian Hierarchical Clustering

procedure BUILD_TREES($\mathcal{D}$, $P(\mathcal{D}_k|\mathcal{H}_1^k)$, α)
  $c \leftarrow n$
  $\mathcal{D}_i \leftarrow \{x^{(i)}\}$ for $i = 1, ..., n$
  $d_i = \alpha$, $\pi_i = 1$ for $i = 1, ..., n$
  while $c > 1$ **do**
    for each $\mathcal{D}_i, \mathcal{D}_j$ from the clusters **do**
      $\mathcal{D}_k = \mathcal{D}_i \bigcup \mathcal{D}_j$
      $d_k = \alpha\Gamma(n_k) + d_{left_k} d_{right_k}$
      $\pi_k = \frac{\alpha\Gamma(n_k)}{d_k}$
      $r_k = \frac{\pi_k P(\mathcal{D}_k|\mathcal{H}_1^k)}{P(\mathcal{D}_k|\mathcal{T}_k)}$
    Find $k = \arg\max_k r_k$
    Merge $\mathcal{D}_k \leftarrow \mathcal{D}_i \bigcup \mathcal{D}_j$
    $\mathcal{T}_k \leftarrow (\mathcal{T}_i, \mathcal{T}_j)$
    Delete $\mathcal{D}_i, \mathcal{D}_j$
    $c \leftarrow c - 1$
  return top node, which contains the whole hierarchy.

III. Description of Implementation

We first implement the BHC algorithm in plain, naive python code, for ease of later comparison, we call it 'bhc plain'. (see implementation in ./bhclust/Bayesian_hclust.py). We provide Normal-Inverse Wishart distribution and Binomial-Beta distribution for calculating likelihood $P(\mathcal{D}_k|\mathcal{H}_1^k)$. We run profiling to check the bottleneck of such implementation. The task here is the simulated 2-dimensional Gaussian dataset with 3 clusters, each of which has 20 data points. The likelihood function is Normal-Inverse Wishart, and the profiling results are shown in Figure 1.

We found that the critical part of the implementation is the module to compute likelihood. Hence, we use c++ to rewrite this part and wrap it up using pybind11. We call it 'bhc c++'. The c++ code is in ./bhclust/log_marginal_prob.cpp and the corresponding python code is in ./bhclust/Bayesian_hclust_cpp.py.

As the calculation of likelihood is the most time-consuming part, we consider store the results in a hash table from previous loops to avoid repeat calculations. To be more specific, suppose at the first outer loop, all pairs of individual clusters are merged and the posterior probabilities r_k are calculated for each pair (in total $\frac{n(n+1)}{2}$ pairs). At the end of first outer loop, only one pair is chosen to merge and the rest clusters remain the same and the process will repeat at later loops. Since we have already calculated the posterior probabilities, we store them in a hash table, i.e. dictionary in python, to avoid the repeat evaluations. This optimization is implemented corresponding to the above two versions, called 'bhc hash' and 'bhc c++ hash'. (see implementations in ./bhclust/Bayesian_hclust_fast.py and ./bhclust/Bayesian_hclust_cpp_fast.py).

For the same task used for profiling mentioned above, we calculate the running time for the for implementations. We run each implementation 7 times and the results are in Table I.

TABLE I
RUNNING TIME OF FOUR BHC IMPLEMENTATIONS

Implementation	Average time (ms)	Standard deviation (ms)
bhc plain	6980	642
bhc c++	1500	422
bhc hash	1010	264
bhc c++ hash	157	11.3

By using hash table to store previously calculated

```
*** Profile stats marshalled to file 'bhc.prof'.
Fri Apr 26 15:28:21 2019    bhc.prof

         4611931 function calls in 6.069 seconds

   Ordered by: internal time, cumulative time
   List reduced from 85 to 5 due to restriction <5>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    72040    2.229    0.000    5.062    0.000 C:\sta-663\sta-663-2019\projects\Bayesian_hclust_fast.py:137(log_marginal_probabi
lity)
   144080    0.713    0.000    1.734    0.000 C:\Users\cheng\Anaconda3\lib\site-packages\numpy\linalg\linalg.py:1737(slogdet)
    35990    0.547    0.000    3.437    0.000 C:\sta-663\sta-663-2019\projects\Bayesian_hclust.py:259(merge)
   216297    0.464    0.000    0.464    0.000 {method 'reduce' of 'numpy.ufunc' objects}
   216179    0.305    0.000    0.862    0.000 C:\Users\cheng\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:1778(sum)
```

Fig. 1. Bottleneck check of naive BHC implementation

nodes results can most significantly reduce the running time cost with some sacrifice in memory. Also, using c++ to rewrite critical function to calculate likelihood can also reduce running time.

