Frequent Item Set Mining and Association Rules

Arno Siebes and Ad Feelders

October 18, 2020

1 Introduction

One of the important categories of data mining problems is that of *associations* between attributes. This gives useful insight for such diverse business problems as *product cross-selling*, *website perception*, and *decision problems*.

There are two ways to look at attribute associations. The first is on the attribute-level, i.e., one looks for statistical dependencies between the attributes. Graphical models are a tool for such associations. The second way is at the value-level. Association rules [2] are the premier tool for this class of problems.

Informally, association rules can be seen as a kind of if-then rules: if a person buys a newspaper, he or she also buys chocolate. The twist association rules bring to classical if-then rules is a *conditional probability*. If a person buys a newspaper, there is a probability that he or she will also buy chocolate. This conditional probability is known as *confidence* in the literature.

Another measure that is generally associated with association rules is that of *support*: the fraction of customers for whom the rule holds, or rather the relative number of customers buying all items occurring in the rule (the so-called underlying itemset). If there is just one customer that buys newspapers and he or she also happens to buy chocolate, the association rule is not very interesting.

Next to being an interestingness measure, the support of a rule also plays a key role in the standard algorithms for association rule discovery. Given thresholds *minsup* and *minconf* for the support and confidence, these algorithms compute all association rules whose support and confidence exceed these thresholds. Itemsets with support at least *minsup* are called frequent.

Originally, association rules were introduced in a binary, non-temporal setting. For example, one considered a collection of transactions at the check-out of a store only recording whether a certain item was bought or not. Later, many extensions have been introduced, e.g., sequences (time), hierarchical clusters of items, and item-counts. Also more complex patterns then itemsets, for example trees and graphs, have been considered extensively.

2 Association Rules: The Binary Case

The traditional setting for association rules [3] consists of a set of transactions in which each transaction is a set of items. Translated to a relational setting, we have a table *db* with schema $R = \{I_1, \ldots, I_n\}$, in which each I_i is a binary attribute. The attributes correspond with the items, the rows in the table with the transactions; a row has value 1 for an attribute if and only if the transaction contains that item.

For $X, Y \subseteq R$, with $X \cap Y = \varnothing$ let:

- $s(X)$ denote the *support* of X, i.e., the number of tuples that have value 1 for all items in X. Sometimes support is expressed as a fraction of the total database size.
- for an *association rule* $X \to Y$, define
  - the support is $s(X \cup Y)$
  - the confidence is $s(X \cup Y)/s(X)$

The problem is to find all association rules that match or exceed the user defined lower thresholds for confidence, *minconf,* and support *minsup.* In a major abuse of notation, you could say that the support of X (when expressed as a fraction) corresonds to $P(X)$ in probability terms, and the confidence of $X \to Y$ with the conditional probability $P(Y \mid X)$.

There are two thresholds we have to satisfy, so ([3]):

1. find all sets Z whose support exceeds the minimal threshold. These sets are called frequent (or large) sets.
2. then test for all non-empty subsets X of frequent sets Z whether the rule $X \to Z \setminus X (= Y)$ holds with sufficient confidence.

2.1 The A Priori Algorithm

The first problem is then: how do we find the frequent sets? In principle, we have to check all subsets of R. However, this is not possible, since

$$|\mathcal{P}(R)| = 2^{|R|}.$$

For example, if we can check 1024 sets/sec. then:

- For 10 items, we are done in 1 second;
- for 20 items, we need 1024 seconds, or 17 minutes
- For 100 items, we need (roughly) 4×10^{18} years, which (far) exceeds the age of the universe!

Luckily, we have the following observation ([3]):

Z can only be frequent if all its (non-empty) subsets are frequent.

