DATA MINING ASSIGNMENT - 02

Name	Contribution %	Contributions	Signature & Date
Austin Baade	33	Generated the Association Rules, Writing the report, testing code	Austin Baade 2/18/18
Venkata Krishna Mohan Sunkara	34	Implemented the Apriori algorithm for generating frequent item sets	Venkata Krishna Mohan Sunkara 2/18/18
ChunQian Moy	33	Formatted the Program, Plot the run time, compared results with Weka	ChunQian Moy 2/18/18

APRIORI ALGORITHM AND RULE GENERATION

Introduction

A dataset consists of many associations and the goal of this assignment is to generate the frequent itemsets among the data and use them to generate association rules based on the support and confidence metrics. We used the Apriori algorithm for frequent itemset generation with the minimum support metric. From, the frequent itemsets obtained we have generated rules which satisfy the minimum confidence metric.

General Design

The implementation is done in the Java programming language. It contains global variables to keep track of itemsets and rules. There are different functions for reading input, storing values into global variables, generating rules, and generating candidates. The main method reads in the command-line arguments, reads the path to the input file, parses the attributes/instances, generates frequent itemsets, generates rules, and outputs the result to the same directory as the input file. The output file is saved as “filenameoutput.txt”.

I. Input Extraction

Functions are declared to handle instances and attributes. We check if a line in the file starts with “@attribute” and save that into a list of attributes. Similarly for instances, we check if the line does not start with any “@” symbol and if it is not empty, then we add it to a list of instances.

II. Handling Classes and Labels

Classes in the input file are preceded with Class after the @attribute, So a function is written to get the class labels and store it in a string matrix. A function stores all the attributes as a hashmap with the index and its corresponding attribute name.

Now we convert the instances in to attribute names as:
I. If an attribute is present then that corresponding index value is used to obtain the attribute name.
II. If an attribute is not present then the corresponding index value is used to obtain the attribute name which is then appended with “-n”

III. Data Structures used

I. A List of strings is used to store each instance and while generating the subsets and power set we used hashset for storing the results.
II. A hashmap is used to store the index of an attribute along with its name. For storing the support counts of different candidate items generated in the process we use a hashmap.

IV. Determine frequent sets

```
// C_k: Candidate itemset of size k
// L_k: Frequent itemset of size k

L_1 = {frequent items};
for (k = 2; L_{k-1} != ∅; k++) {
   C_k = candidates from L_{k-1} with minimum support
   for each transaction t in database {
        C_t = subset(C_k, t); // subsets of t that are candidates
        for each candidate c ∈ C_t c.count ++;
   }
  L_k = {c ∈ C_k : c.count ≥ min_support)
}
return L=∪_k L_k;
```

Figure 1 Pseudocode for generating frequent itemset

To determine the frequent sets the program follows the above algorithm shown in Figure 1. There is a function to generate the candidates and to join them together. This does the majority of the work to determine their support count and save the frequent itemsets.

V. Association Rule Generation

For generating the rules, we follow the general algorithm shown in Figure 2.

for each frequent itemset I,
   generate all nonempty subsets of I;
for each nonempty subset s of I output the rule
   if $\frac{support(I)}{support(s)} \geq \min\_conf$ generate $s \rightarrow (I - s)$

Figure 2 Pseudocode for generating association rules

After we have all of the frequent itemsets, we go through each one and generate subsets. For example, the frequent set {A,B,C} would generate {A}, {B}, {C}, {A,B}, {A,C}, and {B,C}. This is done in nested for loops to generate all possible subsets. If the support of the frequent itemset divided by the support of the subset is greater or equal to the minimum confidence, then a rule is generated as {subset} → {Itemset - subset}.

