GLOBAL K- MEANS CLUSTERING ALGORITHM

Harshit Goyal 2015B4A70815P
Pulkit Agarwal 2015A70050P
Tanay Agrawal 2015B4A80567P
Vaibhav Mathur 2015B4A80567P

1 Introduction

1.1 Background

Clustering is the grouping of a particular set of objects based on their characteristics, aggregating them according to their similarities. Regarding to data mining, this methodology partitions the data implementing a specific join algorithm, most suitable for the desired information analysis. There are several different ways to implement this partitioning, based on distinct models. Distinct algorithms are applied to each model, differentiating its properties and results. One of the most widely used method is the K-means Clustering Algorithm. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed apriori.

1.2 Motivation

K-means Algorithm is a fast and robust way for clustering but it has some drawbacks to it due to which it limits its application. K-means algorithm is a local search procedure and its performance depends heavily on the initial conditions. In the algorithm we have to initialize the k cluster points by randomly taking the points through the vector space. This can change the outcome with varying initial conditions.

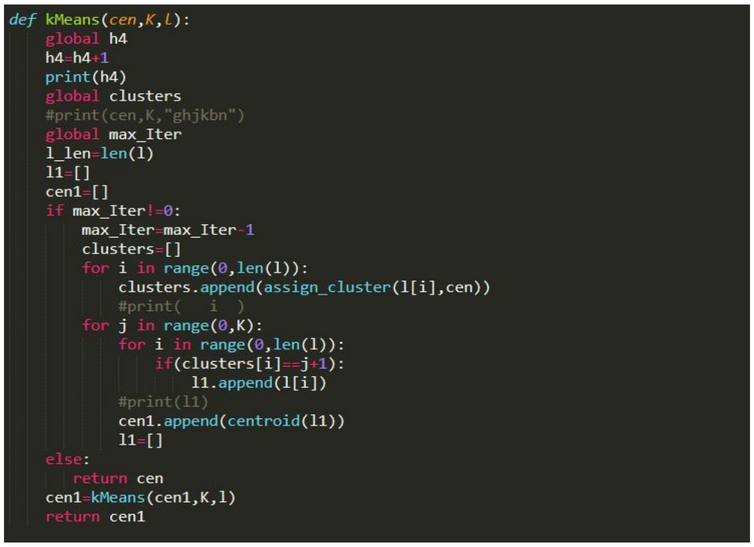
1.3 Objective

The objective of this project is to rectify the above problem by using a Global K-means clustering algorithm. Global K-means algorithm constitutes a deterministic effective global clustering algorithm for the minimization of the clustering error that employs the K-means algorithm as a local search procedure.
Further to improve the running time, some modifications were studied and Fast Global K-means clustering is implemented which gives solutions comparable to Global K-means but works faster than it since it sets an upper bound to the error which significantly reduces the number of iterations and lower the computational load required.
Also an implementation of Kd Trees is used to reduce the initial number data points on which local search procedure is performed by using the bucket centers (which are fewer than the data points) as possible insertion locations for the Global K-means algorithm.

2 Proposed Techniques

K-Means Algorithm

The k-means algorithm finds locally optimal solutions with respect to the clustering error. It is a fast iterative algorithm that has been used in many clustering applications. It is a point-based clustering method that starts with the cluster centers initially placed at arbitrary positions and proceeds by moving at each step the cluster centers in order to minimize the clustering error.

