Collaboration: No collaboration
External resources: https://athena.ecs.csus.edu/~mei/associationcw/RuleGeneration.html

Task 1: Function for the support count

My function has two arguments (dataset and itemset)
I initialize a count at 0 and for each transaction in my dataset, if my itemset is a subset of a transaction, I add one 1 to my count. I return this count as the support count

Task 2 :Level Wise enumeration

I define for this task 3 function, one to get the items, one to get candidates and one for the frequent candidate.

get_frequent return the frequent and non-frequent candidates based of their support count, so we make a loop on a list of candidates and if the support count >=threshold, We add it to our frequent list , else to the non-frequent list. We return both

get_items to have the first line of our enumeration, so we look in each transaction, if we find a new item, we add to our list of items. At the end, we return it.

generate_next_level to generate the next level of our enumeration tree. I generate the next level by adding one item. I look for the index of the last item of the subset so I can add the items which come after to perform a level wise enumeration using a canonical order.

Now I have what I need, I define my function level_wise_enumeration.
It starts by getting the items with the get_item function, get the frequent items.
We initialize our list of frequent items frequent_itemset with the frequent items of the first level . In a while loop , we generate our candidates and after apply the downward closure if our candidate is the superset of a non-frequent itemset .After pruning, we get the frequent itemset by support count.
At the end, we print the number of frequent itemset and return the list of frequent itemset .

Task 3: Test on dataset

I make some experimentation on datasets. I used loading function in the code snippets to load them and run my function level_wise_enumeration to mine the frequent itemset. I report the datasets, parameters and running time in the table below.

datasets	threshold	frequent itemset	running time
abalone	500	529	0.821431
abalone	250	1187	1.617543
pizzas	145	23	0.005154
pizzas	289	11	0.002083
house	260	3	0.003995
house	200	28	0.01699

The implementation actually work on this datasets and outputs the result expected but when I try it on more huge datasets, It never terminates. It seems to not be performant enough.

Task 4: Associations rules

I create a class Rule (antecedent, consequent), a method to calculate the confidence and another one to check If an association is strong or not based on the threshold.

I define a function non_empty_itemset which return all the itemset I need for my rules, another function to generate the rules for each itemset. for the final function association rules, I check if the association rule is strong, I add it to a list. I return this list at the end

I tested this function on pizzas dataset

confidence threshold	Output
0.8	['tuna'])==>(['mozzarella']) (['mozzarella', 'olives'])==>([' ham'])
0.5	([' ham'])==>(['mozzarella']) (['mozzarella'])==>([' ham']) ([' ham'])==>(['olives']) (['olives'])==>([' ham']) (['tuna'])==>([' ham']) (['mozzarella'])==>(['olives']) (['olives'])==>(['mozzarella']) (['tuna'])==>(['mozzarella']) ([' ham', 'mozzarella'])==>(['olives']) ([' ham', 'olives'])==>(['mozzarella']) (['mozzarella', 'olives'])==>([' ham'])
0.7	([' ham'])==>(['mozzarella']) (['mozzarella'])==>([' ham']) (['tuna'])==>([' ham']) (['tuna'])==>(['mozzarella']) ([' ham', 'olives'])==>(['mozzarella']) (['mozzarella', 'olives'])==>([' ham'])