Hiding Co-occurring Sensitive Patterns in Progressive Databases

Amruta Mhatre
Dept. of Computer Science and Engg
Indian Institute of Techniology, Roorkee,
Uttarakhand, India. 247667
amruta.01@gmail.com

Durga Toshniwal
Dept. of Computer Science and Engg
Indian Institute of Techniology, Roorkee,
Uttarakhand, India. 247667
durgatoshniwal@gmail.com

ABSTRACT

These days lot of work is been carried out in the field of privacy preserving data mining. Apart from the standard techniques of privacy preservation, methods are being proposed specific to the data mining tasks carried out. However most of these methods work on static databases. One such a method is downgrading application effectiveness. The effectiveness of applications may be downgraded by hiding sensitive association rules, hiding sensitive sets of patterns etc. Sometimes although a particular pattern is not interesting, its co-occurrence with another pattern may reveal certain sensitive information. In this paper we present a novel technique to hide sensitive co-occurring sequential patterns. The proposed method works on progressive databases. Progressive databases are a generalized model of static, dynamic and incremental databases. The applicability of the method is also extended to suit these different types of databases.

Keywords

Data mining, sequential pattern mining, co-occurrence hiding, progressive databases.

1. INTRODUCTION

Data Mining, better known as knowledge discovery can be described as obtaining possibly unseen information from large data. This process of analyzing data from different perspectives and summarizing it into useful information has a great application in the business world. For example, the mined knowledge can be used to increase revenue, cut costs or make certain marketing decisions. It allows users to analyze data from many different dimensions, categorize it, and summarize the relationships identified. Technically, *data mining can be defined as the process of "mining" knowledge from large amounts of data* [1].

Data collection is an important step in data mining. The type of database used for storing the collected data, depends upon the application of this data. Sometimes the presence of obsolete data in the data used for mining may result into erroneous results. Progressive databases provide a generalized solution to incrementally store all the collected data. These databases allow dynamic addition and deletion of data. This avoids re-mining of the whole data when new data is added [3]. The static and incremental databases are special cases of such databases. As a result progressive databases hence have a greater scope for application in real world applications.

Sequential Pattern mining is a commonly used data mining technique for business applications. The applications of sequential pattern mining are mostly in the fields of targeted marketing, promotions and sales, customer retention etc. The use of sequential pattern mining over progressive databases enhances the applicability of the mining technique since it mines the most recent patterns without the side-effects of presence of obsolete data.

Security and privacy are other important issues for any data collection method .The collected data is intended to be shared used for making strategic decisions. Also, when data is mined for applications like customer profiling, medical analysis etc. large amounts of sensitive and private data about individuals needs to be gathered, stored and processed. Such situations make it necessary to maintain the confidentiality of the data in order to prevent its illegal access. Sometimes data mining results may also disclose some new implicit information about individuals which is against privacy policies. For these reasons, privacy preserving data mining is essentially a sought after field of research in data mining [2].

A lot of work in the field of privacy preservation has been done. Researchers aim to propose generalized methods that could be applied to any type of data irrespective of the underlying mining technique. Some such methods introduced include randomization, l-diversity, k-anonymity, cryptographic techniques etc. However some methods are specific to certain types of mining techniques. These techniques include query auditing [4], association rule hiding [5], frequent itemset hiding [6] etc. These methods can be categorized as methods that preserve privacy by downgrading the effectiveness of the applications.

This paper proposes a method to hide co-occurring frequent patterns while mining for sequential patterns in progressive databases. The paper is organized as follows: Section 2 describes the problem statement and its related concepts. The base approach for blocking co-occurring patterns is discussed in section 3. The proposed work and its integration with its integration with an

PAIS '10, March 22, 2010, Lausanne, Switzerland.

algorithm that mines sequential patterns on progressive databases have been discussed in Section 4. Section 5 discusses results and certain performance issues with respect to the proposed algorithm. The concluding remarks in the work are discussed in Section 6 with certain ideas for further enhancements.

