Pair Triplet Association Rule Generation in Streams

[1]Manisha Thool, [2]Preeti Voditel

[1] Ramdeobaba College of Engineering and Management,
Nagpur, Maharashtra, India

[2] Ramdeobaba College of Engineering and Management
Nagpur, Maharashtra, India

Abstract

Many applications involve the generation and analysis of a new kind of data, called stream data, where data flows in and out of an observation platform or window dynamically. Such data streams have the unique features such as huge or possibly infinite volume, dynamically changing, flowing in or out in a fixed order, allowing only one or a small number of scans. An important problem in data stream mining is that of finding frequent items in the stream. This problem finds application across several domains such as financial systems, web traffic monitoring, internet advertising, retail and e-business. This raises new issues that need to be considered when developing association rule mining technique for stream data. The Space-Saving algorithm reports both frequent and top-k elements with tight guarantees on errors. We also develop the notion of association rules in streams of elements. The Streaming-Rules algorithm is integrated with Space-Saving algorithm to report 1-1 association rules with tight guarantees on errors, using minimal space, and limited processing per element and we are using Apriori algorithm for static datasets and generation of association rules and implement Streaming-Rules algorithm for pair, triplet association rules. We compare the top- rules of static datasets with output of stream datasets and find percentage of error.

Keywords: *Association rule mining, Space-Saving algorithm, Streaming-rules algorithm.*

.

1. Introduction

Data Stream Mining is the process of extracting knowledge structures from continuous, rapid data records. A data stream is ordered sequences of instances that arrive at a rate that does not permit to permanently store them in memory. Data streams are unbounded in size making them impossible to process by most data mining approach. This is because most of them require scans of data to extract the information which is unrealistic for stream data. The characteristics of data stream mining are as follows. It is impossible to store all the data from the data stream. The data stream can change over time and also need to consider the problem of resource allocation in mining data streams due to the large volume and the high speed of streaming data. [1] Data streams can be viewed as a sequence of relational tuples (e.g. call records, web page visits, sensor readings) that arrive continuously at time varying. Due to their speed and size it is impossible to store them permanently.

Many applications involve the generation and analysis of a new kind of data, called stream data, where data flow in and out of an observation platform or window dynamically. Such data streams have the unique features such as huge or possibly infinite volume, dynamically changing, flowing in or out in a fixed order, allowing only one or a small number of scans. As the number of applications on mining data streams grows rapidly, there is an increasing need to perform association rule mining on stream data. For most data stream applications, there are needs for mining frequent patterns and association rules from data streams. An important problem in data stream mining is that of finding frequent items in the stream. This problem finds application across several domains such as financial systems, web traffic monitoring, internet advertising, retail and e-business. [2]

Algorithm for frequent items mining in data streams are generally two techniques: counter-based technique, and sketch-based technique. They are frequent items mining algorithm. Counter based algorithm maintain a summary of the items. The summary consists of a small sunset of the items with associated counters approximating the frequency of the item in the stream. Counter-based algorithm maintains counters for and monitors a fixed number of elements of the stream. If an item arrives in the stream that is monitored, the associated counter is incremented; else the algorithm decides whether to discard the item or reassign an existing counter to this item. They maintain a summary of the items .The summary consists of a small subset of the items with associated counters approximating the frequency of the item in the stream. The counter-based algorithms include Sticky Sampling and Frequent (Freq), Lossy Counting (LC), and Space-Saving (SS).

The sketch-based technique work by hashing the items to a small sketch of the data stream i.e. to maintain a sketch

of the data stream using hashing and updating a corresponding counter. The frequency of the individual items can be estimated by reading a counter in the sketch. Sketch-based techniques maintain approximate frequency counts of all elements in the stream. The Sketch algorithm solve frequency estimation problem, and so need additional data information to solve frequent items problem. In Sketch-based techniques the algorithms include CountSketch (CCFC), GroupTest (CGT), and CountMin-Sketch (CM). The Space-Saving algorithm reports both frequent and top-k elements with tight guarantees on errors. We also develop the notion of association rules in streams of elements.

The Streaming-Rules algorithm is integrated with Space-Saving algorithm to report 1-1 association rules with tight guarantees on errors, using minimal space, and limited processing per element and we are using Apriori algorithm for static datasets and generation of association rules and implement Streaming-Rules algorithm for pair, triplet association rules. We compare the top- rules of static datasets with output of stream datasets and find percentage of error.

In rest of the paper is organized as follows. Section II highlights the related work. In Section III, we introduce proposed work of Space-Saving algorithm, and its associated data structure. The building blocks of Streaming algorithm are explained.

