# K-Means and Locality Sensitive Hashing for Song Clustering

Harish Mohan
Priyanka Narendra
Ramya Lakshminarasimha
University of Southern California

## 1 ABSTRACT

This project aims to cluster songs based on song lyrics using K-means clustering. The first part of the project aims to perform shingling and generate signatures for the shingles. The second part is to utilize the signatures as song representatives and cluster them based on K-means clustering. We use Jaccard similarity as distance metric between song data points and perform LSH to restrict the comparisons.

## 2 INTRODUCTION

Music taste is a very unique, peculiar and characteristic quality of a person. Of all the millions of songs and sounds that exist, the fact that people decide to develop a liking for a particular genre of music is not by mere chance.

We want to identify significant relationships between songs using the lyrics of the song. Clustering on large data sets takes significant amount of time and hence the aim of this project is two fold:

- To cluster songs using K-Means clustering and identify relationships between the clusters that are formed
- Use Jaccard similarity and Locality Sensitive hashing methods to improve the speed of the algorithm and to see how this affects the performance.

### 2.1 Traditional Algorithms Drawbacks

Traditional Algorithms of clustering documents involve converting documents into a vector of scores that denote the importance of each term in the document. This is done by using TF-IDF scores. Similarity of documents is defined then by the cosine similarity between the vectors.

n = Total number of terms in the document. k = Number of times term w appears in a document. m = Total number of documents. d = Number of documents with term w in it.

$$TF(w) = k/w \tag{1}$$

$$IDF(w) = log(m/d) \tag{2}$$

$$cosine(x, y) = (x.y)/(||x|| * ||y||) \tag{3}$$

The problem with this approach is the large corpus of words which results in a sparse matrix leading to large storage requirements and higher computation times.

A key problem in K-Means clustering is comparing the distance between every point in a cluster with one another to find the Clusteroid. Though each cluster can be run in parallel there are N2 comparisons that must be done. So we aim to reduce the N2 comparisons by utilizing Locality Sensitive Hashing.

Furthermore, The term document matrix approach does not include any contextual information and classifies songs with similar words as similar.

### 2.2 Our Contribution

To reduce the space and time complexity, we went about min hashing the term document matrix that was created and only selected x signatures for comparison. To further improve our speed, we used Locality Sensitive hashing with banding to exploit parallelism to the maximum. We used Jaccard similarity instead of cosine similarity as well.

$$Jaccard(x, y) = (x \cap y)/(x \cup y) \tag{4}$$

We also built a k-means clustering model using Cosine similarity to compare the results.

Furthermore, our approach is to use shingles of words to provide some contextual information to the algorithm.

After extensive testing we settled on an optimal shingle size that provided the best results.

# 3 APPLICATIONS

The project has various applications in the research and professional industry:

- The clusters formed can be used to suggest songs to users. Songs can either be randomly picked from the cluster of the cluster can be used as an initial pool of songs to find the most similar song from. The similarity metric can either be Jaccard or Cosine which are both implemented in this project. The project serves as a base for a recommendation system of songs.
- Trends over time can be analyzed within each cluster. It can be checked to see if songs of the same year or decade have been grouped into the same cluster. This would help identify patterns in songs during a time period and if it has changed over time.
- Finding and interpreting relationships between the clusters formed and the genre. This can be done to see if songs of similar genre use similar words and if they would be grouped to the same cluster.

# 4 DATA

In this section we discuss about the structure of the dataset and how the datasets are identified, processed, labeled and utilized.

Description: In order to run the LSH based K-means clustering algorithm, we use lyrics of songs as data and run the clustering algorithm over that. We use the MetroLyrics dataset available from Kaggle. The dataset size is around 99MB and there are about 380,000 song records. Each record is of the form (id, song_name, year, artist, genre, lyrics). Among these fields we use the (id, lyrics) fields to run the algorithm and (year, genre) as the golden standard for analysis purpose. The dataset contains songs from year ranging from 1967 to 2016. The genres available in the data are Country, Electronic, Folk, Hip hop, Jazz, Pop and Others.

