CSE 601: Data Mining and Bioinformatics
Project 2

Clustering of Gene Data:

K-means clustering

Hierarchical clustering

Density based clustering

K-means clustering using MapReduce

University at Buffalo

Prashanth Seralathan(pseralat)	#50204883
Manishkumarreddy Jarugu(mjarugu)	#50206843
Haneesh Reddy Poddutoori(haneeshr)	#50208280

K-means Clustering:

K-means clustering is the most famous unsupervised learning algorithm that has it’s application spread over various fields. K-means generally revolves around the idea of picking centroids at random and assigning the points to one of the chosen cluster based on it’s proximity to a cluster - The proximity is generally measured based out on Euclidean distance, but other distance functions such as mahalanobis distance, cosine distance, correlation distance, etc. In general, the Euclidean distance works well in most of the cases and for our case of clustering the gene dataset we have chosen that.

The k-means works on the basic idea of **Expectation maximization** approach, In the initial iteration the points are assigned to the centroids to which it is closest and after an iteration we tend to recalculate the centroid based on the points that are currently clustered, this will give an idea centroid - so this process of iteration continues until we find the most optimal centroid. The centroid recalculation revolves around the idea of calculating the mean of a points that are present in a particular cluster and use the new value of centroid for the next iteration. This process continues until certain number of iterations or till all the values of old centroid converge with the newly calculated ones.

The good analogy of why we are doing this can be explained considering a problem of opening couple of pizza shops in the neighborhoods, Initially you might start the centroids considering it to close to couple of neighborhood houses and then the centroid recalculation continues until the certain number of iterations or until the centroid value does not change - so as an eventual result we will get an clusters that ideally separates two neighborhoods with pizza shop locations converging centrally at each of the neighborhoods respectively.

There are few issues in k-means clustering as the results are always not so optimal,
So these are few things that needs to be kept in mind before performing k-means clustering,

1) ***What is the ideal number of clusters to start with?*** - This is one the biggest questions in k-means and there is no defined answer for it. In general the distortion increases with fewer number of clusters and distortion decreases with large number of clusters. So we can keep increasing the number of clustering to compare the performance of the algorithm - At one point the number of distortion will stop decreasing and remains fairly constant and that can be chosen for ideal k - This rule is popularly called as **"Elbow method"**.

2) Visualizing the clusters to get an approximate idea of number of clusters - This works very well when the data is two dimensional, we can identify the number of clusters based on how the data is distributed. For multi-dimensional points the visualization can be done using dimensionality reduction methods such as PCA but still identifying the number of clusters has to be combined with the elbow method.

Implementation details:
Program: K-means.py

Point object:

Each point from the input file is considered as a point object, the point object consists of the following properties,

- x - data points of the object,
- True cluster number associated with the point.
- Cluster number that is yet to be determined.

Main Iteration:

kmeans(filename, clusters):

This method takes the filename and number of clusters to be generated, The initial centroid can be taken randomly or the centroid id's can be given via input. The input file is read here and point objects are initialized here.

1. At this point we start the loop of clustering and call the method *performCluster*
2. Once the points have been assigned cluster that is near to it then we call *reCalculateCentroids*.
3. Continue steps 1 and 2 until the centroid converge or the number of max-iterations are reached.

<u>*Helper methods:*</u>

performCluster(dataPoints, clusterPoints):

This is a helper method that is used for calculating the closest centroid to a particular point. It iterates over the current centroid values and keeps track of the closest centroid near which the current point is located - It updates the cluster id to newly found cluster.

The Euclidean distance is obtained by using the helper method *distance* - The convergence condition is also checked here by looking for the condition at which no point is reassigned to a new cluster.

reCalculateCentroids(dataPoints, clusterSize):

It iterates over the datapoints and keeps a running sum of point values for respective clusters. The new centroid is calculated as *(calculated sum for each centroid /no of points in that cluster)* - the process of mean recalculation.

distance(point1, point2):

Gives the euclidean distance between two given points.

Observations running on gene data:

Specifically, to our gene data we observed the **"elbow method"** holded true, With lesser number of clusters the data was distorted and the performance was not very ideal. With the ideal number of clusters we got an optimal results but the jaccard coefficient wasn't really great.