In this paper, we propose an integrated online streaming algorithm for solving both problems of finding the top-k elements, and finding frequent elements in a data stream. Our Space-Saving algorithm reports both frequent and top-k elements with tight guarantees on errors. For general data distribution, Space-Saving answers top-k queries by returning k elements with roughly the highest frequencies in the stream and it use limited space for calculating frequent elements. In this paper, we develop the notion of association rules in streams of elements. The Streaming-Rules algorithm is developed to report association rules with tight guarantees on errors, using minimal space, and limited processing per element and then we compare the top-k rules of static datasets with output of stream datasets.

In rest of the paper is organized as follows. Section 2 highlights the related work. We introduced the Apriori algorithm and Association Rule and techniques in Data Stream mining. We introduce proposed work in Section 3 Space-Saving algorithm, and its associated data structure. And the building blocks of streaming algorithm.

2. Background and Related Work

In this section we provide the background information about Datasets, Apriori algorithm and Association Rule mining.

2.1 Datasets

The synthetic and real-life data sets are available from the Frequent Itemset Mining Dataset Repository at http://fimi.cs.helsinki.fi/data/ [3].

2.2 Apriori Algorithm

Apriori algorithm proposed by R. Agrwal and R. Srikant in 1994 [3] for mining frequent item sets for Boolean association rules. The name of the algorithm is based on the fact that the algorithm uses prior knowledge of frequent itemset properties. Frequent item sets generation and the creation of strong association rule from the frequent item sets pattern are two basic steps in association rule mining.

```
L1 = {large 1-itemsets}
for (k=2; Lk-1≠∅; k++) do begin
        Ck = apriori-gen(Lk-1); // New candidates
        for all transactions t ∈ D do begin
                C't = subset (Ck, t) // Candidates
contained in t
                For all candidates c ∈ Ct do
c.count++
        end
        Lk = {c ∈ Ct | c.count ≥ minsup}
end
Return ∪k Lk
```

Pseudo code of Apriori Algorithm

According to [7] it first scans the database D and calculates the support of each single item in every record I in D, and denotes it as C_I. Out of the itemsets in C_I, the algorithm computes the set L_I containing the frequent 1-itemsets. In the k^{th} scan of the database, it generates all the new itemset candidates using the set L_{k-1} of frequent (k-1) itemsets discovered in the previous scanning and denotes it as C_k. And the itemsets whose support is greater than the minimum support threshold are kept L_k.

2.3 Generating Association rules from Frequent Item sets

According to [8] the following two steps are required to augment the association rule generation.

i) For every frequent itemset "I" , all non-empty subsets of "I" is required to be generated.

ii) For all non-empty subsets of I, if support_count (I) / support_count(s) >= min_conf (min_conf=minimum confidence threshold) then output the rule as "s →(l-s)". According to [12] Apriori Algorithm is the algorithm to extract association rules from datasets. Apriori Algorithm is not an efficient algorithm as it in a time consuming algorithm in case of large datasets. With the time a number of changes proposed in Apriori to enhance the performance in term of time and number of database passes.

2.4 Associations Rules

In data mining, with the increasing amount of data stored in real application system, the discovery of association rule attracts more and more attention. Mining for association rules can help in business, and decision making. [3]

Association rule techniques are used for data mining if the goal is to detect relationship or association between specific values of categorical variables in large data sets. There may be thousands or millions of records that have to be read and to extract the rules for, but in the past user would repeat the whole procedure, which is time – consuming in addition to its lack of efficiency for new data, or there is a need to modify or delete some or all the existing set of data during the process of data mining. Mining association rules is particularly useful for discovering relationship among items from large databases [4]. A standard association rule is a rule of the form X → Y which says that if X is true of an instance in a database, so is Y true of the same instance, with a certain level of significance as measured by two indicators, support and confidence. The goal of standard association rule mining is to output all rules whose support and confidence are respectively above some given support and coverage thresholds. These rules encapsulate the relational associations between selected attributes in the database, for instance, computer →antivirus software: 0.02 support, 0.70 coverage denotes that in the database, 70% of the people who buy computer also buy antivirus software, and these buyers constitute 2% of the database. The mining process of association rules can be divided into two steps.

1. Frequent Item sets Generation: generate all sets of items that have support greater than a certain threshold, called minsupport.

2. Association Rule Generation: from frequent itemsets, generate all association rule that have confidence greater than a certain threshold called minconfidence [11].

2.5 Counter-Based Algorithms

2.5.1 Freq

According to [12] the Frequent algorithm keeps count of k=$1/\emptyset$ number of items. This is based on the observation that there can be at the most $1/\emptyset$ items having frequency more than $\emptyset N$. Freq keeps count of each incoming item by assigning a unique counter for each item, until all the available counters are occupied. The algorithm then decrement all counters by 1 until one of the counters becomes zero. It then uses that counter for newest item. This step deletes all the non-frequent item counters.

