Homework 2 - Finding Frequent Itemsets

Perttu Jääskeläinen Group 11 Gabriele Morello
perttuj@kth.se morello@kth.se

November 21, 2022

Contents

1 Introduction

In this homework, we identified frequent item sets from a set of transactions, representing items bought by customers in a store. Each transaction would contain an arbitrary amount of integer values, which represented the ID's of items bought in the transaction. The sets would be filtered based on some support value, s, given as input to the program.

When each frequent item set was identified, we also mined for association rules - given a frequent item set I, identify if some subset of I (called A) that causes the remainder of the set I to occur - i.e. A $->$ I $\setminus$ A. To identify these rules, we used another input value for the confidence of association rules - c, which calculated association rules based on the frequently identified item sets.

The implementation used the A-Priori algorithm to detect frequent item sets. It was implemented in Scala using the Apache Spark for distributing the data sets and processing data in parallel.

To run the code, see the **README.md** file in the project root.

2 Problem description

A frequent item set is a set of items that occur together frequently. For example, for three transactions containing [A,B,C], [B,C,D], [C,D,E], the frequent item sets would be [B,C] and [C,D] (occurs in 2/3 transactions). Support for a set of items would be the amount of transactions containing the set, divided by the total amount of transactions. Support for the sets mentioned above would be 66%, or s = 2.

With these frequent item sets identified, we discovered association rules. An association rule is a relation between two sets (which are both subsets of a frequent item set) - given a frequent item set I, an association rule would be to detect association rules such that A $->$ I $\setminus$ A. To calculate the confidence for an association rule, one would divide the amount of transactions containing I with the amount of transactions containing A.

For example, for set [B,C] in the example above, we would have association rules [B]$->$[C] would have a confidence of 2/2 = 100%, and [C]$->$[B] would have a confidence of 2/3 or 66%.

The values for support s and confidence c were given as input parameters to our application - which we set to 1% and 50% during testing.

3 A-Priori Implementation

The A-Priori algorithm is based on making passes through the data set, and constructing data as we go - without having to keep everything in memory at once. During each pass, we want to find items that have a support value of s; which represents how many times each element (or combination of elements) occur in the set of transactions.

By avoiding having to keep everything in memory at once, we can greatly improve run time efficiency both by speed (less comparisons to make with a smaller data set), but also in terms of memory. This is useful to identify frequently occurring items - but we might also be interested in *which of the items influences the other*. For example, if we have items [A,B,C] - does including A imply that B and C will also occur? Does [A,C] usually cause B?

This consideration will be addressed in the next section, where we implement confidence of association rules.

4 Confidence of Association Rules

In the basic implementation, confidence of the result wasn't taken into account - which might make it clearer if the result obtained is accurate or not. For example, we might have 100k transactions where items A and B appear in 2000 of them ($support \geq 1000$) - but if A appears in 80k transactions with B in only 2000 of them, the confidence isn't very high that A and B are usually bought together. We used a confidence ratio of 50% to obtain the result.

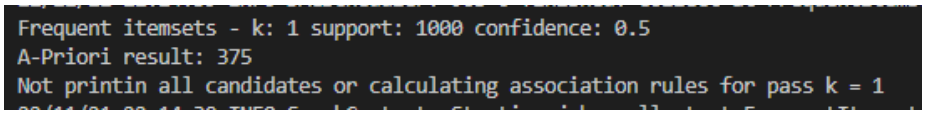

Figure 1: *Results of pass k = 1, note that we don't generate association rules, since we cannot have a subset of 1 item(s).*

To calculate the confidence we create all the possible subset of a candidate with length between one and length of the candidate minus one, we then match each subset with all the other subsets that don't contain him. We iterate through these pairs and we calculate the confidence dividing the support of the union of the first and second element by the support of the first element. We compare the confidence with a given threshold and we return a tuple made by the original candidate, the two partitions and the resulting confidence.

