CS412 Final

Qi Long

qilong2

1

1.1 (a)

Absolute support is number of transactions that a sequential pattern X occur in. Listing all Length-1 sequential patterns and supports in a table:

Length-1 sequential patterns	Support
a	3
b	5
c	4
d	3
e	3
f	2
g	1
h	1

Filter by $min\_sup = 2$, list length-1 frequent sequential patterns in SDB_1 in table:

Length-1 sequential patterns	Support
a	3
b	5
c	4
d	3
e	3
f	2

1.2 (b)

The projected database for prefix $< bb >$:

$< bb >$-projected DB
$< (\_d) >$
$< (ce) >$
$< f >$
$< cb(ade) >$

From the table above, $sup(< bb >) = 4 > 2 = min\_sup$, so $< bb >$ is a length-2 frequent pattern in SDB_1.

1.3 (c)

The projected database for prefix $< (bd) >$:

$< (bd) >$-projected DB
$<>$
$< bcb(ade) >$

From the table above, $sup(< (bd) >) = 2 = min\_sup$, so $< (bd) >$ is a length-2 frequent pattern in SDB_1.

1.4 (d)

The projected database for prefix $< b >$

$< b >$-projected DB
$< (bd) >$
$< (\_f)(fg)b(ce) >$
$< (\_f)(ah)abf >$
$< (\_e) >$
$< (\_d)bcb(ade) >$

1.5 (e)

Based on (d), projected databases:

- $< ba >$: $< (\_h)abf >$, $< (\_de) >$
  frequent: none, stop.

- $< bb >$: $< (\_d) >$, $< (ce) >$, $< f >$, $< cb(ade) >$
  frequent: $< bbc >$, $< bbe >$
  - $< bbc >$: $< (\_e) >$, $< b(ade) >$, stop.
  - $< bbe >$: $< (\_c) >$, $< (\_ad) >$, stop.
- $< bc >$: $< (\_e) >$, $< b(ade) >$
  frequent: none, stop.
- $< bd >$: $< (\_b) >$, $< (\_ae) >$
  frequent: none, stop.
- $< be >$: $< (\_c) >$, $< (\_ad) >$
  frequent: none, stop.
- $< bf >$: $< (\_g)b(ce) >$, $<>$
  frequent: none, stop.
- $< (ba) >$: not frequent, stop.
- $< (bc) >$: not frequent, stop.
- $< (bd) >$: $<>$, $< bcb(ade) >$
  frequent: none, stop.
- $< (be) >$: $<>$, not frequent, stop.
- $< (bf) >$: $< (fg)b(ce) >$, $< (ah)abf >$
  frequent: $< (bf)b >$, $< (bf)f >$
  - $< (bf)b >$: $< (ce) >$, $< f >$, stop.
  - $< (bf)f >$: $< (\_g)b(ce) >$, $<>$, stop.

Therefore, all frequent sub-sequences starting with b includes:

$$< b >, < ba >, < bb >, < bbc >, < bbe >, < bc >, < bd >, < (bd) >,$$

$$< be >, < bf >, < (bf) >, < (bf)b >, < (bf)f >$$

2

2.1 (a)

From Table 2,

$$rel\_sup(A) = \frac{7}{10} = 0.7$$
$$rel\_sup(B) = \frac{7}{10} = 0.7$$

For $X \to Y$, by equation:

$$s = \frac{abs\_sup\{X, Y\}}{Total\ Transactions} \tag{1}$$
$$c = \frac{rel\_sup\{X, Y\}}{rel\_sup(X)} \tag{2}$$

$$s = \frac{abs\_sup\{A, B\}}{Total\ Transactions} = \frac{4}{10} = 0.4$$
$$c = \frac{sup\{A, B\}}{sup(A)} = \frac{0.4}{0.7} = 0.57143$$

2.2 (b)

When generating candidates, self-joining is used. It follows the pseudo code

```
result = [ ]
for each $p$ in $F_k$:
    for each $q$ in $F_k$:
        if $p.item_1 = q.item_1, p.item_2 = q.item_2, ..., p.item_k < q.item_k$:
            result.append(join(p, q))
```

2.2.1 Level-1

First scan C_1:

Length-1 itemsets	Support
A	7
B	7
C	7
D	6
E	3

First Filter F_1:
All length-1 itemsets have $support > min_sup$, so same as C_1
Generate Candidates:
Self-joining $F_1 * F_1$ according to 2.2, get

$$AB, AC, AD, AE, BC, BD, BE, CD, CE, DE$$

2.2.2 Level-2

Second scan C_2:

Length-2 itemsets	Support
AB	4
AC	5
AD	4
AE	3
BC	5
BD	4
BE	2
CD	3
CE	2
DE	2

Second Filter F_2:

Length-2 itemsets	Support
AB	4
AC	5
AD	4
AE	3
BC	5
BD	4
CD	3

Generate Candidates:
Self-joining $F_2 * F_2$ according to 2.2, get

$$ABC, ABD, ABE, ACD, ACE, ADE, BCD$$

Pruning:
Remove ABE, since BE is not frequent.
Remove ACE, since CE is not frequent.
Remove ADE, since DE is not frequent.

