



**HAL**  
open science

## Boosting the Learning for Ranking Patterns

Nassim Belmecheri, Nouredine Aribi, Nadjib Lazaar, Yahia Lebbah, Samir Loudni

► **To cite this version:**

Nassim Belmecheri, Nouredine Aribi, Nadjib Lazaar, Yahia Lebbah, Samir Loudni. Boosting the Learning for Ranking Patterns. *Algorithms*, 2023, 16 (5), pp.218. 10.3390/a16050218 . hal-04130835

**HAL Id: hal-04130835**

**<https://hal.science/hal-04130835>**

Submitted on 11 Oct 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

---

# BOOSTING THE LEARNING FOR RANKING PATTERNS

---

**Nassim Belmecheri**

Lab. LITIO

University of Oran1

Oran, Algeria

belmecheri.nassim@edu.univ-oran1.dz

**Noureddine Aribi**

Lab. LITIO

University of Oran1

Oran, Algeria

aribi.noureddine@gmail.com

**Nadjib Lazaar**

LIRMM

University of Montpellier

Montpellier, France

Nadjib.Lazaar@lirmm.fr

**Yahia Lebbah**

Lab. LITIO

University of Oran1

Oran, Algeria

ylebbah@gmail.com

**Samir Loudni**

TASC (LS2N-CNRS)

IMT Atlantique

Nantes, France

samir.loudni@imt-atlantique.fr

## ABSTRACT

Discovering relevant patterns for a particular user remains a challenging task in data mining. Several approaches have been proposed to learn user-specific pattern ranking functions. These approaches generalize well, but at the expense of the running time. On the other hand, several measures are often used to evaluate the interestingness of patterns, with the hope to reveal a ranking that is as close as possible to the user-specific ranking. In this paper, we formulate the problem of learning pattern ranking functions as a multicriteria decision making problem. Our approach aggregates different interestingness measures into a single weighted linear ranking function, using an interactive learning procedure that operates in either passive or active modes. A fast learning step is used for eliciting the weights of all the measures by means of pairwise comparisons.

This approach is based on Analytic Hierarchy Process (AHP), and a set of user-ranked patterns to build a preference matrix, which compares the importance of measures according to the user-specific interestingness. A sensitivity based heuristic is proposed for the active learning mode, in order to insure high quality results with few user ranking queries. Experiments conducted on well-known datasets show that our approach significantly reduces the running time and returns precise pattern ranking, while being robust to user-error compared with state-of-the-art approaches.

**Keywords** Interactive Data Mining, Machine Learning, Active Learning, Multicriteria decision making, Analytic Hierarchy Process.

## 1 Introduction

Data mining is the study of how to extract relevant information from data and exploit it as useful knowledge. One of its most important subfields, pattern mining, involves searching and enumerating interesting patterns in data. In the last decade, the pattern mining community has witnessed a sharp shift from efficiency-based approaches to methods that can extract more meaningful patterns. Unfortunately, obtaining interesting results with traditional pattern mining methods can be a tough and time-consuming task. The two main issues are that: 1) humongous amounts of patterns are found, of which many are redundant, and 2) preferences and background knowledge of the domain expert are not taken into account. The importance of taking user preferences and knowledge into account was first emphasized by Tuzhilin [1]. The main idea is to model the user's preferences using objective interestingness measures. Here, the interestingness of a pattern is computed from the data only. However, as stated in [2], using objective quality measures are of limited practical use, since interestingness depends on the specific user and task at hand.

In recent years, pattern mining has been challenged by an increasing focus on *user-centered, interactive, and anytime* pattern mining [3, 4, 5]. This new paradigm stresses that users should be presented quickly with patterns likely to

be interesting to them, and typically affect later iterations of the interactive mining process by giving feedback. An important aspect of this framework is the ability to learn user-specific pattern ranking functions from feedback. This idea was first investigated in [6] and recently extended by Boley et al. [4] and Dzyuba et al. [3, 5] in the context of interactive pattern mining. These approaches exploit standard machine learning techniques to learn weighted vectors according to some selected features on patterns (i.e. items, transactions, length . . .).

In [3], a linear ranking function is learned using RankingSVM. In [4, 5], the authors propose to use Stochastic Coordinate Descent (SCD) [7] to learn a logistic function. While these methods allow to exploit user feedback to learn ranking functions, they also result in a more computationally expensive learning task, particularly when the number of pairs of patterns and measures used for ranking increases. However, a vital ingredient of this framework is the ability to quickly present a set of patterns to the user and focus on what should be of interest using some interestingness measures.

In this paper, we tackle the problem of learning pattern ranking functions as a multicriteria decision making problem. We use a *weighted linear function* to aggregate all the individual measures into a global ranking function. A fast and scalable learning algorithm is used to maintain the right expected weights of measures. The proposed approach exploits the Analytical Hierarchy Process (AHP) [8], and a set of user-ranked patterns to learn the measures' weighting vector, that maximizes the correlation between the (unknown) user's ranking function and the learned AHP-based ranking function.

This approach was firstly introduced in our earlier work [9], where we show initial results of the passive version of the algorithm. More precisely, our new contributions compared with the previous version are as follows:

- A detailed presentation of the whole approach with running examples.
- A new generic version of the learning algorithm to learn both in active and passive modes.
- Proposition of a sensitivity based heuristic for selecting patterns when learning in the active mode.
- Implementing three different user simulations.
- Validation of the whole approach with its various settings (i.e., passive and active modes, user simulations, etc.) on large and new datasets.
- Showing the robustness of the approach in an interactive learning process, considering user mistakes and user changing its ranking criterion.

To assess the interest of our approach, we consider the association rules mining problem as a case study. Experiments conducted on a wide range datasets show that our approach significantly reduces the running time, while ensuring accurate pattern ranking compared to the state-of-the-art approaches.

The rest of the paper is organized as follows: Section 2 introduces the related work. Section 3 presents some necessary preliminaries. Section 4 formulates the tackled problem and describes our approach. Section 5 illustrates our approach on a running example. Empirical results are reported and discussed in Section 6. Section 7 concludes the paper.

## 2 Related work

Recent approaches for iterative/user-centric pattern mining are based on learning ranking functions throughout user pairwise pattern rankings. This problem is known as *object ranking*. Several methods have been developed for the object ranking task [10]. A common solving technique involves minimizing pairwise loss, e.g., the number of discordant pairs. In [6], the authors propose a log-linear model for itemsets to learn the user's prior knowledge from her feedback. They use RankingSVM formulations [11] to learn a ranking function. For more complex patterns (e.g., sequential patterns), a belief model is proposed, which exploits belief probabilities assigned to transactions. Later, Dzyuba et al. [12] extend the approach based on RankingSVM for active preference learning for ranking patterns. According to [13], the worst-case time complexity to solve the SVM problem varies between  $k^2$  and  $k^3$  on the number of samples  $k$ ; whereas our approach is linear on  $k$  (see section 4.4). In [4], the same approach is adopted to learn a logistic function using Stochastic Coordinate Descent (SCD) [7]. The same technique is also used in [5], but the learned function is exploited for patterns sampling. In [14], regression techniques are adopted to learn ranking functions.

Our approach is different and tackles the problem of learning pattern ranking functions as a multicriteria decision making problem using AHP. Moreover, our method remains fast when the number of patterns used for ranking increases.

### 3 Background

#### 3.1 Pattern mining and interestingness measures

Data can be abstracted as a triplet  $(\mathcal{O}, \mathcal{A}, \mathcal{L})$  in which  $\mathcal{O}$  is a set of objects,  $\mathcal{A}$  is a set of attributes and  $\mathcal{L}$  is the language used to express objects on attributes. This framework is inspired from inductive databases concepts [15, 16]. Following the language  $\mathcal{L}$ , for instance, we can get classical datasets: (1) when  $\mathcal{L}$  is a binary relation between attributes and objects (i.e.,  $\mathcal{L} \subseteq \mathcal{O} \times \mathcal{A}$ ), we get the classical itemset mining framework; (2) when  $\mathcal{L}$  expresses each object as a sequence of attributes, we get a sequential dataset; (3) when  $\mathcal{L}$  expresses each object as a graph where nodes and vertices are labeled, we get a dataset of graphs. Data mining tasks aim at discovering interesting regularities, namely patterns, between objects in large-scale datasets.

Measuring the interestingness of discovered patterns w.r.t. the user-specific preferences is an active field in Data Mining. Nine concepts have been introduced to determine whether a pattern is interesting or not: *Conciseness, Generality, Reliability, Peculiarity, Diversity, Novelty, Surprisingness, Utility* and *Applicability* [17]. Some concepts are correlated like *Surprisingness* and *Applicability* concepts, are conflicting like *Generality* and *Peculiarity*, or independent like *Novelty* and *Utility*. The user has in mind some of these concepts to say that she prefers a pattern than another. A wide range of measures, and aggregation of these measures, have been proposed to satisfy the user preferences.

