Query-Independent Learning to Rank for RDF Entity Search

Conference Paper · May 2012

DOI: 10.1007/978-3-642-30284-8_39

Blaž Fortuna
Jožef Stefan Institute
80 PUBLICATIONS 1,253 CITATIONS
SEE PROFILE

Thanh Tran
Sheffield Hallam University
25 PUBLICATIONS 480 CITATIONS
SEE PROFILE

Dunja Mladenić
Jožef Stefan Institute
364 PUBLICATIONS 5,216 CITATIONS
SEE PROFILE

Query-Independent Learning to Rank for RDF Entity Search

Lorand Dali[1], Blaž Fortuna[1], Thanh Tran[2], Dunja Mladenić[1,3]

[1] Jožef Stefan Institute, 1000 Ljubljana, Slovenia
{Lorand.Dali, Blaz.Fortuna, dunja.mladenic}@ijs.si

[2] Institute AIFB, Geb. 11.40 KIT-Campus Sd, D-76128 Karlsruhe, Germany
duc.tran@kit.edu

[3] Jožef Stefan International Postgraduate School, 1000 Ljubljana, Slovenia

Abstract. The amount of structured data is growing rapidly. Given a *structured query* that asks for some entities, the number of matching candidate results is often very high. The problem of ranking these results has gained attention. Because results in this setting equally and perfectly match the query, existing ranking approaches often use features that are independent of the query. A popular one is based on the notion of centrality that is derived via PageRank. In this paper, we adopt *learning to rank* approach to this structured query setting, provide a systematic categorization of *query-independent features* that can be used for that, and finally, discuss how to leverage information in access logs to automatically derive the *training data* needed for learning. In experiments using real-world datasets and human evaluation based on crowd sourcing, we show the superior performance of our approach over two relevant baselines.

Keywords: information retrieval, learning to rank, semantic search

1 Introduction

With the development of the Semantic Web as a Web of interlinked resource descriptions represented in RDF (e.g. Linked Data), and the continuous increase in the number of publicly available datasets, the problem of retrieval and ranking RDF resources has gained attention. Basically, a *RDF resource description* is a set of *triples*, which capture the *relations* of that resource to other resources, and its *attribute* values. On the Web today, we can find descriptions for different kind of entities, such as organizations, people, and geographic locations. Data that have been made publicly available through the Linking Open Data initiative for instance, include both encyclopedic knowledge captured by general datasets such as DBpedia and specific knowledge in various domains (music, life science, etc.).

For searching RDF resource descriptions (henceforth also called *entity search* because resources stand for real-world entities), the keyword paradigm commonly used for Information Retrieval (IR) has been adopted. Also, interfaces based on *structured query languages*, SPARQL in particular, are widely employed. Basically, SPARQL rests on the notion of graph pattern matching. It is widely used for retrieving RDF data because RDF triples form a graph, and graph patterns matching subgraphs of this graph can be specified as SPARQL queries. Most endpoints, which provide public Web access to the kind of RDF data mentioned above, support SPARQL queries. While keyword search is clearly easier to use, structured query languages such as SPARQL can provide the expressiveness (technical) users may need in order to capture complex information needs, and to fully harness the structure and semantics captured by the underlying data. In fact, many queries posed on the Web are actually specified using form- and facet-based interfaces (e.g. facetted search provided by Yahoo!, Amazon and EBay). The inputs provided by the users through these interfaces are actually mapped to structured queries.

Since structured queries precisely capture the constraints the candidate answers must satisfy, the underlying engine can return perfectly sound and complete results. That is, all results can be found (*complete*) and every one of them perfectly matches the query (*sound*). However, given the large amount of data, queries may result in a large number of results, while only a few of them may be of interest to the user. In this case, ranking and returning only the top-*k* results is the standard strategy used in practical scenarios to improve efficiency and response time behavior. Studies have shown that users typically scan results beginning from the top, and usually focus only on the top three or four [1]. However, how do we *rank entity search results in this structured query scenario,* given all entities equally (i.e. perfectly) match the query?

A few specific approaches have been proposed to deal with ranking RDF results [5,7,10]. Most of these approaches [5] assume an ambiguous keyword query such that the ranking problem is mainly understood as the one of computing *content relevance*, i.e. to find out whether the resource's content is relevant with respect to the query. In the structured query setting, all resources are equally relevant. Ranking approaches [10,11] that can be used to distinguish resources in this setting are mainly based on *centrality*, a notion of "popularity" that is derived from the data via PageRank. Besides centrality, we study the use of other features and incorporate them into a *learning to rank* (LTR) framework for ranking entity search results, given structured queries. The main contributions of this paper can be summarized as follows:

(1) **Learning to rank over RDF data**. LTR [2] is a state-of-the-art IR technique that learns a ranking function from labeled training data (relevance judgments). We show how LTR can be adopted for ranking entity search results over RDF data.
(2) **Query-independent features**. Critical for the performance of LTR are features. For this specific structured query setting, we systematically identify query-independent features (those that go beyond content relevance) and individually analyze their impacts on ranking performance.
(3) **Access logs based ground truth and training data**. While LTR offers high performance, it critically depends on the availability of relevance judgments for training. We observed from our experiments based on real users (via a crowd sourcing based evaluation recently proposed in [3]) that the final results strongly correlate with the *number of visits* (#visits) that is captured in the access logs. We provide a detailed analysis of this correlation and for the case where training data and ground truth is not easy to obtain, we propose the use of #visits as an alternative.

