Question 1

a. $(2', L1)$ List all candidate 1-itemsets (C1). What will you do to get rid of non-frequent 1-itemsets (Choose one from pruning, self-joining and db-scanning)? List all frequent 1-itemsets (L1).

C1

Item Set	Sup
{a}	1
{b}	2
{d}	2
{f}	5
{g}	3
{h}	3
{j}	2
{k}	1
{l}	2
{m}	4
{n}	1

By DB-Scanning

L1

Item Set	Sup
{f}	5
{g}	3
{h}	3
{m}	4

b. $(2'$,L1) What will you do to generate all candidate 2-itemsets (C2) (Choose one from pruning, self-joining and db-scanning)? List all itemsets in C2.

By Self-Joinning

C2

Item Set	Sup
{f, g}	3
{g, h}	1
{h, m}	3
{m, f}	4
{f, h}	3
{g, m}	2

c. (1′, L1) Take the same action you choose for computing L1 from C1, and list all frequent 2-itemsets (L2) computed from C2.

L2

Item Set	Sup
{f, g}	3
{h, m}	3
{m, f}	4
{f, h}	3

d. (3′, L1) To generate all candidate 3-itemsets (C3) from L2, what is the extra action you need to consider besides the one you take from L1 to C2 in sub-question b (Choose one from pruning, self-joining and db-scanning)? List all itemsets in C3. Are they all frequent?

Item Set	Sup
{f, g, h}	1
{f, g, m}	2
{f, m, h}	3

{f, m, h} is frequent. The other ones are not.

Pruning

e. ($2'$, L2) Is there any frequent 4-itemset? Why?

No, because there are only 1 3-pattern itemset in total.

f. ($2'$,L3) According to your simulation, what part of Apriori involves the heaviest com- putation? Under what circumstances will this be extremely bad? Hints: For the first question, you can still choose from the three steps hinted before and explain. For the second question, think about the impact of min sup.

I think Self-joining would be the costliest. If the number of different item is big, and the min-sup is really small, the number of item set computed would be close to the factorial of n, which is really bad.

Question 2

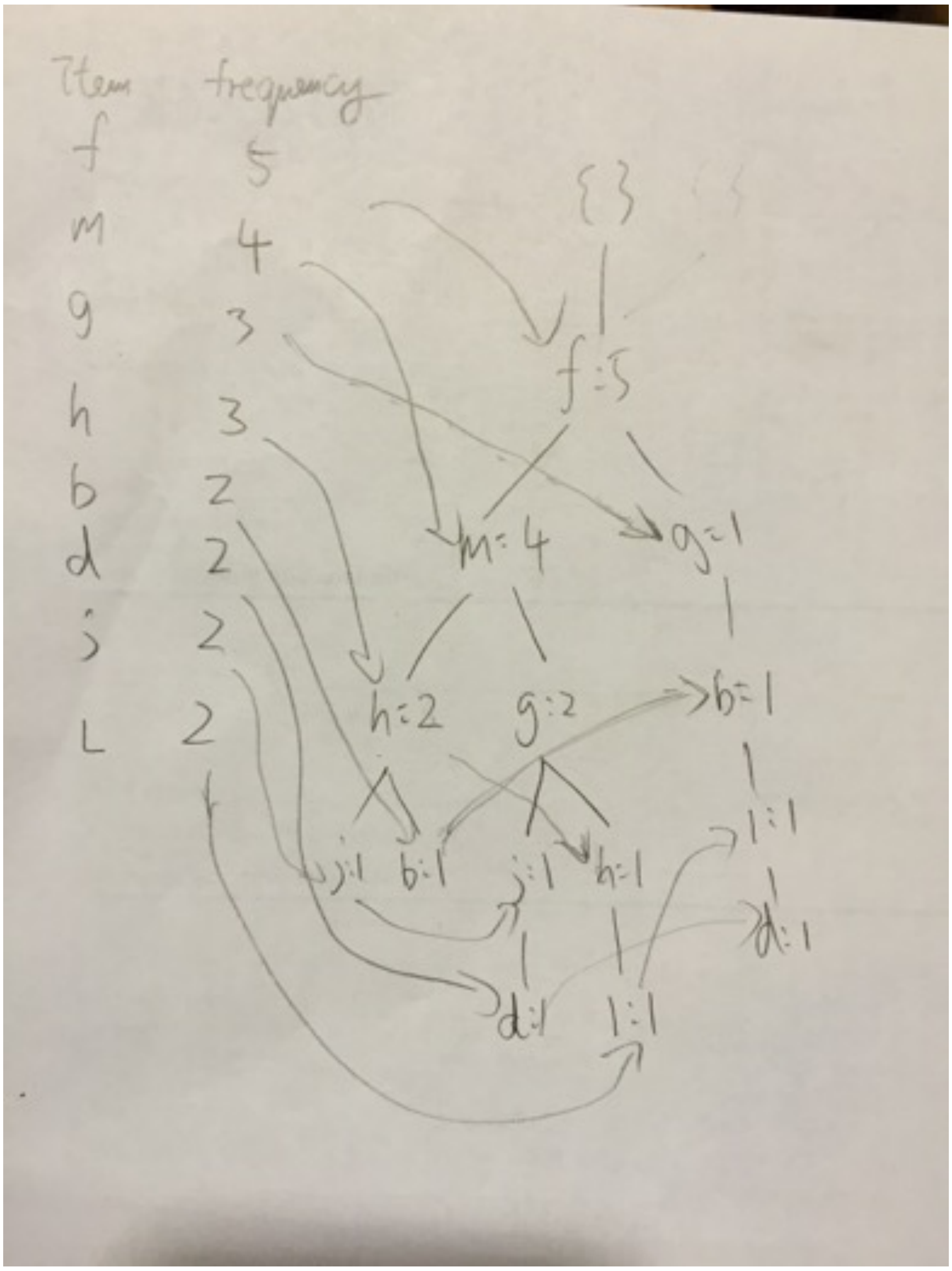
a. ($2'$,L2) Generate an ordered list off recent items based on the raw transaction database. Hints: Reorder items within each transaction according to their frequencies in the whole database.

