Higher Order Apriori

Shenzhi Li, Aditya P. Belapurkar, Xiaoning Yang, Mark J. Dilsizian
William M. Pottenger, Murat Can Ganiz, Christopher D. Janneck

Lehigh University Department of Computer Science and Engineering
19 Memorial Drive West, Bethlehem, PA 18015, USA

{shl3, apb204, xiy204, mjd204, billp, mug3, cdj2}@lehigh.edu

Abstract Frequent itemset mining (FIM) is a well known technique for discovering relationships between items. Most FIM algorithms are based on first-order associations between items in the same record. Although a few algorithms capable of discovering indirect propositional rules exist, they do not extend beyond second-order. In addition, although multi-relational ARM discovers higher-order rules, the rules are non-propositional and the algorithm is NP-complete. This article introduces Higher Order Apriori, a novel algorithm for mining higher-order rules. We extend the itemset definition to incorporate k-itemsets up to nth-order, and present our levelwise order-first algorithm: levelwise meaning that the size of k-itemsets increases in each iteration (as with Apriori), and order-first meaning that at each level, itemsets are generated across all orders. Support is calculated based on the order of itemsets and the number of higher-order associations connecting items.

Keywords Association Rule Mining (ARM), Data Mining, Text Mining, Sequence Mining, Multi-relational ARM, Higher Order ARM, Machine Learning

1 Introduction

Association Rule Mining (ARM) is one of the most widely used algorithms in data mining. Generating rules based on statistics of item co-occurrence, ARM produces output in the form of propositional rules. Co-occurrence refers to instances where two or more items appear in the same context, and is also called 1st-order association. Much work has been done developing techniques for generating, analyzing and measuring 1st-order associations (e.g., [2], [16], [17]), but most techniques do not support mining across transaction boundaries. Notable exceptions include sequence mining and multi-relational ARM. These are examples of approaches that discover *higher-order* associations.

Higher-order associations are formed by linking different contexts (e.g., transactions) through one or more common items. Consider the following example from traditional market-basket analysis: if customer "A" purchases {computer, OS}, customer "B" buys {laptop, OS} and customer "C" gets {laptop, mouse, battery}, then several higher-order associations can be formed. These include computer-to-laptop through OS, OS-to-mouse through laptop, computer-to-battery through OS and laptop, etc. The first two associations are termed 2nd-order since they each span two contexts, while the computer-to-battery association is 3rd-order. Overall, any association greater than 1st-order (i.e., spanning at least two contexts) is termed a higher-order association.

Higher-order associations are currently employed in a number of real world applications including medical research, marketing analysis, law enforcement and homeland defense. In the field of medicine, for example, Literature Based Discovery [18] has been used to uncover a higher-order association between Fish Oil and Raynaud's disease in medical literature, leading to a potential new treatment [8]. An important law enforcement application that employs 2nd-order associations is COPLINK® Detect [9], which assists law enforcement personnel through textual entity extraction and link-generation through the use of a semantic network. Mooney et al. [14] discusses a link discovery algorithm, as part of the DARPA Evidence Extraction and Link Discovery program that uses Inductive Logic Programming in its mining of multi-relational data.

Several other research efforts are also revealing promising results on the utility of higher-order associations. In [12], Kontostathis and Pottenger present experimental evidence suggesting a strong correlation between 2nd-order association of terms and the performance of Latent Semantic Indexing in terms of F_β, the harmonic mean of precision and recall. In related work, based on statistical comparisons of distributions of higher-order association frequencies, Ganiz and Pottenger report that classes of instances in labeled training data may be separable based on the characteristics of the higher-order associations alone (without recourse to a learning algorithm) [7].

As Tan et al. [19] [20] intimate, the future of cross-selling in the e-Marketplace may be greatly impacted by the results of further study of higher-order associations. This is in keeping with our results on simulated e-Marketplace data, which indicate that the latent information contained in higher-order associations can be leveraged to build more effective association rule models.

