Optimization of Bayesian Hierarchical Clustering

Matt Welch, Yamac Isik

April 2018

1 Abstract

We perform a Bayesian agglormerative hierarchical clustering. This algorithm uses a Normal-Inverse-Wishart prior, and we evaluate the marginal likelihoods (derivation in the appendix), incorporating prior information, to attain the highest probability under the merged hypothesis H_1. We then optimize this model using Cython, parallelization, and increasing coding efficiency. We test this model on the glass data presented in our paper, real world data on head and brain size, and simulated data from multivariate normal distributions with different centers. To evaluate performance of the model we use two measures: system time and node impurity. For the data sets with two predictors, we graphically display the clustering. Also displayed are comparisons of output given various different prior knowledge for the parameters on the inverse weishart distribution.

2 Background

For this analysis, we examine a Bayesian Hierarchical Clustering Algorithm based on a paper on this topic by Katherine A. Heller and Zoubin Ghahrmani. First we code the algorithm as described in the paper and then optimize its performance, and compare it to other methods such as a standard hierarchical agglomerative clustering algorithm.

This project addresses clustering a set of data points, into nested clusters in a hierarchical manner. At each step, two new clusters are merged, and a dendrogram can be constructed in this way. The purpose is to create a clustering that matches the structure of the real world. This has been traditionally performed by starting with the entire data set and each of the data points in its own cluster. The algorithm then subsequently merges clusters together based on some distance metric, for example the euclidean distance of the closest points in the cluster.

Possible applications range from biology to medicine to Business marketing and other applications. For example, in a biology setting, clustering can group organisms into different disjoint but similar environments. Clustering can even be used in evolutionary biology to group the genome into different families of

genes. In medicine, clustering has important applications for grouping different antibiotics into different groups.

In a business setting, clustering is advantageous for grouping customers based on buying habits. For insurance companies, clustering can group various geographic locations throughout the country by risk level. This is but a small overview of clustering: it is also useful for fields ranging form climatology with weather patterns to robots, were outliers in sensor data can be grouped together.

It is important to note that there is not one method of clustering that is most optimal for all data sets. The advantages of looking at this problem from a Bayesian framework is to avoid over-fitting, while providing a Bayesian framework given the prior information. If prior information is known, say regarding means for the variables, and how certain we are, then this can be incorporated into the clustering.

Advantages of traditional methods, are that they are easily implementable from packages in Python. Also, it is easy to construct dendrograms and evaluate model efficiency. Downsides are potential overfitting, not incorporating prior information. A disadvantage of using a Bayesian framework is that it is more computationally intensive given now we have probabilities of the merged hypothesis which must be updated on each iteration.

3 Description of algorithm

Bayesian hierarchical clustering goes through two algorithms presented in the paper. The first starts with each data point in its own cluster and goes through the following steps.
(1) Initialize the data x^1, x^2, ... x^n into $c = n$ clusters, so each point has its own cluster
(2) For each pair of clusters calculate:

$$p(D_k|T_k) = \pi_k p(D_k|H_1^k) + (1 - \pi_k)p(D_i|T_i)p(D_j|T_j)$$

i and j stand for any tree i and tree j respectively, which combine to form tree k. We initialize $p(D_i|T_i)$ such that

$$p(D_k|H_1^k) = p(D_i|T_i).$$

To calculate $p(D_k|H_1^k)$ refer to the appendix.
(3) To calculate the π_i in (2) we use the following iterative algorithm shown in (4). First, though, initialize $d_i = \alpha$ and $\pi_i = 1$.
(4) With trees i and j defined above (d_{left} and d_{right} correspond to trees i and j respectively), n_k is the number of data points after merging trees i and j respectively, α is a prior parameter corresponding to approximately the number of expected clusters, calculate π_k as

$$d_k = \alpha\Gamma(n_k) + d_{left}d_{right}$$

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

(5) In each step, compare all possible trees i and j. For any particular tree, use the values of d_{left}, d_{right}, $n_k = n_i + n_j$ from the previous step to calculate π_k and then the values of $p(D_i|T_i)$, $p(D_j|T_j)$, along with $p(D_k|H_1^k)$ from the appendix to calculate $p(D_k|T_k)$. Finally, calculate

$$r_k = \frac{\pi_k p(D_k|H_1^k)}{p(D_k|T_k)}$$

.

(6) Store each r_k from each possible combination above and combine the trees i and j corresponding to the largest value of r_k. For the trees that are not combined, $P(D_i|T_i)$, d_i, n_i, along with the data points in each cluster, are carried along to the cluster in the next step. These determine the updates for the non-merged clusters. d_k from Equation (4), n_k from $n_i + n_j$, and equation (2) for $P(D_k|T_k)$ determine the updates for the merged cluster.

