Assignment 2

CS6720 : Data Mining

CS15M015 : Hasit Bhatt

Indian Institute Of Technology, Madras

1 Apriori & FP-Tree

(a) The frequent item-sets with 10% support(8816 absolute support frequency) are (32), (38), (39), (41), (48), (38, 39), (39, 41), (39, 48), (41, 48).[1]

(b) **Comparision of FP-Tree and Apriori**

(i) Varying Support Threshold

When observations were taken by varying the support thresholds to 5%, 10%, 25%, 50%. The following results were observed.

Support Threshold	APriori	FP-Tree	#Items
1%	11m51.275s	0.588912s	159
5%	13.947s	.51462s	16
10%	7.8s	0.43s	9
25%	5.45s	0.415091s	3
50%	3.396s	0.409492s	1

Table 1: Running time with varying support

From the results, we can observe that reducing the support increases the time required to calculate the frequent items increase exponentially. But, in case of FP-tree it does not seem to increase that rapidly, it increases rather very slowly, seemingly linear in the observations.

This is due to the underlying fact that, Apriori does too many scans on disk. And, as we know disc access is far slower than the main memory (Approximately 100 times). Therefore, when, support is set to lower values, the Apriori does not scale, whereas, FP-Tree does very well.

(ii) Varying Dataset size

When data size is varied to 10%, 25%, 50% and 100%, the results as shown in Table 2 were observed.

[1] can be found under Code/Q1

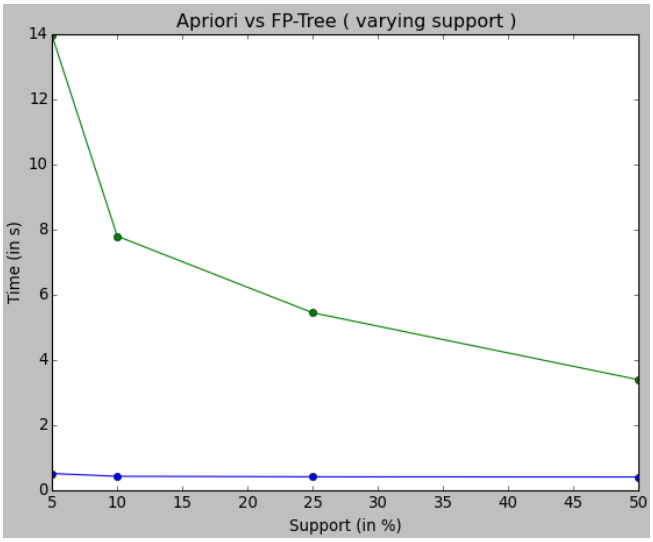

Fig. 1: Varying support threshold

Percentage Data	Apriori	FP-Tree
10%	0.716s	0.048943s
25%	1.931s	0.118351s
50%	3.807s	0.231427s
100%	7.714s	0.420977s

Table 2: Performance with change in datasize

As already observed in previous subquestion, the FP-tree is much faster and takes less time, due to less number of disc accesses. But, with the change in datasize, the graphs of both are not too dissimilar.
With the varying datasize, computation task increases. But, as can be observed from the data, the ratio of the time taken for two different variations (e.g. time taken for 10% and time taken for 25%) is same for a given method. This is due to the fact that, increase in the size of datasets, means, more disk access.

FP-tree reduces the number of disc accesses to 2, but, still it has to read transactions from memory only. So, as, it has more amount of data to read, the time required increases, and, as the computation and main memory access is comparatively much lesser, the effect of datasize is proportional to the computation time. E.g. ratio of time for 25% dataset and 10% dataset is 2.418, which is similar to increase in data (2.5x).