2 Related Work

As noted in the Introduction, traditional ARM algorithms only identify 1st-order associations, i.e., co-occurrence in the same context. Sequential pattern mining, on the other hand, uses 2nd-order associations to discover frequent subsequences as patterns in a sequence database. The sequential pattern mining algorithm was introduced by Agrawal and others in [1] and [3]. In later work Mannila et al. introduce an efficient solution to the discovery of frequent patterns in a sequence database [13]. Chan et al. [4] study the use of wavelets in time-series matching and Faloutsos et al. [6] and Keogh et al. [11] propose indexing methods for fast sequence matching using R* trees, the Discrete Fourier Transform and the Discrete Wavelet Transform. Multi-relational ARM is a type of ARM algorithm designed specifically to mine higher-order rules across tables in a single database [5] [15]. In fact, multi-relational data mining in general (not limited to ARM) is an emerging research area that enables the analysis of complex, structured types of data such as sequences in genome analysis. Similarly, there is a wealth of recent work concerned with enhancing existing data mining approaches to employ relational logic. WARMR, for example, is a multi-relational enhancement of Apriori presented by Dehaspe and Raedt [5]. Although WARMR provides a sound theoretical basis for multi-relational ARM, it does not seriously address the efficiency of computation. In fact the runtime performance of WARMR depends heavily on the implementation of θ-subsumption, and because θ-subsumption is NP-complete, performance is poor. In addition, the model sacrifices the perspicuity of a propositional representation. More recently, methods that address 2nd-order associations have been discussed in the context of the e-Marketplace. Tan et al. [19] proposes a method of discovering indirect (2nd-order) associations between items in a database of transactions. The proposed INDIRECT algorithm is introduced in [19] and uses Apriori to link two items that are both highly dependent on a mediator set. This approach has similarities with sequence mining; in effect, the mediator set replaces the item held in common that is used to create the sequence (e.g., customer_ID). Of the efforts in higher-order ARM surveyed, only multi-relational ARM provides a solid theoretical foundation for discovering higher-order rules based on higher-order associations. Yet this same foundation results in an NP-complete complexity. In addition, none of the approaches surveyed treat associations higher than 2nd-order efficiently; most of them do not consider orders higher than two at all. To address these issues, in the following section we introduce a novel algorithm, Higher Order Apriori, that efficiently takes higher-order associations into account while providing the efficacy of the propositional representation available in existing algorithms in sequence mining and indirect association.

3 Theoretical Framework

Higher Order Apriori discovers itemsets that cross record boundaries. We first extend the itemset definition to incorporate k-itemsets up to nth-order, and then show how to generate and calculate support for higher-order itemsets. Following this, we present the pseudo code for Higher Order Apriori.

Definition 1: *If item a and item b from different transactions can be associated across n distinct records, then we say item a and b are nth-order associated, denoted* $a \sim^{r_1} i_1 \sim^{r_2} i_2 \sim \cdots \sim^{r_{n-1}} i_{n-1} \sim^{r_n} b$ *where ~ represents the co-occurrence relation and all records are distinct. The order of a higher-order association is determined by the number of records n.* This definition includes the constraint that a given record can appear at most once in a higher-order association. This is necessary to be consistent with the original ARM framework. For example, given a higher-order association $a \sim^{r_1} i_1 \sim^{r_2} i_2 \sim^{r_1} b$, based on definition 1, *a* and *b* are 2nd-order associated because there are two distinct records in the link. This conflicts however with the fact that *a* and *b* actually are 1st-order associated since they both come from r_1.

Definition 2: *An nth-order k-itemset is a k-itemset for which each pair of its items is nth-order associated.* For example, if *abc* is a 3rd-order itemset, then there must exist at least three 3rd-order associations between *a* and *b*, *b* and *c* and *a* and *c* respectively. Having defined higher-order itemsets, the remaining question is how to generate them and calculate support. The support for 1st-order k-itemsets is calculated by counting the number of records containing the itemsets. However, this method is infeasible for higher-order itemsets that cross record boundaries. Thus, we introduce the concept of a *higher-order recordset* which is the context of higher-order itemsets.