Handling: As specified above the dataset contains 120,000 song records. But in order to keep it simple and efficient we retain only the songs in English letters and retain only the ASCII characters. Furthermore, we found that some records have identified to contain null values for the lyrics field. So those records were also eliminated from consideration. After performing all these filtering, we roughly get around 50,000 records to perform clustering.

| | song | year | artist | genre | lyrics |
|---|---|---|---|---|---|
| | | | drake | | |
| 0 | ego-remix | 2009 | beyonce-knowles | Pop | Oh baby, how you doing? You know I'm gonna cut right to the chase Some women were made but me, myself I like to think that I was created for a special purpose You know, what's more special than you? Y... |
| 1 | then-tell-me | 2009 | beyonce-knowles | Pop | playin' everything so easy, it's like you seem so sure. still your ways, you dont see i'm not sure if they're for me.. then things come right along our way, though we didn't truly ask. it seems as if t... |
| 2 | honesty | 2009 | beyonce-knowles | Pop | If you search For tenderness It isn't hard to find You can have the love You need to live But if you look For truthfulness You might just As well be blind It always seems to be So hard to give Chorus:... |
| 3 | you-are-my-rock | 2009 | beyonce-knowles | Pop | Oh oh oh I, oh oh oh I [Verse 1:] If I wrote a book about where we stand Then the title of my book would be 'Life with Superman' That's how you make me feel I count you as a privilege This love is so ... |

# 5 ALGORITHM

In this section, we present a clustering algorithm that uses K-means strategy and attempts to integrate it with Locality sensitive hashing in order to reduce the number of comparisons among the nodes in each cluster. The salient features of this algorithm are: (1) the clustering algorithm can combine nodes that possess similar properties (i.e. shingles). (2) the algorithm is stable and can detect outliers (3) the algorithm has linear storage requirement for the node representation. The n nodes are randomly sampled from the original data and they are further partitioned if the algorithm allows parallel execution of data in particular sections. The analysis of issues pertaining to the size of the sample drawn and the partitioning will be presented in section 4.

## 5.1 Intuition and Overview

The main intention of this algorithm is to identify and capture the implicit correlation between different songs and aggregate them in the form of clusters. In order to achieve this goal, we are trying to utilize K-means clustering algorithm in which instead of using the Euclidean distance as the distance measure, we are using similarity scores as the distance metric.

## 5.2 Pre-Processing Algorithm

The algorithm starts by collecting the entire dataset, divide them into a set of partition and generate all the shingles from the lyrics data. Then we collect all the

unique shingles generated from each partition and store it in a list format. Based on the position of shingles in the list each shingle is assigned a shingle id. Each song will be represented as a sub-list of shingle ids indicating the shingles that they contain. Thus the representation of each song is transmogrified from list of words to list of integer ids.

Once the songs are represented as a set of shingle ids, we generate min hash signatures for each song given fixed set of hash functions. These min hash signatures will then be used in locality sensitive hashing to eliminate similarity comparison between unrelated songs.

## 5.3 Clustering Algorithm

The algorithm starts by randomly taking k samples from the given set of n songs and these k songs are assigned as the initial clusteroids for the k clusters. Based on the number of available partitions s, the n songs are split between s partitions and each partitions receives all the k clusteroids. In each partition for each song we calculate its similarity score with every k clusteroid based on the jaccard coefficients between the shingles of the songs in comparison and assign each candidate song to a cluster which has the closest similarity score. This concludes the cluster assignment step.

In the second phase of the algorithm, we aggregate the songs from all the partitions and divide the n songs into k partitions depending on the cluster the song is assigned to. In order to update the clusteroid of the cluster, we find a song having the maximum average similarity score within the cluster and assign it as the clusteroid for that particular cluster. Within each partition, locality sensitive hashing is performed such that for each song, itâĂŹs similarity score is determined only for candidate songs as indicated by locality sensitive hashing. Then after finding the song with maximum average similarity score, the clusteroid is updated.

The algorithm reiterates over the cluster assignment and updating phases until the clusteroids and cluster elements doesnâĂŹt change (i.e. until the cluster converges, the algorithm is run). The algorithm can also be modified to run until each cluster has number of songs less than a certain threshold.

## 5.4 Clustering Procedure

In this subsection we describe in detail the functionality of the clustering algorithm. The Input parameters provided to the algorithm, the hyper-parameters the algorithm utilizes and the output expected from the algorithm is elucidated here. Initially the data set is given in the format of an input file where each line in the file represents a song as (id, year, genre, artist, lyrics) pair. Hence we run the prepossessing algorithm which generates a lighter and compact representation of the given data set such that each line is of the form (id, list of shingle ids). This file is generated by running the generate_shingle_lists() procedure giving the data set as input. This new dataset will be used to run the clustering algorithm.

