1. INTRODUCTION

Data mining is a computing process that extracts hidden predictive information and patterns from large amount of raw data. Also referred to as "knowledge discovery in databases," the term "data mining" wasn’t coined until the 1990s.Its foundation comprises three intertwined scientific disciplines: statistics (the numeric study of data relationships), artificial intelligence (human-like intelligence displayed by software and/or machines) and machine learning (algorithms that can learn from data to make predictions). It uses a variety of data analysis tools to discover patterns and Relationships in data that may be used to make valid predictions. The newest answer to increase revenues and to reduce costs is data mining. The potential returns are enormous. Innovative organizations worldwide are already using data mining to locate and appeal to higher-value customers, to reconfigure their product offerings to increase sales, and to minimize losses due to error.

2. PROBLEM DEFINITION

Understanding the project objectives and requirements from a domain perspective and then converting this knowledge into a data science problem definition with a preliminary plan designed to achieve the objectives. Data science projects are often structured around the specific needs of an industry sector   or even tailored and built for a single organization. A successful data science project starts from a well defined question or need.

3. Measures of Association Rules

Essentially, association mining is about discovering a set of rules that is shared among a large percentage of the data (Zaki, 2000). Association rules mining tend to produce a large number of rules. The goal is to find the rules that are useful to users. There are two ways of measuring usefulness, being objectively and subjectively.
Objective measures involve statistical analysis of the data, such as support and confidence (Agrawal et al., 1993).

3.1 Support

The rule X ⇒ Y holds with support s if s% of transactions in D contain X ∪ Y. Rules that have a s greater than a user-specified support is said to have minimum support.

TID	ITEMS	Support = Occurence / Total Support
1	ABC	Total Support = 5 Support {AB} = 2 / 5 = 40% Support {BC} = 3 / 5 = 60% Support {ABC} = 1 / 5 = 20%
2	ABD	
3	BC	
4	AC	
5	BCD	

3.2 Confidence

The rule X ⇒ Y holds with confidence c if c% of the transactions in D that contain X also contain Y. Rules that have a c greater than a user-specified confidence is said to have minimum confidence.

TID	**ITEMS**	**Given X ⇒ Y** **Confidence = Occurrence {Y} / Occurrence {X}**
1	ABC	Confidence {A ⇒ B} = 2 / 3 = 66% Confidence {B ⇒ C} = 3 / 4 = 75% Confidence {AB ⇒ C} = 1 / 2 = 50%
2	ABD	
3	BC	
4	AC	
5	BCD	

4. LITERATURE SURVEY

4.1. CP-Tree:

Frequent pattern mining is a heavily researched area in the field of data mining with wide range of applications. Finding a frequent pattern (or items) plays as essentials role in data mining . Efficient algorithm to discover frequent patterns is essential in data mining research. A number of research works have been published that presenting new algorithm or improvements on existing algorithm to solve data mining problem efficiently. In that Apriori algorithm is the first algorithm proposed in this field. By the time of change or improvement in Apriori algorithm, the algorithms that compressed large database in to small tree data structure like FP tree, CAN tree and CP tree have been discovered . These algorithms are partitioned based , divide and conquer method used that decompose mining task in to smaller set of task for mining confined patterns in conditional database, which dramatically reduce search space. In this paper I propose a new novel tree structure - extension of CP tree that extract all frequent pattern from transactional database. This tree structure constructs compact prefix free structure with one database scan and it provide same mining performance as FP growth technique by efficient tree restructuring process. It also supports interactive and incremental mining without rescan the original database.

ADV:-Extracting hidden patterns from existing data is the process of data mining. The data is in the structured form and defined to make it compatible for processing .

DISADV:- According to NIST data representation limits the capacity of using traditional algorithms to conduct effective processing and analysis.

Reference from : Bezdek, J. C., & Pal, S. K. (1992). *Fuzzy models for pattern recognition: Methods that search for structures in data*. New York: IEEE Press of cp tree.

4.2. <u>FUFP Tree:-</u>

The frequent pattern tree (FP-tree) is an efficient data structure for association-rule mining without generation of candidate itemsets. It was used to compress a database into a tree structure which stored only large items. It, however, needed to process all transactions in a batch way. In real-world applications, new transactions are usually incrementally inserted into databases. In the past, we proposed a Fast Updated FP-tree (FUFP-tree) structure to efficiently handle new transactions and to make the tree update process become easier. In this paper, we attempt to modify the FUFP-tree construction based on the concept of pre-large itemsets. Pre-large itemsets are defined by a lower support threshold and an upper support threshold. It does not need to rescan the original database until a number of new transactions have been inserted. The proposed approach can thus achieve a good execution time for tree construction especially when each time a small number of transactions are inserted. Experimental results also show that the proposed Pre-FUFP maintenance algorithm has a good performance for incrementally handling new transactions.

Reference from: Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., & Uthurusamy, R. (Eds.). (1996). *Advances in knowledge discovery and data mining.* AAAI/MIT Press of dynamic fp tree.

4.3. <u>RP Tree:</u>

Most association rule mining techniques concentrate on finding frequent rules. However, rare association rules are in some cases more interesting than frequent association rules since rare rules represent unexpected or unknown associations. All current algorithms for rare association rule mining use an Apriori level-wise approach which has computationally expensive candidate generation and pruning steps. We propose RP-Tree, a method for mining a subset of rare association rules using a tree structure, and an information gain component that helps to identify the more interesting association rules. Empirical evaluation using a range of real world datasets shows that RP-Tree itemset and rule generation is more time efficient than modified versions of FP-Growth and ARIMA, and discovers 92-100% of all the interesting rare association rules.

ADV:-Models use flows to define the tree topology and have different objective functions.Computational results use instances from phylogenetic and telecommunications area.

DISADV:-Computational results use sets of instances having up to 15 terminal nodes.One of the models outperforms the previous models proposed in the literature.

Reference from: Han, J., & Kamber, M. (2000). *Data mining: Concepts and techniques*: Morgan Kaufmann.

4.4. **FP Growth:**

A new tree structure, called a FP tree, which is an extended prefix-tree structure for sorting compressed and crucial information in 2000. Consequently, the FP-growth method is a FP-tree based mining algorithm for mining frequent patterns. The Fp-growth approach is based on divide and conquer strategy for producing the frequent itemsets. FP-growth is mainly used for mining frequent itemsets without candidate generation to remove the drawbacks of the Apriori algorithm.

Major steps in FP-growth are:

Step 1.It firstly compresses the database showing frequent itemset into FP-tree. FP-tree is built using 2 passes over the dataset.

Step 2.It divides the FP-tree into a set of conditional database and mines each database separately, thus extract frequent item sets from FP-tree directly.

ADV:-FP-Growth algorithm runs much faster than the Apriori, it is logical to parallelize the FP-Growth algorithm to enjoy even faster speedup.

DISADV:- Recent work in parallelizing FP-Growth suffers from high communication cost, and hence constrains the percentage of computation that can be parallelized.

Reference from: Michie, D., Spiegelhalter, D. J., & Taylor, C. C. (1994). *Machine learning, neural and statistical classification*: Ellis Horwood.

4.5. MSFP Algorithm:

Proposed mining association rules with non-uniform minimum support values in 1999. This algorithm is an extension of Apriori algorithm which allowed users to choose different minsup to different items according to its natural frequency. They also defined the MIS as the lowest minimum supports among the items in the itemset. This is not always useful because it would consider some items that are not worth to be considered because one of the items in this itemset, its minsup was set too low. In some cases it makes sense that the minsup must be larger than the maximum of theminimum supports of the items contained in an item set.

ADV:-Among the various data mining applications, mining association rules is an important one. The strategies for mining frequent item sets, which is the essential part of discovering association rules, have been widely studied over the last decade such as the Apriori, DHP, and FP growth.

DISADV:-In the traditional frequent item sets mining algorithms, a strict definition of support is used to require every item in a frequent item set occurring in each supporting transaction.

Reference from: Jain, A. K., & Dubes, R. C. (1988). *Algorithms for clustering data*. New Jersey: Prentice Hall.

