Computer and Information Science; Vol. 8, No. 3; 2015
ISSN 1913-8989 E-ISSN 1913-8997
Published by Canadian Center of Science and Education

Sequence Pattern Mining in Data Streams

H. M. Hijawi[1] & M. H. Saheb [1,2]

[1] Department of Computer Science, Arab American University, Palestine

[2] IT and Computer Engineering College, Palestine Polytechnic University, Palestine

Correspondence: M. H. Saheb. E-mail: alsaheb@ppu.edu

Received: May 12, 2015 Accepted: June 12, 2015 Online Published: July 31, 2015

doi:10.5539/cis.v8n3p64 URL: http://dx.doi.org/10.5539/cis.v8n3p64

Abstract

Sequential pattern mining in data streams environment is an interesting data mining problem. The problem of finding sequential patterns in static databases had been studied extensively in the past years, however mining sequential patterns in the data streams still an active field for researches. In this research a new greedy sequence pattern mining algorithm for the data streams is introduced, it will be used to find the strongly supported sequences. The proposed algorithm is built based on the sequence tree which is used to find the sequential patterns in static databases. The proposed algorithm divides the streams into patches or windows and each patch will update the sequence tree which built from the previous windows. An example is introduced to explain how this algorithm works. We also show the efficiency and the effectiveness of the proposed algorithm on a synthetic dataset and prove how it is suited for data streams environment. We showed experimentally that the proposed algorithm is more efficient than the PrefixSpan algorithm for patterns with any support less than 30% for CPU time and with any support less than 60% for memory usage.

Keywords: sequential patterns mining, data streams, sequence mining, sequence tree

1. Introduction

In recent years new applications have been emerged such as network traffic analysis, wireless sensor networks and user web clicks, this introduced a new kind of data called data streams (Marascu and Masseglia, 2005). A data stream is an ordered sequence of items which in many applications can be read once without storing them in the database due to the huge size and amount of data. Sequential pattern mining in data streams concerns about finding the sequence patterns in data without the need of multi scan of data, however the data is huge and generated continuously (Muthukrishnan, 2005). In this research we focus on the problem of mining sequence patterns in data streams such as web clicks and wireless sensor networks, and we propose a new algorithm for mining sequential patterns in the data streams.

In recent years many contributions have been published to solve the problem of mining frequent patterns (Tiwari et al., 2010), mining sequential patterns in transaction databases (Agrawal & Ramakishnan, 1998), and web frequent multi-dimensional sequential patterns (Hwang et al., 2011). A recent survey (Rao et al., 2013) provides an overview about mostly used sequence pattern mining algorithms and provides a comparative study between them. Another survey (Vijayarani & Sathya, 2012) presents the various applications of data streams and provides the analysis of frequent pattern mining papers in data streams. Incremental mining algorithm of sequential patterns based on sequence tree was introduced in (Liu et al., 2012), however this algorithm is not used in data streams environments because it stores all patterns even they are not frequent and this causes a lot of problems in data streams environments where there is a limitation in memory usage. Even more, the memory allocation of sequence tree will be larger than the size of sequence data base.