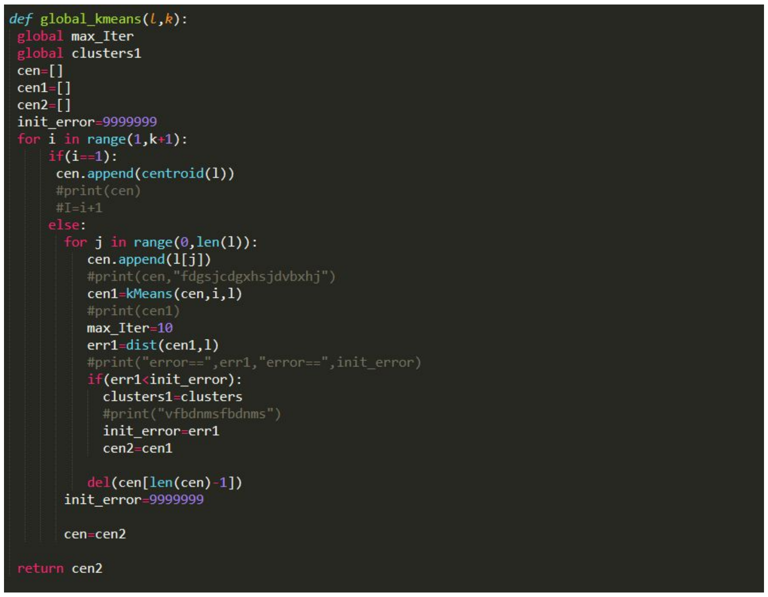


```
def kMeans(cen,K,l):
    global h4
    h4=h4+1
    print(h4)
    global clusters
    #print(cen,K,"ghjkbn")
    global max_Iter
    l_len=len(l)
    l1=[]
    cen1=[]
    if max_Iter!=0:
        max_Iter=max_Iter-1
        clusters=[]
        for i in range(0,len(l)):
            clusters.append(assign_cluster(l[i],cen))
            #print(   i  )
        for j in range(0,K):
            for i in range(0,len(l)):
                if(clusters[i]==j+1):
                    l1.append(l[i])
            #print(l1)
            cen1.append(centroid(l1))
            l1=[]
    else:
        return cen
    cen1=kMeans(cen1,K,l)
    return cen1
```

Global K-means Algorithm

In this technique instead of randomly selecting initial values for all cluster centers as is the case with most global clustering algorithms, the proposed technique proceeds in an incremental way attempting to optimally add one new cluster center at each stage.
The rationale behind the proposed method is based on the following assumption: an optimal clustering solution with k clusters can be obtained through local search (using k-means) starting from an initial state with:

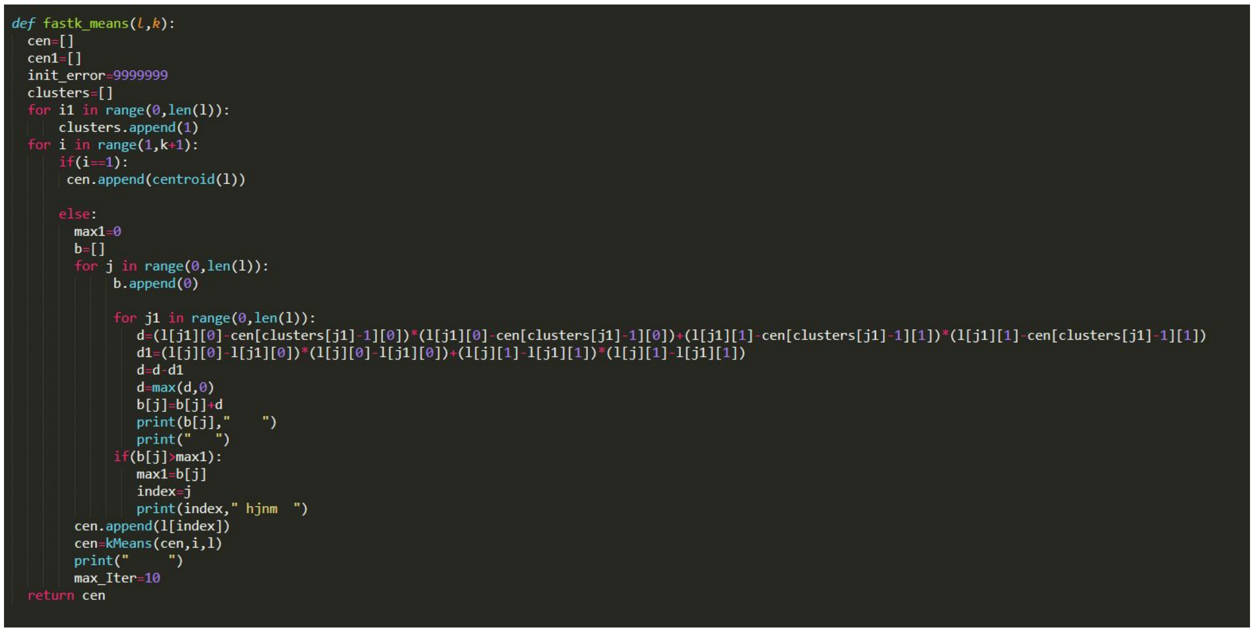
- The k − 1 centers placed at the optimal positions for the (k − 1)-clustering problem and,
- The remaining k-th center placed at an appropriate position to be discovered.



```
def global_kmeans(l,k):
 global max_Iter
 global clusters1
 cen=[]
 cen1=[]
 cen2=[]
 init_error=9999999
 for i in range(1,k+1):
     if(i==1):
      cen.append(centroid(l))
      #print(cen)
      #I=i+1
     else:
       for j in range(0,len(l)):
          cen.append(l[j])
          #print(cen,"fdgsjcdgxhsjdvbxhj")
          cen1=kMeans(cen,i,l)
          #print(cen1)
          max_Iter=10
          err1=dist(cen1,l)
          #print("error==",err1,"error==",init_error)
          if(err1<init_error):
            clusters1=clusters
            #print("vfbdnmsfbdnms")
            init_error=err1
            cen2=cen1

          del(cen[len(cen)-1])
       init_error=9999999

       cen=cen2

 return cen2
```

