Learning-to-rank with Prior Knowledge as Global Constraints

Tiziano Papini and **Michelangelo Diligenti** [1]

Abstract. A good ranking function is the core of any Information Retrieval system. The ranking function can be a simple cosine similarity or, more likely for any advanced IR system like a Web search engine, a function processing a high number of signals which typically requires a lot of hand-tuning to deal with the large variability of queries. The result is a sub-optimal and highly complex ranking function which, in spite of having been crafted by human experts, is hard to control and debug. In the last few years, learning-to-rank from examples has emerged as a more flexible approach to design ranking functions. While learning-to-rank approaches have been proved to significantly outperform hand-tuned solutions, they still feature many disadvantages. First, they rely on a large number of training examples to model the high variability of the input query stream. Unfortunately, constructing a training set is more complicated than labeling examples in classical supervised classification tasks. Indeed, the labeling process for learning-to-rank tasks is inherently error-prone and incomplete. Secondly, learning-to-rank schemas usually do not account for the explicit knowledge that human experts have built over the years. It would be nice to integrate this knowledge without having to rely on a large set of examples to infer it. Finally, the proposed approach opens new ways to integrate unlabeled data into the learning process, as the rules must be respected also by the unlabeled data. This paper presents a general framework to convert prior knowledge in form of First Order Logic (FOL) clauses into a set of continuous constrains and shows how these constraints can be integrated into any learning-to-rank approach which is optimized via gradient descent.

1 Introduction

Modern Information Retrieval systems like Web search engines have available hundreds of features to represent each document to answer users' queries. These features range from query-independent signals like PageRank to others measuring the match between the query and a document. Human experts can compose these signals in order to get a ranking function which is effective in most cases. However, because of the high number of correlated signals, it is hard to manually design ranking functions achieving results that are close to optimality. Learning-to-rank from examples has emerged as a more flexible approach to design ranking functions, which has gained popularity both in research and industrial contexts. Learning-to-rank approaches have been proved to significantly outperform hand-tuned solutions [1], but they still feature many disadvantages. First, they rely on a large number of training examples to model the high variability of the input query stream, in a context where the labeling process is inherently error-prone, leading to inconsistent, incomplete and noisy training sets. Secondly, learning-to-rank schemas usually discard the explicit knowledge that human experts have built over several years, starting the optimization process from scratch.

The basic assumption of this paper is that an expert can express the desired behavior of the ranking function via a set of FOL clauses, and it is required to integrate this knowledge without having to rely on a large set of examples to infer it. The training examples will also respect the rules and a supporter of pure learning-to-rank approaches could claim that the rules can be inferred from the examples. However, the number of training examples is often not sufficient to converge to the optimal ranking function and it is therefore more compact and effective to express the ranking rules in an explicit form. Another advantage of this approach is that it allows to get advantage of the unlabeled data into the learning process, as the data can be used to ensure that the constraints are enforced. In a learning-to-rank setting, unlabeled data means the actual ranking provided by the IR system, with no labeling nor goodness judgments assigned by a human expert. Therefore, unlabeled data can be obtained in a virtually unlimited quantity with little cost.

This paper presents a general framework to encode arbitrary prior knowledge expressed as First Order Logic (FOL) into a set of continuous constraints, which can be added to the cost function to optimize. For example, a constraint could force some degree of diversity or freshness in the results, or make the ranking to be determined by a single feature under a specific condition. The presented framework is not tied to a specific learning-to-rank approach, but it can be integrated into any technique, which is optimized via gradient descent. In particular, the experimental section employs a variant of *LambdaRank*, called *LambdaNeuralRank*, to show the effectiveness of the proposed technique. The experimental results show that the approach improves state-of-the-art results on a dataset built from the logs of a commercial search engine when little supervised data is available.

2 Prior work

Learning-to-rank approaches can be grouped into three main classes: pointwise, pairwise and listwise approaches. A pointwise approach [2] takes document/score pairs as training examples to learn a document scoring function $f : \mathcal{D} \Rightarrow I\!R$, which assigns a score to a document, under the assumption that the final rank of the documents will be obtained by sorting the documents in descending order of score. Pairwise methods like [3] take a set of pairs of documents as input to the training. The training process consists in learning a function, which correctly orders the pairs ($f(u) > f(v)$ if u should precede in the rank v). Pairwise approaches are generally preferred to pointwise methods, because they do not impose specific scores to the learning algorithm, leaving it the freedom to select the score range in which to work. Finally, listwise methods [4, 5] get a set of lists of ranked

[1] Dipartimento di Ingegneria dell'Informazione, Università di Siena, Italy, email: {papinit,diligmic}@dii.unisi.it

documents as training examples, and the optimization is performed using a loss function over the entire list of documents.

