Fuzzy User Preference Model for Top-k Search

Peter Gurský, Veronika Vaneková and Jana Pribolová

Abstract—The task of modeling complex user preferences face the problem of understandability to common users and the problem of querying the dataset for preferred objects. We propose natural and complex model of user preferences decomposed with respect to particular attribute values. Partial preferences are combined by monotone aggregation function. The user model is stored in ontology structure. We also present an extension of top-k objects search algorithm to provide a query evaluation of the proposed model.

I. INTRODUCTION

Exponential growth of web content has recently led to expansion of search engines, recommender systems and personalized e-commerce portals. All these various systems have a similar goal: to restrict the amount of information presented to a user and to provide only the information relevant to user’s requirements. However, the representation and formal model of user preferences might vary from application to application. Search engines use keyword queries, document similarity notion and tf-idf vector model, i.e. a user preference is represented only by few key words. Recommender systems usually utilize explicit user ratings of items or implicit ratings derived from user’s behavior. The most complex user preference models include evidence of preferred attribute values and the importance of particular attributes to a user. We provide a brief survey of various preference models in section II.

Complex user preference models usually reflect real life preferences more accurately and lead to more precise results. Many approaches that model user preferences omit the problem of searching the relevant data in a dataset. The models of complex user preferences that represent data in very bushy structures can lead to complexity explosion, especially over huge datasets.

In practice, users often have only vague idea about what they want to find. We want to cover both precise and vague preferences. We propose a fuzzy model of user preferences, which reflects this fact and enables vague queries.

User preferences are stored in OWL ontology described in section III. We analyze various explicit ways of determining fuzzy preferences in section IV. In section V, we describe top-k search algorithm which provides very fast way of finding best objects for a specific user without need to process all objects.

In summary, the contributions of this paper are:

- We propose natural and complex model of user preferences decomposed to partially linear local preferences and monotone combination function.
- We present the index structures and methods that can be used for effective search of top-k relevant objects based on the proposed model.

II. PREFERENCE MODELS

Preference is a concept which enables a choice between several objects and provides rank ordering of these objects, based on user’s satisfaction they provide. Therefore the simplest representation of user preference is object ordering or ranking. This representation is widely used in recommender systems [2] and provides satisfactory results.

The main problem addressed by recommender systems is to predict preference values for objects unseen by user. There are two main approaches to solve this problem: content-based filtering and collaborative filtering.

Content-based filtering systems [4] allow users to choose keywords that describe their preferences. Keywords are searched within available documents or object descriptions. Users also have possibility to rate objects to provide feedback for the system.

In collaborative filtering systems [3], the user has a possibility to rate some subset of objects (the whole set may contain millions of objects). These ratings are stored in user-object matrix M, where every element $m_{ij}$ represents a rating given to $j^{th}$ object by $i^{th}$ user. Each user’s preference is thus represented as a sparse vector of ratings, i.e. the most values in this vector are undefined. In user-based approach, the recommender system finds a group of users similar to actual user (usually via cosine vector metric or correlations) and predicts a rating of unseen object as an average rating within the group. In item-based approach, the system associates each object (item) with a group of similar objects and then predicts the user’s rating as an average rating of other objects from the group.

Problems arise when new users or new objects enter the system [1]. New users have not rated any items, therefore it is impossible to find similar users and nothing can be found about their preferences. Similarly, new objects are not yet rated by any user, so they are not recommended. Another problem is scalability. User-object matrix becomes very
large when the number of objects and users increases.

The main disadvantage of collaborative filtering systems is that they model only object ratings but they ignore various attributes of these objects. In practice, users always consider some attributes when giving ratings to objects. User preferences to attributes remain hidden to the system.

*Multi-criteria decision* systems enable user to choose preferred values for every attribute. Again, these attribute values can be represented as classical sets or intervals. The system will find only objects with specified attribute values. But when we impose such crisp search conditions, the system will not retrieve objects with similar values. The retrieved set of objects can be too large or too small and there is no specific ordering of objects. These problems are addressed by fuzzy logic and uncertainty theory, see e.g. [10] for fuzzy extension of logic programming.

