

APPLYING MACHINE LEARNING TO
RELEVANCE EVIDENCE FUSION
AT INDEXING TIME

SHEILA DA NÓBREGA SILVA

APPLYING MACHINE LEARNING TO
RELEVANCE EVIDENCE FUSION
AT INDEXING TIME

Tese apresentada ao Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal do Amazonas como requisito parcial para a obtenção do grau de Doutor em Informática.

ORIENTADOR: EDLENO SILVA DE MOURA

Manaus

Julho de 2020

SHEILA DA NÓBREGA SILVA

APPLYING MACHINE LEARNING TO
RELEVANCE EVIDENCE FUSION
AT INDEXING TIME

Dissertation presented to the Graduate Program in Computer Science of the Universidade Federal do Amazonas in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

ADVISOR: EDLENO SILVA DE MOURA

Manaus

July 2020

Silva, Sheila da Nóbrega

S586a Applying Machine Learning to Relevance Evidence Fusion at
Indexing Time / Sheila da Nóbrega Silva. 2020
84 f.: il. color; 31cm.

Orientador: Edleno Silva de Moura

Tese (Doutorado em Informática) - Universidade Federal do
Amazonas.

1. Computação. 2. Recuperação de Informação. 3.
Indexação. 4. Learning to rank. 5. LambdaMART. I. Moura,
Edleno Silva de. II. Universidade Federal do Amazonas III.
Título



PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
INSTITUTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA



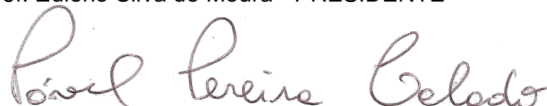
FOLHA DE APROVAÇÃO

"Applying Machine Learning to Relevance Evidence
Fusion At indexing Time"

SHEILA DA NÓBREGA SILVA

Tese de Doutorado defendida e aprovada pela banca examinadora constituída pelos Professores:


Prof. Edleno Silva de Moura - PRESIDENTE


Prof. Pável Pereira Calado - MEMBRO EXTERNO


Prof. Leandro Balby Marinho - MEMBRO EXTERNO


Prof. Thierson Couto Rosa - MEMBRO EXTERNO


Prof. Cláudio Soares da Silva - MEMBRO INTERNO

Manaus, 30 de Julho de 2020

A Deus, por me permitir chegar até aqui com saúde e serenidade.

Aos meus pais, José Herculano da Nóbrega e Maria Gomes da Nóbrega, pelo grande incentivo e amor incondicional.

Aos meus filhos, Leandro e Gustavo, por entenderem cada momento presente, mas na verdade ausente, concentrada nos estudos.

Ao Sidney, que mesmo não estando mais fisicamente ao meu lado, sempre me fortalece de alguma forma.

Aos amigos de vida e do trabalho, pela torcida constante pelo meu sucesso.

Ao Tribunal de Contas do Estado do Amazonas, por ter propiciado as condições necessárias para o avanço desta pesquisa.

Ao meu orientador, por ter colocado os estudos de volta na minha vida em um momento onde tudo estava sem cor.

*“Quando achamos que não somos mais capazes,
chegamos no ponto exato da superação.”*
(Sheila da Nóbrega Silva)

Resumo

O principal objetivo das máquinas de busca é produzir resultados de *ranking* de alta qualidade. Um aspecto importante das máquinas de busca modernas é o uso de um grande número de distintas fontes de evidência de relevância para construir um modelo de *learning to rank* (L2R). Essas evidências coletivamente ajudam a estimar se o documento é relevante ou não para a consulta. O *ranking* com os resultados da consulta é calculado por meio da fusão de todas as fontes de evidência em um único score do documento, para cada documento que compõe o ranking final. Nas últimas décadas vários trabalhos sobre fusão de evidências tem sido feito com a implementação de métodos de L2R.

Os métodos de L2R usam exemplos de consultas com os seus respectivos resultados para treinar modelos de aprendizagem supervisionada que determinam a posição relativa do documento na lista final de resultados. Uma vez treinado, o modelo pode ser usado durante o processamento da consulta para determinar o *ranking* final. Esta abordagem, entretanto, inadvertidamente adiciona custos computacionais para o processamento da consulta, o que pode levar a uma queda no desempenho do tempo de processamento. Para mitigar este problema, foi proposto na literatura uma abordagem alternativa - *Learning to Precompute Evidence Fusion* (LePrEF), baseada em uma técnica de aprendizagem supervisionada com PG (Programação Genética). O modelo LePrEF propõe implementar a fusão de um conjunto de evidências em tempo de indexação, gerando um único índice invertido contendo entradas unificadas representando todas as fontes de evidências. Esses termos unificados são chamados de *Unified Term Impacts* (UTIs). Cada UTI substitui vários atributos por um único valor no índice de documentos, reduzindo assim o esforço para calcular os *scores* dos documentos em tempo de processamento da consulta porque o sistema busca e processa menos valores. A adoção de valores de UTI produz resultados de ranking competitivos. Entretanto, a ausência dos atributos que não estão disponíveis em tempo de consulta pode levar a uma perda de acurácia.

Nesta tese estudamos e propomos uma modificação no LambdaMART, que pas-

samos a chamar de UTI-LambdaMART, um algoritmo de *gradiente boosting* para gerar valores unificados de impactos do termo em tempo de indexação. Também, propomos e avaliamos um modelo híbrido que utiliza valores de UTI com atributos dependentes da consulta. Demonstramos que o nosso método híbrido por entregar resultados com alta qualidade, equivalente aos modelos neurais atuais estado da arte. Os resultados dos experimentos mostram que o nosso melhor modelo híbrido, HLambdaMART, alcança um NDCG@10 igual 0,495 usando apenas 36 atributos em tempo de processamento da consulta, enquanto o melhor baseline alcança 0,490 usando um conjunto maior de atributos em tempo de processamento da consulta. O uso do nosso *framework* híbrido reduz o tempo de execução do LambdaMART em cerca de 35% do tempo que seria executá-lo sem a nossa proposta. Adicionalmente, estudamos e propomos um método simples para obter ganhos significativos na compressão do índice de UTI sem perda na qualidade dos resultados das buscas. Nossa abordagem foi capaz de alcançar 79% de taxa de compressão do índice, enquanto manteve a qualidade dos resultados equivalentes aos métodos que não usam compressão. Conduzimos também experimentos demonstrando o uso do UTI-LambdaMART como um *base ranker*.

Abstract

The production of high quality ranking results is the main goal of web search engines. An important aspect of modern search engines is the use of a large number of distinct sources of relevance evidence to build the learning to rank (L2R) model. Collectively, they determine whether the document is relevant to a query or not. The ranking of query results is computed by fusing all sources of evidence into a single document score, for each document in the final ranking. In the past few decades, most of the works on evidence fusion has been done with the implementation of L2R methods.

L2R methods use examples of queries and their respective results to train supervised learning models that determine the relative position of the documents in the result list. Once trained, the model can be used during query processing to determine the final ranking. This approach, however, inadvertently adds computational costs to query processing, which may lead to a drop in time performance. To mitigate this problem, an alternative approach was proposed in literature — Learn to Precompute Evidence Fusion (LePrEF), based on supervised learning techniques with GP (Genetic Programming). LePrEF proposes to implement the bulk of the evidence fusion during indexing time, generating a single inverted index containing unified entries representing all sources of evidence. These unified entries are called Unified Term Impacts (UTIs). 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.

In this dissertation we study and propose a modified LambdaMART, named UTI-LambdaMART, a gradient boosting algorithm to generate *unified term impacts* (UTI) values at indexing time. We also propose and evaluate 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. The experimental results show that our best hybrid model, HLamb-

daMART, 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 addition, we study and propose a simple method to obtain significant gains in UTI-index compression with virtually no loss in the quality of search results. Our approach was able to achieve 79% compression rate of the index, while keeping the quality of results on par with methods that do not use compression. We also conduct experiments that demonstrate the use of the UTI-LambdaMART as a base ranker.

List of Figures

1.1	Two-stage L2R query processing	2
1.2	Indexing and query processing when using UTI-LambdaMART	5
3.1	The L2R process.	18
3.2	Boosted ensemble of decision trees.	20
3.3	Boosting tree example with N=3 and L=3.	21
3.4	General steps for constructing a LambdaMART model for a ranking problem.	24
3.5	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.	26
3.6	An individual representation in GP	28
4.1	An example of the UTI-LambdaMART model output for indexing and query processing.	41
4.2	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.	42
4.3	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	43
4.4	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. 1.1 are highlighted in blue.	45
5.1	Comparison of the UTI-LambdaMART quality results in MQ2007 with and without the two additional positional features for each term-document . .	53
6.1	Quality results for HiNT and HLambdaMART on MQ2007.	63

6.2	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. . .	67
-----	---	----

List of Tables

5.1	Characteristics of the Datasets	48
5.2	Summary of the data extraction from Gov	49
5.3	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.	50
5.4	Impact in MAP results when varying the number of term-Document positional features	52
5.5	MQ2007 and MQ2008 term-related versions of features 1-15 available in MQ2007 and MQ2008, i.e. features mapping terms to documents.	53
5.6	Query-dependent features of LETOR MQ2007 and MQ2008, plus UTI score and a set of nine features related to the positions of query terms, used in the hybrids methods (HCA, HMART and HLambdaMART)	56
5.7	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).	57
6.1	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.	60
6.2	Comparing the UTI-LambdaMART in MQ2008 test set using the model trained in MQ2007 and in MQ2008. A significant performance degradation is denoted as (∇).	61

6.3	Performances of MART, CA, LambdaMART, DeepRank, HiNT and the hybrid methods, combining UTI-LambdaMART with CA (HCA) and LambdaMART (HLambdaMART). We applied the 9 passage-based features in all L2R methods. A significant performance degradation of HLambdaMART is denoted as (∇).	62
6.4	Performance comparisons of MART with different set of features on MQ2007 and MQ2008.	65
6.5	Performance comparisons of CA with different set of features on MQ2007 and MQ2008.	66
6.6	Performance comparisons of LambdaMART with different set of features on MQ2007 and MQ2008.	66
6.7	LambdaMART and HLambdaMART time (seconds) to index and space (MB) when applied to MQ2007 and MQ2008.	69
6.8	LambdaMART and HLambdaMART average times, in milliseconds, to run base ranker (base) and top ranker (top) on MQ2007 and MQ2008.	70
6.9	Impact on NDCG@10 of UTI-LambdaMART and HLambdaMART when varying the number of decimal places per entry when computing UTI values on MQ2007.	72

Contents

Resumo	xiii
Abstract	xv
List of Figures	xvii
List of Tables	xix
1 Introduction	1
1.1 Problem Statement	3
1.2 Research Goals	4
1.3 Research Questions	5
1.4 Publication	7
2 Related Work	9
2.1 Precomputed Evidence Fusion	9
2.2 Document rank models	10
3 Background	15
3.1 Learning-To-Rank	16
3.2 LambdaMART rank model	19
3.3 Learning UTI values	24
3.3.1 UTI-GP Model	27
3.4 Deep learning model	28
3.5 Evaluation Measures	29
4 The UTI-LambdaMART Model	33
4.1 Modified LambdaMART Algorithm	34
4.1.1 An example of UTI generation using UTI-LambdaMART	39
4.2 Combining UTIs with Query-Time Features	42

4.2.1	Using UTI Values as a Query-Time Feature	43
4.2.2	Using UTI-LambdaMART as a Base Ranker	44
5	Experimental Protocols	47
5.1	Datasets	47
5.2	Baseline Methods	51
5.2.1	UTI Methods	51
5.2.2	L2R Methods	54
5.2.3	Deep Matching Methods	54
5.2.4	Hybrid Methods	55
6	Experiment Results	59
6.1	UTI-LambdaMART for evidence fusion	59
6.2	Hybrid architecture evaluation	61
6.2.1	The impact of UTI on L2R methods	63
6.3	Evaluation the UTI-LambdaMART for base ranker	66
6.4	A performance evaluation	68
6.5	An UTI-Index compression evaluation	71
7	Conclusions and Future Work	73
7.1	Conclusions	73
7.2	Future Work	74
	Bibliography	77

