A7 - Clustering

Ayushi,Manthan Thakar

11/1/2017

Objective

- Using Spark, implement hierarchical agglomerative clustering and k-means clustering on Million Song Daatset.
- Define a commonality graph as an undirected graph. The nodes are artists. There is an edge between artists if they are similar. The weight on the edge is the commonality between the artists, then cluster all artists using k-means in Spark, using the trendsetters as initial centroids and some definition of distance in the commonality graph as distance measure.

System Specification

Local

OS: Mac OS Sierra 10.12.6

Java Version: 1.8

RAM: 8Gb

Disk: 256 Gb Solid State Drive

Processor: 2.3GHz Intel i5 Kaby Lake Processor

Scala Version : 2.11.8

Spark Version : 2.2.0

AWS

The Amazon Map Reduce cluster was setup using Amazon EMR.

Instance Type: m3.xlarge

Memory: 15GB

Storage: 2 x 40 GB SSD

vCPU: 4

Subproblem 1 : Clustering

In this section, we present the findings obtained by implementing & running KMeans clustering and Hierarchical Agglomerative Clustering on Million Song Dataset, to obtain Fuzzy Loudness, Fuzzy Length, Fuzzy Tempo, Fuzzy Hotness and Combined Hotness.

In the following section the implementations of KMeans clustering algorithm and Hierarchical Agglomerative Clustering algorithm are described. After that, we discuss the results.

KMeans Clustering Implementation

Our implementation of KMeans clustering can be summerized as follows:

- **Assign Initial centroids** Since, in all of the tasks we're interested in, the number of clusters is fixed. Therefore, we begin by selecting 3 centroids initially.
    - For Combined Hotness we randomly sample 3 points from the datase to act as our initial centroids
    - For one dimensional data (i.e. Fuzzy Loudness, Fuzzy Length, etc.) we choose these centroids more systematically by selecting `Max`, `Min` and `Mean` of the data points as our initial centroids. Since, KMeans clustering is sensitive to initial centroids, we believe this approach is helpful in this case.
- **Label data points**
    - For the first iteration, we use the initial centroids and calculate the distance from each data point to each centroid. The centroid that is closest to a given data point is assigned as that data point's cluster. Here, the distance measure is euclidean distance.
    - In later iterations, similar approach is used but the distances are calculated with respect to updated centroids.
- **Recalculate Centroids**
    - After labelling data points with cluster IDs, centroids are recalculated by taking the mean of each data point in a cluster.

We run KMeans algorithm for *10 iterations*.

Hierarchical Agglomerative Clustering Implementation

Our implementation of Hierarchical Agglomerative Clustering consists of following steps:

- **Calculate Distance Matrix** First we calculate the distance matrix by calculating distance between each possible pair in the dataset
- **Find Closest Pair & Update Matrix** Then, we find the closest pair in the distance matrix and update the matrix as follows:
    - We add new entries in the distance matrix from the merged cluster by taking the minimum distance. For example, if cluster A and B form the closest pair, then we calculate distance of any point x from pair (A, B), `d({A, B}, {X})` as `min(d{A, X}, d{B, X})`, where `d` denotes the distance between two points. The distance matrix is euclidean distance.
    - The entries in the distance matrix for the closest pair are deleted. For example, entries from `A` and `B` to each point is deleted and they are replaced by a single entry `AB`.
- This above two steps are run iteratively for `n - 3` iterations, where `n = Number of elements in the dataset`.

Results

In this section we discuss the results obtained by KMeans clustering and Agglomerative clustering. We first look at *KMeans clustering* results for both small dataset and big dataset. In the next section, we discuss some of the challenges we faced while obtaining results for Agglomerative Clustering.

Fuzzy loudness

- Aim: To cluster songs into quiet, medium, and loud

For this task, we cluster the loudness using KMeans clustering algorithm. As described above, we take `Min`, `Max` and `Mean` values of Loudness as initial centroids. The results generated are visualized for both small and big dataset.

In the following graphs, the points are colored based on what cluster they belong to. Loudness is on Y-axis and since this is one dimensional data we keep the cluster ID as X-axis.

Figure 1.1. Fuzzy Loudness: Small Dataset

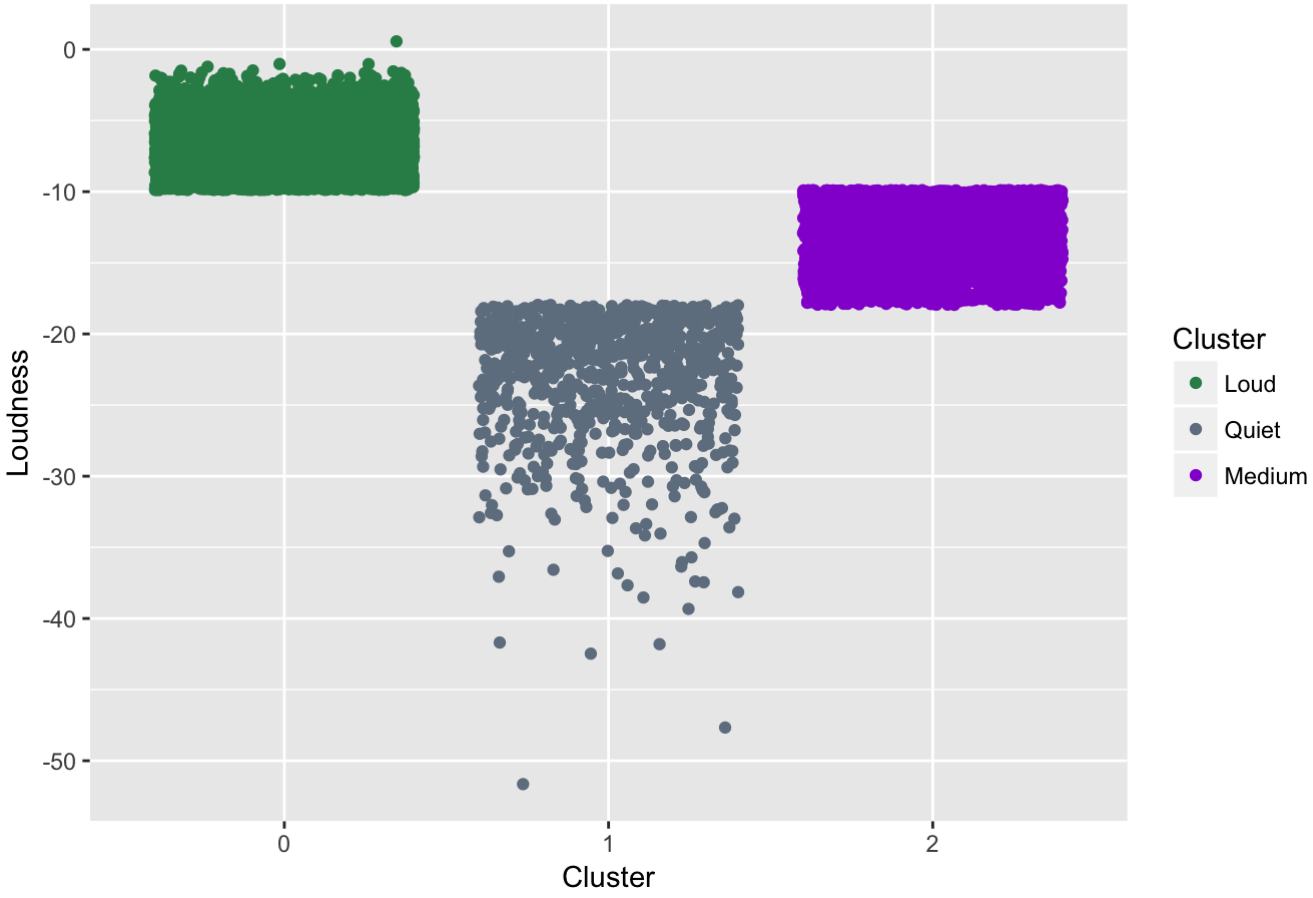
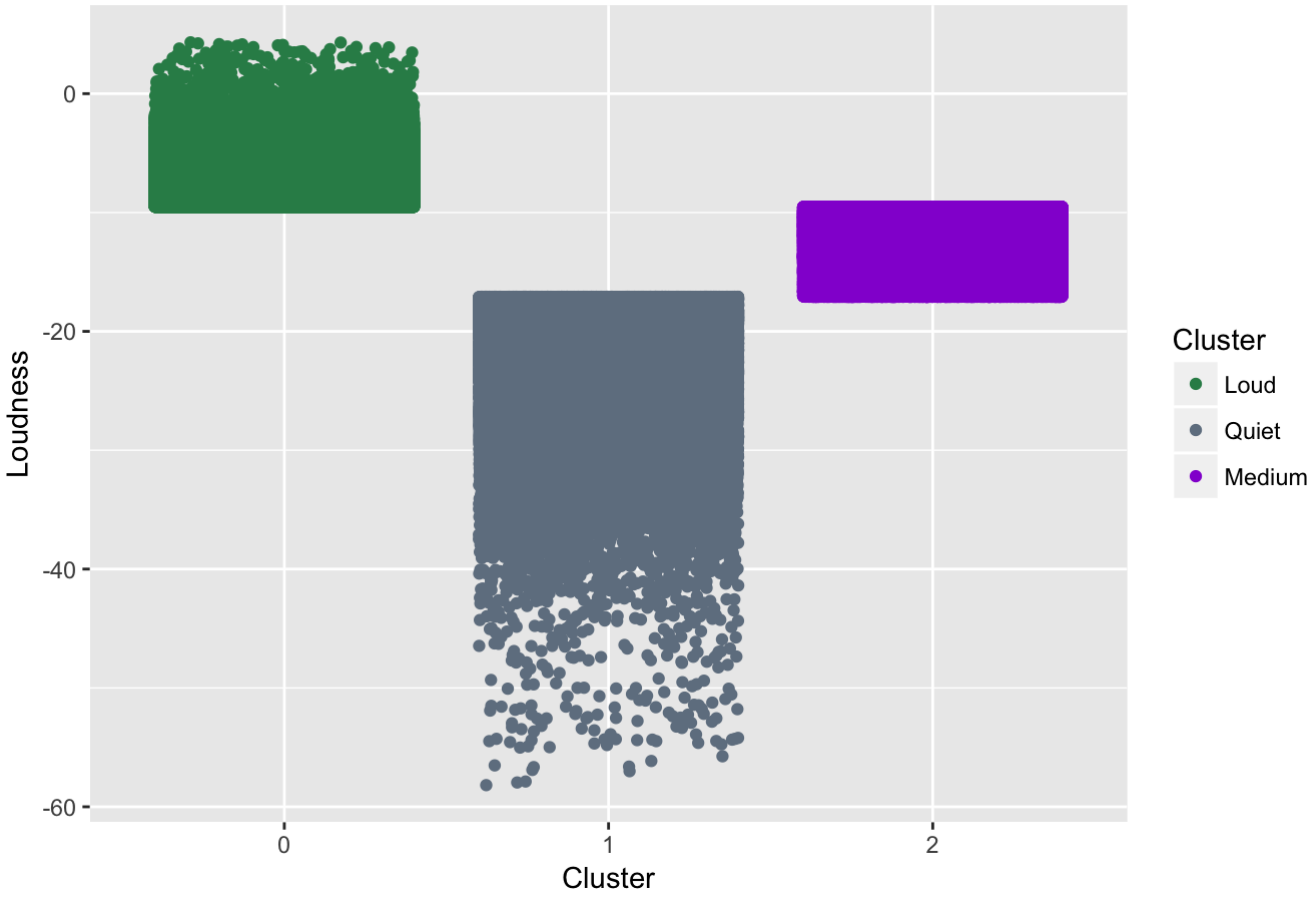


