IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 23, 2020, accepted April 6, 2020, date of publication April 9, 2020, date of current version April 27, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2986943

Effective Lightweight Learning-to-Rank Method Using Unified Term Impacts

SHEILA DA N. SILVA [1], **EDLENO S. DE MOURA** [1], **PÁVEL P. CALADO** [2], **AND ALTIGRAN S. DA SILVA** [1]
[1]Institute of Computing, Federal University of Amazonas, Manaus 69067-005, Brazil
[2]INESC-ID, Instituto Superior Tecnico, 1049-001 Lisbon, Portugal

Corresponding author: Sheila da N. Silva (sheila.nobrega@tce.am.gov.br)

This work was supported in part by the eSpot under Grant CNPq 461231/2014-0, in part by the SocSens 1258 under Grant CAPES/PCGI 88887.130299/2017-01, in part by the CARECO under Grant PROCAD/CAPES 1259 88881.068507/2014-01, and in part by CNPq individual grants of Edleno S. de Moura and Altigran S. da Silva.

ABSTRACT In this study, we propose and evaluate a novel learning-to-rank (L2R) approach that produces results on par with those of the state-of-the-art L2R methods while being computationally effective. We start by presenting a modified gradient boosted regression tree algorithm to generate *unified term impact* (UTI) values at indexing time. Each unified term impact replaces several features with a single value in the document index, thereby reducing the effort to compute the document scores at query processing time because the system fetches and processes fewer values. The adoption of UTI values produces competitive ranking results. However, the lack of features available only at query time might lead to accuracy loss. To solve this problem, we propose a hybrid model that uses UTI values with query-dependent features. We demonstrate that our hybrid methods can deliver high-quality results on par with those of the existing state-of-the-art neural ranking models. Our methods can also reduce the computational costs for processing queries, serving as an interesting alternative for L2R practical applications. Our best hybrid, HLambdaMART, achieves an NDCG@10 value of 0.495 using only 36 features at query processing time when applied to the MQ2007 collection, while the best baseline achieves 0.490 using a larger set of features at query processing time. The use of our hybrid framework reduces the time to run LambdaMART to about 35% of the time to run it without using our proposals. In summary, we present a competitive and lightweight alternative L2R approach to be adopted in search systems.

INDEX TERMS Gradient boosting, indexing, LambdaMART, learning-to-rank, search engines.

I. INTRODUCTION

High-quality ranking results are fundamental for web search engines. Users expect answers to their queries displayed on the page and at the top of the list of search engines [1]. Modern search engines experience fast query processing times regardless of the size of the datasets; however, any noticeable increase in waiting time can dampen their perception of the quality of the system, because computational efficiency cannot be obtained at the cost of quality.

Quality is addressed through machine learning techniques known as *learning-to-rank* (L2R) techniques. Fig. 1 shows query processing performed through a two-step L2R-based search engine [2]–[5]. In the first step, top-*k* ranking results are retrieved using a low-cost ranking strategy, such as BM25 [6], referred to as a *base ranker*. The value of *k* depends on the quality of the first ranking, because the goal is to obtain a list that covers most of the potentially relevant documents for the user. The choice of *k* also influences computational costs because a higher *k* yields a higher cost for processing queries [4]. In the second step, adopting an L2R model referred to as a top ranker, the top-*k* documents are reranked using a more sophisticated ranking method. This step accesses an index that contains dozens or more than a hundred features per document, to be combined into a final score. The main goal is to offer the best final result to the user.

While other architectures could be adopted, such as one that involves processing all documents with the L2R method, the two-step approach described in Fig. 1 is mentioned and adopted in the literature as a solution to allow for fast query processing since the top ranker needs to process the features of only a few documents [2]–[5].

The associate editor coordinating the review of this manuscript and approving it for publication was Jenny Mahoney.

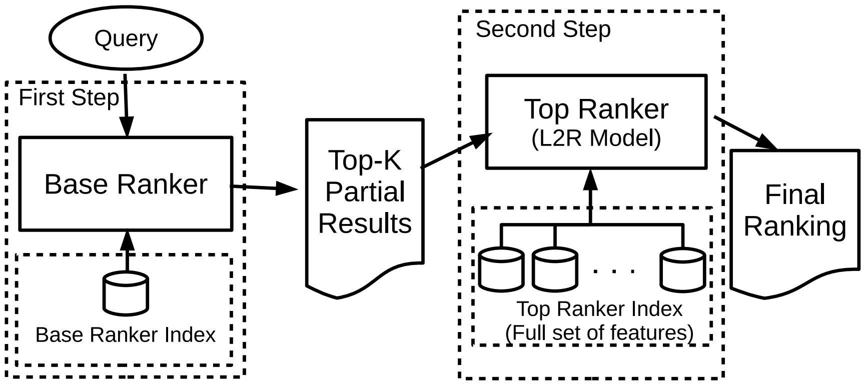


FIGURE 1. Two-stage L2R query processing.

The approach is also useful to create L2R reference collections. For instance, it is adopted in learning to rank (LETOR) collections MQ2007 and MQ2008 [7], where BM25 [6] is adopted as a base ranker. While MQ2007 and MQ2008 were created using GOV2 web page collection, which contains about 25 million documents, these two L2R reference collections provide only the top answers of BM25 as the documents to be processed and ranked by the experimented methods.

Using many distinct sources of *relevance evidence* to build a L2R model is an important aspect of modern search engines. Collectively, these sources are combined to estimate the relevance of a document to a query. Examples of these sources are the frequencies of terms in the text; URLs, titles, and other parts of the document; web link graph analysis; and query log analysis [8]. These features are in the top ranker index in Fig. 1.

In the top ranker, the ranking of the query results is computed, fusing all sources of evidence into a single document score, at the query processing time to produce a final document ranking. In the past few decades, works on evidence fusion have been carried out using L2R implementations [9], such as genetic programming algorithms [10], [11], gradient boosting methods [12], [13], and neural networks [14]–[17]. Thus, L2R methods use example queries and their respective results to train supervised learning models. Then, these models determine the relative position of the documents from the results of a new query and determine the final ranking. However, this approach increases the computational cost of query processing, leading to a drop in query-time performance.

To mitigate this problem, Carvalho *et al.* [18] proposed an alternative that fused evidences at indexing time named *UTI-GP*, based on supervised genetic programming as the underlying learning mechanism. UTI-GP generates a single inverted index that contains unified entries representing all sources of evidence, called *unified term impact* (UTI) values. On the basis of a pretrained L2R model, each UTI value is computed at indexing time. At query time, the search engine obtains the score of each document by adding UTI values that associate it to each of the query terms. Using this approach, several features of the top ranker are substituted with a single feature, i.e., a UTI value. This substitution makes the top ranker extremely simple and lightweight when compared with the traditional L2R strategies.

A limitation of this approach is that several features usually available for search systems are not available at indexing time. For instance, features that depend on the query set, such as the BM25 [6] of the document given a query, are not available at indexing time. Other examples of features that would not be available include any personalized information about the user who is typing the query or information about the most-clicked documents given a query. Thus, UTI-based methods have important limitations.

In the present study, we propose UTI-LambdaMART, an adaptation of LambdaMART [12], [19], a gradient boosting algorithm, to compute UTI values of each pair of document and term during indexing. Compared to LambdaMART, our method uses less resources and reduces the cost of query processing, which is why we call it a lightweight method. Fig. 2 illustrates the process of UTI index generation and how the queries are processed using UTI values. At indexing time (Fig. 2a), the available features are fused into a single index that contains UTI values computed by UTI-LambdaMART. Query processing is performed as depicted in Fig. 2b, where queries are processed in a single step using only the index containing UTI values. A simple ranking method is used, in which the score of a document is the sum of UTI values of each query term. Here, we also address the limitations of UTI-based methods and discuss how to take advantage of UTI-based methods by combining them with traditional L2R methods.

Three main reasons are behind the choice to modify LambdaMART in our research. First, according to experiments presented in a recent survey about this area, LambdaMART represents one of the best L2R methods available in the literature [20]. In addition to the high-quality results produced by LambdaMART, it also has the property of assigning numeric scores to each document given a query, a property that is useful when converting the method to generate UTI values, since they are numeric scores. Finally, we consider it a fast algorithm, both when training a new

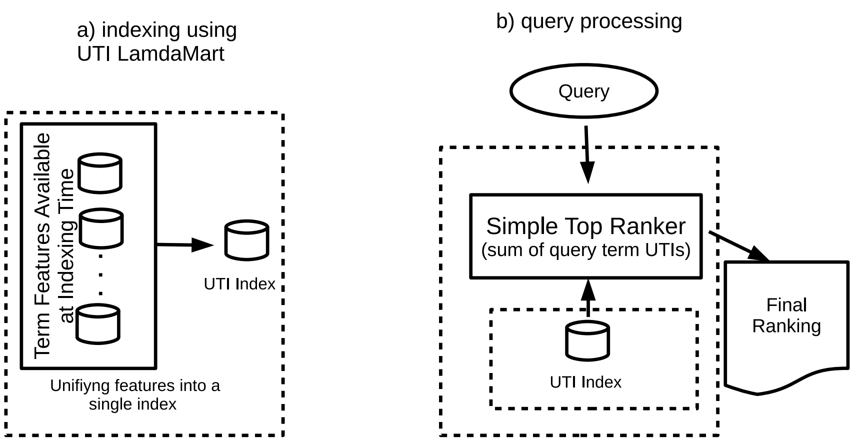


FIGURE 2. **Indexing and query processing when using UTI-LambdaMART.**

L2R model and when processing queries with the generated model.

The methods with performances better than that of LambdaMART in the literature are based on neural networks and can be taken in future work to generate UTI values. We decided to not study this possibility here given that these methods are considered expensive for online query processing, although there are efforts to improve their performance in the literature [21]. Furthermore, as we show in the experiments, our model using LambdaMART is able to achieve a quality of results on par with that of state-of-art neural network models. The great performance of LambdaMART led us to adopt it in our research to produce a high-quality ranking using UTI values.

A. CONTRIBUTIONS AND RESEARCH QUESTIONS

We improve the proposal of Carvalho *et al.* in terms of four aspects: first, we propose a new method for generating UTI values, named UTI-LambdaMART, to significantly reduce the required training compared with UTI-GP while improving the quality of the search results. Second, we demonstrate a method of using our proposed UTI-LambdaMART in a hybrid model and combining it with other L2R methods. The results achieved are on par with those of state-of-the-art neural ranking methods, while still processing fewer features at query processing times. Third, we show that our method offers an extremely low computational effort. The experiments presented indicate it is useful as a practical alternative base ranker for search systems, as it produces a first ranking close to the final result. Finally, using a simple compression scheme, we reduce the space requirements of the inverted index produced, achieving this reduction without significant loss of search quality.

