Name: Viet Duong

CSC 240 Project 1

**6.7 (Implementation project) Using a programming language that you are familiar with, such as C++ or Java, implement three frequent itemset mining algorithms introduced in this chapter:**

**(1) Apriori.**

**(2) FP-growth.**

**(3) One Improvement of choice for Apriori.**

**Compare the performance of each algorithm with various kinds of large data sets. Write a report to analyze the situations (e.g., data size, data distribution, minimal support threshold setting, and pattern density) where one algorithm may perform better than the others, and state why.**

All of my implementations are in Java.

(1) For Apriori, I use the algorithm that closely follows the pseudocode in Figure 6.4.

(2) For FP-Growth, I use the algorithm described in Figure 6.9.

(3) In order to improve the Apriori algorithm, I use a Hash-based technique. Since the original Apriori scan the whole dataset to compute the support count at every iteration which takes a lot of time, I add a hashtable to map each item to its count so that the algorithm only needs to check the hashtable and get the support count for each item immediately.

The data set is the UCI Adult Census Dataset, which consists of 48842 data points and is used to predict whether income exceeds $50000 per year based on census data. I think the size of the dataset is sufficient to test the implementations of the algorithms above, in order to compare their accuracy and efficiency.

The minimal support threshold used is 60%, which is commonly seen to be used for exercises in the textbook.

The runtimes for the Apriori, FP-Growth and improved Apriori algorithms are 116.116, 0.592, and 0.67 seconds respectively. Although the runtimes do vary, the rank of efficiency of the algorithms in decreasing order is Fp-Growth>Improved Apriori>Apriory.

Apriori is the slowest because it has to scan the entire dataset for every iteration for generating new candidates. Meanwhile, the improved version of Apriori has a hashtable as reference. Lastly, FP-Growth is the fastest because it only has to scan the data twice, once for growing the FP-Tree and once for mining the FP-Tree.

Please refer to the README.txt file for instructions to run the codes and sample outputs.

*References:*

Jiawei Han, Micheline Kamber, Jian Pei, Data Mining: Concepts and Techniques, 3rd Edition.

UCI Machine Learning Repository, <http://archive.ics.uci.edu/ml/datasets/Adult>.

Kamran Nobahar, An implementation of FP-Growth algorithm in Java, open-source GitHub repository. <https://github.com/goodinges/FP-Growth-Java>.

Yutong He, Frequent Pattern Mining, open-source GitHub repository, <https://github.com/KellyYutongHe/Frequent-Pattern-Mining>.