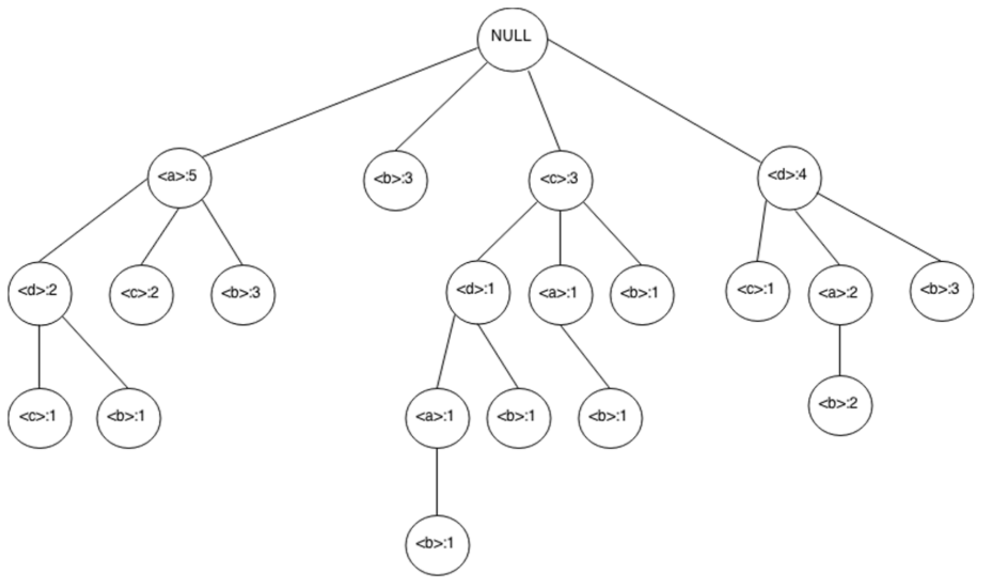


Figure 1. The sequence tree of the sequence data base in Table 1

From sequence tree we can traverse the tree and get all frequent sequences with the requested minimum support. The following are the frequent sequence patterns with Min_sup = 2, for Figure 1.

{<a>, <b>, <c>, <d>, <ad>, <ac>, <ab>, <da>, <dab>, <db>}

To make the sequence tree suitable for data streams where the traffic always active. The windows of sequences will be considered for each specific period of time, for example the size of the window will be 20 sequences. So the sequence tree will be updated for each window. Due to the limitation of memory in data streams environment such in sensor networks, the tree will be pruned after each specific number of windows. However the branches with support less than minimum support will be removed.

2. Problem Definition

Suppose that in the stream environment the monitored items set is I = {i_1, i_2... i_n} where n is the total number items. A sequence of items S is an ordered list from I denoted by <$s_1s_2 \ldots s_m$>. A sequence <$a_1a_2 \ldots a_n$> is a subsequence from < $b_1b_2 \ldots b_m$> if there exist integers $1 \leq \; i_1 < i_2 < \ldots < i_k \leq m$ such that $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i2} \ldots a_n \subseteq b_{in}$.

Given a sequence data base table T (Sid, Sequence), the support of sequence S donated by support(S) is the measure for how many times the sequence S appears in the sequence table. Min_sup is a user defined value to determine if the sequence is frequent or not. If support(S) > Min_sup the sequence S is frequent. Thus the problem of sequence pattern mining in data stream is to find the strongly frequent sequential patterns with support bigger than Min_sup with taking in consideration the memory and processing power limitations (Lee et al., 2014).

The following example is given to illustrate the sequences in data stream environment, however we assume an element has only one item; this is common in sensor networks environment where each sensor is specialized in sensing one phenomena.

Table1. Sequence Database

Id	Sequence
1	adc
2	dab
3	ac
4	cdab
5	adb

Table1 shows a batch of sequenced data obtained from data stream environment. Considering the Min_sup = 2, sequence <ac>, as example, will be frequent because it appears twice.

3. Proposed Algorithm

The idea is to divide the active coming streams into windows or batches. Where window size W is user defined value, for each element in the window we check the support and build the sequence tree only for elements with support larger than minimum support. After each specified number of windows P (prune time) we prune the tree in order to get rid of branches with low support.

The idea behind building the sequence tree is based on the permutation for the sequence. Given any sequence S = {a, b, c, d} then all possible sub sequences are actually the permutation for that sequence {<a>, <b>, <c>, <d>, <ab>, <ac>, <ad>, <bc>, <bd>, <cd>, <abc>, <acd>, <bcd>, <abcd>}. Starting from the root node, permutation set of sequences will be inserted to the tree and the count will be incremented only for the latest item in each sequence. Last item in each sequence will indicate to a new possible path in the tree.

```
DSSPM Algorithm
Input: Set of streams windows W = {w_1, w_2... w_n}, Min_sup, prune time P.
Output: Tree of frequent patterns with support > Min_sup.

Initialize the root node, root = {NULL}
Initialize the supported items list, Sup_list = {}

while(true) /* streams always active so we need to run the algorithm forever */
    foreach item i ∈ w_n do
        calculate support(i)
        if support(i) > Min_sup then
            add i to Sup_list
        end if
    end for
    foreach sequence s ∈ w_n do
        call pruneSequence(s, Sup_list) /* remove items with support less than Min_sup */
        calculate permutation(s)
        foreach sequence ps ∈ permutaion_list do
            add ps to the tree, starting from the root node
              increment the count only for the latest item in the sequence
        end for
    end for
    if n = P /* if number of processed windows = prune time */
        call pruneTree() /* remove sequences with support less than Min_sup */
    end if
end while
```

Lemma 1: In the above algorithm, the item I from a sequence S can be considered as a frequent item only and only if support(I) > Min_sup.

