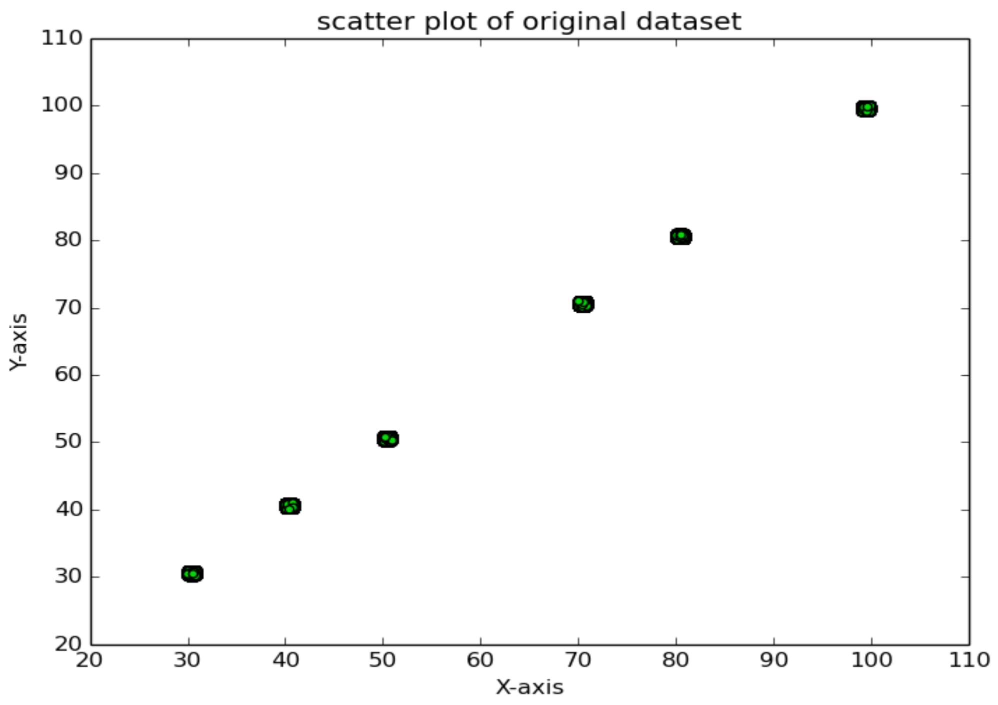
Data Mining: Assignment3

Abhishek Yadav, CS12B032

March 1, 2016

Q1

(a) Scatter plot of the original dataset on the left clearly shows 6 distinct and large cluster aligned along X=Y. Reachability plot obtained as a result of running OPTICS over the dataset also shows 6 distinct clusters. A close observation of the reachability plot shows that there are several smaller clusters within larger ones.



(b) While setting ϵ' to extract clusters from the reachability plot we must know what our requirements are. If We want to extract smaller clusters and more in number we must set smaller value of ϵ' which will extract clusters from within large clusters, but if our requirement is to look at higher level of clustering, we can set bigger values of ϵ' which will result in larger clusters' extraction and less in number giving no information about the