Fast Global K-means Algorithm

The fast global k-means algorithm constitutes a straightforward method to accelerate the global k-means algorithm. The difference lies in the way a solution for the k-clustering problem is obtained, given the solution of the (k − 1)-clustering problem. In this we do not execute the k-means algorithm until convergence to obtain the final clustering error E_n. Instead we compute an upper bound $E_n \leq E - b_n$ on the resulting error E_n for all possible allocation positions x_n, where E is the error in the (k − 1)- clustering problem.

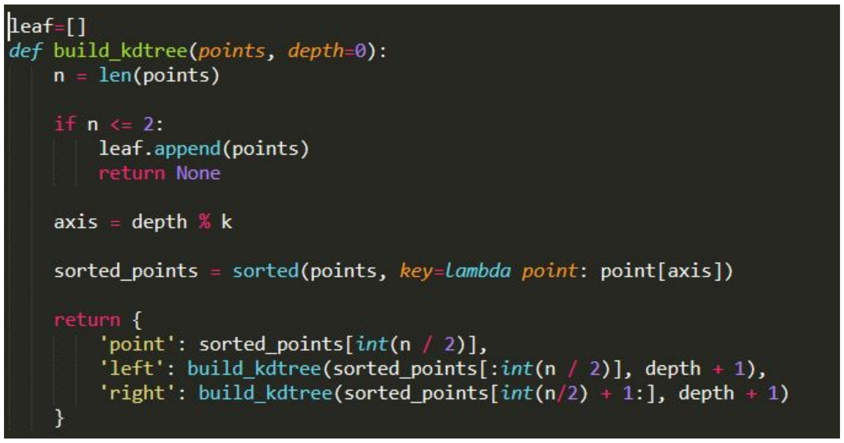


```
def fastk_means(l,k):
    cen=[]
    cen1=[]
    init_error=9999999
    clusters=[]
    for i1 in range(0,len(l)):
        clusters.append(1)
    for i in range(1,k+1):
        if(i==1):
         cen.append(centroid(l))

        else:
          max1=0
          b=[]
          for j in range(0,len(l)):
               b.append(0)

               for j1 in range(0,len(l)):
                  d=(l[j1][0]-cen[clusters[j1]-1][0])*(l[j1][0]-cen[clusters[j1]-1][0])+(l[j1][1]-cen[clusters[j1]-1][1])*(l[j1][1]-cen[clusters[j1]-1][1])
                  d1=(l[j][0]-l[j1][0])*(l[j][0]-l[j1][0])+(l[j][1]-l[j1][1])*(l[j][1]-l[j1][1])
                  d=d-d1
                  d=max(d,0)
                  b[j]=b[j]+d
                  print(b[j],"    ")
                  print("   ")
               if(b[j]>max1):
                  max1=b[j]
                  index=j
                  print(index," hjnm  ")
          cen.append(l[index])
          cen=kMeans(cen,i,l)
          print("     ")
          max_Iter=10
    return cen
```

Kd trees algorithm

A k-d tree is a multi-dimensional generalization of the standard one-dimensional binary search tree, that facilitates storage and search over k-dimensional data sets.

