Weighted Association Rules: Model and Algorithm

G.D. Ramkumar, Sanjay Ranka, Shalom Tsur

October 16, 1997

1 Introduction

Methods for deriving large itemsets and their association rules from transactional data is a thriving research topic in data mining. Numerous methods have been proposed to improve the process: adapting it to parallel computing environments, taking advantage of the special properties of the underlying data, utilizing sampling methods, etc. However, often the result of such an analysis is a very large number of rules. It is not unusual to produce in this process hundreds or even thousands of rules and of course, the lower the confidence and support requirements, the larger the number of rules.

To the user, whose main interest is to gain a better understanding and insight into his/her problem domain represented by the data, this is a daunting prospect; deriving any conclusions from this wealth of knowledge is a very difficult task. Another difficulty with the present approach to association data mining is that it precludes the derivation of certain rules in which the constituent items have very different levels of support. This can be illustrated by the following example[1]:

> Caviar is an expensive and hence a low-support item in any supermarket basket. Vodka, on the other hand, is a high to medium-support item. The association: *caviar* $\Rightarrow$ *vodka* is of very high confidence but will never be derived by the existing methods since the itemset {vodka, caviar} is of low support and will not be included.

[1] We do not know who should be attributed for it; one of the authors was introduced to the example during one of the Stanford Data Mining seminars.

There is therefore a need to improve on the methods of mining derivation rules in at least two ways:

- The resulting knowledge needs to be presented in a more compact, more crystallized manner to be of interest to the user.

- Certain results that are precluded today, need to be presented.

In this paper we describe a few preliminary ideas, intended to overcome these problems. We propose alternative schemes to define association rules and analyze the algorithmic consequences of the new definitions. We argue that the new schemes are more practical and have direct applications in several problem domains. It may be harder to find association rules using the new scheme, but the ones derived are generally more useful.

The current work on association rules has focused on algorithms and analysis in the abstract without focusing on issues that arise when applying them to practical domains. These issues arise when defining the semantics of the association rules, while designing efficient algorithms to find them, as well as when implementing these algorithms.

In several problem domains it does not make sense to assign equal importance to all the items involved in the market basket analysis. For example, in the context of stock market prices—some stocks have much higher value than others and might appreciate a lot more than other stocks in a comparable time period. In the supermarket context, some items like jewelry, designer clothes, etc. are of much higher value than trivia like bubblegum or candy. Rules involving jewelry may have less support than those involving candy but are much more significant in terms of the revenue (and consequently profit) earned by the store.

We have developed a generalization of the itemset support problem which can incorporate item weights as well as transactions weights. However, the algorithm required to generate these "weighted" itemsets is more difficult than the DIC algorithm of [5] or the Apriori Algorithm [2, 4, 1, 3]. If only transaction weights are required for an application the solution is considerably simpler. In the following, we briefly describe some potential ways weights can be attached to items and/or transactions and explore the algorithms required to compute the weighted sets.

While we do not know of any related work that attempts to use the item weights directly in the derivation of association rules we mention the reasearch by Srikant and Agrawal [6] on the mining of quantitative association rules. The approach taken there is to conceptually regard each $(\mathrm{attribute}, \mathrm{value})$ pair as a distinct "item" and to employ clustering methods to control the very large number of rules generated thus. The result is that certain range information on quantitative attributes can be produced as the result; the method cannot however address the issue of itemsets that are valuable *in total* or the "Caviar and Vodka" problem mentioned earlier.

This paper is organized as follows: Section 2 includes the formal definition of the problem and an introduction to the notation used; Section 3 describes the algorithms used; Section 4 presents some experimental results, and we conclude the paper in Section 5.

2 Definitions and Formal Notation

We formalize the problem along the lines of [5]. Let $I = i_1, i_2, \ldots, i_n$ be the set of all distinct items. With each item i_j we associate a weight w_j. The weight is a positive real number that reflects the "significance" of the time relative to other items. We assume that the items $i_1 \ldots i_n$ are sorted by weight in that order, with i_1 and i_n having the minimum and maximum weights respectively. In addition, we can assign weights to transactions. Let T be the set of transactions. Any specific transaction $t \in T$ is associated with a weight $w(t)$.

