Clustering Algorithms

Meghana Ananth Gad (50182335)
Ladan Golshanara (50093954)
Duc Thanh Anh Luong (50094977)

6th November 2016

1 Introduction

In this report we describe the four clustering algorithm we implemented: K-means, hierarchical clustering adn DBSCAN and Map-reduce k-means. The algorithms are implemented in R except Map-Reduce k-means which is implemented in Java. For the PCA and plotting, built-in packages are used. First we visulaize each given dataset based on their ground truth cluster: The "cho.txt" dataset has 368 data points that are clustered into 5 clusters as shown in Figure 1.

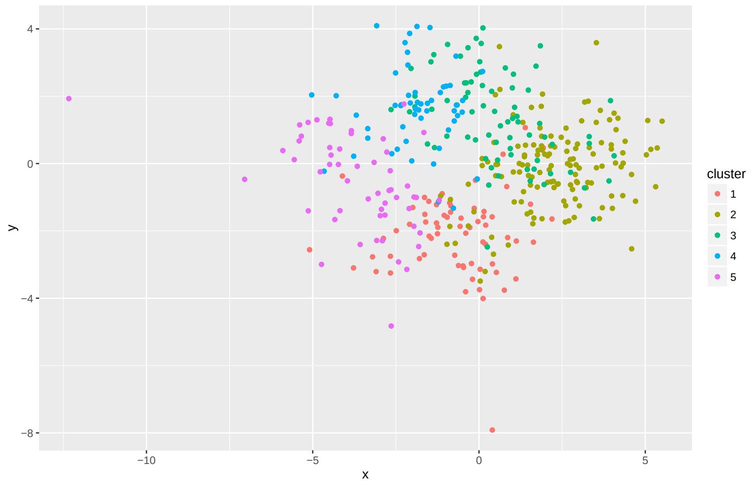

Figure 1: Visualization of cho dataset

The iyer dataset has 517 datapoint that are clustered into 10 groups as showsn in Figure 2

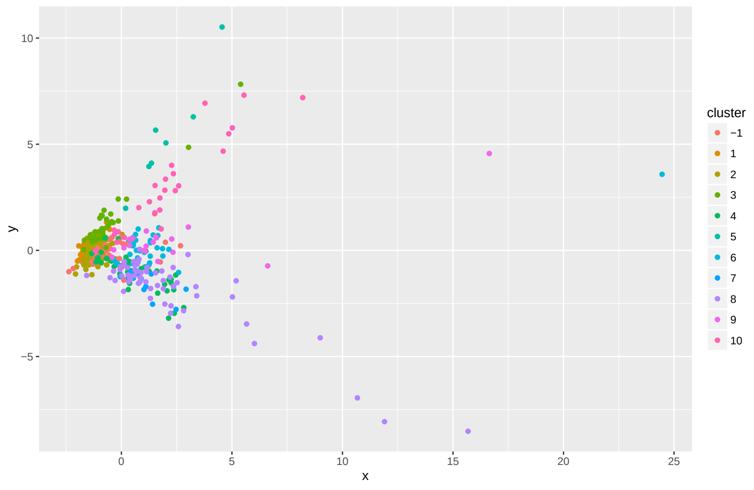

Figure 2: Visualization of iyer dataset

2 K-means algorithm

2.1 Implementation

Our kmeans algorithm works as follows:

1. **Input:** dataset, number of clusters **Output:** centroids, labels
2. MaxIter = 500
3. Initialize centroids to random points in the dataset.
4. Repeat until MaxIter
    - (a) For each point p in dataset
    - (b) Calculate the distance of p to each centroid
    - (c) Assign p the centroid with the minimum distance
5. Update centroids based on the points they were assigned

6. if the centroids do not change stop

K-means function gets the dataset and number of clusters as the input. The initial centroids are chosen randomly from the given dataset. Then for each data point in the dataset, we compare the euclidean distance of that point to each centroid and then cluster it based on the minimum distance. After clutering every point based on the current centroids, we update the centroids. These process continues until either the difference between the new centriods and old centroids are zero or the maximum number of iterations is reached. The maximum iteration in our algorithm is 500.

2.2 Kmeans Result Visualization

The result of K-means algorithm depends on the initial centroids. For the purpose of reproducibilty of the results we generate the centriods to be the first k data points of the data set where k is the number of clusters. The result of our k-means clustering on cho dataset by setting k=5 is shown in Figure 3.

