International Journal of Database Theory and Application
Vol.8, No.3 (2015), pp.279-288
http://dx.doi.org/10.14257/ijdta.2015.8.3.24

An Efficient Algorithm for Approximate Frequent Intemset Mining

Veepu Uppal

Assistant Professor, Departement of Computer Science & Engineering
Manav Rachna College of Engineering, Faridabad, INDIA
veepu.mrce@mrei.ac.in

Abstract

Frequent itemset mining is a focused theme in data mining research and an important step in the analysis of data arising in a broad range of applications. The traditional exact model for frequent itemset requires that every item occur in each supporting transaction. However, real application data is usually subject to random noise. The reasons for noise are human error and measurement error. These reasons pose new challenges for the efficient discovery of frequent Itemset from the noisy data. Approximate frequent itemset mining is the discovery of itemset that are present not exactly but approximately in transactions.Most known approximate frequent Itemset mining algorithms work by explicitly stating the insertion penalty value and weight threshold. This paper presents a new method for generating insertion penalty value and weight threshold using support count of an item.

Keywords*: Frequent item set mining, Weight, Penalty, Support and Weight Threshold*

1. Introduction

Frequent Itemset: An itemset X is frequent if its support $\sigma(X)$ is more than or equal to some threshold minimum support (min_sup) value, i.e. if $\sigma(X) \geq \min_\sup$
Approximate Frequent Itemset: An itemset that may not be present exactly in all supporting transactions but only approximately is called approximate frequent itemset.

Purpose of Using Approximate Frequent Itemset Most of the frequent item set mining algorithms is based on exact matching .But in fact there are many application where exact matching is not required. One of the application is in telecommunication where different devices like switches, routers and other transmission equipments are inter connected, each device produce an alarm where an alarm shows a abnormal situation. The task is to find the alarms which occur at same time. These alarms can be found using a sliding window over sequence. Each window position then captures a specific slice of the alarm sequence [1, 2]. The underlying idea is that in this way the problem of finding frequent episodes is reduced to that of finding frequent itemsets in a database of transactions: each alarm can be seen as an item and the alarms in a time window as a transaction. The support of an episode is the number of window positions, in which the episode occurs. Unfortunately, alarms often get delayed, lost, or repeated due to noise, transmission errors, failing links etc. If alarms are delayed they will not be considered as frequent item set this leads to loss of interesting frequent items. To cope with these situations we rely on notion of approximate frequent itemset.The lost or delayed alarms are associated with lower insertion cost. For example, in telecommunication networks different alarms can have a different probability of getting lost: usually alarms raised in lower levels of the module hierarchy get lost more easily than alarms originating in higher levels. In such cases, it is convenient to be able to associate the former with lower insertion costs than the latter.

ISSN: 2005-4270 IJDTA
Copyright ⓒ 2015 SERSC

Insertions of a certain item may also be completely inhibited by assigning a very high insertion cost.

2. Proposed System

In telecommunication network different alarms have different probability of getting lost. The alarms at lower level get lost easily than alarms at higher level so it is convenient to associate the formers with lower insertion cost. Thus the lost alarm at lower level can be inserted easily. In the proposed system penalty associated with each item depend upon its support count and the weight threshold is not stated explicitly. The number of insertions allowed in a transaction is limited by weight threshold value, which depends upon penalties of items. In this way the maximum number of insertions allowed in a transaction can be limited [3]. The reason behind calculating the weight threshold value from the difference of maximum and minimum penalty is to limit the number of insertions and hence reducing the number of insertions for the items having maximum support.

Data Model for the Proposed System: The data model for proposed system consists of two lists:

1) Transaction List
2) Item List

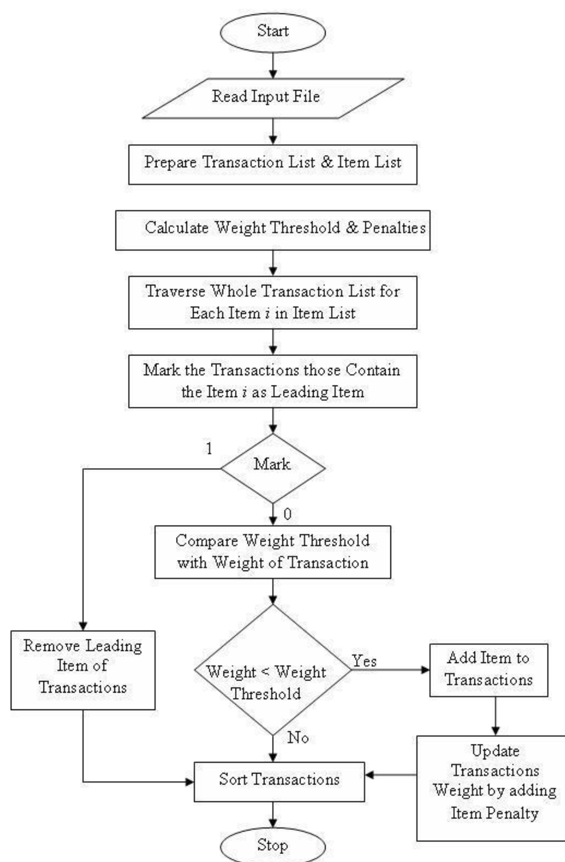

Figure 1. Flow Chart for the Proposed System

Copyright ⓒ 2015 SERSC

Transaction List:

a) Sequence Number: - Represents the sequence number of every transaction in the database.
b) Counter: - Represents the number of occurrences of each transaction
c) Items: - Items contained in a transaction. [4]
d) Weight: - Every transaction is associated with a weight. Initial weight of transaction is 0 and keeps on increasing as items are inserted into the transaction.
e) Mark: - If transaction contains the split item mark is 1, else 0. Initial mark of transaction is 0.
f) Pointer to next transaction: - Point to next transaction in list.

Item List:

a) Name: - Represents the name of item.
b) Support: - Represents the number of occurrences of item in transaction list.
c) Penalty: - Represents the penalty associated with each item. [5]

3. Approximate Frequent Itemset Mining

Table 1. Transaction Set

TID	Transactions
1	ad
2	acde
3	bd
4	bcd
5	bc
6	abd
7	bde
8	bcde
9	bc
10	abd

Step1:-Find out the support value of individual item.
a=4 b=8 c=5 d=8 e=3
Step2:-Arrange the transactions with item in increasing value of initial support.

Table 2. Arranged Items

TID	Items
1	ad
2	eacd
3	bd
4	cbd
5	cb
6	abd
7	ebd
8	ecbd
9	cb
10	abd

Copyright ⓒ 2015 SERSC

Step3:- The transactions are sorted in increasing order of support count of leading item and are then sorted into descending order of size of transaction.

Table 3. Arranged Transaction

TID	Items
1	eacd
2	ecbd
3	ebd
4	abd
5	abd
6	ad
7	cbd
8	cb
9	cb
10	bd

Step4:- Finally prepare the data structure with counters that is to be used in Approximate Frequent Itemset mining algorithm.

Table 4. Transaction Counter

Counter	Transactions
1	eacd
1	ecbd
1	ebd
2	abd
1	ad
1	cbd
2	cb
1	bd

Algorithm for the Proposed System:

```
function SaM_Fuzzy _Modified (a: List of transactions, p: List of items)
var i: char;        (*buffer for split item  name*)
var counter:int;  (*counter for transactions*)
W_min : float;       (*weight threshold value*)
total_support=0: int (*total support of items*)
max: float                    (*maximum item  penalty*)
min: float          (*minimum item penalty*)
begin
while a is not empty do  (* while database is not
                                        empty *)
a[0].wgt=0; (*initialize transactions weight*)
end;
while p is not empty do  (*while item list is not
                                        empty*)
total_support=total_support+p[0].support;
                          (*sum of support of all items*)
end
while p is not empty do
```

 Copyright ⓒ 2015 SERSC

```
p[0].penalty=p[0].support/total_support;                    (*penalty for every item*)
end
max=p[0].penalty;     (*penaltyof first item*)
min=p[0].penalty;
while p is not empty do   (*while item list not
                                      empty*)
if  p[0].penalty>max
then max=p[0].penalty; (*get maximum
                                        penalty*)
end
if p[0].penalty<min
then min=p[0].penalty; (*get minimum
                                       penalty*)
end
end
W_min =max-min;             (*get weight threshold value*)
while p is not empty do   (*while item list not
                                        empty*)
while a is not empty do (* while database not
                                        empty *)
if  a[0].items[0].name = p[0].name
             (*get transaction those have split item *)
then i=a[o].items[0].name; (*get the split item*)
a[0].mark=1; (*mark transactions 1 those contain
                       split item as their leading item*)
remove i from a[0].items; (* remove the split
                                        item *)
else
a[0].mark=0; (*mark transactions 0 those don't
           contain split item as their leading item *)
end
if a[0].mark==0 and a[0].weight< W_min
                  (*check conditions for insertions*)
then a[0].wgt=a[0].wgt+p[0]. penalty;           (*add penalty to transaction weight*)
counter=a[0].counter; (*get counter of
                                    transaction*)
 p[0].support=p[0].support+counter;
end                         (*outer if loop*)
end       (*process transaction one by one*)
end       (*process items one by one*)
end       (*function SAM()*)
```