This work addresses four main research questions:

- **RQ1: If we adapt LambdaMART to compute UTI values, would it result in an effective method in terms of the quality of the results?** To address this question, we demonstrate a method of using LambdaMART to produce a term-based learning model and precompute UTI values.
- **RQ2: Because search systems rely on features not available at indexing time, would the combination of methods that compute UTI values with L2R methods at query processing time produce high-quality results?** Here, we investigate how to use UTI computation as one of the steps in a two-stage learning process, proposing a hybrid approach that uses UTI values to substitute, at the query processing time, the set of features available at indexing time.
- **RQ3: Is UTI-LambdaMART a good alternative base ranker?** Since it produces a high-quality ranking and, at the same time, is quite fast, UTI-LambdaMART can be used as a strong base ranker; i.e., it can produce a ranking closer to the final ranking with minimum compromise in terms of the time efficiency. When used as a base ranker, we can reduce the number k of documents analyzed at query processing time. We investigate this hypothesis in the experiments.
- **RQ4: In addition to quality issues, would the use of UTI-LambdaMART produce a fast L2R alternative solution ?** This final question is about performance. The methods proposed here have as one of their goals reducing the query processing times. We thus investigate the possible impact of the methods on the performance of a search system.

The rest of the paper is organized as follows. In Section II, we discuss related works. Section III presents background information about L2R and UTI learning. Section IV presents the proposed method by describing a method of using LambdaMART to generate UTI scores at indexing time. Section V presents our experimental setup, followed by a comparison with current baselines. Section VI concludes the paper and offers directions for future research.

II. RELATED WORK

In this section, we review the related works on precomputed evidence fusion and document rank models.

A. PRECOMPUTED EVIDENCE FUSION

Anh and Moffat [22] were the first to propose precomputed term impacts on documents, which was further addressed in two other followup articles [23], [24]. Their work aimed at reducing the number of arithmetic computations performed at query processing times using a fixed term impact computation strategy not based on machine learning. It is different from our work because they propose an ad hoc method and do not study the use of multiple features, as we and other authors do. Carvalho *et al.* [18] proposed a method, referred to here as UTI-GP, to learn UTI values using genetic programming (GP), where the concept of UTI was first introduced.

Although the authors showed that computing UTI values is a promising strategy, the longer time required to train the evidence fusion model was a major drawback. We hereby propose an adaptation of LambdaMART [12] to compute UTI values.

As a result, we derived a hybrid method to reduce the query time and to achieve quality results on par with the best baselines found in the literature.

B. DOCUMENT RANK MODELS

Several authors have studied L2R methods in the literature using different machine learning techniques [10]–[13], [15]–[17]. Other examples and details of the L2R methods can be found in the works of Liu [9] and Tax *et al.* [25]. All methods mentioned in this session apply the learning at query processing only, with none of them studying the possibility of computing unified term impacts or applying them in the L2R process.

Wei *et al.* [26] propose a novel learning-to-rank model on the basis of the Markov decision process (MDP), referred to as MDPRank. In the learning phase of MDPRank, the construction of a document ranking is considered as a sequential decision-making, each corresponding to an action of selecting a document for the corresponding position. The policy gradient algorithm of REINFORCE is adopted to train the model parameters. The evaluation measures calculated at every ranking position are utilized as the immediate rewards to the corresponding actions, which guide the learning algorithm to adjust the model parameters so that the measure is optimized. The authors compare their methods to several baselines using the collection MQ2007 [7]; the results, however, are quite inferior to those achieved by the current state-of-the-art methods.

Lucchese *et al.* [27] propose a framework called CLEaVER to optimize L2R models based on ensembles of regression trees. Their method first removes a subset of the trees in the ensemble and then fine-tunes the weights of the remaining trees according to a quality measure. Experiments performed on two publicly available datasets show that CLEaVER can prune up to 80% of the trees. In [28], the authors improved the pruning strategies of CLEaVER and proposed the extension model called X-CLEaVER. Again, their method is orthogonal to the ones proposed here. Their pruning strategy can also be applied when fusing evidence at indexing times, which may reduce the computational training costs.

Issues related to the combination of the efficiency and effectiveness in L2R methods have recently been addressed in the literature. Chen *et al.* [29] explored the importance of integrating feature costs into multistage L2R IR systems, optimizing cascaded ranking models that offer a better balance between the efficiency and quality of ranking results. The proposal in this study can be combined with the cascade strategy because the approaches that compute UTI values may be incorporated in any cascade ranking strategy.

Guo *et al.* [20] present a comprehensive survey about L2R methods, comparing more than 20 methods available in the literature, and show that deep neural networks, such as DeepRank [15] and HiNT [16], are among the best L2R methods. They show that HiNT is the method with the best performance when applied to MQ2007, and DeepRank yields results close to it. DeepRank [15] simulates the human judgment process aggregating relevance signals from a query-centric context. HiNT [16] is composed of two stacked components, namely, the *local matching layer* and the *global decision layer*. The local matching layer focuses on producing a set of local relevance signals by modeling the semantic matching between a query and each passage of the document. The global decision layer accumulates local signals and uses them for the final relevance score. The shortcoming of the high computational cost is identified in these methods, which employ special hardware and GPUs to run at a competitive time. Nonetheless, we adopted these methods owing to their high-quality results. On the basis of the experimental results, our hybrid can achieve quality results on par with those achieved by these baselines, which use lightweight L2R models and fewer features at query processing times.

The research in the area is continuously evolving. Yu *et al.* [30] recently presented a study specifically focused on listwise methods, a specific type of L2R method, and proposed the WassRank method. To validate the effectiveness of WassRank, they conduct a series of experiments on two benchmark collections and present results indicating that their method produces results with a quality superior to that of other listwise methods. We plan to investigate the possible combination of their ideas to our UTI approach as future work.

In another recent study, Xu *et al.* [17] show that the quality of document features can affect the effectiveness of ranking models and study methods to better model the relationship between queries and documents. They perform their study using deep neural network models to generate effective features and incorporate autoencoders in the construction of ranking models based on L2R. While the ideas presented show a potential contribution to the area, their final results, however, are not superior to those achieved by HiNT [16]. We consider as a future work using their ideas as a possible alternative to improve the quality of features and produce even better ranking results in our methods. As the quality of results presented by the method is quite low compared to those of HiNT, we decided to not include this method in our baselines.

Gallagher *et al.* [31] present a framework for learning an end-to-end cascade of rankers using backpropagation. They show that learning objectives at each stage can be chained together and optimized jointly to achieve significantly better tradeoffs globally. Their approach is another alternative to improve the quality of results of L2R methods, such as LambdaMART. As in X-CLEaVER, the work of Gallagher is orthogonal to our research presented here.

Ji *et al.* [21] discuss the high computational costs of recently proposed L2R methods based on the neural network and investigate a method for the fast approximation of three interaction-based L2R neural network ranking algorithms using locality sensitive hashing (LSH). Their method accelerates the query-document interaction computation by using a runtime cache with precomputed term vectors and speeds up the kernel calculation by taking advantage of the limited integer similarity values. Zamani *et al.* [32] propose a standalone ranking model (SNRM) using an indexing method for neural representation. While their ideas reduce the computational costs related to L2R neural network, the high computational costs of these methods, compared to other L2R approaches, are still an important issue to be addressed in the literature.

III. BACKGROUND

In this section, we further explain the main differences between learning-to-rank and UTI learning. We also summarize the LambdaMART model.

In the past, ranking models were usually created without learning techniques, adopting a specific formula based on a reasoning regarding how to represent queries, documents and the similarity scores between them. An example of ranking models is the probabilistic model that was used to propose the BM25 [6] score function. BM25 provides a formula to compute the similarity score to estimate how relevant a document is to a given query, and this score function is adopted to compute the ranking results in search systems.

Solutions employing machine learning techniques to automatically construct ranking models have emerged in the literature in the past few decades. Motivations include the increasing complexity of modern search systems, which now use several sources of relevance evidence, also know as features. For instance, systems currently include information on the user clicks in past occurrences of a query, personalized information about the users, such as geographic information, and so on. In this context, the use of machine learning techniques has become almost a standard solution for automatically producing high-quality models in search systems.

A. LEARNING-TO-RANK

We start by defining a supervised learning-to-rank approach in the two-step L2R-based search engine. The L2R task can be divided into a learning system and a ranking system [33]. To explain the learning system, we first need to define the training data used to create a ranking model. Let $Q = \{q^{(1)}, q^{(2)}, q^{(3)},\ldots,q^{(m)}\}$ denotes a set of queries. For each query $q^{(i)} \subset Q$, there is an associated set of k resulting documents $D_i = \{d_1^{(i)}, d_2^{(i)} \ldots, d_k^{(i)}\}$. Each query document pair $(q^{(i)}, d_j^{(i)})$, with $q^{(i)}$ being a query and $d_j^{(i)} \in D_i$ being a document, has an associated feature set $fs(q^{(i)}, d_j^{(i)})$ containing n related features $fs(q^{(i)}, d_j^{(i)}) = \{f1_{(i,j)}, f2_{(i,j)}, \ldots, fn_{(i,j)}\}$ and its respective document relevance judgment $r(i, j)$, representing the relevance score of document $d_j^{(i)}$ as a result for query $q^{(i)}$.

The training data are the input used to learn a ranking model, or ranking function. In some methods, the ranking model assigns scores to each document in the answer set for a query. In these cases, it can be defined as a function $F(x), x = (q^{(i)}, d_j^{(i)}, fs(q^{(i)}, d_j^{(i)}))$, with F being a function that assigns a numerical score to each document given a query, using the feature set associated with them to compute this score. In the ranking system, a rank model learned is used to predict the rank of documents for a new query, given as input documents not seen during the training and their related features.

We may find small variants of this general model. For instance, other methods produce a ranking model that requires the whole set of documents and features as input when processing queries. This is the case, for instance, for SVMRank [34], which compares all pairs of documents in the answer and ranks them according to the number of times a document is better than other documents in the answer. This approach is known as pairwise L2R. In SVMRank, the whole set of results for a query should be analyzed at query processing time to produce the score of each document in the results set since it does not produce a function F that depends only on the pair of a given document and query, always requiring the whole set of results to produce the ranking.

Some pairwise methods perform pairwise comparisons only when learning the model and produce as the result a function that assigns the scores for a given document and query based only on the feature set associated with them, without comparing them to other results to produce this score. For instance, this is the case of LambdaMART, which is also a pairwise method, but one that produces a function F as described above.

1) LambdaMART RANK MODEL

LambdaMART is a combination of MART [35], an L2R method that adopts a boosted tree model, and LambdaRank [36], a neural network model.

