Clustering Algorithms

Report

Amitha Narasimha Murthy (50098028)

Neeti Narayan (50098029)

Vivekanandh Vel Rathinam (50098075)

This report describes three clustering algorithms to find clusters of genes that exhibit similar expression profiles. Our clustering results are validated using external, internal index and visualized using PCA. MapReduce K-means is also implemented. This report also includes snapshots of the output generated through this implementation.

Introduction

The objective of this project is to implement three clustering algorithms: K-means, Hierarchical Agglomerative with Single Link (Min), and density-based clustering. For each clustering algorithm, we validate our clustering results with

- External index (Rand Index, Jaccard Coefficient)
- Internal index (Silhouette)
- Visualize using PCA (Principal Component Analysis)

We have set up a single-node Hadoop cluster for implementing MapReduce K-means.

Section 1

Clustering Algorithms:

1. K-means

K-means is a partition-based clustering method. The algorithm takes an input parameter ‘k’ and partitions a set of ‘n’ data points in to ‘k’ clusters so that the resulting intra-cluster similarity is high whereas the inter-cluster similarity is low.

For this project, we have set *k* equal to the number of clusters in the ground truth labels. *k*=5 for cho.txt and 10 for iyer.txt.

- **Algorithm Flow**

The initial cluster centers (centroids) are specified. Each data point is then randomly assigned into one of the *k* clusters. The cluster centroid is updated by taking the mean of each cluster.

Iterate until convergence (stop when the difference in sum of squared error (SSE) is below threshold)

- For each data point x:
    - The Euclidean distances between x and the *k* centroids is calculated
    - x is reassigned to the cluster whose centroid is closest to x
- The cluster centroids are updated by calculating the mean of each cluster (based on cluster assignment)

- **Validation**

<u>For cho.txt</u>:

External index (using Rand Index): 80.74%

External index (using Jaccard Coefficient): 39.28%

Internal index (using Silhouette): 24.02%

<u>For iyer.txt</u>:

External index (using Rand Index): 78.72%

External index (using Jaccard Coefficient): 29.47%

Internal index (using Silhouette): 25.59 %

- **Visualization**

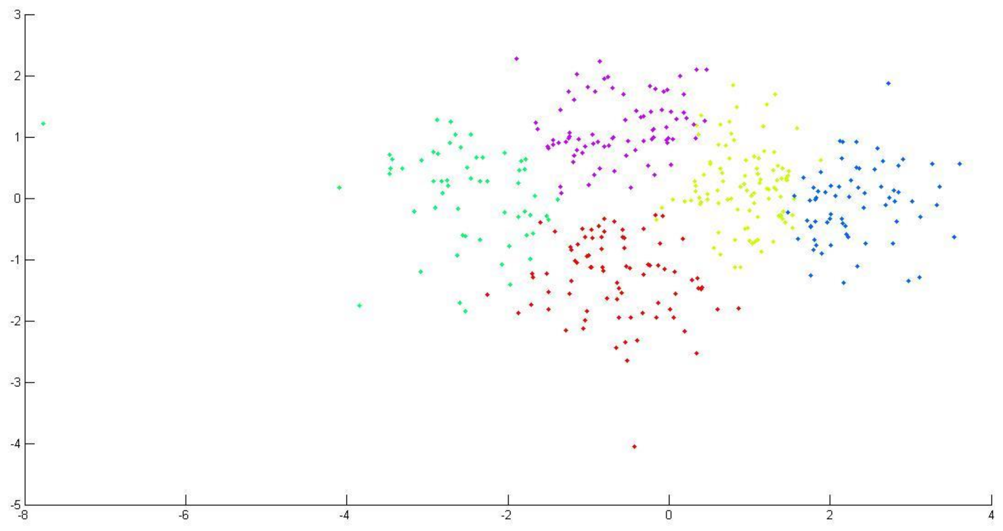


Fig 1: K-means on cho.txt

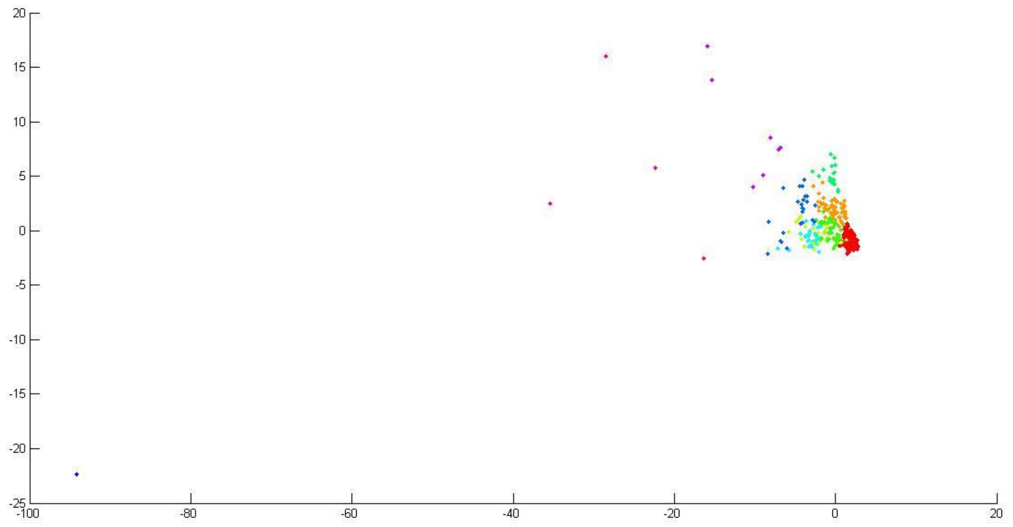


Fig 1: K-means on iyer.txt

- **Result Analysis**

  We observed that initializing with different centroids would give us different results. This is because K means algorithm is inherently sensitive to initial centroids. But we also observed that there is no definite formula for a good result with respect to initial centroids.

- **Pros**
  - It is easy to implement
  - For large number of objects, K-means is very efficient. Let *n* be the number of data points, *k* be the number of clusters, and *i* be the number of iterations. Then the complexity is in the order of O(*nki*)

- **Cons**
  - It requires that the number of clusters *k* be specified
  - Distance computations between the centroids and data points takes most of the execution time
  - It may not reach global minimum
  - It is sensitive to outliers and cannot handle varying sizes, densities or irregular shapes of clusters
  - There is a possibility of getting empty clusters
  - It cannot handle categorical data

2. Hierarchical Agglomerative (with Single Link – Min)

Hierarchical agglomerative clustering is a bottom-up approach.

- **Algorithm Flow**

First, the distance matrix is computed using Euclidean distance metric. Each data point is a cluster by itself.

Repeat until only a single cluster remains:

- Row *i* and column *j* with the smallest entry in the distance matrix is determined. The two closest clusters *i* and *j* are merged
- The distance matrix is updated. *i* and *j* together form one new cluster *N.* Cluster *N*'s distance from each of the remaining clusters, say *A* is computed as:
  - $d_{(i,j) \to A} = \min(d_{iA}, d_{jA})$

- **Validation**

For cho.txt:

External index (using Rand Index): 24.02%

External index (using Jaccard Coefficient): 22.83%

Internal index (using Silhouette): 24.27%

For iyer.txt:

External index (using Rand Index): 18.82%

External index (using Jaccard Coefficient): 15.82%

Internal index (using Silhouette): 62.92%

- **Visualization**

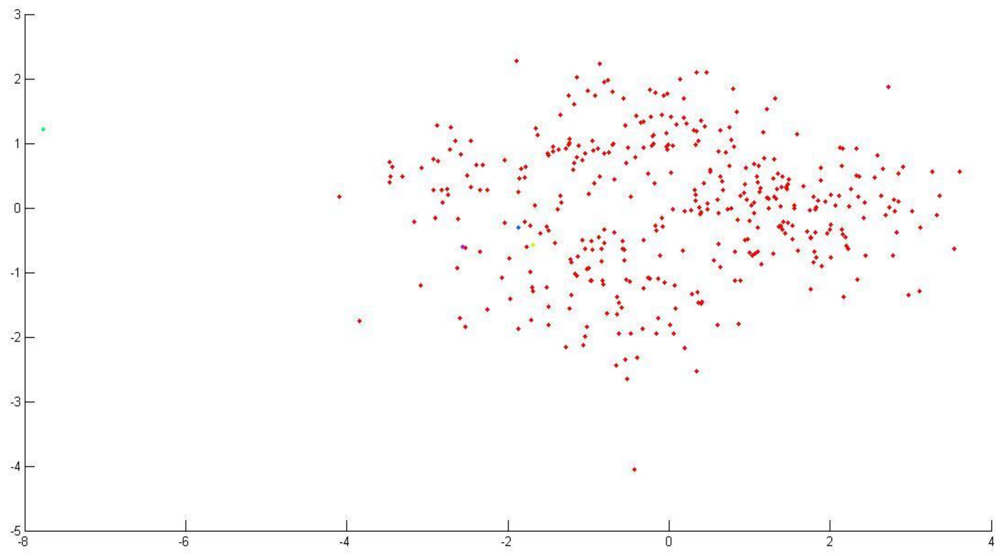


Fig 3: Hierarchical Agglomerative on cho.txt

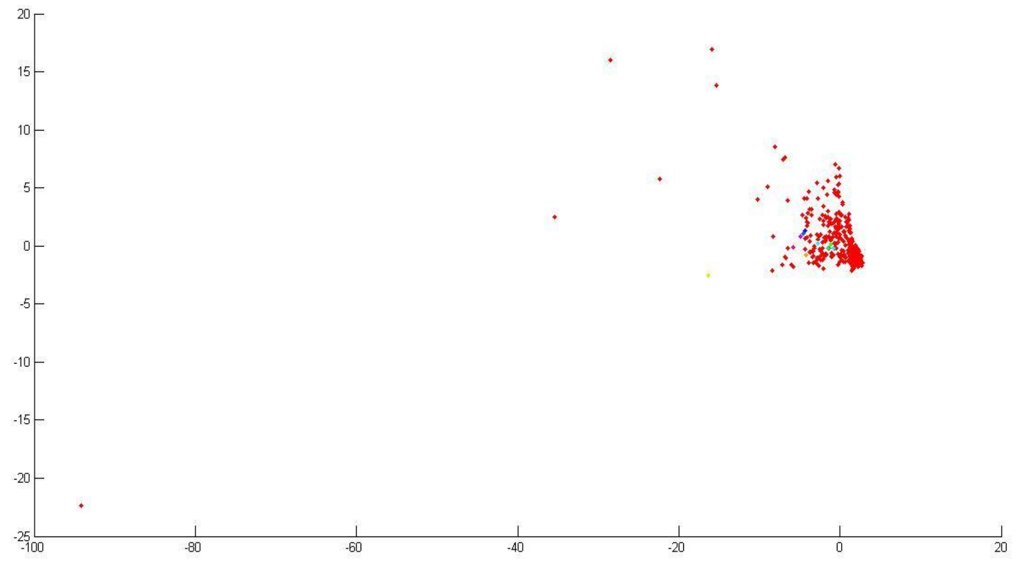


Fig 4: Hierarchical Agglomerative on iyer.txt

- **Result Analysis**

  With single link (Min), we find two “closest” clusters to merge in each step. For cho.txt, 382 data points are assigned to one cluster, remaining 4 data points form 4 individual clusters. This is the effect of using MIN as the approach to define cluster distance. On changing to MAX, we obtained better results.

  Also, we performed normalization on the data. But, it did not make any significant difference.