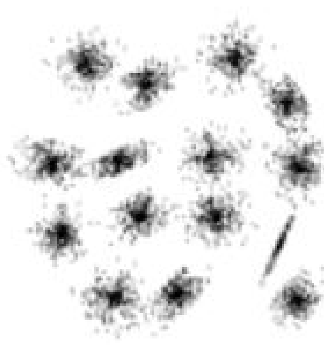
We run the kd tree algorithm until a terminal node (called bucket) is created containing less than a prespecified number of points b (called bucket size) or if a prespecified number of buckets have been created. The idea is to use the bucket centers (which are fewer than the data points) as possible insertion locations for the algorithms presented previously.



```
leaf=[]
def build_kdtree(points, depth=0):
    n = len(points)

    if n <= 2:
        leaf.append(points)
        return None

    axis = depth % k

    sorted_points = sorted(points, key=lambda point: point[axis])

    return {
        'point': sorted_points[int(n / 2)],
        'left': build_kdtree(sorted_points[:int(n / 2)], depth + 1),
        'right': build_kdtree(sorted_points[int(n/2) + 1:], depth + 1)
    }
```

3 Data Sets Used

S-sets



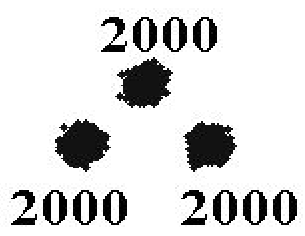
S1

Synthetic 2-d data with N=5000 vectors and k=15 Gaussian clusters with different degree of cluster overlapping

P. Fränti and O. Virmajoki, "Iterative shrinking method for clustering problems", *Pattern Recognition*, 39 (5), 761-765, May 2006. (Bibtex)

S1: ts txt

Unbalance



Synthetic 2-d data with N=6500 vectors and k=8 Gaussian clusters

ts txt

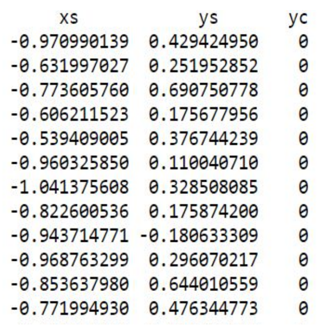
M. Rezaei and P. Fränti, "Set-matching methods for external cluster validity", *IEEE Trans. on Knowledge and Data Engineering*, 28 (8), 2173-2186, August 2016. (Bibtex)

Ground truth centroids: cb and txt

Ground truth partitions: pa

Synth.te

Test set for synthetic two-class problem.



```
       xs            ys        yc
  -0.970990139  0.429424950    0
  -0.631997027  0.251952852    0
  -0.773605760  0.690750778    0
  -0.606211523  0.175677956    0
  -0.539409005  0.376744239    0
  -0.960325850  0.110040710    0
  -1.041375608  0.328508085    0
  -0.822600536  0.175874200    0
  -0.943714771 -0.180633309    0
  -0.968763299  0.296070217    0
  -0.853637980  0.644010559    0
  -0.771994930  0.476344773    0
```

4 Experiments and Results

The above data sets were tested on the techniques discussed in the report. This was done in three stages:

- First normal global K-means was applied with data point as the insertion locations.
- Next, Fast global K-means was tried with the same points.
- At last the Kd trees was used to reduce the number of insertion point and bucket centres were used.

4.1 Results

Global K-means was applied on the data set S1.csv by varying k clusters and the results have been demonstrated in the Figs 2.1-2.3.

