Mining Temporal Features in Association Rules

Xiaodong Chen[1] and Ilias Petrounias[2]

[1] Department of Computing & Mathematics, Manchester Metropolitan University
Manchester M1 5GD, U.K., e-mail: X.Chen@doc.mmu.ac.uk

[2] Department of Computation, UMIST, PO Box 88, Manchester M60 1QD, U.K.,
e-mail: ilias@co.umist.ac.uk

Abstract: In real world applications, the knowledge that is used for aiding decision-making is always time-varying. However, most of the existing data mining approaches rely on the assumption that discovered knowledge is valid indefinitely. People who expect to use the discovered knowledge may not know when it became valid, or whether it still is valid in the present, or if it will be valid sometime in the future. For supporting better decision making, it is desirable to be able to actually identify the temporal features with the interesting patterns or rules. The major concerns in this paper are the identification of the valid period and periodicity of patterns and more specifically association rules.

1. Introduction

The problem of association rules was introduced in [1] and has been extended in different ways. Most existing work overlooks any time components, which are usually attached to transactions in databases. Without this knowledge most of the information resulting from data mining activities is not of great use. For example, it is not useful to look at *all* supermarket transactions that have taken place over the years in order to identify patterns. Most of this information will be outdated. Temporal issues of association rules have been recently addressed in [2] and [4]. [2] focuses on the discovery of association rules with known valid periods and periodicities. The valid period shows the absolute time interval during which an association is valid, while the periodicity conveys when and how often an association is repeated. Valid period and periodicity are specified by calendar time expressions in [2]. In [4], the concept of calendric association rules is defined, where the rule is combined with a calendar that is a set of time intervals and is described by a calendar algebra. Here we focus on two mining problems for temporal features of some known/given association: 1) finding all interesting contiguous time intervals during which a specific association holds (section 2); 2) finding all interesting periodicities that a specific association has (section 3).

2. Discovery of Longest Intervals

Given a time-stamped database and a known association, one of our interests is to find all possible time intervals during which this association holds. Those intervals are composed of a totally ordered set of contiguous constructive intervals (called granular intervals) with a given granularity representing a non-decomposable interval of some fixed length. The interval granularity is the size of each granular interval (e.g. Hour, Day, etc.). Each expected time interval is denoted by $\{G_i, G_{i+1}, ..., G_j\}$, where G_k ($i \le k \le j$) is a granular interval, and the time domain can be also represented by a totally ordered set of all contiguous granular intervals. We define LENGTH(ITVL, GC) as the number of intervals of granularity GC in ITVL.

J.M. Zytkow and J. Rauch (Eds.): PKDD'99, LNAI 1704, pp. 295-300, 1999.

Definition 2.1: Given an association AR, an interval ITVL is *valid* with respect to AR if the temporal association rule (AR, ITVL) satisfies *min_supp* and *min_conf*.

More often than not people are just interested in intervals the duration of which is long enough, since some short intervals may not be periods of particular interest.

Definition 2.2: Given an association AR and an interval granularity GC, an interval ITVL is *long* with respect to AR if: ITVL is valid with respect to AR, and LENGTH(ITVL, GC) ≥ *min_ilen (minimal interval length)*.

Consider a long interval ITVL with respect to AR. It is possible that ∃ ITVL' ⊂ ITVL and LENGTH(ITVL', GC) ≥ min_ilen. ITVL' is not a long interval with respect to AR, since AR may have low support and/or confidence during ITVL', but very high support and confidence during the rest of the period(s) in ITVL.

Definition 2.3: Given an association AR and an interval granularity GC, an interval ITVL is *strictly long* with respect to AR if for any ITVL', ITVL' ⊂ ITVL and LENGTH(ITVL', GC) ≥ *min_ilen*, ITVL' is long with respect to AR.

With respect to a given association AR, for any two strictly long intervals, $ITVL_1$ and $ITVL_2$, if $ITVL_1 \subset ITVL_2$, we say that $ITVL_2$ is strictly longer than $ITVL_1$.

Definition 2.4: Given an association AR and an interval granularity GC, an interval ITVL is *longest* with respect to AR if: 1) the interval ITVL is strictly long with respect to AR, and 2) not ∃ ITVL" ⊃ ITVL, where ITVL" is strictly long with respect to AR.

