Assignment Report - A8(A7)

Shabbir

October 31, 2017

Abstract

This report describes results of different kinds of clustering on the million songs dataset. Detailed requirements can be found here (http://janvitek.org/pdpmr/f17/task-a7-clustering.html). The first section describes the experiment design, setup and brief structure of project. Followed by second section starts with results visualization and analysis of subtasks including run time statistics. Finally we conclude with findings from both sections.

1. Design of experiment

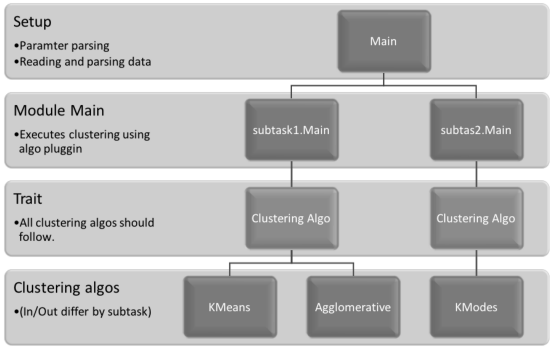

Figure 1

Our clustering architecture for subtask 1 is a single 2d clustering algorithm which takes 2d points RDD and returns 3 clusters from those points as set of points assigned to centeroids. For each problem within subtask we filtered data and passed an RDD of just (x, y) coordinates. For 1d clustering y coordinate is set to static value.

Case classes were intentionally not used in the code as it made the code lines longer and incur added penalty for pattern matching. Instead tuples with comments convention is used throughout the code.

We ran all experiments on a MacBookPro12 with 1 Intel Core i5 processor (2.7 GHz, 2 cores) with 8 GB ram.

2. Results

2.1 Subtask 1

We ran K means clustering on the million song dataset until convergence. Convergence criteria used is when centroids don't change over 2 consecutive iterations. We found clustering takes 5 minutes till convergence for all tasks in subtask 1.

2.1.1 1D clustering

Figure 2 shows various clusters found using K-Means on million songs. The graphs shown are step heat maps. Here intensity of color shows the amount of songs concentrated at particular interval(bucket). As data is fractional we clubbed neighboring points together using buckets. Title of the plot shows bucket size used within that graph. Along the Y-axis we have percentage of songs. Height of the heat-step specifies the amount of songs in that cluster. Added together they span full corpus (100% songs). Relative size of clusters gives an indication of total number of songs present in that cluster. Every step specifies start of a different cluster, whose name can be found on the Y2 axis (right-hand-side).

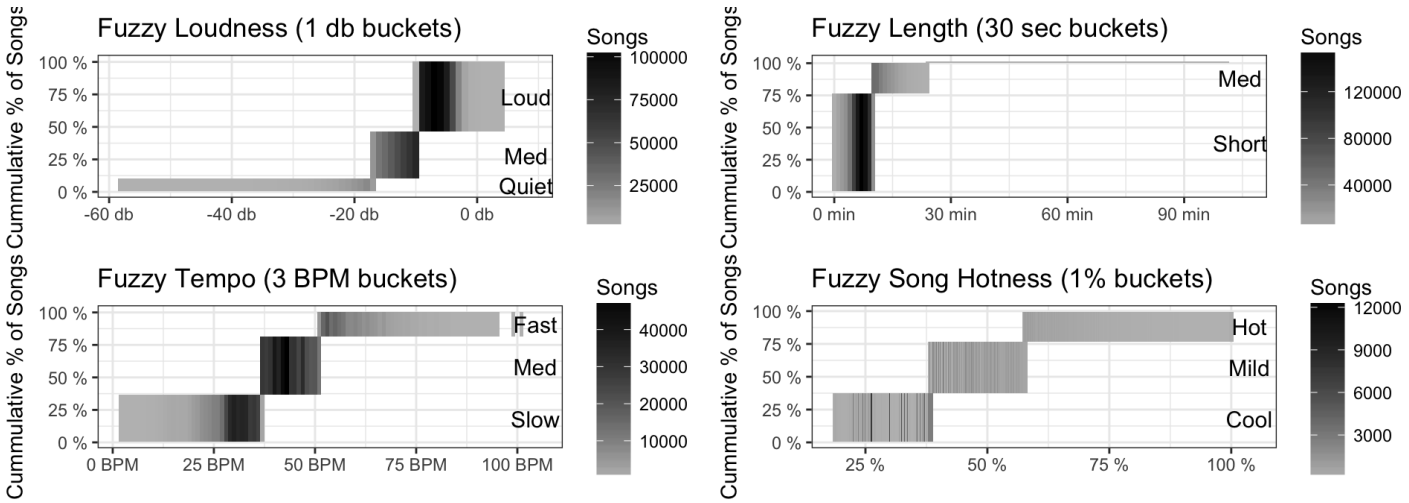

Figure 2

Observations

- Clusters have some overlapping in the transition bucket as we are intentionally removing precision while reporting.
- We see there are a lot more loud songs (more than 51%) and just a few quite songs.
- Most of the density of songs is concentrated within -9 to -5 db in loud songs.
- There is more concentration of songs between 25BPM to 50BPM tempo. Hinting towards a sweet range of listening for audience.
- Most songs lie in the 7-9 Minutes range.
- Interestingly there exists some density of songs all over from 20 plus minutes to 60 minutes.
- Most songs are pretty evenly distributed in the hotness spectrum with gradual gradient transition.
- There seems to be a single odd dense region in the cool songs around 26% which doesn't seem to have a gradual transition.