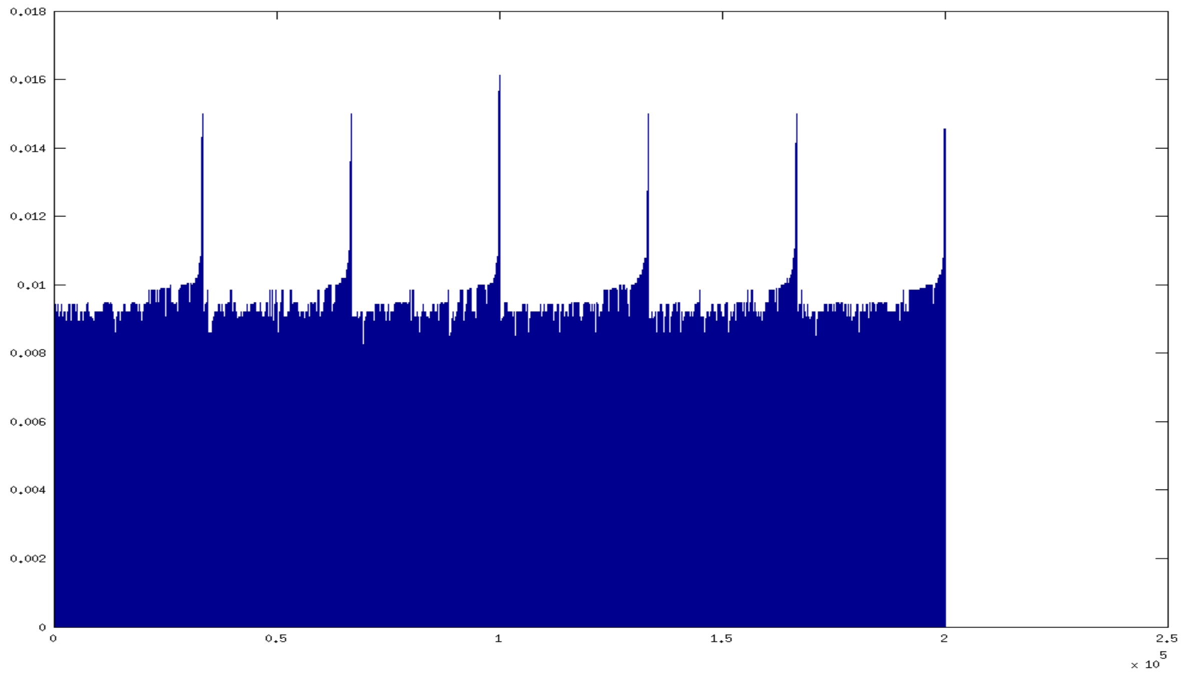
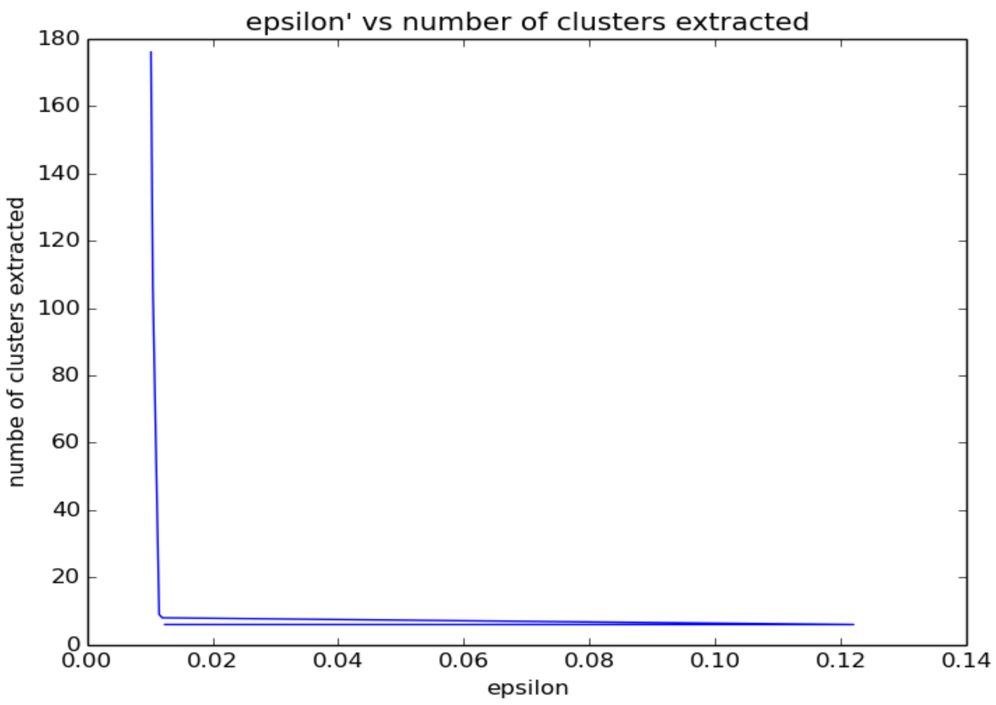


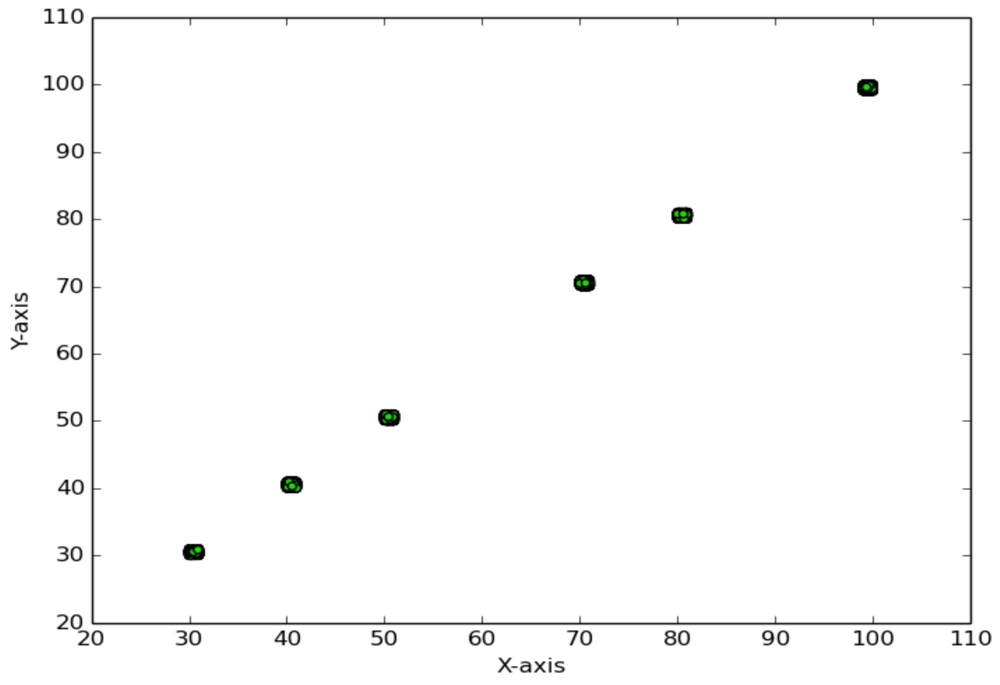
Figure 1: Reachability-plot of the ordered set of the

