Implementation of Bayesian Hierachical Clustering

Lina Yang, Xiaodi Qin

May 1, 2017

1 Abstract

Hierarchical clustering is commonly used in unsupervised learning. There are several limitations to the traditional hierarchical clustering, including no guide to determine the appropriate number of clusters to prune the tree, difficult to choose suitable distance metric, and lack of probability based criterion for model evaluation. To overcome these limitations, Heller and Ghahramani (2005) developed a Bayesian hierarchical clustering algorithm. This algorithm fits a probabilistic model to the data and uses marginal likelihoods as criteria for choosing nodes to merge. Purity score was used to evaluate the performance of the algorithm and to compare with the traditional hierarchical clustering approaches.

Key words: Bayesian Methods, hierarchical clustering, unsupervised learning, marginal likelihood, purity score

2 Background

Agglomerative hierarchical clustering is a bottom up hierarchical clustering method. It assigns each data point to its own cluster and iteratively merges the two clusters that have the least distance until all the clusters are merged to a single cluster. This hierarchical clustering aims to generate a hierarchical structure that consistent with the organization of the real data. There are several disadvantages of traditional hierarchical clustering. 1. It provides no guide to determine the appropriate number of clusters to prune the tree. 2. It is difficult to choose suitable distance metric. 3. It does not incorporate the probabilistic distribution of the data, so it is hard to make comparison between other clustering approaches.

The Bayesian hierarchical clustering (BHC) algorithm fits a probabilistic model to the data and uses marginal likelihoods to choose which clusters to merge. It calculates the probability that the merged two clusters are from the same mixture class, and compares it with all the other potential merge combinations.

This BHC algorithm can be applied to various kinds of data. For example, we can apply it to classify patients who have a specific type of cancer into subgroups based on their genetic expression data. In this paper, they applied it to multivariate Gaussian, Bernoulli and multinomial data.

3 Algorithm

This algorithm calculates the probability of merging for each two subtrees, and build the whole tree bottom-up, and finally returns a matrix in the structure of input matrix for `dendrogram` function in `Scipy` module. It can be used to cluster data with multivariate normal distribution or Bernoulli distribution, and is possible to be extended to other types of data.

For each pair of clusters, the hypotheses we need to test are H_1 vs. H_2, representing either merge or stay unmerged. And the probability of being merged will be calculated to make the decision. This probability, denoted by r, is given by Bayes' rule,

$$r = \frac{\pi \mathrm{P}(D \mid H_1)}{\mathrm{P}(D \mid T)}$$

where $\mathrm{P}(D \mid H_1)$ is the probability of the data under H_1, and it can be computed by distribution function of the data and corresponding conjugate prior. π is the prior that all data in the two clusters belong to one cluster, and $\mathrm{P}(D \mid T)$ is a weighed sum of the probability of the data under each hypothesis, which can be expressed as

$$\mathrm{P}(D_k \mid T_k) = \pi_k \mathrm{P}(D_k \mid H_1^k) + (1 - \pi_k)\mathrm{P}(D_i \mid T_i)\mathrm{P}(D_j \mid T_j)$$

Based on the equation above, $\mathrm{P}(D_k \mid T_k)$ has to be calculated recursively, since it uses probabilities of data under its subtrees to determine the probability of data in current tree. The cutoff for not merging is 0.5 because of only two hypotheses considered.

The algorithm basicly consists of three parts:

1. Line 1 - 4 is the initialization part. $\mathrm{P}(D \mid T)$'s are calculated given distritbution of data assigned to all the leaf nodes, $\mathbb{S}$ is a set of indices containing all the data clusters, and every time two clusters are merged togther, their indices would be removed from $\mathbb{S}$; every time a new cluster is built up, the newly assigned index associated with it would be added to $\mathbb{S}$. Family is a function defined elsewhere with two options, "niw" for multivariate normal data and "bb" for Bernouli.

2. Line 5 - 12 describes the process of obtaining the probabilities of merging for pairwise leaf nodes using the equation above, and each of the potential clusters have an index stored in $\mathbb{P}$. Similar to $\mathbb{S}$, $\mathbb{P}$ keeps adding/deleting indices to maintain a set of combinations of all current available clusters during building the whole tree.

