# Song Clustering

*Rashmi*

*11/1/2017*

# A7 - Song Clustering

## Introduction

The Assignment is to perform iterative computations, graph computations. The entire description of the assignment can be found here: http://janvitek.org/pdpmr/f17/task-a7-clustering.html (http://janvitek.org/pdpmr/f17/task-a7-clustering.html). We are using a dataset based on the metadata in the Million Song Dataset - https://labrosa.ee.columbia.edu/millionsong/ (https://labrosa.ee.columbia.edu/millionsong/)

## Execution Environment

AWS EMR Cluster with 4 M3xlarge machines - Full specifications of the machines can be found here: https://aws.amazon.com/ec2/instance-types/ (https://aws.amazon.com/ec2/instance-types/)

## Clustering - Subproblem 1

### K-Means Clustering Algorithm

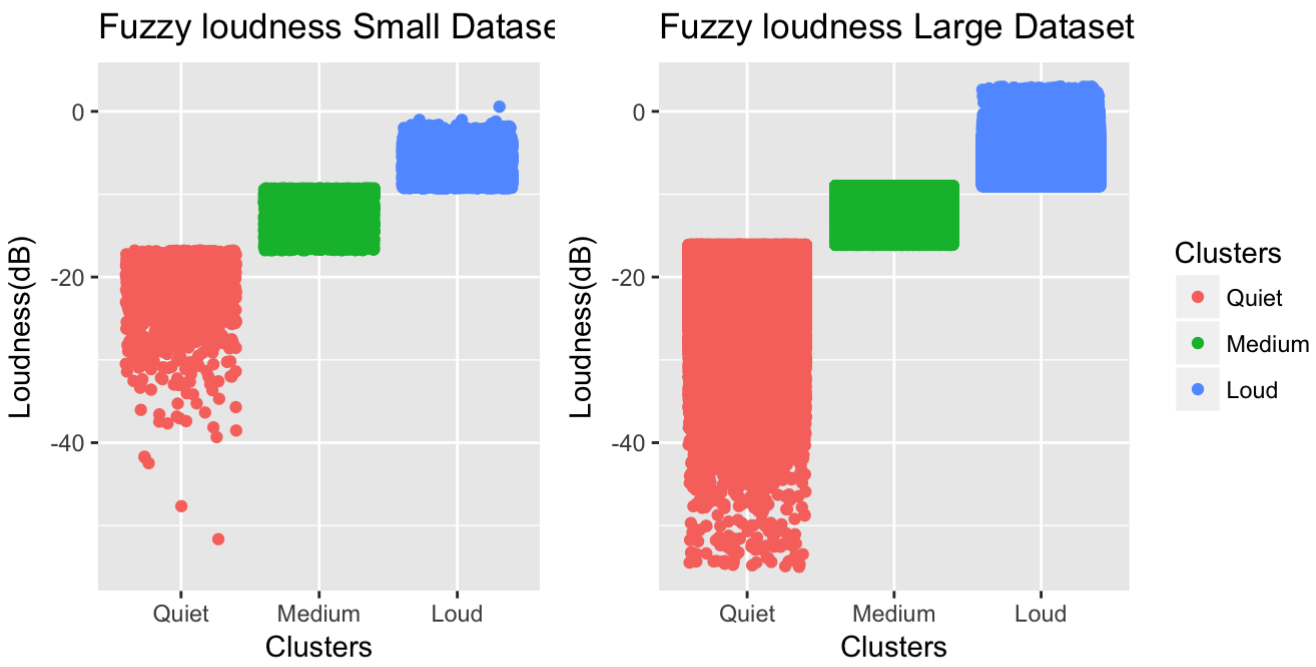
k-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. Since K-Means algorithm is sensitive to the initial centriods, I have chosen the quartiles of the loudness vector as the initial vector. The quartiles would give me points at 25, 50 and 75 percentile.