#### 3.2 Analytical Hierarchy Process (AHP)

AHP is a well-known multicriteria decision-making method developed by Saaty [8]. It is based on structuring and synthesizing the decision problem into a problem hierarchy, and eliciting criteria weights from the decision maker through a series of pairwise comparisons, as opposed to utilizing numerical values directly. After constructing a hierarchy of the decision problem, the importance weights of criteria can be calculated according to the following three main steps:

1. A user is asked to compare the criteria of a given level in a pairwise manner to estimate their relative importance in relation to criteria at the preceding level. The pairwise comparisons are collected into a pairwise comparison matrix,  $\mathbf{A} = (a_{ij})_{m \times m}$ , with  $a_{ij}$  expressing the relative importance of criterion  $i$  to criterion  $j$ : the  $i^{\text{th}}$  criterion is better than the  $j^{\text{th}}$  criterion if  $a_{ij} > 1$ . The preference matrix  $\mathbf{A}$  is filled s.t.  $a_{ij} = 1/a_{ji}$  and  $a_{ii} = 1$ . For simplicity and easiness, AHP uses 9 degrees of preference, where  $a_{ij} = 1$  indicates an indifference and  $a_{ij} = 9$  an absolute preference.
2. Once the pairwise comparison matrix at a given level is built, the weight vector  $w = (w_1, \dots, w_m)^T$  can be computed using several mathematical techniques proposed in the literature. For example, the eigen vector method (EVM) or distance-based minimization methods, such as Least Squares Method, Logarithmic Least Squares Method, Weighted Least Squares Method, Logarithmic Least Absolute Values Method and Singular Value Decomposition [18, 19, 20, 21, 22]. The most used one is the Eigen Vector Method. The EVM method computes the weight vector  $w$  by solving the characteristic equation:

$$\begin{cases} \mathbf{A} \cdot w = \lambda_{max} \cdot w \\ w^T \mathbf{1} = 1 \end{cases} \quad (1)$$

where  $\mathbf{A}$  is the pairwise comparison matrix,  $\lambda_{max}$  is the highest eigen value of  $\mathbf{A}$ , and  $\mathbf{1} = (1, \dots, 1)^T$ . The constraint  $\sum_{i=1}^m w_i = 1$  is added to aid the solving process and avoid the infinitely many solutions. Note that, if  $\mathbf{A}$  is positive then the highest eigenvalue is real (i.e.  $\lambda_{max} \in \mathbb{R}$ ) [22]. Interestingly, Saaty [23] mentioned that the perfect<sup>1</sup> normalized eigenvector  $w$  satisfied the following relation:

$$a_{ij} = \frac{w_i}{w_j}, \forall i, j = 1 \dots m. \quad (2)$$

In this case  $\lambda_{max} = m$ , otherwise  $\lambda_{max} > m$ . The relation given by equation (2) explains why some researchers prefer the distance-based methods to *directly* consider to minimize the distance between  $(a_{ij})_{m \times m}$  and  $(w_i/w_j)_{m \times m}$ . However, the resulting problem is non-linear and difficult to solve. This is why we prefer, in this paper, to use the EVM method instead of distance-based methods.

Obviously, the hierarchy may contain more levels of criteria, especially when the number of criteria exceeds nine. One key idea to handle this particular case consists to establish a hierarchy structure based on existing criteria dependencies. It is important to generate an optimized decision problem hierarchy, to reduce the number of pairwise decisions. For more details, we refer the reader to [22]. Note that, in this paper, we will not cover this point.

<sup>1</sup>When the components of  $\mathbf{A}$  are exactly obtained as ratio between weights.

**Example 1** Consider a multicriteria decision problem with three criteria, for which we build a pairwise comparison matrix  $\mathbf{A}(a_{ij})_{3 \times 3}$  that compares the relative importance of criteria according to the achievement of an overall goal. This matrix is given by:

$$\mathbf{A} = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{4} \\ 2 & 1 & \frac{1}{2} \\ 4 & 2 & 1 \end{pmatrix}$$

To find the criteria weight vector, one could use the EVM method, which consists to solve the equation:  $\mathbf{A}w = \lambda_{max}w$ . The solution is given by the vector  $w = (w_1, w_2, w_3) = (\frac{1}{7}, \frac{2}{7}, \frac{4}{7})$  whose components are the weights of criteria.

## 4 AHP-based learning approach for ranking patterns

Our objectif is to learn a ranking function aggregating a set of interestingness measures while approximating the user-specific preferences.

### 4.1 Problem statement: learning pattern rankings

We show in this section that learning a ranking function from some interestingness measures w.r.t. user-specific preferences can be translated into a multicriteria problem. In our context, the criteria are interestingness measures. The task is to learn a function for ranking patterns from a sample of ordered patterns. Following previous research [24], we use *ordered feedback*, where a user is asked to provide a total order over a (small) number of patterns according to her subjective interestingness on them.

Let  $\triangleright$  be a binary preference relation between patterns. A ranking function based on  $\triangleright$ , denoted by  $r_{\triangleright}(\cdot)$ , accepts as input a subset of patterns  $\mathcal{P} \subseteq \mathcal{L}$  and returns as output a permutation  $S = \langle P_{r_1}, P_{r_2}, \dots, P_{r_n} \rangle$  of  $\mathcal{P}$ , s.t.  $|\mathcal{P}| = |S|$  and  $\forall i < j, P_{r_i} \triangleright P_{r_j}$ .

The user ranking function to learn is denoted by  $r_{\triangleright_u}$ , which is based on the user preference relation  $\triangleright_u$ . Given a set of interestingness measures  $\mathcal{M} = \{M_1, \dots, M_m\}$  and the corresponding ranking functions  $r_{\triangleright_{M_1}}, \dots, r_{\triangleright_{M_m}}$ . We denote by  $rank_{M_i}(P_j)$  the rank of  $P_j \in \mathcal{P}$  w.r.t. the interestingness measure  $M_i$ . Here,  $(P_k \triangleright_{M_i} P_l)$  means that  $M_i(P_k) \geq M_i(P_l)$ .

We propose to state the problem as a learning problem, where we have to find the weights  $w_i$  of a linear aggregation function over the measures from a set of ranked patterns that maximizes the correlation of the aggregation with the user ranking function  $r_{\triangleright_u}$ :

$$f(P) = \sum_{i=1..m} w_i \cdot M_i(P) \quad (3)$$

Our main idea is to exploit the user feedback to perform pairwise comparisons between measures in order to estimate their relative importance on the basis of individual scores of measures. The results of these pairwise comparisons are then collected into a pairwise comparison matrix used by AHP to compute the weights  $w_i$ .

#### Comparing a single measure to a user ranking:

Let  $S = \langle P_1, \dots, P_n \rangle$  be a user ranking. Let  $\mathcal{M} = \{M_1, \dots, M_m\}$  be some interestingness pattern measures. To evaluate the overall ranking accuracy of a given measure compared to a user ranking, we use the Kendall's W concordance coefficient [25]. Given a measure  $M_i$  and a user ranking  $S$ , we denote by  $K_i(S) \in [0, 1]$ , the Kendall's W between the ranking using  $M_i$  and the user ranking  $S$ , where the value 1 (resp., value 0) represents a perfect concordance (resp., no concordance):

$$K_i(S) = \frac{3\alpha}{(|S|^3 - |S|)}, \quad \alpha = \sum_{\ell=1}^{|S|} (R_\ell - \bar{R})^2, \quad (4)$$

where  $R_\ell = rank_{M_i}(P_\ell) + rank_u(P_\ell)$  with  $P_\ell \in S$ , and  $\bar{R} = \frac{1}{|S|} \sum_{\ell=1}^{|S|} R_\ell$ .

### Comparing two measures according to a user ranking:

Let  $M_i$  and  $M_j$  two measures to be compared according to a given user ranking  $S$ . Computing the gap  $\Delta_{i,j}(S) = K_i(S) - K_j(S)$  enables to know how much  $M_i$  is closer to the user ranking  $S$  than  $M_j$ : if  $\Delta_{i,j}(S) \geq 0$ , then  $M_i$  is closer, else the converse. If a set of user rankings  $\mathcal{S} = \{S_1, \dots, S_n\}$  is considered, comparing two measures  $M_i$  and  $M_j$  on  $\mathcal{S}$  comes to estimating the closest measure to  $\mathcal{S}$  by averaging the gaps on all the user rankings of  $\mathcal{S}$ :

$$\Delta_{i,j} = \frac{1}{n} \sum_{S_k \in \mathcal{S}} \Delta_{i,j}(S_k).$$

## 4.2 Learning weights process

Function 1 implements the learning process that computes a weight vector  $w$  (a weight for each measure) by exploiting the pairwise comparisons of measures. It takes as inputs a vector of pairs comparisons  $\Delta$  and the corresponding index  $l$ , a user ranking  $S$  and a set of measures  $\mathcal{M}$ . The learning step is done at line 2 where for each pair of measures, we compute the gap w.r.t the user ranking  $S$ . `learnWeights` is an incremental function that can take into account the previous  $l$  user rankings by updating  $\Delta$ . At line 5, the AHP matrix is built by scaling the  $\Delta_{i,j}$  comparisons to the permitted values (between 1/9 and 9). Once done, the criteria weight vector  $w$  is returned in line 12 by solving the minimization problem described in line 11, using the usual eigen vector method (EVM) [22].

---

**Function 1** `learnWeights`( $\langle \Delta, l \rangle, S, \mathcal{M}$ )

---

**In** : Set of measures  $\mathcal{M} = \{M_1, \dots, M_m\}$ ; user ranking  $S = \langle P_1, \dots, P_n \rangle$ ;

**In Out** : pairs comparisons and index parameter  $\langle \Delta, l \rangle$ ;

**Out** : Weight vector  $w$ ;

---

```

1 A[ $i, j$ ]  $\leftarrow 1, \forall i, j \in \{1, \dots, m\}$ ;
2 foreach  $M_i, M_j \in \mathcal{M} : i < j$  do
3   |  $\Delta_{i,j} \leftarrow \frac{l}{l+1} \Delta_{i,j} + \frac{1}{l+1} (K_i(S) - K_j(S))$ ;
4 end
5 foreach  $\Delta_{i,j} \in \Delta$  do
6   | if  $\Delta_{i,j} < 0$  then
7     | scale  $\Delta_{i,j}$  to  $(-9..-1)$ ; A[ $j, i$ ]  $\leftarrow |\Delta_{i,j}|$ ; A[ $i, j$ ]  $\leftarrow 1/|\Delta_{i,j}|$ ;
8   | else
9     | scale  $\Delta_{i,j}$  to  $(1..9)$ ; A[ $i, j$ ]  $\leftarrow \Delta_{i,j}$ ; A[ $j, i$ ]  $\leftarrow 1/\Delta_{i,j}$ ;
10 end
11  $w \leftarrow solve \left( \begin{array}{l} \mathbf{A} \cdot w = \lambda_{max} \cdot w \\ \text{subject to } \sum_{j=1}^m w_j = 1, \quad w_j > 0, \forall j = 1 \dots m. \end{array} \right)$ 
12 return  $w$ ;
```

---

Once the weight vector  $w$  is learned, we can compute the score of a given pattern  $P_i$  using the following weighted aggregation function:

$$g_w(P_i) = \sum_{M_i \in \mathcal{M}} w_i \text{scale}(M_i(P)) \quad (5)$$

where the *scale* function is used to adjust interestingness values of the different measures to a  $[0, 1]$  scale.