With respect to a given association AR, there may be a series of different longest intervals existing along the time line.

Definition 2.5: Given a set of time-stamped transactions (***D***) over a time domain (***T***), a known association (*AR*), minimum support (*min_supp*), minimum confidence (*min_conf*), and minimum interval length (*min_ilen*), the problem of mining valid time periods is to find *all possible longest intervals* with respect to the association AR.

Suppose time domain $\boldsymbol{T} = \{G_1, G_2, ..., G_n\}$, where G_i ($1 \le i \le n$) is a granular interval. The set of time-stamped transactions ***D*** is ordered by timestamps and is partitioned into $\{D(G_1), D(G_2), ..., D(G_n)\}$. The search problem can be considered as successively looking for all longest sequences along the time domain sequence $\{G_1, G_2, ..., G_n\}$. For each possible longest interval, the search can be performed in two steps: 1) find its *seed interval*; 2) extend this seed interval to the corresponding longest interval.

Definition 2.6: Let an interval $ITVL = \{G_i, G_{i+1}, ..., G_j\}$, ITVL is called as *a seed interval* if it satisfies the following conditions: 1) it is a strictly-long interval; 2) no strictly long interval starting before G_i covers ITVL; and 3) no other interval being covered by ITVL satisfies the previous two conditions.

For example, let *min_ilen* be 3 and assume that $ITVL_1 = \{G_5, G_6, G_7, G_8, G_9\}$ and $ITVL_2 = \{G_7, G_8, G_9, G_{10}, G_{11}, G_{12}\}$ are two longest intervals, then $\{G_7, G_8, G_9, G_{10}\}$ could be a seed interval of $ITVL_2$ if there is no other strictly long interval covering it. However, although $\{G_7, G_8, G_9\}$ is strictly long, it can not be a seed interval of any longest interval since $ITVL_1$ covers it (condition 2). Also, $\{G_7, G_8, G_9, G_{10}, G_{11}\}$ is not regarded as a seed interval because $\{G_7, G_8, G_9, G_{10}\}$ is a seed one (condition 3).

Proposition 2.1: Let an interval $ITVL = \{G_i, G_{i+1}, ..., G_j\}$. If ITVL is *a seed interval*, there must be one and only one longest interval that covers ITVL and this longest interval is an interval starting from G_i (this holds due to definitions 2.4, 2.6).

This says that if we can find all the seed intervals, we can extend them to get all the longest intervals. The questions are: how to find the seed interval and how to extend it to the longest interval. Let's answer the second question first. If ITVL is a seed interval, then the corresponding longest interval can be derived from ITVL as follows:

1) If the last granular interval of ITVL is the last granular interval along the time domain, output ITVL (which is obviously a longest interval) and terminate the search.

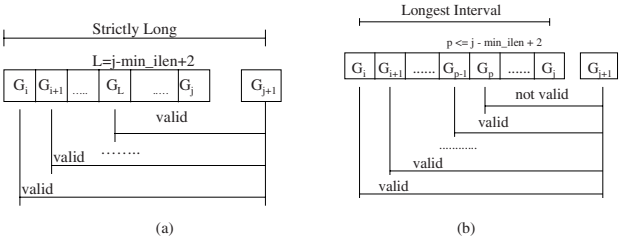

Figure 2.1 Extend a seed interval to a longest interval

2) Let ITVL = $\{G_i,G_{i+1},...,G_j\}$, consider next granular interval G_{j+1} along the time domain and check if $\{G_i,G_{i+1},...,G_j,G_{j+1}\}$ is a strictly long interval by successively checking if each interval $\{G_k, G_{k+1}, ..., G_j, G_{j+1}\}$ (where k = i, i+1, ... , j-*min_ilen*+2) is valid. If all of them are valid (Figure 3.1(a)), interval $\{G_i,G_{i+1},...,G_j,G_{j+1}\}$ is surely a strictly long one. G_{j+1} is added to ITVL and becomes the last granular interval of ITVL. Go back to step 1) to look for a longer interval. Otherwise, once we find an interval $\{G_p,G_{p+1},...,G_j,G_{j+1}\}$ (where i≤p≤ j-*min_ilen*+2) is not valid (Figure 2.1(b)), we can conclude that there is no longer interval starting from G_i that will be strictly long. Output ITVL and finish the search for this corresponding longest interval.