LambdaMART constructs a ranking model F that maps each set of features of instance x into a numerical score $F(x)$. Each instance $x = (q, d, fs(d, q))$ is a tuple composed of a query q, a document d and a set of feature values related to document d and query q, $fs(d, q)$. The model constructs a function that maximizes the quality of the ranking in the training set, using information on the relevance level of each document d with respect to query q, $r_{(d,q)}$, to compute the quality of the results. LambdaMART generates a tree ensem-

1	Create an initial base model F_0
2	For each interaction n (of N boosting rounds) construct a regression tree for all query-document instances. (repeat steps 2.1-2.5)
2.1	Update document scores using current model (F_{n-1})
2.2	Compute λ(gradient) and w(derivate of λ) for each query-document instance x. The gradient λ can be seen as the force to define the direction that each document must follow in rank. This force is based in the document relative position given its relevance and score compared with other documents of the query response set. LambdaMART uses the LambdaRank cost function, ΔNDCG
2.3	Fit a next regression tree, with L leaves, to model the lambda for the entire set of instances
2.4	Do a Newton step [12] to optimize gradient prediction in terminal nodes of the tree created in step 2.3
2.5	Update the model F_n
3	Return F_N

FIGURE 3. **General steps for constructing a LambdaMART model for a ranking problem.**

ble model. Mapping occurs by traversing a regression tree, where the direction to be followed (left or right) is defined using the value of each feature. Each leaf $\gamma_{l,n}$ of the tree has a value learned in the training, $1 \leq l \leq L$ and $1 \leq n \leq N$, where N is the number of trees and L is the number of leaves. The mapping is 1:1; i.e., for each instance, it is assigned a score.

Fig. 3 shows the general steps used by LambdaMART to build a rank model. Refer to the original method [12] for further details of all steps. It is important for this study to understand the model inputs (instances used in the learning process): step 2.1, namely, how the scores are updated, and step 2.2, namely, how the gradient is computed, from the original LambdaMART model.

B. LEARNING UTI VALUES

In this work, we study a problem slightly different from the L2R problem. We need to learn a score for each document and term pair present in the collection, named as UTI. It should summarize the importance of the term to the document by fusing, at indexing times, a set of features that relate the term to a document, the feature set associated with the document and term pair. Thus, the main difference relative to other L2R approaches is that we learn a function that assigns a score to a document given a term, instead of assigning a score to a document given a query.

As it is used to produce an index before the queries arrive at the search system, features that are not available at indexing time cannot be used by models that compute UTI values. For instance, at indexing time, it is not possible to know the BM25 score of the query since it is a feature that depends on the set of query terms present in each query, instead of depending on an isolated term. Another example of a feature that would be unavailable at indexing time is the current geographical location of a user when typing a query. This is information that may change each time the user types the query. Thus, UTI models can adopt only a subset of the features usually available in L2R collections.

In addition to the important difference stated above, the process of learning UTI values is quite similar to the process of L2R. Our main goal is to find a score function that maps each term and document pair to a numeric score. After learning this mapping function, it is then applied in the indexing to generate UTI values and store them in the search engine indexing. Note that the generation of UTI values is performed offline at indexing time.

Fig. 4 shows how to learn a UTI model. In the process of learning UTI values, the inputs are query term-by-term instances, whereas each instance x is a set of term-document features (all features of the term and the document available only at indexing time, such as, the term frequency in the body of the document, first occurrence position of the term in the document and document length) and the relevance judgment regarding the whole query.

The algorithm learns the patterns to compute UTI values using the queries in the learning process, as depicted in Fig. 4. The learning process searches for a function that produces UTI values that optimize the quality of ranking results for the training set. The ranking of UTI values to be optimized is always computed by adding the UTI values of all query terms for each document analyzed.

The result of the learning process is a model that maps the provided values of term-document features to a single numerical value, namely, the UTI. The index processing consists of applying the UTI model, taking as input the term-document features; thus, this model can compute UTI for words not seen in the learning process or that do not exist in the document.

IV. THE UTI-LambdaMART MODEL

Our main goal is to construct a UTI model that will be applied during the indexing process for efficient query processing. We used the LambdaMART algorithm [12] to implement our model. To adapt LambdaMART to generate UTI values, we modified it to (i) obtain information about the query terms, instead of information about the entire queries, as input, and (ii) compute the lambda of each individual term-document considering how good the query term is in the final position of the document. The modified algorithm is explained below.

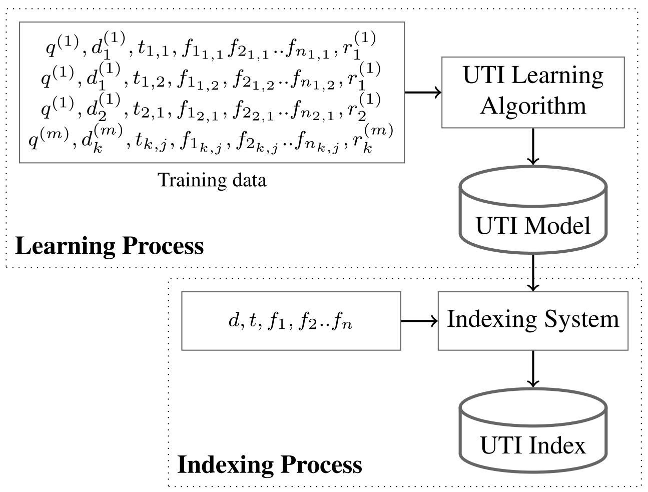

$q^{(1)}, d_1^{(1)}, t_{1,1}, f_{1_{1,1}} f_{2_{1,1}} .. f_{n_{1,1}}, r_1^{(1)}$
$q^{(1)}, d_1^{(1)}, t_{1,2}, f_{1_{1,2}}, f_{2_{1,2}} .. f_{n_{1,2}}, r_1^{(1)}$
$q^{(1)}, d_2^{(1)}, t_{2,1}, f_{1_{2,1}}, f_{2_{2,1}} .. f_{n_{2,1}}, r_2^{(1)}$
$q^{(m)}, d_k^{(m)}, t_{k,j}, f_{1_{k,j}}, f_{2_{k,j}} .. f_{n_{k,j}}, r_k^{(m)}$
Training data
UTI Learning Algorithm
UTI Model
Learning Process
$d, t, f_1, f_2 .. f_n$
Indexing System
UTI Index
Indexing Process

FIGURE 4. The process used to generate a UTI value computation model, the learning process, described in the upper part of the figure, and the indexing process, which takes the model and pairs of terms and documents to produce UTI values.

A. MODIFIED LambdaMART ALGORITHM

UTI-LambdaMART uses gradient boosted decision trees using NDCG@N, N being a constant, as a cost function to learn UTI values given labeled queries. Based on LambdaMART, our approach computes a model Ft that produces UTI scores for each *term* and document pair, instead of a model that computes scores for each query and document pair. In our case, each instance $x = (q, d, fs_{(d,q)})$ is decomposed into $|q|$ training instances in the form $x_t = (t, d, fs_{(d,t)})$), where $t \in q$ and $fs_{(d,t)}$ denote a set of feature values related to the document d and term t. The sum of scores $\sum_{\forall t \in q} Ft(x_t)$ is used in the training to produce rankings and evaluate the quality of the function Ft by using the information on the relevance of the documents, $r_{(d,q)}$. During the training, the document-term impact is computed considering the query-term impact. The sum of UTI values of a query term is taken whenever it is necessary to compute the score of the document given a labeled query. The model Ft is trained and then applied to compute UTI values stored at indexing time, in a process detailed below in Algorithm 1.

Succinctly, UTI-LambdaMART works as follows: the process starts with the model's initialization (lines 1–5). In the iteration to create N trees (lines 6–31), we start by computing a score for each document-query pair ($scores[d, q]$, lines 7–11). This score consists of summing each term's individual UTI, using the model Ft_{n-1}, computed so far by the algorithm. Using the computed scores to sort the documents in descending order, we can estimate the difference in NDCG obtained by switching the order of every pair of documents (d_j, d_k) in a query results list ($\Delta NDCG$, lines 12–23). This is performed when the relevance score for d_j is greater than the relevance score for d_k. $\Delta NDCG$ is then used to compute the λ and w values associated with each term-document-query triple ($\lambda[t, d, q]$ and $w[t, d, q]$, respectively, lines 16–21). Note that since the judgment of relevance is over the entire query, the values calculated here are divided by the number of query terms. Finally, on lines 24–30, the original LambdaMART steps for fitting the regression trees and updating the model are executed, and the model Ft_n is updated. The regression trees map terms and documents features, and the computed UTI value is a linear combination of all created trees. During the learning process, after each interaction, the next regression tree is fitted based on the lambda force, which increases the term impact or reduces its impact. This force considers how good the term is for a given query and the impact of its absence in the document.

The UTI-LambdaMART algorithm generalizes over terms not seen in the query set because the final score obtained is not word-dependent; thus, the term-document feature values define the UTI value. Fig. 5 shows an example using the UTI-LambdaMART algorithm for indexing and query processing. At the end of the interactions (in the illustration, when $n = 3$), the model is generated. In the indexing time, for the document "GX000-14-11495597", given a new term "reheat", the UTI value produced is the linear combination of all regression trees and thus the sum of the terminal nodes reached by its feature values ($f11, f16$, and $f17$). In the query processing, given a query, the document score is the sum of UTI values of each query term.

Fig. 6 presents further practical examples of UTI values produced by our model. Note that UTI values are independent of the query, being unique for each term and document pair. For instance, the terms "food" and "temperature" have the same UTI values when present in queries 8688 and 9513.

A benefit of UTI-LambdaMART, compared to the original LambdaMART is that we can now model individual term features not available in the original model. For instance, we can now include the individual *TF* and *inverse document frequency*(IDF) weight of terms in isolation, whereas in traditional L2R methods, this information exists only as an aggregated value, such as the sum of all *TF* values. Moreover, a lack of information at query processing time, such as BM25 scores or user profiles, may lead to a loss of quality. We performed experiments to examine the final balance between the positive and negative aspects of using UTIs as the only source of information for computing the final ranking. Nevertheless, we also propose a hybrid approach that overcomes the disadvantages of using only UTI values for the ranking.

B. COMBINING UTIs WITH QUERY-TIME FEATURES

We propose two alternative approaches to adopt UTIs in an L2R framework and address the lack of query-time information that occurs when using UTIs. Both alternatives combine UTI values computed using our method with traditional L2R methods.