2.5.2 LC

The Lossy Counting algorithm was proposed by Manku and Motwani in 2002 [5] in addition to a randomized sampling-based algorithm and technique for extracting from frequent items to frequent itemsets. The algorithm maintains a data structure D, which is a set of entries of the form (e, f, Δ), where e is an element in the stream, f is an integer representing the estimated frequency and Δ is the maximum possible error in f. LC conceptually divides the incoming stream into buckets of width w= $1/\varepsilon$ transactions each. If an item arrives that already exists in D, the corresponding f is incrementing, and else a new entry is created. D is pruned by deleting some of the entries at the bucket boundaries. The space requirement is O ($\frac{1}{\varepsilon} Log\varepsilon N$) and time cost O (1).

2.5.3 Space-Saving Algorithm

According to [10] the deterministic Space-Saving algorithm uses a data structure called Stream-Summary. For each corresponding monitor the frequent items the Stream-Summary data structure consist of a linked list of a fixed number of counters. All counters with the same count are associated with a bucket which stores the count. Buckets are created and destroyed as new items arrive. They are stored as an always sorted doubly linked list. Each counters also stores the estimated error in the frequency count of the corresponding item, which is used later to provide guarantees about the accuracy of the frequency estimate returned by and error returned by the algorithm. The space requirement is O ($\frac{1}{\varepsilon}$) and the counts of all stored items solve frequency estimation problem with error n/k.

2.6 Sketch-Based Algorithm

2.6.1 CGT

According to [9] the Combinational Group Testing algorithm is based on a combination of group testing and error correcting codes. Each item is assigned to groups using a family of hash functions. Within each group there is a group counter which indicates how many items are present in the group and a set of log M counters with M being the largest item in the dataset. The group counters and the counters which correspond to the bits 1 in the binary representation of the item are updated accordingly. The space complexity is O ($\frac{1}{\emptyset}\log\frac{1}{\emptyset\delta}\log M$).

2.6.2 Count Sketch

According to [6] CountSketch is an array of t hash tables each containing b buckets. There are two sets of hash functions are used one (h_1……..h_t) hashes items to buckets, and second set is (s_1……s_t) hashes items to the set {+1, -1}. Randomness of O ($t \log M$) required for implementation of these independent hash function. When an item arrives, the t buckets corresponding to that item are identified using first set, and in second set updated by adding +1 or -1.
Space complexity is O ($\frac{1}{\varepsilon^2}\log\frac{N}{\delta}$) and time is O ($\frac{N}{\delta}$).

2.6.3 Count Min Sketch

CountMin Sketch proposed by Cormode and Muthukrishnan [8] described similar to CountSketch. The algorithm maintains an array of $d \times w$ counters. When an item i arrives, one counter in each row is incremented, the counter is determined by the hash functions. The estimated frequency for any item is the minimum of the values of its associated counters. For each new item its estimated frequency is calculated, and if it is greater than the required threshold, it is added to a heap. At the end, all items whose estimated count is still above the threshold are output.
The Space complexity is O ($\frac{1}{\varepsilon}\log\frac{1}{\delta}$).

2.7 Why We Use Counter-Based Algorithm

We used Counter-based algorithm because we are interested to solve the only the frequent elements problem whereas Sketch-based algorithm act as general data stream summaries, and can be used for other types of approximate statistical analysis of the data stream ,apart from being used to find the frequent items. Thus our application was strictly limited to discovering frequent items, counter-based algorithm would be preferable. With the observation [13] sketch-based algorithm require more space than counter-based algorithm. We compare with other algorithm Space-Saving algorithm required less space i.e., O ($\frac{1}{\varepsilon}$) so we implemented Space-Saving Counter-based algorithm which only solve frequent item problem with minimizing space.

3. Proposed Work

3.1 Space-Saving Algorithm

In this we briefly describe the *Space-Saving* algorithm. The algorithm proposed in [10] our counter-based *Space-Saving* algorithm and its associated *Stream-Summary* data structure.

The deterministic Space-Saving algorithm uses a data structure called Stream-Summary. For each corresponding monitor the frequent items the Stream-Summary data structure consist of a linked list of a fixed number of counters.

Algorithm: Space-Saving (m counters, stream S)

begin

```
for each element, e, in S{
   If e is monitored{
      let count_i be the counter of e
       Increment-Counter (Count_i);
    }
else{
    // The replacement step
      let e_m be the element with least hits, min
      Replace e_m with e;
      Increment-Counter (Count_i);
     Assign ε_m the value min;
    }
  }// end for
End;
```

Algorithm: The Space-Saving algorithm

The Increment-Counter algorithm

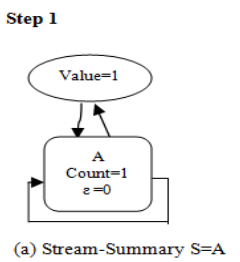
To implement Space-Saving algorithm we need a *Stream-Summary* data structure. [10]

