# Data Mining Assignment - Pattern Mining

Arno Deceuninck

March 16, 2022

## 1 Implementation

I started trying to find some libraries that implemented apriori or eclat, but still implemented my own version of eclat in order to fully understand the algorithm. When reading the dataset, you get a list of baskets, which are sets of items viewed/purchased/... by the same user. So for each user, we have a set of items. We'll now try to search the frequent itemsets, which are sets of items that are frequently bought together (at least minsup (minimum support) times). In eclat, we start by changing this structure and using tidsets (transaction id sets) in our new structure. Our new structure maps each of the item onto the sets of transaction id's of the transactions that contain given item. When generating this set, we can already filter out all the items that have less transactions than the specified minimum support, since they'll never be in a frequent itemset that will satisfy this minimum support. This gives us the frequent itemsets containing just one item. We can now use those itemsets to generate the frequent itemsets containing two items, since this can be found by taking the intersection of the tidsets of the two items in it. Note that after each intersection, we still have to check whether the found itemsets still satisfies our minsup (which can be done by counting the transactions in the tidset). We can keep doing this recursively until we don't find any new frequent itemsets satisfying our minsup.

By doing this, we have found our frequent itemsets satisfying our minsup. To create association rules from those itemsets, we'll generate all possible bi-partitions of each of those sets and see the first partition as the left side of our association rule and the other partition as the right side of our association rule. For each of those partitions, we then check whether it satisfies the required confidence and if this is the case, we have found an association rule.

## 2 Run on dataset

This are the top 10 itemsets in terms of support. The confidence and support are the numbers after each rule.

```
{5677043} -> {5697463} (0.33, 55)
{5814516} -> {5814517} (0.88, 49)
```

```
{5809912} -> {5809910} (0.58, 49)
{5809910} -> {5809911} (0.16, 47)
{5649235} -> {5649236} (0.35, 30)
{5886282} -> {5909810} (0.35, 30)
{5751422} -> {5751383} (0.28, 30)
{5886282} -> {5892179} (0.33, 28)
{5649236} -> {5649271} (0.14, 27)
{5886282} -> {5900651} (0.31, 26)
```

Almost all of the itemsets in this top 10 consist of only one item. This makes sense, because they are sorted on minsup and a subset will always have the same or a higher minsup (but the confidence can be lower).

# 3 Interesting rules

A rule is interesting if it contains both a high minsup and a high confidence. We want the minsup to be as high as possible, so we have a lot of items the rule has effect on. We also want the confidence to be as high as possible, so we're actually outputting rules that are often valid. We can e.g. find interesting rules by defining a minimum threshold confidence and filtering out all rules with a lower confidence than this threshold. By sorting the remaining items on minsup, we can get the interesting results. If we set this way our minimum confidence on 0.7 and sort on the support, we get following rules:

```
{5814516} -> {5814517} (0.88, 49)
{5855091} -> {5855092} (0.82, 18)
{5739056} -> {5739055} (0.74, 17)
{5846436} -> {5846437} (0.74, 17)
{5816166} -> {5809910} (0.78, 14)
{5827356} -> {5827357} (0.92, 12)
{5707826} -> {5692527} (1.00, 9)
{5739122} -> {5739124} (0.82, 9)
{5787479} -> {5692527} (0.75, 9)
{5814516, 5804820} -> {5814517} (0.89, 8)
```

Instead of sorting on the support, you can also sort on the confidence multiplied with the support to get interesting rules. The results with this are similar in our case.

# 4 Extra information

A company will most likely try to recommend items the user will actually buy. With this in mind, you want your right side of your association rule only to have items the user actually purchased. Note that the left side can still contain

the information of viewed/added to cart/...items, since this is also relevant information to get insight on what kind of items the user is searching for. Instead of just taking a look at the itemsets, you can also reapply the same algorithm, but by seeing each (action, item) pair as a different item. This might give you different itemsets based on the action on the item. E.g. where we would initially have:

```
{item1} -> {item2}
```

we would now get:

```
{(viewed, item1)} -> {(purchased, item3)}
{(purchased, item1)} -> {(purchased, item4)}
```

with as mentioned earlier only things that are purchased on the right side, since this is what most stores are interested in. This is what I also implemented (with a minsup=5 to limit the execution time).

# 5 Interesting rules with extras

Our interesting rules remain the rules with a high support and high confidence. The main change is not in how to order them based on level on interest, but how the rules are generated. Our most interesting rules would contain only purchased items on the right side, but this is already taken into account when generating the rules. This are our new rules with 0.7 as minimum confidence sorted on confidence multiplied by support.

```
{'purchase5814516'} -> {'purchase5814517'} (0.90, 46)
{'purchase5814516', 'cart5814516'} -> {'purchase5814517'} (0.90, 43)
{'cart5814516'} -> {'purchase5814517'} (0.86, 43)
{'cart5809911'} -> {'purchase5809910'} (0.78, 36)
{'purchase5819894'} -> {'purchase5823667'} (0.82, 23)
{'purchase5814516', 'purchase5814518'} -> {'purchase5814517'} (1.00, 18)
{'purchase5814518'} -> {'purchase5814517'} (0.86, 19)
{'purchase5855091'} -> {'purchase5855092'} (1.00, 16)
{'cart5814516', 'purchase5814518'} -> {'purchase5814517'} (1.00, 16)
{'purchase5855091', 'cart5855091'} -> {'purchase5855092'} (1.00, 16)
```