1) USING UTI VALUES AS A QUERY-TIME FEATURE

The idea is to run the UTI-LambdaMART algorithm to obtain UTI values. In the following, we use a standard L2R approach. However, we substitute all features adopted to generate UTI values by the UTI value itself, complementing

Algorithm 1 Algorithm LambdaMART for Precomputing UTI

```
input  : number of trees $N$, set of training instances $\mathcal{M}$, number of leaves per tree $L$, learning rate $\eta$
output: Model to compute UTI values $Ft$
foreach $(q, d, r_{(d,q)}) \in \mathcal{M}$ do
    foreach $t \in q$ do
        $Ft_0((t, d, fs_{(d,t)})) \leftarrow 0$
    end
end
for $n \leftarrow 1$ to $N$ do
    foreach $(q, d, r_{(d,q)}) \in \mathcal{M}$ do
        $scores[d, q] \leftarrow 0$;                                                          } The score of each query and
        foreach $t \in q$ do $scores[d, q] \leftarrow scores[d, q] + Ft_{n-1}(t, d, fs_{(d,t)})$ ;   } document pair is the sum
                                                                                          } of all term-document UTI
    end
    foreach Query $q \in \mathcal{M}$ and each pair of documents $(d_j, d_k)$ resulting of $q$ do
        if $r_{(d_j,q)} > r_{(d_k,q)}$ then
            $\Delta_\lambda \leftarrow \left(\Delta NDCG(d_j, d_k, q, scores)/(1 + e^{(scores[d_j,q]-scores[d_k,q])}\right)$;
            $\Delta_w \leftarrow \Delta_\lambda \times (1 - (1 + e^{(scores[d_j,q]-scores[d_k,q])}))$ ;
            foreach $t \in q$ do
                $\lambda[t, d_j, q] \leftarrow \lambda[t, d_j, q] + \Delta_\lambda/|q|$;   }
                $\lambda[t, d_k, q] \leftarrow \lambda[t, d_k, q] - \Delta_\lambda/|q|$;   } Assign $\lambda$ and $w$ for
                $w[t, d_j, q] \leftarrow w[t, d_j, q] + \Delta_w/|q|$;                     } each term-document
                $w[t, d_k, q] \leftarrow w[t, d_k, q] + \Delta_w/|q|$;                     }
            end
        end
    end
    Create $L$ leaf tree $\{R_{ln}\}_{l=1}^{L}$ on $\{\forall(q, d, r_{(d,q)}) \in \mathcal{M}$, and $\forall t \in q|(t, d, fs_{(d,t)}), \lambda[t, d, q]\}$;
    Assign leaf values to $\{R_{ln}\}_{l=1}^{L}$ using the calculated $\lambda$ and $w$ based on Newton step;
    foreach $x = (q, d, r_{(d,q)}) \in \mathcal{M}$ do
        foreach $t \in q$ do
            Update the model $Ft_n(t, d, fs_{(d,t)})$ using parameter $\eta$ and $\{R_{ln}\}_{l=1}^{L}$
        end
    end
end
```

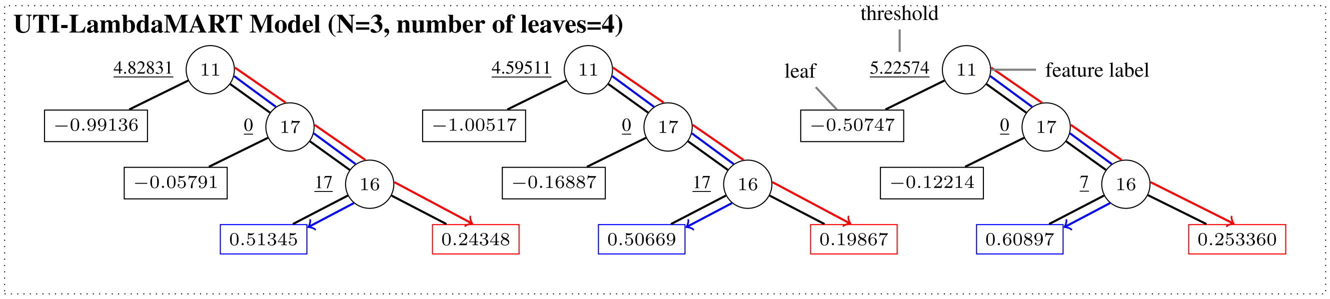
it with the remaining features available at query processing time. As a result, the number of features fetched and processed at query time becomes smaller. The amount of reduction depends on the number of features at indexing time and on the project decision regarding features encoded in UTI values. We show examples in the experimental section.

Fig. 7 shows the complete query processing steps when using our proposed hybrid approach, highlighting the differences from Fig. 1 in blue. The first step is similar to that in any traditional L2R approach—a simple base ranker is applied to obtain a list of k potentially relevant documents. In the second step, UTI value, learned as described in Fig. 2b, is taken as one of the features used by the L2R top ranker as a substitute for the features adopted to generate the UTI values. UTI values of query terms are summed up and combined with the remaining query-dependent features. Note that with this change, we reduce the number of features to be processed and fetched by the hybrid approach compared to the approach described in Fig. 1.

UTI values can encode information not available to traditional L2R methods because they learn weights for the individual term entries, whereas traditional methods address only aggregate query-level features. As a consequence, our hybrid approach can improve the overall quality of results. Using two L2R collections, we performed experiments to validate this hypothesis in the experimental section.

2) USING UTI-LambdaMART AS A BASE RANKER

Another method for creating a hybrid approach is to take advantage of UTI values in the first step of the query processing base ranker. Base rankers are simple methods with a low computational cost for computing query results [3]. When processing queries with UTI values, the final score for each document is computed as a simple sum of UTI values for



Indexing

term	doc GX000-14-11495597				doc GX160-11-2318949			
	f11	f16	f17	UTI	f11	f16	f17	UTI
"temperature"	6.04263	210	227	0.69551	5.09375	18	56	-0.06532
"reheat"	6.04263	828	877	0.69551	5.09375	17	55	0.51267
"food"	6.04263	7	9	1.62911	5.09375	0	0	-0.73425

Query Processing

qid:8688 "temperature for reheat food"

Document	score
GX000-14-11495597	3.02013
GX160-11-2318949	-0.2869

FIGURE 5. An example of the UTI-LambdaMART model output for indexing and query processing.

	docID	term	UTI
qid:8688 "temperature for reheat food"	GX000-14-11495597	temperature reheat food	0.072725927 0.090611666 0.895362371
qid:9513 "cook food left at room temperature"	GX000-14-11495597	cook food left room temperature	0.280159877 0.895362371 0.005127802 -0.081957972 0.072725927

FIGURE 6. Examples of UTI values of terms present in two different queries. Note that given a term and a document, we always take the same UTI value, regardless of what the other query terms are.

the documents in the inverted lists of their query terms. This lowers the computational effort, common in L2R methods and even traditional IR methods, such as BM25. Thus, our UTI-based approach fits perfectly as a base ranker.

We illustrate this idea in Fig. 8 The differences from the architecture described in Fig. 1 are highlighted in blue and consist of (i) changing the base ranker to use UTI values and (ii) using the hybrid approach described in Fig. 7. The goal here is not so much to yield improvements in the ranking quality but rather to reduce the number of top-k documents retrieved in the first step of the query processing, thereby reducing the overall cost of producing the final ranking.

V. EXPERIMENTS

We now present our experimental setup, followed by an experimental evaluation of the proposed approaches.

A. DATASETS

For our experiments, we used two LETOR 4.0 benchmark datasets [7]: Million Query Track 2007(MQ2007) and Million Query Track 2008(MQ2008). The LETOR collection was extracted from the GOV2 collection, a document collection containing about 25 million web pages. The three main reasons for choosing these two datasets are as follows. First, they are largely adopted in other L2R-related works [15]–[17], [30]. Second, their meta feature files are available. Thus, it is possible to extract individual features and relate terms of documents, which is information that is required for UTI-LambdaMART.

Third, the document content is available for this collection, allowing for the extraction of features from the document, which are useful in the experiments with our method and also are required by the best baselines we found [15].

LETOR was created by using BM25 [6] as the base ranker to select an initial set of documents that can be good answers for each query. Then, L2R methods reorder this initial set of ranked documents by applying the entire set of features available to generate the final ranking [37]. Every query-document pair in the MQ2007 and MQ2008 datasets is represented by a 46-feature vector that maps the query to documents. All the features of LETOR are numeric values. Recent studies [16] introduced 9 extra features related to positional information of the term queries in passages of the documents. We adopt this expanded set of features in our experiments. All of these 9 extra features are also numeric values.

Following a procedure adopted in previous works, because the number of terms in MQ2008 is small for training, we merged MQ2007 and MQ2008 when processing MQ2008 such that our training sets became larger. This procedure was adopted by recent research articles [15], [16] to increase the training set when using neural networks. We repeated the same procedure for a fair comparison of the methods. The validation and test sets remained unchanged. In total, there are 1,692 queries, 2,727 terms, 69,623 query-document pairs, and 236,774 term-document-

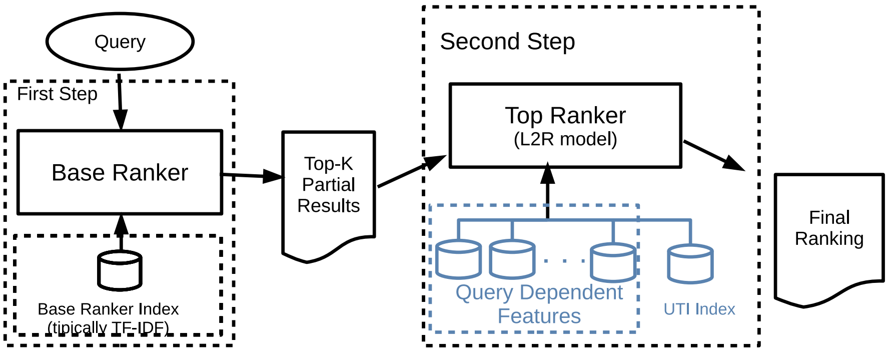


FIGURE 7. Query processing using a hybrid approach that adopts UTI values to substitute all features available at indexing times. The L2R model combines UTI values with the remaining features. Differences from the architecture described in Fig. 1 are highlighted in blue.

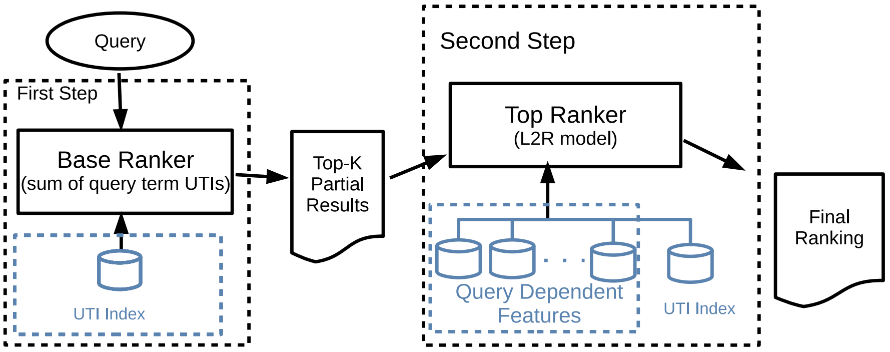


FIGURE 8. Query processing using the hybrid approach and adopting a UTI both as a feature of the L2R process and as a base ranker. Differences from the architecture described in Fig. 1 are highlighted in blue.

query triples in MQ2007. MQ2008, after the merger, contains 2,477 queries, 2,909 terms, 84,834 query-document pairs, and 300,442 term-document-query triples.