The same can be observed for Apriori, and to be more precise, any method used for the same task, because, the ratio of time for data ac-

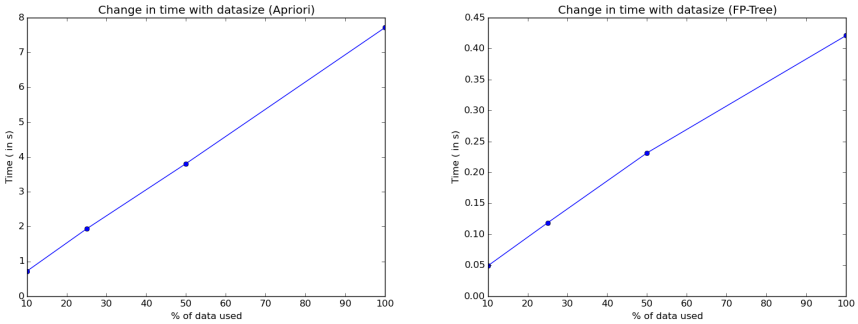

Fig. 2: The change in computation time with change in datasize.

cess from disc will increase, no matter how many scans you need to do. Therefore the plot is found to be linearly increasing.

(iii) Top k% of items with change in k

With considering only top 10%, 25%, 50% and 100% items by frequencies, the following plot was observed. To understand behaviour, both the plots were scaled. For scaling, time taken was divided by the time taken for top 10% items.
Here, it can be observed that changing the value of k in top k% items does not affect much to Apriori algorithm, because, it still needs to do same number of scans, and, only items in transactions are reduced (a bit), and not transactions. So, time taken will still be almost same.

But, in case of FP-Tree, it behaves differently. It gets affected by the change in the value of k. This is because, FP-Growth creates FP-Tree in memory, where, it stores all the data in the structure, so, with more number of items, the nodes will be more. And, therefore, construction will take more time, when number of items are more. And, disc scans does not effect much, as number of disc scans are anyway 2 (constant).

So, for even better performance, we can only consider top X% items in datasets, and then let FP-Tree take it as an input.

2 Finding number of clusters

The number of clusters in the dataset is **10**.

To find the number of clusters k which can be a good fit for clustering can not be known beforehand, as clustering is unsupervised. But, by the fact that

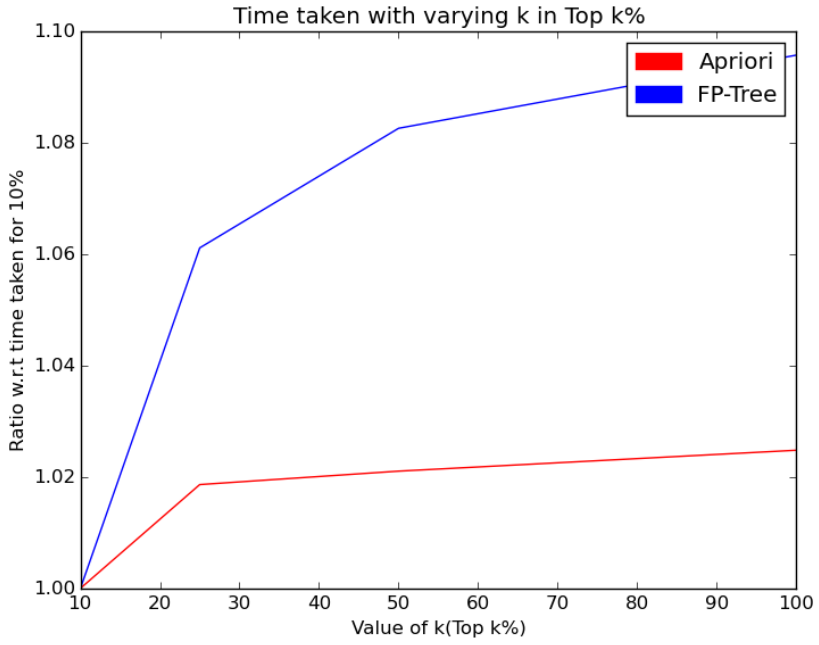

Fig. 3: Time ratio with varying value of k when considering only top-k% items

with number of clusters, the inter cluster distance keep on decreasing, and intra cluster distances keeps on increasing, we can find the fit for number of clusters using the elbow method.