We proposed preliminary fuzzy search system in [11]. Now we extend the presented user model, describe some preference acquisition methods and specify searching algorithms.

Instead of classical (crisp) sets of values, we use *fuzzy sets*. Every element of fuzzy set has a degree of membership, determined by fuzzy set membership function. Degrees of membership range from 0 to 1, where 0 means that the element does not belong to the fuzzy set, 1 means full membership and other values mean partial membership. Thus we can order retrieved objects according to relevancy of attribute values to particular user.

Real-life preferences usually belong to one of the following variants of trapezoidal membership function (see fig. 1):

- **left-trapezoidal function (LT)** – lower attribute values are better,
- **right-trapezoidal function (RT)** – higher attribute values are better,
- **trapezoidal function (TRZ)** – middle values are preferred,
- **inverse-trapezoidal function (INV)** – marginal values are better than middle values.

![Fig. 1. Basic fuzzy set types: left-trapezoidal, right-trapezoidal, trapezoidal and inverse-trapezoidal membership functions.](image)

In the following sections we describe our fuzzy preference model in more detail.

### A. Ordinal and Nominal Attributes

First of all we describe possible attribute types. The most advantageous are *ordinal* attributes. As the name indicates, an ordinal attribute has some natural value ordering, other than lexical ordering. Typical examples are integer or decimal numbers. Creating a fuzzy set over ordinal attribute is straightforward.

*Nominal* attributes have a finite range of possible values, usually strings. These values have no apparent ordering. We usually express user preference as a *crisp set* of suitable values.

However, user can still prefer some values more than other values. It is possible to specify user’s degree of preference to some values. This method is called *fuzzification* of crisp attribute.

### B. Local Preferences to Domain Attributes

Let us consider a real-estate system holding information about flats for sale with following attributes: price, number of rooms, floor, size, age (ordinal attributes) and status, seller type (nominal attributes). Imagine a user looking for a flat with “good price” and “good size”. Such requirements can be subjective and thus different for every user. A concept of “good price” depends on user’s income, whereas “good size” can depend on size of his family. Most people prefer big flats, but some look for middle size or small flats.

For each interesting attribute, the user can determine which values are better. Every attribute value is given a number from [0,1]. In our preference model, these values determine a degree of membership in fuzzy set like “good_price”. Values with greater degree of membership are more preferred by user. Thus he defines an ordering of objects (flats) based on values from attribute domain. We call these orderings of particular attribute domains the user’s *local preferences*.

### C. Global Preference

Local preferences to different attributes can yield different orderings of objects. In fact, this happens very often with real world data. If the user prefers cheap and big flats, we find that flats with better size have worse price and vice versa. Local preferences are not enough to compare two arbitrary objects from the domain of discourse. We need to combine membership values obtained by user local preferences for every object.

The combination of user local preferences that leads to overall values of objects for a concrete user is called *global preference*. There are many possibilities of aggregating fuzzy local preferences to one overall value. We only require such combination function to be *monotone* in all arguments. It can be represented as a composition of fuzzy connectives (conjunctions or disjunctions). Another possibility is a weighted average, e.g.:

$$
good\_flat(x) = \frac{2 \cdot good\_price(x) + good\_size(x)}{3}
$$
where \( good\_price(x), good\_size(x) \) are local preferences. The local preference \( good\_price(x) \) has higher weight than \( good\_size(x) \), so its influence on the overall result will be bigger.

In our preference model we can specify global preferences also by monotone classification rules which can be learned from objects ranked by the user [9]. These rules reflect global preferences in more natural and comprehensible way. The following example shows a set of classification rules:

- \( good\_flat(x) \geq 0.8 \) IF \( good\_price(x) \geq 0.8 \) AND \( good\_number\_of\_rooms(x) \geq 0.3 \)
- \( good\_flat(x) \geq 0.4 \) IF \( good\_price(x) \geq 0.5 \) AND \( good\_age(x) \geq 0.5 \) AND \( good\_floor(x) \geq 0.1 \)

Every rule consists of a head (e.g. \( good\_flat(x) \geq 0.8 \)) and a body, which is a conjunction of clauses (e.g. \( good\_price(x) \geq 0.8 \) AND \( good\_number\_of\_rooms(x) \geq 0.3 \)). If these conditions are fulfilled for some object, then this object has overall preference value at least the same as on the left side of the rule (head).

