CS 422-DATA MINING
HOMEWORK 4
TA SOLUTION

1.

Customer ID	Transaction ID	Items Bought
1	0001	$\{a, d, e\}$
1	0024	$\{a, b, c, e\}$
2	0012	$\{a, b, d, e\}$
2	0031	$\{a, c, d, e\}$
3	0015	$\{b, c, e\}$
3	0022	$\{b, d, e\}$
4	0029	$\{c, d\}$
4	0040	$\{a, b, c\}$
5	0033	$\{a, d, e\}$
5	0038	$\{a, b, e\}$

a) Consider the market basket transactions shown in table to the right. Compute the support for itemsets {e}, {b,d}, and {b,d,e} by treating each transaction ID as a market basket.
b) Compute the confidence for the association rules {b,d} –→ {e} and {e} –→ {b,d}. Explain each step in your calculation and show all calculations.
c) Is confidence a symmetric measure?

Ans.:

1.
a) Support count of an itemset is defined by number of times the itemset appears in the list of transactions.

→ Support $\sigma(X)$ for an itemset X is defined by

$$\sigma(X) = |\{t_i \mid X \subseteq t_i,\ t_i \in T\}|;$$ t_i = Transaction i, T = Set of all transactions

$$\text{Support}(\{e\}) = \frac{\text{No. of times itemset } \{e\} \text{ appears in transactions}}{\text{Total no. of transactions}}$$

$$= \frac{8}{10}$$

$\therefore \boxed{\text{Support}(\{e\}) = 0.8}$

$\text{Support}(\{b,d\}) = \frac{2}{10}$

$\therefore \boxed{\text{Support}(\{b,d\}) = 0.2}$

$\text{Support}(\{b,d,e\}) = \frac{2}{10}$

$\therefore \boxed{\text{Support}(\{b,d,e\}) = 0.2}$

(b) In $X \to Y$ association rule, confidence determines how frequently items in Y appear in transactions that contain X.

Confidence $c(X \to Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$

$$c(bd \to e) = \frac{0.2}{0.2}$$
$$= 1$$
$$= 100\%$$

$$c(e \to bd) = \frac{0.2}{0.8}$$
$$= 0.25$$
$$= 25\%$$

(C) No, confidence is not a symmetric measure as we can see that $c(bd \to e) \neq c(e \to bd)$.
Confidence measures the reliability of the inference made by a rule. For a given rule $X \to Y$, the higher the confidence, the more likely it is for Y to be present in transactions that contain X. Confidence does not preserve the values under inversion of the rule. Confidence for $A \to B$ and $B \to A$ may or may not be the same.

2.

Transaction ID	Items Bought
1	{Milk, Beer, Diapers}
2	{Bread, Butter, Milk}
3	{Milk, Diapers, Cookies}
4	{Bread, Butter, Cookies}
5	{Beer, Cookies, Diapers}
6	{Milk, Diapers, Bread, Butter}
7	{Bread, Butter, Diapers}
8	{Beer, Diapers}
9	{Milk, Diapers, Bread, Butter}
10	{Beer, Cookies}

a) Consider the market basket transactions shown in table to the right. What 2-itemset has the largest support? Show how you calculate the value of the support for that itemset.
b) For the itemset from (a), it is a 2-itemset and contains 2 items, *a* and *b.* Calculate the confidence for the rules {*a*} −→ {*b*} and {*b*} −→ {*a*}. What can you say about the confidence for this rule? How do you explain this results from the data and also from your conclusion in exercise 1?

Ans.:

2.
(a) We can see there total 6 items in the table. We have to find largest support of 2-itemset. Thus, we will consider each item and check with remaining items in 2-itemset and count the largest support as shown below.