Chapter 1

Introduction

High-quality ranking results are fundamental for web search engines. Users expect answers to their queries displayed at the top of the list of search engines result pages [Saraiva et al., 2001]. Modern search engine users experience fast query response regardless of the size of the datasets; however, any noticeable increase in waiting time can dampen their perception of the quality of the system, thus 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. 1.1 shows query processing performed through a two-step L2R-based search engine [Dang et al., 2013; Capannini et al., 2016; Daoud et al., 2017; Sousa et al., 2019]. In the first step, top- k ranking results are retrieved using a low-cost ranking strategy, such as BM25 [Robertson and Walker, 1994], 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 [Daoud et al., 2017]. 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

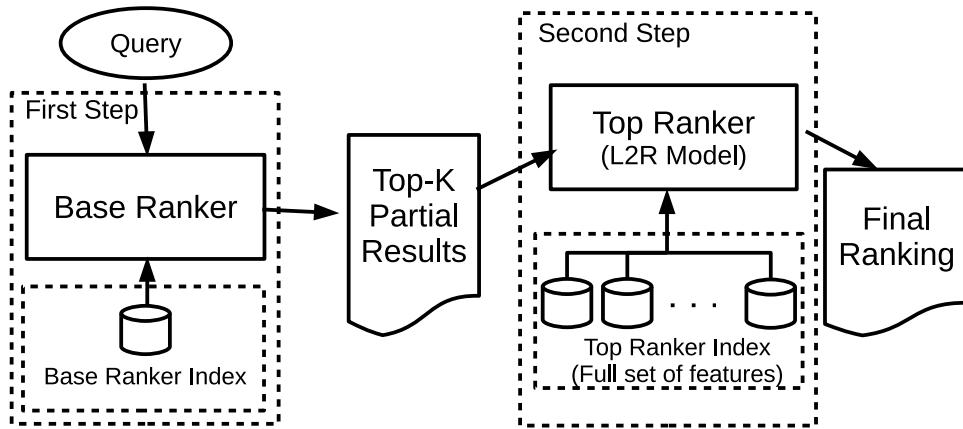


Figure 1.1: Two-stage L2R query processing

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. 1.1 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 [Dang et al., 2013; Capannini et al., 2016; Daoud et al., 2017; Sousa et al., 2019].

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 [Baeza-Yates and Ribeiro-Neto, 2011]. These features are in the top ranker index in Fig. 1.1.

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 [Liu, 2011], such as genetic programming algorithms [de Almeida et al., 2007; Silva et al., 2009], gradient boosting methods [Wu et al., 2010; Lucchese et al., 2018b], and neural networks [Mitra et al., 2017; Pang et al., 2017; Fan et al., 2018; Xu et al., 2019].

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, [Costa Carvalho et al., 2012] 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.

1.1 Problem Statement

A limitation of the UTI-GP 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 [Robertson and Walker, 1994] 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. Other limitation is the

long time required for training the model using genetic programming. Thus, UTI-based methods have important limitations.

1.2 Research Goals

In the present study, we propose UTI-LambdaMART, an adaptation of LambdaMART [Borges, 2010; Wu et al., 2010], 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. Fig. 1.2 illustrates the process of UTI index generation and how the queries are processed using UTI values. At indexing time (Fig. 1.2a), 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. 1.2b, 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.

We improve the proposal of Carvalho *et al.* in terms of four aspects:

1. 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.
2. To address the raised problem, 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.
3. We show that our method offers an extremely low computational effort. The experiments presented indicate it is useful as a practical alternative base ranker

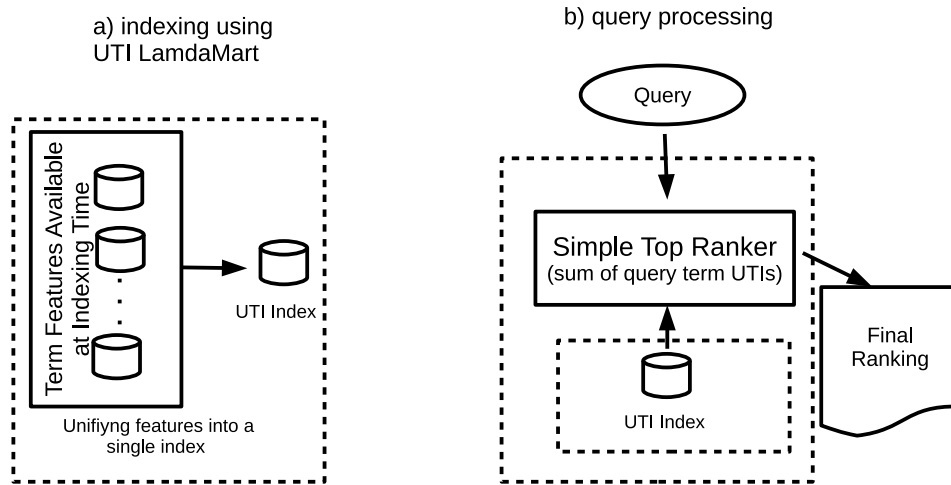


Figure 1.2: Indexing and query processing when using UTI-LambdaMART

for search systems, as it produces a first ranking close to the final result.

4. Using a simple compression scheme, we reduce the space requirements of the inverted index produced, achieving this reduction without significant loss of search quality.

1.3 Research Questions

This work addresses five 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. A term-based learning model produces a score for each document given a single term (as the case of the UTI score), instead of producing a score of a full query, as done by traditional L2R models. Our solution produces competitive results when compared with other L2R methods since, as other methods that produce

UTIs, it encodes a more fine grained information in the learning process, taking features at the term level instead of query level. In our model, we take individual term frequency as a feature, whereas in conventional L2R models, information on the individual frequency of each term is encoded in a single feature, such as a sum of term frequencies or max term frequency among the query terms. We also performed experiments to compare the performance of UTI-LambdaMART with the work of Costa Carvalho et al. [2012], where the results of our LambdaMART adaptation achieves results in 2.8% of the time required for UTI-GP. Using only features at indexing time, we produce results on pair with the original LambdaMART, that uses a full set of features.

- **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. This approach considerably reduces the number of features to be fetched and processed, thus reducing the time to answer the query. We also performed experiments to show that our hybrid approach produces quality scores on pair with the best neural ranking baselines in the literature Pang et al. [2017]; Fan et al. [2018].

- **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 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.
- **RQ5: How much can the UTI index be reduced with virtually no loss in the quality of search results ?** We address the problem of generating UTI values that produce an index that can be compressed. We exploited UTI-LambdaMART to achieve a much smaller index with low impact in terms of its construction time. Our proposal for index size reduction consists of truncating the decimal places in the UTI values generated. We investigate whether it is most profitable to reduce the size of the UTI representation during or after the training.

The rest of this dissertation is organized as follows. In Section 2, we discuss related works. Section 3 presents background information about L2R and UTI learning. Section 4 describe our proposal. Section 5 presents our experimental setup, followed by a comparison with current baselines and the answers to our research question in Section 6. Section 7 concludes the paper and offers directions for future research.

1.4 Publication

The evaluations described in this thesis are also presented in the following paper:

- S. D. N. Silva, E. S. De Moura, P. P. Calado and A. S. Da Silva, *Effective Lightweight Learning-to-Rank Method Using Unified Term Impacts*, in IEEE Access, vol. 8, pp. 70420-70437, 2020, doi: 10.1109/ACCESS.2020.2986943.

Chapter 2

Related Work

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

2.1 Precomputed Evidence Fusion

Anh and Moffat [2002] were the first to propose precomputed term impacts on documents, which was further addressed in two other followup articles [Anh and Moffat, 2005; Anh et al., 2008]. 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.

Costa Carvalho et al. [2012] 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. Basically, the learning process consist of generate individuals to combine all sources of relevance evidence into a single value for each term-document pair. Then, the quality of the rankings generated by the UTIs created for each individual is used as this individual's fitness value. The quality of the results was measured using the

Normalized Discounted Cumulative Gain (NDCG) [Järvelin and Kekäläinen, 2002]. 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.

2.2 Document rank models

Several authors have studied L2R methods in the literature using different machine learning techniques [de Almeida et al., 2007; Silva et al., 2009; Wu et al., 2010; Lucchese et al., 2018b; Pang et al., 2017; Fan et al., 2018; Xu et al., 2019]. Other examples and details of the L2R methods can be found in the works of Liu [2011] and Tax et al. [2015]. 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. [2017] 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 [Liu et al., 2007]; the results, however, are quite inferior to those achieved by the current state-of-the-art methods.

Lucchese et al. [2016] 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 Lucchese et al. [2018a], 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. [2017] 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.

Pang et al. [2017] propose a deep learning approach called DeepRank to relevance ranking in information retrieval. DeepRank simulates the human judgment process aggregating relevance signals from a query-centric context. Basically, the process has three stages: first, the entire document is analyzed to find the relevant locations (where the query terms are); second, the local relevances is determined using CNN and 2D-GRU and, finally, using RNN and a term gating network, these locations are aggregated to a global one for ranking.

Fan et al. [2018] propose a Hierarchical Neural maTching model (HiNT). HiNT 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.

Guo et al. [2019] 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 [Pang et al., 2017] and HiNT [Fan et al., 2018], 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. 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. [2019] 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.

In another recent study, Xu et al. [2019] 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 [Fan et al., 2018]. 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. [2019] 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. In the proposed cascading ranking model, for each stage only the highest

ranked documents are promoted to the next stage, thus the full list of candidate documents is seen only in the earlier stages. They work with different set of features across the stages. 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. [2019] 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. [2018] 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.

Chapter 3

Background

In this section, we further explain the main differences between learning-to-rank and UTI learning. We also summarize the LambdaMART model and the evaluation metrics adopted to evaluate the quality of our experiments.

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 [Robertson and Walker, 1994] 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.

As search systems evolved in the past decades, several new sources of relevance information have been introduced and, in spite of initial efforts provide simple functions to combine distinct sources of relevance, such as combining evidences using a Bayesian Network probabilistic approach [Silva et al., 2000], just performing a linear combination [Westerveld et al., 2001] or using a SIGMOID adjustment function in an attempt to better modeling the importance of the combined features [Craswell et al., 2005]. A common problem with all the mentioned approaches is that they adopt too simplified

models for combining evidences, which tend to achieve low quality of results, specially as the number of evidences to be combined grow or when there are relationships between sources of relevance evidence that are not captured by them.

In a more promise research direction, solutions employing machine learning techniques to automatically construct ranking models also 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.

3.1 Learning-To-Rank

In information retrieval, for a given query, the objective of the L2R is to order a set of resulting documents according to their degree of relevance. 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 [Li, 2011]. To explain the learning system, we first need to define the training data used to create a ranking model. Let $Q = \{q^{(1)}, q^{(2)}, q^{(3)}, \dots, q^{(m)}\}$ denotes a set of queries. For each query $q^{(i)} \in Q$, there is an associated set of k resulting documents $D_i = \{d_1^{(i)}, d_2^{(i)}, \dots, d_k^{(i)}\}$. 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 $fs(q^{(i)}, d_j^{(i)}) = \{fs1_j^{(i)}, fs2_j^{(i)}, \dots, fsn_j^{(i)}\}$ and its respective document relevance judgment $r_j^{(i)}$, 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 (Fig. 3.1), or ranking function. Consider that queries and documents are represented together as vectors in

R^n , basically we need to find a rank function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ which minimizes

$$R(f) = \frac{1}{m} \sum_{q=1}^m L(\pi(f, D_q), r_q) \quad (3.1)$$

where, $\pi(f, D_q)$ is the ranking of documents for query q and L measures the discrepancy between $\pi(f, D_q)$ and r_q . L is the loss function and can be defined according to the adopted L2R approach. There are three approaches used in L2R implementations:

1. **Pointwise**: an approach in which the loss function is defined based on a single document. Pointwise approach ignores the group structure of ranking, then during the training the data is seen in the same way as in conventional supervised learning, when we take the relevance r as a class label and the problem becomes a classification problem. Given a query, we can use the model to ranking documents simply by sorting the documents according to the scores provided by the model.
2. **Pairwise**: in this approach, the loss function is defined based on pairs of feature vectors (related with pairs of documents) with different judgments of relevance. The ranking problem is transformed into pairwise classification or pairwise regression. Ranknet[Burges et al., 2005] and LambdaRank[Burges et al., 2006] are examples of this approach.
3. **Listwise**: an approach that considers the entire list of documents to define the loss function. This approach learns during the training a rank function $f(x)$ that can assign scores to documents (feature vectors) and rank the documents using the scores, such documents with higher scores are ranked higher. ListNet[Xia et al., 2008], AdaRank[Xu and Li, 2007] and LambdaMART[Burges, 2010] implement the listwise approach.

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 only 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.