A recent trend in learning to rank approaches is to attempt a direct optimization of the target metrics [6][7], which typically are either *Mean Average Precision* (MAP) for ranks having two relevance scores or *Normalized Discounted Cumulative Gain* (NDCG) when there is an arbitrary number of relevance scores. This class of approaches falls in the listwise category as the target metrics are functions of ranked lists and not of individual pairs. Those approaches are generally considered to outperform pairwise methods as they can direct the learning toward what's most important with regard to the optimization of the target metric. However, direct optimization of the target metrics is difficult, because all the commonly employed metrics, such as NDCG and MAP, are not expressed in terms of the scoring functions but in terms of the document ranks (which then depend on the functions). This makes the resulting loss function either constant or not differentiable in any point with respect of the training parameters. Most learning approaches solve this issue by employing a continuous approximation of the target metric [8].

3 LambdaRank, LambdaNeuralRank and constraints

The proposed methodology to inject prior knowledge can be integrated into any learning-to-rank approach trained via gradient descent. We start introducing a variant of LambdaRank, a state-of-the-art learning-to-rank algorithm, which will be used as underlying learning mechanism over which FOL knowledge is injected.

Let $C(f, \boldsymbol{d})$ be the following cost function

$$C(f,\boldsymbol{d}) = \underbrace{\sum_{i,j\in\boldsymbol{d}, i>j} L\left(f(d_i), f(d_j)\right)}_{\mathcal{L}(f,\boldsymbol{d})} + \underbrace{\frac{\lambda_r}{2}||\boldsymbol{w}||^2}_{\mathcal{R}(f)} + \underbrace{\sum_{h=1}^{H}\lambda_c^h \Phi_h(f)}_{\mathcal{C}(f)}$$

where $[f(d_1), \cdots, f(d_n)]$ is the vector of scores assigned by f to the set of documents to score $\boldsymbol{d}$, $i > j$ indicates that document d_i should be ranked higher than d_j , $\boldsymbol{w}$ is the vector of parameters of f, λ_r determines the trade-off between fitting the supervised data and the magnitude of the weights, L is a loss function and the H logic clauses forming the prior knowledge are represented as a set of H constraints $\Phi_h(f)$, each assigned a weight λ_c^h. $\mathcal{L}(f,\boldsymbol{d}), \mathcal{R}(f), \mathcal{C}(f)$ indicate the labeled, regularization and constraint contributions to the cost function, respectively.

In particular, L was selected to be a hinge function shifted by a value ϵ which acts as the desired margin, yielding,

$$\begin{aligned} C(f,\boldsymbol{d}) &= \sum_{i,j\in\boldsymbol{d}, i>j} h\left(f(d_i) - f(d_j) - \epsilon\right) + \\ &+ \frac{\lambda_r}{2}||\boldsymbol{w}||^2 + \sum_{h=1}^{H}\lambda_c^h \Phi_h(f)\ . \end{aligned} \tag{1}$$

When $\lambda_c^h = 0, h = 0, \ldots, H$, this cost function is equivalent to the cost function used by SVMRank [9] (in the primal). The derivative of the cost function with respect of a generic weight of f is the following:

$$\begin{aligned} \frac{\partial C(f,\boldsymbol{d})}{\partial w} &= \frac{\partial \mathcal{R}(f)}{\partial w} + \frac{\partial \mathcal{C}(f)}{\partial w} + \frac{\partial \mathcal{L}(f,\boldsymbol{d})}{\partial w} = \\ &= \lambda_r w + \sum_{h=1}^{H}\lambda_c^h \frac{\partial \Phi_h(f)}{\partial w} + \\ &+ \sum_{i,j\in\boldsymbol{d}, i>j} \underbrace{h' \cdot \left(\left.\frac{\partial f}{\partial w}\right|_{d_i} - \left.\frac{\partial f}{\partial w}\right|_{d_j}\right)}_{\lambda_{ij}}, \end{aligned} \tag{2}$$

where λ_{ij} is the contribution to the derivative of a single pair of documents.

LambdaRank [10] provides a learning-to-rank basic mechanism, which acts as a meta-methodology to optimize a target metric. LambdaRank is a meta-approach as it relies on an underlying pairwise algorithm (like the one presented so far) for computing the gradients for each pair. LambdaRank transforms a pairwise approach into a listwise one by weighting the contribution to the gradient of each pair as shown in equation 2 by its utility in improving the target metric. Regardless of the exact formulation of the cost function optimized, LambdaRank has been proved to locally maximize the target metric [11]. This is theoretically justified by observing that it is not needed to explicitly know the cost function we are optimizing, only the gradients are needed.

