Mining frequents itemset and association rules in diabetic dataset

Abdelfatah MAAROUF, Youssef FAKIR, Rachid ELAYACHI

Sultan Moulay Slimane University, Faculty of Sciences and Technics, Beni Mellal, Morocco

Email: info.dec07@yahoo.fr, rachideea@yahoo.fr

***Abstract*—** Data mining is a field of science that unifies techniques from machine learning, pattern processing, statistics, databases, and visualization to handle the problem of retrieving information from large databases. Association rule mining (ARM) is a process of extracting frequent patterns, correlations, and associations between sets of data items in the databases. Several association rule based on diabetes diagnosis and prediction techniques are presented in the literature.
This paper's aim is the extraction of association rules by Frequent Pattern Growth (FP-Growth) algorithm and its variants using a diabetic data set.

***Keywords*—** Data mining, Association rules, frequent patterns, FP-Growth, CFP-Growth, ICFP-Growth.

1. Introduction

Data mining represents several techniques for extracting knowledge form large databases. a lot data mining approaches are being used to extract interesting knowledge, such as , association rule mining techniques extracting association between the entities, clustering techniques to group the unlabeled data into clusters, classification techniques to recognize the different classes existing in categorical labeled data. In this paper we are interested in the association rules methods, especially the FP-Growth based algorithms, we are going to explain how to extract itemset using the FP-growth [1] Algorithm and his based algorithms: CFP-growth and ICFP growth, and also, we will make a com parative study between those three algorithms. The remaining of the paper is organized as follows. In section 1, we will explain the FP-Growth Algorithm with an example, and in section 2 we are going to introduces the FP-Growth-based algorithm called CFP-Growth. Then, we will illustrate another FP-Growth-based algorithm, the ICFP-Growth algorithm, in Section 3. In Section 4, we will make a comparative study between classification models that we implement using the three algorithms, Finally, the conclusion is drawn in Section 5.

2. FP-growth

The FP-Growth is an Algorithm used to find frequent itemset without using candidate generations, thus improving performance. The special thing in this method is the usage of a data structure named frequent-pattern tree (FP-tree), which contain the itemset association information. This algorithm works in this way: first, it compresses the input database by constructing an FP-tree instance to represent frequent items. After this step, it split the compressed database into a set of conditional databases, each one associated with one frequent pattern. Finally, each such database is mined separately. Using this strategy, the FP-Growth reduces the search costs by looking for short patterns recursively and then concatenating them in the long frequent patterns, offering good selectivity [2]. An FP-tree consists of one root labeled as "null", a set of item prefix subtrees as the children of the root and a frequent-item header table. Each node in the prefix subtree consists of three fields: item-name, count and node-link. The count of a node records the number of transactions in the database that share the prefix represented by the node, and node-link links to the next node in the FP-tree carrying the same item-name. Each entry in the frequent-item header table consists of two fields: item-name and head of node-link, which points to the first node in the FP-tree carrying the item-name. Besides, the FP-tree assumes that the items are sorted in decreasing order of their support counts, and only frequent items are included.

After the FP-tree is built, the FP-growth algorithm recursively builds conditional pattern base and conditional FP-tree for each frequent item from the FP-tree and then uses them to generate all frequent itemset[3]

Based on this definition, we have the following FP-tree construction algorithm.

Algorithm 1 (FP-tree construction).
Input: A transaction database DB and a minimum support threshold ξ .
Output: FP-tree, the frequent-pattern tree of DB. Method: The FP-tree is constructed as follows.

1. Scan the transaction database DB once. Collect F, the set of frequent items, and the support of each frequent item. Sort F in support-descending order as FList, the list of frequent items.
2. Create the root of an FP-tree, T , and label it as "null". For each transaction Trans in DB do the following. Select the frequent items in Trans and sort them according to the order of FList. Let the sorted frequent-item list in Trans be [p | P], where p is the first element and P is the remaining list. Call insert tree([p | P], T).
The function insert tree([p | P], T) is performed as follows. If T has a child N such that N.item-name = p.item-name, then increment N's count by 1; else create a new node N, with its count initialized to 1, its parent link linked to T , and its node-link linked to the nodes with the same item-name via the node-link structure. If P is nonempty, call insert tree(P, N) recursively.

2.1. *Exemple.*

Based on the algorithm we will apply the FP-growth on a small transaction data base that contain 9 transactions with 5 item such as: Asparagus (A), Corn (C), Beans (B), Tomatoes (T) & Squash (S)

Table 1: transaction database.

Transaction ID	List of items in the transaction
T1	B, A, T
T2	A, C
T3	A, S
T4	B, A, C
T5	B, S
T6	A, S
T7	B, S
T8	B, A, S, T
T9	B, A, S

Therefore, for example, for the first transaction T1 consists of three items such as Beans (B), Asparagus (A), and Tomatoes (T). Similarly, the transaction T6 contains the items Asparagus (A) and Squash (S). Let us also consider the minimum support for this small transaction data to be 2. Hence, min_support = 2. First, we need to create a table of item counts to all transactional database as shown in Table 2:

Table 2: support count item.

Item	Support Count	Ordered Item
Beans (B)	6	Asparagus (A)
Asparagus (A)	7	Beans (B)
Squash (S)	6	Squash (S)
Corn (C)	2	Corn (C)
Tomatoes (T)	2	Tomatoes (T)

This is simply the count of each item, such as if we see Squash (S) has been bought in 6 transactions viz, T3, T5, T6, T7, T8 & T9, so the support count is 6 for Squash. Next, we just need to sort the item list in descending order for their support count. Hence, the table of support count may now be as represented in Table 2.
Beans (B) & Squash (S) have the same support count of 6 and any of them can be written first. Here, we have written Beans (B) first. Similarly, Corn(C) and tomatoes (T) can also be listed in the same fashion. Now we are ready to start with the construction of the FP Tree. The FP tree root node is usually represented with a NULL root node. Now consider the Transaction T1. T1 consists of Beans (B), Asparagus (A) and tomatoes (T). Now out of these three items, we need to look for the item which has

the maximum support count. Since Asparagus (A) has the highest support count of 7, we will extend the tree from its root node to A as Asparagus. Since this is the first transaction, the count is denoted by A:1. Let's look further, out of Beans(B) and Tomatoes (T) the support count of Beans is 6, and the support count for Tomatoes is 2. From Asparagus, we can extend the tree to Beans (B:1 for the first transaction consisting of beans) and after that T:1 for Tomatoes. The process of tree construction is as shown in Fig.1 (a,b,c,d,e).

