On Ranking Refinements in the Step-by-step Searching through a Product Catalogue

Nenad Stojanovic
Institute AIFB, University of Karlsruhe, Germany
nst@aifb.uni-karlsruhe.de

Abstract

In our previous work we have developed a logic-based approach for the refinement of ontology-based queries that enables a user to search through a repository in a step-by-step fashion. Since the set of refinements in a step can be large, they should be ranked according to their relevance for fulfilling a user's need. In this paper we present such a ranking model, which takes into account the information content (informativeness) of a refinement as well as the preferences of the user.

1. Introduction

In our previous work [1] we present an approach for refining ontology-based, so called logic-based query refinement, that enables a user to navigate through the information content incrementally and interactively. In each refinement step a user is provided with a complete but minimal set of refinements, which enables him to develop/express his information need in a step-by-step fashion. The approach is based on the model-theoretic interpretation of the refinement problem, so that the query refinement process can be considered as the process of inferring all queries which are subsumed by a given query. Due to the complexity of subsumption reasoning, we use an alternative, more tractable generality order, θ-subsumption, frequently used for efficient implementation of inductive logic programming tasks. Since the θ-subsumption is a partial order relation, it generates a lattice of refinements, which serves as the basic structure for the step-by-step query refinement.

Although the logic-based query refinement generates the minimal number of refinements, it is possible that the entire set of refinements is too large for the searching habits of an average user. Indeed, by using an analogy to the browsing results of a query, one can expect that an average user will consider only several top-ranked refinements. Therefore, the set of refinements should be ranked according to their importance for fulfilling a user's need and in this paper we present such a ranking model. In Section 2 we first explain the basic terminology, whereas in Section 3 we present the ranking model. In Section 4 we give some concluding remarks.

2. Background

In this section we give the basic assumption/terminology we use in this paper.

Definition 1: Ontology
An ontology is a structure $O := (C, \leq_c, R, \sigma)$ consisting of:

- two disjoint sets C and R whose elements are called concept identifiers and relation identifiers, resp.,
- a partial order $\leq_c$ on C, called concept hierarchy or taxonomy (without cycles)
- a function $\sigma : R \rightarrow C^+$, called signature.

Definition 2: Knowledge Base
A Knowledge Base is a structure $KB := (C_{KB}, R_{KB}, I, l_c, l_r)$ consisting of:

- two disjoint sets C_{KB} and R_{KB}
- a set I whose elements are called instance identifiers (or instances or objects shortly)
- a function $l_C : C_{KB} \rightarrow I$ called concept instantiation
- a function $l_r : R_{KB} \rightarrow I^+$ called relation instantiation

A relation instance can be depicted as $r(I_1, I_2, ..., I_n)$, where $r \in R_{KB}, I_i \in I$. Similarly, $c(I_i)$, where $c \in C_{KB}$, represents the concept the instance I_i belongs to. r is called a predicate and I_i is called a term.

Definition 3: Query
A conjunctive query is of the form or can be rewritten into the form: $Q(\overline{X}) \equiv forall\ \overline{X}\ \ \overline{P}(\overline{X}, \overline{k})$, with $\overline{X}$ being a vector of variables $(X_1, ..., X_n)$, $\overline{k}$ being a vector of constants (concept instances), $\overline{P}$ being a vector of conjoined predicates (relations).

For example, for the query "forall x,y worksIn(x, KM) and researchIn(x, y)" (1)
we have: $\overline{X} := (x, y)$, $k := (KM)$, $\overline{P} := (P_1, P_2)$, $P_1(a,b,c) :=$ `worksIn(a,b)`, $P_2(a,b,c) :=$ `researchIn(a,c)`.

Since a predicate constrains the interpretation of a variable in a query, in the rest of the text we will use the term *query constraint* as the description of a predicate. For example, `researchIn(x, y)` is a constraint for the interpretation of the variable x.

3. Ranking model

In order to determine the relevance of a refinement for a user's need, we use two sources of information: (a) user's preferences for such a refinement and (b) informativeness (information content) of a refinement.

3.1. User's preferences

Since the users are reluctant to provide an explicit information about the relevance of a result, the ranking has to be based on the implicit information that are captured by observing user's behaviour, so-called implicit relevance feedback [2]. In the query refinement a user interacts subsequently with the system so that, by discovering user's preferences, we have to take into account not only the last query a user made, but rather the whole process of creating a query (query session). We define three types of such an implicit relevance feedback: *Actuality*, *ImplicitRelevance* and *ImplicitIrrelevance*.