The first phase of the algorithm is the cluster assignment phase. Initially k random sample of songs are taken from the given n songs and assigned as the clusteroids. Then we start the map phase assignment_map() procedure to assign songs to clusters based on the jaccard similarity score where the number of partitions is the minimum of available number of partitions and number of songs.

After the songs are assigned their unique cluster ids, each song is assigned to the k clusters by using the k reducers in the reducer phase. Thus the cluster assignment process is completed.

The second phase of the algorithm involves reassigning the clusteroid within each cluster by choosing the song having maximum average similarity score. We calculate the min hash signature from the shingle list of the song using h number of hash functions. Each hash function is of the form,

$$sign(x) = (223 * x + 139 * i) \tag{5}$$

Here x is the smallest shingle id of each song that gives the smallest value for sign(x) and i value changes from 0 to h and L is the length of the unique shingles list.

We call the updation_map_phase1() procedure where each song in the cluster is split across b bands each of size r where h=b*r. Then the updation_reduce_phase1() procedure that performs locality sensitive hashing in each of the bands and generates all the similar song candidate pairs in each cluster. Then for each cluster the updation_map_phase2() contains the list of similar songs for each song in each cluster. Here the jaccard similarity is performed using the shingles ids of songs and the average similarity score for each song in each cluster is calculated. The final updation_reduce_phase2() will

reduce based on the cluster ids and within each cluster assigns the song having maximum average similarity score within that cluster as the clusteroid, thereby concluding the cluster updation phase.

Reducing the computation time taken: Even though we perform map reduce based parallel programming, since we have to use the entire shingle ids list of each song during the jaccard coefficient calculation, it is a time consuming process as the length of shingle ids list is highly varying and can be too long. Hence a better recommendation is to use min hash signatures which are of fixed size h and smaller compared to length of shingle ids list. Even though there lies a tradeoff between the accuracy of clustering and time taken to run the clustering algorithm, based on our experimental analysis it is evident that this loss is tolerable if the number of min hash functions is sufficiently large. Hence the algorithm will run quicker since the algorithm has fewer data to process than the initial approach.

## 5.5 Time and Space Complexity

The worst case time complexity of algorithm can be found by considering the 2 phases of the clustering algorithm. The initial cluster assignment phase takes O(n*k) where each song has to be compared with every clusteroid. The final cluster updation phase will take O((n/k)2). So if run sequentially the complexity of the algorithm is of order of O((n/k)2). Since we have to store all the n data points, the space complexity taken by the algorithm is O(n).

## 5.6 Partitioning for Speedup

In order to reduce the computation time if we partition the dataset based on available number of instances, we get a smaller run time complexity. If we have an arbitrary p number of partitions, the first phase will have run time complexity of O(n*k/p) and the run time complexity of second phase is reduced to O(n/(p*k))2. So when the algorithm is run in parallel, the run time complexity is O(n/(p*k))2. Space complexity is not affected by partitioning.

## 5.7 Psuedocode

```
Procedure: generate_shingle_lists()
Input: (songs[ ])
Output: ( song_shingle_list[ ])
Process:
for each song in songs:
        shingles= get_shingles(song.lyrics)
        for each shingle in shingles:
                shingle_id = get_shingle_id( shingle)
                song.shingle_ids.add(shingle_id)
        add_song(song.id,song.shingle_ids)
```

```
Procedure: assignment_map()
Input: (song, cluster_list [ ])
Output: (cluster.id, song)
Process:
for each cluster in cluster_list:
        get_similarity_score (song,cluster.clusteroid)
yield most_similar_cluster.id,song
```

```
Procedure: assignment_reduce()
Input: (cluster_id, song_list [ ])
Output: (cluster)
Process:
cluster=get_cluster(cluster_id)
for each song in song_list:
        assign_song(cluster,song)
yield cluster
```

```
Procedure: updation_map_phase1()
Input: (cluster.id)
Output: (cluster.id, band_id),(song.id,sign_band)
Process:
cluster=get_cluster(cluster.id)
for each song in cluster.members:
for each band:
        sign_band=get_sign_band (song,band)
        yield (cluster.id,band.id), (song.id,sign_band)
```

```
Procedure: updation_reduce_phase1()
Input: (cluster_id, band_id), song_sign_band_list [ ]
Output: (cluster_id,song_id),similar_songs_ids [ ]
Process:
band_song_mapping={}
for each (song,sign_band) in song_sign_band_list:
        if sign_band in band_song_mapping:
                band_song_mapping[sign_band].add(song)
        else:
                band_song_mapping[sign_band]=[song]
for each sign in band_song_mapping:
        song_list=band_song_mapping[sign]
        for each song in song_list:
                other_song=get_song(song_list,song)
                yield (cluster_id,actual_song_id), other_song
```

```
Procedure: updation_map_phase2()
Input: (cluster_id,song_id),other_songs []
Output: cluster.id, (song_id,score)
Process:
song=get_song(song_id)
score=0
for each other_song in other_songs:
        score+=get_similarity_score(song,other_song)
yield cluster.id,(song_id,score)
```

```
Procedure: updation_reduce_phase2()
Input: (cluster_id, song_score_list [ ])
Output: (cluster_id)
Process:
cluster=get_cluster(cluster_id)
new_centroid=get_new_centroid(song_score_list)
cluster.update_centroid(new_centroid)
yield cluster_id
```