This is known as the *A-Priori property*. In other words, we can search *level wise* for the frequent sets. The level is the number of items in the set. Denote by $C(i)$ the sets of i items that are potentially frequent (the candidate sets) and by $F(i)$ the frequent sets of i items.

```
**Find frequent sets**
$C(1) := R$
$i := 1$
**While** $C(i) \neq \varnothing$ **do**
    $F(i) := \varnothing$
    **For** each $X \in C(i)$ **do**
        **If** $s(X) \geq minsup$ **then** $F(i) := F(i) \cup \{X\}$
    $i := i + 1$
    $C(i) := \varnothing$
    **For** each $X \in F(i-1)$ **do**
        **For** each $Y \in F(i-1)$ that shares $i-2$ items with $X$ **do**
            **If** All $Z \subset X \cup Y$ of $i-1$ items are frequent **then**
                $C(i) := C(i) \cup \{X \cup Y\}$
```

The algorithm fragment:

```
**For** each $X \in F(i-1)$ **do**
    **For** each $Y \in F(i-1)$ that shares $i-2$ items with $X$ **do**
```

Could lead to multiple generations of the set $X \cup Y$, which means unnescessary work! To avoid this, one places an (arbitrary) order on the items and replaces the above fragment by

```
**For** each $X \in F(i-1)$ **do**
    **For** each $Y \in F(i-1)$ that shares *the first* $i-2$ items with $X$ **do**
```

Let's look at an example ([6]). *Minsup* equals 2 and the data is given by:

tid	I_1	I_2	I_3	I_4	I_5
1	1	1	0	0	1
2	0	1	0	1	0
3	0	1	1	0	0
4	1	1	0	1	0
5	1	0	1	0	0
6	0	1	1	0	0
7	1	0	1	0	0
8	1	1	1	0	1
9	1	1	1	0	0

On the first level we find:

Itemset	Support	Check
$\{I_1\}$	6	Ok
$\{I_2\}$	7	Ok
$\{I_3\}$	6	Ok
$\{I_4\}$	2	Ok
$\{I_5\}$	2	Ok

Since all 1-itemsets are frequent, all possible 2-itemsets are candidates at the second level:

Itemset	Support	Check
$\{I_1, I_2\}$	4	Ok
$\{I_1, I_3\}$	4	Ok
$\{I_1, I_4\}$	1	No
$\{I_1, I_5\}$	2	Ok
$\{I_2, I_3\}$	4	Ok
$\{I_2, I_4\}$	2	Ok
$\{I_2, I_5\}$	2	Ok
$\{I_3, I_4\}$	0	No
$\{I_3, I_5\}$	1	No
$\{I_4, I_5\}$	0	No

The third level yields:

Itemset	Support	Check
$\{I_1, I_2, I_3\}$	2	Ok
$\{I_1, I_2, I_5\}$	2	Ok

For example, $\{I_1, I_2, I_3\}$ is a candidate at level 3, because all its level 2 subsets are frequent. The candidate is generated only once: by combining $\{I_1, I_2\}$ and $\{I_1, I_3\}$. The itemset $\{I_2, I_4, I_5\}$ is generated as a "pre-candidate" by combining $\{I_2, I_4\}$ and $\{I_2, I_5\}$, but then it is pruned because its level 2 subset $\{I_4, I_5\}$ is not frequent. Clearly there are no sets that qualify for the fourth level and we are done.

We rejected the naive algorithm because its complexity was $O(2^{|R|})$. So, what is the complexity of level wise search? Take a database with just 1 tuple consisting completely of 1's and set *minsup* to 1. Then, all subsets of R are frequent. Hence, the worst case complexity of level wise search is $O(2^{|R|})$.

However, if we *assume* that *db* is *sparse* (by far the most values are 0), then we *expect* that the frequent sets have a maximal size k with $k << |R|$. If that expectation is met, we have a worst case complexity of:

$$O\left(\sum_{j=1}^{k+1} \binom{|R|}{j}\right) = O(|R|^{k+1}) << O(2^{|R|})$$

This expression is derived as follows: in the worst case everything up to level k

is frequent. This means that at level $j \leq k+1$ we have to consider

$$\binom{|R|}{j}$$

candidates. The total number of candidates is of course given by the sum over all levels.

