Cover Similarity Based Item Set Mining

Marc Segond and Christian Borgelt

European Centre for Soft Computing,
Calle Gonzalo Gutiérrez Quirós s/n, E-33600 Mieres (Asturias), Spain
{marc.segond,christian.borgelt}@softcomputing.es

Abstract. In standard frequent item set mining one tries to find item sets the support of which exceeds a user-specified threshold (minimum support) in a database of transactions. We, instead, strive to find item sets for which the similarity of the covers of the items (that is, the sets of transactions containing the items) exceeds a user-defined threshold. This approach yields a much better assessment of the association strength of the items, because it takes additional information about their occurrences into account. Starting from the generalized Jaccard index we extend our approach to a total of twelve specific similarity measures and a generalized form. In addition, standard frequent item set mining turns out to be a special case of this flexible framework. We present an efficient mining algorithm that is inspired by the well-known Eclat algorithm and its improvements. By reporting experiments on several benchmark data sets we demonstrate that the runtime penalty incurred by the more complex (but also more informative) item set assessment is bearable and that the approach yields high quality and more useful item sets.

1 Introduction

Frequent item set mining and association rule induction are among the most intensely studied topics in data mining and knowledge discovery in databases. The enormous research efforts devoted to these tasks have led to a variety of sophisticated and efficient algorithms, among the best-known of which are Apriori [1,2], Eclat [38,39] and FP-growth [19,16,17].

Unfortunately, a standard problem in this research area is that the output (that is, the set of reported item sets or association rules) is often huge and can easily exceed the size of the transaction database to mine. As a consequence, the (usually few) interesting item sets and rules drown in a sea of irrelevant ones. One of the reasons for this is that the support measure for item sets and the confidence measure for rules are not very informative, because they do not say that much about the actual strength of association of the items in the set or rule: a set of items may be frequent simply because its elements are frequent and thus their frequent co-occurrence can even be expected by chance. In association rule induction adding an item to the antecedent may be possible without affecting the confidence much, because the association is actually brought about by the other items in the antecedent. Therefore a considerable number of redundant and/or irrelevant item sets and rules is often produced.

M.R. Berthold (Ed.): Bisociative Knowledge Discovery, LNAI 7250, pp. 104–121, 2012.

Approaches to cope with this problem include, for instance, [36,37], which rely on subsequent filtering and statistical tests in order to single out the relevant rules and patterns. In this chapter, however, we pursue a different direction, namely changing the search criterion for item sets, so that fewer irrelevant item sets are produced in the first place. The core idea is to replace the support measure with a more expressive measure that better captures whether the items in a set are associated. To obtain such a measure we draw on the insight that for associated items their covers—that is, the sets of transactions containing them—are more similar than for independent items. Since the Jaccard index is a very natural and straightforward measure for the similarity of sets, this leads us to the definition of a Jaccard item set, which is an item set for which the generalized Jaccard index of the covers of its items exceeds a user-specified threshold. This index has the advantage that it is also anti-monotone, so that the same search and pruning techniques can be employed as in frequent item set mining.

We then extend our approach to a total of twelve specific similarity measures that can be generalized from pairs of sets (or, equivalently, binary vectors). We present a generalized form, from which all of these measures can be obtained by proper parameterization, but which also allows for other options. Finally, it turns out that standard frequent item set mining is a special case of this flexible framework, which, however, also offers several better alternatives.

The rest of this chapter is organized as follows: in Section 2 we briefly review frequent item set mining and a core search procedure and introduce our notation. In Section 3 we present the generalized Jaccard index with the help of which we then define Jaccard item sets. Section 4 reviews the Eclat algorithm, the processing scheme of which we employ in the search for Jaccard item sets. In Section 5 we show how the difference set idea for Eclat can be adapted to efficiently compute the value of the denominator of the generalized Jaccard index, thus completing our JIM algorithm (for Jaccard Item set Mining). In Section 6 we consider a total of twelve specific similarity measures that can be used in place of the Jaccard index, together with a generalized form. In Section 7 we apply our algorithm to standard benchmark data sets and to the 2008/2009 Wikipedia Selection for schools to demonstrate the speed and usefulness of our algorithm. Finally, in Section 8, we draw conclusions from our discussion.

2 Frequent Item Set Mining

Frequent item set mining is a data analysis method that was originally developed for market basket analysis. It aims mainly at finding regularities in the shopping behavior of the customers of supermarkets, mail-order companies, online shops etc. In particular, it tries to identify sets of products (or generally items) that are associated or frequently bought together. Once identified, such sets of associated products may be used to optimize the organization of the offered products on the shelves of a supermarket or the pages of a mail-order catalog or web shop. They can also give hints which products may conveniently be bundled or may be suggested to a new customer, or to a current customer after a purchase.

Formally, the task of frequent item set mining can be described as follows: we are given a set B of *items*, called the *item base*, and a database T of *transactions*. Each item represents a product, and the item base represents the set of all products on offer. The term *item set* refers to any subset of the item base B. Each transaction is an item set and represents a set of products that has been bought by an actual customer. Since two or even more customers may have bought the exact same set of products, the total of all transactions must be represented as a vector, a bag, or a multiset, since in a simple set each transaction could occur at most once.[1] Note that the item base B is usually not given explicitly, but only implicitly as the union of all transactions in the given database.

We write $T = (t_1, \ldots, t_n)$ for a transaction database with n transactions. Thus we are able to distinguishing equal transactions by their position in the database vector (that is, the transaction index is an implicit identifier). In order to conveniently refer to the index set of the transactions, we introduce the abbreviation $\mathbb{N}_n := \{k \in \mathbb{N} \mid k \leq n\} = \{1, \ldots, n\}$. Given an item set $I \subseteq B$ and a transaction database T, the *cover* $K_T(I)$ of I w.r.t. T is defined as $K_T(I) = \{k \in \mathbb{N}_n \mid I \subseteq t_k\}$, that is, as the set of indices of transactions that contain I. The *support* $s_T(I)$ of an item set $I \subseteq B$ is the number of transactions in the database T it is contained in, that is, $s_T(I) = |K_T(I)|$. Given a user-specified *minimum support* $s_{\min} \in \mathbb{N}$, an item set I is called *frequent* in T iff $s_T(I) \geq s_{\min}$. The goal of frequent item set mining is to identify all item sets $I \subseteq B$ that are frequent in a given transaction database T. Note that the task of frequent item set mining may also be defined with a *relative* minimum support, which is the fraction of transactions in T that must contain an item set I in order to make I frequent. This alternative definition is obviously equivalent.