2. RELATED WORK

2.1 Problem Statement

Sometimes although a particular pattern is not interesting, its co-occurrence with another pattern may reveal certain sensitive information. Suppressing or blocking frequent patterns are the two methods that have been proposed in order to deal with this problem. One such approach has been discussed by O.Abul in [6]. The method proposed here is however applicable only to static databases. Databases that are progressive in nature are more generalized and hence the methods proposed for static databases need not be always applicable. The limitations of these kinds of databases are that they require dynamic processing since they deal with frequently updating data. Also the memory constraints do not allow storage of a large amount of data. Due to this there is a need for a dynamic method for hiding co-occurring patterns.

The method proposed here suppresses co-occurring frequent patterns by blocking some of the sensitive patterns over progressive databases. As a result if pattern set P = (A, B) is sensitive, the proposed method blocks either A or B; avoiding both A and B to be frequent together at a particular time instance. This avoids the co-occurrence of A, B as frequent patterns at a particular timestamp. This algorithm works on progressive databases and hence can work with any type of databases, be it static, dynamic or incremental and can be said to be based on the idea proposed in [6].

An application of this can be in strategic marketing. For example if many superstores wish to mine global shopping patterns across all their customer transaction databases, they would require a privacy preservation mechanism to keep intact the privacy of the data they are sharing. Also a particular party may consider the co-occurrence of a particular pattern set as sensitive (may be for promotional / sales reasons) and would not be interested in sharing this pattern, if it occurs, with other parties. As a result the global patterns published would contain only certain patterns from the sensitive set but never the whole set.

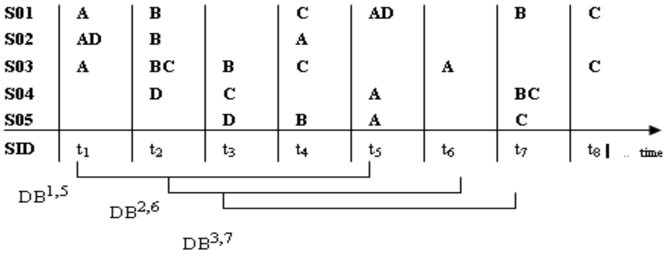
2.2 Preliminaries

Frequent sequential pattern mining, commonly known as sequential pattern mining, was first addressed in [7] by R. Agrawal and R. Srikant as the problem: “Given a database of sequences, where each sequence consists of a list of ordered item sets containing a set of different items, and a user defined minimum support threshold, sequential pattern mining finds all subsequences whose occurrence frequencies are no less than the threshold from the set of sequences”. This term *sequence* can be more formally described as:

Definition 1: Let $X =\{x_1, x_2, x_{3\dots}\ x_n\}$ be a set of different items. An element *e*, denoted by $< x_{1,}\ x_{2,}\ . . >$, is a subset of items belonging to *X* which appear at the same time. A sequence *s*, denoted by $< e_1 ; e_2 ; . . . ; e_m >$, is an ordered list of elements. A sequence database *DB* contains a set of sequences, and | *DB* | represents the number of sequences. in *DB*. A sequence $\alpha = < a_1 ; a_2 ; . . . ; a_n >$ is a subsequence of another sequence $\beta =< b_1 ; b_2 ; . . . ; b_m >$ if there exist a set of integers, $1 \leq i_1 \leq i_2 \leq i_n \leq m$, such that a_1 is a subset of b_{i1}; a_2 is a subset of b_{i2} ; .. . and a_n is a subset of b_{in} [3].

In case of progressive databases, it is required that the obsolete data is pruned out of the database and only recent, updated information remains in the database. The time period under study id thus called the Period of Interest. It is can be represented as:

Definition 2: Period of Interest (*POI*) is a sliding window. The length of the *POI* is a user specified time interval. The sequences having elements whose timestamps fall into this period *POI*, contribute to | *DB* | for sequential patterns generated at that timestamp. On the other hand, the sequences having elements with timestamps older than those in the *POI* are pruned away from the sequence database immediately and do not contribute to the | *DB* | thereafter [3]. This is illustrated in Fig.1.



