HW1 Report

電機碩二
r06921048
李友岐

(1) Implementation and observation:

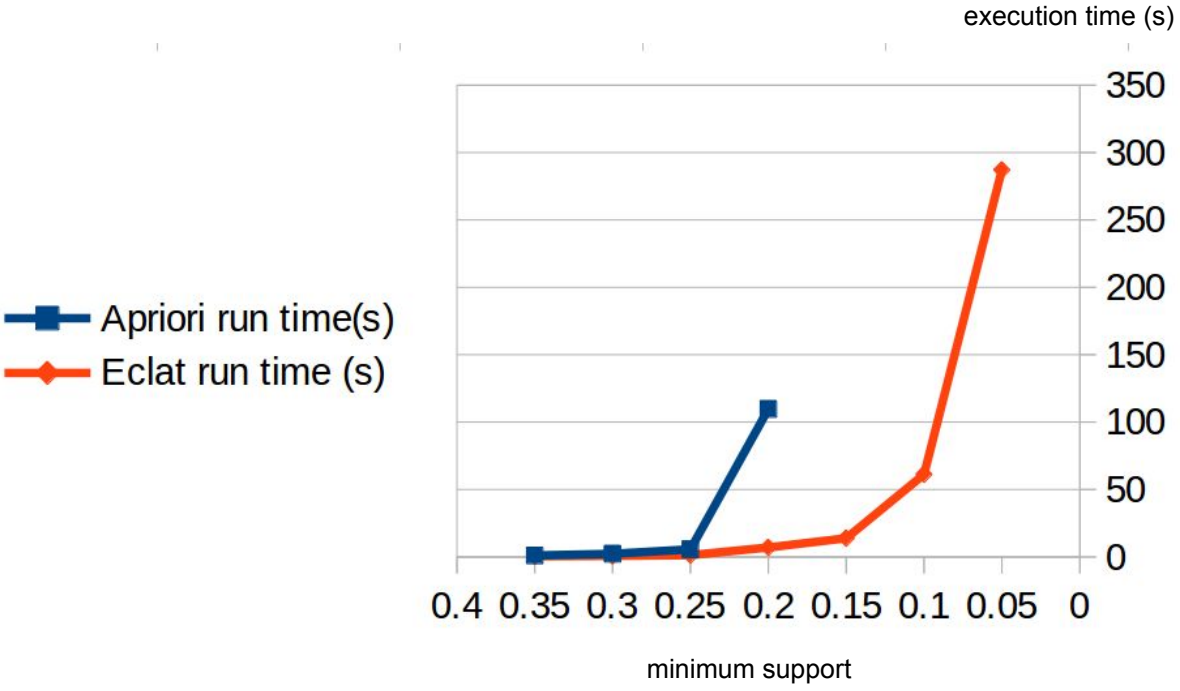
Apriori:

I implemented it by Python and I chose Python's built-in container set as the main data structure. Each row of data.txt represents a set. Besides, every itemset we generate is stored as a set. Therefore, we can directly use the operator of set during the process. if we want to generate L_k by two L_{k-1} (A, B), then the result itemset is the union of A and B (A | B). Note that we should check the intersection of two itemset (A & B) first . For the generation of L_k, the length of the intersection of two L_{k-1} must equal to (k-2). Moreover, we need to find k itemsets(L_{k-1}) totally for this L_k or this L_k is not frequent. For example, if we have two L_3 {1, 2, 3} and {1, 2, 4}, then we first check the length of their intersection, which is eligible. Thus, their union might be a L_4. To make sure it is possible to be frequent, we need to find {1, 3, 4} and {2, 3, 4} by using intersection of set to check whether this L_3 is what we want. To compute the support number of itemset, we just check how many row sets are the superset of this itemset. To improve the performance, we should remove the redundant process. For example, if we generate {1, 2, 3, 4} by {1, 2, 3} and {1, 2, 4} before, then we don't need to check any combination of each two of {1, 2, 3}, {1, 2, 4}, {1, 3, 4}, {2, 3, 4}. Since we already know the result. This method can save a lot of time. However, Apriori algorithm is less efficient than Eclat, because it takes too much time to generate L_k from trying different combinations of L_{k-1} .

Eclat:

This part is also implemented in Python and Python's built-in container set is still the main data structure. In Eclat algorithm, however, we store the information with vertical data layout. Each item has a data set to record which rows contain it. For example, if a number shows up in row 2, row 3 and row 5, then its data set is equal to {2, 3, 5}. If we want to check whether the new itemset generated from these two items is frequent, we can just use the intersection of the data set of these two items. The support number of new itemset is the length of its data set, which is the intersection of the data set of these two items. For example, item 1 and item 2 are included in {1, 2, 3} and {2, 3, 4}, respectively. Then itemset [1, 2] is included in the intersection of {1, 2, 3} and {2, 3, 4}, which is {2, 3}. Therefore, the support number of [1, 2] is the element length of {2, 3}, which is 2. To expand the itemset, we define a function taking current itemset and item index as parameters. If the new itemset is frequent, then we keep calling this function to expand itemset until it becomes not frequent. If the new itemset is not frequent, then there is no need to expand it. This algorithm doesn't have redundant process, which means all possible itemsets are checked by one time at most. As a result, the performance of Eclat using vertical bitvector is way better than Apriori.

(2) Execution time versus minimum support



Minimum support	Apriori run time (s)	Eclat run time (s)
0.35	1.10	0.53
0.30	2.28	0.78
0.25	5.62	1.38
0.20	109.72	7.06
0.15		13.89
0.10		61.23
0.05		287.00