Data Mining

Vismaya Veeramanju Kalyan

1 a) Total number of transactions = 10

Support (X) = $\frac{\textit{Number of transaction containing } X}{\text{Total number of transactions}}$

$$\textit{Number of transaction containing } \{e\} = 8$$

Support (e) = $\frac{\textit{Number of transaction containing } \{e\}}{\text{Total number of transactions}}$ = 8/10 =0.8

$$\textit{Number of transaction containing } \{b, d\} = 2$$

Support (b,d) = $\frac{\textit{Number of transaction containing } b,d}{\text{Total number of transactions}}$ = 2/10 = 0.2

$$\textit{Number of transaction containing } \{b, d, e\} = 2$$

Support (b,d,e) = $\frac{\textit{Number of transaction containing } b,d,e}{\text{Total number of transactions}}$ = 2/10 = 0.2

{e} has higher support compared to {b,d} or {b,d,e} and hence {e} itemset is more frequent in the item-set.

1 b) confidence (x -> y) = support(x U y) / support(x)

b confidence - how often transactions that contain X also contain Y

$$C(\{b,d\} \to \{e\}) = \frac{\sigma(b,d,e)}{\sigma(b,d)} = \frac{0.2}{0.2} = \underline{\underline{1}}$$

The term shows that if a person buys b,d chances of it contain e

→ we find the support for union of those 2 itemset (b,d,e) divide by the support of {b,d}

In this example all the transactions that contain b,d also contains e hence has a very high confidence of 1

$$C(\{b,d\} \to \{e\}) = \frac{\sigma(b,d,e)}{\sigma(b,d)} = \frac{\sigma(b,d,e)/N}{\sigma(b,d)/N}$$

$$\frac{\text{support}(b,d\ e)}{\text{support}(b,d)} = \frac{2/10}{2/10} = \frac{0.2}{0.2} = 1 \qquad \boxed{100\%\ \text{confidence}}$$

$$c(\{e\} \rightarrow \{b,d\}) = \frac{\sigma(e,b,d)}{\sigma(e)}$$

$$= \frac{\frac{2}{10}}{\frac{8}{10}}$$

$$= \frac{0.2}{0.8}$$

$$= 0.25 \quad \boxed{25\% \text{ confidence}}$$

Here see the confidence of {b,d} being bought when item set e is bought.

- Find the support of ({e} U {b,d}) & {e}
- you can calculate the confidence by the support. ~~of~~ ~~all~~ calculated.

The confidence is pretty low for this as many transaction contain just {e} but not {b,d}

c) No confidence is not a symmetric measure as we saw in the above example itself. A customer buying {b,d} always buys {e} as confidence is 1. on the other hand the customer buying {e} has only 0.25 chances of buying {b,d}.

2a. Bread Butter is the 2-itemset that has the largest support.

Support (Bread,Butter) = $\frac{\textit{Number of transaction containing } \{\text{Bread,Butter}\}}{\text{Total number of transactions}}$ = 5/10 =0.5

2 a) {Bread, Butter}

$$\text{Support} = \frac{\sigma(\text{Bread, butter})}{N} = \frac{5}{10} = 0.5 \text{ support.}$$

b) $$c(\text{Bread} \rightarrow \text{Butter}) = \frac{\sigma(\text{Bread, Butter})}{\sigma(\text{Bread})} = \frac{0.5}{5/10} = 1$$

confidence = 100%.

$$c(\text{Butter} \rightarrow \text{Bread}) = \frac{\sigma(\text{Butter, Bread})}{\sigma(\text{Butter})} = \frac{0.5}{5/10} = 1$$

confidence = 100%.

The confidence for this is rule imply that people who take butter take bread and vice- versa.

From the data in the table we can see that bread and butter appear together in the transactions. There is no transaction that contain bread or butter (1-itemset). Hence the support for bread and butter together is same as the support for the bread or butter alone. As the confidence is calculated from the support of either bread or butter and union of bread and butter we get a very high confidence of 1 (100%).

This is special case where the 2-items {bread and butter} always appeared together making the confidence same for the association rule.