## 4.3 AHPRank algorithm

Function 1 implements the key concept of our approach. It enables to learn a weight vector  $w$  from a given user ranking  $S$ . Moreover, function 1 can easily be used in an iterative process and for a passive/active perspective using the incremental computation of  $\Delta$  in line 3. We propose AHPRank (algorithm 2) based on `learnWeights` function to learn a weighted aggregation function in a passive mode as well as in an active one. Algorithm 1 takes as input a set of measures  $\mathcal{M}$ , a set of user-ranked patterns  $\mathcal{S}$  and a triplet of parameter  $\langle \mathcal{P}, \theta, T \rangle$  corresponding respectively to a collection of patterns and two integers to use in the active mode. AHPRank starts by initializing the vector of pairs comparisons to zero in line 1. If AHPRank is fed with a non empty  $\mathcal{S}$ , then it acts in a passive mode by iterating and

calling `learnWeights` on the given user-ranked patterns  $\mathcal{S}$  (lines 3-6). Otherwise, the active mode is triggered by submitting  $T$  queries to the user (lines 7-15). Here, the user is asked to rank a subset of patterns proposed by a query generator. Our query generator is based on a heuristic that exploits the performance of the overall approach on the already learned rankings. More precisely, the active learning mode is performed through the following main iterative steps:

1. As a first step, a subset of patterns is randomly sampled from the global set of patterns  $\mathcal{P}$ : At line 10, `SamplePatterns` returns a sample  $\mathcal{P}'$  of  $\theta$  patterns.
2. Select a pair of patterns using `BasedHeuristic` function. The proposed heuristic aims to select a pair of patterns of good quality to speed up the convergence of the learning process (function 3 detailed in section 4.3.1).
3. Interacting with the user to get her preferences w.r.t to the patterns ranking query feedback (line 12).
4. Preference-based learning of the user ranking function (line 13), by calling `learnWeights` function.

It is important to stress that it is very usual that  $\mathcal{P}$  is of huge size making step 2 very costly. Step 1 enables us to avoid this complexity by sampling a reasonable subset of patterns. Moreover, it also introduces a kind of diversity on the patterns proposed to the user through step 2.

To be effective, our learning method must decide when to stop e.g. maximal number of iterations, albeit being open to any custom stopping criterion. We take into account the case where the user can stop at any time she feels satisfied, and we simulate the user's stopping point using a fixed number  $T$  of learning iterations.

---

**Algorithm 2** `AHPRank( $\mathcal{M}, \mathcal{S}, \langle \mathcal{P}, \theta, T \rangle$ )`

---

**In** :Set of measures  $\mathcal{M} = \{M_1, \dots, M_m\}$ ;  
Passive mode ( $\mathcal{S} \neq \emptyset$ ): Set of user-ranked patterns  $\mathcal{S} = \{S_1, \dots, S_n\}$ ;  
Active mode ( $\mathcal{S} = \emptyset$ ): Collection of patterns  $\mathcal{P}$ ; Sample size  $\theta$  ;  
Number of iterations  $T$ ;

```

1 foreach  $\Delta_{i,j} \in \Delta$  do  $\Delta_{i,j} \leftarrow \emptyset$ ;
2 if  $\mathcal{S} \neq \emptyset$  then
3   foreach  $S_k \in \mathcal{S}$  do
4      $w^t \leftarrow \text{LearnWeights}(\langle \Delta, k \rangle, S_k, \mathcal{M})$ ;           // Learn weights
5   end
6 end
7 else
8    $w^0 \leftarrow \mathbf{1}$ ; ;                                       // Weight vector
9   for  $t = 1, 2 \dots T$  do
10     $\mathcal{P}' \leftarrow \text{SamplePatterns}(\mathcal{P}, \theta)$ ;
11     $Q_t \leftarrow \text{SensitivityBasedHeuristic}(\mathcal{P}', w^{t-1}, \mathcal{M})$ ;
12     $S_t \leftarrow \text{AskRanking}(Q_t)$ ; // Ask user to rank the pair of patterns in  $Q_t$ 
13     $w^t \leftarrow \text{LearnWeights}(\langle \Delta, t \rangle, S_t^*, \mathcal{M})$ ; // Learn weights
14  end
15 end

```

---

#### 4.3.1 A Pattern Selection Heuristic for AHPRank

Learning in active manner is not an easy task, since it is crucial that each learning step improves the learned function. Selecting the right patterns to present to the user is what allows the algorithm to improve the learning process step by step. However generating the right set of patterns is NP-hard [26]. We propose a heuristic (Function 3) based on sensitivity analysis for AHP models [27]. The rationale behind our heuristic is to select a single pair of patterns that are close in terms of overall score predicted by the learned function, relative to the sum of the measures gaps:

$$\sigma(P_i, P_j) = \left| \frac{g_w(P_i) - g_w(P_j)}{\sum_{l=1}^m (M_l(P_i) - M_l(P_j))} \right|; i, j \in 1 \dots k. \quad (6)$$

Intuitively, a successful aggregation should have the following property: if two patterns are close in the overall predicted score, they should be close also in their measures values. Thus, the algorithm will be enforced into learning the correct

---

**Function 3** BasedHeuristic( $\mathcal{P}, w, \mathcal{M}$ )

---

**In** : Collection of patterns  $\mathcal{P}$ ; Weight vector  $w$ ; Set of measures  $\mathcal{M}$ ;

```
1  $\mathcal{P}' \leftarrow \text{Sort}(\mathcal{P}, w)$ ; // sort  $\mathcal{P}$  according to  $g_w$ 
2  $\text{Pair} \leftarrow \emptyset$ ;
3  $\sigma_{\min} \leftarrow +\infty$ ;
4 for  $i \in 1..|\mathcal{P}'| : P_i, P_{i+1} \in \mathcal{P}'$  do
5    $\text{score} \leftarrow \sigma(P_i, P_{i+1})$ ;
6   if  $\text{score} < \sigma_{\min}$  then
7      $\sigma_{\min} \leftarrow \text{score}$ ;
8      $\text{Pair} \leftarrow \{P_i, P_{i+1}\}$ ;
9 end
10 end
11 return  $\text{Pair}$ ;
```

---

preferences of the patterns when having the lowest  $\sigma$  value, since the selected pair is surely close in terms of  $g_w$  values and very distant on the measures values.

**Example 2** Given four interestingness measures  $\mathcal{M} = \{M_1, M_2, M_3, M_4\}$  and two patterns  $\{P_1, P_2\}$ . Suppose that the measure values for both patterns are as follows:  $P_1 = \langle 0.1, 0.2, 0.3, 0.7 \rangle$ ,  $P_2 = \langle 0.7, 0.3, 0.2, 0.1 \rangle$  and that the current learned weight vector is  $w = \langle 0.25, 0.25, 0.25, 0.25 \rangle$ . At this stage,  $g_w(P_1) = 0.26$  and  $g_w(P_2) = 0.26$ . Here  $P_1$  and  $P_2$  are equivalent w.r.t. the current state of  $g_w$ . However, the values of the four measures are different for the two patterns. This pair of patterns is an interesting candidate ( $\sigma(P_1, P_2) = 0$ ), where eliciting a preference on such a pair will make the learning closer to the user ranking.

#### 4.4 Complexity Analysis

**Proposition 1 (Time complexity)** Given a bounded number of interestingness measures, Algorithm 2 runs in  $O(nk)$  in passive learning and in  $O(n)$  in active learning.

**proof** Let  $\mathcal{S} = \{S_1, \dots, S_n\}$  be a set of  $n$  user rankings,  $k$  the size of the largest ranking  $S_t$  and  $m$  a bounded number of measures in  $\mathcal{M}$ . Once the pairwise comparison matrix  $\mathbf{A}$  is built, we could compute the preference vector of weights  $w$  using several mathematical techniques. We have used the eigen-vector based method (i.e., EVM) [22], for which the worst case time complexity is about  $O(m^3)$  [28]. The EVM approach is of constant time complexity  $O(1)$  since that  $m$  is constant. Complexity of computing the matrix at line 3 of the learnWeights function is  $O(mnk)$  since that Kendall's W is in  $O(k)$  (see Definition (4)). The complexity of lines 2, 5 and 11 in learnWeights are respectively  $O(nm^2 + mnk)$ ,  $O(m^2)$  and  $O(m^3)$ . Notice that in AHP, it is demonstrated in [29] that the number of criteria is recommended to be no more than seven  $\pm 2$ . That is, with  $m \leq 9$ , we get an asymptotic quadratic complexity of  $O(nk)$  when AHPRank is in passive mode. In other terms, the approach costs  $O(k)$  for each ranking in the given  $\mathcal{S}$ . Now, when AHPRank is in active mode where queries are pair of patterns ( $k = 2$ ), the complexity is linear on the number of queries submitted to the user  $O(n)$ .