Note that more than one rule can apply for some object. Let us consider an object \( flat1 \) with local preference values \( good\_price(flat1) = 0.91, good\_number\_of\_rooms(flat1) = 0.4, good\_age(flat1) = 0.7 \) and \( good\_floor(flat1) = 0.5 \). It is easy to see that this object satisfies the conditions of both rules stated above. We infer \( good\_flat(flat1) \geq 0.8 \) from the first rule and \( good\_flat(flat1) \geq 0.4 \) from the second rule. The higher value 0.8 is taken as a final result. The inequality \( good\_flat(flat1) \geq 0.8 \) implies also \( good\_flat(flat1) \geq 0.4 \), so both rule heads are fulfilled.

III. USER PREFERENCE ONTOLOGY

The user preference ontology is designed to store all available information about users and their preferences. Ontology structure is especially suitable for storing user preferences because of their rich graph structure and possible irregularities and missing values. The ontology described in this section provides unified schema for our system. It is not used for data interchange and processing on the web, however, such usage is also possible.

The main class \textit{User} has the following properties that express relevant personal information:
- \( \text{hasAge} \)
- \( \text{hasMaritalStatus} \)
- \( \text{hasGender} \)
- \( \text{hasEducationLevel} \)

Every user is then represented as an instance of class \textit{User} with a unique URI identifier. Personal information is gathered as a part of user’s registration. It can be used later to determine initial set of local user’s preferences (see section IV).

User’s personal information is independent from particular domain, but preferences are domain dependent. It means that the user may have different preferences to price in different domains, e.g. flats and notebooks. \textit{User} instance is therefore connected with one \textit{DomainSpecificUser} instance for every domain that we want to model, e.g. one instance for flats and another instance for notebooks. Instances of \textit{DomainSpecificUser} can be acquired from different sources and with different methods. Together they form a complex model of domain specific preferences which are used for finding best objects (see section V).

Class \textit{DomainSpecificUser} has the following properties:
- \( \text{hasAttributePreference} \) – local preference related to a domain specific attribute
- \( \text{hasRuleCharacteristic} \) – global preference

Local preferences are related to fuzzy, fuzzyfied or crisp attributes. In the first case, \textit{AttributePreference} instance is connected with fuzzy set, as is shown on fig. 2. Fuzzy set membership function is piecewise linear and it is specified by a set of points with \((x,y)\) coordinates and string labels. We consider only fuzzy sets that correspond to four basic types mentioned in section II. Thus every fuzzy set in our ontology must have a type specified. On Fig. 2, LT is an abbreviation for left-trapezoidal function, RT for right-trapezoidal function, TRZ denotes trapezoidal function and INV its inverse type.

[Fig.2. Ontology representation of preference to fuzzy attributes.]

In the case of fuzzyfied attributes, we use a set of fuzzyfied values instead (fig. 3). Each fuzzyfied value consists of an original string value and an evaluation from \([0,1]\). From set-theoretic point of view, such fuzzyfication is a fuzzy subset of original (crisp) value set.

And finally, in case of crisp attributes, we can express preference as a crisp set of preferred attribute values (see fig. 4).
As stated in section II, global preferences can be represented as monotone classification rules. Ontology schema for classification rules is shown on fig. 5. Each rule consists of one or more condition clauses and a result value.

Let us consider a rule from our previous example

\[
good_{flat}(x) \geq 0.4 \quad \text{IF} \quad good_{price}(x) \geq 0.5 \quad \text{AND} \quad good_{age}(x) \geq 0.5 \quad \text{AND} \quad good_{floor}(x) \geq 0.1
\]

has a result value 0.4 and it consists of three clauses, namely “good_price(x) ≥ 0.5”, “good_age(x) ≥ 0.5” and “good_floor(x) ≥ 0.1”. Each condition clause will be represented by one ontology instance of Clause. If we put the first clause “good_price(x) ≥ 0.5” to ontology, it will be connected with a domain specific attribute Price (because the fuzzy set “good_price” depends on Price). The inequality “≥” is represented by a corresponding instance of the class Relation and “0.5” is a datatype value.

