Generating Frequent Item-sets

CODE SEQUENCE

Step 1: Convert the mushroom dataset (*mushroom.txt*) to market basket data(*mbd.txt*).

Python Code: *conversion.py* -> convertToMBD ( )

Some Outputs:

```
In [79]: runfile('F:/51/codes/convertToMBD.py', wdir='F:/51/codes')
nameList = [['e', 'p'], ['b', 'c', 'f', 'k', 's', 'x'], ['f', 'g', 's', 'y'], ['b', 'c',
'e', 'g', 'n', 'p', 'r', 'u', 'w', 'y'], ['f', 't'], ['a', 'c', 'f', 'l', 'm', 'n', 'p',
's', 'y'], ['a', 'f'], ['c', 'w'], ['b', 'n'], ['b', 'e', 'g', 'h', 'k', 'n', 'o', 'p',
'r', 'u', 'w', 'y'], ['e', 't'], ['?', 'b', 'c', 'e', 'r'], ['f', 'k', 's', 'y'], ['f',
'k', 's', 'y'], ['b', 'c', 'e', 'g', 'n', 'o', 'p', 'w', 'y'], ['b', 'c', 'e', 'g', 'n',
'o', 'p', 'w', 'y'], ['p'], ['n', 'o', 'w', 'y'], ['n', 'o', 't'], ['e', 'f', 'l', 'n',
'p'], ['b', 'h', 'k', 'n', 'o', 'r', 'u', 'w', 'y'], ['a', 'c', 'n', 's', 'v', 'y'],
['d', 'g', 'l', 'm', 'p', 'u', 'w']]
len(finalList) = 8124
finalList[0] = [[0, 1], [0, 0, 0, 0, 0, 1], [0, 0, 1, 0], [0, 0, 0, 0, 1, 0, 0, 0, 0,
0], [0, 1], [0, 0, 0, 0, 0, 0, 1, 0, 0], [0, 1], [1, 0], [0, 1], [0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0], [1, 0], [0, 0, 0, 1, 0], [0, 0, 1, 0], [0, 0, 1, 0], [0, 0, 0, 0, 0, 0,
0, 1, 0], [0, 0, 0, 0, 0, 0, 0, 1, 0], [1], [0, 0, 1, 0], [0, 1, 0], [0, 0, 0, 0, 1],
[0, 0, 1, 0, 0, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 0]]
len(convertedList) = 8124
convertedList[0] = [0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 1, 0]
```

Step 2: Take the Market basket data (*mbd.txt*) and convert it to weight matrix (*newdatabase.txt*).

Python Code: *conversion.py* -> convertToWeights ( )

Some Outputs:

```
len(database) = 119
len(database[0]) = 119
```

Step 3: Using the weight matrix (newdatabase.txt), create a graph and calculate its eigen value centrality and page rank, and then store them in a text file.

R Code: *compute.R*

This is actually the frequent itemsets of order 1.

Some Outputs:

```
[85, 88, 35, 91, 36, 38, 61, 65, 53, 23]
```

Step 4: Using the frequent itemsets of order 1 (*eigenvalue.txt / pageRank.txt*), we will generate frequent itemsets upto order ‘n’.

Python Code: *getMultiple.py*

Some Outputs:

Eigen Value Centrality:

```
In [94]: runfile('F:/51/getMultiple.py', wdir='F:/51')
Frequent ItemSets of order  1
[85, 88, 35, 91, 36, 38, 61, 65, 53, 23]
Frequent ItemSets of order  2
[[85, 88], [85, 35], [88, 35], [85, 91], [88, 91], [35, 91], [85, 36], [88, 36], [35,
36], [91, 36]]
Frequent ItemSets of order  3
[[85, 88, 35], [85, 88, 91], [85, 35, 91], [88, 35, 91], [85, 88, 36], [85, 35, 36],
[88, 35, 36], [85, 91, 36], [88, 91, 36], [35, 91, 36]]
Frequent ItemSets of order  4
[[85, 88, 35, 91], [85, 88, 35, 36], [85, 88, 91, 36], [85, 35, 91, 36], [88, 35, 91,
36], [85, 88, 35, 38], [85, 88, 35, 61], [85, 88, 91, 38], [85, 35, 91, 38], [85, 88,
35, 65]]
Frequent ItemSets of order  5
[[85, 88, 35, 91, 36], [85, 88, 35, 91, 38], [85, 88, 35, 91, 61], [85, 88, 35, 91, 65],
[85, 88, 35, 36, 38], [85, 88, 35, 91, 53], [85, 88, 35, 36, 61], [85, 88, 35, 91, 23],
[85, 88, 91, 36, 38], [85, 35, 91, 36, 38]]
```

