Jay Patel
CS422 - Data Mining
10/09/18

ASSIGNMENT 4

Association Rule Mining & Map Reduce

Part 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) Compute Support

There are total 10 distinct transactions. for the item set $\{e\}, \{b, d\}, \{b, d, e\}$,we shall check their occurrence in the given transactions.

The support count for item-set is given by : $S(item) = \frac{\#trans\_with\_item}{N}$, Therefore

$$S(\{e\}) = \frac{8}{10} = 0.8 \qquad (1)$$

$$S(\{b, d\}) = \frac{2}{10} = 0.2 \qquad (2)$$

$$S(\{b, d, e\}) = \frac{2}{10} = 0.2 \qquad (3)$$

b) Compute Confidence

The confidence can be calculated based on the values we found in (a). The confidence is give by the formula : $C(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$, Therefore

$$C(\{b, d\} \rightarrow \{e\}) = \frac{0.2}{0.2} = 100\,\% \qquad \text{[note: using (2) and (3)]}$$

$$C(\{e\} \rightarrow \{b, d\}) = \frac{0.2}{0.8} = 25\,\% \qquad \text{[note: using (1) and (3)]}$$

c) Is Confidence a symmetric measure?

In the previous question, we calculated the confidence for the association, and measuring the difference by reversing the values. we see that the answers do not match, since the support for e is 0.8 while reduces the confidence to 25%. Hence we can conclude that the confidence is **not symmetric**.

Part 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) 2 Item-set with largest support.

For this question, we could use the brute force method and find support for each possible pair from the 6 items. However, Since there are only 10 entries, we can analyze the table by looking at it, and notice that item-set $\{bread, butter\}$ appear in 5 out of 10 entries. Just like we calculated support in part 1. the support,

$$S(\{bread, butter\}) = \frac{5}{10} = 0.5$$

b) We are to compare the confidence for the rule $\{bread\} \rightarrow \{butter\}$ and $\{butter\} \rightarrow \{bread\}$. we need support value for both $\{bread\}\ and\ \{butter\}$.

$$S(\{butter\}) = \frac{5}{10} = 0.5$$

$$S(\{bread\}) = \frac{5}{10} = 0.5$$

$$C(\{bread\} \rightarrow \{butter\}) = \frac{0.5}{0.5} = 100\,\%$$

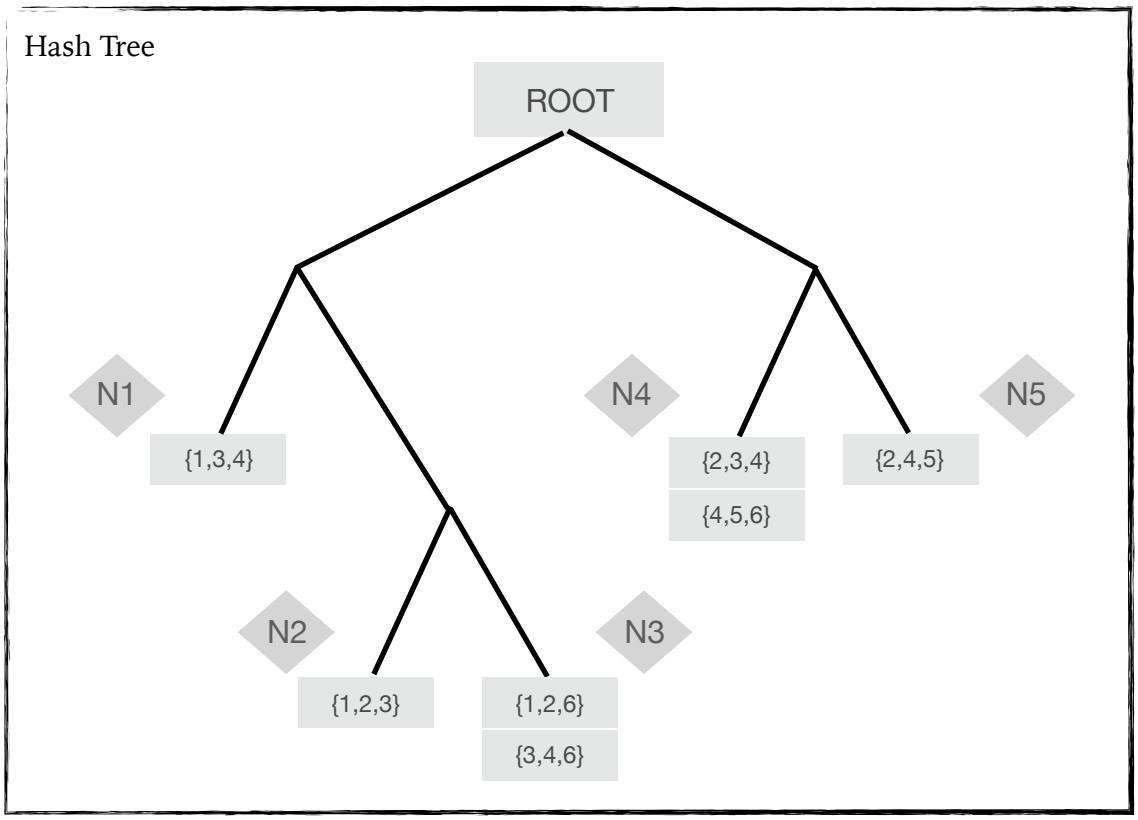
$$C(\{butter\} \rightarrow \{bread\}) = \frac{0.5}{0.5} = 100\,\%$$