***Actuality*.** The *Actuality* parameter reflects the phenomena, accounted in the IR research, that a user may change the criteria about the relevance of a query term, when encountering newly retrieved results. In other words, the constraints most recently introduced in a user's query are more indicative of what the user currently finds relevant for his need. We model it using the analogy to the ostensive relevance proposed in [3]:

$Actuality(c,Qs) = \frac{1}{num\_session(c,Qs)+1}$, where c is a query constraint , Qs is the current query session and *num_session*(c, Qs) is the number of refinement steps, which the constraint c is involved in.

***ImplicitRelevance*.** The theory about the implicit relevance feedback postulates that if a user selects a resource from the list of retrieved results, then this resource corresponds, to some extent, to his information need. However, a click on a particular resource in the resulted list cannot be treated as an absolute relevance judgement, since users typically scan only the top l ranked ($l \approx 10$) resources. For example, maybe a document ranked much lower in the list was much more relevant, but the user newer saw it. It appears that users click on the (relatively) most promising resources in the top l, independent of their absolute relevance. However, if we assume that a user scans the list of results from top to bottom, the relative relevance is evident: all non-clicked-on resources placed above a clicked-on resource are less relevant than the clicked-on resource. Obviously, the relevance is related to some feature that are contained in the clicked-on resource and not contained in non-clicked-on resources. It means that by analysing the commonalities in the attributes of results a user clicked/not clicked, we can infer more information about the intension of the user in the current query session. In order to achieve this, we define the relation *preferred ranking* as $R_i <_{rQ^*} R_j$ for all pairs $1 <= j < i$, with $i \in C$ and $j \notin C$, where (R_1, R_2, R_3, …) is a ranked list of resources, set C contains the ranks of the clicked-on resources and Q is the posted query.

By analysing the difference between the features (attributes, constraints) of the clicked-on and non-clicked-on resources we get a set of so called *Preferred* constraints for a query Q in the following manner:
$Preferred(Q) = \{con \mid con \in \cup_j Preferred_j(Q)\}$, where
$Preferred_j(Q) = \{el \mid el \in Attr(R_j) \setminus \cup_i Attr(R_i), \forall i\ R_i <_{rQ^*} R_j\}$,
$Attr(R_x)$ is the set of constraints (attributes) that are defined for the resource R_x.

Therefore, the set of constraints that seems to be relevant for a user in a query Q_s can be calculated as: (1)

$$ImplRel(c,Qs) = \begin{cases} 0, & c \notin Preferred(Qs) \\ \frac{1}{n} \sum_{i=1,n} \frac{1}{num\_sessions(c,Q_s,i)}, & c \in Preferred(Qs) \end{cases}$$

, where *num_sessions*(c, Qs, i) is the number of refinement steps which the constraint c is involved as a preferred constraint in, whereas n is the total number of times the constraint c is treated as a preferred constraint in the current session Qs.

***ImplicitIrrelevance*.** Since the recommended refinements are presented to a user in the decreasing order of relevance, one can assume that if a user has selected n-th ranked results, then the first n-1 ranked results (constraints) are wrongly ranked on the top of the list of the refinements. We call these constraint *implicit irrelevant*. They are calculated in similar manner as implicit relevant constraints: (2)

$$ImplIrrel(c,Qs) = \begin{cases} 0, & c \notin Nonpreferred(Qs) \\ \sum_{i=1,n} \frac{1}{num\_sessions(c,Q_s,i)}, & c \in Nonpreferred(Qs) \end{cases}$$

, where $NonPreferred(Q) = \{con \mid con \in \cup_j NonPreferred(Q)_j\}$
$NonPreferred_j(Q) = \{el \mid el \in \cup_i Attr(R_i) \setminus Attr(R_j), \forall i\ R_i <_{rQ^*} R_j\}$.
The definition of *num_sessions*(c, Qs, i) is analogue to (2), but regarding implicit irrelevance.

Similar to (1), formula (2) enables the correction of false assumptions (regarding the preferences of the current user) made in the ranking process.

Formulas (1) and (2) ensures the self-adaptivity of the ranking system, e.g. they do not allow that the system repeatedly ranks a non-interesting refinement highly. Finally, the calculated implicit relevance of a refinement c looks like:

$$Relevance(c,Qs) = \frac{ImplRel(c,Qs)+1}{ImplIrrel(c,Qs)+1}$$

3.2. Informativeness

Another source to define relevance of a refinement is

its informativeness or information content. From the machine learning research it is known that usefulness of an attribute (i.e. constraint) for traversing the searching space is proportional to its information content, that is frequently measured using entropy [4]. Indeed, the entropy shows the interestingness of a constraint regarding its relevance for the user's need.
In order to illustrate the relevance model let us consider the knowledge base represented in Figure 1.

car	Type	GPS	Automatic
c1	familyCar	g1	a1
c2	sportsCar	g2	a1
c3	sportsCar	g3	a2
c4	sportsCar	g4	a3
c5	familyCar	g3	a3
c6	familyCar	g1	a2

