Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2007)

Extracting Borderline Associations

Wei Kian Chen, Dustin Baumgartner and Ryan Millikin
Department of Electrical & Computer Engineering and Computer Science
T.J. Smull College of Engineering
Ohio Northern University
Ada, Ohio 45810
Email: w-chen@onu.edu, d-baumgartner@onu.edu, r-millikin@onu.edu

***Abstract*— In this paper, we present an extension of the well known algorithm for association mining, Apriori. This extended algorithm, AprioriBL, considers associations between items which occur together – focusing solely on the borderline cases. These borderline cases occur often enough to provide valuable information; however, there are currently no algorithms that target them. We discuss how the AprioriBL algorithm works and present a comparative analysis of Apriori and AprioriBL.**

I. Introduction

Data mining is a technique developed to look into large datasets for implicit, previously unknown, and potentially useful information [4]. Association mining is one of the many data mining techniques where relationships within the data are the extracted information. Most association mining techniques focus on relationships that have a high frequency of occurrence. However, less obvious but valuable information may be lost along the borderline between frequent and infrequent relationships. For instance, frequent relationships among items at a grocery store may already be utilized, but the less frequent relationships can still be used to increase revenue. We developed an extension to the Apriori algorithm, Apriori Borderline, $AprioriBL$, which targets these relationships. In the following sections, we explain how the Apriori and AprioriBL algorithms work, along with an illustration of each. Next, sample applications of AprioriBL are given. Then, we introduce the testing environment and explain the tests that were run, followed by an analysis of two test cases that display the various observed trends in our results.

II. Background

In order to understand how to utilize association mining, the terminologies used for the process must first be understood. These terminologies are as follows [1]:

- $I = \{i_1, i_2, ..., i_n\}$ represents the set of all items that appear in the dataset.
- $T = \{t_1, t_2, ..., t_n\}$, which represents a set of transactions. Each transaction t is a set of items, such that $t \subseteq I$.
- D represents a dataset, which is the set of transactions, T.
- An itemset is a set of items occurring together on the same transaction. A k-itemset refers to a set with k items.
- S represents the support, which is the lowest percentage of transactions in the dataset on which an itemset must occur. The user-specified support value for the system is the minimum support threshold. Any itemset that satisfies the minimum support threshold is known as a frequent k-itemset.
- L_k represents a list, which is composed of frequent k-itemsets and their supports.
- C_k represents a candidate set, which is a list of k-itemsets that may exist in the dataset. These k-itemsets are derived from L_{k-1}.

A. Apriori

A classic algorithm commonly used for association mining is Apriori. Led by Rakesh Agrawal, Apriori was developed in 1993 at the IBM Almaden Research Center as a fast way to mine associations between items. This algorithm generates all possible frequent itemsets through multiple passes over the data. The first pass calculates the support for each individual item, which is used to determine frequent 1-itemsets, L_1. Subsequent passes start with L_k as a seed for generating new, potentially frequent candidate itemsets, C_{k+1}. Each candidate itemset is a unique combination of two itemsets in L_k. All k-subsets of an itemset in C_{k+1} must exist in L_k. The support for each candidate itemset is calculated during the next pass over the data, resulting in all frequent $k+1$-itemsets, L_{k+1}. This list is then used as a seed for the next generation of frequent-itemsets, which continues until no more candidate itemsets can be generated. Since each pass of the algorithm depends on the frequency at which itemsets occur, choosing a proper support value is essential. This support value varies according to the dataset [2].

B. Apriori Algorithm

In this section the pseudocode for the Apriori algorithm is given [2]. Initially, C_1 is populated with all unique 1-itemsets found by passing through the dataset. Itemsets in C_k are trimmed if they have a frequency less than S. This trimmed set contains the frequent itemsets, L_k, and is output. Before the next iteration, C_{k+1} is generated. The pseudocode of the algorithm is:

```
$C_1$ = {1-itemsets};
for ($k$ = 1; $C_k \neq \phi$; $k++$)
{
  $L_k$ = TrimUnsupported($C_k$);
  PrintApriori($L_k$);
  $C_{k+1}$ = AprioriGen($L_k$);
}
```