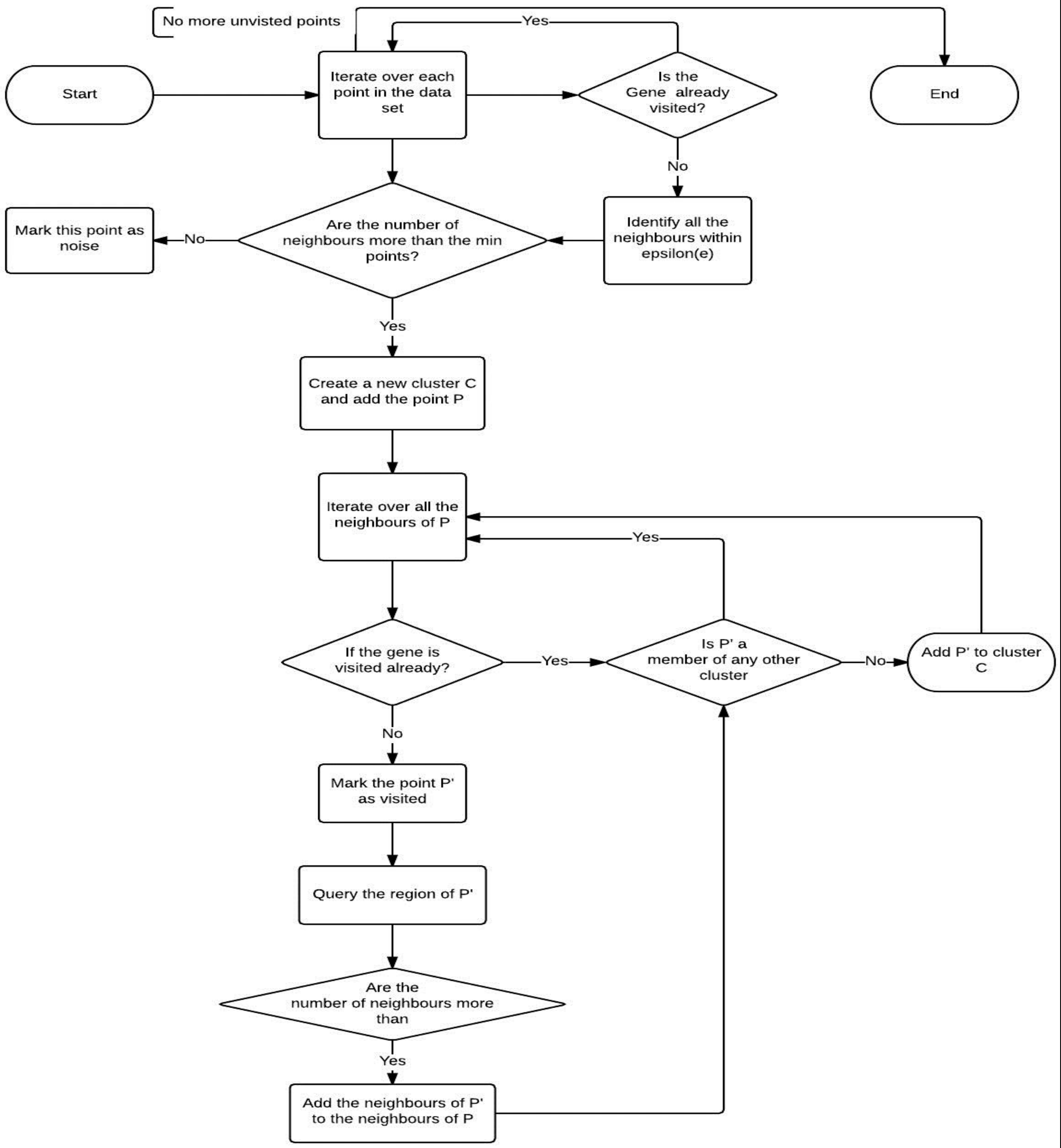
  Non-outliers and outliers are not distinguished when applying the clustering algorithm. All records are considered as our input.

- **Pros**
  - It can handle non elliptical shapes
  - It does not require that the number of clusters be specified (unlike K-means)

- **Cons**
  - It is sensitive to noise and outliers
  - It is slow compared to K-means. Merging is time consuming.