The other issue encountered during implementation and testing is that the algorithm collapses when the number of samples in the datasets exceed 60 given that the probability model is Gaussian. The reason we suppose is that like multivariate Gaussian distribution, the marginal probability of a node involves multiplication of individual probabilities, leading to very small (e.g. 10 to negative 300) floating numbers. Although we perform all the calculations in natural log scale, the small probabilities have no comparative meanings, which can still make the model collapse. The workarounds we think include choosing other proper probability models, but it turns out to be quite challenging and impractical in real world as it is unlikely that a certain kind of data come from a specific probability model. If we choose binary data and use Binomial-Beta distribution as the probability model, this issue disappears.

IV. Applications

A. Simulated Data

First, we show the results of the simulated dataset used for code performance comparison above. We simulate 3 cluster of 60 samples from 2-dimensional multivariate normal distribution with different means and equal variance. We fit the BHC on this data, and draw the dendrogram by the centroid linkage between clusters in Figure 2 and clustering results in Figure 3.

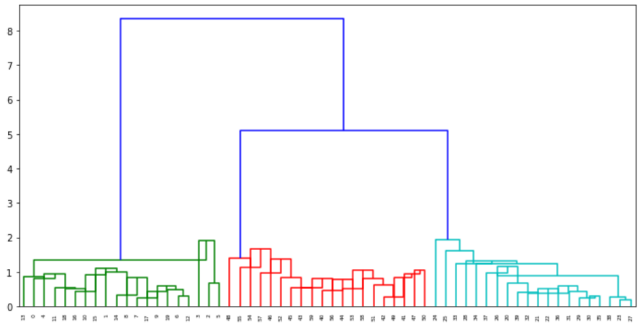

Fig. 2. Dendrogram of BHC on simulated data with 3 clusters

Then, we simulate 5 cluster of 60 samples from 2-dimensional multivariate normal distribution with different means and equal variance. The dendrogram and clustering results are in Figure 4 and Figure 5. For both datasets, BHC performs perfectly. Hence, BHC performs quite well in such small dataset.

B. Real World Data

In the original paper, the author uses the glass dataset from UCI repository[3]. I downloaded the same dataset from UCI repository. The dataset contains 9 attributes, each of which represents a kind of elements, which are all continuous variables. The label is the type of glass. There are 7 types of glass in total. I choose the first two types and sample 10 from each type. The dendrogram is in Figure 6.

We define the goodness of clustering by choosing the cuts so that there are exactly two clusters