3. Line 13 - 28 repeats the process of computing $\mathrm{P}(D \mid T)$ for every two possible clusters, until $\mathbb{S}$ runs out of indices. In each run, the combination of the highest merging probability would be append to a list Z, which is a 4dimensional matrix, within each row first two numbers being the indices of two sub-clusters that have been merged, third number being weight evaluated as reciprocal of log odds, and the last being the number of nodes in the cluster.

The code was checked by the clustering results against the ones in Heller and Ghahramani (2005), using similar data. Unfortunately so far there is no other package implementing the same algorithm to compare with, the only one found on Bioconductor is for multinomial and time-series data, which was not included in our implementation.

The performance of this method comparing with other clustering methods was evaluated using a *purity score*, which will be elaborated in Section 6.

4 Optimization

This algorithm roughly involves twice iterations through all leaf nodes, therefore the overall computational complexity is $O(n^2)$, and the inefficiency can be identified by Cython annotation in the profiling results(`/bhc/tests/bottleneck_check.ipynb`). Since the body of the code is wrapped in a long loop and contains functions from third-party modules, it is hard to optimize the code using `numba` or `pyspark`. And since the algorithm calculates one point at each step, vectorization can hardly be incorporated, and python lists were used instead of `numpy` in case that overhead could not pay off. Some small calculation results were also cached to avoid repeated calculations. However, the code can be partly optimized. For multivariate normal data, the marginal likelihood involves lots of matrix multiplications, therefore, we rewrote all the relevant functions into C++ code (`/bhc/bhc/helper.cpp`), wrapped it using `pybind11`, and made a new function called `bhclust_fast`. The speed is more than three times as fast as for the original code for small data set (Toy data set, 18 obs). For larger sets of multivariate data, we have experienced difficulties in timing the original code, since the docker usually stopped computation before it was done. However, the C++ accelerated code could give a system runtime of 1.02s and CPU time of 53.8s for a 788×2 data set. More comparisons of speed can be found in `/bhc/tests/comparison_speed.ipynb`.

5 Application

Hierarchical clustering is widely used since real-world data often show a hierarchical pattern by nature. We used data in different area to test the performance of method. Here we showed

Algorithm 1: Bayesian Hierachical Clustering

```
Input: Data $X = (X_0, X_1, ..., X_N)$, $family \in \{niw\}$, hyperparameter $\alpha$, scaling factor on
       the prior precision of the mean $r$.
Output: A linkage matrix $Z$
Set $\mathbb{S} = \emptyset$
For $l \in \{0, 1, \ldots, N-1\}$:
    $n_l^0 = 1,\ d_l^0 = \alpha,\ X_l^0 = X_l,\ ml_l = \textsc{family}(X_l^0)$
    $\mathbb{S} = \mathbb{S} \cup \{l\}$
Set $t = 0,\ \mathbb{P} = \emptyset$
For $i \in \{0, 1, \ldots, N-2\}$:
    For $j \in \{i+1, \ldots, N-1\}$:
        $c_{1,t} = i,\ c_{2,t} = j,\ n_t = n_i^0 + n_j^0$
        $X_t = (X_i^0, X_j^0)^T,\ d_t = \alpha\Gamma(n_t) + d_i^0 d_j^0$
        $P_{1,t} = \textsc{family}(X_t)\alpha\Gamma(n_t)/d_t,\ P_{2,t} = ml_i ml_j (d_i^0 d_j^0 / d_t)$
        $logodds_t = \log P_{1,t} - \log P_{2,t}$
        $\mathbb{P} = \mathbb{P} \cup \{t\},\ t = t + 1$
Set $p = 0,\ Z = []$
While $1$:
    $idx = \arg\max_{idx \in \mathbb{P}} logodds$
    $Z.\textsc{append}([c_{1,idx}, c_{2,idx}, logodds_{idx}, n_{idx}])$
    $n_{N+p}^0 = n_{idx},\ X_{N+p}^0 = X_{idx},\ d_{N+p}^0 = d_{idx},\ ml_{N+p} = P_{1,idx} + P_{2,idx}$
    $rm = \{c_{1,idx}, c_{2,idx}\},\ \mathbb{S} = \mathbb{S} \setminus rm$
    If $\mathbb{S} = \emptyset$:
        break
    For $q \in \mathbb{S}$:
        $c_{1,t} = N + p,\ c_{2,t} = q,\ n_t = n_{N+p}^0 + n_q^0$
        $X_t = (X_{N+p}^0, X_q^0)^T,\ d_t = \alpha\Gamma(n_t) + d_{N+p}^0 d_q^0$
        $P_{1,t} = \textsc{family}(X_t)\alpha\Gamma(n_t)/d_t,\ P_{2,t} = ml_{N+p} ml_q (d_{N+p}^0 d_q^0 / d_t)$
        $logodds_t = \log P_{1,t} - \log P_{2,t}$
        $\mathbb{P} = \mathbb{P} \cup \{t\},\ t = t + 1$
    $\mathbb{P} = \mathbb{P} \setminus \{r : c_{1,r} \in rm \vee c_{2,r} \in rm\}$
    $\mathbb{S} = \mathbb{S} \cup \{N + p\},\ p = p + 1$
return $Z$
```