Using both cross-domain and domain-specific real world datasets, we evaluate the proposed LTR approach and show its superior performance over two relevant baselines. Results suggest that combining different features yields high and robust performance. Surprisingly, the use of features that are derived from the external Web corpus (features that are independent of the query and local dataset) yields the best performance in many cases.

Structure. The remainder of the paper is structured as follows. We firstly discuss related work in Section 2. Then, our adaptation of LTR is presented in Section 3. Experimental results are discussed in Section 4 before we conclude in Section 5.

2 Related Work

Approaches for ranking in the RDF setting can be distinguished into those which consider the *relevance* of a resource with respect to the query, and the others, which derive different features (e.g. *popularity*) from cues captured in the data such as centrality and frequency.

2.1 Query-dependent Content Relevance

Approaches using query-dependent content relevance include the main IR approaches that basically, rank a result based on the likelihood its content is relevant, given the query. Different IR approaches have been adopted to rank structured results, and to deal with RDF in particular. An adoption of the *vector space model* has been proposed for ontology-based IR [4]. More specifically focused on the ranking of structured results in RDF is the work from Blanco et al. [5] that is built upon *BM25F*, another IR approach widely adopted in commercial search engines. The idea here is to use different fields for indexing different properties of RDF resources. Different weights are assigned to these fields to recognize that some fields are more important than others in ranking RDF resources. Also, the more recent *language modeling* (LM) paradigm has been adapted to the case of structured results. For ranking structured objects, three different models were studied [6]: The simple unstructured model treats all object attributes and values as vocabulary terms. The structured variant employs different term distributions for different attributes and assigns different weights to attributes (similar to the idea behind BM25F). Recently, LM is also used for ranking in the RDF setting [7]. For this, a language model is proposed for the query, and also RDF graphs matching the queries are represented as language models. As opposed to the approaches mentioned before, which model queries and documents based on words, the models employed here are probability distributions over RDF triples. The probability of a given triple should capture its "*informativeness*", which is measured based on witness counts. The authors implement this by issuing keyword queries for each triple using Web search engines, and used the reported result sizes as witness count estimates.
Ranking as performed by these mentioned approaches is based on the relevance of the content with respect to the given keyword query. We focus on the ranking of results to structured queries, which as opposed to the ambiguous keyword queries, are

precisely defined such that the query semantics can be fully harnessed to produce answers that are equally relevant. Thus in principle, content relevance can be expected to be less important in this case, and other features should be considered for ranking. Among the approaches mentioned above, the only exception that deals with structured queries is the LM-based ranking of RDF triples (graphs). As discussed, this work does not directly capture content relevance but relies on informativeness. We consider this as one baseline and show that using additional features can substantially outperform this. Previous works build upon the vector space model [4], language models [7], and probabilistic IR [5]. In this work, we adopt yet another popular IR paradigm, namely LTR [2]. This paradigm constitutes the state-of-the-art in IR, and is widely used by commercial Web search engines.

2.2 Query-independent Features

Approaches described in this subsection are not taking the query into account, but rather using query-independent features. An example of such features that are independent of the query is *centrality*, which can be derived from the graph-structured nature of the underlying data using algorithms such as PageRank [8] and HITS [9]. The aim of PageRank is to give a global, query-independent score to each page. The score computed by PageRank for a given page captures the likelihood of a random Web surfer to land on that page.

The first adoption of PageRank in the structured data setting was proposed for Entity-Relation graphs representing databases, and specific approaches for dealing with RDF graphs have been introduced recently. For instance, ResourceRank [10] is such a PageRank adapted metric that is iteratively computed for each resource in the RDF dataset. Also, a two layered version of PageRank has been proposed [11], where a resource gets a high rank if it has a high PageRank within its own graph, and if this graph has a high PageRank in the LOD cloud (which is also considered as a graph where nodes represent datasets). The difficulties in adapting PageRank to the structured data setting is that the graph here – as opposed to the Web graph – has heterogeneous nodes and edges (different types of resources and different relations and attribute edges). A solution is to manually assign weights to different relations, but this approach is only applicable in a restricted domain such as paper-author-conference collections [13].