## 5.8 Challenges

We faced the following challenges during the project: The first challenge was cleaning and parsing data. The dataset contained a lot of NULL values and was skewed. It had characters of various encodings and had to be read appropriately. Identifying the parallelization process and the process of cluster centroid updation and new point updation was a significant challenge. Furthermore, since our data was multidimensional, converting higher dimensional data to 2 dimensions for visualization and analysis purposes was a challenge.

# 6 RESULTS

## 6.1 Algorithm configurations

For the size of the shingles, we had to make sure that the shingle size was not too small that the uniqueness of a shingle is not compromised, while at the same time a very large shingle would hide the possible similarities that may exist, we based out shingle size on the average length of words, which is 5.

The correct k value, which denotes the right number of clusters for a dataset with no predefined labels is an ambiguous step. A smaller k, value like 2 would just provide two clusters, from which no significant insight can be drawn, while a large cluster will decrease the error at the cost of having all the observations split into many clusters. Therefore we chose to model our k value based on some of the other attributes present in our dataset, as the goal is to find the relationship of the song lyrics with its attribute values. So after experimenting with every attribute, like choosing k as 21 to depict all years present, k as 40 for the number of artists and k as 5 for the number of major genres, we chose k as 5, as it gave the best results with the highest inter cluster similarity as compared to the other values.

The signature size of each song, which depicts the number of hash functions generated and used, is 100. Min Hashing requires using collections of hash functions which should produce non-colliding results.For the fixed parameters we used high valued prime numbers.

For the first phase, we partitioned the RDD into N partitions, where N depicts the total number of songs in the dataset, as the k centers were broadcasted to each of the partition. For the second phase we partitioned the RDD into k clusters, so that the similarity scorer for every song, with regard to every other song in that cluster can be calculated and hence the new centroid be assigned. But increasing the number of partitions beyond could not be achieved, as all the songs in a cluster is required to calculate the similarity score.

## 6.2 Dataset Experimented

The dataset used for this project, is the MetroLyrics dataset available on Kaggle. It initially contained 380,000 songs, which after cleaning, was reduced to 276,000 songs. The dataset contains 6 columns, the index of each song, the song name, the artist who sang the song, the year the song was released, the genre the song belongs

to, and the lyrics of the song. And each song lyrics contains on an average, 300 to 400 words. For clustering, we considered only the lyrics of the song. But used the other columns to find out relations between the songs and its other attributes.

## 6.3 Quality and Result Analysis

The measure of how good a cluster is, is analyzed based on the content of the cluster. We also analyzed the cluster based on how high the clusterâĂŹs cumulative cluster score is. This was calculated by calculating the sum of the normalized similarity score of a song with every other song in that cluster. The majority songs in each cluster belonged to one of the genres. Upon further inspection we found that majority of the rock song and hip - hop songs were classified into the same cluster, which signified how closely related the two genres are in terms of lyrics. The songs whose category was not available, we could see belonged to the cluster which contained majorly hip-hop songs or to the cluster which contained majorly pop songs. Most of the pop songs were clustered majorly into 2 of the clusters , which upon further inspection we realized, were Spanish pop songs and English pop songs. We generated word clouds from the clusters created. Some of which we have shown here. We can clearly make out from the most popular words in these songs, which genre they may belong to. We have also generated a scatter plot, by transforming each of the song into a tf-idf vector and then using principal component analysis, we have reduced its dimensions from 100 (length of the signature) to 2. While this resulted in f information loss, plotting the same for the genres of the songs and the clusters that was created by our algorithm, we can see that there is a close relation between the genres and our clusters.

