UPRE: User preference based search system

Peter Gurský[1], Tomáš Horváth[1], Róbert Novotný[1], Veronika Vaneková[1], Peter Vojtáš[2]
[1]University of P.J. Šafárik, Košice, Slovakia
{peter.gursky, tomas.horvath, robert.novotny, veronika.vanekova}@upjs.sk
[2]Charles University and ICS AS, Prague, Czech republic
peter.vojtas@mff.cuni.cz

Abstract

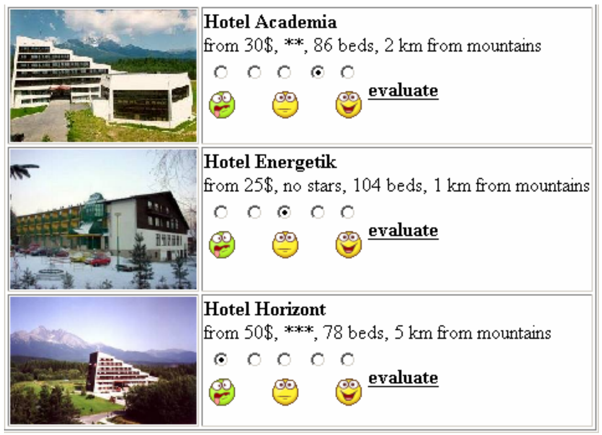
We present a middleware system UPRE system enabling personalized web search for users with different preferences. The input for UPRE is user evaluation of some objects in scale from the worst to the best. Our model is inspired by existing models of distributed middleware search. We use both inductive and deductive tasks to find user preferences and consequently best objects.

1. Introduction and motivation

Today we can find many techniques in searching user relevant objects. Imagine a portal of hotel searching system. Typically such portals are based on a faceted browser, a form search system or a full text search system. We present a quite uncommon approach proposed by our user preference search system UPRE. This system can be used individually or added as a next functionality to the mentioned search approaches.

When a user wants to specify to UPRE the objects he prefers, he needs to evaluate several objects in some scale (from the worst to the best). These objects can be a representative set of samples or objects found during a visit of a portal. After evaluation of some objects our system can learn user preferences based on (possibly hidden) attributes or properties of objects and give to the user the best objects from the whole set of objects. An example of such user evaluation system can be seen on figure 1.

Typical objects we want to search are hotels, job offers, papers, pictures, books, presentations, conferences etc. The decision of which object is better is usually based on object properties. Let us consider the following example: Imagine a user looking for a hotel which has good price, good distance from airport and has good equipment in rooms.


Hotel Academia
from 30$, **, 86 beds, 2 km from mountains
○ ○ ○ ● ○
evaluate

Hotel Energetik
from 25$, no stars, 104 beds, 1 km from mountains
○ ○ ● ○ ○
evaluate

Hotel Horizont
from 50$, ***, 78 beds, 5 km from mountains
● ○ ○ ○ ○
evaluate

Figure 1. Evaluation of objects

The meaning of goodness can be for each user (or group of users) different. For the price of hotels, one user can prefer cheap hotels (student), second prefers expensive hotels (manager) and the other one prefers middle price (professor). The natural meaning of this ordering is that the user determines the relations "'better" or "worst" between any two values of a given property. For each such property, user has a notion about the ordering of objects based on real value of property from an attribute domain. We call these orderings of particular attribute domains the user local preferences.

User usually has to decide preference between objects which are incomparable in particular attribute preference (one hotel can be better in price, another in distance etc.). This human feature is modeled e.g. in multicriterial decision making. With the combination of user local preferences we obtain the user global preferences. Usual way to obtain global preferences is to use a combination function which states the weights of the local preferences. We call this function an aggregation function @. For example the aggregation function @(7*cheap+2*near)/9 states that the user prefers more cheap hotels than near ones (it does not mean that he prefers far hotels).

We assume, and it is not unusual in the semantic web, to have distributed properties of the objects. For example if we want to find a good hotel, the information about price and room equipments should be collected on one server and the distance from airport can be stored on another. Another server should collect information e.g. about year of construction or ranking of hotels by users visiting hotels in the past etc.