In particular, let $\theta(\boldsymbol{d}, \mathbf{rnk})$ be the goodness score of rank $\mathbf{rnk}$ over documents $\boldsymbol{d}$ and $\Delta\theta(\boldsymbol{d}, \mathbf{rnk}, i, j)$ be the difference of the θ values coming from swapping document i and j of $\boldsymbol{d}$ in $\mathbf{rnk}$. LambdaRank weights the the contributions λ_{ij} by the impact on the target metric $\Delta\theta(\boldsymbol{d}, \mathbf{rnk}, i, j)$. Therefore, the derivative of the labeled part with respect to a weight w assumes the following form,

$$\frac{\partial \mathcal{L}(f,\boldsymbol{d})}{\partial w} = \sum_{i,j\in\boldsymbol{d}, i>j} \lambda_{ij} \cdot \Delta\theta(\boldsymbol{d}, \mathbf{rnk}, i, j)\ . \tag{3}$$

This value can be directly used to optimize the function by gradient descent together with $\frac{\partial \mathcal{R}(f)}{\partial w}$ and $\frac{\partial \mathcal{C}(f)}{\partial w}$, which we will show in the following section how to compute.

A common choice for the target metric in learning-to-rank environments is the NDCG, defined as,

$$NDCG(\boldsymbol{d}, \mathbf{rnk}) = \sum_{j=1}^{|\boldsymbol{d}|} \frac{2^{scr(d_j)} - 1}{\log_2(rnk(d_j)) + 1}\ .$$

where $scr(d_j)$ and $rnk(d_j)$ are the relevance score and its rank in the result set for document d_j.

It is easy to show that for NDCG, $\Delta NDCG(Bd, \mathbf{rnk}, i, j))$ is equal to,

$$\begin{aligned} \Delta NDCG(\boldsymbol{d}, \mathbf{rnk}, i, j) &= \frac{(2^{scr(d_i)} - 2^{scr(d_j)})}{\log_2(rnk(d_j) + 1)} - \\ &- \frac{(2^{scr(d_j)} - 2^{scr(d_i)})}{\log_2(rnk(d_i) + 1)}\ . \end{aligned} \tag{4}$$

Equation 4 can be used to replace $\Delta\theta(\boldsymbol{d}, \mathbf{rnk}, i, j)$ in equation 3. The function f can be implemented using any machine learning approach. In our experimental setting, f was implemented by a Neural Network as in RankNet [3]. For this reason, we called this learning-to-rank approach *LambdaNeuralRank*.

4 Prior Knowledge expressed as FOL and its conversion

In order to implement learning-from-constraints as described in the previous section, one needs to devise a conversion process to translate logic formalisms into real-valued functions. We focus our attention on knowledge-based descriptions given by first-order logic (FOL–KB). While the framework can be easily extended to arbitrary FOL predicates, in this paper we will consider only unary predicates to keep the notation simple.

Any FOL clause has an equivalent version in *Prenex Normal form* (PNF), that has all the quantifiers $(\forall, \exists)$ and their associated quantified variables at the beginning of the clause. Standard methods exist to convert a generic FOL clause into its corresponding PNF and the conversion can be easily automated. Therefore, without loss of generality, we restrict our attention to FOL clauses in the PNF form. In the following, we indicate by $\mathcal{V} = \{v_1, \ldots, v_N\}$ the set of the variables used in the KB. In particular, v_i can take values over the document collection $\mathcal{D}$ to be ranked or the set of queries $\mathcal{Q}$. Predicates can be of two types. The first type takes a document and returns $true$ if the document/query meets some predefined requirement (like having a feature above a value or its content belongs to some topical category, etc.). This first set of predicates used in the KB is indicated as $\mathcal{P} = \{p_k | p_k : (v \in \mathcal{D}) \rightarrow \{true, false\},\ k = 1, \ldots, |P|\}$. A second class of predicates $\mathcal{R}$ (*ranking predicates*) checks whether a document d is in the first i positions of a rank over a set of n documents $\boldsymbol{d} = \{d_1, \cdots, d_n\}$, as determined by the ranking function f for a query q. Let $\mathcal{R} = \{r_k | r_k : (q \in \mathcal{Q}, v \in \mathcal{D}, I\!R^n) \rightarrow \{true, false\},\ k = 1, \ldots, |R|\}$. The clauses will be built from the set of atoms $p(v) : (p \in \mathcal{P}, v \in \mathcal{D}) \vee (p \in \mathcal{R}, v \in \mathcal{D} \times \mathcal{Q} \times I\!R^n)$. Predicates $\mathcal{P}$ are given and they only select whether a document has some desired characteristic. On the other hand, predicates $\mathcal{R}$ depend on the ranking functions, which determine the rank of a document for a given query.

We assume that all the clauses are in the following general form:

$$\forall q \in \mathcal{Q}\ \ \{\forall, \exists\} v_1\ \ldots\ \{\forall, \exists\} v_n\ \ E(q, p(v_1), \ldots, p(v_n))$$

where $E(q, p(v_1), \ldots, p(v_n))$ is a generic propositional expression over the atoms.

The FOL–KB will contain a set of clauses corresponding to expressions with no free variables (i.e. all the variables appearing in the expression are quantified) that are assumed to be *true* in the considered domain. These clauses can be converted into a set of constraints that can be enforced during the learning process. As detailed in the following sections, the conversion process of a clause into a constraint functional consists of the following steps: *Conversion of the ranking predicates*, *Conversion of the Propositional Expression* and *Quantifier conversion*.