It is important to stress that in practice, we have in general a set  $\mathcal{S}$  reduced to a large element in the passive learning mode ( $n = 1$  and a large  $k$ ), and to a set of  $n$  queries of pair of patterns (a large  $n$  and a  $k = 2$ ) in the active mode. This makes AHPRank running in a linear time (linear on  $k$  in passive and on  $n$  in active) in most of the cases. This low complexity makes AHPRank a fast approach to learn a user ranking function, which is supported by the experimental evaluation in section 6.

## 5 Running Example

To illustrate our approach, we consider an initial set of patterns  $\mathcal{P} = \{P_1, \dots, P_{10}\}$ , five interestingness measures  $\mathcal{M} = \{M_1, \dots, M_5\}$  and a user-ranking  $r_{\triangleright_u}$ . Columns of Table 1 show the user ranking for  $\mathcal{P}$ , the rankings given by  $M_i$  on  $\mathcal{P}$  and the AHPRank results. It is important to stress that no measure perfectly matches the user ranking.

The data in Table 1 is used by our algorithm to build the AHP Matrix and to learn a weight vector  $w$  over  $\mathcal{M}$ . For the sake of simplicity, let us call AHPRank in the passive mode with  $\mathcal{S} = \{S_a = \langle P_3, P_1, P_5, P_2, P_4 \rangle\}$ . Here, the user prefers  $P_3 \triangleright_u P_1 \triangleright_u P_5 \triangleright_u P_2 \triangleright_u P_4$ . AHPRank learns weights by calling learnWeights on  $S_a$  and by computing the correlation between the user rankings of  $S_a$  and the ones of the measures  $M_i$ . The matrix provided in Equation (7) shows the  $\Delta$

<i>user</i>	<i>S</i>	$M_1$ – rank	$M_2$ – rank	$M_3$ – rank	$M_4$ – rank	$M_5$ – rank	$g_w$ – rank
<b>1</b>	$P_7$	0.95 <b>1</b>	0.48 8	0.79 2	0.30 9	0.80 <b>1</b>	0.72 <b>1</b>
<b>2</b>	$P_3$	0.75 3	0.72 1	0.78 3	0.70 <b>2</b>	0.61 <b>2</b>	0.68 <b>2</b>
<b>3</b>	$P_6$	0.80 2	0.49 7	0.50 9	0.65 4	0.60 <b>3</b>	0.61 <b>3</b>
<b>4</b>	$P_1$	0.47 9	0.47 9	0.76 5	0.56 6	0.59 <b>4</b>	0.54 <b>4</b>
<b>5</b>	$P_8$	0.56 6	0.65 4	0.63 8	0.69 3	0.40 <b>5</b>	0.53 <b>5</b>
<b>6</b>	$P_{10}$	0.57 5	0.50 <b>6</b>	0.80 1	0.4 8	0.02 10	0.34 8
<b>7</b>	$P_5$	0.62 4	0.62 <b>5</b>	0.66 6	0.57 5	0.27 <b>7</b>	0.48 <b>7</b>
<b>8</b>	$P_2$	0.48 <b>8</b>	0.66 3	0.65 7	0.1 10	0.05 9	0.33 9
<b>9</b>	$P_4$	0.50 7	0.68 2	0.77 4	0.50 7	0.35 6	0.50 6
<b>10</b>	$P_9$	0.02 <b>10</b>	0.1 <b>10</b>	0.05 <b>10</b>	0.8 1	0.25 8	0.18 <b>10</b>

Table 1: Running example with 10 patterns and 5 measures.

values of each measure pairs. We recall that for  $|\mathcal{S}| > 1$ ,  $\Delta_{i,j}$  is an averaged value of  $\Delta_{i,j}(S_k)$  where  $S_k \in \mathcal{S}$ . Since we only have  $S_a$ ,  $\Delta_{1,2} = \Delta_{1,2}(S_a)$ . Here,  $\Delta_{1,2}(S_a)$  represents the gap between the two measure rankings  $M_1$  and  $M_2$  w.r.t the user ranking  $S_a$ . For having  $\Delta_{1,2}(S_a)$ , we need to compute Kendall’s W  $K_1(S_a)$  (resp.,  $K_2(S_a)$ ) between the ranking of  $M_1$  (resp.  $M_2$ ) compared to the user ranking  $S_a$ :  $\Delta_{1,2}(S_a) = (K_1(S_a) - K_2(S_a)) = (0.25 - 0.25) = 0$ . After scaling the values of  $\Delta$  to  $(-9.. -1)$  if negatives, and to  $(1..9)$  otherwise. We can see in Equation (7) that, for instance, measure  $M_5$  is better than  $M_4$  with a degree of 6. The same intensity is observed between  $M_5$  and  $M_3$ . We can also observe that  $M_3$  and  $M_4$  are indifferent.

$$\Delta = \begin{matrix} & \begin{matrix} (M1) & (M2) & (M3) & (M4) & (M5) \end{matrix} \\ \begin{matrix} (M1) \\ (M2) \\ (M3) \\ (M4) \\ (M5) \end{matrix} & \begin{pmatrix} & 0 & 0.42 & 0.42 & -0.43 \\ & & 0.20 & 0.28 & -0.25 \\ & & & -0.08 & -0.55 \\ & & & & -0.60 \end{pmatrix} \end{matrix} \xrightarrow{\text{Scaling}} \begin{matrix} & \begin{matrix} (M1) & (M2) & (M3) & (M4) & (M5) \end{matrix} \\ \begin{matrix} (M1) \\ (M2) \\ (M3) \\ (M4) \\ (M5) \end{matrix} & \begin{pmatrix} & 1 & 4 & 4 & -4 \\ & & 2 & 3 & -3 \\ & & & 1 & -6 \\ & & & & -6 \end{pmatrix} \end{matrix} \quad (7)$$

Afterwards, AHPRank computes the AHP matrix  $A$  through `learnWeights` function and the scaled  $\Delta$  matrix (i.e., the average correlation gap) as an input:

$$\mathbf{A} = \begin{matrix} & \begin{matrix} (M1) & (M2) & (M3) & (M4) & (M5) \end{matrix} \\ \begin{matrix} (M1) \\ (M2) \\ (M3) \\ (M4) \\ (M5) \end{matrix} & \begin{pmatrix} 1 & 1 & 4 & 4 & 1/4 \\ 1 & 1 & 2 & 3 & 1/3 \\ 1/4 & 1/2 & 1 & 1 & 1/6 \\ 1/4 & 1/3 & 1 & 1 & 1/6 \\ 4 & 3 & 6 & 6 & 1 \end{pmatrix} \end{matrix} \quad (8)$$

At the end, AHPRank computes the weighting vector  $w$  by solving the minimization problem. Thus, we find the learned weight vector  $w = (w_{M_1}, w_{M_2}, w_{M_3}, w_{M_4}, w_{M_5}) = (0.24, 0.24, 0.065, 0.065, 0.39)$ . This vector reflects the importance of each measure with respect to the achievement of the goal (user ranking function  $g_w$ ).

Now, we can use the AHP interestingness measure  $g_w$  of formula 5 to rank all the patterns  $\mathcal{P}$  provided in Table 1. The overall ranking accuracy of AHPRank on such example is of 91% where it is able to rank accurately the 5 first patterns, the seventh and the tenth ones.

## 6 Experiments

In this section, we experimentally evaluate our fast learning framework AHPRank for ranking patterns. First, we present our case study on association rules mining and different oracles simulating user-specific rankings. Then, we present the studied research questions, the experimental protocol and the obtained results.

### 6.1 Mining associations rules (ARs)

We evaluate our generic approach on a concrete pattern mining task, the association rule mining, one of the most important and well studied task in data mining, first introduced in [30].

An association rule is an implication of the form  $X \rightarrow Y$ , where  $X$  and  $Y$  are itemsets such that  $X \cap Y = \emptyset$  and  $Y \neq \emptyset$ .  $X$  represents the body of the rule and  $Y$  represents its head. The *frequency* of an itemset  $X$  in a dataset, denoted by  $freq(X)$ , is the number of transaction of the dataset containing  $X$ . The frequency of a rule  $X \rightarrow Y$  is the frequency of the itemset  $X \cup Y$ , that is,  $freq(X \rightarrow Y) = freq(X \cup Y)$ .

Several interestingness measures of ARs have been introduced like support, confidence, interest factor, correlation and entropy. Tan et al. [31] made an interesting study on the usefulness of the existing measures related to the application type. In particular, they identified seven independent groups (see Table 2) of consistent measures having similar properties.

Groups	Measures
1	<b>Yules Q</b> , Yules Y, Odds Ratio
2	<b>Cosine</b> , Jaccard
3	<b>Laplace</b> , Support
4	$\phi$ <b>coefficient</b> , CollectiveStrength, piatetskyShapiro's
5	<b>Goodman Kruskal's</b> , Gini Index
6	<b>Interest factor</b> , added value, Klossgen K
7	<b>certainty factor</b> , Mutual Information, Cohen's $\kappa$

Table 2: Independent Subjective Measure Groups

For our evaluation, we pick one measure per group to have 7 measures representing an independent set of measures (see measures in bold in table 2).