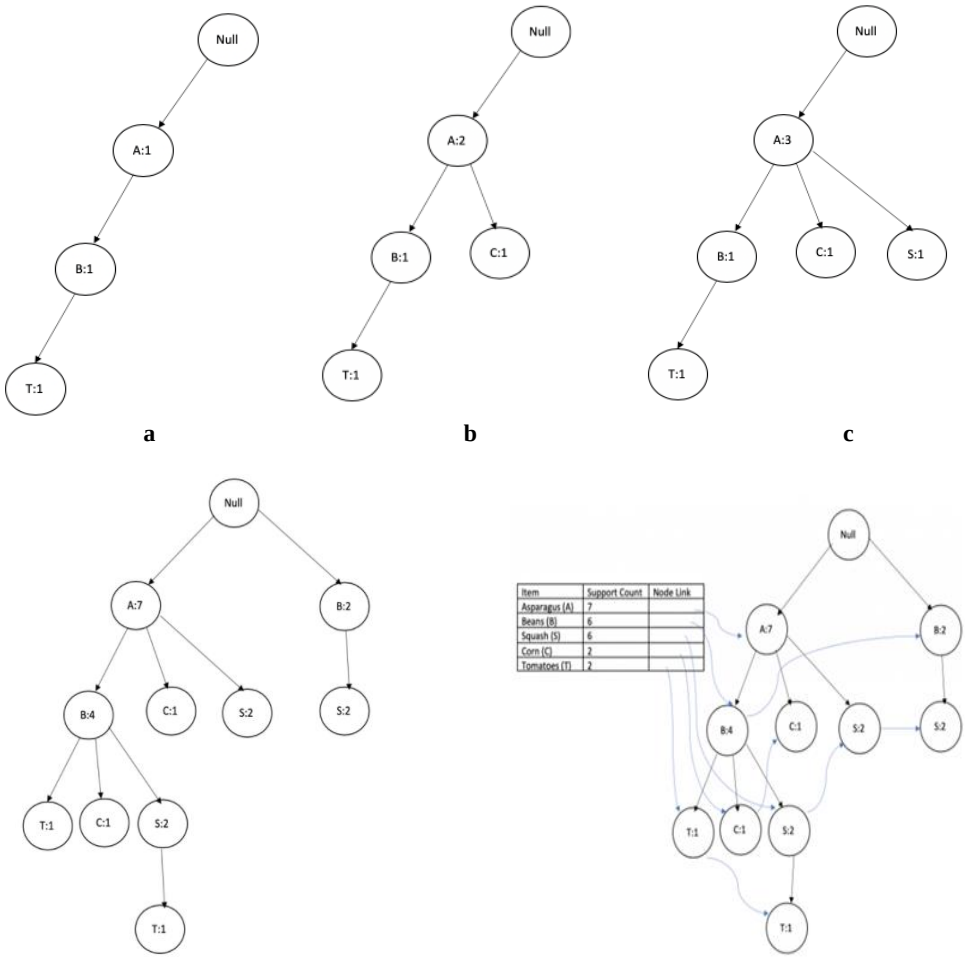

Fig.1. Process of FPtree construction

Going further in the transaction T2, there are two items viz Asparagus(A) and Corn (C). Since the support count for Asparagus is 7 and Corn (C) is 2, we need to consider Asparagus (A) first and going from the root node we need to see if there is any branch that is extended to Asparagus (A) which is true in this case, and we can increase the count as A:2 (Fig.2). But after that, there is no node connected to Asparagus (A) to corn (C) we need to create another branch for Corn as C:1. Similarly, for transaction T3 we have Asparagus (A) then Squash (S) in the descending order of their support count. So, Asparagus(A) count has been increased from A:2 to A:3, and further, we can see that there aren't any nodes for Squash (S) from Asparagus (A), so we need to create another branch going for a Squash node S:1, as described in Fig.1(c). The same process for the other transactions. In the end, we have this tree (Fig.1 (d)).

Now we can cross-verify using Fig.1(d) and Table 3. The count for Asparagus (A) stands at A:7, which is similar in Table 3. If we verify for Beans, there are two nodes B:4 and B:2 hence, the count matches with Beans count of 6 in Table 3. Now we can merge the last tree (figure 4) with a table that contain the items, support count and node link to have the last version of the FP-tree (Fig.1(e). Now we need to construct a table for conditional pattern base and hence, the frequent pattern generation. Now let us

consider the table in Fig.1(e), we need to consider the item, which has the lowest support count, in our case, Tomatoes (T) has the lowest support count of 2. We need to see where the tomatoes are. We can see there are two traversal paths for tomatoes (T) from the root node. One of them being A-B-T, where T is having count 1 i.e., T:1. So we can create the conditional pattern base as in Row 1 of the Table 4 {A, B:1}.

Also, the second traversal path is A-B-S-T and T has a count of 1 (T:1) so the conditional pattern base is {A, B, S;1}. So there are two conditional pattern bases {{A, B:1},{A,B,S:1}}. Now from this, we will construct the conditional FP tree column in Table 4. Here A is in {A, B:1} and {A, B, S:1} so both have a final count of 1 each summing up to <A:2>, in the similar fashion <B:2> and <S:1> . Finally it comes out to be <A:2, B:2, S:1>, but we will not consider S:1 since we have taken the minimum support count to be 2. So finally, the value should be <A:2, B:2> for the first row i.e., Tomatoes (T).

Now to construct the frequent pattern generation (the last column for table 4) we need to join the Conditional FP tree column with Item in Table 4. Joining <A:2> with Tomatoes (T) we can write {A, T:2}, after that join <B:2> with tomatoes (T) it comes out to be {B, T:2}. Then we need to join A and B both with T hence {A, B, T:2} so the frequent pattern generation for the row of Tomatoes (T) becomes {A, T:2}, {B, T:2}, {A, B, T:2}.

Here both the (A) and (B) have a count of 2 so we have written {A, B, T:2}, but sometimes the case may arise that and B have different counts, in that case, we need to consider the minimum count of the item and write in the same fashion.

Now for the Corn (C) rows, we can see in Fig.5 that we can reach Corn (C) through two different paths A-B-C and A-C. In both the traversal paths, the count for Corn (C) is 1. Hence, {{A, B:1}, {A:1}} is our conditional pattern base. Now for the Conditional FP tree column, we need to check for a count of each item in the conditional pattern base, for we have 2 as an account of A is 1 as count of B. Since the minimum support count that we have considered is 2, we need to neglect B. So, the conditional FP Tree columns stands to be <A:2>. After this, join Conditional FP tree column with Item column for Corn (C) which comes out to be {A, C:2}

Now consider the Squash (S) row. We can reach Squash through 3 traversal paths A-B-S, A-S, and B-S. Since the count of Squash is 2 in A-B-S we can write {A, B:2} similarly, for the other two paths we can write {A;2} & {B:2} so the Conditional pattern base stands at {{A, B:2}, {A:2}, {B:2}}. For conditional FP tree columns, we can see that <B:2> has been reached from the right branch from the Null node and the other two through the left branch originating from the Null root.

Hence, we need to write differently and not add in one, we can write <A:4, B:2>, <B:2>. Now joining A:4 from <A:4, B:2> with Squash (S) gives {A, S:4}. Now from <A:4, B:2> joining with Squash (S) gives, {B, S:2} but we have written {B, S:4}. We have written {B, S:4} the count as 4 since we have another <B:2> in the Conditional FP tree column. Summing both B with S from <A:4, B:2> and B with S from <B:2> the count comes out to be {B, S:4}. Now joining <A:4, B:2> with S (the final joining all items) we get {A, B, S:2} not {A, B, S:4} since we need to consider the minimum count which is 2. Taking the minimum count is required since we need to check the frequent counts where both A & B occur with S not only one of the items. These three items, Asparagus (A), Beans (B), Squash (S) occur only twice in this case so {A, B, S:2}.
Similarly, the last row for Bean is constructed. In addition, Asparagus is directly from the Null node and since there are not any in-between nodes to reach Asparagus (A), there is no need to go for another row of Asparagus.

Table 4: Frequent pattern generation.

Item	Conditional Pattern base	Conditional FP-tree	Frequent Pattern Generation
Tomatoes (T)	{{A, B: 1}, {A, B, S: 1}}	<A: 2, B: 2>	{A, T: 2}, {B, T: 2}, {A, B, T: 2}
Corn (C)	{{A, B: 1}, {A: 1}}	<A: 2>	{A, C:2}
Squash (S)	{{A, B: 2}, {A :2}, {B: 2}}	<A: 4, B:2>, <B:2>	{A, S: 4}, {B, S: 4}, {A, B, S: 2}
Bean (B)	{{A: 4}}	<A: 4>	{A, B: 4}

3. CFP-growth

The CFP-growth algorithm is developed based on the FP-growth algorithm. Even though it is an FP-growth-like algorithm, the structure, construction and mining procedures of CFP-growth are different from FP-growth. The CFP-growth algorithm accepts transaction database and MIS values of items as an input. Using the items' MIS values as prior knowledge, it discovers complete set of frequent patterns with a single scan on the transaction database. Briefly, the working of CFP-growth is as follows:

i. Items are sorted in descending order of their MIS values. Using the sorted list of items, an FP-tree-like structure known as MIS-tree is constructed with a single scan on the transaction database. Simultaneously, the support of each item in the MIS-tree is measured.

ii. To reduce the search space, tree-pruning operation is performed to prune the items that cannot generate any frequent pattern. The criterion used is prune the items that have support less than the lowest MIS value among all items.