## 6.4 Comparison

Cosine similarity was used to compare performance with LSH. While cosine was a little slower than LSH, the cluster formed were a little more cohesive with that of the genre, which was to be expected, since LSH trades off accuracy for speed. We also used the signatures to perform LSH as opposed to the shingles, since the number of shingles generated were m - 5, where m is the total number of characters in each song. Using the signatures over the shingles and words, gave a significant boost to the run time.

| | index | song | year | artist | genre | lyrics | text | Cluster |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | ego-remix | 2009 | beyonce-knowles | Pop | Oh baby, how you doing?\nYou know I'm gonna cu... | Oh baby, how you doing?You know I'm gonna cut ... | 1 |
| 1 | 1 | then-tell-me | 2009 | beyonce-knowles | Pop | playin' everything so easy,\nit's like you see... | playin' everything so easy,it's like you seem ... | 1 |
| 2 | 2 | honesty | 2009 | beyonce-knowles | Pop | If you search\nFor tenderness\nIt isn't hard t... | If you searchFor tendernessIt isn't hard to fi... | 1 |
| 3 | 3 | you-are-my-rock | 2009 | beyonce-knowles | Pop | Oh oh oh I, oh oh oh I\n[Verse 1:]\nIf I wrote... | Oh oh oh I, oh oh oh I[Verse 1:]If I wrote a b... | 1 |
| 4 | 4 | black-culture | 2009 | beyonce-knowles | Pop | Party the people, the people the party it's po... | Party the people, the people the party it's po... | 1 |

**Figure 1: Result of 5 songs and their clusters using cosine similarity - Songs of pop genre are classified in cluster 1 together**

```
0 ['Rock', 'Other', 'Rock', 'Other', 'Other']
1 ['Pop', 'Pop', 'Pop', 'Pop', 'Pop']
2 ['Country', 'Country', 'Country', 'Country', 'Other']
3 ['Hip-Hop', 'Hip-Hop', 'Hip-Hop', 'Hip-Hop', 'Hip-Hop']
4 ['Pop', 'Pop', 'Pop', 'Pop', 'Pop']
```

**Figure 2: Genre of the top 5 songs in each of the clusters**

## 6.5 Word Clouds of Cosine Similarity Results

By plotting word clouds of the cluster, we get a representation of the kinds of words in the song. Looking at these words, we can see if there is any relationship between the cluster formed. In the first cluster formed, we see that a lot of terms related to love and life. The second cluster has grouped together songs of a particular language. This is an interesting insight to see that the clustering algorithm can differentiate between songs of different languages. The third cluster seems to have words similar to the first cluster again and we cannot derive too many insights from it. The fourth cluster has terms that mostly occur in rap songs and terms that occur in songs that are offensive or portray hatred.