We further researched about it to figure out that gene data having large number of dimensions the cluster cannot be always attributed to a spherical cluster and k-means in general doesn’t have a great performance data that is distributed in non-spherical manner - The unevenly sized clusters is one of the other reasons for reduced performance of k-means. The performance of k-means was even worse for the second dataset ‘iyer.txt’ which has outliers in it, This is because the outlier distort the centroids that are calculated and there is no pre-processing step to eliminate the outliers.

One alternative to k-means algorithm is using **K-means++** where the initial centroids are not chosen at random instead they are chosen based on weighted probability distribution. The runtime performance on the given datasets were good, but given a large dataset k-means has the possibility of degraded performance due to repeated centroid recalculations.

Pros:

- It’s an simple unsupervised algorithm that works efficiently when the original data has clustered data internally and density of points distributed equally in the clusters.
- K-means is practical algorithm which can be adapted to multiple applications with simple modifications and it’s easier to interpret the results.
- K-means is generally good in terms of computational cost.

Cons:

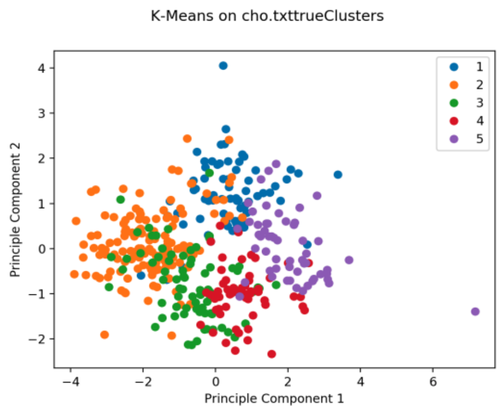
- Finding out correct number of clusters to initialize things has to be via PCA or elbow method.
- K-means performs poorly when the points that differ in densities.
- K-means is sensitive to outliers.
- The centroids might get give poor results because of local maxima.

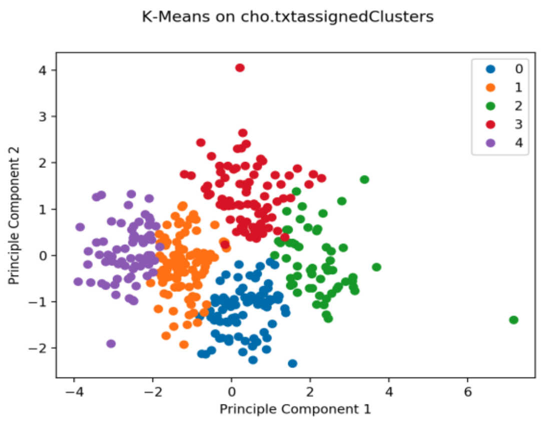
Jaccard Coefficient comparison:

Cluster Points	**DataSource: cho.txt**
3	0.40885748863376 18
4	0.4102173664775971 6
5	0.4216819449980071 5
6	0.3003094662766544
7	0.2772860031453606

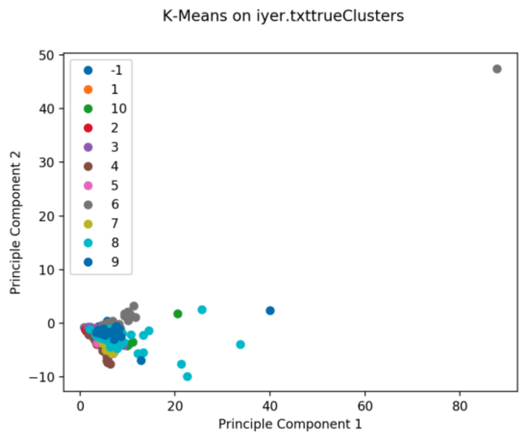
Cluster Points	**DataSource: iyer.txt**
8	0.305953387478329 2
9	0.3069061313260893 5
10	0.3187908154436075
11	0.2970179992444460 5
12	0.3096866796839641 3

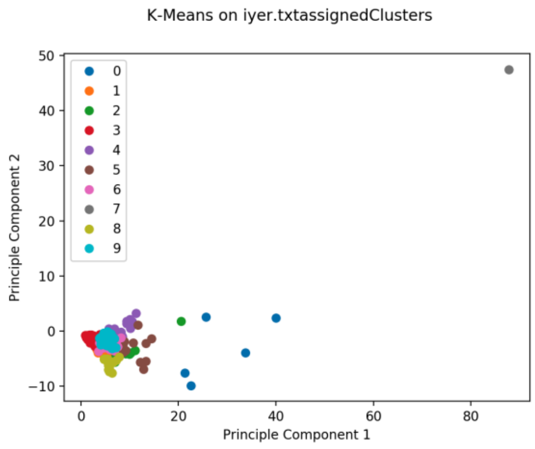
True & assigned clusters on cho.txt:





True & assigned clusters on iyer.txt:





Visually the clustering looks good on cho.txt, it clustered the points together based on the relative distance between the points, on the other hand iyer.txt since the points are relatively close to each other - so Kmeans mis classifies the data. So in essence k-means performs well when the data has good level of separation between them.

