CS6720: HW1

Abhishek Yadav, CS12B032

Tuesday 9th February, 2016; 00:22

Q1

(a) As mentioned in the problem statement, a file named **CS12B032_1a.txt** is included within **CS12B032_HW2/Q1** which contains the frequent itemsets with 10% minimum support threshold. Same is listed as following:

32
38
39
41
48
38 39
39 41
39 48
41 48

(b) I've used **fp_growth** python-implementation from `https://github.com/enaeseth/python-fp-growth/` or `https://pypi.python.org/pypi/fp-growth`. Following are the plots of time vs minsup(%).

From figure2 it is evident that fp_grwoth outperforms apriori for minusp threshold $\geq$ 2%, whereas figure1 shows otherwise, i.e. apriori performs outperforms fp_growth for minsup threshold $\leq$ 2%. For minsups $\geq$ 2%, number of frequent patterns mined by both the algorithms are almost similar and same in number, and fp_growth performs better than apriori which is in accordance with the theoretically proven fact about fp_growth being faster than apriori. But, for smaller minsups($\leq$ 2%) fp_growth doesn't perform well as compared to apriori, which can be attributed to the fact that number of patterns mined by fp_growth are larger than those mined by apriori.
frequent-patterns: minsup(.2%-1.2%)

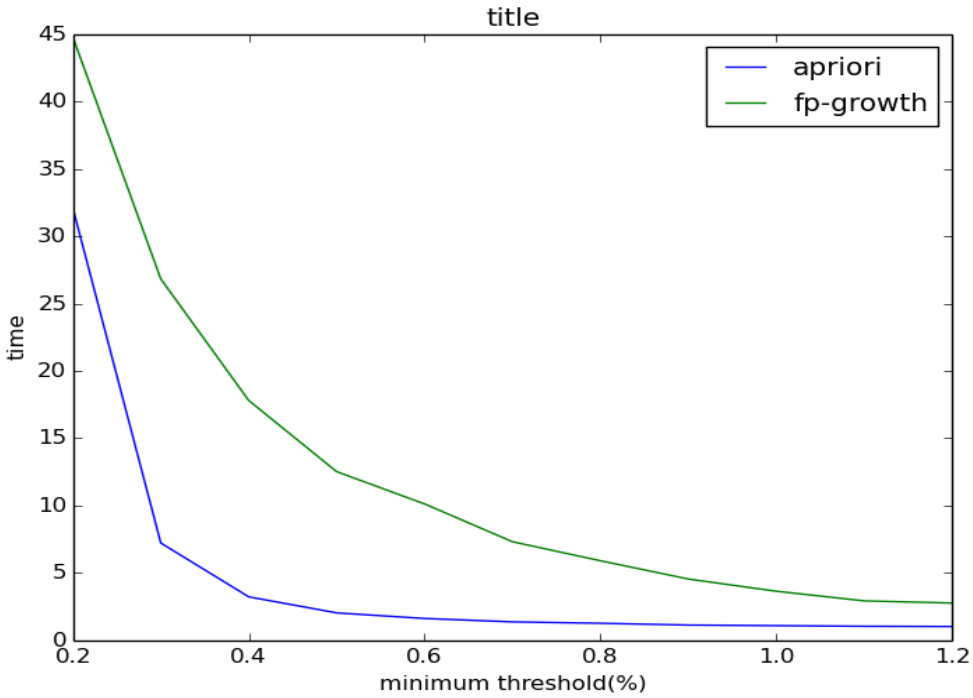


Figure 1: minsup(%)(.2-1.2)