Generating association rules from the frequent sets is done as follows ([3]):

```
Generate Association Rules
For each frequent set $X$ do
    For all non-empty $Y \subset X$ do
        If $s(X)/s(X \setminus Y) \geq minconf$ then
            Output $X \setminus Y \rightarrow Y$
```

Let's continue our example. One of the frequent sets is $\{I_1, I_2, I_5\}$. This generates:

Itemset	Rule	Confidence
$\{I_1, I_2\}$	$\{I_1, I_2\} \rightarrow I_5$	$2/4 = 50\%$
$\{I_1, I_5\}$	$\{I_1, I_5\} \rightarrow I_2$	$2/2 = 100\%$
$\{I_2, I_5\}$	$\{I_2, I_5\} \rightarrow I_1$	$2/2 = 100\%$
$\{I_1\}$	$I_1 \rightarrow \{I_2, I_5\}$	$2/6 = 33\%$
$\{I_2\}$	$I_2 \rightarrow \{I_1, I_5\}$	$2/7 = 29\%$
$\{I_5\}$	$I_5 \rightarrow \{I_1, I_2\}$	$2/2 = 100\%$

Clearly, this algorithms is again exponential. For every X, we consider all $2^{|X|} - 1$ non-empty subsets Y of X. However, as long as $|X| \leq k << |R|$, this is not necessarily a problem. Quite often one generates only those association rules with a singleton Y, which makes the generation algorithm linear.

3 Drowning in Association Rules

In practice, association rules suffer from an embarrassment of richness: one often gets too many results. The number of association rules one discovers is inversely related to both *minsup* and *minconf*. If one sets these thresholds (too) high, one only discovers already well-known associations. If one lowers the thresholds the number of discovered rules grows dramatically. Getting more results than tuples in the database is not unheard of! Recall our earlier example in which the frequent itemset $\{I_1, I_2, I_5\}$ generated 6 association rules.

If all discovered rules would be interesting, the fact that one gets many would be a — perhaps unfortunate — fact of life. However, many of the rules convey little or no useful information. Suppose you discover that 60% of the people that buy bread also buy cheese. How interesting is this if you know that 60% of all people buy cheese?

There are two approaches to this flood of results, i.e., pre-computing and post-processing. Pre-computing means that we trie to generate less rules. Post-processing means that the resulting set of rules is filtered or ordered such that the user only has to consider the more interesting rules. Historically post-processing was considered first, therefore we start with this approach.

3.1 Post-processing the Resultset

If there are so many rules, it may be a good idea to order the results. We can use for example confidence and support; they define a *partial order* on the set of rules. Given rules r_1 and r_2, $r_1 <_{sc} r_2$ iff

1. $s(r_1) \leq s(r_2) \wedge \text{conf}(r_1) < \text{conf}(r_2)$ or
2. $s(r_1) < s(r_2) \wedge \text{conf}(r_1) \leq \text{conf}(r_2)$

Additionally, $r_1 =_{sc} r_2$ iff $s(r_1) = s(r_2)$ and $\text{conf}(r_1) = \text{conf}(r_2)$.
Using this partial order we can present the rules, e.g., as follows:

- order the rules on support, from high to low
- per fixed support level, order the rules on confidence, again from high to low.

Note, we can do this interactively, allowing the user to play with support and confidence levels, presenting only those rules that meet the currently set levels.

We could also present the rules as follows:

- Order the rules by consequent
- per consequent order the rules by confidence and support

While this presentation creates some order in the chaos of rules, they do not solve the problem of uninteresting results. Is there anything we can do?

