

University of Groningen
Information Systems
Assignment 4
Group 18

Satyanarayan Nayak, Swastik
s.nayak.1@student.rug.nl
S4151968

Mitra, Siddharth
s.mitra.2@student.rug.nl
S4138430

December 17, 2019

Contents

1	Problem Definition	2
2	Solution	2
2.1	Histogram	2
2.2	Frequent Items and Support	3
2.3	Generating the Association rules	3
2.4	Brute Force approach (without Anti-Monotone) implementation	4
2.5	Anti-Monotone	6
2.6	Top 30 association rules sorted by confidence in descending order	8
2.7	Implementation	8
2.8	Results	9
3	Conclusion	10
4	GitHub	10

1 Problem Definition

1. Show the histogram of all items in the 9835 transactions [0.5 pts]
2. Use the given Matlab implementation (associationRules.m) and add to it the code where you determine the association rules from the determined frequent item sets. Your implementation must include the anti-monotone property in slide 44. [7 pts]
3. Compare the time needed to generate all association rules when you implement the anti[U+2010]monotone property in slide 44 versus when such property is ignored. [1 pt]
4. How many rules satisfy the following input parameter values? support = 0.001, confidence = 0.8. Sort the association rules by confidence in descending order and list the top 30 rules. [0.5 pt]
5. Deliverable:
 - (a) Complete Matlab script that include the code requested in point 2 above.
 - (b) A report that include the answers to the rest of the questions. [1 pt for clarity]

2 Solution

2.1 Histogram

The following function, 'plotHistogram' function is responsible for plotting the frequency of each item in the 9835 transactions provided in groceries.txt. The resulting histogram is as shown in Figure 1.

```
function r = plotHistogram(data)
    figure('units','normalized','outerposition',[0 0 1 1],'NumberTitle','off','Name','PRODUCTS
        FREQUENCY')
    h = histogram(categorical([data{:}]),'Orientation','vertical',...
        'facecolor',[1 0.301960784313725 0])
    xtickangle(90)
    xlabel(['\fontsize{16}PRODUCTS'])
    ylabel(['\fontsize{16}FREQUENCY'])
    set(gca,'FontSize',8);
    set(gca,'FontName','Times New Roman');
    h.DisplayOrder = 'descend';
    title(['\fontsize{16}PRODUCT TRANSACTION FREQUENCIES'])
    grid on;
end
```