The LETOR collections were originally created by just providing feature values for the top-ranked documents to each query. We here are interested in adding new term features and are also interested in performing experiments to evaluate the indexing and query processing times when applying the experimented L2R methods to these collections. To do so, we extracted the content of all mentioned documents in the collections MQ2007 and MQ2008. As a result of this process, the total number of documents in MQ2007 is 65,216, with a size of 935.2MB of plain text, and the total number of documents in MQ2008 is 78,593 documents, with a total of 1100MB of plain text. The total number of posting lists to represent the frequency of the query terms in the documents is 7,454,889 in MQ2007 and 9,145,089 in MQ2008.

Table 1 reports a complete list of the features available. Some of these features are extracted from the *TF*, *IDF*, and $TF \times IDF$ values of the terms in the document. These features naturally map terms of documents and can be used in UTI-LambdaMART. In the original LETOR, the sum of the values for the query terms maps queries to documents. For instance, LETOR represents as a feature the sum of the *TF* values of query terms instead of each individual *TF* value. The document length (DL, expressed as the number of terms) is also computed as a query-independent feature. Each of these values is computed as a single value for all query terms from different areas in the document — the body of the text, the anchor text, the title, the URL, and the entire document — thereby generating a total of 20 features. Six other features are derived from the link structure and URL — PageRank, InLink Count, OutLink Count, Number of Slashes in the URL, Length of the URL, and Number of Child Pages.

Finally, the remaining original LETOR features are the similarity scores between the documents and queries. Although features such as the *TF* can be used to map terms of documents, these query similarity features are nonusable by UTI methods because they represent maps between queries and documents. These features include the BM25 score and three variations of language model-based (LMIR) functions, all applied to the same five different areas of the document. Although we can decompose these values into term to document scores, they are still intrinsically related to queries. This study [7] provides a detailed description of the LETOR features.

In addition to the 46 original LETOR features, we have also adopted the nine passage-based features described by the authors of DeepRank [15]. They divide documents into smaller portions, named *passages*, and calculate

TABLE 1. Original features (from 1 to 46) of MQ2007 and MQ2008, plus a set of nine features related to the positions of query terms, used in the baseline L2R methods.

Query-Document Features	
(01-05)	Sum of TFs (term frequencies) of the body, anchor, title, URL and whole document.
(06-10)	Sum of IDFs (inverse document frequencies) of body, anchor, title, URL and whole document.
(11-15)	Sum of TF*IDFs of the body, anchor, title, URL and the whole document.
(16-20)	DL (document length) of the body, anchor, title, URL and whole document.
(21-25)	BM25 of the body, anchor, title, URL and whole of document.
(26-30)	LMIR.ABS of the body, anchor, title, URL and whole of document.
(31-35)	LMIR.DIR of the body, anchor, title, URL and whole of document.
(36-40)	LMIR.JM of the body, anchor, title, URL and whole of document.
(41)	PageRank
(42)	Inlink number
(43)	Outlink number
(44)	Number of slashes in the URL
(45)	Length of the URL
(46)	Number of child page
(47-55)	Maximum, minimum and average passage-based features taken from TF*IDF, BM25, and LM

the TF-IDF, BM25 and language model (LM) scores for each query-passage pair, picking the maximum, minimum, and average scores across passages as the nine new features for the L2R process [15].

When examining the list of features in Table 1, LETOR features represent maps between queries and documents, assigning one feature value for each document and query pair. We divide these features into three distinct groups. First, some of these features are more related to terms than to queries. We refer to these as *term-related features*. This is the case for the *TF* and *IDF* values, which are individual properties of terms. In this first group, we include features 1-15 of LETOR, features more related to the queries, such as BM25 scores or LMIR scores, named *query-related* or *query-dependent* features. We include in the second group features from 21-40. The third group of features is composed of document properties that do not depend on the query or the query terms. This is the case for features 41-46. The nine passage-based features are also divided into query-related and term-related groups.

B. BASELINE METHODS

We work with three types of baseline methods for comparison: UTI methods, L2R methods and hybrid methods.

1) UTI METHODS

The baseline adopted to compare with our UTI-LambdaMART is the *UTI-GP* method, which applies genetic programming to produce UTI values at indexing times. This method was proposed by [18] and was included for completeness in the experiments.

The authors provided the UTI-GP implementation; the methodology and parameter settings adopted follow those proposed in their article. More specifically, we used 40 generations, a population size of 1000, a tree depth of 17, a tournament size of 6, a crossover rate of 0.85, a mutation rate of 0.05 and a reproduction rate of 0.10 to perform 10 runs with distinct random seeds.

When learning to produce single UTI values for each term, we are interested in the term-related features. To perform this task, we use the related features as values associated with pairs of terms and documents instead of queries and documents. Thus, we produced the required information term by term to use in the learning process of UTI-LambdaMART and UTI-GP. The passage-based extra features of LETOR are also query-dependent. To replace them as term-document maps, we included information about the first and second positions of each term in each document. These two features represent the positional information available to generate UTI values and were defined after we observed that even varying the number of positions from 2 to 10 resulted in no change in the quality.

The features related to documents can be easily adopted by associating them with terms or queries. In the experiment, we investigated their use at both indexing and query processing times. For instance, PageRank [38] can be encoded as a feature when computing UTI values and as a feature available to the top ranker when processing queries.

Table 2 summarizes the term-document features that we adopted. Features 16-17 were adopted to represent the positions of terms in documents, playing the same role as features 47-55 presented in Table 1. These two features are extracted from the original documents from the GOV2 collection. We also investigated if the first term occurrence position could improve other L2R methods, adding it as an additional feature. However, we concluded that it would not have a significant impact on the final performance of the other L2R methods when passage-based information was adopted.

When producing UTI values with UTI-LambdaMART and UTI-GP, we adopted features 1-17 mentioned in Table 2 and

TABLE 2. MQ2007 and MQ2008 term-related versions of features 1-15 available in MQ2007 and MQ2008, i.e. features mapping terms to documents.

Term-Document Features	
(01-05)	TFs (term frequencies) of the body, anchor, title, URL and whole document
(06-10)	IDFs (inverse document frequencies) of the body, anchor, title, URL and whole document
(11-15)	TF*IDFs of the body, anchor, title, URL and whole document
(16-17)	First and second occurrence position of a term in the document; zero if it does not occur

combined them with the remaining query-independent features available at indexing time. We included features 16-20 in Table 1, producing a total of 22 features available to compute UTI values. These features are exploited only at indexing time. The stop-words were removed using the INQUERY stop-words list [39] before extracting these features. It contains 429 words.

2) L2R METHODS

We chose methods from studies that have achieved the best scores in the existing L2R methods to serve as baselines, namely, *DeepRank* [15] and the hierarchical neural matching model (*HiNT*) [16], the best baselines found in the literature that uses neural network learning methods [20]; the listwise linear feature-based model coordinate ascent (*CA*) [40]; and the original *LambdaMART* [12]. We use QuickRank [3] to implement LambdaMART and CA. CA was trained using 21 samples, a window size of 10, and a reduction factor of 25. All LambdaMART-based models, including our own UTI-LambdaMART, were trained using 100 trees ($N = 100$), with a learning rate of 0.1 ($\eta = 0.1$) and number of leaves of 10 ($L = 10$). During training, we optimized the average NDCG with a cutoff of 10 results.

For HiNT and DeepRank, we collected the results reported by the authors using the MQ2007 and MQ2008 collections (described in the following section). In addition to LETOR (MQ2007 and MQ2008) features, HiNT and DeepRank also extract other features, such as text passages, from the GOV2 collection (the collection used to create LETOR). The authors make these features available for a fair comparison with other methods. We also adopt these extra features in our experiments.

The traditional L2R methods adopted in LambdaMART and CA were tested using the full expanded set of 55 LETOR features mentioned in 1. The deep matching models DeepRank and HiNT adopt passage-level information during learning. DeepRank obtained better experimental results with resources automatically learned from the classification texts and all handcrafted resources (46 standard features in LETOR). The set of features adopted by DeepRank and HiNT is similar to the set adopted by the baselines, although their model does not have an explicit set of features [16].

3) HYBRID METHODS

The experiments combining methods that compute UTI values with query-dependent features were conducted using

TABLE 3. Number of features adopted by each method at indexing times and at query processing times. The HiNT and DeepRank methods learn from the raw text inputs (features equivalent to the 55 adopted by the other methods).

Method	Indexing	Query Processing
UTI-LambdaMART	22	-
UTI-GP	22	-
Hybrid methods (HCA and HLambdaMART)	-	36
L2R Baselines (LambdaMART and CA)	-	55
Neural Baselines (DeepRank and HiNT)	-	raw text

CA and LambdaMART, referred to as HCA and HLambdaMART. When implementing our hybrid approach, we first produced UTI values using features 1-17 in Table2 and features 16-20 in Table 1. The six document features related to LETOR features (41-46 in Table 1) could be applied both at indexing time to produce UTI values and at query processing time. From the observation, they performed better at query processing, despite being excluded when computing UTI values in the hybrid approach. As a consequence, in addition to UTI values, we have features 21-55 in Table 1 at query processing times of the hybrid approach. Thus, in our hybrid approach, 22 features are adopted at indexing times and 36 at query processing times. This combination produced high-quality ranking results.

Table 3 summarizes the features adopted by each method, describing whether the features are adopted at indexing or at query processing time. All methods were compared fairly with the information allotted to them being the same, respecting the restrictions and settings of each method. We also performed an embedded feature selection, taking the subset that optimized the results in training [41] for each of the tested methods. Thus, we selected the best result for each method and each feature set tested.

4) EVALUATION MEASURES

To evaluate the methods, we adopted MAP, P@N and NDCG@N. These three evaluation metrics [42] are commonly adopted when evaluating the quality of results produced by search systems [8]. Their values vary from 0, the worst possible result, to 1, the best possible result. We further explain the metrics bellow:

MAP is the mean average precision. It is a metric that is useful to compare the overall ranking provided by the systems. The MAP of a set of query results QR, with each element $qr_i \in QR$ being a list of ranked documents provided as a result of a system for a specific query q_i in the set of queries Q, is defined as follows:

$$MAP(QR) = \frac{\sum_{i=1}^{|QR|} AvPel(qr_i)}{|QR|} \tag{1}$$

where $AvPel(qr_i)$ is the average precision of query result qr_i. When examining the list of results for a system, whenever we find a relevant document, we say that we improve the recall, and may compute the precision by dividing the number of relevant documents found so far by the number of examined results so far. We compute the precision for all positions in qr_i at which we find a relevant answer. We assign a precision of zero for recall levels not reached by the system. For instance, a system that gives 3 of 7 relevant results in a list of results will have 3 non-zero precision values and 4 zero precision values. The average precision takes the average precision at all recall levels. It is important to say that $AvPel(qr_i)$ is not defined if q_i has no relevant answer, since there are zero possible recall levels in this case. For these situations, most of the authors assign average precision of zero. We follow this strategy here.

