Mining Rank Data

Sascha Henzgen and Eyke Hüllermeier

Department of Computer Science
University of Paderborn, Germany
{sascha.henzgen,eyke}@upb.de

Abstract. This paper addresses the problem of mining rank data, that is, data in the form of rankings (total orders) of an underlying set of items. More specifically, two types of patterns are considered, namely frequent subrankings and dependencies between such rankings in the form of association rules. Algorithms for mining patterns of this kind are proposed and illustrated on three case studies.

1 Introduction

The major goal of data mining methods is to find potentially interesting patterns in (typically very large) data sets. The meaning of "interesting" may depend on the application and the purpose a pattern is used for. Quite often, interestingness is connected to the *frequency* of occurrence: A pattern is considered interesting if its number of occurrences in the data strongly deviates from what one would expect on average. When being observed much more often, ones speaks of a *frequent pattern*, and the problem of discovering such patterns is called *frequent pattern mining* [6]. The other extreme is outliers and exceptional patterns, which deviate from the norm and occur rarely in the data; finding such patterns is called *exception mining* [10].

Needless to say, the type of patterns considered, of measures used to assess their interestingness, and of algorithms used to extract those patterns being highly rated in terms of these measures, strongly depends on the nature of the data. It makes a big difference, for example, whether the data is binary, categorical, or numerical, and whether a single observation is described in terms of a *subset*, like in itemset mining [6], or as a *sequence*, like in sequential pattern mining [9].

In this paper, we make a first step toward the mining of *rank data*, that is, data that comes in the form of *rankings* of an underlying set of items. This idea is largely motivated by the recent emergence of *preference learning* as a novel branch of machine learning [5]. While methods for problems such as "learning to rank" have been studied quite intensely in this field, rank data has not yet been considered from a data mining perspective so far.

To illustrate what we mean by rank data, consider a version of the well-known SUSHI benchmark, in which 5000 customers rank 10 different types of sushi from

S. Džeroski et al. (Eds.): DS 2014, LNAI 8777, pp. 123–134, 2014.

most preferred to least preferred.[1] This data could be represented in the form of a matrix as follows:

```
5  7  3  8  4 10  2  1  6  9
6 10  1  4  8  7  2  3  5  9
2  7  3  1  6  9  5  8  4 10
.  .  .  .  .  .  .  .  .  .
```

In this matrix, the value in row i and column j corresponds to the position of the jth sushi in the ranking of the ith customer. For example, the first customer likes the eighth sushi the most, the seventh sushi the second best, and so on.

The above data consists of *complete* rankings, i.e., each observation is a ranking of the complete set of items (the 10 types of sushi). While we do assume data of that kind in this paper, there are many applications in which rank data is less complete, especially if the underlying set of items is larger. We shall come back to corresponding generalizations of our setting in the end of this paper.

The rest of the paper is organized as follows. In the next two sections, we explain more formally what we mean by rank data and rank patterns, respectively. An algorithm for mining rank patterns in the form of what we call *frequent subrankings* is then introduced in Section 4. Some experimental results are presented in Section 5, prior to concluding the paper in Section 6.

2 Rank Data

Let $\mathbb{O} = \{o_1, \ldots, o_N\}$ be a set of items or objects. A ranking of these items is a total order that is represented by a permutation

$$\boldsymbol{\pi} : \, [N] \rightarrow [N] \enspace ,$$

that is, a bijection on $[N] = \{1, \ldots, N\}$, where $\boldsymbol{\pi}(i)$ denotes the position of item o_i. Thus, the permutation $\boldsymbol{\pi}$ represents the order relation

$$o_{\boldsymbol{\pi}^{-1}(1)} \succ o_{\boldsymbol{\pi}^{-1}(2)} \succ \cdots \succ o_{\boldsymbol{\pi}^{-1}(N)} \enspace ,$$

where $\boldsymbol{\pi}^{-1}$ is the inverse of $\boldsymbol{\pi}$, i.e., $\boldsymbol{\pi}^{-1}(j)$ is the index of the item on position j.

We assume data to be given in the form of a set $\mathbb{D} = \{\boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \ldots, \boldsymbol{\pi}_M\}$ of (complete) rankings $\boldsymbol{\pi}_i$ over a set of items $\mathbb{O}$. Returning to our example above, $\mathbb{O} = \{o_1, \ldots, o_{10}\}$ could be the 10 types of sushi, and $\boldsymbol{\pi}_i$ the ranking of these sushis by the ith customer.