2.2.3 Level-3

Third scan C_3:

Length-3 itemsets	Support
ABC	3
ABD	2
ACD	2
BCD	2

Third Filter F_3:

Length-3 itemsets	Support
ABC	3

Stop.
In conclusion, all frequent itemsets include:

$$A, B, C, D, E, AB, AC, AD, AE, BC, BD, CD, ABC$$

2.3 (c)

Scan DB once, sort single item frequent pattern in frequency descending order:

$$F - list = A - B - C - D - E$$

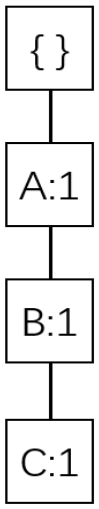
FP Trees see next page.

3

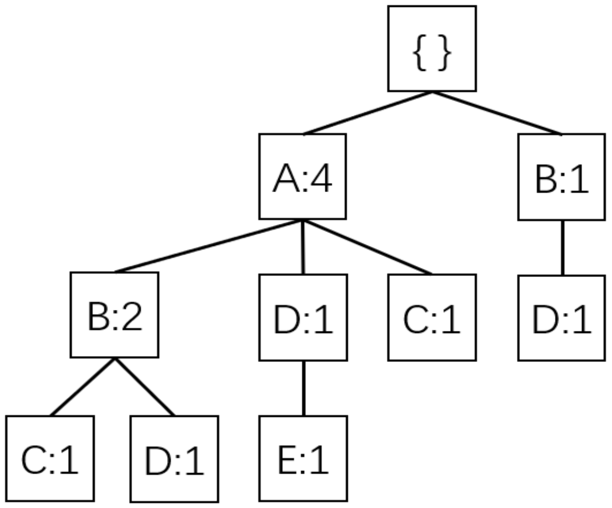
3.1 (a)

Newton's method takes the equation:

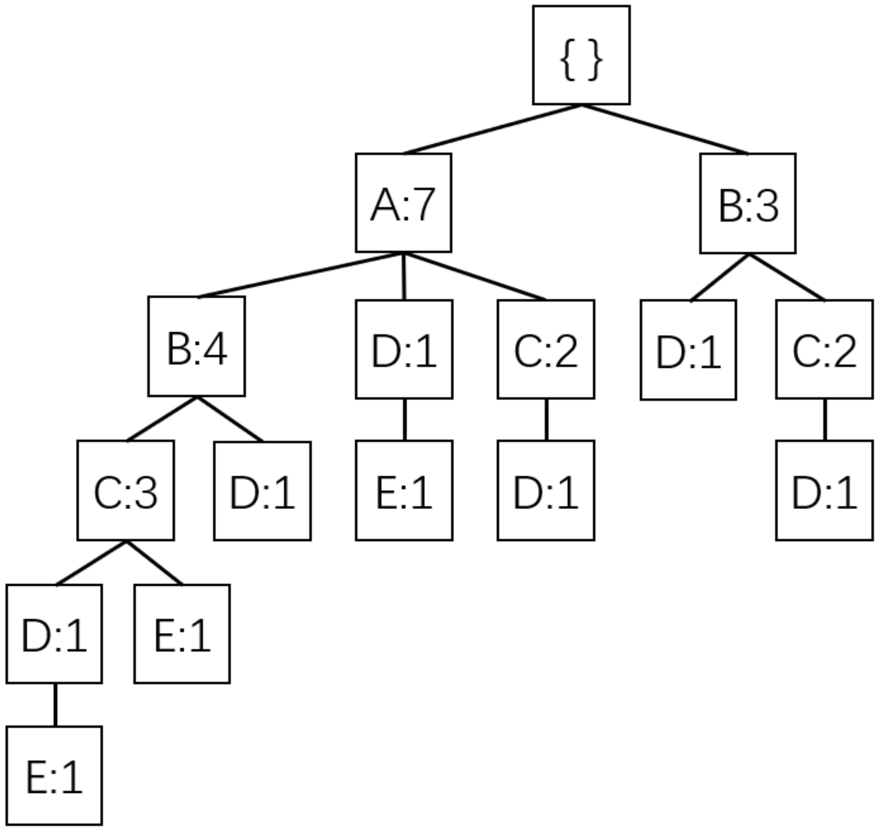
$$\theta_{t+1} \leftarrow \theta_t - [\nabla^2 L(\theta_t)]^{-1} \nabla L(\theta_t) \tag{3}$$



