International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 4, June - 2012

Improved technique in Sequential Sequence Mining in large database of transaction

Kiran Amin[1], J. S. Shah[2]

[1]*Assoc. Prof. & Head, Department of Computer Engineering,*
U. V. Patel College of Engineering, Kherva, Gujarat

[2]*Prof. & Head, Department of Computer Engg.,*
L.D. College of Engineering, Ahmedabad

Abstract

Sequential Sequence Mining finds useful sequential sequence from the customers' database. It is equal necessary to consider time interval between the consecutive item purchased by the customers. Here we have discussed improved technique in finding this sequence. Many algorithms have been developed to associate the huge amount of sequence. This paper focuses on Sequential Sequence Mining algorithms to generate large sequence from customers' database of transaction.

Keywords- Sequential Sequence Mining, Support, Transformation, Maximal Phase

1. Introduction

The Sequential Sequence Mining finds useful sequence from the large sequence. The comparisons with various techniques are given in terms of memory, generated pattern and various time intervals. We have presented various Sequential Mining techniques with examples. So here we will focus on the time interval sequential pattern. This is really challenging and new field and it will help to find out more specific information to take decision. The Association Rule Mining finds only frequent patterns but it not pay attention on the time interval between two events occurred. Sometimes these things really make importance. Suppose that customer buy item 'A' and then customer come to buy item 'B' after long time so this things sometimes not much importance. But if we know that how much time spends between item 'A' and 'B' than it will help us to take decision. Previous research addresses time intervals in two typical ways which are (a) by the time-window approach, and (b) by completely ignoring the time interval. First, the time-window approach requires the length of the time window to be specified in advance. A sequential pattern mined from the database is thus a sequence of windows, each of which includes a set of items. Items in the same time window are bought in the same period.

2. Association Rule Mining

Association Rule Mining is used to find the frequent sequence which occurred in maximal way. Sequence mining is to find out the frequent sequence which occurred in maximum no of sequences[1]. The working of it is shown in the following example.

SID	Sequences
1	<a(bc)(ef)ad>
2	<bcd>
3	<adb>

Example 1

The small data set of typical transaction is shown in Example 1. The Sequence ID is represented by SID. The sequences are shown against the SID. We consider here as the Customer id and Sequences which we will consider as the items bought by particular customer. Sequences represent in <…> brackets. Here for SID 1, we find out the Sequence <a(bc)(ef)ad>. Here a,b,c,d,e,f are the item code. The (…) bracket indicates that these items are bought by the customer at same time means in single transaction. No bracket is shown for the customer buys a single item in transaction. Similarly all transactions are shown with various SID. We have 5 transactions for SID=1. (1st transaction is a, 2nd truncation we have 2 items b and c, 3rd transaction we have e and f, 4th transaction we have a, and last 5th transaction we have d). In sequence mining 'ab' and 'ba' are the different things. Also we have one more point to be noted in the sequence mining is, how to find out the support. Support in sequence mining is indicating that sequence occurred in how many SIDs. The support for item 'a' is 2 because it is occurred in SID 1 and 3. First sequence mining concept found by the srikant and agrawal in IBM research center. In that they found various algorithms for the sequence mining then lots of research was done in the field of sequential mining.

The association rule mining gives only the frequent sub-sequence but it doesn't give the time interval between successive items. For that new method found is useful to find time interval sequence patterns. It don't only find out the sub-sequences but also gives the time interval between two successive items. Now consider Example 2 in which time interval is added between various transactions.

SID	Sequences
1	<(a,2)(bc,4)(ef,7)(a,8)(d,9)>
2	<(b,4)(c,6)(d,7)>
3	<(a,2)(d,3)(b,6)>

Example 2

Here in Example 2 you can see that the item or itemset you will find the time stamp. Here (a,2), means item 'a' is occurred at time stamp 2.

Now here we will see the real life example for Association Rule Mining,

a) Having bought a laser printer, a customer will come back to buy a scanner and then a CD burner.

While Time interval sequential mining

b) Having bought a laser printer, a customer will come back to buy a scanner in three months and then a CD burner in six months.

Now we will see the some of the algorithms for the sequential mining.