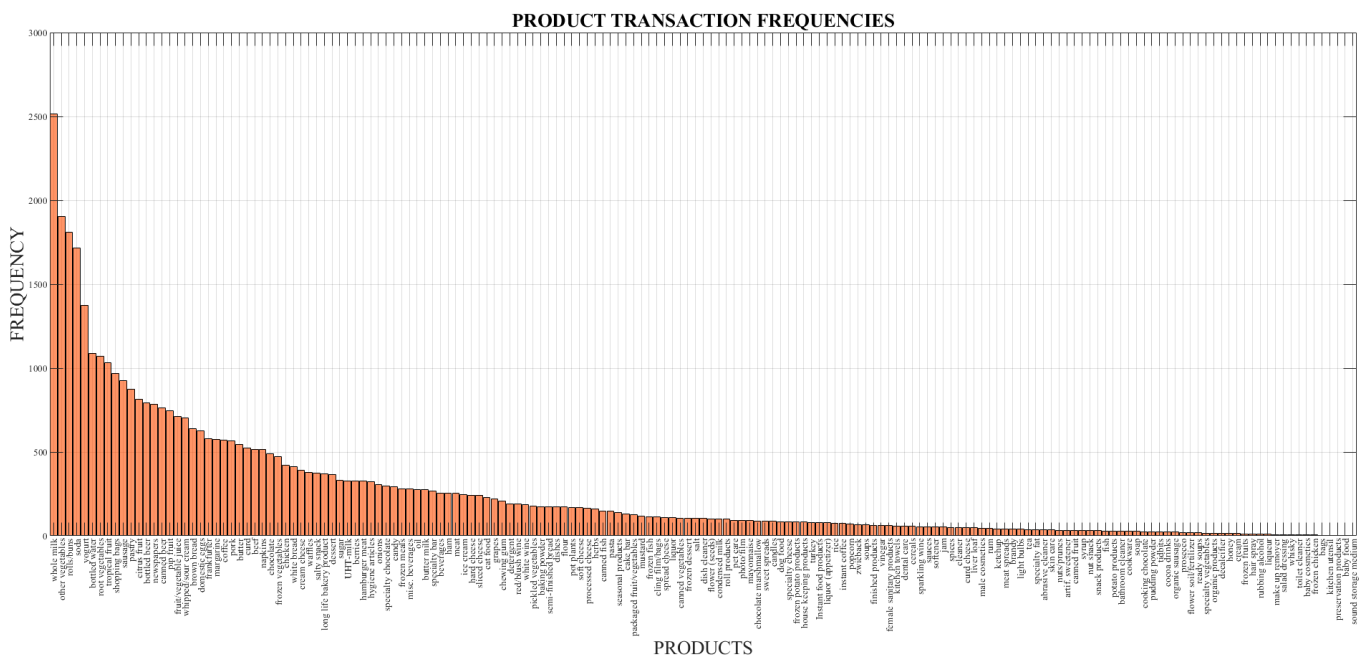


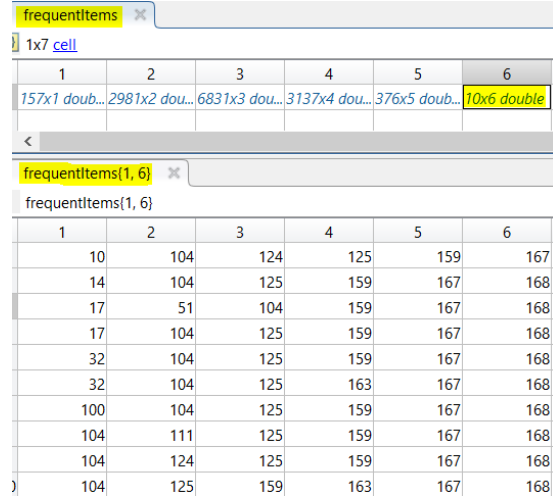
Figure 1: Product frequency plot

2.2 Frequent Items and Support

Frequent items and support are two data-structure provided and their functionalities are as follows,

Frequent Items: Data-structure that contains items or group of items that are frequent in the data-set, i.e the items or set of items that has a support value greater than the minimum support (0.001). A sample structure of this data-structure is shown in Figure 2.

Support: Data-structure that contains support values for every combination of antecedents and consequent derived from the data-set provided. A sample structure of this data-structure is shown in Figure 3.



frequentItems

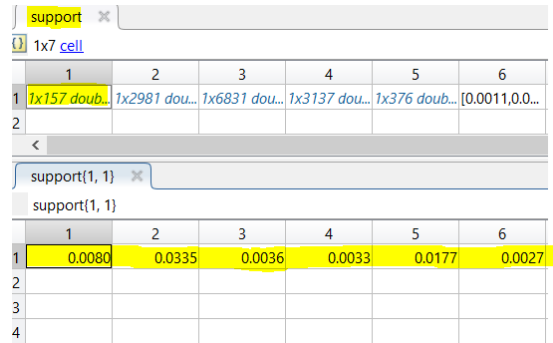
1x7 cell

1	2	3	4	5	6
157x1 doub...	2981x2 dou...	6831x3 dou...	3137x4 dou...	376x5 doub...	10x6 double

frequentItems(1, 6)

1	2	3	4	5	6
10	104	124	125	159	167
14	104	125	159	167	168
17	51	104	159	167	168
17	104	125	159	167	168
32	104	125	159	167	168
32	104	125	163	167	168
100	104	125	159	167	168
104	111	125	159	167	168
104	124	125	159	167	168
104	125	159	163	167	168

Figure 2: Frequency sets



support

1x7 cell

1	2	3	4	5	6
1x157 doub...	1x2981 dou...	1x6831 dou...	1x3137 dou...	1x376 doub...	[0.0011,0.0...