Instead of centrality, more simple features based on *frequency* counts have also been used in the RDF setting. For instance, structured queries (graph patterns) representing interpretations of keyword queries have been ranked based on the frequency counts of nodes and edges [16]. Just like PageRank scores, these counts aim to reflect the popularity of the nodes and edges in the query pattern such that more popular queries are preferred. The use of frequency has long tradition in IR. Term and inverse document frequencies are commonly used to measure the importance of a term for a document relative to other terms in the collection.

These query-independent features can be directly applied to our structured query setting to distinguish between the results that are equally relevant. In a systematic fashion, we identify different categories of features that can be used for our LTR approach, including centrality and frequency. We show that besides the featurbes derived from the corpus (i.e. the underlying RDF graph), external information on the Web provides useful features too. We compare and show that the use of different features can outperform the ResourceRank baseline, which is based on centrality.

3 Query Independent Learning to Rank over RDF

We describe how we adapt LTR to the RDF entity search setting, followed by a detailed description of the features which the learning algorithm uses.

3.1 Learning to Rank

LTR [2] is a machine learning technique used to induce a ranking model from training data. We use the pairwise setting, which means that a training example is provided as a pair of entities and we know which of the two entities should be ranked higher in the result set. In what follows we formally describe the pairwise method and how it is adapted to our case.

For a given dataset D, let $Q = \{q_1, q_2, \cdots, q_n\}$ be the set of queries. For every query q_i let $R(q_i) = \{r_1^i,\ r_2^i,\ \cdots,\ r_k^i\}$ be the set of answers to q_i. We define a feature set as $F \coloneqq (f_1, f_2, \cdots, f_P)$ where f_j are the functions $f_j : R(q_i) \to [0, 1]$, which assign a real value to each answer, and we normalize the feature values to values between 0 and 1 for each query separately. We refer to $f_j(r)$ as a feature of the resource r. A target feature f_t (also called *label*) is a special feature, which determines the correct ranking as a descending ordering of the resources. It is the ranking based on the target feature f_t, which we want to obtain using a LTR algorithm.

For every answer r_j^i we compute a feature vector $x_j^i \coloneqq (f_1(r_j^i), f_2(r_j^i), \ldots, f_P(r_j^i))$. The feature vector x_j^i does not contain any target feature f_t. The training set consists of all pairs

$$\left(x_R^i, x_N^i\right), \quad \forall i \in \overline{1,n}, \quad \forall r_R^i, r_N^i \in R(q_i), \quad R \neq N$$

such that

$$f_t\left(r_R^i\right) > f_t\left(r_N^i\right).$$

In other words, for each query, we take all the pairs of the feature vectors of the answers to the query such that we put the answer with a higher target feature on the first place. To each pair $\left(x_R^i, x_N^i\right)$ we associate a cost C:

$$C \coloneqq \frac{2 f_t(r_R^i)}{f_t\left(r_R^i\right) + f_t\left(r_N^i\right)} - 1.$$

Intuitively we can think of C as the confidence in the correct ordering of the pair $\left(r_R^i, r_N^i\right)$ or as the penalty, which the learning algorithm receives if it makes a mistake on this pair. We can observe that if $f_t\left(r_R^i\right) = f_t\left(r_N^i\right)$ then $C = 0$, so we are not confident at all that r_R^i should be ranked higher than r_N^i. On the other hand if $f_t\left(r_R^i\right) \gg$

$f_t(r_N^i)$ then the value of C gets close to 1, and the learning algorithm obtains a big penalty for making a mistake on this pair.
The list of pairs $\left(x_R^i, x_N^i\right)$ with their associated cost C is the input to the RankSVM [17] algorithm described below. The goal is to learn a weight vector $w \in \mathbb{R}^P$ of the same dimensions as the training vectors x. Then given a new vector y, representing the feature vector of an answer to be ranked, we can compute the score of the answer, which is equal to the inner product between the weight vector w and the vector y,

$$score = w^T y\,.$$

The ranking is then obtained by sorting answers by their scores.

3.2 Rank SVM

Linear SVM [21] is a popular way of learning the weight vector w. Originally SVM is formulated as a binary classification problem, where w is the separating hyperplane with maximum margin. The linear soft margin SVM for classification can be adapted to the pairwise ranking problem. The objective is to make the inner product $w \cdot x_R^i$ greater than $w \cdot x_N^i$ by the margin 1 and allowing for some errors ξ. We have

$$w \cdot \left(x_R^i - x_N^i\right) \geq 1 - \xi_i, \forall i\,.$$

The maximum margin separating hyperplane is the one which minimizes

$$\frac{1}{2}\|w\|^2 + C\sum \xi_i\,.$$

This is called the primal problem, and it is the one which we shall solve as described in [22]. By substituting ξ_i we get the hinge loss

$$\frac{1}{2}\|w\|^2 + C\sum(1 - w \cdot (x_R^i - x_N^i))_+\,,$$

where the function $(\cdot)_+$ is defined as $(\cdot)_+ \coloneqq \max(0,\ \cdot)$. We minimize the hinge loss by using the subgradient method, which gives a fast but approximate solution.