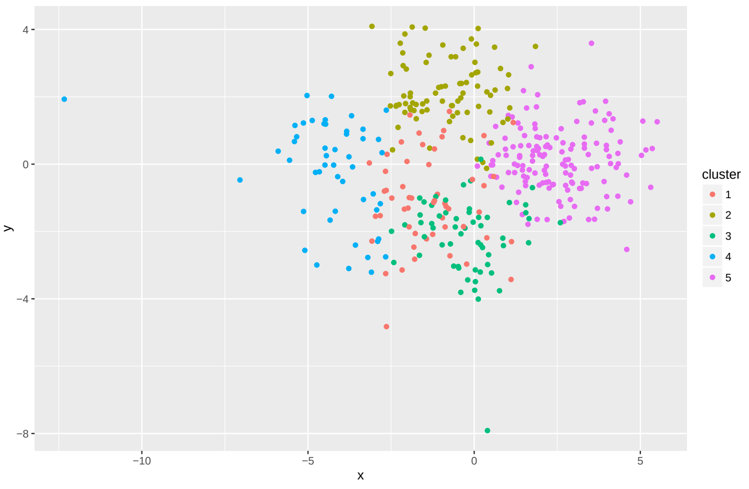

Figure 3: Visualization of k-means clustering on Cho dataset where k=5

The results on iyer dataset is shown in Figure 4.

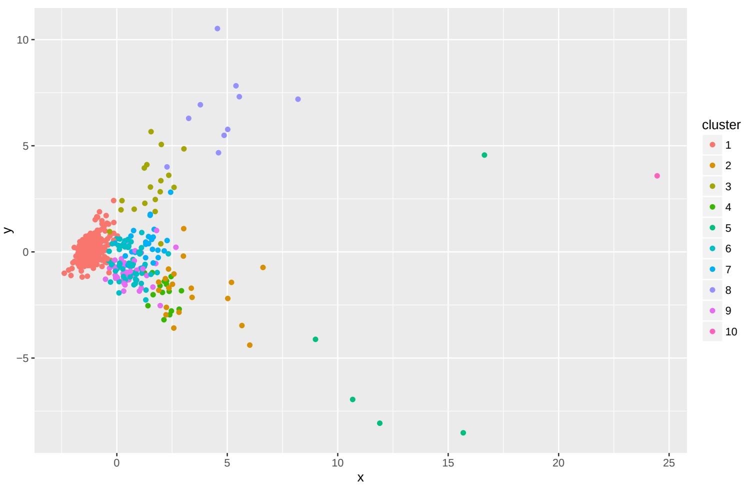

Figure 4: Visualization of k-means clustering on iyer dataset where k=10

2.3 Kmeans Result Evaluation

For the evaluation we use Jaccard and Rand index which are shown in Table 2.

Dataset	Jaccard index	Rand index
cho	0.3990425	0.7978067
iyer	0.3143917	0.7136358

Table 1: External Index for Kmeans clustering

We observed that the jaccard coefficient of cho.txt is higher than iyer.txt and this could be because the original clusters in cho.txt are more sperical when compared to the clusters in iyer.txt.

2.4 Pros and Cons of Kmeans

Pros

1. K-means has linear run time. The run time for n data points can be approximated as $O(n)$
2. The algorithm is easy to implement as the important computations involved in this algorithm are distance and centroid computation.

Cons

1. K-means can identify only sperical clusters.

2. The algorithm takes number of clusters as input and it is not easy to fix this parameter (k). Without prior knowledge of k, it would be difficult to obtain optimal results.

3. The result of the algorithm is subjective to the initial centroids. In other words, the results of a particular run cannot be reproduced if the initial centroids are different.

4. K- means algorithm does not identify any point as outlier, it cluster all the points in the data set.

3 Hierarchical Clustering with Single Linkage

The implementation of our hierarchical clustering method is in function *my_hclust*. The input of our algorithm is the distance matrix, which is the euclidean distance between each pair of points. The algorithm works by iteratively merging a pair of nodes that have smallest distance into a bigger cluster. This process finishes when there is one node remaining. By default, every data point is the leaf node at the beginning and was merged iteratively to form a tree. The distance between two nodes is measured by the minimum distance between one data point from one node and another data point from the other node.

In order to keep track of merging process, variable *merge* is used and it has the form of $(n-1)\times 2$ matrix where each row i indicates that node $merge[i,1]$ and $merge[i,2]$ are merged in i^{th} step. Since hierarchical clustering iteratively merge a pair of nodes and produce a new node until there is only one node remaining, this process clearly has $n-1$ iterations where n is the number of data points in our dataset.

Variable *height* is used to keep track the height of the tree for each merging step. For this reason, it is a vector of size $n-1$.

Variable $group\_membership$ is used to keep track of the membership of each node in the tree.

By default, the leaf node (individual data point) will has the node label $-i$ where i is the row index of that data point. All non-leaf node in the tree wil be labeled increasingly from 1 to $n-1$.

The distance matrix is the input of hierarchical clustering and will be gradually updated when the tree is being constructed.

For merging step i^{th}, we find the minimum distance in the distance matrix and it will be the height of the new node that we are going to form. After that, we find the pair of nodes that has this minimum distance and update row i of matrix *merge* with these two nodes. We also have to update the group membership to reflect the changes when merging two nodes. Finally, we have to update the distance matrix when two nodes are merged.