Algorithm: Increment-counter (counter $Count_i$)
begin
let Bucket $_i$ be the Bucket of count$_i$
let Bucket $_i^+$ be the Bucket $_i$'s neighbor of larger value
Detach count$_i$ from Bucket $_i$'s child-list;
count$_i$ ++;
//Finding the right bucket for count$_i$
If (Bucket $_i^+$ does not exist **AND** count$_i$= Bucket $_i^+$)
Attach count$_i$ to Bucket $_i^+$'s child-list ;
else{
// A new bucket has to be created
Create a new Bucket Bucket $_{new;}$
Assign Bucket $_{new}$ the value of count$_i$
Attach count$_i$ to Bucket $_{new}$'s child-list
Insert Bucket $_{new}$ after Bucket $_i$
}
//Cleaning up
If Bucket $_i$'s child-list is empty{
Detach Bucket $_i$ from the Stream-Summary;
Delete Bucket $_i$;
}
End;

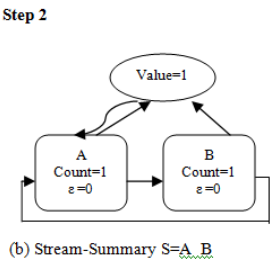
Algorithm: The Increment-Counter algorithm

In Stream-Summary, all elements with the same counter value are linked together in a linked list. They all point to a parent bucket. The value of the parent bucket is the same as the counters value of all its elements. Every bucket points exactly one element among its child list, and buckets are kept in a doubly linked list, sorted by their values. Initially, all counters are empty, and are attached to a single parent bucket with value 0. Stream-Summary can be sequentially traversed as a sorted list, since the buckets list is sorted.

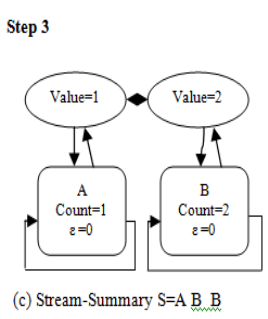
Example: Assuming m=2 and Stream is A B B C. In step 1 the stream S =A, the stream-summary in step (a). For Stream-Summary S=A B, the bucket shown in step 2.When another B arrives, a new bucket is created with value=2, and B get attached to it in step 3.When C arrives, the element with the minimum counter, A is replaced by C. C has error will be 1.The final stream summary is shown in step 4.



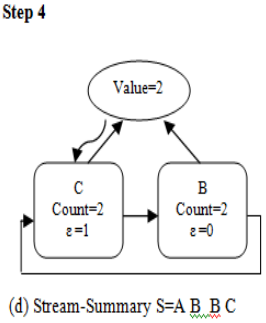
(a) Stream-Summary S=A



(b) Stream-Summary S=A B



(c) Stream-Summary S=A B B



(d) Stream-Summary S=A B B C

Fig.1 Example of Space-Saving Algorithm with Stream-Summary

3.2 The Streaming-Rules Algorithm

The algorithm proposed in [14], given a stream $q_1,q_2 \ldots q_I \ldots q_N$, and maxspan is δ.

The algorithm maintains a *Stream-Summary* data structure for m elements. For each element e_i, of these m counters, the algorithm maintains a consequent *Stream-Summary*e_i *data* structure of n elements.

The j^{th} element in *Stream-Summary*e_i will be denoted e_i^j, and will be monitored by counter *Count* (e_i, e_j), whose error bound will be $\varepsilon(e_i, e_j)$. Each element, q_I, in the current window has a consequent set s_I. In addition, the last observed element has an antecedent set t_I.

For each element, q_I, in the data stream, if there is a counter, *Count* (e_i), assigned to q_I ,i.e., $e_i = q_I$,increment *Count* (e_i). Otherwise, replace e_m , the element that currently has the least estimated hits, *min*, with q_I; assign *Count*(q_I) the value *min +1;* set $\varepsilon(q_I)$ to min ;re- initialize *Stream-Summary*.

For association rule we are using Nested Data Structure i.e. the antecedent data structure and consequent data structure. Delete the consequent set, $s_{I-\delta-1}$, of the expired element, $q_{I-\delta-1}$ .Assign an empty consequent s_I set to q_I . Delete the antecedent set t_{I-1} and create an empty antecedent set t_I for q_I. Scan the current window $q_{I-\delta}$ to q_{I-1}. For each scanned element q_J, the algorithm checks if q_I has been inserted into s_J, and whether q_J has been inserted into t_I. If both condition do not hold, insert q_I into s_J ;and q_J into t_I.