Algorithm Runtime

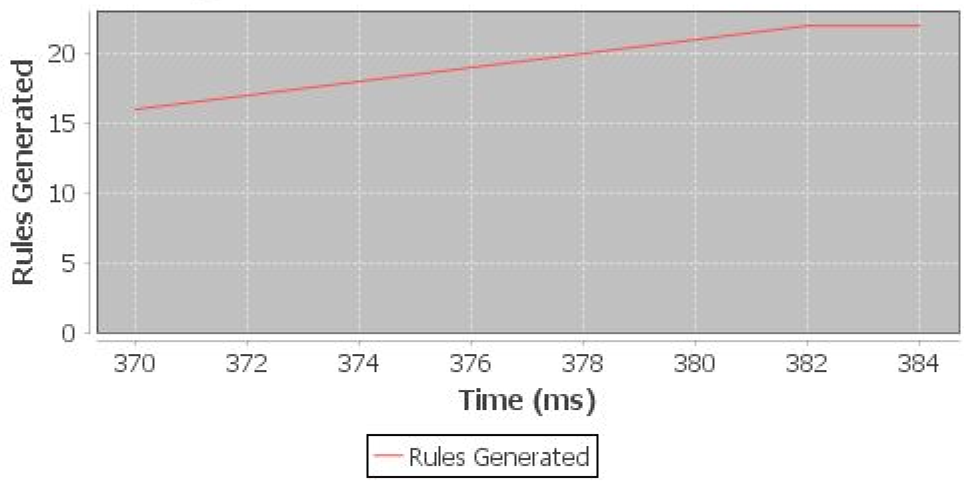


Figure 3 Plot of Run Time Against the Number of Rules Generated

The plot in Figure 3 shows the amount of time in milliseconds to generate the associate rules on our Apriori implementation. We ran our implementation on the vote.arff dataset with minimum support threshold of 0.55 (239 instances), and minimum confidence threshold of 0.5.

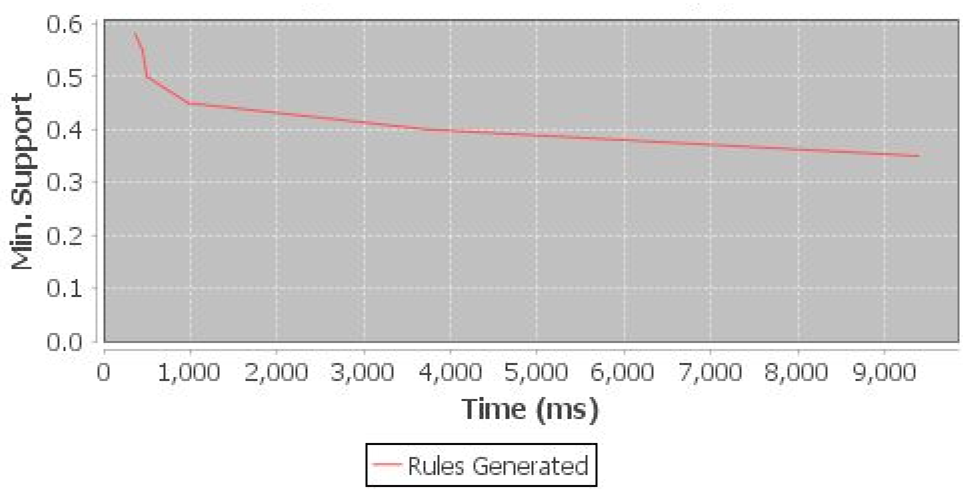


Figure 4 Plot of Run Time Against Different Min. Supports

Figure 4 shows how different values of minimum support threshold with the same minimum confidence threshold (0.5) affect the run time of our implementation. As we can observe, the lower the minimum support threshold, the more time it takes to generate the rules. In general, we can expect to get more frequent itemsets with decreasing the support count. Even in our

algorithm implementation we got more frequent itemsets with decreasing the support count which in turn resulted in generating more rules which affects the run time of the program.
We can also observe that there is a little difference in the run time for min supports of 0.6 and 0.5, this is due to a little difference in the number of frequent item sets generated for those higher support counts. But, as we keep on decreasing the support count more itemsets satisfy the min support threshold and hence more frequent itemsets are generated which in turn results in more association rules which is responsible for an increase in run time. That is the reason we observe a large difference in the run times for min supports of 0.4 and 0.3.

We can also notice that as the min support becomes 0 then all the itemsets are considered as frequent itemsets which is the same as brute force approach without any pruning. So, the run time at that min support becomes computationally expensive as the size of data set increases.