iii. After tree-pruning operation, tree-merging operation is performed on the MIS-tree to merge the child nodes of a parent node that share same item. The resultant MIS-tree is called compact MIS-tree.

iv. Finally, choosing each item in the compact MIS-tree as the suffix item (or pattern), its conditional pattern base (i.e., prefix sub-paths) is built to discover complete set of frequent patterns. Since frequent patterns do not satisfy downward closure property, CFP-growth tries to discover complete set of frequent patterns by building suffix patterns until its respective conditional pattern base is empty [5]**.**

3.1. MIS tree

The MIS-tree is constructed as follows. First, the items are sorted in descending order of their MIS values, say L1 and their frequency values are set at zero. Next, a root node of the tree is constructed by labeling with "null". Next, for each transaction in the dataset the following steps are performed to generate MIS-tree. They are:
i. The items in each transaction are sorted in L1 order. Next, update the support of the items, which are present in the transaction by incrementing the support value of the respective item by 1.
ii. A branch is created for each transaction such that nodes represent the items, level of nodes in a branch is based on the sorted order and the count of each node is set to 1. However, in constructing the new branch for a transaction, the count of each node along a common prefix is incremented by 1, and nodes for the items following the prefix are created, linked accordingly and their values are set to 1[6].
There is a very important difference between the FP-tree and the MIS-tree: the FP-tree only contains frequent items, but the MIS-tree consists of not only all frequent items but also those infrequent items with supports no less than MIN.

3.2. Algorithm

Algorithm 2: (MIS-tree construction)
Input: a transaction database DB and minimum support threshold of each item.
Output: its Multiple Item tree, MIS-tree.

---

1. Create the root of a MIS-tree, T, and label it as "null".

For each Transaction Trans inDB do the follwing:

a. Sort all itmes in Trans according to their MIS-value, in nonincreasing order.
b. Count the support values of any item in Trans.
c. Let the sorted items in Trans be [p|P], where p is the first element and P is the remaining list.

Call insert_tree([p|P, T])

2. Let F denote the set of those items with supports smaller than MIN and f denote an item in F.

for each f in F do the following:

a. Delete the entry in the header table with item-name = f.
b. Call MIS-Pruning(Tree, f).

3. Name the resulting table as MN frequent item header table
4. Call MIS_Merge(Tree).

Procedure **insert_tree([p|P], T)**

While(P is nonempty){
If the achild N such that p.item = N.item-name then N.count++;
Else
Createa new node N, and let count be 1;
Let its parent link be linked to T;
Let its node=link be linked to the nodes with the same item-name via the node-link structure;
}

Function **MIS_Pruning(Tree, a)**{
For each node in the node-link of a in Tree do{
If the node is a leaf then remove the node directly;
Else remove the node and then its parent node will be linked to its child node(s);
}
}
Function **MIS_Merege(Tree)**{
For each iem a in the MIN frequent header table do{
If there are childnode with the same itam-name then
Merge these nodes and set the count as the summation of these nodes counts;
}
}

Algorithem 3 :(CFP-Growth)
Input : MIS-tree, a set of MIN frequent item F, MIS-values
Of each item in F.
Output: the complete set of all f's conditional frequent patterns and the complete set of f's conditional patterns.

Method : **call CFP_growth(MIS-tree, null)**

Procedure **CFP_growth(Tree, f)**{
For each a in header of Tree do{
Generate pattern b = a ∪ a' with support = a.support;
Construct b's conditional pattern base and b's conditional MIS-tree Tree b;
If Tree b≠ ∅ then call CP_growth(Tree b, b, MIS(a'));
}
}

Procedure **CP_growth(Tree, f)**
Input: MIS-tree, MIN frequent item f and its minsup
Output : f's conditional patterns (include the complete set of f's conditioanl frequent petterns and all support values of its subsets).
Method: call CP_growth(Tree, a', MIS(a'))
Procedure **CP_growth(Tree, a, MIS(a'))**{
For each a in the header of Tree do{
Generate pattern b = a ∪ a' with support = a.support;
Construct b's conditional pattern base and then b's conditional MIS-tree Treeb;
If Treeb ≠ ∅ then call CP_growth(Treeb, b, MIS(a'))
}
}

3.3. Exemples

We will consider the transaction database in Table 5, to apply on it the CFP growth algorithm.

Table 5: Transaction database.

TID	Item bought	Item bought (ordered)
1	d, c, a, f	a, c, d, f
3	g, c, a, f, e	a, c, e, f, g
3	b, a, c, f, h	a, b, c, f, h
4	g, b, f	b, f, g
5	b, c	b, c

The MIS value of each item is shown in Table 6. According to Algorithm 1, the order of the items in the MIS-tree is arranged according to their MIS values in non-increasing order. For ease of discussion, the rightmost column of Table 5 lists all the items in each transaction following this order.

Table 6: items MIS values.

Item	a	b	c	d	e	f	g	h
MIS value	4	4	4	3	3	2	2	2

To create the MIS-tree, we first create the root of the tree, labeled as "null". The scan of the first transaction leads to the construction of the first branch of the MIS-tree: ((a:1), (c:1), (d:1), (f:1)). Notice that all items in the transaction would be inserted into the tree according to their MIS values in non-increasing order. The second transaction (a, c, e, f, g) shares the same prefix (a, c) with the existing path (a, c, d, f). So, the count of each node along the prefix is increased by 1 and the remaining item list (e, f, g) in the second transaction would be created as the new nodes. The new node (e:1) is linked as a child of (c:2); node (f:1) as a child of (e:1); node (g:1) as a child of (f:1). For the third transaction (a, b, c, f, h), it shares only the node (a). Thus, a's count is increased by 1, and the remaining item list (b, c, f, h) in the third transaction would be created just like the second transaction. The remaining transactions in DB can be done in the same way.
To facilitate tree traversal, a MIN_frequent item header table is built in which each item points to its occurrences in the tree via the head of node-link. Nodes with the same item-name are linked in sequence via such node-links. After all the transactions are scanned, the tree with the associated node-links is shown in Fig.2.

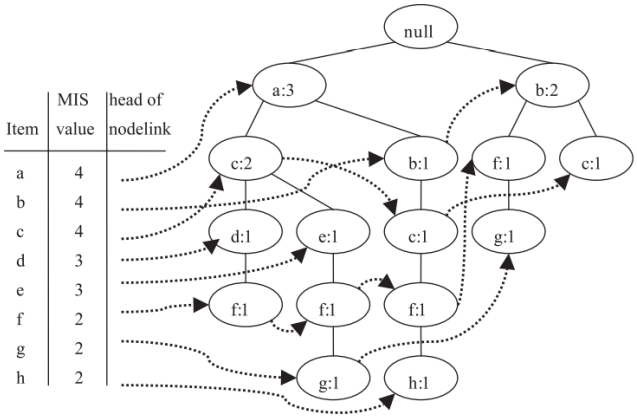

Fig.2. Tree with the associated node-links

After scanning all the transactions, we will get the count of each item as (a:3, b:3, c:4, d:1, e:1, f:4, g:2, h:1) and the initial MIS-tree shown in Fig. 2. We retain those items with supports no less than MIN=2 in our MIS-tree. Therefore, we remove the nodes with item-name= (d, e, h) and the result is shown in Fig.3.
After these nodes are removed, the remaining nodes in the MIS-tree may contain child nodes carrying the same item-name. For the sake of compactness, we traverse the MIS-tree and find that node (c:2) has two child nodes carrying the same item-name f. We merge these two nodes into a single node with item-name=f, and its count is set as the sum of counts of these two nodes (shown in Fig.4).
At last, the complete and compact MIS-tree is shown in Fig.5.
Now we are complete the first step is to build the MIS tree (Fig.4), let go to the second step is to extract f conditional frequent pattern from the MIS tree.

