a8-report

Jiangtao Xu

11/13/2017

Environment

OS: OSX Processor Name: Intel Core i7 Processor Speed: 2.8 GHz Number of Processors: 1 Total Number of Cores: 4 L2 Cache (per Core): 256 KB L3 Cache: 6 MB Memory: 16 GB SSD: 256 GB

Implementing K-means and HAC

K-Means Clustering

Get initial centroid: get data from input file, map each row of data into `SongInfo` object. `SongInfo` provides function to calculate distance like `calculateDistance()` , valid checking function `isValid()` , function to get required filed like `getSymbol()` , which will be used later.

Select 3 `SongInfo` randomly by using `takeSample()` , if selected data is invalid, then reassign random value to that data.

Get cluster by centroids: filter the data if it's valid. According to the distance from 3 centroids, clustering data into 3 parts by using `getClusterByCentroids()` function.

Recalculate centroids: for the new 3 clustering, recalculate the centroid nodes for each part, use `getCentroids()` function.

Output: repeat the above steps 10 times. and output the final centroids and clusters.

Hierarchical Agglomerative Clustering (HAC):

Sort SongInfo: collect and order SongInfo data by required field where use `getCoordinate()` fucntion to get values to be sorted, according to the input symbol, return the different coordinate for 1 dimension and 2 dimension data. At the end, map each data into `List[SongInfo]` and append with index by using `zipWithIndex` function. This step return clusters named `clusters` in program.

For example, for input value {2,3,5,8}we have {(list(2),1), (list(3),2), (list(5),3), (list(8),4)} from 1st step

Get min pair: Calculate the distance for each possible combination of `List[SongInfo]` , select the minimal distance pair.

For example, we could get minimal distance pair {list(2),list(3)} from 2nd step.

Combine the min pair: remove the element of the minimal pair in `clusters` , and insert the minimal pair into the removed place.

For example,we would delete(list(2),1), (list(3),2) from `clusters` , and insert (list(2,3),1) in the removed place, here we get { (list(2,3),1), (list(5),3), (list(8),4)}

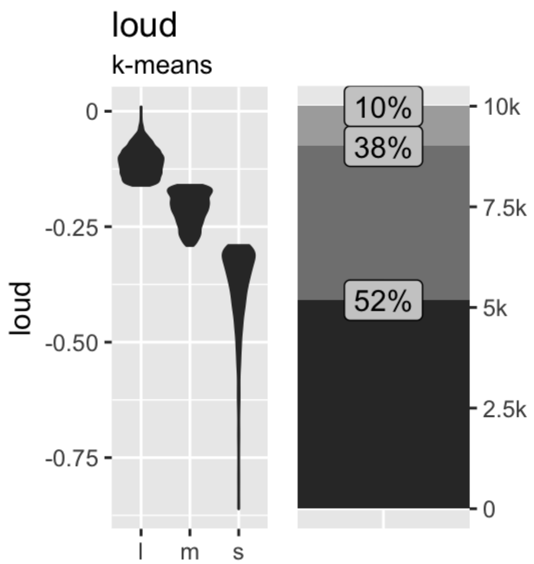
Output repeat the 2nd, 3rd steps until the length of `clusters` is equals to 3, then output the clusters.

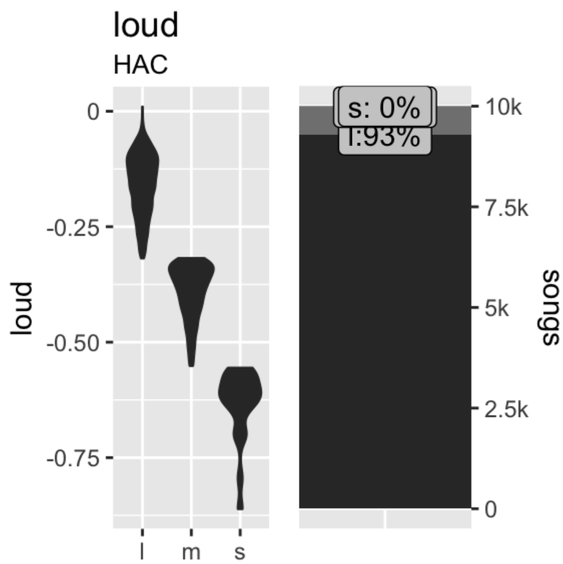
Result

To show and compare the result, use MillionSongSubset as input data. The flowing are results and comparisons between K-Means and HAC.