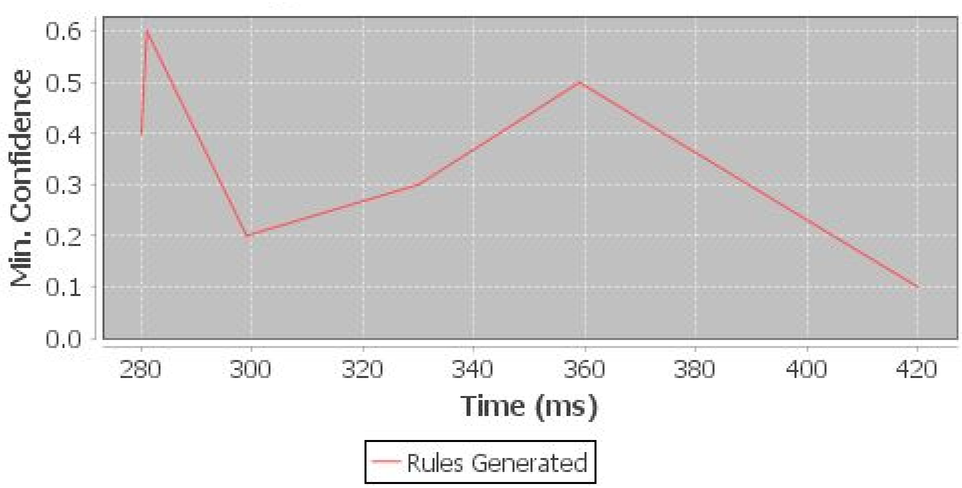


Figure 5 Line Graph of Time Against Different Min. Confidence

Figure 5 shows how different values of minimum confidence threshold with the same minimum support threshold (0.5) affect the run time of our implementation. The run time varies with different values of minimum confidence threshold. We even tried running it few more times but couldn’t manage to get a consistent graph pattern.

Weka Association Rules

To run association rules in Weka:

1. Open the data set in Weka.
2. Select the Associate tab in weka then the window as shown in fig.06 appears.
3. Now we can choose the algorithm we want by clicking on the choose tab
4. Since we want to use the apriori algorithm, we click on the tab, then a window as shown in the Fig. 7 appears.

5. We can set the upperBoundMinSupport to desired value. Ensure that the metricType is confidence and select desired minMetric value that will treated as the minimum confidence threshold. (Fig. 7)

