Apriori implementation using Python and Matlab

Luting Chen 50133507, Yuze Liu 50207903, Vicky Zheng 50037709

Introduction

The apriori algorithm is used to determine association rules. In this homework, we implemented the apriori algorithm by using Matlab. We also implemented it using python. The dataset we tested our implementation on is gene expression data. The applications of this are straight forward: with association rules we can see how genes relate to each other and what the relation between genes can tell us about the data.

Data Preprocesing

The data provided is a .txt file that contains a 100 by 102 matrix. There are 100 samples each corresponding to the genes of a patient with a disease. The diseases are: ALL, AML, Breast Cancer and Colon Cancer. In order to optimize our program, we converted each of these diseases into integers: 1,2,3,4 respectively. The data set also gives us a sequence of 100 genes labeled as either up or down. Each gene at index i is treated as 1 if it is up and 0 if it is down. We did this to optimize our program by having fewer string comparsions since integer comparisons are much faster.

Implementation

For this homework, we choose to use Matlab because our input data is a matrix. Matlab has helpful functions for matrix manipulation like determining whether a set of numbers is in a row of the matrix. It also has helpful functions such as the union of two sets which helped us do the pruning necessary for an efficient apriori algorithm. We also implemented the same algorithm using Python independently. We implemented it twice independently to verify that we have the correct answers. In our implementation, we enumerate through all possible tuples where a tuple can be comprised of genes or genes and diseases. We start at the shortest length of tuples, and work our way up. We based our implementation off the diagram provided in class which is shown in Fig. 1.

The total number of possible tuples is greater than O(n!) where n is the number of attributes. In order to reduce our search space, we use pruning. Pruning significantly reduces our runtime by reducing the number of tuples we have to explore. This is because we expliot the fact that a superset of an infrequent set must also be infrequent. In Fig. 1, $BCD => A$ was infrequent therefore all of it's supersets are also infrequent so we don't have to explore it.

For part 2, we generated all possible rules for all item sets, calculated the confidence for each of them and then counted the ones that meet our confidence requirement of .6 and other rule requirements. We prefiltered the rules that we generated by picking from the item sets that had a support of at least .5. Another observation that we made was that the only way that the confidence could be be lower than .6 is if the frequency of one of the subsets was greater than 83. This is because we know that if an item is in the frequent set, it has at least a frequency of 50. The only way for confidence to be lower than .6 is if frequency of subset is greater or equal to 83 because $\frac{50}{83} = .6$. We found that all of our frequencies were in the range $50 \leq freq < 83$. This simplified our counting.