Trans	Items
1	f, g, b, l, d
2	f, m, g, h, l
3	f, m, h, b
4	f, m, h, j
5	f, m, g, j, d

b. ($5'$, L2) Generate Header Table and FP-tree based on the frequent item list. Link nodes to the corresponding positions in the Header Table.

Item Set	Sup
{b}	2

{d}	2
{f}	5
{g}	3
{h}	3
{j}	2
{l}	2
{m}	4



c. (4′,L2)Generate Conditional Pattern Bases and Conditional FP-trees for items m,h,b,j based on the FP-tree, and list the frequent patterns computed based on each of the Conditional FP-trees. (You only need to show the Conditional Pattern Bases and the frequent patterns.)

Item Set	Sup
{f}	5
{m}	4
{g}	3
{h}	3
{b}	2
{d}	2
{j}	2
{l}	2

d. (2′,L3) Why do we order the items in each transaction by their frequency before constructing the FP-tree? Hints: Think about the purpose of FP-tree and how this order will affect its structure.

The most frequent element will be on the top of tree. If we sort the tree by frequency in advance, it would be clearer and easier for us to build the tree.

Question 3

a. (3′,L1) List all closed patterns among the frequent patterns computed in Question 1.

(h, m), (f, h, m), (f, g), (f, m)

b. (3′,L1) List all maximal patterns among the frequent patterns computed in Question 1.

(f, g), (f, h, m)

c. ($2'$, L2) List at least 2 association rules with min conf = 0.6 from the frequent patterns computed in Question 1.

f→g = (60%, 60%)

m→f = (80%, 100%)

d. ($2'$,L3) Under what circumstances should we prefer maximal patterns than closed patterns? What about the other way around? Hints: Think about the differences between the two special frequent patterns.

We would prefer Closed Pattern if we wish to investigate into the information of frequent itemsets and and figure out the specific pair-items. When we only need to find the frequent pattern but don't need to know the counting details, Max Pattern would be better.

Question 4

a. ($3'$,L3) A max-pattern must be a closed pattern. If yes, briefly explain your idea; otherwise, give a counter example.

True. The support would either decrease or be the same if an element is added to the set. Therefore, if max-pattern is always a subset of close pattern.

b. ($3'$, L2) At each step of Apriori, we will generate all non-repeated (k+1)-itemsets joined by each pair of k-itemsets that agree on (k − 1) items, and then test the frequency of those (k + 1)-itemsets on the transaction database to remove all infrequent ones.

False, it doesn't necessarily have k-1 items.

c. ($3'$,L2) For FP-Growth, in order to mine all frequent patterns, we have to recursively generate conditional frequent bases and conditional FP-trees until the FP-tree generated has only one node.

False. We keep searching until the tree has one or no path left.

d. ($3'$,L1) CLOSET is based on Apriori, while MaxMiner is based on FP-Growth.

False. Instead, CLOSET is based on FP-Growth.

e. ($3'$, L3) To measure the correlations between frequent items, Lift is always better than Confidence (as defined in association rule). Use examples to analyze.

False. Lift sometimes has a more accurate evaluation than confidence, in the but not always.

X, Y are two items in the list of items. When X has 2, Y has 1000, the lift would be very big. Their value would be overestimate.

