Approximate Association Rule Mining

Jyothsna R. Nayak and Diane J. Cook
Department of Computer Science and Engineering
Arlington, TX 76019
cook@cse.uta.edu

Abstract

Association rule algorithms typically only identify patterns that occur in the original form throughout the database. In databases which contain many small variations in the data, potentially important discoveries may be ignored as a result. In this paper, we describe an associate rule mining algorithm that searches for approximate association rules. Our ~AR approach allows data that approximately matches the pattern to contribute toward the overall support of the pattern. This approach is also useful in processing missing data, which probabilistically contributes to the support of possibly matching patterns. Results of the ~AR algorithm are demonstrated using the Weka system and sample databases.

Contact Author:

Diane J. Cook
Department of Computer Science and Engineering
Box 19015
University of Texas at Arlington
Arlington, TX 76019
Office: (817) 272-3606
Fax: (817) 272-3784
Email: cook@cse.uta.edu

Keywords: knowledge discovery, association rules, inexact match, missing data

Abstract

Association rule algorithms typically only identify patterns that occur in the original form throughout the database. In databases which contain many small variations in the data, potentially important discoveries may be ignored as a result. In this paper, we describe an associate rule mining algorithm that searches for approximate association rules. Our ~AR approach allows data that approximately matches the pattern to contribute toward the overall support of the pattern. This approach is also useful in processing missing data, which probabilistically contributes to the support of possibly matching patterns. Results of the ~AR algorithm are demonstrated using the Weka system and sample databases.

Introduction

A number of data mining algorithms have been recently developed that greatly facilitate the processing and interpreting of large stores of data. One example is the association rule mining algorithm, which discovers correlations between items in transactional databases.

The Apriori algorithm is an example association rule mining algorithm. Using this algorithm, candidate patterns which receive sufficient support (occur sufficiently often) from the database are considered for transformation into a rule. This type of algorithm works well for complete data with discrete values.

One limitation of many association rule mining algorithms, such as the APriori algorithm [Agrawal 1990], is that only database entries which exactly match the candidate patterns may contribute to the support of the candidate pattern. This creates a problem for databases containing many small variations between otherwise similar patterns, and for databases containing missing values.

Missing and noisy data is prevalent in data gathered today, particularly in business databases. For example, U.S. census data reportedly contains up to 20% erroneous data. Important features may also be frequently missing from databases if collection was not designed with mining in mind.

The goal of this research is to develop an association rule algorithm that accepts partial support from data. By generating these "approximate" rules, data can contribute to the discovery despite the presence of noisy or missing values.

The approximate association rule algorithm, called ~AR, is built upon the Apriori algorithm and uses two main steps to handle missing and noisy data. First, missing values are replaced with a probability distribution over possible values represented by existing data. Second, all data contributes probabilistically to candidate patterns. Patterns which receive a sufficient amount of full or partial support are kept and expanded.

To demonstrate the capabilities of ~AR, we incorporate the algorithm into the Weka implementation of Apriori. Results are shown on several sample databases.

Association Rule Mining

A number of data mining algorithms have been introduced to the community that perform summarization of the data, classification of data with respect to a target attribute, deviation detection, and other forms of data characterization and interpretation. One popular summarization and pattern extraction algorithm is the association rule algorithm, which identifies correlations between items in transactional databases.

Given a set of transactions, each described by an unordered set of items, an association rule X Y may be discovered in the data, where X and Y are conjunctions of items. The intuitive meaning of such a rule is that transactions in the database which contain the items in X, tend to also contain the items in Y. An example of such a rule might be that many observed customers who purchase tires and auto accessories also buy some automotive services. In this case, X = {tires, auto accessories} and Y = {automotive services}.

Two numbers are associated with each rule, that indicate the support and confidence of the rule. The support of the rule X Y represents the percentage of transactions from the original database that contain both X and Y. The confidence of rule X Y represents the percentage of transactions containing items in X that also contain items in Y. Applications of association rule mining include cross marketing, attached mailing, catalog design and customer segmentation.