Figure 1.2. Fuzzy Loudness: Big Dataset



It is observed that the data points belonging to the same loudness range appear in the same cluster in a Loudness vs Cluster graph, for the most part. But, for both datasets the loudness values near -20 db seems to be overlapping. There appears multiple points having similar loudness but they appear in different clusters. This might be because, of less (in our case, 10) number of iterations.

Figure 2. Distribution

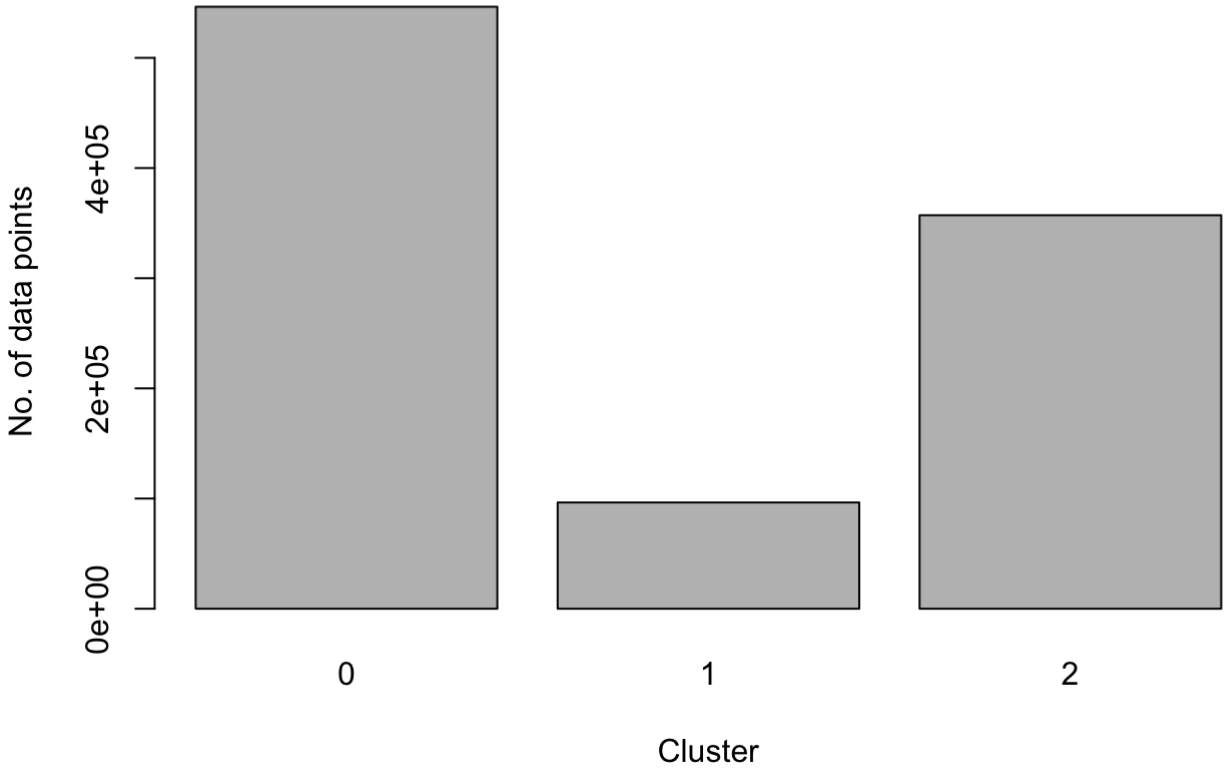


Figure-2

It is observed that maximum number of points are in cluster 0 indicating that most songs are `loud` followed by cluster 2, which represents `medium` cluster.

Cluster	Mean Loudness	Standard Deviation
0	-6.523788	1.775146
1	-21.633129	4.396574
2	-12.524099	2.057522

Mean loudness of clusters are shown in the above table. Cluster 1 shows maximum standard deviation indicating that the values are spreadout and farther away from the mean. Cluster 0 has minimum standard deviation indicating that values are close to the mean.Both the facts,Mean and Standard deviation, can also be clearly seen in Figure 2.

Fuzzy length

- Aim: To cluster songs into short, medium, and long

For this task, we cluster the length of songs using KMeans clustering algorithm. As described above, we take `Min` , `Max` and `Mean` values of length as initial centroids. The results generated are visualized for both small and big dataset.

As can be seen in `Figure 3.1` and `Figure 3.2`, the `short` and `medium` clusters are very dense, where as, the `long` cluster is more scattered. This shows that there are very few long songs in the dataset. Most of the songs seem to be either medium or short.

Also note that there are a few values around length 0, this is because while processing data we replace `NA` values with 0.