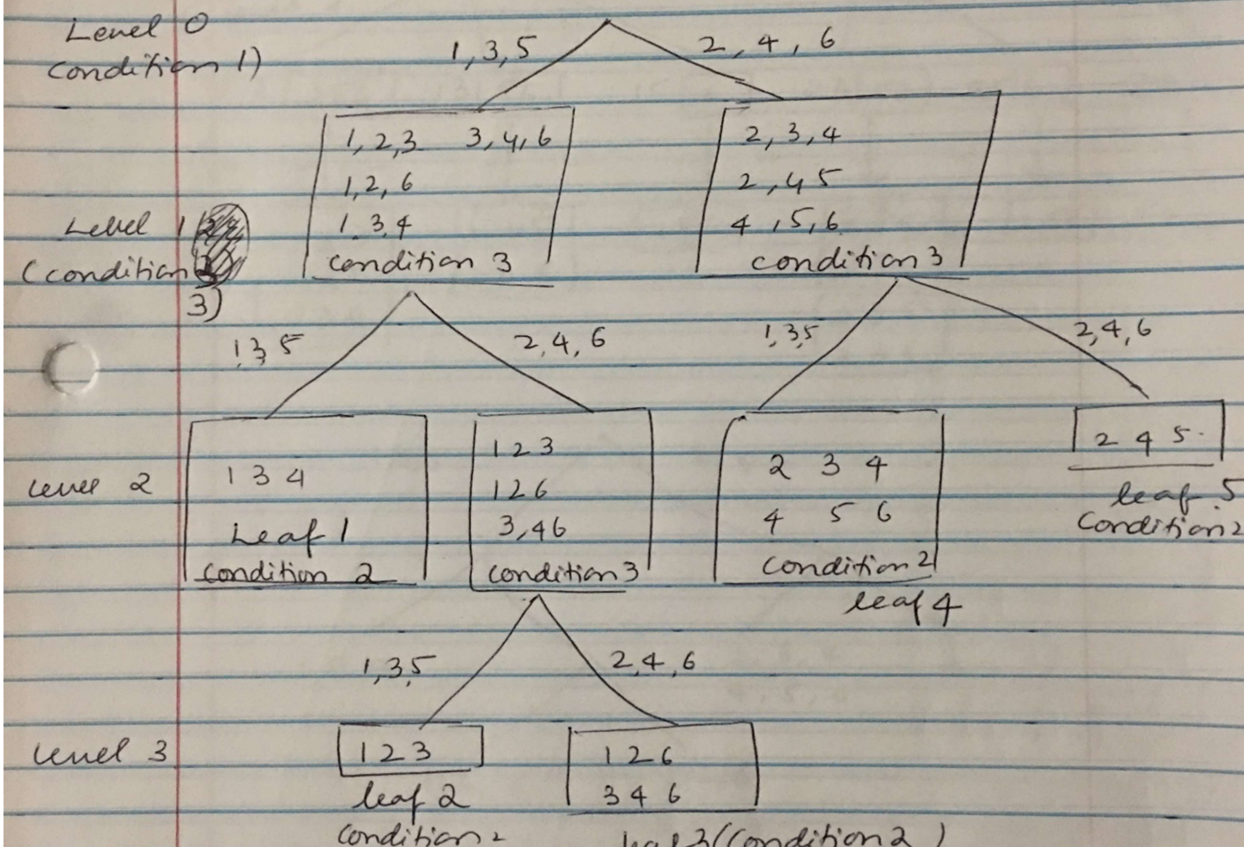
In the 1 a question the {b,d,e} do not always appear together in the transactions. e appears independently in many transactions making the confidence of e ->b,d low. On the other hand b, d always appear with e hence b,d -> e has a high confidence of 1.

3a.