Machine Problem

Brief Explanation:

For Step one, I first parse the topic files as an array of set, a set for each line. (def parseText(text)) Then I would count how much time each element exist, and put them into a dictionary. (def doFirst(rawData, frequentPattern)) . Afterward, I would keep doing the DB scanning, Self joining and Pruning until no element is left. (doTheRest(data, num, frequentPattern))

For Step two, I use one function to compute the Close Pattern and another one to compute the Max Pattern.

Question to ponder A: How do you choose min sup for this task? Explain your criteria; any reasonable choice will be good.

I set the min_sup as 100. After trying different value range from 50 to 300, 100 seems to be the perfect value. With 100, it generated a reasonable numbers of frequent patterns that are valuable. With min_sup set to a over-small value, the result list would contain many insignificant value and the computing time would be long. With min_sup set to a over-big value, the result list would be too short and not enough to reveal the association rules.

Question to ponder B: Can you figure out which topic corresponds to which domain based on patterns you mine? Write down your observations.

Topic-0.txt: data mining

Topic-1.txt: machine learning

Topic-2.txt: information retrieval

Topic-3.txt: database

Topic-4.txt: theory

Question to ponder C: Compare the results of frequent patterns, closed patterns and maximal patterns, is there any difference? If so, what kind of patterns give more satisfying results? Write down your analysis.

The closed patterns and frequents patterns are mostly the same, since the number of support varies and the chance that two pattern has the exact same support number is low. The max patterns are different. Most of the single words were filtered. It helps filter out the combined words and is more helpful for determine the topic.

Question to ponder D: What are the differences between phrases which satisfy only the min sup criterion and phrases (association rules) which satisfy both min sup and min conf criteria? Compare the results of Step 1 and Step 3 and write down your observations.

In Step 3, the confidence of the data is considered in the calculation. The result is slightly more accurate than the calculation without the confidence, since confidence considers the relationship between the data. For example, when A and B have a great chance of appearing together, confidence will bring this into account.

Topic-0:

```
1. [series=1]: 209 ==> [time=1]: 194   <conf:(0.93)> lift:(16.65) lev:(0.02)
conv:(12.33)

 2. [association=1]: 336 ==> [rule=1]: 233   <conf:(0.69)> lift:(16.75) lev:
(0.02) conv:(3.1)

 3. [rule=1]: 416 ==> [association=1]: 233   <conf:(0.56)> lift:(16.75) lev:
(0.02) conv:(2.19)

 4. [association=1]: 336 ==> [mining=1]: 159   <conf:(0.47)> lift:(4.09)
lev:(0.01) conv:(1.67)

 5. [pattern=1]: 528 ==> [mining=1]: 203   <conf:(0.38)> lift:(3.32) lev:
(0.01) conv:(1.43)

 6. [rule=1]: 416 ==> [mining=1]: 159   <conf:(0.38)> lift:(3.3) lev:(0.01)
conv:(1.43)

 7. [mining=1]: 1163 ==> [data=1]: 413   <conf:(0.36)> lift:(2.77) lev:
(0.03) conv:(1.35)

 8. [time=1]: 560 ==> [series=1]: 194   <conf:(0.35)> lift:(16.65) lev:
(0.02) conv:(1.49)

 9. [data=1]: 1288 ==> [mining=1]: 413   <conf:(0.32)> lift:(2.77) lev:
(0.03) conv:(1.3)

10. [mining=1]: 1163 ==> [pattern=1]: 203   <conf:(0.17)> lift:(3.32) lev:
(0.01) conv:(1.15)
```

Topic-1:

1. [machine=1, support=1]: 117 ==> [vector=1]: 115   <conf:(0.98)> lift:(42.26) lev:(0.01) conv:(38.09)

2. [machine=1, vector=1]: 123 ==> [support=1]: 115   <conf:(0.93)> lift:(41.68) lev:(0.01) conv:(13.36)

3. [neighbor=1]: 137 ==> [nearest=1]: 121   <conf:(0.88)> lift:(60.6) lev:(0.01) conv:(7.94)

4. [nearest=1]: 141 ==> [neighbor=1]: 121   <conf:(0.86)> lift:(60.6) lev:(0.01) conv:(6.62)

5. [neural=1]: 128 ==> [network=1]: 101   <conf:(0.79)> lift:(16.49) lev:(0.01) conv:(4.35)

6. [vector=1, support=1]: 146 ==> [machine=1]: 115   <conf:(0.79)> lift:(24.66) lev:(0.01) conv:(4.42)

7. [support=1]: 217 ==> [vector=1]: 146   <conf:(0.67)> lift:(28.93) lev:(0.01) conv:(2.94)

8. [vector=1]: 225 ==> [support=1]: 146   <conf:(0.65)> lift:(28.93) lev:(0.01) conv:(2.75)

9. [semi=1]: 165 ==> [supervised=1]: 105   <conf:(0.64)> lift:(34.98) lev:(0.01) conv:(2.66)