Figure 3.1. Fuzzy Length: Small Dataset

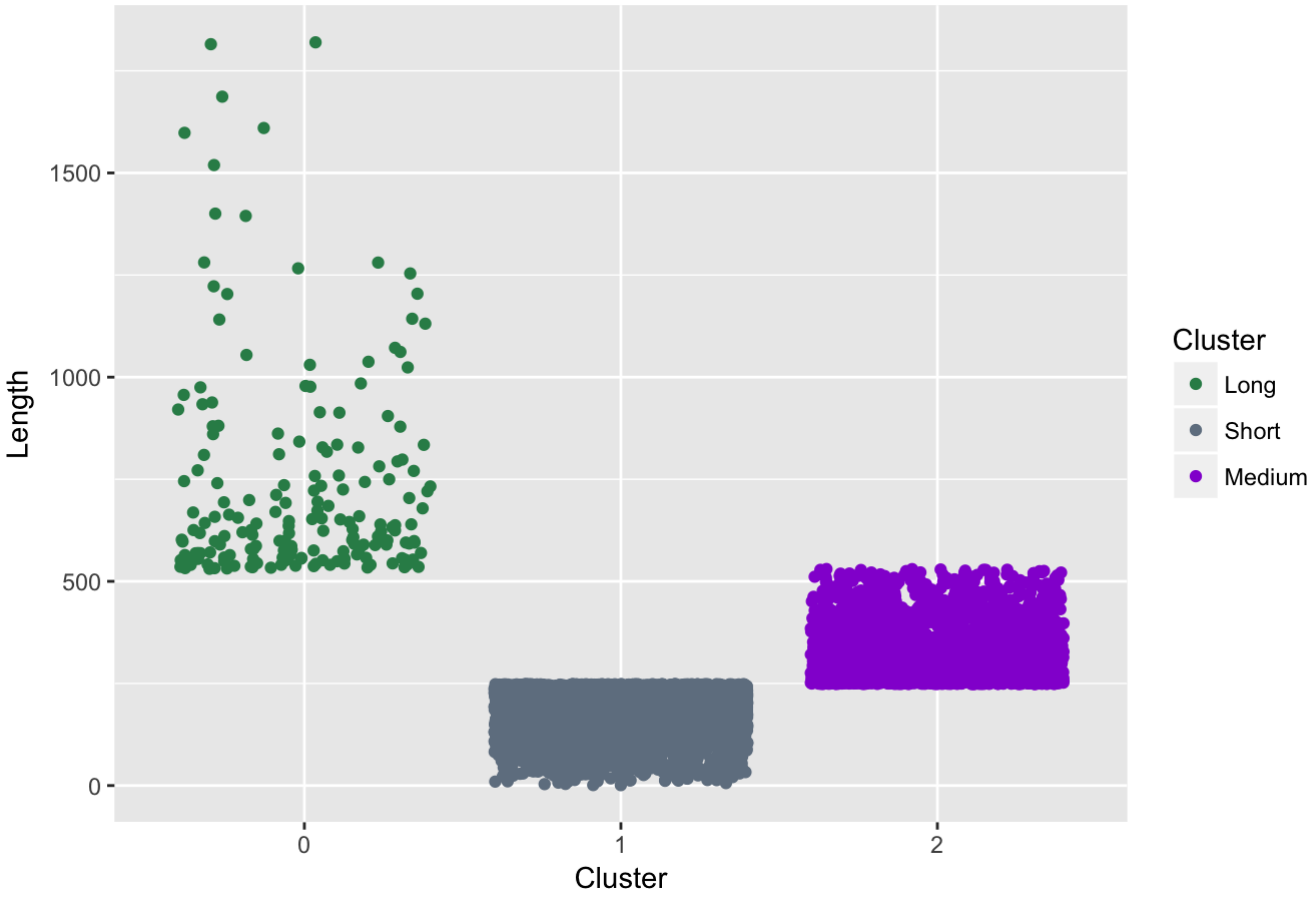
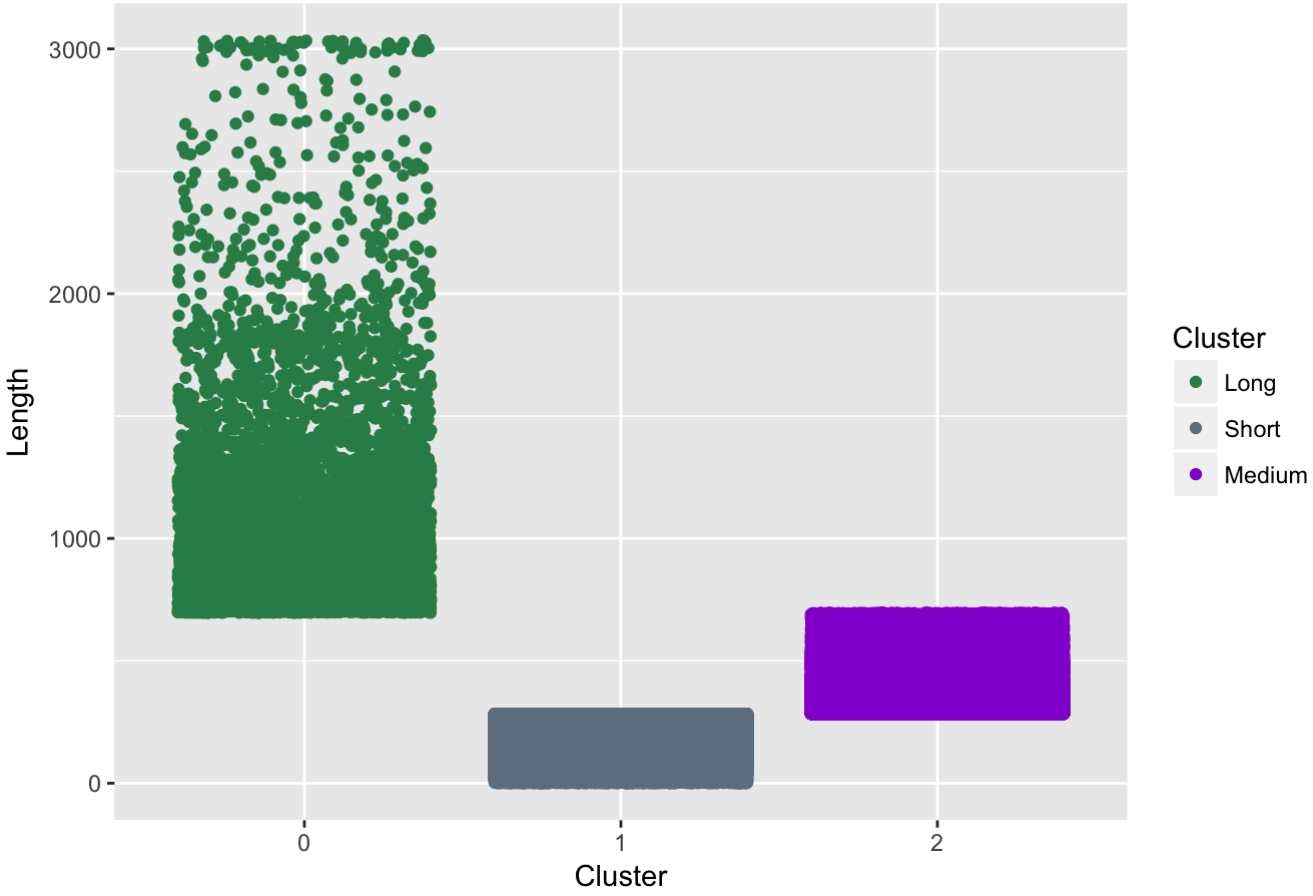
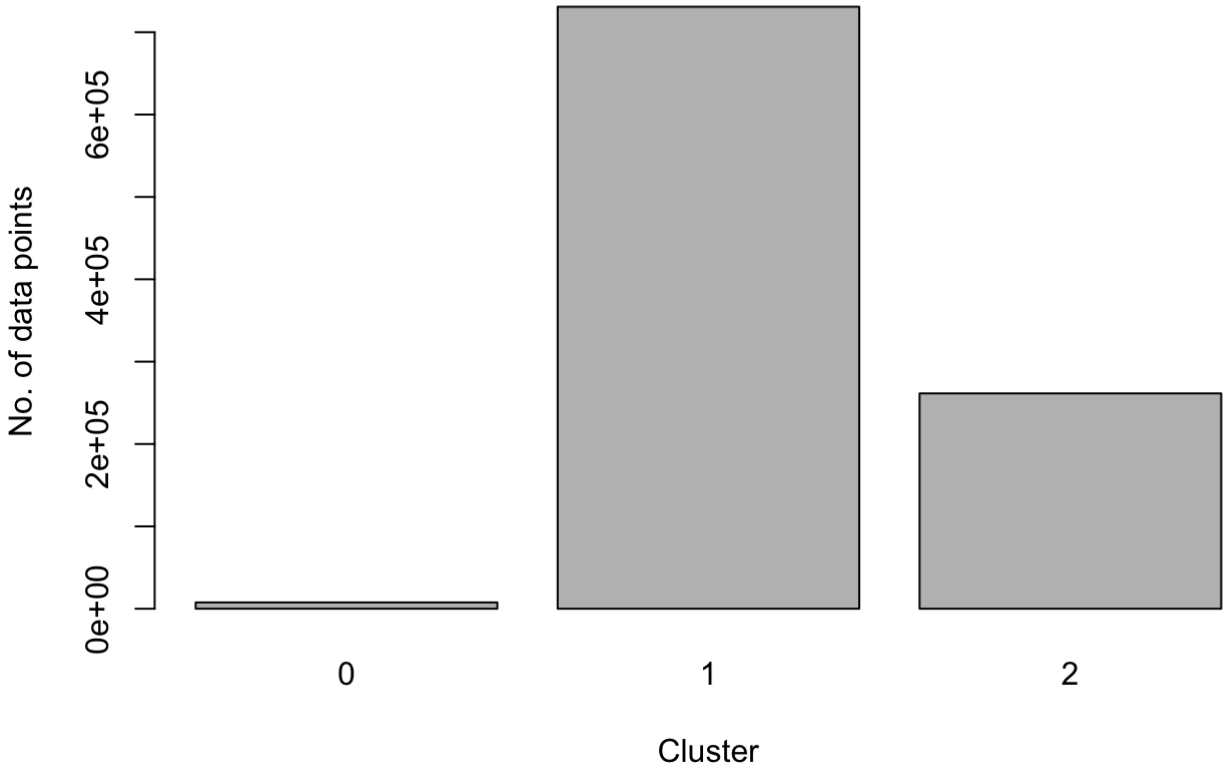


Figure 3.2. Fuzzy Length: Big Dataset



It is observed that the data points belonging to the same length range appear in the same cluster in a length vs Cluster graph, validating the fact that the clustering has been exceuted properly.