Hierarchical Clustering:

Hierarchical Clustering is a clustering algorithm in which clustering is done by maintaining clusters at different levels which works based on greedy algorithm of finding an optimal suboptimal solution, in other words maintain hierarchy of clusters. The hierarchy of clusters can be maintained in two different ways:

1) Agglomerative (Bottom Up):

Each object is initially a cluster, and the close proximity clusters are merged into one cluster. This merging is continued until there are K clusters remaining. In other words it's like merging many number of clusters to K clusters.

2) Divisive Approach(Top Down):

Initially all the objects are considered to be in one single cluster and then each cluster is split into 2 clusters. This process is repeated until we have K clusters. So what's eventually happening is all the objects are considered to be in one single cluster and kept on dividing into two, until we have k clusters.

We use a bottom up Agglomerative approach for our implementation:
So the algorithm proceedings are as follows:

1) Each object is assumed to be a cluster. So let’s say there are 5 objects A,B,C,D,E. Now we define them as clusters.
2) We need to calculate distance matrix for all the given clusters. So the distance matrix will be a 5x5 matrix and contains the distance between every pair of

clusters. So the diagonal matrix doesn't contain represent the distance between the same clusters which is marked as 0.

3) Closest two clusters can be merged into one cluster. So let's say cluster B and C are close and they are merged together to form one cluster say BC. So now the cluster set will become A, **BC**, D, E.
4) The distance matrix is now calculated for these 4 clusters (4x4 matrix) and the step 3 is repeated.
5) As we have exactly K clusters, the process is terminated.

This approach can be visualized as constructing of dendrograms in each of the phases And the most important thing to be noted is that while calculating the distance matrix, for the distance between two clusters, the distance between the closest proximity data objects which belong to different clusters are used.

So it can be something like,

Distance between
cluster A and cluster B = Min(distance between point a in A and point b in B)
It can be represented as follows,

$$d_{min}(C_{i,}C_j) = min_{P \exists C_{i,} Q \exists C_j} distance(P, Q)$$

The main advantage of using this type of approach of calculating distance between two clusters based on the min distance between points between two clusters is being able to detect non-elliptical shape clusters.

The major disadvantage of this method is that, this cannot detect noise and presence of outliers.

Implementation Details:

Class Point:

We define a class Point from which all points are instantiated as objects. Each object has four different properties,

x - data points of the object,

clusterNumber- initially None,

trueCluster - number associated with the point,

And has two methods: getTrueCluster for getting the true cluster number, getPointValue for getting actual data point.

Distance function:

When we are given two points, say point1 and point2 to this function, it calculates the difference of the array and then square each element of the array and then calculate the sum of the entire array and then finally return the square root of the summed value. Basically it returns the euclidean distance between the given two points.

findNeighbours function:

When we are passed a point and epsilon value, all the neighbours within the distance epsilon are being calculated and added to a list. This neighbourhood list is returned as a return value.

hierarchical function:

Initially each and every point is labelled as cluster, then we find two clusters for which the distance between the cluster is minimum and then we finally merge these two clusters. Then the same process is repeated until the number of clusters is equal to k.

Observations running on gene data:

The performance wasn’t really great on the gene data - We observed a weird pattern of points had really close values with each other and it wasn’t able to segregate the clusters between the data points, hence we had an observation where the data points clustered together in a single connected component.

The single connected component can be more clearly explained based on observing the data in “iyer.txt” - the entire gene data provided in the file is very close to each other, so after calculating the euclidean distance it gets clustered to the already formed component. Some of the outliers which have a large distance variation doesn’t get connected to the dendrogram and remains unconnected.

Some of the example points in “iyer.txt” that doesn’t get combined are,

```
491 9  1.0    1.18   5.53   11.9   3.86   28.22  20.07  16.54  9.58   3.3    5.88   8.33
442 8  1.0    0.88   0.91   1.27   1.4    4.48   17.43  16.41  9.5    7.37   4.45   3.97
```

From the above observations we can infer that hierarchical clustering do not work well on points that are closely related to each other, they tend to consider them as a single connected component in dendrogram.