In principle, a rule is interesting if it gives useful (actionable) information. But is that something you can decide syntactically? Lots of different measures [5] have been defined as an attempt to do just that, e.g., lift, interest, conviction, collective strength, gain, gini, entropy, χ^2, ... We'll discuss one of these, the lift. The *lift* of an association rule is defined (in terms of probabilities, and, once more, in a major abuse of notation) as

$$\text{lift}(A \to C) = \frac{P(C|A)}{P(C)} = \frac{P(A,C)}{P(A)P(C)}$$

In other words, if a rule has a confidence $P(C|A)$ of 0.9 while $P(C) = 0.2$, then the lift of the rule is 4.5. Hence, the lift measures whether people that buy A actually have an increased probability of buying C. The lift should be bigger than one for a rule to be interesting.

3.2 Generating Less Rules

Rather than post-processing the rules, one can also try to generate just the more interesting rules. One of the approaches is that one puts extra constraints on the frequent sets. Two important concepts in this field are the *maximal* frequent itemsets and the *closed* frequent itemsets. The underlying idea of both concepts is that the set of all maximal/closed frequent itemsets represent all frequent itemsets but is far smaller than the set of all frequent itemsets. Both are an example of what is also known as a *condensed* representation.

Maximal frequent itemsets are frequent itemsets that have no frequent supersets. Clearly, each frequent itemset is a subset of a maximal frequent itemset. MaxMiner [4] is an algorithm that directly mines the maximal frequent itemsets from the database. This is especially useful if one expects large frequent itemsets. Mining for just the maximal frequent itemsets is cheaper than the Apriori algorithm in that case.

Closed frequent itemsets are itemsets that completely characterise their associated set of transactions. That is, a frequent itemset I is closed if I contains all items that occur in all transactions in the support of I. We discuss the A-Close algorithm for mining closed frequent itemsets [7] in some more detail.

For an itemset I, denote by $\sigma(I)$ the set of tuples in which all items in I are bought, that is,

$$\sigma(I) = \{t \in db \mid \forall i \in I, i \in t\}.$$

For a set of transactions T let $f(T)$ denote the set of items that are bought in all transactions in T, that is,

$$f(T) = \{i \in R \mid \forall t \in T, i \in t\}$$

The closure of an itemset is obtained by first applying σ and then f:

$$c(I) = f(\sigma(I)).$$

Hence, $c(I)$ is the set of items that are bought in all transactions in which all items in I are bought. To give a procedural description of applying the closure operator to an itemset I: first get the transactions in which all items in I are bought, and then see whether there are any more items that are common to all these transactions. If so, add them to I and return the result. Clearly, $I \subseteq c(I)$ and I has the same support as $c(I)$. An itemset I is closed if and only if $c(I) = I$. An itemset I is a closed frequent itemset iff it is both frequent and closed.

For example, the closed frequent itemsets for

Transaction	Items
1	A, C, D
2	B, C, E
3	A, B, C, E
4	B, E
5	A, B, C, E

with minimum support 2/5 are

$$\{C\}, \{A, C\}, \{B, E\}, \{B, C, E\}, \{A, B, C, E\}$$

The A-Close Algorithm for finding all frequent closed itemsets consist of 2 phases:

Phase 1: Discover all frequent closed itemsets in *db*.

Phase 2: Derive all frequent itemsets from the frequent closed itemsets found in phase 1.

In phase 1 the algorithm determines a set of so-called *generators* that will produce all frequent closed itemsets by application of the closure operator c. An itemset I is a *generator* of a closed itemset J if it is one of the smallest itemsets with $c(I) = J$. For example: BC is a generator of the closed itemset BCE because

$$c(\{B, C\}) = f(\sigma(\{B, C\})) = f(\{2, 3, 5\}) = \{B, C, E\},$$

and there is no smaller set with closure $\{B, C, E\}$. $\{C, E\}$ is another generator of $\{B, C, E\}$.

