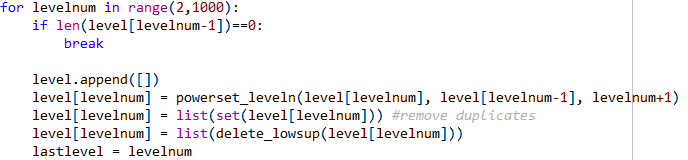
**DM Project1 Report: Association Rule**

**0 Environment:** anaconda spyder 3.2.8, python 3.6, cpu: amd 2600X

* 1. **Apriori implementation**:

Code is appended called “apriori\_1018.py”

Main code snapshot



Level: a two level set here represents “two items in one set”

*Powerset\_leveln*: make candidate power sets (combinations) according to last level of chosen sets.

*Delete\_lowsup*: From all candidate sets in this level, choose the correct sets that meets the requirement of minimum support.

* 1. **Apriori Analysis:**

Easy to implement, however time-consuming (especially when generating two-item sets, since they meets the minimum support easily thus the number of candidate set is huge )

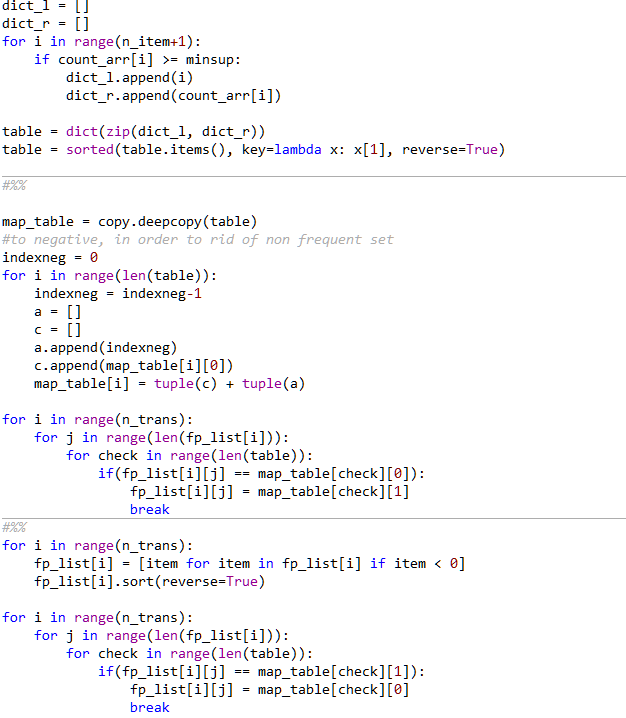
**2.1 Fp-Growth:**

Code is appended called “fpgrowth.py”

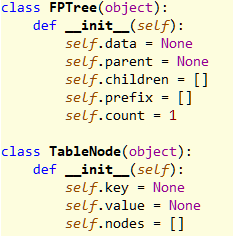
Main code snapshot

* Data preprocessing (count frequency + remove and sort data)

Using Dictionary as table to save frequent sets and its counts, and modify the original data, give negative values for frequent data, and remove the positive ones, which means infrequent data.



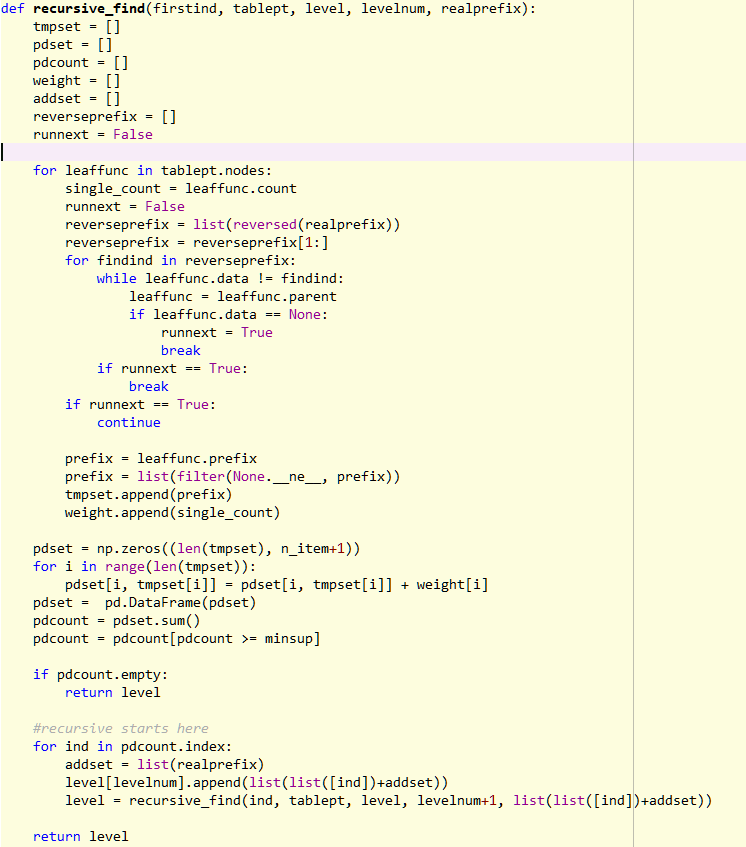
* Build tree:



*FPTree*: The FP Tree node, record prefix for each node for further usage.