3 Rank Patterns

In the context of rank data as outlined above, there is a large variety of *rank patterns* that might be of interest. In this section, we introduce two examples of rank patterns that we shall elaborate further in subsequent sections.

[1] http://kamishima.new/sushi/

3.1 Subrankings

An obvious example of such a pattern is a *subranking.* We shall use this term for a ranking π of a subset of objects $O \subset \mathbb{O}$. Here, $\pi(j)$ is the position of item o_j provided this item is contained in the subranking, and $\pi(j) = 0$ otherwise. For example, $\pi = (0, 2, 1, 0, 0, 3)$ denotes the subranking $o_3 \succ o_2 \succ o_6$, in which the items o_1, o_4, o_5 do not occur.

In the following, we will write complete rankings $\boldsymbol{\pi}$ in bold font (as we already did above) and subrankings in normal font. The number of items included in a subranking π is denoted $|\pi|$; if $|\pi| = k$, then we shall also speak of a k-ranking.

We denote by $O(\pi)$ the set of items ranked by a subranking π. The other way around, if $O' \subset O(\pi)$, then $(\pi|O')$ denotes the restriction of the ranking π to the set of objects O', i.e.,

$$(\pi|O')(j) = \begin{cases} \#\{o_i \in O' \,|\, \pi(i) \leq \pi(j)\} & \text{if } o_j \in O' \\ 0 & \text{if } o_j \notin O' \end{cases}$$

If π is a subranking of $O = O(\pi)$, then $\boldsymbol{\pi}$ is a (linear) extension of π if $(\boldsymbol{\pi}|O) = \pi$; in this case, the items in O are put in the same order by $\boldsymbol{\pi}$ and π, i.e., the former is consistent with the latter. We shall symbolize this consistency by writing $\pi \subset \boldsymbol{\pi}$ and denote by $E(\pi)$ the set of linear extensions of π.

Now, we are ready to define the notion of *support* for a subranking π. In analogy to the well-known problem of itemset mining (see also Section 4 below), this is the relative frequency of observations in the data in which π occurs as a subranking:

$$\text{supp}(\pi) = \frac{1}{M} \cdot \#\{\boldsymbol{\pi}_i \in \mathbb{D} \,|\, \pi \subset \boldsymbol{\pi}_i\} \tag{1}$$

A *frequent subranking* is a subranking π such that

$$\text{supp}(\pi) \geq min_{supp} \enspace ,$$

where min_{supp} is a user-defined support threshold. A frequent subranking π is *maximal* if there is no frequent subranking π' such that $\pi \subset \pi'$ and $\pi' \not\subset \pi$.

3.2 Association Rules

Association rules are well-known in data mining and have first been considered in the context of *itemset mining.* Here, an association rule is a pattern of the form $A \rightharpoonup B$, where A and B are itemsets. The intended meaning of such a rule is that a transaction containing A is likely to contain B, too. In market-basket analysis, where a transaction is a purchase and items are associated with products, the association $\{\texttt{paper}, \texttt{envelopes}\} \rightharpoonup \{\texttt{stamps}\}$ suggests that a purchase containing paper and envelopes is likely to contain stamps as well.

Rules of that kind can also be considered in the context of rank data. Here, we look at associations of the form

$$\pi_A \rightharpoonup \pi_B \enspace , \tag{2}$$

where π_A and π_B are subrankings of $\mathbb{O}$ such that

$$\#\big(O(\pi_A) \cap O(\pi_B)\big) \leq 1 \ . \tag{3}$$

For example, the rule $b \succ e \succ a \rightharpoonup d \succ c$ suggests that if b ranks higher than e, which in turn ranks higher than a, then d tends to rank higher than c. Note that this rule does not make any claims about the order relation between items in the antecedent and the consequent part. For example, d could rank lower but also higher than b. In general, the (complete) rankings $\boldsymbol{\pi}$ that are consistent with a rule (2) is given by $E(\pi_A) \cap E(\pi_B)$.

