Indian Journal of Science and Technology, *Vol 9(21), DOI: 10.17485/ijst/2016/v9i21/95292, June 2016*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

A Multi-Class Based Algorithm for Finding Relevant Usage Patterns from Infrequent Patterns of Large Complex Data

Sujatha Kamepalli[1*], Rajasekhara Rao Kurra[2] and Y. K. Sundara Krishna[3]

[1]Krishna University, Machilipatnam, Andhra Pradesh, India; sujatha101012@gmail.com
[2]Sri Prakash College of Engineering and Technology, Thuni, Andhra Pradesh, India; krr_it@yahoo.co.in
[3]CSE Department, Krishna University, Machilipatnam, Andhra Pradesh, India; yksk2010@gmail.com

Abstract

Background/Objective: With the development of data mining techniques to analyze large amount of complex data has played an essential role in several areas like cloud computing, medical databases, geographical information retrieval, etc. The automatic evaluation of cloud patterns is a challenging task due to the large amount of interesting patterns can be extracted. However, how to find infrequent patterns is still an open issue in cloud computing. **Methods:** Conventional approaches are mainly depends on quantitative datasets with support and confidence measures. Due to the large amount of cloud storage data, it is very difficult to extract the weighted association rules based on the server usage statistics. Traditional techniques are implemented on the data samples with the same attribute type. Due to this fact, a multi-class algorithm is proposed to find relevant usage patterns of large complex data. **Findings:** Proposed approach does not rely on any probabilistic closure measures and quantitative data. This approach minimizes the database scans and optimizes the infrequent cloud patterns. **Applications/Improvements:** Experimental results show that, proposed work generates high quality cloud patterns compared to traditional quantitative rule mining techniques.

Keywords: Complex Cloud Data, Infrequent Association Rules, Multiclass Attributes

1. Introduction

Complex data is growing at a phenomenal rate due to the e-commerce, internet and social networks. With the rapid growth of big-data, the need for efficiently extracting the cloud properties in a reliable or scalable way is unprecedentedly more. Complex data mining has become crucial for may cloud vendors to extract relevant patterns from the huge data sets in order to support their operations and to take right decisions. Given a set of transaction T, the goal of association rule mining is to find all rules having support ≥ minsup threshold, confidence ≥ minconf threshol[1].

For association rule mining, our goal is to extract valuable and storage knowledge from large amounts of data by using large scale data mining. Those large volumes of data are huge wealth for any cloud provider or enterprise. As a result, large set of infrequent patterns provides decision making for any organizations. Hadoop is one of the open source distributed framework used to process the cloud data and its efficiency without generating decision patterns.

Infrequent rule mining is a method to find the hidden patterns in complex data and extract inferences on how a subset of attributes influences the existence of other super subsets.

** Author for correspondence*

All infrequent patterns are not optimal due to the redundancy in the frequent patterns. Usually, frequent association rule mining approaches focus on finding frequent relationship between the attributes. Negative Association Rule Mining (ARM) works similar to positive ARM, but in reverse manner. But the problem with the negative ARM is to consume more memory and time. The association rule mining challenge can be classified as categorical and numerical attributes in the dataset. Most of the traditional[2-6] techniques are implemented on numerical attributes for generating frequent patterns.

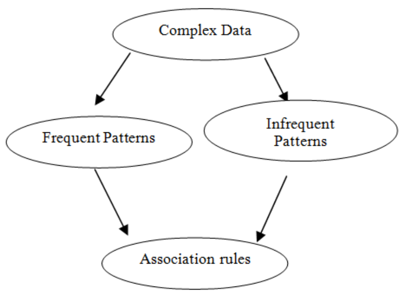

Figure 1. Two-Phase ARM process.

ARM is a two phase process as shown in Figure 1.

- 1) Generating frequent item sets.
- 2) Quality association rules generation from the frequent item sets.

Since the first step represents the computational knowledge extraction, optimal solutions have been proposed [6-10] to generate patterns on multi-core processors.

Based on the parallel and grid computing constraints under the cloud environment, a set of data partitioning and allocation approaches have been proposed in the literature. Data partitioning and replication are two major ways to optimize the scalability and availability of distributed databases. However, partitioning approaches used to distribute the cloud user workloads in an effective way and decreases reading latencies. On the other hand, replication optimizes the chance of data inconsistencies and also system capabilities. Therefore, the efficient rule based detection on the cloud data should be used to eliminate their effects.