The elbow method is nothing but, the graph of ratio of intra cluster vs inter cluster over k (number of clusters). So, given dataset D, with centroid (mean) as μ, the ratio can be defined as,

$$\frac{intra - clusterdistance}{inter - clusterdistance} = \frac{\sum d(p, C(p))}{\sum_{p \epsilon D, q=C(p)} d(q, \mu)} \tag{1}$$

where $C(p)$ is centroid of cluster to which the point is assigned.

Using simple k-means algorithm [2], this ratio can be calculated easily. Using 10k iterations of k-means, over 10 simulations, the ratio was calculated.

As per the plot (Fig. 4), till k=10, the ratio keeps on increasing, but, then, for few value of k it does not change much, and then it decreases, but not monotonically. Sometimes, spikes are also formed after this value, but, it doesn't reach

[2] can be found under Code/Q2

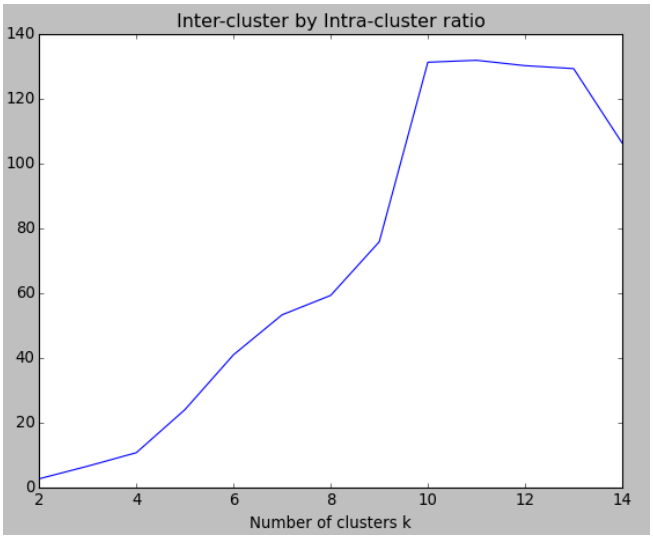

Fig. 4: Elbow method

the maximum as that is found at k=10 in the plot.

When the value of k is less, the centroids are comparatively far away, and clusters contain too many points, covering considerable amount of region in space, therefore, the intra cluster distance is high. And, gradually, with increase in number of clusters, it (intra-cluster distance) decreases. The similar way, inter-cluster decreases, because, now new clusters emerging, the overall distance between centroids of clusters decrease.

But, after a point, the intra-cluster does not change much. But, inter-cluster varies with value k, and the behaviour can not be determined. e.g. two clusters partitioned in 3 clusters will increase the intra cluster distance, but, may have lesser distance when partitioned in 4 clusters. So, after optimal value of k, the graph does not behave regular, and some peaks and flats may be observed depending on the data.

The plot of intra over inter (Fig. 5)can also used to determine the value of k. It looks similar to the reflection the graph over X-axis (also known as knee method).

3 Single Link Hierarchical Clustering

(a) From the data point, we can generate a distance matrix where, d[i][j] gives distance between point i and point j.
The distance matrix for the input will be as shown in Table 3.
Therefore, dendrogram will look similar to the Fig. 5.