A-Close performs a levelwise search like Apriori: G_{i+1} (the set of generators at level $i + 1$) is constructed using G_i. Using their support, and the support of their i-subsets in G_i,

- infrequent itemsets, and
- itemsets that have the same support as one of their subsets

are deleted from G_{i+1}. The rationale of the second pruning rule is that if an itemset has a subset with the same support, then this subset also has the same closure.

Let's apply A-close to our example, to see how it works.

Candidate	Support
A	3
B	4
C	4
D	1
E	4

$\Longrightarrow$

Generator	Support
A	3
B	4
C	4
E	4

The first step is no different then Apriori. Now we find the level 2 generators:

Candidate	Support
AB	2
AC	3
AE	2
BC	3
BE	4
CE	3

$\Longrightarrow$

Generator	Support
AB	2
AE	2
BC	3
CE	3

AC is pruned, because subset A has the same support (and therefore the same closure). BE is pruned for the same reason. Level 3 candidate ABE cannot be a generator, because one of its subsets (BE) is not a level 2 generator. The correctness of this last step may not be immediately obvious. BE is not a level 2 generator because subset B has the same support as BE. It follows that AB must have the same support as ABE, and hence ABE is not a generator.

Now that the generators have been determined, we compute their closures, where

$$c(I) = \cap t \in db : I \subseteq t$$

Different generators may produce the same closure; in the table on the right the duplicates have been romoved:

Generator	Closure	Support
A	AC	3
B	BE	4
C	C	4
E	BE	4
AB	ABCE	2
AE	ABCE	2
BC	BCE	3
CE	BCE	3

$\Longrightarrow$

Closure	Support
AC	3
BE	4
C	4
ABCE	2
BCE	3

In phase 2 we determine all frequent itemsets and their support. We use the following properties:

1. All maximal frequent itemsets are closed.
2. Every frequent itemset is a subset of a maximal frequent itemset.
3. The support of an itemset equals the support of the smallest closed itemset in which it is contained (its closure).

Our strategy is therefore to select the maximal itemsets, generate all their subsets, and determine their support using the third property.

To continue our example, $\{A, B, C, E\}$ is the only maximal frequent itemset. Hence, we generate all its subsets and compute their support. In the table below the closed sets themselves have been left out:

Subset	Support
A	3
B	4
E	4
AB	2
AE	2
BC	3
CE	3
ABC	2
ABE	2
ACE	2

The performance of A-Close is comparable to that of Apriori on sparse, weakly correlated data (e.g. supermarket basket data). In that case almost all itemsets are closed, and A-Close can perform very little extra pruning compared to Apriori. For dense, strongly correlated data, the difference between the number of frequent itemsets, and the number of *closed* frequent itemsets will be larger, and therefore A-Close will outperform Apriori.

4 Conclusions

Association rules are a powerful tool in the toolbox of the data miner. Association rules can be computed cheaply and provide useful insight in the data at hand. Although one tends to generate too many association rules, there are many useful techniques to filter the more interesting results from this flood. The concept of a frequent set and its monotonicity (the Apriori property) has been generalized and adapted to many other cases. One can easily fill a book on this topic; see [1, 6].

References

[1] J-M. Adamo. *Data Mining for Association Rules and Sequential Patterns: Sequential and Parallel Algorithms.* Springer Verlag, 2001.

[2] R. Agrawal, T. Imilinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. SIGMOD 1993.* ACM Press, 1993.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th VLDB.* Morgan Kaufmann, 1994.

[4] Bayardo. Efficiently mining long patterns from data. In *Proc. SIGMOD 1998.* ACM Press, 1998.

[5] Bayardo and Agrawal. Mining the most interesting rules. In *Proc. 5th KDD.* ACM Press, 1999.

[6] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufman, 2001.

[7] Pasquier, Bastide, Taouil, and Lakhal. Discovering frequent closed itemsets for association rules. In *Proc. 7th International Conference on Database Theory (ICDT).* Springer Verlag, 1999.