A standard approach to find all frequent item sets w.r.t. a given database T and a minimum support $s_{\min}$, which is adopted by basically all frequent item set mining algorithms (except those of the Apriori family), is a *depth-first search* in the subset lattice of the item base B. Viewed properly, this approach can be seen as a simple *divide-and-conquer* scheme. For some chosen item i, the problem to find all frequent item sets is split into two subproblems: (1) find all frequent item sets containing the item i and (2) find all frequent item sets *not* containing the item i. Each subproblem is then further divided based on another item $j \neq i$: find all frequent item sets containing (1.1) both items i and j, (1.2) item i, but not j, (2.1) item j, but not i, (2.2) neither item i nor j etc.

All subproblems that occur in this divide-and-conquer recursion can be defined by a *conditional transaction database* and a *prefix*. The prefix is a set of items that has to be added to all frequent item sets that are discovered in the conditional database, from which all items in the prefix have been removed. Formally, all subproblems are tuples $S = (T_C, P)$, where T_C is a conditional transaction database and $P \subseteq B$ is a prefix. The initial problem, with which the recursion is started, is $S = (T, \emptyset)$, where T is the given transaction database to mine and the prefix is empty. A subproblem $S_0 = (T_0, P_0)$ is processed as follows: Choose an

[1] Alternatively, each transaction may be enhanced by a unique *transaction identifier*, and these enhanced transactions may then be combined in a simple set.

item $i \in B_0$, where B_0 is the set of items occurring in T_0. This choice is arbitrary, but usually follows some predefined order of the items. A common choice is to process the items in the order of increasing frequency in the transaction database to mine, as this often leads to the shortest search times. If $s_{T_0}(i) \geq s_{\min}$, then report the item set $P_0 \cup \{i\}$ as frequent with the support $s_{T_0}(i)$, and form the subproblem $S_1 = (T_1, P_1)$ with $P_1 = P_0 \cup \{i\}$. The conditional transaction database T_1 comprises all transactions in T_0 that contain the item i, but with the item i removed. This also implies that transactions that contain no other item than i are entirely removed: no empty transactions are ever kept. If T_1 is not empty, process S_1 recursively. In any case (that is, regardless of whether $s_{T_0}(i) \geq s_{\min}$ or not), form the subproblem $S_2 = (T_2, P_2)$, where $P_2 = P_0$ and the conditional transaction database T_2 comprises all transactions in T_0 (including those that do not contain the item i), but again with the item i removed. If T_2 is not empty, process S_2 recursively.

Eclat, FP-growth, and several other frequent item set mining algorithms all follow this basic recursive processing scheme [15,5]. They differ mainly in how they represent the conditional transaction databases. There are basically two fundamental approaches, namely horizontal and vertical representations. In a horizontal representation, the database is stored as a list (or array) of transactions, each of which is a list (or array) of the items contained in it. In a vertical representation, a transaction database is stored by first referring with a list (or array) to the different items. For each item a list of transaction identifiers is stored, which indicate the transactions that contain the item.

However, this distinction is not pure, since there are many algorithms that use a combination of the two forms of representing a database. For example, while Eclat [38,39] uses a purely vertical representation and SaM (Split and Merge) [6] uses a purely horizontal representation, FP-growth [19,16,17] combines in its FP-tree structure a (compressed) horizontal representation (prefix tree of transactions) and a vertical representation (links between the tree branches).[2]

The basic processing scheme outlined above can easily be improved with so-called *perfect extension pruning*, which relies on the following simple idea: given an item set I, an item $i \notin I$ is called a *perfect extension* of I, iff I and $I \cup \{i\}$ have the same support, that is, if i is contained in all transactions containing I. Perfect extensions have the following obvious properties: (1) if the item i is a perfect extension of an item set I, then it is also a perfect extension of any item set $J \supseteq I$ as long as $i \notin J$ and (2) if I is a frequent item set and K is the set of all perfect extensions of I, then all sets $I \cup J$ with $J \in 2^K$ (where 2^K denotes the power set of K) are also frequent and have the same support as I.

These properties can be exploited by collecting in the recursion not only prefix items, but also, in a third element of a subproblem description, perfect extension items. Once identified, perfect extension items are no longer processed in the recursion, but are only used to generate all supersets of the prefix that have the

[2] Note that Apriori, which also uses a purely horizontal representation, is not mentioned here, because it relies on a different processing scheme: it traverses the subset lattice level-wise rather than depth-first.

same support. Depending on the data set, this method, which is also known as *hypercube decomposition* [34,35], can lead to a considerable acceleration of the search. It should be clear that this optimization can, in principle, be applied in all frequent item set mining algorithms.[3]

3 Jaccard Item Sets

As outlined in the introduction, we base our item set mining approach on the similarity of item covers rather than on item set support. In order to measure the similarity of a set of item covers, we start from the Jaccard index [22], which is a well-known statistic for comparing sets. For two arbitrary sets A and B it is defined as

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

Obviously, $J(A,B)$ is 1 if the sets coincide (i.e. $A = B$) and 0 if they are disjoint (i.e. $A \cap B = \emptyset$). For overlapping sets its value lies between 0 and 1.

The core idea of using the Jaccard index for item set mining lies in the insight that the covers of (positively) associated items are likely to have a high Jaccard index, while a low Jaccard index rather indicates independent or even negatively associated items. However, since we consider also item sets with more than two items, we need a generalization to more than two sets (here: item covers). In order to achieve this, we define, in a perfectly straightforward manner, the *carrier* $L_T(I)$ of an item set I w.r.t. a transaction database T as

$$L_T(I) = \{k \in \mathbb{N}_n \mid I \cap t_k \neq \emptyset\} = \{k \in \mathbb{N}_n \mid \exists i \in I : i \in t_k\} = \bigcup_{i \in I} K_T(\{i\}).$$

The *extent* $r_T(I)$ of an item set I w.r.t. a transaction database T is the size of its carrier, that is, $r_T(I) = |L_T(I)|$. Recall also that, in analogy, the *cover* $K_T(I)$ of an item set I w.r.t. a transaction database T is

$$K_T(I) = \{k \in \mathbb{N}_n \mid I \subseteq t_k\} = \{k \in \mathbb{N}_n \mid \forall i \in I : i \in t_k\} = \bigcap_{i \in I} K_T(\{i\})$$

and that the *support* $s_T(I)$ of an item set I is the size of this cover, that is, $s_T(I) = |K_T(I)|$. With these two notions we can simply define the generalized Jaccard index of an item set I w.r.t. a transaction database T as its support divided by its extent, that is, as

$$J_T(I) \;=\; \frac{s_T(I)}{r_T(I)} \;=\; \frac{|K_T(I)|}{|L_T(I)|} \;=\; \frac{|\bigcap_{i \in I} K_T(\{i\})|}{|\bigcup_{i \in I} K_T(\{i\})|}.$$

[3] Note that perfect extension pruning is *not* the same as restricting the output to *closed* frequent item sets [26], even though a closed item set can be defined as an item set that does not possess a perfect extension. The reason is that the search, in order to avoid redundant work, usually does not consider all possible extensions. Hence there may be perfect extensions which are not detected in the search.