S01	A	B		C	AD		B	C
S02	AD	B		A				
S03	A	BC	B	C		A		C
S04		D	C		A		BC	
S05			D	B	A		C	
SID	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8

Figure 1: Sample Database

Another concept to be borrowed from association rule hiding, frequent itemset hiding can be described as:

Definition 3: Let $X =\{x_1, x_2, x_{3\dots}\ x_n\}$ be the set of itemsets to be hidden from *C* .Given a threshold *α* the frequent itemset hiding problem requires to transform the database *D* into a database *D’* such that*:*

$$\text{Support}_{D'}(x_i) < \alpha \ \ \textit{and}$$

$$|\ \text{Support}_{D}(x_i)\text{-Support}_{D'}(x_i)| \quad \text{is minimized}$$

The first requirement asks for lowering the support of sensitive itemsets below a level, so the receiver of D’ can not mine any of the sensitive itemsets at support cnt = α. The second requirement is the minimization objective which claims for solutions destroying supports of itemsets as less as possible [6].

3. PROPOSED WORK

The proposed method blocks the co-occurrence of sets of patterns, the occurrence of which is considered sensitive. The method maintains information each set of patterns considered sensitive in a structure called *blockSet*. The *blockSet* maintains updated information about the status of all patterns, along with their support counts. The *threshFlg* associated with each pattern indicates whether the support of a particular pattern has crossed the threshold value. The algorithm updates the status of the pattern status if the support count of a pattern in the *blockSet*, changes. The support count of the pattern changes when an instance of that pattern occurs in the database. The method can be briefly explained as follows: The algorithm identifies each occurrence of a sensitive pattern in the database. Every occurrence of the sensitive pattern is thus noted in order to keep track of the support count of the pattern. When the support count of a pattern crosses the threshold, the *threshFlg* of that particular pattern is set. The set status of the threshFlg indicates that the co-occurrence of this pattern may reveal some sensitive information.

While updating the support count of a sensitive pattern, the algorithm checks for the support counts of other patterns in the same set. If all of the patterns except the pattern under consideration are already frequent, i.e their support counts have already crossed the threshold, this current occurrence of the pattern is ignored. In other words this pattern is *blocked.* Incrementing the support count of this pattern would result into the co-occurrence of all the patterns in that *blockSet,* and hence revealing some sensitive information. On the other hand, if not all patterns in the *blockSet* are infrequent, the occurrence of the pattern is acknowledged and the support count of the pattern is increased. The *threshFlg* of the pattern is updated in case the support count crosses the threshold. This is algorithmically expressed in Fig 2.Here *bthresh* is the blocking threshold, which indicates the maximum value of the support count beyond which the pattern may be a candidate for blocking.

```
Procedure hidingCo-occurring Patterns( blockSet )
For (every element in the dataset)
   if (element creates pattern that exists in blockSet )
      add element and set pattern.bcandid = 1;
      if (  new instance of the pattern exists)
         if ( pattern.bcandid ==1)
            if (pattern.support > threshold for rest n-1 patterns in set)
               block sequence;
            else
               add sequence;
               if(pattern.support >=bthresh)
                     pattern.threshFlg =1;
End
```

Figure 2: Procedure hidingCo-occurringPatterns

4. INTEGRATING CO-OCCURRENCE BLOCKING WITH PROGRESSIVE SEQUENTIAL PATTERN MINING

In this section we apply the concept proposed in the previous section to a sequential pattern mining algorithm over progressive databases. The algorithm used for progressive sequential pattern mining is *PISA* [3].The algorithm hence obtained by merging the two ideas, progressively mines patterns and blocks the co-occurrence of patterns in sensitive sets.

The algorithm is described in Fig.3. At the beginning of the algorithm, the algorithm gathers information about the sets of patterns whose co-occurrence is considered sensitive in a data

```
Algorithm co-occurBlock( minSup, poi)
Var currentTime;
Var thresh;
Var blockSet;
Mtree root;
while ( there is new data in the Db)
      Ele = Data obtained at current timestamp.
      bthresh = |Db|* minSup;
      for ( every pattern in every set in blockSet)
            prune obsolete sequences
            if ( pattern.support < bthresh)
                  reset pattern.threshFlg;
      insert (root,Ele);
      currentTime ++;
End
```