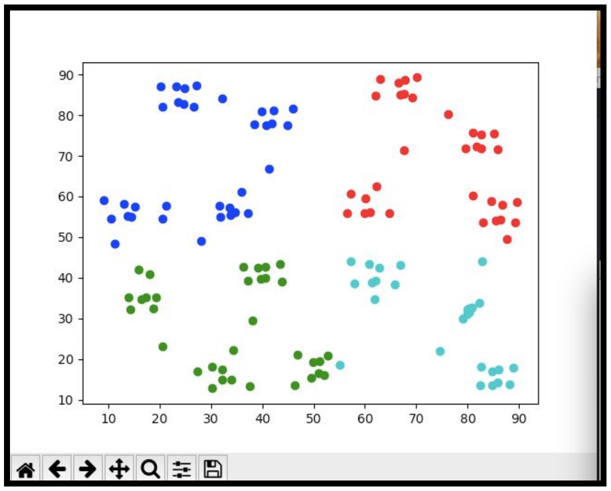


Fig 2.1, k=4

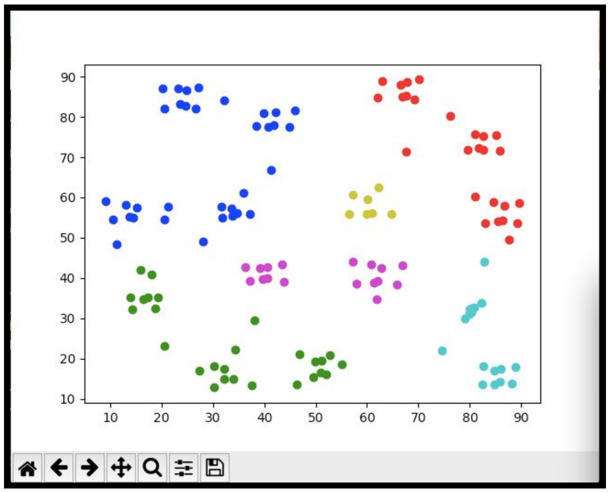


Fig 2.2, k=7

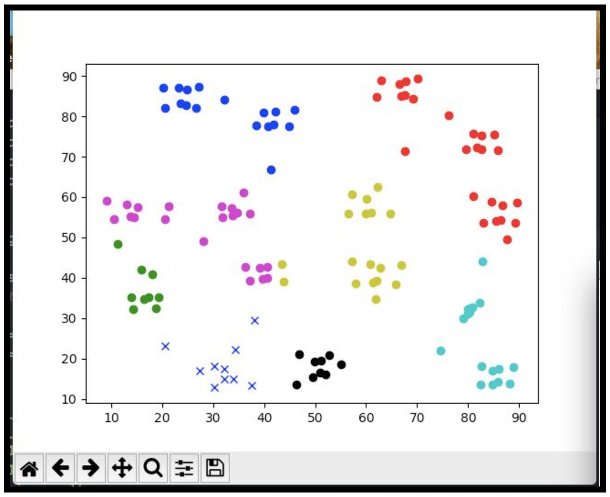


Fig: 2.3, k=8

Fast global k-means on the data set cassini250.cv (Figs 3.1-3.2)

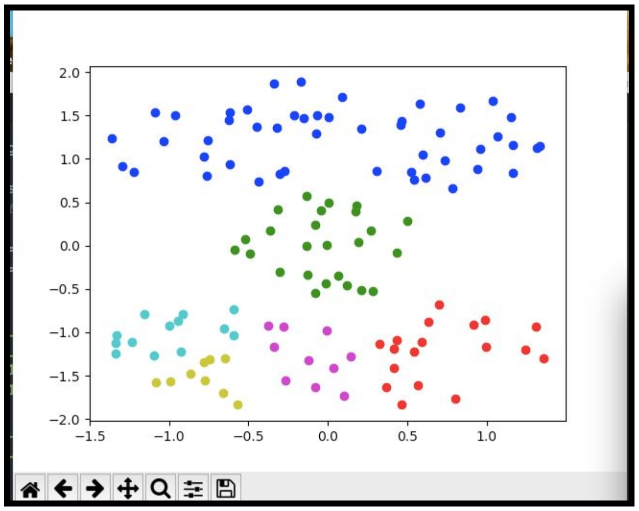


Fig 3.1, k=6

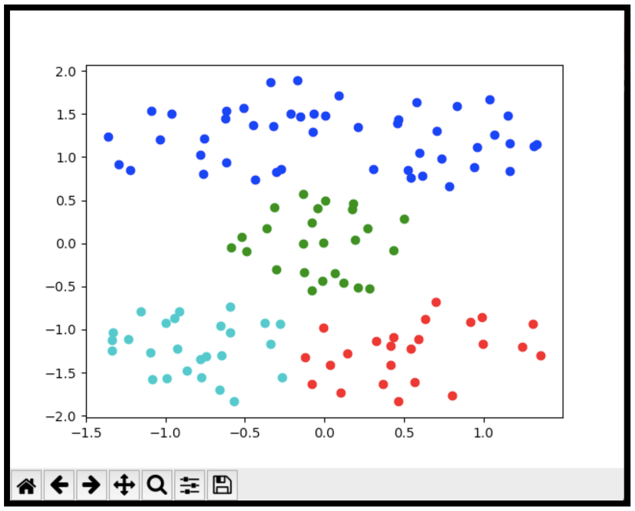


Fig 3.2, k=4

Next, Kd trees was used to find the bucket centres of the data point on the set (Fig 4.1). The blue dots represent the bucket centre and the red dots are all the data points.