Clearly, this is a very natural and straightforward generalization of the Jaccard index. Since for an arbitrary item $a \in B$ it is obviously $K_T(I \cup \{a\}) \subseteq K_T(I)$ and equally obviously $L_T(I \cup \{a\}) \supseteq L_T(I)$, we have $s_T(I \cup \{a\}) \leq s_T(I)$ and $r_T(I \cup \{a\}) \geq r_T(I)$. From these two relations it follows

$$J_T(I \cup \{a\}) \leq J_T(I).$$

Therefore the generalized Jaccard index w.r.t. a transaction database T over an item base B is an anti-monotone function on the partially ordered set $(2^B, \subseteq)$.

Given a user-specified minimum Jaccard value $J_{\min}$, an item set I is called *Jaccard-frequent* if $J_T(I) \geq J_{\min}$. The goal of Jaccard item set mining is to identify all item sets that are Jaccard-frequent in a given transaction database T. Since the generalized Jaccard index is anti-monotone, this task can be addressed with the same basic scheme as the task of frequent item set mining. The only problem to be solved is to find an efficient scheme for computing the extent $r_T(I)$.

4 The Eclat Algorithm

Since we will draw on the scheme of the well-known Eclat algorithm for mining Jaccard item sets, we briefly review some of its core ideas in this section. As already mentioned, Eclat [38] uses a purely vertical representation of conditional transaction databases. That is, it uses lists of transaction indices, which represent the cover of an item or an item set. It then exploits the obvious relation

$$K_T(I_1 \cup I_2) = K_T(I_1) \cap K_T(I_2),$$

which can easily be verified by inserting the definition of a cover. In particular, Eclat exploits the special case

$$K_T(I \cup \{a, b\}) = K_T(I \cup \{a\}) \cap K_T(I \cup \{b\}),$$

which allows to extend an item set by an item. This is used in the recursive divide-and-conquer scheme described above by intersecting the list of transaction indices associated with the split item with the lists of transaction indices of all items that have not yet been considered in the recursion. In this case the set I in the formula above is the prefix P of the conditional transaction database.

An alternative to the intersection approach, which is particularly useful for mining dense transaction databases[4], relies on so-called *difference sets* (or *diffsets* for short) [39]. The diffset $D_T(a \mid I)$ of an item a w.r.t. an item set I and a transaction database T is defined as

$$D_T(a \mid I) = K_T(I) - K_T(I \cup \{a\}).$$

[4] A transaction database is called *dense* if the average fraction of all items that occur per transaction is relatively high. Formally, we may define the density of a transaction database T as $\delta(T) = \frac{1}{n \cdot |B|} \sum_{k=1}^{n} |t_k|$, which is equivalent to the fraction of ones in a binary matrix representation of the transaction database T.

That is, a diffset $D_T(a \mid I)$ lists the indices of all transactions that contain I, but not a. Since obviously

$$s_T(I \cup \{a\}) = s_T(I) - |D_T(a \mid I)|,$$

diffsets are equally effective for finding frequent item sets, provided one can derive a formula that allows to compute diffsets with a larger conditional item set I without going through covers (using the above definition of a diffset). However, this is easily achieved, because the following equality holds [39]:

$$D_T(b \mid I \cup \{a\}) = D_T(b \mid I) - D_T(a \mid I).$$

This formula allows to formulate the search entirely with the help of diffsets. It may be started either with the complements of the covers of the items, which are the diffsets for an empty condition, or by forming the differences of the covers of individual items to obtain the diffsets for condition sets with only a single item.

5 The JIM Algorithm (Jaccard Item Set Mining)

The diffset approach as it was reviewed in the previous section can easily be transferred in order to find an efficient scheme for computing the *carrier* and thus the *extent* of item sets. To this end we define the *extra set* $E_T(a \mid I)$ as

$$E_T(a \mid I) = K_T(\{a\}) - \bigcup_{i \in I} K_T(\{i\}) = \{k \in \mathbb{N}_n \mid a \in t_k \wedge \forall i \in I : i \notin t_k\}.$$

That is, $E_T(a \mid I)$ is the set of indices of all transactions that contain a, but no item in I. Thus it identifies the extra transaction indices that have to be added to the carrier if item a is added to the item set I. For extra sets we have

$$E_T(a \mid I \cup \{b\}) = E_T(a \mid I) - E_T(b \mid I),$$

which corresponds to the analogous formula for diffsets reviewed above. This relation is easily verified as follows:

$$\begin{aligned}
&E_T(a \mid I) - E_T(b \mid I) \\
&= \{k \in \mathbb{N}_n \mid a \in t_k \wedge \forall i \in I : i \notin t_k\} - \{k \in \mathbb{N}_n \mid b \in t_k \wedge \forall i \in I : i \notin t_k\} \\
&= \{k \in \mathbb{N}_n \mid a \in t_k \wedge \forall i \in I : i \notin t_k \wedge \neg(b \in t_k \wedge \forall i \in I : i \notin t_k)\} \\
&= \{k \in \mathbb{N}_n \mid a \in t_k \wedge \forall i \in I : i \notin t_k \wedge (b \notin t_k \vee \exists i \in I : i \in t_k)\} \\
&= \{k \in \mathbb{N}_n \mid (a \in t_k \wedge \forall i \in I : i \notin t_k \wedge b \notin t_k) \\
&\qquad \vee \underbrace{(a \in t_k \wedge \forall i \in I : i \notin t_k \wedge \exists i \in I : i \in t_k)}_{=\text{false}}\} \\
&= \{k \in \mathbb{N}_n \mid a \in t_k \wedge \forall i \in I : i \notin t_k \wedge b \notin t_k\} \\
&= \{k \in \mathbb{N}_n \mid a \in t_k \wedge \forall i \in I \cup \{b\} : i \notin t_k\} \\
&= E_T(a \mid I \cup \{b\})
\end{aligned}$$

In order to see how extra sets can be used to compute the extent of item sets, let $I = \{i_1, \ldots, i_m\}$, with some arbitrary, but fixed order of the items that is indicated by the index. This will be the order in which the items are used as split items in the recursive divide-and-conquer scheme. It is

$$\begin{aligned} L_T(I) &= \textstyle\bigcup_{k=1}^{m} K_T(\{i_k\}) \;=\; \bigcup_{k=1}^{m} \big(K_T(\{i_k\}) - \bigcup_{l=1}^{k-1} K_T(\{i_l\})\big) \\ &= \textstyle\bigcup_{k=1}^{m} E(i_k \mid \{i_1, \ldots, i_{k-1}\}), \end{aligned}$$

and since the terms of the last union are clearly all disjoint, we have immediately

$$r_T(I) = \sum_{k=1}^{m} |E(i_k \mid \{i_1, \ldots, i_{k-1}\})| = r_T(I - \{i_m\}) + |E(i_m \mid I - \{i_m\})|.$$

Thus we have a simple recursive scheme to compute the extent of an item set from its parent in the search tree (as defined by the divide-and-conquer scheme).