1. **Quartile Calculation**
    - Sort the RDD[(trackId, measure)] with respect to the measure.
    - Q1 - count/4 th index of the sorted RDD
    - Q2 - count/2 th index of the sorted RDD
    - Q3 - (3/4)*count th index of the sorted RDD
2. **Initial Centroids** - Assign the above computed points as the initial centroid
3. **Assigning to a cluster** - *Distance Measure* is the absolute difference between the data point measure and centroid measure. For each data point - (trackId, measure), calculate the distance to all centroid. Find the centroid which is at the smallest distance and assign the data point to that centroid
4. **Re-calculating centroids** - For all the data points in a cluster, calculate the average of the measure. That will be the new centroid for that cluster

The K means clustering algorithm is said to have converged when the centroids do not change. But, I ran K-Means for 10 iterations, and I noticed not much significant changes in the centroid and hence considered it be converged.

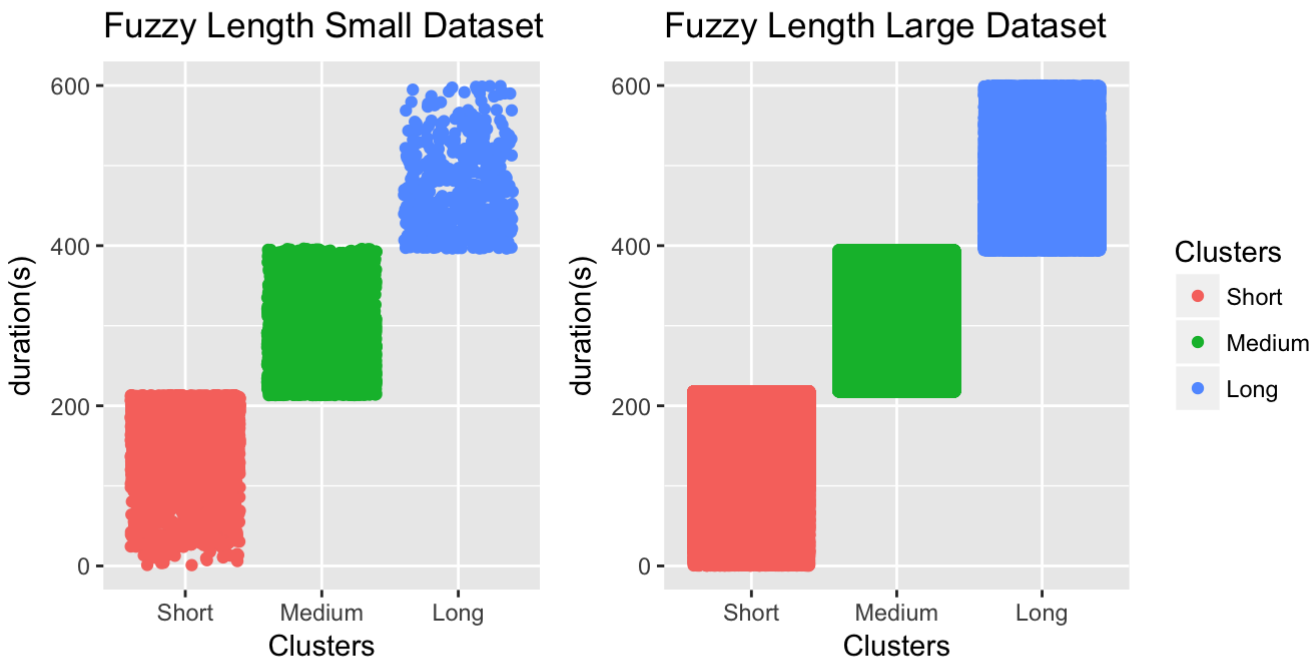
#### Fuzzy Loudness

The below graphs represent the fuzzy loudness clustering on *loudness* metric for small and large datasets.The graph represents the Quiet, Medium and Loud clusters. There are few overlapping in the conflicts that can be due to the conflicts in minimum cluster assignment.