Pros:

- Hierarchical clustering can give different partitionings depending upon the level that we are looking at it.
- Hierarchical clustering works well on the data that is small.

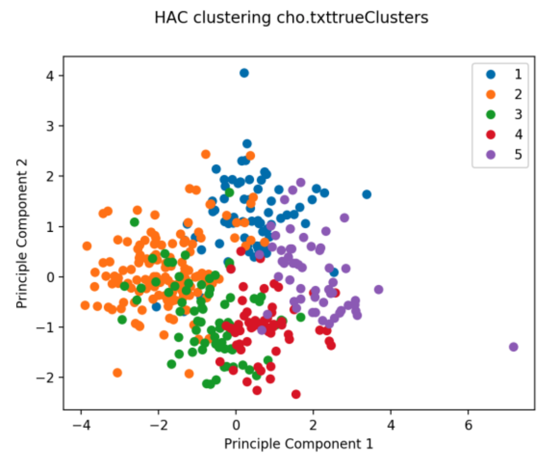
Cons:

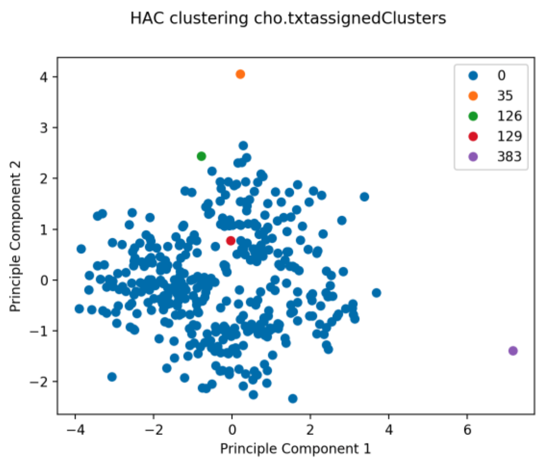
- Hierarchical clustering doesn’t give great results in the presence of outliers and noise.
- Hierarchical cluster doesn’t have a great performance on large datasets and the distortion is high.

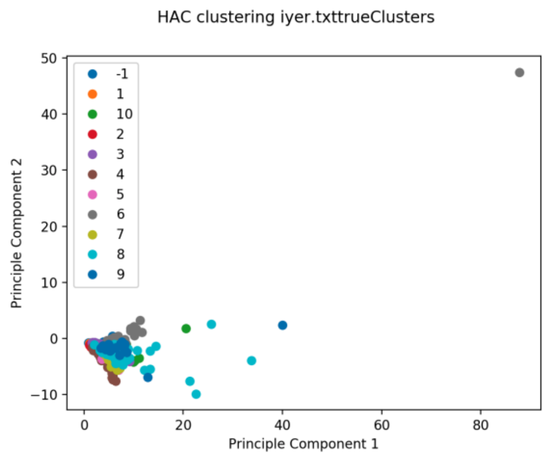
Jaccard coefficient:

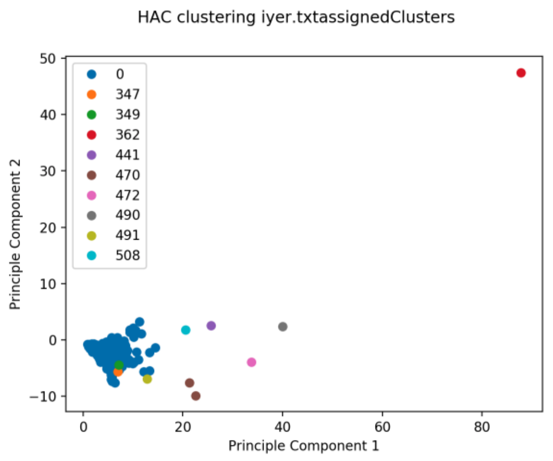
Cluster Points	**DataSource: cho.txt**
3	0.22933831189036352
4	0.22942934782608695
5	0.22839497757358454
6	0.22849318819743958
7	0.22879187984815977

Cluster Points	**DataSource: iyer.txt**
8	0.15710452461338867
9	0.1582508564533713
10	0.15824309696642858
11	0.1584602101134175
12	0.15845239630094118









From the above PCA plots we can say that hierarchical clustering did not give the most optimized performance since both the gene data given here are closely similar to each other, The plots would have been interesting if the datasets had two separate groups of similar points, ideally the separation between the two group of points would be captured by HAC. Comparing the runtime, HAC is the slowest as it considers each individual point as individual clusters and clusters them recursively until certain number of points.