Figure 3: Algorithm co-occurBlock

```
Procedure insert (Tc , Mtree)
for (each node of Mtree in post order)
  if (node is Root node)
    for (ele of every seq in eleSet)
      for (all combination of elements in ele)
        if( element ==label of one of node.child)
          if (seq is in node.child.seq_list)
            update timestamp of seq to Tc;
          else
            create a new sequence with timestamp = Tc;
        else   // create a child node
            create child node with element, seq, timestamp = Tc;
 else  //  for a common node
    for (every seq in the seq_list)
      if (seq.timestamp <= Tc - POI)
        delete seq from seq_list and move to next seq;
      if (there is new ele of seq in eleSet)
        for  (all combinations of elements in ele)
          if (element is not on the path from Root)
            if (element == label of one of node.child)
              if(seq is in node.child.seq_list)
                child.seq_list.seq.timestamp = node.seq.timestamp;
              else
                if (node.child.bcandid =1)
                  find set corresponding to pattern in blockSet;
                  if  (threshFlg of not more than n-2 elements in the  set is set)
                    create new sequence with timestamp = seq.timestamp,
                    update pattern.threshFlg;
                  else
                    block sequence
                else
                    create new sequence with timestamp = seq.timestamp,
          else  //create a child
            create a new child with element,seq, timestamp =
                seq, timestamp.
      if  (path from Root to node lies in blockSet)
      child.bcandid =1;
        if  (seq_list.size ==0)
          delete this node and all of its children from its parent;
      if (seq_list.size>=support*sequence number)
        output labels of path from Root to this node as sequence pattern;
```

Figure 4: Procedure Insert

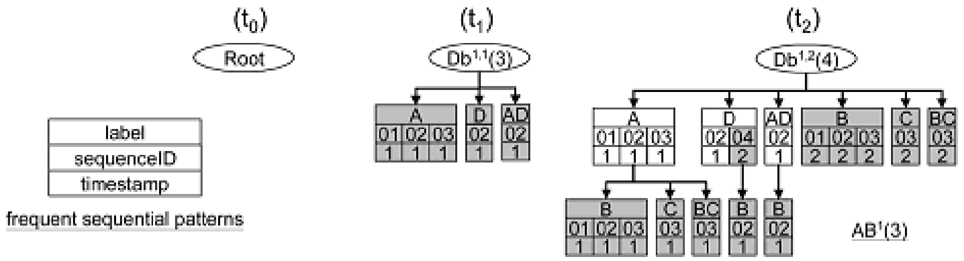


Figure 5: Sample working of procedure insert (with *blockSet* as *null*)

structures called *blockSets.* (described in Section 3). At every timestamp the algorithm computes the threshold, to determine if the support count of a pattern is large enough to be considered as a *frequent pattern.* The threshold of a pattern is calculated as the number of distinct sequences present in the database multiplied by the minimum support. The data about support counts of patterns in the *blockSet* is also updated so as to prune the obsolete sequences and update values of the corresponding thresh's fiags. The algorithm then extracts data from the database corresponding to the current timestamp and integrates it into the data structure used for storing candidate patterns (*Mtree*).The method of assimilation of data into the *Mtree* and the corresponding procedure for blocking is explained in Fig. 4.

Whenever a series of elements appear in a sequence (refer *SID* in Fig.1), path from the root is created labeled by the respective elements of the pattern with the corresponding sequence number on which this pattern occurred. This path from root to node called a candidate pattern. If this pattern exists in the *blockSet*, the *bcandid* field of the last node in the pattern is set to 1, indicating that the sequences occurring at this node may be blocked in order to suppress co-occurrence of sensitive patterns. If a path already exists the concerned fields of the nodes are updated with the respective information. The timestamp for each node of the candidate sequential pattern is marked according to timestamp of the starting element of the candidate pattern. While adding sequences to nodes with bcandid value *set*, the procedure first checks for the values of *thresh*Flg of other patterns in the set. If the *threshFlg* of not more than n-2 patterns in a set are 1 then the pattern adds the sequence else blocks the sequence. An obsolete element (i.e. element which lies out of the *POI*) and a node having no sequence numbers in its sequence list are pruned from the sequence list of the node and the *Mtree* respectively, ensuring only up to date candidate patterns in the *Mtree*.

After all the candidate sequential patterns are generated, the algorithm checks for the number of sequence IDs in a sequence list of all nodes. If the number of sequence IDs in a particular node is larger than minimum support multiplied number of sequences in the current *POI*, the path from the root till that node is considered as a frequent sequential pattern.