The condition (3) may call for an explanation. It plays the same role as the condition of empty intersection between items in the rule antecedent A and rule consequent B that is commonly required for association rules $A \rightharpoonup B$ in itemset mining ($A \cap B = \emptyset$), and which is intended to avoid trivial dependencies. In fact, assuming an item a in the rule antecedent trivially implies its occurrence in all transactions to which this rule is applicable. In our case, this is not completely true, since a subranking is modeling relationships between items instead of properties of single items. For example, a rule like $a \succ b \rightharpoonup a \succ c$ is not at all trivial, although the item a occurs on both sides. A redundancy occurs, however, as soon as two or more items are included both in π_A and π_B. This is why we restrict such occurrences to at most one item.

In itemset mining, the confidence measure

$$\mathrm{conf}(A \rightharpoonup B) = \frac{\mathrm{supp}(A \cup B)}{\mathrm{supp}(A)}$$

that is commonly used to evaluate association rules $A \rightharpoonup B$ can be seen as an estimation of the conditional probability

$$\mathbf{P}(B \,|\, A) = \frac{\mathbf{P}(A \text{ and } B)}{\mathbf{P}(A)} \ ,$$

i.e., the probability to observe itemset B given the occurrence of itemset A. Correspondingly, we define the confidence of an association $\pi_A \rightharpoonup \pi_B$ as

$$\mathrm{conf}(\pi_A \rightharpoonup \pi_B) = \frac{\#\{\boldsymbol{\pi}_i \in \mathbb{D} \,|\, \pi_A, \pi_B \subset \boldsymbol{\pi}_i\}}{\#\{\boldsymbol{\pi}_i \in \mathbb{D} \,|\, \pi_A \subset \boldsymbol{\pi}_i\}} = \frac{\mathrm{supp}(\pi_A \oplus \pi_B)}{\mathrm{supp}(\pi_A)} \tag{4}$$

As an important difference between mining itemsets and mining rank data, note that the class of patterns is closed under logical conjunction in the former but not in the latter case: Requiring the simultaneous occurrence of itemset A *and* itemset B is equivalent to requiring the occurrence of their union $A \cup B$, which is again an itemset. The conjunction of two subrankings π_A and π_B, denoted $\pi_A \oplus \pi_B$ in (4), is not again a subranking, however, at least not a single one; instead, it is represented by a *set* of subrankings $\pi_A \oplus \pi_B$, namely the rankings π such that $O(\pi) = O(\pi_A) \cup O(\pi_B)$, $(\pi|O(\pi_A)) = \pi_A$, $(\pi|O(\pi_B)) = \pi_B$. Correspondingly, the *joint support* of π_A and π_B,

$$\mathrm{supp}(\pi_A \oplus \pi_B) = \#\{\boldsymbol{\pi}_i \in \mathbb{D} \,|\, \pi_A, \pi_B \subset \boldsymbol{\pi}_i\} \ , \tag{5}$$

is the support of this subset. As we shall see later on, this has an implication on an algorithmic level.

Finally, and again in analogy with itemset mining, we define a measure of *interest* or *significance* of an association as follows:

$$\operatorname{sign}(\pi_A \rightharpoonup \pi_B) = \operatorname{conf}(\pi_A \rightharpoonup \pi_B) - \operatorname{supp}(\pi_B) \tag{6}$$

Just like for the measure of support, one is then interested in reaching certain thresholds, i.e., in finding association rules $\pi_A \rightharpoonup \pi_B$ that are highly supported ($\operatorname{supp}(\pi_A \rightharpoonup \pi_B) \geq min_{supp}$), confident ($\operatorname{conf}(\pi_A \rightharpoonup \pi_B) \geq min_{conf}$), and/or significant ($\operatorname{sign}(\pi_A \rightharpoonup \pi_B) \geq min_{sign}$).

3.3 Comparison with Itemset Mining

The connection between mining rank data and itemset mining has already been touched upon several times. Moreover, as will be seen in Section 4, our algorithm for extracting frequent subrankings can be seen as a variant of the basic Apriori algorithm, which has first been proposed for the purpose of itemset mining [1].