## 6.2 User feedback emulators

It is difficult to evaluate an interactive approach since users are scarce, for this reason we emulate the user feedback using three different objective target ranking functions:

- **RAND-EMU**: The user-specific ranking is equivalent to a random weighted aggregation function. For that, we generate for each measure  $M_i$  a random weight  $w_i \in [0, 1]$  such that  $\sum_{i \in 1..m} w_i = 1$ .
- **LEX-EMU**: The user-specific ranking follows a lexicographic order on the measures. That is, given a lexicographic order  $lex$  on  $\mathcal{M}$ :

$$lex(\mathcal{M}) = \langle l_1, \dots, l_m \rangle, s.t., \bigcup_{i \in 1..m} l_i = \mathcal{M}$$

Here, given two patterns  $(P_1, P_2)$ ,  $P_1$  is preferred to  $P_2$  iff  $(l_i(P_1) > l_i(P_2))$  or  $(l_i(P_1) = l_i(P_2) \wedge l_{i+1}(P_1) > l_{i+1}(P_2))$  (for  $i = 1$  to  $m - 1$ ).

- **CHI-EMU**:  $\chi^2$  as a statistical measure is a good candidate to emulate the user feedback [32].  $\chi^2$  is a rather complex function to approximate with non-trivial correlations, it is also a quality measure suggested in [5]. Here, we use  $\chi^2$  as the target user-specific ranking function over ARs. Given some association rule  $X \rightarrow Y$ , the  $\chi^2$  value is as follows:

$$\chi^2(X \rightarrow Y) = \frac{(freq(X \rightarrow Y) - \frac{freq(X)freq(Y)}{N})^2}{\frac{freq(X)freq(Y)}{N}} \quad (9)$$

with  $N$  the number of transactions of the dataset.

## 6.3 Research questions

Our evaluation aims to answer the following four research questions:

- **RQ1**: *Given a sample of ranked patterns, is it possible to infer the user-specific preferences over all the patterns? If yes, how much of data are required as input? How long does the learning process last?*
- **RQ2**: *How does the proposed new learning based on AHP compare with the baseline one using SVM when they are used to automatically learn user-specific ranking function?*

Dataset	$ \mathcal{T} $	$ \mathcal{I} $	Density(%)	Type of Data	#Rules
Hepatitis	137	68	50.00	Disease	0.5 M
Connect	67,557	129	33.33	Game steps	1 M
Mushroom	8,124	119	18.75	Species of mushrooms	1.5 M
T40	100,000	1,000	4.20	Synthetic dataset	2 M
Retail	88,162	16,470	0.06	Retail market basket data	2,5 M

T40:T40I10D100K

Table 3: Dataset Characteristics.

- **RQ3:** *How effective is AHPRank from an active learning perspective? Is the sensitivity heuristic a wise choice to use in query selection?*
- **RQ4:** *How effective is AHPRank from an interactive data mining perspective?*

## 6.4 Experimental protocol

### 6.4.1 Implementation settings

We have implemented in Java our AHPRank approach with its two modes: the passive mode denoted by AHPRank.0 and the active mode denoted by AHPRank.1. The code is publicly available at [github.com/lirmm/AHPRank](https://github.com/lirmm/AHPRank). We compared our approach to the state of the art, the SVM based approach RankingSVM [5]. We denote by RankingSVM.0 the passive version and by RankingSVM.1 the active one following [3]. All experiments were conducted on an Intel core i7, 2.4Ghz with 16Gb of RAM using an overall timeout of one hour.

### Metrics

We consider three metrics to evaluate the performance of our approach:

- the Spearman’s rank correlation coefficient  $\rho$  (sum of squared differences between learned  $rank_L$  and target  $rank_T$  pattern ranks over  $n$  patterns) in order to evaluate the overall ranking accuracy:

$$\rho = 1 - \frac{6 \sum_{i \in 1..n} (rank_L(P_i) - rank_T(P_i))^2}{n(n^2 - 1)}, \quad (10)$$

- the Recall metric  $R@k$  in order to evaluate the effectiveness of our approach in identifying the top  $k$  most interesting patterns:

$$R@k = \frac{|\{rank_L(P_i) \leq k : i \in 1..n \wedge rank_T(P_i) \leq k\}|}{k} \quad (11)$$

- CPU time needed to finish the whole learning process and the waiting time between two queries for the active mode (in seconds).

## 6.5 Benchmark Datasets

We selected several real-sized datasets from the FIMI repository.<sup>2</sup> These datasets have various characteristics representing different application domains. Table 3 reports for each dataset the number of transactions  $|\mathcal{T}|$ , the number of items  $|\mathcal{I}|$ , its application domain, and the number of valid rules (#Rules) corresponding to the initial rules minded using a standard association rules algorithm and without any knowledge about the user. The datasets are presented by increasing order of #Rules.

## 6.6 Passive learning results

In this section, we address the two first research questions (**RQ1** and **RQ2**). To that end, we perform a 5-folds cross-validation on #Rules for each dataset. In each fold, we randomly select 20% of #Rules to form the training data and use the remaining 80% ARs for evaluation (testing data). We have chosen this way of using k-folds cross-validation in order to see how effective are the approaches in learning from relatively small training sets.

<sup>2</sup>[fimi.uantwerpen.be/data/](http://fimi.uantwerpen.be/data/)

Datasets	measures							
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	VBM
RAND-EMU								
Hepatitis	0.78	0.92	0.29	<b>0.97</b>	0.79	0.85	0.45	<b>0.97</b>
Connect	0.68	0.62	<b>0.95</b>	0.70	0.60	0.45	0.57	<b>0.95</b>
Mushroom	0.84	<b>0.97</b>	0.28	0.83	0.55	0.73	0.36	<b>0.97</b>
T40	0.71	<b>0.99</b>	0	0.76	<b>0.99</b>	<b>0.99</b>	0	<b>0.99</b>
Retail	0	<b>0.98</b>	0.43	0.71	0.84	<b>0.98</b>	0.51	<b>0.98</b>
LEX-EMU								
Hepatitis	0.79	0.64	0	0.68	<b>0.92</b>	0.63	0	<b>0.92</b>
Connect	0.21	0.47	<b>0.76</b>	0.26	0	0.23	0.50	<b>0.76</b>
Mushroom	0.21	<b>0.82</b>	0.38	0.77	0.78	0.37	0	<b>0.82</b>
T40	0.76	0.98	0	0.77	<b>0.99</b>	0.97	0	<b>0.99</b>
Retail	0.10	0.78	0.54	0.49	<b>0.84</b>	0.80	0.58	<b>0.84</b>
CHI-EMU								
Hepatitis	0.20	0.92	0.36	<b>0.96</b>	0.73	0.85	0.43	<b>0.96</b>
Connect	0.09	0.15	0	0.02	<b>0.29</b>	0.01	0	<b>0.29</b>
Mushroom	0.51	0.64	0	0.46	0.28	<b>0.88</b>	0	<b>0.88</b>
T40	0.71	0.98	0.17	0.64	0.98	<b>0.99</b>	0.29	<b>0.99</b>
Retail	0	<b>0.94</b>	0.53	0.60	0.83	0.57	0.61	<b>0.94</b>
(1): YulesY (2): Cosine (3): Laplace (4): Leverage (5): Lambda (6): InterstFactor (7): Certainty								

Table 4: Correlation results with user ranking for using the emulators.

**A) Analysing the different user feedback emulators.** Table 4 reports the rank correlation  $\rho$  between the user-specific ranking functions (RAND-EMU, LEX-EMU and CHI-EMU) and the seven interestingness measures. We also report VBM, the *Virtual Best Measure*, which returns the best rank correlation  $\rho$  provided by one of the seven measures. We observe from table 4 that given a measure, the results are chaotic. For instance, Lambda measure is highly correlated to CHI-EMU on T40 dataset ( $\rho = 98\%$ ) but it is weakly correlated on Connect ( $\rho = 29\%$ ) dataset. The same observation can be made with RAND-EMU and LEX-EMU functions. However, the theoretical construction of VBM is of a high accuracy (mean of 91%), which brings us to believe that a weighted aggregation of the selected measures can lead to a good trade-off.

**B) Comparing AHPRank.0 with RankingSVM.0.** Table 5 is dedicated to the k-folds cross-validation results of RankingSVM.0 and AHPRank.0 using averaged rank correlation  $\rho$ , recall values (R@10% and R@1%) and CPU time in seconds averaged over the folds. Under a time contrast of one hour, RankingSVM.0 is able to deal with a training set not exceeding 100K rules, and thus we are able to compare it with our approach only on Hepatitis. In terms of ranking accuracy, we observe that RankingSVM.0 outperforms AHPRank.0 for all user feedback emulators. However, AHPRank.0 remains competitive with an acceptable accuracy. Same observations can be made in terms of recall at the 10% and 1% top of the ranking (R@10% and R@1%). AHPRank.0 is able to reach a high correlation with the user ranking functions on most of the datasets. However, we can also observe a weak correlation with CHI-EMU on Connect, where a high correlation is observed on that dataset with RAND-EMU and LEX-EMU. This stems from the fact that RAND-EMU and LEX-EMU are linear functions expressed with the given 7 interestingness measures, which explains the high accuracy. However, CHI-EMU is a complex function and the statistics extracted from Connect on the 7 measures are not sufficient to learn such a function. However, the high accuracy of RankingSVM.0 is at the expense of the running time. For a training set of 100K rules, RankingSVM.0 needs more than 15 minutes to learn. Exceeding 100K rules, RankingSVM.0 needs more than one hour, where AHPRank.0 is able to deal with 7.5M of rules in a time not exceeding 4 minutes.

In what follows, the observations and the conclusions drawn from CHI-EMU remain true for RAND-EMU and LEX-EMU. For the sake of simplicity, we only report the results on the complex function CHI-EMU.