An association rule discovery algorithm searches the space of all possible patterns for rules that meet the user-specified support and confidence thresholds. One example of an association rule algorithm is the Apriori algorithm designed by Srikant and Agrawal [Srikant and Agrawal 1997]. The problem of discovering association rules can be divided into two steps:

1. Find all *itemsets* (sets of items appearing together in a transaction) whose support is greater than the specified threshold. Itemsets with minimum support are called *frequent itemsets*.
2. Generate association rules from the frequent itemsets. To do this, consider all partitionings of the itemset into rule left-hand and right-hand sides. Confidence of a candidate rule X Y is calculated as support(XY) / support(X). All rules that meet the confidence threshold are reported as discoveries of the algorithm.

```
$L_1$: = {frequent 1-itemsets};
k:= 2;          // k represents the pass number
While ($L_{k-1} \neq \emptyset$)
    $C_k$ = New candidates of size k
        generated from $L_{k-1}$
    For all transactions $t \in D$
        Increment count of all candidates in $C_k$
            that are contained in t
    $L_k$ = All candidates in $C_k$ with
            minimum support
    k = k+1
Report $\cup_k L_k$ as the discovered frequent itemsets
```

k-itemset	An itemset containing k items
L_k	Set of frequent k-itemsets (k-itemsets with minimum support)
C_k	Set of candidate k-itemsets (potentially frequent itemsets)
$\cup_k L_k$	Set of generated itemsets

Figure 1. The Apriori algorithm.

Figure 1 summarizes the Apriori algorithm. The first pass of the algorithm calculates single item frequencies to determine the frequent 1-itemsets. Each subsequent pass *k* discovers frequent itemsets of size *k*. To do this, the frequent itemsets L_{k-1} found in the previous iteration are joined to generate the candidate itemsets C_k. Next, the support for candidates in C_k is calculated through one sweep of the transaction list.

From L_{k-1}, the set of all frequent (k-1) itemsets, the set of candidate k-itemsets is created. The intuition behind this Apriori candidate generation procedure is that if an itemset X has minimum support, so do all the subsets of X. Thus new itemsets are created from (k-1) itemsets p and q by listing p.item1, p.item2, .., p.item(k-1), q.item(k-1). Items p and q are selected if items 1 through k-2 (ordered lexicographically) are equivalent for p and q, and item k-1 is not equivalent. Once candidates are generated, itemsets are removed from consideration if any (k-1) subset of the candidate is not in L_{k-1}.

Mining in the Presence of Missing Data

Mining in the presence of missing data is a common challenge. A variety of approaches exist to deal with missing data. By far the most common response is to omit cases with missing values. Because deleting data with missing values may waste valuable data points, missing values are often filled. Missing values may be replaced with a special symbol that the mining algorithm ignores. The missing value may also be induced using standard learning techniques, though these approaches yield the most successful results when only one attribute is missing [Lakshminarayanan 1996, Quinlan 1993, Ragel 1998]. Breiman uses a surrogate split to decide the missing value [Breiman 1983]. The surrogate value is the one with the highest correlation to the original value.

The most common approach is to impute missing values by globally replacing them with a single value such as the feature average before initiating the mining algorithm. The Weka system, on which we implement our ~AR algorithm, substitutes the mode of a nominal attribute for missing values throughout the database.

The ~AR Algorithm

Our approach to approximate association rule mining is embodied in the ~AR algorithm. The ~AR algorithm represents an enhancement of the Apriori algorithm included as part of the Weka suite of data mining tools [Weka]. The Weka algorithms, including the basic Apriori algorithm, are written in Java and include a uniform interface.

The first step of the ~AR algorithm is to impute missing values. Each missing value is replaced by a probability distribution. In order to adopt this approach, we make the assumption that fields are named or ordered consistently between data entries. This probability distribution represents the likelihood of possible values for the missing data, calculated using frequency counts from the entries that do contain data for the corresponding field.

For example, consider a database that contains the following transactions, where "?" represents a missing value.

- A, B, C
- E, F, E
- ?, B, E
- A, B, F

The missing value is replaced by a probability distribution calculated using the existing data. In this case, the probability that the value is "A" is P(A) = 0.67, and the probability that the value is "E" is P(E) = 0.33.

The second step of the ~AR algorithm is to discover the association rules. The main difference between ~AR and the Apriori algorithm is in the calculation of support for a candidate itemset. In the Apriori algorithm, a transaction supports a pattern if the transaction includes precise matches for all of the items in the candidate itemset.