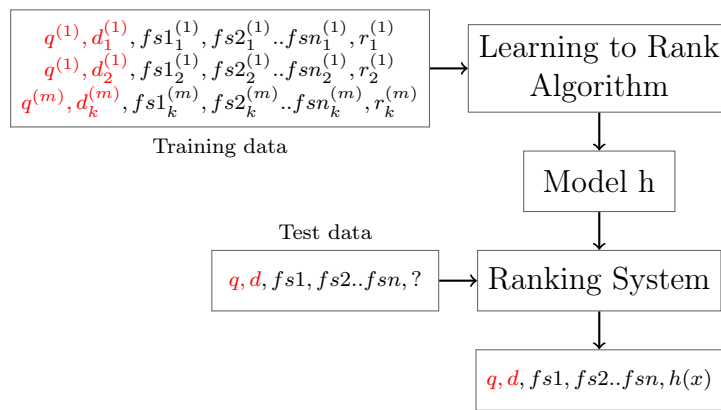


Figure 3.1: The L2R process.

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 [Joachims, 2002], 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 methods perform pairwise and listwise 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 a listwise method, but one that produces a function f as described above.

3.2 LambdaMART rank model

LambdaMART [Burges, 2010] is a combination of MART [Friedman, 2001], an L2R method that adopts a boosted tree model, and LambdaRank [Burges et al., 2006], which is based in RankNet [Burges et al., 2005]. LambdaMART is the MART method using the cost function of the LambdaRank.

LambdaMART constructs a ranking model F that maps each set of features of instance x into a numerical score $F(x)$. At training time, each instance $x = (q, d, r)$ is a tuple composed of a set of features values related to query q and document d , and a set of relevance r that indicates the relevance of the document d to the query q .

The model constructs a function that maximizes the quality of the ranking in the training set, using information on the relevance level r of each document d with respect to query q to compute the quality of the results. LambdaMART generates a tree ensemble model (Fig. 3.2). The goal is to find a set of trees that, together, minimize the loss function. The output $F(x)$ can be defined as

$$F(x) = \sum_{i=1}^N f_i(x) \quad (3.2)$$

where N is the number of trees and each f_i is a function assigned for a individual regression tree.

Mapping occurs by traversing each regression tree, where the direction to be followed (left or right) is defined using the value of each feature and the threshold learned. The leaves are the output of the tree. In decision trees, we have two parameters: the parameter region assignments R_n that assigns an instance x_n to a leaf node l , and the

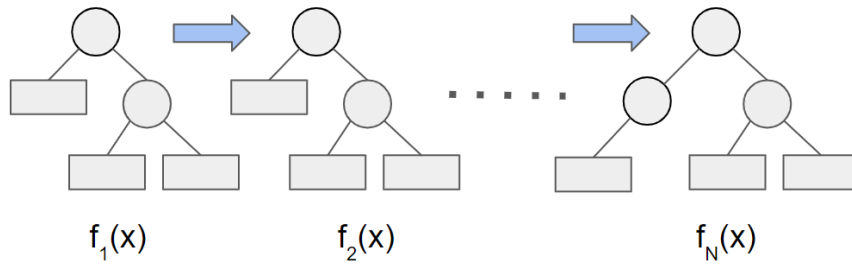


Figure 3.2: Boosted ensemble of decision trees.

leaf outputs γ_l that are the tree's output for all instances assigned for the region R_l . 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 parameters defined by the user are N , L and the learning rate η (value that is multiplied for every $\gamma_{l,n}$ for every tree).

An example of the model generated by the LambdaMART is shown in Fig. 3.3. The parameters are $N = 3$ and $L = 3$. We consider the instance x , with features $f_1 = 0.5$ and $f_2 = 38$, then we parsing x down the tree f_1 , where f_1 and f_2 are the features assigned to the nodes of this tree with their respective thresholds 0.43 and 36. In our example, for the instance x the value of f_1 is greater than the threshold 0.43, so we proceed to the right of the root node, for the right child of the root node the value of the feature f_2 is also greater than the threshold value, so we turn right again and finally reaches a leaf node $\gamma_{3,1}$ with the predicted value 0.3. We repeat the same steps for the others trees. The final score is the sum of the predicted values in each model tree. The mapping is 1:1; i.e., for each instance, it is assigned a score.

The Cost Function

LambdaMART objective function is based on LambdaRank, which is based on RankNet. The Ranknet model uses a probabilistic cost function implemented in a neural network. We begin describing the RankNet objective function. The RankNet training data is partitioned by query. For a query q , for each pair of documents d_j , d_k ,

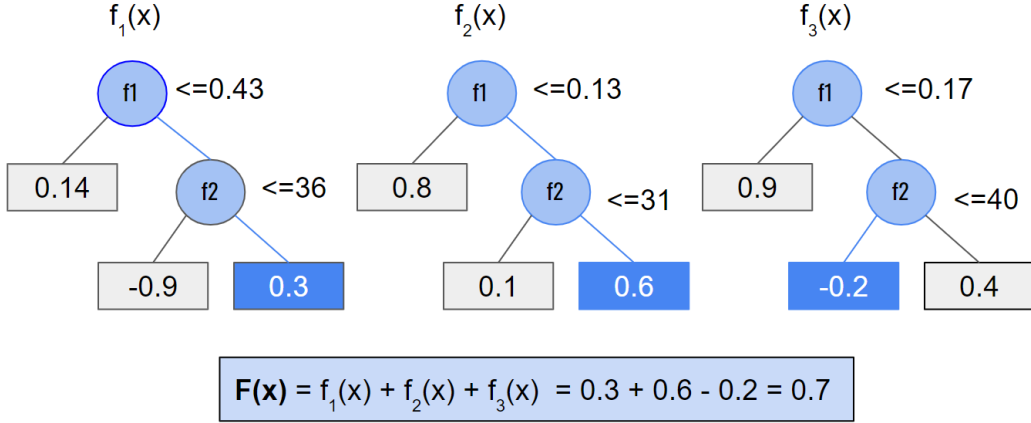


Figure 3.3: Boosting tree example with N=3 and L=3.

with different labels, and feature vectors x_j, x_k , the outputs of the model are mapped to a learned probability of the document d_j be ranked higher than d_k (because the document d_j is labeled as 'relevant' and document d_k 'not relevant') is expressed as a sigmoid function:

$$P_{jk} = \frac{1}{1 + e^{-\sigma(s_j - s_k)}} \quad (3.3)$$

where the scores $s_j = f(x_j)$, $s_k = f(x_k)$, and the parameter σ determines the shape of the sigmoid. The cost function calculates the cross entropy loss:

$$C = -\bar{P}_{jk} \log P_{jk} - (1 - \bar{P}_{jk}) \log(1 - P_{jk}) \quad (3.4)$$

where \bar{P}_{jk} is the model probability and P_{jk} the actual probability (1 for relevant judgments and 0 for not relevant judgments). The RankNet combines Eq. 3.3 and 3.4 then the cost when d_j must be ranked higher than d_k is:

$$C = s_j - s_k + \log(1 + e^{s_j - s_k}) \quad (3.5)$$

LambdaRank is based on RankNet, but instead of pairwise error, the Lamb-

daRank optimization cost C has the property that

$$\frac{\partial C}{\partial s_j} \gg \frac{\partial C}{\partial s_k} \quad (3.6)$$

whenever $j \gg k$. The main idea is to express how the rank order of documents change after sorting them by score for a given query. We use $S = s_1, \dots, s_n$ to denote the scores assigned by F to D , and $r = r_1, \dots, r_n$ to denote how relevant the document d_j is for query q_i . The gradient of the cost C with respect to the score of the document d_j (at rank position j) for the query q_i , as

$$\frac{\partial C}{\partial s_j} = -\lambda_j(s_1, r_1, s_2, r_2, \dots, s_{n_i}, r_{n_i}) \quad (3.7)$$

where positive λ_j means that the document must move up the ranked list to reduce the cost, and negative value means it must moves down. Thus, the λ function depends on the ranked order (based on the computed scores) of the all documents returned for a query q_i , which makes LambdaMARTa listwise approach. LambdaRANK solved the problem of information retrieval measures being discontinuous or flat, computing the gradients after the documents have been sorted by their scores. Burges et al. [2006] prove that to optimize some IR measure we just multiply the $|\Delta Z|$ by λ , where $|\Delta Z|$ is the difference of an IR measure when d_j and d_k are swapped. Here, we are interested in optimizing the NDCG, then we just multiply the λ 's by the $\Delta NDCG$ (Eq. 3.8), that is the change in $NDCG$ when swapping d_j and d_k .

$$\lambda_{jk} = \frac{\partial C(s_j - s_k)}{\partial s_j} = \frac{1}{1 + e^{(s_j - s_k)}} * \Delta NDCG \quad (3.8)$$

We use Normalized Discounted Cumulative Gain (NDCG) to evaluate the agreement between the ranking produced by F and the relevance labels r for query q .

Gradient Boosting Regression Tree

LambdaMART is a boosted tree model, so the output of the model is a linear combination of the outputs of a set of regression trees. During the training suppose we are given a data set $\{f_{s_i}, r_i\}$, $i = 1, \dots, m$ where the set of features $f_{s_i} \in R^d$ and the labels $r_i \in R$. For a give vector f_{s_i} we index its features values by $f_{s_{i,j}}$, $j = 1, \dots, d$. First we consider all data to be resident on the root node, and we need to do the first split and distribute the data on two leaves connect to the root. To do it, we loop through all training instances to find the feature $f_{s_{i,j}}$ and threshold t such that, if all training instances with $f_{s_{i,j}} \leq t$ fall to the left child node and the rest fall to the right child node, then the sum

$$S_j = \sum_{i \in L} (\lambda_i - \bar{\lambda}_L)^2 + \sum_{i \in R} (\lambda_i - \bar{\lambda}_R)^2 \quad (3.9)$$

is minimized. Where $\bar{\lambda}_L$ is the mean of λ 's of the set of instances that fall to the left child node and $\bar{\lambda}_R$ the mean of λ 's of all instances that fall to the right child node. After calculating S_j for all features, the feature j and threshold t attached to the rood node are those that reach the minimal S_j . To build a tree with a total of L leaves this process is executed $L - 1$ times. For each leaf nodes l we assign the value of γ_l , which is the mean of the λ 's of the instances that reach the leaf node l . The leaf output γ_l is the tree's output for all instances assigned to region R_l . LambdaMART can decrease the utility for some queries while increasing the general utility.

Each regression tree f_i , with L leaves, models the λ for the entire set of instances. The least squares is used to compute the splits, it defines the choice of a specific threshold t and node j of the generated regression tree. For each leaf is assigned the mean of λ 's of the training instances that fall there. Each tree contributes to a gradient step in the direction that minimizes the loss function.

The lambdaMART algorithm

Fig. 3.4 shows the general steps used by LambdaMART to build a rank model. Refer to the original method [Wu et al., 2010] 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.

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[Wu et al., 2010] 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 3.4: General steps for constructing a LambdaMART model for a ranking problem.

3.3 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. The impact of a term for a document is learned based on its overall impact on queries.

To explain the learning process, we first define the training data used to create a UTI model. Let $Q = \{q^{(1)}, q^{(2)}, q^{(3)}, \dots, q^{(m)}\}$ denotes a set of queries. For each query $q^{(i)} \in Q$, there is an associated set of k resulting documents $D_i = \{d_1^{(i)}, d_2^{(i)}, \dots, d_k^{(i)}\}$, and j query terms $t_i = \{t_1^{(i)}, t_2^{(i)}, \dots, t_j^{(i)}\}$. Each triple, query, document, and term $(q^{(i)}, d_j^{(i)}, t_k^{(i)})$, with $q^{(i)}$ being a query, $d_j^{(i)} \in D_i$ being a document, and $t_k^{(i)} \in t_i$ being a term, has an associated feature set $fs(q^{(i)}, d_j^{(i)}, t_k^{(i)})$ containing n related features $fs(q^{(i)}, d_j^{(i)}, t_k^{(i)}) = \{fs1_{(j,k)}, fs2_{(j,k)}, \dots, fsn_{(j,k)}\}$ and its respective document relevance judgment $r_{(j)}^{(i)}$, representing the relevance of document $d_j^{(i)}$ with the term $t_k^{(i)}$ as a result for query $q^{(i)}$. The training data are the input used to learn a UTI model (Fig. 3.5).

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. 3.5 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.