In the unweighted setting, the support of an itemset was defined as the fraction of transactions that contain the itemset. The user defined a support threshold that specified the minimum support an itemset must have before it is deemed worthy of attention. It now remains to define the weighted support threshold in the general setting in which both items and transactions have weights. The derivation of rules from large weighted itemsets remains identical to the unweighted setting. For each large weighted itemset, every rule that has the requisite confidence is generated. As in the unweighted case, the confidence of a rule is the conditional probability that in a transaction if all the left items are present then the right items are also present.

Association Rule: An implication of the form,

A and B $\ldots \Rightarrow D$ and E $\ldots$

where $A, B, D, E, \ldots$ are (patient stay) attributes. This rule indicates that attributes $A, B, \ldots$ occurring together in a patient stay record *usually* imply that attributes $D, E, \ldots$ are also present in that record. For example,

Age is 70-80 and *DRG is Psychoses* ⇒ *Payor is Medicare*
is a sample rule.

Weighted support: Given a rule of the form,
A and B ... ⇒ D and E ...,
its weighted support is the fraction of the weight of patient stays that contain *all* of the attributes $A, B, D, E, \ldots$ to the weight of all patient stays. For example, the weighted support of the rule
Age is 70-80 and *DRG is Psychoses* ⇒ *Payor is Medicare*
is the fraction of weight of Medicare patient stays with DRG Psychoses and of age between 70-80 to the weight of all patient stays. If the weight corresponds to charge of a patient stay, then the weighted support is the fraction of total revenue of RUSH related to occurrence of attributes $A, B, D, E, \ldots$ in patient stay records.

The weighted support of an itemset can be defined as the product of the total weight of the itemset (sum of the weights of its items) and the fraction of transactions the itemset occurs in. The purpose of the association rule miner is then to determine all rules that are above a user specified minimum weighted support threshold having a minimum user specified confidence. Formally, let the weighted support of an itemset $J \subseteq I$, $J = j_1, \ldots, j_k$ be an itemset. The weighted support of J is denoted $w_s(J)$ and is defined as:

$$w_s(J) = \sum_{t \in T \wedge J \subseteq t} w(t) * \left(\sum_{j_l \in J} w(j_l) + \kappa\right); \tag{1}$$

where κ is a constant.

Notice that the above definition of weighted support is unnormalized. The normalized weighted support $w_{ns}(J)$ can be defined as:

$$w_{ns}(J) = \frac{w_s(J)}{\sum_{t \in T} w(t) * \left(\sum_{j \in t} w(j) + \kappa\right)} \tag{2}$$

where j ranges over each item contained in transaction t.

The user-specified weighted support threshold is denoted as ω. Typically, we are interested in itemsets with weighted support above ω. We use the convention that ω is unnormalized. Of course, the normalized weighted support ω_n is easily determined as:

$$\omega_n = \frac{\omega}{\sum_{t \in T} w(t) * \left(\sum_{j \in t} w(j) + \kappa\right)}$$

In general, we can apply a combination of weights to items and weights to transactions and obtain maximum flexibility in the determination of large itemsets and hence association rules. In the rest of this section we describe how to chose these weights for solving real world applications.

Item weights For example, in the supermarket setting the weight could be the profit made by the store by selling one unit of the item. In the health center setting, the weight of an item could be the cost normally associated with that specific procedure or symptom. If the weights are suitably assigned they provide us with a means of effectively filtering the rules.

Transaction weights In some contexts it may make sense to not only assign weights to items but also to transactions. We may be able to identify an automatic means of assigning weights to transactions either based on the purchaser or the set of items that were purchased.

In the context of medical center data the weight of a transaction (patient stay) may be assigned to the profit generated on a patient. This will help in identification of rules which generate profit either based on few patients with huge profit or a large number of patients each with small profits.

In the supermarket setting, the weight of a transaction could be:

1. the total profit made by the store on that transaction,
2. the "significance" of the customer who made the purchase. This may be a measure of whether or not he/she is a frequent customer at the store. It may be the case that a customer whose average store purchases are much higher than the store average should contribute more towards the itemset counts than most other customers.

Assigning weights to transactions allows us to additionally bias the rule-finding mechanism so that rules that apply to transactions of high importance are more likely to be found.

Weighted Support The use of weighter support would lead to useful mechanisms to prioritize the rules according to their importance, instead of their support and confidence alone. In fact, the thresholds defined by the user would be specified in terms of the margin of significance of cost. This margin is probably easier for the user to specify than the relatively arbitrary support threshold.