smaller clusters. Now, in this case if we mean optimal clustering as the clustering setting to extract the 6 main clusters then we can keep setting parameters from some minimum value and observe the number of clusters extracted for each of them keep increasing them, finally we get the desired number of clusters with great accuracy, i.e. almost same number of data-points can be extracted for each cluster by setting suitable parameter. Following graph represents the number of clusters extracted against the value of ϵ':

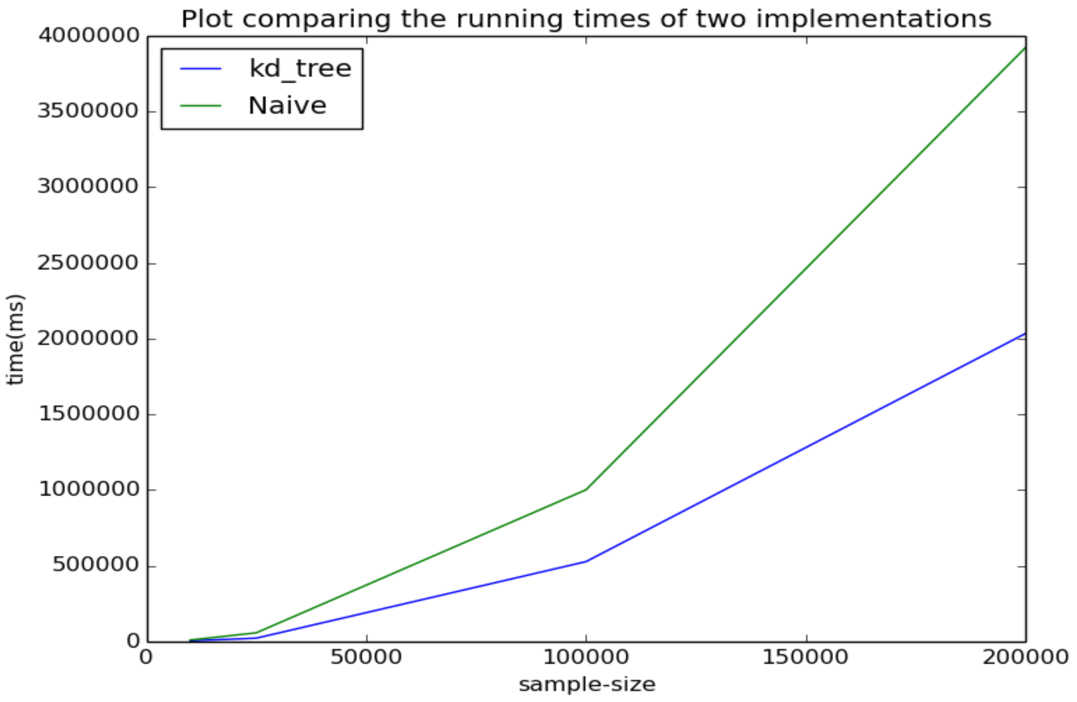


For ϵ'=0.0122 we get 6 clusters each containing almost 33,330 or so points which, infact is the case with real number of clusters. So optimal value of ϵ' is 0.0122. Further increasing the value of ϵ' slightly doesn't change the number of clusters. But after a certain value it will result in even less number of clusters.