Definition 3: *Two records are n*th*-order linked if they can be linked through n-2 distinct records. The n*th*-order link between records is denoted* $r_1 \sim^{i_1} r_2 \sim^{i_2} \cdots \sim^{i_{n-1}} r_n$. *An n*th*-order k-itemset is a k-itemset for which each pair of records is n*th*-order linked.* Clearly, given an *n*th-order link between two records $r_1 \sim^{i_1} r_2 \sim^{i_2} \cdots \sim^{i_{n-1}} r_n$, if $a \in r_1$, $b \in r_n$ and $a \neq b$, then there must exist at least one *n*th-order association between item *a* and *b*: $a \sim^{r_1} i_1 \sim^{r_2} i_2 \sim \cdots \sim^{r_{n-1}} i_{n-1} \sim^{r_n} b$. Given the higher-order links between records and the derived higher-order associations between items, the corresponding higher-order itemsets can be generated.

Definition 4: *A n*th*-order itemset* $i_1 i_2 \ldots i_n$ *is supported by a n*th*-order recordset* $r_1 r_2 \ldots r_n$ *if no two items come from the same record.* For example, given the following three records <r_1, *abc*>, <r_2, *adef*>, <r_3, *bfgh*>, then $r_1 r_2 r_3$ is a *2*nd-order recordset as we have $r_1 \sim^{a} r_2$, $r_1 \sim^{b} r_3$ and $r_2 \sim^{f} r_3$. Since for example $b \in r_1$, $e \in r_2$ and $g \in r_3$, *beg* is a *2*nd*-order* itemset supported by the *2*nd*-order* recordset $r_1 r_2 r_3$. To calculate the support of a higher-order itemset we need to know both the number of recordsets supporting the itemset and the order of each recordset. A recordset may appear frequently because there may be many different higher-order links supporting the recordset. For example, if there are two 2nd-order links between r_1 and r_2, say $r_1 \sim^{i_1} r_2$ and $r_1 \sim^{i_2} r_2$, then recordset r_1 r_2 could be formed based on either link and thus has a support of two. To simplify the problem, we group higher-order links that support the same recordset into a *link group*, written as $r_1 \sim^{I_1} r_2 \sim^{I_2} \cdots \sim^{I_{n-1}} r_n, I_j = r_j \cap r_{j+1}$. The *size* of a link group, i.e. the number of links in the group, can be calculated by taking the product of the sizes of I_j. For example, given a 3rd-order link group L_g: $r_1 \sim^{acd} r_2 \sim^{ef} r_3$, the number of links in L_g is *size(*L_g*)*=|*acd*|*|*ef*|=6.

Similar to links, recordsets can also be grouped together. For example, the 2nd-order 4-recordset in Figure 1.a and the 2nd-order 4-recordset in Figure 1.b can be grouped together to form the 2nd-order 4-recordset in Figure 1.c. The only difference is that the recordset in

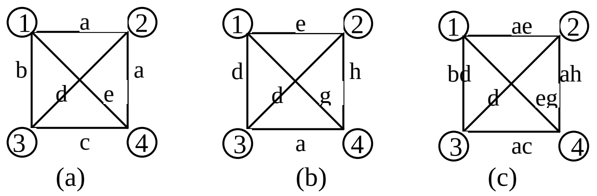

Figure 1. Grouping recordsets

Figure 1.c is composed of link groups instead of links. In what follows, when we discuss recordsets, we refer to recordsets composed of link groups as exemplified in Figure 1.c.

Similarly, the *size* of a recordset composed of link groups can be calculated by taking the product of the sizes of each link group. For example, the size of the recordset in Figure 1.c is 2*2*2*2*2*1=32. It is also important to note that a given recordset might be composed of different link group combinations. This may happen for nth-order recordsets when n is greater than two. For example, Figures 2.a and 2.b represent the same 3rd-order 3-recordset $r_1r_2r_3$ formed in two ways. Recordset $r_1r_2r_3$ in Figure 2.a may have a size different than in Figure 2.b because the link groups connecting r_1 and r_2 (through r_5 and r_4 respectively) may have different sizes. The total size of $r_1r_2r_3$ must incorporate both recordsets. Generalizing from this example, given j instances of n^{th}-order k-recordset rs, its size is defined as: $size_{n\_k}(rs) = \sum_{t=1}^{j}(\prod_{u=1}^{k(k-1)/2}\prod_{v=1}^{n-1}|I_v|)$, where j represents the number of different ways to form rs, and the double product inside the parentheses represents the size of the tth recordset.

