Homework frequent itemsets

Arno Strouwen

April 7, 2023

Exercise 6.1.1

Suppose there are 100 items, numbered 1 to 100, and also 100 baskets, also numbered 1 to 100. Item i is in basket b if and only if i divides b with no remainder. Thus, item 1 is in all the baskets, item 2 is in all fifty of the even-numbered baskets, and so on. Basket 12 consists of items 1, 2, 3, 4, 6, 12, since these are all the integers that divide 12. Answer the following questions:

(a) If the support threshold is 5, which items are frequent?
The items 1 trough 20 are frequent, because item i is present in 100 divided by i baskets, rounded down to the nearest integer.

(b) If the support threshold is 5, which pairs of items are frequent?
Because there are 20 frequent singleton items, there are 20*19/2=190 possible frequent doubleton. A doubleton is frequent if the least common multiple of its two elements is smaller than 20, e.g. {2, 11} is not frequent because this doubleton will only appear in baskets divisible by 22, and we know 22 is not frequent because of part (a). This can thus by calculated be the following Julia code snippet:

```
frequent_doubletons = Tuple{Int,Int}[]
for i in 1:20, j in i+1:20
    lcm(i,j)<=20 && (push!(frequent_doubletons,(i,j)))
end
julia> println(frequent_doubletons)
[(1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (1, 9),
(1, 10), (1, 11), (1, 12), (1, 13), (1, 14), (1, 15), (1, 16),
(1, 17), (1, 18), (1, 19), (1, 20), (2, 3), (2, 4), (2, 5),
(2, 6), (2, 7), (2, 8), (2, 9), (2, 10), (2, 12), (2, 14),
(2, 16), (2, 18), (2, 20), (3, 4), (3, 5), (3, 6), (3, 9),
(3, 12), (3, 15), (3, 18), (4, 5), (4, 6), (4, 8), (4, 10),
(4, 12), (4, 16), (4, 20), (5, 10), (5, 15), (5, 20), (6, 9),
(6, 12), (6, 18), (7, 14), (8, 16), (9, 18), (10, 20)]
julia> length(frequent_doubletons)
56
```

(c) What is the sum of the sizes of all the baskets?
The sum of the sizes of the baskets is equivalent to the sum of in how many baskets each item occurs. Item i occurs in 100 divided by i baskets, rounded down to the nearest inter. In Julia code:

```
sum_basket_sizes = 0
for i in 1:100
    sum_basket_sizes += 100÷i
end
julia> sum_basket_sizes
482
```

Exercise 6.1.5

For the data of Exercise 6.1.1, what is the confidence of the following association rules?

(a) $\{5, 7\} \rightarrow 2$.
The doubleton {5, 7} is in baskets 35 and 70. Only 70 is divisible by 2, thus the confidence is 1/2 or 50%.

(b) $\{2, 3, 4\} \rightarrow 5$.
b) The tripleton {2,3,4} is in baskets 12, 24, 36, 48, 60, 72, 84 and 96. Only 60 is divisible by 5, thus the confidence is 1/8 or 12.5%.

Exercise 6.2.5(a)

Suppose the support threshold is 5. Find the maximal frequent itemsets for the data of Exercise 6.1.1.
An itemset cannot be frequent if it contains an item i greater than 20, see 6.1.1(a).
An itemset cannot be maximal if there exists an item j, which divides an item of the itemset i, but is not in the itemset, e.g. {10} cannot be maximal because 5 divides 10 and thus every basket containing 10 will also contain 5. The superset {5, 10} is thus also frequent.
Any item i in a maximal itemset must divide the largest element of the itemset m. Suppose this was not the case, then adding the least common multiple of i and m to the itemset, would result in a superset that is still frequent.
Using these three lemmas, it is easy to verify that the following are the only maximal itemsets:

```
{20,1,2,4,5,10}, {19,1}, {18,1,2,3,6,9}, {17,1}, {16,1,2,4,8},
{15,1,3,5}, {14,1,2,7}, {13,1}, {12,1,2,3,4,6}, {11,1}
```

Exercise 6.3.1

Here is a collection of twelve baskets. Each contains three of the six items 1 through 6.

```
{1, 2, 3} {2, 3, 4} {3, 4, 5} {4, 5, 6}
{1, 3, 5} {2, 4, 6} {1, 3, 4} {2, 4, 5}
{3, 5, 6} {1, 2, 4} {2, 3, 5} {3, 4, 6}
```