Assign the initial weight for every transaction as 0.Prepare the transaction list and item list from txt file. Find out the total support of items in the item list. Associate the penalty(c(i)) with every item.

$$c(i) = \frac{\sigma(i)}{\sum_{i=1}^{n} \sigma(i)} \quad (1) \qquad n = \text{total no of items}$$

Copyright ⓒ 2015 SERSC

Then find out the maximum P_{max} and minimum penalty P_{min} in item list and calculate the weight threshold value W_{min}.

$$W_{min} = P_{max} - P_{min} \quad (2)$$

Traverse the transactions list for every item in item list. Mark the transaction those contain first item as splitting item as 1 and mark other transactions as 0. Check if mark is equal to 0 and weight of transaction is less than the weight threshold, add the penalty value to weight of the transaction. The updated weight of transaction will be $w_{(i)}(t) = f(w(t), c(i))$, where f is a function that combines the weight w(t) before editing and the insertion cost $c(i)$.

$$w_{(i)}(t) = w(t) + c(i) \quad (3)$$

Get the counter for the transaction. Add the counter to support of item in item list. Then sort the transactions in transaction list in lexicographical order and repeat the process for every item in item list.

4. Implementation

Transaction list along with sequence number, size of transaction, initial weight, counter of transaction (number of occurrence of same transaction) and items in that transaction .

```
C:\Dev-Cpp\Proposed.exe
Intial Item Support
Item:e  Support:3
Item:a  Support:4
Item:c  Support:5
Item:d  Support:8
Item:b  Support:8

before inserion transactions
------------------------------------------------------------
transaction 1 has size:4 counter:1 weight:0.000000 and transcation is as :eacd
transaction 2 has size:4 counter:1 weight:0.000000 and transcation is as :ecdb
transaction 3 has size:3 counter:1 weight:0.000000 and transcation is as :edb
transaction 4 has size:3 counter:2 weight:0.000000 and transcation is as :adb
transaction 5 has size:2 counter:1 weight:0.000000 and transcation is as :ad
transaction 6 has size:3 counter:1 weight:0.000000 and transcation is as :cdb
transaction 7 has size:2 counter:2 weight:0.000000 and transcation is as :cb
transaction 8 has size:2 counter:1 weight:0.000000 and transcation is as :db
```

Figure 1. Transaction with Counters & Weights

The transactions which occur more than one time are assigned counter equal to number of their occurrence and the repeated transactions are deleted. Total support count of items (total number of items) in given database, along with list of items where each item is associated with support count and penalty. The weight threshold value calculated from penalties is also displayed here.

 Copyright ⓒ 2015 SERSC

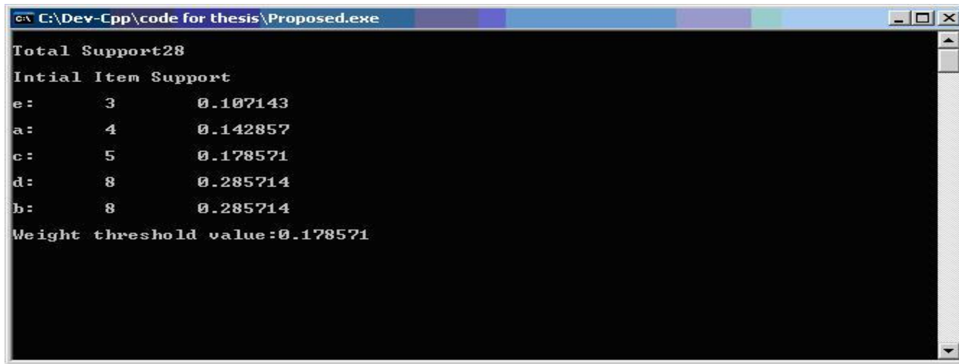

Figure 2. Items Support, Penalties & Weight Threshold Value

Transactions are displayed with their mark and weight. The transactions having ‘e’ as their leading item are marked as 1. The leading item is removed from these transactions and their weights remain same as initially assigned 0. The transactions not having ‘e’ as leading item is marked as 0 are assigned updated weight, the penalty value associated with item ‘e’. The transactions are sorted firstly according to increasing order of support count of leading item and then descending order of size of transactions.

