COMP5331 Knowledge Discovery in Databases (Fall Semester 2019)
Homework 1 Solution

Q1

(a)
Yes. It can be adapted.

First of all, we obtain the 2-sequence by scanning the database.
(NOTE: The 2-sequence contains two kinds of sequences – (1) the sequence contains only one *timestamp entry* (e.g. <{D, E}>) and (2) the sequence contains two or more timestamp entries (e.g. <{D}, {E}>).

The Apriori-like algorithm is described as follows.

1. k=2
2. Find all frequent 2-sequences and store them in L_k
3. repeat
4. k=k+1
5. Generate candidate k-sequences from L_{k-1} (which will be described later) and store them in C_k
6. Scan the database and count the support of each candidate in C_k
7. Find the k-sequence in C_k with support ≧ minsupport and store them in L_k
8. until L_k = empty set
9. return L_i for i=2,…k

e.g.
We obtain the following 2-sequences.
{<{A}, {D}>, <{A}, {E}>, <{A}, (G}>, <{D, E}> }

Next, we generate the candidate 3-sequences by the join-and-prune process.

The *join* step of the generation process is described as follows.
A sequence $s^{(1)}$ is joined with another sequence $s^{(2)}$ only if the subsequence obtained by dropping the first item in $s^{(1)}$ is identical to the subsequence obtained by dropping the last item in $s^{(2)}$. The resulting candidate is the sequence $s^{(1)}$, concatenated with the last item from $s^{(2)}$. The last item from $s^{(2)}$ can either be joined into the same timestamp element as the last item in $s^{(1)}$ or different timestamp elements depending on the following conditions.

1. If the last two items in $s^{(2)}$ belong the same timestamp element, then the last item in $s^{(2)}$ is part of the last timestamp element in $s^{(1)}$ in the joined sequence. (e.g. Suppose we have frequent sequences <{A},{B},{C}> and <{B},{C, D}> in L_{k-1}. Candidate <{A},{B},{C, D}> is obtained by joining <{A},{B},{C}> and <{B), {C, D}>).
2. If the last two items in $s^{(2)}$ belong to different timestamp elements, then the last item in $s^{(2)}$ becomes a separate timestamp element appended to the end of $s^{(1)}$ in the joined sequence. (e.g. Suppose we have frequent sequences <{A},{B},{C}> and <{B},{C},{D}> in L_{k-1}. Candidate <{A},{B},{C},{D}> is obtained by joining <{A},{B},{C}> and <{B},{C},{D}>).

e.g. In the running example, we obtain one candidate 3-sequence <{A}, {D,E}> (by joining <{A}, {D}> and <{D, E}>) after the join step.

The *prune* step of the generation process is described as follows.
A candidate k-sequence is pruned if at least one of its (k-1)-sequence is infrequent.

For example, <{A}, {D,E}> is a candidate 3-sequence. We need to check whether <{A}, {E}> is a frequent 2-sequence (NOTE: We do not need to check whether <{A}, {D}> and <{D, E}> are frequent 2-sequence because <{A}, {D,E}> was generated from these two frequent sequences). Since <{A}, {E}> is frequent, <{A}, {D,E}> is also considered as a candidate 3-sequence after the prune step.

Then, we do the *counting* step to count the support of each candidate in the set.
As the support of <{A}, {D,E}> is 2, then it is one of the final results.

We repeat the process until L_k is an empty set.

In our running example, all sequences with support at least 2 are {<{A}, {D,E}> }

(b)
No. This is because if a k-sequence is frequent, it is not necessarily true that all its sub-sequences are frequent.

Consider an example with the following 4 transactions.

Customer W, Rich = Yes, time 1, items A, B, C
Customer X, Rich = Yes, time 2, items A, B, C
Customer Y, Rich = No, time 3, items A, B, C
Customer Z, Rich = No, time 4, items A, B

The above transactions can be transformed into four sequences as follows.
W, Yes: <{A, B, C}>
X, Yes: <{A, B, C}>
Y, No: <{A, B, C}>
Z, No: <{A, B}>

The supports of <{A, B, C}> with respect to values "Yes" and "No" are 2 and 1, respectively.
Thus, the important ratio of <{A, B, C}> is 2/1 = 2.