Density Based Clustering:

Density based clustering in other words Density-based spatial clustering of applications with Noise, shortly known as DBSCAN is an approach in which, from the given points in a spatial system, the points which are closely occurred together are combined together forming a cluster and the points which are loosely present are marked as outliers or infrequent points. This clustering algorithm is one of the most commonly used clustering algorithms of all times.

The set of neighbours to a point is called as Neighbourhood. The set of neighbourhood points within distance ε is called ε-neighbourhood points.

In this approach, two main parameters are considered. They are ε and MinPts. ε is nothing but the maximum distance within which if the points are present, they will be considered as neighbours. MinPts are the minimum number of neighbours to be present in order to be considered as a neighbourhood.

Based on the metrics described above, each and every point can be one of the three points: Core point, border point and Noise point.

Core point:
A core point is a point within the cluster and this point has more than or equal to the MinPts parameter and these points should be within the range Epsilon(ε). These points constitute majority of a cluster and they are present interior in a cluster.

Border point:
A border point is a point which is present in the border region of a cluster. Technically speaking, border point has fewer points in the neighbourhood than MinPts within ε. And it is also to be noted that all the border points are present in the neighborhood of some of the core points. These points constitute to the cluster.

Noise point:
The point which is neither a core point nor a border point is a Noise point. These points are outlier/ infrequent points.

So the algorithm proceeds in such a way that, as a core point is detected it propagates in such a way if it has MinPts within the ε distance, then these neighbours are included in the cluster and each of the neighbours is checked whether they are core points or border points and if they are core points, the algorithm proceeds accordingly.

Let’s say there are two points A and B where A is a core point. The point B is said to be density reachable from A if B is present in A’s ε distance neighbourhood. Here B is said to be directly density reachable from A. And let’s say B is also a core point and there is this point C which is density reachable from C, then C is like indirectly density reachable from A.

The following are the steps done to implement the actual DBSCAN algorithm:

1) All the data points, epsilon value and MinPts value is given as input.
2) We iterate over the dataset and if it has ε-neighbourhood size more than MinPts then they are assigned a cluster number and each of the cluster is expanded.
3) If it has ε-neighbourhood size less than MinPts, then it is marked as noise.
4) For the cluster expansion, for each point in the neighbourhood, new neighbourhood is calculated and if they have size more than MinPts within ε, then the cluster expansion is called for each of the points in the neighbour.

Implementation Details:

The implementation of DBSCAN is done in python.

Class Point:

We define a class Point from which all points are instantiated as objects. Each object has four different properties,

x - data points of the object,

clusterNumber- initially None,

trueCluster - number associated with the point,

visited - boolean indicating whether the boolean is set or not.

The main function is dbscan and it has three parameters: input text file, epsilon value and MinPts value. So in dbscan, the input text file is read line by line and they are converted to point objects of list called datapoints.

Main Iteration:

For each and every data point, if its not visited then we go inside the condition and mark the point as visited and calculate the neighbours for this point within epsilon distance. If the number of neighbourhood points is more than MinPts then the node is assigned a cluster and its neighbours are sent to expandCluster function else the point is marked with cluster number -1.

Helper functions:

Distance function:

When we are given two points, say point1 and point2 to this function, it calculates the difference of the array and then square each element of the array and then calculate the sum of the entire array and then finally return the square root of the summed value. Basically it returns the euclidean distance between the given two points.

findNeighbours function:

When we are passed a point and epsilon value, all the neighbours within the distance epsilon are being calculated and added to a list. This neighbourhood list is returned as a return value.

expandCluster function:

The input parameter for this function is neighbour points, cluster number, epsilon, MinPts. What we do over here is, we iterate the entire neighbours. If they are not visited, we mark it as visited and set the cluster number to the input cluster number. We again calculate new neighbours for each and every point and if the number of points are more than the MinPts then we call the *expandCluster* function again with the new parameters.

