STA 663 Final Project

Optimization of Bayesian Hierarchical Clustering

Matt Welch, Yamac Isik
Netids: mtw25,yai3

May 3, 2018

Github Repository:

https://github.com/mtw25/Stat663FinalProject-Matt_Yamac

1 Abstract

We perform a Bayesian agglomerative hierarchical clustering. This algorithm uses a Normal-Inverse-Wishart prior, and we evaluate the marginal likelihoods[1], incorporating prior information, to attain the highest probability under the merged hypothesis H_1. The algorithm uses a tree node data structure to calculate the output tree. We then optimize this model using better data structures,looping techniques, and Cython to increase 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 purity measure. For the data sets with two predictors, we graphically display the clustering. Our results also show the comparisons of output given various different prior knowledge for the parameters on the Normal-Inverse-Wishart distribution.

2 Background

For this analysis, we examine a Bayesian Hierarchical Clustering Algorithm based on the paper by Katherine A. Heller and Zoubin Ghahrmani(2005). First we code the algorithm as described in the paper, optimize its performance, and then compare it to other methods such as a standard k-means and hierarchical agglomerative clustering. Our work differs from these models by using a Bayesian posterior calculation rather than a distance metric to calculate the clusters.

[1]derivation can be found in the appendix

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, 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 the model efficiency. Downsides are potential overfitting and not incorporating prior information. Bayesian clustering framework is more computationally intensive given we have to update the probabilities of the merged hypothesis on each iteration. However, it potentially can improve the purity measure.

3 Description of the Algorithm

Bayesian hierarchical clustering consists of 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 contains the optimal Bayesian clustering for any desired number of clusters.

For the tree structure that keeps track of the clustering process we defined a new class called "Node".The Node class has the regular tree node data structure with the following instance variables:

· **points** : A set which keeps track of all the observations belonging to the tree node.

· **d** :The d_k value of each cluster, initially set to α.

· **number** :The number which represents the clusters, initially set the indices of each observations. Newly formed trees take the number of the tree with the lower number.

· **left,right** :Node variables which represents the nodes that have been used to form the cluster. These variables allow us to travel through the tree using recursive methods.

· **ph** : A variable that keeps track of the probability of the tested hypothesis, if the node is a leaf it just gives the probability distribution for that point.

We have also created class functions for the tree node structure. Specifically, the function combine, creates a new node by combining two existing nodes and updating the points and left and right variables accordingly.

4 Optimization and Performance

4.1 Run Time

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, 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 created three different models. The first function is called bayes_clust and calculates the tree structure by creating two Numpy matrices and saves the respective π and r values using a double loop for all the possible combinations. For example, if the first step goes through all n^2 nodes, the next step should go through all $(n-1)^2$. The last step goes through k^2 nodes.

Our second model is called bayes_clust_fast which exploits the symmetry in our Numpy arrays, improving efficiency by a factor of two. It does that by calculating only the upper triangle (without the diagonal) of r_k that two clusters i and j would be merged. By looping through the combinations that are only in upper triangle parts of the matrix, this reduces the run time of our algorithm by more than 50 percent.

Lastly, we also used Cython to further reduce the run time of the our algorithm. By defining types of each variable and using memory view structures we were able to optimize our calculations. Defining and writing the commonly used math functions within Numpy such as, numpy.max, numpy.factorial and numpy.mean in C reduced the speed of our calculations further. From our tests, the cythonized version was 70-80 percent faster than the fast model.[2]

4.2 Accuracy and Precision

For comparing the accuracy of our models, we use the purity measure defined by Heller and Ghahrmani. The purity measure is calculated from known discrete class labels c_1, c_2, ..., c_n, by first picking a leaf, a sample point, uniformly at random from the data. Then we pick another point uniformly at random that belongs to the same class in the actual data. Next, we find the smallest sub tree which contains both leaves. The dendrogram purity is given by the expected value of the fraction of leaves which are in the same class as these two points.

We further improved the precision of our code by using decimal fixed point arithmetic instead of floating point for the probability calculations. 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.This trade of between speed and precision proved to be useful because of the large exponential values within our probability calculations.