The search algorithm for Jaccard item sets can now easily be implemented as follows: we start be creating a vertical representation of the given transaction database. The only difference to the Eclat algorithm is that we have not only one, but two transaction lists per item i: one represents $K_T(\{i\})$ as in standard Eclat, and the other represents $E_T(i \mid \emptyset)$, which happens to be equal to $K_T(\{i\})$. That is, for the initial transaction database the two lists are identical. However, this will obviously not be maintained in the recursive processing. In the recursion the first list for the split item is intersected with the first list of all other items to form the lists representing the covers of the corresponding pairs. The second list of the split item is subtracted from the second list of all other items, thus yielding the extra sets of transactions for these items given the split item. From the sizes of the resulting lists the support and the extent of the enlarged item sets and thus their generalized Jaccard index can easily be computed.

Note that the support computation may, as in the Eclat algorithm, also be based on diffsets. Likewise, an analogous scheme can be derived for the extent computation. In addition, Jaccard item set mining can also exploit perfect extension pruning. The only difference is that an item a is now called a perfect extension of an item set I w.r.t. a transaction database T only if $s_T(I \cup \{a\}) = s_T(I)$ and $r_T(I \cup \{a\}) = r_T(I)$, while standard frequent item set mining only requires the first equality. Such perfect extensions are handled exactly in the same way: they are not employed as split items, but collected in a third element of the subproblem description, and are used only to generate all supersets of an item set that share the same generalized Jaccard index.

6 Other Similarity Measures

Up to now we focused on the generalized Jaccard index to measure the similarity of sets (item covers). However, there is a large number of other similarity measures for sets (or, equivalently, for binary vectors, because a set may be represented by its indicator vector w.r.t. some base set). Recent extensive overviews of such measures for the pairwise case include [7] and [8].

Table 1. Quantities in terms of which the considered similarity measures are specified, together with their behavior as functions on the partially ordered set $(2^B, \subseteq)$

quantity	requirement on transaction	behavior
n_T	none (independent of the set I)	constant
$s_T(I) = \lvert K_T(I)\rvert = \lvert\bigcap_{i\in I} K_T(\{i\})\rvert$	contains all items	anti-monotone
$r_T(I) = \lvert L_T(I)\rvert = \lvert\bigcup_{i\in I} K_T(\{i\})\rvert$	contains at least one item	monotone
$q_T(I) = r_T(I) - s_T(I)$	contains some, but not all items	monotone
$z_T(I) = n_T \quad - r_T(I)$	contains no item	anti-monotone

By relying on the same scheme that we used to generalize the Jaccard index to more than two sets, a large number of such set similarity or binary vector similarity measures can be generalized beyond pairwise comparisons as follows: with the JIM algorithm we presented in the preceding section, we can easily compute the five quantities listed in Table 1. These quantities count the number of transactions that satisfy different requirements w.r.t. a given item set I (see the second column of Table 1). With these quantities a wide range of similarity measures for sets or binary vectors can be generalized.

Exceptions are measures for comparing two sets X and Y that refer explicitly to the number $|X - Y|$ of elements that are contained in the set X, but not in the set Y, and distinguish this number from the number $|Y - X|$ of elements that are contained in the set Y, but not in the set X. This distinction is difficult to generalize beyond the pairwise case, because the number of possible containment patterns of an element to the members of a family of sets grows exponentially with the number of the sets (here: covers, and thus: items). As a generalization would have to consider all of these containment patterns separately, it becomes quickly infeasible. Note, however, that an occurrence of the sum $|X-Y|+|Y-X|$ does not pose a problem, because this sum corresponds to the value $q_T(I)$.

By collecting from [8] similarity measures that can be specified in terms of the quantities listed in Table 1, we compiled Table 2. Note that the index T and the argument I are omitted to make the formulas more easily readable. Note also that the Gower&Legendre measure $S_{\mathrm{G}} = \frac{s+z}{s+q/2+z}$ [18] listed in [8] is exactly the same as the second Sokal&Sneath measure (it is just written differently, with a factor of 2 canceled from both numerator and denominator). Furthermore, note that the Hamann measure $S_{\mathrm{H}} = \frac{x+z-s}{n} = \frac{n-2s}{n}$ [20] listed in [8] is equivalent to the Sokal&Michener measure S_{M}, because $S_{\mathrm{H}} + 1 = 2S_{\mathrm{M}}$, and hence omitted. Likewise, the second Baroni-Urbani&Buser measure $S_{\mathrm{U}} = \frac{\sqrt{xz}+x-q}{\sqrt{xz}+o}$ [4] listed in [8] is equivalent to the one given in Table 2, because $S_{\mathrm{U}} + 1 = 2S_{\mathrm{B}}$. Finally, note that all of the measures listed in Table 2 have range $[0, 1]$ except S_{K} (Kulczynski) and S_{O} (Sokal&Sneath 3), which have range $[0, \infty)$.

Table 2 is split into two parts depending on whether the numerator of a measure refers only to the support s or to both the support s and the number z of transactions that do not contain any of the items in the considered set I. The former are referred to as based on the inner product, because in the pairwise case

Table 2. Considered similarity measures for sets/binary vectors

Measures derived from inner product:

Russel & Rao [28]	$S_{\mathrm{R}} = \frac{s}{n} = \frac{s}{r+z}$
Kulczynski [25]	$S_{\mathrm{K}} = \frac{s}{q} = \frac{s}{r-s}$
Jaccard [22] Tanimoto [33]	$S_{\mathrm{J}} = \frac{s}{s+q} = \frac{s}{r}$
Dice [10] Sørensen [32] Czekanowski [9]	$S_{\mathrm{D}} = \frac{2s}{2s+q} = \frac{2s}{r+s}$
Sokal & Sneath 1 [31,29]	$S_{\mathrm{S}} = \frac{s}{s+2q} = \frac{s}{r+q}$

Measures derived from Hamming distance:

Sokal & Michener Hamming [30,21]	$S_{\mathrm{M}} = \frac{s+z}{n} = \frac{n-q}{n}$
Faith [12]	$S_{\mathrm{F}} = \frac{2s+z}{2n} = \frac{s+\frac{1}{2}z}{n}$
AZZOO [7] $\sigma \in [0,1]$	$S_{\mathrm{Z}} = \frac{s+\sigma z}{n}$
Rogers & Tanimoto [27]	$S_{\mathrm{T}} = \frac{s+z}{n+q} = \frac{n-q}{n+q}$
Sokal & Sneath 2 [31,29]	$S_{\mathrm{N}} = \frac{2(s+z)}{n+s+z} = \frac{n-q}{n-\frac{1}{2}q}$
Sokal & Sneath 3 [31,29]	$S_{\mathrm{O}} = \frac{s+z}{q} = \frac{n-q}{q}$
Baroni-Urbani & Buser [4]	$S_{\mathrm{B}} = \frac{\sqrt{sz}+s}{\sqrt{sz}+r}$

s is the value of the inner (or scalar) product of the binary vectors that are compared. The latter measures (that is, those with both s and z in the numerator) are referred to as based on the Hamming distance, because in the pairwise case q is the Hamming distance of the two vectors and $n - q = s + z$ their Hamming similarity. The decision whether for a given application the term z should be considered in the numerator of a similarity measure or not is difficult. Discussions of this issue for the pairwise case can be found in [29] and [11].

Note that the Russel & Rao measure is simply normalized support, demonstrating that our framework comprises standard frequent item set mining as a special case. The Sokal & Michener measure is simply the normalized Hamming similarity. The Dice/Sørensen/Czekanowski measure may be defined without the factor 2 in the numerator, changing the range to $[0, 0.5]$. The Faith measure is equivalent to the AZZOO measure (Alter Zero Zero One One) for $\sigma = 0.5$ and the Sokal & Michener/Hamming measure results for $\sigma = 1$. AZZOO is meant to introduce flexibility in how much weight should be placed on z, the number of transactions which lack all items in I (zero zero), relative to s (one one).

All measures listed in Table 2 are anti-monotone on the partially ordered set $(2^B, \subseteq)$, where B is the underlying item base. This is obvious if in at least one of the formulas given for a measure the numerator is (a multiple of) a constant or anti-monotone quantity or a (weighted) sum of such quantities, and the numerator is (a multiple of) a constant or monotone quantity or a (weighted) sum of such quantities (see Table 1). This is the case for all but S_{D}, S_{N} and S_{B}.

That S_{D} is anti-monotone can be seen by considering its reciprocal value

$$S_{\mathrm{D}}^{-1} = \frac{2s+q}{2s} = 1 + \frac{q}{2s}.$$

Since q is monotone and s is anti-monotone, S_{D}^{-1} is clearly monotone and thus S_{D} is anti-monotone. Applying the same approach to S_{B}, we arrive at

$$S_{\mathrm{B}}^{-1} = \frac{\sqrt{sz}+r}{\sqrt{sz}+s} = \frac{\sqrt{sz}+s+q}{\sqrt{sz}+s} = 1 + \frac{q}{\sqrt{sz}+s}.$$

Since q is monotone and both s and $\sqrt{sz}$ are anti-monotone, S_{B}^{-1} is clearly monotone and thus S_{B} is anti-monotone. Finally, S_{N} can be written as

$$S_{\mathrm{N}} = \frac{2n-2q}{2n-q} = 1 - \frac{q}{2n-q} = 1 - \frac{q}{n+s+z}.$$

Since q is monotone, the numerator is monotone, and since n is constant and s and z are anti-monotone, the denominator is anti-monotone. Hence the fraction is monotone and since it is subtracted from 1, S_{N} is anti-monotone.

Note that all measures in Table 2 can be expressed as

$$S = \frac{c_0 s + c_1 z + c_2 n + c_3\sqrt{sz}}{c_4 s + c_5 z + c_6 n + c_7\sqrt{sz}} \tag{1}$$

by specifying appropriate coefficients $c_0, \ldots, c_7$. For example, we obtain S_{J} for $c_0 = c_6 = 1$, $c_5 = -1$ and $c_1 = c_2 = c_3 = c_4 = c_7 = 0$, since $S_{\mathrm{J}} = \frac{s}{r} = \frac{s}{n-z}$. Similarly, we obtain S_{O} for $c_0 = c_1 = c_6 = 1$, $c_4 = c_5 = -1$ and $c_2 = c_3 = c_7 = 0$, since $S_{\mathrm{O}} = \frac{s+z}{q} = \frac{s+z}{n-s-z}$. This general form allows for a flexible specification of various similarity measures. Note, however, that not all selections of coefficients lead to an anti-monotone measure and hence one has to carefully check this property before using a measure that differs from the pre-specified ones.

7 Experiments

We implemented the described item set mining approach as a C program that was derived from an Eclat implementation by adding the second transaction identifier list for computing the extent of item sets. All similarity measures listed in Table 2 are included as well as the general form (1). This implementation has been made publicly available under the GNU Lesser (Library) Public License.[5]

In a first set of experiments we applied the program to five standard benchmark data sets, which exhibit different characteristics, especially different densities, and compared it to a standard Eclat search. The data sets we used are: BMS-Webview-1 (a web click stream from a leg-care company that no longer exists, which has been used in the KDD cup 2000 [23,40]), T10I4D100K (an artificial data set generated with IBM's data generator [41]), census (a data set derived from an extract of the US census bureau data of 1994, which was preprocessed by discretizing numeric attributes), chess (a data set listing chess end game positions for king vs. king and rook), and mushroom (a data set describing poisonous and edible mushrooms by different attributes). The first two data

