DATA ANALYTICS

ARM PROJECT

2018101119 - Pullamma Mayakuntla
2019101101 - Stella Sravanthi

Introduction :

In this project, we did the implementation of frequent itemset mining algorithms; Apriori and FPgrowth.

Data and Data Cleaninge

For the same, we used the datasets from SPMF: Open dataset library which contains the itemsets in the numbered format and in the form of text files. We chose 2 datasets and implemented the algorithms on both these datasets. Following are the links for the same;
http://www.philippe-fournier-viger.com/spmf/datasets/mushrooms.txt (MUSHROOM DATASET)

Data cleaning : We did the necessary data preprocessing to get the processed data from the text file. We named/labeled the first column as ‘items’.
Mushroom dataset: 8416 transactions and 119 distinct items.

```
[19] df.rename(columns={df.columns[0]: 'new'},inplace=True)
```

```
df
```

	new
0	1 5 12 21 23 25 36 39 42 53 56 57 67 71 79 88 ...
1	1 5 12 21 23 25 36 39 42 53 56 57 67 71 79 88 ...
2	1 5 12 21 23 25 36 39 42 50 56 57 67 71 79 88 ...
3	1 5 12 21 23 25 36 39 42 50 56 57 67 71 79 88 ...
4	1 5 12 21 23 25 36 39 42 44 56 57 67 71 79 88 ...
...	...
8411	1 7 12 13 24 31 34 38 41 44 55 63 67 71 76 85 ...
8412	1 7 12 13 24 31 34 38 41 44 55 63 67 71 76 85 ...
8413	1 7 12 13 24 31 34 38 41 44 55 63 67 71 76 85 ...
8414	1 7 12 13 24 31 34 38 41 44 55 63 67 71 76 85 ...
8415	1 7 12 13 24 31 34 38 41 44 55 63 67 71 76 85 ...

Task 1 - Implementation of FP Growth Algorithm

Main steps in the algo are:

1) Constructing FP tree, header table with cleaned itemset
2) Mining the tree and conditional FP trees

With merging strategy which helps us to optimise in time and space while generating conditional pattern bases during pattern-growth mining

Steps:

1) Each node of tree is represented in struct named FP_Node

```
struct FP_Node
{
    FP_Node* parent;
    int val;
    int cnt;
    bool visited;
    map<int,FP_Node*>children;
};
```

- cnt : Count of an item in the FP_Tree
- val : Stores the item's numeric value
- FP_TreeNode* parent : address of the parent node.
- map<int, FP_TreeNode*>children : point to all children of a node
- bool visited : whether the node is visited before or not

2) Storing the transactions
   `vector<pair<set<int>,int> > datum`
3) All possible subsets are calculated by a simple logic of single and multiple branches
   a) Single branch is just to check for the presence of that item in the Subset.

```
bool traverse_tree(FP_Node* root)
{
   FP_Node* t =  root;
   while(t != nullptr)
   {

        if(t->children.size() == 0) return true;
        else if(t->children.size() > 1) return false;
        map<int, FP_Node*>::iterator itr; itr =
t->children.begin();
        t = itr->second;
   }
   return true;
```

b) Multiple branches means need to check conditional pattern which leads to look up header table here **tb.**

4) Merging technique helps us to avoid visited nodes and only unvisited will be traversed till root with the address vector.
5) For all unvisited nodes of a path, the conditional transaction will be a subset and further marked and subset added in the conditional pattern base of the element.

Task-2: Apriori Algorithm Generate Closed frequent Itemsets

(Used Mushroom dataset with minimum support count set as 5050 threshold)

1. **How I calculated Support count :** We have scanned the database only once. First We have obtained all unique items and for each item, we made a bit string of length of database (8416). Traverse through the database , if an item occurs in transaction i of the database , then ith position of the string is made as 1. Like this, we obtained bit strings for all and kept all in a dictionary with unique items as keys. This is a more optimized way of counting support count because, if we want support count of a pattern [x,y], simply perform bitwise and of bit strings of x and y and we will get the result as bitstring and count number of ones in that , that gives the support count.
2. Length 1 frequent itemsets are obtained as follows.

```
df2
```

	items	support_count	pattern_size
0	[36]	8200	1
1	[38]	6824	1
2	[41]	5880	1
3	[67]	5316	1
4	[71]	5076	1
5	[90]	8416	1
6	[94]	8216	1
7	[97]	7768	1