Impurity Score for Bayesian Clustering [3]

(kappa, v0)	Brain Data Set	Synthetic Data Set
$lenth(X)/20$, $\kappa 0 * 2$	0.46593	0.63607
$lenth(X)/20$, $\kappa 0 * 5$	0.47699	0.68771
$lenth(X)/40$, $\kappa 0 * 4$	0.41333	0.56407
$lenth(X)/5$, $\kappa 0$	0.36291	0.63076
$lenth(X)/20$, $\kappa 0 * 3$	0.46043	0.62939

[2]It is important to mention, although we used our cythonized model for our testing, due to the compiling challenges it presented we were not able to implement it for use in our package bayes_cluster. Although this is an issue we want to address in the future our bayes_fast function was enough for our data calculations.

[3]The α values for the calculations are set to be 4

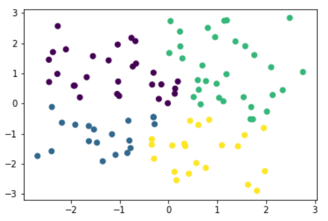
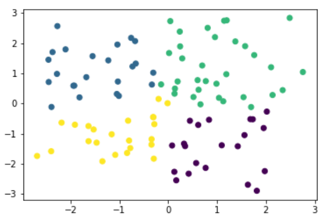



Figure 1: Left: Bayesian clustering on synthetic data, right: traditional k means on synthetic data

Comparison of three functions run time for three data sets

Function	Brain Data Set	Synthetic Data Set	Glass Data Set
Bayesian cluster	26.5 s	1.75 s	48 s
Bayesian cluster fast	12.8 s	852 ms	23.3 s
Bayesian cython	10.2 s	667 ms	21 s

5 Applications to Simulated and Real Word Data Sets

For our comparison to real world data sets, we examined the glass data set presented in the paper as well as an additional real world data set on brain weight and head size versus gender and age of individuals[4].

5.1 Glass Data Set

This data set is taken 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 as the 0 vector and covariance as the identity matrix. We also simulated 20 observations from 4 different cluster means $(-1,-1)$, $(-1,1)$, $(1,-1)$, and $(1,1)$.

[4]Initial data set had four features for gender, age, brain weight and head size. We created a class variable with 4 classifiers that combines gender and age together,this gave us a data set with 2 features and 1 classification value.

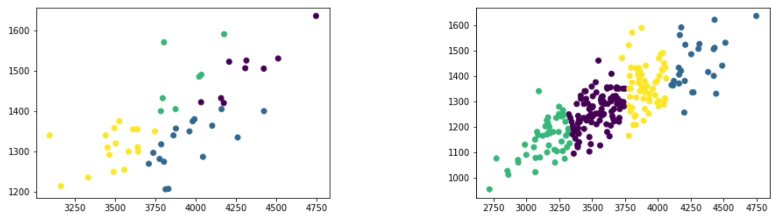


Figure 2: Left: Bayesian clustering on 50 points in Brain Data , right: traditional method on entire Brain Data set

5.3 Brain Weight an Head Size

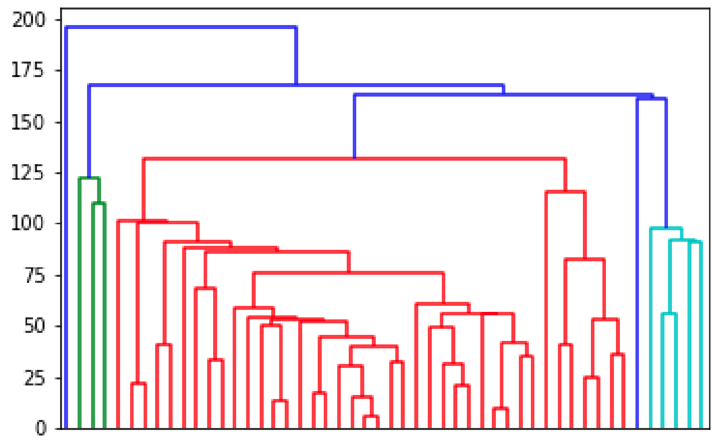
This data set had groups for male and female has well as a second variable for age range between 20-46 and older than 46. We converted this to a data set with four classes: younger and male; younger and female; older and male; and older and female. Head size was given in cubic centimeters and brain size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

6 Analysis With the Competing Algorithms

As mentioned above, our comparison involves examining a Bayesian hierarchical model and comparing it to a standard k-means/ 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 to K-means, the standard agglomerative clustering starts with each point having its own cluster center. In subsequent steps, the algorithm first finds the closest points in separate clusters. For each pair of clusters, it then merges the two clusters with the closest points. Our Bayesian model is an agglomerative clustering model which uses a different measure method for the linkages compared to the standard one.

