STA 663 Final Project: Bayesian Hierarchical Clustering

Sam Voisin & Jonathan Klus

April 30, 2019

Abstract

Bayesian Hierarchical Clustering (BHC) is an agglomerative tree-based method for identifying underlying population structures ("clusters"). BHC was introduced by K. Heller and Z. Ghahramani as a way to approximate the more computationally intensive Dirichlet process mixture model with Gaussian components introduced by C. Rassmusen. The advantage of these models over their counterparts lies in the fact that an *ex ante* number of clusters does not need to be specified. Instead, the Bayesian approach allows for regularized flexibility via a prior placed on the cluster concentration parameter α. We utilized the object-oriented programing (OOP) capabilities of Python to build a module that can be used similar manner to algorithms included in the popular scikit-learn library. As with many Bayesian methods, the increased flexibility of BHC comes at a computational cost and the increased risk of poor results due to misspecified priors and likelihoods for a given problem. Throughout this paper we will describe methods we used to mitigate these issues where possible.

Background

The paper we chose to analyze and reproduce is *Bayesian Hierarchical Clustering* by Katherine Heller and Zoubin Ghahramani (2005).

Cluster analysis is a frequent unsupervised learning problem encountered in statistics/ probability and machine learning disciplines. It involves assigning observed data to discrete groups based on shared or similar attributes. Hierarchical clustering is a particular subset of methods of clustering that take into account the *degree* to which data points are similar. Data points that are more similar are clustered first. Then clusters that are less similar are grouped until only one all-encompassing cluster remains. This is also known as *agglomerative clustering*.

The frequentist hierarchical clustering method requires the user to specify the number of clusters prior to the actual clustering process. This is potentially problematic because the true number of clusters that exist within the data distribution is not usually known. This problem is addressed by BHC through the use of a cluster concentration parameter that regularizes, rather than restricts, the number of possible data clusters. It also typically uses a distance metric (e.g. Euclidean distance) as a decision rule, while the Bayesian hierarchical clustering method uses marginal likelihoods based on a prior and likelihood specified by the user. This provides a more flexible framework.

There are two key limitations to the BHC algorithm according to K. Heller. The first limitation is the computational cost which is $O(n^2)$ using Big-O notation. This is caused by the algorithm's need to compare every possible pair of clusters during each iteration. The second limitation of BHC is the lack of a tree uncertainty metric like a probabilistic interpretation of arriving at one particular tree structure versus another.

The Algorithm

The core of the BHC algorithm relies on a Bayesian hypothesis test in which two alternatives are compared:

1) H_1 is the hypothesis that two clusters D_i and D_j were generated from the same distribution $p(x|\theta)$ with the prior distribution for θ being $p(\theta|\beta)$. The probability of clusters i and j being generated from the same distribution is defined as $p(D_k|H_1)$. The posterior for this hypothesis is:

$$r_k = \frac{\pi_k p(D_k|H_1)}{\pi_k p(D_k|H_1) + (1-\pi_k)p(D_i|T_i)p(D_j|T_j)}$$

Note that $\pi_k = p(H_1)$ is the prior probability of a merge occuring for clusters i and j. This makes the denominator of this expression the Bayesian *evidence*.

2) H_2 is the hypothesis that the two clusters D_i and D_j were generated from two independent distributions and therefore should *not* be joined together as cluster D_k. The probability of H_2 is calculated as $p(D_k|H_2) = p(D_i|T_i)p(D_j|T_j)$ where T_i and T_j are the subclusters being examined.

All existing clusters are compared and joined based on the combination of clusters with the highest posterior merge probability r_k. In our module this iterative action of comparing and merging clusters occurs in the `HierarchyTree` object. `HierarchyTree` is initialized by passing in the data matrix `X`, the parameters of the prior distributions, and the family of distributions. Currently, support for the Multivariate Normal-Inverse Wishart family exists with support for the Beta-Binomial and Multinomial-Dirichlet planned as the next steps in development of this module. The `HierarchyTree` is "grown" using the `grow_tree` method. This method is analogous to the `fit` method for a class in `sklearn`.

The BHC algorithm is initialized by setting all data points within their own singleton cluster represented by a "leaf" in what will become the clustered hierarchy tree. The algorithm iterates over all possible n choose 2 combinations of these one-point clusters and calculates the posterior possibility of each combination. The combination of clusters with the highest posterior probability are merged to form a new cluster in a "greedy" fashion. This concludes one iteration of the algorithm.