2. Frequent Itemsets

Frequent itemsets are itemsets whose support is greater than or equal to given minimum support cunt(here=5050)

	items	support_count	pattern_size
0	[36]	8200	1
1	[38]	6824	1
2	[41]	5880	1
3	[67]	5316	1
4	[71]	5076	1
5	[90]	8416	1
6	[94]	8216	1
7	[97]	7768	1

8	[36, 38]	6608	2
9	[36, 41]	5664	2
10	[36, 67]	5124	2
11	[36, 90]	8200	2
12	[36, 94]	8192	2
13	[36, 97]	7576	2
14	[38, 90]	6824	2
15	[38, 94]	6632	2
16	[38, 97]	6464	2
17	[41, 90]	5880	2
18	[41, 94]	5688	2
19	[41, 97]	5232	2
20	[67, 90]	5316	2
21	[67, 94]	5124	2
22	[71, 90]	5076	2
23	[90, 94]	8216	2
24	[90, 97]	7768	2
25	[94, 97]	7568	2
26	[36, 38, 90]	6608	3
27	[36, 38, 94]	6608	3
28	[36, 38, 97]	6272	3

29	[36, 41, 90]	5664	3
30	[36, 41, 94]	5664	3
31	[36, 67, 90]	5124	3
32	[36, 67, 94]	5124	3
33	[36, 90, 94]	8192	3
34	[36, 90, 97]	7576	3
35	[36, 94, 97]	7568	3
36	[38, 90, 94]	6632	3
37	[38, 90, 97]	6464	3
38	[38, 94, 97]	6272	3
39	[41, 90, 94]	5688	3
40	[41, 90, 97]	5232	3
41	[67, 90, 94]	5124	3
42	[90, 94, 97]	7568	3
43	[36, 38, 90, 94]	6608	4
44	[36, 38, 90, 97]	6272	4
45	[36, 38, 94, 97]	6272	4
46	[36, 41, 90, 94]	5664	4
47	[36, 67, 90, 94]	5124	4
48	[36, 90, 94, 97]	7568	4
49	[38, 90, 94, 97]	6272	4

50	[36, 38, 90, 94, 97]	6272	5

3. Time take to generate all frequent itemsets is

Time taken to extract frequent itemsets using apriori

```
t2-t1
```

```
17.718475341796875
```

4. Closed Frequent Itemsets

Closed Frequent Itemsets are itemsets whose supersets' support count is not equal to itself.

	items	support_count	pattern_size
0	[90]	8416	1
1	[36, 90]	8200	2
2	[38, 90]	6824	2
3	[41, 90]	5880	2
4	[67, 90]	5316	2
5	[71, 90]	5076	2
6	[90, 94]	8216	2
7	[90, 97]	7768	2
8	[94, 97]	7568	2
9	[36, 90, 94]	8192	3
10	[36, 90, 97]	7576	3
11	[36, 94, 97]	7568	3
12	[38, 90, 94]	6632	3

13	[38, 90, 97]	6464	3
14	[38, 94, 97]	6272	3
15	[41, 90, 94]	5688	3
16	[41, 90, 97]	5232	3
17	[67, 90, 94]	5124	3
18	[90, 94, 97]	7568	3
19	[36, 38, 90, 94]	6608	4
20	[36, 38, 94, 97]	6272	4
21	[36, 41, 90, 94]	5664	4
22	[36, 67, 90, 94]	5124	4
23	[36, 90, 94, 97]	7568	4
24	[38, 90, 94, 97]	6272	4
25	[36, 38, 90, 94, 97]	6272	5

Optimizations :

1. **Hash Based Technique**:

When scanning each transaction in the database to generate the frequent 1-itemsets , we can generate all the 2-itemsets for each transaction, hash (i.e., map) them into the different buckets of a hash table structure, and increase the corresponding bucket counts. A 2-itemset with a corresponding bucket count in the hash table that is below the support threshold cannot be frequent and thus should be removed from the candidate set.

As specified above we have used bit strings method for support count hence not need to scan the database again and again. The Bit Strings method is also hashing with optimization. Hence not implemented this technique.(Did Not implement apriori with loops of scanning database again and again).

2. **Partition based Technique:**
   We have implemented partition based technique by dividing a given dataset into 6 parts and so the 5050 support count by 6.
   Again we merge all the itemses got in different itemsets.