The following figure provides a dendrogram containing the hierarchical cluster information for comparison.



The Dendogram Example for K-Means with k=3 For the Synthetic Data Set

Comparison of System Run Times With Competing Algorithms[5]

Data set	K-means	Agglormerative Link	Bayesian Hierarchical
Glass Data	39.2 ms	2.54 ms	20.2 s
Brain Data	30.3 ms	1.22 ms	10.2 s
Synthetic Data	27.8 ms	895 us	686 ms

Comparison of Impurity Measures With the Ones in the Paper

Data set	Agg. Single Link	Agg. Avg. Link	Bayesian Hierarchical
Glass Data	0.478	0.491	0.669

7 Further Improvements

Our code currently operates with a time complexity of $O(n^3)$. One can reduce this to $O(n^2)$, if on each iteration of the algorithm, we keep track of the previous clusters that were calculated and update only the rows and columns of the merged cluster. This becomes complicated since after merging a cluster, we must delete the previous clusters that becomes part of the merged cluster. However, the clusters being deleted could be part of the original matrix (where each individual point is a cluster), or a newer cluster formed. Therefore, it is tricky to update the matrix.

Furthermore, working with multiprocessing packages with pool options such as multicoreprocessing, the speed of our calculations can be reduced further. One must note however in order improve the run time, the double loop in our calculations has to be converted to a single loop by creating a $n^2 by 2$ Numpy

[5]The prior used for these calculations are set to be the same as the ones used in the purity calculations (the one with the most accurate purity).

array that gives all the index combinations to a single loop. With the single loop we can use parallel programming to further increase the speed of our model.

8 Conclusion

Our model implemented a Bayesian Clustering Algorithm using various prior values to decide which clusters to combine. Using a tree node structure we are able keep track of each combination at each step. Furthermore, unlike conventional clustering methods from packages such as sklearn, we can use a recursive method to calculate the purity scores for our classifications.

Our results on both the real and synthetic data sets show that Bayesian clustering can produce similar results to K-Means and Single Link Agglomerative algorithms. However, setting different initial values for our parameters can lead to different results from conventional clustering.This means with proper prior knowledge about our data set, our algorithm can produce more accurate results.

9 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

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

Working through the final algebra, plugging into the initial equation above, gives us.

$$p(D|B) = (\frac{1}{\pi})^{\frac{np}{2}}\frac{\Gamma_p(\frac{\nu+n}{2})}{\Gamma_p(\frac{\nu}{2})}\frac{|\eta|^{\frac{\nu}{2}}}{|\eta_n|^{\frac{\nu+n}{2}}}[\frac{\lambda}{\lambda+n}]^{\frac{p}{2}}$$

Where $\nu_n = \nu + n$ p is the number of parameters in the model, n is the number of rows in D, and

$$\eta_n = \eta + \sum_i(y_i-\bar{y})(y_i-\bar{y})^T + \frac{n}{\lambda+n}(\bar{y}-\mu_0)(\bar{y}-\mu_0)^T$$

10 References/Bibliography

[1] Katherine A. Heller and Zoubin Ghahramani , Bayesian Hierarchical Clustering, Duke University, 2005.

[2] Kevin P. Murphy, Conjugate Bayesian analysis of the Gaussian distribution, The University of British Columbia, October 3, 2007 (Equation 266).

[3] Filipovych, Roman; Resnick, Susan M.; Davatzikos, Christos (2011). ”Semi-supervised Cluster Analysis of Imaging Data”. NeuroImage. 54 (3): 2185–2197. doi:10.1016/j.neuroimage.2010.09.074. PMC 3008313Freely accessible. PMID 20933091.

[4] Bewley, A.; et al. ”Real-time volume estimation of a dragline payload”. IEEE International Conference on Robotics and Automation. 2011: 1571–1576.