(7) Continue merging clusters in this way by repeating steps 1 to 6. For example, after two data points are merged, we have $n - 1$ clusters. In each iteration of the first 6 steps, we combine one new cluster i and j. Stop once the desired number of clusters is reached, or we have 1 large cluster with all the data in it. The history, starting from every point in its own cluster, to one large cluster, contains the optimal Bayesian clustering for any desired number of clusters.

4 Optimization and performance

First, we performed a prior analysis, to examine different prior knowledge. For each analysis, we used the covariance matrix as η, and the mean of the data as μ_0, however μ_0 in a real world analysis, one could choose μ_0 after consulting literature about typical expected values of the variables in the glass data. To optimize our performance, we relied on Cython, parallelization, coding our probability function in C, as well as improving the efficiency of our code in general. To examine the performance, we looked at three comparisons: system running time, node impurity measure, and graphs to demonstrate the simulated data.

Note that the node impurity measure is calculated from known discrete class labels c_1, c_2, ..., c_n, by first picking a leaf uniformly at random from the data. Then pick another leaf uniformly at random, which has the same real class. Next, find the smallest sub tree which contains both leaves. The dendrogram purity is the expected value of the fraction of leaves which are in the same class.

4.1 Prior analysis

			System Run Times		
(alpha, kappa, v0)			Bayesian Clust	Bayesian Fast	Cython
alpha			cell1	cell2	cell3
alpha			cell4	cell5	cell6
alpha			cell7	cell8	cell9

d	e	f
cell1	cell2	cell3
cell4	cell5	cell6
cell7	cell8	cell9

4.2 Cython

We used Cython, defining each of the data variables in C, while using python code where it could not be improved. We coded a recursive factorial function which was faster than the Python version, a cythonized function to calculate the max value of the matrix, and the mean of a row in the matrix, as well as cythonized each of the component functions of the final bayesian$_c lust_c yt$.

4.3 General efficiency of code

We improved the efficiency by a factor of two, by calculating only the upper triangle of r_k that two clusters i and j would be merged. This is due to the lower triangle of clusters j and i being merged being the same r_k. We further improved the efficiency of the code by using decimal fixed point arithmetic instead of floating point. This allows us to work with numbers that are ten times larger on the log scale. One other improvement worth noting is decimals allow the user to represent values exactly, in contrast to binary representations where one can expect round off. There are also commands to alter the precision and make it as large as need be.

As a side note: Our code currently operates on a time complexity of $O(n^3)$. One can reduce this to $O(n^2)$, if on each iteration of the algorithm described above, one kept track of the previous clusters that were calculated. This becomes complicated because after merging a cluster, we must delete the previous clusters that become part of the merged cluster. However, this the clusters being deleted could be part of the original matrix (each individually point a cluster), or a newer cluster formed, and it is tricky to update the matrix.

5 Applications to simulated data sets and real data sets

For our comparison to real world data sets, we examined the glass data set presented in the paper as well as a real world data set with brain weight against head size.

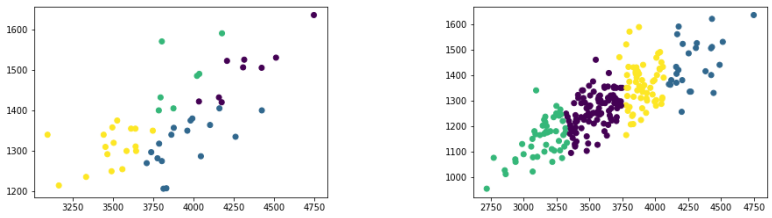


Figure 1: Left: Bayesian clustering on 50 points, right: traditional method on entire dataset

5.1 Glass Data Set

This dataset came from the UCI repository. There was a refractive index, percentage weights of various elements such as sodium or magnesium, and 6 different types of glass.

5.2 Simulated Data

We simulated 20 observations from a multivariate normal distribution, with mean the 0 vector and covariance the identity matrix. We also simulated 20 observations from 4 different cluster means $(-1, -1)$, $(-1, 1)$, $(1, -1)$, and $(1, 1)$.

5.3 Brain Weight an Head Size

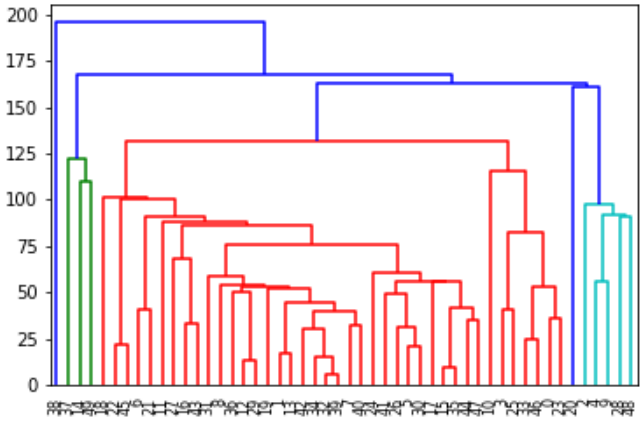
This data set had groups for male and female has well as a second variable for age range between 20-46 and older than 46. We interpreted this as the data having four classes: younger and male; younger and female; older and male; and older and female. Head size was given in cubic centimeters and brain size was given in grams. As shown below:

6 Analysis with competing algorithms

As mentioned above, our comparison involves examining a Bayesian hierarchical model to a standard k-means and standard hierarchical agglomerative clustering. The standard k-means model first picks points at random in the data to be cluster centers. It then assigns points to cluster centers based on which center each point is closest to. In the next step, we change the cluster centers to be the center of the points in each cluster from the previous step. We then repeat, assigning the points to the nearest cluster center as before.