$P@N$ is the average value of precision achieved by the system when considering the precision when inspecting exactly N results for each query. The precision for a specific query when inspecting N results is the number of relevant results found at the top N results divided by N. $P@N$ presented in our results is the average for all queries. It is usually adopted in experiments related to web search, being useful to compare the quality of the systems at the top of the ranking.

$NDCG@N$ is the average of normalized discounted cumulative gain computed for all queries at the top N results of the ranking. For each query, $NDCG@N$ is computed as

$$NDCG@N(QR, REL) = \frac{\sum_{i=1}^{|QR|} \frac{DCG@N(qr_i)}{IDCG@N(rel_i)}}{|QR|} \tag{2}$$

where QR is a set of result lists, $qr_i \in QR$ is the list of results of the system for query i and is sorted by in decreasing order of relevance score, REL is also a set of lists, and $rel_i \in REL$ is the list of relevant results for query i sorted in decreasing order of score.

$DCG@N(qr_i)$ is the discounted cumulative gain achieved by the system for the results of query i (qr_i) and is computed as

$$DCG@N(qr_i) = \sum_{j=1}^{N} \frac{2^{relScore(qr_i,j)} - 1}{\log_2(j+1)} \tag{3}$$

where qr_i represents the list of query results provided by the system to the query i, j is the j-th position of the list and $relScore(qr_i, j)$ is the relevance score of the j-th element of the list qr_i. In the MQ2007 and MQ2008 collections, the relevance score varies from 0 (non-relevant) to 1 (relevant).

The $IDCG(rel_i)$ is the ideal discounted cumulative gain, and is computed as

$$IDCG@N(rel_i) = \sum_{j=1}^{N} \frac{2^{relScore(rel_i,j)} - 1}{\log_2(j+1)} \tag{4}$$

where rel_i is the list of relevant results for query i, sorted in decreasing order of scores, j is the j-th position of the list and $relScore(rel_i, j)$ is the relevance score of the j-th element of the list. NDCG is a metric broadly adopted when comparing ranking results of search systems. It gives better values for systems that provide relevant results closer to the top of the ranking.

We conducted a two-sided paired t-test for statistical significance tests, with a p-value ≤ 0.05. The t-test is the most adopted and studied statistical test when comparing rankings of search results [43]–[45].

C. RESULTS

The experimental results are reported according to the research questions (RQs) presented by us.

1) RQ1: IF WE ADAPT LambdaMART TO COMPUTE UTI VALUES, WOULD IT RESULT IN AN EFFECTIVE METHOD IN TERMS OF THE QUALITY OF THE RESULTS ?

The first question references the possibility of modifying LambdaMART to produce a competitive model to generate UTI values, applying the learning process at indexing time to learn weights associated with terms instead of rank queries. Table 4 reports the results of the MQ2007 and MQ2008 datasets. Interestingly, when comparing the UTI-LambdaMART results to those of the original LambdaMART, the results of UTI-LambdaMART are quite close to those of the original method. A lower computational cost when ranking at the query processing time is another benefit of this version. A further investigation reveals that UTI-LambdaMART takes advantage of the more fine-grained features owing to a lack of query-dependent features. Moreover, LETOR contains a small set of features available at the query processing time. Thus, information such as clicks and other personalized user information could be adopted to offer more advantages to the original method when compared to the UTI-LambdaMART version.

Another interesting observation is the quality of results achieved by UTI-LambdaMART, being superior to that achieved by UTI-GP. The differences in the results of the two methods are statistically significant in terms of all metrics reported. In addition to the quality of results, it is also important to report the time required to train the models in both UTI alternatives. The time required for the training of UTI-LambdaMART is far less than the time required for the training of UTI-GP. In our experiments, the UTI-GP model took approximately three training hours [18], whereas UTI-LambdaMART took only 5 minutes, just about 2.8% of the training time required for UTI-GP.

Regarding the questions of how to adapt LambdaMART to produce UTI values, we need to address the problem of generating UTI values that produce an index that can be compressed, since compression is a common practice in search engines, being applied to improve the time and space efficiency [8]. A compressed index will take less storage space and require less I/O operations, generally yielding

TABLE 4. Performance of UTI-LambdaMART compared to that of UTI-GP (the previous UTI method available in the literature) and to that of the original LambdaMART. UTI methods apply the learning process at indexing time and use a limited set of features, while the original LambdaMART method performs the learning at query processing time and uses the full set of features available in the collection.

MQ2007							
Model	**NDCG@1**	**NDCG@5**	**NDCG@10**	**P@1**	**P@5**	**P@10**	**MAP**
UTI-GP	0.402▽	0.415▽	0.442▽	0.468▽	0.418▽	0.384▽	0.462▽
LambdaMART	0.424	0.432	0.459	0.486	0.427	0.392	0.477
UTI-LambdaMART	0.429	0.437	0.465	0.499	0.435	0.414	0.481
MQ2008							
Model	**NDCG@1**	**NDCG@5**	**NDCG@10**	**P@1**	**P@5**	**P@10**	**MAP**
UTI-GP	0.383▽	0,472▽	0.228▽	0.449▽	0.345▽	0.246▽	0.473▽
LambdaMART	0.413	0.491	0.237	0.464	0.356	0.251	0.490
UTI-LambdaMART	0.397	0.489	0.237	0.464	0.352	0.249	0.486

TABLE 5. Impact on NDCG@10 of UTI-LambdaMART and HLambdaMART when varying the number of decimal places per entry when computing UTI values on MQ2007.

	Decimal Places	**Bits per entry**	**CR**	**UTI-LambdaMart** (NDCG@10)	**HLambdaMart** (NDCG@10)
	0	2.68	92%	0.450	0.488
UTI	**1**	**6.59**	**79%**	**0.465**	**0.494**
	2	12.70	60%	0.466	0.496
	3	19.18	40%	0.465	0.492
	All	32.00	-	0.465	0.495

a gain in performance. In our proposal, we exploited UTI-LambdaMART to achieve a much smaller index with low impact in terms of its construction time. Our approach still allows for the complementary application of standard compression methods, yielding even greater gains.

Our proposal for index size reduction consists of truncating the decimal places when computing UTI values. We investigate whether it is most profitable to reduce the size of the UTI values representation during or after the training. During the training, in Algorithm 1, every time the function Ft_n is evaluated, only the first α decimal places are considered. A smaller value for α means fewer bytes required for storage per UTI value. When the reduction occurs after the training, we simply truncate the final UTI score. The final result analysis of the quality shows that the second approach is superior with not only lower quality loss but also lower implementation cost.

Table 5 reports the compression rates achieved with the number of digits truncated in when computing UTI values. We report only the results for MQ2007, because the conclusions and results for MQ2008 are similar. We also compressed the truncated UTI values using the commonly used technique of *Elias Delta coding* [46]. The results are expressed as the number of bits per entry. In reference to the amount of compression, each original UTI value (i.e., nontruncated) would be represented as a 32-bit floating point number; thus, a 3-bit representation is equivalent to a 90% compression rate.

From the results, a small tradeoff between quality and compression rate is observed as we vary the number of truncated decimal point values. Nevertheless, the results of truncating UTI values to only one decimal digit yield virtually no reduction in NDCG@10, thereby achieving a compression rate of approximately 79%. When truncating to zero decimal digits, we observed a clear negative impact on the NDCG@10 values, indicating that the best tradeoff was achieved when setting the system to use just 1 decimal place. In conclusion, we may say that the experiments with compression indicate the possibility of reducing the average number of bits required to store each UTI value from 18 to 6 bits, without significant loss in the quality of the results.

2) RQ2: WOULD THE COMBINATION OF UTI METHODS AND L2R METHODS AT QUERY PROCESSING TIME PRODUCE HIGH-QUALITY RESULTS ?

As previously discussed, UTI models are not able to handle features that are available only at query processing times. This limitation raises questions about how to take advantage of them in practical systems. One of the solutions is to use UTI values in the hybrid architecture proposed in Section IV-B. We here perform experiments to evaluate the quality of results provided by this combination.

When considering the results of the methods presented in Tables 4 and 6, we can see that when comparing UTI-LambdaMART to the best baselines found in the literature, both DeepRank and HiNT produce better results, superior of both of the collections and in terms of all metrics considered. However, Table 6 shows that our hybrid HLambdaMART produces scores results superior to or on par with those of the best baselines, according to the metric and collection tested. HLambdaMART was superior to HiNT

TABLE 6. Performances of CA,LambdaMART, DeepRank, HiNT and the hybrid methods, combining UTI-LambdaMART with CA (HCA) and LambdaMART (HLambdaMART). A significant performance degradation of HLambdaMART is denoted as (▽).

MQ2007							
Model	**NDCG@1**	**NDCG@5**	**NDCG@10**	**P@1**	**P@5**	**P@10**	**MAP**
CA	0.401▽	0.417▽	0.441▽	0.451▽	0.412▽	0.375▽	0.463▽
LambdaMART	0.424▽	0.432▽	0.459▽	0.486▽	0.427▽	0.392▽	0.477▽
DeepRank [15]	0.441▽	0.457	0.482	0.508▽	0.452	0.412	0.497
HiNT [16]	0.447▽	0.463	0.490	0.515▽	**0.461**	**0.418**	0.502
HCA	0.466	**0.467**	0.492	0.536	0.458	0.415	0.501
HLambdaMART	**0.469**	0.466	**0.495**	**0.538**	0.456	0.416	**0.503**
MQ2008							
Model	**NDCG@1**	**NDCG@5**	**NDCG@10**	**P@1**	**P@5**	**P@10**	**MAP**
CA	0.395▽	0.477▽	0.229▽	0.452▽	0.348▽	0.248▽	0.478▽
LambdaMART	0.413	0.491▽	0.237▽	0.464▽	0.356▽	0.251▽	0.490▽
DeepRank [15]	0.406	0.496	0.240	0.482	0.359	0.252	0.498
HiNT [16]	0.415	0.501	0.244	**0.491**	**0.367**	**0.255**	0.505
HCA	**0.416**	0.505	0.246	0.485	0.364	0.253	**0.506**
HLambdaMART	0.414	**0.506**	**0.247**	0.483	0.361	0.253	0.503

in NDCG@1 and P@1 on MQ2007, with statistically significant differences in the results and improvements of 4.9% and 4.5%, respectively. For the remaining comparisons, there were no statistically significant differences. For instance, on MQ2007, the NDCG@10 for HLambdaMART was 0.495, whereas it was 0.490 for HiNT. Differences between HiNT and HLambdaMART were not statistically significant, except for NDCG@1 and P@1 on MQ2007, where HLambdaMART outperforms HiNT.