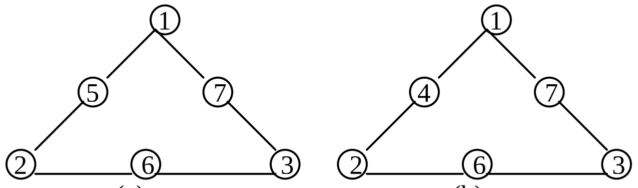

Figure 2. Two forms of recordset $r_1r_2r_3$

Given the size values for all recordsets, we define the support of a k-itemset is as:

$$\sup_k(is) = \sum_{u=1}^{\max\_order} \frac{\log_{10}\sqrt{\sum size_{n\_k}(rs)+1}}{u}$$

The sum results from the fact that since the same k-itemset can be generated at different orders, the *global* support for a given k-itemset must include the *local* support at each order u. Thus the ratio $\log_{10}\sqrt{\sum size_{n\_k}(rs)+1}\big/u$ calculates the *local* support for an itemset at a specific order u. The idea behind this ratio is simply to account for both the number of higher-order links supporting a given itemset as well as the order of the itemset. As order grows, our intuition is that support ought to decrease – thus the denominator u. This reflects the assumption that the longer the link between records, the weaker the itemset association. In contrast, the more links connecting records in a recordset, the stronger support ought to be. These two intuitions are just that – clearly, extensive experimentation is required to ascertain the utility of this definition of support.

The challenge arises when we consider the exponential growth of $size_{n\text{-}k}(rs)$ – although order grows linearly, $size_{n\text{-}k}(rs)$ grows exponentially. Again, our intuition is that both of these factors are equally important. Thus, in order to constrain $size_{n\text{-}k}(rs)$ to grow linearly with order, we first take the square root of $size_{n\text{-}k}(rs)$ and then the $\log_{10}$. The square root accounts for the $O(n^2)$ growth in the number of edges in a recordset as order grows; the $\log_{10}$ accounts for the exponential growth of $size_{n\text{-}k}(rs)$. We add one to $size_{n\text{-}k}(rs)$ in the numerator to ensure that support is non-zero.

4 Higher Order Apriori Algorithm

In this section, we present the Higher Order Apriori algorithm, which discovers rules based on higher-order associations. Our higher-order ARM is structured in a level-wise order-first manner. Level-wise means that the size of *k*-itemsets increases in each iteration (as is the case for Apriori), while order-first means that at each level, itemsets are generated across all orders. We will first introduce the notation used in the algorithm, and then present the algorithm in detail.

Table 1. Notation

RS_k	Set of *k*-recordsets. Each member has: i) recordset; ii) order; iii) size of the recordset.
RS_{n_k}	Set of n^{th}-order *k*-recordsets. Each member has: i) recordsets; ii) size.
IS_k	Set of *k*-itemsets. Each member has: i) itemset; ii) global support of the itemset.
IS_{n_k}	Set of n^{th}-order *k*-itemsets. Each member has: i) itemsets; ii) local support.

The input to *HO-Apriori* is a connected undirected graph, where each vertex represents a record, an edge between two vertices exists when two records share at least one common item, and the weight of the edge represents the number of common items. In terms of the nomenclature in section 3, a path in G is actually the link group between two records. Thus, the problem of discovering link groups is the problem of finding all simple paths.

```
HO-Apriori (G)
1.  Enumpath(G, max_order)
2.  for ( $k$ = 3; $RS_{k-1} \neq \phi$; $k$++) // k: size of the recordset
3.   for ( $n$ = 2; $n$ < max_order; $n$++)     // n: order
4.      $RS_{n\_k}$ = Gen_RS($RS_{n\_k-1}$ );
5.      foreach recordset $rs \in RS_{n\_k}$
6.          Enum_IS($rs$, $n$);
7.   foreach itemset $is$ where $|is|=k$
8.      for (u=2; u<max_order; u++)
9.          if $is \in IS_{u\_k}$
10.             $IS_k(is).\text{sup} += \log_{10} \sqrt{IS_{n\_k}(is).\text{sup}+1} / u$
11.         if $IS_k(is)$.sup < $min\_sup$
12.         remove $is$ from $IS_k$
13. Generate rules
```