In contrast agglomerative clustering is like the Bayesian hierarchical clustering: each point starts out with its own cluster center. In subsequent steps: the algorithm first finds the closest points in separate clusters, for each pair of clusters, and then merges the two clusters with the closest points.

Also, for these standard algorithms we provide dendrograms containing the hierarchical cluster information for comparison.



7 Discussion/conclusion

8 Appendix: Calculations of Marginal Likelihood

First note that:

$$p(D|B) = \frac{p(D,\theta|B)}{p(\theta|B)}$$

therefore,

$$p(D|B)p(\theta|B,D) = p(D,\theta|B) = p(\theta|B)P(D|B,\theta)$$

Using the Normal Inverse Wishart as the prior we obtain

$$p(D|B)NIW(\theta|\mu_n,\lambda_n,\eta_n,\nu_n) = \prod_i^n MVN(y_i|\mu,\Sigma)NIW(\mu.\Sigma|\mu_0,\lambda,\eta,\nu)$$

Therefore, we are tasked with calculating

$$\frac{\prod_i^n MVN(y_i|\mu,\Sigma)NIW(\mu.\Sigma|\mu_0,\lambda,\eta,\nu)}{NIW(\theta|\mu_n,\lambda_n,\eta_n,\nu_n)}$$

First, the likelihood function is:

$$\prod_{i=1}^n |2\pi\Sigma|^{-\frac{1}{2}} exp(-\frac{1}{2}(y_i-u)^T\Sigma^{-1}(y_i-u)$$

$$= (2\pi)^{\frac{-n}{2}} |\Sigma|^{\frac{n}{2}} exp(\frac{-1}{2}[\sum_i (y_i - \bar{y})^T \Sigma^{-1} (y_i - \bar{y}) + (\bar{y} - \mu)^T \Sigma^{-1} (\bar{y} - \mu) + 2(y_i - \bar{y})^T \Sigma^{-1} (\bar{y} - \mu)])$$

$$= (2\pi)^{\frac{-n}{2}} exp(\frac{-n}{2}[\sum_i (\bar{y} - \mu)^T \Sigma^{-1} (\bar{y} - \mu)) exp(\frac{-1}{2} tr[\Sigma^{-1} \sum_i (y_i - \bar{y})(y_i - \bar{y})^T])$$

$$= (2\pi)^{\frac{-n}{2}} exp(\frac{-n}{2} \sum_i (\bar{y} - \mu)^T \Sigma^{-1} (\bar{y} - \mu)) exp(\frac{-n}{2} tr(\Sigma^{-1} S)),$$

where

$$S = \frac{1}{n} (\sum_i (y_i - \bar{y})(y_i - \bar{y})^T$$

Multiplying this we get the product of the two expressions below

$$= (2\pi)^{\frac{-n}{2}} |\Sigma|^{\frac{-n}{2}} exp(\frac{-n}{2} \sum_i (\bar{y} - \mu)^T \Sigma^{-1} (\bar{y} - \mu)) (2\pi)^{\frac{-1}{2}} |\Sigma|^{\frac{-1}{2}} exp(\frac{-1}{2} (\mu - \mu_0)^T \lambda \Sigma^{-1} (\mu - \mu_0))$$

$$C * |\Sigma|^{\frac{-(\nu+p+1)}{2}} exp(\frac{-1}{2} * tr[\eta \Sigma^{-1}])$$

where

$$C = \frac{|\eta|^{\frac{\nu}{2}}}{2^{\frac{\nu p}{2}} \Gamma_p(\frac{\nu}{2})}$$

and Γ_p is the multivariate Gamma function.
Working through the expression in the exponent of the exp terms in the first line above, we get

$$-[\frac{n}{2} (\bar{y} - \mu) \Sigma^{-1} (\bar{y} - \mu) + \frac{1}{2} (\mu - \mu_0)^T \lambda \Sigma^{-1} (\mu - \mu_0)]$$

Completing the square then gives the sum of the two terms below:

$$\frac{-1}{2} [(\mu - \frac{\lambda \mu_0 + n\bar{y}}{(\lambda + n)})^T (\lambda + n) \Sigma^{-1} (\mu - \frac{\lambda \mu_0 + n\bar{y}}{(\lambda + n)})]$$

and

$$-(\frac{\lambda \mu_0 + n\bar{y}}{\lambda + n}$$

9 References/bibliography