5. <u>Analysis of different type of Algorithm</u>:

5.1. <u>FP Growth Algorithm</u>:

➨ The FP-Growth Algorithm, proposed by Han in, is an efficient and scalable method for mining the complete set of frequent patterns by pattern fragment growth, using an extended prefix-tree structure for storing compressed and crucial information about frequent patterns named frequent-pattern tree (FP-tree).

```
Step1:  if Tree contains a single prefix path then { // Mining single prefix-path FP-tree}
Step2: let P be the single prefix-path part of Tree;
Step3: let Q be the multipath part with the top branching node replaced by a null root;
Step4: for each combination (denoted as ß) of the nodes in the path P do
Step5: generate pattern ß ∪ a with support = minimum support of nodes in ß;
Step6: let freq pattern set(P) be the set of patterns so generated;
Step7: else let Q be Tree;
Step8: for each item ai in Q do { // Mining multipath FP-tree}
Step9: generate pattern ß = ai ∪ a with support = ai .support;
Step10: construct ß's conditional pattern-base and then ß's conditional FP-tree Tree ß;
Step11: if Tree ß ≠ Ø then
Step12: call FP-growth(Tree ß , ß);
Step13: let freq pattern set(Q) be the set of patterns so generated;
Step14: return(freq pattern set(P) ∪ freq pattern set(Q) ∪ (freq pattern set(P) × freq pattern set.
```

<u>Example:</u>

TID	*Items bought*	*(ordered) frequent items*
100	***{f, a, c, d, g, i, m, p}***	***{f, c, a, m, p}***
200	***{a, b, c, f, l, m, o}***	***{f, c, a, b, m}***
300	***{b, f, h, j, o, w}***	***{f, b}***
400	***{b, c, k, s, p}***	***{c, b, p}***
500	***{a, f, c, e, l, p, m, n}***	***{f, c, a, m, p}***

min_support = 3

1. Scan DB once, find frequent 1-itemset (single item pattern)
2. Sort frequent items in frequency descending order, F-list
3. Scan DB again, construct FP-tree

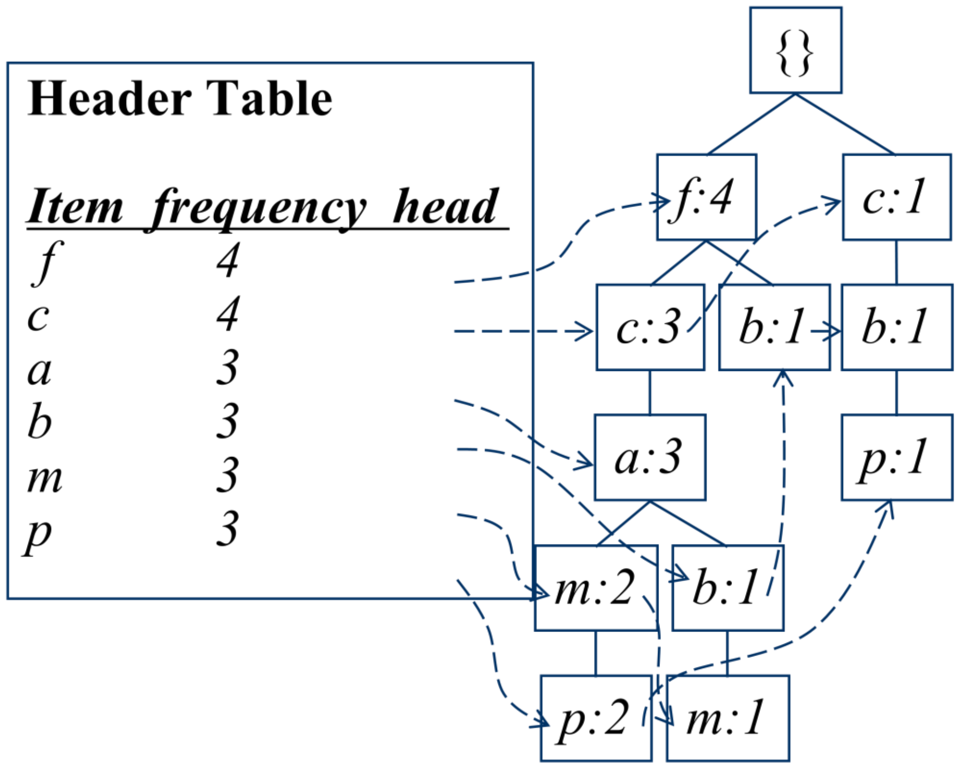


Fig1: fp growth

5.2. **Parallel FP Growth Algorithm:**

- Proposed by Min Chen et.al , 2009. [“**An efficient parallel FP-Growth algorithm”,** ***M Chen, X Gao, H.F. Li***, *IEEE Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2009*]
- It exploits the power of cluster computing.
- Multiple Computers/Processors works on the transaction database concurrently to construct the FP tree.
- This algorithm executes faster that normal FP Growth algorithm.
- Amount of speed-up achieved largely depends on number of processors/computers working concurrently.

Input: A transaction database Db and a minimum support threshold.

Output: It's frequent pattern tree. FP Tree.

Method: The FP-Tree is constructed in the following steps.

```
Suppose  there are n processor p1,p2…..pn;

Divide database Db into n part DB1….DBn with the same size.

Processor p,scan DBj once.And create total set of frequent items F and their supports.Short F in support descending order as L.

for(j=l;j≤ n; j=j=2)

{

Processor p1,parallel scan DBj, construct FP-tree, do not create head table and node link;

}

While(n>1) do

for (j=1;j≤n;j=j+2)

{

Proccesssor p1,parallel call FP-merge(FP-treej,FP-tree,+1,int (j/2)+1);

n=n/2;

}

Create head table  and same nodes link for FP-tree1;

Return FP-tree1;

End;
```

5.3. <u>Compact Pattern Tree:</u>

- It was first proposed by Syed Tanbeer et.al. [**CP-tree: a tree structure for single-pass frequent pattern mining**, *Syed Tanbeer et. Al. , Proceedings of the 12th Pacific-Asia conference on Advances in knowledge discovery and data mining*, 2008]
- Unlike FP Tree it requires only 1 database scan.
- CP-tree provides better performance than existing algorithms for interactive and incremental mining

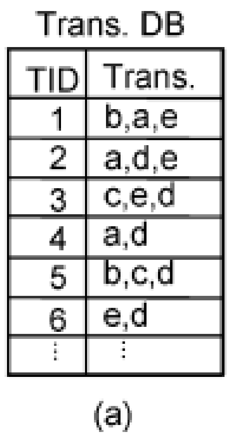
Input: T and l

Output: $T_{sort}$and l_{sort}

```
Step1:  Computer $l_{sort}$ from in frequency-descending order using merge sort technique
Step2: For each  branch $B_i$  in T
Step3: For each unprocessed path $p_i$ $B_i$
Step4: If  $P_i$  is  a sorted path
Step5: Process_Branch($P_{i)}$
Step6: Else sort_path($P_i$)
Step7: Terminate when all the branches are sorted and output $T_{sort}$  and $l_{sort}$.
Step8: Process_Branch (P){
Step9: For each branching node $n_b$ in p form $leaf_p$ node
Step10: For each sub-path form  $n_b$ to $leaf_x$ with $k \neq p$
Step11: If items ranks of all nodes between $n_b$ and $leaf_x$ are greater than that of $n_b$
Step12: P=sub-path from $n_b$ to $leaf_x$
Step13: if P is a sorted path
Step14: Process_Branch(P)
Step15: Elase P=path form the root to $leaf_x$
Step16: Sort_Path(P)
Step17: }
```

```
Step18: Sort_path(Q){
Step19: reduce the count of all nodes of Q by the value of $leaf_Q$ count
Step20: using merge sort, sort Q in an array according to $l_{sort}$ order
Step21: delete all nodes having count zero from Q
Step22: Insert sorted Q into T at the location from where it was taken
Step23:}
```

<u>Example:</u>



Trans. DB

TID	Trans.
1	b,a,e
2	a,d,e
3	c,e,d
4	a,d
5	b,c,d
6	e,d
⋮	⋮

(a)