(a) After inserting T_1



(b) After inserting T_5



(c) After inserting T_{10}

In this equation, the gradient is a vector of length of the number of parameters and so the Hessian ($\nabla^2 L(\theta_t)$) will be a matrix with two dimensions both having length of the number of parameters. Therefore, for models with trillions of parameters, the gradient term is extremely large in size, which is extremely hard to form and take the inverse of. In addition, Hessian may not be positive definite, causing further computational issues.

3.2 (b)

Adaptive gradient descent (Adagrad) takes the form

$$\theta_{t+1,i} \leftarrow \theta_{t,i} - \eta_{t,i} g_{t,i} \tag{4}$$

$$\eta_{t,i} = \frac{\eta}{\sqrt{\Sigma_{\tau=1}^{t} g_{\tau,i}^2}} \tag{5}$$

The parameter update for Adagrad algorithm is by subtracting the gradient of the loss function from the respective parameter values at the previous iteration with a dynamic learning rate ($\eta_{t,i}$). Besides, $\eta_{t,i}$ is related to the basic learning rate and inversely proportional to sum of gradient descent of that specific parameter throughout out time. It is calculated separately at each iteration. From the equations, $\eta_{t,i}$ changes with t and i, so at different epochs and for different parameters, the learning rate $\eta_{t,i}$ varies, so the change in parameters varies.
"Relevant" parameters means larger influence on the optimization, thereby larger gradient. Since the learning rate $\eta_{t,i}$ is inversely proportional to sum of gradients of corresponding parameter throughout time, they result in smaller learning rate.
"Irrelevant" parameters means smaller influence on the optimization, thereby smaller gradient. Since the learning rate $\eta_{t,i}$ is inversely proportional to sum of gradients of corresponding parameter throughout time, they result in larger learning rate.

3.3 (c)

I disagree with Professor ScatterBrain. From the equation (5), at each iteration, the squared gradient is added to the sum of the squared of gradient in the past, which means we only care about the cumulative value ($\sqrt{\Sigma_{\tau=1}^{t} g_{\tau,i}^2}$)

instead of every individual gradient. So not "all past gradients" have to be stored, instead, only store and update the cumulative value ($\sqrt{\Sigma_{\tau=1}^{t} g_{\tau,i}^2}$) is enough. A large number of steps of the algorithm lead to larger storage demands to some extent because of larger cumulative value, but not because storing all steps' gradient separately.

3.4 (d)

Adam makes use of two moments when updating the parameters. The two moments take the form:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{6}$$

where first moment estimate discount the past with β_1.

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{7}$$

where second moment estimate discount the past with β_2.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{8}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{9}$$

After computing bias corrected versions of first and second moment, the parameter is updated using two moments and combined with two constants, by formula:

$$\theta_{t+1} \leftarrow \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t \tag{10}$$

According to the equation (5), Adagrad algorithm has the primary concern that the step sizes of all parameters keep getting smaller with more iterations, since the denominator is a cumulative summation of gradient with equal importance overtime. If the step size keeps decreasing, it will take many iterations to converge if it decrease too fast and will hardly improve as the t goes large and step size goes to zero.
Adam attempt to fix it by "forgetting the past". According to what is discussed above, compared with (4), adam makes step size able to increase or decrease with time by using two moments that forget some portions of the previous values of themselves and aggregate them with the new gradient (g_t). So the information from the past iterations keeps vanishing gradually and the step size does not keep getting smaller.

4

4.1 (a)

The PAM algorithm takes the step:

```
Select initial K medoids randomly
Repeat
    Object re-assignment
    Swap medoid m with another if it improves the clustering quality
Until convergence criterion is satisfied
```

Analyze the steps above:

- It takes $(n - K)$ times of computation to compute distance of non-medoid points to a medoid.
- The object re-assignment needs to compute the distance from each non-medoid data point to all K medoids, so $K(n - K)$ computational time for each object re-assignment.
- The algorithm needs to repeat swapping medoid to candidate points and thereby repeat object re-assignment and calculating SSE. Except medoids, all other points can be candidate points, so $(n-K)$ candidate points need to repeat distance calculation $(n - K)$ times..