For the MIS-tree in Fig.5, let us consider how to build a conditional pattern base and conditional MIS-tree for item g. First, the node-link of item g is followed. Each such path in the MIS-tree ends at a node "g". However, we exclude the node "g" itself and add it to the conditional pattern base and the conditional MIS-tree for item g. Counter of each node in the path is set to that of the node "g" itself. In this example, following the node-link for g, we get two paths in the MIS-tree: (a:3, c:2, f:2, g:1) and (b:2, f:1, g:1). To build the conditional pattern base and conditional MIS-tree for g, we exclude the node g in these two paths, (a:1, c:1, f:1) and (b:1, f:1). Notice that counters of the nodes in these two paths are all set to 1, because the counter values of both nodes g in the paths (a:3, c:2, f:2, g:1) and (b:2, f:1, g:1) are 1. Whether an item is frequent in the g's conditional MIS-tree is checked by following the node-link of each item, summing up the counts along the link and seeing whether it exceeds the MIS value of item g. In the conditional MIS-tree for g, the support count of a is 1, that of b is 1, that of c is 1 and that of "f" is 2. Since the MIS value of item g is 2, only item f is frequent in the g's conditional MIS-tree here. Therefore, we find g's conditional frequent pattern. Therefore, we apply same process for the other items to achieve the result showing in the Table 2.

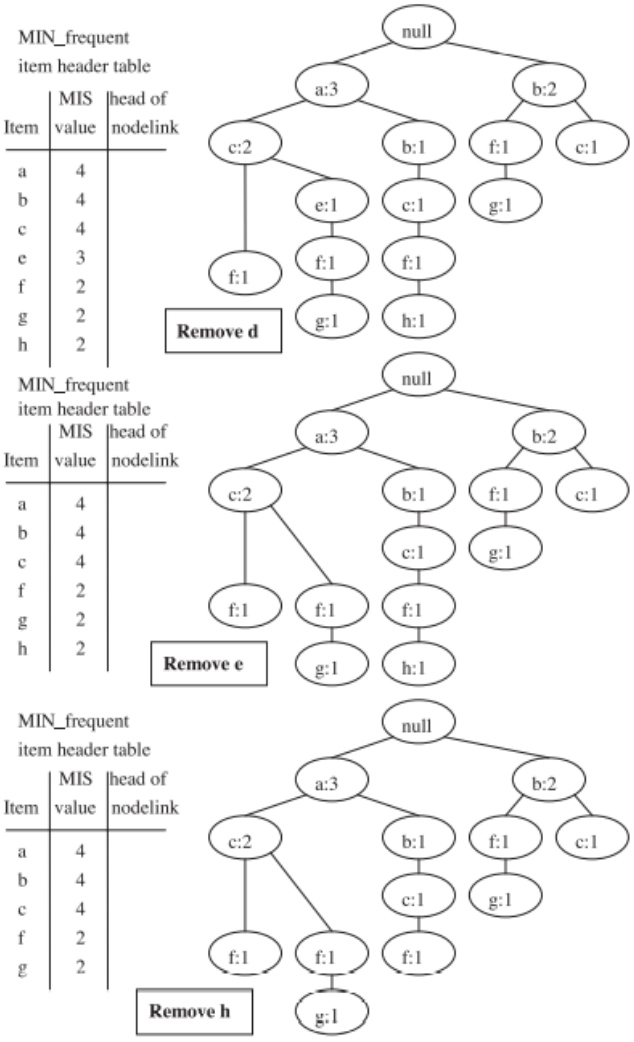

Fig. 3. Result after removing nodes

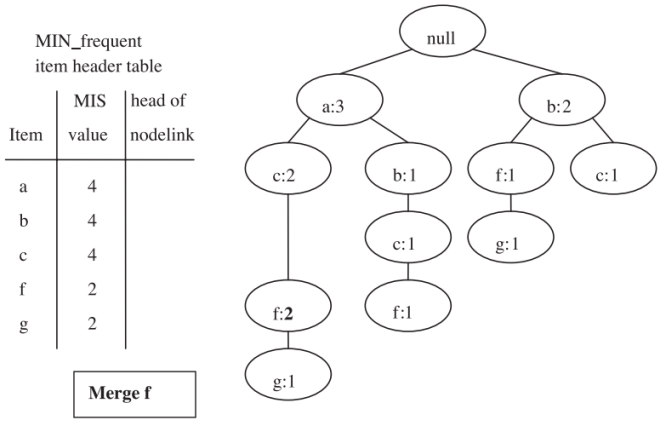

Fig.4. Result after merging

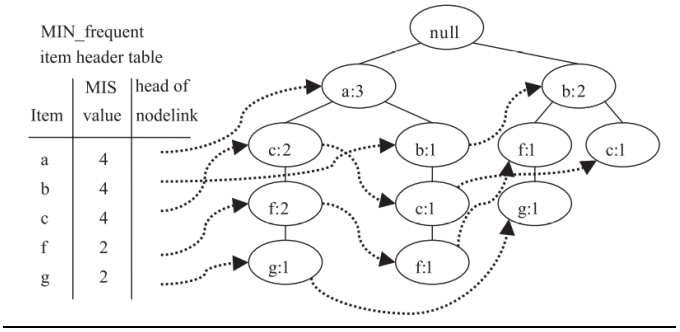

Fig.5. Final result

Table 7: Conditional frequent patterns.

Item	Conditional pattern base	Conditional frequent patterns
g	{(a:1, c:1, f:1), (b:1, f:1)}	fg:2
f	{(a:2, c:2), (a:1, b:1, c:1), (b:1)}	acf:3, bf:2
c	{(a:2), (a:1, b:1), (b:1)}	null
b	{(a:1)}	null
a	Null	null

4. ICFP-growth Algorithm

The ICFP-growth pre-assumes that for every item, user specifies the MIS values prior to its execution. Therefore, using the priori information i.e., MIS values of the items, the frequent patterns are generated with a single scan on the dataset. The proposed algorithm generalizes the CFP-growth algorithm for finding frequent patterns. This approach involves three steps. They are constructing the MIS-tree, extracting compact MIS-tree and mining the compact MIS-tree to mine frequent patterns. We now discuss each of these steps in detail.

4.1. Constructing MIS-tree:

The construction of MIS-tree in ICFP-growth algorithm is shown in Algorithm 1 and described as follows. The ICFP-growth algorithm accepts transaction dataset (Trans), Itemset (I) and minimum item support values (MIS) of the items as input parameters. Using the input parameters, the ICFP-growth

creates an initial MIS-tree, which is similar to MIS-tree created by CFP-growth (Lines 1 to 7 in Algorithm 1).

4.2. Extracting Compact MIS-tree:

Next, starting from the last item in the item-header table (i.e., item having lowest MIS value) perform tree-pruning operating by calling MisPruning procedure to remove the infrequent items from the item-header table and MIS-tree. After one item is pruned, move to immediate next item in item-header table and perform tree-pruning. However, when the frequent item is encountered, stop tree-pruning process. The MIS value of this frequent item represents the LMS value. Let the resultant item header table be MinFrequentItemHeaderTable. The items in this header table may contain both frequent and infrequent items having support greater than the lowest MIS value among all frequent items. The items in the header table are referred as "quasi-frequent items". Call MisMerge procedure to merge the tree. Finally, call InfrequentLeafNodePruning procedure to prune the infrequent leaf nodes in the MIS-tree. The resultant MIS-tree is the compact MIS-tree.

4.3. Mining Frequent Patterns from Compact MIS-tree:

Mining the frequent patterns from the compact MIS-tree is shown in Algorithm 6. The process of mining the compact MIS-tree in ICFP-growth is almost same as mining the compact MIS-tree in CFP-growth. However, the variant between the two approaches is that before generating conditional pattern base and conditional MIS-tree for every item in the header of the Tree, the ICFP-growth approach verifies whether the suffix item in the header of the Tree is a frequent item. If a suffix item is not a frequent item then the construction of conditional pattern base and conditional MIS-tree are skipped. The reason is as follows. In every frequent pattern, the item having lowest MIS value should be a frequent item. In constructing the conditional pattern base for a suffix item, the suffix item represents the item having lowest MIS value. Therefore, if the suffix item is an infrequent item, then all its prefix-paths will also be infrequent.