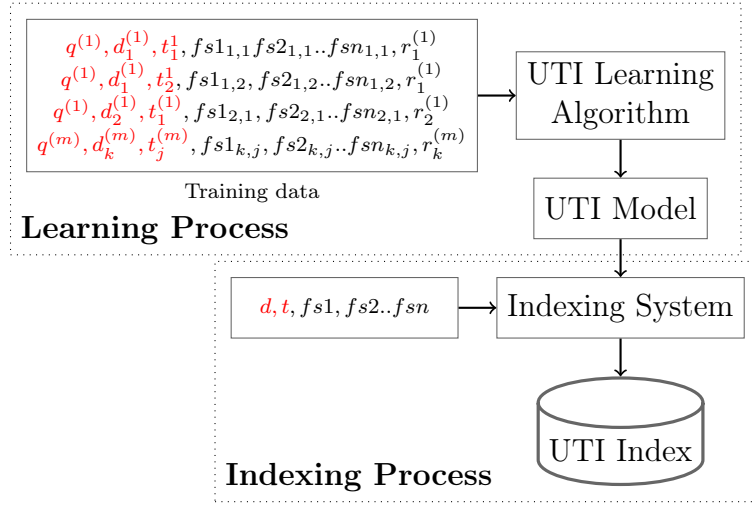


Figure 3.5: The process used to generate a UTI value computation model, the learning process, described in the upper part of the figure, and the indexing process, which takes the model and pairs of terms and documents to produce UTI values.

The algorithm learns the patterns to compute UTI values using the queries in the learning process, as depicted in Fig. 3.5. 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.

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 because during the learning process we discard all features not available to compute UTIs, such as the query dependent features.

3.3.1 UTI-GP Model

In a previous study, [Costa Carvalho et al., 2012] proposed LePrEF, referred to here as UTI-GP, a method to fuse relevance evidences at indexing time using a genetic programming method. LePrEF adopts the concept of a single numerical value to represent the whole set of relevance evidences, storing Unified Term Impacts (UTIs) in the inverted index instead of storing several values for each isolated source of evidence. Instead, it fuses the sets of index entries for all relevance evidences into a single value.

UTI-GP is a Learning to Rank method that is based on the idea of performing the fusion of different sources of relevance at indexing time. To do so, it introduced the concept of Unified Term Impacts (UTIs), which are single numerical values representing the impact that a term has in a document, taking into consideration all sources of relevance evidence. To achieve this fusion, it uses Genetic Programming (GP) in order to generate individuals (mathematical formulas, as can be seen in Fig. 3.6) to combine all sources of relevance evidence into a single value for each term-document pair. Then, the quality of the rankings generated by the UTIs created for each individual is used as this individual's fitness value. In the original paper, the quality of the results was measured using the well-known Normalized Discounted Cumulative Gain (NDCG) Järvelin and Kekäläinen [2002].

After finding the best individual result during the training stages, the whole dataset has to be indexed according to that individual, generating an UTI inverted index

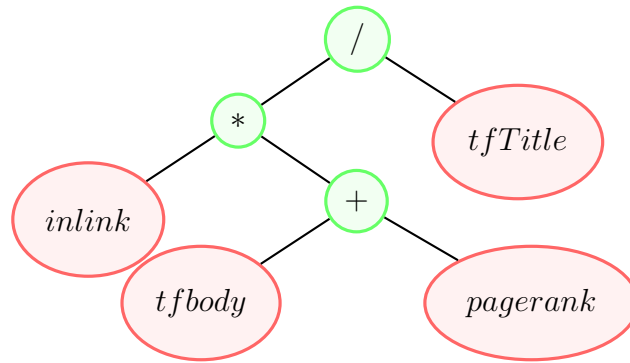


Figure 3.6: An individual representation in GP

3.4 Deep learning model

One of the problems faced by the L2R methods is about the available features during the learning of the model. The more effective the features, the better the results. In this context, deep learning methods have been applied for automatically learning ranking features. Huang et al. [2013] use a deep neural network (DNN) for mapping the raw text features (query and documents) into the features in a semantic space. The implementation consists of two vector: a high-dimensional vector for the input of the raw text features and, another vector for the output the model in a low-dimensional semantic feature space. The cosine similarity between these vectors is used to compute the relevance score.

Recent studies are focused on discovering contexts [Pang et al., 2017; Fan et al., 2018], trying to simulate the way people judge the document as relevant or improving the identification of parts of the document with high relevance in relation to other parts, and the impact of this on the overall relevance of the document. Pang et al. [2017] use convolutional neural network (CNN) or two dimensional gated recurrent units (2D-GRU) to determine the local relevances. Their model, DeepRank, was the first deep IR model to outperform learning to rank models. [Fan et al., 2018] use deep matching networks to automatically learn the passage-level relevance signals.

3.5 Evaluation Measures

MAP, P@N and NDCG@N are evaluation metrics [Robertson, 2000] commonly adopted when evaluating the quality of results produced by search systems [Baeza-Yates and Ribeiro-Neto, 2011]. 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:

$$MAP(QR) = \frac{\sum_{i=1}^{|QR|} AvPel(qr_i)}{|QR|} \quad (3.10)$$

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. MAP is designed for binary relevance levels. 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. In this thesis we use the LambdaMART algorithm to maximize the NDCG function. For each query, $NDCG@N$ is computed as

$$NDCG@N(QR, REL) = \frac{\sum_{i=1}^{|QR|} \frac{DCG@N(qr_i)}{IDCG@N(rel_i)}}{|QR|} \quad (3.11)$$

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

$$DCG@N(qr_i) = \sum_{j=1}^N \frac{2^{relScore(qr_i, j)} - 1}{\log_2(j + 1)} \quad (3.12)$$

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 2 (relevant). All documents with a relevance score greater than zero are considered relevant.

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

$$IDCG@N(rel_i) = \sum_{j=1}^N \frac{2^{relScore(rel_i, j)} - 1}{\log_2(j + 1)} \quad (3.13)$$

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 [Cormack and Lynam, 2007; Smucker et al., 2007, 2009].

Chapter 4

The UTI-LambdaMART Model

Currently, deep learning models are among the most successful methods applied to the L2R problem [Guo et al., 2019]. They achieve superior quality when compared to most of the alternative L2R models available nowadays. However, they have several practical disadvantages that still need to be solve to allow them to be viable when considering time performance issues. One of them is that during the learning process, while the L2R models use manually extracted features, the deep models use the raw text. This property can be seem as an advantage at a first glance, since there is less engineering effort to find and model useful features. These models might, for instance, capture interesting and useful features, such as the possibility of catching the relationship between the relevance of the query-terms and their location in the document. While at first these methods might have an advantage when considering the potential modeling alternatives, this access to the raw text is a disadvantage because it makes the models more expensive both at learning and, most importantly, at query processing times. Further, the main problem faced by neural models is the high cost of learning. The implementation of learning raw text features is complex due to the high-dimensional data and requires more computational resources and time to be implemented than other simpler learning to rank methods.

When considering efficiency in L2R, the LePrEF approach becomes a good alternative to capture features of each term in the document without adding costs to the query processing. It is also able to capture and model term to document relationships, while traditional learning to rank methods only allow learning query to document relationships. As we will show in the Chapter 6 that presents experimental results, we can explore term to document modeling to produce effective and light weigh L2R models.

We now show our proposal to adapt a fast L2R algorithm to learn UTI at the indexing time, and we exploit the position of the term in the document as a new feature. Our experimental results (described in Section 5.2.1.1) show that the use of the first two positions of the terms in the document during the UTI learn reaches a higher quality than that obtained by the LambdaMART top rank algorithm using all features available at query time. We also propose a hybrid model to address the lack of features available only at query time. In this new model, the sum of the UTI values of all query-terms is used as a feature for a top ranker algorithm.

In Section 4.1 we present our proposal for the UTI model, which is an adaptation of the LambdaMART algorithm [Wu et al., 2010]. In Section 4.2 we show how to combine UTIs with query-time features.

4.1 Modified LambdaMART Algorithm

We use the LambdaMART algorithm to implement our model. 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 [Guo et al., 2019]. 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 L2R model and

when processing queries with the generated 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.

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 x_t as described in Eq. 4.1.

$$x_t = (t, d, fs_{(d,t)}) \quad (4.1)$$

where $t \in q$ and $fs_{(d,t)}$ denote a set of feature values related to the document d and term t . Each instance x has $|q|$ instances x_t , where $|q|$ is the total number of query terms. The query-document score is calculated according of Eq. 4.2.

$$Score_{(q,d)} = \sum_{\forall t \in q} Ft(x_t) \quad (4.2)$$

where each query-document score 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.