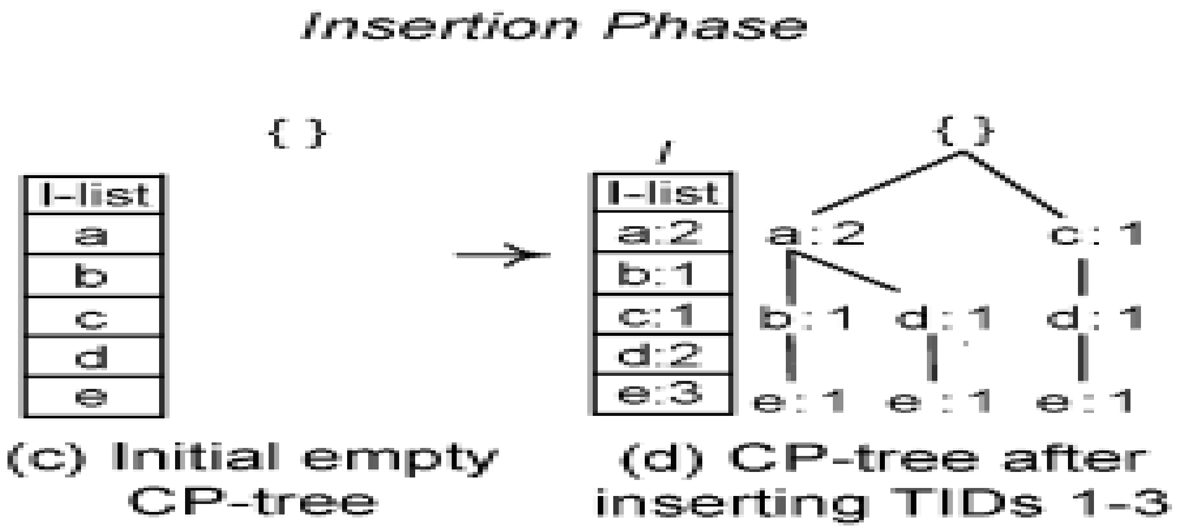


Fig 2 cp tree pic

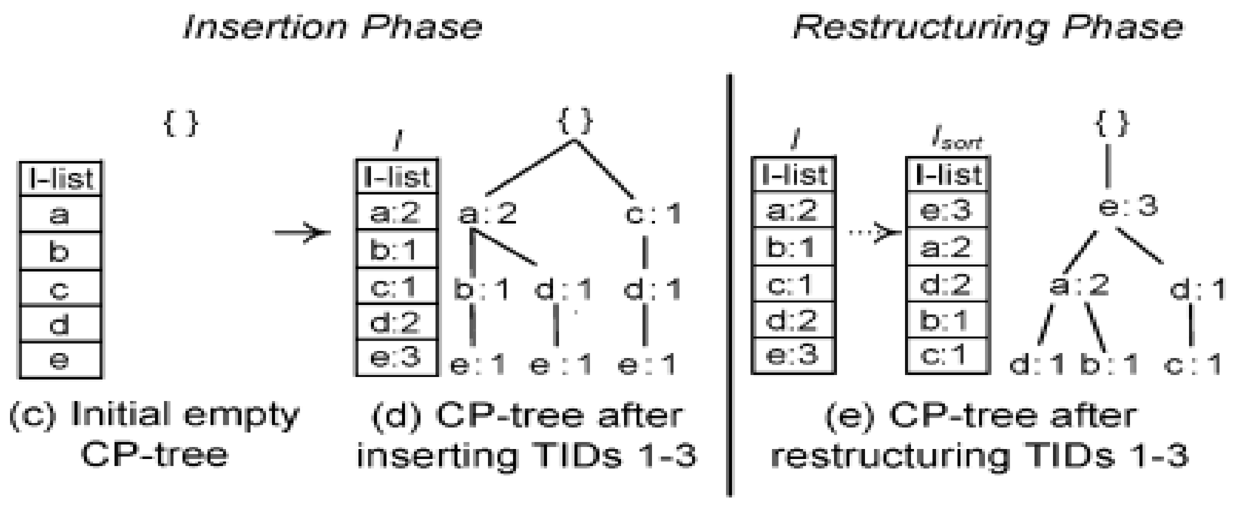

Insertion Phase
Restructuring Phase
(c) Initial empty CP-tree
(d) CP-tree after inserting TIDs 1-3
(e) CP-tree after restructuring TIDs 1-3

Fig 3: cp tree

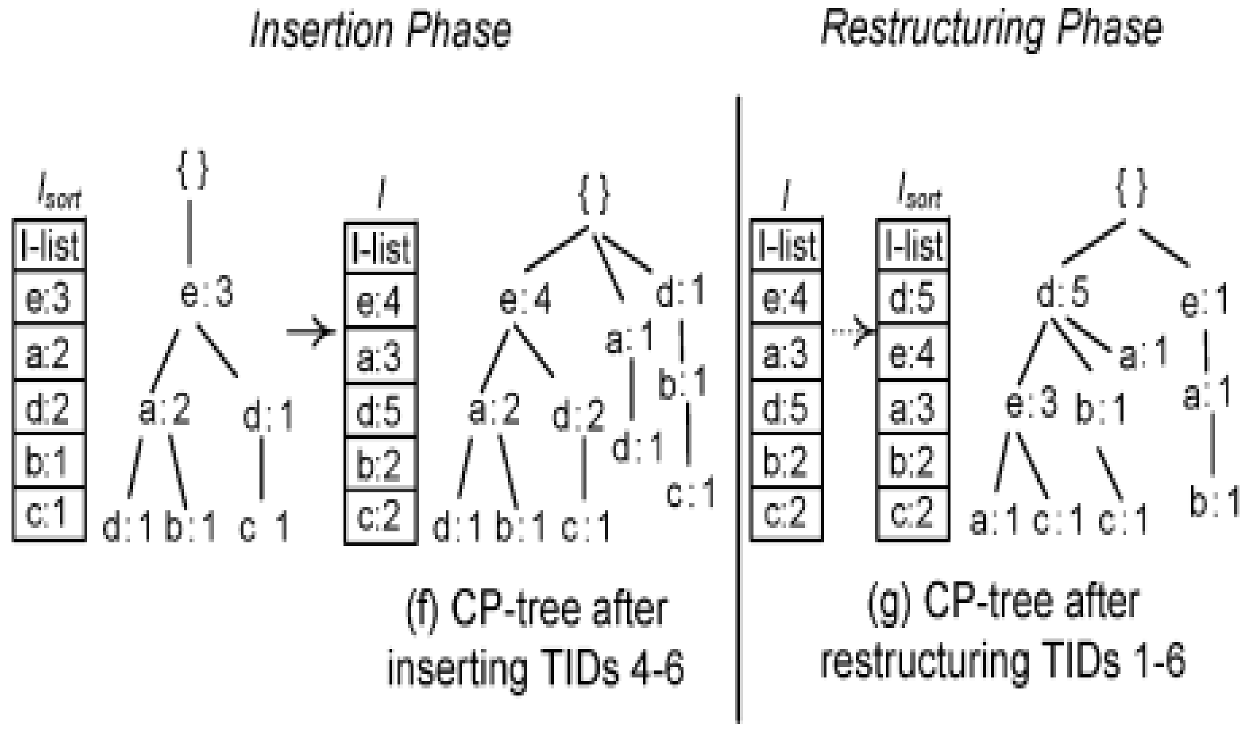

Insertion Phase
Restructuring Phase
(f) CP-tree after inserting TIDs 4-6
(g) CP-tree after restructuring TIDs 1-6

Fig 4 cp tree

5.4. Ascending Frequency Order Prefix tree(AFOPT):

- This algorithm was first proposed by Liu et.al. [**Ascending frequency ordered prefix-tree: Efficient mining of frequent patterns,** *Liu et.al, Database Systems for Advanced Applications, 2003.]*
- This algorithm uses a simple while compact data structure—ascending frequency ordered prefix tree(AFOPT) to organize the conditional databases.
- It uses arrays to store single branches to further save space.
- It then traverses the prefix-tree structure using a top-down strategy to mine pattern rules.
- This ascending frequency item ordering method achieves significant performance improvement over other pattern mining algorithms.