3. Density-based (DBSCAN)

- **Algorithm Flow**



No more unvisted points
Yes
Start
Iterate over each point in the data set
Is the Gene already visited?
End
No
Mark this point as noise
No
Are the number of neighbours more than the min points?
Identify all the neighbours within epsilon(e)
Yes
Create a new cluster C and add the point P
Iterate over all the neighbours of P
Yes
If the gene is visited already?
Yes
Is P' a member of any other cluster
No
Add P' to cluster C
No
Mark the point P' as visited
Query the region of P'
Are the number of neighbours more than
Yes
Add the neighbours of P' to the neighbours of P

- **Validation**

<u>For cho.txt</u>:
External index (using Rand Index): 70.15%
External index (using Jaccard Coefficient): 28.27 %
Internal index (using Silhouette): 5.79%

<u>For iyer.txt</u>:
External index (using Rand Index): 68.12%
External index (using Jaccard Coefficient): 29.02%
Internal index (using Silhouette): 40.84%

- **Visualization**

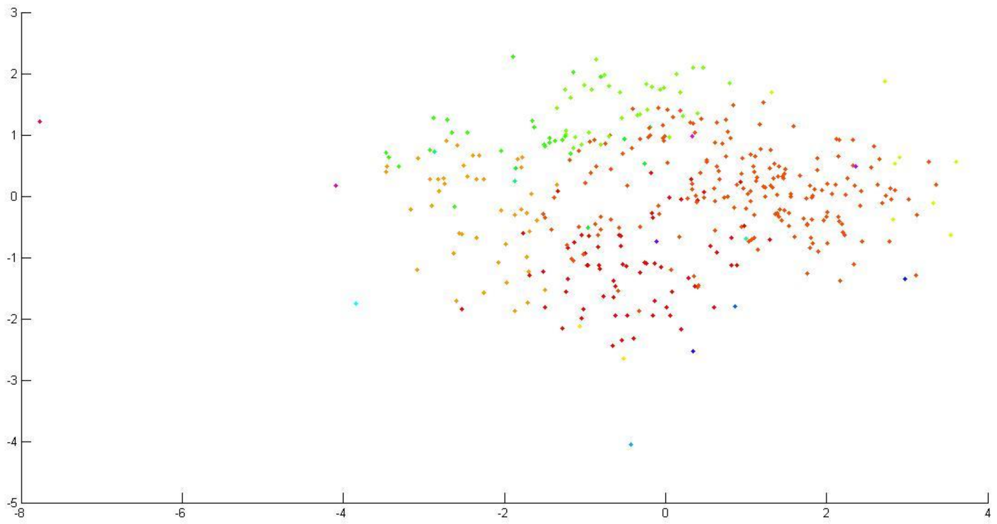


Fig 5: DBSCAN on cho.txt

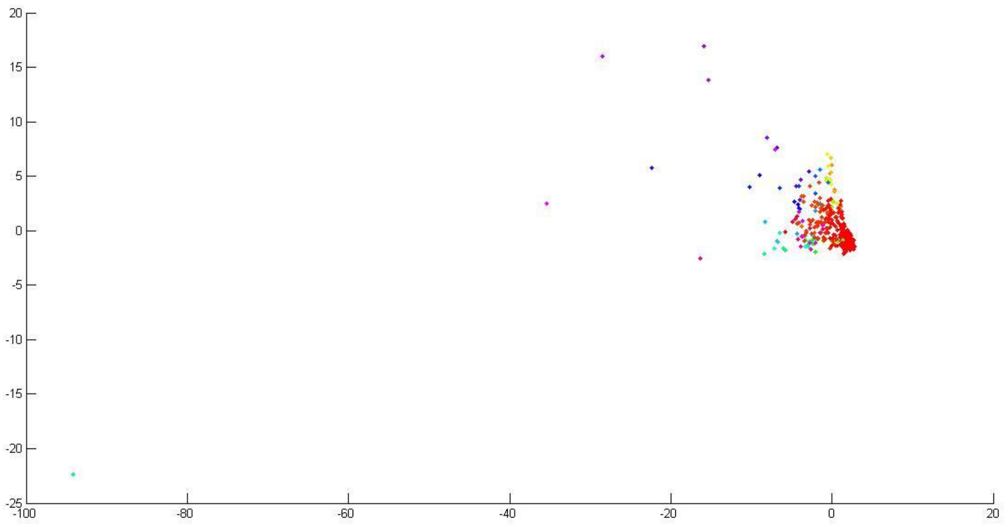


Fig 6: DBSCAN on iyer.txt

- **Result Analysis for different combination of Epsilon and MIN points**

Epsilon	**Number of min points**	**Rand Index**	**Jaccard Coefficient**	**Silhouette Coefficient**
2	55	68.21	30.10	0.27
2.5	55	69.53	30.34	0.367
3	55	68.12	29.02	0.408
2	65	68.21	30.10	0.28
2.5	65	69.53	30.40	0.365
3	65	68.12	29.02	0.4084

From the above table, we can deduce that Epsilon of 3 gives very good performance for clustering the data set. We tried and tested our algorithm with many epsilon values. Epsilon values lesser than 2 and greater than 4 resulted in 30 or more clusters and 2 clusters respectively which is bad.

The number of Minimum points and the Epsilon values are proportional to each other as we can see from the table i.e. Clusters cannot be formed with more points and lesser diameter (epsilon)

- **Pros**
  - It is resistant to noise
  - It can handle clusters of different shapes and sizes (unlike K-means)

- **Cons**
  - It is sensitive to parameters
  - It cannot handle clusters of varying densities

Section 2

MapReduce K-means