Figure 3. Higher Order Apriori

The first step, *Enumpath*, discovers all the higher-order 2-recordsets within max_order by computing all the link groups between records. The pseudocode for *Enumpath* is shown in Figure 4. The first step in *Enumpath* uses an algorithm in [21] to find all simple paths (i.e., link groups) between two vertices. The worst case time complexity of this step is O($|V||E|$) for a given path where V is the set of vertices and E the edges in G [21].

The resulting link groups are 2-recordsets of different orders with different sizes. The order and size information is kept in RS_2.

```
Enumpath (G, max_order)
For each record pair rp in the graph
    Find all simple paths using the algorithm in [21]
For each path found
        order = number of records on the path
        size = size of the path
        If $RS_2(rp,order)$ is valid
                $RS_2(rp,order).size$ += size
        Else $RS_2(rp,order)=size$
```
Figure 4. *Enumpath*

Steps 2 through 12 of the *HO-Apriori* algorithm in Figure 3 comprise one outer and two inner loops. The outer loop proceeds level-wise and keeps track of the sizes of recordsets. Although the ($k+1$)-recordsets are generated in an Apriori-like fashion based on k-recordsets from the previous iteration, naturally no pruning is performed for recordsets. The first inner loop, from steps 3 through 6, proceeds order-first and generates different order itemsets from the k-recordsets, and calculates the corresponding support. Figure 6 depicts the level-wise order-first structure of our higher-order association rule mining algorithm.

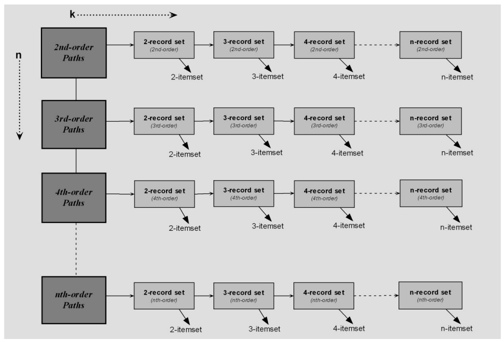

Figure 6. Level-wise Order-First Structure

In more detail, step 4 (Figure 5) generates the n^{th}*-order k-recordsets* based on the n^{th}*-order (k-1)-recordsets* using Apriori's candidate generation function. The size of the recordset is calculated based on the equation for $size_{n\text{-}k}(rs)$ in section 3.

```
Gen_RS ($RS_{n\_k}$)
$RS_{n\_k+1}$ = apriori_gen($RS_{n\_k}$)
foreach recordset $rs \in RS_{n\_k+1}$
        foreach rp in rs
                $rs.size \times= RS_2(rp,n).size$
```
Figure 5. Generate Recordset

For each n^{th}*-order k-recordset* generated, step 6 (Figure 7) enumerates all possible n^{th}*-order k-itemsets* from the recordset.

```
Enum_ IS (rs, n)
k = number of records in the rs
Pick one item from each record
If the items are different
        $IS_{n\_k}(is).sup$ += rs.size
```

Figure 7. Enumerate Itemsets from Recordset

Steps 8 to 10 calculate the global support for a single *k*-itemset across orders from 2 to max_order based on the support metric described in section 3. If the global support does not meet the threshold, the *k*-itemset is discarded in step 12. The frequent *k*-itemsets are kept in IS_k. The algorithm proceeds with the *(k+1)*-recordsets in the next iteration. After discovering all the frequent itemsets (both first and higher-order), association rules are generated in step 13 using a standard ARM algorithm such as Apriori.