Input:

root is the root of the prefix tree;

min_sup is the minimum support threshold;

Description:

Step 1: if there is only one branch in tree root then

Step 2: Output patterns and return;

Step 3: end if

Step 4: for all children c of root do

Step 5: traversal subtree rooted at c and find the set of frequent items F'. sort the items in F' in ascending order of their frequencies;

Step 6: if $|F'|> 1$ then

Step 7: traversal subtree rooted at c and build anew prefix

Tree newroot which contains only the items in F';

Step 8: AFOPT(newroot,min_sup);

Step 9:end if;

Step 10: for all children subroot of c do

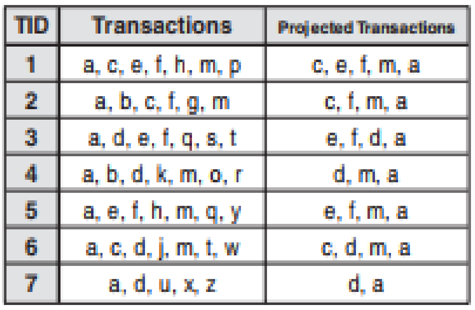
Step 11: sibroot= the right sibling of c whose item equal to subroot item;

Step 12: Merge(sibroot, subroot);

Step 13: end for

Step 14: end for

Example:



TID	Transactions	Projected Transactions
1	a, c, e, f, h, m, p	c, e, f, m, a
2	a, b, c, f, g, m	c, f, m, a
3	a, d, e, f, q, s, t	e, f, d, a
4	a, b, d, k, m, o, r	d, m, a
5	a, e, f, h, m, q, y	e, f, m, a
6	a, c, d, j, m, t, w	c, d, m, a
7	a, d, u, x, z	d, a

min_sup = 40%, F = {c:3, e:3, f:4, d:4, m:5, a: 7}

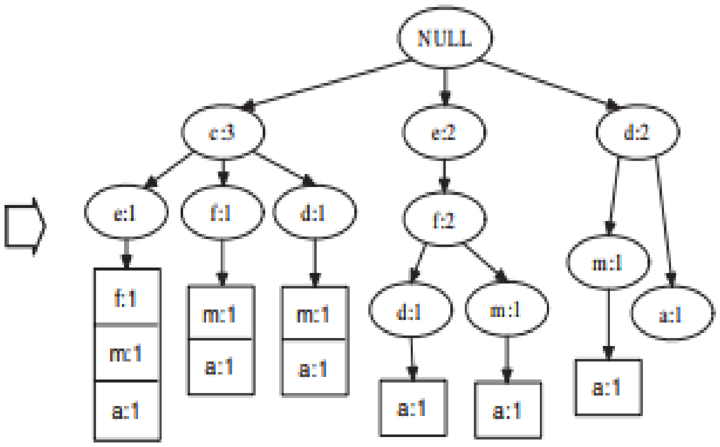


Fig 5 AFOPT tree

6. SOFTWARE REQUIREMENT SPECIFICATION

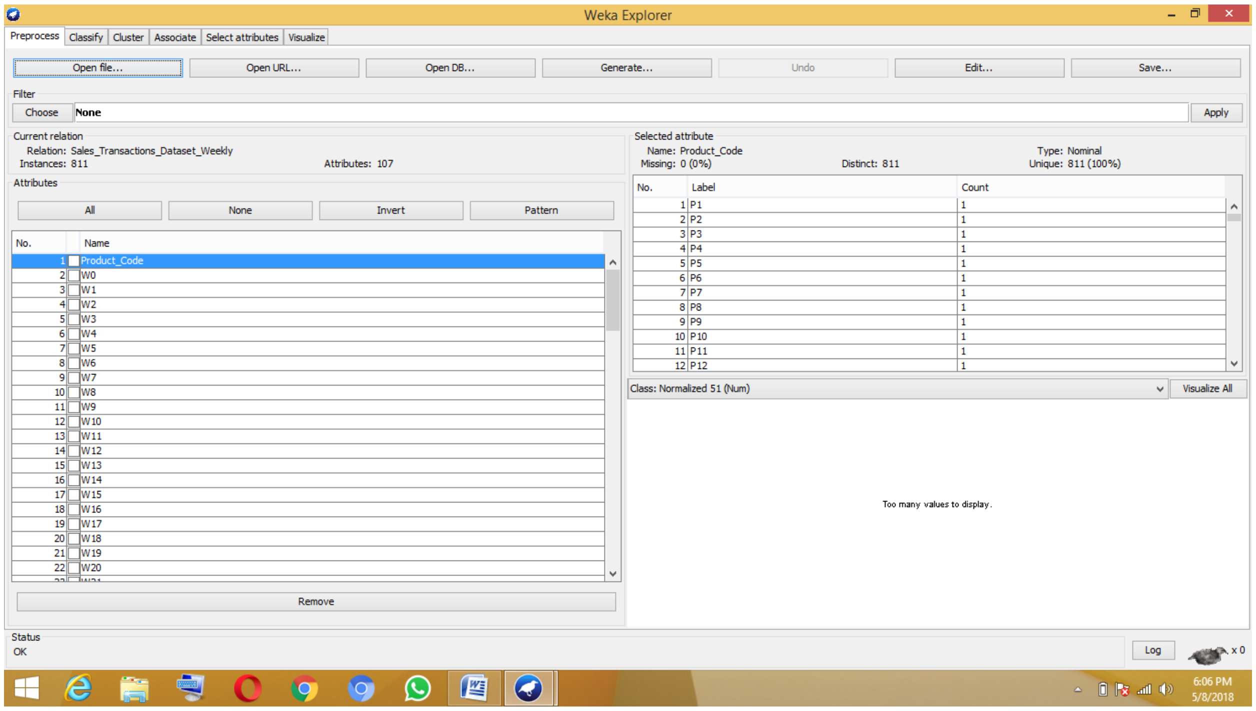
The main software for studying DATA MINNING is WEKA.

Weka (Waikato Environment for Knowledge Analysis) is a popular suite of machine learning software written in Java, developed at the University of Waikato, New Zealand. Weka is free software available under the GNU General Public License. The Weka workbench contains a collection of visualization tools and algorithms for data analysis and predictive modeling, together with graphical user interfaces for easy access to this functionality . Weka is a collection of machine learning algorithms for solving real-world data mining problems. It is written in Java and runs on almost any platform. The algorithms can either be applied directly to a dataset or called from your own Java code. The original non-Java version of Weka was a TCL/TK frontend to (mostly third-party) modeling algorithms implemented in other programming languages, plus data preprocessing utilities in C, and a Makefile-based system for running machine learning experiments. This original version was primarily designed as a tool for analyzing data from agricultural domains, but the more recent fully Java-based version (Weka 3), for which development started in 1997, is now used in many different application areas, in particular for educational purposes and research.
Advantages of Weka include:

- **I.** Free availability under the GNU General Public License
- **II.** Portability, since it is fully implemented in the Java programming language and thus runs on almost any modern computing platform
- **III.** A comprehensive collection of data preprocessing and modeling techniques
- **IV.** Ease of use due to its graphical user interfaces Weka supports several standard data mining tasks, more specifically, data preprocessing, clustering, classification, regression, visualization, and feature selection

7. Coding And Screenshot

I. Open the .arff file which is present in the system, by choosing it from the OPEN FILE option. All the attribute and the instance that are present in the chosen .arff file while be shown as follow.



Filter: None

Current relation
Relation: Sales_Transactions_Dataset_Weekly
Instances: 811
Attributes: 107

Attributes

No.	Name
1	Product_Code
2	W0
3	W1
4	W2
5	W3
6	W4
7	W5
8	W6
9	W7
10	W8
11	W9
12	W10
13	W11
14	W12
15	W13
16	W14
17	W15
18	W16
19	W17
20	W18
21	W19
22	W20

Selected attribute
Name: Product_Code
Type: Nominal
Missing: 0 (0%)
Distinct: 811
Unique: 811 (100%)

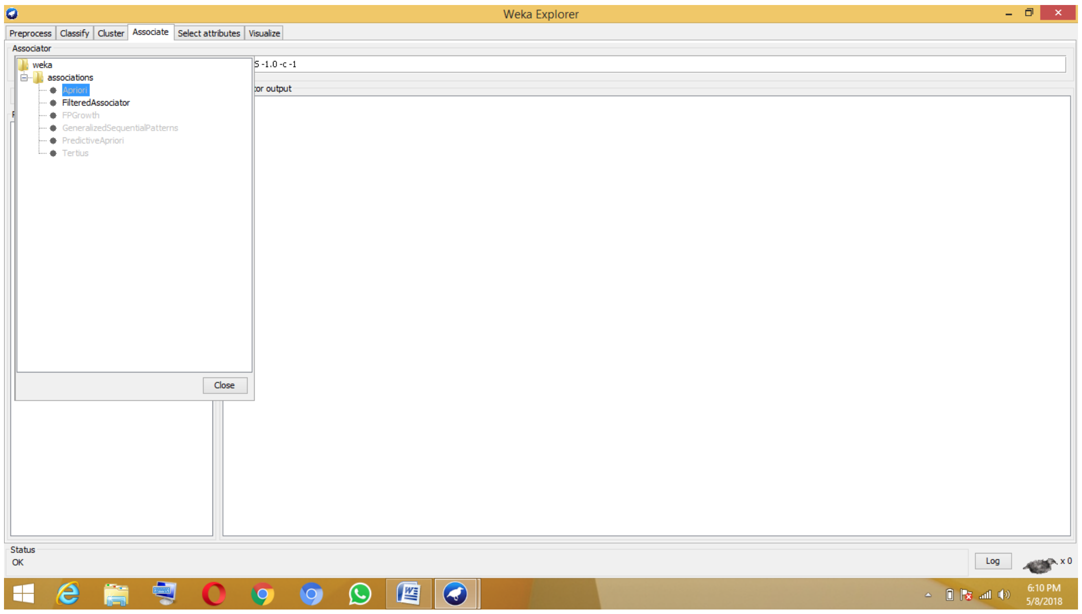
No.	Label	Count
1	P1	1
2	P2	1
3	P3	1
4	P4	1
5	P5	1
6	P6	1
7	P7	1
8	P8	1
9	P9	1
10	P10	1
11	P11	1
12	P12	1

Class: Normalized 51 (Num)

Too many values to display.