GPS	GPSType	Language
g1	auto	eng
g2	semi	ger
g3	semi	fra
g4	auto	fra

Automatic	AutomaticType	Level
a1	type1	low
a2	type2	high
a3	type1	high

Figure 1. The knowledge base used for illustrating relevance model

However, there are two problems in direct applying the information theory for selecting the most informative attribute in the ontology-based querying:

1. Since in an ontology-based query an attribute is assigned to a concept (i.e. variable), before determining the informativeness of an attribute (using the entropy), the informativeness of a concept has to be determined. For example, in the query (3)
Q = "forall x,y,z Car(x) and GPS(y) and Automatic(z) and hasType(sportsCar, x) and hasLuxury(y, x) and hasLuxury(z, x)", the question is which of the variables x, y, z is the most suitable for the further refinement.
We define the suitability of each of these variables in the following manner:
Suitability(X, Q) = *VariableAmbiguity*(X, Q)/*Gain*(X, Q). (4)

$$VariableAmbiguity(X,Q) = \frac{|Relation(Type(X))| + 1}{|Assigned\,Relations(Type(X),Q)| + 1},$$

where *Relation*(*C*) is the set of all relations defined for the concept *C* in the ontology, *AssignedRelations*(*C*,*Q*) is the set of all relations defined in the set *Relation(C)* and which appear in the query *Q*.
If we assume that there two relations are defined for the concept GPS, then regarding (3):
VariableAmbiguity(y, Q) = (2+1)/1 = 3.
Gain(X, Q) = *Info*(T) – *Info*(X, T) is the standard measure of the informativeness [4], i.e. for the probability distribution of the values (instances) that belongs to the variable X, X = (x1, ..., xn)

$$Info(X,T) = \sum_{i=1,n} \frac{|x_i|}{|T|} E(x_i) \quad (5)$$

where T is the set of all examples relevant for the query Q and E is the standard measure for the entropy:

$$E(w) = -\sum_{i \in Category} w_i * log(w_i) \quad (6)$$

where *Category* is the set of all categories for the classification and wi is the distribution of instances that belong to the category *i* (w = (w_1, ..., w_c)).
In order to compensate the effect of multivalued attributes the *SplitInfo* factor is introduced:
Gain(X, T) = *Gain*(X, T)/*SplitInfo*(X, T) (7)

$$SplitInfo(X,T) = -\sum_{i=1,n} \frac{x_i}{T_i} * log(\frac{x_i}{T}).$$

For the given problem domain we define only two classes an example (instance) can belong to: *Relevant* and *Irrelevant*, representing the set of relevant and irrelevant examples regarding a users need (i.e. query). The set of relevant examples for the query (3) is depicted by the rectangle in the first table in Figure 1 (all sportsCars).
Further, for the set T of examples given in Figure 1,
Info(T) = E(3/6, 3/6)
Info(y, T) = ((2/6)*E(0,1) + (1/6)*E(1,0) + (2/6)*E(1/2,1/2) + (1/6)*E(1,0))
SplitInfo(y, T) = ((2/6)*log(2/6) + (1/6)*log(1/6) + (2/6)*log(2/6)+ (1/6)*log(1/6))

By applying (4) on variable x, y, z from query (3) we get that the variable y (GPS) is most suitable for the further refinement.

2. The second problem is the determination of the informativeness of an attribute of a variable from the query, since the relevance categories are defined on the level of the concepts. For example, by considering left-bottom table it is difficult to say if the example g_3 is relevant or irrelevant since according to the query (3) it can be treated as both of them.

Therefore, we need a notation of fuzzy entropy, which we introduced in the following manner:

For each example x_i (instance) we introduce a partial membership to a category p(x_i) that is calculated from the set T. For example p(g1)=0, p(g3)=0.5.
Further the calculation of Entropy includes this fuzzy probability

$$E(w) = -\sum_{i \in Category} \frac{\sum_{j=1,k} p(w_j)}{k} * log(\frac{\sum_{j=1,k} p(w_j)}{k}),$$

where k is the number of examples that belong to a category.
For example, regarding attribute GPSType from Figure 1:
E(semi) = -((1+1/2)/2)*log(1+1/2)/2) + (0+1/2)/2)*log(0+1/2)/2).

Formulas (5) and (7) are used to determine the informativeness of an attribute.

Finally, for a constraint *e* the *Informativeness* is calculated as: *Informativeness*(e,Q)= *Suitability*(*Variable*(e),Q))**Gain*(*Predicate*(e), Q), where *Variable* retrieves the variable from a constraint

For example, a possible refinement of the query (3) can be e = " and GPSType(var, y)".

Informativeness(e,Q)=*Suitability*(*y*,Q))* *Gain*(GPSType, Q),

Note that for numeric features, the values have to be discretized, for instance in equi-depth buckets [4].