2.1.2 2D Clustering

Here we cluster songs based on artist hotness and song hotness. The graph in Figure 3 shows a density plot. The plot is segregated by shades of grey signifying different cluster categories. Graph also has a legend on the side showing the color to category correlation along with relative percentage of songs within them.

Observations

- There seems to a correlation between artist hotness and song hotness. This can be seen from the lack of hot songs near less than 20% artist hotness.
- When artist hotness is between 20-60% songs hotness can range from 20-100%.
- There is a large concentrations of songs near 40% artist hotness and 50% song hotness.

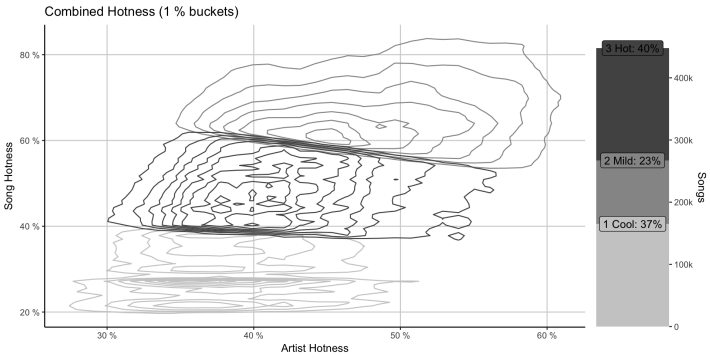

Figure 3

2.1.3 Agglomerative 1D Clustering

We implemented agglomerative clustering technique with spark on 1D data points using following version of algorithm:

1. Sort all points and assign an index to every point using `zipWithIndex` and join it with consecutive indices to compute their distances.

2. Loop until count(unClusteredPoints) is greater than required number of clusters
    1. Find minimum dist in unClusteredPoints.
    2. Filter out **all** elements that have the same distance as min dist from unClusteredPoints. (Cluster multiple points in single iteration).
    3. The above assumption is valid as all points are sorted and are 1D. And after merging new cluster distance to any neighbor could never be less than min distance found before. (This helps us speed up algo by many folds.)
3. Finally we are left with a minimal set of points in the unClusteredPoints RDD.
4. We set original point distances to $+\inf$ and compute final cluster assignment by replacing these $+\inf$ distances with unClusteredPoints.
5. Then do a groupBy and collect all points on the driver to sequentially loop through them.
6. In this design whenever we find a non infinite value (point from the unclustered set) we get a cluster boundary.

Below figure shows a comparison of Agglomerative clustering output vs Kmeans for Fuzzy loudness on the million songs dataset.

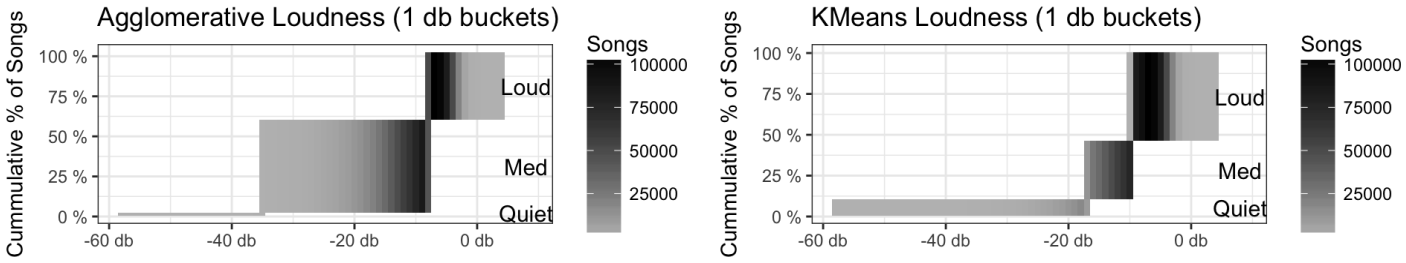

Figure 4

Observations

- We found that Agglomerative clustering is difficult to parallelize due to one update at a time constraint. We can parallelize the search for updates and filters. However we still have to go through a lot of iterations to achieve the final clustering result.
- It took twice as much time as Kmeans to run on the million songs dataset. (K-Means took 70sec, while, agglomerative took 137sec.)
- We can also see some imbalance between the clusters found by Agglomerative and K-Means. It comes from the fact that Agglomerative chooses to merge the closest clusters first, while K-Means picks centroids and tries to assign points to it while updating the centroids.

2.2 Subtask 2

The goal of this assignment is to cluster artists based on their genre.