The return value of hierarchical clustering will be a tree which is represented by two components:

- $(n-1) \times 2$ matrix that represents merging process (variable *merge* that described above)
- vector of size $n-1$ that represents the height at which the merging process happens.

3.1 Hierarchical Clustering Result Visualization

The result of the hierarchical clustering on the cho dataset is shown in Figure 5 and its dendrogram is shown in Figure 6. Note that the dendrogram has to be sclaed to be fitted in the page. The graph is also provided as a separate file in the submitted project folder.

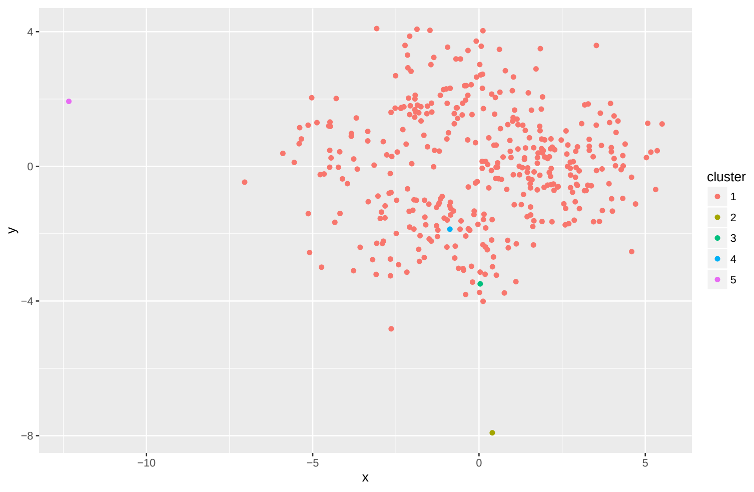

Figure 5: Visualization of hierarchical clustering on cho dataset

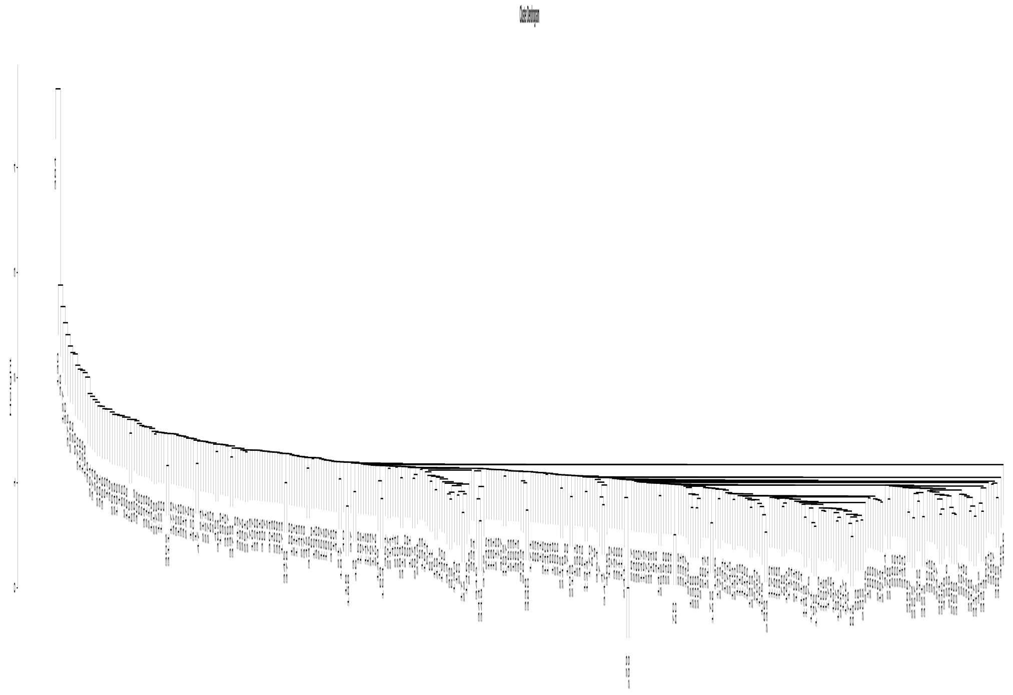

Figure 6: Dendrogram of Hierarchical clustering on Cho dataset

As shown in Figure 5 , hierarchical clustering found five clusters, however one of the clusters is much bigger than the other ones.

Note: The dendograms of results were two big and so they are not clearly visible on the report. But we have included the (*.pdf*) versions of the images in our submission for your perusal.

The result of the algorithm on the iyer data set is shown in Figure 7 and the corresponding dendrogram is depicted in Figure 8.