support{1, 1}

1	2	3	4	5	6
0.0080	0.0335	0.0036	0.0033	0.0177	0.0027

Figure 3: Support values

2.3 Generating the Association rules

The function generateAssociation function creates a set of association rules based on the frequent set of items, support matrix and minimum confidence value provided as parameters (0.8). The Association rules can be generated in two ways,

1. Brute force approach
2. Anti-monotone approach

Brute force approach: In the brute force approach, every antecedent is independently checked for association with a consequent. We know that the confidence decreases as the number of consequent elements increases.

$$confidence(ABC \rightarrow D) \geq confidence(AB \rightarrow CD) \geq confidence(A \rightarrow BCD) \quad (1)$$

Since the brute force process ignore the above rule, it will undertake several unnecessary iterations that could have been avoid.

Anti-Monotone: The Anti-Monotone will consider the above mentioned rule of confidence and will prune the entire set of subsets belonging to a rejected association rule. This reduces the number of association rules to be checked for confidence, in turn increasing the performance of the code without any loss of accuracy.

Running the simulation multiple times, the amount of time taken by Brute force and Anti-Monotonic methods are as shown in 1

Brute Force Method	Anti-Monotone
28.5 - 30.0 seconds	20-22 seconds
413 Association Rules	413 Association Rules

Table 1: Time taken and number of Association rules generated

2.4 Brute Force approach (without Anti-Monotone) implementation

The implementation of the brute force technique is as shown below,

```
function generateAssociation(frequentItems, support, minconf)
tic
% Data-Structure to store the association rules that satisfy min-confidence
associationRules = {};
% Iterate over the Frequent Items level in reverse order
for i=size(frequentItems,2)-1:-1:2
    % Iterate over the Frequent items
    for k = 1:size(frequentItems{1,i},1)
        % Store the Frequent Item at level 'i' and Position 'k'
        row = frequentItems{1,i}(k,:);
        % Get the support value for the antecedent (sup(AUB) or numerator)
        support_nume = support{1, size(row, 2)}(k);
        % Iterate over row to generate association rules
        for j=length(row)-1:-1:1
            % Create subsets of row with size 'j'
            subsets = nchoosek(row,j);
            % Iteration over the subsets
            for x = 1:size(subsets,1)
                % Get the antecedent
                X_S = subsets(x,:);
                % Get the Frequent Item set that belongs to antecedent
                % Ex: if antecedent = [1, 2]; FrequentItem(2) will be used
                fi_X_S = frequentItems{size(X_S,2)};
                % Find the index of the antecedent in the FrequentItem
                % Use the Index found to fetch the support of antecedent
                support_X_S = support{size(X_S,2)}(ismember(fi_X_S, X_S, 'rows'));
                % Calculate confidence value
                confidence = round(support_nume/support_X_S,2);
                % If confidence is greater than minimum confidence store it
                % in the associationRule Data-structure as results
                if confidence >= minconf
                    associationRules = [associationRules;[{X_S}, ...
                        {setdiff(row,X_S)}, {confidence}]];
                end
            end
        end
    end
end
toc
% Display the top 30 association rules
show_top_30(associationRules);
end
```

Listing 1: Brute Force complete code implementation

The code is supplied with code comments that are sufficient to explain most of the simple steps, some key aspects of this implementation are as follows,

subsets = nchoosek(row,j) The code snippet is responsible for generating the subsets of a parent set. Consider the example,

```

FrequentItem = [10, 104, 124]
j = 1
subsets = nchoosek(FrequentItem,j)
# Subset
# [10]
# [104]
# [124]
j = 2
subsets = nchoosek(FrequentItem,j)
# Subset
# [10, 104]
# [10, 124]
# [104, 124 ]
and so on,,,
Note: If we take the first subset element [10] and FrequencyItem [10, 104, 124],
we can visualize them as an association rule as follows, [10, 104, 124] => [10]

```

Calculate Confidence

```

%.....
row = frequentItems{1,i}(k,:);
% Get the support value for the antecedent (sup(AUB) or numerator)
support_nume = support{1, size(row, 2)}(k);
%.....
%.....
%.....
for x =1:size(subsets,1)
    X_S = subsets(x,:);
    fi_X_S = frequentItems{size(X_S,2)};
    support_X_S = support{size(X_S,2)}(ismember(fi_X_S, X_S, 'rows'));
    confidence = round(support_nume/support_X_S,2);
    if confidence >= minconf
        associationRules = [associationRules;[{X_S}, ...
            {setdiff(row,X_S)}, {confidence}]];
    end
end