Given two sequences α=< a_1, a_2 … a_n> and β=< b_1 b_2 … b_m>. α is called a subsequence of β, denoted as $\alpha \subseteq \beta$, if there exist integers $1 \leq j_1 < j_2 < \ldots < j_n \leq m$ such that $a_1 \subseteq b_{j1}$, $a_2 \subseteq b_{j2}$,…, $a_n \subseteq b_{jn}$.β is a super sequence of α. Here <abf> is the subsequence of the <a(bc)(ef)ad>.

The length of a sequence is the number of itemsets in the sequence. A sequence of length *k* is called a *k*-sequence. i.e

Candidate 1-subsequences:

<i_1>, <i_2>, <i_3>, …,<i_n>

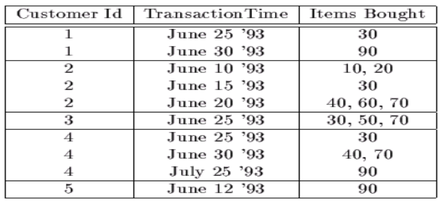
Candidate 2-subsequences:

<i_1, i_2>, <i_1, i_3>, …,<(i_1 i_1)>, <(i_1 i_2)>, …, <(i_{n-1} i_n)>

AprioriAll [1]

It was developed by the srikant and agawal in IBM research lab. It works like typical Apriori. It will find out all the frequent subsequence which will satisfy the minimum support threshold. Here also subsequence item sets will be generated with the help of the candidate itemsets. Also we have to scan dataset every time to fine out *k*-large sequences. AprioriALL have five phases and at the end of all the phases we get the large frequent sequences.

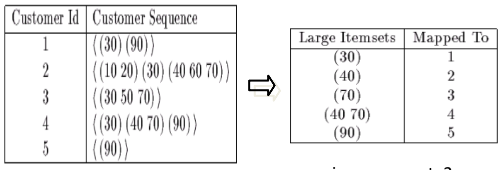
1. Sort Phase : The database is sorted, with customer-id as the major key and transaction-time as the minor key. This step converts the dataset in the sequential order as shown in Figure 1.



Customer Id	TransactionTime	Items Bought
1	June 25 '93	30
1	June 30 '93	90
2	June 10 '93	10, 20
2	June 15 '93	30
2	June 20 '93	40, 60, 70
3	June 25 '93	30, 50, 70
4	June 25 '93	30
4	June 30 '93	40, 70
4	July 25 '93	90
5	June 12 '93	90

Figure 1

2. LitemsetPhase : In this phase we find the set of all Litemsets (Large itemset) L. We are also simultaneously finding the set of all large 1-sequences. Here we arrange the data in form of sequential dataset. Like you can see that we combine the all items by particular customer. i.e. here you can see that customer 1 buy the item 30 and 90 in different transaction and we show it <(30)(90)>. Here '<…>' will indicate as a sequence id or customer ID and '(…)' will indicate the 1 transaction. So here, we continue with our older example and note that we have minimum support threshold id 2(40%).



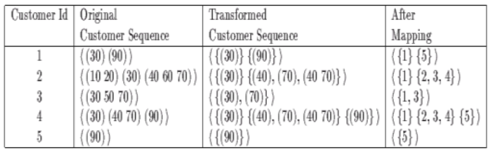
Customer Id	Customer Sequence
1	⟨(30) (90)⟩
2	⟨(10 20) (30) (40 60 70)⟩
3	⟨(30 50 70)⟩
4	⟨(30) (40 70) (90)⟩
5	⟨(90)⟩

Large Itemsets	Mapped To
(30)	1
(40)	2
(70)	3
(40 70)	4
(90)	5

min_sup_count=2

Figure 2

3.Transformation Phase: we need to repeatedly determine which of a given set of large sequences are contained in a customer sequence. To make this test fast, we transform each customer sequence into an alternative representation. In a transformed customer sequence, each transaction is replaced by the set of all litemsets contained in that transaction. If a transaction does not contain any litemset, it is not retained in the transformed sequence.