Proof: Before the sequence batch entered to the algorithm, a process of counting the repetition of each element in the batch is made and this grantee that only elements with count greater than the minimum support are considered.

Lemma 2: For the sequence S, if length(S) > number of tree levels; then S is not a frequent sequence.

Proof: Suppose we have a sequence S = {i_1, i_2... i_n} with n items and the number of tree levels is m where $m < n$. Since length of the sequence is n so we need n levels in the tree to store the complete sequence but we have m levels and m is less than n. So sequences S will not entirely fit in the tree, hence it is not a frequent sequence.

Using Lemma 1 and Lemma 2 we can see that the algorithm will prune the Items with support < Min_sup. Some of the frequent patterns will be dropped from the final list of the frequent sequences due to the fact that the proposed algorithm will prune the sequences that are not frequent after ending the preset prune time (p).

Complexity

In the worst case, the time complexity to process one window using DSSPM will be the complexity to calculate the sequence permutation $O(m^2)$; where m is the number of items in the sequence, in addition to the complexity when building the sequence tree for that window O(n.m); where n is the number of sequences in the window. The total time complexity will be O(n^2), and this suit data streams environments such as web clicks and wireless sensors network.

Structure of Sequence Tree

Sequence tree is a data storage structure; it is used to store the frequent patterns with their support values. The root node of the tree is an empty node and just used as a link to the frequent sequences. Each of the tree nodes has two attributes; one stores the item and the other stores the support of the item inside the sequence. Any path from the root node to the leaf node is a frequent pattern and it is support is the support of the leaf node. The support of any parent node is equal or larger than support of it is children.

Suppose the following two windows, as shown in table 2, each of five sequences are coming to the analysis node in data stream environment. The goal is to find sequence patterns with Min_sup = 3. Consider the prune time is 2 and we have the following items set I = {a, b, c, d, e}.

Table 2. Two input windows

W1		W2	
Id	Sequence	Id	Sequence
1	bad	1	aceb
2	abec	2	cae
3	aec	3	ecad
4	bda	4	ade
5	abde	5	ae

First we should find the support for each element in W1, {a = 5, b = 4, c = 2, d = 3, e = 3}. Since support (c) < Min_sup, we will ignore item c when we build the sequence tree. After processing W1, we will have the tree shown in Figure 2.

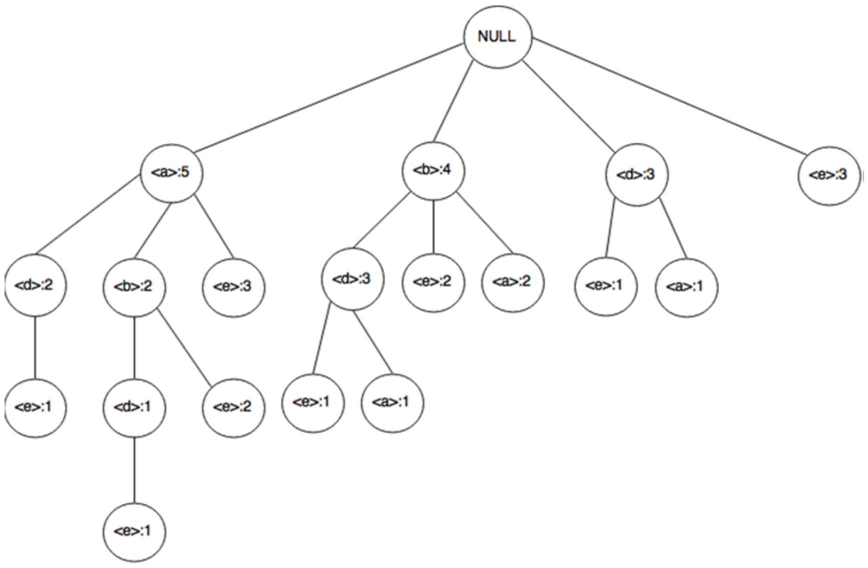


Figure 2. Sequence tree after processing W1

Now we find the support for elements in W2, {a = 5, b = 1, c = 3, d = 2, e = 5}. Since support (b) < Min_sup and support (d) < Min_sup, we will ignore both items b and d. Sequence tree after processing W2 will be as shown in Figure 3.