If $\mathbf{q_J}$ is monitored, say at $\mathbf{e_j}$, i.e., Stream-Summarye_j is Stream-Summaryq_I, and then insert $\mathbf{q_I}$ into Stream-Summarye_j as follows. If there is a counter, **Count** $(\mathbf{e_j}, \mathbf{q_I})$,assigned to $\mathbf{q_I}$ in Stream-Summarye_j, increment it .If **Count** $(\mathbf{e_j}, \mathbf{q_I})$does not exist ,let $\mathbf{e_j^n}$ be the element with currently the least estimated hits, $\mathbf{min_j}$ in Stream-

Summary$_j$ .Replace $\mathbf{e_j^n}$ with $\mathbf{q_I}$;set **Count** ($\mathbf{e_j}$, $\mathbf{q_I}$)to $\mathbf{min_j}$ **+1**and set $\boldsymbol{\varepsilon}$ ($\mathbf{e_j}$, $\mathbf{q_I}$)to $\mathbf{min_j}$.

If q_I has been inserted into s_J, or q_J has been inserted into t_I, or q_J is not monitored in Stream-Summary, the algorithm skips to q_{J+1}. Streaming-Rules is sketched in Figure 4.

Algorithm streaming-Rules (nested Stream-Summary (m,n))
begin
For each element,q_I, in the stream S {
If q_I is monitored {
Let Count (ei) be the counter of q_I
Count (ei) ++;
} else {
//The replacement step
Let e_m be the element with least hits, min
Replace e_m with q_I
Assign ε (q_I)the value min;
Count (e_m) ++;
Re-initialize Stream-Summaryq_I;
};
Delete $s_{I-\delta-1}$ of the expired element, $q_{I-\delta-1;}$
Create an empty set s_I for $q_{I;}$
Delete the set $t_{I-1;}$
Create an empty set t_I for $q_{I;}$
For each element ,q_J in the stream S, where (I-δ)≤J<I{
If q_J is monitored **AND** $q_I \not\in s_J$ **AND** $q_J \not\in t_I$ {
If q_J contains more than one element {
If each element of $q_J \in t_I$ **AND** $q_I \notin s_J$ **AND** $q_J \notin t_I$ {
Insert q_I into s_J;
Insert q_J into $t_{I;}$
//The association counting step
Let q_J be monitored at e_j
If q_I is monitored in Stream-Summarye_j {
Let Count (e_j, q_I) be the counter of
Count (e_j, q_I) ++;
} } }else {
//The nested replacement step
Let e_j^n be the element with least hits,min_j
Replace e_j^n with q_I;
Assign ε (e_j, q_I) the value min_j ;
Count (e_j, q_I) ++;
}
Insert [(e_j, q_I)] at (i+1)
}
} //end for
}//end for
End:

Algorithm: The Streaming-Rules algorithm

3. 3 Find-Forward Algorithm

Find-Forward [14] scans Stream-Summary in order of estimated frequencies, starting by the most frequent element e1, until it reaches an element that does not satisfy minsup.

Algorithm: Find-Forward (Stream-Summary (m, n))
begin
Integer
i = 1;
While (Count (ei) > [φN] AND
i ≤ m) {
Integer j = 1;
while (Count+ (ei , ej) > [ψ(Count(ei) − ε(ei))] AND
j ≤ n){
output ei → ej ;
j++;
}// end while
i++;
}//end
while
end;

Algorithm: The Find-Forward algorithm

For each scanned element ei, Find-Forward scans its Stream-Summaryei, in order of estimated frequencies, starting by the most frequent, e1, until it reaches an element that does not satisfy min-conf, and outputs all the element that satisfy minconf.

Hence, to guarantee that Find-Forward always approximate by over-estimation only, it reports the estimated count of association x → y as Count(x, y)+ ε(x), and we denote it Count+ (x, y). Any element y, whose Count+ (x, y) satisfies ψ(Count(ei) − ε(ei)) should be reported as an association of the form x → y.

4. Experimental Results

We are using synthetic dataset T10I4D100K total transaction is 100000, http://fimi.cs.helsinki.fi/data/ is used to evaluate the performance of the proposed algorithm, where Test system lusing a Windows XP, experimental environment is Jdk 6.9.1 with support and confidence.