The $TrimUnsupported$ function loops through all the transactions in the dataset to count the frequency of the itemsets in C_k. When an itemset is found, its counter is incremented. After the traversal is complete, only those itemsets whose frequency is greater than or equal to S are added to L_k. The pseudocode for $TrimUnsupported$ is:

```
// Loop through transactions
foreach transaction $t \in D$
  // Loop through candidate itemsets
  foreach itemset $i \in C_k$
    if ($i \subseteq t$) then
      i.count++;
$L_k$ = {$i \in C_k$ | i.count $\geq$ $S$}
```

The $AprioriGen$ function checks each itemset in L_k for other itemsets that differ only by the last item. The itemsets found are joined together to form candidate itemsets. After the pass through L_k is finished, the possible candidate sets are pruned. The pseudocode for $AprioriGen$ is:

```
// Loop through seed itemsets
foreach itemset $i \in L_k$
  // Loop through subsequent seed itemsets
  foreach itemset $j \in L_k$ where
            i.item$_1$ = j.item$_1$, ...,
            i.item$_{k-2}$ = j.item$_{k-2}$,
            i.item$_{k-1}$ < j.item$_{k-1}$
    add itemset i.item$_1$, ..., i.item$_{k-1}$,
                j.item$_{k-1}$ to $C_{k+1}$;
PruneCombinations($C_{k+1}$);
```

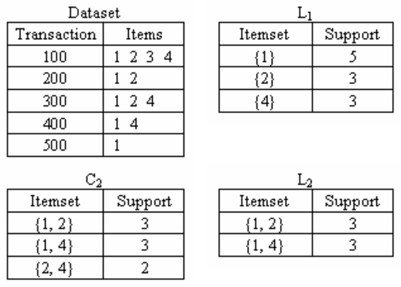
The $PruneCombinations$ function removes unfeasible itemsets from the possible candidates, C_{k+1}. An itemset is unfeasible when any of its k-subsets does not exist in L_k. If a subset is not found in L_k, then that subset has a frequency less than S. Therefore, any of its supersets will not have the proper support. The pseudocode for $PruneCombinations$ is:

```
// Loop through possible candidate itemsets
foreach itemset $i \in C_{k+1}$
  // Loop through subsets of the itemset
  foreach $k$-subset $s$ of $i$
    if $s \notin L_k$ then
      remove $i$ from $C_{k+1}$;
```

C. Apriori Illustration

Let us consider the following example in Fig. 1 to illustrate the Apriori algorithm. Assume there are four items available in a store: milk, bread, butter, and cheese. For simplicity of illustration, these items are represented numerically as 1, 2, 3, and 4, respectively. Assume that the dataset is composed of five transactions occurring over a short period of time. Also, let the minimum support threshold, S, be 60%. Since this is a rather small example, the minimum support is expressed as the exact count, which is 3.

All frequent 1-itemsets are generated based on the support value when Apriori makes its first pass over the dataset. All items, except for item 3, satisfy the minimum support threshold; therefore, L_1 is {{1}, {2}, {4}}. C_2 consists of all possible candidate 2-itemsets and their supports. None of the candidate 2-itemsets are removed because all the 1-subsets exist in L_1. However, the trimming of infrequent 2-itemsets removes the candidate 2-itemset {2, 4} based on its support. The result is L_2. The only possible candidate 3-itemset is {1, 2, 4}; however, the subset {2, 4} does not exist in L_2. Therefore, {1, 2, 4} is unfeasible and C_3 is not generated. Since $C_3 = \phi$, the algorithm terminates.



Dataset

Transaction	Items
100	1 2 3 4
200	1 2
300	1 2 4
400	1 4
500	1

L_1

Itemset	Support
{1}	5
{2}	3
{4}	3

C_2

Itemset	Support
{1, 2}	3
{1, 4}	3
{2, 4}	2

L_2

Itemset	Support
{1, 2}	3
{1, 4}	3

Fig. 1. Illustration of Apriori using five transactions.