$s(\{Milk, Beer\}) = \sigma(Milk \cup Beer)/\sigma(N) = 1/10 = 0.1$

$s(\{Milk, Butter\}) = \sigma(Milk \cup Butter)/\sigma(N) = 3/10 = 0.3$

$s(\{Milk, Diapers\}) = \sigma(Milk \cup Diapers)/\sigma(N) = 4/10 = 0.4$

$s(\{Milk, Cookies\}) = \sigma(Milk \cup Cookies)/\sigma(N) = 1/10 = 0.1$

$s(\{Milk, Bread\}) = \sigma(Milk \cup Bread)/\sigma(N) = 3/10 = 0.3$

$s(\{Beer, Butter\}) = \sigma(Beer \cup Butter)/\sigma(N) = 0/10 = 0$

$s(\{Beer, Diapers\}) = \sigma(Beer \cup Diapers)/\sigma(N) = 3/10 = 0.3$

$s(\{Beer, Cookies\}) = \sigma(Beer \cup Cookies)/\sigma(N) = 2/10 = 0.2$

$s(\{Beer, Bread\}) = \sigma(Beer \cup Bread)/\sigma(N) = 0/10 = 0$

$s(\{Butter, Diapers\}) = \sigma(Butter \cup Diapers)/\sigma(N) = 3/10 = 0.3$

$s(\{Butter, Cookies\}) = \sigma(Butter \cup Cookies)/\sigma(N) = 1/10 = 0.1$

$\boxed{s(\{Butter, Bread\}) = \sigma(Butter \cup Bread)/\sigma(N) = 5/10 = 0.5}$

$s(\{Diapers, Cookies\}) = \sigma(Diapers \cup Cookies)/\sigma(N) = 2/10 = 0.2$

$s(\{Diapers, Bread\}) = \sigma(Diapers \cup Bread)/\sigma(N) = 3/10 = 0.3$

$s(\{Cookies, Bread\}) = \sigma(Cookies \cup Bread)/\sigma(N) = 1/10 = 0.1$

→ From above results, we can see that 2-itemset {Butter, Bread} has the highest support of 0.5.

(b) a = Butter, b = Bread

$$\text{Confidence } c(X \to Y) = \frac{\text{support}(X \cup Y)}{\text{support}(X)} = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

$$c(\text{Butter} \to \text{Bread}) = \frac{\sigma(\text{Butter} \cup \text{Bread})}{\sigma(\text{Butter})}$$

$$= \frac{5}{5}$$

$$= 1$$

$$= 100\%$$

$$c(\text{Bread} \to \text{Butter}) = \frac{\sigma(\text{Bread} \cup \text{Butter})}{\sigma(\text{Bread})}$$

$$= \frac{5}{5}$$

$$= 1$$

$$= 100\%$$

→ Here, we can see that both rules {Butter}→{Bread} and {Bread}→{Butter} have same confidence. Because, support for individual itemset {Butter} and {Bread} are equal. But this may not be the case always as seen in above 1(c). Hence, confidence is assymetric.

3.

For the questions below: For each step of your tree build process show what condition you used. The answers should be concise. Only show the details to prove that you build the tree yourself. For part (b) show the details of placing itemsets down each level of the tree.

Consider the following set of candidate 3-itemsets:

$$\{1,2,3\},\{1,2,6\},\{1,3,4\},\{2,3,4\},\{2,4,5\},\{3,4,6\},\{4,5,6\}$$

(a) Construct a hash tree for the above candidate 3-itemsets. Assume the tree uses a hash function where all odd-numbered items are hashed to the left child of a node, while the even-numbered items are hashed to the right child. A candidate k-itemset is inserted into the tree by hashing on each successive item in the candidate and then following the appropriate branch of the tree according to the hash value. Once a leaf node is reached, the candidate is inserted based on one of the following conditions:

Condition 1: If the depth of the leaf node is equal to k (the root is assumed to be at depth 0), then the candidate is inserted regardless of the number of itemsets already stored at the node.

Condition 2: If the depth of the leaf node is less than k, then the candidate can be inserted as long as the number of itemsets stored at the node is less than $maxsize$. Assume $maxsize$ = 2 for this question.

Condition 3: If the depth of the leaf node is less than k and the number of itemsets stored at the node is equal to $maxsize$, then the leaf node is converted into an internal node. New leaf nodes are created as children of the old leaf node. Candidate itemsets previously stored in the old leaf node are distributed to the children based on their hash values. The new candidate is also hashed to its appropriate leaf node.

(b) Consider a transaction that contains the following items: {1,2,3,5,6}. What are the candidate 3-itemsets contained in the transaction?

(c) Consider a transaction that contains the following items: {1,2,3,5,6}. Using the hash tree constructed in part (a), which leaf nodes will be matched against the transaction?