Suppose the support threshold is 4. On the first pass of the PCY Algorithm we use a hash table with 11 buckets, and the set $\{i, j\}$ *is hashed to bucket* $i \times j$ *mod 11.*

All answers are generated using the following Julia program:

```
using LinearAlebra
dataset = [[1, 2, 3],
           [2, 3, 4],
           [3, 4, 5],
           [4, 5, 6],
           [1, 3, 5],
           [2, 4, 6],
           [1, 3, 4],
           [2, 4, 5],
           [3, 5, 6],
           [1, 2, 4],
           [2, 3, 5],
           [3, 4, 6]]

hashbucket_dict = Dict{Int64,Array{Tuple{Int64, Int64}}}()
for i = 1:6
    for j = i+1:6
        bucket = (i*j)%11
        if haskey(hashbucket_dict,bucket)
            hashbucket_dict[bucket] = push!(hashbucket_dict[bucket],(i,j))
        else
            hashbucket_dict[bucket] = [(i,j)]
        end
    end
end

singleton_counter = zeros(Int,6)
doubleton_counter = UpperTriangular(zeros(Int,6,6))
hashbucket = zeros(Int,11)
for basket in dataset
    for i in eachindex(basket)
        singleton_counter[basket[i]] += 1
        for j in i+1:lastindex(basket)
            doubleton_counter[basket[i],basket[j]] += 1
            hashbucket[((basket[i]*basket[j])%11)+1] += 1
        end
    end
end

hashbucket_frequent = hashbucket .>= 4

in_second_pass = Tuple{Int64, Int64}[]
for i in eachindex(hashbucket_frequent)
    if hashbucket_frequent[i]
        for pair in hashbucket_dict[i-1]
            if singleton_counter[pair[1]] >= 4 && singleton_counter[pair[2]] >= 4
                push!(in_second_pass,pair)
            end
        end
    end
end
```

(a) By any method, compute the support for each item and each pair of items. We store the support of singletons in a vector and the support of doubletons in an upper triangular matrix. Ignore the diagonal of this matrix. The support of singletons is calculated by looping over every basket in the dataset. Within each basket we loop over every item and increment the elements of the singleton counting vector whose indices correspond to the items in the basket. We also iterate over each pair of items in the basket and similarly increment the doubleton counting matrix.

```
julia> singleton_counter
6-element Vector{Int64}:
4
6
8
8
6
4

julia> doubleton_counter
6×6 UpperTriangular{Int64, Matrix{Int64}}:
0  2  3  2  1  0
   0  3  4  2  1
      0  4  4  2
         0  3  3
            0  2
               0
```

(b) Which pairs hash to which buckets?
A dictionary is created with keys equal to the numbers 0 through 11. The value corresponding to a key is an array of pairs[1] whose multiplication modulo 11 equals the key. The key 0 is missing, because nothing hashes to 0.

```
julia> hashbucket_dict
Dict{Int64, Array{Tuple{Int64, Int64}}} with 10 entries:
5  => [(1, 5)]
4  => [(1, 4), (3, 5)]
6  => [(1, 6), (2, 3)]
7  => [(3, 6)]
2  => [(1, 2), (4, 6)]
10 => [(2, 5)]
9  => [(4, 5)]
8  => [(2, 4), (5, 6)]
3  => [(1, 3)]
1  => [(2, 6), (3, 4)]
```

(c) Which buckets are frequent?
The support of each bucket is stored in a vector, where the i+1th element contains the support of all pairs who hash to the value i. This is because Julia arrays start at index one, and the first element of the vector stores pairs which hash to zero. The support is calculated in a similar way as in (a), by looping over all the pairs of items in each basket, and incrementing the element of the bucket vector whose index is determined by the hash function. The 2nd, 3rd, 5th and 9th bucket are frequent, which corresponds to hash values 1, 2, 4 and 8.

```
julia> hashbucket_frequent
11-element BitVector:
0
1
1
0
1
0
0
0
1
0
0
```

(d) Which pairs are counted on the second pass of the PCY Algorithm?
We iterate over the vector containing the supports of the buckets from (c). For the frequent buckets we look up the corresponding pairs, using the dictionary

[1]Strictly speaking it is an array of tuples. A pair is a datatype with another meaning in Julia.

created in (b). We verify if the singletons corresponding to these pairs are also frequent, using the vector from (a). This leads to the following pairs being in the second pass:

```
julia> in_second_pass
8-element Vector{Tuple{Int64, Int64}}:
(2, 6)
(3, 4)
(1, 2)
(4, 6)
(1, 4)
(3, 5)
(2, 4)
(5, 6)
```

Exercise 6.4.1

Suppose there are eight items, A, B, . . . , H, and the following are the maximal frequent itemsets: {A, B}, {B, C}, {A, C}, {A, D}, {E}, and {F}. Find the negative border.