Figure 4 Distribution



It is observed that maximum number of points are in cluster 1 indicating that most songs are short followed by cluster 2 - medium.

```
d1<-aggregate(lvalues$V2, by=list(Cluster=lvalues$V1), FUN=mean)
d2<-aggregate(lvalues$V2, by=list(Cluster=lvalues$V1), FUN=sd)
d1$sd<-d2$x
colnames(d1)<-c("Cluster","Mean Length","Standard Deviation")
library(formattable)
formattable(d1)
```

Cluster	Mean Length	Standard Deviation
0	1020.0029	403.15582
1	196.6269	54.82508
2	374.7408	82.98670

Mean length of clusters are shown in the above table. Cluster 0(long) shows maximum standard deviation indicating that the values are spreadout and farther away from the mean. Cluster 1 has minimum standard deviation indicating that values are close to the mean. Both the facts, Mean and Standard deviation, can also be clearly seen in `Figure 3.2`.

Fuzzy Tempo

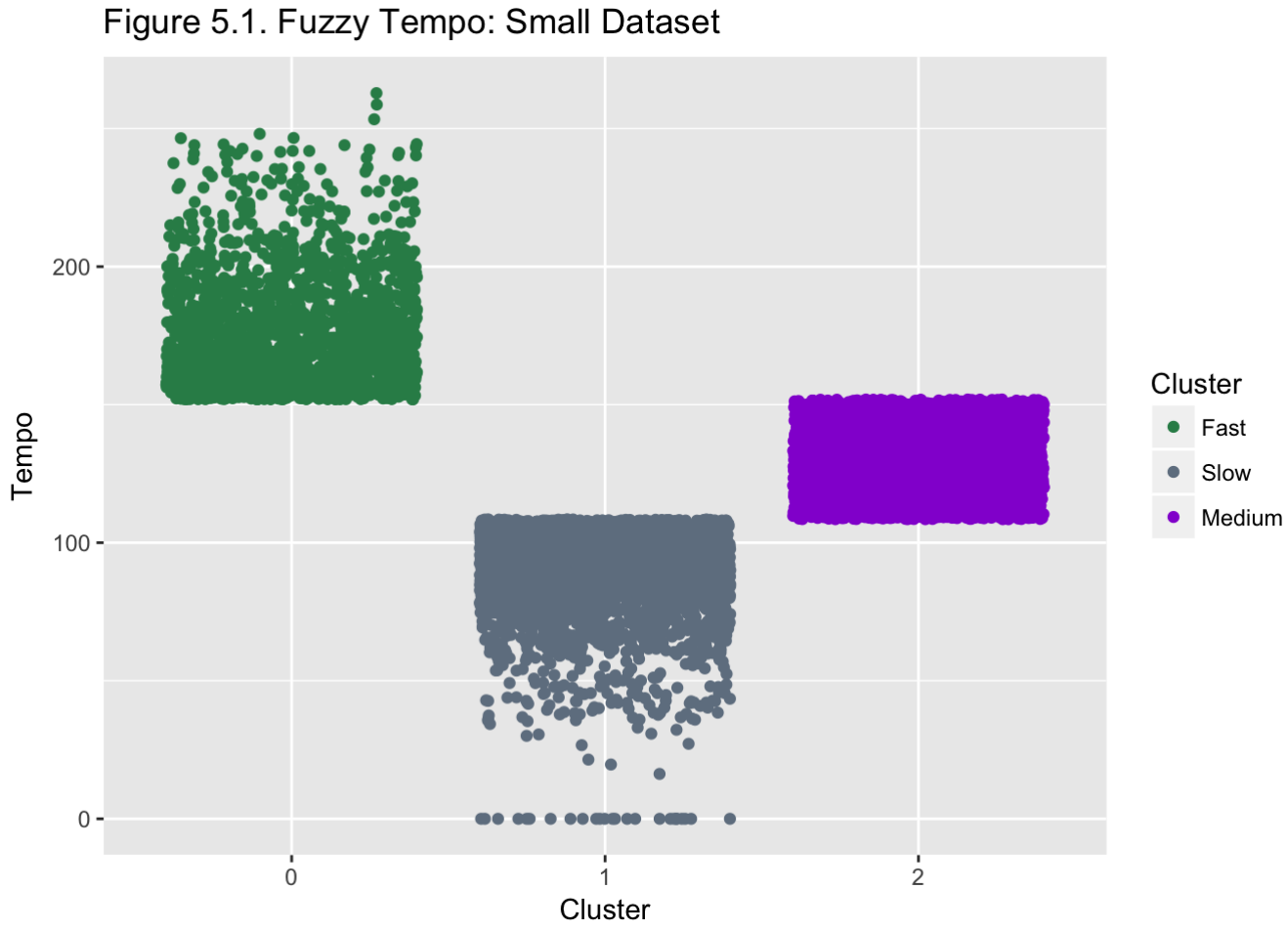
- Aim: To cluster songs into slow, medium, and fast

For this task, we cluster the tempo of songs using KMeans clustering algorithm. As described above, we take `Min`, `Max` and `Mean` values of length as initial centroids. The results generated are visualized for both small and big dataset.

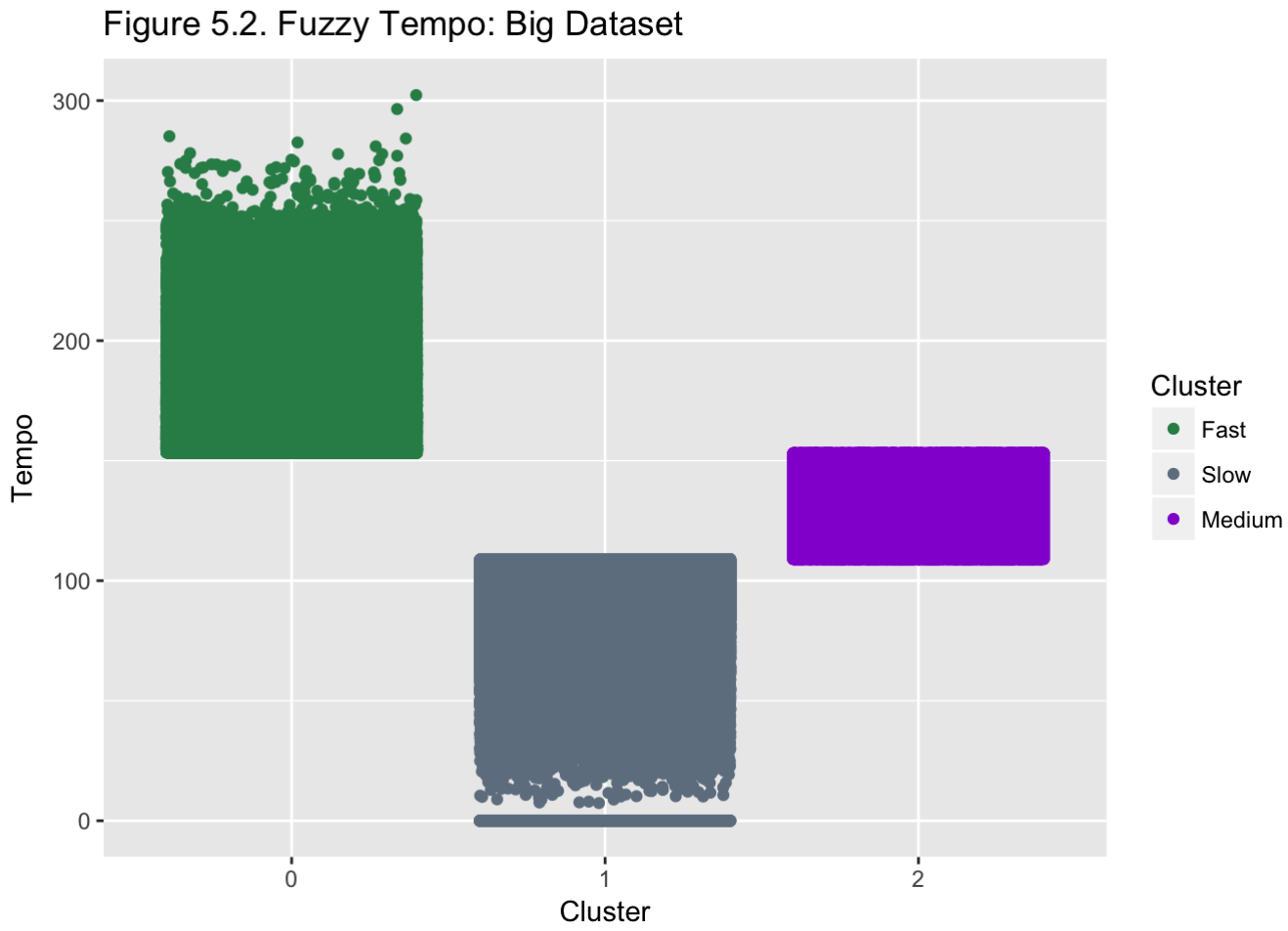
In both `Figure 5.1` and `Figure 5.2`, the clusters for fast songs and slow songs seem to be having a range where the cluster is very dense. For example, the values between tempo 150 to 200 have a lot of values clustered for `Fast` cluster . But above 200, the values are scattered. This indicates that there are very few songs that are either fast or slow.