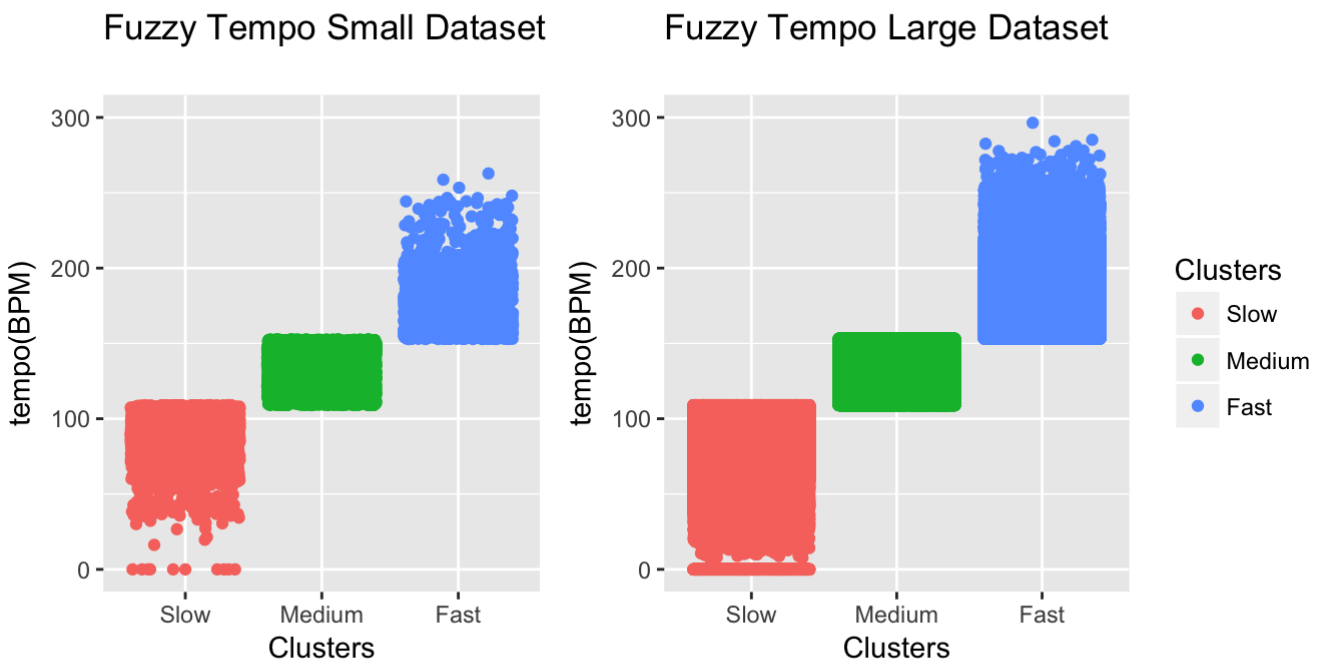

## Fuzzy Length

The below graphs represent the fuzzy length clustering on *duration* metric for small and large datasets. The graph represents the Short, Medium and Long clusters.There are few overlapping in the conflicts that can be due to the conflicts in minimum cluster assignment.