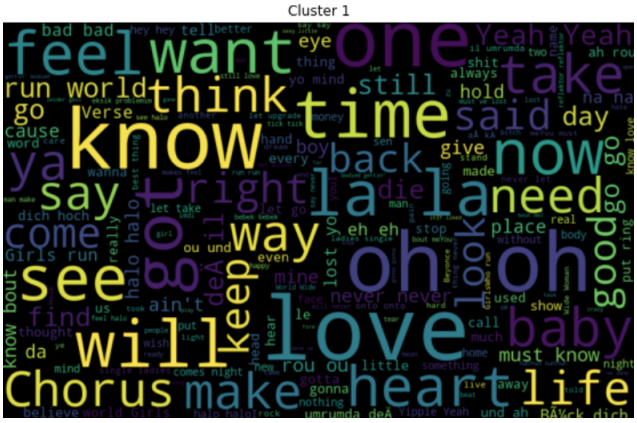


**Figure 3: Cosine similarity: Word cloud for cluster 1**


**Figure 6: Cosine similarity: Word cloud for cluster 4**

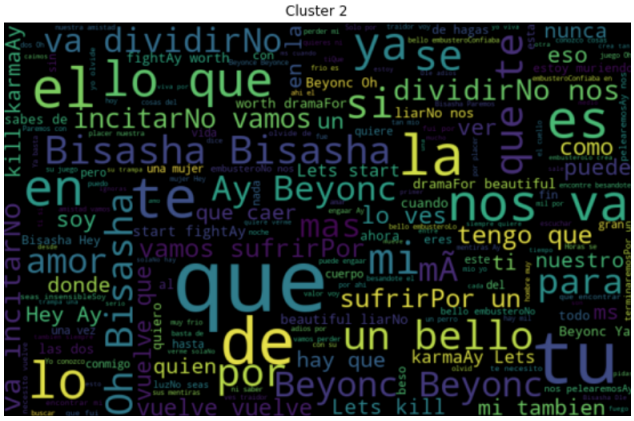


**Figure 4: Cosine similarity: Word cloud for cluster 2**

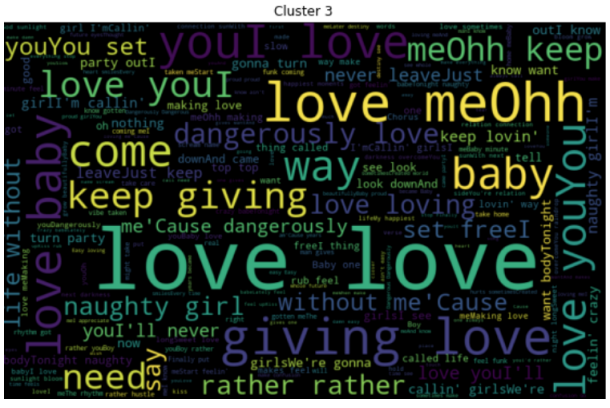


**Figure 5: Cosine similarity: Word cloud for cluster 3**

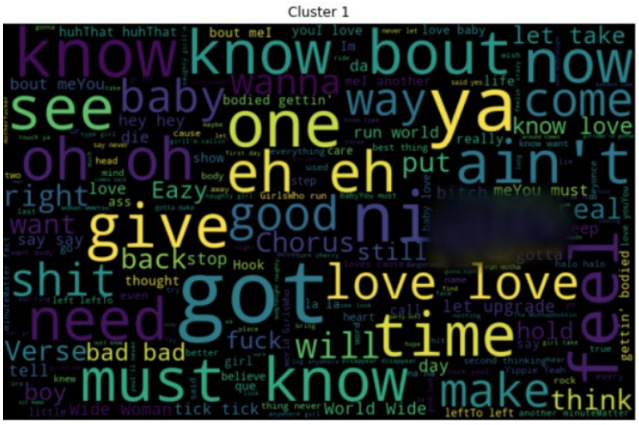


**Figure 7: Jaccard similarity: Word cloud for cluster 1**

## 6.6 Word Clouds of Jaccard similarity results

The first word cloud here looks like it can be classified into pop genre. The second word cloud represents similar words but can be classified into rock songs The third word cloud has words that represent hip hop songs and the fouth word cloud could represent pop songs again.

## 6.7 Scatter Plots of the songs based on genre

By noticing figure 11 and figure 12, we see that there is a strong correlation between the actual genre of the songs and the clusters that have been formed. We see that cluster 3 in the graph correlates with Hip-hop and