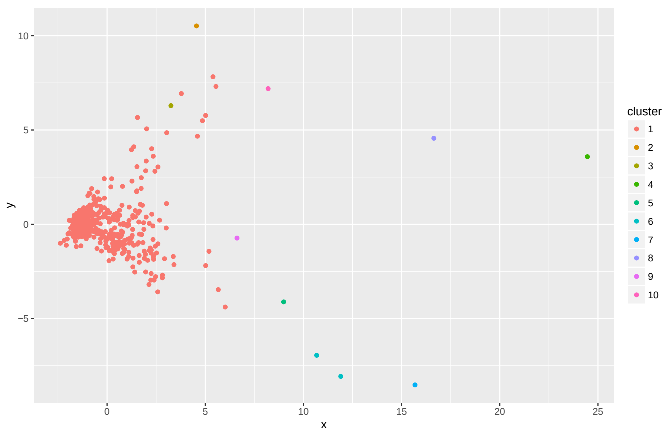

Figure 7: Visualization of hierarchical clustering on iyer dataset

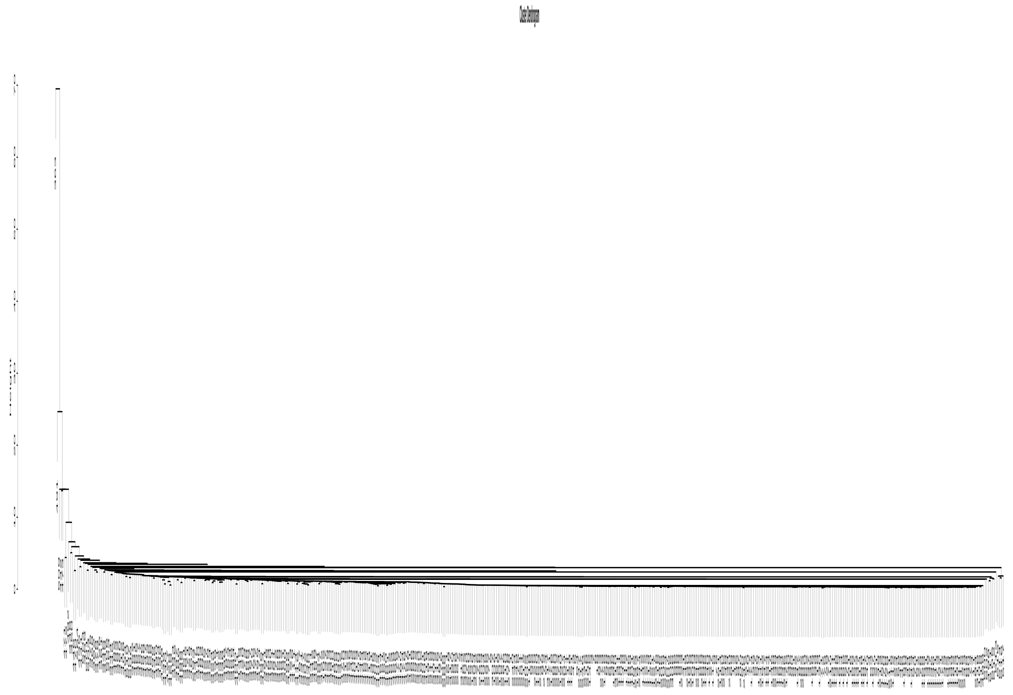

Figure 8: Dendrogram of Hierarchical clustering on iyer dataset

3.2 Hierarchical Clustering Result Evaluation

Again, we use the external indexes to validate our clustering results.

Dataset	Jaccard index	Rand index
cho	0.228395	0.2402749
iyer	0.1582431	0.1882868

Table 2: External Index for Hierarchical clustering

We observed that the jaccard co-efficient obtained after performing hierarchical clustering on cho.txt and iyer.txt is low as most of the data points get clustered into one group. This could be because the ground truth clusters of the original data in cho.txt and iyer.txt have no hierarchy or taxonomy.

3.3 Pros and Cons of Hierarchical Clustering

Pros

1. Prior knowledge of number of clusters to be found is not required as the final result of the algorithm is a tree, which can be cut at different levels to obtains desired number of clusters.

2. Multiple run of the algorithm produce the same result.

3. This algorithm produces good results with data that has heirarchy or taxonomy.

Cons

1. This algorithms is slower when comapred to K-means as it takes approximately quadratic run time $O(n^2)$ for n data points.

2. Sometimes places majority of the data points into the same cluster.

3. At any stage the algorithm does not reconsider the clustering decision made at earlier stages.

4 DBSCAN Algorithm

4.1 Implementation of DBSCAN

The implementation of Dbscan algorithm was done as expplained below. We maitain two global lists, one to identify the if a point has been visited and the other to hold the assigned cluster number. We also compute a distance matrix in the beginning of the algorithm. We have two functions to comute neighbors and to expand cluster. These functions are called inside the my_dbscan function in our implementation.

The below steps are carried out for every point in the dataset

1. Dbscan algorithm
   Input: Dataset, epsilon, minpts
   Output: List of assigned clusters
   Here we check if the point has been visited, if it has been visited then we move on to the next point in the dataset else we compute the neighbors of this point and mark it as visited.
   For every point that we mark as visited, we then check if the number of nieghbors of this point are less than minpts and if so we mark it as noise. Otherwise, we start a new cluster with this point and expand the cluster by calling expandcluster function in our implementation.