In contrast, ~AR allows transactions to partially support a candidate pattern. Given *f* fields or items in the candidate itemset, each item in the database entry may contribute toward a total of *1/f* of the total support for the candidate itemset. If the database entry completely matches the corresponding item, support is incremented by 1/f. Thus if there is a complete match between the transaction and the candidate itemset, the support will be incremented by *f * 1/f* = 1, in a manner similar to the Apriori algorithm.

Two types of inexact match may occur. In the first case, the transaction entry exists but does not match the corresponding entry in the candidate itemset. In this case, the support is incremented according to the similarity of the two values. In this case of nominal attributes, the difference is maximal and support is not incremented. In the case of numeric values, the support is incremented by the absolute value of the difference between the values, divided by the maximum possible value for the given item.

Consider a candidate itemset containing four items:

- C = A, B, C, D

A database transaction may exist that fully matches the candidate itemset:

- T_1 = A, B, C, D

In this example, support for candidate C is incremented by ¼ + ¼ + ¼ + ¼ = 1

Using the second example, the transaction does not completely match the candidate itemset:

- T_2 = A, E, C, D

Support for candidate C is incremented based on transaction T_2 by ¼ + 0 + ¼ + ¼ = ¾.

The second type of inexact match considers a missing value which has been replaced by a probability distribution, and is considered for possible support of a candidate itemset. The support for the pattern is then incremented by 1/f * the probability that the missing value corresponds to the value in the candidate itemset. For example, if the transaction is:

- T_3 = A, B, ?, D and P(C = ½), P(E = ¼), and P(F = ½)

Support for candidate itemset C is incremented by ¼ + ¼ + (¼ * ½) + ¼ = 7/8.

One danger with this approach is that every transaction can potentially support every candidate itemset. To prevent transactions from supporting patterns that differ greatly from the transaction, a minimum match threshold is set by the user. If the support provided by any transaction falls below the threshold, then the transaction does not contribute any support to the candidate pattern.

```
FindSupport(C, D)
  support = 0
  for each transaction T ∈ D
    f = number of items in T
    for i = 1 to f
      if T[i] = "?"
        support = support + (1/f * P(D[i] = C[i]))
      else if D[i] ≠ C[i]
        support = support + (1/f * (|C[i] – D[i]|))
      else support = support + 1/f
    if support > MatchThreshold
      return support
    else return 0
```

Figure 2. FindSupport function.

The pseudocode for the FindSupport function is shown in Figure 2. This function determines the amount of support that exists in the database for a specific candidate itemset.

In the Weka implementation of the ~AR algorithm, a minimum support threshold of 100% is initially specified. Multiple iterations of the discovery algorithm are executed until at least N itemsets are discovered with the user-specified minimum confidence, or until the user-specified minimum support level is reached.

The ~AR algorithm is composed of 3 steps. First, all of the transactions are read from a database stored in the ARFF format. Second, itemsets are generated that meet the support and confidence thresholds. Finally, all possible rules are generated from the large itemsets.

Experimental Results

We demonstrate the ability of ~AR to discover approximate association rules in the presence of noisy and incomplete data. Specifically, we test the ~AR algorithm on a labor transactional database. The database is provided in the ARFF form as shown in Figure 3, in which each field describes a property of labor. This database describes final settlements in labor negotiations in Canadian industry. The database contains a large amount of missing data.

```
1,5,?,?,?,40,?,?,2,?,11,'average',?,?,
   'yes',?,'good'
2,4.5,5.8,?,?,35,'ret_allw',?,?,'yes',
   11,'below_average',?,'full',?,
   'full','good'
?,?,?,?,?,38,'empl_contr',?,5,?,11,
   'generous','yes','half','yes',
   'half','good'
3,3.7,4,5,'tc',?,?,?,?,'yes',?,?,?,?,
   'yes',?,'good'
3,4.5,4.5,5,?,40,?,?,?,?,12,'average',
   ?,'half','yes','half','good'
2,2,2.5,?,?,35,?,?,6,'yes',12,
   'average',?,?,?,?,'good'
3,4,5,5,'tc',?,'empl_contr',?,?,?,12,
   'generous','yes','none','yes',
   'half','good'
3,6.9,4.8,2.3,?,40,?,?,3,?,12,
   'below_average',?,?,?,?,'good'
2,3,7,?,?,38,?,12,25,'yes',11,
   'below_average','yes','half','yes',
   ?,'good'
1,5.7,?,?,'none',40,'empl_contr',?,4,?,
   11,'generous','yes','full',?,?,
   'good'
3,3.5,4,4.6,'none',36,?,?,3,?,13,
   'generous',?,?,'yes','full','good'
2,6.4,6.4,?,?,38,?,?,4,?,15,?,?,'full',
   ?,?,'good'
2,3.5,4,?,'none',40,?,?,2,'no',10,
   'below_average','no','half',?,
   'half','bad'
```