Well known solution is faceted browser (see [14]). User can reduce the total number of displayed instances by enabling one or more restrictions thus decreasing the size of the visible information space. Individual restrictions are further combined to form complex restrictions ultimately allowing the user to perform more precise queries. These restrictions specify the user requirements very strictly. This can be disadvantage in many real-world situations. For example if we specify a restriction "price ≤ 100$" the system does not find any hotel of the cost 101$, however this hotel can be better in other attributes than hotels satisfying the given restriction. Furthermore, we can not form restrictions in the natural way like "cheap", "near to", "good", etc.

The second approach in a distributed environment is a monotone aggregation function firstly proposed by R.Fagin [5]. As mentioned above, by monotone aggregation function user can say the importance of each property e.g. by weighted sum. The handicap of this approach is the requirement of explicit weights of properties. Many users cannot imagine what it means that "price is 3 times more important than distance from the airport". So, the results may not be relevant to the user's intended requirements.

The third way to specifying the objects to retrieve is a skyline. In the skyline, there are objects that are not dominated by any other object. Object x dominates object y if x has greater or equal score in all properties and is strictly greater in at least one property than y. Skyline was firstly presented by S. Börzsönyi et al. [2]. Authors put a proposed algorithm to the database query processor. First solution in the field of distributed middleware querying was presented by W. Balke et al. [1]. The lack of this approach is that it gives many results if we consider more properties and they are not ordered by the grade of relevancy to user requirements.

The main idea of our approach is the following: for many users it is hard to specify their local or global preferences exactly (by functions, equations, etc). Instead, they rather express their local preferences in natural language (cheap hotels, fast cars, etc), similarly global preferences (they prefer nearest and expensive hotels before the far and cheap ones, etc.). If we give to users a form and let them to explain local preferences for all of our properties and then do so for the global preference, it could be easily boring for them. Thus we have to use different way to obtain local and global preferences. We follow the idea that everybody can easily say, for a concrete object, how suitable it is. Thus we will require from user to evaluate the objects in scale from the worst to the best.

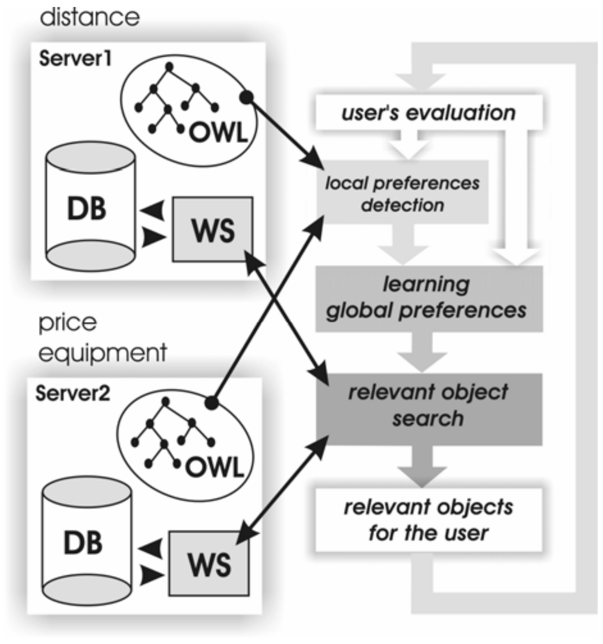


Figure 2. Flow diagram of system UPRE

This paper is structured in compliance with the figure 2:

In our system UPRE if user wants to find suitable objects we give him a set of objects, representative samples, and their properties (e.g. some hotels and their prices, distances from airport, pictures, etc.). User evaluates/ranks these objects according to his/her impression, belief. Using the ranking, our system detects the user's local preferences (in chapter 3), so we determine the type of a user specified by ontology (in chapter 2). From these local preferences and from the user's evaluation we learn the user's global preferences (in chapter 4). Consecutively, by means of these global preferences we find relevant objects for the user (in chapter 5). User can evaluate them and start the whole process again. In chapter 6 we outline the implementation aspects of UPRE and chapter 7 concludes our paper.

2. Model of User Preference Ontology

Data that we use in our examples are stored in domain ontology of hotels. Hotels are resources with properties like price, room equipment, distance from the airport etc. We want to order the values of these properties according to local preference of some user type, but the ordering method depends on the type of property.

