Assignment 2: Apriori Rule Generation

TDT4300 Data Warehousing and Data Mining

1. Apriori Frequent Itemset Generation

1. Brute-force method

The brute-force method generates all possible candidate frequent itemsets while traversing the lattice, and only starts pruning when all candidates have been generated. This is the simplest approach, but requires by far the most computation of all the methods, and does not scale well with larger sets of data. This becomes clear when the brute-force method is unable to complete within reasonable time (several minutes) for the large data set, and in fact gets "stuck" as early as level 2.

2. F_{k-1} x F_1 method

This method creates new candidate itemsets of size *k* from the frequent itemsets of size *k-1*, by combining them with the frequent 1-itemsets. We do this because we know that an itemset containing an infrequent 1-itemset (single item) won't be a frequent itemset. This results in there being generated a much lower amount of candidates, thus leading to a much shorter run time.

On average, this method ran 86 % faster than the brute-force method on the small data set, and some huge but unknown amount faster on the large data set.

3. F_{k-1} x F_{k-1} method

This method combines frequent item sets of size *k-1* to create the candidate frequent itemsets of size *k*, but only those where the first *k-2* items of the two sets are the same (this requires the sets to be sorted lexicographically). This ensures that we don't get multiples sets that are actually the same – e.g. {bread, egg, ham, milk} and {bread, ham, milk, egg}. For the first two levels (k=1 and k=2), this method is the same as the previous, but beyond that, we can see significant improvements – although not as large as those of the previous method compared to the brute-force method.

On average, this method ran 31 % faster than the previous method, and 90% faster than the brute-force method on the small data set, and 35 % faster than the previous method on the large data set.

4. Pruning and About the Numbers

We ran each of the methods five times while timing each run, and used these numbers as a basis for our analysis. The algorithms were run on a computer with an i5 quadcore processor at 1.7GHz per core.

The following tables show the number of candidates generated vs. candidates kept at each level (each value of *k*) for each of the methods. The first table is for the small data set, while the second is for the large data set.

Small data set

	Brute-force		**F_{k-1} x F_1**		**F_{k-1} x F_{k-1}**	
	Generated	**Kept**	**Generated**	**Kept**	**Generated**	**Kept**
Level 1	*6*	*6*	*6*	*4*	*6*	*4*
Level 2	*15*	*15*	*6*	*4*	*6*	*4*
Level 3	*150*	*150*	*4*	*1*	*1*	*1*
Level 4	*300*	*300*	*1*	*0*	*0*	*0*
Level 5	*150*	*150*	*N/A*	*N/A*	*N/A*	*N/A*

Large data set

	Brute-force		**F_{k-1} x F_1**		**F_{k-1} x F_{k-1}**	
	Generated	**Kept**	**Generated**	**Kept**	**Generated**	**Kept**
Level 1	*122*	*122*	*122*	*17*	*122*	*17*
Level 2	*N/A*	*N/A*	*136*	*15*	*136*	*15*
Level 3	*N/A*	*N/A*	*182*	*2*	*18*	*0*

We see that already on a small data set, there is noticeable difference between the brute-force method and the methods that prune as they go. When we get a bigger, realistic data set, the brute-force method simply isn't viable. With large data sets, we want efficient algorithms to avoid unnecessary computation, such as the Fk-1 x Fk-1 method.

2. Apriori Rule Generation

To calculate the confidence for an association, you look at the support for the itemsets on either side of the rule (the *antecedent* and the *consequent*). By dividing the support for the left hand side (the *antecedent*) by the support for the right hand side (the *consequent*), you get the confidence value for the rule. Below is a table with the number of rules generated after each of the previous algorithms.

	Brute-force	**F_{k-1} x F_1**	**F_{k-1} x F_{k-1}**
Small data set	*180*	*11*	*11*
Large data set	*N/A*	*30*	*30*

By comparing the confidence value of each rule to a minimum confidence value (*minconf*), we can select a set of rules to keep and use in further predictions. We did not do this in our implementation, but it's quite obvious that not all the rules generated after the brute-force algorithm are as useful as some of them (too low confidence), and should be discarded in actual use.