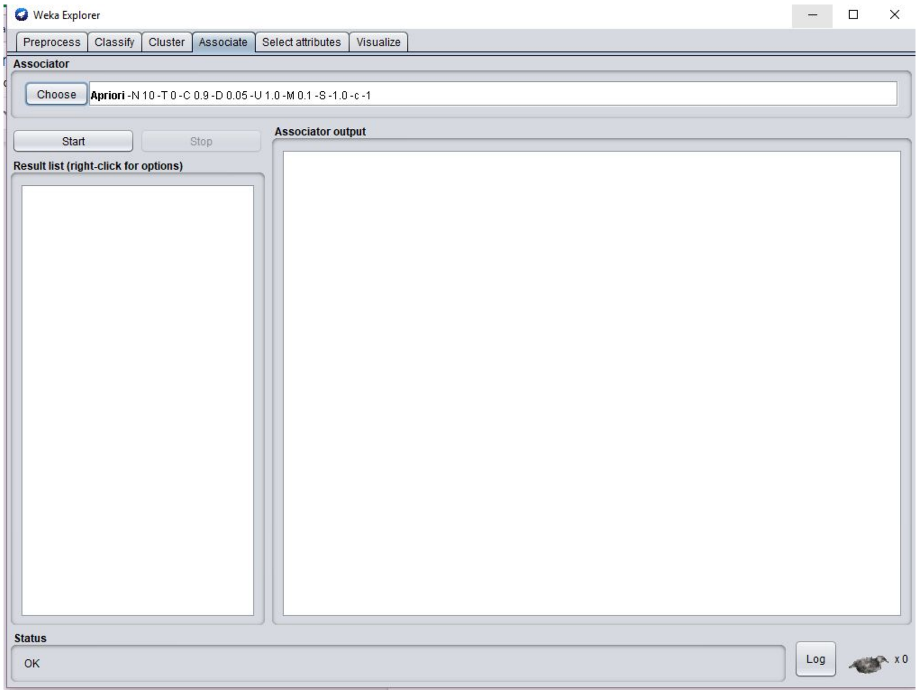


Figure 6 Associator in Weka

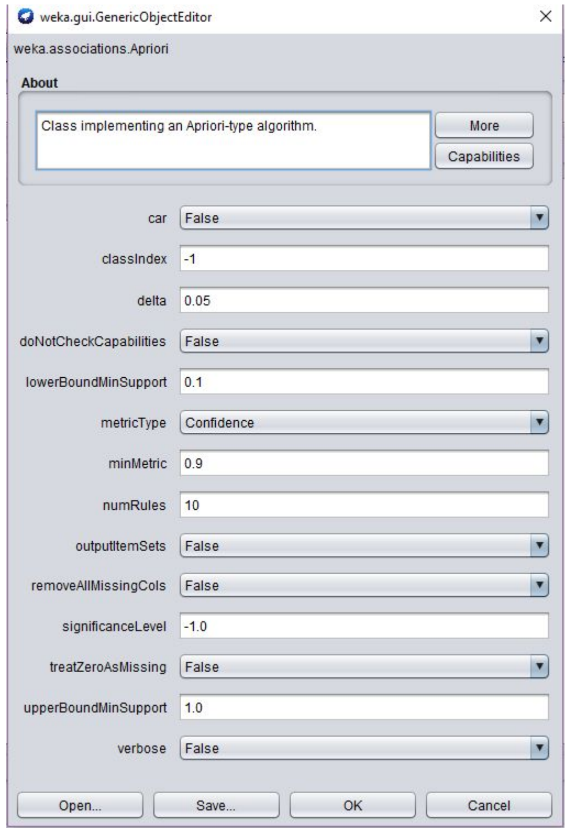


Figure 7 Setting Up Apriori in Weka

Comparison

Running the Apriori algorithm on the vote.arff dataset in weka with a minimum support threshold of 0.55 (239 instances) and minimum confidence threshold of 0.5 provided us 22 associated rules. (Fig. 8). Whereas, withholding the same dataset,minimum support and confidence our java implementation also resulted in 22 associated rules as well (Fig. 9). Our implementation were able to produce 100% accuracy as compared to Weka.

Moreover, running Apriori with higher minimum confidence threshold (0.8) and remain both the same dataset and the same minimum support threshold (0.55), weka was able to produce 14 associated rules (Fig. 10) and our java implementation produced 14 associated rules (Fig. 11).

However, when running the Apriori with lower minimum support value (0.35) using the same dataset and helding the minimum confidence to be 0.5, weka produced 422 associated rules (Fig. 12) whereas our Java implementation produced 19689 associated rules (Fig. 13). The difference is due to our implementation generating all possible rules whereas weka doesn't. For example, a,c → b is the superset of rule including a → b and c → b, our implementation will count them as 3 different rules whereas weka will only consider a,c =>b as a single rule.

Therefore as the number of subsets get higher, the number of rules generated got higher in our implementation.

We also think that our implementation generates more rules because it doesn't check for the cases where the rule is already present in some other format. For example, if a rule like a,b → c,d exists then other rules such as b,a → d,c and a,b → c,d are exactly the same as previous one.

The output generated by our implementation is not in the exact same format as the weka output. Our output just displays the rule along with the support of the frequent itemset from which it is generated along with the confidence for that rule.

Note: For our implementation, if the dataset consists of 2 class labels then the first label as described in the file is treated as class and the other is considered as class-n. For example, in the vote.arff data set it consists of 2 class labels as democrat and republican. So, our program considers democrat as class and republican as class-n. Our program works only for the data set consisting as 'y' and 'n' and only two class labels.