III. AprioriBL

There are itemsets that occur at the borderline of the minimum support threshold. As subsequent iterations of Apriori are performed, these itemsets are unlikely to provide combinations that will have sufficient support for continued generation; however, they could still provide useful information. Borderline itemsets have various business applications in which revenue can be increased [3]. These itemsets are the focus of our algorithm, AprioriBL. In contrast to Apriori, only itemsets with frequency within a specified range are reported. The range, R, is defined as $S \pm \theta$, where θ is the maximum deviation from the user-specified support value. All k itemsets with a frequency $S - \theta$, the minimum support threshold for AprioriBL, or greater are placed in a seeding pool, P. P is used for the next candidate itemset generation. P is similar to the L of Apriori, except that its minimum support threshold is $S - \theta$ rather than S. Since each pass of the algorithm depends on the frequency at which itemsets occur, choosing a proper support and range is essential. These values vary according to the dataset.

A. AprioriBL Algorithm

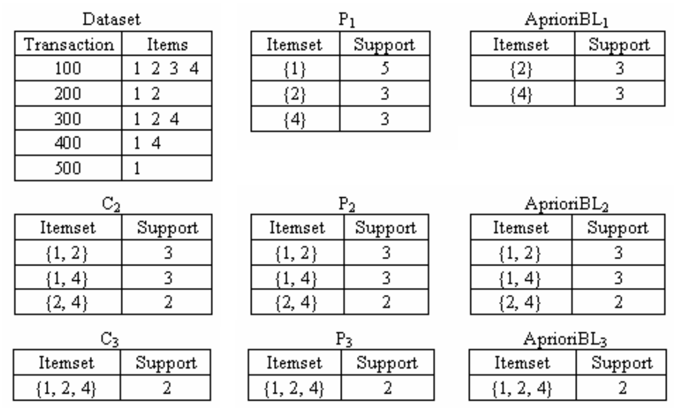
To implement AprioriBL, two changes to the Apriori algorithm provided in Section II-B are required. The first change converts $PrintApriori$ to $PrintAprioriBL$. Itemsets from P_k that occur between $S - \theta$ and $S + \theta$ times are output, rather than the itemsets which occur more often than or equal to S times. The second modification effects the $TrimUnsupported$ function. Once the frequency for all itemsets in C_k is obtained, only itemsets whose frequency is greater than or equal to $S - \theta$ are added to P_k. The resulting pseudocode for $TrimUnsupported$ is:

```
// Loop through transactions
foreach transaction $t$
  // Loop through candidate itemsets
  foreach itemset $i \in C_k$
    if ($i \subseteq t$) then
      i.count++;
$P_k$ = {$i \in C_k$ | i.count $\geq$ $S - \theta$}
```

B. AprioriBL Illustration

The same example dataset used to illustrate Apriori in Fig. 1 is used again to illustrate AprioriBL in Fig. 2. Let us assume that S = 60% (3) and θ = 20% (1).

On the first pass through the dataset, all 1-itemsets that satisfy the minimum support threshold, $S - \theta$, are generated and placed into P_1, {{1}, {2}, {4}}. All of these itemsets that fall within the range, 3±1, are reported as $AprioriBL_1$, {{2}, {4}}. The itemset



Dataset

Transaction	Items
100	1 2 3 4
200	1 2
300	1 2 4
400	1 4
500	1

P_1

Itemset	Support
{1}	5
{2}	3
{4}	3

$AprioriBL_1$

Itemset	Support
{2}	3
{4}	3

C_2

Itemset	Support
{1, 2}	3
{1, 4}	3
{2, 4}	2

P_2

Itemset	Support
{1, 2}	3
{1, 4}	3
{2, 4}	2

$AprioriBL_2$

Itemset	Support
{1, 2}	3
{1, 4}	3
{2, 4}	2

C_3

Itemset	Support
{1, 2, 4}	2

P_3

Itemset	Support
{1, 2, 4}	2

$AprioriBL_3$

Itemset	Support
{1, 2, 4}	2

Fig. 2. Illustration of AprioriBL with the dataset used for the Apriori.