Bayesian hierarchical clustering results against hierarchical clustering with average linkage for 1 synthetic data set and 2 real data sets.

5.1 Simulated data sets

This is a data simulated by Gionis et al. (2007), with dimension 788×2. The scatter plot (Figure **??**) displays that the seven clusters in the data have clear boundaries between each other, suggesting this might be a easy classification question. The results for the two types of clustering are shown in Figure **??**. The different colors in the texts on x-axis are the true labels for all the classes in this data, which are numbers 0 - 7.

The Bayesian hierarchical clustering shows a feature similar to single linkage, but with more condensed distance.

Figure 1: Scatter plot of the original aggregation data

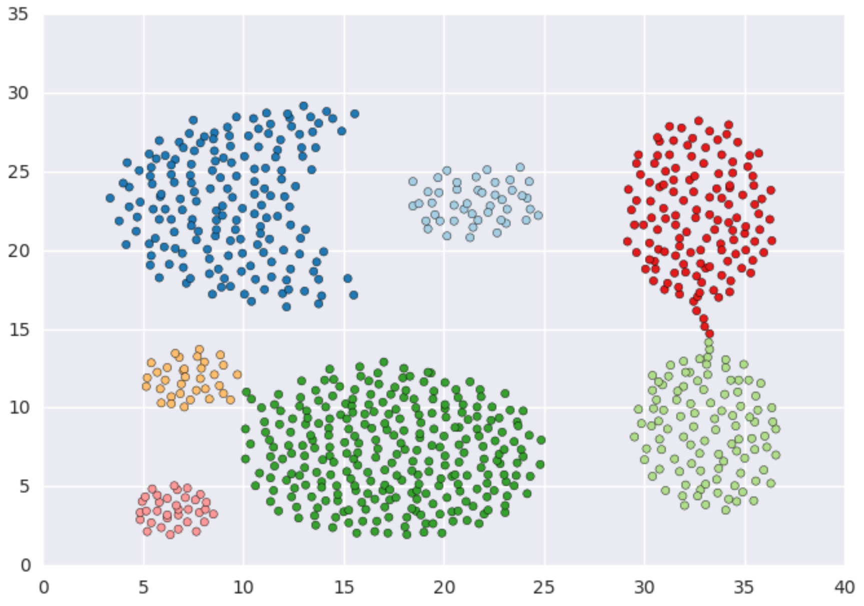
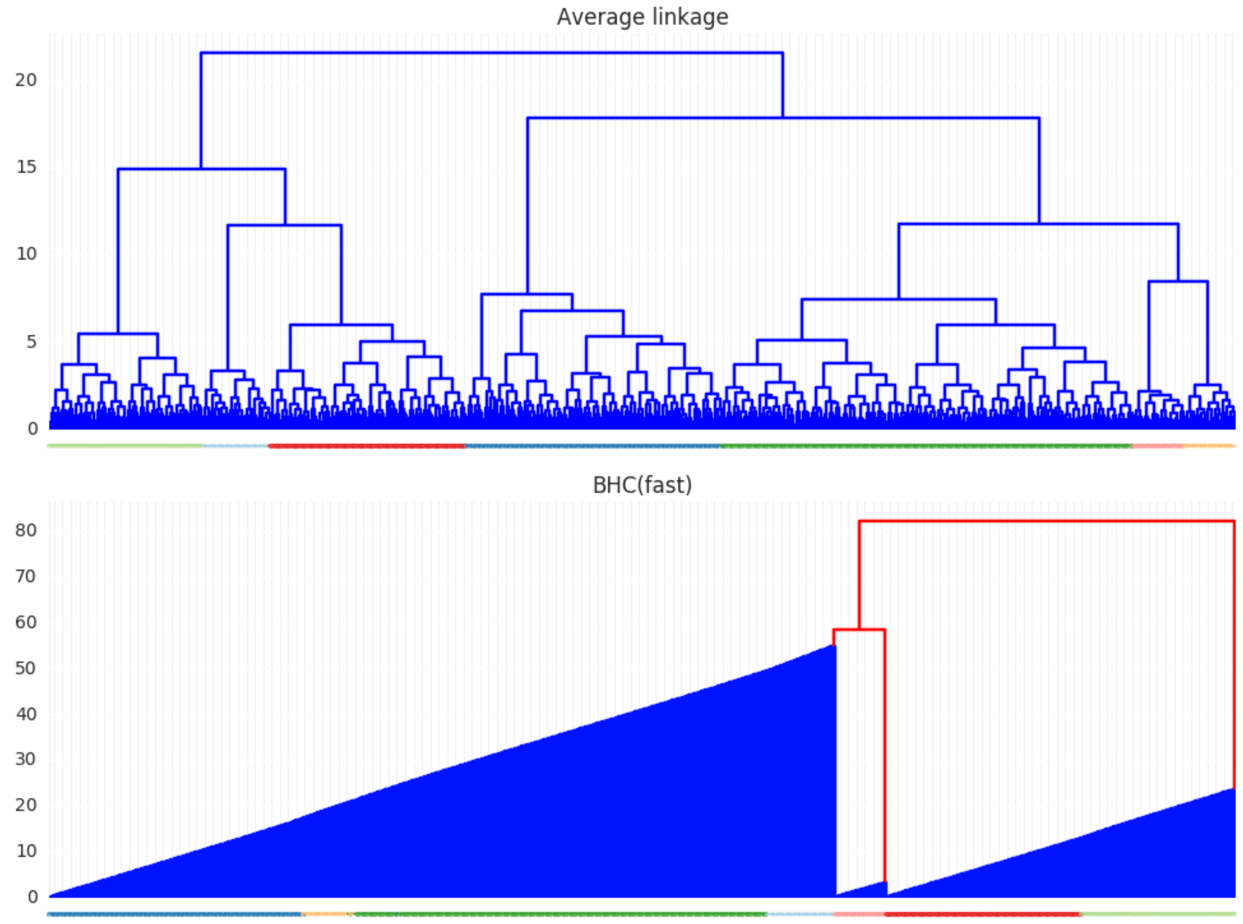


Figure 2: Dendrograms for hierarchical clustering of average link and Bayesian method