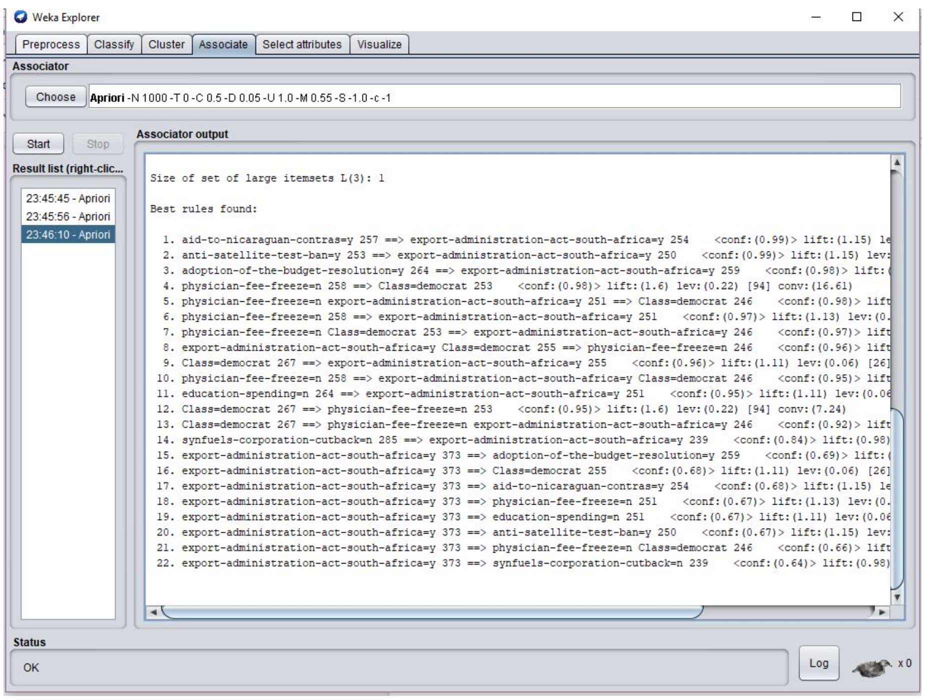
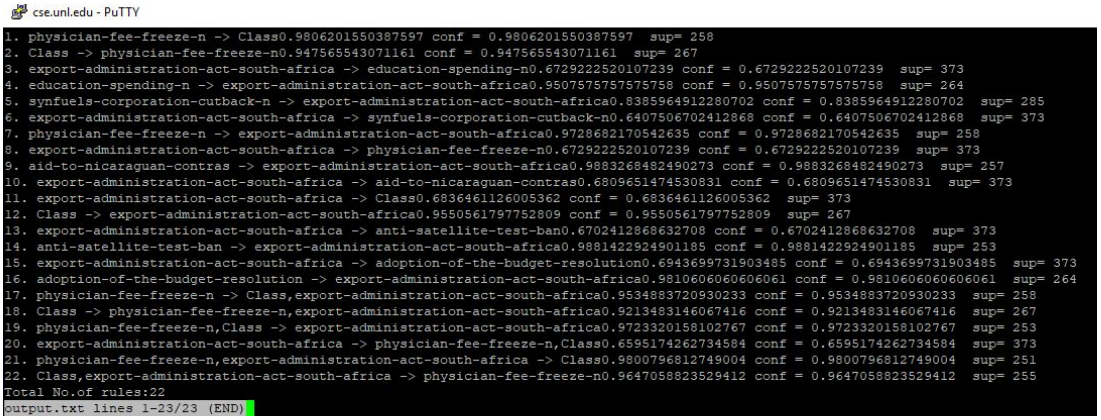
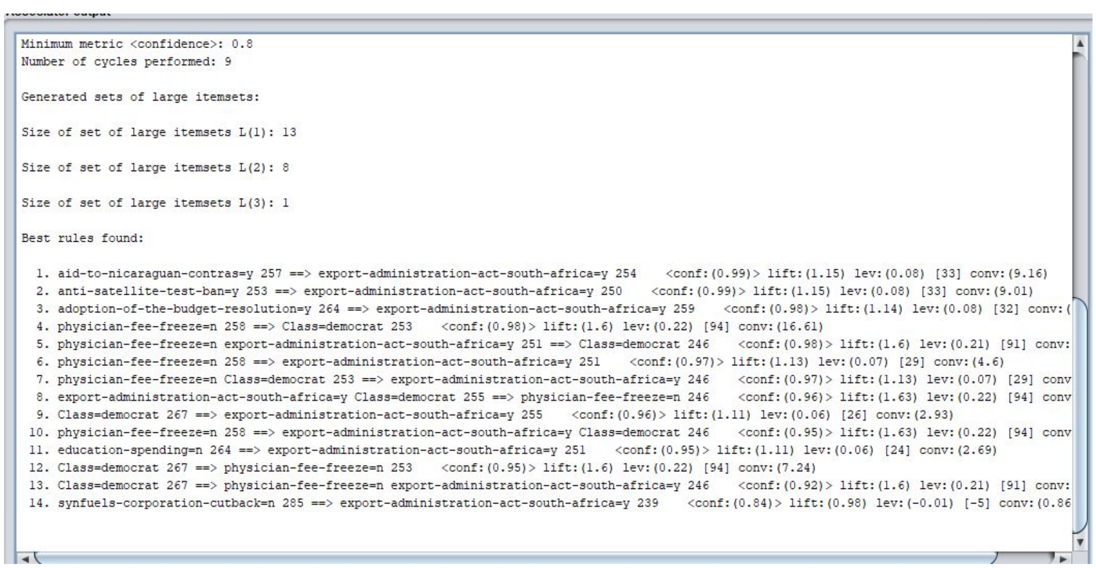


Figure 8 Apriori with 0.55 min. Support (Weka)



