**CSE 601 – Project 2**

**Clustering Algorithms**

**Akash Mandole 50168447**

**Bhavani Sundara Raman 50169253**

**Harsha Sudarshan 50169593**

**Motivation**

Finding groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects in other groups.

**Applications**

* Explore data distribution.
* Pattern recognition, spatial data analysis, image processing, market research, WWW, …
* Cluster documents
* Cluster web log data to discover groups of similar access patterns
* Marketing: Help marketers discover distinct groups in their customer bases, and then use this knowledge to develop targeted marketing programs
* City-planning: Identifying groups of houses according to their house type, value, and geographical location
* Climate: understanding earth climate, find patterns of atmosphere and ocean

**Data Representation**



**Clustering Requirements**

* Scalability
* Ability to deal with different types of attributes
* Minimal requirements for domain knowledge to determine input parameters
* Able to deal with noise and outliers
* Discovery of clusters with arbitrary shape
* Insensitive to order of input records
* High dimensionality
* Incorporation of user-specified constraints
* Interpretability and usability

**Cluster Validation**

Comparing clustering results to Ground Truth.

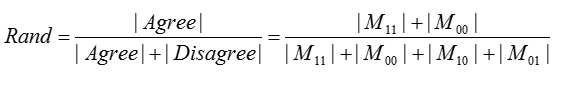
Let *P*={*P*1,…,*Ps*}: the set of “ground truth” clusters and *C*={*C*1,…,*Ct*}: the set of clusters reported by a clustering algorithm.

Create incidence matrix such that, *Pij*= 1 if *Oi* and *Oj* belong to the same “ground truth” cluster in *P*; *Pij*=0 else *Cij* = 1 if *Oi* and *Oj* belong to the same cluster in *C*; *Cij*=0 otherwise

A pair of data object (Oi,Oj) falls into one of the following categories

* M11 : Cij=1 and Pij=1; (agree)
* M00 : Cij=0 and Pij=0; (agree)
* M10 : Cij=1 and Pij=0; (disagree)
* M01 : Cij=0 and Pij=1; (disagree)

**Rand index**

****

**Jaccard Coefficient**



**Clustering Algorithms, Results an Evaluation**

**K-Means**

K-means is a simple unsupervised learning algorithms. It follows a simple and easy way to classify a given data set through a certain number of clusters assuming **k** clusters. The idea is to define k centroids, one for each cluster. A better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early clustering is done. At this point we need to re-calculate k new centroids of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new centroid. As a result of this loop we may notice that the k centroids change their location step by step until no more changes are done.

**Algorithm Description**

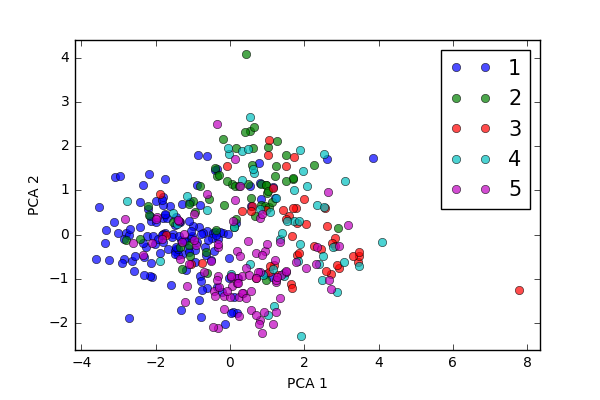
1. Partition points {*x1*,…,*xn*} into *K* clusters where *K* is predefined
2. Specify the initial cluster centers (centroids)
3. Repeat 4-7 until no change
4. For each object *xi*
5. Calculate the distances between *xi* and the *K* centroids
6. (Re)assign *xi* to the cluster whose centroid is the closest to *xi*
7. Update the cluster centroids based on current assignment

