Ian Delbridge
CSC240
Luo
10/20/14

Write Up for Frequent Pattern-Mining Implementation Project

Prompt: *6.7 (**Implementation project**) Using a programming language that you are familiar with, such as C++ or Java, implement threefrequent itemset mining algorithms introduced in this chapter: (1) Apriori [AS94b], (2) FP-growth [HPY00], and (3) Eclat [Zak00](mining using the vertical data format). 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.*

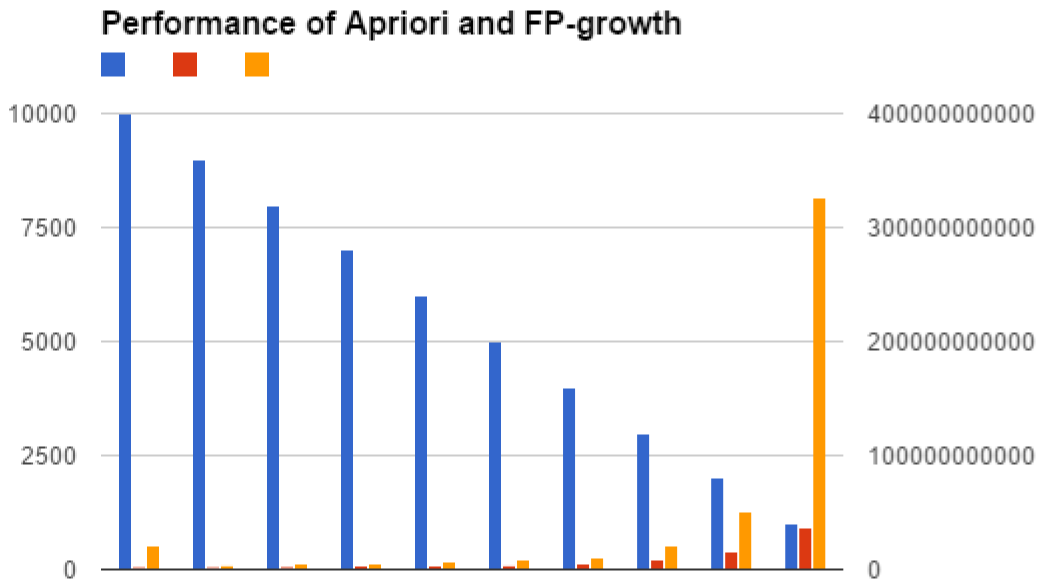
In this project I implemented in Java (1) Apriori and (2) FP-growth algorithms, and also implemented (3) a separate version of Apriori that intends to improve slightly on the efficiency of Apriori. I wrote these with the focus of using them on the data set given at http://archive.ics.uci.edu/ml/datasets/Adult. In order to keep target this data set as well as keep the algorithms abstract to any set of transactions, I created separate preprocessing files to deal with the challenges of this data set. That is, though in the original document they are not labeled, 0 might mean zero education level, zero losses, or zero gains, etc. depending on the placement in the transaction, so I simply concatenated the field and an equals sign to each transaction as a string to differentiate capital-gain=0, capital-loss=0, etc. This may amplify the time required to scan, since it makes the data file longer, and each string itself longer, but it brings more clarity.

I should also note that the continuous data here was not suited for this type of data mining. I planned on partitioning the continuous fields into discrete classes, but time did not permit, so these algorithms consider 10000001 completely different from 10000000.

I implemented a postprocessing algorithm to determine the association rules separate from generating the frequent itemsets. It functions entirely separately as (1), (2), and (3) create output files containing the frequent itemsets.

Both Apriori and FP-growth generate the same itemsets, so I am relatively confident that for a given support level they will generate correct frequent itemsets. However, FP-growth is notably faster and for good reason. Firstly, candidates must be generated which is not an inconsequential task - it takes up a large amount of runtime. However, it's very costly the fact that the algorithm must scan the data set for each set of candidates that it creates. As a result, it wastes a lot of time on the candidates. FP-growth, however, is not the same breadth-first type search that Apriori is. Instead, it only “scans” the part of the data set that is relevant to the potentially frequent itemset by using conditional trees to, more or less, narrow down the space that it must search. We end up with just two scans through the data set, and the rest comes directly from the tree that is built.

I compared the runtime of the the two algorithms below:



Here the blue represents the ,minimum support count of the frequent itemsets, red represents FP-growth runtime in nanoseconds, and orange represents Apriori runtime in nanoseconds. You can see experimentally that Apriori has explosive runtime that seems exponential as the size of the minimum support count decreases. Further, while FP-growth does have increased runtime for lower support, it grows at a much slower rate. FP-growth is the scalable algorithm in this situation. It is the only of the two that could stand up to exceedingly large data and have reasonable runtime.

My small improvement on apriori is simply a way of skipping the transactions in the database that will not be used anymore. That is, I just mark the transaction and check for the mark when I am about to read the transaction. It does seem to save a little time, but not as substantial as FP-growth - instead it seems to growth at approximately the same rate, but with slightly smaller times at each support-count size. This is about what I would expect, because the reduction in calculations is the size of one transaction times the number of transactions not being looked at, which start at zero and increase to eventually be all of the transactions.

From this analysis, I think that FP-growth is the preferred method of the three when finding frequent itemsets for large amounts of data, especially when the support minimum is small. However, for smaller projects and/or higher supports, apriori *does* function, as well as its small improvement algorithm.