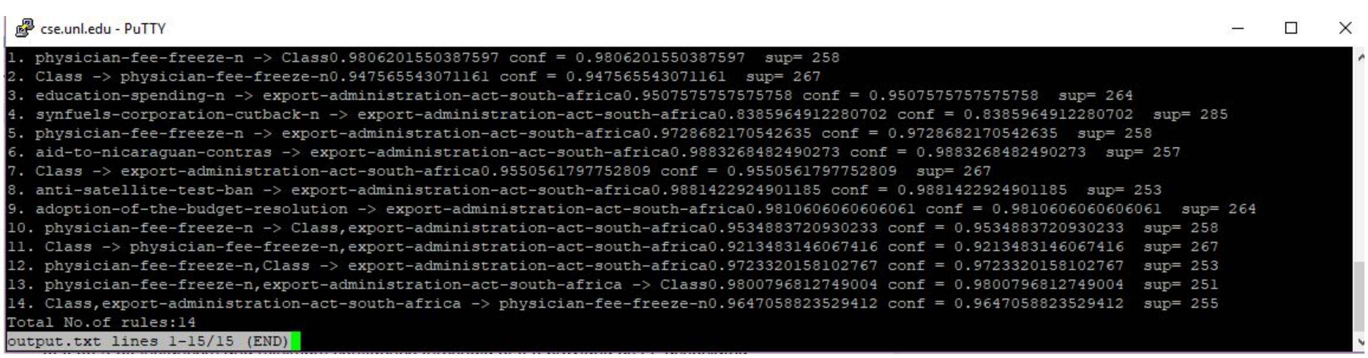
```
1. physician-fee-freeze-n -> Class0.9806201550387597 conf = 0.9806201550387597  sup= 258
2. Class -> physician-fee-freeze-n0.947565543071161 conf = 0.947565543071161  sup= 267
3. export-administration-act-south-africa -> education-spending-n0.6729222520107239 conf = 0.6729222520107239  sup= 373
4. education-spending-n -> export-administration-act-south-africa0.9507575757575758 conf = 0.9507575757575758  sup= 264
5. synfuels-corporation-cutback-n -> export-administration-act-south-africa0.8385964912280702 conf = 0.8385964912280702  sup= 285
6. export-administration-act-south-africa -> synfuels-corporation-cutback-n0.6407506702412868 conf = 0.6407506702412868  sup= 373
7. physician-fee-freeze-n -> export-administration-act-south-africa0.9728682170542635 conf = 0.9728682170542635  sup= 258
8. export-administration-act-south-africa -> physician-fee-freeze-n0.6729222520107239 conf = 0.6729222520107239  sup= 373
9. aid-to-nicaraguan-contras -> export-administration-act-south-africa0.9883268482490273 conf = 0.9883268482490273  sup= 257
10. export-administration-act-south-africa -> aid-to-nicaraguan-contras0.6809651474530831 conf = 0.6809651474530831  sup= 373
11. export-administration-act-south-africa -> Class0.6836461126005362 conf = 0.6836461126005362  sup= 373
12. Class -> export-administration-act-south-africa0.9550561797752809 conf = 0.9550561797752809  sup= 267
13. export-administration-act-south-africa -> anti-satellite-test-ban0.6702412868632708 conf = 0.6702412868632708  sup= 373
14. anti-satellite-test-ban -> export-administration-act-south-africa0.9881422924901185 conf = 0.9881422924901185  sup= 253
15. export-administration-act-south-africa -> adoption-of-the-budget-resolution0.6943699731903485 conf = 0.6943699731903485  sup= 373
16. adoption-of-the-budget-resolution -> export-administration-act-south-africa0.9810606060606061 conf = 0.9810606060606061  sup= 264
17. physician-fee-freeze-n -> Class,export-administration-act-south-africa0.9534883720930233 conf = 0.9534883720930233  sup= 258
18. Class -> physician-fee-freeze-n,export-administration-act-south-africa0.9213483146067416 conf = 0.9213483146067416  sup= 267
19. physician-fee-freeze-n,Class -> export-administration-act-south-africa0.9723320158102767 conf = 0.9723320158102767  sup= 253
20. export-administration-act-south-africa -> physician-fee-freeze-n,Class0.6595174262734584 conf = 0.6595174262734584  sup= 373
21. physician-fee-freeze-n,export-administration-act-south-africa -> Class0.9800796812749004 conf = 0.9800796812749004  sup= 251
22. Class,export-administration-act-south-africa -> physician-fee-freeze-n0.9647058823529412 conf = 0.9647058823529412  sup= 255
Total No.of rules:22
output.txt lines 1-23/23 (END)
```