Consider a sub-sequence of <{A,B,C}>, i.e., <{A, B}>.
The supports of <{A, B}> with respect to values "Yes" and "No" are both 2.
Thus, the important ratio of <{A, B}> is 2/2 = 1 < 2.

Note that <{A, B}> is a subsequence of <{A, B, C}> but the important ratio of <{A, B}> is smaller than the important ratio of <{A, B, C}>. Thus, the apriori property is not satisfied. Thus, we cannot adapt the Apriori algorithm.

Q2

(a)
(i) {C}, {D}

(ii) {C, D}, {A, C}, {A, D}

(iii) {A, C, D}, {B, E, F}

(b)
Yes. It can be adapted.

There is a support threshold s which changes over time.
Initially, it is set to 1.
According to the current threshold value equal to 1, we build an FP-tree.

Initially, we have three variables, L1, L2 and L3.
L1 is a variable storing the current-best $S_{1,2}$.
L2 is a variable storing the current-best $S_{2,2}$.
L3 is a variable storing the current-best $S_{3,2}$.
Initially, each of these variables is set to an empty set.

We also have three variables, x1, x2 and x3.
x1 is a variable storing the 2^{nd} (or l-th) greatest value of the multi-set containing the supports of all itemsets in L1
x2 is a variable storing the 2^{nd} (or l-th) greatest value of the multi-set containing the supports of all itemsets in L2
x3 is a variable storing the 2^{nd} (or l-th) greatest value of the multi-set containing the supports of all itemsets in L3
Initially, each of these variables is set to 1.

From the FP-tree above, construct the FP-conditional tree for each item (or itemset) according to the current value of the support threshold s.

The method is the same as the original FP-growth algorithm.
However, there is the following modification.
Whenever we generate the frequent itemsets from the current conditional FP-tree according to the current threshold s (which is being updated), for each frequent itemset just generated, if the size of the itemset is K where K = 1, 2 and 3, we do the following. (Otherwise, we do nothing).

a. Set a variable x to the 2^{nd} (or l-th) greatest value of the multi-set containing the supports of all itemsets in LK (If there are fewer than 2 (or l) values in this multi-set, x is set to 1.)
b. If the support of this itemset is greater than x, we insert it into LK.
c. Update variable x again to be the 2^{nd} (or l-th) greatest value of the multi-set containing the supports of all itemsets in the updated LK
d. Remove all itemsets in LK with support smaller than x
e. Update xK to max{xK, x}
f. Update the support threshold s to be min{x1, x2, x3}

Example

Counting:

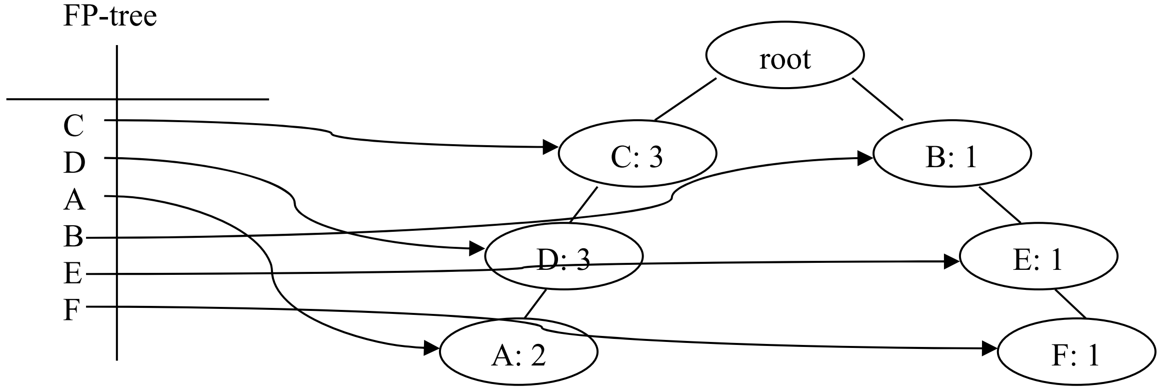
A	2
B	1
C	3
D	3
E	1
F	1

Sorting

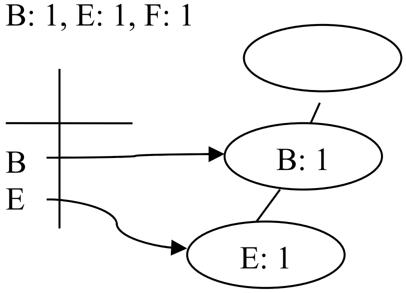
C	3
D	3
A	2
B	1
E	1
F	1

Items	(Ordered) frequent items
C, D	C, D
B, E, F	B, E, F
A, C, D	C, D, A
A, C, D	C , D, A

FP-tree



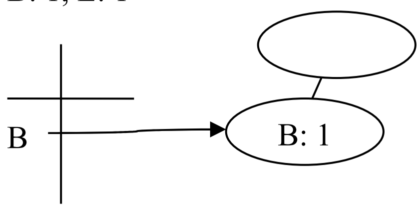
Conditional FP-tree on F (count = 1)
B: 1, E: 1, F: 1



$L_1 = \{F: 1\}$
$L_2 = \{BF: 1, EF: 1\}$
$L_3 = \{BEF: 1\}$

Conditional FP-tree on E (count = 1)

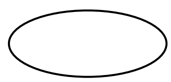
B: 1, E: 1



L_1 = {E:1, F: 1}
L_2 = {BE: 1, BF: 1, EF: 1}
L_3 = {BEF: 1}

Conditional FP-tree on B (count = 1)

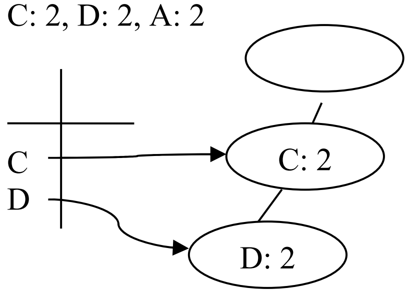
B: 1



L_1 = {B: 1, E:1, F: 1}
L_2 = {BE: 1, BF: 1, EF: 1}
L_3 = {BEF: 1}

Conditional FP-tree on A (count = 2)

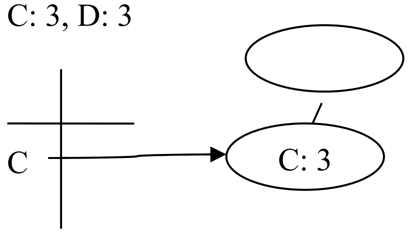
C: 2, D: 2, A: 2



L_1 = {A: 2, B: 1, E: 1, F: 1}
L_2 = {AC: 2, AD: 2}
L_3 = {ACD: 2, BEF: 1}

Conditional FP-tree on D (count = 3)

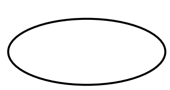
C: 3, D: 3



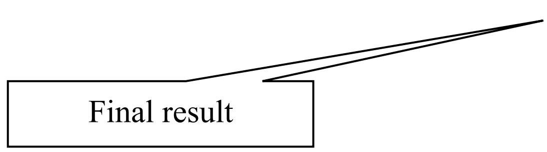
L_1 = {A: 2, D: 3}
L_2 = {AD: 3, AC: 2, AD: 2}
L_3 = {ACD: 2, BEF: 1}

Conditional FP-tree on C (count = 3)

C: 3



L_1 = {C: 3, D: 3}
L_2 = {AD: 3, AC: 2, AD: 2}
L_3 = {ACD: 2, BEF: 1}



(c)

1. By finding $S_{k,l}$, we control the number of the resulting frequent itemsets (e.g., in case that finding frequent itemsets in the traditional problem returns a huge number of frequent itemsets, finding the frequent itemsets of $S_{k,l}$ can reduce the resulting frequent itemsets by returning some top ones only.).

2. In the traditional problem of finding frequent itemsets, setting the hard threshold of support is not user-friendly. To deal with this issue, instead of specifying a hard "absolute" threshold, we can use a soft "relative" threshold.

Q3.