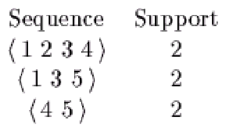


Customer Id	Original Customer Sequence	Transformed Customer Sequence	After Mapping
1	⟨(30) (90)⟩	⟨{(30)} {(90)}⟩	⟨{1} {5}⟩
2	⟨(10 20) (30) (40 60 70)⟩	⟨{(30)} {(40), (70), (40 70)}⟩	⟨{1} {2, 3, 4}⟩
3	⟨(30 50 70)⟩	⟨{(30), (70)}⟩	⟨{1, 3}⟩
4	⟨(30) (40 70) (90)⟩	⟨{(30)} {(40), (70), (40 70)} {(90)}⟩	⟨{1} {2, 3, 4} {5}⟩
5	⟨(90)⟩	⟨{(90)}⟩	⟨{5}⟩

Figure 3

4.Sequence Phase: Here traditional Apriori algorithms for the sequence phase. The algorithm scans the dataset multiple times. In each scan, we start with a seed set of large sequences. We use the seed set for generating new potentially large sequences, called candidate sequences. We find the support for these candidate sequences during the scan over the data. At the end of the scan, we determine which of the candidate sequences are actually frequent. These large candidates become the seed for the next scan. We have two families of algorithms, which we call count-all and count-some. The count-all algorithms count all the large sequences, including non-maximal sequences. The non-maximal sequences must then be pruned out (in the maximal phase). We present one count-all algorithm, called AprioriAll, based on the Apriori algorithm. We present two count-some algorithms: AprioriSome and DynamicSome.

5.Maximal Phase : Find the maximal sequences among the set of large sequences. Having found the set of all large sequences S in the sequence phase, the following algorithm can be used for finding maximal sequences. Let the length of the longest sequence be n. Then,



Sequence	Support
⟨1 2 3 4⟩	2
⟨1 3 5⟩	2
⟨4 5⟩	2

Figure 4

Here we show that how Apriori will work. Now we will just take over view of AproriSome [1] and DynamicApriori [1]. Here just take over view and compression because they are not use in normal as they have some of the limitations.

AprioriSome algorithm runs in forward n backward pass. In forward pass, we only count sequence of certain lengths. For example, we might count sequences of length 1, 2, 4 and 6 in the forward pass and count sequences of length 3 and 5 in the backward pass. It saves to the time by not count those sub-sequences which are not maximum. But some time s we required all the frequent sub-sequences rather than only max-subsequences. So it also saves the time and memory.But as we know today memory is never an issue so we can eliminate this advantage.

DynamicSome algorithm is work as same as the AprioriSome. Just difference is in generating the candidate sequence. The candidate sequences that are counted, is determined by the variable step. In the initialization phase, all the candidate sequences of length upto and including step are counted. Then in the forward phase, all sequences whose lengths are multiples of step are counted. Thus, with step set to 3, we will count sequences of lengths 1, 2, and 3 in the initialization phase, and 6,9,12,... in the forward phase. We really wanted to count only sequences of lengths 3,6,9,12,... We can generate sequences of length 6 by joining sequences of length 3. We can generate sequences of length 9 by joining sequences of length 6 with sequences of length 3, etc. However, to generate the sequences of length 3, we need sequences of lengths 1 and 2, and hence the initialization phase.As in AprioriSome, during the backward phase, we count sequences for the lengths we skipped over during the forward phase. However, unlike in AprioriSome, these candidate sequences were not generated in the forward phase. The intermediate phase generates them. Then the backward phase is identical to the one for AprioriSome.

Comparison

For the large dataset dynamic take lots of time to generate the candidate sequences compare to AprioriAll and AprioriSome algorithms, while AprioriAll and AprioriSomealmost same output. DynamicSome performs worse than the other two algorithms mainly because it generates and counts a much larger number of candidates in the forward phase because it will include those candidate sequence also which subsequences are not frequent and large. The major advantage of AprioriSome over AprioriAll is that it avoids counting many non-maximal sequences. However, this advantage is reduced because of two reasons. First, candidates C_k in AprioriAll are generated using L_{k-1}, whereas AprioriSome sometimes uses C_{k-1} for this purpose. Since $L_{k-1} \subseteq C_{k-1}$, the number of candidates generated using AprioriSome can be larger. Second, although AprioriSome skips over counting candidates of some lengths, they are generated nonetheless and stay memory resident. For lower supports, there are longer large sequences, and hence more non-maximal sequences, and AprioriSome does better. That means for the lower number of customer AprioriSome perform better than the AprioriAll. And That's why we say that AprioriAll used more in recent days because we deal with the huge amount of data and as well as we have more than enough memory to save the file so memory will be the no issue here.