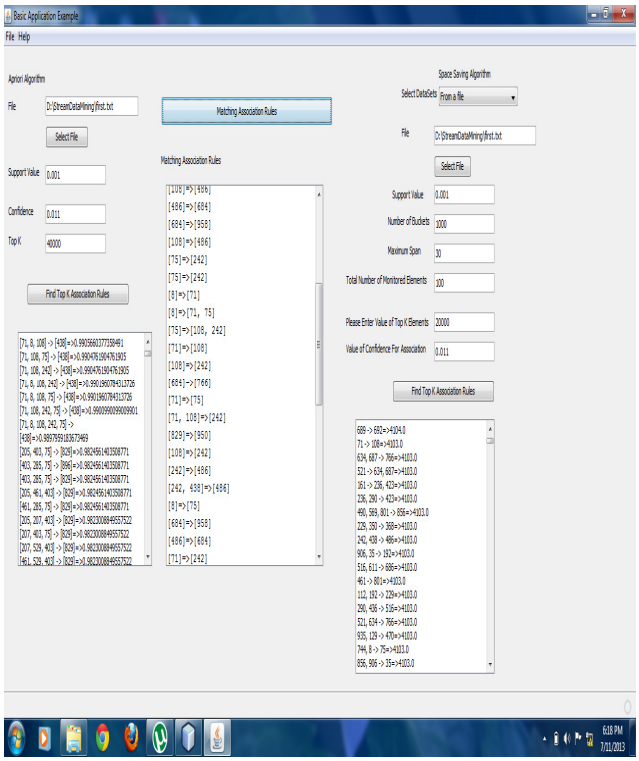
We are using 46461 transactions of T10I4D100K dataset with support= 1% and confidence=11%.

```
( [71, 8, 242, 438, 75] ) => ( [684, 486] ) = 94.84536082474226 %
( [71, 8, 108, 75] ) => ( [684, 486] ) = 95.09803921568627 %
( [71, 8, 108, 438, 75] ) => ( [684, 486] ) = 95.04950495049505 %
( [71, 8, 108, 242, 75] ) => ( [684, 486] ) = 94.89795918367348 %
( [71, 8, 108, 242, 438, 75] ) => ( [684, 486] ) = 94.84536082474226 %
( [438] ) => ( [958] ) = 5.511069241639189 5 %
( [438] ) => ( [766] ) = 9.138012246820537 %
( [438] ) => ( [958, 766] ) = 4.992934526613283 %
```

Fig.2 Association Rule of Apriori Algorithm

```
ei->ej : 332, 362, 469 --> 689
ei->ej : 362, 469, 689 --> 692
ei->ej : 689, 692, 991 --> 392
ei->ej : 692, 991, 392 --> 461
ei->ej : 991, 392, 461 --> 490
ei->ej : 461, 490, 569 --> 801
ei->ej : 490, 569, 801 --> 856
ei->ej : 569, 801, 856 --> 862
ei->ej : 856, 862, 906 --> 35
ei->ej : 862, 906, 35 --> 112
ei->ej : 906, 35, 112 --> 192
ei->ej : 192 --> 209
ei->ej : 192 --> 267
ei->ej : 192 --> 209, 267
ei->ej : 192 --> 296
ei->ej : 192 --> 209, 296
ei->ej : 192 --> 267, 296
ei->ej : 192 --> 209, 267, 296
```

Fig.3 Association Rule by Streaming-Rule Algorithm



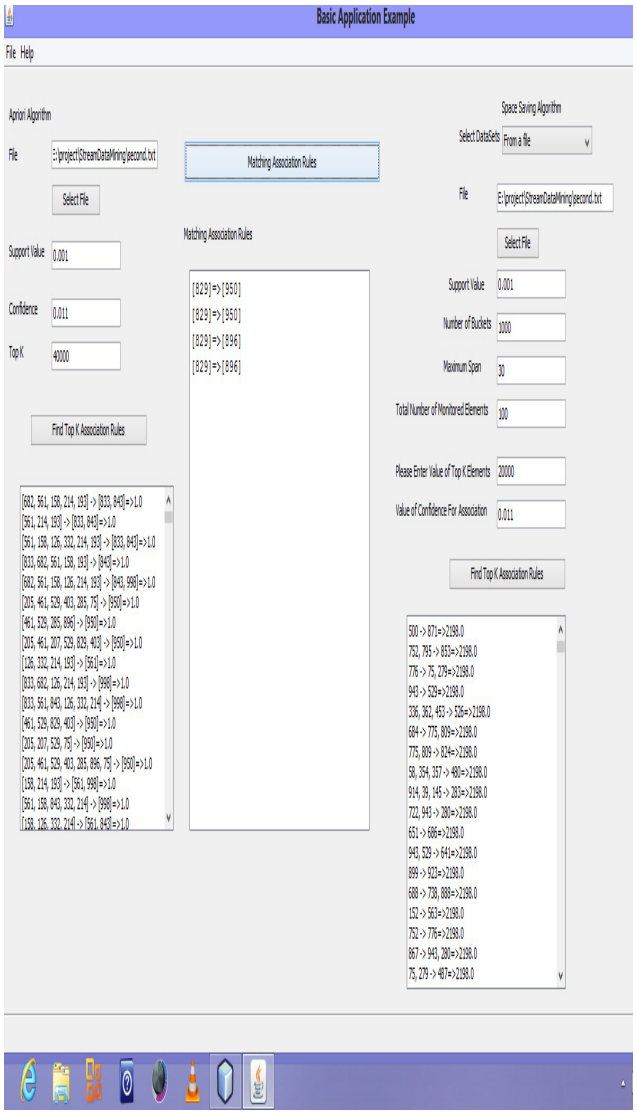
We are using 24862 transactions of T10I4D100K dataset with support= 1% and confidence=11%.

```
( [207, 896] ) => ( [950] ) = 97.55102040816327 %
( [207, 829] ) => ( [950] ) = 86.04651162790698 %
( [207, 829, 896] ) => ( [950] ) = 97.86324786324786 %
( [207, 529] ) => ( [950] ) = 84.13793103448276 %
( [207, 529, 896] ) => ( [950] ) = 97.86324786324786 %
( [207, 529, 829] ) => ( [950] ) = 97.00854700854701 %
( [207, 529, 829, 896] ) => ( [950] ) = 97.77777777777777 %
( [461, 207] ) => ( [950] ) = 94.90196078431372 %
( [461, 207, 896] ) => ( [950] ) = 97.86324786324786 %
( [461, 207, 829] ) => ( [950] ) = 97.88135593220339 %
( [461, 207, 829, 896] ) => ( [950] ) = 97.78761061946902 %
( [461, 207, 529] ) => ( [950] ) = 97.85407725321889 %
( [461, 207, 529, 896] ) => ( [950] ) = 97.77777777777777 %
```