Figure 3. Portion of the labor database.

```
Best rules found:

 1. duration='(2.8-inf)' vacation=generous 3 ==> bereavement-assistance=yes class=good 3 (1)
 2. duration='(2.8-inf)' bereavement-assistance=yes 3 ==>vacation=generous class=good 3 (1)
 3. vacation=generous bereavement-assistance=yes 3 ==> duration='(2.8-inf)' class=good 3 (1)
 4. duration='(2.8-inf)' vacation=generous bereavement-assistance=yes 3 ==> class=good 3 (1)
 5. duration='(2.8-inf)' vacation=generous class=good 3 ==> bereavement-assistance=yes 3 (1)
 6. duration='(2.8-inf)' bereavement-assistance=yes class=good 3 ==> vacation=generous 3 (1)
 7. wage-increase-second-year='(3.85-4.3]' class=good 3 ==> bereavement-assistance=yes 3 (1)
 8. wage-increase-first-year='(3.47-3.96]' 3 ==> wage-increase-second-year='(3.85-4.3]' shift- differential='(-inf-4.3]' 3
(1)
 9. wage-increase-first-year='(3.47-3.96]' wage-increase-second-year='(3.85-4.3]' 3 ==> shift-differential='(-inf-4.3]' 3
(1)
10. wage-increase-first-year='(3.47-3.96]' shift-differential='(-inf-4.3]' 3 ==> wage-increase-second-year='(3.85-4.3]' 3
(1)
11. wage-increase-second-year='(3.85-4.3]' shift-differential='(-inf-4.3]' 3 ==> wage-increase-first-year='(3.47-3.96]' 3
(1)
12. duration='(2.8-inf)' shift-differential='(-inf-4.3]' 3 ==> class=good 3 (1)
```

Figure 4. Labor database rules discovered using the Apriori algorithm.

```
Best rules found:
 1. wage-increase-first-year='(3.47-3.96]' shift-differential='(-inf-4.3]' 4 ==> wage-increase-second- year='(3.85-4.3]' 4
(1)
 2. wage-increase-first-year='(3.47-3.96]' 3 ==> shift-differential='(-inf-4.3]' 3 (1)
 3. duration='(2.8-inf)' vacation=generous bereavement-assistance=yes 3 ==> class=good 3 (1)
 4. wage-increase-second-year='(3.85-4.3]' shift-differential='(-inf-4.3]' 4 ==> wage-increase-first-year='(3.47-3.96]' 4
(1)
 5. duration='(2.8-inf)' vacation=generous 3 ==> bereavement-assistance=yes 3 (1)
 6. vacation=generous bereavement-assistance=yes 3 ==> duration='(2.8-inf)' 3 (1)
 7. wage-increase-first-year='(3.47-3.96]' 4 ==> wage-increase-second-year='(3.85-4.3]' shift-differential='(-inf-4.3]' 4
(1)
 8. duration='(2.8-inf)' bereavement-assistance=yes 3 ==> vacation=generous 3 (1)
 9. duration='(2.8-inf)' shift-differential='(-inf-4.3]'vacation=generous 5 ==> class=good 5 (1)
10. bereavement-assistance=yes class=good 5 ==> wage-increase-second-year='(3.85-4.3]' 5 (1)
11. statutory-holidays='(11.4-12]' 4 ==> class=good 4 (1)
12. working-hours='(39.5-inf)' 4 ==> shift-differential='(-inf-4.3]' 4 (1)
```

Figure 5. Labor database rules discovered using the ~AR algorithm.

The Weka system generates association rules after imputing missing values in a preprocessing step. Missing values are generated using the method described earlier. We expect the number of rules generated using ~AR to increase over the Weka original implementation, because transactions that do not exactly match the candidate itemset can still contribute to the support of the pattern.