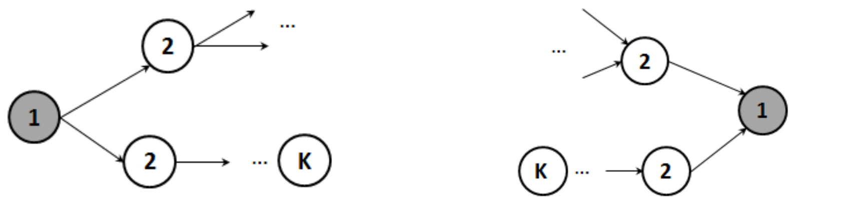
3.3 Feature Extraction

The proposed approach uses features which can be grouped into *dataset specific* or *dataset independent* features. Dataset specific features are extracted from the RDF graph. Dataset independent features are extracted from external sources like web search engines or N-gram databases. Note that although the dataset specific features are specific to the dataset, the methodology to extract these features can be applied to any RDF dataset.

We also classify the features into *frequency*-based features obtained by counting different patterns in RDF graphs or counting the number of occurrences in web search results or n-gram databases, and *centrality*-based features obtained by applying graph theoretic algorithms like PageRank or HITS on the RDF graph.

In the following subsections we shall describe each feature in detail.

3.3.1 Features Extracted From the RDF Graph



$f@K = |\{tr \in RDF \mid subj(tr) = K\}|$ $\quad$ $f@K = |\{tr \in RDF \mid obj(tr) = K\}|$

Figure 1 Number of subjects (left) and objects (right) at level K

In this section we describe several features extracted from the RDF graph. We look at the RDF dataset as a directed graph with resources as nodes and properties as edges. We define the concept of a *feature at level K* as follows. We call anchor node the node, which corresponds to the resource we want to extract the feature for. In Figure 1 and Figure 2 the anchor nodes are shaded. The feature at level 1 is the feature extracted from the anchor node. The feature at level 2 is computed from the nodes one step away from the anchor node. One step away means that we go to the neighbor either in the direction of the edge or in the opposite direction. In general the feature at level K is computed from the nodes which are K-1 steps away from the anchor node. In the experiments presented in this paper we have used the features at levels 1 and 2. In what follows we provide the list of features we extract from the RDF graph

Number of subjects @ K. This feature is a count of the triples, which have as subject the node for which we extract the feature. In Figure 1 on the left, the value of this feature at level 1 is 2 (because two arrows go out) and the value of this feature at level 2 is 3 (= 2 + 1).

Number of objects @ K. This feature is computed in a similar way to the number of subjects @ K, the difference being that now the number of triples with the node as object is counted (arrows coming in). The graph on the right side of Figure 1 illustrates the computation of this feature.

Number of types of outgoing predicates @ K. At each level this feature is the count of the elements of the set of predicates occuring at that level. The anchor node is the subject. This feature is illustrated on the left side of Figure 2.

Number of types of incoming predicates @ K. At each level this feature is the count of the elements of the set of predicates occuring at that level. The anchor node is the object. This feature is illustrated on the right side of Figure 2.

Average frequency of outgoing predicate @ K. We compute the frequency counts of all predicates in the dataset. At level K we take the set of predicates P_K and we compute the feature as the average of the frequency counts of the predicates in P_K.

Average frequency of incoming predicate @ K. This feature is similar to the previous feature, the difference being that we take into account the predicates which correspond to edges pointing towards the anchor node.

Number of literals. This feature counts how many times the anchor node occurs as the subject in an RDF triple where the object is a literal.

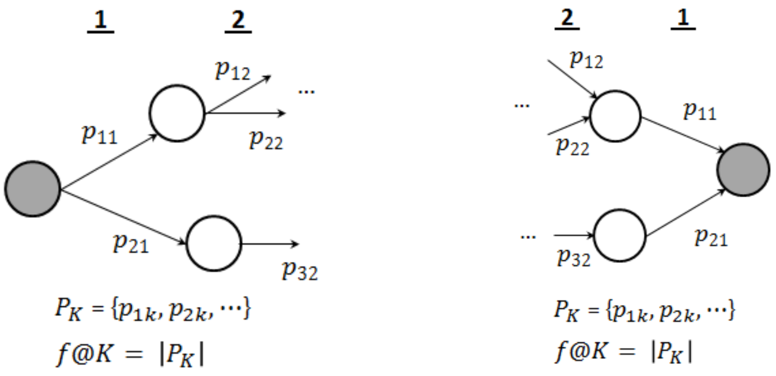


Figure 2 Number of types of outgoing and incoming predicates at K

3.3.2 PageRank

This section briefly describes the PageRank algorithm and how it applies to our case. PageRank was introduced in the early days of web search out of a need for a global, query independent ranking of the web pages. PageRank assumes a directed graph as input and will give a score to each of the nodes as a result. PageRank is based on the random walk model, which assumes that a very large number of users walk the graph choosing at each step a random neighbor of the current node or jumping to any node in the graph. The score of a node is given by the expected number of users being at the given node at a moment in time. The scores are computed recursively from the following equation:

$$\boldsymbol{p} = d \cdot \boldsymbol{M} \cdot \boldsymbol{p} + (1-d) \cdot \boldsymbol{u}, \qquad \boldsymbol{p}, \boldsymbol{u} \in \mathbb{R}^n, \boldsymbol{M} \in \boldsymbol{\mathcal{M}}(n)$$