As pointed before, we see a lot of 0 values, this time a bit more than before. This shows that there were many records for the Tempo data that had `NA` values in the data. Since the data seems to have very dense values in the clusters, we keep the 0 values as is. But keeping 0 values in the dataset might change cluster results for different shapes of data.

```
OUTPUT_PATH = "output/kmeans-tempo/part-00000"
tvalues = read.csv(OUTPUT_PATH, header = FALSE)
tvalues$V1 = as.factor(tvalues$V1)
p<- ggplot(tvalues, aes(x=V1, y=V2)) +
  geom_jitter(aes(col = tvalues$V1))
p+ labs(x = "Cluster",y="Tempo",col = "Cluster", title="Figure 5.1. Fuzzy Tempo: Small D
ataset")+ scale_color_manual(labels=c("Fast","Slow","Medium"),values=c("seagreen4","slat
egrey","darkviolet"))
```



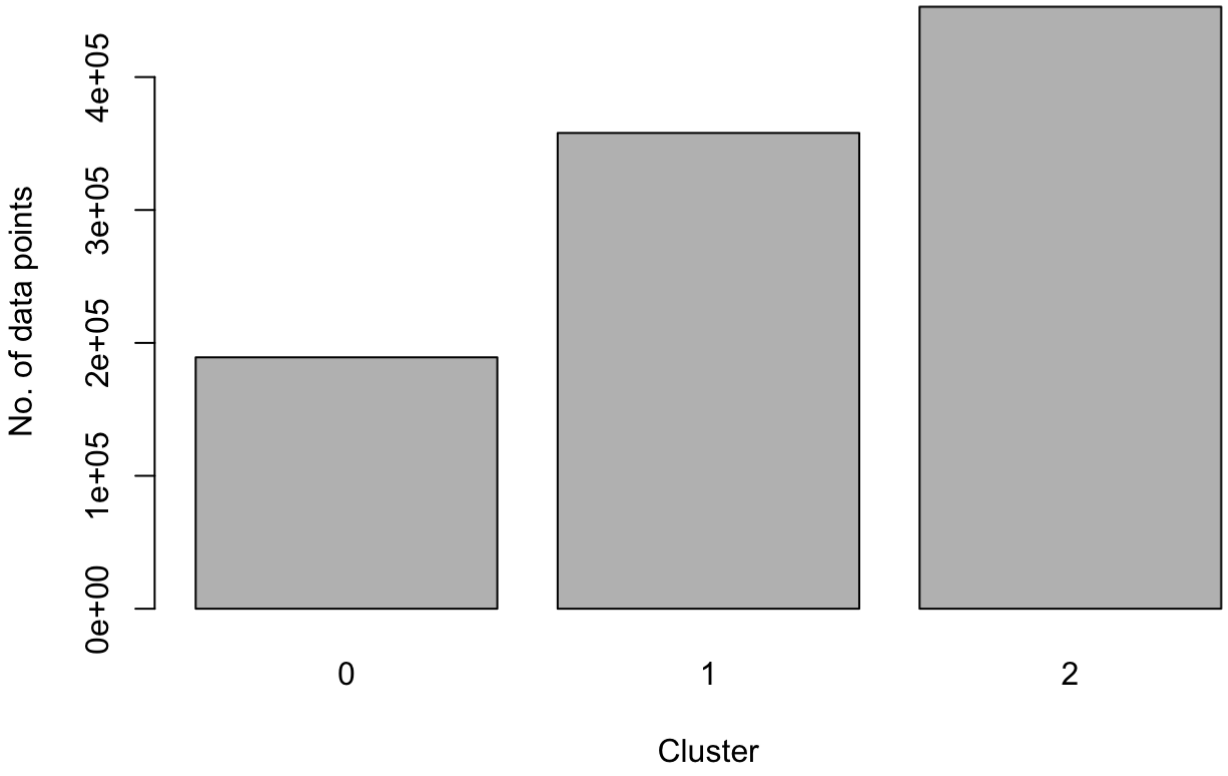
```
OUTPUT_PATH = "output/full/kmeans-tempo/part-00000"
tvalues = read.csv(OUTPUT_PATH, header = FALSE)
tvalues$V1 = as.factor(tvalues$V1)
p<- ggplot(tvalues, aes(x=V1, y=V2)) +
  geom_jitter(aes(col = tvalues$V1))
p+ labs(x = "Cluster",y="Tempo",col = "Cluster", title="Figure 5.2. Fuzzy Tempo: Big Dat
aset")+ scale_color_manual(labels=c("Fast","Slow","Medium"),values=c("seagreen4","slateg
rey","darkviolet"))
```



It is observed that the data points belonging to the same tempo range appear in the same cluster in a tempo vs Cluster graph, validating the fact that the clustering has been exceuted properly.

```
counts <- table(tvalues$V1)
barplot(counts, main="Figure 6. Distribution",
    xlab="Cluster", ylab="No. of data points")
```

Figure 6. Distribution



It is observed that maximum number of points are in cluster 2 indicating that most songs are medium tempo followed by cluster 1 - slow.

```
d1<-aggregate(tvalues$V2, by=list(Cluster=tvalues$V1), FUN=mean)
d2<-aggregate(tvalues$V2, by=list(Cluster=tvalues$V1), FUN=sd)
d1$sd<-d2$x
colnames(d1)<-c("Cluster","Mean Tempo","Standard Deviation")
library(formattable)
formattable(d1)
```

Cluster	Mean Tempo	Standard Deviation
0	177.38748	20.41034
1	88.86622	16.30811
2	129.22764	11.66267

Mean tempo of clusters is shown in the above table. Cluster 0(fast) shows maximum standard deviation indicating that the values are spreadout and farther away from the mean. Cluster 2 - medium has minimum standard deviation indicating that values are close to the mean.

Fuzzy hotness

- Aim: To cluster songs into cool, mild, and hot based on song hotness

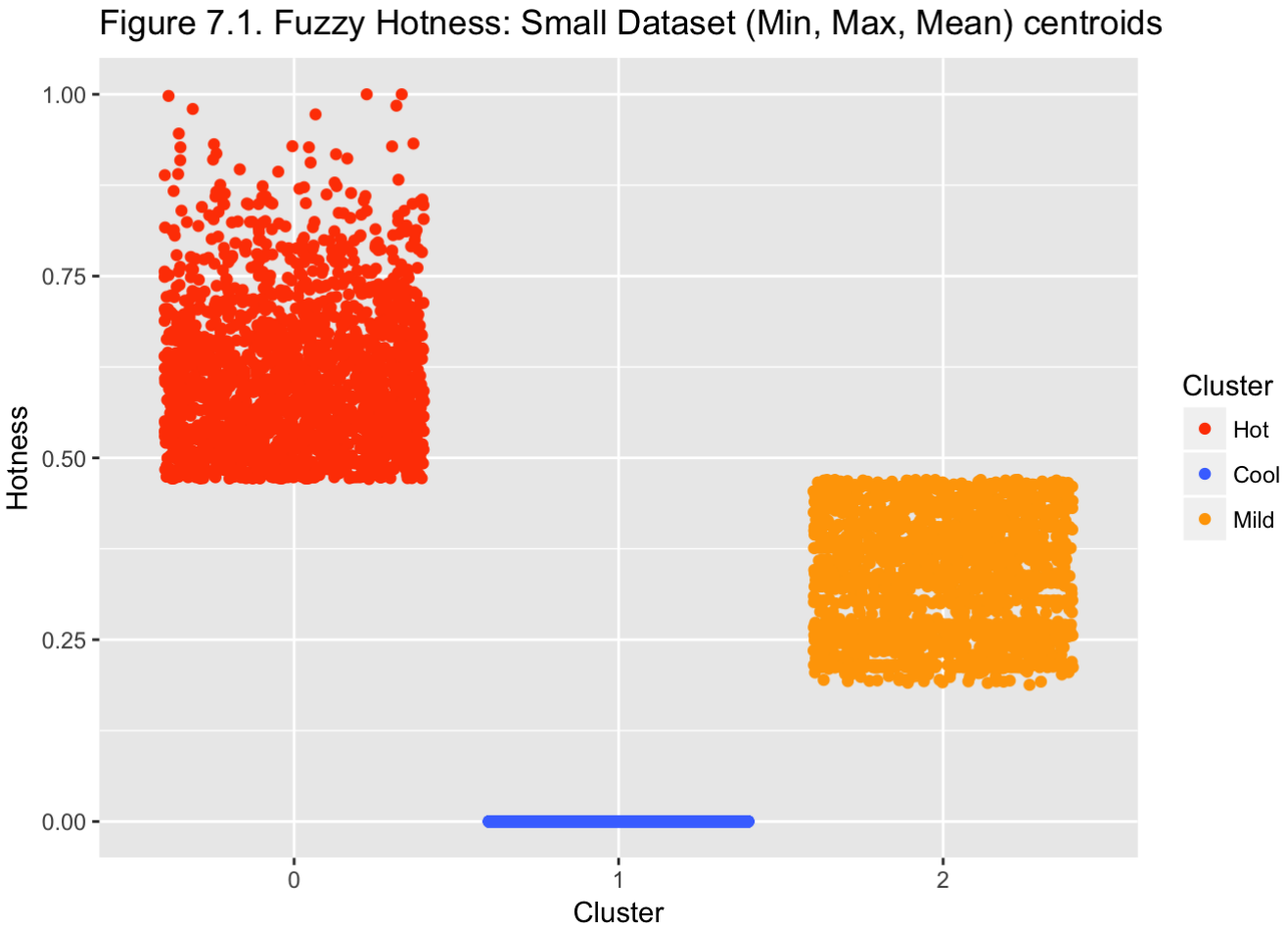
For this task, we cluster songs by their hotness using KMeans clustering algorithm. As described above, we take `Min`, `Max` and `Mean` values of length as initial centroids. The results generated are visualized for both small and big dataset.