Associative classification is one of the major tasks in knowledge discovery. Existing studies have shown that associative algorithms are used to handle unstructured data. The first method used to handle unstructured data is CBA,which considers the apriori approach to get association patterns based on class labels. Rule mining and classification is used to mine relationships between attributes and class labels.

In the literature, many studies have been introduced to find rules for large data and construct an association based classifier such as C45, RIPPER, Multi-class Classification based on Association Rules (MCAR), CBA, etc. Most of the research work is to solve single class association mining. Most of the traditional methods fail to consider only one class label and ignoring the other classes. At present, most of the methods identify the optimal rules based on the interesting measures. In the multiclass complex data, conventional approache does not find the optimal rules based on single class interesting measures. CAN tree, FP tree and CP tree have been implemented in the literature on quantitative or transactional databases. These algorithms are divide and conquer and partitioned based methods are used that divide data into small sets for mining patterns in transactional database, which gradually reduce search and memory space. In CP tree contains frequent and infrequent items at the end of the frequent sets generation. Since "Can tree" is not stored the frequencies in descending order, it usually results poor computation in tree size compared to FP-tree datastructure. The CP-tree implements the concept of dynamic tree construction to generate a compact tree at runtime[11]. FP-tree is a compact tree structure used to get frequent items in transacrtion mining. However, it only handles the frequent items in a dataset and it is a two-phase solution. CanTree provides a single-phase solution which maintains complete dataset information suitable for interactive and incremental mining. However, it suffers very high mining time due to the canonical concept of its tree structure[11-14].

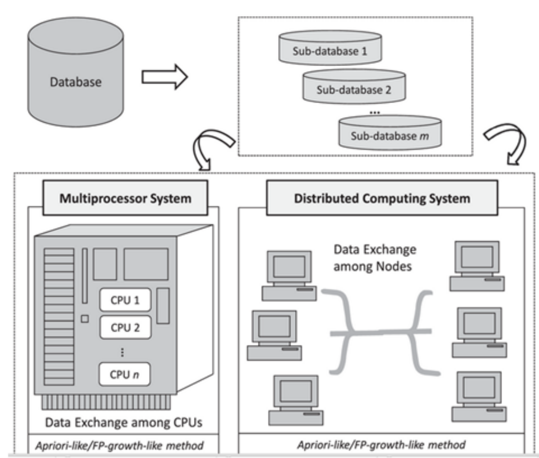

Figure 2. Traditional Parallel Fp-Tree Mechanism.

The compaction achieved in FP-tree is small when the data is distributed unevenly. Due to this, FP-growth would require a lot of effort to combine fragment patterns with no frequent itemsets being found. Parallel FP –tree has been proposed in based on FP-tree data structure to mine association rules on multi-processor systems.

This method splits the database in several non-overlapping sets based on the number of cloud processors and each processor use FP-tree to exchange information between the multiple processsors. Balanced Tidset –based parallel FP-tree technique is used to extract frequent processor related rules as shown in Figure 2 .

2. Multi-Class Based Infrequent Mining Algorithm for Large Complex Dataset

In this proposed approach a real time cloud server is used for data preparation. The overall workflow is described in the Figure 3. Cloud server has 'n' number of virtual instances with different types of services. Each cloud instance, includes instance id, instance name, application status, network overhead, load balancing, memory usage, etc. Each cloud provider need to optimize his cloud services based on the cloud instance properties. In this work, complex distributed data was prepared using cloud virtual instance properties.

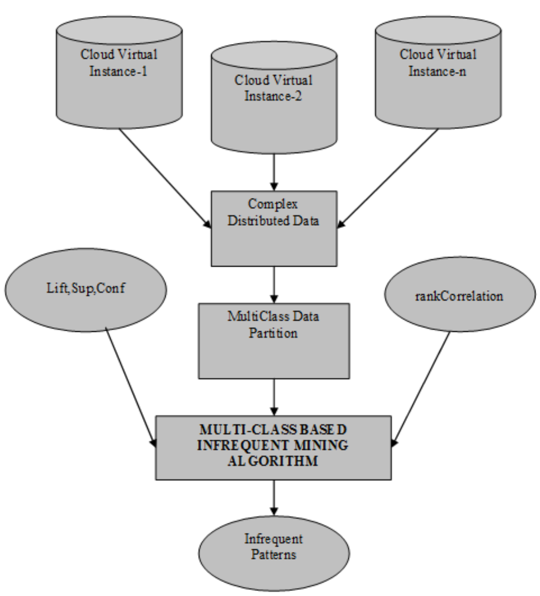

Figure 3. Proposed Flow Chart.