Algorithm 1. MIS-tree (Tran:transaction dataset, I: itemset containing n items, MIS: minimum item support values for n items).

1: Let L represent the set of items sorted in non-decreasing order of their MIS values.
2: create the root of a MIS-tree, T, and label it as "null".
3: for each transaction t ∈ Tran do
4: sort all the items in t in L order.
5: count the support values of any item I, denoted as S(i) in t.
6: Let the sorted items in t be [p|P], where p is the first element and P is the remaining list. Call InsertTree([p|P], T).
7: end for
8: for j=n-1; $j \geq 0$; –j do
9: if $S[I_j] < MIS[I_j]$ then
10: Delete the item I_j in header table.
11: call MisPruning(Tree, $L[I_j]$).
12: else
13: break; //come out of pruning step.
14: end if
15: end for
16: Name the resulting table as MinFrequentItemHeaderTable.
17: Call MisMerge(Tree).
18: Call InfrequentLeafNodePruning(Tree).

Procedure 2. InsertTree ([p|P], T).

1: while P is nonempty do
2: if T has a child N such that p.item-name=N.item-name then 3: N.count++.
4: else
5: create a new node N, and let its count be 1.

6: let its parent link be linked to T.
7: let its node-link be linked to the nodes with the same item-name via the node-link structure;
8: end if
9: end while

Procedure 3. MisPruning (Tree, I_j).

1: for each node in the node-link of I_j in Tree do
2: if the node is a leaf then
3: remove the node directly;
4: else
5: remove the node and then its parent node will be linked to its child node(s);
6: end if
7: end for

Procedure 4. MisMerge (Tree).

1: for each item I_j in the MinFrequentItemHeaderTable do
2: if there are child nodes with the same item-name then then
3: merge these nodes and set the count as the summation of these nodes' counts.
4: end if
5: end for

PROCEDURE 5. INFREQUENTLEAFNODEPRUNING(TREE).

1: choose the last but one item I_j in MinFrequentItemHeader-Table. That is, item having second lowest MIS value.
2: repeat
3: if I_j item is infrequent item then
4: using node-links parse the branches of the Tree.
5: repeat
6: if I_j node is the leaf of a branch then
7: drop the node-link connecting through the child branch.
8: create a new node-link from the node in the previous branch to node in the coming branch.
9: drop the leaf node in the branch.
10: end if
11: until all the branches in the tree are parsed
12: end if
13: choose item I_j which is next in the order.
14: until all items in MinFrequentItemHeaderTable are completed

Algorithm 6. ICFP-growth (Tree: MIS-tree, L: set of quasi-frequent items, MIS: minimum item support values for the items in L).

1: for each item I_j in the header of the Tree do
2: if I_j is a frequent item then
3: generate pattern $\beta = I_j \cup \alpha$ with support = I_j .support;
4: construct β's conditional pattern base and β's conditional MIS-tree Tree β.
5: if $Tree_\beta \neq \phi$ then
6: call CpGrowth($Tree_\beta$, β, MIS(I_j).
7: end if
8: end if
9: end for

Procedure 7. CpGrowth(Tree, α, MIS(α)).

1: for each I_j in the header of Tree do
2: generate pattern $\beta = I_j \cup \alpha$ with support = I_j .support.
3: construct β's conditional pattern base and then β's conditional MIS-tree Treeβ .
4: if $Tree_\beta \neq \phi$ then
5: call CpGrowth(Treeβ , β, MIS(α)).
6: end if
7: end for

4.4. Example

Based on the algorithm we will apply the ICFP-growth on a small transaction database that contain 10 transactions with 8 items (Table 8).

Table 8: Transaction database.

TID	Items
1	A, C
2	A, C, B
3	A, C, G
4	A, C, H
5	A, D, B
6	F, E
7	F, E
8	D, E
9	B, D
10	B, D

Table 9: Items MIS values.

Item	A	B	C	D	E	F	G	H
MIS value	5	5	5	5	2	2	2	2

We will consider the items MIS values in Table 9. The first step in the ICFP-growth algorithm is to constructing MIS-tree, it's the same process as in the CFP-growth to get the result shown in Figure 6.

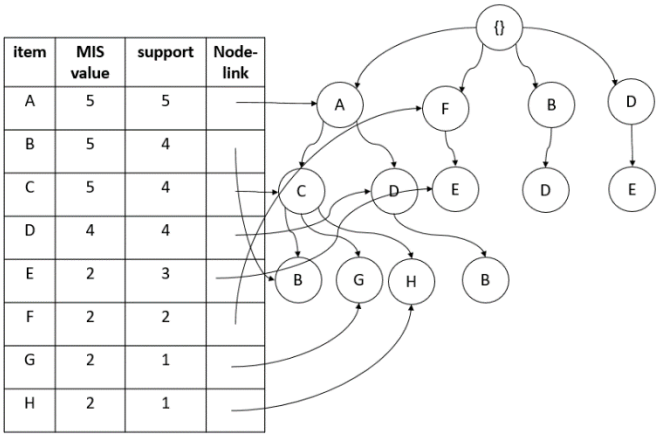

Fig.6. MIS-Tree

After constructing MIS-tree, new we will applay to the MIS tree the MIS pruning and also MIS merge as we have seen in the CFP-growth, new we have as a result the MIS tree compact, the result shown in the figure (Fig.7). After extracting the compact MIS-tree, we will apply the Infrequent Leaf Node Pruning to reduce the search space. First, we will scan the MinFrequentItemHeaderTable to extract the infrequent items. The infrequent item is the item that have the support less than the MIS value. We have (B, C) are infrequent items, we are going to start with the item B we have to search the for B nodes in the compact MIS tree that represent the leaf of the tree and delete them the result are shown in the Fig. 8.

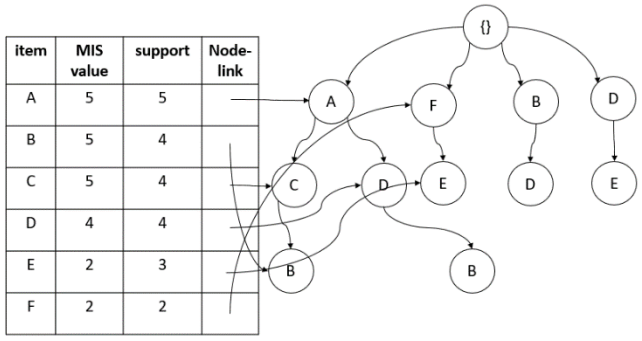

Fig.7

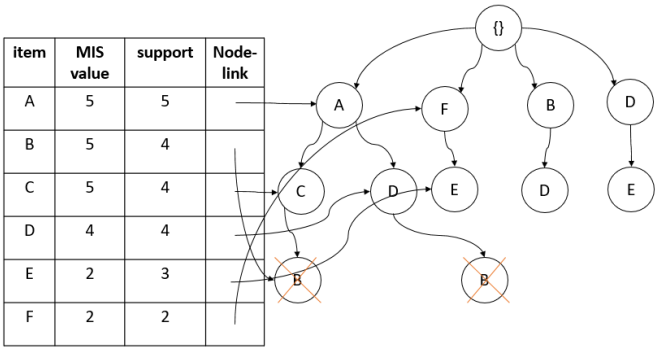

Fig.8

And we will the same process to the item C by search for the C nodes that represent the leaf node of the tree, the result shown in Fig.9.

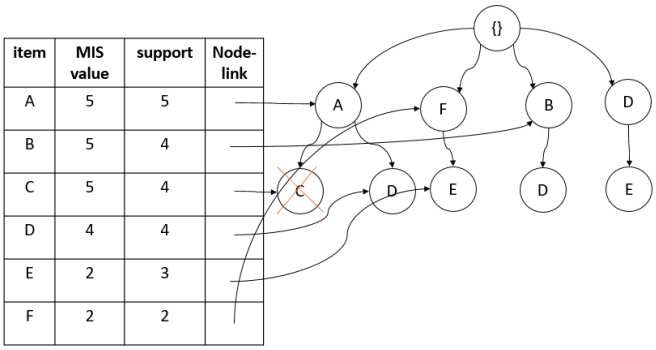

Fig.9

In the end we will have, the MIS tree reduced as we can see in Fig.10. Now we can extract the frequent pattern from the final tree, the result shown in the Table 10.

