Jakefim: Frequent Itemset Mining

by Jake Cronin

Summary

Jakefim is an apriori frequent itemset mining algorithm written in java.

Given a list of transactions, jakefim constructs a list of all frequently appearing item combinations. Jakefim is an interative approach, first finding frequent itemsets of size 1, and looking for increasingly larger itemsets on each iteration.

Interesting Optimizations

Data Pruning

The original dataset is traversed only twice, for counting frequency of size 1 and size 2 item sets. In subsequent iterations, the data set is built up as lists of i-sized item combinations for each transaction. The advantage of this method is that no time is spent checking a superset whose subset is already known to be infrequent.

Item Object -> Fast Combination of Items

At the start of an iteration, it is possible to spend a lot of time deciding which supersets of items to count in the data set. If done poorly, an algorithm could also end up double-counting combinations of items or looking for supersets whose subsets are already known to be infrequent. To improve performance in deciding which supersets to check, jakefim uses an object called an Item. An Item represents a set of items, which it stores as a string. It also has a property called the prefix, which is a string of all items in the Item except for the most recently added item.

The advantage of using the Item object is that it has a function, combine(), that makes it very fast to check if two item sets should be combined into a superset. If two Items are frequent and share the same prefix, their combination will be a unique superset that may also be frequent. If the two items do not have an identical prefix, however, the superset will either not be unique, or will be visited at a later iteration of the algorithm.

The Item object allows jakefim to quickly prune the transaction data, which is critical to limiting the amount of permutations of items that need to be counted.

High Level Overview

Step One - Loading in Data

The outer linkedlist represents transactions. The inner linked list holds each individual item of the transaction

Step Two - Counting Individual Items

On one pass through the 2d linked list of data, jakefim builds a hashmap to count occurances of each item. When an item’s counter reaches the mininmum support threshold, it is added to a second hashmap containing only frequent items.

Step Three - Counting Size 2 Item Sets

Running through the original dataset, transactions are removed and checked one at a time. For each item in a transaction, if the item is not present in the frequents hashmap from step two, it is removed from the transaction. Frequent items are compared against every other remaining item in the transaction. If both items are frequent, they are combined and counted into a hashmap similarly to how items are counted in step two. In addition, the items are added into a new transaction list to replace the list of individual items. After this step, the count of all frequent size-2 items are saved in a hashmap, and the original data list of transactions is replaced with a list of size-2 item combinations of frequent items from each transaction.

Step Four - Counting Size 2+ Itemsets

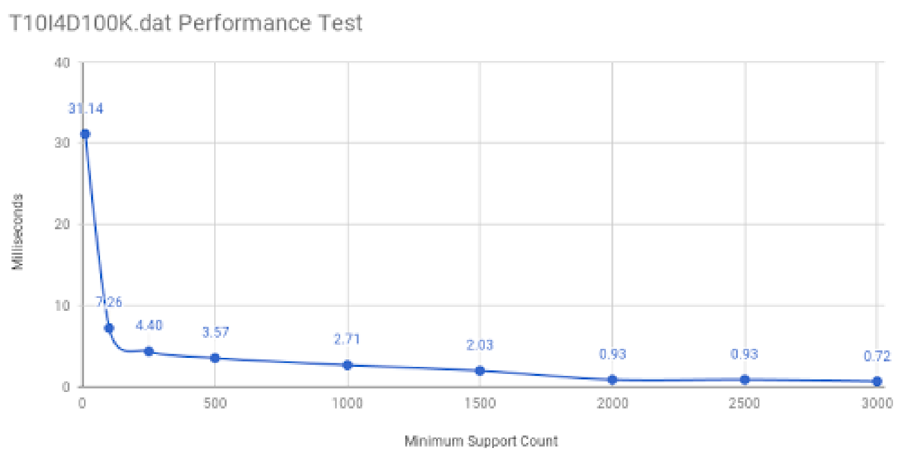
Passing through the modified data from the previous iteration, each transaction is looked at individually. For each i-size item set in the transaction, if it is not present in the frequents hashmap from the previous iteration, it is removed from the transaction. The item sets that are present are compared to every other frequent item set in the transaction. Item sets that are identical up to the i-1’th item are combined to form a superset of i+1 items. This superset is counted in a hashmap and added to a new transaction list of i+1 sized item sets to be used for the next iteration.

Step 5 - Saving Data

The frequency list containing a hashmap of frequent item sets of each length is iterated over, sorted, and formatted into strings that are written to the output file.

Performance:

Running on a 2015 Macbook Pro with a 2.2 Ghz processor



T10I4D100K.dat	
Minimum Support Count	Average Runtime (ms)
10	31139
100	7264
500	3573
1000	2711
2000	925

chess.dat	
Minimum Support Count	Runtime (ms)
2500	42534
3000	754