2. Expand cluster for point P
   Input: Data point P, neighbors of P, epsilon, minpts
   Output: All the density reachable points are assigned to same cluster as point P
   In this function we check if every neighbor point of point P has been visited. If a neighbor point (P') has not been visited,then it is marked visited and we compute its (P') neighbors and check if number of neighbors are greater than or equal to minpts and if so we append these points to the original list of neighbor points (P). This way we find all the density reachable points from point P. We also check if the neighboring points of point P have already been assigned cluster, if not we assign them to cluster to which point P belongs to.

3. Computing neighbors function
   Input: Data point P, epsilon
   Output: List of neighbors of point P
   In this function we check if the eucledian distance between P and every other point in the data set. We append the points who are at a distance less than epsilon to the list of neighbors and return the final list of neighbors.

4.2 Visualization of the Results of DBscan

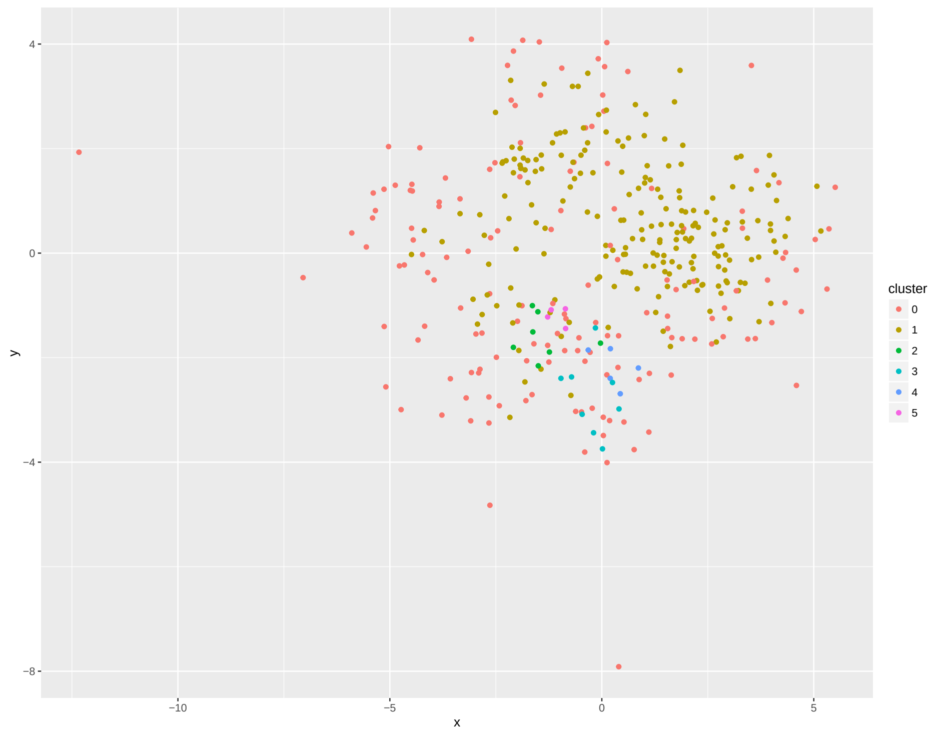

Figure 9: Visualization of Dbscan clustering on cho.txt when epsilon=1.2 and minpts=7