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

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  $Ft$ 
1 foreach  $(q, d, r_{(d,q)}) \in \mathcal{M}$  do
2   foreach  $t \in q$  do
3      $Ft_0((t, d, fs_{(d,t)})) \leftarrow 0$ 
4   end
5 end
6 for  $n \leftarrow 1$  to  $N$  do
7   foreach  $(q, d, r_{(d,q)}) \in \mathcal{M}$  do
8      $scores[d, q] \leftarrow 0$ ;
9     foreach  $t \in q$  do  $scores[d, q] \leftarrow scores[d, q] + Ft_{n-1}(t, d, fs_{(d,t)})$  ;
10    end
11  end
12  foreach Query  $q \in \mathcal{M}$  and each pair of documents  $(d_j, d_k)$  resulting of  $q$ 
    do
13    if  $(r_{(d_j,q)} > r_{(d_k,q)})$  and  $(j > k)$  then
14       $\Delta\lambda \leftarrow \left( \Delta NDCG(d_j, d_k, q, scores) / (1 + e^{(scores[d_j,q] - scores[d_k,q])}) \right)$ ;
15       $\Delta w \leftarrow \Delta\lambda \times (1 - (1 + e^{(scores[d_j,q] - scores[d_k,q])}))$  ;
16      foreach  $t \in q$  do
17         $\lambda[t, d_j, q] \leftarrow \lambda[t, d_j, q] + \Delta\lambda/|q|$ ;
18         $\lambda[t, d_k, q] \leftarrow \lambda[t, d_k, q] - \Delta\lambda/|q|$ ;
19         $w[t, d_j, q] \leftarrow w[t, d_j, q] + \Delta w/|q|$ ;
20         $w[t, d_k, q] \leftarrow w[t, d_k, q] + \Delta w/|q|$ ;
21      end
22    end
23  end
24  Create  $L$  leaf tree  $\{R_{ln}\}_{l=1}^L$  on  $\{\forall (q, d, r_{(d,q)}) \in \mathcal{M}, \text{ and } \forall t \in q | (t, d, fs_{(d,t)}), \lambda[t, d, q]\}$ ;
25  Assign leaf values to  $\{R_{ln}\}_{l=1}^L$  using the calculated  $\lambda$  and  $w$  based on
    Newton step;
26  foreach  $x = (q, d, r_{(d,q)}) \in \mathcal{M}$  do
27    foreach  $t \in q$  do
28      Update the model  $Ft_n(t, d, fs_{(d,t)})$  using parameter  $\eta$  and  $\{R_{ln}\}_{l=1}^L$ 
29    end
30  end
31 end

```

computing a score for each document-query pair ($scores[d, q]$, lines 7–11). Different from the original algorithm, here we need to compute this score considering that the

model generates a single UTI for each term-document, instead of producing a single score for all the query-term of the document. So, this score consists of summing each term's individual UTI, using the model Ft_{n-1} , computed so far by the algorithm. Thus:

$$scores[d, q] = scores[d, q] + Ft_{n-1}(t, d, fs_{(d,t)}) \quad (4.3)$$

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 , and $j > k$. This happens if the document at position j is below the document at position k in the rank although it is more relevant, which means that the document d_j was ranked below the document d_k .

$\Delta NDCG$ is then used to compute the $\Delta\lambda$ and Δw values associated with each pair of documents (d_j, d_k) as shown in Eq. 4.4. The $\Delta NDCG$ is computed for $O(n(i)^2)$ documents pairs, where $n(i)$ is the number of documents for query q_i . These two steps (lines 14–15) are exactly equal to the original LambdaMART. The variable $\Delta\lambda$ computes the loss function optimization on each train point. This is one of the points that made LambdaMART popular, the ease of optimizing any IR metric, simply replacing the $\Delta NDCG$ with the desired information retrieval metric.

$$\Delta\lambda \leftarrow \left(\Delta NDCG(d_j, d_k, q, scores) / (1 + e^{(scores[d_j, q] - scores[d_k, q])}) \right) \quad (4.4)$$

The Δw (show in Eq. 4.5) is computed here to be used in the Newton's optimization step.

$$\Delta w \leftarrow \Delta\lambda \times (1 - (1 + e^{(scores[d_j, q] - scores[d_k, q])})) \quad (4.5)$$

Each term of the query q has its value of λ increased by $\Delta\lambda/|q|$ for the document d_j , and decreased by the same amount for the document d_k as shown in Eq. 4.6.

$$\begin{aligned}\lambda[t, d_j, q] &= \lambda[t, d_j, q] + \Delta\lambda/|q| \\ \lambda[t, d_k, q] &= \lambda[t, d_k, q] - \Delta\lambda/|q|\end{aligned}\tag{4.6}$$

All query-terms in documents d_j and d_k have their values of w increased as described in Eq.4.7

$$\begin{aligned}w[t, d_j, q] &= w[t, d_j, q] + \Delta w/|q| \\ w[t, d_k, q] &= w[t, d_k, q] + \Delta w/|q|\end{aligned}\tag{4.7}$$

Note that since the judgment of relevance is over the entire query, the values computed are divided by the number of query terms $|q|$ ($\lambda[t, d, q]$ and $w[t, d, q]$, respectively, lines 16–21).

After computing the gradients λ and weighs w for the entire dataset, the algorithm fits the regression tree (lines 24–25). First, it creates the tree n with L leaves $\{R_{ln}\}_{l=1}^L$. Suppose for a given vector of features $fs_{(d,t)_i}$, we index its feature values by $fs_{(d,t)_{ij}}$, $j = 1, \dots, z$. $fs_{(d,t)_i} \in R^z$ and labels $r_{(d,q)_i} \in R$, where z is the dimension of the region. Basically the region assignment R_i assigns a training example x_{ti} to a leaf node l .

Every feature $fs_{(d,t)_{ij}}$ runs through the tree until it reaches a leaf node. All data that falls on a specific node is taken into account when deciding to split the values of the current set of leaf nodes. For a better understanding, lets consider that all data is on the root node. For a given feature, the algorithm iterates for all samples until it finds the threshold h , that is the value that better determines the split, so the samples with $fs_{(d,t)_{ij}} \leq h$ fall to the left child node, and the rest fall to the right child node. The feature and threshold assigned to the root node are those that minimize the loss function.

After building the tree, it is applied a single step of Newton's method to optimize

lambda predictions in terminal nodes¹. The next regression tree, for each training point evaluated, models the m derivatives of the cost related with the current model, where m is the total of training instances.

$\gamma_{l,n}$ is the output of the tree and it is a fixed value associate for the tree n , and leaf l . Thus, as shown in the Eq. 4.8 each tree leaf $\gamma_{l,n}$ is the sum of Δ_λ of all instances i where l is the final node, divided by the sum of Δ_w values from the same instances. Since w values are always positive, this step serves to reduce the magnitude of the lambdas. This fact makes the algorithm conservative when it is in the right direction, but it ends up having a positive impact when it is in the wrong direction.

$$\gamma_{l,n} = \frac{\sum_{(t,d,fs_{(d,t)})_{i \in R_{l,n}}} \Delta \lambda_i}{\sum_{(t,d,fs_{(d,t)})_{i \in R_{l,n}}} \Delta w_i} \quad (4.8)$$

Finally, on lines 26–30, the original step for updating the model are executed, and the model Ft_n is updated. Each tree models the gradient of the cost for a score model, and a new regression tree is added to the ensemble applying the learning rate η for each leaf $\gamma_{l,n}$.

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.

4.1.1 An example of UTI generation using UTI-LambdaMART

The first step is to use UTI-LambdaMART to train the model that will be applied during the indexing time to generate the UTIs for each term-document. The general process to learning UTI values is detailed in Section 3.3.

¹Newton's method is an iterative method for finding the roots of a differentiable function F , which are solutions to the equation $F(x) = 0$. In optimization, Newton's method is applied to the derivative f' of a twice-differentiable function f to find the roots of the derivative (solutions to $f'(x) = 0$), also known as the stationary points of f

Fig. 4.1 shows an example using the UTI-LambdaMART algorithm for indexing and query processing. In this example, the model is generated at the end of the interactions. Fig. 4.1 illustrates the output of the method, which is a model with three trees ($N = 3$) and each tree with four leaves ($L = 4$). The node represents the feature label and the values of the threshold, and the final nodes are learned during the training of the model. The actual value of the feature determines the direction to follow in the tree, on the left if the value is low and on the right if it is above the threshold. Each next regression tree model was created based on the previous model. The threshold and leaf of each tree were optimized during the training process, for this reason their values are different in each of the trees created. The goal during the learning is finding a set of trees that when together minimize the loss function.

In the indexing time, for the example shown in Fig. 4.1, the document GX000-14-11495597 and the word "reheat" have as input of the UTI-Model three features, f_{11} (length of the body of the document), f_{16} and f_{17} (the first and second positions where "reheat" appears in the document). These features can be seen on the trees with the labels "11", "16" and "17". Let's consider that this model was trained with only these three features.

The UTI value produced is the linear combination of all regression trees and thus the sum of the terminal nodes reached by their feature values (f_{11} , f_{16} , and f_{17}). 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. The cost to calculate the UTI is the time to go through each tree, accumulating the output values of each one for the set of input features.

In the query processing, for the same document, given a query "temperature for reheat food", the document score is a simple sum of UTI values of each query term, thus the score is 3.02013 in the given example. For the document GX160-11-2318949, the score is -0.2869. We note that the impact of not having the term in the document is learned based on features document-dependent and on the relevance of the term to

other documents.

Regarding time performance, the improvement achieved by the UTI-model in comparison to methods that process all the features at query processing times in the query processing is to reduce the time to compute the document score for a given query, because all the features available at the time of indexing do not need to be processed again at the query-time.

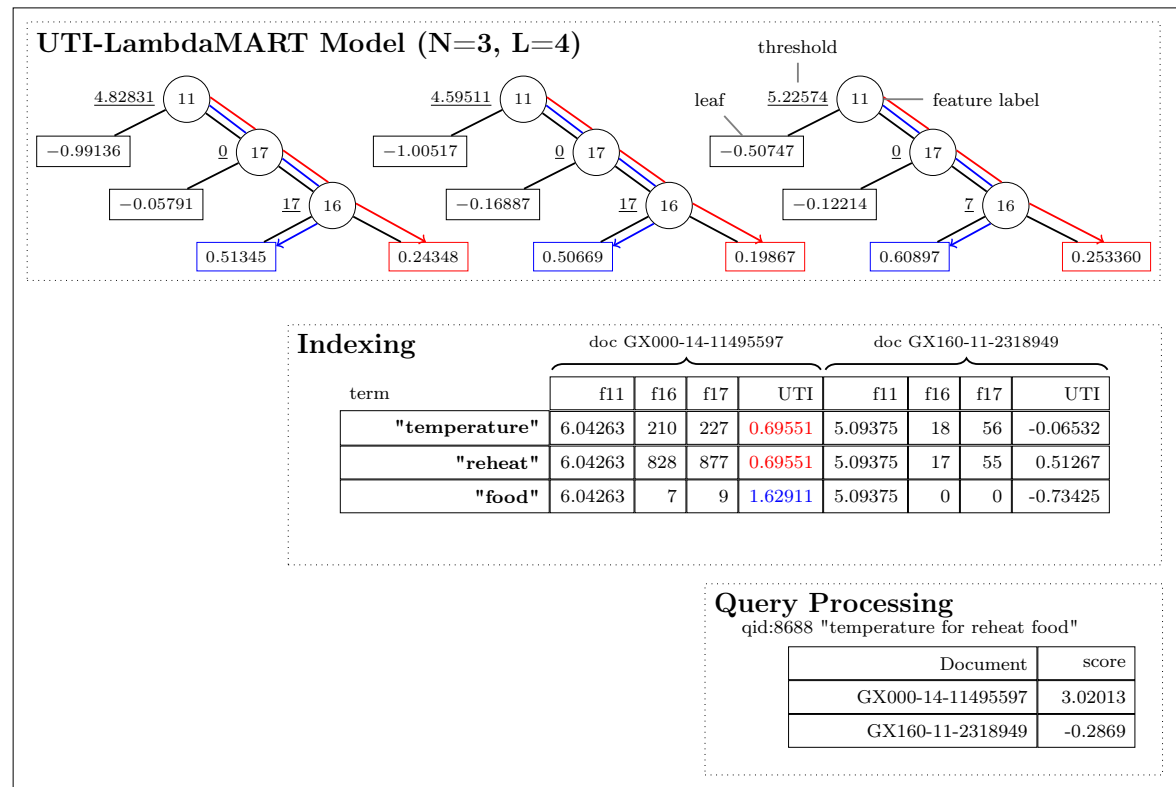


Figure 4.1: An example of the UTI-LambdaMART model output for indexing and query processing.

Fig. 4.2 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. We can conclude with the generated UTI values that the word "food" is the one that has the highest impact on document GX000-14-11495597, followed by the words "cook" and "reheat".

Regarding quality of results, a possible benefit of UTI-LambdaMART, compared

	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 4.2: 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.

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, and features that indicate the position of the term in the document, whereas in traditional L2R methods, this information exists only as an aggregated value, such as the sum of all *TF* values or are just not considered, such as positional features about each individual term. On the other hand, when using UTI there is a lack of information that is only available at query processing time, such as BM25 scores or user profiles, and this absence of information may lead to a loss of quality. In the next Chapter, we present 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 and study here hybrid approaches that may overcome the possible disadvantages of using only UTI values for the ranking.

4.2 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.

4.2.1 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 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 Section 5.

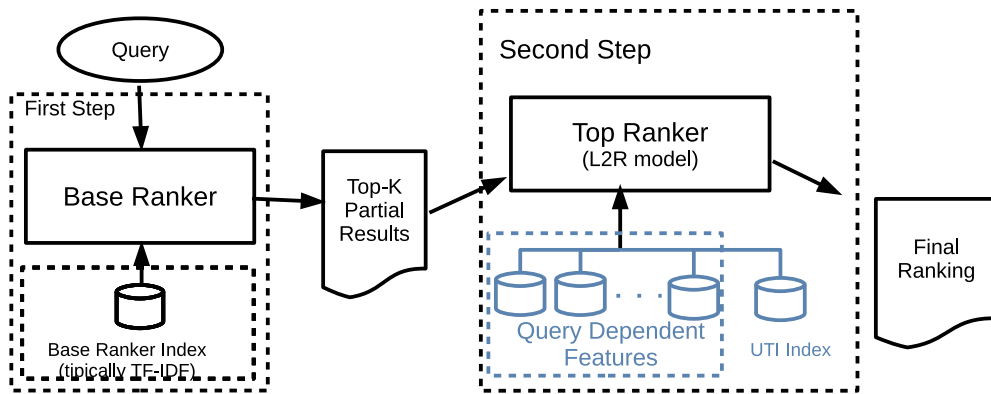


Figure 4.3: Query processing using a hybrid approach that adopts UTI values to substitute all features available at indexing times. The L2R model combines UTI values with the remaining features. Differences from the architecture described in Fig. 1 are highlighted in blue

Fig. 4.3 shows the complete query processing steps when using our proposed hybrid approach, highlighting the differences from Fig. 1.1 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. 1.2b, 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. 1.1.

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.

4.2.2 Using UTI-LambdaMART as a Base Ranker