**Figure 8: Jaccard similarity: Word cloud for cluster 2**

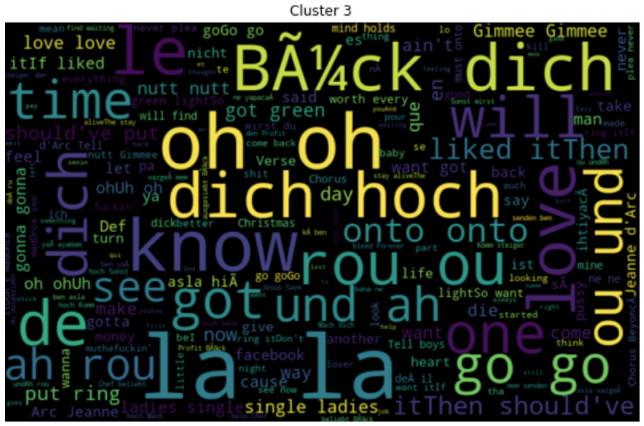


**Figure 9: Jaccard similarity: Word cloud for cluster 3**

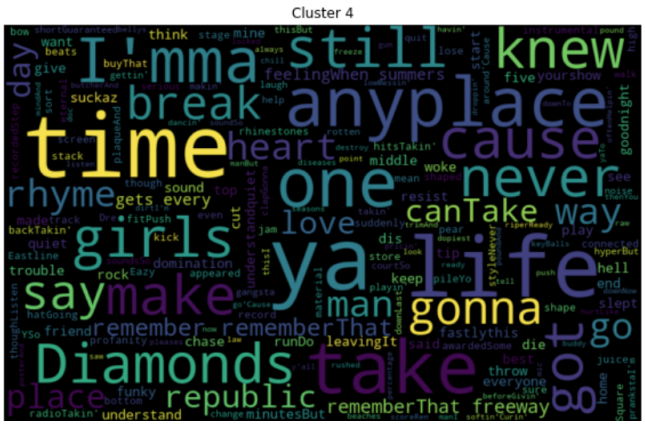


**Figure 10: Jaccard similarity: Word cloud for cluster 4**

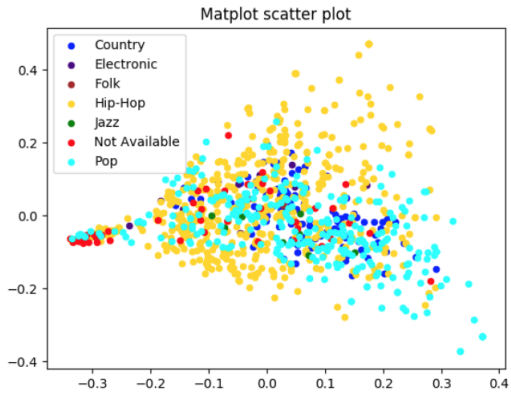


**Figure 11: Scatter plot of songs by genre - From the actual dataset**

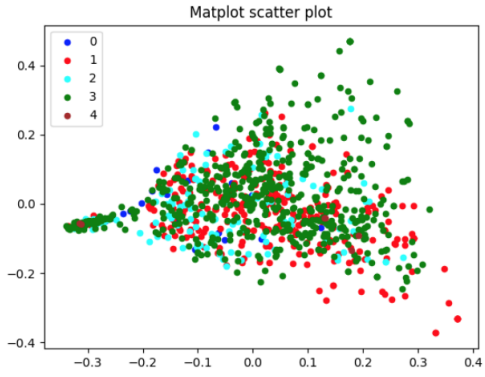


**Figure 12: Scatter plot of the clusters that were formed**

cluster 4 with pop genre. This is similar to the results we got above.

## 7 CONCLUSION AND FUTURE WORK

Based on the experiments performed, we can observe that k-means combined with locality sensitive hashing performs well for large scale data. For large scale data, using LSH clearly outperforms using cosine similarity when their speed- accuracy tradeoffs are compared. Studying the clusters obtained, a strong correlation between the lyrics and genre was observed. We were also able to uncover some relationships between different

genres when their lyrics are compared. We were also able to improve the runtime speed of the algorithm by utilizing the signatures to compute the jaccard similarity, instead of the shingles generated for the songs. We have run the code on an octa core system, so most of the partitions were not truly parallelized. Running this on a distributed network would give a better performance. Also, because we are comparing song lyrics, we could integrate some natural language processing techniques to get some contextual information, and gain better insights about the data.

# REFERENCES


[1] Mayer, Rudolf and Neumayer, Robert and Rauber, Andreas., "Rhyme and Style Features for Musical Genre Classification by Song Lyrics" , pp. 337-342, 2008.

[2] Li, Qiuhong and Wang, Peng and Wang, Wei and Hu, Hao and Li, Zhongsheng and Li, Junxian.,"An efficient K-means clustering algorithm on MapReduce " *International Conference on Database Systems for Advanced Applications*, 2014.

[3] Silva, Eliezer S and Valle, Eduardo.,"K-medoids LSH: a new locality sensitive hashing in general metric space. " *SBBD (Short Papers)*, 2014.

[4] Shen, Xiaobo and Liu, Weiwei and Tsang, Ivor and Shen, Fumin and Sun, Quan-Sen.,"Compressed k-means for large-scale clustering" *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[5] Elkahky, Ali Mamdouh and Song, Yang and He, Xiaodong.,"A multi-view deep learning approach for cross domain user modeling in recommendation systems" *Proceedings of the 24th International Conference on World Wide Web*, 2015.

[6] Wu, Yi-Pu and Guo, Jin-Jiang and Zhang, Xue-Jie.,"A linear dbscan algorithm based on lsh" *2007 International Conference on Machine Learning and Cybernetics*, 2007.