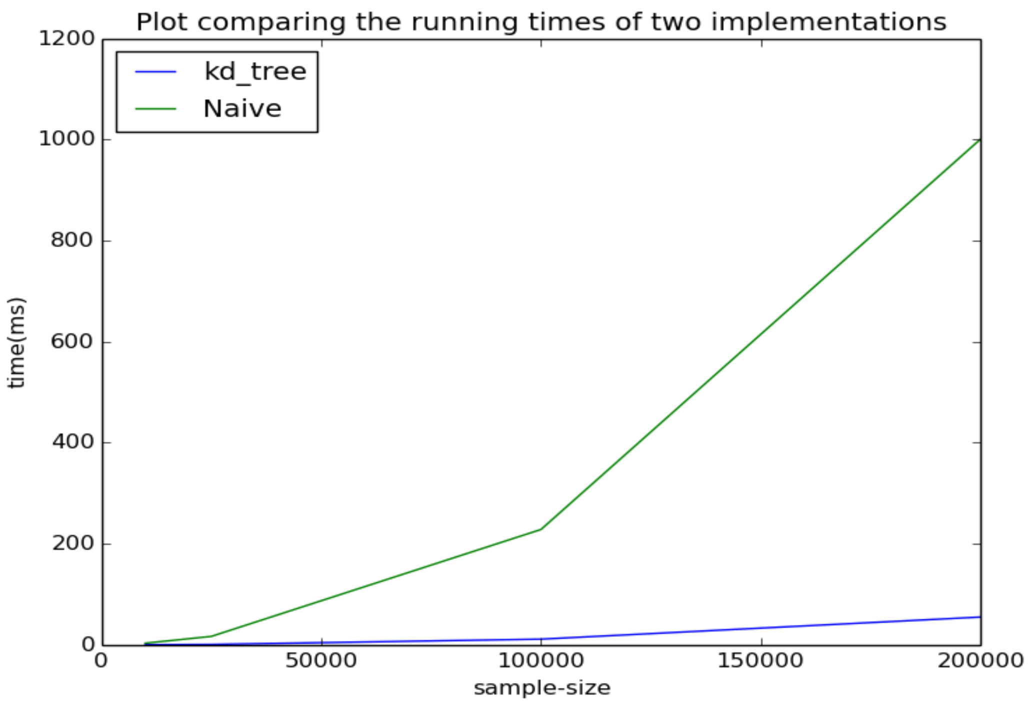
(c) Following are the plots of 'execution-time vs sample-size' for various sample sizes using naive and kd-implementation of OPTICS. As observed in the output of both(KD-Tree and naive) implementations, each of the 6 clusters contain almost 33334 ± 1 hence randomly sampled datasets have the equal probability of having almost same number of points from each cluster and hence forming the same number of clusters as the original dataset with each new cluster containing almost same number of data points. Following scatter plot shows the distribution of 10000 randomly sampled datapoints:



Y-axis
X-axis



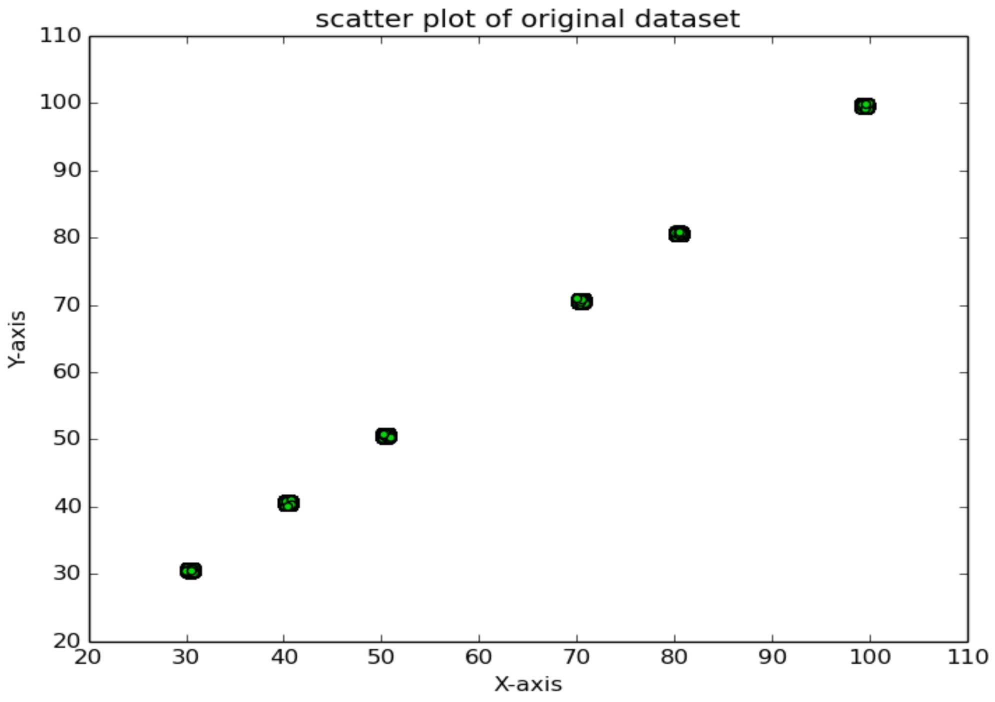
Plot comparing the running times of two implementations
kd_tree
Naive
time(ms)
sample-size