The most intensive calculation in the K-Means algorithm is the calculation of distances. Every iteration requires a total of (nk) computations where n is the number of data points and k is the number of clusters. The distance computations of one objects with its center is irrelevant to distance computations of other data points with their corresponding centers. Therefore, the distance computations of data points with their respective centers can be executed in parallel.

At each iteration, the new centers that are used in the next iteration should be updated. Therefore, this iterative procedure must be executed serially.

Mapper:

Input: <key,value> pairs, where each <key,value> pair represents a record. key is the offset in bytes of the record. The value is a string of the contents of the record.

Output: <key1,value1> where key1 is the centroid to which the data point belongs to. Value1 is the data point.

1. Populate a static data structure called clusterCentroids with the initial centroids
2. Calculate the Euclidean distance between the data point and every centeroid in clusterCentroids
3. Label the data point with the cluster centroid of the nearest cluster
4. Output<key1,value1>

```
5.  End
```

Reducer:

Input: <key1,V> where key is any centroid and V is the list of data points having the centroid 'key1'

Output: <key2, value2> where key2 is the new centroid and value 2 is the string comprising of all the data points with this new centroid

```
1. Initialize an arraylist to hold the sum of each dimension for all the data points belonging
   to that cluster.
2. Initialize a counter NUM to keep track of the data points belonging to the same cluster
3. for(String point: V)
           Update the sum in arraylist by adding the dimensions of this point to the
           corresponding indices
           NUM++
4. Divide each of the entries in the arraylist by NUM to get the new centroid
5. Output <new centroid, datapoints>
6. End
```

- **Validation**

  <u>For cho.txt</u>:

  External index (using Rand Index): 80.82%

  External index (using Jaccard Coefficient): 43.00%

  Internal index (using Silhouette): 24.23%

  <u>For iyer.txt</u>:

  External index (using Rand Index): 73.02%

  External index (using Jaccard Coefficient): 32.99%

  Internal index (using Silhouette): 48.03%

- **Visualization**

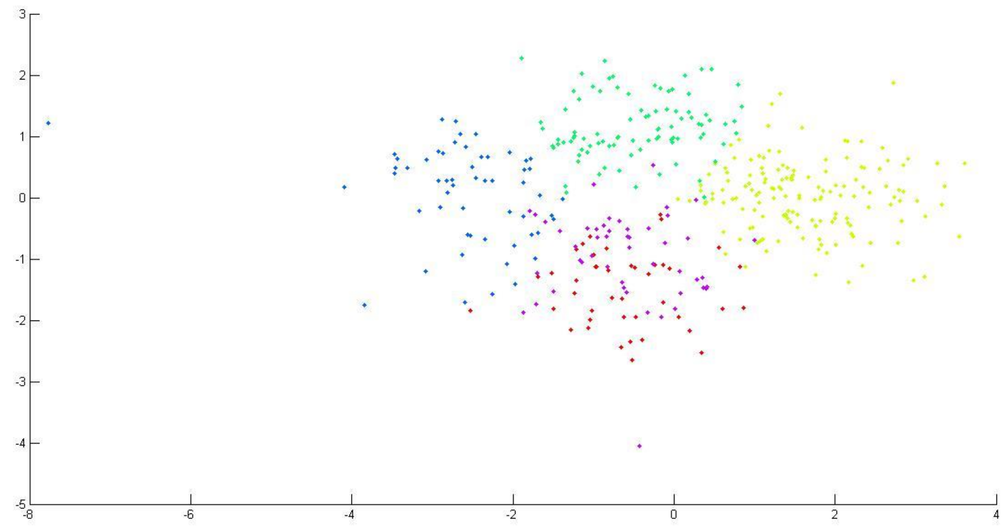


Fig 7: K-Means MapReduce on cho.txt

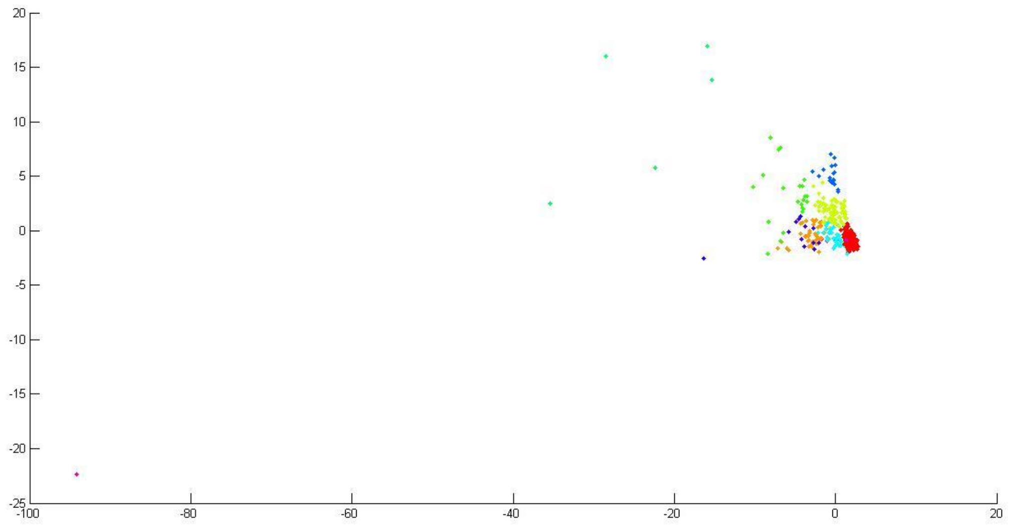


Fig 8: K-Means MapReduce on iyer.txt

- **Result Analysis**
  We observed that the results of the non-parallel K-Means and parallel K-Means are very similar. However, for these particular datasets, MapReduce implementation of K -Means took longer time than the non-parallel K-means which leads to our conclusion that MapReduce is efficient only for large datasets.

- **Pros**
  - Performs well for large datasets
  - We don’t have to be bothered about the nitty-gritty of parallelization, synchronization and concurrence
  - Simple to implement
- **Cons**
  - Does not perform well with small datasets like the ones used in this project
  - Takes a lot of time to set up the environment
  - Takes higher number of iterations to converge than non-parallel K-Means