5. RESULTS AND DISCUSSION

The results are computed for a synthetic dataset [7] with 300 sequences and 30 records. These results are compared with the results of frequent itemset hiding. It is seen that the results (Fig. 6) are better since the number of patterns suppressed using frequent pattern /item set hiding are *n* times that of the proposed method, where *n* is the number of patterns in the set. Also the number of patterns which get blocked due to the by blocking the patterns in a set of patterns, also reduces by a factor of *n*. Fig. 7 compares the number of patterns reconstructed in with the increase in the size of the set to be blocked.

No of blocking sets	% ge of patterns mined after co-occurrence blocking		%ge of patterns mined after frequent itemset hiding	
	At ts =15	At ts = 30	At ts = 15	At ts = 30
1	98	99.4	97.4	98.8
2	89	91	79	83.8
3	81	83	62	67.6

Figure 6: Comparison of co-occurrence pattern blocking algorithm with frequent pattern hiding

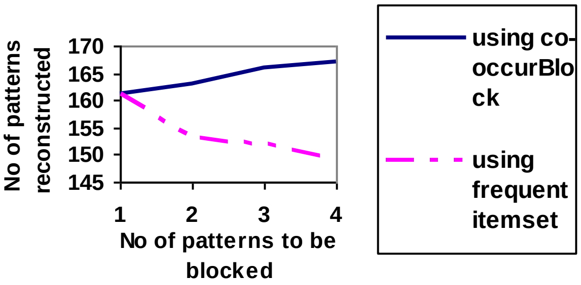


Figure 7: Number of patterns reconstructed with size of blocking set

Unlike the other approaches to blocking co-occurring frequent sequential patterns, this approach dynamically chooses the pattern to be suppressed. Hence, the constraint 2 mentioned in Definition 3 cannot be applicable in this scenario. Moreover the percentage of supersets of patterns blocked as a result of blocking a particular pattern also depends upon the pattern that is blocked.

6. CONCLUSIONS

The proposed method presents a novel technique to downgrade the effectiveness of a group of sensitive patterns over progressive databases. As compared to the previous methods of suppressing frequent sensitive patterns by manipulating support counts, the approach presented in this paper avoids the co-occurrence of sensitive frequent patterns by suppressing one of the patterns and keeping it from being frequent. The pattern to be suppressed is selected at runtime. Since this method blocks only a single pattern in order to avoid co-occurrence, the number of consequently suppressed sequences is limited. However the dynamic choice of the pattern to be blocked may also serve as a limitation in certain cases. But this limitation can be overlooked keeping in mind the generalized nature and corresponding limitations of progressive databases.

This area can be further explored in order to develop methods to select the pattern to be blocked. This will enable to keep a check on the number of patterns lost in case of a wrong choice of pattern to be blocked.

7. REFERENCES

[1] J.Han and M.Kamber, “Data Mining: Concepts and Techniques”, Series Editor Morgan Kaufmann Publishers, ISBN 1-55860-489-8, 2000.

[2] V.S. Verykios, E. Bertino, I.N. Fovino, L.P. Provenza, Y. Saygin, and Y. Theodoridis, “State-of-the-Art in Privacy Preserving Data Mining,*” ACM SIGMOD* Record, Vol. 3, No. 1, pp. 50 - 57, 2004.

[3] J.W Huang, C.Y.Tseng, J.C Ou, and M. S.Chen, “A General Model for Sequential Pattern Mining with a Progressive Database,” *IEEE Transactions on Knowledge and Data Engineering*, Vol. 20, No. 9, pp. 1153 - 1167, 2008.

[4] Verykios V. S., Elmagarmid A., Bertino E., Saygin Y.,, Dasseni E.: Association Rule Hiding. *IEEE Transactions on Knowledge and Data Engineering*, 16(4), 2004.

[5] Adam N., Wortmann J. C.: Security-Control Methods for Statistical Databases: A Comparison Study. *ACMComputing Surveys*, 21(4), 1989.

[6] O. Abul. “Hiding Co-Occurring Frequent Itemsets,” 2nd Intl Workshop on Privacy and Anonymity in the Info. Soc. (PAIS’09), Russia, 2009.

[7] http://www.datasetgenerator.com/