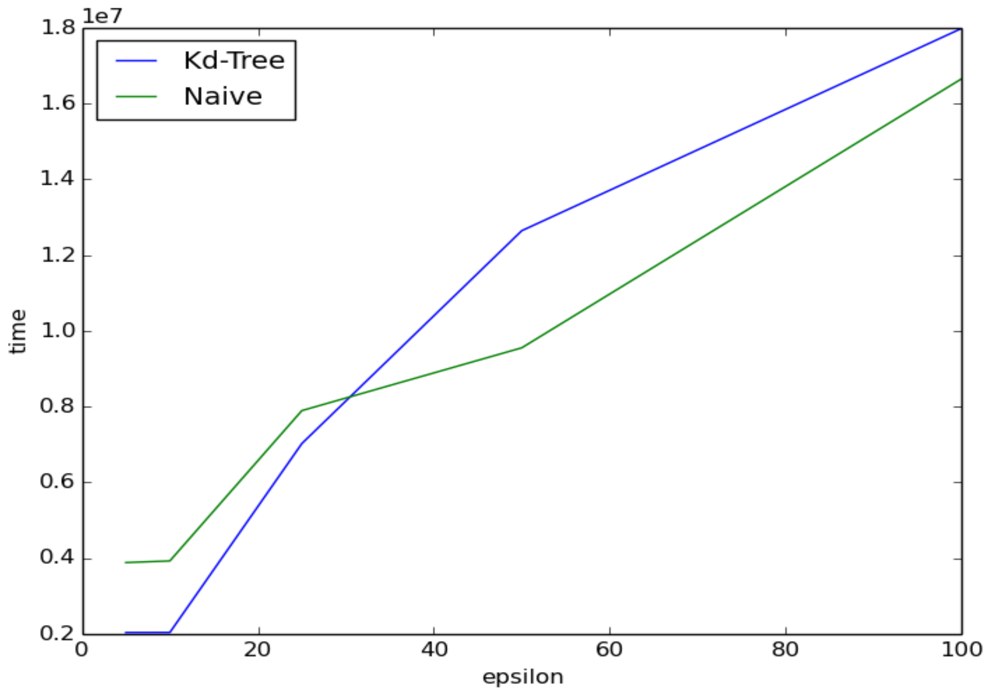


Observation: For small value of ϵ, gap between the two lines, i.e. difference in time taken for the same sample size, increases rapidly with increasing sample size, as compared to larger value of ϵ. I've calculated the execution times ignoring the preprocessing times hence , it can be ignored in performance evaluation of the two implementations. In the naive implementation, for every point, entire database is scanned to find out the ϵ-neighbors of the point, which is accompanied by linear lookup of an array of size **minPts** to efficiently update the core distance of the current object and the reachability distance of the nearby objects for which {d(currObject, p) < core_dist} followed by the insertion of the same neighbors in the seedList. So total complexity of the execution for all n data points is:
Time = scan-time + core-distance and reachability-distance update time + seedList update time.

Last two steps, i.e. core and reachabiility-distance updates and seedList-updates are performed even in Kd-Tree implementation. So, distinguishing factor between the two implementations is the **"ϵ-neighborhood"** query time. Kd-Tree returns the ϵ-neighborhood of a point after performing O(log(N)+K){typical case}, O($\sqrt{n}+k$) {worst-case}operations, where K is the number of data-points returned, on the tree resulting in total O(nlog(n)+n*k) or O(n$\sqrt{n}$+n*k) complexity for complete execution whereas the naive implementation results in O(n^2) complexity. So difference in the execution times is : d(T)= n^2 - n($\sqrt{n}$+k) = n(n$\sqrt{n}$ - K), which is an increasing function in n. This answers the question why the difference in execution-times increase with increasing sample-sizes. Moreover, with increasing, ϵ, number of points returned in each ϵ-neighborhood query results in increased value of K at every step of the execution, resulting in decreased growth in the execution-time difference.