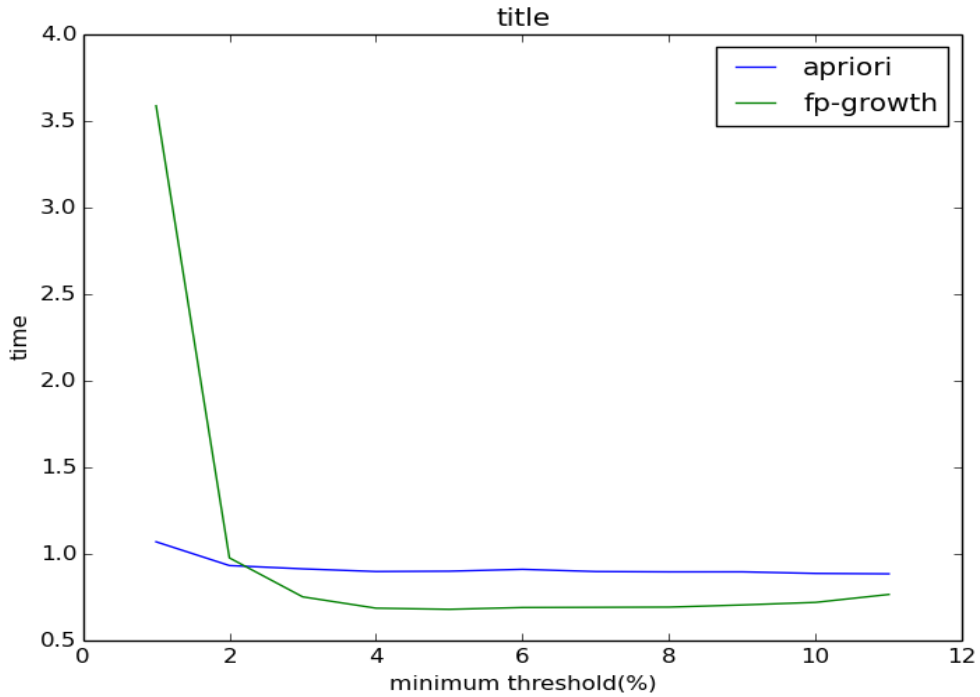
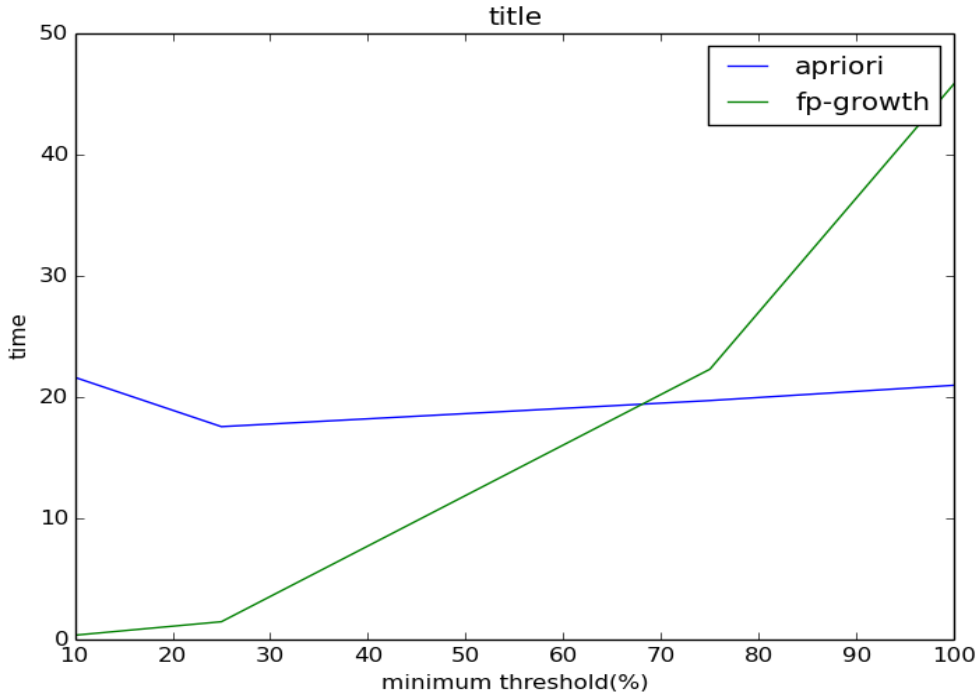


Figure 2: minsup(%)(1-10)

minsup(%)	fp_growth	apriori
.2	C1:956, C2:1631, C3:642, C4:94, C5:7	C1:963, C2:1145, C3:508 C4:92, C5:7
.3	C1: 515, C2: 834, C3: 298, C4: 43, C5: 4	C1:521, C2:582, C3:255, C4:47, C5:4
.4	C1: 317, C2: 504, C3: 172, C4: 25, C5: 2	C1:320, C2:339, C3:150, C4:26,C5:2
.5	C1: 221, C2: 359, C3: 119, C4: 19, C5: 1	C1:221, C2:237, C3:103, C4:19, C5:1
.6	C1: 165, C2: 250, C3: 83, C4: 11	C1:165, C2:173 C3:69, C4:11

It can be easily seen that fp_growth mines more patters/rules than apriori which may be causing it to perform poor as compared to apriori.



(c) At .5% minimum support threshold, when larger sample of the dataset is randomly picked, which is comparable to original dataset, apriori beats fp_growth because fp_growth mines more rules than apriori for a given minsup whereas for smaller dataset fp_growth outperforms apriori due to reduced transactions and hence less rules mining.