Status
OK

II. In the association tab, we have to associate the FP Growth to generate the rule, but it is disable , as in the figure.


Weka Explorer

Preprocess | Classify | Cluster | Associate | Select attributes | Visualize

Associator

S -1.0 -c -1

- weka
  - associations
    - Apriori
    - FilteredAssociator
    - FPGrowth
    - GeneralizedSequentialPatterns
    - PredictiveApriori
    - Tertius

Close

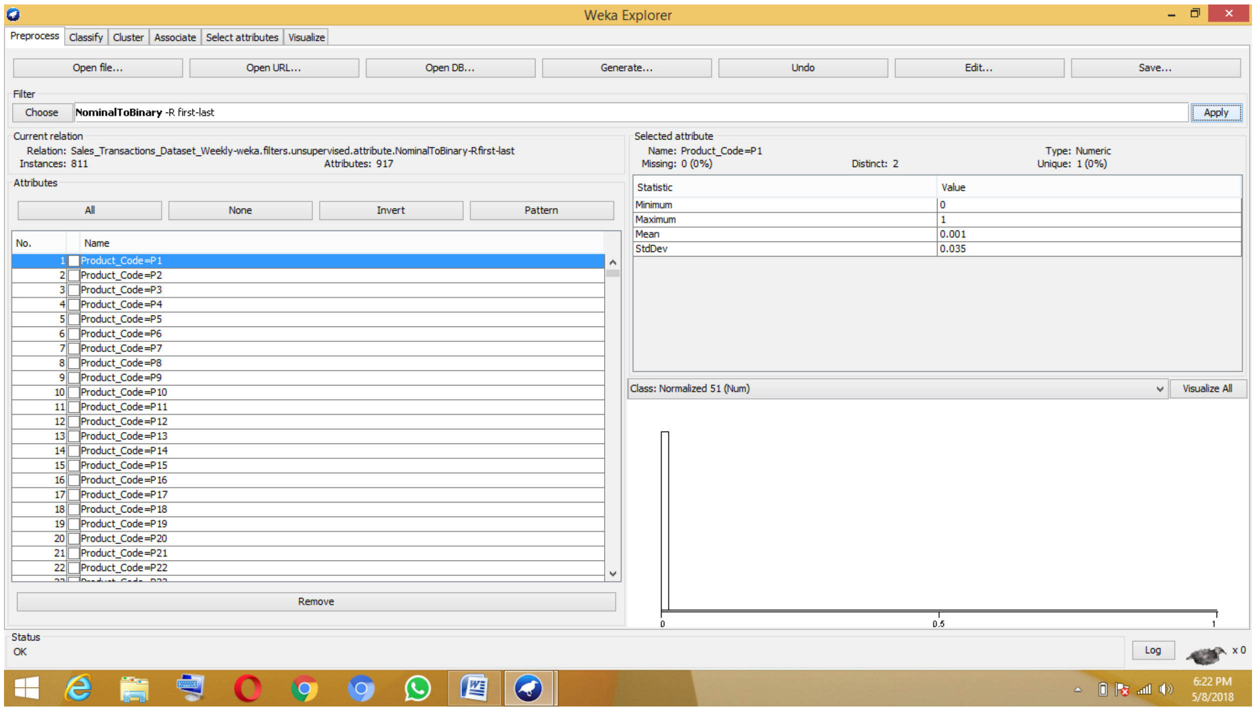
tor output

Status
OK

Log x 0

6:10 PM
5/8/2018

III. To enable the disable FP growth associator, we have to apply filter to the the .arff file from the preprocess. We have to choose from the unsupervised attribute filter, NOMINAL TO BINARY to convert all nominal value to binary, as the data only contain the nominal value and select the CLASS as NO CLASS and apply it. The following figure show the output of the step.


Weka Explorer

Preprocess | Classify | Cluster | Associate | Select attributes | Visualize

Open file... | Open URL... | Open DB... | Generate... | Undo | Edit... | Save...

Filter
Choose **NominalToBinary** -R first-last Apply

Current relation
Relation: Sales_Transactions_Dataset_Weekly-weka.filters.unsupervised.attribute.NominalToBinary-Rfirst-last
Instances: 811 Attributes: 917

Selected attribute
Name: Product_Code=P1 Type: Numeric
Missing: 0 (0%) Distinct: 2 Unique: 1 (0%)

Statistic	Value
Minimum	0
Maximum	1
Mean	0.001
StdDev	0.035

Attributes
All | None | Invert | Pattern

No.	Name
1	Product_Code=P1
2	Product_Code=P2
3	Product_Code=P3
4	Product_Code=P4
5	Product_Code=P5
6	Product_Code=P6
7	Product_Code=P7
8	Product_Code=P8
9	Product_Code=P9
10	Product_Code=P10
11	Product_Code=P11
12	Product_Code=P12
13	Product_Code=P13
14	Product_Code=P14
15	Product_Code=P15
16	Product_Code=P16
17	Product_Code=P17
18	Product_Code=P18
19	Product_Code=P19
20	Product_Code=P20
21	Product_Code=P21
22	Product_Code=P22

Remove

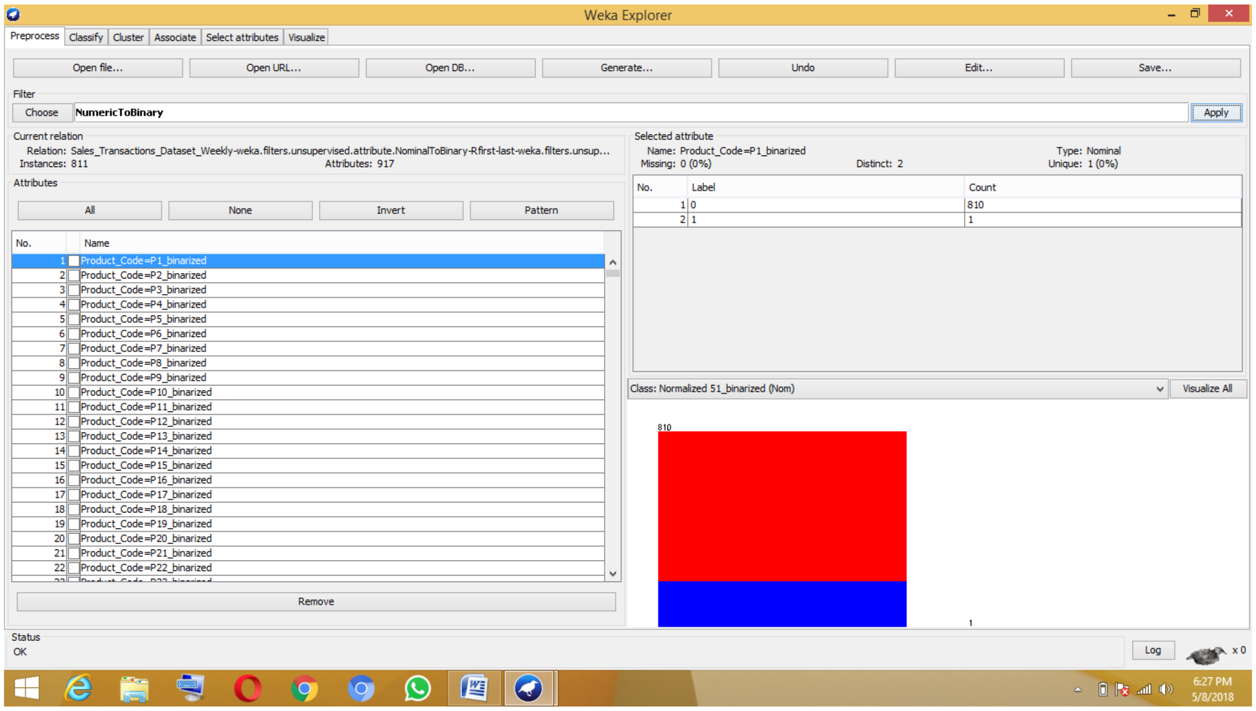
Class: Normalized 51 (Num) Visualize All

Status
OK

Log x 0

6:22 PM
5/8/2018

IV. Now we also have to apply NUMERIC TO BINARY filter form the unsupervised attribute filter to convert all the numeric value to binary select the CLASS as NO CLASS and apply it. The following figure show the output of the step.