Where n is the number of nodes in the graph, $\boldsymbol{p}$ is the PageRank vector containing the score for each node and is initialized with 0, $\boldsymbol{M}$ is the transition matrix constructed such that $\boldsymbol{M}[i,j] = 1$ if there is an edge from node i to node j and 0 otherwise. Moreover, to eliminate nodes which do not link to any other node we consider a sink node k such that $\boldsymbol{M}[i,k] = 1,\ \forall\, i$ and $\boldsymbol{M}[k,i] = 0,\ \forall\, i$. Finally the columns of M are normalized to sum up to 1; $\boldsymbol{u}$ is the jump vector and its entries are $\boldsymbol{u}[i] = \frac{1}{n}, \forall i$; d is the damping parameter, and represents the probability of walking to a neighboring node versus jumping. In our experiments we have set the value of d to its typical value of 0.85, and ran the iteration until it converged.

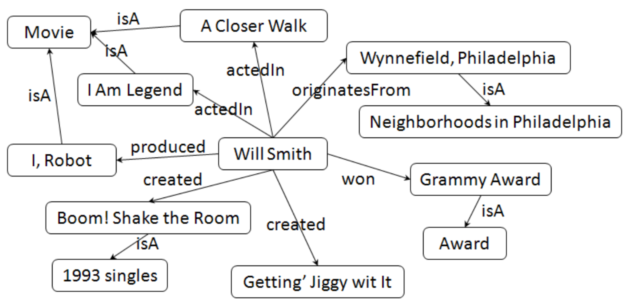


Figure 3 An example graph representation of a part of the Yago knowledge base

In case of the web, the graph is made of the web pages as nodes and the hyperlinks as edges. In our case the nodes are DBpedia or Yago resources or categories, and the edges are properties. For illustration, Figure 3 shows a subgraph from the Yago knowledge base.

3.3.3 Hubs and Authorities

Hubs and authorities, also known as Hyperlink-Induced Topic Search (HITS) algorithm, is an iterative algorithm which takes as input a directed graph and assigns two scores to each of its nodes. The two scores, the hub score and the authority score, are defined recursively in terms of each other such that a node gets a high hub score if it points to nodes with high authority scores, and a node gets a high authority score if it is pointed to by nodes with high hub scores.

$$\forall p, \qquad auth(p) = \sum_{i=1}^{n} hub(i)$$

$$\forall p, \qquad hub(p) = \sum_{i=1}^{n} auth(i)$$

where p is a node in the graph, n is the total number of nodes connected to p, and i is a node connected to p. $auth(p)$ and $hub(p)$ are initialized to 1.

3.3.4 Search Engine Based

We have used the search services provided by Yahoo! BOSS[1] to measure how many times the label of a resource (which corresponds to an answer to a query) appears on the internet. We do this by searching the web with the resource's label as a query and taking the number of hits as a feature. For instance to compute this feature for the resource corresponding to the person Neil Armstrong, we make a web search with the query ‘Neil Armstrong’ and obtain that the number of search results is 3720000.

[1] http://developer.yahoo.com/search/boss/

3.3.5 Google N-grams

Google released a dataset of all n-grams[2] (1-grams up to 5-grams) which appear on the internet at least 40 times, together with their frequency counts. We consider as a feature of a resource the frequency count of its label in the n-gram dataset. When the label of a resource is composed of many words we generate all 3-grams from the label and take the sum of the frequencies of the 3-grams as a feature. For instance, to compute this feature for the resource corresponding to the person Neil Armstrong we search for the 2-gram 'Neil Armstrong' and obtain that it occurs 132371 times in the Google N-gram database.

4 Experiments

Given RDF datasets and SPARQL queries, we obtained results using a Triple store. In the experiments, we run different versions of the proposed LRT algorithm and baselines to compute different rankings of these results. The goals of the experiments are (1) to compare LTR against the baselines and (2) to analyze the performance of individual features (feature sets). As performance measures, we use the standard measures NDCG and Spearman's correlation coefficient. We build upon the data, queries and methodology proposed by the recent SemSearch Challenge evaluation initiative [3]

4.1. Datasets and Queries

We have two sets of queries[3]. The first set is a subset of the entity queries provided by the SemSearch Challenge dataset. It consists of 25 queries, for which we obtain answers from DBpedia and Yago, two datasets containing encyclopedic knowledge that were extracted from Wikipedia infoboxes. Answers from these datasets correspond to Wikipedia articles. We used the Wikipedia access logs from June 2010 to January 2011 (available at http://dammit.lt/wikistats/).