Cloud complex data has multiple attributes with different classes and values. Complex data is partitioned based on the cloud instance properties. Proposed multi-class based infrequent mining algorithm is used to find interesting patterns and its relationship within cloud instances.

Notations:
CSD: Class Based Sub-Datasets
CBM: Class Based Boolean Matrix
PAR:Positive Association Rules
IAR:Infrequent Association Rules

Algorithm Explanation:
In the proposed algorithm, Dataset is initialized and then data objects are extracted using different class labels. For each sub dataset in the class based datasets generate both transactions based boolean and class based boolean matrices for infrequent association patterns. Afterwards generate 1-item and m-item candidate sets to each sub dataset. In the step 5, lift and correlation computations are performed on the associated items and then positive and infrequent items are classified . Infrequent items are inserted into CPTree to generate infrequent association patterns using the correlation threshold condition. Finally, infrequent association patterns are generated from the CPTree.

Algorithm:
Step 1: Extract class based sub-datasets CSD.
Step 2: For each dataset in CSD.
Do
Generate Transaction based Boolean Matrix TBM.
Generate Class based Boolean Matrix $CBM_1 ... CBM_n$ where n is number of classes.
Done
Step 3: Generate 1 to n items candidate sets CS .
Step 4:
ρ_{min} minimum threshold
PAR $\leftarrow \emptyset$; IAR $\leftarrow \emptyset$
scan the database CS and extract 1-item frequent sets (f_1)
for ($m = 2, f_{m-1} \mathrel{!}= \emptyset, m++$)
Do
$R_m = Join(f_{m-1}, f_1)$;
Done
Step 5:
For each item $i \in R_m$

do
lfv=lift (D,i)
if lfv≥ρ_{min} then
$f_m = f_m \cup \{i\}$
For each items associated with item {i} in D

φ_1=getBooleanOccurences(CBM);
$\varphi_2 \ldots \varphi_{n+1}$ =getBooleanOccurences($CBM_1 \ldots CBM_n$);
For each class k in $\varphi_2 \ldots \varphi_{n+1}$
Do

σ_{corr} = Correlation(i, φ_1, φ_k)
if $\sigma_{corr} \geq \rho_{min}$
then
if conf(φ_1, φ_k) ≥ $conf_{min}$ then
PAR ← PAR ∪ {φ_1, φ_k}
else if conf(φ_1, φ_k) ≥$conf_{min}$and
sup(¬φ_1, ¬φ_k) ≥ρ_{min}then
IAR ← IAR ∪ {¬φ_1, ¬φ_k }
Insert CPtree(IAR,ρ_{min})
endif
endif
if $\sigma_{corr} \leq -\rho_{min}$
then
if conf(φ_1, ¬φ_k) ≥ $conf_{min}$ then
IAR ← IAR ∪ {φ_1, ¬φ_k }
Insert CPtree(IAR,ρ_{min})
endif
if conf(¬φ_1, φ_k) ≥ $conf_{min}$ then
IAR ← IAR ∪ {¬φ_1, φ_k }
Insert CPtree(IAR,ρ_{min})
endif
endif
Infrequent Rules ← getPatterns(CPtree)
done
done
Done

Lift calculates the ratio between the rules support and confidence of the itemset in the rule consequent based on the each selected class.

$$\text{lift} = \text{prob}(c_i / D_i) / \text{prob}(c_i, D) \qquad (1)$$

prob(c_i, / D): Probability of occurrence of an item in samples of ith class .
prob(c_i, D): Probability of occurrence of an item in a dataset of ith class.

Correlation
$$|D_i|\,\text{lift}(i,\phi_1) - |D_i|\,\text{lift}(i,\phi_2) / |D| \sqrt{\text{lift}(i,\phi_1)^2 - \text{lift}(i,\phi_2)^2}$$

Correlation formula is used to find the correlation between the associated items with different class labels. Here correlation computation is performed using lift computation between the associated items. Correlation computation has three different ranges such as negative ,zero and positive. If the computed value is positive, then the items are highly associated to each other. If the computed value is negative, items are not associated for infrequent patterns. If the value is zero, then the items are not related to each other for pattern generation.

3. Experimental Results

In this experimental study, dynamic data from the cloud server was used with attributes such as instance id, load Moodle balancing number, duration, cloud data size,server status etc. All experiments are performed with the real time Amazon cloud instances and client configurations as Intel(R) CPU 2.13GHz, 4 GB RAM, and the minimum OS platform is Microsoft Windows 7 Professional (SP2).