For MQ2007 and MQ2008, the hybrid models outperform their respective traditional models LambdaMART and CA, which indicates the superiority of our hybrid approach. For instance, on MQ2007, the NDCG@10 for HLambdaMART was 7.8% higher than the one achieved by LambdaMART, and the NDCG@10 for HCA was 11.6% higher than the one achieved by CA. In both cases, the improvements are statistically significant.

3) RQ3: IS UTI-LambdaMART METHOD A GOOD ALTERNATIVE BASE RANKER ?

UTI-LambdaMART is useful for compiling a large set of initial features and producing a first cut ranking result to be used at query processing times by other L2R methods, which may be included in our HLambdaMART. In these cases, UTI-LambdaMART is used as a base ranker. We performed experiments to verify the impact of this alternative to compare with BM25, the base ranker adopted to create the MQ2007 and MQ2008 datasets. Fig. 9 presents the results of MQ2007. We do not present the results of MQ2008 because the conclusions and results are similar to those of MQ2007.

In Fig. 9, UTI-LambdaMART outperforms BM25 as a base ranker. For example, from the top-10 results to be processed, UTI-LambdaMART achieves a higher NDCG@10 score than that of BM25 for the top-20 results. This result shows that UTI-LambdaMART is effective as a base ranker in web

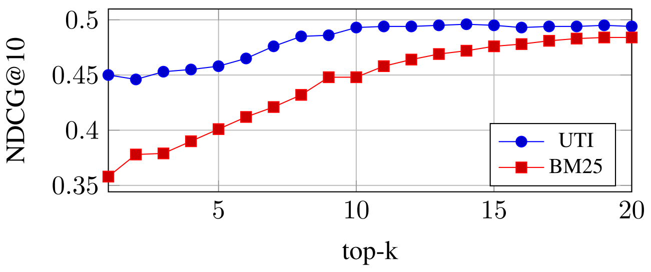


FIGURE 9. Quality of results achieved by HLambdaMART when using UTI-LambdaMART and BM25 as base rankers of MQ2007 for distinct sizes of the top-*k* results transferred from the base ranker to HLambdaMART.

search, since we can use it to obtain the same quality by processing fewer results in the top ranker. The combined advantages of our method are discussed in the next experiment.

4) RQ4: IN ADDITION TO QUALITY ISSUES, WOULD THE USE OF UTI-LambdaMART PRODUCE A FAST L2R ALTERNATIVE SOLUTION ?

To make it easier to understand the advantages and disadvantages of adopting HLambdaMART, we compare the time performance of our method to that of the original LambdaMART method, comparing a system that adopts the architecture presented in Fig. 1 to our proposal presented in Fig. 8, thus using UTI values to produce the base ranker and our proposed hybrid method that combines UTI values with features not available at indexing times. The time performance experiment presented here was executed using a machine with an Intel Core i5-4200U 1.6-GHz CPU, and 8 GB of memory. The time performance experiment described here adopts the indexing and searching systems provided by Daoud et al [4], using BMW as the query processing algorithm.

We start by comparing the time and space required by each method when indexing the collections. For LambdaMART,

TABLE 7. LambdaMART and HLambdaMART time (seconds) to index and space (MB) when applied to MQ2007 and MQ2008.

	MQ2007		MQ2008	
Method	Time	Space	Time	Space
LambdaMART	72	180.1	87	213.4
HLambdaMART	83	229.7	103	272.0

TABLE 8. LambdaMART and HLambdaMART average times, in milliseconds, to run base ranker (base) and top ranker (top) on MQ2007 and MQ2008.

	MQ2007		MQ2008	
Method	base	top	base	top
LambdaMART	0.024	0.037	0.032	0.018
HLambdaMART	0.015	0.014	0.021	0.006

we need to store the frequency of the term in the collection, which is used by the base ranker and by the top ranker, and the maximum, minimum and average frequencies of each term in the document passages in the collection, information required to compute passage-based features from 47-55 of Table 1. Features from 16-20 and 41 to 46 require a smaller storage space that is proportional to the number of indexed documents. The remaining features can either be computed at query processing time or have a storage space proportional to the vocabulary of the collection, such as the IDF values in features 6-10 of Table 1.

For HLambdaMART, it is necessary to add the index of UTI values, which has a number of entries equal to the number of frequency posting lists in the collection. Table 7 presents the space required by the index of MQ2007 and MQ2008. The HLambdaMART required an index that is approximately 27% greater both in MQ2007 and MQ2008. This is the additional cost of using this method. Regarding the time for indexing, HLambdaMART required 15% extra time when indexing MQ2007 and approximately 18% extra time for indexing MQ2008. UTI index is generated by taking the indexes already built to process queries, which reduces the overhead to create it.

Table 8 presents the average time for processing queries in the base ranker and top ranker in both collections. When looking to the results at the top ranker, we see that the time needed to run HLambdaMART was approximately 38% the time required by LambdaMART on MQ2007 and 35% of the time on MQ2008. The performance gain is a natural consequence of (i) the reduction in the number of documents that are inspected by the system, since our approach using UTI method as the base ranker reduces the number of documents inspected to half the number inspected when using BM25, and (ii) the reduction in the number of features processed since HLambdaMART fetches and processes only 36 features, while the original LambdaMART method processes 55 features. We thus note that the input size given to HLambdaMART was approximately 32% the input given to LambdaMART, which explains the gain in performance. This experiment illustrates the potential gain in time performance at query processing time achieved when using our proposed ideas, results that are achieved with a gain in the quality of results compared to those of the original LambdaMART.

When looking to the times achieved by the base ranker, HLambdaMART was 38% faster for MQ2007 and 37% faster for MQ2008. The base ranker of HLambdaMART is faster because it: i) computes a simple ranking function that adds only UTI values, whereas BM25 requires math operations to compute the ranking, and ii) it requires a smaller number of top results to produce the final ranking, as shown in Fig. 9.

A comparison of our method to the baselines HiNT and DeepRank other methods based on neural networks recently proposed in the literature, would be not only a difficult task but also unfair. First, these two methods use the raw text and data from the collection as the input to the L2R process at query processing time; this simple procedure slows down the performance of the methods compared to methods that take explicitly precomputed feature values, as discussed recently by Ji *et al.* [21]. Adding the raw text to the L2R process opens the possibility of improving the quality of results, with the two methods being among the state-of-the-art L2R methods when considering the quality of results, but it also makes the methods expensive, since the neural network adopted by them needs to process the raw data at query processing time. A second important factor is that neural network methods require specialized hardware to run, which makes a time comparison difficult to perform.

If we compare the methods using the same hardware, without specialized hardware with GPUs, the neural network methods would become extremely slow, since these methods take advantage of massive parallel operations and use the parallelism of GPUs to accelerate the query processing. From these observations, we limit ourselves and say only that LambdaMART is known to be a fast L2R method and requires less computational resources than current methods based on neural networks, such as HiNT and DeepRank. By obtaining a quality of results similar to that of these methods, we conclude that our proposal is a competitive alternative solution for L2R.

VI. CONCLUSION AND FUTURE WORK

In this work, we propose UTI-LambdaMART, a modified LambdaMART ranking algorithm designed to generate UTI scores for each term and document pair at indexing time. We also propose hybrid methods that combine UTI-LambdaMART with other L2R methods, producing new L2R methods named HCA and HLambdaMART. Our experimental results show that HCA and HLambdaMART produce results on par with those of the state-of-the-art L2R method HiNT, which adopts a neural network model. Furthermore, our architecture resulted in a reduction in the time needed to process queries, which is an important property for real-world search systems.

As a future work, we plan to investigate a combination of HiNT and our hybrid approach. The idea is to study a hybrid approach where HiNT would be used as a top ranker. We also intend to investigate the relationships between the different rankings as an additional feature in our hybrid approach. In this case, the final ranking would be computed by combining results produced by a set of other ranking methods.

As another important future direction, we plan to combine our proposal with other optimization methods related to L2R, which can further improve either the quality of the results or the performance of the L2R methods. For instance, we plan to explore the possible combination of the methods proposed here with the cascade methods proposed in the literature [28], [29], [31].

We also plan to study the effect of methods developed to produce more stable rankings, namely, risk-sensitive L2R methods, and study their impact when applied to UTI methods. Risk-sensitivity is a subarea of L2R that tries to learn models that are good on average while at the same time reducing the risk of performing poorly in a few but important queries (e.g., medical or legal queries) [5]. Their usage when combined with UTI values represents a challenge, since when computing UTI values, the learning is performed at indexing times.

ACKNOWLEDGMENT

The authors would like thank the authors of the HiNT and DeepRank methods for providing their datasets and experimental results to allow for a fair comparison between their work and their proposals.

REFERENCES

[1] P. C. Saraiva, E. S. de Moura, N. Ziviani, W. Meira, R. Fonseca, and B. Riberio-Neto, "Rank-preserving two-level caching for scalable search engines," in *Proc. 24th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. (SIGIR)*, 2001, pp. 51–58.

[2] V. Dang, M. Bendersky, and W. B. Croft, "Two-stage learning to rank for information retrieval," in *Advances in Information Retrieval*. Berlin, Germany: Springer, 2013, pp. 423–434.

[3] G. Capannini, C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, and N. Tonellotto, "Quality versus efficiency in document scoring with learning-to-rank models," *Inf. Process. Manage.*, vol. 52, no. 6, pp. 1161–1177, Nov. 2016.

[4] C. M. Daoud, E. S. de Moura, D. Fernandes, A. S. da Silva, C. Rossi, and A. Carvalho, "Waves: A fast multi-tier top-k query processing algorithm," *Inf. Retr. J.*, vol. 20, no. 3, pp. 292–316, Jun. 2017.

[5] D. X. Sousa, S. Canuto, M. A. Gonçalves, T. C. Rosa, and W. S. Martins, "Risk-sensitive learning to rank with evolutionary multi-objective feature selection," *ACM Trans. Inf. Syst.*, vol. 37, no. 2, pp. 24:1–24:34, Mar. 2019.

[6] S. E. Robertson and S. Walker, "Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval," in *Proc. 17th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. (SIGIR)*. New York, NY, USA: Springer-Verlag, 1994, pp. 232–241.

[7] T.-Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li, "Letor: Benchmark dataset for research on learning to rank for information retrieval," in *Proc. SIGIR Workshop Learn. Rank Inf. Retr.*, 2007, pp. 3–10.