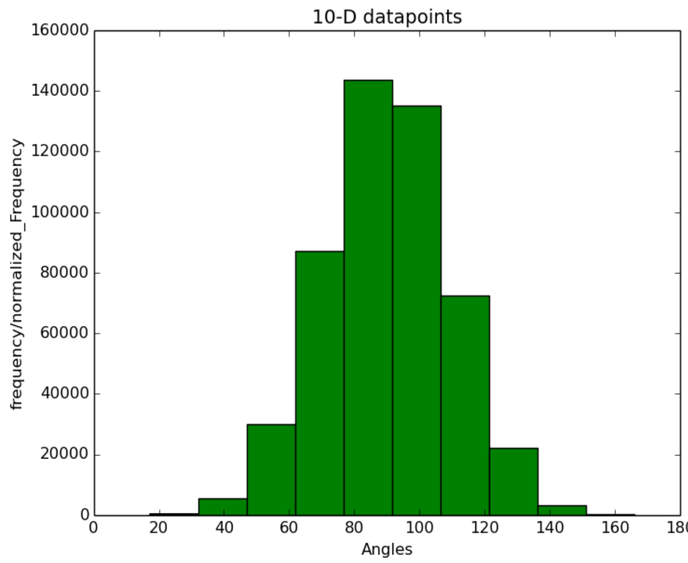
(d) As discussed in (a) scatter plot of the original dataset, depicted below, reveals that clusters are quite dense and each pair of cluster is separated by more than 10 units of distance, i.e. closest points between any two clusters are as far apart as 10units. -