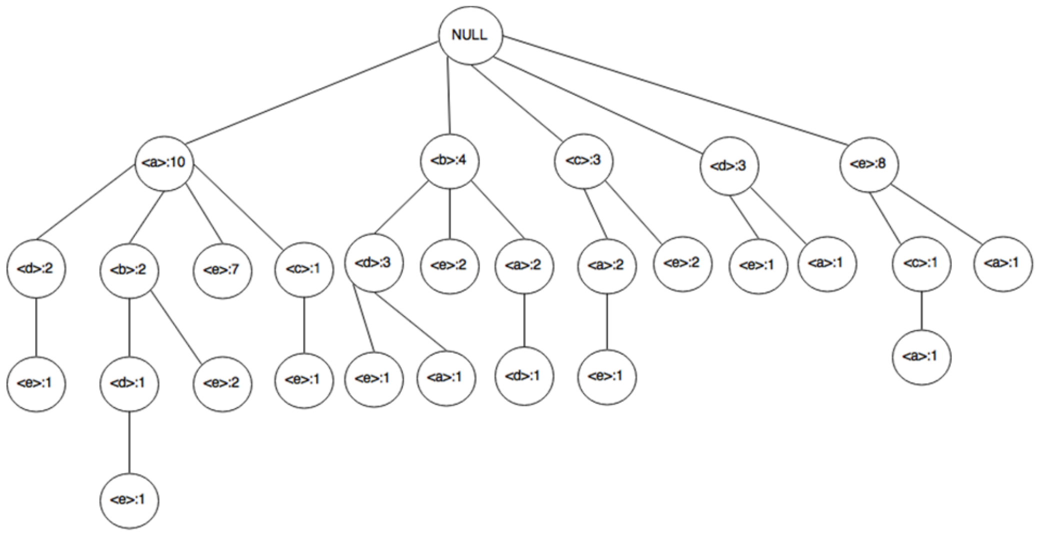


Figure 3. Sequence tree after processing W2

Now we have to prune the tree; we will remove all paths with support less than Min_sup. The final tree will be as shown in Figure 4.

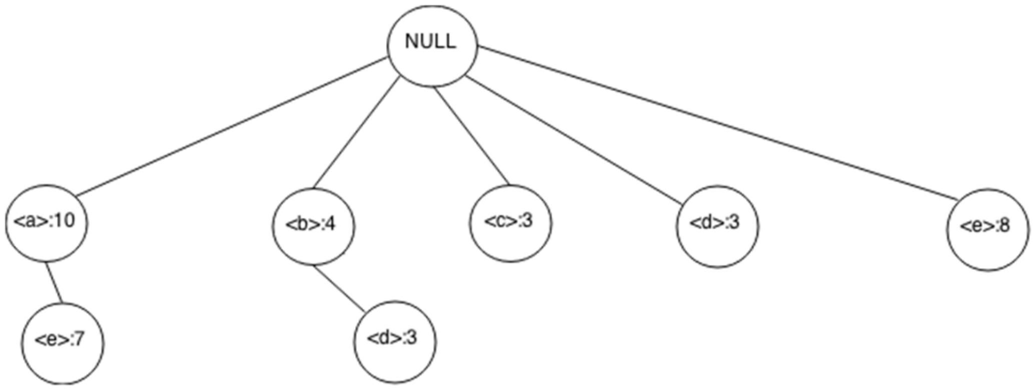


Figure 4. Sequence tree after pruning

Traverse the tree we can get the sequence patterns with support > Min_sup = {<a>:10, <b>:4, <c>:3, <d>:3, <e>:8, <ae>:7, <bd>:3}.

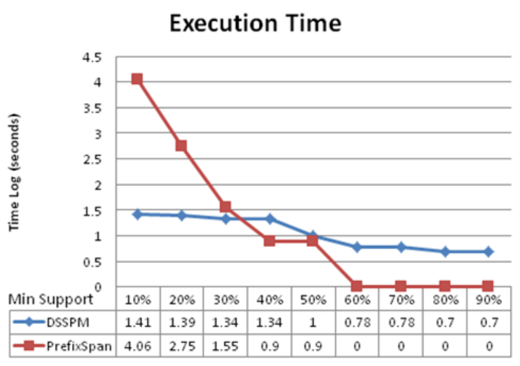
Prune time and Min_sup values can be tuned according to the processing node memory limitation; decrease Min_sup and increase prune time will give an accurate result and will increase the memory required to build the tree.