IV. ACQUISITION METHODS

In general, user preferences can be acquired explicitly or implicitly. Explicit methods require direct user input. It can be a ranked set of objects, selection of preferred values, text query, etc. Direct user input is an ideal case; it assumes that the user has a clear idea about what he wants to find and that he is able to pass his requirements precisely to the system.

In practice, this may not be the usual case. The user sometimes cannot specify his preferences before he orientates himself in the system and explores its functionality. Explicit specification of one’s preferences could be too complicated task for a new user, so that it would discourage him from using the system.

Implicit methods extract preferences from user’s behavior. For example, if a customer of online store purchases some product, the system automatically fills corresponding element in the customer’s preference vector. Other significant user’s actions include downloading or printing a page, downloading a document or using some GUI elements. This approach typically obtains more data than explicit methods, because it gathers preferences even from those users who would never enter their explicit ratings. However, it takes a long time before the system gathers enough information about a new user [1].

Many systems use hybrid methods that combine both explicit and implicit approaches. It is possible to use an inductive procedure to find local and global preferences from explicit user’s ratings of objects [9]. An alternative approach is SVM based system described in [8].

We employ an explicit acquisition method which lets user skip explicit specification of his preferences if it is too complicated or boring task for him. Missing parts of user profile are filled by modified collaborative filtering method. User profile can be also modified later.

The first part of user registration is compulsory. It consists of personal information questionnaire. This information is stored in the domain independent part of our user ontology (see section III).

The user has an opportunity to specify his local preferences via graphical interface. This interface is different for ordinal and nominal attributes.

- Ordinal attributes – the interface shows possible values of this attribute in the first column. Every value has a slider in the second column. The slider enables user to specify a degree of preference for corresponding
attribute value (see fig. 6).

Nominal attributes – the interface shows a combo box with possible string values. If the user selects some value, it appears on the right side of the GUI together with a slider and a “remove” button. The slider has the same functionality as with ordinal attributes: it specifies a degree of preference for corresponding value. Fig. 7 shows the interface for attribute “seller type”. Two values (corporate entity and estate agency) have been previously selected from the combo box and their degrees of preference have been set with sliders. Remaining two values (mediator and private person) are still visible in the opened combo box.

All user preferences specified with this interface are stored to ontology and used for searching the best objects (see section V).

Note that the result of the hash function does not have to be found in the hash table. In our example, Table I contains only 5 user types and it does not contain e.g. type 11110. This would mean that no user of the same type (11110) ever entered our system before. In this case the user would receive fuzzy sets prevailing for all users, not only for users of the same user type. The new user 11110 would have right-trapezoidal fuzzy set for floor and left-trapezoidal fuzzy set for size.

Thus we can acquire approximate preferences even for those users who do not provide them explicitly. These preferences form an input for top-k searching algorithm described in the next section.

V. TOP-K SEARCH

Searching the datasets is a very important part of the process of retrieving relevant objects to users. We assume that a user wants to find a few best objects only. This assumption is quite natural especially over huge datasets.

The aim of the class of top-k search algorithms is to retrieve k best objects from the dataset without a scan of the whole dataset. Thus we can find top 5 objects by scanning e.g. 20% of all data in a dataset.

Top-k querying deals with tradeoff between query expressive power and computational complexity. Approaches that are optimal in computational complexity usually do not support high expressive queries.

In the family of Threshold algorithms (TAs) [7] the basic assumption is a monotone combination function. The monotone combination function can be used to express our global preferences only. On the other hand these algorithms are optimal [7] and fast.

In [5,6] authors face the problem of more expressive queries (that cover our preference model) by analyzing complicate ranking functions and offer a kind of multidimensional search. These approaches are slower and the analysis of the function is rather hard. Moreover the algorithm in [5] can find the best object, but not the next ones.