The algorithm repeats the steps described above for merging clusters until only one super-cluster of all the data points remains. Due to the greedy nature of the algorithm, this tree will have merged clusters with low posterior merge probabilities. These merges are considered *unjustified* and usually occur at the top of the tree. In the final step of the algorithm, unjustified merges seperated if their posterior merge probability is less than a user defined threshold.

To summarize the above in step-by-step form:

1) Vectors $\{x_1, ..., x_n\}$ are data points generated with distribution $p(x|\theta)$ with prior $p(\theta|\beta)$. These vectors are initialized into seperate clusters $\{1, ..., n\}$.

2) All possible combinations of the clusters are compared to find the highest posterior merge probability r_k. This combination is joined into a new cluster in a higher *tier* within the tree.

3) The number of clusters is then reduced by 1, and the process repeats itself from step 2 while the number of clusters is greater than 1.

4) The tree is cut at points where the posterior probability of merging is less than some specified threshold. Typically the threshold value is chosen to be $r_k < 0.5$.

Optimization for Performance

After writing the code for the complete algorithm including the `HierarchyTree`, `Split`, and `Leaf` classes described in the *Background* section, we began looking for areas in which we could improve the speed of the BHC algorithm. It is important to note that, because of the algorithm requirement that *all* clusters be tested for merging with all other clusters, the computational burden of the algorithm grows extremely quickly with large n datasets and datasets with high dimensions. As described above, Bayesian hierarchical clustering is quadratic in the number of datapoints in terms of complexity.

One challenge of the BHC algorithm exists in the recursive calculation of $p(D_k|H_2)$. As the size of the dataset increases, the number of recursions must also increase as we continue to "grow" the `HierachyTree`. We quickly reach a point at which continued recursion becomes infeasible. To combat this problem we developed two additional classes for storing various information about a given cluster. An instance of the `Leaf` object is created for each data pont when the `HierarchyTree` is initialized. Once the `grow_tree` method is invoked instances of the `Split` object are created as clusters merge. The `Split` instances are designed to save the probability of a posterior merge. In this manner a true recursive calculation is unnecessary and the computing cost of the algorithm is greatly reduced.

We employed the `cProfile` package in order to find the rate limiting steps in the `grow_tree` method. Profiling our code showed us that, while our functions were themselves not very resource intensive, the algorithm requires that they be called many times during each iteration. Most notably the function for calculating the marginal likelihood of a cluster (i.e. the Bayesian evidence) `marginal_clust_k` was calling for several repeated calculations of the probability of merging clusters $p(D_k|H_k)$. Through some simple code restructuring we were able to reduce the number of times this calculation was made. This reduced our computational time from $12s$ to $8s$ on a toy data set of 30 points. This translates to significant time savings on much larger data sets.

Application to Simulated Data

In Heller, et. al. (2005), she describes one rationale for the Bayesian hierarchical clustering algorithm as being that it is a fast approximation to a Dirichlet process model, like a mixture of Gaussians. This model has a known solution as described by Rasmussen (2000), but the marginal likelihood is intractable and requires the use of adaptive importance sampling. The problem is therefore quite time complex, with Rasmussen describing that it took 18 minutes of CPU time to achieve just 100 independent samples from the posterior distribution. To demonstrate the applicability of Heller's algorithm to this problem, we generated synthetic three-dimensional data from three multivariate Gaussian distributions (i.e. there were 3 classes) and specified the weights for each distribution. The resulting data frame was then used to train the BHC algorithm, and the predicted class labels were compared to the true class labels.

Performance of the algorithm was assessed by calculating the purity of the final class labels. The purity metric sums the the maximum number of points in each predicted class that were from the same original class, then divides them by the total number of observations in the data set. This provides a measure of how well the algorithm clustered the data after pruning has been performed (i.e. is each predicted class majority from the same true class, is it "pure"), but is rather simplistic in that it can easily be fooled if each observation is assigned to its own class. In this case, the purity metric would be 1.0, a perfect score. To compensate for this shortcoming, we propose an adjusted purity score which only considers the top k predicted class labels (where k is the true number of classes in the data set). This adjusted purity score provided a more realistic assessment of the performance of the clustering algorithm, especially in cases when there are many singletons in the final class assignments.