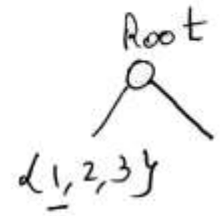
Ans.:

3. {1,2,3}, {1,2,6}, {1,3,4}, {2,3,4}, {2,4,5}, {3,4,6}, {4,5,6}

(a) Here, the tree uses a hash function where all odd-numbered items are hashed to the left child and even-numbered items are hashed to the right child.

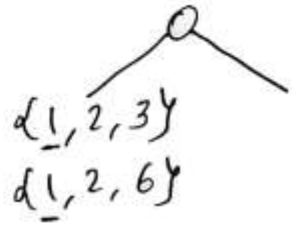
∴ Hash function = p mod 2. So, items 1, 3, 5 are hashed to the left branch because they all have same remainder after dividing by 2. Items 2, 4, 6 are hashed to the right branch.

(1) {1,2,3} : Condition-1 is followed here.



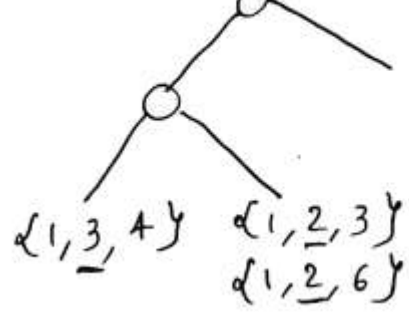
- 1st item in the itemset which is '1', so it is hashed to the left branch.

(2) {1,2,6} : Condition-2 is followed here.



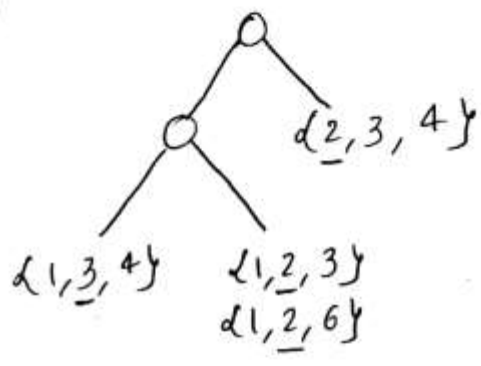
- {1,2,6} is hashed to the left branch.

(3) {1,3,4} : Condition-3 is followed here.



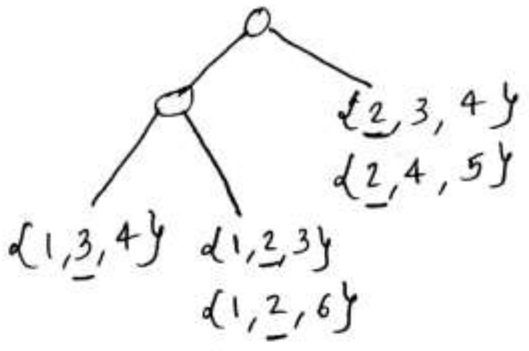
- New leaves are created. {1,3,4} is hashed to the left, {1,2,3}, {1,2,6} are hashed to the right according to 2nd item in the itemset.

(4) $\{2,3,4\}$: Condition-1 is followed here.



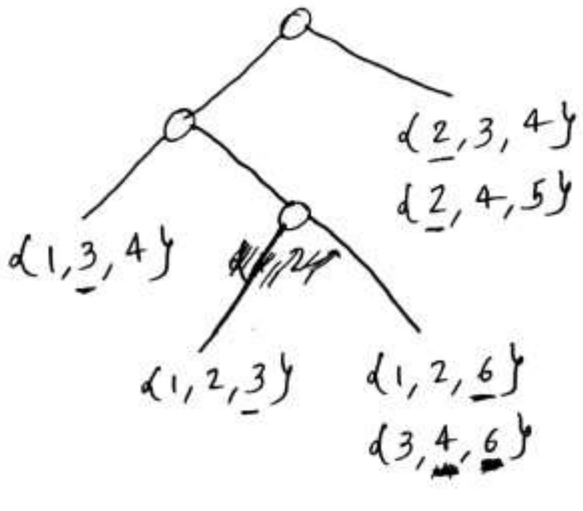
- $\{\underline{2},3,4\}$ is hashed to the right according to first item which is 2.

(5) $\{2,4,5\}$: Condition-2 is followed here.