There are two graphs in each section that decribe the result of cluster by k-means and HAC. For each graph, there are 2 parts of the clustering: left(up for the combined hotness) is the distribution of 3 clusters (l: large, m: medium, s: small which is named according to the value of centroids of each parts); right(down for the combined hotness) is the percent portion of each cluster.

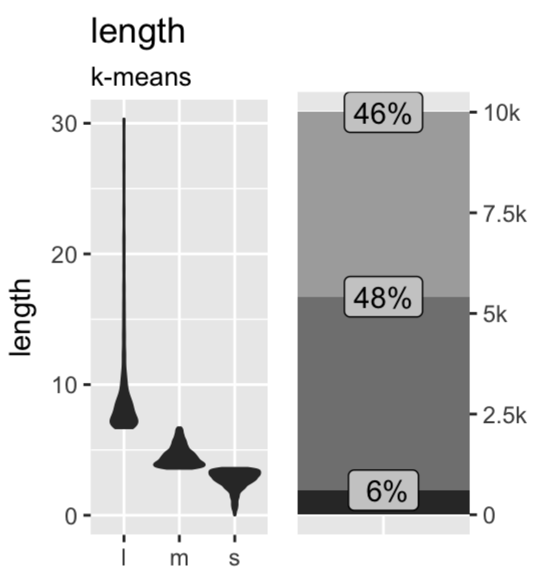
Fuzzy Loudness