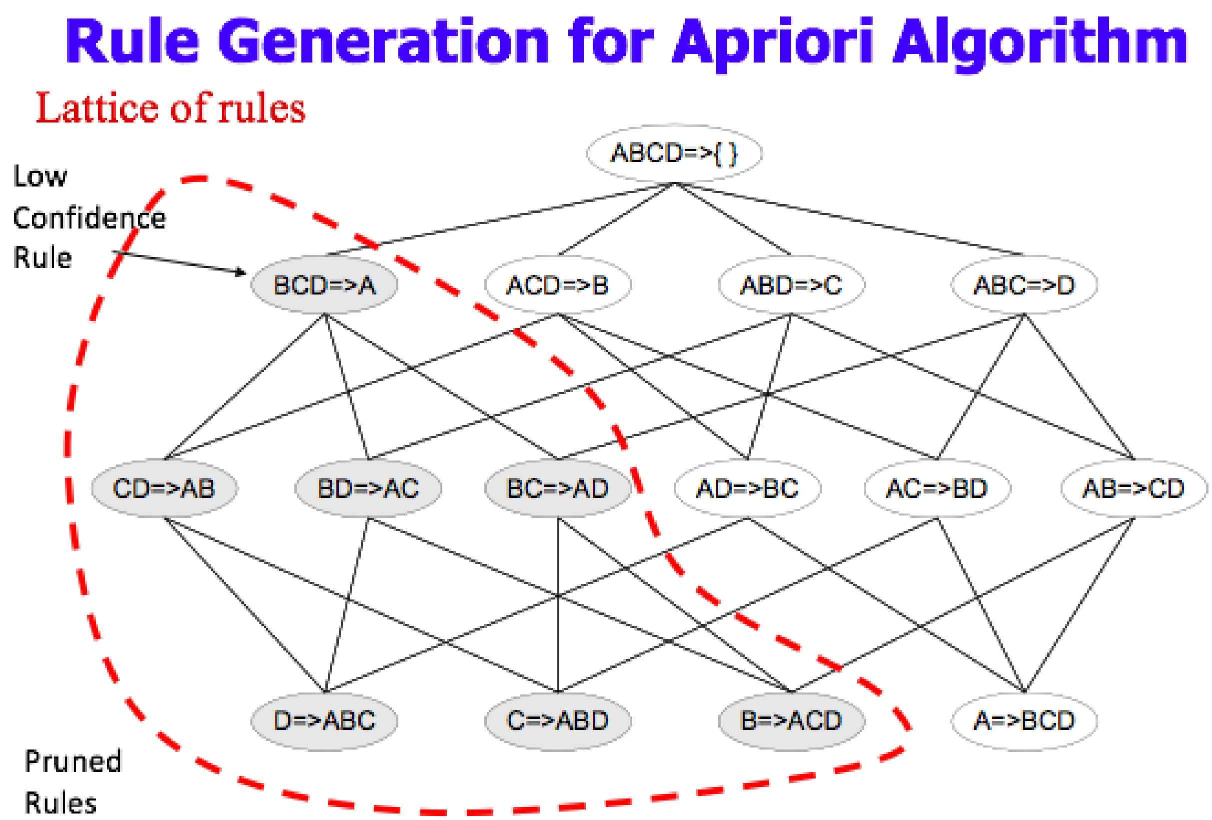

Figure 1: Apriori Pruning Search Space

Results from Part 1

With a support level of 30%, we obtain the values:

196 of length 1 frequent item sets.

5340 of length 2 frequent item sets.

5287 of length 3 frequent item sets.

1518 of length 4 frequent item sets.

438 of length 5 frequent item sets.

88 of length 6 frequent item sets.

11 of length 7 frequent item sets.

1 of length 8 frequent item sets.

With a support level of 40%, we obtain the values:

167 of length 1 frequent item sets.

753 of length 2 frequent item sets.

149 of length 3 frequent item sets.

7 of length 4 frequent item sets.

1 of length 5 frequent item sets.

With a support level of 50%, we obtain the values:

109 of length 1 frequent item sets.

63 of length 2 frequent item sets.

2 of length 3 frequent item sets.

With a support level of 60%, we obtain the values:

31 of length 1 frequent item sets.

2 of length 2 frequent item sets.

With a support level of 70%, we obtain the values:

7 of length 1 frequent item sets.

Results from Part 2

The results from Part 2 were obtained by combining our results from Part 1 and combining it with the counting method described in piazza post 113. Confidence is calculated from the equation:

$$c = \frac{\sigma(HEAD)}{\sigma(BODY)}$$

1. RULE HAS ANY OF G6_UP

   Result: 10

2. RULE HAS 1 OF G1_UP

   Result: 14

3. RULE HAS 1 OF (G1_UP, G10_DOWN)

   Result: 26

4. BODY HAS ANY OF G6_UP

   Result: 5

5. BODY HAS NONE OF G72_UP

   Result: 124

6. BODY HAS 1 OF (G1_UP, G10_DOWN)

   Result: 15

7. HEAD HAS ANY OF G6_UP

Result: 5

8. HEAD HAS NONE OF (G1_UP, G6_UP)

   Result: 126

9. HEAD HAS 1 OF (G6_UP, G8_UP)

   Result: 6

10. RULE HAS 1 OF (G1_UP, G6_UP, G72_UP)

    Result: 48

11. RULE HAS ANY OF (G1_UP, G6_UP, G72_UP)

    Result: 50

12. SIZE OF RULE >= 3

    Result: 6

13. SIZE OF BODY >= 2

    Result: 6

14. SIZE OF HEAD >= 2

    Result: 6

15. BODY HAS ANY OF G1_UP AND HEAD HAS 1 OF G59_UP

    Result: 1

16. BODY HAS ANY OF G1_UP OR HEAD HAS 1 OF G6_UP

    Result: 12

17. BODY HAS 1 OF G1_UP OR HEAD HAS 2 OF G6_UP

    Result: 7

18. HEAD HAS 1 OF G1_UP AND BODY HAS 0 OF DISEASE

    Result: 7

19. HEAD HAS 1 OF DISEASE OR RULE HAS 1 OF (G72_UP, G96_DOWN)

    Result: 24

20. BODY HAS 1 of (G59_UP, G96_DOWN) AND SIZE OF RULE >= 3

    Result: 6

Conclusion

In this homework, we implemented the apriori algorithm to find association rules. With our implementation, we calculated the number of frequent sets for various lengths for support levels ranging from 0.3-0.7. With the frequent item sets, we are able to calculate the results from the queries given to us on piazza. We are able to do this because we have already calculated the frequent item sets for Support = .5 and we can easily calculate the confidence with the equation given to us to see if it meets the threshold of .6.

References and Notes

[1] Jing Gao *Association1 Slides*.