In order to see why $w_S(J)$ is a good measure, let us now look at the supermarket setting for a moment. Suppose we set the weight of each item to the profit it generates to the store. We claim that in this setting, when the weights of the transactions are all set to 1, the weighted support of an itemset J is the actual total profit made by the store due to the *joint* sale of the items of J (that is, the profit due to customer transactions that included the sale of all the items of J and possibly others). Clearly, the weighted support is a good indication of the level of significance of the itemset to store profits.

One advantage of using weighted support in this scenario is that the user can specify the weighted support threshold ! as a profit measure instead of an arbitrary fraction. For example, the user may state that he/she is only interested in itemsets by which the store generates a profit of at least $40,000. Setting the threshold by using this measure is more natural than using an arbitrary fraction of transactions which may result in a large number of relatively uninteresting itemsets.

Assigning weights to transactions allows us to additionally bias the rule-finding mechanism so that rules that apply to transactions of high importance are more likely to be found. Weighting of transactions can be used in cases where the store cares more/less for particular groups of customers or transactions. For example, the store may be interested in studying the patterns of behavior of their most valued returning customers. In this scenario, the user may set both the item weights as well as the transaction weights.

2.1 Special cases

Our generalized setting can be effectively used in several application scenarios. Setting all the item weights to zero, transaction weights to one and ˇ to one corresponds to the simple itemset support problem described in the literature. In the following we describe other settings which constrain the item weights and/or transaction weights and correspond to nontrivial extensions of the simple itemset support problem.

Zero Item weights and nonuniform transaction weights Assuming that ˇ is non-zero, this corresponds to assigning non-uniform weights to transactions only. An application of this setting may be generating all the itemsets which corresponds to frequent customers only. The weight of the transaction can be assigned to the frequency of the customer.

Uniform Transaction Weights Assuming that ˘ is zero, this corresponds to assigning weights to items only i.e. we are interested in itemsets with a weighted support above a threshold. This setting allows us to set variable size frequency levels for different itemsets.

Itemset with larger number of items are less frequent. A special case corresponds to setting all the item weights to unity. In this case, for a given itemset, the frequency should be inversely proportional to the number of items in the itemset.

3 Algorithm for Weighted Large Itemsets

In this section we look at methods to compute weighted large itemsets: that is, itemsets that have weighted support above a user specified minimum weighted support threshold !. Recall that the weighted support is unnormalized, unless specified otherwise.

3.1 Weighted Transactions

If all the item weights are assigned to zero and the transactions have arbitrary weights, we claim that traditional algorithms like Apriori and DIC can be used to find the weighted itemsets. The reasoning is as follows.

The critical assumption that is made in the Apriori and DIC algorithms is that if an itemset is large then all its subsets are also large. This allows the algorithms to build large itemsets of increasing size, by adding items to itemsets that are already discovered to be large. We now show that this assumption also holds in the case where each transaction is assigned a weight. If a given transaction tffl T contains an itemset $J \subset I$ then all subsets of the itemset J are also contained in the transaction t. Hence, if an itemset J is large then all its subsets $K \subseteq J$ are also large.

Hence, the algorithms of [5] and [2, 4, 1, 3] can be applied with minor modifications to obtain all the large weighted itemsets. The only change required is to update the support of an itemset using the transaction weight, as opposed to incrementing it by 1 as earlier.

The complexity issues, the running time improvements, and the scalability remains exactly the same as before.

3.2 Weighted Transactions *and* Items

In the case where transactions as well as items have weights, the situation turns out to be considerably more difficult. The assumption discussed in the preceding subsection does not hold anymore: that is, an itemset may be large (that is, have enough weighted support) even though some of its subsets are small.

For example, among supermarket purchases the itemset {toothpaste, candy} may be small but its superset caviar, toothpaste, candy} may be large, because the weight of caviar greatly increases the weighted support of the latter itemset. This suggests that in order to make sure that all the large itemsets are discovered, we may have to keep around itemsets that may be small but later may serve to "derive" large itemsets: that is, one of their future children (a superset) in the set lattice could be a large itemset.

In order to make sure that we keep around all the itemsets we need, we have worked out a scheme by which itemsets are assigned lower weighted support thresholds than the input threshold !. For an itemset J, the reduced weighted support threshold is denoted as $!_r(J)$; it is obtained by scaling ! with an appropriate fraction $Œ(J)$. That is, $!_r(J) = Œ(J) * !$. The factor $Œ(J)$ is assigned so that if any superset of J (that is, a derived itemset of greater cardinality yet to be discovered) has enough weighted support, then the weighted support of J is above $!_r(J)$ (that is, $Œ(J) * !$). The next subsection will discuss how the $Œ(J)$ factor is set.