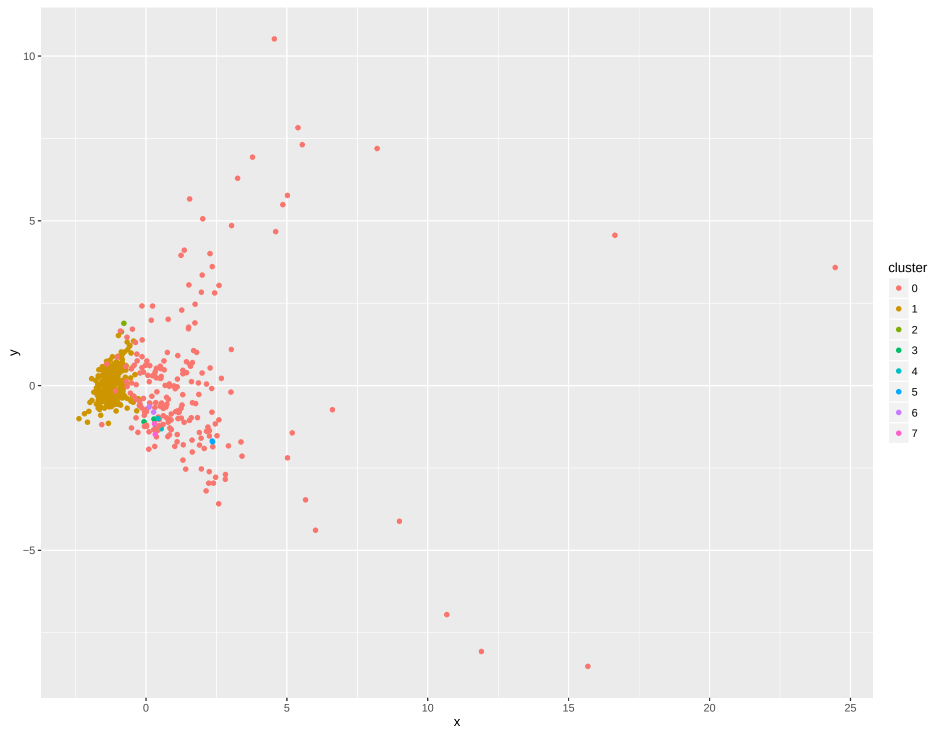

Figure 10: Visualization of Dbscan clustering on iyer.txt when epsilon=0.7 and minpts=2

4.3 DBSCAN Result Evaluation

Epsilon	Minpts	Number of Clusters	Jaccard Coefficient	Rand Index
1.0	7	3	0.2031064	0.4056619
1.0	15	1	0.2127672	0.2713093
1.0	20	0	0.2300733	0.2300733
1.2	7	5	0.2214344	0.565183
1.2	15	2	0.2283315	0.5473033
1.2	20	1	0.2201191	0.4850868
1.4	7	1	0.2234849	0.3992322
1.4	15	1	0.2400562	0.4986174
1.4	20	1	0.238631	0.5195173
1.8	7	1	0.2250466	0.2691213
1.8	15	1	0.2251729	0.2723295
1.8	20	1	0.2244563	0.2772155
2.0	7	1	0.2250547	0.263054
2.0	15	1	0.2251581	0.2662622
2.0	20	1	0.2250466	0.2691213

Table 3: Result of Dbscan on cho.txt

Epsilon	Minpts	Number of Clusters	Jaccard Coefficient	Rand Index
0.5	2	4	0.2501578	0.6134297
0.5	3	2	0.2840324	0.6343808
0.5	4	1	0.2461848	0.6028643
0.6	2	6	0.2833062	0.6422524
0.6	3	1	0.2803471	0.6301456
0.6	4	2	0.2757101	0.6268683
0.7	2	7	0.285019	0.6444074
0.7	3	2	0.2840324	0.6343808
0.9	2	16	0.2896267	0.6685984
0.9	3	5	0.2843014	0.6488595
1.0	2	15	0.2884626	0.6685984
1.0	3	5	0.2835567	0.6526756

Table 4: Result of Dbscan on iyer.txt

The values of epsilon and minimum points (minpts) listed above were estimated based on observations made during visualization of the original data in cho.txt and iyer.txt after performing PCA. We noticed that the majority of data points in iyer.txt were very close to each when compared to the data points in cho.txt. We also observed that the clusters in iyer.txt are more compact when compared to clusters in cho.txt and so we start our experiment with epsilon=1.0 and minpts=7 in case of cho.txt and epsilon=0.5 and minpts=2 in case of iyer.txt.

We noticed that varying epsilon and minpts by very small value greatly impacts the total number of clusters that are found by the algorithm. The optimal values are difficult to find as Dbscan is very sensitive to small changes in epsilon and minpts. The values of epsilon and minpts can be chosen based on the requirements. In the below paragraph we have explained how choosing a large value of epsilon and small value for minpts can help us in isolating points that far away from majority of points in cho.txt and iyer.txt

The Dbscan algorithm can identify clusters of arbitrary shapes and this property can be used to identify points which are significantly far from the majority of the points in the data. Setting epsilon=4.0 and minpts=8 in case of cho.txt helps us in identifying points which are significantly far from the majority of the points in the data. In the below figure, the point marked pink is identified as an outlier by Dbscan as it is at a significantly far distance from all other points towards right.