We are using greedy algorithm for finding the frequent patterns; this will not give an optimal answers, which is normal for this class of algorithms. We are removing elements from sequences with support < Min_sup. Also we are pruning the sequences with support < Min_sup after prune time P.

4. Experimental Results and Performance Analysis

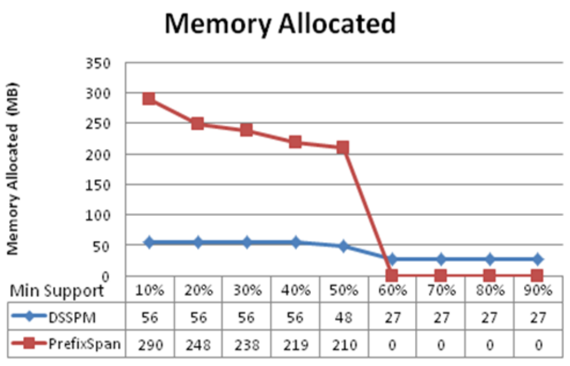
The algorithm is implemented in Java using PC running windows XP with dual core processor (2.8 GHz) and 4GB of memory. We evaluated our proposed algorithm with synthetic data generated by sequence pattern mining framework SPMF (Fournier et al., 2015), each sequence has 17 unique transactions and each transaction is represented by a letter {a-q}. In order to show the efficiency of our proposed algorithm, we did a comparison with PrefixSpan algorithm (Pei et al., 2013). We report in Figure 5 the time needed to find all frequent patterns in 100,000 sequences with different values of minimum support, each window contains 100 sequences and the pruning time is 100 windows. AS shown in Figure 5 (a) the execution time for PrefixSpan grows as the minimum support decrease. Clearly, in Figure 5 (a) the execution time for DSSPM increases slightly while decreasing the minimum support. Moreover DSSPM outperforms PrefixSpan in execution time for all minimum support values.

Figure 5 (b) shows the average memory allocated for both algorithms with different values of minimum support. With low minimum support values, PrefixSpan needs more memory than DSSPM. The value of memory allocated is decrease while increasing the minimum support.



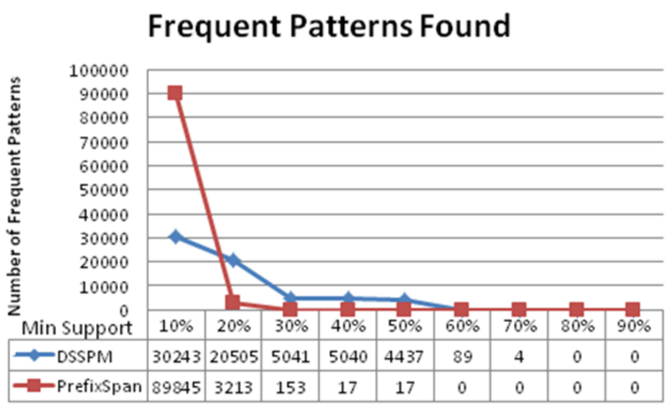
Min Support	10%	20%	30%	40%	50%	60%	70%	80%	90%
DSSPM	1.41	1.39	1.34	1.34	1	0.78	0.78	0.7	0.7
PrefixSpan	4.06	2.75	1.55	0.9	0.9	0	0	0	0

(a)



Min Support	10%	20%	30%	40%	50%	60%	70%	80%	90%
DSSPM	56	56	56	56	48	27	27	27	27
PrefixSpan	290	248	238	219	210	0	0	0	0

(b)



Min Support	10%	20%	30%	40%	50%	60%	70%	80%	90%
DSSPM	30243	20505	5041	5040	4437	89	4	0	0
PrefixSpan	89845	3213	153	17	17	0	0	0	0

(c)

Figure 5. Comparison between DSSPM and PrefixSpan

Table 3. Most frequent sequences with minimum support of 10%

Sequence	DSSPM Support	PrefixSpan Support
adhji	154	17604
iebfa	152	17557
ebfgh	148	17676
bfegi	147	17668
gfaej	153	17709
fhgbc	152	17677
cadhj	171	17622
dhjib	151	17763
hjieb	144	17689
jiebf	154	17642

In Figure 5 (c) we compared both algorithms in terms of number of frequent patterns found, it is clearly that with low minimum support PrefixSpan is more effective while with high support values DSSPM become more effective and can find more frequent patterns.