Noting that a ranking can be represented (in a unique way) in terms of a *set* of pairwise preferences, our problem could in principle even be reduced to itemset mining. To this end, a new item $o_{i,j}$ is introduced for each pair of items $o_i, o_j \in \mathbb{O}$, and a subranking π is represented by the set of items

$$\{o_{i,j} \,|\, o_i, o_j \in O(\pi),\ \pi(i) < \pi(j)\} \ .$$

This reduction has a number of disadvantages, however. First, the number of items is increased by a quadratic factor, although the information contained in these items is largely redundant. In fact, due to the transitivity of rankings, the newly created items exhibit (logical) dependencies that need to be taken care of by any mining algorithm. For example, not every itemset corresponds to a valid ranking, only those that are transitively closed.

Apart from that, there are some important differences between the two settings, for example regarding the number of possible patterns. In itemset mining, there are 2^N different subsets of N items, which is much smaller than the $N!$ number of rankings of these items. However, the N we assume for rank data (at least if the rankings are supposed to be complete) is much smaller than the N in itemset mining, which is typically very large. Besides, the itemsets observed in a transaction database are normally quite small and contain only a tiny fraction of all items. In fact, assuming an upper bound K on the size of an itemset, the number of itemsets is of the order $O(N^K)$ and grows much slower in N than exponential.

4 Algorithms

Our algorithm for mining frequent subrankings is based on the well-known Apriori algorithm for mining frequent itemsets [1]. This algorithm constructs itemsets in a level-wise manner, starting with singletons and increasing the size by 1 in each iteration. Candidate itemsets of size k are constructed from the frequent itemsets of size $k-1$ already found. In this step, Apriori exploits an important monotonicity property for pruning candidates: If A is a frequent itemset, then any subset of A must be frequent, too. Thus, by contraposition, any superset of a non-frequent itemset cannot be frequent either.

This monotonicity property also holds for subrankings: If a subranking π is frequent, then all rankings $\pi' \subset \pi$ are frequent, too. Thus, the Apriori approach is in principle applicable to rank data. Nevertheless, since rankings and sets have different properties, the construction and filtering steps need to be adapted. The basic structure of our algorithm for finding frequent subrankings, that will subsequently be explained in more detail, is the following:

1. Initial search for frequent 2-rankings $\mathcal{F}^{(2)}$ (set $k=3$).
2. LOOP:
   (a) Construct potential frequent k-rankings from the set of frequent $(k-1)$-rankings $\mathcal{F}^{(k-1)}$.
   (b) Filter frequent k-rankings from the potential frequent k-ranking set.
   (c) Stop the LOOP if no k-ranking passes the filtering.
   (d) Set $k=k+1$.

4.1 Searching Frequent 2-Rankings

While the smallest unit in itemset mining is an item, the smallest unit in the case of subrankings is a preference pair $a \succ b$. Therefore, the initial step is to exhaustively search for all frequent 2-rankings in the data set of rankings.

4.2 Construction of Candidate k-Rankings

Every k-ranking $\pi^{(k)}$ can be decomposed into a set of $(k-l)$-rankings

$$C^{(l)}\left(\pi^{(k)}\right) = \left\{\pi_i^{(k-l)} \,|\, \pi_i^{(k-l)} \subset \pi^{(k)}\right\}$$

with $0 \leq l < k$. A k-ranking $\pi^{(k)}$ is *fully consistent* with $\mathcal{F}^{(k-1)}$ if $C^{(1)}(\pi^{(k)}) \subset \mathcal{F}^{(k-1)}$.

In this step, we search for all k-rankings that are fully consistent with $\mathcal{F}^{(k-1)}$. For this purpose, the algorithm iterates over all pairs $(\pi_i^{(k-1)}, \pi_j^{(k-1)}) \in \mathcal{F}^{(k-1)} \times \mathcal{F}^{(k-1)}$ with $i<j$ and builds *partially consistent* k-rankings. These are rankings $\pi^{(k)}$ such that $\{\pi_i^{(k-1)}, \pi_j^{(k-1)}\} \subset C^{(1)}(\pi^{(k)})$. For example, from the 2-rankings $a \succ b$ and $c \succ d$, we are not able to build any 3-ranking, whereas from $a \succ b$ and $b \succ c$, we can build the 3-ranking $a \succ b \succ c$. Likewise, from the 2-rankings $a \succ c$ and $b \succ c$, we can build the two rankings $a \succ b \succ c$ and $b \succ a \succ c$.