**Implementation notes:**

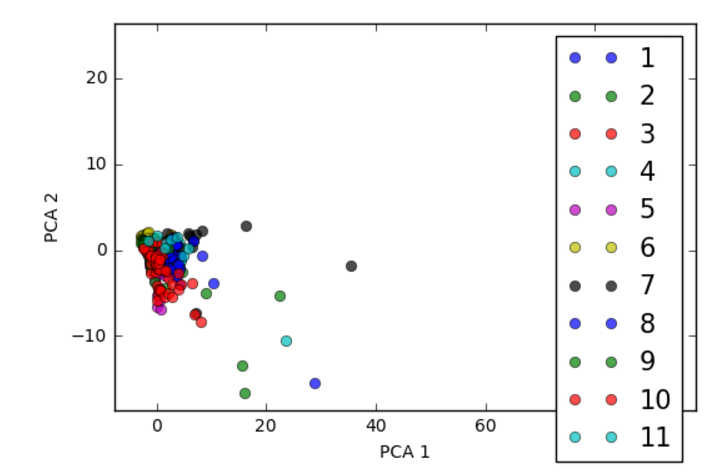
* We created a n\*n euclidean distance matrix for reference.
* Initialisation of centroids were random sampling of input indexes

**Result Visualization**

Data set: cho.txt



Data set: Iyer.txt

****

**Result Evaluation**

Data set: Iyer.txt

Jaccard coefficient = 0.35009671179883944

Data set: cho.txt

Jaccard coefficient = 0.3400

**Pros and Cons of algorithm**

* Pros: Efficient: O(*tkn*), where *n* is # objects, *k* is # clusters, and *t* is # iterations. Normally, *k*, *t* << *n*. Easy to implement
* Cons: Need to specify *K*, the number of clusters. Local minimum– Initialization matters. Empty clusters may appear

**Hierarchical agglomerative clustering**

Hierarchical clustering algorithms are either top-down or bottom-up. Bottom-up algorithms treat each document as a singleton cluster at the outset and then successively merge (or agglomerate) pairs of clusters until all clusters have been merged into a single cluster that contains all documents. Bottom-up hierarchical clustering is therefore called hierarchical agglomerative clustering or HAC. Top-down clustering requires a method for splitting a cluster. It proceeds by splitting clusters recursively until individual documents are reached.

**Algorithm Description**

* 1. Compute the distance matrix
  2. Let each data point be a cluster
  3. **Repeat**
  4. Merge the two closest clusters as per MIN or Single Link approach
  5. Update the distance matrix
  6. **Until** only a single cluster remains

**MIN or Single Link approach**

The distance between two clusters is represented by the distance of the closest pair of data objects belonging to different clusters. Determined by one pair of points, i.e., by one link in the proximity graph. Such that