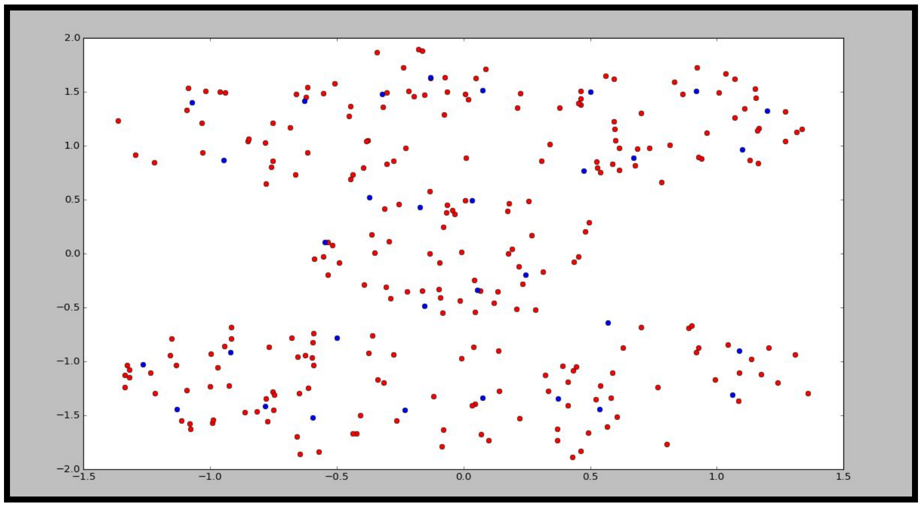


Fig 4.1

When Kd trees were used in collaboration with the global K-means algorithm on the data set synth.te, the results came to something like in Fig 5.1, a significant reduce in computational time was found with almost similar results.

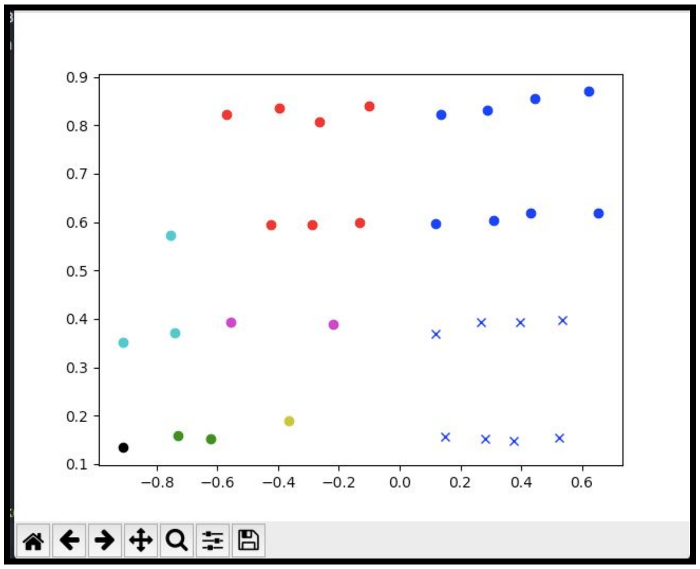


Fig 5.1

Now to check the error with increasing number of clusters(k), a graph between mean square error and k was plotted(Fig, 6.1) using the data set unbalance. The y-axis represents the mean square and the x-axis, the number of clusters. It can be clearly seen that the error decrease if we increase the number of clusters.