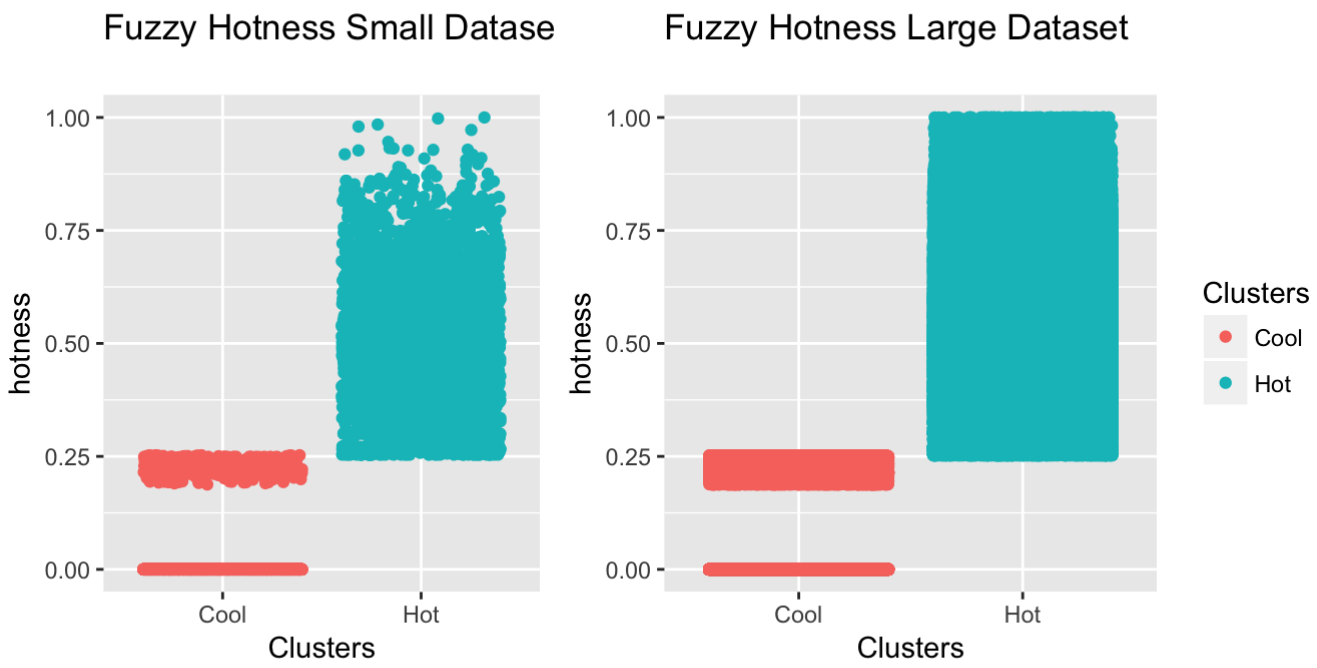

## Fuzzy Tempo

The below graphs represent the fuzzy tempo clustering on *tempo* metric for small and large datasets. The graph represents the Slow, Medium and Fast clusters.There are few overlapping in the conflicts that can be due to the conflicts in minimum cluster assignment.In the slow cluster, there are few zero points, which are invalid measure caught and replaces as 0.



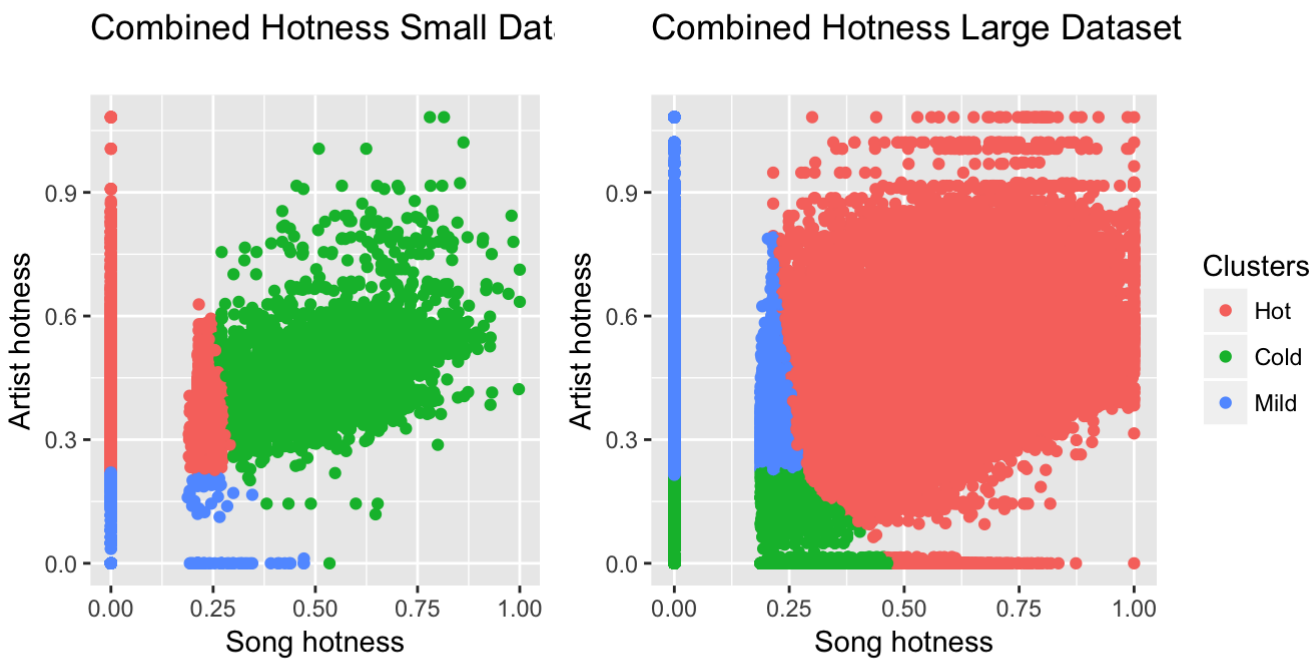

## Fuzzy Hotness

The below graphs represent the fuzzy hotness clustering on *song hottness* metric for small and large datasets. The graph represents the Cool, Mild and Hot clusters. There are few overlapping in the conflicts that can be due to the conflicts in minimum cluster assignment. In the cool cluster, there are few zero points, which are invalid measure caught and replaces as 0.