In our approach we propose an extension of TAs family that allows computing of queries that cover our preference
model. Let objects have \( m \) attributes. Original TAs assume to have \( m \) ordered lists \( L_1, \ldots, L_m \) and for every \( i \), \( L_i \) is ordered by local preference function \( f_i \) considered to be the same for all users. Consider the object \( X \) with attribute values \( x_1, \ldots, x_m \). The monotone combination function \( F(f_1(x_1), \ldots, f_m(x_m)) \) expresses the user preference to the object \( X \).

In our approach we assume that users have different local preferences as well as different global preference. Reordering the sources for each local preference is unsuitable, because we want to access only necessary data from the sources. A query expressing the user preferences in Reordering the sources for each local preference is unsuitable, because we want to access only necessary data from the sources. A query expressing the user preferences in

L*(fuzzyfied and crisp) attributes separately.

worst entries in each sorted list one by one from search for objects relevant to user preferences.

We omit the description of original TAs due to space limitation. They can be found in [7]. The TAs access the entries in each sorted list one by one from “the best” to “the worst” with respect to the local preference. We consider the simulation of sorted access over ordinal (fuzzy) and nominal (fuzzyfied and crisp) attributes separately.

\[
\text{SortedStreamB+tree}
\]

\[
\text{Input:} \quad \text{membership function } f \text{ of a fuzzy set (as a black box)}
\]

\[
\text{-Set } X := \{x: f(x) \text{ is a representative of each local maximum of } f\}
\]

\[
\text{Output:} \quad \text{The next best object with its fuzzy value in the sorted output stream.}
\]

\[
\text{Function getNext()}
\]

If (first call) then

\[
\text{For all } x \in X \text{ do}
\]

\[
\text{Traverse from the root of B+ tree to find neighbor records } <a, x_a>, <b, x_b> \text{ in a leaf (or leafs) such that } a \leq x \leq b \text{ and add triples } [p(o), f(x), \text{left}] \text{ and } [p(b), f(x), \text{right}] \text{ to } T \text{, if there were no such records on the left or on the right do not add the corresponding triple } [p(o), f(x), \text{direction}] := \text{max}(T).
\]

\[
\text{Return } <x, f(x)>.
\]

Else

\[
\text{If } T=\emptyset \text{ return "no more objects" and exit;}
\]

\[
[p(o), f(x), \text{direction}] := \text{max}(T); \text{ (last returned object)}
\]

\[
\text{Remove } [p(o), f(x), \text{direction}] \text{ from } T
\]

\[
\text{If direction, from p(o) shows to the leaf on the disk, load it to the memory, if there is not such a leaf return getNext() and exit;}
\]

\[
\text{Traverse to the record } <a, x_a> \text{ defined by } p(o) \text{ and direction,}
\]

\[
\text{If } [p(o), f(x), \text{right}] \text{ in } T \text{ (in local minimum)}
\]

\[
\text{Replace it by } [p(o), f(x), \text{null}] \text{ in } T
\]

\[
\text{Else add } [p(o), f(x), \text{direction}] \text{ to the set } T
\]

\[
[p(v), f(x), \text{direction}] := \text{max}(T); \text{ (next returned object)}
\]

\[
\text{If direction} \neq \text{null remove } [p(v), f(x), \text{direction}] \text{ from } T
\]

\[
\text{Return } <x, f(x)>.
\]

Fig. 8. Sorted stream algorithm over B+ tree.

A. Ordinal attributes

The most common attributes are ordinal ones. Typical structures for indexing the ordinal attributes are B+trees. Pages of leaf data are double linked and we can easily traverse between them in both directions. The leaf traversal supported by B+ tree allows us to follow records (attribute values) from high preferred to low preferred. When a user defines a fuzzy set over the ordinal attribute, he/she actually defines the ordering of the attribute domain.

Our simulation assumes that we can find local maximums of the membership function. With partially linear membership function as represented in our model, the identification of local maximums is trivial. Local maximums are the starting points of traversing the B+ tree. After identification the local maximums, the whole membership function is used as a black box.