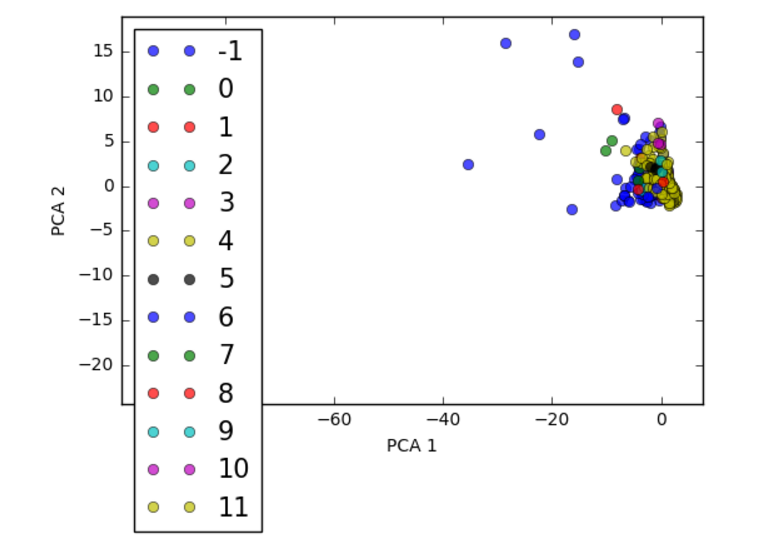


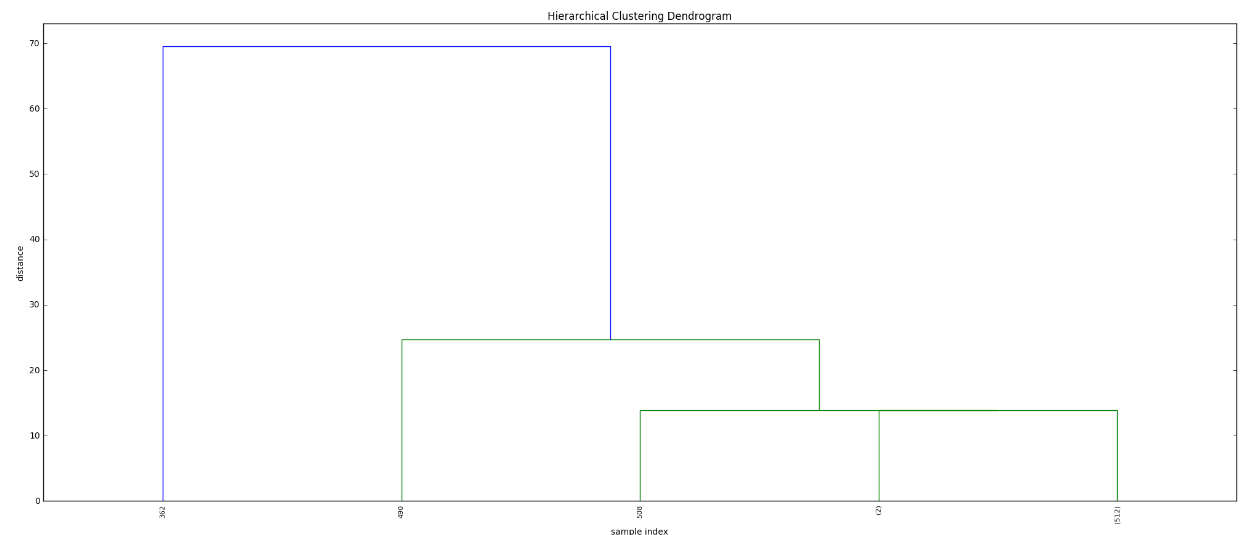
**Implementation notes:**

* We created a n\*n euclidean distance matrix for reference.
* We have taken a cut off distance from top of 75% of max distance value