*Tablenode*: Connect each FP Tree nodes that have same data.

* Find Frequent dataset:



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 |  |  | 4 | 3 |
|  | 1 | 5 | 4 | 3 |
| 2 |  | 5 |  | 3 |

Instead of building new trees to find frequent sets, use recursive finding method based on sorted prefix.

3

First, orange circle is table node, and blue circle is fp tree leaf nodes with data=3, count occurrence for each leaf node prefix, for the example above, if min support=2, we have sets (2,3), (5,3), (4,3), then, for (4,3) (least frequent item), we do the recursive find and with different prefix (brown area), if support meets, then new itemset (?, 4, 3) will be generated, no new itemset generate in the example, then we go to next one (5,3), and so on.

**2.2 FP-growth Analysis:**

Much more faster on generating frequent datasets since it doesn`t need to scan the original data for each candidate set, however it`s difficult to implement.

1. **Analysis on both methods:**

**Dataset:** “data3.data”: with 1000 transactions and 1000 items, average length of transaction = 100

In this case, we change the minimum support with confidence 0.6, and compare two methods, apriori and fp-growth (they have identical results but different performance)

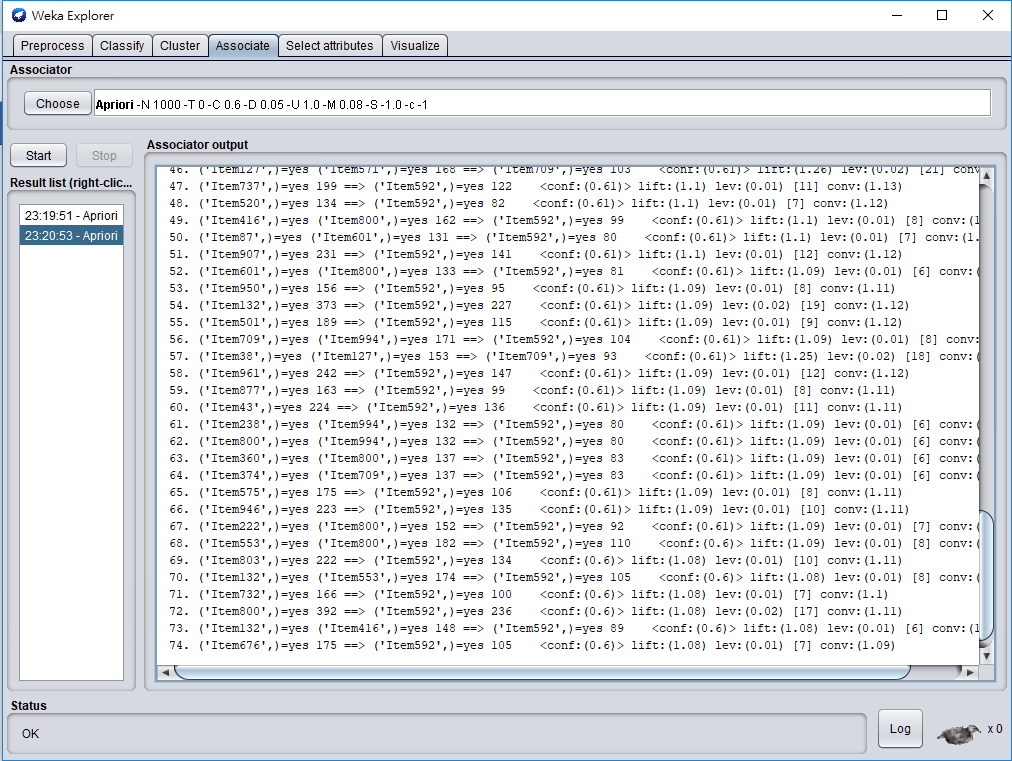
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| sup |  | .120 | .100 | .08 | .05 |
| level | **1** | 303 | 383 | 459 | 610 |
|  | **2** | 711 | 1715 | 4113 | 16134 |
|  | **3** | 4 | 38 | 322 | 8095 |
|  | **4** | 0 | 0 | 0 | 50 |
| rule |  | 12 | 27 | 74 | 954 |
| time | **apr** | 20 min | 1hr | 1.5 hr | >4 hr |
|  | **fp** | 12 s | 17 s | 25 s | 66 s |

We can observed that when minimum support is small, we will have more frequent sets for each level. Another observation is that fp-growth runs much more faster than apriori.

Next we run different confidence with same data and support = 0.8

|  |  |  |  |
| --- | --- | --- | --- |
| conf | 0.5 | 0.6 | 0.7 |
| rules | 757 | 74 | 1 |

**4. Rule Correctness:**

We check the correctness of the generated associate rules using WEKA, a tool for machine learning. 

Data input for WEKA is in .arrf data type, which need to run the preprocess data code “data\_pre\_weka.py” to have .csv first, then use the Tools -> arffViewer to transform to .arff.

74 rules with 0.08 support and 0.6 confidence, which exactly the same as the result of handmake apriori and fp-growth algorithm (saved to “result80.csv”)

**5. Conclusion**

In this project, we implement the two algorithms for dealing with associate rule problem, Apriori and Frequent Pattern Growth. Then use the tool WEKA to check correctness. Among these two algorithms, FP-growth may be more efficient for generating associate rules.