GSP

After sometime they introduce the GSP algorithm [2]. It is working same like the Apriori but the difference is that it is used for the more generalization purpose. So it is used the some of the limitation which is really useful for the some applications. List of the limitations are mention below.

1. **Absence of time constraints**

Users often want to specify maximum and/or minimum time gaps between adjacent elements of the sequential pattern. For example,a book club probably does not care if someone bought “C++ book", followed by“JAVA book" three years later; they may want to specify that a customer should support a sequential pattern only if adjacent elements occur within aSpecified time interval, say three months. (So for a customer to support this pattern,the customer should have bought “JAVA book" within three months of buying “C++ book".)

2. **Rigid definition of a transaction**

For many applications, it does not matter if items in an element of a sequential pattern were present in two different transactions, as long as the transaction-times of those transactions are within some small time window. That is, each element of the pattern can be contained in the union of the items bought in a set of transactions, as long as the difference between the maximum and minimum transaction-times is less than the size of a sliding time window. For example, if the book-club specifies a time window of a week, a customer who ordered the “C++ book" on Monday, “C book" on Saturday, and then “JAVA book" and “HTML book" in a single order a few weeks later would still support the pattern “`C++ book' and `C book', followed by `JAVA book' and `HTML book' ".

3. **Absence of taxonomies**

Many datasets have a user-defined taxonomy (is-a hierarchy) over the items in the data, and users want to find patterns that include items across different levels of the taxonomy.

This way GSP is work better than AprioriAll because it generates the less amount of the frequent itemsets than AprioriAll. Next algorithm will be followed the Apriori Techniques.

MEMISP

Ming and lee come in 2005 with some new concept to find sequences[5]. In this paper, we propose a memory indexing approach for fast sequential pattern mining, named MEMISP[5]. This algorithm scans the sequence database only once for reading data sequences into memory. The find-then-index technique recursively find the item which constitute a frequent sequence and constructs a compact index set which indicates the set of data sequences for further exploration. Though effective index advancing, fewer and shorter data sequences need to be processed in MEMISP as discovered patterns getting longer. It is better algorithm than the GSP and PrefixSpan[3][4], because it works without any candidate generation or projection dataset. MEMISP work same as the pseudoprojection PrefixSpan. But some way it work different and better than the pseudoprojection PrefixSpan.We will see all of that differences here. The pseudoprojection PrefixSpan is not effective and efficient in large dataset because we have to load the dataset in main memory for the pseudoprojection. MEMISP do the same process but have one partition-and-divide technique which deals with the extra-large database. It will take one more scan to find the frequent patterns from the large dataset. This difference noted when dataset can’t store in the main memory.
Another difference is when dataset can store in the main memory means the dataset can’t be as large as the above. MEMISP doesn’t remove any of the in-frequent 1-length dataset, where PrefixSpan remove the in-frequent 1-length dataset. And store memory index as list (MEMISP) instead of pair (PrefixSpan).

Improved PrefixSpan(I-PrefixSpan)

Dhanysaputra come with the improve version of the PrefixSpan[6]. It is called I-PrefixSpan(improved PrefixSpan). This algorithm improves PrefixSpan in two ways: (1) it implements sufficient data structure for Seq-Tree framework to build the in-memory database sequence and to construct the index set which help to reduce the load on the main memory, and (2) instead of keeping the whole in-memory database, it implements separator database to store the transaction alteration signs. Seq-Tree framework with sufficient data structure is the other contribution in I-PrefixSpan which is used to store in-memory sequence database and to construct the index set. Seq-Tree is a general tree with two certain characteristics: (1) all leaves must be located at the same depth and (2) the height of the tree is at least 2. It is shown in Figure 5.

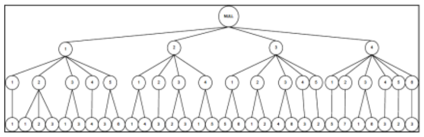


Figure 5

I-PrefixSpan

1st it was introduce by dr. Yen Chen in 2003[7]. He finds the sequential patterns that include time intervals, called time-interval sequential patterns. This work develops two efficient algorithms for mining time-interval sequential patterns[10]. So we will learn it with example. Lets see,

Here you can see small dataset which will use to find the Time Interval Sequential Mining.