Now, we consider how to find the seed interval. During the course of the search for all longest intervals, two cases in which we will start to look for the seed interval are:

1) At the beginning of the search, we need to find the seed interval of the first longest interval along the time domain. We can start from the beginning of the time line and check successively each interval with the length of *min_ilen* until we find that it is valid. This interval will be the seed interval of the first longest interval.

2) At the time when a longest interval is just found (see Figure 2.2), we need to find the seed interval of the next longest interval. Assume the currently found longest interval is ITVL = $\{G_i, G_{i+1}, ..., G_j\}$. As discussed above, once we find an interval $\{G_p, G_{p+1}, ..., G_j, G_{j+1}\}$ (where i ≤ p ≤ j - *min_ilen* + 2) is not valid, we terminate the search for this current longest interval. Obviously, it is impossible that any sub-interval of ITVL or any interval $\{G_k, G_{k+1}, ..., G_j, G_{j+1}\}$ (where i ≤ k ≤ p) would be a new seed interval. So, the next seed interval must be the first following strictly-long interval that does not end before G_{j+1}. Figure 2.2 shows two possible cases in which the next seed interval will be found. In Figure 2.2(a), the next seed interval is the first following strictly-long interval which ends by G_{j+1}. The length of this interval may be greater than *min_ilen*. In second case, as shown in Figure 2.2(b), the seed is the first following valid interval with the length of *min_ilen*. Here, G_q is any granular interval after G_L.

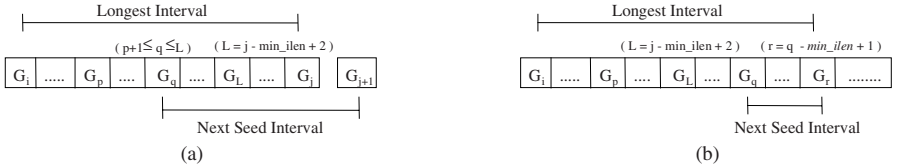

Figure 2.2 Finding the next seed interval

Figure 2.3 shows an one-pass algorithm (LISeeker) for searching for all the longest intervals of a given association rule AR in a database ***D*** over a time domain ***T***. For

simplicity and without loss of the generality, suppose $\boldsymbol{T}$ = {G_1,G_2,...,G_n} and $\boldsymbol{D}$ = {D[G_1],D[G_2],...,D[G_n]}. The search is gradually made by scanning all the partitions of the database, D[G_1],D[G_2],...,D[G_n]. To monitor the search process and avoid multiple passes over the database, we build a structured queue G_QUEUE, which is an ordered list of granular intervals. This ordered list can be considered as a candidate interval of the next expected seed interval or the next longest interval, depending on its status. In G_QUEUE, each element G_i consists of the three fields: *trans_num* (number of transactions in D[G_i]), *body_num* (number of transactions containing AR.body, in D[G_i]) and *rule_num* (number of transactions containing AR.body ∪ AR.head, in D[G_i]). IN(G_QUEUE, G_i) adds a granular interval G_i with relevant numbers into the rear of G_QUEUE and OUT(G_QUEUE) removes a granular interval from the front of G_QUEUE. The search starts with the first interval with the length of *min_ilen* along the time line. In the algorithm, ptr1 and ptr2 always point to the start and end of the current candidate interval and go forward alternately. In the outer iteration in the algorithm (starting at line 3), a seed interval is firstly being looked for (lines 4 to 17) and the corresponding longest interval is then being derived from this (lines 18 to 39).

```
(1)   for ( i = 1; i≤ min_ilen; i++ ) { SCAN(D[Gi]); IN(G_QUEUE, Gi); }
(2)   ptr1= 1;  ptr2 =min_ilen;
(3)   for ( ptr2 ≤ n ) do {
(4)       for ( ptr2 ≤ n ) do {  /* looking for the next strictly long interval */
(5)           i = ptr1;  j = ptr2;
(6)           for ( i≤ j -min_ilen+1  ) do {
(7)               if ( NotValid({Gi , ..., Gj}) ) break;
(8)               i++;
(9)           }
(10)          if ( i > j -min_ilen+ 1 ) break;  /* found a seed {Gptr1, ...., Gptr2} */
(11)          else if ( i = j -min_ilen+ 1 && j = n ) exit;  /* no any more seed */
(12)          else {
(13)              for (k = ptr1; k≤ i; k++) do OUT(G_QUEUE);
(14)              if ( i = j -min_ilen+ 1 )
(15)                  { j++;  SCAN(D[Gj]);  IN(G_QUEUE, Gj); }
(16)              ptr1 = i + 1;  ptr2 = j;
(17)          } }
(18)      for ( ptr2≤ n ) do {  /* looking for the next longest interval */
(19)          if (ptr2 = n){
(20)              OUTPUT({Gptr1, ..., Gptr2 }) ;   /* found a longest interval */
(21)              exit;
(22)           }
(23)          i = ptr1; j = ptr2 + 1;
(24)          for ( i≤ j -min_ilen+ 1  ) do {
(25)              if ( NotValid({Gi , ..., Gj}) ) break;
(26)              i++;
(27)          }
(28)          if ( i≤ j - min_ilen+ 1 ) {
(29)              OUTPUT({Gptr1, ..., Gptr2 });    /* found a longest interval */
(30)              for (k = ptr1; k≤ i; k++) do OUT(G_QUEUE);
(31)              if ( i = j -min_ilen+ 1 )
(32)                  { j++;  SCAN(D[Gj]);  IN(G_QUEUE, Gj); }
(33)              ptr1 = i + 1;  ptr2 = j;
(34)              break;
(35)          }
(36)          else { /* extending {Gptr1, ..., Gptr2+1} with Gj */
(37)              SCAN(D[Gj]); IN(G_QUEUE, Gj);
(38)              ptr2 = j;
(39)          }  } }
```