Being partially consistent with a single pair $\{\pi_i^{(k-1)}, \pi_j^{(k-1)}\} \subset \mathcal{F}^{(k-1)}$ is necessary but not sufficient for being fully consistent with $\mathcal{F}^{(k-1)}$. Let us again consider the 2-rankings $a \succ b$ and $b \succ c$, and the only partially consistent 3-ranking $a \succ b \succ c$. In order to assure that this ranking is fully consistent with $\mathcal{F}^{(k-1)}$, we have to check whether the ranking $a \succ c$ is in $\mathcal{F}^{(k-1)}$, too.

Instead of explicitly searching for $a \succ c$ in $\mathcal{F}^{(k-1)}$, we store $a \succ b \succ c$ in a hash map with a key (based on the object sequence abc) and a count as value. This count is set to 1 the first time we put in a key and incremented every time we apply the `put(key)` operation. The idea is that if a ranking $\pi^{(k)}$ is fully consistent with $\mathcal{F}^{(k-1)}$, we find exactly $|C^{(1)}(\pi^{(k)})|(|C^{(1)}(\pi^{(k)})|-1)/2 = k(k-1)/2$ pairs of $(k-1)$-rankings from which $\pi^{(k)}$ can be built. After iterating over all pairs, we pass through the entries in our hash map and collect the keys with value $k(k-1)/2$. These rankings form the set of potentially frequent k-rankings. The whole procedure is described in Algorithm 1.

Algorithm 1.

```
procedure NCR($\mathcal{F}^{(k-1)}$)
    $\mathcal{K} = \emptyset$                                    ▷ set of candidates
    initialize HashMap $M$
    for $i = 1$ to $|\mathcal{F}^{(k-1)}| - 1$ do
        for $j = i + 1$ to $|\mathcal{F}^{(k-1)}|$ do
            $\mathcal{C}_{i,j} \leftarrow$ CONSTRUCT($(\pi_i^{(k-1)}, \pi_j^{(k-1)})$)    ▷ constructs sub consistent $k$-rankings
            for $\pi^{(k)} \in \mathcal{C}_{i,j}$ do
                if $M.getValue(\pi^{(k)}) = null$ then              ▷ $\pi^{(k)}$ is key
                    $M.put(\pi^{(k)}, 1)$
                else
                    $M.incrementValue(\pi^{(k)})$
                end if
            end for
        end for
    end for
    for $entry \in M$ do
        if $entry.getValue() = k(k-1)/2$ then
            $\mathcal{K}.add(entry.getKey())$
        end if
    end for
    return $\mathcal{K}$
end procedure
```

The task of CONSTRUCT is to take two $(k-1)$-rankings and construct all partially consistent k-rankings. The way CONSTRUCT works is actually quite easy, although the implementation is a bit intricate. Roughly speaking, the two rankings are compared position by position, and the algorithm has a special handling for the last two positions. CONSTRUCT can be divided into two steps, an alignment step and a construction step. Before we explain the basic principle of operation, let us make a few observations.

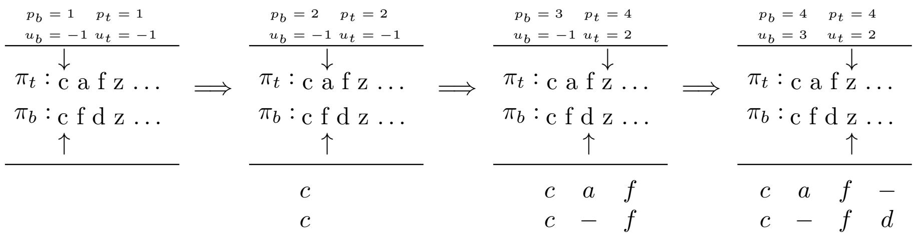

Fig. 1. Illustration of the CONSTRUCT procedure. The second row shows how the object pointers are set after every iteration—the upper arrow is the top object pointer p_t and the lower arrow the bottom object pointer p_b. The alignment, which is implicitly constructed by the algorithm, is shown in the bottom row.