The testing described above is fully documented in the reproducible Comparative_Analysis Jupyter Notebook, available on this project's GitHub repo. The BHC algorithm performs performs reasonably okay on the synthetic data, achieving 95% purity and 52% adjusted purity. From a visual inspection of the predicted clusters in 2D, the assignments also appear to reasonable. However, there is certainly room for improvement. The clustering appears to be particularly sensitive to the values assigned to the model hyperparameters, especially that of the Dirichlet hyperparameter α.

In addition to the Gaussian likelihood, Heller's paper also suggested that clustering could also be computed using Bernoulli-Beta or Multinomial-Dirichlet conjugate families. The former is implemented in our version of Heller's BHC algorithm, and the package is structured such that the latter conjugate family, or other likelihoods, can be easily added and are compatible with the other helper functions. We also tested the BHC algorithm on a synthetic Bernoulli data set that was constructed similarly to the Gaussian described above (except the Bernoulli likelihood was used instead of the multivariate Gaussian). The BHC algorithm appeared to have difficulty clustering the data, which has 3 dimensions and 3 classes (i.e. there should be 3 clusters), and achieved only 37% purity.

Application to Real Data

The algorithm was also tested on several real data sets, including the canonical Iris data set for clustering data, which consists of 4 dimensions (variables) and 3 class labels. Also tested was the wine data set, which consists of 13 variables that describe the attributes of a particular type of wine and 3 class labels corresponding to the region in which they were grown. Both data sets are available for download through SciKit-Learn in Python through sklearn.datasets.

	Gaussian Purity	Gaussian Adj	Iris Purity	Iris Adj	Wine Purity	Wine Adj	Bern Purity	Bern Adj
BHC	0.95	0.52	0.53	0.35	0.42	0.19	0.55	0.55
AHC	0.90	0.68	0.76	0.59	0.80	0.60	0.55	0.55
K-means	0.90	0.66	0.78	0.54	0.84	0.64	0.55	0.55

Figure 1: Table 1: Performance of two benchmark algorithms versus BHC on purity and adjusted purity

The BHC algorithm performance was less than stellar on the Iris data set, with 53% purity and 35% adjusted purity.

Performance on the wine data set was worse, with 42% purity and only 19% adjusted purity. This data set was handled a bit differently, as PCA was used to reduce the dimensionality of the problem from 13 to 3, and a subsample of the original data set was used as there was some numerical instability noticed as the number of training observations approached 200. These first three principal components accounted for approximately 67% of the variance, however the results were still subpar. This result may call into question whether the assumption of a normal likelihood was a good one, particularly given the dismal results for the wine data set.

Comparative Analysis

In addition to the four data sets that were tested on the BHC algorithm, performance on each data set was also compared to built-in SciKit-Learn clustering algoriths. Both the Hierarchical Agglomerative Clustering package (the non-Bayesian version of BHC) and the k-Means package were implemented, using a Euclidean distance metric as their decision rule.

To compare the run time of each algorithm, all were run on the synthetic Gaussian data that was previously described, with 3 runs of 3 loops each. The BHC algorithm was predictably the slowest, given its computationally intensive nature.

In addition to comparing the run time, the performance of all three algorithms was compared using the purity and adjusted purity metrics for all four data sets that were tested as a part of this paper. It is worth noting that these algorithms all require the user to indicate the resulting number of clusters. This hyperparameter makes these models less flexible, but also introduces some bias when testing them on a data set for which the number of class labels is known. For this reason, we cross-validated the results using a range of class label hyperparameters from 1-9. The results are described below.

Comparison to Non-Bayesian Hierarchical Clustering

SciKit-Learn's hierarchical clustering algorithm was much faster than BHC, and performed well for small numbers of clusters, or when the correct number of class labels was specified as the hyperparameter. But as the hyperparameter was increased, performance, as measured by cluster purity, declined steeply. This is somewhat analogous to the process by which BHC arrives at a predicted set of cluster labels, since the expectations are vaguely defined through the prior, but a specific number of clusters is not specified.

Comparison to another algorithm (K-means?)

The k-means algorithm was also compared to BHC

Discussion and Conclusions

References

Heller, K. A., & Ghahramani, Z. (2005). Bayesian hierarchical clustering. Proceedings of the 22nd International Conference on Machine Learning - ICML 05.

Rasmussen, C. E. (2000). The Infinite Gaussian Mixture Model. Advances in Neural Information Processing Systems 12, 554-560.