Figure 2.3 Search Algorithm for Longest Intervals (LISeeker)

Function SCAN passes over all the transactions in D[G_i] counting the number of those transactions, the number of the transactions containing the body of AR, and the number of transactions containing both the body and head of AR. Function NotValid

checks if the interval $\{G_i,...,G_j\}$ is valid in terms of the given minimum support and confidence. Since the relevant counts (*trans_num*, *body_num*, *rule_num*) in each data partition $D[G_k]$ ($i \le k \le j$) have been recorded in G_QUEUE, the support and confidence of AR in $D[\{G_1,G_2,...,G_n\}]$ can be computed by the sums of those relevant counts. The function OUTPUT will convert the longest interval that was found in the form of $\{G_{ptr1}, ..., G_{ptr2}\}$ into a time period described by an understandable representation.

3. Discovery of Longest Periodicities

Given a time-stamped database and a known association, another temporal feature is a set of regular intervals in cycles, during each of which this association exists. A periodic time can be represented as a triplet <Cycle, Granule, Range>. Cycle is the length (given by a calendar) of a cycle, Granule is the duration (given by a calendar) of a granular interval, and Range is a pair of numbers which give the position of regular intervals in the cycles. Given a periodic time PT=<CY,GR,[x:y]>, its interpretation $\Phi(PT)=\{P_1,P_2,...,P_j,...\}$ is regarded as a set of intervals consisting of the x-th to y-th granular intervals of GR, in all the cycles of CY. If we partition the time domain ***T*** by CY and express it as $\{C_1,C_2,...,C_j,...\}$ (where C_j is an interval of CY), we have $P_j \subseteq C_j$ (for any j>0). For example, let PT=<Year,Month,[10:12]>, ***T*** can be expressed as $\{year_1,year_2,..., year_j,...\}$ and $\Phi(PT)$ as $\{Q_1,Q_2,...,Q_j,...\}$ (Q_j is the last quarter of year $_j$).

Definition 3.1: Given an association AR, a periodic time PT=<CY, GR, RR> is valid with respect to AR if there are not less than *min_freq%* of intervals in $\Phi(PT)$, which are strictly long with respect to AR.

Definition 3.2: Given association AR, periodic time PT=<CY, GR, RR> is longest with respect to AR if PT is valid with respect to AR, and not (PT' = <CY, GR, RR'> such that RR' $\supset$ RR and PT' is strictly long with respect to AR.

Definition 3.3: Given a set of time-stamped transactions (***D***) over a time domain (***T***), a minimum support (*min_supp*), a minimum confidence (*min_conf*), a minimum frequency (*min_freq*), a minimum interval length (*min_ilen*), as well as the cycle of interest (CY) and granularity (GR), the problem of mining the periodicities of a known association (AR) is to find *all possible periodic times* <CY, GR, RR>, which are longest with respect to the association AR. Here, RR is expected to be discovered.