5.2 Real data sets

Figure 3: Heatmap of the original seeds data

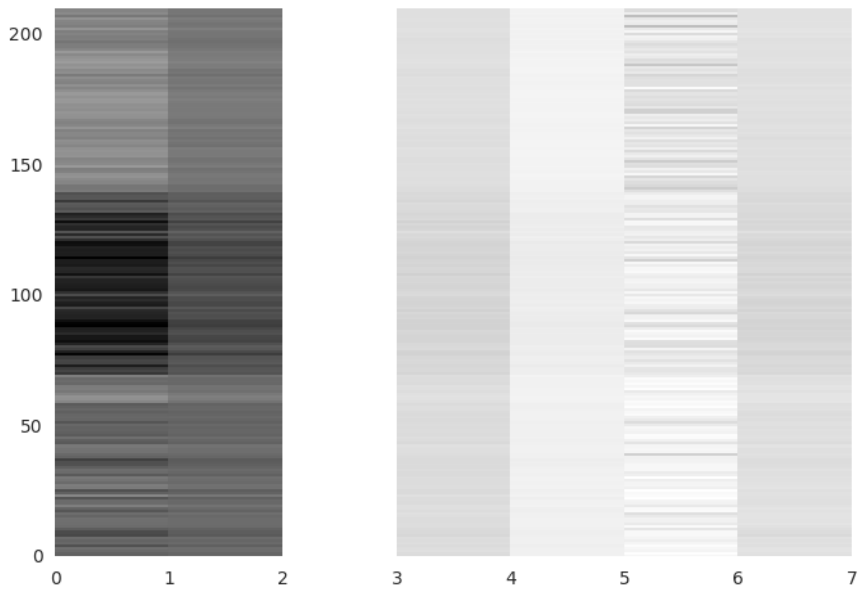


Figure 4: Dendrograms for hierarchical clustering of average link and Bayesian method