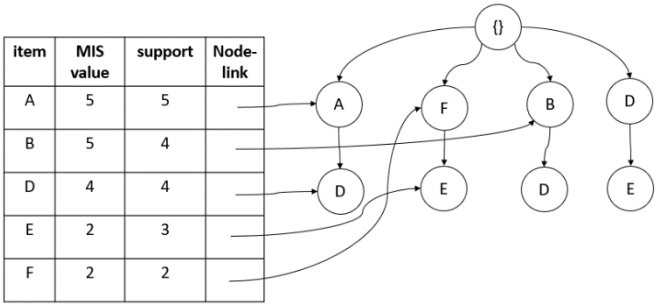

Fig.10: Final Tree

Table 10: Conditional frequent patterns.

item	Conditional pattern base	Conditional frequent patterns
F	null	null
E	{(F:2), (D:1)}	FE:2
D	{(A:1), (B:2)}	BD:2
B	null	null
A	null	null

5. Implementation

In this implementation, we are going to make three-classification system using the three algorithms FP-growth, CFP-Growth and ICFP-growth, to compare their performances.

In this study, we will use python 3 as programming language, vs code as IDE and windows 10 machine with 1.8 GHz and 8GB memory as environment.

We are going to apply the three algorithms on a classification diabetes female's dataset, with two '.csv' files the first for train dataset, and the other for the test dataset, the two '.csv' files contain 8 features:

- Pregnancies: Number of times pregnant.
- Glucose: Plasma glucose concentration 2 hours in an oral glucose tolerance test.
- BloodPressure: Diastolic blood pressure (mm Hg).
- SkinThickness: Triceps skin fold thickness (mm).
- Insulin: 2-Hour serum insulin (mu U/ml).
- BMI: Body mass index (weight in kg/(height in m)^2).
- DiabetesPedigreeFunction: Diabetes pedigree function.
- Age: Age (years).

The datasets are available in Kaggle platform (https://www.kaggle.com/mathchi/diabetes-data-set)

5.1. Database transformation

The dataset that we have is numerical it contains just numerical values; the FP-Growth, CFP-growth and ICFP-Growth accept the transactional datasets, database that contains transactions, a group of items. In this step, we try to transform the diabetes dataset (numerical datasets) into a transactional dataset.
To do this transformation we are going to visualize each feature to know how it change next to the number of individuals, to divide each feature into domains that regroups several individuals.
The first feature is the age, the result of visualization illustrated in the Fig.11.

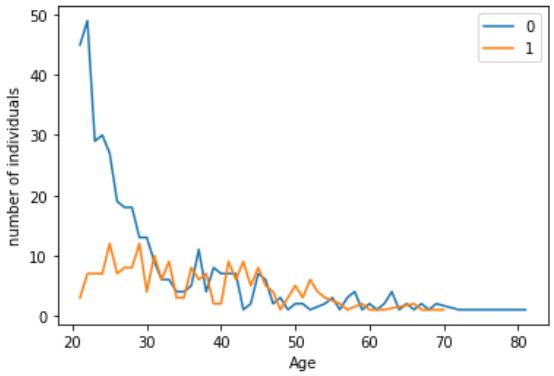

Fig.11: Age Vs number of individuals

We have in this Figure 11 a graph of the age feature compared to the number of individuals. As we can see in the range [20, 30] we have, hight number of no diabetes in comparison to the numbers of individuals with diabetes, and for the range [30, 80] we have allmost the same number of individuals for both classes 0 and 1 (0: without 1: diabetes , with diabetes), so we can just divide the range of the feature into two domains: A1 : [0, 30] and A2 : [30, 80]. The second feature is the Blood Pressure, the result shown in the Fig.12.

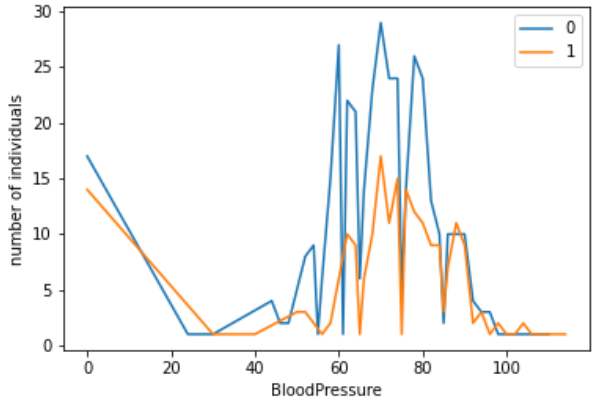

Fig.12: Blood Pressure Vs number of individuals

As we can see in the graph of the Blood pressure in hard to divide it into domains, but we are going to make approximate domains, in the range [0, 40] we have the same variation for the two classes 0 and 1, and in [40, 90] the class 0 is highest then the 1 class, and in the range [90, 120] also we have the same variation of the classes 0 and 1, so we will divide this feature to three domains: B1: [0, 40]; B2 : [40, 90]; B3: [90, 120]. The thread feature is BMI, the result of visualization exemplified in the Fig.13.

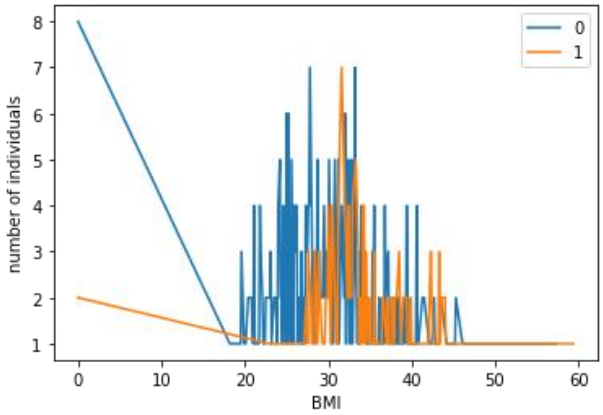

Fig.13: BMI vs number of individuals

In this graph, we have the MBI index compared to the number of individuals. As we see in this graph, we can divide the range of the BMI feature into two domains the first BMI1: [0, 30], where we have individuals' number of the class 0 highest than the 1 class, and the second is BMI2: [30, 60] where the two classes have almost the same variation. The fourth feature is the DiabetesPedigreeFunction, the visulisation is in the Fig.14.

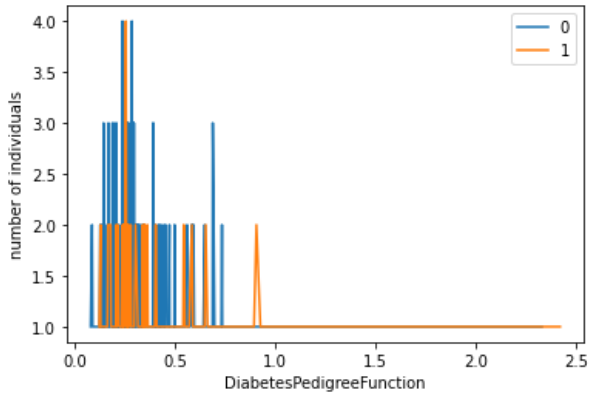

Fig.14: Diabetes of function of number of individuals

In this figure we can see in [0, 0.8] the 0 class have almost the highest number of individuals than the 1 class, and for the range [0.8, 2.5] the opposite, the class 1 have the highest number of individuals, therefore we can divide the feature into two domains: D1: [0, 0.8] and D2: [0.8, 2.5]. The fifth feature is the Glucose, the result of visualization illustrated in the Fig.15.