4.1 Conversion of the ranking predicates

A ranking predicate $r_k(\boldsymbol{d}, d, q)$ can be implemented by computing $\boldsymbol{f}$ over $\boldsymbol{d}$ for q, then by sorting the scores and, finally, by checking that the rank condition on the target document is verified. The issue with this ranking predicate is that it is either constant or not differentiable in any point with respect of the parameters of the ranking function f. This makes optimization with respect of the parameters difficult. As commonly done in the context of learning to rank [8], it is possible to approximate the ranking predicate with a continuous and differentiable surrogate approximation of the predicate. This can be done by defining the expected position of d in the rank established by f over the set of document $\boldsymbol{d}$ as:

$$\hat{pos}(\boldsymbol{d}, d, f) = \sum_{i=1}^{|\boldsymbol{d}|} P(d_i > d) = \sum_{i=1}^{|\boldsymbol{d}|} \frac{e^{-\alpha[f(d) - f(d_i)]}}{1 + e^{-\alpha[f(d) - f(d_i)]}}$$

where $P(d_i > d)$ models the probability that d_i precedes d using a sigmoidal function and $\alpha > 0$ is a constant. The predicate $r_k(\boldsymbol{d}, d, q)$ can now be approximated by using a sigmoidal function on top of the estimate of the rank

$$\hat{r_k}(\boldsymbol{d}, d, q) = sigm(k - \hat{pos}(\boldsymbol{d}, d, f))$$

.

4.2 Conversion of the Propositional Expression

Prior work in the literature [12] concentrated on conversion schema based on t-norms. A different approach based on mixtures of Gaussians is exploited in this paper. This approach has been inspired by the methodology described in [13] to integrate symbolic knowledge into neural networks. The advantage of this approach relies in the fact that it does not perform any independence assumption among the variables, like it happens for t-norms.

In particular, let us consider a propositional logic clause involving n logic variables. The logic clause is equivalent to its truth table containing 2^n rows, each one corresponding to a configuration of the variables. The continuous function approximating the clause is based on a set of Gaussian functions, each one centered on a configuration corresponding to the true value in the truth table. The mixture function generalizes the propositional clause into a continuous setting by summing up the single Gaussians:

$$\begin{aligned} &t(x_1, \ldots, x_n) = \\ &= \sum_{[c_1, \ldots, c_n] \in \mathcal{T}} \exp\left(- \frac{||[x_1, \ldots, x_n]' - [c_1, \ldots, c_n]'||^2}{2\sigma^2} \right), \end{aligned} \tag{5}$$

where $x_1, \ldots, x_n$ is the set of variables in the propositional clause and $\mathcal{T}$ is the set of all possible configurations of the input variables which correspond to the true value in the table.

For example, let us consider the clause $x \vee y$, which is verified by the three configurations $[true, true]$, $[true, false]$, and $[false, true]$. The clause is converted as $t(x, y) = \exp\left(-\frac{||[x,y]' - [1,1]'||^2}{2\sigma^2}\right) + \exp\left(-\frac{||[x,y]' - [1,0]'||^2}{2\sigma^2}\right) + \exp\left(-\frac{||[x,y]' - [0,1]'||^2}{2\sigma^2}\right)$.

It is also possible to generalize the clause by setting the default value to true and modeling the false configurations in the truth table:

$$\begin{aligned} &t(x_1, \ldots, x_n) = \\ &= 1 - \sum_{[c_1, \ldots, c_n] \in \mathcal{F}} \exp\left(- \frac{||[x_1, \ldots, x_n]' - [c_1, \ldots, c_n]'||^2}{2\sigma^2} \right), \end{aligned}$$

where $\mathcal{F}$ is the set of all possible configurations of the input variables which correspond to a false value in the table.

If $[x, y]'$ assumes values corresponding to a configuration verifying the clause, $t(x, y) \geq 1$ holds. For a generic input $[x, y]'$, the value of $t(x, y)$ will decrease depending on the distance from the closest configuration verifying the clause. The variance σ^2 is a parameter that can be used to determine how quickly $t(x, y)$ decreases when moving away from a configuration verifying the constraint. Please note that each configuration verifying the constraint is always a global maximum of t.

4.3 Quantifier conversion

The quantified portion of the expression is processed recursively by moving backward from the inner quantifier in the PNF expansion.