We recognize four types of properties. The first type includes properties that are graded in some way to finite number of classes. For example, hotels can have different types of internet connection: none, dial-up, EDGE, satellite, wireless or optical. This is the set of possible values. We sort them by the speed of connection and assign them the values from 0 to 1. Bigger evaluation means better speed. The second type is real or integer number, for example price, number of beds etc. We order the values in usual way and normalize the results to the interval [0,1]. The third type is boolean or yes/no property e.g. aspect at a sea or breakfast included. These properties have evaluation 0 or 1. The last type of properties is text. We can use this kind of attribute to reduce searching space (user wants to have internet in a hotel). We can also derive some other properties from a text. These new properties should be one of the three previous types. For example from the name of a town or ZIP code we can deduce a distance from a specific location.

We suppose that properties are potentially stored on different places or in different data stores in their domain ontologies. We proposed UPRE ontology to explain data sources. When we learn user local and global preferences (see next 2 chapters) from the user evaluated data (or if we get them some other way), we can map them to relevant part of "user preference ontology" UPRE (Figure 3). The schema of this ontology is independent from the domain ontology and it can be used for data other than hotels as well. The local preference specifies the ordering of values for a given attribute while the global preference determines the importance of every attribute.

We consider three general types of ordering of properties: ascending (bigger values are better), descending (smaller values are better), and middle (values in the middle of the range are better - neither big, nor small). These three types of ordering cover most of examples of local preferences from the real life. The case of middle values is more complicated than the other two: we know the range of possible values of the attributes, but the preferred middle value does not have to be exactly in the middle of the range. Though this is the most usual case, we assume that the preferred middle can also be in 1/4 and 3/4 of the range.

We consider also a special case, where the overall attribute value is counted from more partial values of properties. For example the price of accommodation is not only number, but we have to multiply it with the course of the particular currency. The currency itself has a finite set of possible values. So in our notion of local preference, every attribute can contain several partial results and a formula to combine them together.

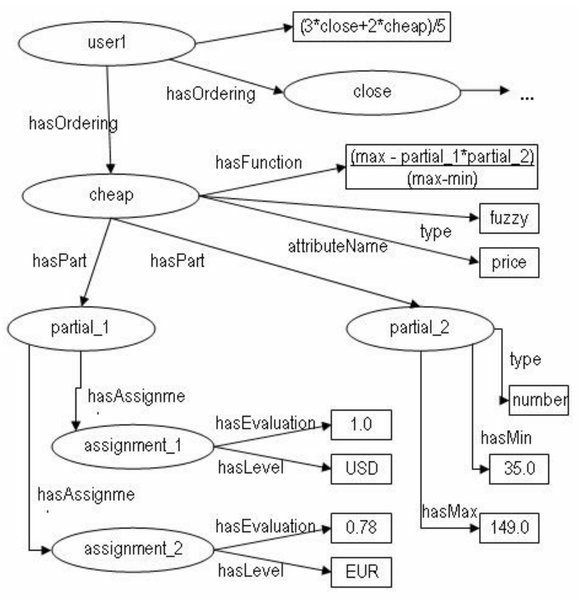


Figure 3. Instance of UPRE ontology

Instances of the user preference ontology are created offline from the instances of domain ontology. For every interesting property and every type of ordering we create an instance of *upre:attribute*[1] class and we give it a suitable name. For example if the property is a price of hotel and the ordering is descending, we name this instance "cheap" because the ordering returns cheap hotels first. For the same property and ascending ordering, we name the instance "expensive". We repeat the process for every interesting property from the domain ontology (mentioned 4 types). Then we define the user types as instances of *upre:user_type*. One type of user may prefer cheap hotels and hotels close to the center, other type may prefer expensive and close hotels. The resulting instance of *upre:user_type* is shown on figure

[1] prefix upre: stands for http://ics.upjs.sk/upre/upre.owl where the user preference ontology can be found

3. We leave out the prefix *upre:* for every URI reference in the graph to simplify it and reduce its size.

3. Detecting local preferences

To learn local preferences, we give to user a (representative) sample of hotels (see table 1) with several attributes. The attributes in our example are: distance from the city center, price of the accommodation and equipment of rooms (without equipments, TV, internet or both). The user classifies every hotel into one of the three classes (categories): poor, good and excellent according to the relevance of the hotel to him. In real world we use 7 classes of Likert scale.