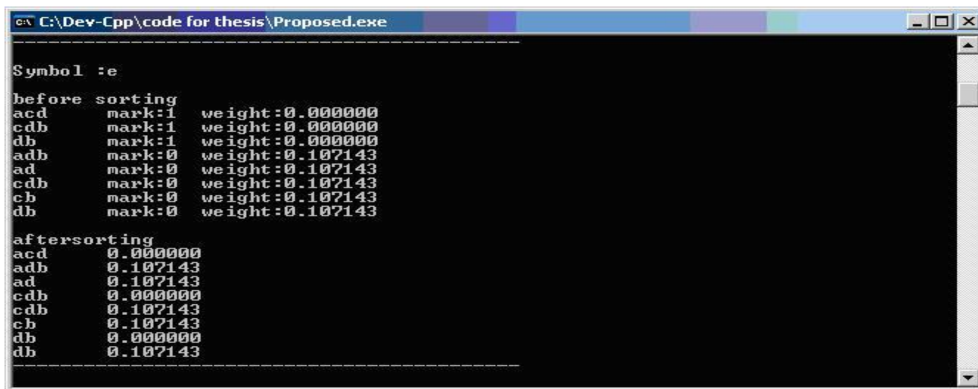

Figure 3: Insertion of Item e

The transactions having ‘a’ as their leading item are marked as 1 and number of insertions associated with these remain unchanged .The leading item ‘a’ is removed from these transactions and their weights remain unchanged. Transactions not having ‘a’ as leading items are marked as 0.The weight of transactions with mark 0 and number of insertions less than weight threshold value is updated by multiplying the previous weight with penalty and their number of insertions associated with these transactions are incremented by 1. The transactions are sorted firstly according to increasing order of support count of leading item and then descending order of size of transactions.

Copyright ⓒ 2015 SERSC

```
C:\Dev-Cpp\Orignal.exe
------------------------------------------------
Symbol :a

before sorting
db      mark:1  weight:0.200000 insertions:1
cd      mark:1  weight:1.000000 insertions:0
d       mark:1  weight:0.200000 insertions:1
cdb     mark:0  weight:0.040000 insertions:2
cdb     mark:0  weight:0.200000 insertions:1
cb      mark:0  weight:0.040000 insertions:2
db      mark:0  weight:0.040000 insertions:2
db      mark:0  weight:0.200000 insertions:1

aftersorting
cdb     0.200000
cdb     0.040000
cb      0.040000
cd      1.000000
db      0.200000
db      0.040000
db      0.200000
d       0.200000
------------------------------------------------
```

Figure 4. Insertion of Item a

In this way all the items are inserted one by one according to their support count. Support count of every item after insertion is shown in figure 1

```
C:\Dev-Cpp\code for thesis\Proposed.exe
ItemSuport after insertion
--------------------------------
e:      10
a:      10
c:      9
d:      8
b:      9
```

Figure 5: Item Support after Insertion

5. Conclusion

Different items should be associated with different penalty values. The items with lower support have higher probability that they have been lost or missed due to reasons like noise or traffic while transferring the data over network. The items with lower support should have lower insertion costs, so that they can be inserted in more transactions and can increase their support count. Thus the penalty values should be depending upon the support count of the items and should be calculated from database instead of being stated explicitly. The weight threshold value should be calculated from database using $P_{\max} - P_{\min}$. If on the other hand the weight threshold value is stated explicitly by the user, the number of insertions will be high for higher weight threshold value hence resulting in more number of insertions. This results in increasing the memory requirement to store missed items. The reason behind calculating the threshold value from the difference of maximum and minimum penalty is to limit the number of insertions and hence reducing the numbers of insertions for the item having maximum support.

References:

[1] B. Christian and W. Xiaomeng, “(Approximate) Frequent Item Set Mining Made Simple with a Split and Merge Algorithm”, Chapter 10 of: Anne Laurent and Marie-Jeanne Lesot (eds.), Scalable Fuzzy Algorithms for Data Management and Analysis: Methods and Design, IGI Global, Hershey, PA, USA **(2009)**, pp.254-272.

Copyright ⓒ 2015 SERSC

[2] H. Mannila, H. Toivonen and A. I. Verkamo, “Discovery of Frequent Episodes in Event Sequences”, Report, **(1997)**.

[3] R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules”, Proc. 20th Int. Conf. on very Large Databases (VLDB 1994, Santiago de Chile), 487–499. Morgan Kaufmann, San Mateo, CA, USA **(1994)**.

[4] B. Christian and W. Xiaomeng, **“**SaM: A Split and Merge Algorithm for Fuzzy Frequent Item Set Mining**”,** Proc. 13th Int. Conf on Fuzzy Systems Association World Congress and 6th Conf. of the European Society for Fuzzy Logic and Technology, IFSA/EUSFLAT Organization Committee, Lisbon, Portugal, **(2009)**, pp.968-973.

[5] X. Wang, C. Borgelt and R. Kruse, “Mining Fuzzy Frequent Item Sets”, Proc. 11th Int. Fuzzy Systems Association World Congress (IFSA’05, Beijing, China), **(2005)**, pp.528– 533.

Copyright ⓒ 2015 SERSC

 Copyright ⓒ 2015 SERSC