```

A confidence of an association $A,B \Rightarrow C$ is determined by $\text{Confidence} = \text{sup}(AUBUC)/\text{sup}(AUB)$. In the above code **X_S** represents the AUB component or the transaction count of A and B together without C in it. The support for this component is found by,

1. Find the FrequentItem Data-Structure to which X_S belongs using its size.
2. Find the position of the X_S in FrequentItem Data-structure using *ismember(fi_X_S, X_S, 'rows')*
3. Use the index derived from step-2 to derive the support value of X_S from the support data-structure using, *support{size(X_S,2)}(<Index From Step2>)*

The support value of antecedent or AUBUC is found at a parent level, as the antecedent does not change for its subsequent subsets. This is an optimization used to reduce the fetch from FrequentItems data-structure. The steps to find the support value is as follows,

1. Directly fetch the support value using *support1, size(row, 2)(k)*; where k is the index at which the support values for the given antecedent is stored.

The above step works, as FrequentItems and support data-structure are related by index, i.e a Frequent-item at index 'i' will have its support value present in index 'i' in the support data-structure.

Once we determine the support values of AUBUC and AUB, we find the confidence by using the formula,

$$\text{confidence} = \frac{\text{sup}(AUBUC)}{\text{sup}(AUB)} \quad (2)$$

The code which is implementing the formula is, *round(support_nume/support_X_S,2)*

If the confidence value obtained is greater than the minimum confidence, we store the association in a data-structure as a result. And the process is continued until every association is checked. The confidence value is rounded by 2 decimal points for better accuracy.

2.5 Anti-Monotone

The Anti-Monotone implementation is exactly similar to the Brute-force approach but with the addition of the following code snippets,

```
function generateAssociation(frequentItems, support, minconf)
associationRules = {};
    %%%%%%%%%% ITERATIONS TO ACCESS FREQUENT ITEMS %%%%%%%%%%
    % Create a list to store the subsets with confidence less than min-conf
    % The subsets will be pruned and it is discontinued susequent stages
    pruneList = [];
    for j=length(row)-1:-1:1
        subsets = nchoosek(row,j);
        if isempty(pruneList) isSubset = []; ...
        else isSubset = ismember(subsets, pruneList{1}); end
        subsets(all(isSubset==1,2),:) = [];
        for x = 1:size(subsets,1)
            %%%%%%%%%% COMPUTE CONFIDENCE %%%%%%%%%%
            if confidence >= minconf
                associationRules = [associationRules; [{X_S}, {setdiff(row,X_S)}, {confidence}]];
            else
                pruneList{end+1,1} = X_S;
            end
        end
    end
end
end
end
```

Listing 2: Anti-Monotone key logic

pruneList: A collection to hold the antecedent that have not met the minimum confidence and have been rejected or pruned. This collection will be looked up to avoid computing associations rules for subsets of pruned antecedent's.

Prune Subsets: Prune from the collection of subsets, the subsets that belong to rejected antecedents. The code snippet used to implement this is as follows,

```
if isempty(pruneList) isSubset = []; ...
else isSubset = ismember(subsets, pruneList{1}); end
subsets(all(isSubset==1,2),:) = [];
```

The logic is built over matrix operations to avoid the introduction of another loop, the steps involved are,

1. Check if the subsets generated is a subset in the pruneList using *ismember(subsets, pruneList{1})*, this will generate a matrix with 0's and 1's.
2. If all the elements in a row of the matrix generated in step-1 is 1's, then prune the row from subsets at that row index using, *subsets(all(isSubset==1,2),:) = [];*

After the pruning, the process is similar to the Brute-Force model where the support values for {**antecedent**} and {**antecedent - consequent**} is determined, and the confidence value is calculated.