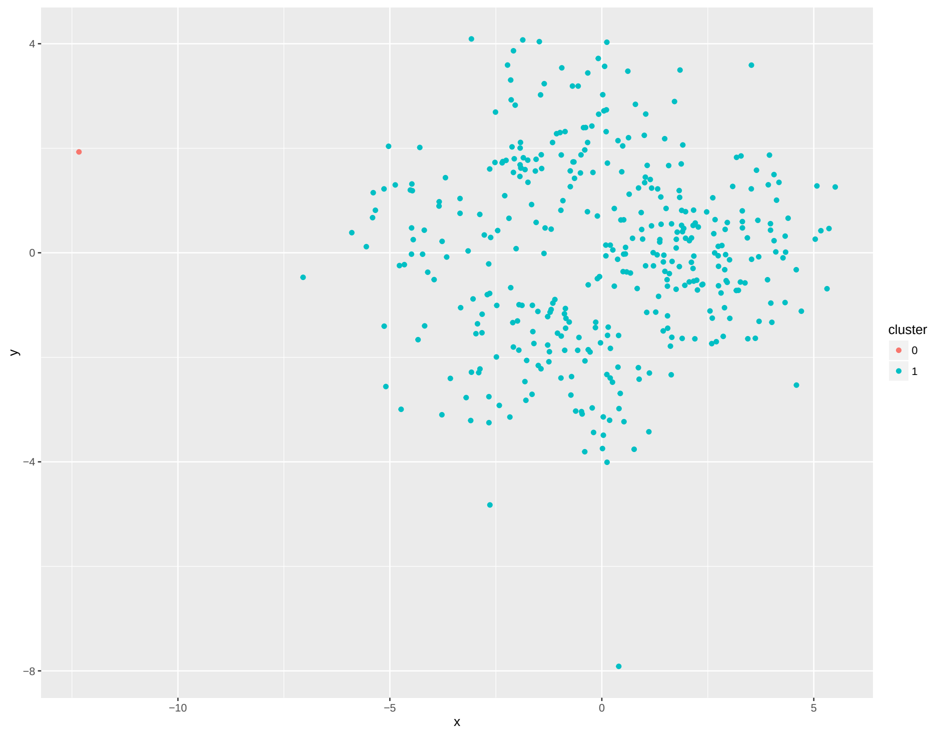

Figure 11: Visualization of Dbscan clustering on iyer dataset when epsilon=4.0 and minpts=8

Since the mpjority of data points in iyer.txt are very close to each other, chosing epsilon=0.7 and minpts=3. The points marked pink are quite far away the points marked blue.

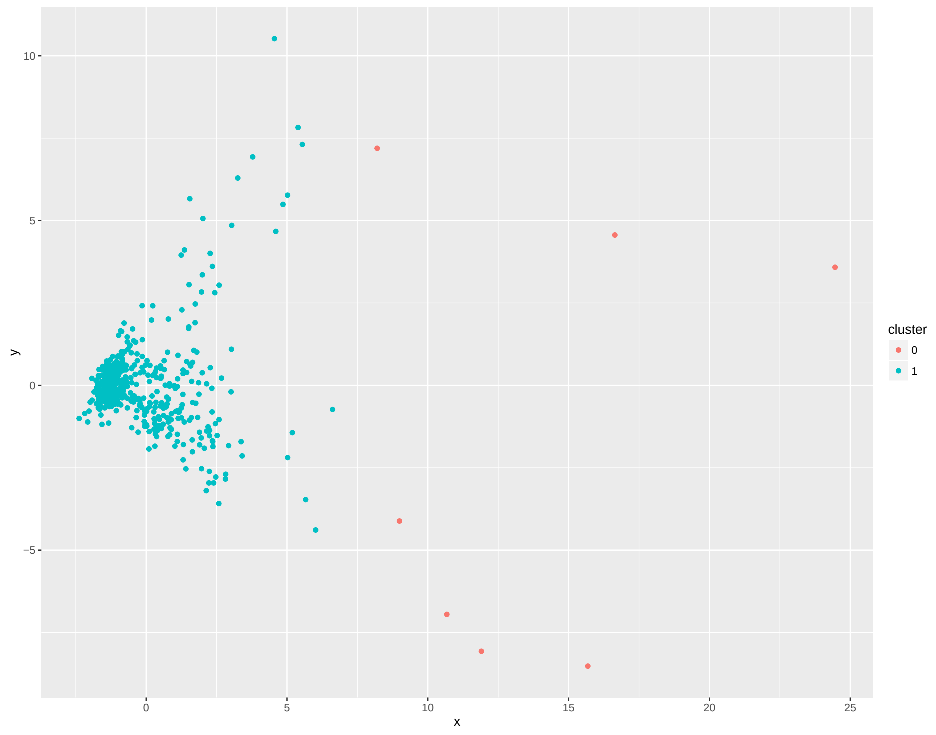

Figure 12: Visualization of Dbscan clustering on iyer dataset when epsilon=0.7 and minpts=3

4.4 Pros and Cons of Dbscan Clustering

Pros

1. This algorithm can identify arbitrary shaped clusters.
2. Prior knowledge of number of clusters to be found is not required.
3. The final results of the algorithm is not greatly impacted by presence of outliers in the data.
4. The results of the algorithm are reproducible for fixed epsilon and minpts.
5. The algorithm is fast and has an approximate run time of $O(nlogn)$, if finding neighbors of point is implemented efficiently.

Cons

1. Cannot identify clusters with varying density.
2. The final result of the algorithm is highly dependent on the parameter epsilon and minpts. And there are no predefined methods to approximate epsilon and minpts that could give optimal results.

5 Map-Reduce Kmeans

In this section, we describe our implementation of Kmeans algorithm in Map-Reduce framework. We have used java for the map-reduce implementation. In order to compare the Map-Reduce implementation with the non-parallel kmeans, the initial centroids are set to the first k datapoints in the given data files, so that the result be deterministic and reproducible.