Page Rank Centrality:

```
In [95]: runfile('F:/51/getMultiple.py', wdir='F:/51')
Frequent ItemSets of order  1
[85, 88, 35, 91, 36, 38, 61, 65, 23, 74]
Frequent ItemSets of order  2
[[85, 88], [85, 35], [88, 35], [85, 91], [88, 91], [35, 91], [85, 36], [88, 36], [35,
36], [91, 36]]
Frequent ItemSets of order  3
[[85, 88, 35], [85, 88, 91], [85, 35, 91], [88, 35, 91], [85, 88, 36], [85, 35, 36],
[88, 35, 36], [85, 91, 36], [88, 91, 36], [35, 91, 36]]
Frequent ItemSets of order  4
[[85, 88, 35, 91], [85, 88, 35, 36], [85, 88, 91, 36], [85, 35, 91, 36], [88, 35, 91,
36], [85, 88, 35, 38], [85, 88, 91, 38], [85, 35, 91, 38], [85, 88, 35, 61], [85, 88,
35, 65]]
Frequent ItemSets of order  5
[[85, 88, 35, 91, 36], [85, 88, 35, 91, 38], [85, 88, 35, 91, 61], [85, 88, 35, 36, 38],
[85, 88, 35, 91, 65], [85, 88, 35, 91, 23], [85, 88, 91, 36, 38], [85, 35, 91, 36, 38],
[85, 88, 35, 36, 61], [85, 88, 35, 91, 74]]
```

Step 5: The outputs observed in *step 4* are then compared with Apriori algorithm.

R code for Apriori: *apriori.R*

Output of Apriori: Support = 0.7

	items	support
1	{V36}	0.8385032
2	{V91}	0.9217134
3	{V35}	0.9741507
4	{V88}	0.9753816
5	{V85}	1.0000000
6	{V36,V91}	0.7956672
7	{V35,V36}	0.8126539
8	{V36,V88}	0.8148695
9	{V36,V85}	0.8385032
10	{V35,V91}	0.8980798
11	{V88,V91}	0.8970950
12	{V85,V91}	0.9217134
13	{V35,V88}	0.9731659
14	{V35,V85}	0.9741507
15	{V85,V88}	0.9753816
16	{V35,V36,V91}	0.7720335
17	{V36,V88,V91}	0.7720335
18	{V36,V85,V91}	0.7956672
19	{V35,V36,V88}	0.8126539
20	{V35,V36,V85}	0.8126539
21	{V36,V85,V88}	0.8148695
22	{V35,V88,V91}	0.8970950
23	{V35,V85,V91}	0.8980798
24	{V85,V88,V91}	0.8970950
25	{V35,V85,V88}	0.9731659
26	{V35,V36,V88,V91}	0.7720335
27	{V35,V36,V85,V91}	0.7720335
28	{V36,V85,V88,V91}	0.7720335
29	{V35,V36,V85,V88}	0.8126539
30	{V35,V85,V88,V91}	0.8970950
31	{V35,V36,V85,V88,V91}	0.7720335

To be noted: Apriori gives the top results of frequent itemsets of each order but in *ascending manner depending upon the support.* But our algorithm gives results in *descending manner* depending upon the variable *topX - number of top elements to be filtered in each order* and *maxOrder – upto what order should our algorithm generate the frequent itemsets.*

As the output shown in step 4:

We use topX=10 and maxOrder=5.

Our algorithm is flexible enough to take any for topX and maxOrder (10 and 5 are not fixed forever).