Base rankers are simple methods with a low computational cost for computing query results [Capannini et al., 2016]. Another method for creating a hybrid approach is to take advantage of UTI values in the first step of the query processing, in the base ranker. When processing queries with UTI values, the final score for each document is computed as a simple sum of UTI values for 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. 4.4 , where UTI values are used in the first step of query processing. The differences from the architecture described in Fig. 1.1 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. 4.3. The goal here is not so much to yield improvements in the final ranking quality achieved by the top-ranker, but rather to reduce the number of top- k documents retrieved in the first step of the query processing by improving the quality of results provided by the base ranker, thereby reducing the overall cost of producing the final ranking. We remember that the cost of query processing may be influenced by the number of top- k results that need to be

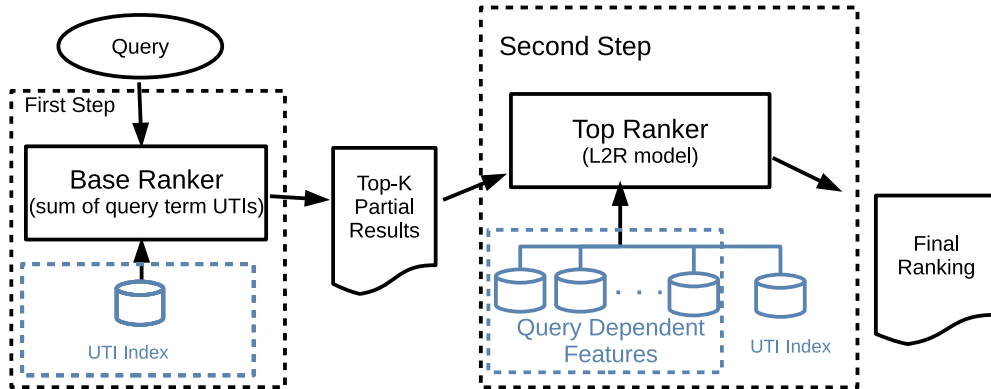


Figure 4.4: Query processing using the hybrid approach and adopting a UTI both as a feature of the L2R process and as a base ranker. Differences from the architecture described in Fig. 1.1 are highlighted in blue.

retrieved Daoud et al. [2017].

The proposal is to apply the same trained UTI-model used by a top ranker algorithm for the UTI generation in the initial selection of documents, making the model act as a base ranker. The use of UTI values have also the potential to reduce the base ranker computation given that the score function is quite simple, with final scores by just adding the UTI values, which is a low cost computation when compared, for instance, to the cost of computing BM25 score.

Chapter 5

Experimental Protocols

In this chapter we present our experimental setup. We begin by describing the dataset adopted, the baseline methods and the parameter settings.

5.1 Datasets

For our experiments, we used two LETOR 4.0 benchmark datasets [Liu et al., 2007]: 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 [Pang et al., 2017; Fan et al., 2018; Yu et al., 2019; Xu et al., 2019]. 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 [Pang et al., 2017].

LETOR was created by using BM25 [Robertson and Walker, 1994] 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 [Cambazoglu et al., 2010]. 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 [Fan et al., 2018] 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 [Pang et al., 2017; Fan et al., 2018] 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 query-document-term triples in MQ2007. MQ2008, after the merger, contains 2,477 queries, 2,909 terms, 84,834 query-document pairs, and 300,442 query-document-term triples. The details are summarized in Table 5.1.

Table 5.1: Characteristics of the Datasets

Dataset	Queries	Query-document pairs	Query-dependent Features	Standard features
MQ2007	1,692	69,623	26	46
MQ2008(Merge)	2,477	84,834	26	46

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. The details are summarized in Table 5.2.

Table 5.2: Summary of the data extraction from Gov

Dataset	Documents	Query-document-term triples	Posting lists
MQ2007	65,216	236,774	7,454,889
MQ2008(Merge)	78,593	300,442	9,145,089

Table 5.3 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 can be extracted individually, term by term, and 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

Table 5.3: 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.

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

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 [Liu et al., 2007] 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 [Pang et al., 2017]. They divide documents into smaller portions, named *passages*, and calculate 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 [Pang et al., 2017].

When examining the list of features in Table 5.3, 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.

5.2 Baseline Methods

We work with four types of baseline methods for comparison: UTI methods, L2R methods, deep matching model, and hybrid methods.

5.2.1 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 [Costa Carvalho et al., 2012] 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.

UTI-LambdaMART was 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.

5.2.1.1 Term-related features

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, as shown in Table 5.4.

Table 5.4: Impact in MAP results when varying the number of term-Document positional features

Number of positional features	-	1	2	3	4	5
MAP	0.466	0.476	0.480	0.480	0.479	0.480

We decided to add these two positional features after evaluating the results of the experiments with and without the extra features. As shown in figure 5.1, the results with the positional features are superior, the $NDCG@10$ using 22 features was 0.465, whereas it was 0.449 for 20 features.

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 [Page et al., 1999] can be encoded as a feature when computing UTI values and as a feature available to the top ranker when processing queries.

Table 5.5 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 5.3. These two features are extracted from

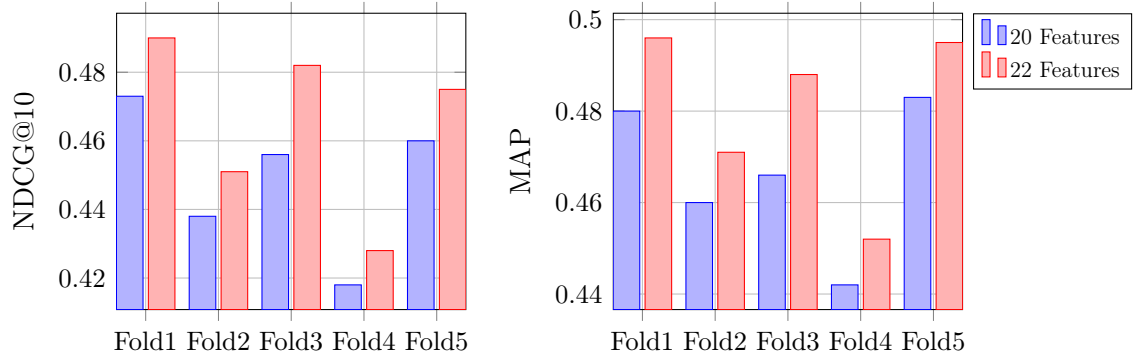


Figure 5.1: Comparison of the UTI-LambdaMART quality results in MQ2007 with and without the two additional positional features for each term-document

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.

Table 5.5: MQ2007 and MQ2008 term-related versions of features 1-15 available in MQ2007 and MQ2008, i.e. features mapping terms to documents.

Seq	Term-Document Features	Data Source
(01-05)	TFs (term frequencies) of the body, anchor, title, URL and whole document	Meta data for LETOR datasets
(06-10)	IDFs (inverse document frequencies) of the body, anchor, title, URL and whole document	Meta data for LETOR datasets
(11-15)	TF*IDFs of the body, anchor, title, URL and whole document	Meta data for LETOR datasets
(16-17)	First and second occurrence position of a term in the document; zero if it does not occur	Original documents of GOV2

When producing UTI values with UTI-LambdaMART and UTI-GP, we adopted features 1-17 mentioned in Table 5.5 and combined them with the remaining query-independent features available at indexing time. We included features 16-20 in Table 5.3, 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 [Callan et al., 1995] before extracting these features. It contains 429 words.

5.2.2 L2R Methods

We chose methods from studies that have achieved the best scores in the existing L2R methods to serve as baselines, the listwise linear feature-based model coordinate ascent (CA) [Donald Metzler, 2007]; MART [Friedman, 2001] and the original *LambdaMART* [Wu et al., 2010].

We use QuickRank¹ [Capannini et al., 2016] to implement LambdaMART, MART and CA. CA was trained using 21 samples, a window size of 10, and a reduction factor of 25. All boosting models, including MART, 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.

The traditional L2R methods adopted (LambdaMART, MART and CA) were tested using the original features available in LETOR, and the full expanded set of 55 LETOR features mentioned in 5.3. Since the deep matching model use passage-level information, we also introduced 9 passage-based features for fair comparison. We applied the full set of features (original+ 9 passage features) on all three learning to rank models presented in Table 6.3,

5.2.3 Deep Matching Methods

Only in recent studies have deep learning models surpassed conventional L2R models. We chose methods from studies that have achieved the best scores to serve as baselines, namely, *DeepRank* [Pang et al., 2017] and the hierarchical neural matching model (*HiNT*) [Fan et al., 2018], the best baselines found in the literature that uses neural network learning methods [Guo et al., 2019];

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

¹<https://github.com/hpclub/quickrank>

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 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 [Fan et al., 2018].

5.2.4 Hybrid Methods

The experiments combining methods that compute UTI values with query-dependent features were conducted using CA, MART and LambdaMART, referred to as HCA, HMART and HLambdaMART. When implementing our hybrid approach, we first produced UTI values using features 1-17 in Table 5.5 and features 16-20 in Table 5.3.

The six document features related to LETOR features (41-46 in Table 5.3) 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 5.3 at query processing times of the hybrid approach. The final set of features adopted at query-time can be seen in Table 5.6. 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.

We use QuickRank [Capannini et al., 2016] to implement HLambdaMART, HMART and HCA. HCA was trained using 21 samples, a window size of 10, and a reduction factor of 25. All boosting models, HMART and HLambdaMART, 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.

Table 5.6: Query-dependent features of LETOR MQ2007 and MQ2008, plus UTI score and a set of nine features related to the positions of query terms, used in the hybrids methods (HCA, HMART and HLambdaMART)

Seq	Query-Document Features	Data Source
(1-5)	LMIR.ABS of the body, anchor, title, URL and whole of document.	LETOR query sets
(6-10)	LMIR.DIR of the body, anchor, title, URL and whole of document.	LETOR query sets
(11-15)	LMIR.JM of the body, anchor, title, URL and whole of document.	LETOR query sets
(16-20)	BM25 of the body, anchor, title, URL and whole of document.	LETOR query sets
(21)	PageRank	LETOR query sets
(22)	Inlink number	LETOR query sets
(23)	Outlink number	LETOR query sets
(24)	Number of slashes in the URL	LETOR query sets
(25)	Length of the URL	LETOR query sets
(26)	Number of child page	LETOR query sets
(27)	Sum of UTIs	UTI Index
(28-36)	Maximum, minimum and average passage-based features taken from TF*IDF, BM25, and LM	LETOR query sets with extra features

Table 5.7 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 [Lai et al., 2013] for each of the tested methods. Thus, we selected the best result for each method and each feature set tested.

Table 5.7: 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, HLambdaMART and HMART)	-	36
L2R Baselines (LambdaMART, MART and CA)	-	55
Neural Baselines (DeepRank and HiNT)	-	raw text

Chapter 6

Experiment Results

In this chapter we present experiments performed to evaluate the ideas proposed by us in this thesis. We provide a detailed description of our results and answer each research question raised in the Chapter 1.

6.1 UTI-LambdaMART for evidence fusion

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 6.1 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.

Table 6.1: 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

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 Costa Carvalho et al. [2012], whereas UTI-LambdaMART took only 5 minutes, just about 2.8% of the training time required for UTI-GP.

We conducted an experiment to complement the assessment of the generalization of the UTI model for new instances. In this experiment, we trained the model using the training and validation set in MQ2007, after generating the final model it is applied

to the MQ2008 test set. We compared this experiment with the quality achieved when all training and testing is performed on the MQ2018 data set.

The quality results presented in Table 6.2 show practically similar results between the two training processes, for instance, with the final model trained on MQ2007 the MAP was 0.485 and with the model trained on MQ2008, the MAP was 0.486. We observe a significant degradation (when the training process is on MQ2007) only on metrics NDCG@1 and P@1. Considering that they are sets with the same features, but with different data context, the results were quite satisfactory.

Table 6.2: Comparing the UTI-LambdaMART in MQ2008 test set using the model trained in MQ2007 and in MQ2008. A significant performance degradation is denoted as (∇).

UTI-LambdaMART	MQ2008						
	NDCG@1	NDCG@5	NDCG@10	P@1	P@5	P@10	MAP
Trained model in MQ2007	0.380 ∇	0.482	0.233	0.454 ∇	0.349	0.249	0.485
Trained model in MQ2008	0.397	0.489	0.237	0.464	0.352	0.249	0.486

6.2 Hybrid architecture evaluation

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 4.2. We here perform experiments to evaluate the quality of results provided by this combination.