Therefore, $O(K(n - K)^2)$ computational complexity is needed.

4.2 (b)

Yes. The k-medians algorithm is more computationally demanding.
The steps of the two algorithms are the same, the only different part is reassigning the center point of each cluster by calculating the mean of values of all points or by finding the median of all data points.
Calculating mean of a dataset can be done in $O(n)$, but finding median of a dataset requires sorting the data points first, which needs $O(nlogn)$ using comparison-based sorting algorithm.

4.3 (c)

Use complete link method:

$$Distance((a,b),c) = max(Distance(a,c), Distance(b,c)) \tag{11}$$

4.3.1 (Step 1)

ab has the smallest distance 2.1, so cluster a, b should be merged.
Update distance by formula 11:

$$Distance((a,b),c) = max(Distance(a,c), Distance(b,c)) = 6.1$$

$$Distance((a,b),d) = max(Distance(a,d), Distance(b,d)) = 8.6$$

$$Distance((a,b),e) = max(Distance(a,e), Distance(b,e)) = 7.3$$

	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

4.3.2 (Step 2)

de has the smallest distance 3.1, so cluster d, e should be merged.
Update distance by formula 11:

$$Distance((d,e),(a,b)) = max(Distance(d,(a,b)), Distance(e,(a,b))) = 8.6$$

$$Distance((d,e),c) = max(Distance(d,c), Distance(e,c)) = 7.3$$

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

4.3.3 (Step 3)

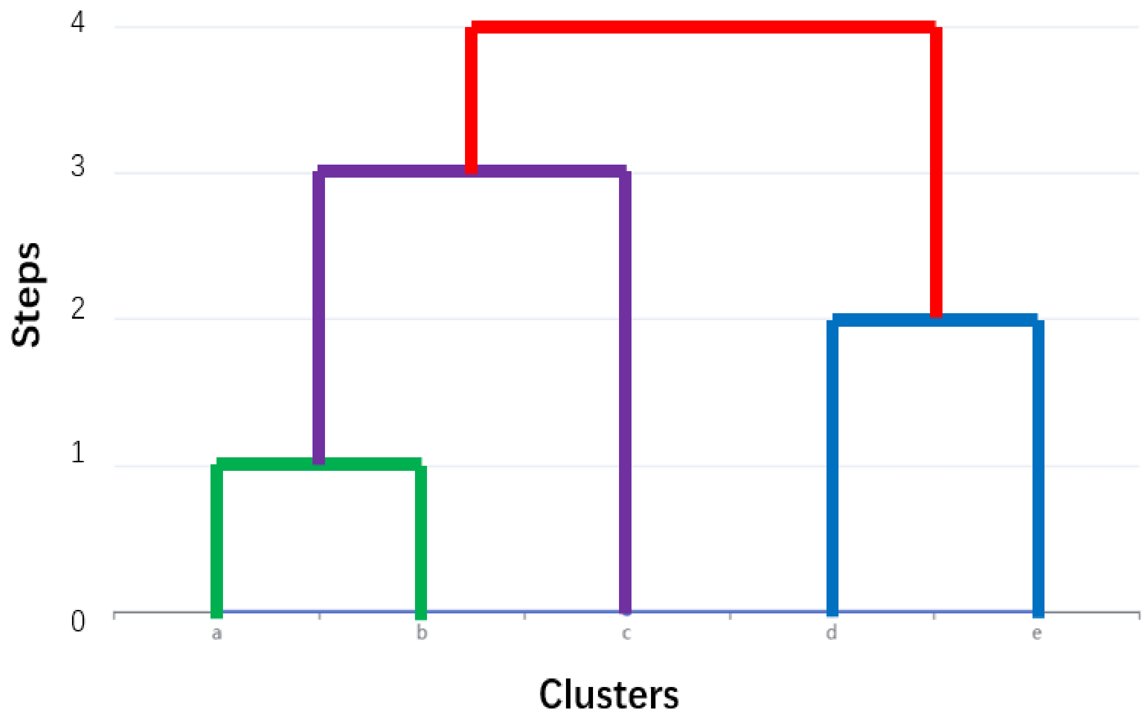
abc has the smallest distance 6.1, so cluster a, b, c should be merged.
Update distance by formula 11:

$$Distance((a,b,c),(d,e)) = max(Distance((a,b),(d,e)), Distance(c,(d,e))) = 8.6$$

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

4.3.4 (Step 4)

Merge a, b, c, d, e and stop.



(a) Agglomerative Clustering Result