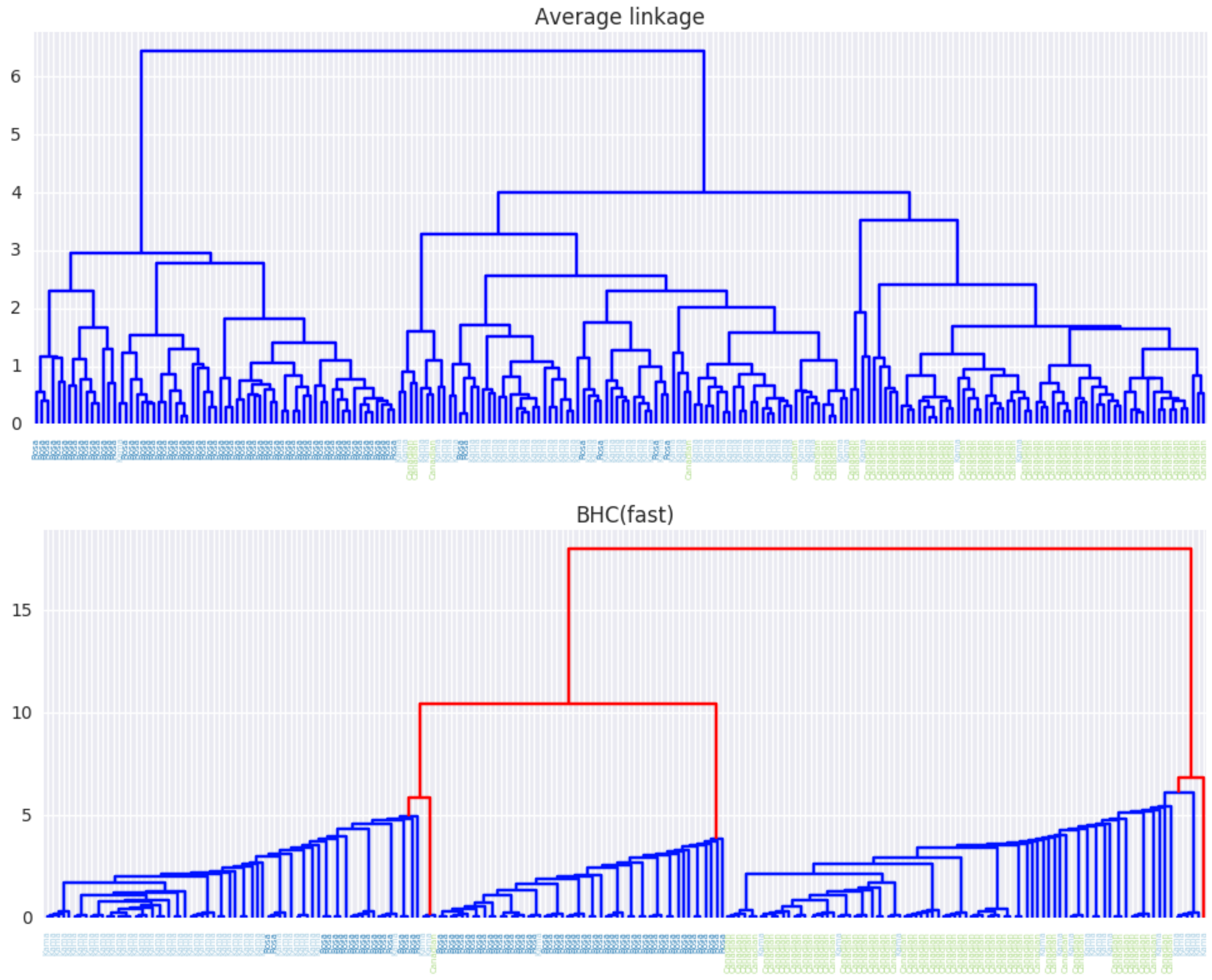


Figure 5: Heatmap of the original CEDAR data