No set with four or more items can be in the negative border, because no tripletons are frequent.
Tripletons in the negative border cannot contain E,F,G or H because no frequent doubletons contains these. No tripleton containing D can be in the negative border, because there are no two frequent doubletons containing D, leaving only the tripleton containing {A, B, C}. All three direct subsets of this tripleton are frequent, it is thus in the negative border.
No doubleton containing G or H can be in the negative border, because no frequent singleton contains these. Since the singletons {A} through {F} are frequent, all doubletons consisting of two different letters from A trough F are in the negative border unless the doubleton is frequent itself. There are $\binom{8-2}{2} - 4 = 11$ such doubletons: {A, E}, {B, E}, {C, E}, {D, E}, {A, F}, {B, F}, {C, F}, {D, F}, {B, D} , {C, D}, {E, F}.
Finally, the singletons {G} and {H} are on the negative border, since the empty set is considered frequent.

Exercise 6.4.2

Apply Toivonen's Algorithm to the data of Exercise 6.3.1, with a support threshold of 4. Take as the sample the first row of baskets: {1, 2, 3}, {2, 3, 4}, {3, 4, 5}, and {4, 5, 6}, i.e., one-third of the file. Our scaled- down support threshold will be 1.
Important: The notes define a set to be frequent if it has support at least equal to some threshold. A threshold of one thus leads to the above 4 tripletons

and all their subsets to be frequent. I suspect that the authors intended the threshold to be larger than but not equal to 1, even though this is not the strict definition of a frequent set. I will solve the exercise both ways.

Scaled down support larger than or equal to 1:

(a) What are the itemsets frequent in the sample?

The above four tripletons are frequent. All doubletons of the form $\{i, i+1\})$ and $\{i, i+2\})$ are frequent, namely: {1, 2}, {1, 3}, {2, 3}, {2, 4}, {3, 4}, {3, 5}, {4, 5}, {4, 6}, {5, 6}. All singletons are frequent. The empty set is frequent.

(b) What is the negative border? Since all singletons are frequent, the doubletons not listed in (a) are in the negative border: {1, 4}, {1, 5}, {1, 6}, {2, 5}, {2, 6}, {3, 6}. No tripletons are in the negative border, e.g. {1, 2, 4} cannot be in the negative border because {1, 4} is not of the form $\{i, i+1\})$ or $\{i, i+2\})$, similar reasoning holds for all other tripletons that are not already frequent.

(c) What is the outcome of the pass through the full dataset? Are any of the itemsets in the negative border frequent in the whole?

Looking at the matrix calculated in 6.3.1 (a) we see that none of the elements in the matrix corresponding to pairs in the negative border have a value larger than or equal to 4. None of the elements of the negative border are thus frequent in the whole sample. For the sets that are frequent in the subsample, we see that all singletons are still frequent, all doubletons except for {2, 4}, {3, 4} and {3, 5} become infrequent in the whole. We did not store support for tripletons but there exist no tripleton whose direct subsets are all frequent in the whole, so no tripleton is frequent in the whole. The empty set is, of course, still frequent in the whole. The algorithm correctly found all the frequent itemsets and terminates.

Scaled down support strictly larger than 1:

(a) What are the itemsets frequent in the sample?

The empty set. All singletons except for {1} and {6}. The doubletons {2, 3}, {3, 4} and {4, 5}. No tripletons or larger sets.

(b) What is the negative border?

{1} and {6}. {2, 4}, {2, 5} and {3, 5}.

(c) What is the outcome of the pass through the full dataset? Are any of the itemsets in the negative border frequent in the whole?

The singletons in the negative border are frequent in the whole. For the doubletons in the negative border {2, 4} and {3, 5} are frequent in the whole. All singletons that are frequent in the subsample remain frequent in the whole. The doubletons {2, 3} and {4, 5} become infrequent in the whole. The doubleton {3, 4} remains frequent in the whole. Since there are frequent itemsets in the negative border, the algorithm must start again with another random sample, because we cannot be sure all the frequent itemsets are found.