[8] R. Baeza-Yates and B. Ribeiro-Neto, *Introduction to Information Retrieval*, 2nd ed. Boston, MA, USA: Addison-Wesley, 2011.

[9] T.-Y. Liu, "Learning to rank for information retrieval," *Found. Trends Inf. Retr.*, vol. 3, no. 3, pp. 225–331, 2009.

[10] H. M. de Almeida, M. A. Gonçalves, M. Cristo, and P. Calado, "A combined component approach for finding collection-adapted ranking functions based on genetic programming," in *Proc. 30th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. (SIGIR)*, 2007, pp. 399–406.

[11] T. P. C. Silva, E. S. de Moura, J. M. B. Cavalcanti, A. S. da Silva, M. G. de Carvalho, and M. A. Gonçalves, "An evolutionary approach for combining different sources of evidence in search engines," *Inf. Syst.*, vol. 34, no. 2, pp. 276–289, Apr. 2009.

[12] Q. Wu, C. J. Burges, K. M. Svore, and J. Gao, "Adapting boosting for information retrieval measures," *Inf. Retr.*, vol. 13, no. 3, pp. 254–270, 2010.

[13] C. Lucchese, F. M. Nardini, R. Perego, S. Orlando, and S. Trani, "Selective gradient boosting for effective learning to rank," in *Proc. 41st Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. (SIGIR)*, 2018, pp. 155–164.

[14] B. Mitra, F. Diaz, and N. Craswell, "Learning to match using local and distributed representations of text for Web search," in *Proc. 26th Int. World Wide Web Conf. (WWW)*, Geneva, Switzerland, 2017, pp. 1291–1299.

[15] L. Pang, Y. Lan, J. Guo, J. Xu, J. Xu, and X. Cheng, "DeepRank: A new deep architecture for relevance ranking in information retrieval," in *Proc. 26th ACM Int. Conf. Inf. Knowl. Manage. (CIKM)*, 2017, pp. 257–266.

[16] Y. Fan, J. Guo, Y. Lan, J. Xu, C. Zhai, and X. Cheng, "Modeling diverse relevance patterns in ad-hoc retrieval," in *Proc. 41st Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. (SIGIR)*, 2018, pp. 375–384.

[17] B. Xu, H. Lin, Y. Lin, and K. Xu, "Incorporating query constraints for autoencoder enhanced ranking," *Neurocomputing*, vol. 356, pp. 142–150, Sep. 2019.

[18] A. L. C. Carvalho, C. Rossi, E. S. Moura, A. S. Silva, and D. Fernandes, "LePrEF: Learn to precompute evidence fusion for efficient query evaluation," *J. Amer. Soc. Inf. Sci. Technol.*, vol. 63, no. 7, pp. 1383–1397, Jul. 2012.

[19] C. J. C. Burges, "From RankNet to LambdaRank to LambdaMART: An overview," Microsoft Res., Bengaluru, India, Tech. Rep. MSR-TR-2010-82, 2010. [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/MSR-TR-2010-82.pdf

[20] J. Guo, Y. Fan, L. Pang, L. Yang, Q. Ai, H. Zamani, C. Wu, W. B. Croft, and X. Cheng, "A deep look into neural ranking models for information retrieval," *Inf. Process. Manage.*, early access, Jul. 9, 2019, doi: 10.1016/j.ipm.2019.102067.

[21] S. Ji, J. Shao, and T. Yang, "Efficient interaction-based neural ranking with locality sensitive hashing," in *Proc. World Wide Web Conf. (WWW)*, New York, NY, USA, 2019, pp. 2858–2864.

[22] V. N. Anh and A. Moffat, "Impact transformation: Effective and efficient Web retrieval," in *Proc. 25th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. (SIGIR)*, New York, NY, USA, 2002, pp. 3–10.

[23] V. N. Anh and A. Moffat, "Simplified similarity scoring using term ranks," in *Proc. 28th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. (SIGIR)*, New York, NY, USA, 2005, pp. 226–233.

[24] V. N. Anh, R. Wan, and A. Moffat, "Term impacts as normalized term frequencies for bm25 similarity scoring," in *Proc. Int. Symp. String Process. Inf. Retr.* Berlin, Germany: Springer, 2008, pp. 51–62.

[25] N. Tax, S. Bockting, and D. Hiemstra, "A cross-benchmark comparison of 87 learning to rank methods," *Inf. Process. Manage.*, vol. 51, no. 6, pp. 757–772, Nov. 2015.

[26] Z. Wei, J. Xu, Y. Lan, J. Guo, and X. Cheng, "Reinforcement learning to rank with Markov decision process," in *Proc. 40th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. (SIGIR)*, New York, NY, USA, 2017, pp. 945–948.

[27] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, F. Silvestri, and S. Trani, "Post-learning optimization of tree ensembles for efficient ranking," in *Proc. 39th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. (SIGIR)*, New York, NY, USA, 2016, pp. 949–952.

[28] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, F. Silvestri, and S. Trani, "X-CLEaVER: Learning ranking ensembles by growing and pruning trees," *ACM Trans. Intell. Syst. Technol.*, vol. 9, no. 6, 2018, Art. no. 62.

[29] R.-C. Chen, L. Gallagher, R. Blanco, and J. S. Culpepper, "Efficient cost-aware cascade ranking in multi-stage retrieval," in *Proc. 40th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. (SIGIR)*, New York, NY, USA, 2017, pp. 445–454.

[30] H.-T. Yu, A. Jatowt, H. Joho, J. M. Jose, X. Yang, and L. Chen, "WassRank: Listwise document ranking using optimal transport theory," in *Proc. 12th ACM Int. Conf. Data Mining (WSDM)*, 2019, pp. 24–32.

[31] L. Gallagher, R.-C. Chen, R. Blanco, and J. S. Culpepper, "Joint optimization of cascade ranking models," in *Proc. 12th ACM Int. Conf. Web Search Data Mining (WSDM)*, New York, NY, USA, Jan. 2019, pp. 15–23.

[32] H. Zamani, M. Dehghani, W. B. Croft, E. Learned-Miller, and J. Kamps, "From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing," in *Proc. 27th ACM Int. Conf. Inf. Knowl. Manage. (CIKM)*, New York, NY, USA, 2018, pp. 497–506.
[33] H. Li, *Learning to Rank for Information Retrieval and Natural Language Processing*, vol. 1. San Rafael, CA, USA: Morgan & Claypool, 2011.
[34] T. Joachims, "Optimizing search engines using clickthrough data," in *Proc. 8th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2002, pp. 133–142.
[35] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, 2001.
[36] C. Burges, R. Ragno, and Q. V. Le, "Learning to rank with nonsmooth cost functions," in *Advances in Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 2006, pp. 392–402.
[37] B. B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt, "Early exit optimizations for additive machine learned ranking systems," in *Proc. 3rd ACM Int. Conf. Web Search Data Mining (WSDM)*, 2010, pp. 411–420.
[38] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the Web," Stanford InfoLab, Stanford, CA, USA, Tech. Rep. SIDL-WP-1999-0120, 1999.
[39] J. P. Callan, W. B. Croft, and J. Broglio, "TREC and TIPSTER experiments with inquery," *Inf. Process. Manage.*, vol. 31, no. 3, pp. 327–343, May 1995.
[40] D. Metzler and W. B. Croft, "Linear feature-based models for information retrieval," *Inf. Retr.*, vol. 10, no. 3, pp. 257–274, Jun. 2007.
[41] H.-J. Lai, Y. Pan, Y. Tang, and R. Yu, "FSMRank: Feature selection algorithm for learning to rank," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 6, pp. 940–952, Jun. 2013.
[42] S. Robertson, "Evaluation in information retrieval," in *European Summer School on Information Retrieval*. Berlin, Germany: Springer, 2000, pp. 81–92.
[43] G. V. Cormack and T. R. Lynam, "Validity and power of t-test for comparing MAP and GMAP," in *Proc. 30th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. (SIGIR)*, 2007, pp. 753–754.
[44] M. D. Smucker, J. Allan, and B. Carterette, "A comparison of statistical significance tests for information retrieval evaluation," in *Proc. 16th ACM Conf. Conf. Inf. Knowl. Manage. (CIKM)*, 2007, pp. 623–632.
[45] M. D. Smucker, J. Allan, and B. Carterette, "Agreement among statistical significance tests for information retrieval evaluation at varying sample sizes," in *Proc. 32nd Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. (SIGIR)*. New York, NY, USA: Association for Computing Machinery, 2009, pp. 630–631, doi: 10.1145/1571941.1572050.
[46] P. Elias, "Universal codeword sets and representations of the integers," *IEEE Trans. Inf. Theory*, vol. IT-21, no. 2, pp. 194–203, Mar. 1975.

SHEILA DA N. SILVA received the B.S. and M.S. degrees in computer science from the Federal University of Amazonas (UFAM), Amazonas, Brazil, in 1996 and 2007, respectively, where she is currently pursuing the Ph.D. degree. Her research interests include information retrieval, machine learning, and recommendation systems.

EDLENO S. DE MOURA received the Ph.D. degree in computer science from the Federal University of Minas Gerais (UFMG), Brazil, in 1999. He is currently a Full Professor of computer science with the Federal University of Amazonas (UFAM), Brazil, where he heads the Database and Information Retrieval Group. He is the author of more than 100 research articles covering topics related to information retrieval, such as efficiency issues in information retrieval, text indexing, ranking algorithms, text classification, text compression, and image search. He has also been a member of the program committee of important conferences in his area, such as ACM SIGIR, ACM CIKM, and WWW Conference.

PÁVEL P. CALADO received the degree in computer engineering from the Instituto Superior Técnico, University of Lisbon, and the M.S. degree in computer science and the Ph.D. degree from the Federal University of Minas Gerais (UFMG), in 2000 and 2004, respectively. During this time, he was awarded two Best Student Paper awards for work on evidence combination for Web search and keyword-based queries in structured databases. He is currently an Associate Professor with the Instituto Superior Técnico and a Researcher with INESC-ID, Lisboa, where he is a member of the IDSS group. His research interests include information retrieval, information extraction, machine learning, digital libraries, and recommendation systems. He has authored several publications in major conferences and journals in these areas and has supervised and participated in several research projects. He also currently teaches courses in databases, information retrieval, and management of data and information.

ALTIGRAN S. DA SILVA received the Ph.D. degree in computer science from the Federal University of Minas Gerais (UFMG), in 2002. He is currently an Associate Professor with the Institute of Computing, Federal University of Amazonas (IComp/UFAM). His research interests involve data management, information retrieval, and data mining with emphasis on the worldwide web and social media environment. He has published more than 100 scientific publications in high-quality journals and conference proceedings and served on program committees and editorial boards of prestigious publication venues and conferences in Brazil and abroad. He has been a board member of the Brazilian Computer Society (SBC) and a cofounder of successful technology ventures.