In this calculation, we got same results for both $\{bread\}\ and\ \{butter\}$. This is due to the fact that they both occur 5 times, and that they occur in same transaction as well. Hence, we get that the confidence is symmetric both case. Although, the confidence is symmetric in this case, it's not necessary to be symmetric for other item-set. Each association rules works on different support and confidence measure.

Part 3:

a) {1,2,3}, {1,2,6}, {1,3,4}, {2,3,4}, {2,4,5}, {3,4,6}, {4,5,6} is the provided candidate 3 Item-set. The following Hash Tree satisfy all the conditions that were to be followed in the question.

- Candidates are stored in the leaf at depth k
- When the depth is less than the leaf, then we add another candidate like we did in node-2, & node-4. maximize is 2.
- since the max size is reached for N2 and N4, we had to create new children at depth.



b) The given transaction is {1,2,3,5,6}. Starting at the root, we use the increment count to hast the transaction t. As we encounter the numbers in set, we add more value in the counter, and then once you read the node, you match the 3 candidate set with your transaction t. Performing the traverse, the candidate item-set contained in the transactions are {1,2,3} *and* {1,2,6}.

c) Starting from the left, N1, N2, N3, N4 will be the leaf nodes that will be checked against the transaction. During the traverse, we will try to match the node that contains the transaction t's data set in the candidate set. since N5 will not be match become the traverse will fail to reach the node. For N5, the next number is 4, which is not in the transaction.

Part 4:

The Grep tool is a command-line functions that would extract matching string, that are passed as an argument, and would return the number of times it occurred in a file. We are tasked to implement the functions of Grep tool using MapReduce Algorithm. MapReduce is implemented over various computers(Distributed_Computing) that can run the same program divided into multiple threads (machine) to get faster result. This can be used to scale the program over a large data, and thats the advantage of it.

The MapReduce, as the name suggest, contains two main tasks(jobs), Map and Reduce. Map processes the input file. It is passed to the function line by line, and the program creates small packets of data. In the reduce stage, the program shuffles the data chunks into orderly manner, and then combines the same packages increasing the counts and produces a new output file.

Implementing the Grep Tool, Our program will consist of two map/reduce jobs in a sequence. As the mapper counts the number of times matching string occur in a file, the second job sorts the matching strings into frequency and saves it all together in an output file. Following is a proposed pseudocode.

Algorithm : Implementation of MapReduce inspired Grep Tool.

```
class MAPPER                                        //mapper program
    function map(docid, document d)
        for line in d
            words = line.strip()                    // break down lines into words
            for word in words
                ( key, value ) —> emit(word, 1)     //display or save the word, and count.
        return —> tuple = [(key, value)...(key, value)]

class REDUCER                                       //reducer program
    function reduce(tuple)
        sum = 0
        currentWord = NULL                          // set place holder for current word
        for (key, value) in  tuple
            if currentWord != key                   // check if word not repeated
                if currentWord is not NULL
                currentWord = key
            sum = sum + value                       //add the value for all the like words

        return -> outFile = emit(currentWord , sum)
```

```
class GREP                                  //grep tool
    function init('string',collection)      //pass string and collection of data file
        outFile_collection                  // place to store output file from first Job
        for file in collection              // FIRST MAP/REDUCE JOB SET
            tuple = map(file)
            outFile = reduce(tuple)
            outFile+=outFile_collection     //add in output collection

                                            //SECOND MAP/REDUCE JOB SET
            tuple = map(outFile_collection) //Inverse Mapping begin
            outFile = reduce(tuple)

        for (word , sum) in outFile
            if word == string
                return emit(word, sum)      //output final answer
```

Here, we have used 3 distinct programs that would implement the grep tool. As mentioned previously, our program will have 2 map reduce jobs. The first map job will take (doc id, and document d) as the key value pair, where it will split the lines, and then into words, and assign a count of 1 to it. the output of the first Map job will be a key value pair of tuple into (word, 1). This then goes to the reducer program that will basically combine the like keys and sum up their values. The second map job is basically it will split the words and values again taking input and key value pair from previous reduce, and then combine into a single output file using reduce. Now, we have also implemented the grep function , which is basically our main, function that will take in the (string, collection) as the key value pair, and will in return, after the 2 job cycle receive a (string, sum) as the output.