The number of partially consistent k-rankings that can be constructed from two $(k-1)$-rankings is 0, 1 or 2. It is 0 if there is no k-ranking that is consistent with $\pi_i^{(k-1)}$ and $\pi_j^{(k-1)}$. By construction, $\pi_i^{(k-1)} = \pi_j^{(k-1)}$ only if $i = j$, and we only compare rankings with $i \neq j$. Consequently, if there is a partially consistent k-ranking, there is also exactly one object $o_{i,u_i} = O(\pi_i^{(k-1)}) \setminus O(\pi_j^{(k-1)})$ and exactly one object $o_{j,u_j} = O(\pi_j^{(k-1)}) \setminus O(\pi_i^{(k-1)})$. Using terminology from sequence alignment, this means there are exactly two gaps in the alignment of the sequences $\pi_i^{(k-1)}$ and $\pi_j^{(k-1)}$, one in the former and one in the latter (Figure 1). However, the existence of one gap in each sequence is only a necessary but not a sufficient condition. Additionally, we must have $o_{i,u_i} \neq o_{j,u_j}$. For instance, the sequences in Figure 2 contain the same elements but in another order. In the alignment, there is one gap in each sequence, yet there is no consistent $(k-1)$-ranking.

The last observation is that the number of consistent k-rankings which can be constructed from $\pi_i^{(k-1)}$ and $\pi_j^{(k-1)}$ is one if $u_i \neq u_j$ and two if $u_i = u_j$.

Thus, the key tasks consist of finding and counting the gaps—see Algorithm 2 for a description in pseudo code. The meaning of top- and bottom are like in Figure 1.

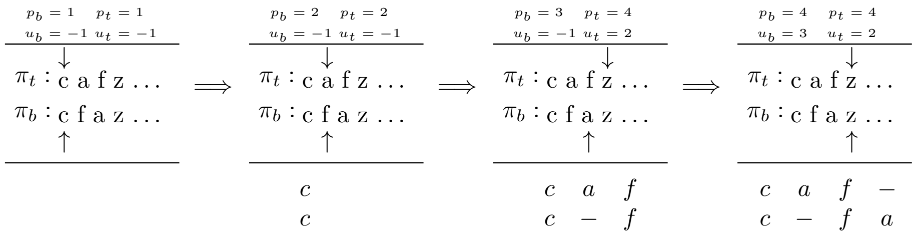

Fig. 2. Here, the $(k-1)$-rankings differ by a swap of the objects a and f. Although it is not possible to build a k-ranking, the alignment is the same as in Figure 1. Therefore, the objects o_{b,u_b} and o_{t,u_t} need to be checked for equality.

Algorithm 2.

```
procedure CONSTRUCT($\pi_b^{(k-1)}$, $\pi_t^{(k-1)}$)
    $u_b \leftarrow -1$; $u_t \leftarrow -1$                    ▷ bottom and top gap position
    $p_b \leftarrow 1$; $p_t \leftarrow 1$                      ▷ bottom and top object pointer
    for $p_b$ to $|\pi_b^{(k-1)}| - 2$ do
        if $o_{j,p_b} = o_{i,p_t}$ then
            $p_t \leftarrow p_t + 1$
        else if $o_{j,p_b} \neq o_{i,(p_t+1)}$ then
            if $u_b > -1$ then
                return null                             ▷ bottom gap already found
            end if
            $u_b \leftarrow p_b$                        ▷ bottom gap is found
        else
            if $u_b > -1$ then
                return null                             ▷ top gap already found
            end if
            $u_t \leftarrow p_t$ and $p_t \leftarrow p_t + 2$      ▷ the top gap is found
        end if
    end for
    ⋮  ▷ Here the procedure deals with the cases $p_b > |\pi_b^{(k-1)}| - 2$, where you have
       to be careful with the incrementation of $p_t$
    if $o_{b,u_b} = o_{t,u_t}$ then
        return null          ▷ necessary gaps are caused by a swap, see Figure 2
    end if
    return INNERCONSTRUCT($\pi_b^{(k-1)}$, $\pi_t^{(k-1)}$, $u_b$, $u_t$)
end procedure
```

While the "outer" CONSTRUCT procedure performs the alignment step, the INNERCONSTRUCT procedure performs the construction step. The idea here is simply to add o_{b,u_b} and o_{t,u_t} to $\pi_t^{(k-1)}$ with the help of the position information u_b and u_t, so that the resulting k-rankings are consistent with $\pi_b^{(k-1)}$ and $\pi_t^{(k-1)}$.

4.3 Filtering Frequent *k*-Rankings

Like in the original Apriori algorithm, we need to check for every potentially frequent k-ranking whether or not it is indeed frequent. For this purpose, we have to go through all rankings and count the appearance of the k-rankings.