Let us consider the universal quantifier first. The universal quantifier expresses the fact that the expression must hold for any result in the set of documents to rank $\boldsymbol{d}$. When considering the real–valued mapping of the original boolean expression, the universal quantifier can be naturally converted measuring the degree of non-satisfaction of the expression over the domain $\boldsymbol{d}$. More generally, let $\boldsymbol{v}_E$ be the vector of variables contained in the expression E, the satisfaction measure can be implemented by computing the overall distance of the penalty associated with E, i.e. $\varphi_E(\boldsymbol{v}_E, f)$ from the constant function equal to 0 over the domain $\boldsymbol{d}$. Using the infinity norm on discrete domains, this measure is

$$\forall v_q \in \boldsymbol{d} \quad E(\boldsymbol{v}_E, \mathcal{P}) \rightarrow \max_{v_q \in \boldsymbol{d}} |\varphi_E(\boldsymbol{v}_E, \mathcal{P})| \,, \tag{6}$$

where the resulting expression depends on all the variables in $\boldsymbol{v}_E$ except v_q, and $\varphi_E(\boldsymbol{v}_E, \mathcal{P}) = max(1 - t_E(\boldsymbol{v}_E, \mathcal{P}), 0)$. Hence, the result of the conversion applied to the expression $E_q(\boldsymbol{v}_{E_q}, \mathcal{P}) = (\forall v_q \in \boldsymbol{d} \quad E(\boldsymbol{v}_E, \mathcal{P}))$ is a functional $\varphi_{E_q}(\boldsymbol{v}_{E_q}, \mathcal{P})$, assuming values in $[0, 1]$ and depending on the set of variables $\boldsymbol{v}_{E_q} = [v_{s(1,E_q)}, \ldots, v_{s(n_{E_q},E_q)}]$, such that $n_{E_q} = n_E - 1$ and $v_{s(j,E_q)} \in \{v_r \in \mathcal{V} | \exists i \;\; v_r = v_{s(i,E)}, \; v_r \neq v_q\}$. The variables in $\boldsymbol{v}_{E_q}$ need to be quantified or assigned a specific value in order to obtain a constraint functional depending only on the functions $\mathcal{P}$.

If we consider the conversion of the PNF representing a FOL constraint without free variables, the variables are recursively quantified until the set of the free variables is empty. In the case of the universal quantifier we apply again the mapping described previously. The existential quantifier can be realized by enforcing the De Morgan law $(\exists v_q \in \boldsymbol{d} \;\; E(\boldsymbol{v}_E, \mathcal{P}) \iff \neg \forall v_q \in \boldsymbol{d} \;\; \neg E(\boldsymbol{v}_E, \mathcal{P}))$ to hold also in the continuous mapped domain. Using the conversion of the universal quantifier of equation (6), we obtain the following conversion for the existential quantifier

$$\exists v_q \in \boldsymbol{d} \;\; E(\boldsymbol{v}_E, \mathcal{P}) \quad \rightarrow \quad \min_{v_q \in \boldsymbol{d}} |\varphi_E(\boldsymbol{v}_E, \mathcal{P})| \,.$$

It is also possible to select a different norm on the discrete domain to convert the universal quantifier. For example, when using the $\|\cdot\|_1$ norm for discrete domains, yields the conversion rule

$$\forall v_q \in \boldsymbol{d} \;\; E(\boldsymbol{v}_E, \mathcal{P}) \rightarrow \frac{1}{|\boldsymbol{d}|} \sum_{v_q \in \boldsymbol{d}} |\varphi_E(\boldsymbol{v}_E, \mathcal{P})| \,.$$

As an example of the conversion procedure, let q be a query and $s(\cdot), n(\cdot)$ two predicates checking whether a result is about topic $sport$ or $nutrition$, respectively. The clause $\forall v \in \boldsymbol{d} \;\; r_k(\boldsymbol{d}, v, q) \Rightarrow s(v) \vee n(v)$ expresses the fact that any result in the first k positions of results for q must be about topic $sport$ or $nutrition$. The only false configuration for the quantifier-free expression is $r_k(\boldsymbol{d}, v, q) = true$, $s(v) = false$, $n(v) = false$. Therefore, $t(q, v) = 1 - \exp\left(-\frac{(r_k(\boldsymbol{d},v,q)-1)^2 + s(v)^2 + n(v)^2}{2\sigma^2}\right)$. Then, converting the quantifier, adding the loss and inserting the continuous approximation of the rank predicate, we get the following constraint:

$$\Phi(f) = \max_{v \in \boldsymbol{d}} \exp\left(-\frac{\left(sigm\left(k - \sum_{i=1}^{|\boldsymbol{d}|} \frac{e^{-\alpha[f(d)-f(d_i)]}}{1+e^{-\alpha[f(d)-f(d_i)]}}\right) - 1\right)^2 + s(v)^2 + n(v)^2}{2\sigma^2}\right) = 0 \,.$$

5 Experimental Results

Unfortunately, publicly available learning-to-rank datasets like LETOR[2] are not suited for testing the proposed method. Indeed, these datasets have been designed for pure learning-to-rank approaches, where only the features are visible. For example, in order to avoid any privacy concern which could follow the public release of sensible data, LETOR does not provide any explicit information about the underlying queries and documents, which are used in the dataset. Furthermore, no additional information like the document content is provided. For all these reasons, it is very hard to introduce any relevant prior knowledge in this limited context.