Ignoring the 0 values clustered in `Cool`, which are noise value introduced while converting `NA` values to 0, we see that the remaining points perfectly split in two clusters. This could happen due to the way we run KMeans clustering or it could be because of the characteristics of the data.

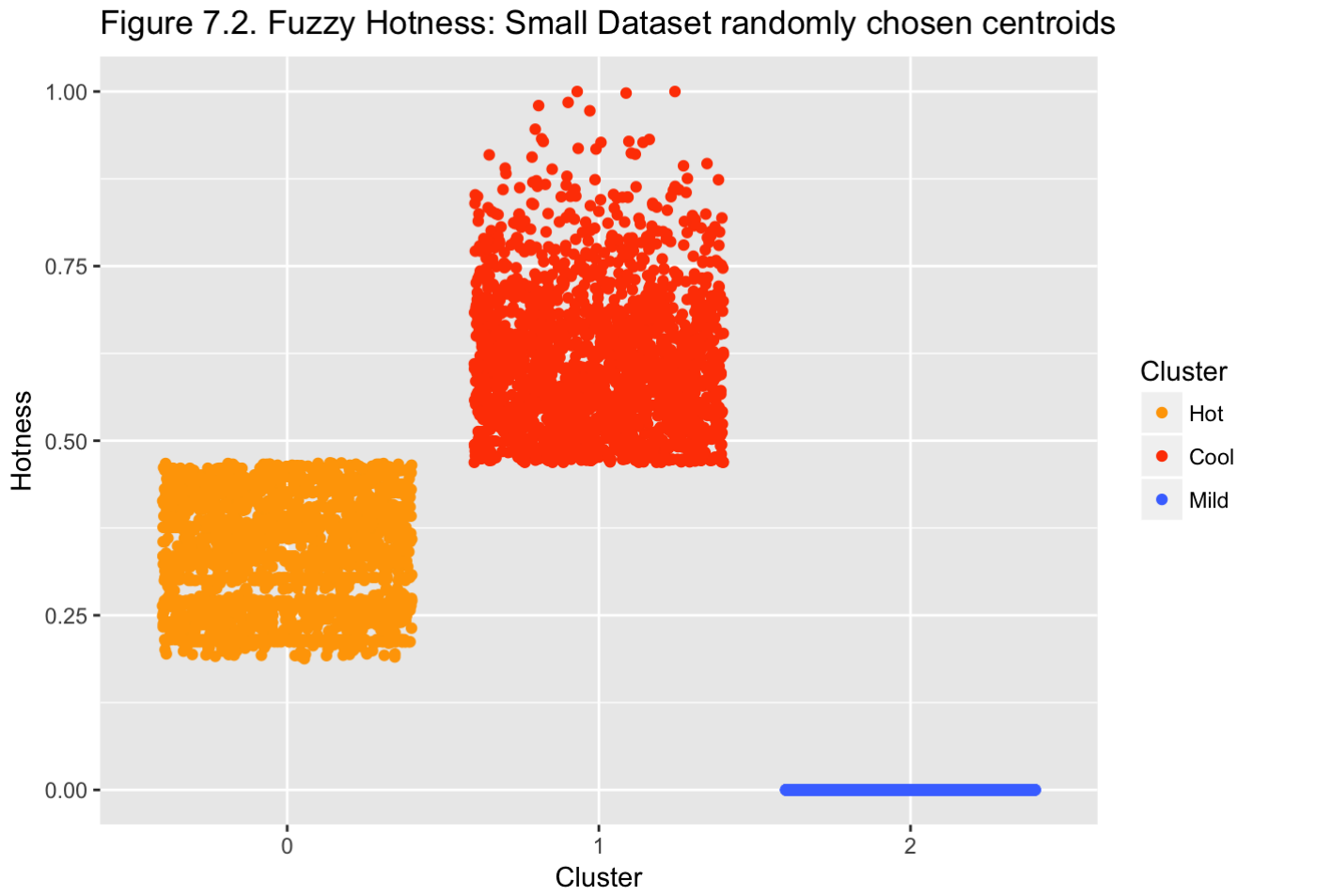
To test whether our implementation doesn't affect the outcome of clustering, instead of having Min, Max and Mean of song hotness as initial centroids, we take random samples as initial centroids. The clusters generated by this approach are visualized in `Figure 7.2`.

Firstly, note that the cluster IDs are different in `Figure 7.2`. That is because the centroids were chosen randomly. Secondly, the data still partitions into two clusters. Which means, our initial approach works correctly and the clustering behavior can be attribute to the dataset.

```
OUTPUT_PATH = "output/kmeans-hotness/part-00000"
hvalues = read.csv(OUTPUT_PATH, header = FALSE)
hvalues$V1 = as.factor(hvalues$V1)
p<- ggplot(hvalues, aes(x=V1, y=V2)) +
  geom_jitter(aes(col = hvalues$V1))
p+ labs(x = "Cluster",y="Hotness",col = "Cluster", title="Figure 7.1. Fuzzy Hotness: Sma
ll Dataset (Min, Max, Mean) centroids")+ scale_color_manual(labels=c("Hot","Cool","Mild"
),values=c("orangered1","royalblue1","orange1"))
```



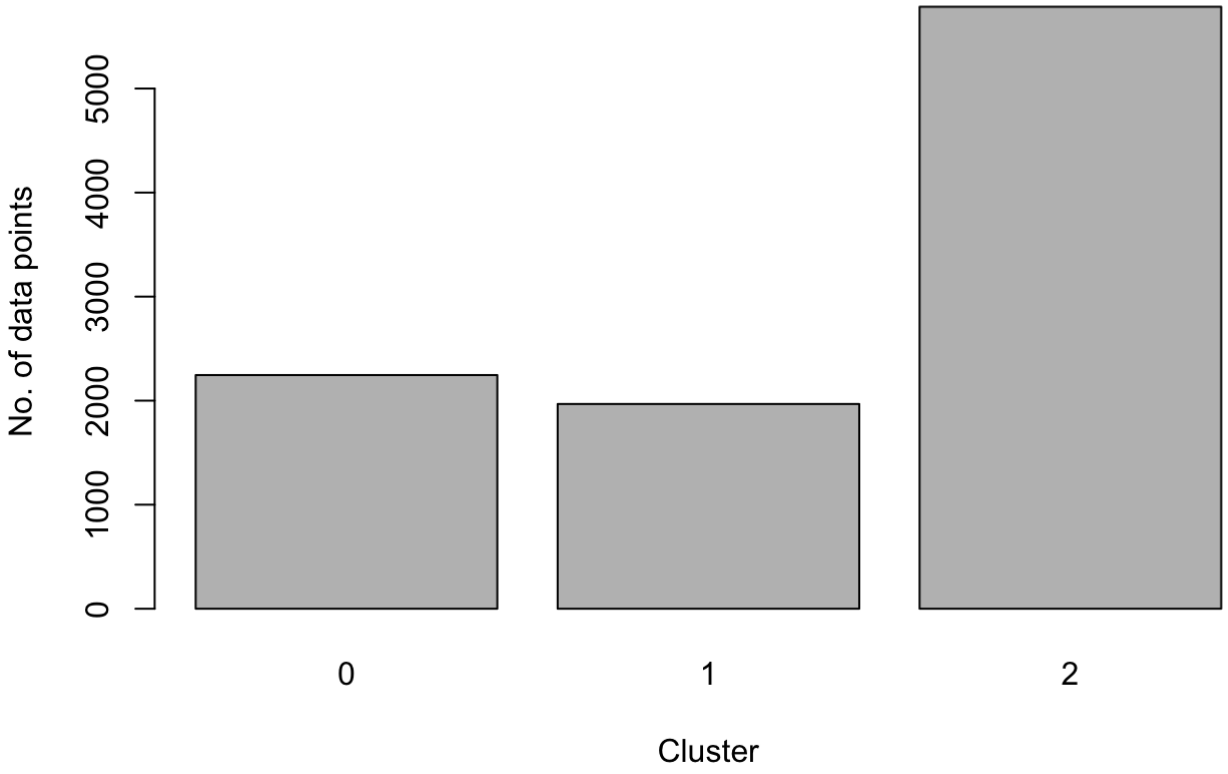
```
OUTPUT_PATH = "output/kmeans-hotness-random/part-00000"
hvalues = read.csv(OUTPUT_PATH, header = FALSE)
hvalues$V1 = as.factor(hvalues$V1)
p<- ggplot(hvalues, aes(x=V1, y=V2)) +
  geom_jitter(aes(col = hvalues$V1))
p+ labs(x = "Cluster",y="Hotness",col = "Cluster", title="Figure 7.2. Fuzzy Hotness: Sma
ll Dataset randomly chosen centroids")+ scale_color_manual(labels=c("Hot","Cool","Mild")
,values=c("orange1","orangered1","royalblue1"))
```



It is observed that the data points belonging to the same hotness range appear in the same cluster in a hotness vs Cluster graph, validating the fact that the clustering has been exceuted properly.

```
counts <- table(hvalues$V1)
barplot(counts, main="Figure 8. Distribution",
    xlab="Cluster", ylab="No. of data points")
```

Figure 8. Distribution



It is observed that maximum number of points are in cluster 1 indicating that most songs are cool followed by cluster 2 - mild.

```
d1<-aggregate(hvalues$V2, by=list(Cluster=hvalues$V1), FUN=mean)
d2<-aggregate(hvalues$V2, by=list(Cluster=hvalues$V1), FUN=sd)
d1$sd<-d2$x
colnames(d1)<-c("Cluster","Mean Hotness","Standard Deviation")
library(formattable)
formattable(d1)
```

Cluster	Mean Hotness	Standard Deviation
0	0.3265712	0.07966564
1	0.6111677	0.10094007
2	0.0000000	0.00000000

Mean hotness of clusters are shown in the above table.Cluster 0(hot) shows maximum standard deviation indicating that the values are spreadout and farther away from the mean. Cluster 1 - cool has minimum standard deviation indicating that values are close to the mean.