A decision is made based on the obtained confidence value,

$$\text{decision} = \begin{cases} \text{Store the Association rule in the AssociationRule result data-structure,} & \text{if confidence} \geq \text{min-confidence} \\ \text{Store the antecedent in the pruneList,} & \text{Otherwise} \end{cases}$$

This is achived by simple if and else statements,

```
if confidence >= minconf
    associationRules = [associationRules; [{X_S}, {setdiff(row,X_S)}, {confidence}]];
else
    pruneList{end+1,1} = X_S;
end
```

And the process is continued until all the unpruned subsets have been tried for minimum confidence.

The design used to implement the Anti-Monotone association rule mining is as follows,

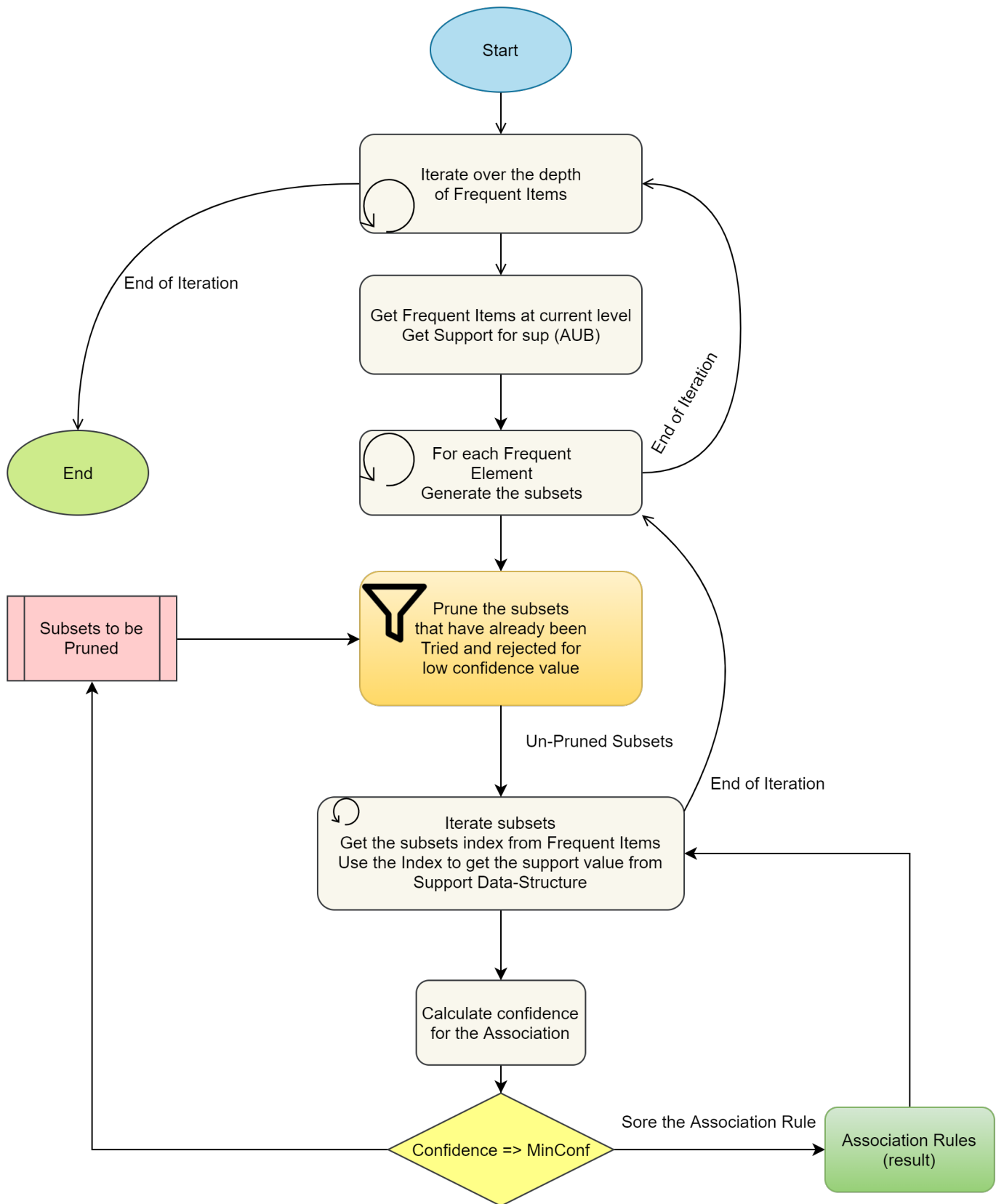


Figure 4: Anti-Monotone Association Rule Flow chart

The complete implementation of the Anti-Monotone association rule mining is as follows,

```
function generateAssociation(frequentItems, support, minconf)
tic
% Data-Structure to store the association rules that satisfy min-confidence
associationRules = {};
% Iterate over the Frequent Items level in reverse order
for i=size(frequentItems,2)-1:-1:2
    % Iterate over the Frequent items
    for k =1:size(frequentItems{1,i},1)
        % Store the Frequent Item at level 'i' and Position 'k'
        row = frequentItems{1,i}(k,:);
        % Get the support value for the antecedent (sup(AUB) or numerator)
        support_nume = support{1, size(row, 2)}(k);
        % Create a list to store the subsets with confidence less than min-conf
        % The subsets will be pruned and it is discontinued susequent stages
        pruneList = [];
        % Iterate over row to generate association rules
        for j=length(row)-1:-1:1
            % Create subsets of row with size 'j'
            subsets = nchoosek(row,j);
            % Prune the subsets from the list of subsets generated with
            % confidence less than min-confidence
            if isempty(pruneList) isSubset = []; else isSubset = ismember(subsets, pruneList{1})
                ; end
            subsets(all(isSubset==1,2),:) = [];
            % Iteration over the subsets that are not pruned
            for x =1:size(subsets,1)
                % Get the antecedent
                X_S = subsets(x,:);
                % Get the Frequent Item set that belongs to antecedent
                % Ex: if antecedent = [1, 2]; FrequentItem(2) will be used
                fi_X_S = frequentItems{size(X_S,2)};
                % Find the index of the antecedent in the FrequentItem