We are now ready to describe the algorithm. The algorithm is closely based on the Apriori (or DIC) algorithms. Let L_k be the set of large k-itemsets. For example, L_3 might contain $\{\{1;3;5\};\{2;4;5\};\{2;5;6\};:::\}$. Let C_k be the set of *candidate* k-itemsets; this is always a superset of L_k. Here is a summary of the algorithm in the weighted setting:

Input : Database D, weighted support threshold !,
    Maximum itemset cardinality fi
Result $:= \emptyset$;
$k := 1$;
C_1 = set of all 1-itemsets;
while $C_k \neq \emptyset$ **do**
    create a weight counter for each itemset in C_k;
    forall transactions in database **do**
        Increment by transaction weight the weight counters of itemsets in C_k

which occur in the transaction;
L_k := All candidates J in C_k
such that $w_s(J) \geq !_r(J)$;
Result := Result $\cup L_k$;
C_{k+1} := all $k+1$-itemsets
which have all of their k-item subsets in L_k.
$k := k + 1$;

end

The algorithm described above is essentially similar to the Apriori algorithm, with the added specification of weights. Itemsets of cardinality k are derived from itemsets of cardinality $k-1$. Notice that the main difference is in the test of the support threshold: instead of checking the weighted support of J versus !, it is checked against $!_r(J)$, the reduced weighted support to enable finding future itemsets of higher cardinality that may be large. It remains to be explained how Œ(J) is determined; this will be the subject of the next subsection.

3.3 Three strategies

There seem to be three possible strategies to decide whether or not to count an itemset, and to decide on the reduced threshold to use for an itemset. They are:

1. The conservative strategy: set the reduced threshold of an itemset J by considering the largest weight "global" ancestor, i.e. the largest weight ancestor among all ancestors including the "indirect" ancestors which have items inserted into as well as appended to the list of items of J. To compute the reduced threshold, scale the user-defined threshold by the ratio of weights of the largest weight global ancestor and of J.

   In this method, we can be sure that all the descendants of a given node that is being considered have all been counted. This may increase the number of smaller cardinality itemsets but could serve to reduce (by filtering) the number of larger cardinality itemsets.

2. The aggressive strategy: set the reduced threshold of J by considering the largest weight "direct" ancestor, i.e. the largest weight ancestor

among direct ancestors which have items appended to the list of items of J; the items that are appended must have weights lower than that of the items of J. Again, to compute the reduced threshold using this technique, scale the user-defined threshold by the ratio of weights of the largest weight direct ancestor and of J.

In this method, we can only be sure that the direct descendants of a given node have been counted. This might increase the number of larger cardinality itemsets that are counted. As a specific example, if an itemset J has been generated then *all* of its direct parents of the type Ji_l may have to be generated where the weight of i_l is lower than the weight of each item of J. Since we only keep direct descendants of a given node, it maybe the case that only J (and its direct descendants) are kept among all the descendants of Ji_l. This could mean that potentially a large number of itemsets (almost as many as the number of items) are generated via a single itemset J.

3. The hybrid strategy: count and keep around *some* extra itemsets that may not be needed by the aggressive strategy. For example, we may choose to count all itemsets of cardinality 2. This will help cut down the number of items that are generated later. For example in considering an itemset Ji_l that could be generated via J, we are able to check that all 2-itemsets involving items of J and i_l have the required weight. This may be a helpful step in reducing the number of itemsets generated via a single itemset J.

   This method tries to derive the advantages of both the conservative and the aggressive strategy by attempting to reduce the number of itemsets that are generated of both smaller and larger cardinality. For a given itemset we can only be sure that its direct descendant has been counted, but often we will find that several other itemsets have been counted that help cut down the space of itemsets.

3.4 Setting the Œ(J) ratio

The objective of setting the Œ(J) ratio is to relax the restriction on the support threshold of an itemset J so that if any its derived item supersets is large then J passes the reduced weighted support threshold test. As before, suppose that the itemset J contains items $\{j_1; j_2; \ldots; j_k\}$.

In order to set $\phi(J)$ to a reasonable value we need the user to specify a parameter β, the maximum cardinality of large itemsets that he/she is interested in. This helps raise the ratio $\phi(J)$ to reasonable levels and thereby reduce the number of itemsets of small cardinality that exceed the reduced weighted support threshold $\omega_r(J)$ (and hence have to be computed).