5.1 Implementation

We maintain an arraylist of centroids, which is updated in every iteration. The final centroids are written to output file. The intermediate centroids are stored in arraylists prev_centroids and centroids respectively and these arraylists are compared at the end of each iteration to check for convergence of the algorithm.

5.1.1 Mapper

Input Key: The byte offset of the input file.
Input Value: A single record in the file.
Output Key: Centroid
Output Value: Data row (vector of double values separated by tab).

In the mapper we compute the euclidean distance of each data point with the centroids and assign the point to the centroid to which it has minimum distnce.

```
public static class AssignmentMapper extends
                    Mapper<Object, Text, IntWritable, Text> {
    @Override
      protected void map(Object key, Text value, Context context)
```

```
            throws IOException, InterruptedException {

            ArrayList<Double> data_row = convertDataRow(value.toString());
            double minDist = Double.MAX_VALUE;
            int minClust = -1;
            for (int i = 0; i < centroids.size(); ++i) {
                double dist = distance(data_row, centroids.get(i));
                if (dist < minDist) {
                    minDist = dist;
                    minClust = i;
                }
            }
            context.write(new IntWritable(minClust), value);
        }
    }
```

5.1.2 Reducer

Input Key: Centroid
Input Value: List of data row (each row is in Text format)
Output Key: New centroid
Output Value: Data row (vector of double values separated by tab).

In the reducer the data rows associated with a paritcular centroid are used to find the new centroid for next iteration. The way this works is, hadoop internally groups all the values associated with particular key and gives it as input to reducer.

```
Public static class UpdateReducer
        extends Reducer<IntWritable, Text, Text, IntWritable> {
        @Override
    protected void reduce(IntWritable clustID, Iterable<Text> data_points,
    Contex context)
        throws IOException, InterruptedException {

        ArrayList<Double> newCentroid = null;
        int count = 0;
        for (Text data_row : data_points) {
```

```
        count++;
        if (newCentroid == null) {
            newCentroid = convertDataRow(data_row.toString());
            } else {
                ArrayList<Double> vector = convertDataRow(data_row.toString());
                    for (int j = 0; j < vector.size(); ++j) {
                        newCentroid.set(j, newCentroid.get(j) + vector.get(j));
                    }
            }
            Text outputKey = new Text(data_row.toString().split("\t")[0]);
            context.write(outputKey, clustID);
        }
        for (int i = 0; i < newCentroid.size(); ++i) {
            newCentroid.set(i, newCentroid.get(i) / count);
        }
        centroids.set(clustID.get(), newCentroid);
    }
}
```

5.2 Improvements

Normally in map reduce, to run a program iteratively the results of one iteration (or one map-reduce job) would be written to a file and in the subsequent iterations these results would have to be read again from the file that holds the intermediate results (results of one map-reduce job).

In our implementation, we do not write the intermediate results between map-reduce jobs to a file. Instead, we maintain two arraylists to hold the values of centroids of previous iteration and the centroids of current iteration.

The function that we have implemented to achieve this is as below

```
// deep copy centroids

public static ArrayList<ArrayList<Double>> deepCopyCentroids
    (ArrayList<ArrayList<Double>> centroids) {
    ArrayList<ArrayList<Double>> result = new ArrayList<>();
    for (int i = 0; i < centroids.size(); ++i) {
        ArrayList<Double> new_vec = new ArrayList<>();
        ArrayList<Double> vec = centroids.get(i);
        for (Double d : vec) {
            new_vec.add(d);
        }
        result.add(new_vec);
    }
    return result;
}
```

5.3 MapReduce kmeans Results Visualization and Evaluation

The result visualization and the Jaccard index is the same as non-parallel k-means (Figure 3, 4 and Table 2 respectively) because we chose the initial cantroids to be the the first k points. Therefore, we compare the performance based on the running time of the two implementation (Table 5.)

Algorithm	cho.txt	iyer.txt
k-means	54.941s	237.675s
MapReduce Kmean	29.376s	36.637s

Table 5: Comparison of the running time of kmeans and Map Reduce Kmeans (in sec)

As can be seen from the table, the map reduce implementation is much faster even for this small datsets. Escpecially in the case of iyer.txt which has more data points the perfomance is almost one order of magnitude faster in case of Map Reduce implemntation.

5.4 Pros and Cons of MapReduce K- means

Pros

1. Run time is much less when compared to run time of non-parallel version.
2. This version of k-means is scalable for larger datasets.

Cons

1. Implementation is not as simple as non-parallel version.
2. Requires good understanding of Hadoop environment to implement the algorithm efficiently.

6 Comparing the performance of Kmeans, Hierarchical and Dbscan algorithm

The best values of jaccard co-efficients were observed in case of Kmeans algorithm for both cho.txt and iyer.txt. But on visulaizing the results we found that the clusters identified by Kmeans were not very similar to the clusters based on the ground truth given in the original data files. We have listed the pros and cons of each algorithm and for these specific datasets the k-means performs better.