Table 1. (Representative) sample of hotels evaluated by user

hotel	distance	price	equipment	evaluation
Apple	100 m	99 $	nothing	poor
Danube	1300 m	120 $	tv	good
Cherry	500 m	99 $	internet	good
Iris	1100 m	35 $	internet,tv	excellent
Lemon	500 m	149 $	nothing	poor
Linden	1200 m	60 $	internet,tv	excellent
Oak	500 m	149 $	internet,tv	good
Pear	500 m	99 $	tv	good
Poplar	100 m	99 $	internet,tv	good
Rhine	500 m	99 $	nothing	poor
Rose	500 m	99 $	internet,tv	excellent
Spruce	300 m	40 $	internet	good
Themse	100 m	149 $	internet,tv	poor
Tulip	800 m	45 $	internet,tv	excellent

We detect the local preferences by applying simple multivariate polynomial regression. By this way we obtain the following:

```
eval = +0.000837224*distance -
0.0100111*price +2.4805
```
(1)

The result (1) means that the price has negative influence and the distance has positive influence on the user evaluation/ranking. In other words, the user classification decreases with increasing price of a hotel and it increases with increasing the distance of a hotel from an airport.

The local preferences can be detected by other statistical methods. The system QUIN [3] for a qualitative data mining which discovers trends in data gives promising results (2).

```
  M-,+(price,distance)
(Cov=97%=62(?31%)/64)
class=eval | distance price
{97%=62(?31.25%)/64}
```
(2)

This result is similar to (1). The attribute distance has positive influence and the attribute price has negative influence to the user evaluation on the whole domain of attributes.

Moreover with the QUIN it is able to detect preferences, in which the middle or the marginal values of the domains of attributes are the best.

Note that "equipment" is a text attribute so it needs no ordering detection. We focus on distance and price.

4. Learning global preferences

We learn user's global preferences by the method of Ordinal Classification with Monotonicity Constraints described in [9,10]. This method is based on Inductive Logic Programming ILP system ALEPH [11], the well-known relational data mining model [4].

In the learning process we compute the user's global preferences by using his/her local preferences (learned by QUIN). We use ILP because our local preferences (orderings) can be represented well in this framework (by definite clauses). For illustration, consider that we have discretized values of distance to three classes: near, middle and far. The different orderings of these classes for different users can be:

`near ≤ middle ≤ far` (non-decreasing)
`near ≤ far ≤ middle`
`far ≤ middle ≤ near` (non-increasing)
`far ≤ near ≤ middle`
`middle ≤ near ≤ far` (the middle values are "better" than near or far)
`middle ≤ far ≤ near`

A usual aggregation function can be easily simulated by monotone classification rules in the sense of many valued logic. The results of our approach (the user's global preferences) computed from the data in our illustrative example are the following classification rules about the relevance of hotels to user preferences (see figure 4):

- evaluation = excellent IF distance>=500 AND price<=99 AND services={tv,internet}
- evaluation = good IF (distance>=500 AND services={tv})
- evaluation = good IF (price<=99 AND services={internet})

From our results we can obtain also additional information about crisp attributes (equipment) The meaning of any classification rule is as follows: If the attributes of object x fulfill expressions on the right side of the rule (body) then the overall value of x is at least the same as on the left side of the rule (head). We can, of course, assign the explicit values to vague concepts like excellent or good. During the simulation of computation of aggregation function we can simply test the validity of requirements of the rules from the strongest rule to weaker ones. When we find the rule that holds, we can say that the overall value of the object is the value on the left side of the rule. Since we test the sorted rules, we always rank the object with the highest possible value.