(a) No. This is because this algorithm terminates when k is equal to the total number of data points. In this case, e_k is equal to 0. Even if k is larger, e_k is also equal to 0 and thus e_k converges. The value k is not what we desire because we want to group "similar" points.

Alg:

a. First, multiply each attribute of each data point by a positive real number Δ such that the "closest" pair between two points is at least 1.0.

b. Second, define d_k to be the product of the distances between any two clusters according to distance "single linkage".

c. We change Step 4 from e_k to d_k .

d. We change Step 5 to the following:

We repeat Step 3 to Step 4 for different possible values of k and obtain the corresponding values of d_k.

e. We find the k s.t. d_k is maximized.

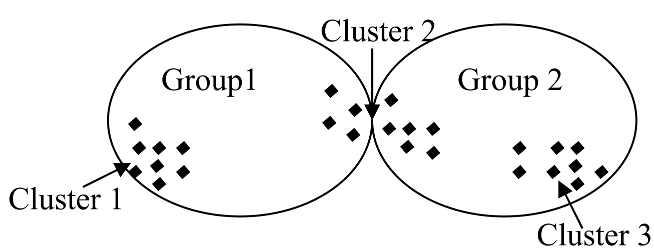
The reasons are:

1. If k is larger than the number of clusters, in k-means, one single cluster will be represented by more than one means. In other words, the real cluster is split into a number of groups in k-means. It is expected that these groups are very close and thus d_k is small.

2. If k is smaller than the number of clusters, in k-means, two or more clusters will be represented by one mean/group.

There are two cases:

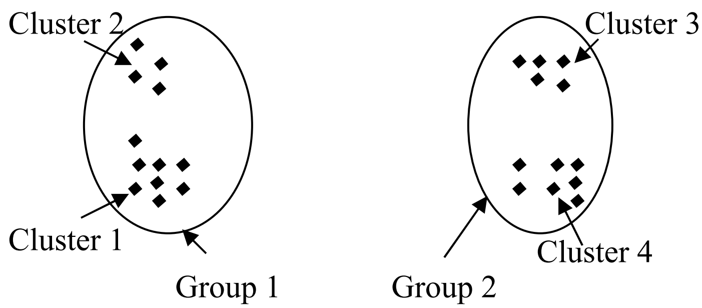
- Case 1:



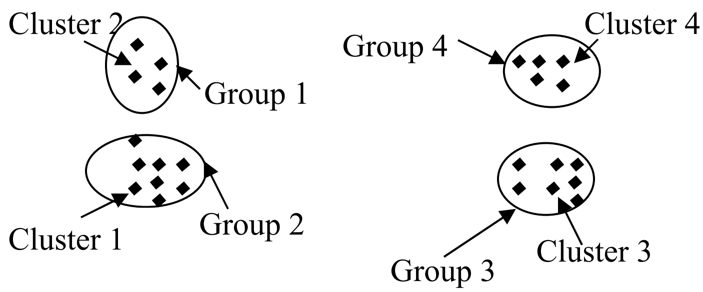
One cluster is split into two or more groups.

In the example, when k=2, cluster 2 is split into group 1 and group 2. It is easy to see that d_k is small.

- Case 2:



In this example, when k=2, cluster 1 and cluster 2 completely belong to group 1 while cluster 3 and cluster 4 completely belong to group 2.
In this case, we know that the separation between group 1 and group 2 is large and thus d_k is large.
However, when k=4, we have



Obviously, d_k in this case is larger.
In conclusion, if we find k s.t. d_k is maximized, we can find a good value for the number of clusters.

(b) The distance metric between a point p and the cluster center c can be modified as follows.
Let X_p be a "special" cluster containing only p.
Let X_c be the cluster representing the cluster center c which contains some assigned points.
The first assigned point of X_c is the point closest to c for each c.
The distance metric between a point p and the cluster enter c is equal to the distance between X_p and X_c according to the single linkage distance in an iterative manner.
While there are still remaining points, find a remaining point p and a cluster X such that the distance between p and X is the shortest and assign p to cluster X.

Q4.