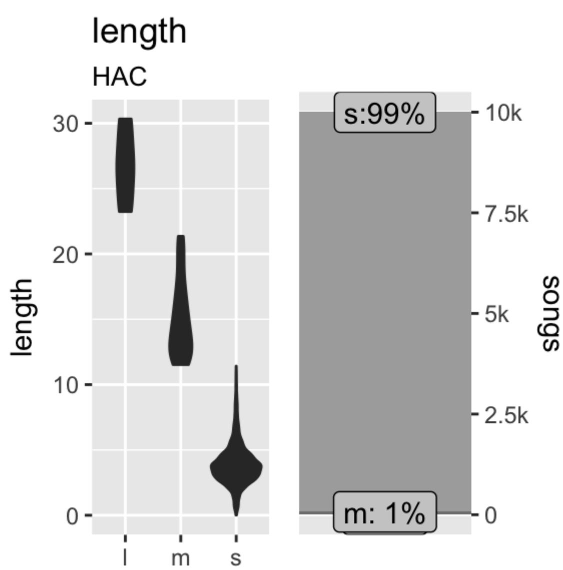




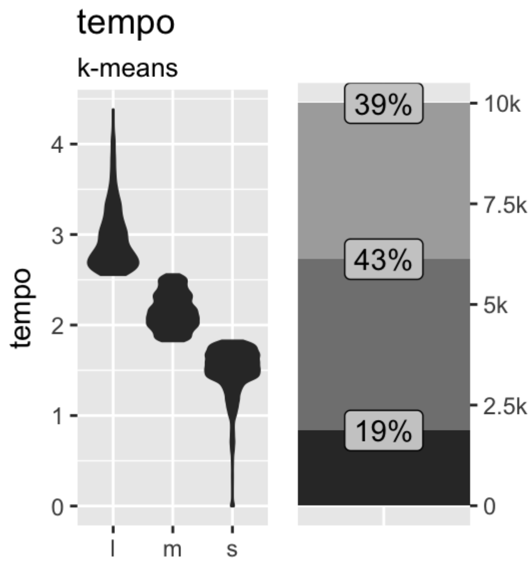
Fuzzy Length

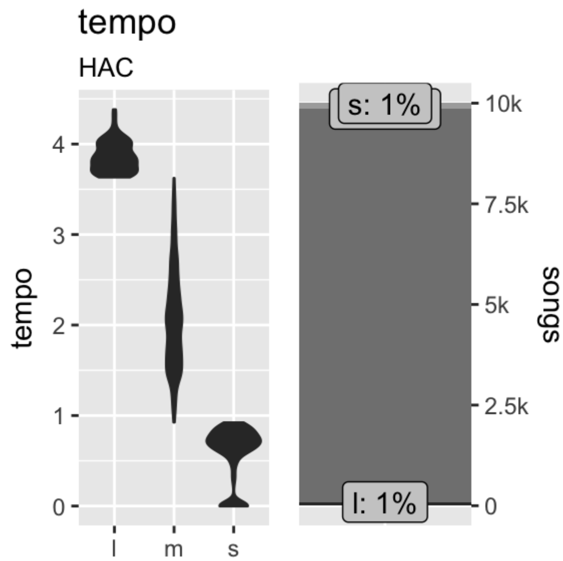
The two graph below





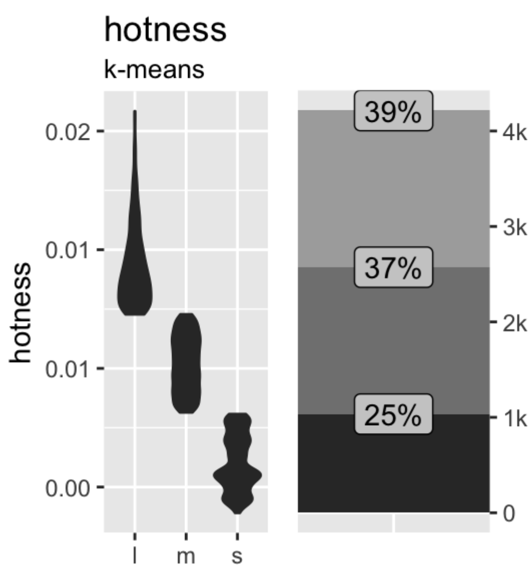
Fuzzy Tempo

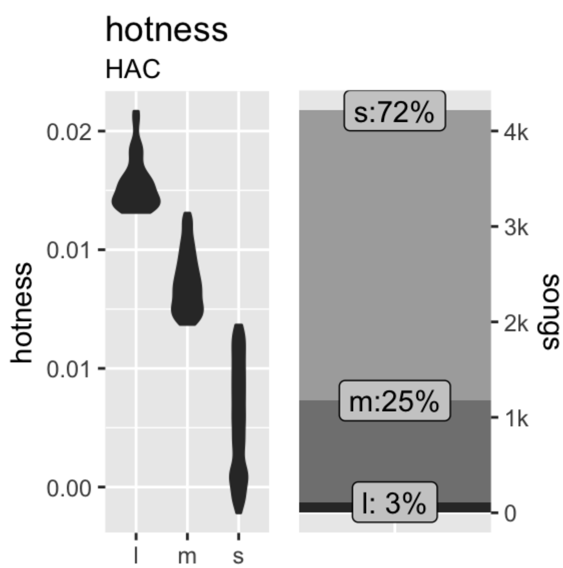




Fuzzy Hotness

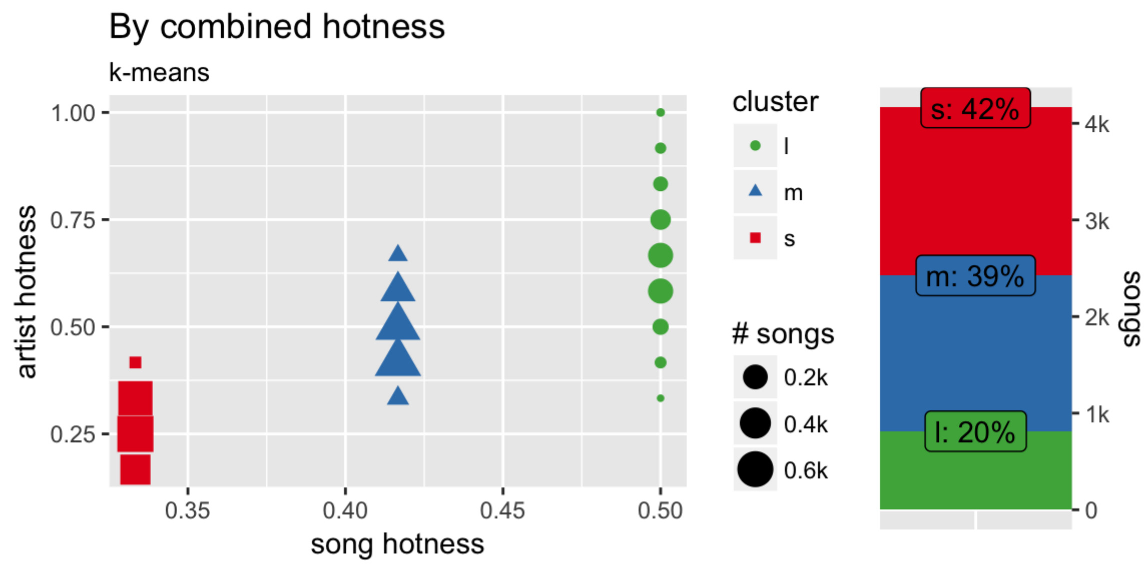
For song hotness, since there are a lot of NA, empty and 0 of hotness filed, so I just filter data with that filed.