$\{1\}$ does not appear in $AprioriBL_1$; however, it is used to seed C_2 because it is in P_1. Notice that none of the candidate 2-itemsets are removed because all the 1-subsets exist in P_1. All these itemsets satisfy the minimum support threshold and are within the range, so they appear in P_2 and $AprioriBL_2$. For C_3, the itemset {1, 2, 4} is generated because all of its 2-subsets exist in P_2. This itemset appears in both P_3 and $AprioriBL_3$ because it occurs as often as the minimum support threshold; however, the regular Apriori algorithm disregarded this potentially useful information. There is only one itemset in P_3, so C_4 cannot be generated and the algorithm is complete.

C. *Apriori vs. AprioriBL*

AprioriBL is an extension of Apriori that extracts borderline associations. Itemsets generated by AprioriBL can be equivalently obtained by performing exclusive disjunction on two iterations of Apriori using minimum support thresholds of $S - \theta$ and $S + \theta$. However, this approach is much more costly in terms of execution time. Not only is executing Apriori twice not desirable, but another algorithm would also have to be created to find the exclusive disjunction. Thus, the ability to extract the borderline associations with one run is a more reasonable approach.

IV. APPLICATIONS

Since AprioriBL provides a different view of a dataset than Apriori, it can be used to further improve the profit of a business. In the following sections, several applications of AprioriBL are discussed [3].

A. *Grocery Store*

A common application of association mining is evaluating the sales at a grocery store. Each customer purchases a set of items, which is added to the dataset as a transaction. Finding associations between items commonly purchased together would prove a valuable marketing tool for increasing sales. For example, let an itemset be {milk, bread, butter} and assume its support is 25%. These three items have a strong association; therefore, they can be used to increase revenue. Placing displays of bread and butter near milk or discounting one of the items may further increase the occurrence of this itemset, in effect increasing overall sales.

Typical association mining looks for itemsets that satisfy the minimum support threshold, most of which are known by an experienced grocer. However, the itemsets at the borderline may not be known but could still be useful. AprioriBL is designed in this manner, to offer alternatives that help a grocer improve sales. Another application closely related to this example is a virtual shopping cart used for e-commerce. When a customer adds an item to their cart, other associated items can be suggested for purchase. Targeting the pricing and marketing of borderline cases as an alternative to strictly highly associated itemsets will help to further increase sales.

B. *Risk Assessment*

Insurance companies use risk assessment for determining which customers they should provide coverage to and for calculating those customers' premiums. A lower risk customer will usually be charged a lower premium because they are less likely to file a claim. Car insurance customer characteristics such as age, driving history, and the type of car are associated with the amount of risk involved in providing coverage. Typical association mining can be used to target customers below a certain amount of risk, giving the insurance company favorable customers. If the borderline cases are considered, more precise characteristics can be obtained and the risk of customers can be better defined. This would allow an insurance company to improve their computational methods for premiums of customers with moderate risk who may not have normally been insured. With these additional customers, the company will be able to generate more revenue.

V. EXPERIMENT ENVIRONMENT

A. *Platform*

We developed the Apriori and AprioriBL algorithms using C++. The testing platform was a Linux environment with the 2.6.18-1.2200.fc5 kernel, a Pentium D 3.20 GHz dual core processor, and 4 GB of primary memory.

B. *Quest Data Generator*

The Quest data generator was used for creating test cases. Quest was developed by Agrawal and the IBM group and is a widely used and accepted tool for simulating real world datasets. Data generation is based on two phenomena that commonly occur in real datasets. First, different frequent itemsets often have items in common, which is caused by the existence of higher order itemsets that may or may not have proper support. Secondly, not all items from a frequent itemset will always occur together; for example, some people may only buy two items from a 3-itemset because they do not need the third item [2].

Data generation using Quest is based on several parameters, which are defined in Table I. The standard naming convention for Quest generated datasets is Tα.NβK.DγK.dat, where the values of β and γ are in thousands. For example, a dataset

TABLE I
DIFFERENT VARIABLES USED IN THE QUEST DATA GENERATOR.