[5] See `http://www.borgelt.net/jim.html`

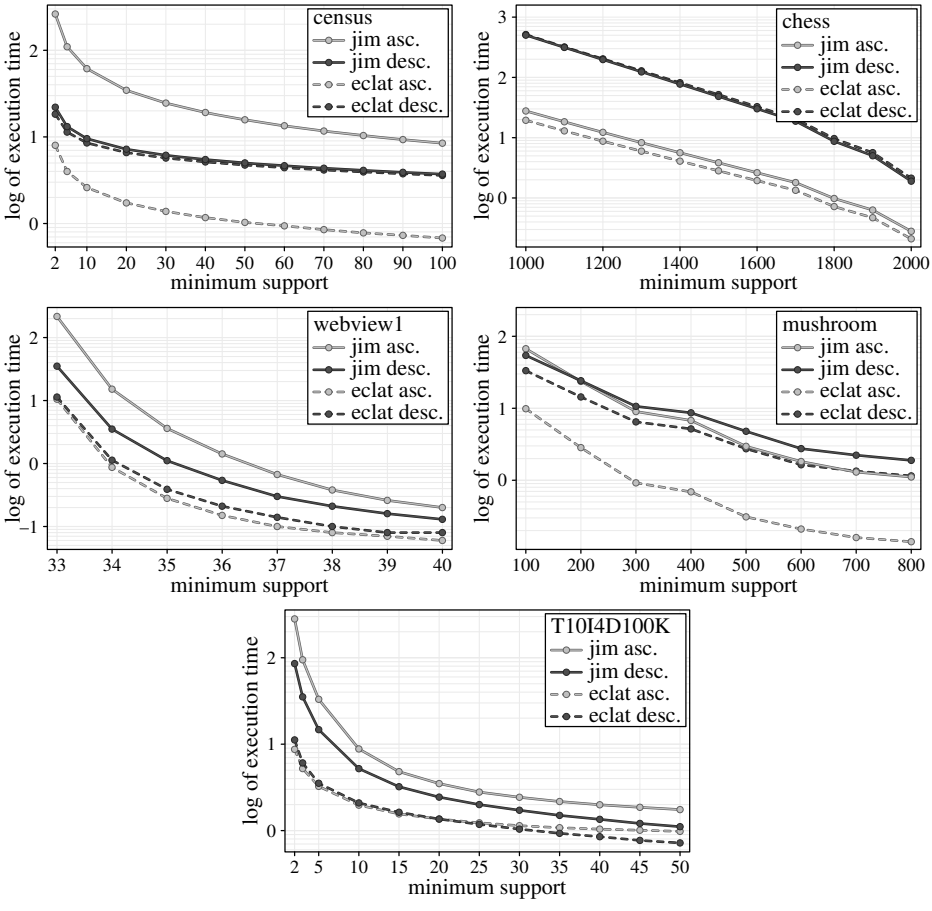

Fig. 1. Logarithms of execution times, measured in seconds, over absolute minimum support for Jaccard item set mining compared to standard Eclat frequent item set mining. Items were processed in ascending or descending order w.r.t. their frequency. Jaccard item set mining was executed with $J_{\min} = 0$, thus ensuring that exactly the same item sets are found.

sets are available in the FIMI repository [14], the last three in the UCI machine learning repository [3]. The discretization of the numeric attributes in the census data set was done with a shell/gawk script that can be found on the web page given in Footnote 5 (previous page). For the experiments we used an Intel Core 2 Quad Q9650 (3GHz) machine with 8 GB main memory running Ubuntu Linux 10.04 (64 bit) and gcc version 4.4.3.

The goal of these experiments was to determine how much the computation of the carrier/extent of an item set affected the execution time. Therefore we ran the JIM algorithm without any threshold for the similarity measure (we used the Jaccard index, i.e. $J_{\min} = 0$, but any other measure gives basically the same results), using only a minimum support threshold (which is supported by

our implementation in parallel). As a consequence, JIM and Eclat always found exactly the same set of frequent item sets for a given minimum support value and thus any difference in execution time comes from the additional costs of the carrier/extent computation. The difference in the generated output consists only in the Jaccard index that the JIM program computes, but standard Eclat can not compute as it lacks knowledge of the quantity $r_T(I)$. In addition, we explored whether the common rule of thumb of frequent item set mining, namely that it is best to process the items in the order of increasing frequency (cf. page 107), also holds for cover similarity based item set mining. Therefore we tried both ascending and descending frequency order for the items.

The results are depicted in the diagrams in Figure 1, which show the decimal logarithm of the execution time in seconds over minimum support (as an absolute number, that is, as a number of transactions). We observe first that for Eclat (dashed lines) processing the items in increasing order of frequency (light gray) almost always works better, since the execution times are shorter than for the reverse order (dark gray)—as expected. For JIM (solid lines), however, the picture is not so clear cut. On three data sets, namely census, BMS-Webview-1, and T10I4D100K, it is better to process the items in descending order of their frequency (the dark gray curve is lower than the light one). On chess it is better to use ascending order (the light gray curve is lower than the dark one), while on the fifth data set (mushroom) it depends on the minimum support which order yields the shorter execution time (the two curves intersect).

We interpret these findings as follows: for the support computation (which is all that Eclat does) it is clearly better to process the items in ascending order of their frequency, because this reduces the average length of the transaction identifier lists. By intersecting with short lists early, the lists processed in the recursion tend to be shorter and thus are processed faster. The same obviously also holds for the support computation part of JIM. However, for the extent computation it is plausible that the opposite order is preferable. Since it works on extra sets, it is advantageous to add frequent items as early as possible to the carrier, because this increases the size of the already covered carrier and thus reduces the average length of the extra lists that are processed in the recursion. Therefore, since there are different preferences, it depends on the data set which operation governs the complexity and thus which item order is better.

From Figure 1 we conjecture that dense data sets (high fraction of ones in a bit matrix representation), like chess and mushroom, favor ascending order, while sparse data sets, like census, BMS-Webview-1 and T10I4D100K, favor descending order. This is plausible, because in dense data sets the intersection lists tend to be long, so it is important to reduce them. In sparse data sets, however, the extra lists tend to be long, so here it is more important to focus on them. The mushroom data set behaves more like a dense data set for lower minimum support and more like a sparse data set for higher minimum support.

Naturally, the execution times of JIM are always greater than those of the corresponding Eclat runs (with the same order of the items), but the execution times are still bearable. This shows that even if one does *not* use a similarity measure to *prune* the search, this additional information can be computed fairly

Table 3. Jaccard item sets found in the 2008/2009 Wikipedia Selection for schools

item set	s_T	J_T
Reptiles, Insects	12	1.0000
phylum, chordata, animalia	34	0.7391
planta, magnoliopsida, magnoliophyta	14	0.6667
wind, damag, storm, hurrican, landfal	23	0.1608
tournament, doubl, tenni, slam, Grand Slam	10	0.1370
dinosaur, cretac, superord, sauropsida, dinosauria	10	0.1149
decai, alpha, fusion, target, excit, dubna	12	0.1121
conserv, binomi, phylum, concern, animalia, chordata	14	0.1053

efficiently. However, it should be kept in mind that the idea of the approach is to set a threshold for the similarity measure, which can effectively prune the search, so that the actual execution times found in applications are much lower. In our own practice we basically always achieved execution times that were lower than for the Eclat algorithm (but, of course, with a different output).

In another experiment we used an extract from the 2008/2009 Wikipedia Selection for schools[6], which consisted of 4861 web pages. Each of these web pages was taken as a transaction and processed with standard text processing methods (like name detection, stemming, stop word removal etc.) to extract a total of 59330 terms/keywords. The terms occurring on a web page are the items occurring in the corresponding transaction. The resulting data file was then mined for Jaccard item sets with thresholds of $J_{\min} = 0.1$ and $s_{\min} = 10$. Some examples of term associations found in this way are listed in Table 3.