Fig.4 Association Rule of Apriori Algorithm

```
ei->ej : 494 --> 678
ei->ej : 494 --> 606, 678
ei->ej : 494 --> 641
ei->ej : 494 --> 614, 641
ei->ej : 494 --> 614, 676
ei->ej : 494 --> 641, 676
ei->ej : 494 --> 623, 626
ei->ej : 494 --> 541
ei->ej : 494 --> 597
ei->ej : 494 --> 541, 597
ei->ej : 494 --> 541, 614
ei->ej : 722 --> 910
ei->ej : 722 --> 803
ei->ej : 722 --> 804
ei->ej : 722 --> 867
ei->ej : 722 --> 943
ei->ej : 722 --> 752
ei->ej : 722 --> 752, 804
```

Fig. 5 Association rule of Streaming-Rule algorithm



We are using 100000 transaction of T10I4D100K dataset with support= 1% and confidence=11%

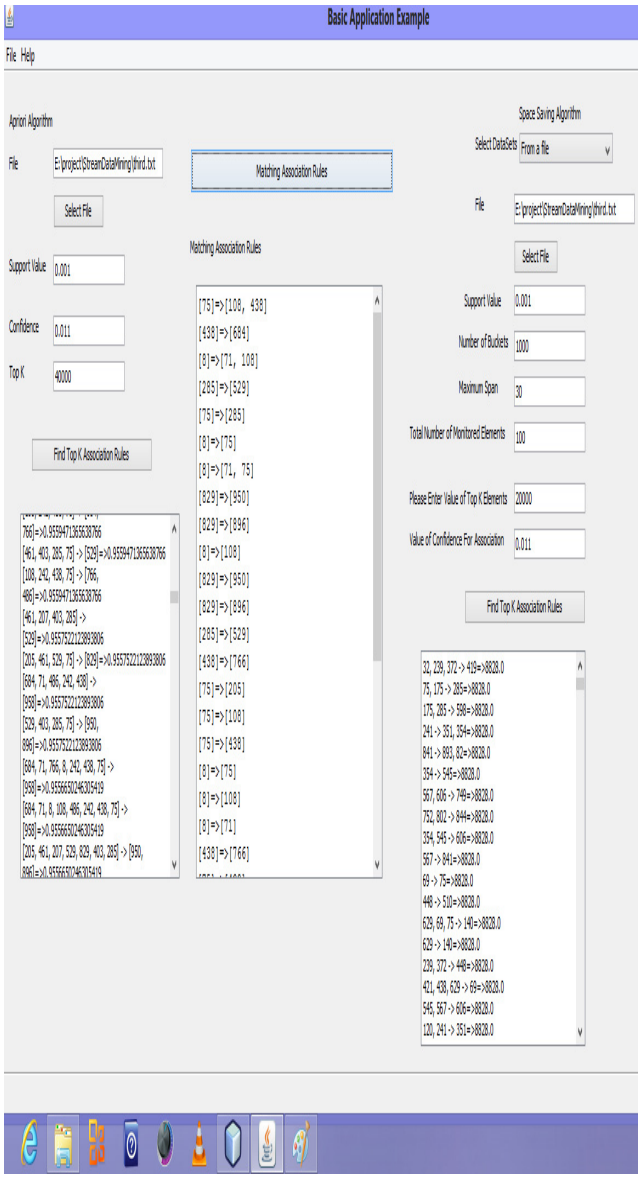
```
(  [8]  ) =>  (  [958,  766,  486,
75]  ) =  7.209612817089453  %
(  [8]  ) =>  (  [684,  486,  75]  ) =
7.743658210947933  %
(  [8]  ) =>  (  [958,  684,  486,
75]  ) =  7.476635514018691  %
(  [8]  ) =>  (  [684,  766,  486,
75]  ) =  7.476635514018691  %
(  [8]  ) =>  (  [958,  684,  766,
486,  75]  ) =  7.209612817089453  %

(  [8]  ) =>  (  [438,  75]  ) =
8.277703604806408  %
(  [8]  ) =>  (  [958,  438,  75]  ) =
7.476635514018691  %
(  [8]  ) =>  (  [766,  438,  75]  ) =
7.209612817089453  %
(  [8]  ) =>  (  [958,  766,  438,
75]  ) =  7.076101468624834  %
(  [8]  ) =>  (  [684,  438,  75]  ) =
8.144192256341789  %
(  [8]  ) =>  (  [958,  684,  438,
75]  ) =  7.343124165554073  %
(  [8]  ) =>  (  [684,  766,  438,
75]  ) =  7.209612817089453  %
(  [8]  ) =>  (  [958,  684,  766,
438,  75]  ) =  7.076101468624834  %
```