                % Use the Index found to fetch the support of antecedent
                support_X_S = support{size(X_S,2)}(ismember(fi_X_S, X_S, 'rows'));
                %Calculate confidence value
                confidence = round(support_nume/support_X_S,2);
                % If confidence is greater than minimum confidence store it
                % in the associationRule Data-structure as results
                if confidence >= minconf
                    associationRules = [associationRules;[{X_S}, {setdiff(row,X_S)}, {confidence}
                        ]];
                else
                    % If the confidence is less than the min-confidence
                    % add an entry in the prune list to skip the
                    % computation of its subset association rules.
                    pruneList{end+1,1} = X_S;
                end
            end
        end
    end
end
toc
% Display the top 30 association rules
show_top_30(associationRules);
end
```

Listing 3: Anti-Monotone complete code implementation

2.6 Top 30 association rules sorted by confidence in descending order

2.7 Implementation

The function show_top_30 creates provides a list of 30 association rules sorted on the basis of their confidence values in the descending order.

```
function show_top_30(associationRules)
    % Open the dataset file
    fid = fopen('myDataFile.csv');
    % Get the header of the CSV file that contains the item names
    items = textscan(fid,'%s',1);
    % close the file
```



```

fclose(fid);
% Split the header by delimiter comma to obtain the list of item names
header = split(items{1},',');
% Sort the association rules in descending order
sortedRes = sortrows(associationRules,3,'descend');
% pick the top 30 association rules from the sorted list
sortedRes = sortedRes(1:30,:);
% For each of the picked association rules
for i=1:size(sortedRes,1)
    % retrieve the original name of the item from the header extracted
    % from the dataset file and concatenate it to a string
    antecedent = sprintf(' %s,' , header{sortedRes{i,1}});
    consequent = sprintf(' %s,' , header{sortedRes{i,2}});
    % Get the confidence value for the antecedent => consequent
    confidence = sprintf(' %f' , sortedRes{i,3});
    % Print the association Rule
    fprintf("{%s} => {%s} ::: Confidence = %s \n", antecedent(1:end-1), consequent(1:end-1),
        "+"confidence)
end
fprintf("The Number of Association that satisfy the input constraints is = %d", size(
    associationRules,1));
end