RAND-EMU								
	RankingSVM.0				AHPRank.0			
	$\rho$	R@10%	R@1%	t(s)	$\rho$	R@10%	R@1%	t(s)
Hepatitis	0.94	0.74	0.77	851	0.93	0.79	0.71	9
Connect	-	-	-	TO	0.95	0.72	0.63	26
Mushroom	-	-	-	TO	0.89	0.69	0.66	32
T40	-	-	-	TO	0.99	0.93	0.58	63
Retail	-	-	-	TO	0.93	0.95	0.96	66
<i>mean</i>	-	-	-	-	0.94	0.82	0.71	39

LEX-EMU								
	RankingSVM.0				AHPRank.0			
	$\rho$	R@10%	R@1%	t(s)	$\rho$	R@10%	R@1%	t(s)
Hepatitis	0.99	0.99	0.99	902	0.92	0.86	0.81	13
Connect	-	-	-	TO	0.60	0.46	0.50	21
Mushroom	-	-	-	TO	0.86	0.56	0.65	32
T40	-	-	-	TO	0.99	0.93	0.46	71
Retail	-	-	-	TO	0.78	0.68	0.65	68
<i>mean</i>	-	-	-	-	0.83	0.70	0.61	41

CHI-EMU								
	RankingSVM.0				AHPRank.0			
	$\rho$	R@10%	R@1%	t(s)	$\rho$	R@10%	R@1%	t(s)
Hepatitis	0.99	0.97	0.92	979	0.94	0.90	0.77	10
Connect	-	-	-	TO	0.15	0.28	0.67	17
Mushroom	-	-	-	TO	0.98	0.91	0.93	28
T40	-	-	-	TO	0.99	0.99	0.96	84
Retail	-	-	-	TO	0.91	0.96	0.95	50
<i>mean</i>	-	-	-	-	0.79	0.81	0.86	38

Table 5: 5-folds cross-validation results (passive learning).

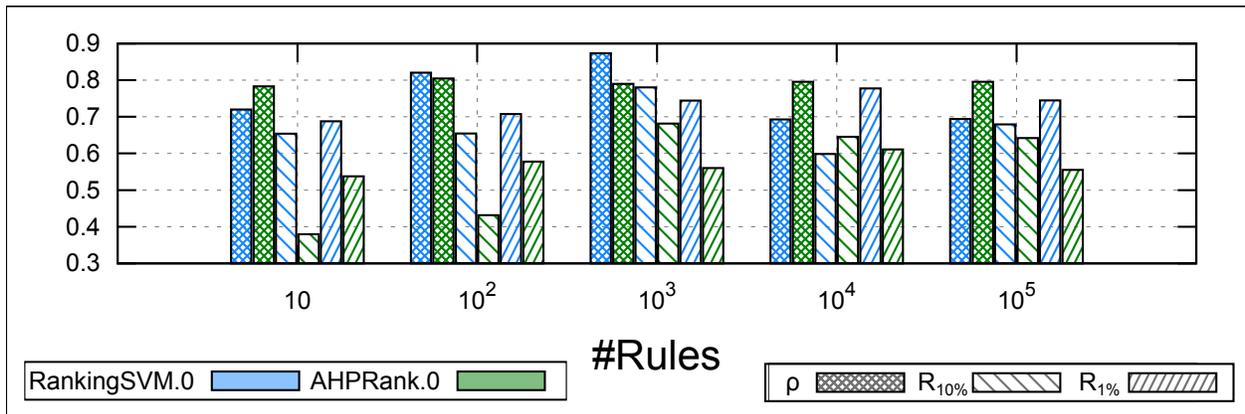


Figure 1: Learning accuracy comparison between RankingSVM.0 and AHPRank.0 learning CHI-EMU on different training data size.

**C) Impact of varying the size of the training data on the learning.** It is important to stress that the proposed AHPRank aims to ensure a fast learning process to reveal a pattern ranking function, while offering good accuracy. To strengthen our observation regarding the performance scalability of AHPRank.0 compared to RankingSVM.0, we report in figures 1 and 2 a performance comparison by varying the size of the training data on learning CHI-EMU function. Similar findings have been made on RAND-EMU and LEX-EMU functions. For each dataset, we select randomly  $nb$  rules and we call the two approaches ( $nb \in \{10, 100, 1K, 10K, 100K\}$ ). The results are averaged on ten runs.

In terms of Spearman correlation  $\rho$ , figure 1 shows a discrepancy of 5% between the two approaches when the training data is not exceeding  $1K$ . However, the gap becomes particularly important (exceeding 10%) and in favor of AHPRank.0 when the training data contains  $10K$  and  $100K$  rules. In terms of recall at 10% and 1%, RankingSVM.0 outperforms AHPRank.0 with a gap of 23% and 15% at R@10% and R@1% when the training data contains only 10 rules. The gap get tighter when the training data is increasing in size at R@10% (less than 5%) and remains more or less the same for R@1%.

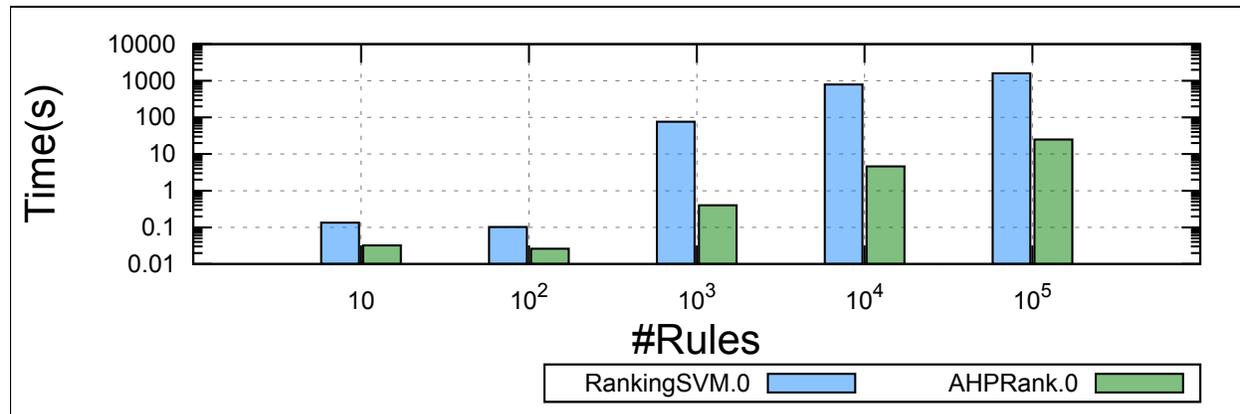


Figure 2: CPU time comparison between RankingSVM.0 and AHPRank.0 learning using CHI-EMU emulator on different training data sizes averaged overall datasets.

In terms of CPU time, figure 2 shows that RankingSVM.0 needs less than a second to deal with a training data not exceeding 100 rules, more than one minute for  $1K$  rules,  $13min$  for  $10K$  rules, and more than  $26min$  for  $100K$  rules. For comparison, AHPRank.0 is able to deal with a training data of a size ranged between 100 and  $100K$  in a time ranged between 0.03 and 24.86 seconds for an overall ranking accuracy close to the one reported by RankingSVM.

## 6.7 Active learning results

In this section, we attempt to answer the two research questions (**RQ2** and **RQ3**) where we compare the active learning versions of RankingSVM and AHPRank. For that end, we consider a simple process, where we ask the user some ranking queries on pairs of patterns (e.g. *do you prefer  $P_i$  to  $P_j$ ?*).

**A) Evaluating the effectiveness of our sensitivity-based heuristic.** Our first experiment aims to evaluate the usefulness of our Sensitivity-Based Generator (SBG) (see section 4.3). For that, we compare SBG to a Random Generator (RG), where the latter selects randomly a pair of patterns from #Rules to submit to the both learners RankingSVM.1 and AHPRank.1. For our experiment and with a human-in-the-loop, we set the number of queries (i.e., number of iterations  $T$  of algorithm 2) to a reasonable number not exceeding 20 queries. We repeat 10 times the experiment and take the average result in order to avoid the bad-luck effect with RG and the sampling step of our SBG. After few tests, we set the size of the sample  $\mathcal{X}$  picked at line 10 of algorithm 2 to  $\theta = 10^3$ . A sample size that provide a good trade-off between time selection and the accuracy of the selected pair.

Figure 3 reports a comparison between RankingSVM.1 and AHPRank.1 in a scatter plot of 20 iterations. We report the Spearman correlation  $\rho$  and the recall at 10% and 1% of each iteration. We observe from the scatter plot that the use of SBG outperforms RG either with our approach or with RankingSVM.1.

In order to support our observation on the use of SBG, we carried out a statistical test using the *Wilcoxon Signed-Rank Test*. Here, the one-tailed alternative hypothesis is used with the following null hypothesis (*RG is more efficient than SBG*):  $H_0$  : The accuracy using RG  $\geq$  The accuracy using SBG. That is, the alternative hypothesis  $H_1$  states that our SBG outperforms RG. With this statistical test, we are able to conclude that the use of SBG is more efficient than RG ( $H_1$  accepted). Table 6 reports the *p-value*, the *z-score* and the confidence interval (CI) of each test. According to the *p-value* column, and except the case of (RankingSVM.1,R@10%), we have strong evidence to reject the null hypothesis. The confidence interval column shows clearly that the use of SBG is better than RG. Consequently, we will use in what follows the SBG heuristic in RankingSVM.1 and AHPRank.1.