Fuzzy combined hotness

- Aim: To cluster songs into cool, mild, and hot based on two dimensions: artist hotness, and song hotness

For this task, we cluster songs by their hotness and their artists' hotness using KMeans clustering algorithm. All the previous taks only performed KMeans algorithm on a one dimensional data. In this task, two dimensions are used. Instead of calculating `Min`, `Max` and `Mean` values to initialize the centroids, we randomly take sample data from the dataset and initialize the centroids.

`Figure 9.1` and `Figure 9.2` show the clusters in Small Dataset and Big Dataset respectively.

Figure 9.1. Combined Hotness: Small Dataset

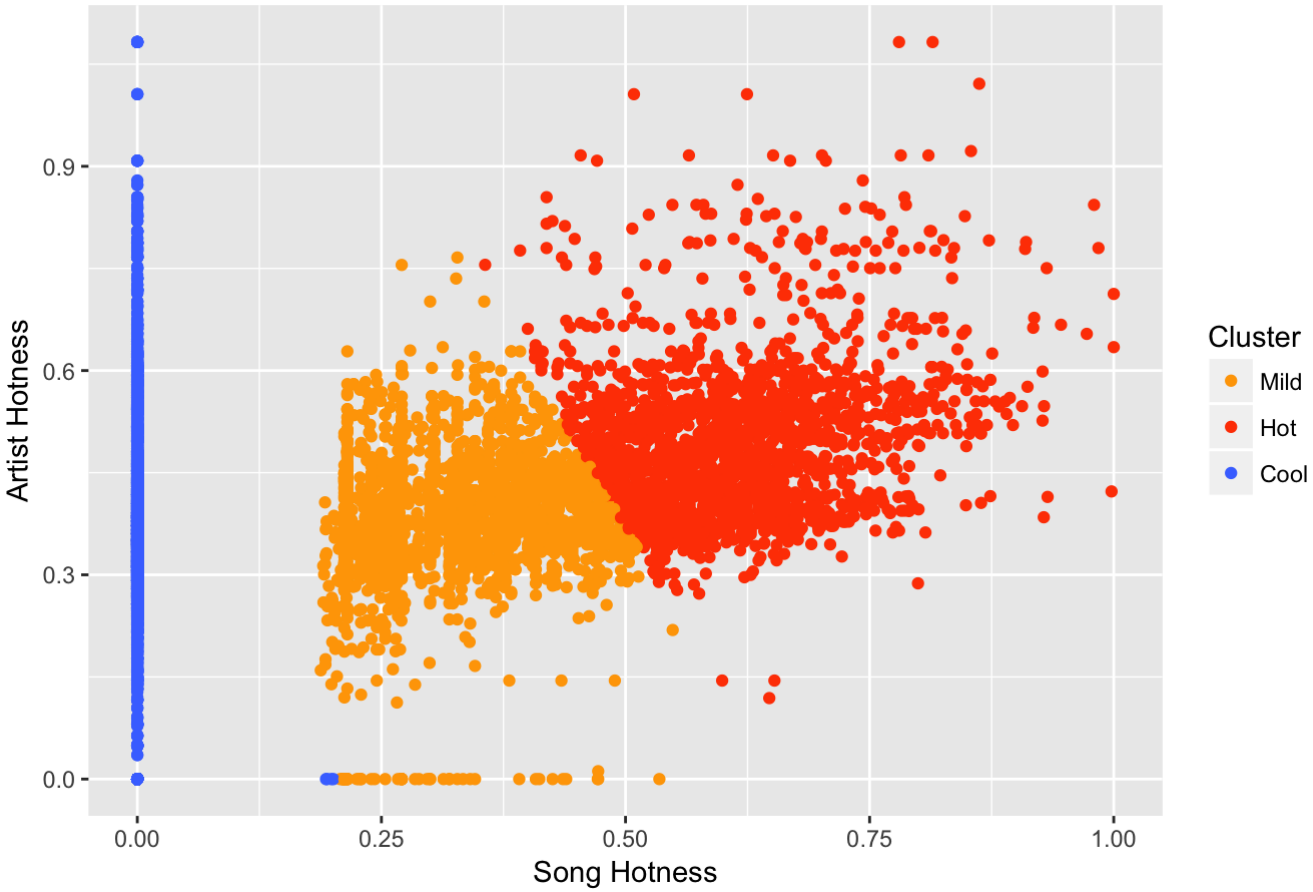


Figure 9.2. Combined Hotness: Big Dataset

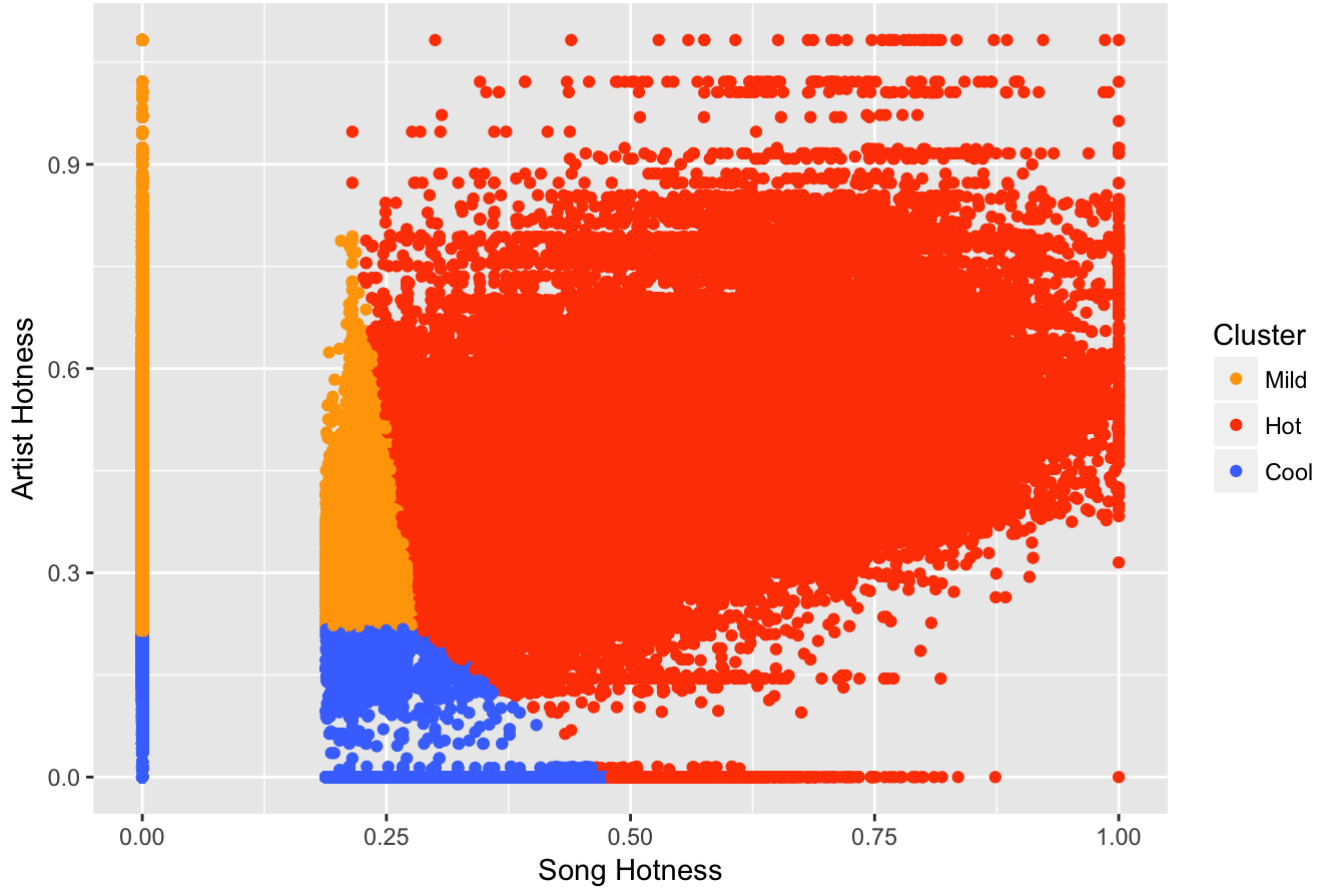
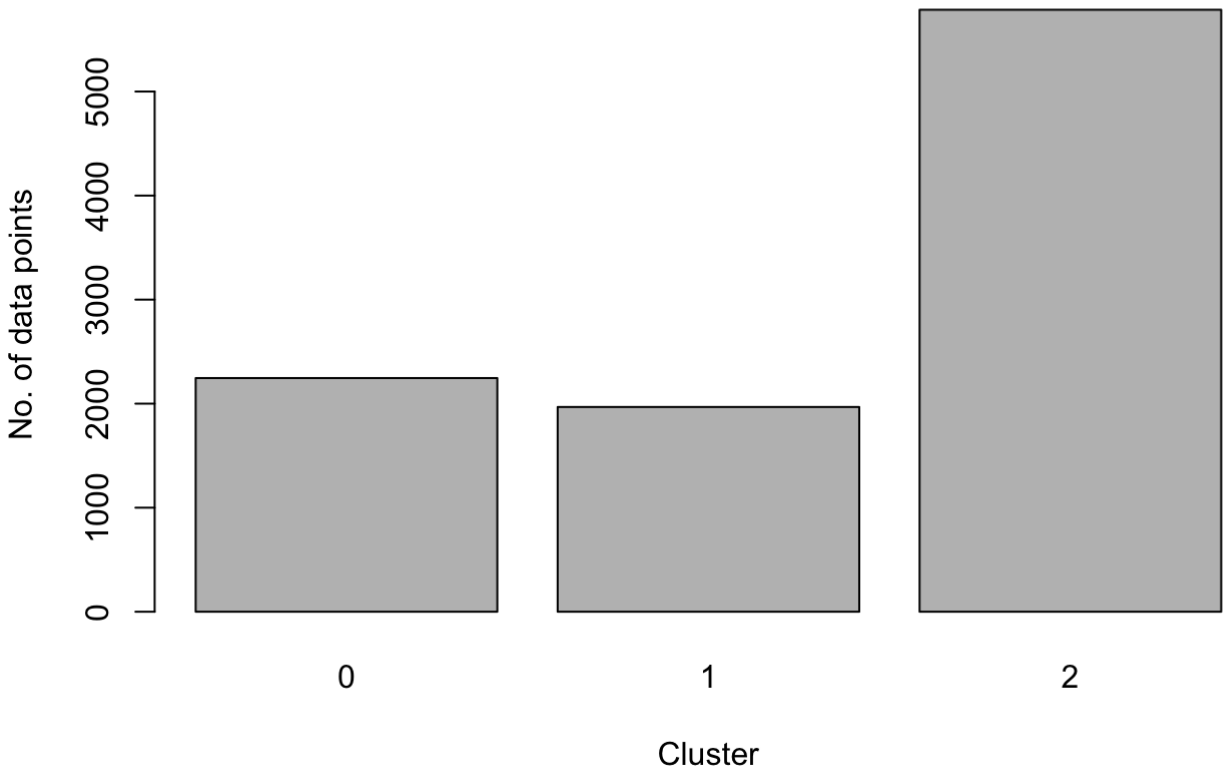


Figure 10. Distribution



Cluster	Mean Combined Hotness	Standard Deviation
0	0.02582084	0.07321603
1	0.49279587	0.14266922
2	0.04218886	0.10411512

KMeans Performance

- **Local Execution Time** We ran all the KMeans clustering jobs on local machine using the small dataset, which took about **10 seconds**.
- **Time of execution on AWS** We ran all the KMenas jobs on AWS using the Big Dataset, which took **6 minutes** on a single node.

We think that this performance is reasonable and we were able to verify the results by visualizing them. Compared to Agglomerative clustering, KMeans performs significantly faster due to its top-down approach. We discuss more on Agglomerative Clustering performance in the next section.

Hierarchical Agglomerative Clustering Results

So far, we have looked at the results for KMeans clustering for all the tasks. Now, we discuss the lack of results from Agglomerative Clustering.

We failed to obtain any presentable results with Agglomerative Clustering. Here are a few things we tried:

- We tried running our Agglomerative Clustering implementation on the small dataset (10K songs) on local machine, which resulted in *Java Heap Space error* with default configuration.
- Next we tried running the algorithm on `1000` records and `100` records respectively. Fortunately, these runs didn't result in heap space errors, but the runtimes were really slow and we couldn't finish both runs completely.
- We tried runningn with `1000` records and `100` records on AWS. With `1000` records, it took about `4 hours` to finish a run, and with `100` records, it took about 40 minutes.