Observations running on gene data:

The DBscan algorithm had a comparatively better performance since it gives a consistent result, but it has it's own disadvantage of choosing eps and mindistance value. We went through the idea of choosing ideal eps and minpoints - Since the data is relatively less we kept the minpoints value to 4 where we obtained maximum performance - As a general thumb rule it's advised not to use minpoints lesser than 3. DB scan was very good adapting to outliers since it completely discards them from consideration - The DBscan wasn't great in cho.txt since the data isn't clustered uniformly, the points are scattered and distributed all over the space - Meanwhile in iyer.txt we observe a better performance since DBscan can identify the points into clusters that are closely related to each other and isolate the outliers. The adaptive DB scan algorithm even learns to optimize on eps and minpoints on an incremental basis.

Pros:

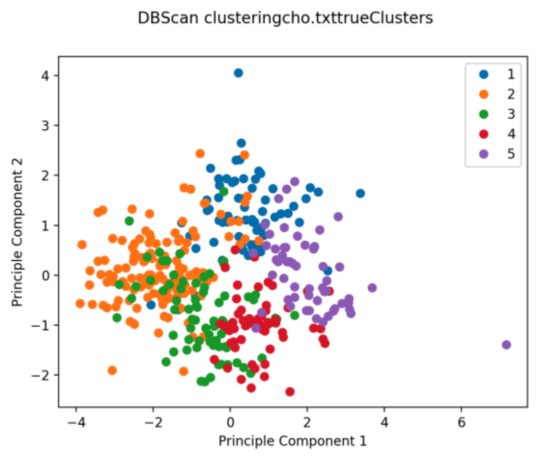
1) Most widely used clustering algorithm as it can identify clusters of arbitrary shapes - clustering works based on combining closely related points that are identified by minpoints and eps.

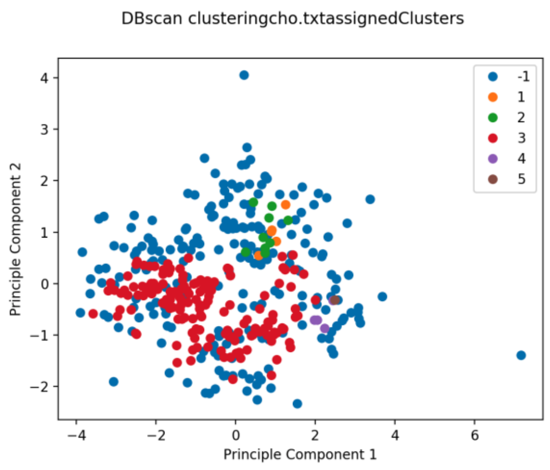
2) Eliminate the possibility of exponential runtime complexity and influence of outliers.

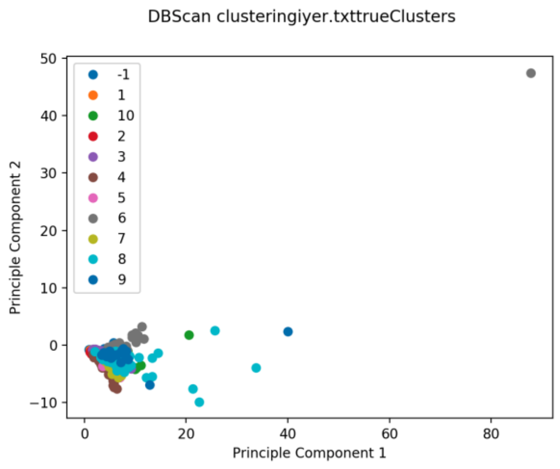
Cons:

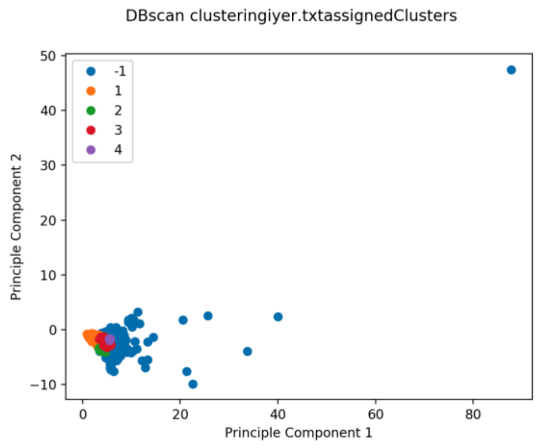
1) Choosing a larger value of eps would cluster almost all the points together and thus it distorts the relationship of the original clusters - Increasing the number of minpoints has the same effect on density based clustering.
2) Choosing too small value of eps would result in marking too many points as noises - but in reality those points are not outliers.