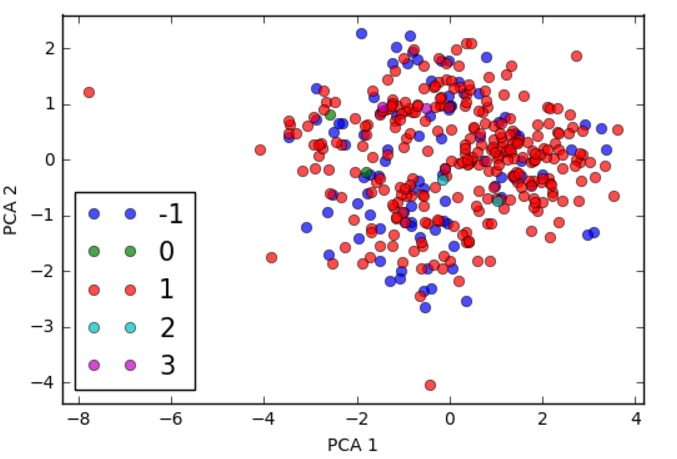
**Result Visualization**

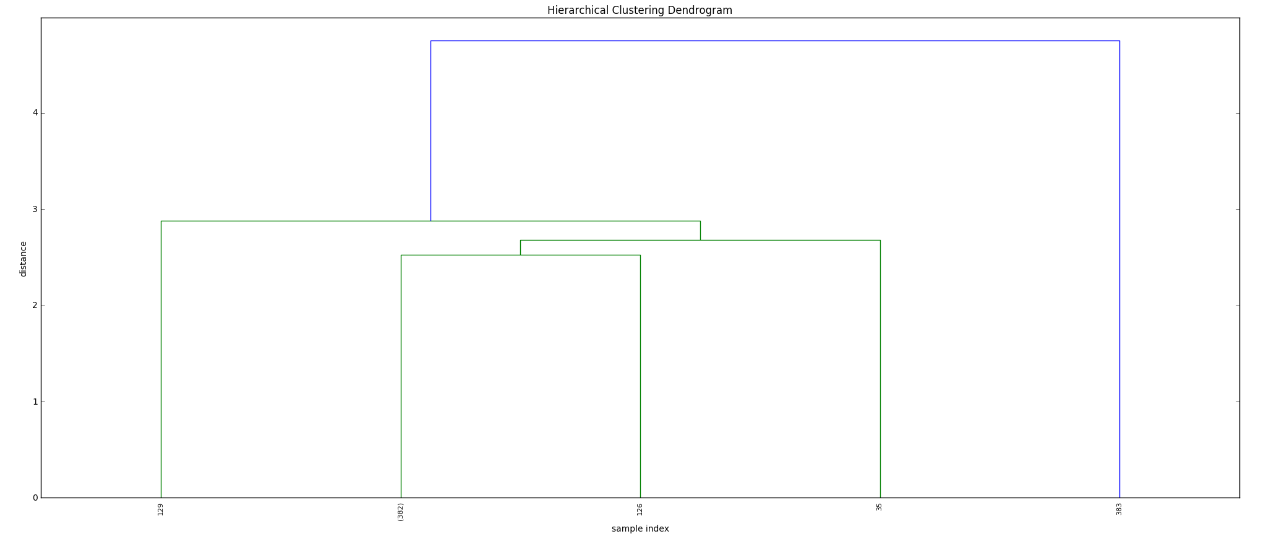
Data set: Iyer.txt

****

****

Dataset: cho.txt





**Result Evaluation**

Data set: Iyer.txt

Jaccard coefficient = 0.079303675048355893

Data set: cho.txt

Jaccard coefficient = 0.098445595854922283

**Pros and Cons of algorithm**

* Pros: Can handle non-elliptical shapes
* Cons: Sensitive to noise and outliers

**Density-based Clustering**

It is a density-based clustering algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away). Density definition is given by two parameters.

* ε-Neighborhood – Objects within a radius of *ε* from an object.
* “High density” - ε-Neighborhood of an object contains at least *MinPts* of objects.

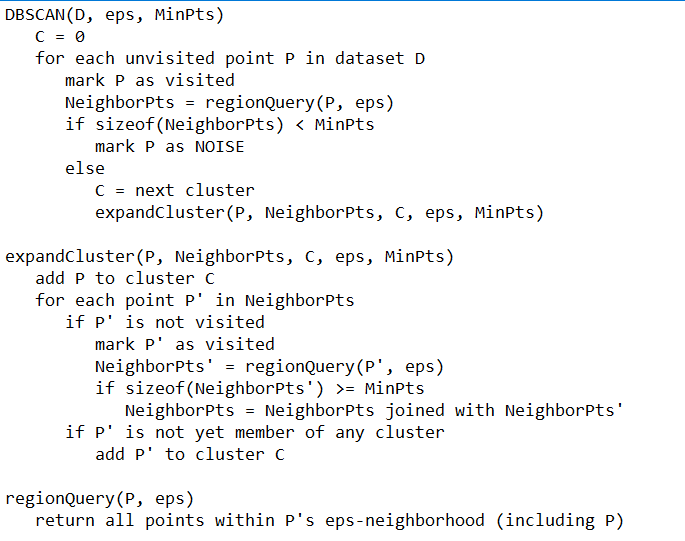
A point can be classified as one of these three:

**Core point:** if it has more than a specified number of points (MinPts) within Eps—These are points that are at the interior of a cluster.