Here's a few reasons due to which the runtime is too slow for Agglomerative Clustering:

- As described in the implementation section, for Agglomerative clustering we create a distance matrix of size `n X n`, where `n` is the number of datapoints we want to consider. For example, for a file containing 10000 songs, the number of distance pairs to be maintained will be `10M`. On top of this, in each iteration, we filter values from this matrix and make changes to it.
- Moreover, this method merges only two clusters in to one in each iteration, so the convergence times can be very slow compared to KMeans algorithm.

Subproblem 2 : Graphs

In this task, we cluster artists using common terms shared between them. For this, we use KMeans clustering.

Approach

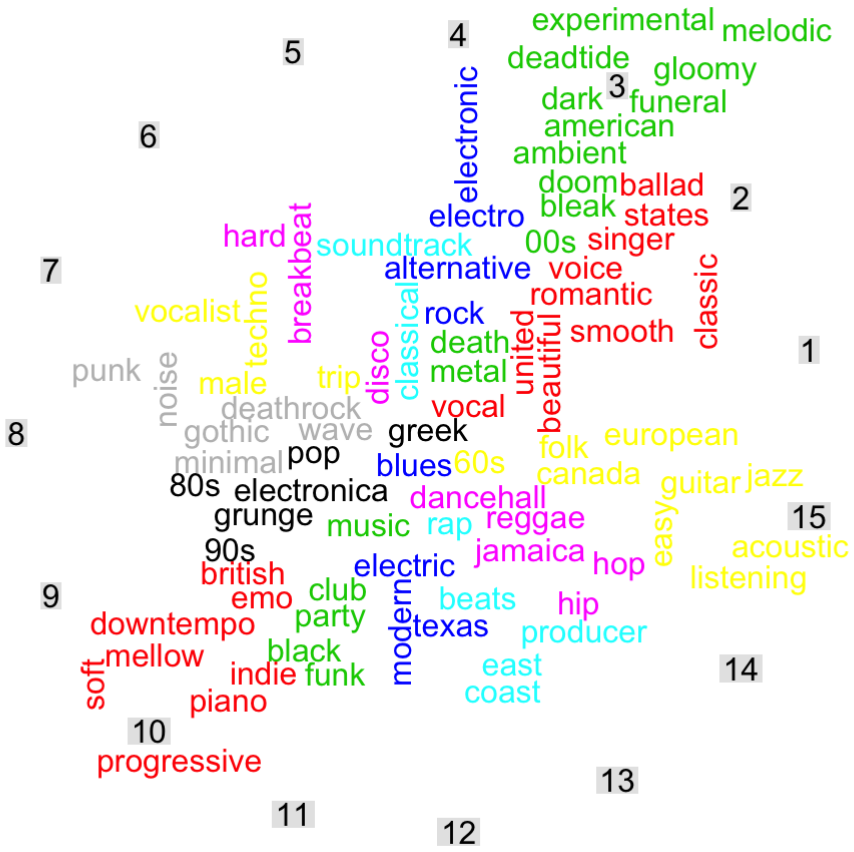
- **Initial Centroids**
  - As Initial Centroids, we take the artist terms of top 30 most popular artists (Trendsetters)
- **Cluster Datapoints**

    - We assign clusters to each artist in the dataset by calculating common terms between each artist and each centroid. The cluster is assigned based on the maximum number of common terms.
- **Recalculate Centroids**
- Centroids are recalculated by finding the common artist terms within the cluster
- We run KMeans for 10 iterations

The initial centroids and centroids after 10 iterations are visualized below by plotting a word cloud. Note that the numbers in the image shows the ID for the cluster and the surrounding values depict the terms inside that centroid.

Below is the visualization for centroids initially, which represents the artist terms for the top 30 popular artists. For the lack of space, we only represent 15 of these centroids. In the beginning, the centroids seem to be containing a lot of terms.

Below is the visualization for centroids after 10 iterations. After running Kmeans clustering for 10 iterations, we can see that the centroids have become more specific than the initial centroids and they only represent specific terms. Again, for the lack of space, we only plot 15 centroids out of 30.



Conclusion

Spark makes it easier to write iterative jobs that maintain state across iterations. By examining the findings from running clustering using KMeans Algorithm and Agglomerative Clustering, we can conclude that when the number of clusters is known beforehand, KMeans is a better clustering approach. Since, it runs significantly faster than Agglomerative Clustering.