Eps	MinPts	Jaccard for cho.txt	Jaccard for iyer.txt
1	3	0.13658391430028907	0.3018213036213756
1	4	0.1303469326340226	0.30483191511222785
1	5	0.11560481839417396	0.3062034832845623
1	6	0.10254122866235896	0.3098390359412898
1.02	3	0.14242509778620105	0.3019449649684762
1.02	4	0.13924256600089494	0.30493466219561177
1.02	5	0.1317536534446764	0.30616717299498686
1.02	6	0.12238518309594322	0.31059263162937767
1.03	3	0.14367884250474383	0.3020592008713069
1.03	4	0.14056705347529608	0.30493466219561177
1.03	5	0.132930822758903	0.3061851890153901
1.03	6	0.1312147847644805	0.3104772668499239









The DBScan algorithm works best even when there is random shaped clusters but when the data has distinct separation between them, From the above plots we can say that Density based clustering gives a better performance compared to Hierarchical clustering but still it’s not the best performance in identifying the relationship between the points as the points do not have clear level of separation between them.

Comparison between algorithms:

- K-means algorithm works well when the given data points have equal densities, well this might not be the case of all data sources.
- For K-means algorithm we need to define the initial number of clusters, which needs to be determined by other sorts of algorithms or randomly generated.

- The number of iterations we can not predict- it depends on the initial centroids we take.
- K-means performs well for the optimal k and data densities that are similar to each other.
- Hierarchical clustering performance degrades for huge datasets as there is huge number of points that needs to be clustered together.
- Hierarchical clustering on the other hand is scalable and it’s performance will be consistent large sample and clusters.
- In DBSCAN we need to pass two mandatory parameters, epsilon and MinPts and hence we need to check for different values of these two parameters to arrive at an optimal solution.
- And the main advantage of DBSCAN is that we can easily detect outlier i.e. unwanted points.
- The main drawback is that the varying densities cannot be detected. Since we consider all the points within a given epsilon as a cluster, we won’t be able differentiate between clusters.

K-means clustering using MapReduce:

MapReduce is an approach which is used for processing big data sets where the data are processed parallelly and they are combined in reducers to form the output. This type of approach allows parallelism.

There are two procedures:

Map procedure:

The map procedure performs filtering and sorting,i.e The input data is feed and split and fed into multiple mappers and each of them output the <Key, Value> - This output from mapper is sorted so that the items having same key can be identified and fed into the same reducer.

Reduce Procedure:

Once a sorted set of <Key, Value> pairs are obtained in reducer, It summarizes the iterable values belonging to that particular key by iterating over the values. It reduces the output we get from the mapper and performs the summary operation on the entire data.

Pros:

- MapReduce is good for large input data.
- Hadoop and mapreduce actually solves the problem of parallelization
- Hadoop and mapreduce helps in Distribution
- Hadoop and mapreduce handles failures and helps in fault-tolerance.
- Can be used like read once and use many times.
- MapReduce and hadoop framework is usually scalable.

Cons:

- Processing intensive operation. In our case, it's CPU intensive operation.
- The amount of data shared is large.
- It usually has low efficiency.
- Doesn't support high level languages like SQL.
- MapReduce requires to parse each item at reading input and transform it into data objects for data processing, causing performance degradation

Implementation Details:

So in our implementation, we maintain a data file for maintaining initial centroids. And we use this file for defining our initials centroids. So like any other mapreduce algorithm we have our own mapper and own reducer.

Mapper class:

For each and every map function, value will be passed and for each and every point, getClusterId is called which compares this point with all the available temporary centroids and returns the centroid which is close to the point. This emits the pair id of the closest centroid and the point.

Mapper <part of the input file> = Emit pair<id of the closest centroid, input point>

Reducer Class:

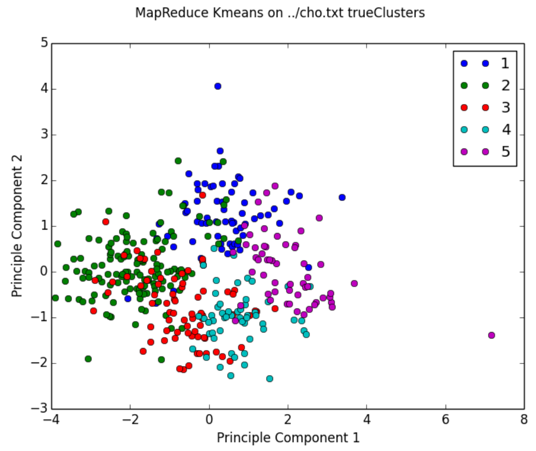
In reducer class, for the reduce method we will be inputting <key, List<point>>. Here we will iterate through the list of points and add all the values to find a sum. Then the sum is divided by the size of the list to get the average sum which will be the new centroid. This new centroid calculated will be maintained in a temporary variable. Finally it will emit index of the point and the clusterId to which the point belongs to for each and every point.