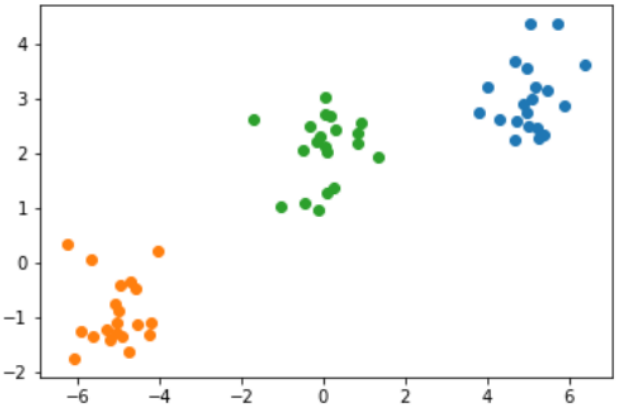

Fig. 3. Clustering results of BHC on simulated data with 3 clusters

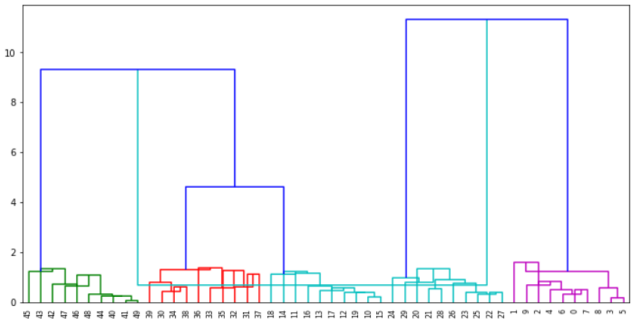

Fig. 4. Dendrogram of BHC on simulated data with 5 clusters

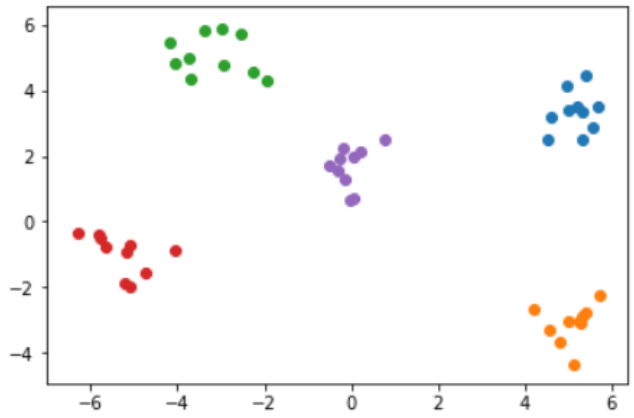

Fig. 5. Clustering results of BHC on simulated data with 5 clusters

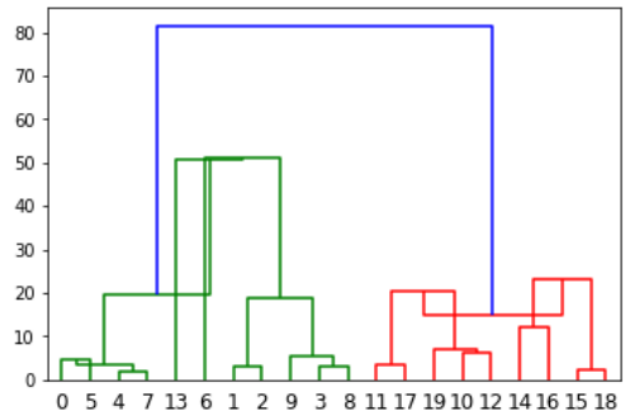

Fig. 6. Dendrogram of BHC on glass data

remaining, and count the number of points correctly clustered according to the known labels of data. The lower bound of the goodness is 0.5. The goodness of this example is 0.95.

We also downloaded the spam email data from UCI repository[4]. The dataset is high-dimensional, containing 57 features and known labels. We sample 50 from each class, (spam and non-spam), binary each feature (zero and non-zero) and run BHC using Binomial-Beta likelihood. The dendrogram is shown in Figure 7) and the goodness is 0.79.

V. Comparative Analysis

At this section, we fit agglomerative clustering, which is the traditional version of hierarchical clustering on the four datasets in the previous section. We use the most populat 'ward' linkage, which chooses the proper merge to minimize the variance of clusters. The implementations have already been in both scipy and sklearn. The four dendrograms are shown in Figure 8, 9, 10, 11. And the goodness of BHC and agglomerative clustering on the four datasets are summarized in Table II. We observe that BHC performs the same with agglomerative clustering with 'ward' linkage as long as BHC can work properly. However, as BHC need to evaluate probability, the running time is much slower than agglomerative clustering when the number of data becomes larger.

VI. Discussion

Bayesian Hierarchical Clustering is a probabilistic approach to do bottom-up hierarchical clustering. It does not need to specify a proper distance metric, but instead, choosing a suitable probability