Clearly, there are several term sets with surprisingly high Jaccard indices and thus strongly associated terms. For example, "Reptiles" and "Insects" always appear together (on a total of 12 web pages) and never alone (as their Jaccard index is 1, so their covers are identical). A closer inspection revealed, however, that this is an artifact of the name detection, which extracts these terms from the Wikipedia category title "Insects, Reptiles and Fish" (but somehow treats "Fish" not as a name, but as a normal word). All other item sets contain normal terms, though (only "Grand Slam" is another name), and are not artifacts of the text processing step. The second item set captures several biology pages, which describe different vertebrates, all of which belong to the phylum "chordata" and the kingdom "animalia". The third set indicates that this selection contains a surprisingly high number of pages referring to magnolias. The remaining item sets show that term sets with five or even six terms can exhibit a quite high Jaccard index, even though they have a fairly low support (only 10–20 transactions, which corresponds to 0.2–0.4% of the 4861 transactions/web pages).

An impression of the filtering power can be obtained by comparing the size of the output to standard frequent item set mining: for $s_{\min} = 10$ there are 83130 frequent item sets and 19394 closed item sets with at least two items. A threshold of $J_{\min} = 0.1$ for the generalized Jaccard index reduces the output

[6] See `http://schools-wikipedia.org/`

Table 4. Some Jaccard item sets that were found in BMS-Webview-1

item set	s_T	J_T
35201, 35205, 35193, 35189, 35197, 35209	37	0.1034
18767, 18751, 18755, 18763, 18743, 18747, 18759	33	0.1467
18543, 18567, 18751, 18539, 18763, 18743, ... 18747, 18571, 18759	27	0.1089
18543, 18567, 18751, 18539, 18763, 18743, ... 18747, 18571, 18759, 18767	27	0.0951

to 5116 (frequent) item sets. From manual inspection, we gathered the impression that the Jaccard item sets contained more meaningful sets and that the Jaccard index was a valuable additional piece of information. It has to be conceded, though, that whether item sets are more "meaningful" or "interesting" is difficult to assess in a convincing fashion. Such an assessment would require an objective measure, which is not available (and if it were available, it could be used directly for the mining). What can be said, though, is that the support and the generalized Jaccard index assess item sets in very different ways, since for the 5116 item sets mentioned above, the correlation coefficient of the support and the generalized Jaccard index is merely 0.18. That is, neither does a high support imply a high generalized Jaccard index nor vice versa.

As an additional example, Table 4 lists Jaccard item sets that were found in BMS-Webview-1. Despite their low support (25–40 transactions, which corresponds to 0.04%–0.07% of the 59602 transactions), they could quickly and effectively be identified with Jaccard item set mining. This result is particularly impressive, because standard frequent item set mining without a restriction to e.g. closed item sets is not possible in reasonable time on BMS-Webview-1 for a minimum support less than about 32 transactions. Restricting the output to closed item sets makes mining feasible and yields 110427 item sets for $s_{\min} = 32$. Jaccard item set mining with thresholds of $s_{\min} = 32$ and $J_{\min} = 0.1$ (but without a restriction to closed item sets) reduces the output to 982 item sets. Again item sets with fairly many items and surprisingly high generalized Jaccard index are found. As for the 2008/2009 Wikipedia Selection for schools the correlation coefficient of the support and the generalized Jaccard index is very low, in this case actually even slightly negative, namely -0.02.

An example of a Jaccard item set from the census data set is

{loss=none, gain=none, country=United-States, race=White,
workclass=Private, sex=Male, age=middle-aged,
marital_status=Married-civ-spouse, relationship=Husband},

that is, an item set with 9 items with a support of 5245 (10.7%) and a Jaccard index of 0.1074. Again it is surprising to see how large an item set can possess a high generalized Jaccard index. Although this set would be discovered with standard frequent item set mining as well, the generalized Jaccard index provides a relevant additional assessment and thus distinguishes it from other item sets.

As a final remark we would like to point out that the usefulness of our method is indirectly supported by a successful application of the Jaccard item set mining approach for (missing) concept detection (see [24] as well as the chapter by Kötter and Berthold in this book, which describes the application).

8 Conclusions

In this chapter we introduced the notion of a Jaccard item set as an item set for which the generalized Jaccard index of the covers of its items exceeds a user-specified threshold. In addition, we extended this basic idea to a total of twelve similarity measures for sets or binary vectors, all of which can be generalized in the same way and can be shown to be anti-monotone. By exploiting an idea that is similar to the difference set approach for the well-known Eclat algorithm, we derived an efficient search scheme that is based on forming intersections and differences of sets of transaction indices in order to compute the quantities that are needed to compute the similarity measures. Since it contains standard frequent item set mining as a special case, mining item sets based on cover similarity yields a flexible and versatile framework. Furthermore, the similarity measures provide highly useful additional assessments of found item sets and thus help us to select the interesting ones. By running experiments on standard benchmark data sets we showed that mining item sets based on cover similarity can be done fairly efficiently, and by evaluating the results obtained with a threshold for the cover similarity measure we demonstrated that the output is considerably reduced, while expressive and meaningful item sets are preserved.

References

1. Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules. In: Proc. 20th Int. Conf. on Very Large Databases, VLDB 1994, Santiago de Chile, pp. 487–499. Morgan Kaufmann, San Mateo (1994)
2. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.: Fast Discovery of Association Rules. In: [13], pp. 307–328
3. Asuncion, A., Newman, D.J.: UCI Machine Learning Repository. School of Information and Computer Science, University of California at Irvine, CA, USA (2007), `http://www.ics.uci.edu/~mlearn/MLRepository.html`
4. Baroni-Urbani, C., Buser, M.W.: Similarity of Binary Data. Systematic Zoology 25(3), 251–259 (1976)
5. Bayardo, R., Goethals, B., Zaki, M.J. (eds.): Proc. Workshop Frequent Item Set Mining Implementations, FIMI 2004, Brighton, UK, Aachen, Germany. CEUR Workshop Proceedings, vol. 126 (2004), `http://www.ceur-ws.org/Vol-126/`