Sample Data:
Ins_02318,Load_8183,23:21,586,4584,500,205,fail
Ins_02318,Load_8183,11:21,704,9804,300,1532,fail
Ins_02 322,Load_8223,8:55,814,9323,500,6975,success
Ins_02324,Load_8243,17:48,530,37130,500,1414,fail
Ins_02331,Load_8313,19:18,518,58120,500,8986,fail
Ins_02344,Load_8443,11:49,380,30858,200,5617,success
Ins_02339,Load_8393,3:8,771,19776,404,6716,fail
Ins_02337,Load_8373,5:36,385,58968,500,7070,success
Ins_02321,Load_8213,22:13,632,58018,200,3398,success
Ins_02345,Load_8453,3:22,579,44768,500,9425,fail
Ins_02333,Load_8333,23:7,471,65753,500,844,fail
Ins_02345,Load_8453,21:13,645,45029,300,4695,fail
Ins_02345,Load_8453,5:7,564,17987,500,8782,fail
Ins_02335,Load_8353,1:23,808,31945,404,3652,success
Ins_02317,Load_8173,12:16,704,3593,300,353,success
Ins_02334,Load_8343,4:53,561,21533,200,8614,success
Ins_02345,Load_8453,4:12,462,47757,404,8509,success
Ins_023 25,Load_8253,13:46,705,1547,404,1119,success
Ins_02318,Load_8183,17:33,848,27626,200,440,fail
Ins_02310,Load_8103,6:2,794,35446,300,765,success

Generated Sample Infrequent Rules

usage <= 889.0 AND appsize >= 173.0 -> appexetime >= 59.0
appexetime >= 59.0 AND appsize >= 173.0 -> serverstat != success
usage >= 250.0 -> serverstat != success
appexetime <= 9947.0 AND appexetime >= 59.0 -> loadbal != Load_8383
appexetime <= 9947.0 -> usage <= 889.0
appexetime <= 9947.0 -> appsize >= 173.0
appexetime <= 9947.0 AND appexetime >= 59.0 AND usage <= 889.0 -> serverstat != success
loadbal != Load_8383 AND appexetime >= 59.0 -> appsize >= 173.0
loadbal != Load_8383 -> usage >= 250.0
serverstat != success -> loadbal != Load_8383
loadbal != Load_8383 -> timestamp != 3:52
appexetime >= 59.0 -> serverstat != success
usage <= 889.0 -> appexetime >= 59.0
usage <= 889.0 AND appsize >= 173.0 AND usage >= 250.0 -> serverstat != success
appexetime >= 59.0 AND appsize >= 173.0 AND usage >= 250.0 -> serverstat != success
appexetime >= 59.0 AND usage >= 250.0 -> appsize >= 173.0
appexetime <= 9947.0 -> loadbal != Load_8383
serverstat != success AND usage >= 250.0 -> loadbal != Load_8383
appexetime >= 59.0 AND usage <= 889.0 -> appsize >= 173.0
loadbal != Load_8383 -> appsize >= 173.0
loadbal != Load_8383 AND appsize >= 173.0 -> appexetime >= 59.0
usage <= 889.0 -> appsize >= 173.0
appsize >= 173.0 AND usage <= 889.0 -> serverstat != success
sage >= 250.0 AND usage <= 889.0 -> appsize >= 173.0
appsize >= 173.0 -> usage <= 889.0
usage <= 889.0 -> loadbal != Load_8383
usage >= 250.0 -> timestamp != 3:52
usage >= 250.0 -> appsize >= 173.0
serverstat != success AND usage <= 889.0 -> appexetime >= 59.0
appexetime >= 59.0 -> usage >= 250.0
appsize >= 173.0 -> loadbal != Load_8383
appsize >= 173.0 AND usage >= 250.0 -> serverstat != success
appexetime <= 9947.0 -> serverstat != success
appexetime >= 59.0 AND appsize >= 173.0 -> serverstat != success
usage >= 250.0 AND appexetime >= 59.0 -> appsize >= 173.0
appexetime >= 59.0 -> usage <= 889.0
serverstat != success AND appsize >= 173.0 -> usage <= 889.0
appexetime <= 9947.0 AND usage >= 250.0 -> loadbal != Load_8383
appsize >= 173.0 AND usage >= 250.0 -> loadbal != Load_8383
appexetime >= 59.0 AND loadbal != Load_8383 -> serverstat != success
appexetime >= 59.0 -> usage >= 250.0
appexetime >= 59.0 AND appsize >= 173.0 -> serverstat != success
loadbal != Load_8383 -> serverstat != success
usage >= 250.0 AND serverstat != success -> appexetime >= 59.0
appexetime >= 59.0 AND usage >= 250.0 -> serverstat != success
serverstat != success -> usage <= 889.0
usage <= 889.0 AND usage >= 250.0 -> serverstat != success
loadbal != Load_8383 AND serverstat != success -> appsize >= 173.0
usage <= 889.0 -> appsize >= 173.0
appexetime >= 59.0 AND usage <= 889.0 -> serverstat != success
usage >= 250.0 -> loadbal != Load_8383
usage <= 889.0 AND appsize >= 173.0 -> appexetime >= 59.0
serverstat != success AND appexetime >= 59.0 -> usage >= 250.0
appsize >= 173.0 AND loadbal != Load_8383 -> serverstat != success
usage <= 889.0 AND usage >= 250.0 -> loadbal != Load_8383
appexetime >= 59.0 -> appsize >= 173.0
appexetime >= 59.0 -> appsize >= 173.0
usage >= 250.0 AND appsize >= 173.0 -> appexetime >= 59.0
appexetime >= 59.0 AND usage >= 250.0 -> serverstat != success
usage >= 250.0 -> appexetime >= 59.0
appexetime >= 59.0 AND loadbal != Load_8383 AND