Variable	Meaning	Default Values
T	Average number of items per transaction	10
N	Number of items	100
D	Number of transactions	1,000

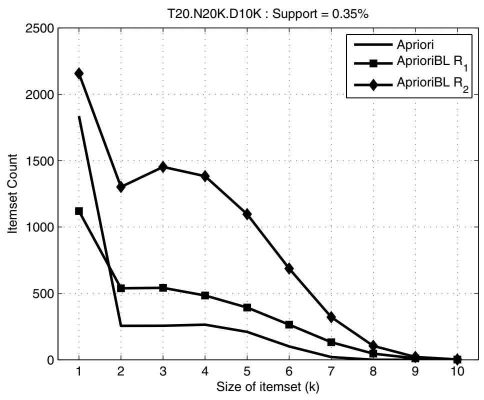


Fig. 3. Graphical illustrations of the results from Apriori and AprioriBL.

created with the default values listed in Table I would be named T10.N0.1K.D1K.dat.

VI. Test and Analysis

A. Datasets and Parameters

Three datasets were created to test the usefulness of the AprioriBL algorithm. All have a T value of 20, because grocery stores commonly use this number as a break point between express lanes and regular lanes. One dataset, T20.N10K.D10K, is dense such that the transaction to item ratio is one. The other datasets, T20.N10K.D5K and T20.N20K.D10K, are sparse with a lower transaction to item ratio. The Apriori algorithm processed these datasets with multiple support values. AprioriBL was executed using corresponding support values and various ranges. The following section provides an analysis of the T20.N20K.D10K dataset using a minimum support threshold S = 0.35% and maximum deviations θ_1 = 0.05% and θ_2 = 0.10%. This yields ranges of R_1 = 0.35% ± 0.05% and R_2 = 0.35% ± 0.10%, respectively. All the test cases show similar dynamics governing the change of itemset count as k increases; however, this particular set of cases displays the various observed trends.

B. Analysis

The graph in Fig. 3 depicts the results of Apriori and AprioriBL. The graph's independent axis displays the size of the itemsets, k, and the dependent axis displays the itemset count. There are two interesting points to note on the graph.

The first, most obvious, observation is the drastic drop in count from 1-itemsets to 2-itemsets. Many single items occur more frequently than the minimum support threshold; however, combinations of these, the 2-itemsets, have a much lower probability of occurring. Thus, many itemsets will not have sufficient support. In general, as k increases the itemset count decreases. However, the itemset counts actually increase for each case from 2-itemsets to 3-itemsets. This second observation is quite apparent in AprioriBL R_2. Here, the increase could be caused by 2-itemsets that were above the range producing 3-itemsets that fall into the range. AprioriBL R_1 did not show such a drastic increase, which may be caused by its smaller range. The 2-itemsets above R_1 could produce 3-itemsets which are within R_2 but below R_1. This "trickle down" effect can be used explain the moderate increase in the itemset counts for AprioriBL R_1 and the larger increase for AprioriBL R_2; however, this is not the case with Apriori. The subtle increase shown in Apriori can only be attributed to the combinatorial nature of the algorithm. Fig. 4 displays an example where 1-itemsets generate a larger number of 2-itemsets, assuming at least five of the 2-itemsets are properly supported.

$$\{\{1\},\{2\},\{3\},\{4\}\} \longrightarrow \{\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\}\}$$

Fig. 4. Four 1-itemsets can create six 2-itemsets if each has proper support.

TABLE II
Results of algorithms: $S = 0.35\%$ and $\theta = 0.05\%$ (AprioriBL R_1).

k-itemset	**Apriori**	**Intersection**	**AprioriBL**
1	1287	549	571
2	72	183	355
3	29	227	315
4	8	256	228
5	1	209	184
6	0	100	164
7	0	20	112
8	0	0	46
9	0	0	10
10	0	0	1

Another representation of the data is given in Table II for AprioriBL R_1 and Table III for AprioriBL R_2. The four columns on the tables are k-itemset, Apriori, Intersection, and AprioriBL. The k-itemset column contains the size of the itemsets for each row. The remaining columns contain the quantity of itemsets which belong to only Apriori, both Apriori and AprioriBL, and only AprioriBL, respectively. For example, of all the 3-itemsets in Table III, 8 are unique to Apriori, 248 satisfy both Apriori and AprioriBL, and 1205 are unique to AprioriBL.

This tabular view of the data gives an alternative perspective. Each column indicates a different set of characteristics for its itemsets, as shown in Table IV. These columns can be compared relative to each other. Apriori is frequent and has high support, the intersection is frequent but has moderate support, and AprioriBL is infrequent but has moderate support.