According to the above, the cyclicity (CY) and granularity (GR) of the periodic time that are of interest, are given. So, we can suppose time domain $\boldsymbol{T} = \{C_1, C_2, ..., C_m\}$, where C_i ($1 \le i \le m$) is a cycle, so that the data set ***D*** can be partitioned into $\{D[C_1], D[C_2], ..., D[C_m]\}$. The search can be decomposed into two sub-problems:

1) search for all the longest intervals over each C_i from $D[C_i]$;
2) derive the possible periodicities from all longest intervals found in each cycle C_i.

The algorithm in section 2 can be used for the search for the longest intervals over each C_i from dataset $D[C_i]$. We only focus on the second sub-problem: how to derive the periodic time from all longest intervals that are found in each cycle C_i. We use C_i.ITVLSET to express the set of all longest intervals found in each cycle C_i. The algorithm (PIDeriver) used for the derivation is based on the following steps:

1) Scanning each C_i.ITVLSET and adding all longest intervals that are found into an ordered list A_LIST, which is ordered by the starting point and the ending point of the interval. Intervals in A_LIST are called essential intervals.
2) Looking for all candidate intervals by splitting essential intervals in A_LIST. If any two intervals in A_LIST intersect and the intersection is long enough, then the intersection is added into the candidate interval set C_LIST.

3) For each candidate interval in C_LIST, counting the number of cycles in which there exists a longest interval covering this interval; computing the frequency for this candidate interval; and removing it from C_LIST if it does not satisfy the minimum frequency (*min_freq* %). For each interval $ITVL_i$ in C_LIST, removing it from C_LIST if there is another interval $ITVL_j$ in C_LIST, $ITVL_i \subseteq ITVL_j$.

4. Implementation and Experimental Results

The algorithms described have been implemented in a prototype mining system [3]. The kernel of the system is a temporal mining language [3], which has been integrated with SQL on the basis of ORACLE. For testing the performance of the algorithms, we generated three datasets that mimic the transactions within one year in a retailing application. Each transaction is stamped with the time instant at which it occurs. We run the algorithm LISeeker to look for longest intervals of a given association of items with the fixed interval granularity, minimum support and minimum confidence, but different minimum interval lengths. The results show that no matter how much the given minimum interval length is, the escaped CPU times are just slightly different. The expense for the search is mostly spent on the scanning of the database and it is scanned only once in any case of different minimum interval lengths. Therefore, the search time depends almost exclusively on the size of the dataset. The escaped CPU time rises almost linearly with the sizes of the datasets. Since the search for longest periodicities is based on algorithm LISeeker and the cost for running LPDeriver can be almost neglected, compared with the cost for running LISeeker, its performance feature is very similar to the search for longest intervals.

5. Conclusions and Future Work

This paper concentrated on the identification of interesting temporal features (valid period and periodicity) of association rules. Based on the concepts of long intervals and longest periodicities, the mining problems were defined and the search techniques were discussed with the corresponding algorithms. We believe that the identification of similar temporal features of other types of patterns can occur naturally within the same framework. Work is now concentrating on the development of algorithms for the identification of similar temporal features for the different types of patterns. An interactive temporal data mining system for supporting the described tasks has been developed with an appropriate SQL-based language [3]. It is currently being extended to support other mining tasks.

References

1. Agrawal, R., Imielinski, T., and Swami, A, *Mining Associations between Sets of Items in Massive Databases*, Proceedings of ACM SIGMOD International Conference on Management of Data, Washington D.C., May 1993.
2. Chen, X., Petrounias, I., and Heathfield, H., *Discovering Temporal Association Rules in Temporal Databases*, Proceedings of International Workshop on Issues and Applications of Database Technology (IADT'98), Berlin, Germany, July 1998.
3. Chen, X. and Petrounias, I., *A Framework for Temporal Data Mining*, Proceedings of 9th International Conference on Database and Expert Systems Applications (DEXA'98), Vienna, 1998.
4. Ramaswamy, S., Mahajan, S., and Silberschatz, A., *On the Discovery of Interesting Patterns in Association Rules*, Proceedings of 24th VLDB Conference, New York, pp.368-379, 1998.