Filter: NumericToBinary

Current relation
Relation: Sales_Transactions_Dataset_Weekly-weka.filters.unsupervised.attribute.NominalToBinary-Rfirst-last-weka.filters.unsup...
Instances: 811 Attributes: 917

Selected attribute
Name: Product_Code=P1_binarized Type: Nominal
Missing: 0 (0%) Distinct: 2 Unique: 1 (0%)

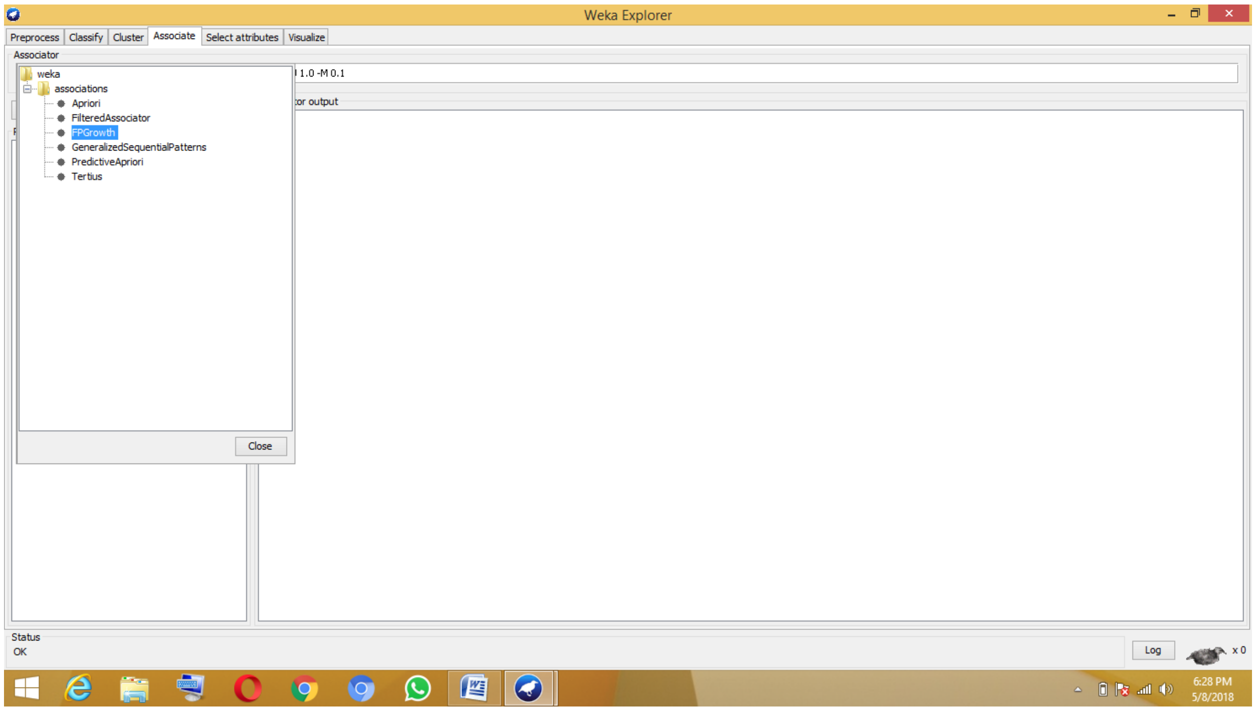
No.	Label	Count
1	0	810
2	1	1

No.	Name
1	Product_Code=P1_binarized
2	Product_Code=P2_binarized
3	Product_Code=P3_binarized
4	Product_Code=P4_binarized
5	Product_Code=P5_binarized
6	Product_Code=P6_binarized
7	Product_Code=P7_binarized
8	Product_Code=P8_binarized
9	Product_Code=P9_binarized
10	Product_Code=P10_binarized
11	Product_Code=P11_binarized
12	Product_Code=P12_binarized
13	Product_Code=P13_binarized
14	Product_Code=P14_binarized
15	Product_Code=P15_binarized
16	Product_Code=P16_binarized
17	Product_Code=P17_binarized
18	Product_Code=P18_binarized
19	Product_Code=P19_binarized
20	Product_Code=P20_binarized
21	Product_Code=P21_binarized
22	Product_Code=P22_binarized

Class: Normalized 51_binarized (Nom)

Status
OK

V. Now in the associate tab, the FP Growth associator is enable and ready to choose to generate the FP Growth rule.

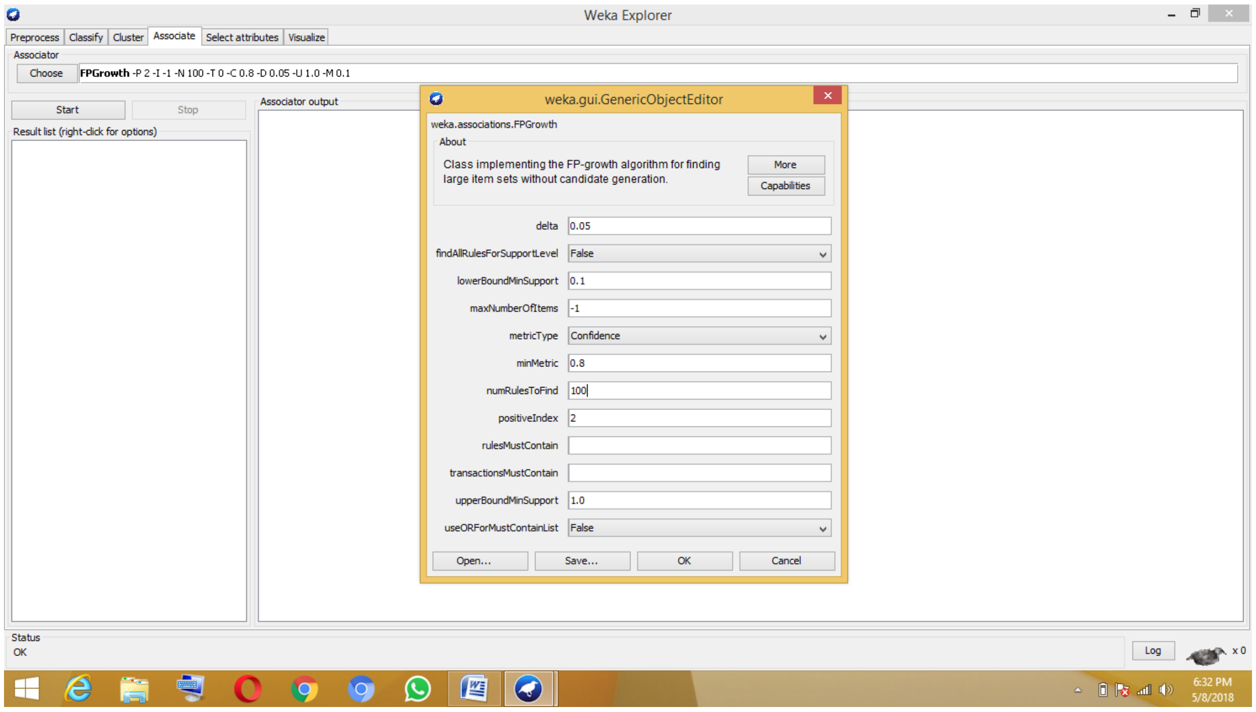


Associator
weka
associations
Apriori
FilteredAssociator
FPGrowth
GeneralizedSequentialPatterns
PredictiveApriori
Tertius

1.0 -M 0.1

Status
OK

VI. Now choosing the FP Growth associator we can also change its parameter by just clicking on the FP Growth chosen option. We change the num of rule to find to 100 and set the minimum support to 0.8 and other parameter accordingly and save it.