6 Implementation

To verify the feasibility and effectiveness of Higher-Order Apriori, a prototype has been developed. The input to the prototype is provided through an XML file that describes the instances of sample data. In a pre-processing step, the input is transformed into an adjacency matrix. The core of the prototype lies in the Algorithm class which implements several task oriented, independent but related methods. The *run()* method of the Algorithm class invokes the *populate()* method of an instance of the RecordManager class to read in the input records and store them for further use. The higher-order paths are collected from UnoInterface ([21]) by invoking Enumpath method, passing one pair of records at a time. These paths are processed and objects of the PathEntry class are instantiated to represent higher-order paths between records along with other essential properties. An instance of the PathCollection class is responsible for sorting, storing and retrieving these paths during subsequent processing of the algorithm.

The *run()* method also employs two instances of the RecordSetCollectionKItem AllOrders class to store two consecutive columns of recordsets (depicted as columns in Figure 6 moving in the k direction). One instance stores objects of type RecordSetCollection to represent the k^{th} column of Figure 6, while the other is used to store the $(k-1)^{th}$ column, thus structuring the process level-wise. Instances of class RecordSetCollection are primarily used to achieve order-wise separation of the generation and storage of the RecordSet objects. Further more, each RecordSet objects holds an instance of the ItemSetCollection class to manage itemsets created using that particular record set. The Algorithm class has its own attribute of type ItemSetCollection to maintain all the itemsets being generated as the process advances level-wise. This ItemSetCollection object represents the result of the Higher Order Apriori algorithm described in the previous section, and all the itemsets in this collection satisfy the minimum support criteria.

The prototype was designed and documented using Rational Rose Enterprise Suite, and implemented in C++ using GNU C/C++ compiler version 3.2.2 to compile and link the code under Red Hat Linux 9.0. Adequate use of OO design principles during the development of this prototype will lead to low maintainability and modification costs. The code and documentation is available online at hddi.cse.lehigh.edu.

7 Results

We conducted experiments to evaluate Higher Order Apriori on multiple systems, including at the National Center for Supercomputing Applications (NCSA). To validate the algorithm we prepared a simulated e-Marketplace dataset of four records and five products: Microsoft Wheel Mouse Optical (b); Building a PC for Dummies, Fifth Edition (c); Microsoft Internet Keyboard (d); Build the Ultimate Custom PC (e) and AMD Athlon 64 X2 (f). For simplicity we will refer to these items by their letter designation (b, c, d, e and f). Table 2 depicts the outcome of executing Higher Order Apriori on this dataset. First, the transaction database (Table 2, left) is converted to a graph representation and passed to HO-Apriori. The link groups discovered by Enumpath are shown in Table 2 in the middle, and Table 2 (right) shows the top 10 itemsets with their local and global support. Seven of these itemsets cannot be discovered by Apriori. Based on these higher order itemsets, many interesting and novel rules can be formed. For example: *Mouse & Keyboard => Building a PC for Dummies*, *Fifth Edition* and *Mouse & Keyboard => AMD Athlon*. From these results we observe the following facts: (1) Higher Order Apriori discovers itemsets that cannot be discovered by Apriori; (2) Higher Order Apriori increases the strength of 1st-order k-itemsets. Higher Order Apriori also respects the well-known Apriori property, making it feasible to generate k-itemsets in the usual manner.

Table 2. Starting Record IDs, Link Groups and Itemsets Discovered by Higher Order Apriori

RID	Itemsets
1	b, c
2	c, d, e
3	c, f
4	d, e, f

Order	Paths (Link groups)
2nd	1-2; 1-3; 2-3;2-4; 3-4
3rd	1-3-2;1-2-3;1-2-4;1-3-4; 2-1-3;2-4-3;2-3-4;3-2-4;
4th	1-3-4-2;1-2-4-3;1-2-3-4; 1-3-2-4;2-1-3-4;3-1-2-4;

Itemsets	2nd	3rd	4th	Total
cdf	0.15	0.21	0.16	0.52
cef	0.15	0.21	0.16	0.52
cd	0.19	0.17	0.12	0.49
ce	0.19	0.17	0.12	0.49
cf	0.19	0.17	0.12	0.49
bcd	0.07	0.18	0.16	0.42
bce	0.07	0.18	0.16	0.42
bcf	0.07	0.18	0.16	0.42
bdf	0.07	0.18	0.16	0.42
bef	0.07	0.18	0.16	0.42

The next set of experiments involved the automatic generation of a test set. Based on a dataset containing 100 unique items we randomly generated records with an average of four items per record. Table 7 depicts examples of nth-order 3-itemsets generated from differing number of records (two through thirty-two). Items are represented by integer values. The first column depicts the number of records, and columns two through five portray examples of higher-order 3-itemsets discovered by Higher Order Apriori. Note that not all runs resulted in higher order itemsets – this is expected because, like Apriori, Higher Order Apriori is a datadriven rule mining algorithm. Table 8 gives a different perspective by portraying example k-itemsets for various orders.