Sid	Sequence
10	((a,1) , (c,3) , (a,4) , (b,4) , (a,6) , (e,6) , (c,10))
20	((d,5) , (a,7) , (b,7) , (e,7) , (d,9) , (e,9) , (c,14) , (d,14))
30	((a,8) , (b,8) , (e,11) , (d,13) , (b,16) , (c,16) , (c,20))
40	((b,15) , (f,17) , (e,18) , (b,22) , (c,22))

Time interval Sequence dataset

Now first we will apply I-PrefixSpan on this dataset and will get the result. It will provide the Time interval Sequences patterns. Also only consider the Time Interval between 2 successive items only.

So this algorithm works same as the typical PrefixSpan but have to deal with the Time Interval. So here we have to take all the possible projection of frequent items. In the first step we will follow the same step like original PrefixSpan. Find out the 1-length sequences. So here we will scan the dataset (table-1) and find out the 1-sequences. If support are 50% and TI= $\{I_0, I_1, I_2, I_3\}$, where $I_0 : t = 0$, $I_1 : 0 < t \leq 3$, $I_2 : 3 < t \leq 6$ and $I_3 : 6 < t \leq \infty$, then we find out the frequent items are (a), (b), (c), (d) and (e). We eliminate the (f) as it is not frequent and not pass the minimum support thresh hold. Now we have to make projection table for all the frequent 1-sequence. Let us take 'a' as the frequent items and make projection table base on that. Here you can see that 'a' item occurred 3 time in the 1st sequence so we have to take make projection for the all the 3 times because we deal with the time interval along with the sequence. We can use different notation for that so we can identify. We use [*Sid* :*Pos*]**,** where *Sid* is the identifier of the sequence and *Pos*is the position of item. Now we have 5 postfix sequences in the projected database and they are as follow.
[10:1] ((*c*, 3)(*a*, 4)(*b*, 4)(*a*, 6)(*e*, 6)(*c*, 10)),
[10:4] ((*b*, 4)(*a*, 6)(*e*, 6)(*c*, 10)),
[10:6] ((*e*, 6)(*c*, 10)),
[20:7] ((*b*, 7)(*e*, 7) (*d*, 9)(*e*, 9)(*c*, 14)(*d*, 14)),
[30:8] ((*b*, 8)(*e*, 11)(*d*, 13)(*b*, 16)(*c*, 16), (*c*, 20)).
Now have to scan this projected dataset again and have to find out the frequent sequences. Now we have to make the table for that. In that table 1st column indicated the time interval and 1st raw will indicate the frequent items. Remaining cell we show that frequency of data occurrence with particular time interval in the projected dataset and base on that we will get the 2-length sequences base on table-2.

As we can see from the table that cell belongs to B-I_0 are frequent because it satisfy the minimum support (which is 2 here). So we can say that (a, I_0, b) is the frequent sequence. That means customer come to by item 'b' after I_0 time of purchase 'a'

Table	**A**	**B**	**C**	**D**	**E**
I_0	0	3	0	0	2
I_1	1	1	1	1	3
I_2	1	0	1	1	1
I_3	0	1	3	1	0

References:

[1]. R. Agrawal and R. Srikant, "Mining Sequential Patterns," Proc. 1995 Int'l Conf. Data Eng. (ICDE '95), pp. 3-14, Mar. 1995.
[2]. R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," Proc. Fifth Int'l Conf. Extending Database Technology (EDBT '96), pp. 3-17, Mar. 1996.
[3]. J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu.Freespan: Frequent pattern-projected sequential pattern mining. In *Proc. 2000 Int. Conf. KnowledgeDiscovery and Data Mining (KDD'00)*, pages 355–359, Boston, MA, Aug. 2000.
[4]. J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth," Proc. 2001 Int'l Conf. Data Eng. (ICDE '01), pp. 215-224, Apr. 2001.
[5]. L.M. Yen and L. S.Y. Lee, "Fast discovery of sequential patterns through memory indexing and database partitioning," Journal Information Science and Engineering, vol. 21, pp. 109-128, 2005.
[6]. Dhany , Saputra and RambliDayang, R.A. and Foong, Oi Mean (2007) Mining Sequential Patterns Using I-PrefixSpan. In: World Academy of Science, Engineering and Technology , 14-16 December, 2007.
[7]. Chen, Y.L., Chiang, M.C. and Ko, M.T. (2003). "Discovering time- interval sequential patterns in sequence databases," Expert Syst. Appl., Vol. 25, No. 3, (pp. 343-354).