usage <= 889.0 -> serverstat != success
appexetime >= 59.0 AND appsize >= 173.0 -> serverstat != success
appexetime >= 59.0 -> loadbal != Load_8383
usage <= 889.0 -> loadbal != Load_8383
serverstat != success -> appsize >= 173.0
serverstat != success AND appsize >= 173.0 -> loadbal != Load_8383
loadbal != Load_8383 -> serverstat != success
usage <= 889.0 AND appsize >= 173.0 -> serverstat != success
appexetime <= 9947.0 AND appsize >= 173.0 -> serverstat != success
appsize >= 173.0 AND usage <= 889.0 AND appexetime >= 59.0 -> serverstat != success
appexetime >= 59.0 AND serverstat != success -> appsize >= 173.0
usage >= 250.0 AND appsize >= 173.0 -> loadbal != Load_8383
serverstat != success -> appexetime >= 59.0
loadbal != Load_8383 AND usage <= 889.0 AND usage >= 250.0 -> serverstat != success
usage >= 250.0 -> appsize >= 173.0
appsize >= 173.0 AND serverstat != success -> appexetime >= 59.0
usage >= 250.0 AND loadbal != Load_8383 -> appexetime >= 59.0
appexetime >= 59.0 -> usage <= 889.0
appexetime <= 9947.0 AND appsize >= 173.0

4. Performance Metrics

Table 1, describes the number of cloud instances operated for infrequent patterns. In this experiment the cloud instance id,computational time, number of rules and memory space are computed and listed in the table.

Table 1. Proposed model computation results

Number of Cloud Instances	Computation Time(ms)	Rules Count	Memory Space(kbytes)
1000-ins	1344	10	4.66
2000-ins	2433	13	5.77
3000-ins	3533	17	7.43
4000-ins	4333	15	8.44
5000-ins	5227	9	1.022
10000-ins	5988	16	1.744

Figure 4, describes the number of cloud instances operated for infrequent patterns. In this experiment the number of rules and memory space are compared under different experiments.

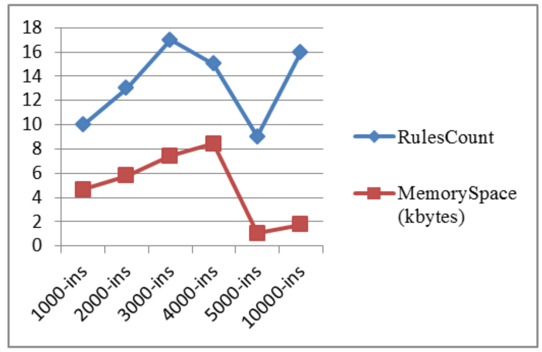

Figure 4. Infrequent Cloud rules and Memory Space.

Figure 5, describes the number of cloud instances operated for infrequent patterns. In this experiment the number of cloud instances and computational are compared under different experiments.