The other set consists of 24 queries, whose answers are computed from the Semantic Web Dog Food (SWDF) dataset [18]. SWDF contains information about people, conferences, workshops, papers and organizations from the Semantic Web field. The dataset is built from metadata about conferences such as ISWC and ESWC, starting from the year 2006. For the USEWOD 2011 Data Challenge [19], a dataset[4] of access logs on the SWDF corpus was released.

While the first set of queries is used to evaluate ranking in a general setting, the second one is used to analyze how the approaches perform in a domain-specific setting.

[2] http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html
[3] http://aidemo.ijs.si/paper_supplement/dali_eswc2012_queries.zip
[4] http://data.semanticweb.org/usewod/2011/challenge.html

4.2 Ground Truth: Human Judgments vs. Access Logs Information

We follow the crowd sourcing approach of SemSearch [3] to obtain relevance judgments from human users. We give human evaluators the following task: Given a question and the answers computed by the system, they should vote for the one answer, which should be ranked first. Moreover, they should indicate the confidence in their choice. We used the number of votes (#votes) for an answer as the ranking criterion. For each question twenty evaluators have voted.

We also propose an automatic way to obtain the ground truth by using access logs. We take as the ranking score of a resource the number of times that resource has been visited, where #visits is obtained from access logs.

We now discuss the correlation between the ranking resulting from human judgments and the one based on #visits. Figure 4 shows one example question, namely "List of boroughs in New York City". The upper bar (red) shows the percentage of visits, and the lower one (blue) shows the percentage of votes this answer has received. It can be seen that the ranking based on #visits is almost the same as the ranking based #votes, especially for the higher ranked answers. To quantify this correlation, we used #votes as the ground truth and computed NDCG for the ranking based on #visits. We obtained NDCG = 0.993 for this question, and the average confidence was 0.675, where 1 is the maximum confidence, and 0 is the lowest.

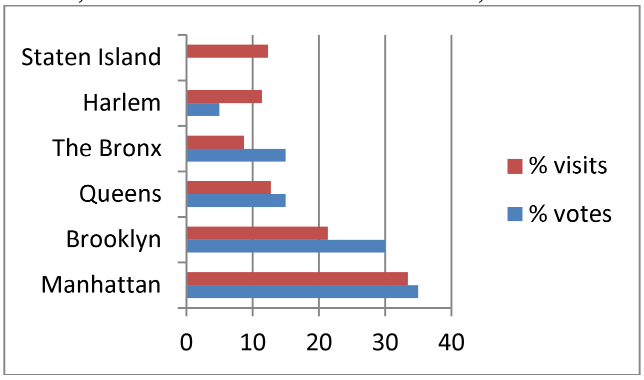


Figure 4 Percentage of votes and visits for the query "List of boroughs of New York City", NDCG = 0.993, average confidence = 0.675.

There are also a few questions where the rankings based on #votes and #visits are not so similar. For instance for the question "Books of Jewish Canon" the NDCG score is only 0.57. However, the average user confidence is also lower in this case (only 0.476). Other questions of this type are "Names of hijackers in the September 11 attacks", "Ratt albums" and "Ancient Greek city-kingdoms of Cyprus". All these questions are relatively specific. We observed in these cases, users indicated relative low confidence, and the agreement between users is also low, suggesting that it was difficult for them to choose the correct answers.

Figure 5 shows the correlations between NDCG scores computed for the ranking based on #visits, confidence and agreement values for each question. By agreement between users we mean the percentage of votes the answer with the highest number of votes has obtained. We can see that in general the ranking based on #votes is quite similar to the ranking based on #visits. More exactly, the average NDCG score is 0.86. For 15 of the 25 queries, the answer with most votes corresponds to the article that is most visited on Wikipedia. Further, we see that the higher the confidence of the users, the higher is also the NDCG based on #visits. Also, NDCG based on #visits correlates with agreement. This means that when users are confident and agree on the results, the ranking based on #visits closely matches the ranking based on #votes.

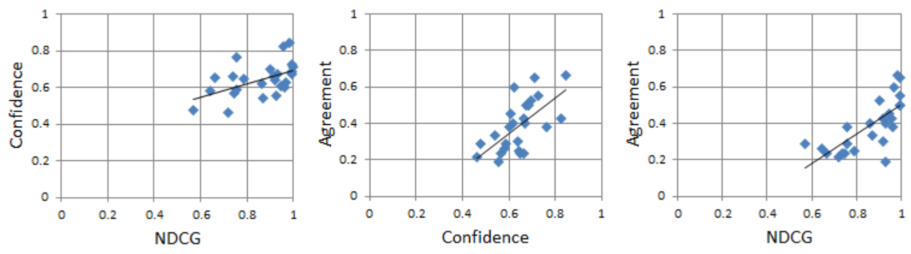


Figure 5 Correlations between confidence, agreement and NDCG values

In conclusion, we have constructed a golden standard for ranking by asking users to vote. The results suggest that for questions where human users can provide correct answers, this golden standard correlates well with the ranking based on #visits. It does not closely match the standard in a few cases, which represent difficult questions for which the user judgments were also not reliable. Thus, #visits can be seen as a good approximation of the ground truth. This has important implications because then, #visits not only can be used as ground truth to evaluate ranking performance, but also as a target feature for training the LTR model.