10. [supervised=1]: 176 ==> [semi=1]: 105   <conf:(0.6)> lift:(34.98) lev:(0.01) conv:(2.4)

Topic-2:

1. [page=1]: 131 ==> [web=1]: 107   <conf:(0.82)> lift:(6.63) lev:(0.01) conv:(4.59)

2. [natural=1]: 228 ==> [language=1]: 170   <conf:(0.75)> lift:(15.15) lev:(0.02) conv:(3.67)

3. [engine=1]: 181 ==> [search=1]: 122   <conf:(0.67)> lift:(9.49) lev:(0.01) conv:(2.8)

```
 4. [service=1]: 267 ==> [web=1]: 117   <conf:(0.44)> lift:(3.56) lev:(0.01)
conv:(1.55)

 5. [retrieval=1]: 1114 ==> [information=1]: 475   <conf:(0.43)> lift:(3.51)
lev:(0.03) conv:(1.53)

 6. [information=1]: 1211 ==> [retrieval=1]: 475   <conf:(0.39)> lift:(3.51)
lev:(0.03) conv:(1.46)

 7. [language=1]: 490 ==> [natural=1]: 170   <conf:(0.35)> lift:(15.15) lev:
(0.02) conv:(1.49)

 8. [semantic=1]: 414 ==> [web=1]: 104   <conf:(0.25)> lift:(2.04) lev:
(0.01) conv:(1.17)

 9. [document=1]: 564 ==> [retrieval=1]: 141   <conf:(0.25)> lift:(2.23)
lev:(0.01) conv:(1.18)

10. [search=1]: 707 ==> [web=1]: 173   <conf:(0.24)> lift:(1.99) lev:(0.01)
conv:(1.16)


Topic-3:

 1. [oriented=1]: 163 ==> [object=1]: 102   <conf:(0.63)> lift:(28.51) lev:
(0.01) conv:(2.57)

 2. [reinforcement=1]: 224 ==> [learning=1]: 117   <conf:(0.52)> lift:
(9.51) lev:(0.01) conv:(1.96)

 3. [discovery=1]: 202 ==> [knowledge=1]: 104   <conf:(0.51)> lift:(7.04)
lev:(0.01) conv:(1.89)

 4. [object=1]: 223 ==> [oriented=1]: 102   <conf:(0.46)> lift:(28.51) lev:
(0.01) conv:(1.8)

 5. [base=1]: 337 ==> [data=1]: 138   <conf:(0.41)> lift:(8.1) lev:(0.01)
conv:(1.6)

 6. [relational=1]: 375 ==> [database=1]: 129   <conf:(0.34)> lift:(3.25)
lev:(0.01) conv:(1.36)
```

```
 7. [base=1]: 337 ==> [knowledge=1]: 111   <conf:(0.33)> lift:(4.5) lev:
(0.01) conv:(1.38)

 8. [data=1]: 514 ==> [base=1]: 138   <conf:(0.27)> lift:(8.1) lev:(0.01)
conv:(1.32)

 9. [system=1]: 928 ==> [database=1]: 195   <conf:(0.21)> lift:(1.99) lev:
(0.01) conv:(1.13)

10. [learning=1]: 558 ==> [reinforcement=1]: 117   <conf:(0.21)> lift:
(9.51) lev:(0.01) conv:(1.23)
```

Topic-4:

```
 1. [concurrency=1]: 133 ==> [control=1]: 107   <conf:(0.8)> lift:(20.73)
lev:(0.01) conv:(4.73)

 2. [oriented=1]: 183 ==> [object=1]: 141   <conf:(0.77)> lift:(14.37) lev:
(0.01) conv:(4.03)

 3. [real=1]: 260 ==> [time=1]: 151   <conf:(0.58)> lift:(19.38) lev:(0.01)
conv:(2.29)

 4. [stream=1]: 212 ==> [data=1]: 112   <conf:(0.53)> lift:(5) lev:(0.01)
conv:(1.88)

 5. [time=1]: 295 ==> [real=1]: 151   <conf:(0.51)> lift:(19.38) lev:(0.01)
conv:(1.98)

 6. [processing=1]: 637 ==> [query=1]: 313   <conf:(0.49)> lift:(2.82) lev:
(0.02) conv:(1.62)

 7. [optimization=1]: 380 ==> [query=1]: 173   <conf:(0.46)> lift:(2.62)
lev:(0.01) conv:(1.51)

 8. [system=1]: 767 ==> [database=1]: 276   <conf:(0.36)> lift:(3.05) lev:
(0.02) conv:(1.37)
```

```
 9. [object=1]: 528 ==> [database=1]: 154   <conf:(0.29)> lift:(2.47) lev:
(0.01) conv:(1.24)

10. [control=1]: 382 ==> [concurrency=1]: 107   <conf:(0.28)> lift:(20.73)
lev:(0.01) conv:(1.37)
```