When considering the results of the methods presented in Tables 6.1 and 6.3, 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

Table 6.3: Performances of MART, CA, LambdaMART, DeepRank, HiNT and the hybrid methods, combining UTI-LambdaMART with CA (HCA) and LambdaMART (HLambdaMART). We applied the 9 passage-based features in all L2R methods. A significant performance degradation of HLambdaMART is denoted as (∇).

MQ2007							
Model	NDCG@1	NDCG@5	NDCG@10	P@1	P@5	P@10	MAP
MART	0.405 ∇	0.421 ∇	0.446 ∇	0.475 ∇	0.413 ∇	0.383 ∇	0.465 ∇
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	0.441 ∇	0.457	0.482	0.508 ∇	0.452	0.412	0.497
HiNT	0.447 ∇	0.463	0.490	0.515 ∇	0.461	0.418	0.502
HMART	0.457 ∇	0.464	0.491	0.527 ∇	0.456	0.416	0.498
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
MART	0.385 ∇	0.483 ∇	0.230 ∇	0.455 ∇	0.354 ∇	0.250 ∇	0.484 ∇
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	0.406	0.496	0.240	0.482	0.359	0.252	0.498
HiNT	0.415	0.501	0.244	0.491	0.367	0.255	0.505
HMART	0.406 ∇	0.501	0.245	0.482	0.360	0.253	0.500
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

the collections and in terms of all metrics considered. However, Table 6.3 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 in NDCG@1 and P@1 on MQ2007 (Fig. 6.1), 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. Our results show that the automatically learned features in the deep learn-

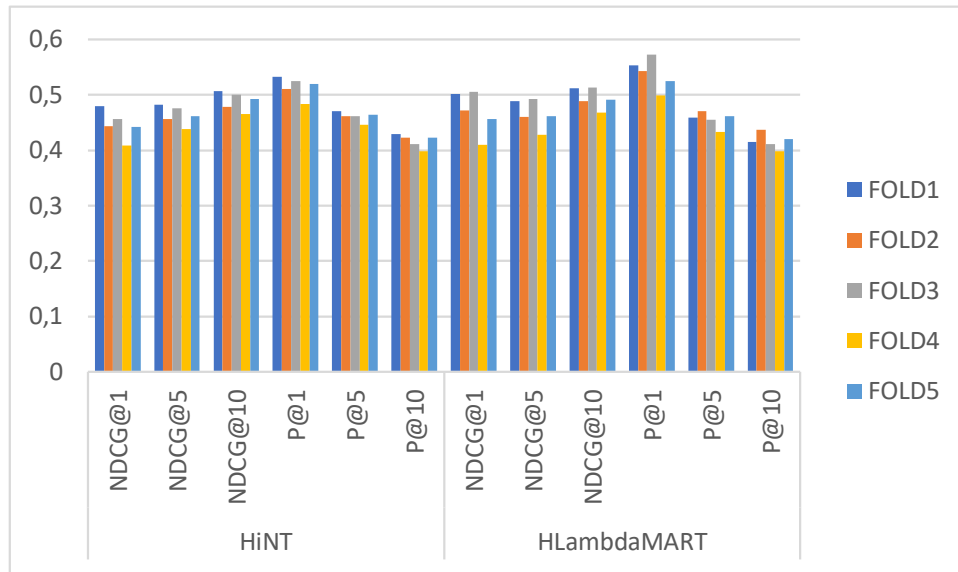


Figure 6.1: Quality results for HiNT and HLambdaMART on MQ2007.

ing baselines adopted in this study, using complex models for training, have quality results on par with the traditional use of features in L2R models.

6.2.1 The impact of UTI on L2R methods

We conducted experiments applying four different sets of features on L2R methods (MART, CA and LambdaMART) and on our hybrids methods (HMART, HCA and HLambdaMART).

Each method and set of features used is detailed below:

- L2R (Original): L2R method applied to the original set of features available in LETOR MQ2007 and MQ2008. Total of 46 features.
- L2R (Original+9 passage): L2R method applied to the original features plus 9 passage-based features proposed by Pang et al. [2017]. Total of 55 features.

- Hybrid (QDF+UTI): Hybrid method. In this set, we consider only the query-dependent features (QDF) available in original LETOR dataset and we added one more feature, the UTI score feature, which is the sum of all UTIs of the query-terms. Total of 27 features.
- Hybrid (QDF+UTI+9 passage): Hybrid method. Here, we consider only the query-dependent features (QDF) available in original set and we added ten (10) more features, the UTI score feature and the 9 passage-based features. Total of 36 features.

When comparing the results of the methods in Tables 6.4, 6.5 and 6.6 we can see that all hybrids models improves significantly the results of the learning methods on MQ2007 and MQ2008. The results of the comparisons between the models are shown below:

Comparison between L2R(Original) model and Hybrid (QDF+UTI) model

The NDCG@10 for HMART was 10% higher than the MART model using all original features of LETOR MQ2007. For the same sets, we can see that there was also an increase in the hybrid models HCA and LambdaMART, comparing to CA and LambdaMART, which were 11.9% and 6.3%, respectively. In all cases, the improvements are statistically significant.

Even applying only 27 features, the results presented by the hybrid models are better than any other L2R model published in the literature in a recent L2R methods survey Guo et al. [2019].

Comparison between L2R(Original+9 passage) model and Hybrid (QDF+UTI+9 passage) model

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. We demonstrated our final results with the 9 passage-base features for the reason that it was considered in the experiments done by the authors of the deeprank and HiNT models.

Comparison between L2R(Original) model and L2R(Original+9 passage) model

NDCG@10 for MART, CA and LambdaMART, in comparison with the experiment using all the original features of the LETOR MQ2007 plus the 9 passage-based features, were 2.1%, 1.3%, and 0.6%, respectively. That was the worst result, it demonstrates that the effort to extract the passage-based features to match with the features used in neural network models does not have a significant impact on the improvement of the L2R models. The same was observed in the MQ2008 dataset,

Table 6.4: Performance comparisons of MART with different set of features on MQ2007 and MQ2008.

MQ2007								
Model	Features	NDCG@1	NDCG@5	NDCG@10	P@1	P@5	P@10	MAP
MART	Original	0.396	0.414	0.437	0.454	0.414	0.379	0.457
MART	Original+ 9 passage	0.405	0.421	0.446	0.475	0.413	0.383	0.465
HMART	QDF+UTI	0.449	0.458	0.486	0.520	0.452	0.414	0.495
HMART	QDF+UTI+ 9 passage	0.457	0.464	0.491	0.527	0.456	0.416	0.498
MQ2008								
Model	Features	NDCG@1	NDCG@5	NDCG@10	P@1	P@5	P@10	MAP
MART	Original	0.372	0.476	0.229	0.439	0.352	0.250	0.477
MART	Original+9 passage	0.385	0.483	0.230	0.455	0.354	0.250	0.484
HMART	QDF+UTI	0.405	0.499	0.242	0.485	0.361	0.251	0.500
HMART	QDF+UTI+9 passage	0.406	0.501	0.245	0.482	0.360	0.253	0.500

Table 6.5: Performance comparisons of CA with different set of features on MQ2007 and MQ2008.

MQ2007								
Model	Features	NDCG@1	NDCG@5	NDCG@10	P@1	P@5	P@10	MAP
CA	Original	0.379	0.410	0.435	0.438	0.405	0.370	0.450
CA	Original+9 passage	0.401	0.417	0.441	0.451	0.412	0.375	0.463
HCA	QDF+UTI	0.464	0.460	0.487	0.535	0.453	0.414	0.499
HCA	QDF+UTI+9 passage	0.466	0.467	0.492	0.536	0.458	0.415	0.501

MQ2008								
Model	Features	NDCG@1	NDCG@5	NDCG@10	P@1	P@5	P@10	MAP
CA	Original	0.372	0.471	0.226	0.437	0.346	0.245	0.474
CA	Original+9 passage	0.367	0.473	0.228	0.439	0.346	0.249	0.475
HCA	QDF+UTI	0.407	0.499	0.243	0.480	0.360	0.252	0.501
HCA	QDF+UTI+9 passage	0.416	0.505	0.246	0.485	0.364	0.253	0.506

Table 6.6: Performance comparisons of LambdaMART with different set of features on MQ2007 and MQ2008.

MQ2007								
Model	Features	NDCG@1	NDCG@5	NDCG@10	P@1	P@5	P@10	MAP
LambdaMART	Original	0.411	0.429	0.451	0.475	0.425	0.385	0.468
LambdaMART	Original+9 passage	0.424	0.432	0.459	0.486	0.427	0.392	0.477
HLambdaMART	QDF+UTI	0.452	0.453	0.481	0.524	0.449	0.410	0.493
HLambdaMART	QDF+UTI+9 passage	0.469	0.466	0.495	0.538	0.456	0.416	0.503

MQ2008								
Model	Features	NDCG@1	NDCG@5	NDCG@10	P@1	P@5	P@10	MAP
LambdaMART	Original	0.373	0.476	0.232	0.453	0.347	0.247	0.478
LambdaMART	Original+9 passage	0.373	0.482	0.233	0.440	0.350	0.248	0.477
HLambdaMART	QDF+UTI	0.389	0.501	0.243	0.468	0.358	0.251	0.496
HLambdaMART	QDF+UTI+9 passage	0.414	0.506	0.247	0.483	0.361	0.253	0.503

6.3 Evaluation the UTI-LambdaMART for base ranker

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. We computed a first ranking with the UTI values produced by UTI-LambdaMART and then adopted the other learning to rank method to reorder the top 20 results produced by the base ranker, providing a final ranking using the whole set of features available. The same procedure was then executed using BM25 as the base ranker.

Fig. 6.2 presents the results of MQ2007. We do not present the results of MQ2008 because the conclusions and results are similar to those of MQ2007.

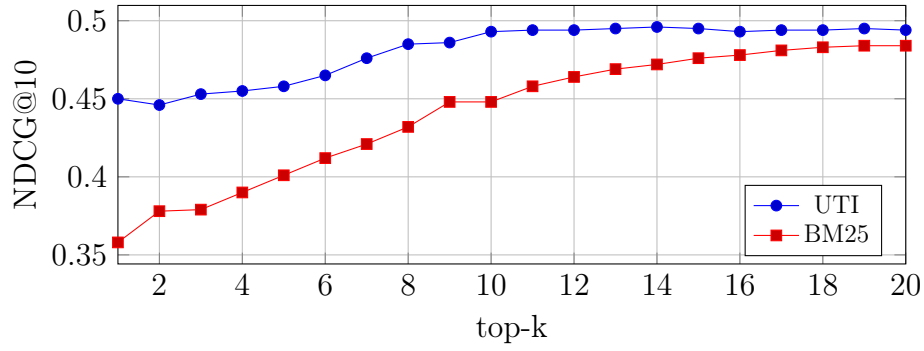


Figure 6.2: Quality of results achieved by HLambdaMART when using UTI-LambdaMART and BM25 as base rankers of MQ2007 for distinct sizes of the top- k results transferred from the base ranker to HLambdaMART.

In Fig. 6.2, 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. For instance, on MQ2007, the NDCG@10 for HLambdaMART using BM25 score as base ranker was 0.448, whereas it was 0.493 for UTI score, thus 10% higher than the one achieved by BM25 on the same top-10 results.

This result shows that UTI-LambdaMART is effective as a base ranker in web 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.

6.4 A performance evaluation

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. 1.1 to our proposal presented in Fig. 4.4, 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. [2017], 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, 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 5.3. 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 5.3.

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 6.7 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 6.7: LambdaMART and HLambdaMART time (seconds) to index and space (MB) when applied to MQ2007 and MQ2008.

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

Table 6.8 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. 6.2.

Table 6.8: LambdaMART and HLambdaMART average times, in milliseconds, to run base ranker (base) and top ranker (top) on MQ2007 and MQ2008.

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

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. [2019]. 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.

6.5 An UTI-Index compression evaluation

RQ5: How much can the UTI index be reduced without loss of quality ?

In our study, 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.

A common approach for improving search engine time and space efficiency is to compress the inverted index Baeza-Yates and Ribeiro-Neto [2011]. A compressed index will take less storage space and thus require less I/O operations, generally yielding a gain in performance. In our proposal, we can exploit UTI-LambdaMART to achieve a much smaller index with low impact on its construction time. In addition, our approach will still allow the complementary application of standard compression methods, thus possibly yielding even greater gains.

Our proposal for index size reduction consists of, quite simply, truncating the decimal places in the UTI values generated. In Algorithm 1, every time the function Ft_n is evaluated, only the first d decimal places are considered. A smaller value for d means less bytes required for storage per UTI. Interestingly, even though this will limit the precision of the document score calculation, UTI-LambdaMART will still try to optimize the final ranking, thereby minimizing the impact that this loss of information might have on the quality of the results.

Our process of 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 6.9 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* Elias [1975]. 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.

Table 6.9: 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
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

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.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, we propose UTI-LambdaMART, a modified LambdaMART ranking algorithm designed to generate UTI scores for each term and document pair at indexing time. Our method surpasses the UTI-GP baseline in quality and training time. The time required to training the UTI-LambdaMART is 2.8% of the time to training the UTI-GP and the quality of its generated model is statistically superior.

Passage-based features are extensively explored by deep matching models. Thus, we also explored information about the term’s position in the document as a feature and our experiments showed that above 2 positions there is no improvement in the quality of the model. On MQ2007, the MAP of the UTI-LambdaMART model was 0.466 and using this two positional features the map is 0.480.

Besides UTI-LambdaMART, we also propose a hybrid methods that combine UTI-LambdaMART with other L2R methods, producing new L2R methods named HCA, HMART and HLambdaMART. Our experimental results show that HCA, HMART and HLambdaMART produce results on par with those of the state-of-the-art L2R method HiNT, which adopts a neural network model. Further, for P@1 and

NDCG@1 the results produced by our methods were better than the ones achieved by HiNT. The result offers a significant reduction in the number of features fetched and processed at query processing times, reducing the from 55 to 36 in collections MQ2007 and MQ2008. Furthermore, our architecture resulted in a reduction in the time needed to process queries, which is an important property for real-world search systems.