**Border point:** if it has fewer than MinPts within Eps, but is in the neighborhood of a core point.

**Noise point:** if it is any point that is not a core point nor a border point.

**Algorithm Description**

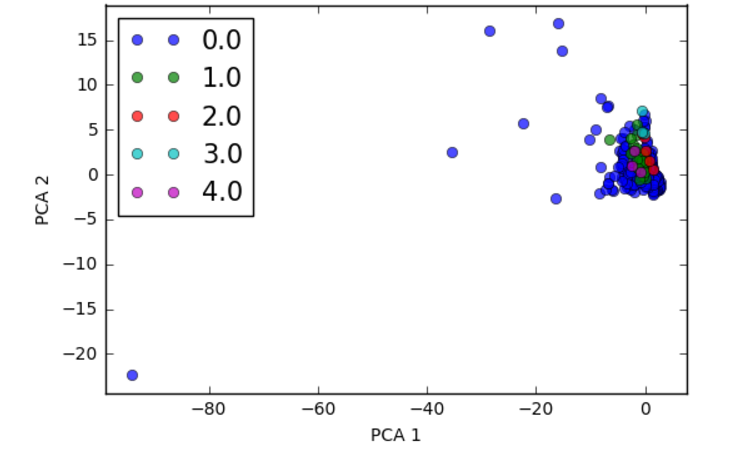
****

**Implementation notes:**

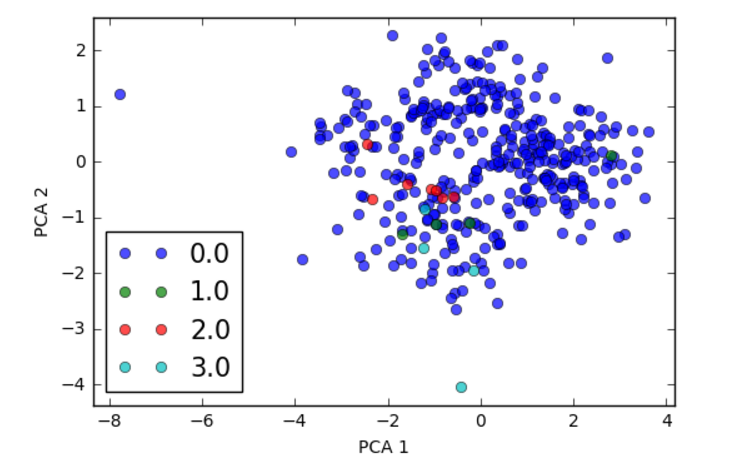
* We have used dfs depth first search algorithm to find the reachable points from core points

**Result Visualization**

Data set: Iyer.txt



Data set: cho.txt



**Result Evaluation**

Data set: Iyer.txt

Jaccard coefficient = 0.1200456782500

Data set: Cho.txt

Jaccard coefficient = 0.0077720207253886009

**Pros and Cons of algorithm**

* Pros: Resistant to Noise. Can handle clusters of different shapes and sizes.
* Cons: Cannot handle varying densities. Sensitive to parameters—hard to determine the correct set of parameters.

**K-Means on MapReduce**

The efficiency of K Means clustering can be leveraged if used along with MapReduce programming style. MapReduce is a programming style that is used for handling high volume data over a distributed computing environment. If we use mapper and combiner functions along with KMeans we can decrease the number of read write operations to a great extent. Clustering algorithm on MapReduce performs better than normal implementation of K Means and reduces time of computation as low as 21 ms.

**Algorithm Description**

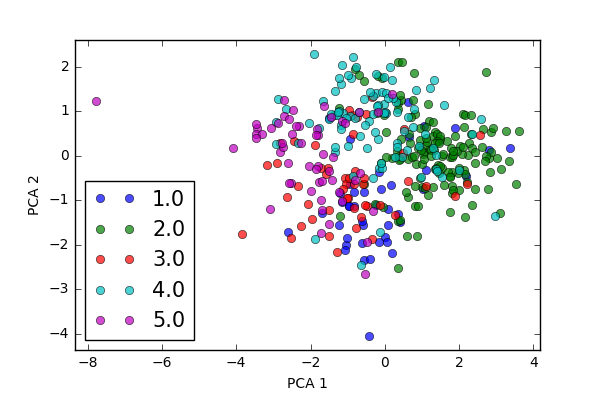
We being with specific centroids and compute the Euclidean distance of each input data with the centroid. We associate the input data with the centroid that is closest to the input data. In doing so we associate each input data with one of the centroids. The centroid and the input data associated with it are treated to be one cluster.

We then re compute the centroid value to be the average of all the points in that cluster. The centroid of each cluster has moved now. We repeat the same process again till the centroids no longer move. We have reached convergence at this stage and complete our algorithm.

**Implementation notes:**

In hadoop environment we create jobs to perform various tasks. Our tasks are divided between mapper and reducer. We give the location where the code must be shipped if this job is to be run across clusters. We determine the datatype of keys and values of output of the mapper and reducer and set them accordingly. In our program we are outputting text. As the job starts we begin out timer. The mapper reads the input file line by line and be parse the line based on our separators and populate input data map and ground truth values. The job of the mapper ends when it had reached end of file. The reducer takes this input data and takes in the given centroid values and begins our algorithm. We have used HashMaps to store input data key as geneids and value as the list of attributes. Cluster data contains the centroid and its coordinates. Cluster reference map is used to store the centroid ids and the nodes associated with that centroid. When the centroid are converged we end the algorithm. We write our output to a file and record the system. The total time to run is calculated as difference of end time and start time. We then use this text file with output to calculate Jaccard coefficient.

**Result Visualization**

****

**Result Evaluation**

Jaccard coefficient = 0.5777202072538806

**Pros and Cons of algorithm**

* Pros: With the use of mapper Reading/Writing to a file is fast. MapReduce is linearly scalable
* Cons: Difficult to implement. Need to specify *K*, the number of clusters. Local minimum– Initialization matters. Empty clusters may appear