CS412: Introduction to Data Mining, Fall 2023, Final

Name: Teng Hou (tenghou2)

Q1

a. *We need to count the time that a single letter appears in a sequence. Their supports are:*

patterns	**support**
$\langle a \rangle$	3
$\langle b \rangle$	5
$\langle c \rangle$	4
$\langle d \rangle$	3
$\langle e \rangle$	3
$\langle f \rangle$	2
$\langle g \rangle$	1
$\langle h \rangle$	1

As the minimum support is 2, so g and h are both removed, then the length-1 frequent sequential patterns in SDB_1 are:

patterns	**support**
$\langle a \rangle$	3
$\langle b \rangle$	5
$\langle c \rangle$	4
$\langle d \rangle$	3
$\langle e \rangle$	3
$\langle f \rangle$	2

b. *The projected database for prefix* $\langle bb \rangle$ *is:*

sequence
$\langle (\_d) \rangle$
$\langle (ce) \rangle$
$\langle f \rangle$
$\langle cb(ade) \rangle$

As the size of the projected database for prefix $\langle bb \rangle$ is 4, which is larger than minimum support which is 2. So it is a length-2 frequent pattern in SDB_1

c. *The projected database for prefix* $\langle bd \rangle$ *is:*

sequence
$\langle bcb(ade)\rangle$

As it could be found in S_1 and S_5, so the support of it is 2. And as it is equal to minimum support 2. So it is a length-2 frequent pattern in SDB_1.

d. *The projected database for prefix* $\langle b\rangle$*:*

sequence
$\langle (bd)\rangle$
$\langle (\_f)(fg)b(ce)\rangle$
$\langle (\_f)(ah)abf\rangle$
$\langle (\_e)\rangle$
$\langle (\_d)bcb(ade)\rangle$

e. All the subsequences and corresponding support are:

subsequences	**sup**
$\langle b\rangle$	5
$\langle ba\rangle$	2
$\langle bb\rangle$	4
$\langle bbc\rangle$	2
$\langle bbe\rangle$	2
$\langle bc\rangle$	2
$\langle bd\rangle$	2
$\langle (bd)\rangle$	2
$\langle be\rangle$	2
$\langle bf\rangle$	2
$\langle (bf)\rangle$	2
$\langle (bf)b\rangle$	2
$\langle (bf)f\rangle$	2

Q2

a. *To find the support s of rule $A \longrightarrow B(s,c)$, we need to find the frequency that A and B both appear in all TIDs, and by the table, the frequency is 4, and the total amount of TIDs is 10. So the support $s = 4/10 = 0.4$. to find the confidence of $A \longrightarrow B(s,c)$, we need to calculate $c = \frac{sup(A,B}{sup(A)}$, and the $sup(A,B)$ is 4 from table, and the $sup(A)$ is 7 from the table. So the confidence $c = \frac{sup(A,B)}{sup(A)} = \frac{4}{7} \approx 0.57143$*

b.

TID	**Items**
T1	A, B, C
T2	A, D, E
T3	B, D
T4	A, B, D
T5	A, C
T6	B, C
T7	A, C, D
T8	A, B, C, D, E
T9	B, C, D
T10	A, B, C, E

Then the C1 will be like:

Itemset
$\{A\}$
$\{B\}$
$\{C\}$
$\{D\}$
$\{E\}$

After the first scan, the $C1$ will be like:

Itemset	**sup**
$\{A\}$	7
$\{B\}$	7
$\{C\}$	7
$\{D\}$	6
$\{E\}$	3

And because the minsup is 3, so no item will be filtered at this time. So the F1 will be:

Itemset	**sup**
$\{A\}$	7
$\{B\}$	7
$\{C\}$	7
$\{D\}$	6
$\{E\}$	3

Then the C2 will be like:

Itemset
$\{A, B\}$
$\{A, C\}$
$\{A, D\}$
$\{A, E\}$
$\{B, C\}$
$\{B, D\}$
$\{B, E\}$
$\{C, D\}$
$\{C, E\}$
$\{D, E\}$

After the second scan, the C2 will be like:

Itemset	**sup**
$\{A, B\}$	4
$\{A, C\}$	5
$\{A, D\}$	4
$\{A, E\}$	3
$\{B, C\}$	5
$\{B, D\}$	4
$\{B, E\}$	2
$\{C, D\}$	3
$\{C, E\}$	2
$\{D, E\}$	2

As the minsup=3, after the second filter, the F2 will be like:

Itemset	**sup**
$\{A, B\}$	4
$\{A, C\}$	5
$\{A, D\}$	4
$\{A, E\}$	3
$\{B, C\}$	5
$\{B, D\}$	4
$\{C, D\}$	3

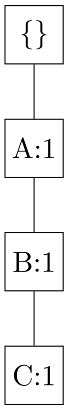
For C3, we apply a trick, we can do self-joining F2. From table of F2, we could get $\{A, B, C\}$ from $\{A, B\}$ and $\{A, C\}$. And we could get $\{A, D, E\}$ from $\{A, D\}$ and $\{A, E\}$. And we could get $\{B, C, D\}$ from $\{B, C\}$ and $\{B, D\}$. And we could get $\{A, C, D\}$ from $\{A, C\}$ and $\{C, D\}$. And we could get $\{A, B, D\}$ from $\{A, B\}$ and $\{B, D\}$. And $\{A, D, E\}$ is removed because we don't have $\{D, E\}$ in table F2. So C3 will be like:

Itemset	**sup**
$\{A, B, C\}$	3
$\{A, B, D\}$	2
$\{A, C, D\}$	2
$\{B, C, D\}$	2

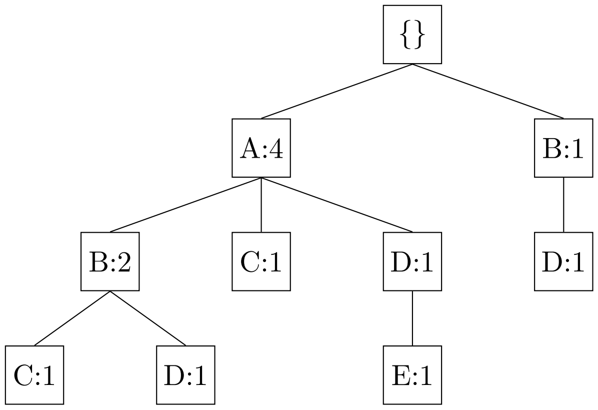
And as minsup=3, so $\{A, B, D\}$, $\{B, C, D\}$ and $\{A, C, D\}$ is also removed. Then the final result F3 is:

Itemset	**sup**
$\{A, B, C\}$	3

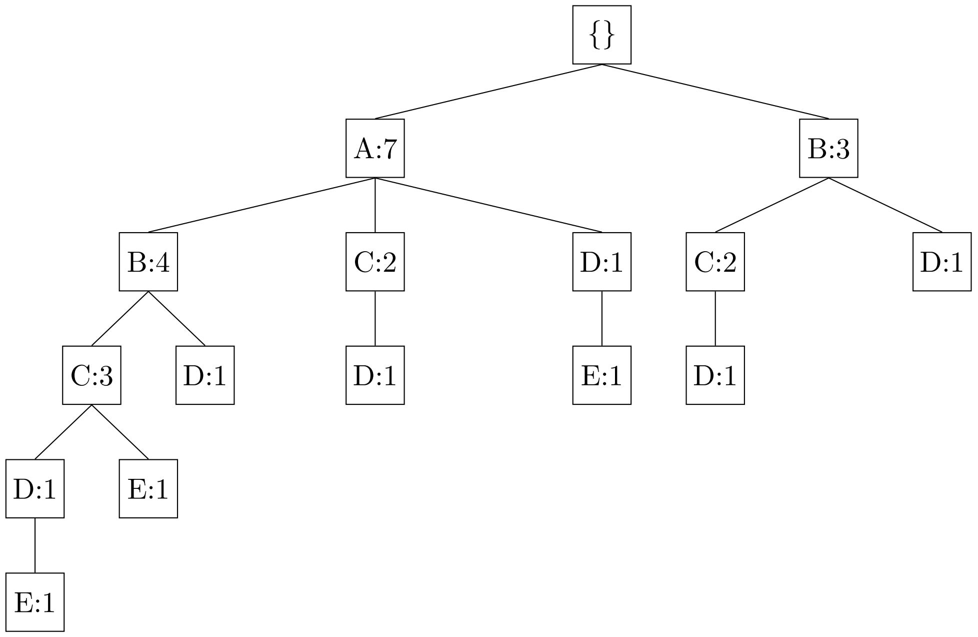
c. For the tree T_1:



For the tree T_5:



For the tree T_{10}:


{}
A:7
B:3
B:4
C:2
D:1
C:2
D:1
C:3
D:1
D:1
E:1
D:1
D:1
E:1
E:1

Q3

a. *Newton's method calculate each parameter using the equation:*

$$\theta_{new} = \theta_{old} - H^{-1}\nabla L(\theta_{old})$$

where the Hessian matrix is:

$$H_{ij} = \frac{\partial^2 L(\theta)}{\partial\theta_i \partial\theta_j}$$

The Hessian matrix has dimensions N^2, N is the number of parameters in the training model. As GPT-4 is rumored to have trillion parameters. N here is really large that will make the cost of computing Hessian matrix becomes extremely high.
So the main computational challenge in training such a model using Newton's method is that it will cause really high memory space and the extremely large cost of computing.

b. *For Adagrad, the update equation is*

$$\theta_{t+1,i} \leftarrow \theta_{t,i} - \eta_{t,i} g_{t,i}$$

where $g_{t,i} = \nabla L(\theta_t)$ *and* $\eta_{t,i} = \frac{\eta}{\sqrt{\Sigma_{\tau=1}^{t} g_{\tau,i}^2}}$
The η *is the learning rate. And the for conclusion, the parameters that whose historical accumulative gradients is smaller will receive larger learning rate, and on the contrary, parameters that whose historical accumulative gradients is larger will receive smaller learning rate.*
This is because that Adagrad algorithm will update each parameter by the historical gradient information. Furthermore, in denominator, there is the sum of the squares of the historical gradients for the i-th parameter. So the larger historical gradients, the smaller learning rate, the smaller historical gradients, the larger learning rate.

c. *No I will disagree with Professor ScatterBrain.*
Because although we need historical gradients. But we could get it by adding the square of the present gradient to the sum of each iteration. Or we can say that we only need to store the accumulative sum value of the historical gradients. And this means that we don't need to store all the historical gradients for each iteration. So the space we need to store the gradients is not such large. So I disagree with Professor ScatterBrain.

d. *To find the update equation of Adam algorithm, we can use below equations:*
For first moment estimate, we could calculate by:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

For second moment estimate, we could calculate by:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

And Compute bias corrected versions of first and second moment by:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Then the final update equation is:

$$\theta_{t+1} \leftarrow \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$$

All above is the method that Adam try to update the parameter.
And for the primary concern regarding Adagrad algorithm, it is that Adagrad remembers $g_{\tau,i}$ *for all* τ *in the history, which makes all historical* τ *get the same weight. And as the times of iteration get larger, the cumulative gradient will get larger and larger which makes the learning rate get really small in the end. And this may lead to the model terminate to learn when the model does not converge.*
And for the method: Adam tries to solve this by calculating the first moment estimate and second moment estimate. And also the bias corrected versions of first and second moment. With this method, The Adam optimization algorithm overcomes the limitations of Adagrad. It adjusts the update step for each parameter using the square root of the second-order moment estimate and the first-order moment estimate, which leverages the benefits of momentum for accelerating gradient descent while preventing the learning rate from diminishing too rapidly. And for bias corrected versions, it ensures the algorithm starts off correctly. These features that Adam will not have a really small learning rate like in Adagrad, leading to better performance in deep learning, especially in handling non-convex optimization problems.

Q4

a. *The computational complexity of k-medoids is $O(k(n-k)^2)$ For PAM method, first that we need to randomly select k medoids, and then we need to consider that for each of the chosen medoids, it need to consider all the other unchosen medoids, which the computational complexity is $O(k(n-k))$. But we need to consider the worst case, for each iteration, if we choose the worst medoid in all possibilities. So we need to find iterate all the remaining $O(n-k)$ points to find the best set of medoids. Then the final computational complexity is $O(k(n-k)*(n-k))$=$O(k(n-k)^2)$*

b. *In general, the computational complexity of k-medians algorithm is larger than that of k-means. And the reason is that we need to sort the whole data to find the median of the set while we don't need to do the same thing we want to find the mean value.*
For each iteration in k-means, we need to only loop through the dataset and the computational complexity is $O(n)$. And for each iteration in k-medians, we need to only loop through the dataset and sort the dataset. The computational complexity of loop is $O(n)$, and the total computational complexity is $O(nlog(n))$. And this is the best situation when we used binary tree to sort the dataset. So the computational complexity of k-medians algorithm is larger than that of k-means.

c. *First we need to marge a and b:*

	a,b	c	d	e
a,b	0			
c	6.1	0		
d	8.6	6.1	0	
e	7.3	7.3	3.1	0

Second then we need to merge d and e:

	a,b	c	d,e
a,b	0		
c	6.1	0	
d,e	8.6	7.3	0

Third then we need to merge a, b and c

	a,b,c	d,e
a,b,c	0	
d,e	8.6	0

And finally, we need to merge a, b, c and d, e

	a,b,c,d,e
a,b,c,d,e	0