**B) Comparing AHPRank.1 with RankingSVM.1.** Table 7 shows a comparison between RankingSVM.1 and AHPRank.1 on the 5 datasets. We report  $\rho$ , R@10% and R@1% within stopping criterion  $T$  at 10, 50 and 100 queries. We also

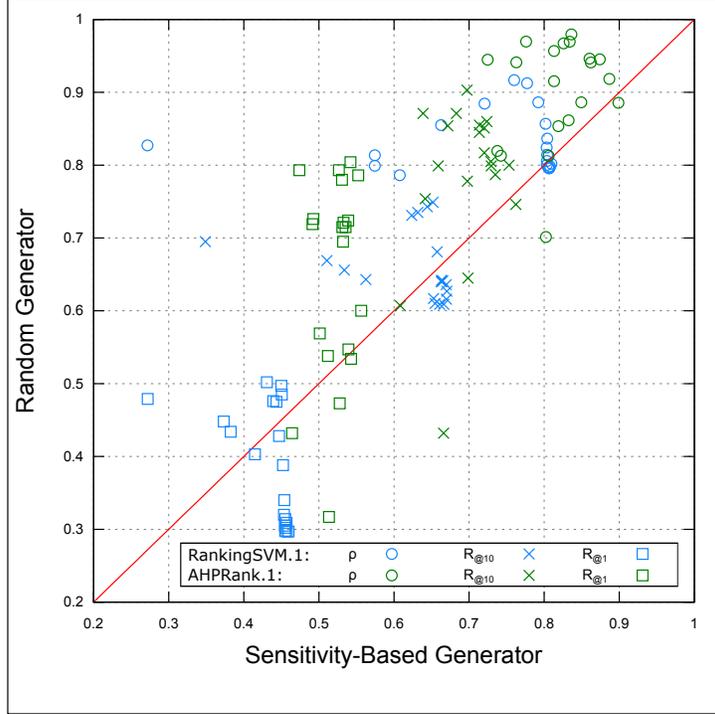


Figure 3: Random vs Sensitivity-Based Patterns Generator.

Approaches	Metrics	<i>p-value</i>	<i>z-score</i>	CI
RankingSVM.1	$\rho$	.00226	-2.8373	99%
	R@10%	.12302	-1.1573	70%
	R@1%	.06301	-1.5306	90%
AHPRank.1	$\rho$	.00034	-3.3973	99.9%
	R@10%	.00205	-2.8746	99%
	R@1%	.00139	-2.9866	99%

Table 6: Wilcoxon Signed-Rank Test (RG vs SBG).

report the averaged latency time between two queries *Time* for RankingSVM.1, knowing that *Time* never exceeds the latency bound of 0.1 seconds. The findings are the same for RAND-EMU, LEX-EMU and CHI-EMU functions.

The main observation that we can draw from table 7 is that AHPRank.1 outperforms RankingSVM.1. Let us take a close look at *Hepatitis* results. For this dataset, AHPRank.1 is able to rank 500K patterns with an accuracy of 78% using only 10 queries (73% observed with RankingSVM.1). The accuracy increases to 95% when learning on 50 queries for AHPRank.1 and 94% for RankingSVM.1. AHPRank.1 reaches 98% of accuracy with 100 queries while RankingSVM.1 remains stable at 94%. In terms of recall values, AHPRank.1 is able to discover the most relevant patterns on the first 50K and 5K patterns (out of 500K) with an accuracy of, respectively, 78% and 66% over 10 queries (61% and 44% for RankingSVM.1). Let us now take a close look at *Retail* dataset. We observe that AHPRank.1 reaches a high accuracy over 10 queries and such accuracy remains stable over the following 90 queries. The RankingSVM.1 results are less impressive on *Retail* and particularly on the recall metric. But the main difference is on the waiting time *Time*, where RankingSVM.1 can keep the user waiting more than 10 seconds between two queries. Same observation can be made on the other datasets where RankingSVM.1 can be hampered by overall waiting time even with queries of size 2, where AHPRank.1 shows an instantaneous behavior (less than 0.1 seconds between two queries). This represents a limitation in the use of RankingSVM.1 especially when the learning is integrated to an interactive data mining process, where a reasonable latency time for a human user is around few seconds [33].

		(a)	(b)	(c)	(d)	(e)
<b>10 Queries</b>						
(1)	$\rho$	0.73	-0.04	0.69	0.99	0.97
	R@10%	0.61	<b>0.28</b>	0.38	0.92	0.91
	R@1%	0.44	0.49	0.36	0.45	0.67
	Time	0	0	0	0	0
(2)	$\rho$	<b>0.78</b>	<b>0.10</b>	<b>0.93</b>	0.99	<b>0.99</b>
	R@10%	<b>0.66</b>	0.36	<b>0.80</b>	0.97	0.91
	R@1%	<b>0.51</b>	<b>0.50</b>	<b>0.71</b>	<b>0.81</b>	<b>0.93</b>
<b>50 Queries</b>						
(1)	$\rho$	0.94	<b>0.20</b>	0.71	0.99	0.98
	R@10%	0.77	0.10	0.39	0.92	0.91
	R@1%	0.57	0	0.36	0.45	0.67
	Time	<b>0</b>	<b>0</b>	<b>0</b>	2	3
(2)	$\rho$	<b>0.95</b>	0.10	<b>0.93</b>	0.99	<b>0.99</b>
	R@10%	<b>0.83</b>	<b>0.40</b>	<b>0.72</b>	<b>0.94</b>	0.91
	R@1%	<b>0.66</b>	<b>0.45</b>	0.51	<b>0.64</b>	<b>0.93</b>
<b>100 Queries</b>						
(1)	$\rho$	0.94	<b>0.29</b>	0.72	0.99	0.98
	R@10%	0.75	0.20	0.41	0.92	0.91
	R@1%	0.53	0	0.36	0.45	0.67
	Time	0	2	3	3	10
(2)	$\rho$	<b>0.98</b>	0.10	<b>0.81</b>	0.99	<b>0.99</b>
	R@10%	<b>0.89</b>	<b>0.35</b>	<b>0.49</b>	<b>0.95</b>	0.91
	R@1%	<b>0.78</b>	<b>0.42</b>	<b>0.37</b>	<b>0.74</b>	<b>0.93</b>
					(1): RankingSVM.1	(2): AHPRank.1
		(a): Hepatitis	(b): Connect	(c): Mushroom	(d): T40	(e):Retail

Table 7: Qualitative evaluation of RankingSVM.1 vs AHPRank.1.

## 6.8 Interactive learning results

In this section, we address the last research questions **RQ4**. In addition to the active learning results, which are part of the interactive perspective with a human-in-the-loop, we conduct two more experiments by focusing on the hazard characteristics linked to the presence of a human in the learning process.

Our first experiment aims to evaluate the robustness of our approach in front of a human likely to make mistakes. To simulate such situations involving incorrect user feedback, we pick randomly (with a percentage ratio  $Err$ ) a set of queries and we swap the user preference. That is, a user that prefers a pattern  $P_i$  to  $P_j$  will make a mistake by preferring  $P_j$  to  $P_i$  with a probability  $Err$ .

Figure 4 reports a comparison between RankingSVM.1 and AHPRank.1 under mistakes present with a probability  $Err$  varying from 0% to 40%. The reported results are averaged on 10 runs and on the whole set of datasets on learning CHI-EMU and by submitting 20 queries to the user. We observe that AHPRank.1 is quite stable and robust even with 40% of mistakes (8 mistakes out of 20). The overall correlation between the learned function and the user one  $\rho$  is at 77% without mistakes and becomes 70% under  $Err=40\%$ . However, RankingSVM.1 is stable till  $Err=20\%$  and then, the accuracy is highly impacted where  $\rho$  decreases from 71% to 54%. In terms of recall, we observe a decline not exceeding 2% at R@10% and 8% at R@1% under AHPRank.1. Using RankingSVM.1, the decline is exceeding 10% at R@10% and R@1%. In terms of CPU time and between two queries, we observe a waiting time ranged between 6 and 12 seconds under RankingSVM.1, where under our approach is never over 0.02 seconds.

Our second experiment aims to evaluate the robustness of our approach in front of an undecided user that starts with a set of preferences  $A$  in mind to finish with preferences  $B$ . Here, the initial user preferences  $A$  are biased along the presented patterns till being the user more comfortable with preferences  $B$ . To simulate such situations where the user target function is changing during the learning process, we conduct an experiment where we start with LEX-EMU as a target to learn and after  $x$  queries, we swap to RAND-EMU where the learning process takes  $(20-x)$  more queries (we coined it LEX2RAND target). Note that we choose a swap from LEX-EMU to RAND-EMU because of the two functions are linear. For a total number of 20 queries, we compare RankingSVM.1 and AHPRank.1 on LEX2RAND( $x$ ) target ( $x \in \{0, 5, 10\}$ ). Note that with  $x=0$ , LEX2RAND(0) is equivalent to RAND-EMU. Figure 5 reports the averaged

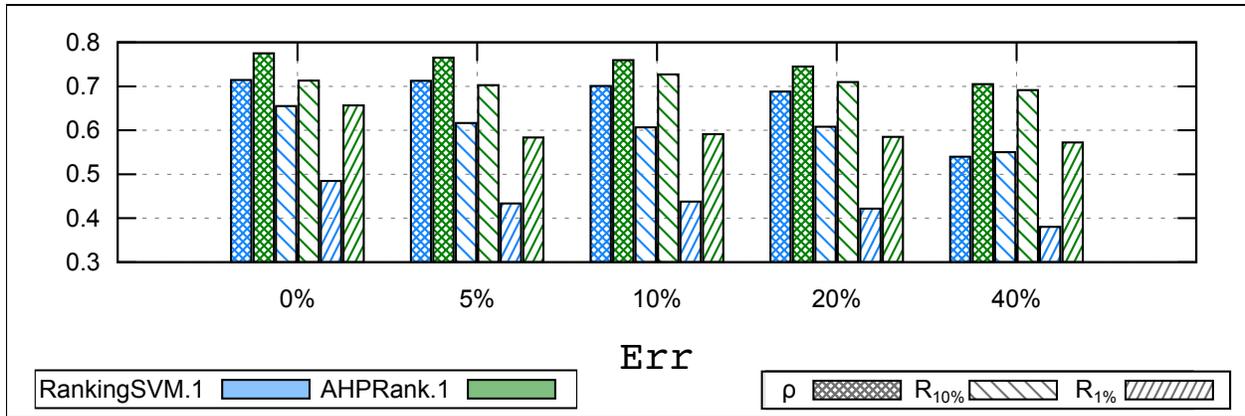


Figure 4: Learning under mistakes (20 queries).

results of 10 runs on all datasets over 20 queries. The main observation that we can draw is that AHPRank.1 is stable in front of a changing linear function to learn even with a changeable mind in the middle of the learning (10 out of 20 queries). However, a target alteration during the learning process can drastically impacts the accuracy of RankingSVM.1 (decline of 44% in terms of  $\rho$ ). In terms of CPU time, our approach is 50 times faster than RankingSVM.1.