To complement our proposal, we investigated the UTI-LambdaMART as a base ranker and the UTI-LambdaMART as the main ranker. In the first case, the UTI values produce high-quality first cut rankings actually closer to the final ranking produced by the best L2R methods. This reduces the number of documents inspected by the top ranker, which ultimately allows faster query processing when compared to a system that adopts BM25 as the base ranker. In the second case, UTI-LambdaMART is directly applied as the top ranker. In this case, it does not require a learning model at a query processing time, yielding an extremely simple ranking strategy. Considering the features only available at the indexing time, the UTI-LambdaMART produces quality rankings compared with other L2R methods. Unlike previous methods, UTI-LambdaMART is a simple and lightweight ranking method that does not fetch several features at a query processing time.

Finally, we investigated an alternative of reducing the storage space when representing UTI values. The reduction is achieved by truncating the UTI numbers produced, mapping them to lower precision numbers, compressed using methods commonly applied to search indexes. We show that this strategy may reduce the average number of bits required to store each UTI value from 18 to 6 bits, without significant loss in the quality of results.

7.2 Future Work

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 Chen et al. [2017]; Lucchese et al. [2018a]; Gallagher et al. [2019].

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) Sousa et al. [2019]. Their usage when combined with UTI values represents a challenge, since when computing UTI values, the learning is performed at indexing times.

Bibliography

- Anh, V. N. and Moffat, A. (2002). Impact transformation: Effective and efficient web retrieval. In *Proceedings of 25th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'02)*, pages 3--10, New York, NY, USA. ACM.
- Anh, V. N. and Moffat, A. (2005). Simplified similarity scoring using term ranks. In *Proceedings of 28th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'05)*, pages 226--233, New York, NY, USA. ACM.
- Anh, V. N., Wan, R., and Moffat, A. (2008). Term impacts as normalized term frequencies for bm25 similarity scoring. In *International Symposium on String Processing and Information Retrieval*, pages 51--62. Springer.
- Baeza-Yates, R. and Ribeiro-Neto, B. (2011). *Modern Information Retrieval*. Addison-Wesley Professional, Boston, MA, USA, 2 edition.
- Burges, C., Ragno, R., , and Le, Q. (2006). Learning to rank with nonsmooth cost functions. In *In Advances in Neural Information Processing Systems*, pages 392--402. MIT Press.
- Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. (2005). Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89--96.

- Burges, C. J. C. (2010). From RankNet to LambdaRank to LambdaMART: An overview. Technical report, Microsoft Research.
- Callan, J. P., Croft, W. B., and Broglio, J. (1995). Trec and tipster experiments with inquiry. *Information Processing & Management*, 31(3):327--343.
- Cambazoglu, B. B., Zaragoza, H., Chapelle, O., Chen, J., Liao, C., Zheng, Z., and Degenhardt, J. (2010). Early exit optimizations for additive machine learned ranking systems. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining (WSDM'10)*, pages 411--420. ACM.
- Capannini, G., Lucchese, C., Nardini, F. M., Orlando, S., Perego, R., and Tonellotto, N. (2016). Quality versus efficiency in document scoring with learning-to-rank models. *Information Processing & Management*, 52(6):1161--1177.
- Chen, R.-C., Gallagher, L., Blanco, R., and Culpepper, J. S. (2017). Efficient cost-aware cascade ranking in multi-stage retrieval. In *Proceedings of 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'17)*, pages 445--454, New York, NY, USA. ACM.
- Cormack, G. V. and Lynam, T. R. (2007). Validity and power of t-test for comparing map and gmap. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 753--754.
- Costa Carvalho, A. L., Rossi, C., Moura, E. S., Silva, A. S., and Fernandes, D. (2012). Lepref: Learn to precompute evidence fusion for efficient query evaluation. *Journal of the American Society for Information Systems and Technology (JASIST)*, 63(7):1383--1397.
- Craswell, N., Robertson, S., Zaragoza, H., and Taylor, M. (2005). Relevance weighting for query independent evidence. In *Proceedings of the 28th annual international ACM*

- SIGIR conference on Research and development in information retrieval*, pages 416–423.
- Dang, V., Bendersky, M., and Croft, W. B. (2013). Two-stage learning to rank for information retrieval. In *Advances in Information Retrieval*, pages 423–434. Springer.
- Daoud, C. M., de Moura, E. S., Fernandes, D., da Silva, A. S., Rossi, C., and Carvalho, A. (2017). Waves: a fast multi-tier top-k query processing algorithm. *Information Retrieval*, 20(3):292–316. ISSN 1573-7659.
- de Almeida, H. M., Gonçalves, M. A., Cristo, M., and Calado, P. (2007). A combined component approach for finding collection-adapted ranking functions based on genetic programming. In *Proceedings of 30th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR’07)*, pages 399–406. ACM.
- Donald Metzler, W. B. C. (2007). Linear feature-based models for information retrieval. *Information Retrieval*, 10(3):257–274.
- Elias, P. (1975). Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2):194–203.
- Fan, Y., Guo, J., Lan, Y., Xu, J., Zhai, C., and Cheng, X. (2018). Modeling diverse relevance patterns in ad-hoc retrieval. In *Proceedings of 41st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR’18)*, pages 375–384. ACM.
- Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. In *The Annals of Statistics*, pages 1189–1232.
- Gallagher, L., Chen, R.-C., Blanco, R., and Culpepper, J. S. (2019). Joint optimization of cascade ranking models. In *Proceedings of the 12th ACM International Conference*

- on Web Search and Data Mining (WSDM'19)*, pages 15--23, New York, NY, USA. ACM.
- Guo, J., Fan, Y., Pang, L., Yang, L., Ai, Q., Zamani, H., Wu, C., Croft, W. B., and Cheng, X. (2019). A deep look into neural ranking models for information retrieval. *Information Processing & Management*, page 102067.
- Huang, P.-S., He, X., Gao, J., Deng, L., Acero, A., and Heck, L. (2013). Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2333--2338.
- Järvelin, K. and Kekäläinen, J. (2002). Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (ACM TOIS)*, 20(4):422--446. ISSN 1046-8188.
- Ji, S., Shao, J., and Yang, T. (2019). Efficient interaction-based neural ranking with locality sensitive hashing. In *Proceedings of the 28th International World Wide Web Conference (WWW'19)*, pages 2858--2864, New York, NY, USA. ACM.
- Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133--142. ACM.
- Lai, H.-J., Pan, Y., Tang, Y., and Yu, R. (2013). Fsmrank: Feature selection algorithm for learning to rank. *IEEE transactions on neural networks and learning systems*, 24(6):940--952.
- Li, H. (2011). *Learning to rank for information retrieval and natural language processing*, volume 1. Morgan & Claypool Publishers.
- Liu, T.-Y. (2011). *Learning to rank for information retrieval*. Springer Science & Business Media.

- Liu, T.-Y., Xu, J., Qin, T., Xiong, W., and Li, H. (2007). Letor: Benchmark dataset for research on learning to rank for information retrieval. In *SIGIR 2007 workshop on learning to rank for information retrieval*, pages 3--10.
- Lucchese, C., Nardini, F. M., Orlando, S., Perego, R., Silvestri, F., and Trani, S. (2016). Post-learning optimization of tree ensembles for efficient ranking. In *Proceedings of 39th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'16)*, pages 949--952, New York, NY, USA. ACM.
- Lucchese, C., Nardini, F. M., Orlando, S., Perego, R., Silvestri, F., and Trani, S. (2018a). X-cleaver: Learning ranking ensembles by growing and pruning trees. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 9(6):62.
- Lucchese, C., Nardini, F. M., Perego, R., Orlando, S., and Trani, S. (2018b). Selective gradient boosting for effective learning to rank. In *Proceedings of 41st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'18)*, pages 155--164. ACM.
- Mitra, B., Diaz, F., and Craswell, N. (2017). Learning to match using local and distributed representations of text for web search. In *Proceedings of the 26th International World Wide Web Conference (WWW'17)*, pages 1291--1299, Republic and Canton of Geneva, Switzerland.
- Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.
- Pang, L., Lan, Y., Guo, J., Xu, J., Xu, J., and Cheng, X. (2017). Deeprank: A new deep architecture for relevance ranking in information retrieval. In *Proceedings of the 26th ACM International Conference on Information and Knowledge Management (CIKM'17)*, pages 257--266. ACM.

- Robertson, S. (2000). Evaluation in information retrieval. In *European Summer School on Information Retrieval*, pages 81--92. Springer.
- Robertson, S. E. and Walker, S. (1994). Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of 17th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'94)*, pages 232--241. Springer-Verlag New York, Inc.
- Saraiva, P. C., Silva de Moura, E., Ziviani, N., Meira, W., Fonseca, R., and Ribeiro-Neto, B. (2001). Rank-preserving two-level caching for scalable search engines. In *Proceedings of 24th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'01)*, pages 51--58. ACM.
- Silva, I., Ribeiro-Neto, B., Calado, P., Moura, E., and Ziviani, N. (2000). Link-based and content-based evidential information in a belief network model. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '00, page 96--103, New York, NY, USA. Association for Computing Machinery.
- Silva, T. P. C., de Moura, E. S., Cavalcanti, J. M. B., da Silva, A. S., de Carvalho, M. G., and Gonçalves, M. A. (2009). An evolutionary approach for combining different sources of evidence in search engines. *Information Systems*, 34(2):276--289.
- Smucker, M. D., Allan, J., and Carterette, B. (2007). A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings of the 16th ACM Conference on information and knowledge management (CIKM'07)*, pages 623--632. ACM.
- Smucker, M. D., Allan, J., and Carterette, B. (2009). Agreement among statistical significance tests for information retrieval evaluation at varying sample sizes. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and De-*

- velopment in Information Retrieval*, SIGIR '09, page 630–631, New York, NY, USA. Association for Computing Machinery.
- Sousa, D. X., Canuto, S., Gonçalves, M. A., Rosa, T. C., and Martins, W. S. (2019). Risk-sensitive learning to rank with evolutionary multi-objective feature selection. *ACM Transactions on Information Systems (ACM TOIS)*, 37(2):24:1--24:34. ISSN 1046-8188.
- Tax, N., Bockting, S., and Hiemstra, D. (2015). A cross-benchmark comparison of 87 learning to rank methods. *Information processing & management*, 51(6):757--772.
- Wei, Z., Xu, J., Lan, Y., Guo, J., and Cheng, X. (2017). Reinforcement learning to rank with markov decision process. In *Proceedings of 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'17)*, pages 945--948, New York, NY, USA. ACM.
- Westerveld, T., Kraaij, W., and Hiemstra, D. (2001). Retrieving web pages using content, links, urls and anchors. In *TREC*, volume 1, pages 663--672.
- Wu, Q., Burges, C. J., Svore, K. M., and Gao, J. (2010). Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3):254--270.
- Xia, F., Liu, T.-Y., Wang, J., Zhang, W., and Li, H. (2008). Listwise approach to learning to rank: theory and algorithm. In *Proc. of the 25th international conference on Machine learning*, pages 1192--1199. ACM.
- Xu, B., Lin, H., Lin, Y., and Xu, K. (2019). Incorporating query constraints for autoencoder enhanced ranking. *Neurocomputing*, 356:142--150.
- Xu, J. and Li, H. (2007). Adarank: a boosting algorithm for information retrieval. In *Proceedings of 30th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'07)*, pages 391--398. ACM.

- Yu, H.-T., Jatowt, A., Joho, H., Jose, J. M., Yang, X., and Chen, L. (2019). Wassrank: Listwise document ranking using optimal transport theory. In *Proceedings of the 12th ACM International Conference on and Data Mining (WSDM'19)*, pages 24--32. ACM.
- Zamani, H., Dehghani, M., Croft, W. B., Learned-Miller, E., and Kamps, J. (2018). From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM'18)*, pages 497--506, New York, NY, USA. ACM.