For an itemset J the derived superset that has the maximum itemset weight with allowable maximum cardinality β is

$$J' = \{j_1, j_2, \ldots, j_k, i_{n-\beta+k+1}, i_{n-\beta+k+2}, \ldots, i_n\}$$

Recall that the set of items are sorted by weight in the order $i_1 \ldots i_n$, with i_n having the maximum weight. The worst-case possibility we have to worry about when deciding whether to keep J or not is the eventuality that every transaction that contains J also contains J' and that J' is large because it has high itemset weight. In this case, J' has weighted support ω iff J has weighted support:

$$\omega_r(J) = \omega * \frac{\sum_{i=1\ldots k} w(j_i)}{\sum_{i=1\ldots k} w(j_i) + \sum_{l=n-\beta+k+1\ldots n} w(i_l)}$$

The fraction that multiplies ω is the ratio of the total weight of items of J to that of J': the ratio by which the weighted support of J' is greater than that of J in the worst case (that is, if the set of transactions in which J' is present is the same as those in which J is present). Hence, $\phi(J)$ can be calculated based on the item weights of J and the rest of the items as:

$$\phi(J) = \frac{\sum_{i=1\ldots k} w(j_i)}{\sum_{i=1\ldots k} w(j_i) + \sum_{l=n-\beta+k+1\ldots n} w(i_l)}$$

We are looking at ways of optimizing the algorithm by determining possibly higher values of ϕ.

Performance Results

We implemented the WIS algorithm and the three strategies proposed in this paper. The platform for the implementation was a Hitachi SR4300 MPP parallel machine (almost identical to the IBM SP2 machine and environment) running AIX; the implementation was in C.

Based on experimental runs, we found that the Aggressive strategy almost uniformly performs better than the Conservative and Hybrid strategies. It counts fewer itemsets as well as takes less time than the other strategies. The WIS algorithm counts many more itemsets than Apriori for comparable values of weighted and unweighted supports. However, the key advantage of WIS is that the results from our algorithm come with a strong evidence of relevance to the user, since the user assigns weights to items based on their importance.

We verified experimentally that in the special case when all the item weights are 0, the algorithm runs in almost the same amount of time as Apriori. The only additional step is that of computing the reduced support thresholds; recall that in this case, all the reduced support thresholds are identical to the support threshold specified by the user.

In table **??** we show the running time of WIS under the three strategies for various input sizes. The weighted support was set to 0.5. We can see clearly that the Aggressive strategy out-performs the Conservative strategy. So we decide to focus on the aggressive strategy for further performance studies. Table **??** shows the performance of the aggressive strategy for various weighted support values when the input size is kept fixed. Table **??** considers the Aggressive strategy under an adaptation of the Dynamic Itemset Counting (DIC) method to WIS. We refer to this adaptation as WIS-DIC. As we can see from this table, the WIS-DIC performs better than WIS for certain input size and weighted support combinations. Since WIS-DIC devolves to WIS when the number of stops per pass is 1 (please see [] for a full discussion of DIC), in general WIS-DIC is faster than WIS and more powerful than DIC.

Conclusion

We presented a new model for market basket analysis that takes into account priorities of items; the priority is a weight assigned to an item based on, for example, its profit or cost attributes. Our model provides for costs assigned both to items as well as transactions in the market basket data. We presented the outline of an algorithm called WIS to compute itemsets that have sufficient weighted supports. These itemsets are used in turn to generate association rules that have high weighted support. This provides a good means of prioritizing the results of the association rule generator.

Acknowledgments

We would like to thank Vineet Singh, Anurag Srivastava, and Yoichi Shintani for useful discussions during the formative stages of the paper.

References

[1] A. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smith, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996.

[2] R. Agrawal, T. Imilienski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pages 207–216, May 1993.

[3] R. Agrawal and J.C. Shafer. Parallel mining of association rules: Design, implementation and experience. Technical Report RJ10004, IBM Research Division, Almaden Research Center, 1996.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th VLDB Conference, Santiago, Chile*, 1994.

[5] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proc. of 1997 ACM-SIGMOD Int. Conf. on Management of Data*, 1997.

[6] R. Srikant, and R. Argrawal Mining Quantitative Association Rules in Large Relational Tables In *Proceedings of the ACM SIGMOD Conference on Management of Data*, Montral, Canada, June 1996.