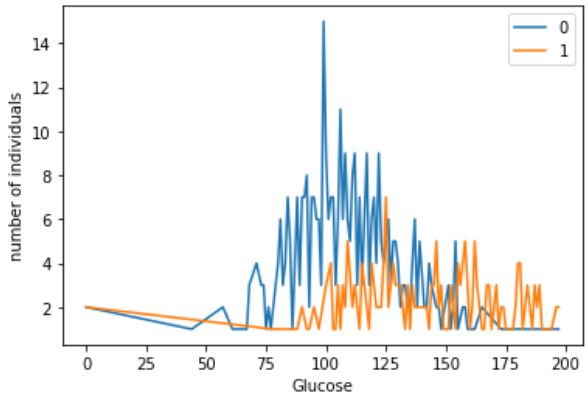

Fig.15: Glucose Vs number of individuals

We have in this figure a graph of the Glucose feature compared to the number of individuals. As we can see in the range [0, 125] we have, hight number of the class 0 in comparison to the numbers of individuals of the class 1, and for the range [125, 200] we have allmost the same number of individuals for both classes 0 and 1, so we can just divide the range of the feature into two domains: G1 : [0, 125] and G2 : [125, 200]. The sixth feature is the Insulin, the result shown in the Fig.16.

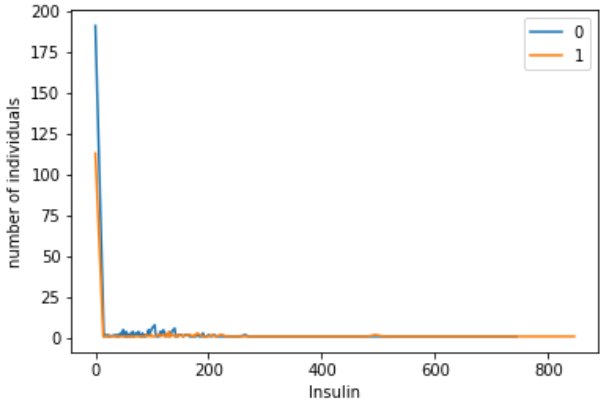

Fig.16: Insulin Vs number of individuals

As we can see in the graph of the Insulin in the range [0, 30] we have almost the same variation for the two classes 0 and 1, and in [30, 150] the class 0 is highest then the 1 class, and in the range [150,

800] also we have the same variation of the classes 0 and 1, so we will divide this feature to three domains: I1: [0, 30]; I2 : [30, 150]; I3: [150, 800].

The festure number seven is the Pregnancies, the result of visualization exemplified in the Fig.17.

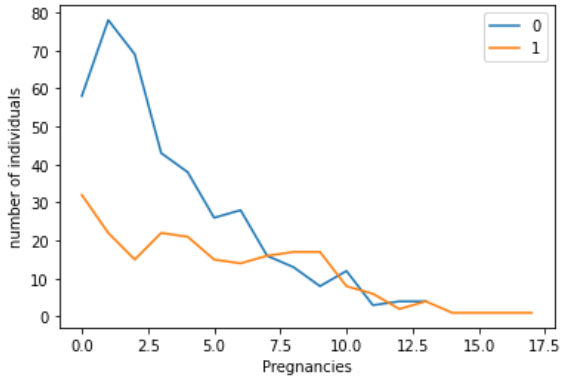

Fig.17: Pregnancies VS number of individuals

In this graph, we have the Pregnancies feature compared to the number of individuals. As we see in this graph, we can divide the range of the Pregnancies feature into two domains the first P1: [0, 7], where we have individuals' number of the class 0 highest than the 1 class, and the second is P2: [7, 17] where the two classes have almost the same variation. The last feature is the SkinThickness, the visulisation is in the Fig.18.

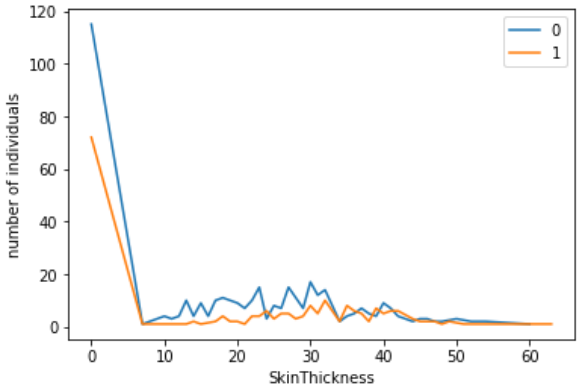

Fig.18: Skin thickness VS number of individuals

In this figure we can see in [0, 8], the proximately the same variation for the two classes, and in [8, 45] the 0 class have almost the highest number of individuals than the 1 class. In addition, for the range [45, 60] also we have the same variation for the both classes, therefore we can divide the feature into three domains: S1: [0, 8], S2: [8, 45] and S3: [45, 60].

After all these the graphs and analysis, we can briefly resume all the information in the Table below.

feature	domains
Age	A1: [0, 30]; A2: [30, 80]
Pregnancies	P1: [0, 7]; P2: [7, 17]
Glucose	G1: [0, 125]; G2: [125, 200]
Blood Pressure	B1: [0, 40]; B2: [40, 90]; B3: [90, 120]
Skin Thickness	S1: [0, 8]; S2: [8, 45]; S3: [45, 60]
Insulin	I1: [0, 30]; I2: [30, 150]; I3[150, 800]
Diabetes Pedigree Function	D1: [0, 0.8]; D2: [0, 2.5]
BMI	BMI1: [0, 30]; BMI2: [30, 60]

Now we can use the domains to transform the dataset to a transactional dataset. As we can see in the Fig.19, we have a part of result that we obtain from the transformation.

```
[['P1', 'G1', 'B2', 'S2', 'I2', 'BMI2', 'D1', 'A1', '0'],
 ['P2', 'G2', 'B3', 'S2', 'I2', 'BMI2', 'D1', 'A2', '1'],
 ['P1', 'G1', 'B2', 'S2', 'I3', 'BMI2', 'D1', 'A1', '1'],
 ['P1', 'G1', 'B2', 'S3', 'I3', 'BMI2', 'D2', 'A2', '0'],
 ['P1', 'G2', 'B2', 'S2', 'I3', 'BMI2', 'D1', 'A1', '1'],
 ['P1', 'G1', 'B2', 'S1', 'I1', 'BMI1', 'D1', 'A2', '0'],
 ['P1', 'G1', 'B1', 'S2', 'I1', 'BMI1', 'D1', 'A1', '0'],
 ['P1', 'G1', 'B2', 'S3', 'I2', 'BMI2', 'D1', 'A1', '0'],
 ['P1', 'G1', 'B2', 'S2', 'I2', 'BMI2', 'D1', 'A1', '0'],
 ['P1', 'G1', 'B2', 'S2', 'I2', 'BMI2', 'D1', 'A1', '0'],
 ['P1', 'G1', 'B2', 'S2', 'I2', 'BMI1', 'D2', 'A1', '0'],
```

Fig. 19

5.2. Extracting association rules*:*

First, we have to initialize the minsupport for FP-Growth, and MIS-values for CFP-Growth and ICFP-Growth. To assign MIS-values for CFP-Growth, we are going to use the equation (2).

$$MIS(i) = maxinum(\beta f(i), LS) \quad (2)$$

- MIS (i) is the MIS-value of the item "i".
- $\beta \in [0, 1]$ is a parameter that controls how the MIS values for items should be related to their frequencies.
- $f(i)$ is the frequencies value for the item "i".
- LS is a use specified value, represent the least minimum support allowed.

In addition, for the ICFP-Growth we will use the equation (3).

$$MIS(i) = \begin{cases} M(i) \ \ if\ M(i) > LMS \\ LMS\ else\ if\ M(i) < LMS\ and\ S(i) > LMS \\ LMIS\ else \end{cases} \quad (3)$$

with:

$$M(i) = S(i) \ - SD$$

- $SD \in [0, 1]$ is a user specified value.
- $S(i) = \frac{f(i)}{N}$ is the support of the item "i".
- $f(i)$ is the frequencies value for the item "i".
- N represent the number of transactions in the dataset.
- LMS is a user specified value, stand for lowest minimum support, and represent lowest MIS value of a frequent item
- LMIS is also a user specified value, stand for least minimum item support, and represent the lowest MIS value among all items in the transaction dataset.
- The LMIS value should always be less than or equal to LMS.