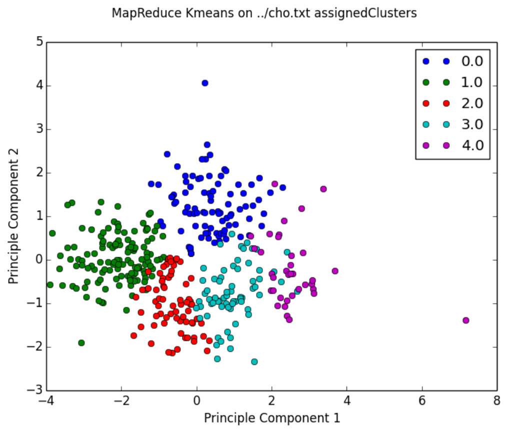
Data Source : cho.txt

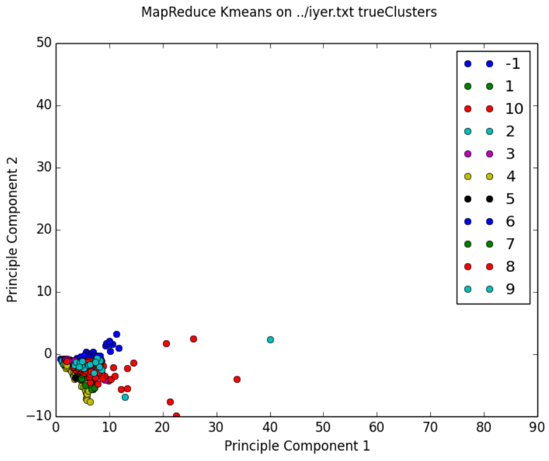
K-Number of Cluster	Jaccard Coeffecient	Time Elapsed(in seconds)
3	0.34652840	30
4	0.41174895	25
5	0.34132258	23
6	0.29775449	38
7	0.25453988	55

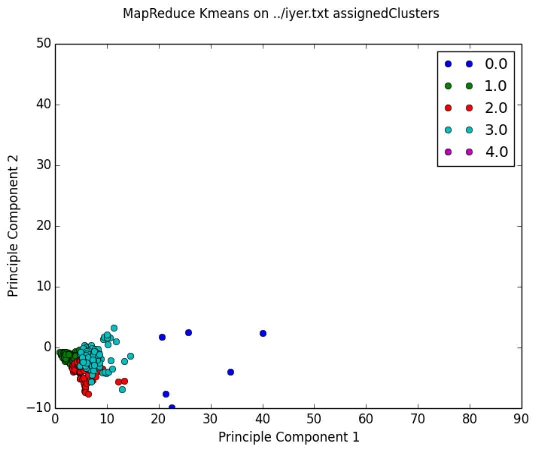
Data Source : iyer.txt

K-Number of Cluster	Jaccard Coeffecient	Time Elapsed(in seconds)
8	0.30813397	40
9	0.31188974	44
10	0.35151760	34
11	0.34686785	55
12	0.31236593	53









Performance Improvement trails:

1. We observed the passing the points in a object like Array or ArrayList, emitting it as a TextWritable gave better performance.
2. Using a combiner would give a great performance improvement since it aggregates the outputs from mapper belonging to a particular key, thus reducing the overhead of the manually combining all the values in the reducer.

References:

1) https://www.edureka.co/blog/k-means-clustering/
2) https://medium.com/@neil.liberman/k-means-clustering-e00408493a40
3) https://stats.stackexchange.com/questions/133656/how-to-understand-the-drawbacks-of-k-means
4) http://people.revoledu.com/kardi/tutorial/Clustering/Numerical%20Example.htm
5) https://en.wikipedia.org/wiki/DBSCAN
6) https://www2.cs.arizona.edu/~bkmoon/papers/sigmodrec11.pdf