Q2

Approach:

(a) **Sum of Squared Error(SSE)**: This approach uses the fact that the **KMeans** algorithm tries to minimize the average **intra-cluster** sums of squared-distanced and maximize the inter-cluster squared distances which are calculated as following:

$$\text{Intra} = \frac{1}{N}\sum_{i=1}^{K}\sum_{x \in C_i} ||x - z_i||^2$$

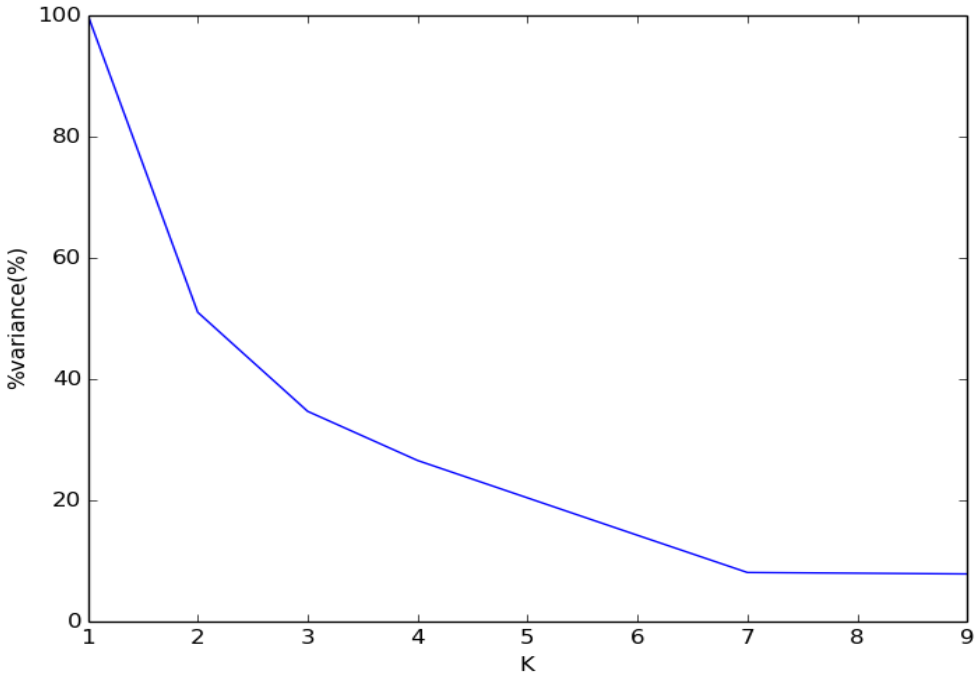


Figure 3: intra-cluster Variance vs K plot for 10 values of K

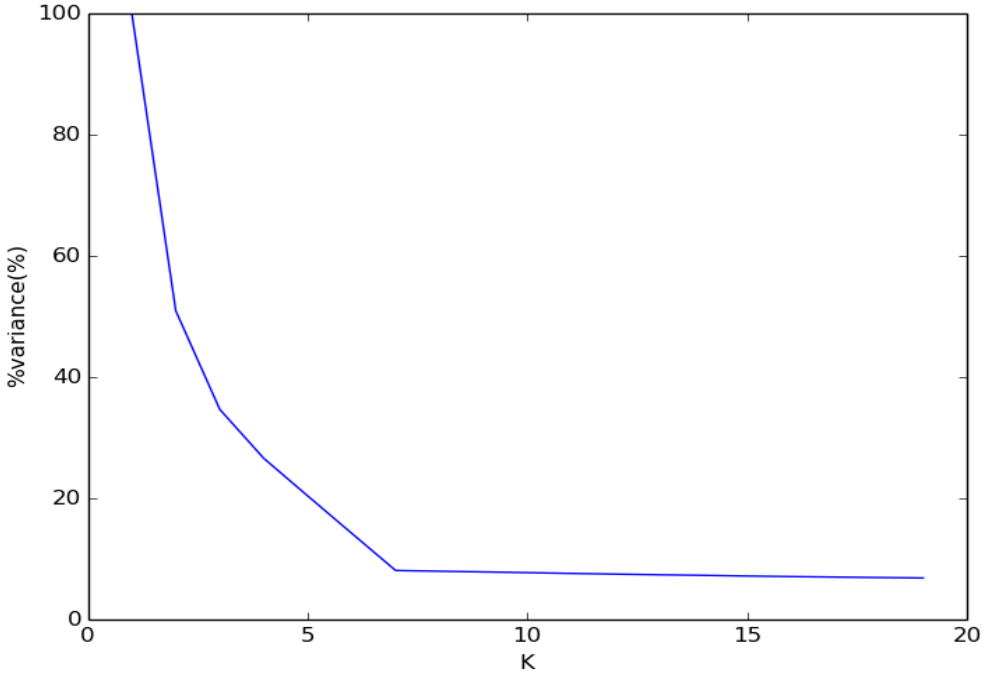


Figure 4: intra-cluster Variance vs K plot for 20 values of K

$$\text{Inter} = \min(||z_i - z_j||^2)$$

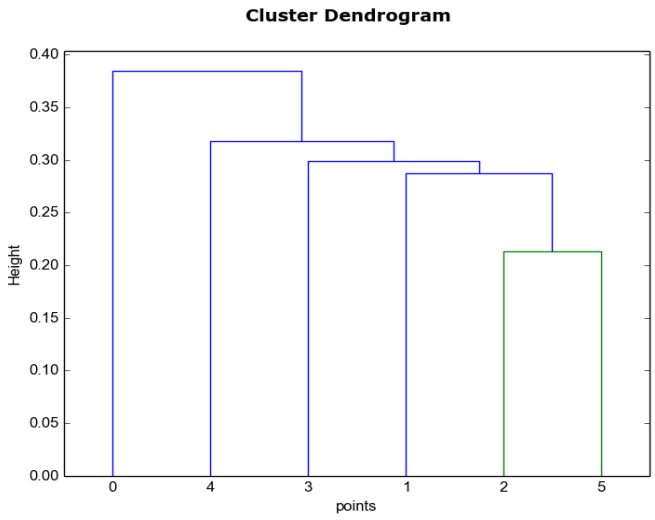
where, z_i and z_j are the cluster centroids and $i \neq j$. Just considering the average of sums of squared-intra-cluster-distances, we get the first two plots for two different values of K(= 10, 20). Here, we see that the graph bends and becomes almost a straight line after k=7, which implies that the average of sums squared-intra-cluster-distances remain constant $\forall K \geq 7$ which means that optimum number of clusters in the given data-set are **7**.

Next, I tried calculating the **Intra/Inter** ratio, which is also a good indicator of optimum clustering in various cases. Smaller the Intra and larger the Inter values, smaller the ratio will be. Hence, smallest ratio of these two will indicated the optimal clustering. From figure3 and figure4 it can be seen that at K=7 the ratio is very small and $\rightarrow 0$, which means that K=7, is the optimal number of clusters for the given dataset.

Note: The associated script is in the directory CS12B032_HW2/Q2

Q3

(a)



(b) If no data structures(i.e. no distance matrix) are used, computational complexity of clustering n data points will be $\mathbf{O}(n^3)$ which can be explained as follows(Assuming that data points along with their attributes are stored somewhere):

Since distances are not stored, at every step of the algorithm we need to calculate the distance between each cluster, which is initially n. So total number of distance calculations are:

$$\binom{n}{2} + \binom{n-1}{2} + \binom{n-2}{2} + \ldots\ldots + \binom{2}{2}$$

$$= \text{coefficient of } x^2 \text{ in } (1+x)^n + (1+x)^{n-1} + \ldots\ldots + (1+x)^2$$