In order to show the effectiveness of our proposed algorithm, we compared the top 10 most frequent sequences found when minimum support was set to 10%. The results are shown in Table 3.

DSSPM is a greedy algorithm; it looks only on the current window pruning the infrequent sequence patterns and this explains the low support values compared with PrefixSpan, but it still can find the frequent sequences which make it suited for data streams environments.

5. Conclusion

A new algorithm based on sequence tree was introduced for sequence pattern mining in data streams environment. The depth of the tree will be equal to the max length of supported sequences and this makes it suitable for data streams environment where there is a limitation in the memory. Algorithm inputs Min_sup and prune time are user defined variables which can be tuned according to the environment and the users need. However if the processing node has enough memory and a good processing power, user can decrease the Min_sup or increase the prune time to get more precise results; this improves the greedy behavior of this algorithm.

We showed experimentally that the proposed algorithm is more efficient than the PrefixSpan algorithm for patterns with any support less than 30% for CPU time and with any support less than 60% for memory usage. While the proposed algorithm detects less frequent sequence patterns than the PrefixSpan algorithm, it still produces the top frequent patterns.

References

Agrawal, R., & Ramakrishnan S., (1998). System and method for mining sequential patterns in a large database. U.S. Patent No. 5819266.

Fournier-Viger, P., Gomariz, A., Soltani, A., Lam, H., & Gueniche, T., (2015). Spmf: Open-source data mining platform. Retrieved from http://philippe-fournier-viger.com/spmf

Huang, G., Na, Z., & Ren, J. D. (2011). Mining Web Frequent Multi-dimensional Sequential Patterns. *Information Technology Journal, 10*, 2434-2439. http://dx.doi.org/10.3923/itj.2011.2434.2439

Lee, Victor E., Ruoming Jin, and Gagan Agrawal, 2014. Frequent Pattern Mining in Data Streams. Frequent Pattern Mining. Springer International Publishing, 199-224. ISBN 13:9783319078212, http://dx.doi.org/10.1007/978-3-319-07821-2_9

Liu, J. X., Yan, S. T., & Ren, J. D. (2012). A Fast Incremental Mining Algorithm of Sequential Patterns Based on Sequence Tree. *International Conference on Mechanical Engineering and Automation Advances in Biomedical Engineering, 10*. ISSN 978-1-61275-028-6/10

Marascu, A., & Florent. M. (2005). Mining sequential patterns from temporal streaming data. Proc. First ECML/PKDD Workshop Mining Spatio-Temporal Data (MSTD'05). http://dx.doi.org/10.1.1.112.7655

Muthukrishnan, S. V. (2005). Data streams: Algorithms and applications. Now Publishers Inc. http://dx.doi.org/10.1561/0400000002

Pei, J., Han, J. W., Behzad, M. A., Helen, P., & Chen, Q. M. (2013). Umeshwar Dayal, and Mei-Chun Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In 2013 IEEE 29th International Conference on Data Engineering (ICDE), pp. 0215-0215. IEEE *Computer Society, 2001*. http://dx.doi.org/10.1109/icde.2001.914830

Rao, V., Chandra, S., & Sammulal, P. (2013). Survey on Sequential Pattern Mining Algorithms. International Journal of computer application (0975-8887) 76.12. http://dx.doi.org/ 10.5120/13301-0782

Tiwari, A., Gupta, R. K., & Agrawal, D. P. (2010). A Survey on Frequent Pattern Mining: Current Status and Challenging Issues. *Information Technology Journal, 9*, 1278-1293. http://dx.doi.org/10.3923/itj.2010.1278.1293

Vijayarani, S., & P. Sathya, 2012. A Survey on Frequent Pattern Mining Over Data Streams. *International Journal of Computer Science and Information Technology & Security 2.5* (2012), 1046-1050. ISSN 2249-9555.