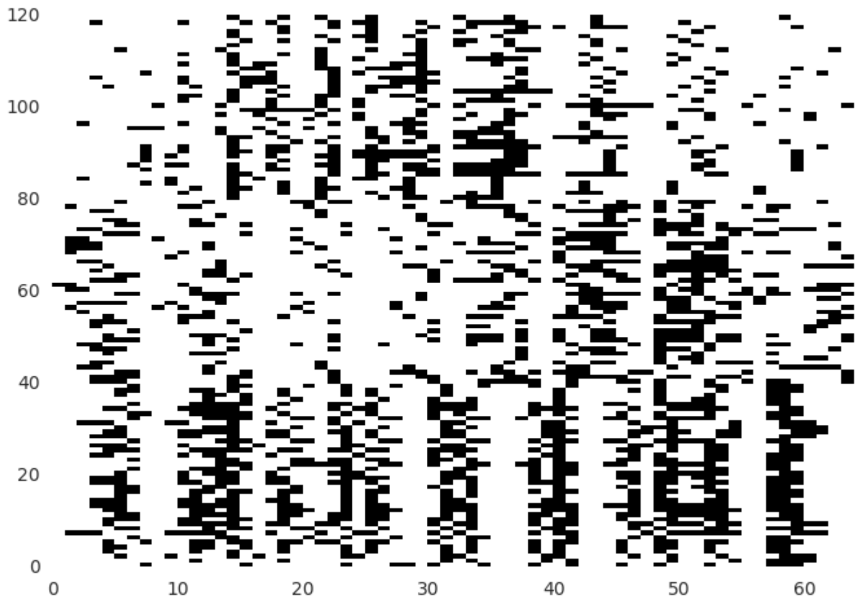
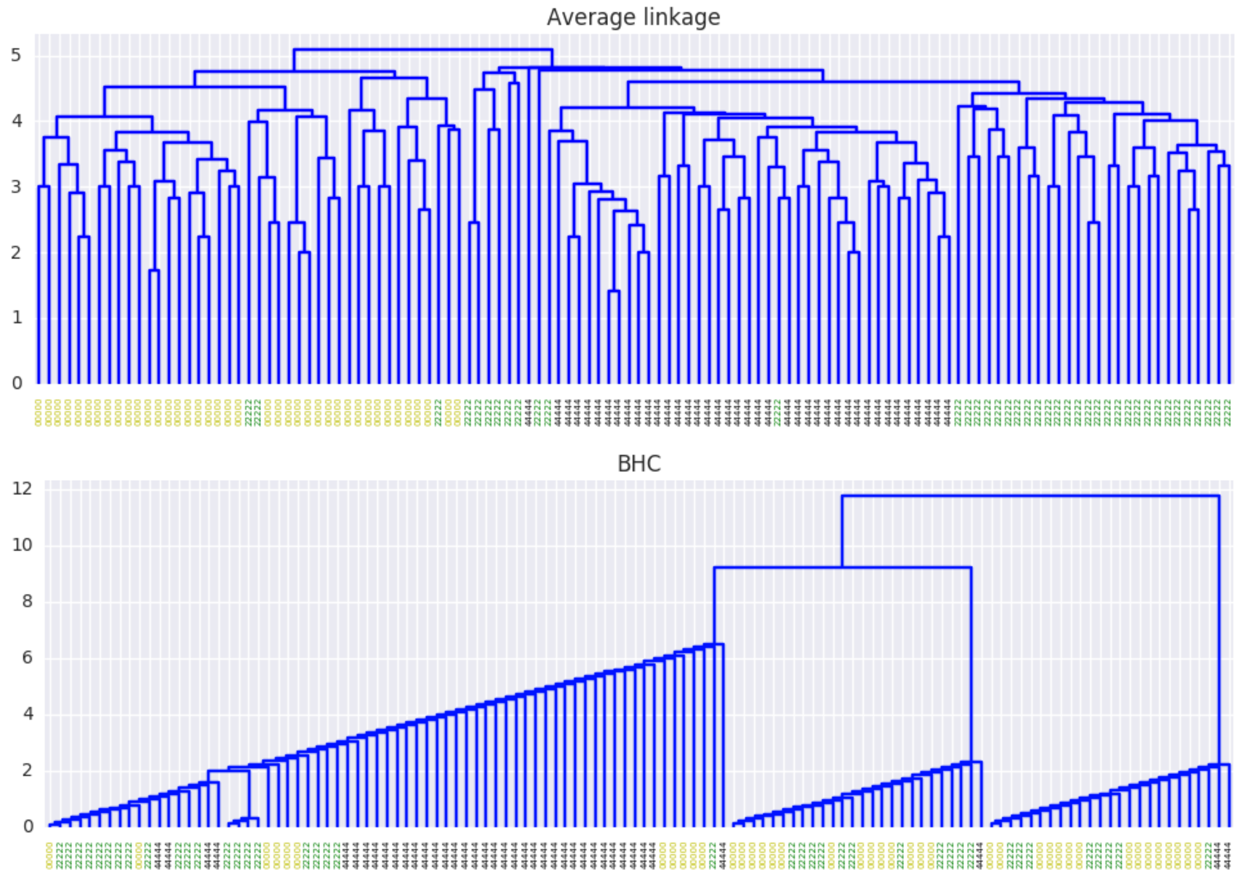