(a) (i)

	1	2	3	4	5	6	7	8
1	0							
2	12	0						
3	2	12.17	0					
4	12.65	4	12.17	0				
5	4.24	15.3	3.16	15.03	0			
6	14.32	3.61	14.87	7.28	18.03	0		
7	10.3	11.4	12.08	14.76	14.42	10.82	0	
8	20.62	21.19	18.68	17.46	18.79	24.7	29	0

(ii)

	1	2	3	4	5	6	7	8
1	0							
2	12	0						
3	2	12.17	0					
4	12.65	4	12.17	0				
5	4.24	15.3	3.16	15.03	0			
6	14.32	3.61	14.87	7.28	18.03	0		
7	10.3	11.4	12.08	14.76	14.42	10.82	0	
8	20.62	21.19	18.68	17.46	18.79	24.7	29	0

Mean of the following clusters:

{1} : (20, 15)
{2} : (8, 15)
{3} : (20, 17)
{4} : (8, 19)
{5} : (23, 18)
{6} : (6, 12)
{7} : (15, 6)
{8} : (15, 35)

	(1,3)	2	4	5	6	7	8
(1,3)	0						
2	12.04	0					
4	12.37	4	0				
5	3.61	15.3	15.03	0			
6	14.56	3.61	7.28	18.03	0		
7	11.18	11.4	14.76	14.42	10.82	0	
8	19.65	21.19	17.46	18.79	24.7	29	0

Mean of the following clusters:
{1, 3} : (20, 16)
{2} : (8, 15)
{4} : (8, 19)
{5} : (23, 18)
{6} : (6, 12)
{7} : (15, 6)
{8} : (15, 35)

	(1,3,5)	2	4	6	7	8
(1,3,5)	0					
2	13.11	0				
4	13.21	4	0			
6	15.71	3.61	7.28	0		
7	12.24	11.4	14.76	10.82	0	
8	19.29	21.19	17.46	24.7	29	0

Mean of the following clusters:
{1, 3, 5} : (21, 16.67)
{2} : (8, 15)
{4} : (8, 19)
{6} : (6, 12)
{7} : (15, 6)
{8} : (15, 35)

	(1,3,5)	(2,6)	4	7	8
(1,3,5)	0				
(2,6)	14.35	0			
4	13.21	5.59	0		
7	12.24	10.97	14.76	0	
8	19.29	22.94	17.46	29	0

Mean of the following clusters:

{1, 3, 5} : (21, 16.67)
{2, 6} : (7, 13.5)
{4} : (8, 19)
{7} : (15, 6)
{8} : (15, 35)

	(1,3,5)	(2,4,6)	7	8
(1,3,5)	0			
(2,4,6)	13.73	0		
7	12.24	12.08	0	
8	19.29	21.11	29	0

Mean of the following clusters:

{1, 3, 5} : (21, 16.67)
{2, 4, 6} : (7.33, 15.33)
{7} : (15, 6)
{8} : (15, 35)

	(1,3,5)	(2,4,6,7)	8
(1,3,5)	0		
(2,4,6,7)	12.31	0	
8	19.29	22.74	0

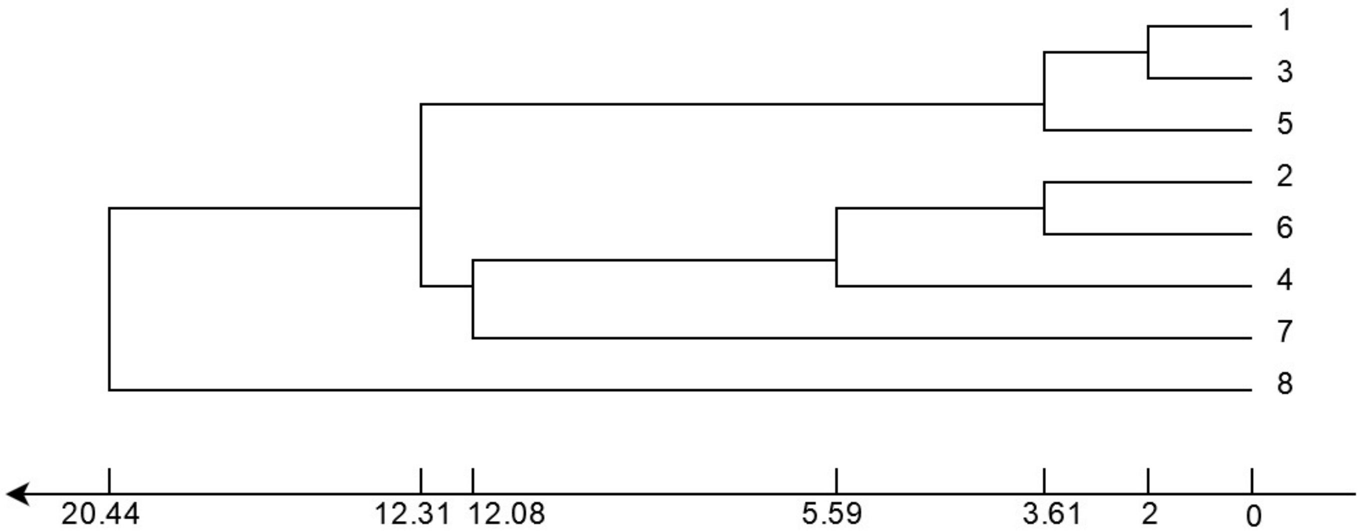
Mean of the following clusters:

{1, 3, 5} : (21, 16.67)
{2, 4, 6, 7} : (9.25, 13)
{8} : (15, 35)

	(1,2,3,4,5,6,7)	8
(1,2,3,4,5,6,7)	0	
8	20.44	0

Mean of the following clusters:
{1, 2, 3, 4, 5, 6, 7} : (14.29, 14.57)
{8} : (15, 35)

Dendrogram:



(b)
No.

In the worst case, there is a need to execute the whole algorithm from scratch.
Consider the case when the new data point is involved in the "closest" pair with another point (out of the 8 data points). If this is the case, all steps computed in (a) have to be re-computed.

Q5

(a)

Algorithm:

1. For each data point p
   Perform a range query from p with radius ε
   N(p) ← the result of the range query
   If |N(p)|≥ MinPts,
     Mark p as a core point.
2. For each core point p
   Generate a cluster C for p.
3. While there exist two clusters C1 and C2 such that there exist p1 ∈ C1 and p2 ∈ C2 where N(p1) contains p2
   Merge these two clusters.
4. For each point p where |N(p)| < MinPts,
   if N(p) contains a core point q,
     Assign p to the cluster q belongs to.

(b)

(i) Let p' be the new data point.

Algorithm:

1. Perform a range query from p' with radius ε
2. Find a set S of points such that each point p in this set satisfies (1) the ε-neighborhood of p (on the original dataset together with the new point), denoted by N(p), includes p', and (2) |N(p)| == MinPts.
3. For each point p in S (together with p' when |N(p')| >= MinPts),
     Mark p as a core point
     Generate a cluster C for p
4. While there exist two clusters C1 and C2 such that there exist p1 ∈ C1 and p2 ∈ C2 where N(p1) contains p2
   Merge these two clusters.
5. If |N(p')| < MinPts and N(p') contains a core point q,
     Assign p' to the cluster q belongs to.

(ii) Let n be the number of data points.

Step 1 could be done in O(log n) with a range query on all points (with the preprocessing phase of building an index on all points which takes O(n log n)).

Step 2 could be done in O(log n + l) where l is the total size of S.

(This could be implemented by a point query on a set of all spheres with the radius equal to ε.)

Step 3 could be done in O(l) time.

Consider Step 4. Consider that we build an index on the spheres of all points with the radius equal to ε for each separate cluster. Let m be the greatest number of points in a cluster. The time of building a single index is O(m log m). Let k be the total number of clusters. The total time of building all indices is O(k m log m). Step 4 could be done as follows. For each data point p (which belongs to a cluster C1) and for each index corresponding to a cluster C2 not containing this point p, we issue a point query on this index. If the answer of this query is non-empty, we merge C1 with C2 conceptually. Otherwise, we do nothing.

Consider Step 5. It takes O(1).

Finally, we maintain the index for the final merged cluster by re-building all indices from scratch which takes O(k m log m). Note that k and m are typically much smaller than n in practice.

Thus, the overall time complexity is O(log n + l + k m log m).