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

odd numbered - left child node

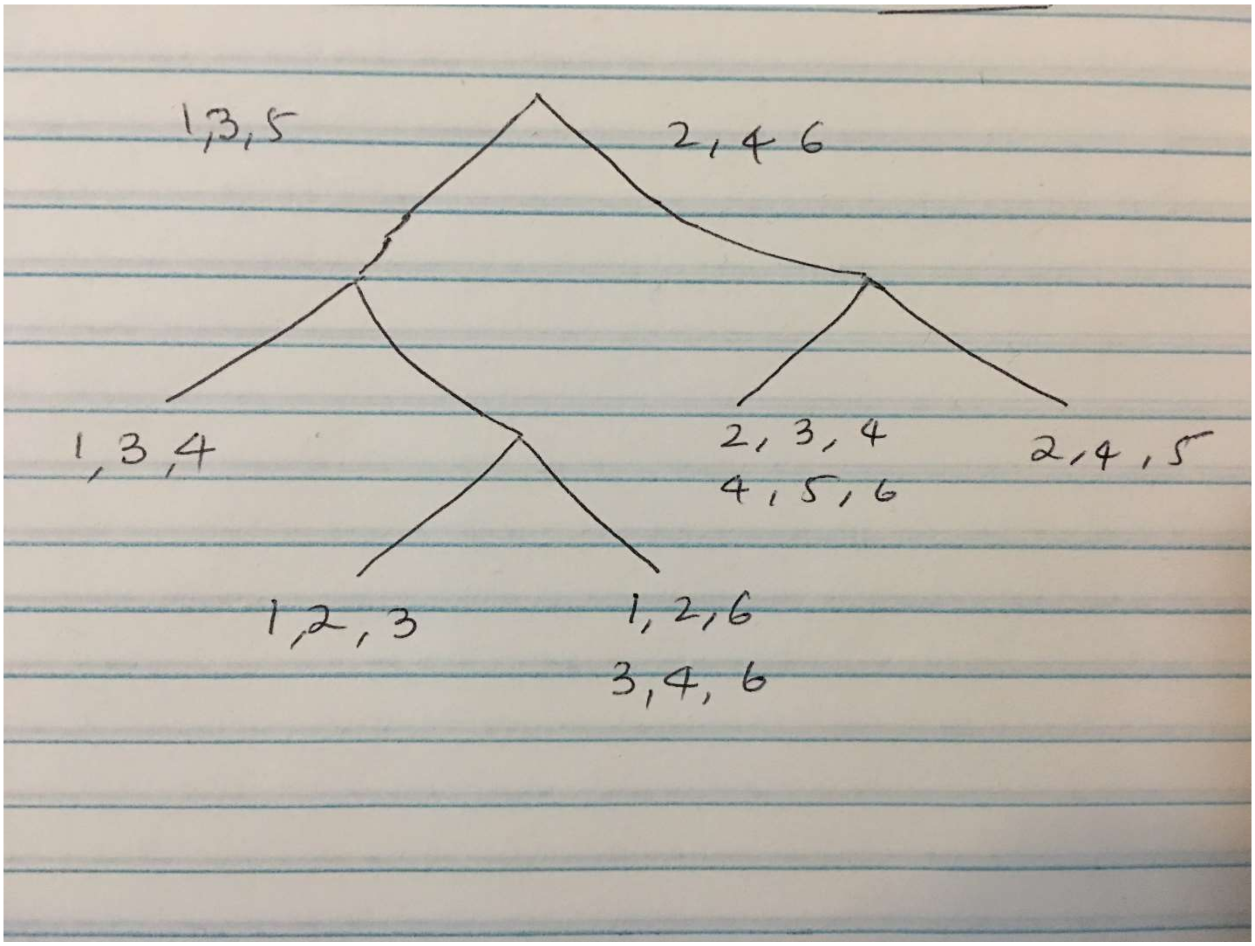
even numbered - right child node



At depth 0 all the candidate itemset is inserted as stated by the condition 1. It follows the hash function where all odd numbered items are hashed to the left tree and all even numbered are hashed to the right tree.

At level 1, the itemset stored becomes equal to the maxsize so its converted into a internal node and the same hash function is applied (here condition 3 is used) for both the left and the right tree.

At level 2 we have 3 leaf node which follow the condition 2. And 1 node which follows condition 3 and hence splits again to children node. The proper diagram of the hash tree is given below.