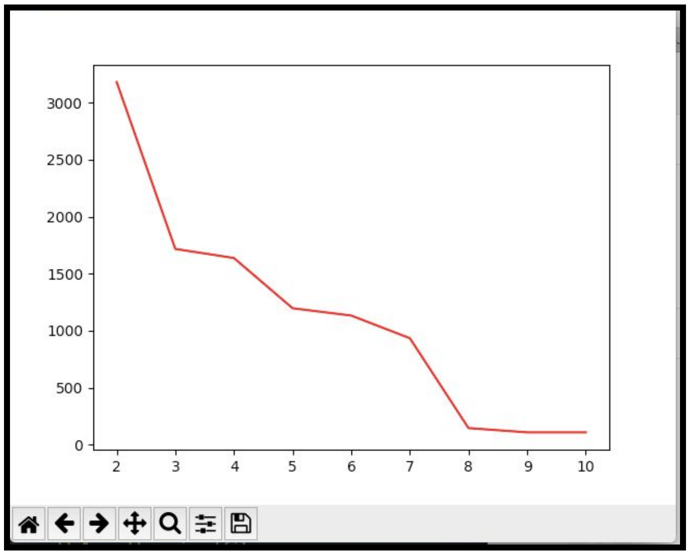
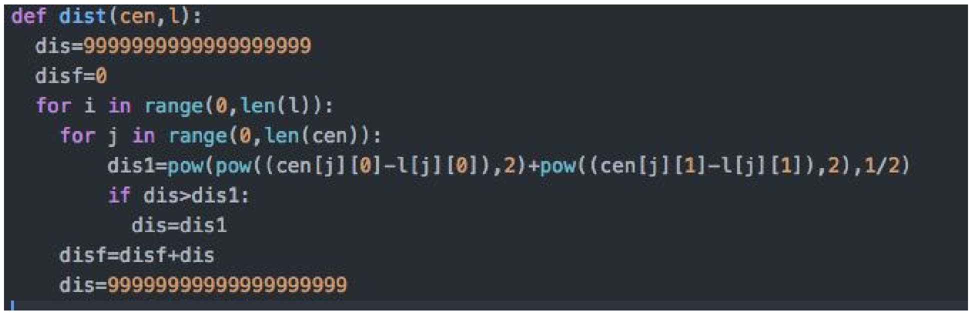


Fig 5.1

4.2 Error Analysis

Error analysis is done by computing the squared error using centroids. Squared distance of each point with the centroid of the cluster it belongs to is computed and then we sum them up to get the error.



```
def dist(cen,l):
  dis=9999999999999999999
  disf=0
  for i in range(0,len(l)):
    for j in range(0,len(cen)):
        dis1=pow(pow((cen[j][0]-l[j][0]),2)+pow((cen[j][1]-l[j][1]),2),1/2)
        if dis>dis1:
          dis=dis1
    disf=disf+dis
    dis=99999999999999999999
```

For global k means:

While fixing the k-1 cluster centroids and then varying the kth centroid over the n dataset elements.

We apply K-means each time and store the error obtained, by fixing each element as kth centroid and finally, the one for which error(computed on centroids obtained after convergence for that point) is minimum we choose the centroids obtained after convergence for that point correspondingly.

For fast K-means :

We do not apply K-means each time instead we find an upper bound for error and one for which error is minimum that point along with the k-1 cluster centroids is used and then k means is applied to obtain the k clusters and their respective centroids.

5 Conclusions and Future work

Using Kd trees the computational overload is decreased and we are able to reduce the dataset at the cost of increasing error rate.Applying different variants of K-Means we come to know that one is better than other in some way. Global K-means is better in terms of error obtained after clustering but it works slow due to K-means running each time for each point.So Fast K-Means is applied in order to increase the speed at the cost of less accuracy.

- K-means and its variants can be applied to n dimensions by changing the formula for centroid.
- PCA can be incorporated for reducing dataset.
- Preprocessing techniques can be applied on dataset for better results.
- Working with high number attribute dataset
- Fast K-means is not sometimes able to properly classify the cluster for sparse and outlier points so increasing kmeans of iteration or changing the way of computing error.

6 References

- https://ac.els-cdn.com/S0031320302000602/1-s2.0-S0031320302000602-main.pdf?_tid=cd8b0d07-62cd-4d93-b762-de41962374c7&acdnat=1524569808_4c55ac39b64c797819aea01e43087f74
- https://matplotlib.org/api/pyplot_api.html
- http://www.numpy.org
- https://www.programiz.com/python-programming