Fig.6 Association Rule of Apriori Algorithm

```
ei->ej  :  217  -->  472
ei->ej  :  217  -->  268,  472
ei->ej  :  217  -->  395,  472
ei->ej  :  120,  241,  268,  351  -->
354
ei->ej  :  285  -->  368
ei->ej  :  285  -->  377
ei->ej  :  285  -->  368,  377
ei->ej  :  285  -->  919
ei->ej  :  285  -->  947
ei->ej  :  285  -->  919,  947
ei->ej  :  285  -->  8
ei->ej  :  285  -->  919,  8
ei->ej  :  285  -->  415
ei->ej  :  285  -->  470
ei->ej  :  285  -->  415,  470
ei->ej  :  285  -->  529
```

Fig. 7 Association rule of Streaming-Rule algorithm



6. Conclusion

We are generated association rules by using Apriori algorithm and Streaming –Rules Algorithm using synthetic dataset T10I4D100K http://fimi.cs.helsinki.fi/data/ . Apriori Algorithm generates Static Rules while Streaming-Rules Algorithm generates dynamic. We compare the top-k rule of both algorithms. The static rules are matched with dynamic but some of the errors are in dynamic database. We are using streaming rules i.e. dynamic in applications such as sensor network, in web block data for advertisement where memory is small and processor speed is slow

References

[1] Elena Ikonomovska, Suzana Loskovska, and Dejan Gjorgjevik," A survey of stream data mining",2005.

[2] Hebah H. O. Nasereddin, "Stream Data Mining", International Journal of Web Applications, Volume 3,Number 2, June 2011.pp 90

[3] Rakesh Agrawal, Ramakrishnan Srikant,"Fast Algorithms for Mining Association Rules in Large Databases" VLDB 1994: 487-499 Proceeding of the 20th Conference on Very Large Data Bases.

[4] Li Y. C., Yeh J. S., Chang, C. C ,"Efficient algorithms for mining share-frequent itemsets", 2005 In Proceedings of the 11th World Congress of Intl. Fuzzy Systems Association, pp. 543–539.

[5] G. S. Manku and R. Motwani.,"Approximate frequency counts over data streams", 2002 In Proc. Of the 28th Int'l Conference on Very Large Databases pages 346–357.

[6] Wang H, Yang J,Wang W,Yu P,"Clustering by pattern similarity in large data sets." Proceedings of the 2002 ACM SIGMOD international conference on Management of data Pages 394-405

[7] Biswaranjan Nayak and Srinivas Prasad,"A Pragmatic Approach on Association Rule Mining and its Effective Utilization in Large Database" in May 2012 IJCSI, vol. 9, Issue 3, No 1.

[8] G. Cormode and S. Muthukrishnan," An improved data stream summary: the count-min sketch and its applications.J. Algorithms ", 2005 55(1):58–75.

[9] Frequent items in streaming data: An experimental evaluation of the state-of-the-art.Nishad Manerikar, and Themis Palpanas. Data Knowl. Eng. 68(4):415-430 (2009)

[10] A. Metwally, D. Agrawal, and A. E. Abbadi, "An integrated efficient solution for computing frequent and top-k elements in data streams", 2006 ACM Trans. Database Syst., vol. 31, no. 3, pp. 1095–1133.

[11] Vikarm Singh and Sapna Nagpal Integrating User's Domain Knowledge with Association Rule Mining in March 2010 IJCSI Vol. 7, Issue 2, No.

[12] L. Golab, D. DeHaan, E. D. Demaine, A. Lopez-Ortiz, and J. I. Munro,"Identifying frequent items in sliding windows over on-line packet streams", 2003 In Proceedings of the Internet Measurement Conference, pp.173-178.

[13] G. Cormode and M. Hadjieleftheriou.,"Finding frequent items in data streams" , 2008 PVLDB, 1(2):1530–1541.

[14] Ahmed Metawally, Divyakant Agrawal and Amr EI Abbadi,"Using Association Rules for Fraud Detection in Web Advertising Networks". In Proceedings of the 31st International Conference on Very Large Databases 2005.