```

Listing 4: Code to display the top 30 association rules

2.8 Results

```

>> associationRules(0.001, 0.8)
Elapsed time is 28.525089 seconds.
{ oil, other_vegetables, root_vegetables, tropical_fruit, yogurt } => { whole_milk } ::: Confidence
= 1.000000
{ bottled_water, other_vegetables, pip_fruit, root_vegetables } => { whole_milk } ::: Confidence =
1.000000
{ butter, domestic_eggs, other_vegetables, whipped_sour_cream } => { whole_milk } ::: Confidence =
1.000000
{ butter, fruit_vegetable_juice, tropical_fruit, whipped_sour_cream } => { other_vegetables } :::
Confidence = 1.000000
{ butter, other_vegetables, pork, whipped_sour_cream } => { whole_milk } ::: Confidence = 1.000000
{ butter, other_vegetables, root_vegetables, white_bread } => { whole_milk } ::: Confidence =
1.000000
{ citrus_fruit, root_vegetables, tropical_fruit, whipped_sour_cream } => { other_vegetables } :::
Confidence = 1.000000
{ citrus_fruit, pastry, rolls_buns, whipped_sour_cream } => { whole_milk } ::: Confidence =
1.000000
{ grapes, tropical_fruit, whole_milk, yogurt } => { other_vegetables } ::: Confidence = 1.000000
{ ham, pip_fruit, tropical_fruit, whole_milk } => { other_vegetables } ::: Confidence = 1.000000
{ ham, pip_fruit, tropical_fruit, yogurt } => { other_vegetables } ::: Confidence = 1.000000
{ newspapers, rolls_buns, soda, whole_milk } => { other_vegetables } ::: Confidence = 1.000000
{ oil, other_vegetables, root_vegetables, yogurt } => { whole_milk } ::: Confidence = 1.000000
{ oil, root_vegetables, tropical_fruit, yogurt } => { whole_milk } ::: Confidence = 1.000000
{ rolls_buns, root_vegetables, sausage, tropical_fruit } => { whole_milk } ::: Confidence =
1.000000
{ brown_bread, pip_fruit, whipped_sour_cream } => { other_vegetables } ::: Confidence = 1.000000
{ butter, domestic_eggs, soft_cheese } => { whole_milk } ::: Confidence = 1.000000
{ butter, hygiene_articles, pip_fruit } => { whole_milk } ::: Confidence = 1.000000
{ butter, rice, root_vegetables } => { whole_milk } ::: Confidence = 1.000000
{ citrus_fruit, root_vegetables, soft_cheese } => { other_vegetables } ::: Confidence = 1.000000
{ cream_cheese_, domestic_eggs, napkins } => { whole_milk } ::: Confidence = 1.000000
{ cream_cheese_, domestic_eggs, sugar } => { whole_milk } ::: Confidence = 1.000000
{ curd, domestic_eggs, sugar } => { whole_milk } ::: Confidence = 1.000000
{ flour, root_vegetables, whipped_sour_cream } => { whole_milk } ::: Confidence = 1.000000
{ hygiene_articles, pip_fruit, root_vegetables } => { whole_milk } ::: Confidence = 1.000000
{ hygiene_articles, root_vegetables, whipped_sour_cream } => { whole_milk } ::: Confidence =
1.000000
{ canned_fish, hygiene_articles } => { whole_milk } ::: Confidence = 1.000000
{ rice, sugar } => { whole_milk } ::: Confidence = 1.000000
{ root_vegetables, sausage, tropical_fruit, yogurt } => { whole_milk } ::: Confidence = 0.940000
{ cream_cheese_, other_vegetables, sugar } => { whole_milk } ::: Confidence = 0.940000
-----
The Number of Association that satisfy the input constraints is = 413