FPGrowth -P 2 -I -1 -N 100 -T 0 -C 0.8 -D 0.05 -U 1.0 -M 0.1

weka.gui.GenericObjectEditor

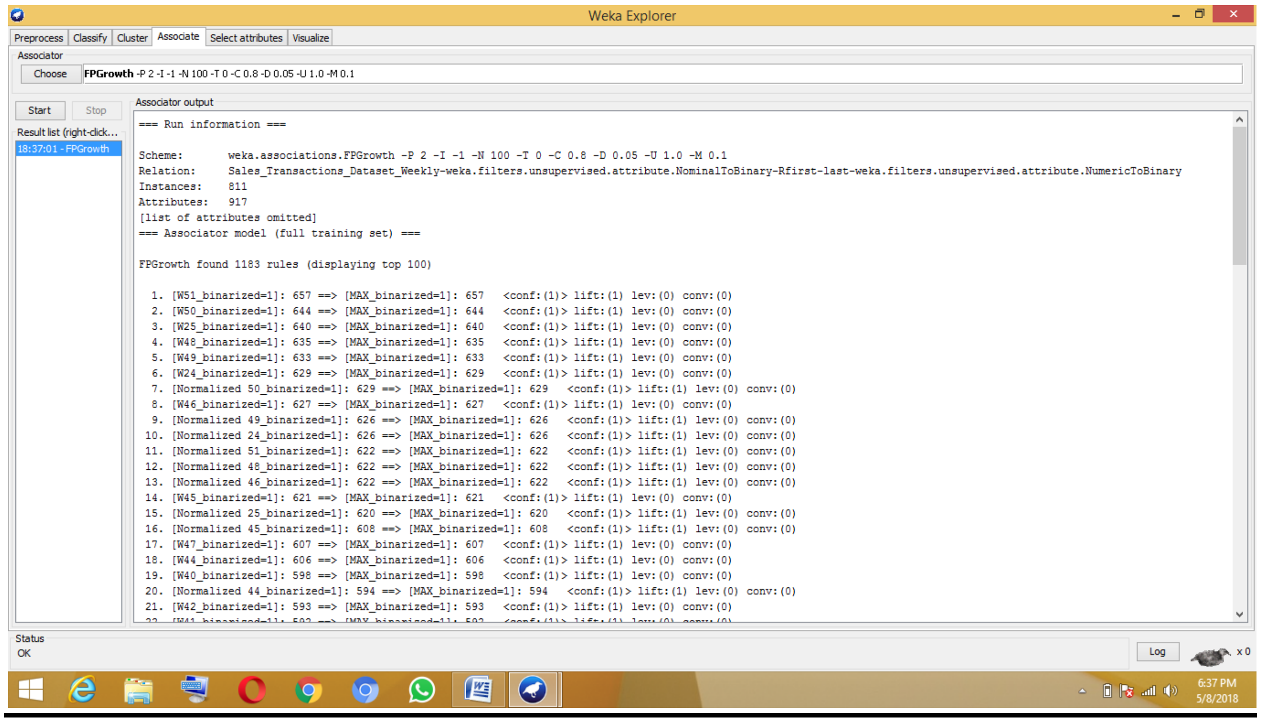
weka.associations.FPGrowth

About

Class implementing the FP-growth algorithm for finding large item sets without candidate generation.

delta	0.05
findAllRulesForSupportLevel	False
lowerBoundMinSupport	0.1
maxNumberOfItems	-1
metricType	Confidence
minMetric	0.8
numRulesToFind	100
positiveIndex	2
rulesMustContain	
transactionsMustContain	
upperBoundMinSupport	1.0
useORForMustContainList	False

VII. After setting all the parameter we just need to click on the start button to generate the rules. Accordingly all FP rule with the given parameter are generated.