Figure 9 Apriori with 0.55 min. Support (Java)



```
Minimum metric <confidence>: 0.8
Number of cycles performed: 9

Generated sets of large itemsets:

Size of set of large itemsets L(1): 13

Size of set of large itemsets L(2): 8

Size of set of large itemsets L(3): 1

Best rules found:

  1. aid-to-nicaraguan-contras=y 257 ==> export-administration-act-south-africa=y 254    <conf:(0.99)> lift:(1.15) lev:(0.08) [33] conv:(9.16)
  2. anti-satellite-test-ban=y 253 ==> export-administration-act-south-africa=y 250    <conf:(0.99)> lift:(1.15) lev:(0.08) [33] conv:(9.01)
  3. adoption-of-the-budget-resolution=y 264 ==> export-administration-act-south-africa=y 259    <conf:(0.98)> lift:(1.14) lev:(0.08) [32] conv:(
  4. physician-fee-freeze=n 258 ==> Class=democrat 253    <conf:(0.98)> lift:(1.6) lev:(0.22) [94] conv:(16.61)
  5. physician-fee-freeze=n export-administration-act-south-africa=y 251 ==> Class=democrat 246    <conf:(0.98)> lift:(1.6) lev:(0.21) [91] conv:
  6. physician-fee-freeze=n 258 ==> export-administration-act-south-africa=y 251    <conf:(0.97)> lift:(1.13) lev:(0.07) [29] conv:(4.6)
  7. physician-fee-freeze=n Class=democrat 253 ==> export-administration-act-south-africa=y 246    <conf:(0.97)> lift:(1.13) lev:(0.07) [29] conv
  8. export-administration-act-south-africa=y Class=democrat 255 ==> physician-fee-freeze=n 246    <conf:(0.96)> lift:(1.63) lev:(0.22) [94] conv
  9. Class=democrat 267 ==> export-administration-act-south-africa=y 255    <conf:(0.96)> lift:(1.11) lev:(0.06) [26] conv:(2.93)
 10. physician-fee-freeze=n 258 ==> export-administration-act-south-africa=y Class=democrat 246    <conf:(0.95)> lift:(1.63) lev:(0.22) [94] conv
 11. education-spending=n 264 ==> export-administration-act-south-africa=y 251    <conf:(0.95)> lift:(1.11) lev:(0.06) [24] conv:(2.69)
 12. Class=democrat 267 ==> physician-fee-freeze=n 253    <conf:(0.95)> lift:(1.6) lev:(0.22) [94] conv:(7.24)
 13. Class=democrat 267 ==> physician-fee-freeze=n export-administration-act-south-africa=y 246    <conf:(0.92)> lift:(1.6) lev:(0.21) [91] conv:
 14. synfuels-corporation-cutback=n 285 ==> export-administration-act-south-africa=y 239    <conf:(0.84)> lift:(0.98) lev:(-0.01) [-5] conv:(0.86
```

Figure 10 Apriori with 0.8 min.Confidence (Weka)



```
1. physician-fee-freeze-n -> Class0.9806201550387597 conf = 0.9806201550387597  sup= 258
2. Class -> physician-fee-freeze-n0.947565543071161 conf = 0.947565543071161  sup= 267
3. education-spending-n -> export-administration-act-south-africa0.9507575757575758 conf = 0.9507575757575758  sup= 264
4. synfuels-corporation-cutback-n -> export-administration-act-south-africa0.8385964912280702 conf = 0.8385964912280702  sup= 285
5. physician-fee-freeze-n -> export-administration-act-south-africa0.9728682170542635 conf = 0.9728682170542635  sup= 258
6. aid-to-nicaraguan-contras -> export-administration-act-south-africa0.9883268482490273 conf = 0.9883268482490273  sup= 257
7. Class -> export-administration-act-south-africa0.9550561797752809 conf = 0.9550561797752809  sup= 267
8. anti-satellite-test-ban -> export-administration-act-south-africa0.9881422924901185 conf = 0.9881422924901185  sup= 253
9. adoption-of-the-budget-resolution -> export-administration-act-south-africa0.9810606060606061 conf = 0.9810606060606061  sup= 264
10. physician-fee-freeze-n -> Class,export-administration-act-south-africa0.9534883720930233 conf = 0.9534883720930233  sup= 258
11. Class -> physician-fee-freeze-n,export-administration-act-south-africa0.9213483146067416 conf = 0.9213483146067416  sup= 267
12. physician-fee-freeze-n,Class -> export-administration-act-south-africa0.9723320158102767 conf = 0.9723320158102767  sup= 253
13. physician-fee-freeze-n,export-administration-act-south-africa -> Class0.9800796812749004 conf = 0.9800796812749004  sup= 251
14. Class,export-administration-act-south-africa -> physician-fee-freeze-n0.9647058823529412 conf = 0.9647058823529412  sup= 255
Total No.of rules:14
output.txt lines 1-15/15 (END)
```

Figure 11 Apriori with 0.8 min. Confidence (Java)