6. Borgelt, C., Wang, X.: SaM: A Split and Merge Algorithm for Fuzzy Frequent Item Set Mining. In: Proc.13th Int. Fuzzy Systems Association World Congress and 6th Conf. of the European Society for Fuzzy Logic and Technology, IFSA/EUSFLAT 2009. IFSA/EUSFLAT Organization Committee, Lisbon (2009)
7. Cha, S.-H., Tappert, C.C., Yoon, S.: Enhancing Binary Feature Vector Similarity Measures. J. Pattern Recognition Research 1, 63–77 (2006)
8. Choi, S.-S., Cha, S.-H., Tappert, C.C.: A Survey of Binary Similarity and Distance Measures. Journal of Systemics, Cybernetics and Informatics 8(1), 43–48 (2010); Int. Inst. of Informatics and Systemics, Caracas, Venezuela (2010)
9. Czekanowski, J.: Zarys metod statystycznych w zastosowaniu do antropologii (An Outline of Statistical Methods Applied in Anthropology). Towarzystwo Naukowe Warszawskie, Warsaw, Poland (1913)
10. Dice, L.R.: Measures of the Amount of Ecologic Association between Species. Ecology 26, 297–302 (1945)
11. Dunn, G., Everitt, B.S.: An Introduction to Mathematical Taxonomy. Cambridge University Press, Cambirdge (1982)
12. Faith, D.P.: Asymmetric Binary Similarity Measures. Oecologia 57(3), 287–290 (1983)
13. Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R. (eds.): Advances in Knowledge Discovery and Data Mining. AAAI Press / MIT Press, Cambridge (1996)
14. Goethals, B. (ed.): Frequent Item Set Mining Dataset Repository. University of Helsinki, Finland (2004), `http://fimi.cs.helsinki.fi/data/`
15. Goethals, B., Zaki, M.J. (eds.): Proc. Workshop Frequent Item Set Mining Implementations, FIMI 2003, Melbourne, FL, USA. CEUR Workshop Proceedings 90, Aachen, Germany (2003), `http://www.ceur-ws.org/Vol-90/`
16. Grahne, G., Zhu, J.: Efficiently Using Prefix-trees in Mining Frequent Itemsets. In: Proc. Workshop Frequent Item Set Mining Implementations, FIMI, Melbourne, FL [15] (2003)
17. Grahne, G., Zhu, J.: Reducing the Main Memory Consumptions of FPmax* and FPclose. In: Proc. Workshop Frequent Item Set Mining Implementations, FIMI, Brighton, UK [5] (2004)
18. Gower, J.C., Legendre, P.: Metric and Euclidean Properties of Dissimilarity Coefficients. Journal of Classification 3, 5–48 (1986)
19. Han, J., Pei, H., Yin, Y.: Mining Frequent Patterns without Candidate Generation. In: Proc. Conf. on the Management of Data, SIGMOD 2000, Dallas, TX, pp. 1–12. ACM Press, New York (2000)
20. Hamann, V.: Merkmalbestand und Verwandtschaftsbeziehungen der Farinosae. Ein Beitrag zum System der Monokotyledonen. Willdenowia 2, 639–768 (1961)
21. Hamming, R.V.: Error Detecting and Error Correcting Codes. Bell Systems Tech. Journal 29, 147–160 (1950)
22. Jaccard, P.: Étude comparative de la distribution florale dans une portion des Alpes et des Jura. Bulletin de la Société Vaudoise des Sciences Naturelles 37, 547–579 (1901)
23. Kohavi, R., Bradley, C.E., Frasca, B., Mason, L., Zheng, Z.: KDD-Cup 2000 Organizers' Report: Peeling the Onion. SIGKDD Exploration 2(2), 86–93 (2000)
24. Kötter, T., Berthold, M.R.: Concept Detection. In: Proc. 8th Conf. on Computing and Philosophy, ECAP 2010. University of Munich, Munich (2010)
25. Kulczynski, S.: Classe des Sciences Mathématiques et Naturelles. Bulletin Int. de l'Acadamie Polonaise des Sciences et des Lettres Série B (Sciences Naturelles) (Supp. II), 57–203 (1927)

26. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering Frequent Closed Itemsets for Association Rules. In: Beeri, C., Bruneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 398–416. Springer, Heidelberg (1998)
27. Rogers, D.J., Tanimoto, T.T.: A Computer Program for Classifying Plants. Science 132, 1115–1118 (1960)
28. Russel, P.F., Rao, T.R.: On Habitat and Association of Species of Anopheline Larvae in South-eastern Madras. J. Malaria Institute 3, 153–178 (1940)
29. Sneath, P.H.A., Sokal, R.R.: Numerical Taxonomy. Freeman Books, San Francisco (1973)
30. Sokal, R.R., Michener, C.D.: A Statistical Method for Evaluating Systematic Relationships. University of Kansas Scientific Bulletin 38, 1409–1438 (1958)
31. Sokal, R.R., Sneath, P.H.A.: Principles of Numerical Taxonomy. Freeman Books, San Francisco (1963)
32. Sørensen, T.: A Method of Establishing Groups of Equal Amplitude in Plant Sociology based on Similarity of Species and its Application to Analyses of the Vegetation on Danish Commons. Biologiske Skrifter / Kongelige Danske Videnskabernes Selskab 5(4), 1–34 (1948)
33. Tanimoto, T.T.: IBM Internal Report, November 17 (1957)
34. Uno, T., Kiyomi, M., Arimura, H.: LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets. Proc. Workshop Frequent Item Set Mining Implementations, FIMI 2004, Brighton, UK. CEUR Workshop Proceedings 126, Aachen, Germany (2004)
35. Uno, T., Kiyomi, M., Arimura, H.: LCM ver. 3: Collaboration of Array, Bitmap and Prefix Tree for Frequent Itemset Mining. Proc. 1st Open Source Data Mining on Frequent Pattern Mining Implementations, OSDM 2005, Chicago, IL, pp. 77–86. ACM Press, New York (2005)
36. Webb, G.I., Zhang, S.: k-Optimal-Rule-Discovery. Data Mining and Knowledge Discovery 10(1), 39–79 (2005)
37. Webb, G.I.: Discovering Significant Patterns. Machine Learning 68(1), 1–33 (2007)
38. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: New Algorithms for Fast Discovery of Association Rules. In: Proc. 3rd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, KDD 1997, Newport Beach, CA, pp. 283–296. AAAI Press, Menlo Park (1997)
39. Zaki, M.J., Gouda, K.: Fast Vertical Mining Using Diffsets. In: Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, KDD 2003, Washington, DC, pp. 326–335. ACM Press, New York (2003)
40. Zheng, Z., Kohavi, R., Mason, L.: Real World Performance of Association Rule Algorithms. In: Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, KDD 2001, San Francisco, CA, ACM Press, New York (2001)
41. Synthetic Data Generation Code for Associations and Sequential Patterns. Intelligent Information Systems, IBM Almaden Research Center, `http://www.almaden.ibm.com/software/quest/Resources/index.shtml`