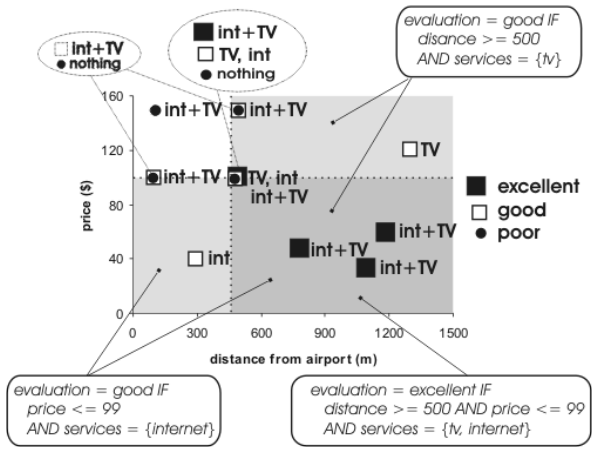


Figure 4. The results of our approach in the case of our illustrative example.

We can see that the ordered meaning of the classification is preserved in our results: hotels classified in the grade "excellent" by the user also fulfills requirements for "good" and "poor" hotels, and "good" hotels fulfills requirements for "poor" ones (e.g. the hotel with 800 m distance from airport and 45$ price equipped with internet and TV is "at least as appropriate as" the hotel 300 m far from the airport and 40$ price equipped just with internet) according to the local preferences (more far and more cheap is the better).

5. Relevant object search (top-k aggregation)

We stated that our model allows distributed properties of the objects. We want use the capability of web services to provide needed properties. The main idea of related approaches is to browse only necessary data from web services until the system is sure that it has top-*k* objects already. Retrieving only *k* best objects can save time and is usually sufficient for user. Saving of accesses grows with the number of objects stored on sources and also with the number of properties we consider.

First approach to such a middleware system was proposed by R. Fagin [5]. He considered two kinds of accesses to sources (web services): sorted and random access. Sorted access means the query for the next best object from the source, so the service send data ordered from the best to the worst. Random access asks for the value of attribute of a concrete object. When we want to use web services we need to minimize the number of accesses to web services and of course the time of searching. The random access has one big disadvantage. Each random access requires searching of the value of concrete object. In the case of sorted access the results are prepared immediately (values can be preordered to the several orderings before user comes) and, moreover, data can be sent in blocks. Because of this feature in our system UPRE we prefer mainly the algorithms which use only sorted access but the random + sorted access can be used too.

From the random access only approaches U. Güntzer et al. [8] proposed stream-combine algorithm as the first one. Much better and supported by good formal model was No Random Access algorithm presented by Fagin et al. [6]. As an improving P. Gurský [7] presented 3P-NRA (3 phased No Random Access) algorithm which reduces unnecessary operations from No Random Access algorithm and proposes some heuristics.

6. Implementation

Our implementation is based on the multiple tools, which are depicted on the Figure 5.

Each of the tools is able to function as stand-alone components. However, this is not our goal. The integration of these components is a fundamental part of our middleware. This is achieved via the Spring Framework [16], which serves for two purposes. Mainly, it glues together and manages the associations between classes, thus maintaining low coupling and high cohesions.

The second purpose is the use of some utility classes that the Spring Framework provides. E. g. Spring's JDBC abstraction unifies the access to the relational databases by various tools.

6.1. The Top-*k* Search

The Top-k search component serves as a tool for searching relevant objects. It is able to retrieve the best k object according to user requirements. The architecture of our top-k search component proposes the using of several algorithms, heuristics and types of sources to map the best searching strategy for a given data sources and types of data. We have implemented