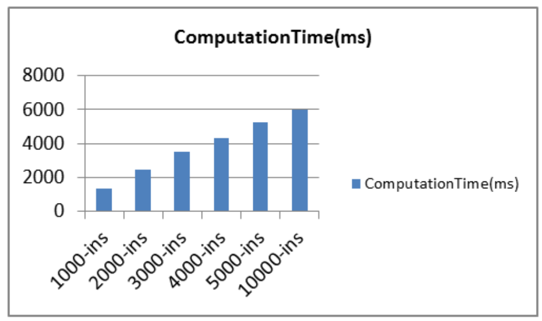

Figure 5. Infrequent Rules Computation Time and Number of Instances.

Table 2 describes the performance analysis of the proposed model with the traditional models in terms of accuracy. As the number of instances size increases the true positive rate in the proposed model increases compare to traditional models.

Table 2. Proposed and Traditional models accuracy comparison

No.of Cloud Instances	CP-Tree (Accuracy)	Parallel FP-Tree	Proposed Model
1000-ins	87.45	93.34	98.79
2000-ins	85.67	89.76	97.67
3000-ins	89.76	91.56	98.18
4000-ins	84.78	89.95	97.98
5000-ins	88.96	92.789	98.19
10000-ins	87.12	91.06	97.89

5. Conclusion

In this proposed work, an improved infrequent mining algorithm was implemented in the real time distributed cloud data. Proposed approach implemented on the data samples with the distinct attribute types. This approach generates high quality cloud usage patterns of large complex data. Proposed approach does not rely on any probabilistic closure measures and quantitative data. This approach minimizes the database scans and optimizes the infrequent cloud patterns. Experimental results show that, proposed work generates high quality cloud patterns compared to traditional quantitative rule mining techniques.

6. References

1. Shabana Asmi P, Justin Samuel S. An analysis and accuracy prediction of heart disease with association rule and other data mining techniques. Journal of Theoretical and Applied Information Technology. 2015; 79(2):254-60.
2. Cohen, Fast effective rule induction, in the Proceeding of the 12 International Conference on Machine Learning, Morgan Kaufmann, San Francisco, 1995, p.115-23.
3. Liu B, Hsu W, Ma Y. Integrating Classification and association rule mining. In: KDD '98, New York, NY, 1998 Aug, p.12-24.
4. Thabtah F, Cowling P, Peng YH. MMAC: A New Multi-Class, Multi-Label Associative Classification Approach. Fourth IEEE International Conference on Data Mining (ICDM'04). 2015 April; 7(1):34-56.
5. Lai Y, Zhong Zhi S. An efficient data mining framework on Hadoop using java persistence API, Computer and Information Technology, 2010. p. 434-40.
6. Caglierro L, Garza P. Infrequent Weighted Item set Mining using Frequent Pattern Growth. IEEE Transactions on Knowledge and Data Engineering. 2014; 26(4):1041-4347.
7. Delgado M, Ruiz MD, Sánchez D, Serrano JM. A formal model for mining fuzzy rules using the *RL* representation theory. Information Sciences. 2011; 181(23):5194-5213.
8. Gupta A, Mittal A, Bhattacharya A. Minimally Infrequent Itemset Mining Using Pattern-Growth Paradigm and Residual Trees, Proc. Int'l Conf. Management of Data (COMAD), 2011. p. 57-68.
9. Shyamala K. An Analysis on Efficient Resource Allocation Mechanisms in Cloud Computing. Indian Journal of Science and Technology. 2015 May; 8(9):814-21.
10. Sugunadevi P. Efficient Algorithm for Mining High Utility Itemsets. The International Journal of Science and Technoledge. 2014 May; 2(5):45-49.
11. Luca Cagliero and Paolo Garza. Infrequent Weighted Itemset Mining using Frequent Pattern Growth. IEEE Transactions on Knowledge and Data Engineering. 2013; 1-14.
12. Syed Khairuzzaman Tanbeer, Chowdhury Farhan Ahmed, Byeong-Soo Jeong, Young-Koo Lee. CP-Tree: A Tree Structure for Single-Pass Frequent Pattern Mining. In: SpringerVerlag Berlin Heidelberg: 2008. p. 1-6.
13. Leung, CK, Khan QI, Li Z, Hoque T. CanTree: A Canonical-Order Tree for Incremental Frequent-Pattern Mining. In: Knowledge and Information Systems. 2007; 11(3):287-311.
14. Jiawei Han, Jian Pei, Yiwen Yin, Runying Mao. Mining Frequent Patterns Without Candidate Generation: A Frequent-Pattern Tree Approach. In: Kluwer Academic Publishers. 2004. p. 53-87.