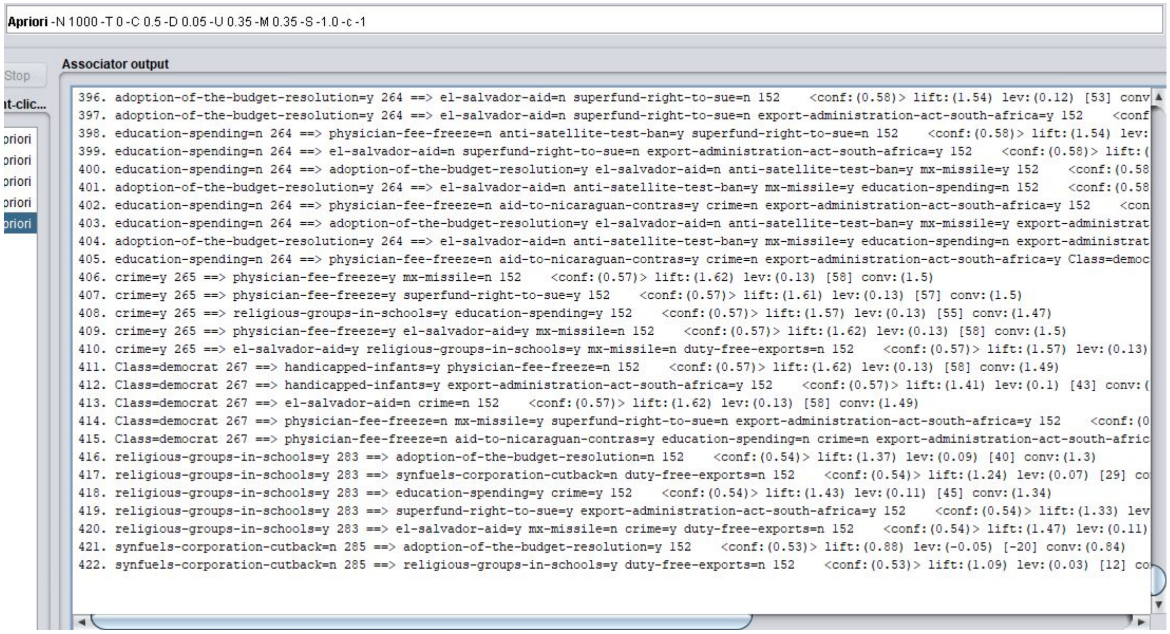


Figure 12 Apriori with 0.35 min. Support(Weka)

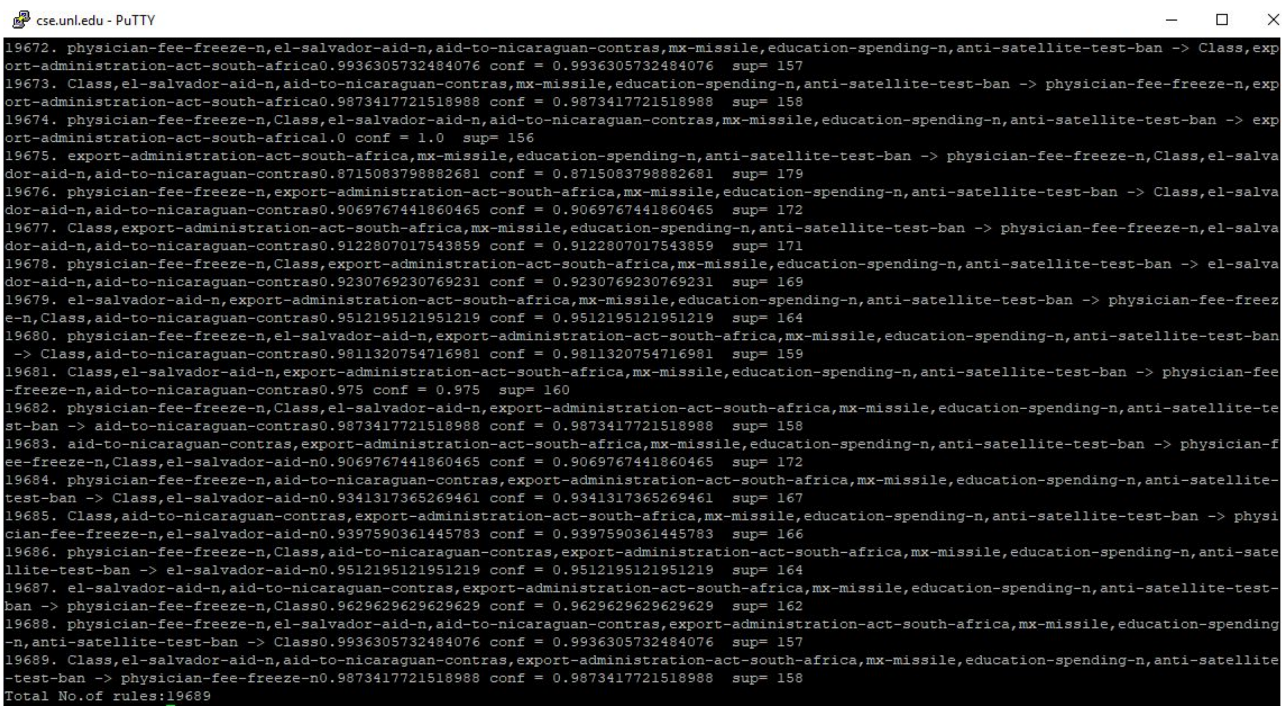


Figure 13 Apriori with 0.35 min. Support(Java)