Therefore, we decided to leverage the AOL dataset, released in 2005, to build a new learning-to-rank benchmark. The AOL dataset has been constructed from the logs of the AOL commercial search engine and it contains a sample of the search activity of 658000 anonymized US-based users over a three month period (March-May 2005), which has been estimated to consist of approximately 1.5% of the overall AOL users in the considered period. The dataset contains 4.8 million queries and 1.8 million URLs. Since we are aware of the privacy concerns of this dataset, the logs have been pruned to remove queries that have been issued less than 30 times and by fewer than four distinct users. This should remove personal queries and documents that could allow associating any anonymous user id to a real person. In this dataset we follow a similar approach to that proposed in [14] by assuming that the relevance of a document for a query is proportional to the number of times the users selected it (click-through-rate), in particular, the click-though-rate ranges in the $[0, 1]$ interval and it has been split into 7 portions of equal size. The first sub-interval is associated to a 0 relevance level (non relevant result), and the relevance level is constantly increased by 0.5 sub-interval by sub-interval up to a maximum relevance level equal to 3 (essential result). Therefore, the learning-to-rank problem consists in predicting the order of the documents in order to maximize the NDCG of the rank.

The dataset has been constructed by randomly selecting 10000 queries issued more than thirty times in the dataset. All the documents that have been selected for the query at least once by a user have been downloaded from the Internet. All the documents that were not available anymore at downloading time have been discarded, resulting into 140740 (query, document pairs). 40% of these queries are kept as candidates for inclusion in a training set. The validation and test sets have been created by randomly splitting the remaining queries into two groups containing 20% and 40% of the initial set, respectively. All the experiments presented in this section have been obtained as an average over 5-folds, obtained by subsampling the pool of reserved training data.

The downloaded HTML documents have been parsed and processed together with their associated query. The output of this process is a vectorial representation of each (query, document) pair composed by 140 features, 5 of which depending on the document only and 135 on the document and query. In particular, the entire document and 4 sections of the document are taken into account: title, body, url and anchor. For each portion, 27 features compute the match between the query and specific sub-portions of the document like the BM25, cosine similarity, etc. Most of these features have been implemented consistently to the LETOR dataset as reported in [1]. Furthermore, using a set of SVM text categorizers [3] built over the data available in the DMOZ taxonomy [3], the documents have

[2] http://research.microsoft.com/users/LETOR/

[3] http://www.dmoz.org

	RankSVM	RankNet	LambdaRank	SortNet	LambdaNeuralRank
NDCG@1	0.462	0.333	0.390	0.450	0.477
NDCG@2	0.563	0.400	0.542	0.568	0.579
NDCG@3	0.603	0.424	0.596	0.615	0.622
NDCG@4	0.628	0.438	0.624	0.639	0.644
NDCG@5	0.647	0.452	0.644	0.658	0.662
NDCG@6	0.663	0.464	0.660	0.672	0.676
NDCG@7	0.675	0.476	0.673	0.685	0.689
NDCG@8	0.686	0.486	0.683	0.696	0.699
NDCG@9	0.696	0.497	0.692	0.704	0.708
NDCG@10	0.705	0.50.7	0.700	0.712	0.716

Table 1: NDCG@n results on the AOL Web logs dataset for the proposed method and other state-of-the-art learning-to-rank approaches.

	Baseline (no constraints)	**Adult**	**Shopping**	**Health**	**Society**	**Sports**	**Recreation**	**Combined**
NDCG@1	0.375	0.425	0.427	0.423	0.398	0.436	0.423	0.435
NDCG@2	0.493	0.544	0.545	0.543	0.523	0.555	0.544	0.554
NDCG@3	0.547	0.600	0.596	0.591	0.573	0.602	0.584	0.598
NDCG@4	0.580	0.625	0.625	0.622	0.604	0.603	0.610	0.620
NDCG@5	0.603	0.645	0.645	0.642	0.624	0.650	0.628	0.640
NDCG@6	0.620	0.661	0.661	0.657	0.639	0.665	0.643	0.655
NDCG@7	0.635	0.672	0.673	0.670	0.653	0.678	0.657	0.668
NDCG@8	0.647	0.685	0.685	0.681	0.664	0.689	0.668	0.679
NDCG@9	0.657	0.694	0.694	0.690	0.674	0.698	0.678	0.689
NDCG@10	0.666	0.702	0.702	0.697	0.684	0.705	0.686	0.697
P@1	0.402	0.451	0.452	0.450	0.424	0.462	0.452	0.463
P@2	0.338	0.368	0.370	0.367	0.355	0.373	0.369	0.377
P@3	0.295	0.315	0.316	0.312	0.304	0.317	0.309	0.316
P@4	0.262	0.276	0.276	0.276	0.268	0.277	0.267	0.274
P@5	0.236	0.247	0.247	0.247	0.240	0.248	0.237	0.246
P@6	0.216	0.226	0.224	0.223	0.218	0.224	0.216	0.222
P@7	0.198	0.203	0.203	0.203	0.200	0.204	0.198	0.203
P@8	0.183	0.187	0.187	0.187	0.185	0.188	0.188	0.187
P@9	0.170	0.174	0.173	0.173	0.172	0.174	0.170	0.174
P@10	0.158	0.162	0.161	0.161	0.160	0.161	0.159	0.162
MAP	0.518	0.555	0.556	0.536	0.561	0.530	0.550	0.561