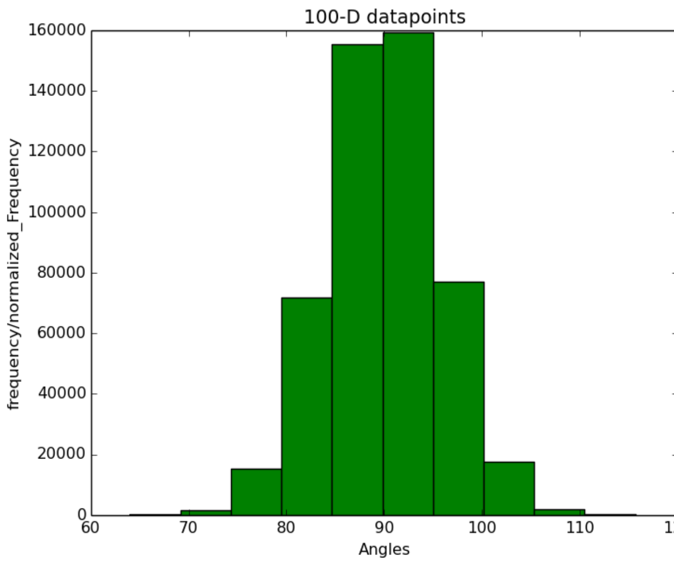


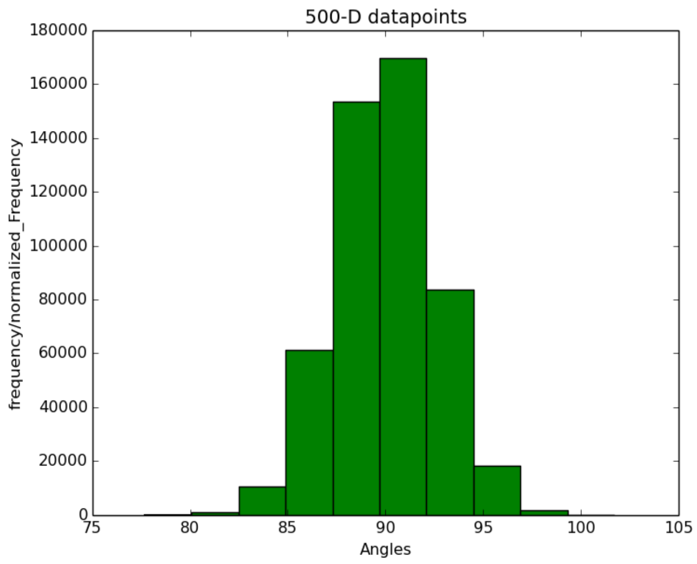


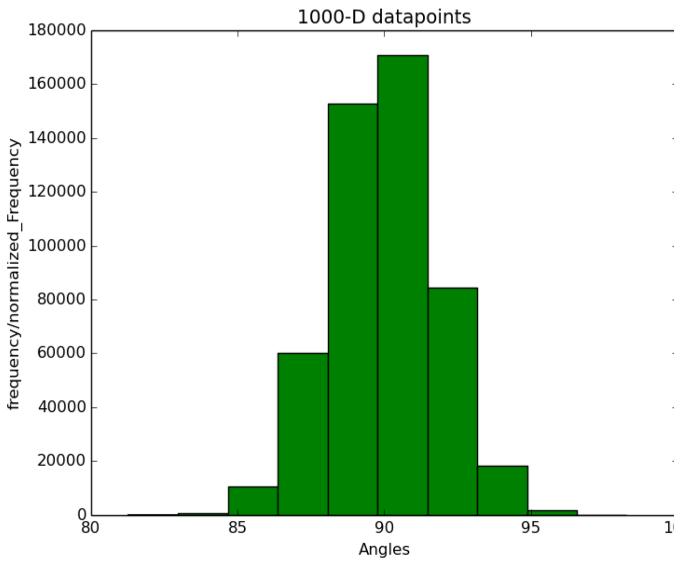
Given the above description, naive and kd-tree implementations get the same number of points, all belonging to the same cluster, in the seed set for ϵ=5 and 10s. Being densely connected, all of them are included within the seedSet for the first random point, and later no other points get added to the **seedSet** which results in almost the same execution time for both these values of ϵ and the operations are scanning the database and updating the core and reachability distance of the neighboring points. Next, for $\epsilon = 25$, we get three clusters, one containing clusters at 30, 40 and 50 and another containing those at x=70 and 80 and last containing the one at x=100. Thus ϵ=25, adds more points to the seedSet, in few steps, than those at 5 and 10, resulting in more number of core and reachability-distance updates and costly seedSet update due to increased size, (nlog(n)), hence total increase in time than that for ϵ= 5 and 10, and as proved previously kd-tree still performs better. Finally, we see that no two consecutive clusters are separated a distance greater than 50, hence for ϵ= 50 and 100 we get only one cluster and containing all 6-sub clusters. For these two values of ϵ kd-tree performs worse than naive, because worst case complexity for kd-tree is O($\sqrt{n}$+K) where K is the number of points being returned as a result of the ϵ-neighborhood query, which in this case, is almost equal to n, because for any point in the middle clusters, almost all the points will be returned as a result of neighborhood-query and total complexity will become O(n*n+n*$\sqrt{n}$) which is worse than that of naive implementation hence resulting in upper kd-tree line.

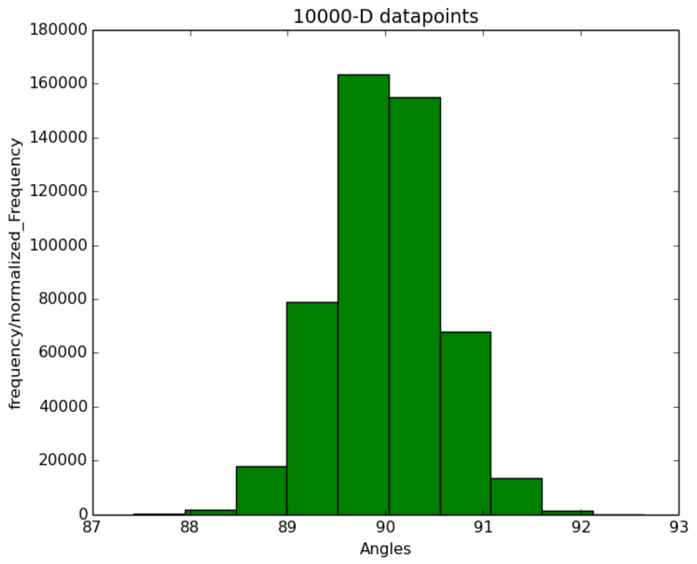
Q2











Observation: As dimensions increase, angles between pairwise points tend to converge around $90°$. **Explanation:** Dot product of two d-Dimensional vectors say, A=$[a_1, a_2......, a_d]^T$ and B=$[b_1, b_2, ..., b_d]^T$(which are essentially two points but referenced from origin) is calculated as follows:

$$A \cdot B = |A||B|cos\theta$$

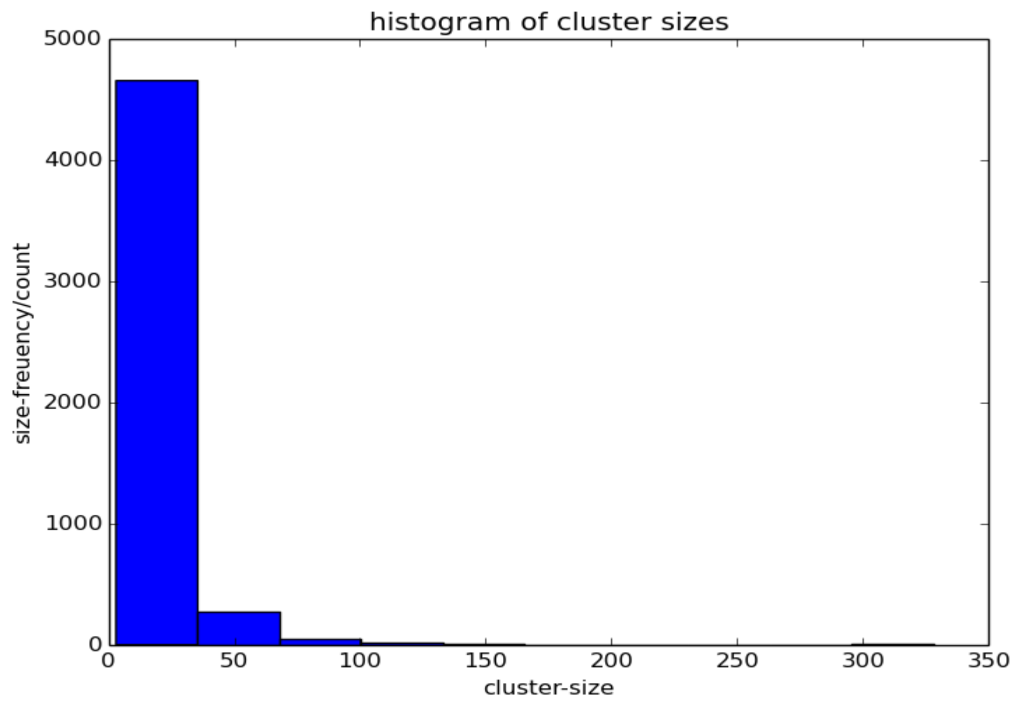
So, angle between the two can be calculated using the above as:

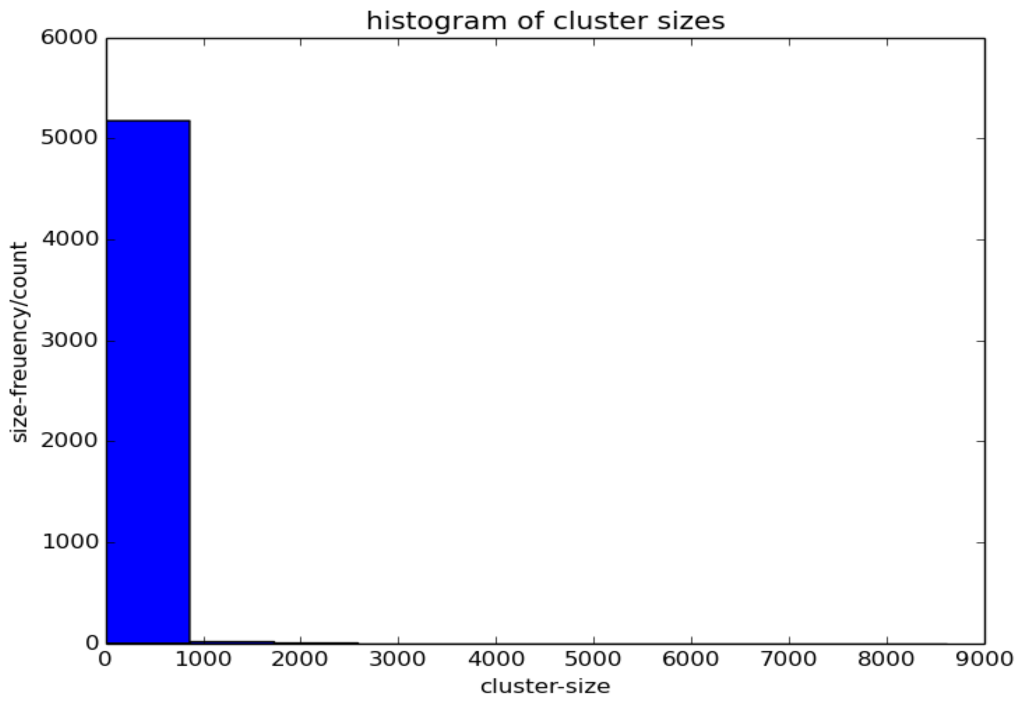
$$\theta = cos^{-1}\left(\frac{A \cdot B}{|A||B|}\right) = cos^{-1}\left(\frac{\sum_{i=1}^{d} a_i b_i}{\sqrt{\sum_{i=1}^{d} a_i^2}\sqrt{\sum_{i=1}^{d} b_i^2}}\right)$$

Here, numerator is the sum of random values multiplied together which are as likely to be negative as positive and hence expected value of zero whereas the denominator grows linearly with d. Thus for large dimensions, cosine of two vectors is close to zero implying that the angles tend to 90°.

1 Q3

As described in the class, as the value of inflation parameter is increased the granularity of clustering also increases as a result small clusters start forming. As mentioned by the author, varying the value of expansion parameter doesn't affect the clustering structure hence set it's value to be constant for all the values of inflation parameter(I). Considering all the communities as the standard I started performing MCL on both the graphs but for none of the values in the given range did we get the results matching with the total number of clusters in the given file. Following is the distribution of the clusters according to their sizes in the amazon's top-5000 community and mcl performed on amazon-graph for I=1.23


histogram of cluster sizes
size-freuency/count
cluster-size
0
1000
2000
3000
4000
5000
0
50
100
150
200
250
300
350


histogram of cluster sizes
size-freuency/count
cluster-size
0
1000
2000
3000
4000
5000
6000
0
1000
2000
3000
4000
5000
6000
7000
8000
9000