In case of support equal to 1000, tripletons and confidence threshold equals to 0.5 we get four partitions of the same single tripleton - three of them in the form doubleton - singleton with a confidence around 0.9 and the other in the form singleton - doubleton with a lower confidence around 0.5. The outputs of passes 1-3 can be seen in figures 1, 2 and 3.

5 Discussion

We had some difficulties in figuring out why the implementation wasn't working - we were not finding any candidate pairs with support = 1000, so we had to spend some time debugging. We realized that it was because we weren't constructing the candidate pairs correctly from a set of ID's - for example, if we had ID's [1,2,3] , we only made pairs [1,2] and [2,3] (for k = 2), and forgot to take into account that pairs such as [1,3] should also be included. When we realized this, we implementing the fix and achieved the result mentioned in the 3:nd section.

We also had slight difficulties understanding how the association rules are intended to work - first, we assumed that we would take any subset I of a candidate pair A, and see the amount of transactions in which I occurs, and some random element(s) S occur (where S is not part of A). When we realized that the rule should be I - I A - it made sense, and was easier to implement.

We also had some performance issues with our implementation, where we were repeatedly looping through the datasets in each pass (k = 1, k = 2) etc. - so we spent some time looking into how to improve the Spark implementation.

At the end, we feel like we learned a lot about identifying itemsets, while learning more about Spark and Scala at the same time. We're looking forward to the remaining homeworks to go even more in depth!

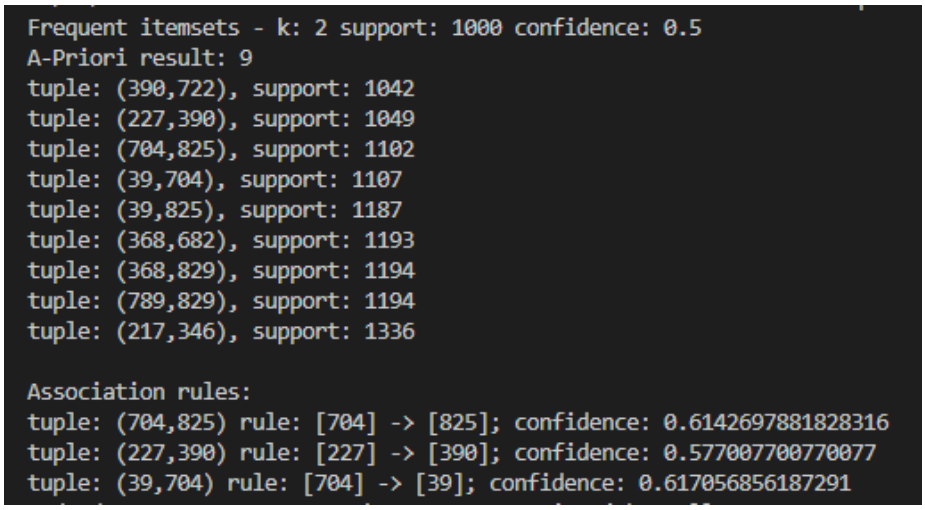

Figure 2: *Results of A-priori algorithm of pass $k = 2$.*

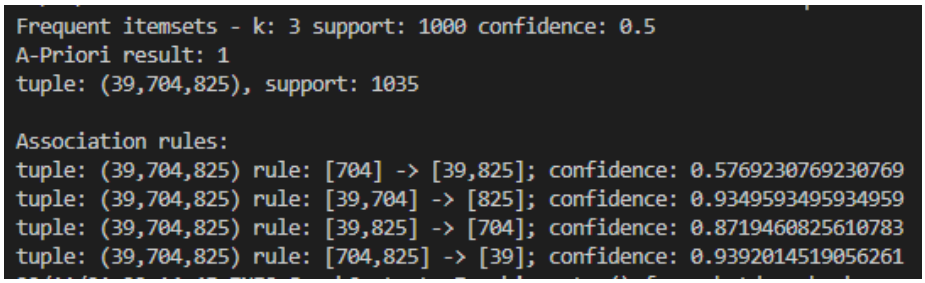

Figure 3: *Results of A-priori algorithm of pass $k = 3$.*