Table 2: NDCG@n, Precision@n and MAP results on the AOL Web logs dataset as a 5-fold average when using LambdaNeuralRank on 4 supervised and 4000 unsupervised queries per fold and a rule for each single category or all the categories combined. The *Baseline* column represents the results obtained by training without the rules.

been assigned a vector of eight scores in $[-1, 1]$ depending on how much their content belongs to the following categories: *Adult*, *Arts*, *Business*, *Shopping*, *Health*, *Society*, *Sports* and *Recreation*. These topicality scores has also been added to the vector of features. The queries, represented as their bag-of-words, have also been classified into the same categories (non-exclusively, meaning that a query can belong to multiple classes). Let $\mathcal{Q}$ be the set of all queries, we indicate with $\mathcal{Q}_c$ the subset of queries classified as belonging to category c. The ranking functions have been implemented by a 2-layer neural network with 40 hidden neurons with tahn activations. Cross-validation on the validation set has been performed to select the best performing neural network during the training process.

LambdaNeuralRank is a non-trivial extension of LambdaRank. Therefore, even if not the main focus of this paper, we start evaluating its performance before showing how it can be further improved using prior knowledge. In particular, table 1 reports the NDCG scores obtained on this dataset by various learning-to-rank approaches in the pairwise (RankSVM [9], RankNet [15] and SortNet [16]) and listwise category (LambdaRank [10]) against LambdaNeuralRank. This table shows that LambdaNeuralRank over-performs with a significant margin the other approaches for all values of NDCG@i. Therefore, LambdaNeuralRank provides a very solid learning-to-rank package, which is not trivial to improve.

The tested ranking rules where in the following form:

$$\forall q \in \mathcal{Q}_T \;\; \exists d \in \boldsymbol{d} \;\; r_k(q,d) \wedge T(d)$$

where $r_k(q,d)$ is a predicate in $\mathcal{R}$ returning true if d is one of the first k position in the rank for q and $T(d)$ is a predicates returning true if d is about topic T. This rule states that at least one result in the first k should be about a category if also the query is about that category.

In our experimental setting, $k = 3$ and, as said before, all the first level DMOZ categories have been considered.

Table 2 and 3 reports the NDCG and MAP scores on the test set as an average over 5-folds by randomly subsampling 4 and 40 queries as training set for each fold, respectively. The *Baseline* column reports the results provided by LambdaNeuralSort without integrating the logic knowledge. The other columns (excluding the last) show the results of the integration of the logic knowledge for a single class. The weight of the rule in each experiment has been determined via crossvalidation on the validation set. Finally, the *Combined* column reports the results provided by a classifier integrating all the logic knowledge at the same time. The weight of each rule λ_c^h in equation 1 has been set to a value proportional to the gain introduced by the rule on the validation set (more informative rules get higher weights). The benefit of the integration is very significant for both datasets, and it is very large on the experiment using few supervised queries. Please note that the additional logic helps generalization also for queries that do not belong to the category that is directly involved in the rule, for this reason it is possible to substantially increase the NDCG and P scores even when using singe rules. The *Combined* classifier is able

	Baseline (no constraints)	Adult	Shopping	Health	Society	Sports	Recreation	Combined
NDCG@1	0.419	0.426	0.444	0.435	0.416	0.414	0.436	0.454
NDCG@2	0.538	0.540	0.561	0.552	0.537	0.541	0.552	0.566
NDCG@3	0.586	0.589	0.608	0.600	0.585	0.587	0.589	0.608
NDCG@4	0.610	0.615	0.631	0.625	0.611	0.614	0.614	0.631
NDCG@5	0.632	0.635	0.651	0.646	0.629	0.635	0.633	0.649
NDCG@6	0.647	0.651	0.666	0.660	0.646	0.650	0.647	0.662
NDCG@7	0.660	0.664	0.678	0.672	0.659	0.663	0.660	0.676
NDCG@8	0.671	0.675	0.689	0.683	0.671	0.675	0.672	0.687
NDCG@9	0.681	0.684	0.698	0.692	0.680	0.685	0.683	0.696
NDCG@10	0.690	0.692	0.705	0.701	0.689	0.693	0.691	0.704
P@1	0.446	0.451	0.471	0.462	0.443	0.441	0.465	0.483
P@2	0.368	0.367	0.383	0.374	0.365	0.370	0.374	0.385
P@3	0.312	0.315	0.324	0.319	0.312	0.313	0.308	0.323
P@4	0.270	0.272	0.278	0.277	0.272	0.273	0.268	0.278
P@5	0.243	0.246	0.249	0.249	0.242	0.246	0.240	0.247
P@6	0.221	0.223	0.226	0.224	0.220	0.223	0.216	0.222
P@7	0.202	0.204	0.206	0.205	0.201	0.204	0.198	0.203
P@8	0.186	0.188	0.190	0.188	0.186	0.188	0.183	0.188
P@9	0.173	0.174	0.175	0.174	0.173	0.174	0.171	0.174
P@10	0.161	0.162	0.163	0.162	0.161	0.162	0.159	0.161
MAP	0.550	0.554	0.569	0.562	0.550	0.550	0.556	0.571