Table 7. Example N-order 3-Itemsets, N=2,3,4,5

Records	2nd order	3rd order	4th order	5th order
2				

4	{13 26 2}
	{50 13 2}
		{13 26 37}
		{63 13 2}		

8				
16				
32	{0 1 29}	{21 16 20}	{36 0 9}	{21 50 0}

Table 8. Example N-order K-Itemsets, K=2,3,4

Records	2-itemsets	3-itemsets	4-itemsets
2			

4	{13 26}
	{50 2}
		{74 13 2}
		{89 26 37}	

8	{67 9}
	{24 11}		

16	{47 11}
	{18 23}		

32	{ 21 17}
	{36 32}
		{0 1 88 }
		{98 80 0 }	

Higher Order Apriori discovers itemsets that Apriori does not. For example, the 2nd-order 3-itemsets {13, 26, 2} and {0, 1, 29} do not occur in any single record in the input data. Likewise, the 2nd-order 2-itemset {50, 2} is a novel nugget. This demonstrates the capability of Higher Order Apriori not only to discover new knowledge, but also to enhance the support for existing 1st-order association rules. Although not depicted here, the rule generation process of Higher Order Apriori results in 1st and higher-order rules.

8 Conclusions and Future Work

We have developed a framework to extend association rule mining from 1st-order to all possible orders. We have defined and identified the context for higher-order itemsets. A mechanism to calculate support of higher-order itemsets was also presented. We have also designed, implemented and tested our higher-order association rule mining algorithm, which discovers propositional rules based on higher-order associations.

In future work we plan to address both theoretical and practical aspects of Higher Order Apriori. First, we will apply the algorithm to real world data including e-Marketplace, law enforcement and public healthcare data as part of our ongoing NSF-funded work in distributed higher-order ARM development. To evaluate Higher Order Apriori, we plan to develop methods for comparing high-confidence, high-support rules generated using Higher Order Apriori with the rules generated by Apriori. This should be very interesting. Second, we need to evaluate the support metric. The current support metric was developed empirically, and we need to further explore whether there is a theoretical basis for calculating support in this way, and also what other factors must be considered in calculating support. Third, we plan to incorporate Higher Order Apriori into our Text Mining Infrastructure (TMI) [10]. The TMI is an open-source framework designed for high-end, scalable text mining, and aims to provide a robust software core for research and development of text mining applications. Based on the TMI, Higher Order Apriori can be easily applied in a distributed environment. Finally, we plan to address issues related to scalability of the algorithm. Although the current algorithm rests on a firm foundation, much of value remains to be accomplished in terms of improving the performance.

Acknowledgements

The authors wish to thank Lehigh University, the Pennsylvania State Police, the Lockheed-Martin Corporation, the City of Bethlehem Police Department, the National Science Foundation and the National Institute of Justice, US Department of Justice. This work was supported in part by NSF grant number 0534276 and NIJ grant numbers 2005-93045-PA-IJ and 2005-93046-PA-IJ. Points of view in this document are those of the authors and do not necessarily represent the official position or policies of Lehigh University, the US Department of Justice, the National Science Foundation, the Pennsylvania State Police or the Lockheed Martin Corporation.

We are also grateful for the help of other co-workers, family members and friends. Co-authors Shenzhi Li, Mark J. Dilsizian, Christopher D. Janneck and William M. Pottenger also gratefully acknowledge the continuing help of their Lord and Savior, Yeshua the Messiah (Jesus the Christ) in our lives and work. Amen.

References

[1] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient Similarity Search In Sequence Databases. In D. Lomet, editor, Proceedings of the 4th International Conference of Foundations of Data Organization and Algorithms (FODO), pages 69–84, Chicago, Illinois, 1993. Springer Verlag.