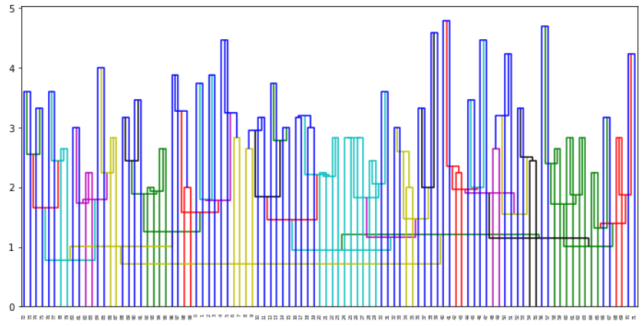

Fig. 7. Dendrogram of BHC on spam email data

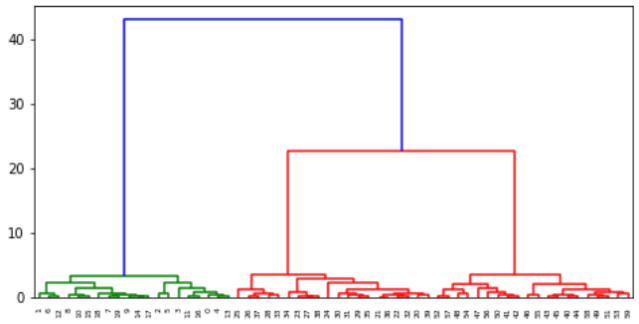

Fig. 8. Dendrogram of agglomerative clustering on simulated data

model becomes even more critical. If the probability model is not proper, like the multivariate Gaussian, the marginal probability of the node when it contains more data will become extremely small, and comparing in such ranges is meaningless. Because the original paper said very few about how to choose the proper probability model, we find it hard to reproduce the excellent results in every case for BHC. At least, we find that for binary data, BHC can perform as good as tradition methods.

REFERENCES

[1] Heller, K. A., & Ghahramani, Z. (2005, August). Bayesian hierarchical clustering. In Proceedings of the 22nd international conference on Machine learning (pp. 297-304). ACM.

[2] Barlow, H. B. (1989). Unsupervised learning. Neural computation, 1(3), 295-311.

[3] https://archive.ics.uci.edu/ml/datasets/glass+identification

[4] https://archive.ics.uci.edu/ml/datasets/glass+identification

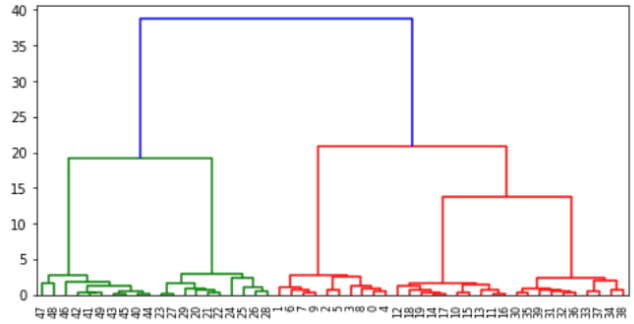

Fig. 9. Dendrogram of agglomerative clustering on simulated data

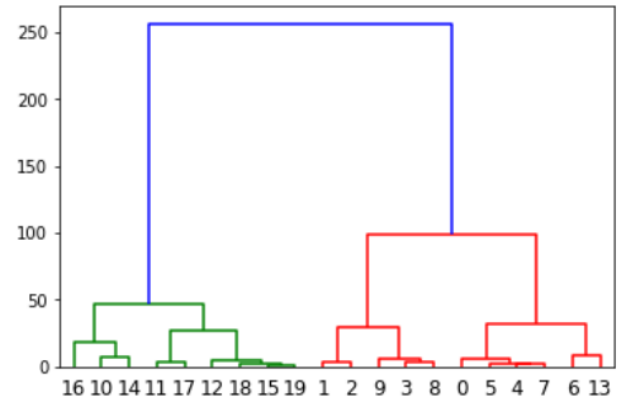

Fig. 10. Dendrogram of agglomerative clustering on glass data

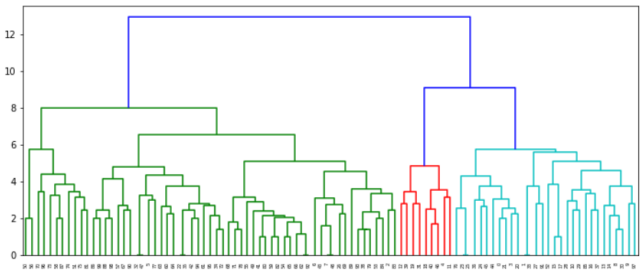

Fig. 11. Dendrogram of agglomerative clustering on spam email data

TABLE II

GOODNESS OF BHC AND AGGLOMERATIVE CLUSTERING

Dataset	BHC	Agglomerative Clustering
Simulated with 3 clusters	1	1
Simulated with 5 clusters	1	1
Glass data	0.95	0.95
Spam email data	0.79	0.79