Figure 4 shows a sample of the generated rules using the original Apriori algorithm, and Figure 5 shows a sample of the generated rules used the ~AR algorithm. The support threshold is 0.4, and the confidence threshold is 0.9. The number preceding the “==>” symbol indicates the rule’s support. Following the rule is the number of those items for which the rule’s consequent holds as well. In parentheses is the confidence of the rule. The Weka implementation of the Apriori algorithm orders rules according to their confidence and uses support as a tiebreaker. Preceding the rules are the numbers of itemsets found for each support size considered.

The original Apriori database discovered 24 itemsets of size 1, 35 itemsets of size 2, 13 itemsets of size 3, and 1 itemset of size 4. Using ~AR and the probability distribution substitution for missing values, 24 itemsets were discovered of size 1, 35 itemsets were discovered of size 2, 24 itemsets were discovered of size 3, and 6 itemsets were discovered of size 4. These discovers represents an increase in the discovered itemsets and corresponding rules over the original Apriori algorithm.

The main reason for this difference is the increased support provided by transactions with missing values to a number of candidate itemsets that possibly match the transaction. We tested the results of the ~AR algorithm with increasing numbers of missing values (due to artificially replacing known values with a missing value symbol). As expected, the number of rules generated by ~AR decreases with the corresponding increase in missing values, because the transactions do not support the itemsets at the required level.

As a second experiment, a small database is used to compare the results of the ~AR with those of the original Apriori algorithm. The artificial grocery database contains transactions containing three items. The first item indicates the type of break purchased (either WHB, WB, or W). The second item indicates the type of cheese purchased (CC), and the third item indicates the type of milk that was purchased (WH). The database transactions are:

1. WHB, CC, WH
2. WHB, CC, WH
3. WB, CC, WH
4. WB, CC, WH
5. W, CC, WH

For this experiment, the support threshold is 0.4, the confidence threshold is 0.9, and the match threshold is 0.6 The original Apriori algorithm discovers 4 itemsets of size 1, 5 of size 2, and 2 of size 3. The ~AR algorithm discovers 5 itemsets of size 1, 7 of size 2, and 3 of size 3.

```
Best rules found:
 1. breadtype=WHB 2 ==> cheese=CC 2 (1)
 2. breadtype=WB 2 ==> cheese=CC 2 (1)
 3. breadtype=WHB 2 ==> milk=WH 2 (1)
4.breadtype=WB 2 ==> milk=WH 2 (1)
 5. milk=WH 5 ==> cheese=CC 5 (1)
 6. cheese=CC 5 ==> milk=WH 5 (1)
 7. cheese=CC milk=WH 5 ==> breadtype=WHB 5 (1)
 8. cheese=CC milk=WH 5 ==> breadtype=WB 5 (1)
```

Figure 6. Results of the grocery database.

Figure 6 summarizes the rules discovered by ~AR. Notice that rule 7, “cheese=CC and milk=WH implies breadtype=WHB”, received a support of 5 from the database. The database contains only two exact matches to this rule. However, transactions 3 through 5 match two of the three items in the candidate itemset and therefore contribute 2/3 support each to the itemset. As a result, the combined support is 2*1 + 3*2/3 = 5.

6 Conclusions

In this paper we introduce an enhancement to the Apriori association rule algorithm, called ~AR, that generates approximate association rules. The ~AR algorithm takes into consideration missing values and noisy data. Missing values are replaced by probability distributions over possible values for the missing feature, which allows the corresponding transaction to support all itemsets that could possibly match the data. Transactions which do not exactly match the candidate itemset may also contribute a partial amount of support, proportionate to the similarity between the transaction and the candidate itemset.

We demonstrate the effectiveness of the ~AR algorithm using sample databases. Results indicate that ~AR successfully generates rules that approximate true correlations in the input database. This behavior is beneficial for databases with many missing values or containing numeric data.

References

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C.J. 1983. *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA.

Ragel, A. 1998. Preprocessing of Missing Values Using Robust Association Rules. In *Proceedings of the Second Pacific-Asia Conference*.

http://www.cs.waikato.ac.nz/~ml/weka/.

Lakshminarayan, K., Harp, S., Goldman, R., and Samad, T. 1996. Imputation of missing data using machine learning techniques. In *Proceedings of the Second International Conference on Knowledge Discovery in Databases and Data Mining*.

Quinlan, J.R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Srikant, R. and Agrawal, R. 1997. Mining Generalized Association Rules. *Future Generation Computer Systems*, 13(2-3).