4.4 Association Rule Mining

The mining of association rules of the form (2) is done on the basis of frequent subrankings, just like in itemset mining. However, as mentioned before, the conjunction $\pi_A \oplus \pi_B$ of two subrankings π_A and π_B is not again a subranking. Therefore, the support (5) of a candidate rule $\pi_A \rightharpoonup \pi_B$ cannot simply be looked up.

Instead, for each candidate rules $\pi_A \rightharpoonup \pi_B$, where π_A and π_B are frequent subrankings that meet the consistency constraint (3), we again pass through the data in order to compute the support (5) as well as measures of confidence (4) and interest (6).

5 Experiments

We used three real data sets for our experiments: The SUSHI data, that was already mentioned in the introduction, consists of 5000 rankings of 10 types of sushis. The STUDENTS data [2] comes from a psychological study and consists of 404 rankings of 16 goals (want to get along with my parents, want to feel good about myself, want to have nice things, want to be different from others, want to be better than others, etc.), each one reflecting what a student considers to be more or less important for himself or herself to achieve. Finally, the ESC14 data is derived from the European Song Contest 2014 in Denmark. It consists of rankings of the 26 countries that reached the final. Since each of the 36 participating countries selected 5 jurors, the total number of rankings is 180. There was a need for one adjustment, however: Since jurors are not allowed to rank their own country, we completed such rankings (of length 25 instead of 26) by putting that country on the bottom rank.

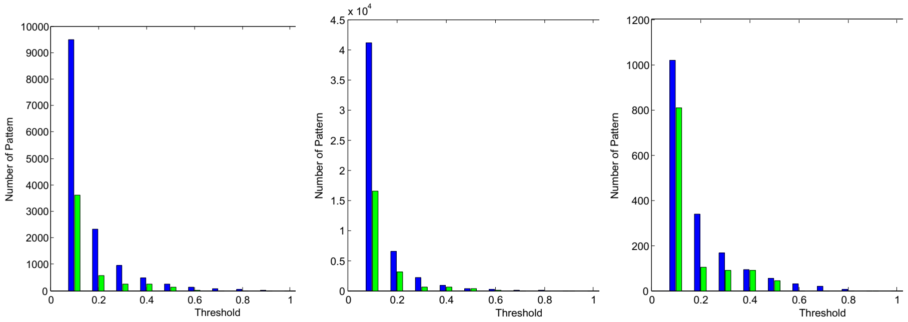

Fig. 3. Number of patterns reaching a threshold min_{supp} (left bar) for the data sets STUDENT, ESC14 and SUSHI (from left to right), compared to the same number for synthetic data sets of the same dimension, in which rankings are generated uniformly at random (right bar)

Figure 3 shows the number of frequent subrankings found in the data sets, compared with the number of frequent subrankings found in synthetic data sets taken from a uniform distribution.

For each of the three data set, we derived a most representative subranking, namely the subranking π that maximizes the relation between its support and the support one would *expect* under a uniform distribution (which is $1/|\pi|!$). Figure 4

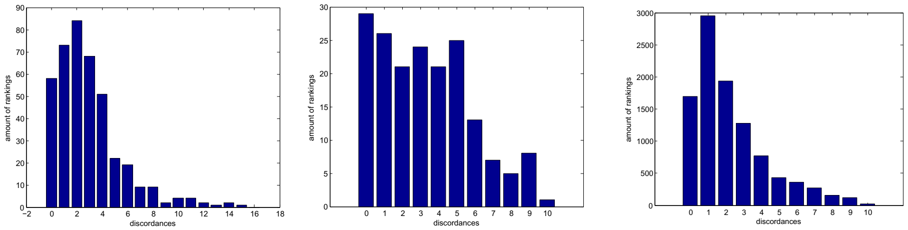

Fig. 4. Distribution of the distance of rankings from the most representative pattern; from left to right: STUDENTS (goal 2 $\succ$ goal 1 $\succ$ goal 9 $\succ$ goal 11 $\succ$ goal 14 $\succ$ goal 16), ESC14 (Netherlands $\succ$ Finland $\succ$ UK $\succ$ Italy $\succ$ Greece), SUSHI (sushi 8 $\succ$ sushi 3 $\succ$ sushi 9 $\succ$ sushi 7 $\succ$ sushi 10)

shows the distribution of the distances of all rankings from that representative, where the distance is determined as the number of pairwise disagreements. As can be seen, the pattern is indeed representative in the case of STUDENTS and SUSHI, in the sense that the large majority deviates by at most 2-3 pairwise inversions. For ESC14, a representative pattern is more difficult to find, suggesting that the preferences are more diverse in this case.