[2] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, pages 207–216, Washington, D.C., 26–28 1993.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In P. S. Yu and A. S. P. Chen, editors, Eleventh International Conference on Data Engineering, pages 3–14, Taipei, Taiwan, 1995. IEEE Computer Society Press.

[4] K. Chan and A. Fu. Efficient Time-Series Matching by Wavelets. In Proc. of 1999 Int. Conf. on Data Engineering, Sydney, Australia, March, 1999.

[5] L. Dehaspe and L. D. Raedt. Mining association rules in multiple relations. In ILP ’97: Proceedings of the 7th International Workshop on Inductive Logic Programming, pages 125–132, London, UK, 1997. Springer-Verlag.

[6] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast Subsequence Matching in Time-Series Databases. In Proc. of the 1994 ACM SIGMOD Int. Conf. on Management of Data, Minneapolis, Minnesota, May, 1994.

[7] M. Ganiz, W. M. Pottenger and X. Yang. Link Analysis of Higher-Order Paths in Supervised Learning Datasets. In the *Proceedings of the Workshop on Link Analysis*, *Counterterrorism and Security*, *2006 SIAM Conference on Data Mining*. Bethesda, MD, April 2006.

[8] M. Ganiz, W. M. Pottenger and Janneck, C. D. Recent Advances in Literature Based Discovery. *Journal of the American Society for Information Science and Technology, JASIST* , 2006 (Submitted)

[9] R. V. Hauck, H. Atabakhsh, P. Ongvasith, H. Gupta, H. Chen, Using Coplink to analyze criminal-justice data, IEEE Computer 35 (3), 2002, pp. 30– 37.

[10] L.E. Holzman, T.A. Fisher, L.M. Galitsky, A. Kontostathis and W. M. Pottenger. A Software Infrastructure for Research in Textual Data Mining. *The International Journal on Artificial Intelligence Tools*, volume 14, number 4, pages 829-849. 2004.

[11] E. Keogh, K. Chakrabarti, M. Pazzani and S. Mehrotra. Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. Springer-Verlag, Knowledge and Information Systems, p. 263–286, 2001.

[12] A. Kontostathis and W.M. Pottenger, W. M. A Framework for Understanding LSI Performance. *Information Processing & Management*, 42(1), 2006.

[13] H. Mannila, H. Toivonen, and A.I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, vol. 1, no. 3, 259-289, 1997.

[14] R. J. Mooney, P. Melville, L.R. Tang, J. Shavlik, I.C. Dutra, D. Page and V.S. Costa. Relational Data Mining with Inductive Logic Programming for Link Discovery. Proceedings of the National Science Foundation Workshop on Next Generation Data Mining, Nov. 2002, Baltimore, MD.

[15] S. Nijssen and J. Kok. Faster Association Rules for Multiple Relations. In IJCAI01, Seattle, Washington, USA, pages 891—896, 2001.

[16] J. S. Park, M. Chen, and P.S. Yu, "An Effective Hash Based Algorithm for Mining Association Rules," Proc. ACM SIGMOD Conf., ACM Press, New York, 1995, pp. 175–186.

[17] A. Savasere, E. Omiecinski, and S. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases," Proc. 21st Int'l Conf. Very Large Databases., Morgan Kaufmann, San Francisco, 1995, pp. 432–444.

[18] D.R. Swanson. Complementary structures in disjoint science literatures. In A. Bookstein, Y. Chiaramella, G. Salton, & V.V. Raghavan (Eds.), *Proceedings of the 14th Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*, pp. 280–289, 1991. New York: ACM Press.

[19] P.N. Tan, V. Kumar, and J. Srivastava. Indirect association: Mining higher order dependencies in data. Technical Report TR00-037, University of Minnesota, 2000.

[20] Tan, Pang-Ning Tan and Vipin Kumar. Mining Association Patterns in Web Usage Data. University of Minnesota, 2002.

[21] T. UNO. An Output Linear Time Algorithm for Enumerating Chordless Cycles. *92nd SIGAL of Information Processing Society Japan*, 47-53, 2003.