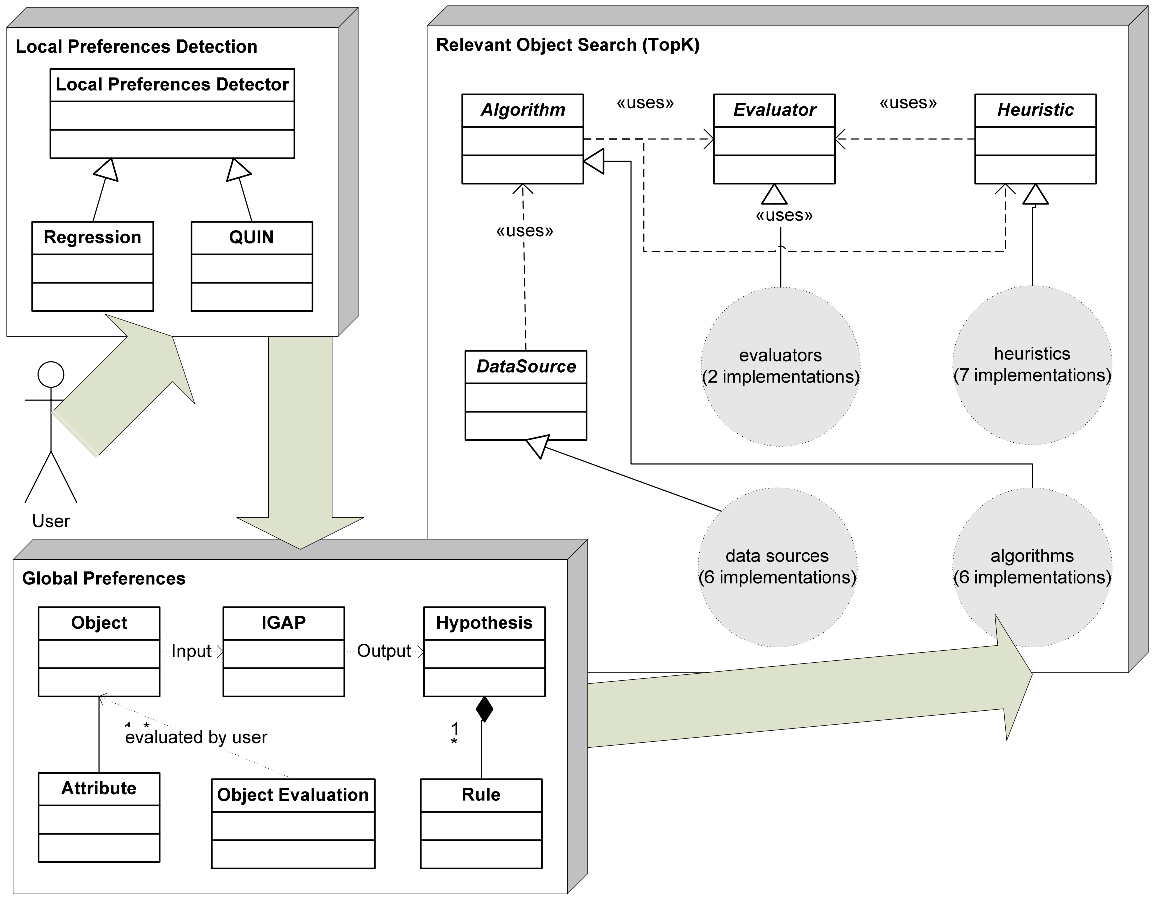

Local Preferences Detection
Local Preferences Detector
Regression
QUIN
User
Relevant Object Search (TopK)
Algorithm
«uses»
Evaluator
«uses»
Heuristic
«uses»
«uses»
DataSource
evaluators (2 implementations)
heuristics (7 implementations)
data sources (6 implementations)
algorithms (6 implementations)
Global Preferences
Object
Input
IGAP
Output
Hypothesis
evaluated by user
1
*
Attribute
Object Evaluation
Rule

Figure 5. The overall schema of the system.

all mentioned approaches plus several others. The architecture counts also with aggregation function as well as classification rules. Sorted access is implemented also using SOAP web services and data are sent in blocks of several objects to minimize the number of messages.

The overall design comprises of four classes: an *Algorithm* represents the procedure used to evaluate the object order. We provide multiple implementations, which use random or sorted accesses to the lists (or their combinations). *DataSource* allows to abstract from the physical location of the input data. We implemented descendants of *DataSource* for many possible sources like SOAP, database, b+ tree, files, ontology and memory.

Heuristic is used by *Algorithm* to optimize the time and space requirements. Their implementations are based on several theoretical researches [1, 5, 7, 8] – so far we have programmed 7 various heuristics.

The final important class, *Evaluator*, is used in the process of calculation of the overall value of an object, which is used in the determination of the final order of the objects. One subclass is based on the concept of the aggregation functions and the other uses the rules of the monotone graded classification obtained from the user evaluation .