In this experiment, we will define the minsupport of the FP-Growth equal to 40, for the CFP-Growth the β equal to 0.1 and *LS* equal to 40, and for the ICFP-Growth, the *SD* value is 0.1, LMS equal to 50 and LMIS equal to 40.

The result of the MIS-values generation, concerns the CFP-Growth is in table below.

item	P1	G1	B2	S2	I2	BMI2	D1	A1	0
MIS-value	51	40	55	40	40	40	53	40	40

P2	G2	B3	A2	1	I3	S3	D2	S1	BMI1	B1
40	40	40	40	40	40	40	40	40	40	40

And for ICFP-Growth the result of MIS-values initialization in the table.

item	P1	G1	B2	S2	I2	BMI2	D1	A1	0
MIS-value	40	40	40	40	40	40	40	40	40

P2	G2	B3	A2	1	I3	S3	D2	S1	BMI1	B1
40	40	40	40	40	40	40	40	40	40	40

After we apply the three algorithms on our dataset, we are getting three models that contains the association rules as you can see in Fig.20.

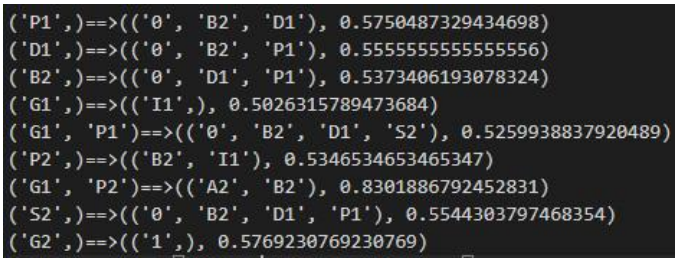

Fig. 20

We have the structure of our model is: (left) ➔(right, Confidence)
The left is causes, the right is the consequence, confidence is number in range of [0, 1] that can represent how much left can lead us to the right, who much the causes can lead as to a consequence, and we can use the equation (4) to calculate the confidence.

$$Confidence(right \rightarrow \text{left}) = \frac{support(right \cup left)}{support(right)} \quad (4)$$

In the figure we have the association rules between all the features, but in our case, we want to do a classification model, for that we will filter the association rules to have in the consequences (right) just the items that represent the classes (‘0’ and ‘1’), the result is shown in Fig.21.

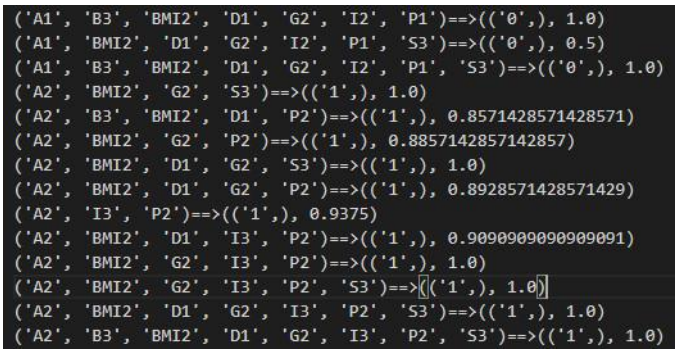

Fig.21

In the figure we have the association for the classification model for example we have this association rule (‘A2’, ‘BMI2’, ‘G2’, ‘P2’) ➔ ((‘1’), 0.89).
That mean if the individual has A2 the age between [30, 80]. The BMI2 the Body mass index in range of [30, 60], the G2 Plasma glucose concentration 2 hours in an oral glucose tolerance test in range of [125, 200], and the P2 number of times pregnant between [7, 17], so we can see that the induvial has a diabetes with the confidence of 0.89.

5.3. Evalutions

In this step, we will evaluate the three models of the algorithms FP-Growth, CFP-Growth and ICFP-Growth. To evaluate our models, we will use the test datasets. In addition, do the same prepressing that we were applied on the train dataset, to transform the numerical dataset into a transactional dataset. After that, we take a transaction from the dataset and calculate the distance between the test transaction

and the left of the association's rules of the models, in our case we have items how we can calculate the distance? In this case, we use an approach to calculate the distance. For example, we have T test transaction and G the left of an association rule that exist in the model.
T = ['P1', 'G1', 'B2', 'S2', 'I3', 'BMI2', 'D2', 'A2'],
G = ['A1', 'B3', 'BMI2', 'D1', 'G2', 'I2', 'P1']
First, we have 8 features in datasets. We initialized the distance by 8, and we check for each item of the T if it exist in the G and for each item exist we decrement the distance by one. In this example, we have P1 exist in the T and G, so the distance is 7. In addition, G1 not exist in G and so we still have distance is 7, and B2, S2, I3, D2, A2 also doesn't exist in G, and we have BMI2 exist so we decrement the distance by 1 so we have the distance equal to 6, the distance between G and T is 6. In addition, after calculating the distances, we chose the three closet association rules, we count who much votes for the '0' and for '1', and we choose the class that have the highest number of votes. After this testing process, we calculate the accuracy for each module and we get the result in Fig.22.

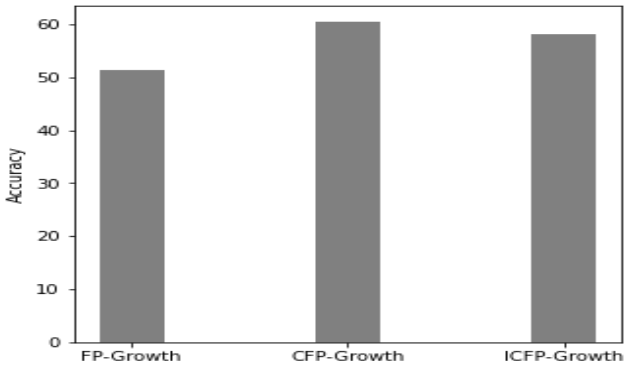

Fig.22. Accuracy graph

In the figure, we have the accuracy of the three classification models of the three algorithms FP-Growth, CFP-Growth and ICFP-Growth. For FP-Growth model we have 51.30%, for the CFP-Growth model we have 60.5%.

6. Conclusion

In this paper, we have explained the three association rules algorithms, FP-Growth, CFP-Growth, and ICFP-Growth, and we made a comparative study on three diabetes classification models that we implemented using the three algorithms FP-Growth, CFP-Growth and ICFP-Growth.

REFERENCES

[1] Jiawei Han, Jian Pei, and Yiwen Yin, 'Mining frequent patterns without candidate generation', in Proceedings of the 2000 ACM SIGMOD international conference on Management of data - SIGMOD '00, Dallas, Texas, United States, 2000, pp. 1–12.

[2] JV Joshua et al., 'Data Mining: A book recommender system using frequent pattern algorithm', Journal of Software Engineering and Simulation, vol. 3, no. 3, pp. 01–13, 2016.

[3] Ya-Han Hu and Yen-Liang Chen, 'Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism', Decision Support Systems, vol. 42, no. 1, pp. 1–24, Oct. 2006.

[4] R. Uday Kiran and P. Krishna Reddy, 'Novel techniques to reduce search space in multiple minimum supports-based frequent pattern mining algorithms', in Proceedings of the 14th International Conference on Extending Database Technology - EDBT/ICDT '11, Uppsala, Sweden, 2011, p. 11.

[5] Shikha Pathania and Harpreet Singh, 'A New Associative Classifier Based on CFP-Growth++ Algorithm', in Proceedings of the Sixth International Conference on Computer and Communication Technology 2015, New York, NY, USA, 2015, pp. 20–25.

[6] R. Uday Kiran and P. Krishna Reddy, 'An Improved Frequent Pattern-Growth Approach to Discover Rare association rules':, in Proceedings of the International Conference on Knowledge Discovery and Information Retrieval, Funchal - Madeira, Portugal, 2009, pp. 43–52.