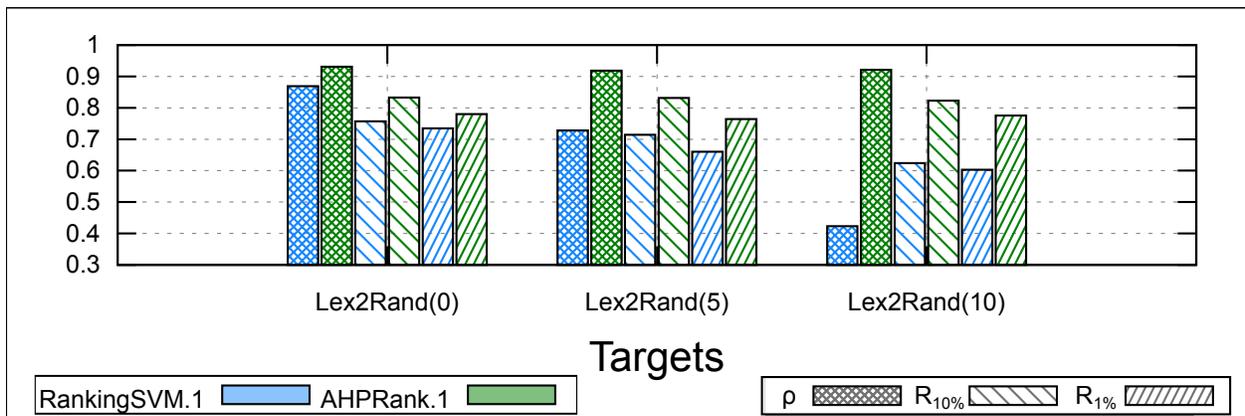


Figure 5: Learning under swaps (20 queries)

To sum up, our main finding is that AHPRank represents a good trade-off between ranking accuracy and running time. AHPRank is able to reach a good overall ranking accuracy within a time being linear on the size of the training data and thus, not exceeding few seconds, which makes it suitable for industrial exploitation.

## 7 Conclusion

We presented a generic and efficient framework for learning pattern ranking functions using a multicriteria decision making method based on AHP. The proposed AHPRank algorithm can be applied in both passive and active learning modes. It requires the user to rank subsets of patterns according to her preferences. Our approach exploits the learned weights to aggregate all the measures into a single ranking function, using a weighted linear formulation. The resulting aggregation function was experimentally and statistically proven to be as close as possible to the user ranking. We applied this framework to the case study of Association Rules Mining, for which experiments showed that AHPRank algorithm is able to learn the ranking function efficiently compared with the results of state-of-the-art learning approaches.

## References

- [1] Abraham Silberschatz and Alexander Tuzhilin. On subjective measures of interestingness in knowledge discovery. In *KDD*, pages 275–281. AAAI Press, 1995.
- [2] Tijl De Bie. An information theoretic framework for data mining. In *KDD*, pages 564–572. ACM, 2011.
- [3] Vladimir Dzyuba and Matthijs van Leeuwen. Interactive discovery of interesting subgroup sets. In *International Symposium on Intelligent Data Analysis*, pages 150–161. Springer, 2013.
- [4] Mario Boley, Michael Mampaey, Bo Kang, Pavel Tokmakov, and Stefan Wrobel. One click mining: interactive local pattern discovery through implicit preference and performance learning. In *IDEA@KDD*, pages 27–35. ACM, 2013.
- [5] Vladimir Dzyuba and Matthijs van Leeuwen. Learning what matters - sampling interesting patterns. In *PAKDD (1)*, volume 10234 of *Lecture Notes in Computer Science*, pages 534–546, 2017.
- [6] Dong Xin, Xuehua Shen, Qiaozhu Mei, and Jiawei Han. Discovering interesting patterns through user’s interactive feedback. In *KDD*, pages 773–778. ACM, 2006.
- [7] Shai Shalev-Shwartz and Ambuj Tewari. Stochastic methods for  $l_1$ -regularized loss minimization. *J. Mach. Learn. Res.*, 12:1865–1892, 2011.
- [8] Thomas L Saaty. What is the analytic hierarchy process? In *Mathematical models for decision support*, pages 109–121. Springer, 1988.
- [9] Nassim Belmecheri, Nouredine Aribi, Nadjib Lazaar, Yahia Lebbah, and Samir Loudni. Une méthode d’apprentissage par optimisation multicritère pour le rangement de motifs en fouille de données. *Revue des Nouvelles Technologies de l’Information*, Extraction et Gestion des Connaissances, RNTI-E-38:289–296, 2022.
- [10] Toshihiro Kamishima, Hideto Kazawa, and Shotaro Akaho. A survey and empirical comparison of object ranking methods. In *Preference Learning*, pages 181–201. Springer, 2010.
- [11] Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. Adapting ranking SVM to document retrieval. In *SIGIR*, pages 186–193. ACM, 2006.
- [12] Vladimir Dzyuba and Matthijs van Leeuwen. Interactive discovery of interesting subgroup sets. In *IDA*, volume 8207 of *Lecture Notes in Computer Science*, pages 150–161. Springer, 2013.
- [13] Léon Bottou and Chih-Jen Lin. Support vector machine solvers. *Large scale kernel machines*, 3(1):301–320, 2007.
- [14] Mansurul A Bhuiyan and Mohammad Al Hasan. Priime: A generic framework for interactive personalized interesting pattern discovery. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 606–615. IEEE, 2016.
- [15] Luc De Raedt. A perspective on inductive databases. *SIGKDD Explor.*, 4(2):69–77, 2002.
- [16] Tomasz Imielinski and Heikki Mannila. A database perspective on knowledge discovery. *Commun. ACM*, 39(11):58–64, 1996.
- [17] Liqiang Geng and Howard J. Hamilton. Interestingness measures for data mining: A survey. *ACM Comput. Surv.*, 38(3):9, 2006.
- [18] Thomas L. Saaty and Luis G. Vargas. Comparison of eigenvalue, logarithmic least squares and least squares methods in estimating ratios. *Mathematical Modelling*, 5(5):309–324, 1984.
- [19] S.I. Gass and T. Rapcsák. Singular value decomposition in ahp. *European Journal of Operational Research*, 154(3):573–584, 2004.
- [20] E. Takeda, K.O. Cogger, and P.L. Yu. Estimating criterion weights using eigenvectors: A comparative study. *European Journal of Operational Research*, 29(3):360–369, 1987.
- [21] Eric Blankmeyer. Approaches to consistency adjustment. *Journal of Optimization Theory and Applications*, 54:479–488, 1987.
- [22] Matteo Brunelli. *Introduction to the analytic hierarchy process*. Springer, 2014.
- [23] Thomas L Saaty. A scaling method for priorities in hierarchical structures. *Journal of Mathematical Psychology*, 15(3):234–281, 1977.
- [24] Vladimir Dzyuba, Matthijs van Leeuwen, Siegfried Nijssen, and Luc De Raedt. Interactive learning of pattern rankings. *Int. J. Artif. Intell. Tools*, 23(6), 2014.
- [25] M. G. Kendall and B. Babington Smith. The problem of  $m$  rankings. *Ann. Math. Statist.*, 10(3):275–287, 09 1939.

- [26] Nir Ailon. An active learning algorithm for ranking from pairwise preferences with an almost optimal query complexity. *J. Mach. Learn. Res.*, 13:137–164, 2012.
- [27] ERHAN ERKUT and MURAT TARIMCILAR. On Sensitivity Analysis in the Analytic Hierarchy Process. *IMA Journal of Management Mathematics*, 3(1):61–83, 01 1991.
- [28] Victor Y. Pan and Zhao Q. Chen. The complexity of the matrix eigenproblem. In Jeffrey Scott Vitter, Lawrence L. Larmore, and Frank Thomson Leighton, editors, *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 507–516. ACM, 1999.
- [29] Thomas L Saaty and Mujgan S Ozdemir. Why the magic number seven plus or minus two. *Mathematical and computer modelling*, 38(3-4):233–244, 2003.
- [30] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, May 26-28, 1993*, pages 207–216. ACM Press, 1993.
- [31] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. Selecting the right objective measure for association analysis. *Information Systems*, 29(4):293–313, 2004.
- [32] Jeffrey L. Ringuest. A chi-square statistic for validating simulation-generated responses. *Comput. Oper. Res.*, 13(4):379–385, 1986.
- [33] Carine Lallemand and Guillaume Gronier. Enhancing user experience during waiting time in hci: Contributions of cognitive psychology. In *Proceedings of the Designing Interactive Systems Conference, DIS '12*, pages 751–760, New York, NY, USA, 2012. ACM.