```

Listing 5: Top 30 Association rules without Anti-monotone

```

>> associationRulesAntinMonotone(0.001, 0.8)
Elapsed time is 20.027905 seconds.
{ oil, other_vegetables, root_vegetables, tropical_fruit, yogurt} => { whole_milk} ::: Confidence
    = 1.000000
{ bottled_water, other_vegetables, pip_fruit, root_vegetables} => { whole_milk} ::: Confidence =
    1.000000
{ butter, domestic_eggs, other_vegetables, whipped_sour_cream} => { whole_milk} ::: Confidence =
    1.000000
{ butter, fruit_vegetable_juice, tropical_fruit, whipped_sour_cream} => { other_vegetables} :::
    Confidence = 1.000000
{ butter, other_vegetables, pork, whipped_sour_cream} => { whole_milk} ::: Confidence = 1.000000
{ butter, other_vegetables, root_vegetables, white_bread} => { whole_milk} ::: Confidence =
    1.000000
{ citrus_fruit, root_vegetables, tropical_fruit, whipped_sour_cream} => { other_vegetables} :::
    Confidence = 1.000000
{ citrus_fruit, pastry, rolls_buns, whipped_sour_cream} => { whole_milk} ::: Confidence =
    1.000000
{ grapes, tropical_fruit, whole_milk, yogurt} => { other_vegetables} ::: Confidence = 1.000000
{ ham, pip_fruit, tropical_fruit, whole_milk} => { other_vegetables} ::: Confidence = 1.000000
{ ham, pip_fruit, tropical_fruit, yogurt} => { other_vegetables} ::: Confidence = 1.000000
{ newspapers, rolls_buns, soda, whole_milk} => { other_vegetables} ::: Confidence = 1.000000
{ oil, other_vegetables, root_vegetables, yogurt} => { whole_milk} ::: Confidence = 1.000000
{ oil, root_vegetables, tropical_fruit, yogurt} => { whole_milk} ::: Confidence = 1.000000
{ rolls_buns, root_vegetables, sausage, tropical_fruit} => { whole_milk} ::: Confidence =
    1.000000
{ brown_bread, pip_fruit, whipped_sour_cream} => { other_vegetables} ::: Confidence = 1.000000
{ butter, domestic_eggs, soft_cheese} => { whole_milk} ::: Confidence = 1.000000
{ butter, hygiene_articles, pip_fruit} => { whole_milk} ::: Confidence = 1.000000
{ butter, rice, root_vegetables} => { whole_milk} ::: Confidence = 1.000000
{ citrus_fruit, root_vegetables, soft_cheese} => { other_vegetables} ::: Confidence = 1.000000
{ cream_cheese_, domestic_eggs, napkins} => { whole_milk} ::: Confidence = 1.000000
{ cream_cheese_, domestic_eggs, sugar} => { whole_milk} ::: Confidence = 1.000000
{ curd, domestic_eggs, sugar} => { whole_milk} ::: Confidence = 1.000000
{ flour, root_vegetables, whipped_sour_cream} => { whole_milk} ::: Confidence = 1.000000
{ hygiene_articles, pip_fruit, root_vegetables} => { whole_milk} ::: Confidence = 1.000000
{ hygiene_articles, root_vegetables, whipped_sour_cream} => { whole_milk} ::: Confidence =
    1.000000
{ canned_fish, hygiene_articles} => { whole_milk} ::: Confidence = 1.000000
{ rice, sugar} => { whole_milk} ::: Confidence = 1.000000
{ root_vegetables, sausage, tropical_fruit, yogurt} => { whole_milk} ::: Confidence = 0.940000
{ cream_cheese_, other_vegetables, sugar} => { whole_milk} ::: Confidence = 0.940000
-----
The Number of Association that satisfy the input constraints is = 413
-----

```

Listing 6: Top 30 Association rules with Anti-monotone

The time taken to generate the Association Rules **without Anti-monotone** = **28.5 to 30 seconds (Approx)**.

The time taken to generate the Association Rules with **Anti-monotone** = **20 to 22 seconds (Approx)**.

As shown in the Listings 5 and 6, we can observe that both techniques created exactly **413 association rules**, and the count is in agreeance with the result generated from the Weka tool.

3 Conclusion

The Anti-Monotone helps in reducing the number of iterations performed on the data-set, but more importantly avoid the computation of unnecessary association rules candidates. On a smaller data-set (9835 transactions) we see a considerable gain in performance over the magnitude of few seconds, this difference in performance between Brute-force technique and Anti-Monotone technique is bound to widen as the size of the data-set increases. To conclude, there is no arguments in favour to be use the Brute-force approach over the Anti-Monotone approach.

4 GitHub

[Assignment-4 GitHub Link](#)