Table III shows that there are very few highly supported 2-itemsets and 3-itemsets that are uniquely generated by the Apriori algorithm. There are significantly more moderately supported itemsets from the intersection and AprioriBL. Furthermore, of these moderately supported itemsets, many more are infrequent

TABLE III
Results of algorithms: $S = 0.35\%$ and $\theta = 0.10\%$ (AprioriBL R_2).

k-itemset	**Apriori**	**Intersection**	**AprioriBL**
1	982	854	1303
2	30	225	1077
3	8	248	1205
4	0	264	1119
5	0	210	887
6	0	100	587
7	0	20	301
8	0	0	104
9	0	0	21
10	0	0	2

TABLE IV
CLASSIFICATION OF ITEMSETS BASED ON FREQUENCY.

Apriori	$S+\theta$ to 1
Intersection	S to $S+\theta$
AprioriBL	$S-\theta$ to S
Low Support	0 to $S-\theta$

than frequent. Although infrequent itemsets are not as tightly associated as frequent itemsets, this large increase allows more options and alternatives to be explored by users of the data. Since AprioriBL produces a large number of small itemsets, this allows for an increased possibility of higher order itemsets to be generated.

Table II shows that there are no frequent itemsets past k = 7, but there are moderately supported itemsets generated by AprioriBL up to the order of k = 10. These higher order itemsets show the relationship between the lower order itemsets more clearly. For example, at a grocery store a subset of 2-itemsets could be {milk, bread} and {bread, butter}, while a 3-itemset could be {milk, bread, butter}. Although the 3-itemset may not have a frequency above the minimum support threshold, it is still useful to know that this extended relationship exists.

VII. CONCLUSION

The Apriori algorithm generates all frequent itemsets that satisfy the minimum support threshold; however, itemsets with a frequency close to the minimum support threshold are generally not contained in higher order itemsets. AprioriBL targets these borderline itemsets – providing potentially advantageous information about the data that may have been overlooked. We explained how the Apriori and AprioriBL algorithms work, and offered several business related applications of AprioriBL. We ran tests and compared the results of Apriori and AprioriBL on the dataset T20.N20K.D10K. We discussed two possible reasons for the increase in itemset count as k increases. The "trickle down" effect can cause this increase in AprioriBL, though the combinatorial nature of the algorithms can cause this in both Apriori and AprioriBL. Finally, we discussed how infrequent borderline itemsets can yield useful information that is not given by the more frequent itemsets of Apriori.

VIII. FUTURE WORK

Our current implementation of AprioriBL is on the same order of efficiency as Apriori; however, in our future work we plan to significantly increase the performance. This will be done by trimming itemsets outside of the range $S \pm \theta$, which yields a seeding pool only containing those itemsets. We expect this to have a dramatic increase in performance at the cost of accuracy. We will complete a comparative analysis of Apriori, AprioriBL, and the new algorithm. Performance and accuracy will be measured relative to each algorithm, which will determine how useful each algorithm is in specific situations.

REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami, "Database mining: A performance perspective," *IEEE Transactions on Knowledge and Data Engineering, Special Issue on Learning and Discovery in Knowledge-Based Databases*, vol. 5, no. 6, pp. 914–925, December 1993. [Online]. Available: http://www.almaden.ibm.com/cs/people/ragrawal/pubs.html

[2] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Databases, VLDB '94*, J. B. Bocca, M. Jarke, and C. Zaniolo, Eds., Santiago, Chile, September 1994, pp. 487–499, expanded version available as IBM Research Report RJ 9389, June 1994. [Online]. Available: http://www.almaden.ibm.com/cs/people/ragrawal/pubs.html

[3] W. K. Chen, D. Baumgartner, and R. Millikin, "Improving business applications with borderline associations," in *Proceedings of the International Conference for Business IT*, Kuala Lumpur Malaysia, August 2006.

[4] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus, "Knowledge discovery in databases: An overview," in *Knowledge Discovery in Databases*, G. Piatetsky-Shapiro and W. J. Frawley, Eds. Menlo Park, California: AAAI Press, 1991, pp. 1–27.