

## Machine Learning

### Digits Data Set – FP Growth

Use the digits dataset available under SKLearn. Consider the data corresponding to classes 0 and 1 only. Each pattern is a  $8 \times 8$  sized character where each value is an integer in the range 0 to 16. Convert it into a binary form by replacing a value below 8 by 0 and other values ( $\geq 8$ ) by 1. Use this binary data in the following tasks

Obtain the frequent itemsets, for each class using FP-growth, by viewing each binary pattern as a transaction of 64 items. Repeat this task with different minsup values in 0.1, 0.3, 0.5, 0.7.

#### SOLUTION

##### CODE:

Please find the code committed as [FPGrowth\\_DigitsDataSet\\_Impl.py](#)

- For this task, we need to view the dataset as a transaction of 64 items (8\*8 sized pattern)
- **TransactionEncoder** is used to transform the dataset into an array that is suitable for the FP growth apis.
- **Fpgrowth** is applied over the modified data frames and we find for various minsup values

##### RESULT:

The complete output is attached here.



Class\_0\_freq\_items.csv



Class\_1\_freq\_items.csv

Top 10 Result is shown for different minsup values.

For Class 0 and Class 1 :

Freq Items with **Min Sup = 0.3**

support itemsets

0	0.994382	(60)
1	0.983146	(4)
2	0.977528	(19)
3	0.977528	(43)
4	0.949438	(51)
5	0.949438	(61)
6	0.938202	(27)
7	0.938202	(12)
8	0.926966	(22)
9	0.926966	(54)

support itemsets

0	0.989011	(20)
1	0.945055	(37)
2	0.939560	(61)
3	0.934066	(53)
4	0.928571	(29)
5	0.928571	(28)
6	0.906593	(45)
7	0.890110	(21)

8 0.884615 (13)  
 9 0.824176 (12)  
 For Class 0 and Class 1 :  
 Freq Items with **Min Sup = 0.5**

support itemsets

0 0.994382 (60)  
 1 0.983146 (4)  
 2 0.977528 (19)  
 3 0.977528 (43)  
 4 0.949438 (51)  
 5 0.949438 (61)  
 6 0.938202 (27)  
 7 0.938202 (12)  
 8 0.926966 (22)  
 9 0.926966 (54)

support itemsets

0 0.989011 (20)  
 1 0.945055 (37)  
 2 0.939560 (61)  
 3 0.934066 (53)  
 4 0.928571 (28)  
 5 0.928571 (29)  
 6 0.906593 (45)  
 7 0.890110 (21)  
 8 0.884615 (13)  
 9 0.824176 (12)

For Class 0 and Class 1 :

Freq Items with **Min Sup = 0.7**

support itemsets

0 0.994382 (60)  
 1 0.983146 (4)  
 2 0.977528 (19)  
 3 0.977528 (43)  
 4 0.949438 (51)  
 5 0.949438 (61)  
 6 0.938202 (27)  
 7 0.938202 (12)  
 8 0.926966 (22)  
 9 0.926966 (54)

support itemsets

0 0.989011 (20)  
 1 0.945055 (37)  
 2 0.939560 (61)  
 3 0.934066 (53)  
 4 0.928571 (28)  
 5 0.928571 (29)  
 6 0.906593 (45)  
 7 0.890110 (21)  
 8 0.884615 (13)  
 9 0.824176 (12)

Complete output format:

Sample reference like this for class 0

support itemsets

0 0.994382 frozenset({60})

1	0.983146	frozenset({4})
2	0.977528	frozenset({19})
3	0.977528	frozenset({43})
4	0.949438	frozenset({51})
5	0.949438	frozenset({61})
6	0.938202	frozenset({27})
7	0.938202	frozenset({12})
8	0.926966	frozenset({22})
9	0.926966	frozenset({54})
10	0.91573	frozenset({35})
11	0.910112	frozenset({11})
12	0.865169	frozenset({5})
13	0.848315	frozenset({46})
14	0.842697	frozenset({14})
15	0.825843	frozenset({13})
16	0.730337	frozenset({53})
17	0.674157	frozenset({30})
18	0.617978	frozenset({39})
19	0.617978	frozenset({38})
20	0.719101	frozenset({52})
21	0.977528	frozenset({4, 60})
22	0.97191	frozenset({19, 60})
23	0.960674	frozenset({19, 4})
24	0.955056	frozenset({19, 4, 60})
25	0.97191	frozenset({43, 60})
26	0.960674	frozenset({43, 4})
27	0.955056	frozenset({43, 19})
28	0.955056	frozenset({43, 4, 60})
29	0.949438	frozenset({43, 19, 60})
30	0.938202	frozenset({43, 19, 4})
31	0.932584	frozenset({43, 19, 4, 60})
32	0.949438	frozenset({51, 60})
33	0.938202	frozenset({19, 51})
34	0.932584	frozenset({51, 4})

**INFERENCE/ANALYSIS:**

- **FP-Growth implementation generates the entire tree structure which is used for frequent itemset.**
- For different minsup values the output remains largely similar for the top 10 values in the range 0.3 to 0.7
- For smaller minsup values(0.1, 0.3) although the top 20 candidates are not varying much, but the time taken to generate the complete frequent itemsets, is very high.
- For relatively higher minsup values(o.5,0.7) the tree is generated faster.

**RESOURCES USED FOR THE ASSIGNMENT:**

<ul style="list-style-type: none"><li>• <b>Environment:</b> Anaconda, Jupyter notebook</li></ul>
<ul style="list-style-type: none"><li>• <b>Software :</b> Python <b>Python libraries/modules:</b> Pandas, Numpy, SkLearn etc</li></ul>