Figure 6: Dendrograms for hierarchical clustering of average link and Bayesian method



seeds Charytanowicz et al. (2010) aggregationGionis et al. (2007)

6 Comparative analysis

We wrote a function to calculate the purity score which is the evaluation method used in Heller's BHC paper. The purity score is calculated as follows:

Pick two leaves i, j uniformly at random with restriction that i, j are from same class, and then find the smallest subtree containing i and j. Find all leaves that locate in this subtree and calculate the proportion of leaves that are from the same class as i (j) in the subtree. This proportion is the purity score we need. We compared this purity score calculated from BHC to 3 kinds of traditional agglomerative clustering using average, single, and complete linkage. We tested their performance on synthetic data generated by us s well as synthetic dataset obtained from other publications. We also obtained some real data from CEDAR dataset for testing.

We find that the BHC algorithm outperforms traditional agglomerative clustering approaches when clustering binary data (Table). BHC algorithm performs better than others when it was applied to our synthetic Gaussian multivariate data. BHC does not perform as good as others when applied to "aggregation" data, but its purity score is still reasonable high (0.788). But for the "spiral" data, only clustering method using single-linkage has a high purity score (1.0), BHC and other two clustering methods all have low purity score (0.3).

	SINGLE	COMPLETE	AVERAGE	BHC
SYNTHETIC_multivariate	0.689	0.6	0.689	1.0
Aggregation	0.85	1.0	1.0	0.788
Spiral	1.0	0.332	0.334	0.35
SYNTHETIC_binary	0.641	0.561	0.608	0.681
CEDAR	0.572	0.73	0.863	0.978

Table 1: Comparison results of four methods on six data sets

7 Discussion

8 Code

The repository can be found at `https://github.com/qxxxd/bhc`.

9 References

Charytanowicz, M., J. Niewczas, P. Kulczycki, P. A. Kowalski, S. ukasik, and S. ak (2010). Complete Gradient Clustering Algorithm for Features Analysis of X-Ray Images. In *Information Technologies in Biomedicine*, pp. 15–24. Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-642-13105-9_2.

Gionis, A., H. Mannila, and P. Tsaparas (2007, March). Clustering Aggregation. *ACM Trans. Knowl. Discov. Data 1*(1).

Heller, K. A. and Z. Ghahramani (2005). Bayesian Hierarchical Clustering. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, New York, NY, USA, pp. 297–304. ACM.