$$= \text{Coefficient of } x^2 \text{ in } \frac{(1+x)^{n+1} - (1+x)^2}{x}$$

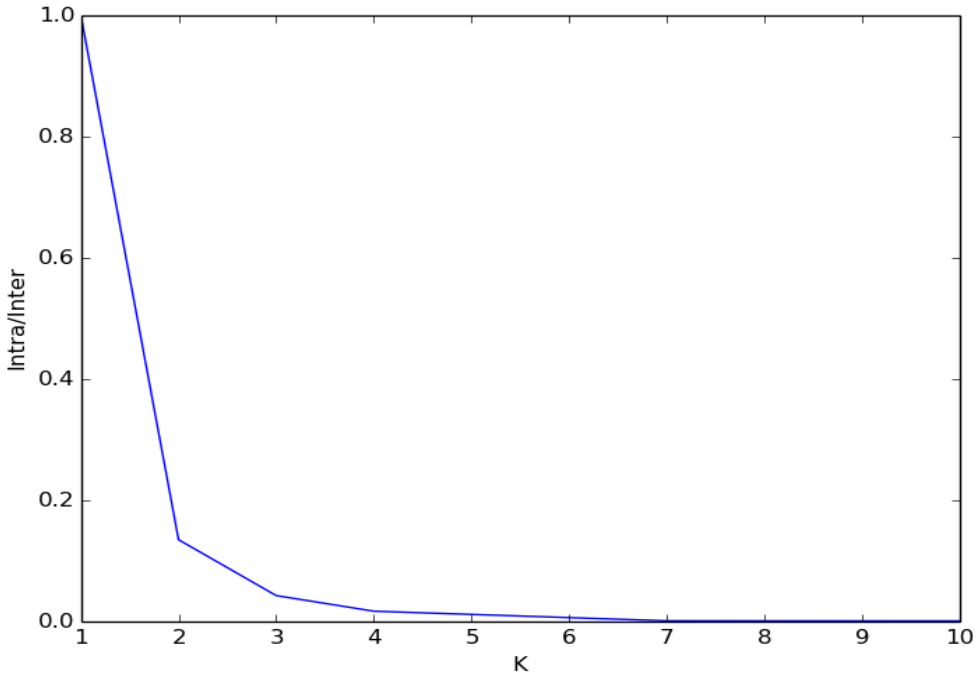


Figure 5: Intra/inter Vs K

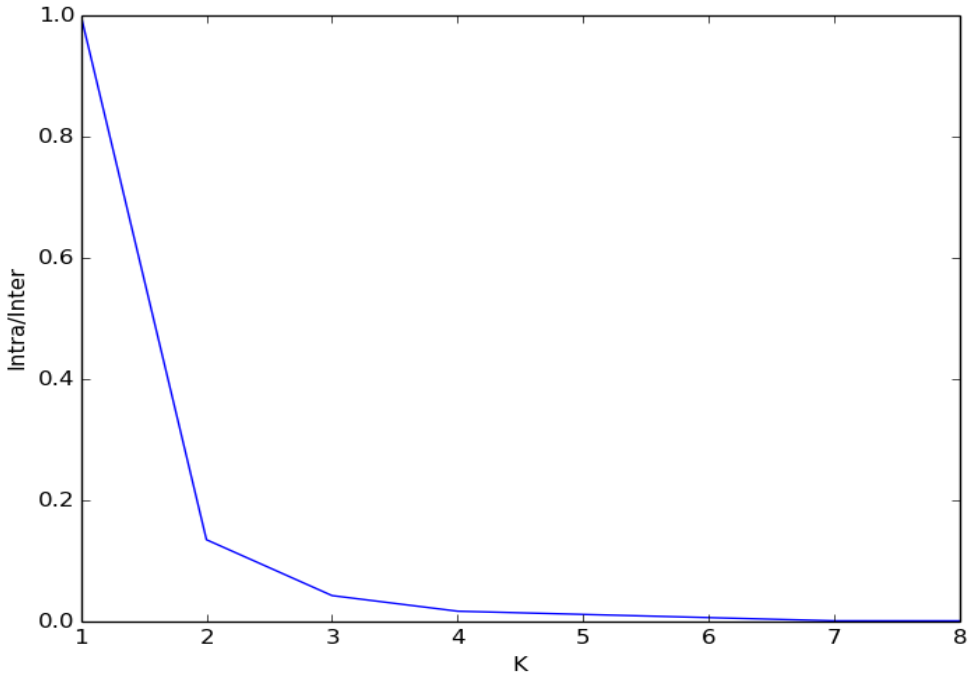


Figure 6: Intra/Inter Vs K

$= \frac{n(n+1)(n-1)}{3!} = \mathbf{O}(n^3)$

(c) $\mathbf{O}(n^2\mathbf{log(n)})$ algorithm to cluster n data points can be achieved using min-priority queue. The steps of the algorithm are as follows:

- First, compute the distance between all pairs of data points which is an $\mathbf{O}(n^2)$ step.
- Store the pairs and the distance between them in a priority queue. Since, there are $O(n^2)$ pairs, so this step also takes $O(n^2)$.
- While merging the clusters C_1 and C_2, we remove all the entries in the priority-queue involving one of these clusters. In this step, since there can be maximum 2n deletions, and and element in a priority-queue can be deleted in **O(log n)** so this step takes **O(nlog n))**.
- Finally, we compute the distance between the new cluster and the remaining clusters, which is an **O(nlog n)** because there can be maximum n insertion in the priority-queue and each insertion takes O(log n) time. Since, last two steps can be executed at most n times, and first two are executed only once, the complexity of the algorithm is:

$$O(n^2log(n)) + O(n^2log(n)) + 2 * O(n^2) = O(n^2log(n))$$