Leaf 1: {1,3,4}

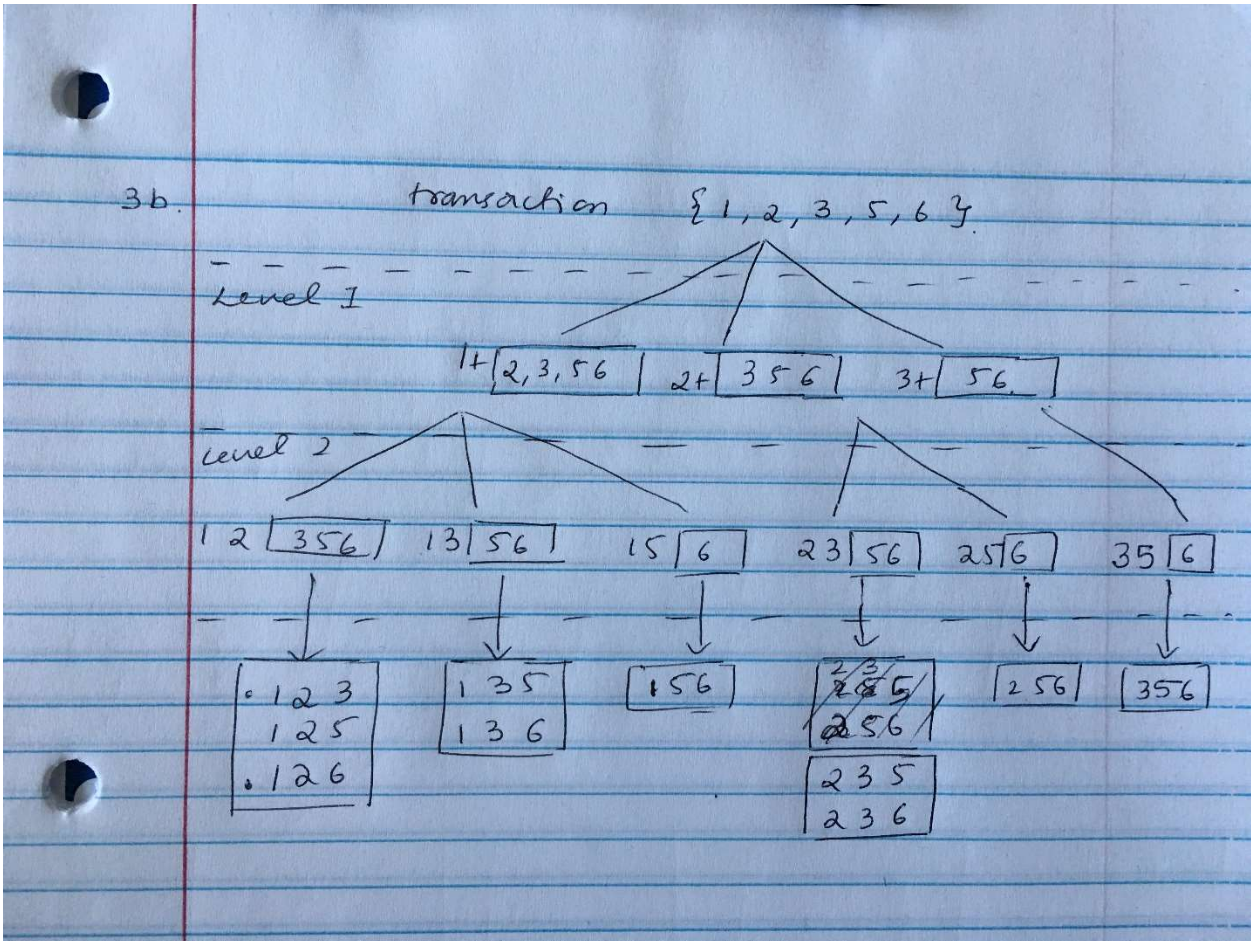
Leaf 2: {1,2,3}

Leaf 3: {1,2,6} {3,4,6}

Leaf 4: {2,3,4} {4,5,6}

Leaf 5: {2,4,5}

3b.



The 3 candidate item-set contained in the transaction are

(1,2,3),(1,2,5),(1,2,6),(1,3,5),(1,3,6),(1,5,6),(2,3,5),(2,3,6),(2,5,6),(3,5,6).

Each itemset keeps its items in increasing lexicographic order.

Transaction {1,2,3,5,6},

All the 3- itemsets contained in transaction must begin with item 1, 2, or 3. It is not possible to construct a 3-itemset that begins with items 5 or 6 because there are only two items in transaction whose labels are greater than or equal to 5.

1+[2 3 5 6] represents a 3-itemset that begins with item 1, followed by two more items chosen from the set {2,3,5,6}. (Similar operation is carried out for 2+[3,5,6] and 3+[5,6]). Level 2 prefix structures represent the different ways in which the second item is selected. For e.g., 1 2 [3 5 6] corresponds to itemsets that begin with (1 2) and are followed by 3, 5 or 6. Level 3 enumerates all the 3-itemsets contained in the transaction. For example, the 3-itemsets that begin with prefix {1 2}are {1,2,3}, {1,2,5}, and {1,2,6} and so on.

3c .

Transaction is {1,2,3,5,6}

leaf nodes that will be matched against the transaction are leaf node 2 {1,2,3} and leaf node 3 {1,2,6}

4.

The program runs two map/reduce jobs in sequence.

- The first job counts how many times a matching string occurred.
- The second job sorts matching strings by their count and stores the output in a single output file.

First job

Each mapper of the first job takes a line as input and matches the user-provided regular expression against the line. It extracts all matching strings and emits (matching string, 1) pairs.

Input< fileID_lineNumber,line>

```
    String matching_string = user_defined.

    String[] tokens = line.split("\n"); // Tokenize the line

    For( String token : tokens){

        If ( tokens.matches (matching_string )

        emits (String matching_string , count 1)
```

Each reducer sums the count of each matching string. The output is sequence files contaning the matching string and count. The reduce phase is optimized by running a combiner that sums the frequency of strings from local map output. As a result, it reduces the amount of data that needs to be shipped to a reduce task.

Input< String matching_string , counts [c1, c2,…] >

```
    Sum =0

    For all count c  in [c1, c2,…] do

        Sum = sum + c

    Emit(String matching_string, count sum)
```

The pseudo code doesn't show a combiner.

Second job

The second job takes the output of the first job as input.

The mapper is an inverse map, which means it swaps the keys and values.

Input< String matching_string , count sum>

Swap(String matching_string , count sum)

Emit(count sum, String matching_string)

The reducer is an identity reducer. It sorts by count in the descending order but there will be no aggregation ie grouping.

The number of reducers is one, so the output is stored in one file. The output file is text, each line of which contains count and a matching string

Input< count sum, String matching_string >

Sort(count sum, String matching_string, decending_order)

Emit(count sum, String matching_string)