All of the four components are based on the Strategy design pattern. This allows great flexibility in the methods used in the evaluation of the best objects by Ordinal Classification with Monotonicity Constraints.

7. Conclusions

In our model we improved R.Fagin's model [5] of middleware search (based on aggregation function) [9] by adding the inductive (learning) part [9, 10] and preprocessing part. The inductive part was used to learn local and global preferences. In preprocessing we combined domain ontology on possibly different sources and user preference ontology.

The main result of the model is the integration of all mentioned components in the process of finding best objects to users according to their own preferences.

Our approach is used also in the Slovak project *NAZOU – Tools for acquisition, organization and maintenance of knowledge in an environment of heterogeneous information resources* to find relevant job offers for the user according to him/her preferences. The domain of this project is different from the one used in this paper, thus we have another domain for testing purposes. We can constitute that our model is domain-independent.

However, our system works sufficiently, we have in plans to make some further improvements, e.g. implementation of heuristics in the learning process of preferences and implementation of M-trees for handling geographically-dependent information (distance of two cities of the world, important for stating the attributes near from, far from, etc.). So far, we have only promising experiments, now.

Furthermore we want to implement the automatic detection of user types according to their common local preferences and thus to compute just the global preferences (e.g. young people like cheap hotels not to be near to a centre while the older ones prefers hotels near to centre, however these are a bit more expensive, etc.). The presentation module of our system needs further improvements, too.

Finally, since our system is domain-independent, it is easy to install to any search applications what makes our approach useful in the commercial sphere, too. It can be used individually or added as a new functionality.

References

[1] Balke, W., Güntzer,U., Zheng, J.: Efficient Distributed Skylining for Web Information Systems, EDBT 2004, LNCS 2992, Heraklion, Crete, Greece, Springer (2004)

[2] Börzsönyi, S., Kossmann, D., Stocker, K.: The Skyline Operator, ICDE 2001, Heidelberg, Germany (2001) 421-430

[3] Bratko, I., Šuc, D.: Learning qualitative models. AI Magazine 24 (2003) 107-119

[4] Džeroski, S., Lavrač, N.: An introduction to inductive logic programming. Relational data mining, S. Džeroski, N. Lavrač eds., Springer 2001, 48-73.

[5] Fagin, R.: Combining fuzzy information from multiple systems, J. Comput. System Sci. (1999) 58:83-99

[6] Fagin, R., Lotem, A., Naor, M.: Optimal Aggregation Algorithms for Middleware. In Proc. 20th ACM Symposium on Principles of Database Systems (2001) 102-113

[7] Gurský, P.: Towards better semantics in the multifeature querying. Proceedings of Dateso 2006, ISBN 80-248-1025-5, (2006), 63-73

[8] Güntzer, U., Balke, W., Kiessling, W.: Towards efficient multi-feature queries in heterogeneous enviroments. ITCC, Las Vegas, Nevada, IEEE Computer society (2001) 622 – 628

[9] T. Horváth, P. Vojtáš: Ordinal Classification with Monotonicity Constraints. In. Proceedings of 6th Industrial Conference on Data Mining ICDM 2006, Leipzig, Germany: LNAI 4065, Springer, 2006, pp. 217 – 225.

[10] Horváth, T., Krajči, S., Lencses, R., Vojtáš, P.: An ILP model for a graded classification problem. J. KYBERNETIKA 40 (2004), No. 3, (2004), p:317–332.

[11] Srinavasan, A. The Aleph Manual. Technical Report, Comp.Lab., Oxford University

[12] Vojtáš, P.: Fuzzy logic aggregation for Semantic Web search for the best (top-k) answers, in FLSW - Fuzzy logic and the semantic web, E. Sanchez ed. Capturing Intelligence Series, 1, Elsevier (2006) 341-360

[13] Vojtáš, P.: A fuzzy EL description logic with crisp roles and fuzzy aggregation for web consulting, accepted for IPMU Paris (2006)

[14] Wynar, B. S., Taylor, A. G.: Introduction to cataloging and classification, 8th edition. Libraries Unlimited Inc. (1992)

[15] system MPR: <http://fuzzy.cs.uni-magdeburg.de/~borgelt/software.html#regress>

[16] The Spring Framework. <http://www.springframework.org/>