Finally, we also extracted association rules from the data sets, and found a number of rules with high confidence and interest (for example, the rule goal 10 (material gain) $\succ$ goal 3 (belongingness) $\succ$ goal 7 (management) $\rightharpoonup$ goal 10 (material gain) $\succ$ goal 5 (social responsibility) in the STUDENTS data with confidence 0.9038, interest 0.6544, and support 0.1149). Many of these rules also have a quite interesting semantic interpretation. Due to reasons of space, however, we refrain from a detailed discussion here.

6 Summary and Conclusion

In this paper, we introduced the problem of mining rank data as a novel data mining task—to the best of our knowledge, mining patterns in this type of data has not been studied systematically in the literature so far. Moreover, we have given two concrete examples of rank patterns, namely frequent subrankings and associations between such rankings, and proposed an algorithm for extracting them from rank data. Our algorithm is a rather straightforward generalization of the basic Apriori algorithm for itemset mining. Needless to say, there is much scope for improving this approach, so as to make it scalable to very large data sets. To this end, one may try to adopt ideas of faster algorithms for itemset mining, such as Eclat [11] or FP-growth [7,8], although the data structures used there are not immediately applicable to rank data.

More importantly, however, the problem of mining rank patterns itself can be generalized in various directions:

- For example, as already mentioned, rank information will not always be provided in the form of complete rankings of all items, i.e., the data itself may already be given in the form of subrankings, partial orders or "bags" of order relations.

- In this regard, one may also think of a combination of mining rank and itemset data. For instance, preference information is often given in the form of top-k rankings, i.e., a ranking of the k most preferred alternatives [4]—obviously, information of that kind can be seen as a *ranking* of a *subset* of all items.
- Since the space of rankings is equipped with a natural topology, it would make sense to search for *approximate patterns*, also allowing a ranking to be supported by *similar* rankings, for example [3].
- Yet another direction is the incorporation of quantitative information about rank positions. It could make a difference, for example, whether two objects a and b share adjacent ranks (suggesting that a is only slightly preferred to b), or whether a appears on the top and b on the bottom of the ranking.

Extensions and generalizations of that kind provide interesting challenges for future work.

Acknowledgments. The authors gratefully acknowledge support by the German Research Foundation (DFG). Moreover, they would like to thank Prof. Louis Fono, Université de Douala, Cameroun, for very stimulating discussions on the topic of this paper.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proc. VLDB, 20th Int. Conf. on Very Large Data Bases, pp. 487–499 (1994)
2. Boekaerts, M., Smit, K., Busing, F.M.T.A.: Salient goals direct and energise students' actions in the classroom. Applied Psychology: An International Review 4(S1), 520–539 (2012)
3. de Sá, C.R., Soares, C., Jorge, A.M., Azevedo, P., Costa, J.: Mining association rules for label ranking. In: Huang, J.Z., Cao, L., Srivastava, J. (eds.) PAKDD 2011, Part II. LNCS, vol. 6635, pp. 432–443. Springer, Heidelberg (2011)
4. Fagin, R., Kumar, R., Sivakumar, D.: Comparing top-k lists. SIAM Journal of Discrete Mathematics 17(1), 134–160 (2003)
5. Fürnkranz, J., Hüllermeier, E. (eds.): Preference Learning. Springer (2011)
6. Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: current status and future directions. Data Mining and Knowledge Discovery 15, 55–86 (2007)
7. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. ACM SIGMOD Record 29, 1–12 (2000)
8. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Mining and Knowledge Discovery 8(1), 53–87 (2004)
9. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. Springer (1996)
10. Suzuki, E.: Data mining methods for discovering interesting exceptions from an unsupervised table. J. Universal Computer Science 12(6), 627–653 (2006)
11. Zaki, M.J.: Scalable algorithms for association mining. IEEE Transactions on Knowledge and Data Engineering 12(3), 372–390 (2000)