4.3 Systems

As baselines for evaluating the proposed ranking models, we have implemented two ranking methods described in the related work. The first is ResourceRank (*ResRank*) [10], which provides a global, query-independent PageRank inspired ranking score. The second baseline (*LM*) is based on building language models for the query and results [20]. As discussed, it actually relies on a rather query-independent metric called witness count, which measures the "informativeness" of RDF triples. This count is estimated based on the number of results obtained from searching the Web with the labels of the subject, predicate and object of the triple as queries. Because the number of triples in DBpedia and Yago is large, it was not feasible for us to submit the resulting search requests. For this baseline, we could obtain results only for the smaller SWDF dataset. The last one called *Wikilog* is considered as an upper limit baseline, which rank results based on #visits in the access logs.

We have implemented several LTR systems based on different features and labels (target features). In particular, we used the four different categories discussed before, namely (1) features based on graph *centrality*, (2) features based on *external* sources, (3) features based on the *RDF* dataset, and (4) the *complete* set of all features. Two target features were used, namely #votes (systems using these labels for training are denoted by the prefix '**H_**') and #visits (systems denoted by prefix '**L_**').

4.4 Evaluation of Learning to Rank

The rankings produced by our LTR models are evaluated based on leave-one-out cross-validation. This means the model is trained on the data from all queries except one; then the model is tested to rank the answers of the left-out query. The procedure is repeated for each query. For queries on DBpedia and Yago, the metrics are computed using the ground truth obtained from human evaluators. For queries on SWDF, the ground truth is obtained from access logs. This is because SWDF is too specific such results cannot be reliably evaluated by people who are not domain experts. The results are computed for each query separately, and the average values are summarized in Table 1. (detailed experimental results are also available[5])

WikiLog gives best performance which cannot be surpassed even when training on data with human judgments as target feature (rows with prefix '**H_**'). Moreover, we notice that the performance of models trained using ground truth obtained from logs ('**L_**') is actually higher than the performance of models trained using ground truth obtained from humans. The main reason for this is that many answers get the same number of votes. This holds especially for the answers, which get few votes or no votes at all. Therefore, many pairs of answers in the training data have no or very low confidence, resulting in much fewer valuable training examples (see Section 3.1). A possible solution is to ask human evaluators to do complete orderings instead of votes. Clearly, this results in a complex task that may be not practical for crowd sourcing. In other words, obtaining training data from humans is difficult.

Comparing to the baselines we noticed that the proposed models are comparable in the general domain (DBpedia and Yago). However, for the domain-specific dataset of SWDF, the improvements are more significant. Further, for all systems, the performance in the specific domain case is lower than in the general domain.

External and Complete seem to be better than other for DBpedia and Yago. For SWDF however, External performs badly, and Centrality and RDF are much better. We think the weak performance of External in the specific domain is because specific resources rarely appear in n-grams and search results. Centrality and RDF, being specific to the dataset, perform much better. Complete have stable and good performance in both settings, mainly because it contains more features which compensate for each other. Notably, for SWDF, Complete, which contains the weakly performing external feature set, still comes out as the best.

Looking at individual features we found that features like the number of search results, the number of objects, number of objects @ 2, the number of different incoming predicates @ 2 and the ngram count are among the best features for both DBpedia and Yago achieving NDCG scores of about 0.8

5 Conclusions and Future Work

We have presented a LTR approach for ranking RDF entity search results that considers a multitude of query-independent features. These features are particularly

[5] http://aidemo.ijs.si/paper_supplement/dali_eswc2012_eval.zip

Table 1 Experimental results

	DBpedia		Yago		SWDF	
	NDCG	Spearman	NDCG	Spearman	NDCG	Spearman
WikiLog	0.8602	0.5000	0.8602	0.5000	-	-
ResRank	0.8329	0.2552	0.8206	0.3276	0.6803	0.2287
LM	-	-	-	-	0.7191	0.2548
H_Centrality	0.7837	0.1524	0.8035	0.2751	-	-
H_External	0.8339	**0.3544**	0.8339	**0.3544**	-	-
H_RDF	0.8339	0.3078	0.7832	0.1627	-	-
H_Complete	0.8322	0.2755	0.7999	0.2955	-	-
L_Centrality	0.8294	0.2593	0.8118	0.3055	0.7376	0.2868
L_External	**0.8380**	0.3383	0.8380	0.3383	0.6149	0.1201
L_RDF	0.8076	0.2353	0.8228	0.2239	0.7401	0.3019
L_Complete	**0.8374**	0.2861	**0.8435**	**0.3510**	**0.7533**	**0.3160**

important in this setting where all results are equally relevant with respect to the query. We show that LTR can outperform the baselines, and the improvement is particularly large for the domain specific dataset. We have analyzed the impact of individual features on the LTR performance. The complete combination of features yields high and consistent performance. Surprisingly, good results could also be obtained when only external features derived from the Web are used.