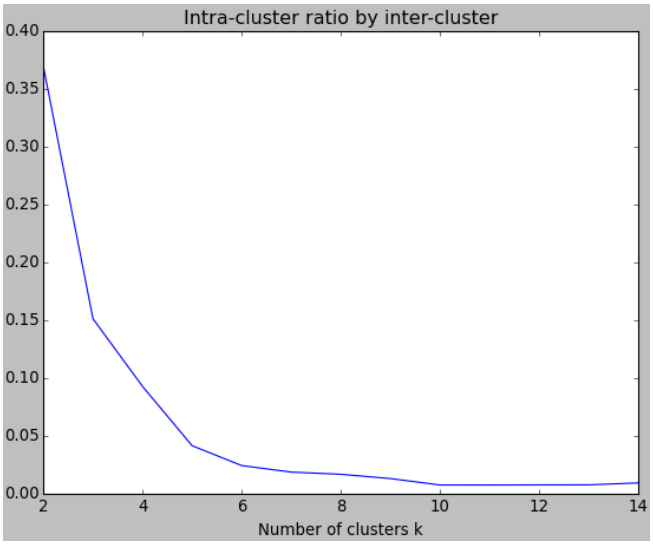

Fig. 5: Intra vs Inter cluster distances graph

0	0.234307	0.21587	0.367696	0.34176	0.235372
0.234307	0	0.143178	0.194165	0.143178	0.243516
0.21587	0.143178	0	0.158114	0.284605	0.10198
0.367696	0.194165	0.158114	0	0.284253	0.219545
0.34176	0.143178	0.284605	0.284253	0	0.386005
0.235372	0.243516	0.10198	0.219545	0.386005	0

Table 3: Distance Matrix for input

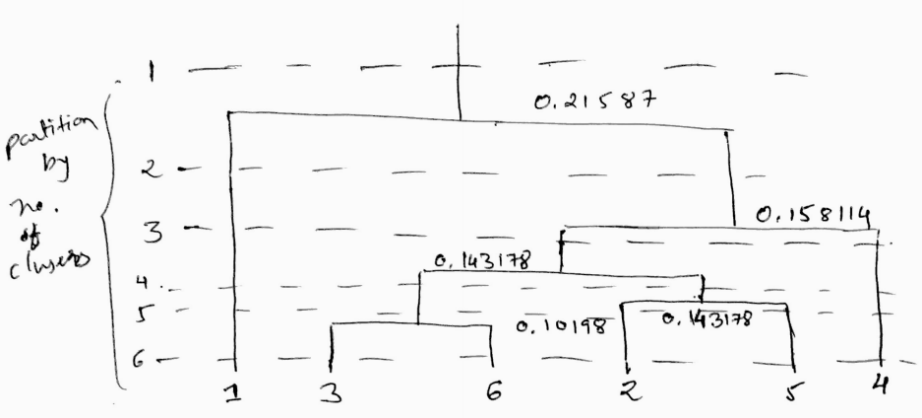

Fig. 6: Dendrogram

Based on the number of partition we want, we can cut the dendrogram accordingly at that level, and get required clusters, without re-processing it for different values of k.
Here, 2-3 and, 2-5 distances are same, therefore, when combining points, 2-5 were given preference over 2-3 because, we would rather like to have points distributed among clusters than a cluster with too many points. But, depending on application, 2-3-6 or 3-5-6 can also be made as a single cluster, for partitioning points into 4 clusters.

(b) In single-link hierarchical clustering, we consider minimum distance from one point from one cluster to second point from other cluster, and each iteration combines two clusters. So, starting with each point as an individual cluster, we need to do n iterations for the algorithm.
Without support of any data structure, for each iteration to combine two clusters (even single points are also considered clusters) we need to find minimum distance from all the possible distances between the clusters to combine it. Going through all distances is of order n^2 and, we need to update the distances for all the distances of points in the cluster. So, for each iteration $c_1n^2 + c_2n^2$ in the worst case.

So, overall complexity would be $n(c_1n^2 + c_2n^2)$. So, the worst time complexity will be $O(n^3)$.

(c) The complexity can be reduced by using **Priority Queue** data structure. The idea is pretty simple. Keep a priority queue of distances, which takes computational complexity of$O(n)$ to get built. And, after that, each removal from priority queue takes at most logn time. Therefore, for each iteration (when two clusters combine to form a single cluster), we need to remove the distance from priority queue, and, update the clusters.

To consider single link, we also need to maintain disjoint set structure, which we can join, because, addition of one point will change the minimum distance to other points, too. This takes $O(n)$ time. So, for n distances, when actually merge operation is done, it will take, $cn * n$ time, and for remaining $\frac{n(n-1)}{2} - n$ distances, the iteration takes $O(logn)$ time.

Therefore, Computational complexity $= c_1 \times (\frac{n(n-1)}{2} - n) + c_2 \times n \times n + c_3$
$= O(n^2logn) + O(n^2) = O(n^2logn)$.