Table 3: NDCG@n, Precision@n and MAP results on the AOL Web logs dataset as a 5-fold average when training LambdaNeuralRank on 40 supervised and 4000 unsupervised queries per fold and a rule for each single category or all the categories combined. The *Baseline* column represents the results obtained by training without the rules.

to get use of the overall knowledge and it performs the best on most experiments.

6 Conclusions

This paper presents a novel approach to integrate prior knowledge in form of FOL clauses into learning-to-rank approaches. This methodology allows to get advantage of the usually extensive prior knowledge developed by human experts to build a given ranking function, without relying on a very large amount of training data to infer it. The experimental results show that this approach provides a large precision increase of the ranking function when little training data is available. This should allow to apply learning-to-rank approaches also in contexts where it is not economically possible to invest a large amount of time and money to label the training data.

7 Acknowledgments

This work has been partially founded by a research award from Google Inc. and by a grant from the MPS foundation. We thank Marco Gori, Marco Maggini, Nino Freno and Edmondo Trentin for the fruitful discussions that helped us to define this model.

REFERENCES

[1] T. Liu, J. Xu, T. Qin, W. Xiong, and H. Li, "Letor: Benchmark dataset for research on learning to rank for information retrieval," in *Proceedings of SIGIR 2007 workshop on learning to rank for information retrieval*, pp. 3–10, Citeseer, 2007.

[2] K. Crammer and Y. Singer, "Pranking with ranking," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 641–647, MIT Press, 2001.

[3] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *Proceedings of the 22nd international conference on Machine learning*, pp. 89–96, ACM, 2005.

[4] Z. Cao, T. Qin, T. Liu, M. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *Proceedings of the 24th international conference on Machine learning*, pp. 129–136, ACM, 2007.

[5] T. Qin, X. Zhang, M. Tsai, D. Wang, T. Liu, and H. Li, "Query-level loss functions for information retrieval," *Information Processing & Management*, vol. 44, no. 2, pp. 838–855, 2008.

[6] Y. Yue, T. Finley, F. Radlinski, and T. Joachims, "A support vector method for optimizing average precision," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 271–278, ACM, 2007.

[7] M. Volkovs and R. Zemel, "Boltzrank: learning to maximize expected ranking gain," in *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1089–1096, ACM, 2009.

[8] T. Qin, T. Liu, and H. Li, "A general approximation framework for direct optimization of information retrieval measures," *Information retrieval*, vol. 13, no. 4, pp. 375–397, 2010.

[9] Y. Cao, J. Xu, T. Liu, H. Li, Y. Huang, and H. Hon, "Adapting ranking svm to document retrieval," in *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 186–193, ACM, 2006.

[10] C. Burges, R. Ragno, and Q. Le, "Learning to rank with nonsmooth cost functions," *Advances in neural information processing systems*, vol. 19, p. 193, 2007.

[11] P. Donmez, K. Svore, and C. Burges, "On the local optimality of lambdarank," in *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pp. 460–467, ACM, 2009.

[12] M. Diligenti, M. Gori, M. Maggini, and L. Rigutini, "Multitask Kernel-based Learning with Logic Constraints," in *Proceedings of the 19th European Conference on Artificial Intelligence*, pp. 433–438, IOS Press, 2010.

[13] P. Frasconi, M. Gori, M. Maggini, and G. Soda, "Representation of finite state automata in recurrent radial basis function networks," *Machine Learning*, vol. 23, no. 1, pp. 5–32, 1996.

[14] N. Craswell and M. Szummer, "Random walks on the click graph," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 239–246, ACM, 2007.

[15] P. Li, C. Burges, and Q. Wu, "Learning to rank using classification and gradient boosting," in *Proceedings of the International Conference on Advances in Neural Information Processing Systems (NIPS)*, Citeseer, 2007.

[16] L. Rigutini, T. Papini, M. Maggini, and F. Scarselli, "SortNet: learning to rank by a neural-based sorting algorithm," in *In proceedings of the SIGIR 2008 Workshop on Learning to Rank for Information Retrieval (LR4IR)*, vol. 42, pp. 76–79, 2008.