The getNext() function simulating the sorted access over B+ tree is formally described in fig. 9. Any triple \([p(o), f(x), \text{direction}]\) in \( T \) represents the following: \( p(o) \) is the position of object \( o \) in a materialized leaf in memory, \( f(x) \) is the relevance of the attribute value \( x \), of object \( o \) given by membership function \( f \) and the third value represents the direction of the next move after returning the pair \(<o, f(x)>\) to the master algorithm.

Function max(T) returns triple with the highest score. If there are more triples with same score, max(T) returns the one with the lowest object id.

Fig. 9. B+ tree traversing.

Example 1: To illustrate the simulation of sorted access consider the situation on fig. 9. Under the B+tree we can see the fuzzy set. The membership function has one local maximum: \( X \) = the interval <200, 250>. When top-k algorithm calls for the first object by sorted access, we need to search the random value in the interval (e.g. 225) by hierarchical traversal. We find 2 neighbor objects around 225 i.e. we add triples \([p(10), 1, \text{right}]\) and \([p(1), 1, \text{left}]\) to \( T \). Seeing that the both fuzzy values are the same we return object with lower object id i.e. <1,1>. In further sorted access calls we will follow one of the gray arrows to get new objects to return. Next we traverse left for the object 3 with fuzzy value 0.9 but return better object 10 with fuzzy value 1.0. After the next sorted access call we follow the direction of the last returned object (i.e. to the right) and compute fuzzy value 1.0 of object 7. After the next sorted access call and computing fuzzy value 0.6 of object 4 we need to sent object 3 and traverse to the left in the next call.

Traversing of the B+ tree is much easier when the membership function \( f \) is monotone (left-trapezoidal of right-trapezoidal). In this case we follow only one direction (the size of \( T \) is 1).

It can be easily shown that our getNext() function over B+ tree access only necessary leafs, thus it does the optimal
number of disk accesses – only a few more pages (if any) than the sorted access over the sorted list used by original TAs.

B. Nominal attributes

Usual types of nominal attributes are Strings and/or lists of domain values (Enumeration). The unordered character of nominal attributes often causes their absence in top-k algorithms. Usually, the nominal attributes are used to express the restrictive parts of queries only.

The suitability of the nominal attribute values can be expressed by assigning fuzzy values (membership in a fuzzy set) to the domain attribute values one by one. The assignment is expressed by a user and can be completely different for each user.

We created a very simple index structure Dis-index depicted on fig. 10. In Dis-index each domain value is stored in memory together with the corresponding pointer to the first page on the disk with objects ids having appropriate domain value. Every inner page on the disk stores also the pointer to the next page holding ids with the same attribute value. Dis-index stores much more objects per page than B+ tree i.e. needs less accesses to the disk. We focus especially on small attribute domains where the fuzzyfication made by a user is easy to specify. The main problem of Dis-tree over bigger domains is many ties. If the cardinality of an active domain of a nominal attribute is huge, then B+ tree is a better choice.

The simulation of the sorted access over Dis-index is straightforward.

If we have a fuzzyfication \( f \) of the actual attribute domain, we sort the domain values according to \( f \) (ordering of a domain in memory is very fast). First we return ids of the value with the best fuzzyfication value. If all objects under the best value are returned we follow the pages under the next best value etc.

For crisp local preference we simply access the pages under preferred domain values in a similar way as in previous case.

The correctness of the sorted access over Dis-index is evident. The big number of stored ids per page makes the Dis-index an effective variant to the B+ tree for small attribute domains especially for nominal attributes.

VI. CONCLUSION AND FUTURE WORK

The user model proposed in this paper is complex, but natural and suitable for fast searching. It is possible to extend this model with other kinds of, namely hierarchical and spatial attributes.

A typical hierarchical attribute is geographical region. We recognize regions on different levels like state, country, city, etc. These regions form a natural hierarchy. If we look for a flat in Manhattan, we consider New York counties (like Staten Island) more similar to Manhattan than other counties.

Other interesting type is spatial attribute. To support such attributes, we need to store GPS coordinates of flats and other places. This allows user to find flats located close to his working place or some special place of interest.

Other domains may include even more specific attributes which can enrich our preference model in the future.

REFERENCES