Data Mining
Assignment 2

Sahar and Safoora

We approached the implementation of the optimized version of the Apriori algorithm by dividing the problem into two parts; first, generating the candidate sets (done by Sahar), and second, counting the supports for each candidate inside the dataset (done by Safoora). We decided to use C++, which is faster than our other alternatives in terms of execution time.

For the first, part after counting the support for one-item item sets and removing the sets with lower support than the minimum support, we sort and concatenate the remaining sets to generate two-item candidate sets. For generating 3-item candidate set, we combine each pair of elements of two-item item set and generate new set. If the size of the resulting set is equal to 3 then we add this new set to a map<string,int> and add its value by 1. If the value of each set in our map is less than 3, it means that one of its subset was not in the two-item item set so we discard it. Pass the remaining to the counting part and do the same for finding 4-item candidate sets, but now the value of each set in our map should be 4 because, for generating for example “abcd” from its 3-sized subsets all “abc”,”acd”,”abd”, and “bcd”, should be seen which results in an at least 4 instance of “abcd”. Using this optimization we don’t need to check all subset of “abcd” to see if all of them are frequent. We do the same for the other k-element candidate sets.

As for the second part, we first sort each transaction and generate all k-sized subsets of each transaction in the dataset based on the k-1-sized subsets. Since our transactions are all sorted, we only append items that are greater than the last item in a k-1-sized subset for generating a k-sized subset. This optimization improved the results a lot since we no longer generate all permutations of k items.
Instead of using a recursive function to generate the subsets, we store all k-sized subsets of all transactions in a map that maps the transaction id to the latest generated subsets. Later on, when it is time to generate k+1 sized subsets of all transactions; we only need to build upon the existing k-sized datasets.
After coming up with all k-sized subsets of a transaction, we match those subsets against the candidates using the prefix tree. To do so, we first feed the k-item candidate sets into a prefix tree. The prefix tree saves us a lot of effort in counting since it helps us rule out transaction subsets as soon as we realize no candidate set starts with the first item in the subset, without having to try to match the whole subset with the whole tree. We take advantage of the fact that both the candidate sets and the transaction subsets are sorted.

We are happy we gave C++ a shot and forced ourselves to remember its syntax and structure. The performance complexity is fairly decent and we are hoping that we are doing better than the most naïve implementation.