3.3. The model

We assume that with every decision taken by the user more information is gained about the user's intensions and background. Each transition (refinement) from the query *e* to query *d* involves a conscious choice by the user by preferring the constraints contained in *d* to the other options in the constraint *e*. The probability of a constraint *d* being the target of the navigation process is related to the distance fluctuations while travelling the navigation path. Recent fluctuations have more impact then past fluctuations. For this purpose we will transform the search space into a transition network, allowing the use of Markov chains theory. First, the set of states is defined as the set of *query constraints* augmented with a special state called *stop*. This state represents the termination of the search process. The transition between states are defined as follows: if constraints *x* and *y* are connected regarding the ontology structure, then they are connected in the transition network.

We assume for each transaction *e* a probability $q_e > 0$ of occurring. In a transition network, for each state *a*: $\sum_{b:a\to b} q_{a\to b} = 1$. The transition matrix T is defined as

$$T(x,y) = \{ \begin{matrix} q_s & if \quad x \in Related(y) \\ 0 & otherwise \end{matrix},$$

where q_s is uniformly distributed over all related constraints and *Related*(*c*) is a function that retrieves constraints that are related to *c* regarding the underlying ontology.

Further, $T'(x,y)$ is the probability of reaching *y* starting from *x* in *i* transitions. $T^0 = I$, the identity matrix. Thus, the sum $\sum_{i=0}^{\infty} T^i(e,d)$ is the probability of reaching *d* from *e* in any number of steps. Next we focus on the probability *Pr*(*d* | *e*) of a constraint *d* being the search target of a navigation path. The destination probability for *d* after traversing a path from *e* is defined as follows: $Pr(d \mid e) = \sum_{i=0}^{\infty} T^i(e,d) \cdot T(d,stop)$.

The infinite sum converges to (*I*-*T*)-1(*e*,*d*). This requires the calculation of the inverse matrix of an |Ω(*O*)| x |Ω(*O*)| matrix, where Ω(*O*) is the set of elementary constraints regarding given ontology *O*. Note that this operation can be calculated off-line.

However, in a step-by-step refinement a user is navigating through queries Q_x that contain several constraints. In that case we expand the probability to include this information as follows:

$$Pr(Q_d \mid Q_e) = \frac{1}{|Q_d|} \cdot \sum_{d \in Q_d} \frac{1}{|Q_e|} \cdot \sum_{e \in Q_e} Pr(d \mid e),$$

where *d* and *e* are constraints from queries Q_d and Q_e respectively. | Q_d | depicts the number of constraints in Q_d.

We use this destination probability function as a starting point for computing which neighbours bring the user most direct towards the highest probable destination constraint. Indeed, as we mentioned above the refinement process depends on the searching history (parameters *Actuality* and *Relevance*) and the content of the repository (parameters *Informativeness*).

This is formalized by assigning the coefficient *Rank* to each query Q_d that belongs to the lattice of the refinement for the query Q_e:

$$Rank(Q_d \mid Q_e) = \frac{1}{|Q_d|} \cdot \sum_{d \in Q_d} \frac{1}{|Q_e|} \cdot$$

$$(\sum_{e \in Q_e} Pr(d \mid e) \cdot (\frac{\lambda \cdot Relevance(d) \cdot Actuality(e) + Informativeness(e, Q_e)}{\lambda + 1})),$$

where λ is a forgetfulness coefficient that model the impact on the past user's behaviour on the ranking process: λ = 0 - the past is forgotten, λ < 1 - the past carries less weight then the present, usually λ = 1/2.

4. Conclusion

In this paper we have presented a comprehensive model for ranking refinements in a query refinement process. It uses two sources of information: a user's preferences for such a refinement and informativeness of a refinement. In that way the approach prioritises highly relevant refinements, i.e. the refinements that are related to the very characteristic (regarding a query) constraints and tailored to the user's need. Moreover, by using a user's implicit relevance feedback, the approach learns from its failures and successes, i.e. it has a self-improvement nature.

Acknowledgement. Research for this paper was partially financed by BMBF in the project "SemIPort" (08C5939).

5. References

[1] N. Stojanovic, "A Logic-based Approach for Query Refinement", WI 2004, IEEE, in press

[2] G. Salton, and C. Buckley, "Improving retrieval performance by relevance feedback", *Journal of the American Society for Information Science*, 41(4): 288-297, 1990.

[3] I. Ruthven, M. Lalmas, and C.J. van Rijsbergen, "Incorporating user search behaviour into relevance feedback", *Journal of the American Society for Information Science and Technology*, 54. 6. pp.528-548. 2003.

[Wit00] I.H. Witten, E. Frank, Data Mining, Morgan Kaufmann Publisher, 2000