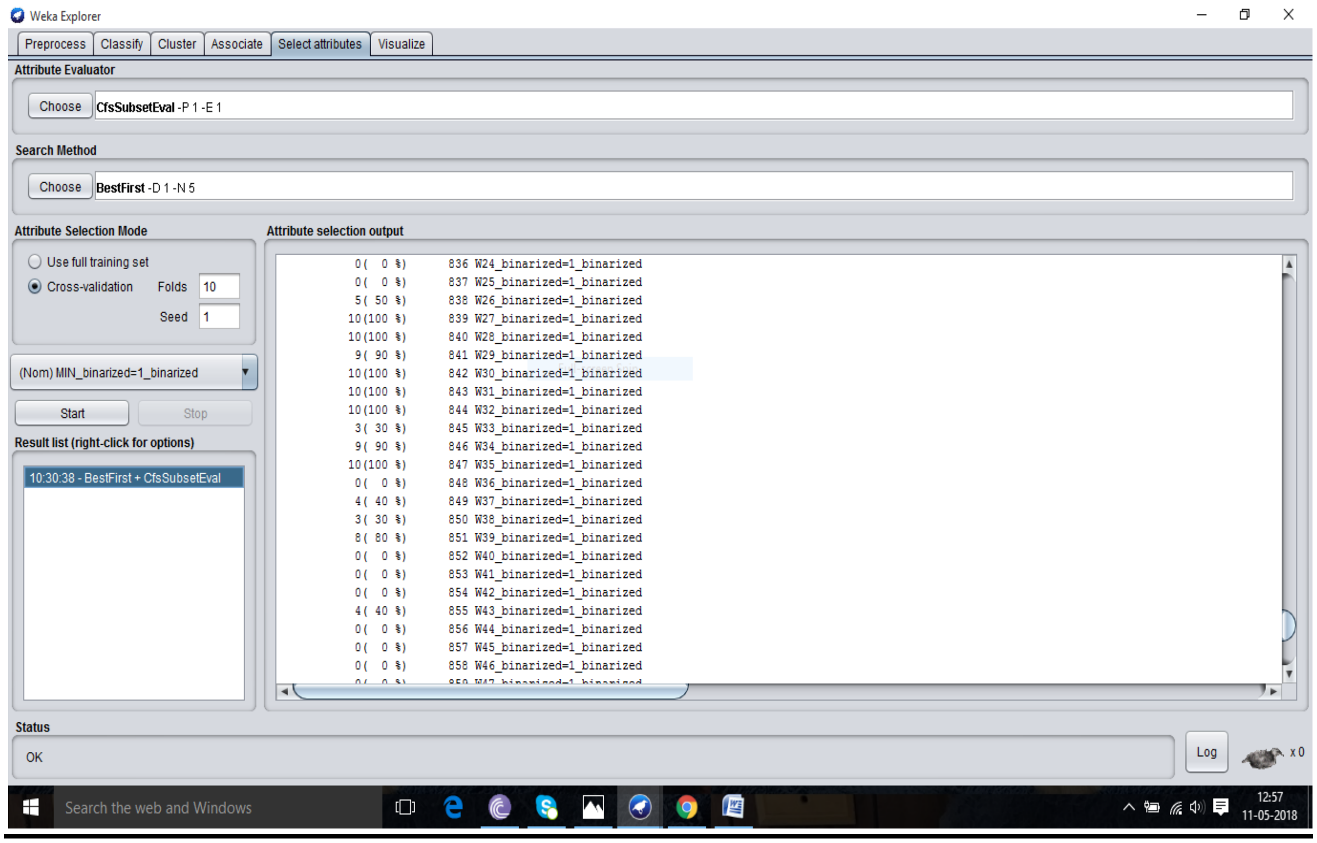


FPGrowth -P 2 -I -1 -N 100 -T 0 -C 0.8 -D 0.05 -U 1.0 -M 0.1

```
=== Run information ===

Scheme:       weka.associations.FPGrowth -P 2 -I -1 -N 100 -T 0 -C 0.8 -D 0.05 -U 1.0 -M 0.1
Relation:     Sales_Transactions_Dataset_Weekly-weka.filters.unsupervised.attribute.NominalToBinary-Rfirst-last-weka.filters.unsupervised.attribute.NumericToBinary
Instances:    811
Attributes:   917
[list of attributes omitted]
=== Associator model (full training set) ===

FPGrowth found 1183 rules (displaying top 100)

  1. [W51_binarized=1]: 657 ==> [MAX_binarized=1]: 657   <conf:(1)> lift:(1) lev:(0) conv:(0)
  2. [W50_binarized=1]: 644 ==> [MAX_binarized=1]: 644   <conf:(1)> lift:(1) lev:(0) conv:(0)
  3. [W25_binarized=1]: 640 ==> [MAX_binarized=1]: 640   <conf:(1)> lift:(1) lev:(0) conv:(0)
  4. [W48_binarized=1]: 635 ==> [MAX_binarized=1]: 635   <conf:(1)> lift:(1) lev:(0) conv:(0)
  5. [W49_binarized=1]: 633 ==> [MAX_binarized=1]: 633   <conf:(1)> lift:(1) lev:(0) conv:(0)
  6. [W24_binarized=1]: 629 ==> [MAX_binarized=1]: 629   <conf:(1)> lift:(1) lev:(0) conv:(0)
  7. [Normalized 50_binarized=1]: 629 ==> [MAX_binarized=1]: 629   <conf:(1)> lift:(1) lev:(0) conv:(0)
  8. [W46_binarized=1]: 627 ==> [MAX_binarized=1]: 627   <conf:(1)> lift:(1) lev:(0) conv:(0)
  9. [Normalized 49_binarized=1]: 626 ==> [MAX_binarized=1]: 626   <conf:(1)> lift:(1) lev:(0) conv:(0)
 10. [Normalized 24_binarized=1]: 626 ==> [MAX_binarized=1]: 626   <conf:(1)> lift:(1) lev:(0) conv:(0)
 11. [Normalized 51_binarized=1]: 622 ==> [MAX_binarized=1]: 622   <conf:(1)> lift:(1) lev:(0) conv:(0)
 12. [Normalized 48_binarized=1]: 622 ==> [MAX_binarized=1]: 622   <conf:(1)> lift:(1) lev:(0) conv:(0)
 13. [Normalized 46_binarized=1]: 622 ==> [MAX_binarized=1]: 622   <conf:(1)> lift:(1) lev:(0) conv:(0)
 14. [W45_binarized=1]: 621 ==> [MAX_binarized=1]: 621   <conf:(1)> lift:(1) lev:(0) conv:(0)
 15. [Normalized 25_binarized=1]: 620 ==> [MAX_binarized=1]: 620   <conf:(1)> lift:(1) lev:(0) conv:(0)
 16. [Normalized 45_binarized=1]: 608 ==> [MAX_binarized=1]: 608   <conf:(1)> lift:(1) lev:(0) conv:(0)
 17. [W47_binarized=1]: 607 ==> [MAX_binarized=1]: 607   <conf:(1)> lift:(1) lev:(0) conv:(0)
 18. [W44_binarized=1]: 606 ==> [MAX_binarized=1]: 606   <conf:(1)> lift:(1) lev:(0) conv:(0)
 19. [W40_binarized=1]: 598 ==> [MAX_binarized=1]: 598   <conf:(1)> lift:(1) lev:(0) conv:(0)
 20. [Normalized 44_binarized=1]: 594 ==> [MAX_binarized=1]: 594   <conf:(1)> lift:(1) lev:(0) conv:(0)
 21. [W42_binarized=1]: 593 ==> [MAX_binarized=1]: 593   <conf:(1)> lift:(1) lev:(0) conv:(0)
 22. [W41_binarized=1]: 592 ==> [MAX_binarized=1]: 592   <conf:(1)> lift:(1) lev:(0) conv:(0)
```

8. Result And Testing

Now choosing select attribute, go to the search method and choose the Best First- D 1 –N 5 method, then go to the attribute selection mode and choose the cross-validation. Set the Folds and Seed. Choose the “(Nom) MIN _ binarized=1_binarized” and click the “Start” button. And we are choosing only those item set which have 100% s


Attribute Evaluator: CfsSubsetEval -P 1 -E 1

Search Method: BestFirst -D 1 -N 5

Attribute Selection Mode: Cross-validation, Folds 10, Seed 1

(Nom) MIN_binarized=1_binarized

Result list: 10:30:38 - BestFirst + CfsSubsetEval

Attribute selection output

```
         0(  0 %)      836 W24_binarized=1_binarized
         0(  0 %)      837 W25_binarized=1_binarized
         5( 50 %)      838 W26_binarized=1_binarized
        10(100 %)      839 W27_binarized=1_binarized
        10(100 %)      840 W28_binarized=1_binarized
         9( 90 %)      841 W29_binarized=1_binarized
        10(100 %)      842 W30_binarized=1_binarized
        10(100 %)      843 W31_binarized=1_binarized
        10(100 %)      844 W32_binarized=1_binarized
         3( 30 %)      845 W33_binarized=1_binarized
         9( 90 %)      846 W34_binarized=1_binarized
        10(100 %)      847 W35_binarized=1_binarized
         0(  0 %)      848 W36_binarized=1_binarized
         4( 40 %)      849 W37_binarized=1_binarized
         3( 30 %)      850 W38_binarized=1_binarized
         8( 80 %)      851 W39_binarized=1_binarized
         0(  0 %)      852 W40_binarized=1_binarized
         0(  0 %)      853 W41_binarized=1_binarized
         0(  0 %)      854 W42_binarized=1_binarized
         4( 40 %)      855 W43_binarized=1_binarized
         0(  0 %)      856 W44_binarized=1_binarized
         0(  0 %)      857 W45_binarized=1_binarized
         0(  0 %)      858 W46_binarized=1_binarized
```

Status: OK

9. FUTURE ASPECT

Abundant literature is published in research into frequent pattern mining However, based on our view; there are still several critical research problems that need to be solved before frequent pattern mining can become a cornerstone approach in data mining applications. First, the most focused and extensively studied topic in frequent pattern mining is perhaps scalable mining methods. When we are working with data streams still it is a research challenge to derive a compact but high quality set of patterns that are most useful in applications. The set of frequent patterns derived by most of the current pattern mining methods including ours give approximate patterns as stream is flowing continuously and some data is lost in the process of analyzing the stream. There are proposals on reduction of such a huge data set, including closed patterns, maximal patterns, approximate patterns, condensed pattern bases, representative patterns, clustered patterns, and discriminative frequent patterns, but still it is research issue to mine pattern sets in both compactness and representative quality for a particular application. To make frequent pattern mining an essential task in data mining, much research is needed to further develop pattern-based mining methods. For example, classification is an essential task in data mining. Construction of better classification models using frequent patterns than most other classification methods is again a research issue.

On one side, it is important to go to the core part of pattern mining algorithms, and analyze the theoretical properties of different solutions. Much work is needed to explore new applications of frequent pattern mining. For example, bioinformatics has raised a lot of challenging problems, and we believe frequent pattern mining may contribute a good deal to it with further research efforts.

10. Conclusion

In our project we have analysed various pattern mining algorithms. FP growth is the most popular algorithm of the lot. Parallel FP Growth is the algorithm that runs on parallel processors ,which has superior run time than FP growth. Compact Pattern tree has added advantage over FP Tree because it scans the DB only once. FP Tree is an efficient algorithm to find frequent patterns in transactional database. FP-Growth is the first successful tree base algorithm for mining the frequent item sets by using its various techniques mentioned its performance can be increased as per the requirements. As in the case of large database its structure fails to fit into main memory hence for this purpose new techniques have been came into existence for reducing data-set and generating tree-structure that may consist of the variations of the classic FP-Tree and result in higher performance.
We have tried to implement the basic FP Growth algorithm for sufficient research work and also we have utilized WEKA for referring the process of association rule mining. In future we plan to implement other pattern mining algorithms with Weka.

11. REFERENCES

1. Bezdek, J. C., & Pal, S. K. (1992). *Fuzzy models for pattern recognition: Methods that search for structures in data*. New York: IEEE Press
2. Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., & Uthurusamy, R. (Eds.). (1996). *Advances in knowledge discovery and data mining.* AAAI/MIT Press.
3. Han, J., & Kamber, M. (2000). *Data mining: Concepts and techniques*: Morgan Kaufmann.
4. Hastie, T., Tibshirani, R., & Friedman, J. H. (2001). *The elements of statistical learning: Data mining, inference, and prediction:* New York: Springer.
5. Jain, A. K., & Dubes, R. C. (1988). *Algorithms for clustering data*. New Jersey: Prentice Hall.
6. Jensen, F. V. (1996). *An introduction to bayesian networks*. London: University College London Press.
7. Kaufman, L., & Rousseeuw, P. J. (1990). *Finding groups in data: An introduction to cluster analysis*. New York: John Wiley.
8. Michie, D., Spiegelhalter, D. J., & Taylor, C. C. (1994). *Machine learning, neural and statistical classification*: Ellis Horwood.

12. LIST OF FIGURES