## Combined hotness

For Combined hotness, the distance measure use is the euclidean distance.The below graphs represent the Combined hotness clustering on *song hottness* and *artist hottness* metric for small and large datasets. The graph represents the Cool, Mild and Hot clusters based song and artist hotness. There are few overlapping in the conflicts that can be due to the conflicts in minimum cluster assignment. In the cool cluster, there are few zero points, which are invalid measure caught and replaces as 0.




#### Performance

- K Means Clustering for small dataset, it took around 1 minute
- K Means Clustering for large dataset, it took around 7 minute

# Agglomerative Clustering Algorithm

Agglomerative Clustering Algorithm is the bottom up approach for hierarchical clustering of data points. Initially, each data point is in its own cluster. With each iteration, pairs of clusters are merged as one moves up the hierarchy. In general, the complexity of agglomerative clustering is O(n^2) which makes it a very slow algorithm for large datasets. I have chosen the linkage criterion as the minimum distance which is the absolute difference between the 2 measures. For, combines hotness, the minimum distance is the euclidean distance between the 2 measure pair.

**Step1:**

1. Compute the Cartesian join of the RDD[(Song, measure)]
2. Filter the resulting RDD to keep only one pair of points, as the distance is symmetric.

**Step2**

1. Find the pair with minimum distance and merge them.
2. Filter all the pairs containing the above data points.
3. Find the minimum of the data point in the merged cluster, and recalculate the distance from the new cluster as the minimum distance to all the points.
4. Remove the filtered data and union with the new merged pair with updated distance to all points.
5. The above changes are tracked and maintained using an RDD - distance matrix.

**Step3**
Repeat Step2 till 3 clusters are obtained, that would be (RDD.count - 3).

The results for Agglomerative Hierarchical Clusters for 100 Song records can be found in the path `output/MillionSongSubset/EMR_Subset/agg_*` files.

### Performance

Agglomerative cluster is a **O(n^2)** time complexity algorithm. Since running on the entire dataset took a lot of time,execution was halted for the entire dataset.However,the algorithm execution completed for 100 song records and was completed in **1 hr 14 minutes for all types of clustering**

# Observations:

**A song's loudness, length, or tempo predict its hotness** - We can infer from the graphs that the hotness is relative to song's loudness, length or tempo. When we do you supervised prediction, the co-relation co-efficient of these parameters would be high.

**A song's loudness, length, tempo, or hotness predict its combined hotness** - We can infer from the graphs that the combined hotness is relative to song's loudness, length or tempo. When we do you supervised prediction, the co-relation co-efficient of these parameters would be high.

# Graphs - Subproblem 2

## K-Means Clustering Algorithm

We define, Popularity = ArtistFamiliarity * ArtistSongCount * SimilarArtistCount

1. **Initial Centroids** - Top 30 Popular Artist based on the above measure is considered as the initial centroid for the kmeans
2. **Re-calculating centroids** - Intersection of all terms in the cluster is chosen as the new centroid artist terms
3. **Assigning to a cluster** - Check the number terms similar between a point and each of its centroid. The point is assigned to a centroid based on the maximum similarity among the centroids.
4. Repeat step 2 and 3 for 10 iterations

Each iterations results are in the path output/MillionSongSubset/EMR_Subset/kmeans_commonality. I ran the code for small subset on local machine. It took 89s to run on laptop - (MacOS, 16GB RAM, i7 Dual Core). Ran into errors while running the same on EMR and unable to run it for full dataset.

# Conclusion

KMeans Clustering was very efficient in Spark. K Means algorithm works well with spherical clusters and same sized clusters. Since the data points we are exploring is not spherical and not of same size, we can see some merging or overlapping points at the borders. We might get more accuracy, if we choose initial centroids with more precision as K-Means algorithm is very sensitive to initialization step. Implementation wise, K-Means clustering performed well with spark. Agglomerative clustering turned out to be costly algorithm. It took 1hr 14min to run 5 clustering on 100 records of data. Implementation wise, Agglomerative clustering dint performed well with spark.