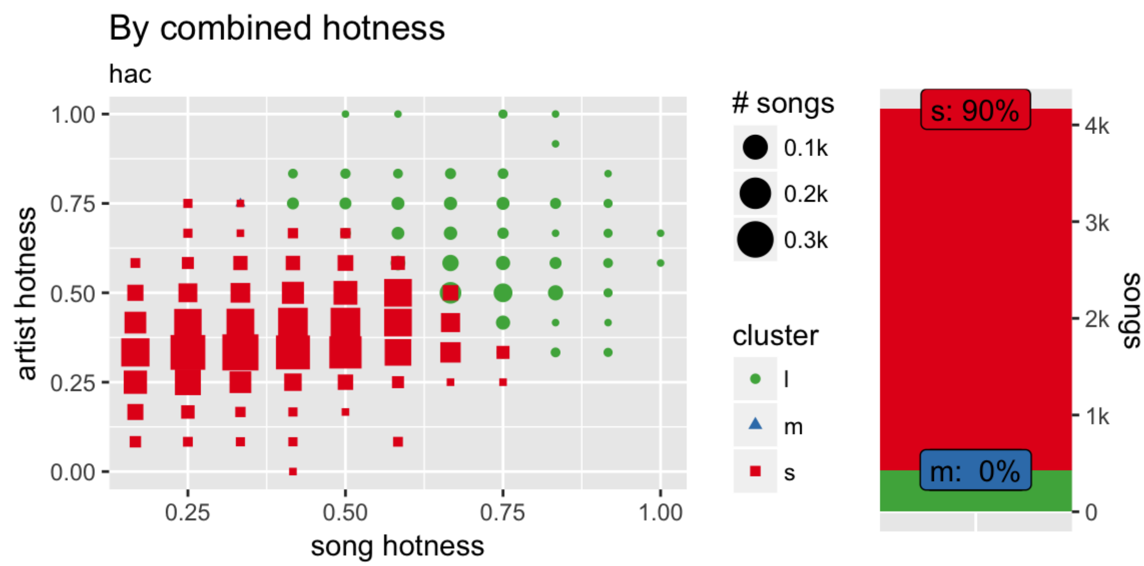




Combined Hotness

For song hotness and artist hotness, since there are a lot of NA, empty and 0 of hotness filed, so I just filter data with that filed.





Obervation and Conclusion

Performance

I run it on local machine, for K-Means, it takes around 6 seconds; for HAC, it takes around 20 seconds to finish. Possible Reason: for HAC, it takes O(n) time complexity to get every pair when using `zip()` function , then it need O(n) for both find and inserting min pair operation. It seems like not much time consuming, however, before this step, using `collect()` function would be very space expensive, especially for big data input, that maybe the factor that influence the time, and that's also why I cannot get result from big input for HAC.

Result

According to the plot, it seeams kmeans render a more reasonable result, the percentage of each clusters is more balanced than HAC. For example: (39%, 43%, 19%) for the tmpo of kmeans compared to (1%, 98% 1%) of HAC.

So, K-Means might be a more powerful cluster algorithm when it comes to performance and specification.