As future work, we will investigate the use of LTR for ranking RDF results in the keyword query setting, which will require both query-independent and query-specific features.

6 Acknowledgements

This work was supported by the Slovenian Research Agency, the ICT Programme of the EC under PASCAL2 (IST-NoE-216886), RENDER (ICT-257790-STREP) and PlanetData (ICT-NoE-257641).

References

1. Edward Cutrell, Zhiwei Guan: What are you looking for?: an eye-tracking study of information usage in web search. CHI 2007, pp. 407-416
2. Tie-Yan Liu. Learning to Rank for Information Retrieval. Foundations and Trends in Information Retrieval. Vol. 3: No 3, pp. 225–331, 2009
3. Roi Blanco, Harry Halpin, Daniel M. Herzig, Peter Mika, Jeffrey Pound, Henry Thompson, Duc Thanh Tran. Repeatable and Reliable Search System Evaluation using Crowd-Sourcing. SIGIR 2011, pp. 923-932, 2011

4. Pablo Castells, Miriam Fernández, David Vallet. An Adaptation of the Vector-Space Model for Ontology-Based Information Retrieval. IEEE Trans. Knowl. Data Eng. 2007, pp. 261-272.
5. Roi Blanco, Peter Mika, Sebastiano Vigna. Effective and Efficient Entity Search in RDF Data, ISWC 2011, pp. 83-97
6. Zaiqing Nie, Yunxiao Ma, Shuming Shi, Ji-Rong Wen, Wei-Ying Ma. Web Object Retrieval. WWW 2007, pp. 81-90
7. Gjergji Kasneci, Shady Elbassuoni, Gerhard Weikum. MING: mining informative entity relationship subgraphs. Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, pp. 1653-1656
8. Page, L., Brin, S., Motowani, R., Winograd, T., The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Libraries, 1998
9. Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. J. ACM Vol. 46, No. 3, pp. 604-632, 1999
10. A. Hogan, A. Harth, S. Decker. ReConRank: A Scalable Ranking Method for Semantic Web Data with Context. SSWS 2006
11. Renaud Delbru, Nickolai Toupikov, Michele Castata, Giovanni Tummarello, Stefan Decker, Hierarchical Link Analysis for Ranking Web Data, ESWC 2010, pp.225-239
12. Glen Jeh, Jennifer Widom, Scaling personalized web search, WWW 2003, pp. 271-279
13. Vagelis Hristidis, Heasoo Hwang, Yannis Papakonstantinou. Authority-Based Keyword Search in Databases, ACM Transactions on database Systems, Vol. 33, No. 1, 2008.
14. Soumen Chakrabarti. Dynamic Personalized Pagerank in Entity-Relation Graphs, WWW 2007, pp. 571-580
15. Zaiqing Nie, Yuanzhi Zhang, Ji-Rong Wen, Wei-Ying Ma. Object-Level ranking: Bringing Order to Web Objects, WWW 2005, pp. 567-574
16. Tran Thanh, Haofen Wang, Sebastian Rudolph, Philipp Cimiano. Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data. ICDE 2009, pp. 405-416
17. Thorsten Joachims, Optimizing search engines using clickthrough data. KDD 2002, pp. 133-142.
18. Knud Moller, Tom Heath, Siegfried Handschuh, John Domingue. Recipes for semantic web dog food -the eswc and iswc metadata projects. In Recipes for Semantic Web Dog Food - The ESWC and ISWC Metadata Projects, ISWC/ASWC 2007, pp. 802-815.
19. Bettina Berendt, Laura Hollink, Vera Hollink, Markus Luczak-Rösch, Knud Möller, David Vallet. USEWOD2011, WWW (Companiaon Volume) 2011, pp. 305-306.
20. Shady Elbassuoni, Maya Ramanath, Ralf Schenkel, Marcin Sydow, Gerhard Weikum. Language-Model-Based Ranking for Queries on RDF-Graphs. CIKM 2009, pp. 977-986.
21. Thorsten Joachims, Making Large-Scale SVM Learning Practical. Advances in Kernel-Methods - Support Vector Learning, B. Scholkopf and C. Burges and A. Smola (ed.), MIT-Press, 1999.
22. Rupnik, J. Stochastic subgradient approach for solving linear support vector machines, SiKDD, 2008.