Assumptions and Specifications

- **Our centroids are not contrained to be a valid point in the graph.**: It could be a subset of any combination of *terms*. Thus, we cannot use the artist similarity to construct edges between artist and centroids. In our case we assume all artists are connected to each other iff they have at least 1 common term. And distance between them is the inverse of the intersection of matching *terms*.
- We assume artist `A` is similar to artist `B` reverse is **NOT** true. As in case of $A \subset B$.
- `KMode` : It is based on KMode clustering technique.
    1. Initially we assign 1 as weight to all terms associated with any artist.
    2. We then take intersect count of *artist terms* to *centroid terms* as a distance measure.
    3. To calculate new centroid position we take mode (most repeating values) of, ***set of artist terms***, for all artists belonging to that cluster.
    4. Finally, we calculate distance to this new centroid by using same intersection technique.

Visualization

Key information to read below figure:

1. All clusters are shown as bunch of tag clouds.
2. Every tag cloud is in row major format. And are sorted by `CLUSTER_ID`.
3. The size of the word indicates the number of artists assigned to that cluster.
4. While the color of the word(darker grey) cluster represents that term is agreeing with the respective cluster centroid.
5. Note: Some centroids are not plotted due to shortage of screen space as decided by wordcloud library!

Cluster Assignment

First Iteration

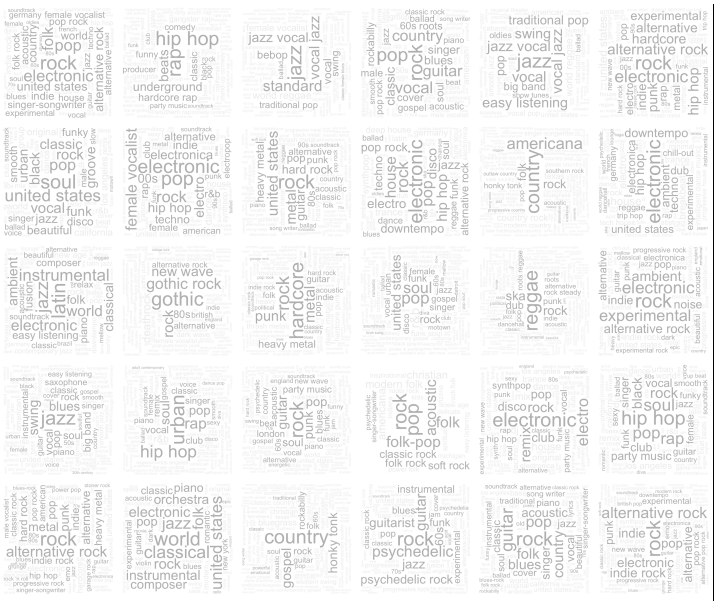

Figure 5.1

Last Iteration

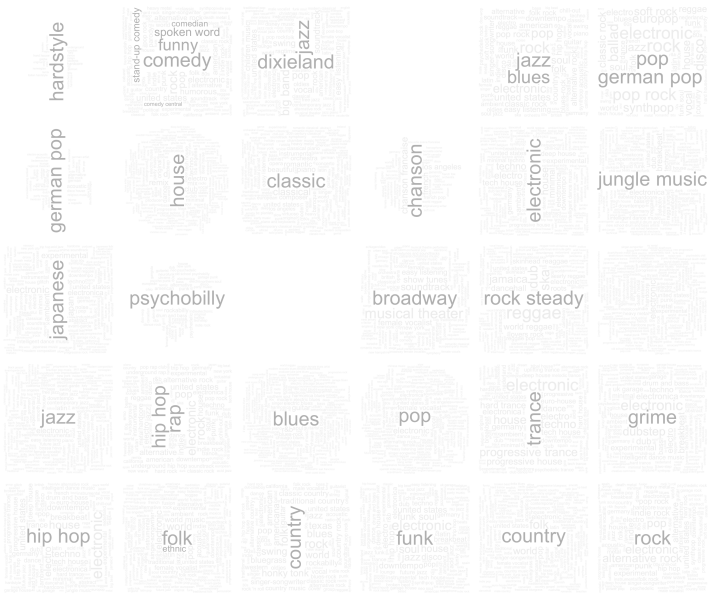

Figure 5.2

Observations

- As algorithm progresses the clusters seems to merge together to form a bigger distinction between *terms*.
- This can be seen in the case of `POP` which was prominent in every cluster earlier in first iteration. It is later aggregated to its own *term* towards the end of the algorithm.
- Clustering also found a related genre of comedy with *terms* like *funny*, *commedian*, *comedy* etc.
- There is a strong correlation between *rap* and *hip-hop* as seen in the last iteration where both terms are present together with almost equal weight.

Conclusion

1. From this exercise it can be seen K-Means is much more faster than the Agglomerative approach used here.
2. It can become difficult to compute Agglomerative for multidimensional points as every iteration Cartesian product has to calculated to compute all pairs distance.
3. Using simultaneous update of multiple min length points million songs clustering size was shrunk from $999,056$ songs to just $32,828$ songs in the first iteration itself. This optimization can be very useful for 1D data points.
4. K-Mode clustering shows promising results in terms of end clusters with much more specialized keywords for centroids after few iterations compared to the start of the algorithm. It would be interesting to explore and contrast results of some other algorithm on the same dataset.