- $\{\underline{2},4,5\}$ is hashed to the right according to first item which is 2.

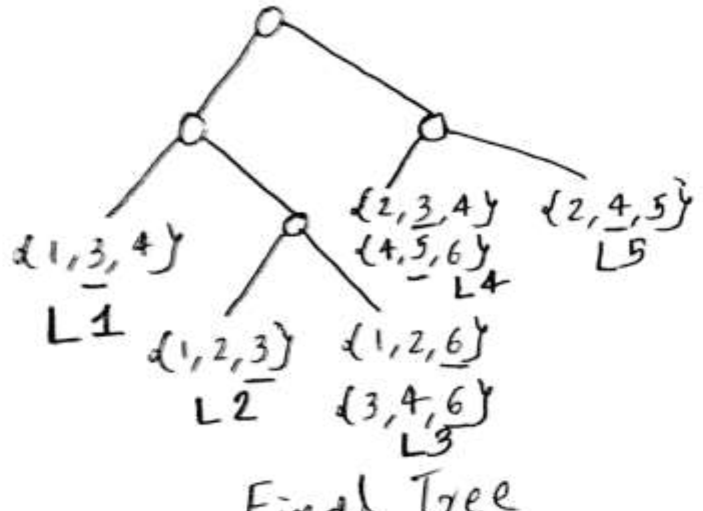
(6) $\{3,4,6\}$: Condition-3 is followed here.



- New leaves are created.
$\{3,\underline{4},6\}$ is hashed to the right according to second item.
$\{1,2,\underline{3}\}$ is hashed to the left and $\{1,2,\underline{6}\}$ is hashed to the right.

(7) {4,5,6} : Condition-3 is followed here.

- New leaves are created. {2,3,4}, {4,5,6} are hashed to the left. {2,4,5} are hashed to the right.



Final Tree

(b) A transaction with items: {1,2,3,5,6}. Given above hash tree, we can say that leaf nodes L1, L2, L3 and L4 will be checked against the transaction. The 3 candidate itemsets contained in the transaction include {1,2,3} and {1,2,6}

(C) From above hash tree, leaf nodes L1, L2, L3 and L4 will be checked against the transaction.

4. How will you implement the Grep tool using MapReduce? The tool has to extract matching strings from text files and counts how many time they occurred. How many MapReduce job are required? Define the Input and Output key-value pairs for each MapReduce job. Write the pseudocode for the implementation of the Mapper and Reducer for each MapReduce job.

Ans.:

4. Grep tool using MapReduce ①

→ Grep tool searches for a string in a text file and return the count of how many times the string has appeared.

# of MR jobs:

→ The program will need 2 map/reduce jobs. The first job counts how many times a matching string occured and the second job sorts matching strings by their frequency and stores the output in a single output file.

MR job-1:

Mapper: It takes line as input and matches the string against the line. It extracts all matching strings and emits (matching string, 1) key-value pairs.

Input: key = file ID
value = line

Output: key = matching string
value = 1

Psuedocode:

```
class Mapper
    method Map(fileID, line)
        for each line
            if line matches findString
                Emit(matching String, 1)
```

findString = Userdefined input string.

Reducer: It will sum up the count of each matching string. It will create output as a file containing matching string and count.

Input: key = matching string
value = counts

Output: key = matching string
value = sum of counts

Psuedocode:

```
class Reducer
    method Reduce(matching string, counts [c1, c2, c3, ...])
        sum = 0
        for each count in counts [c1, c2, c3, ...]
            sum = sum + count
        Emit(matching string, sum)
```

Combiner: The reducer can be optimized by running combiner which sums up frequency of strings from output of map function to reduce amount of data needs to be shipped to reducer.

→ Input, Output and Psuedocode are same as shown in Reducer.

MR job-2:

Mapper: The mapper is an inverse map, it swaps the keys and values.

Input: key = matching string
value = sum

Output: key = sum
value = matching string

Psuedcode:

```
class Mapper
    method Map(matching string, sum)
        Emit(sum, matching string)
```

Reducer: It will sort the counts of matching string and store output in a single text file.

Input: key = sum
value = matching string

Output: key = sum
~~mat~~ value = matching string

Psuedocode:

```
class Reducer
    method Reduce(sum, matching string)
        sort matching string by sum
        Emit(sum, matching string)
```