Learning to Adaptively Rank Document Retrieval System Configurations

Romain Deveaud, Josiane Mothe, Md Zia Ullah, Jian-Yun Nie

Learning to Adaptively Rank Document Retrieval System Configurations

ROMAIN DEVEAUD, JOSIANE MOTHE, and MD ZIA ULLAH, IRIT, UMR5505 CNRS, Université de Toulouse, France
JIAN-YUN NIE, Université de Montréal, Canada

Modern Information Retrieval (IR) systems have become more and more complex, involving a large number of parameters. For example, a system may choose from a set of possible retrieval models (BM25, language model, etc.), or various query expansion parameters, whose values greatly influence the overall retrieval effectiveness. Traditionally, these parameters are set at a system level based on training queries, and the same parameters are then used for different queries. We observe that it may not be easy to set all these parameters separately, since they can be dependent. In addition, a global setting for all queries may not best fit all individual queries with different characteristics. The parameters should be set according to these characteristics. In this article, we propose a novel approach to tackle this problem by dealing with the entire *system configurations* (i.e., a set of parameters representing an IR system behaviour) instead of selecting a single parameter at a time. The selection of the best configuration is cast as a problem of ranking different possible configurations given a query. We apply learning-to-rank approaches for this task. We exploit both the query features and the system configuration features in the learning-to-rank method so that the selection of configuration is query dependent. The experiments we conducted on four TREC ad hoc collections show that this approach can significantly outperform the traditional method to tune system configuration globally (i.e., grid search) and leads to higher effectiveness than the top performing systems of the TREC tracks. We also perform an ablation analysis on the impact of different features on the model learning capability and show that query expansion features are among the most important for adaptive systems.

Additional Key Words and Phrases:

Information systems, information retrieval, learning to rank, retrieval system parameters, adaptive information retrieval, query features, data analytics

This article is supported by ANR Agence nationale de la recherche CAAS project, ANR-10-CORD-001 01.

Authors' addresses:

R. Deveaud, J. Mothe, and Md Z. Ullah, IRIT, UMR5505 CNRS, Université de Toulouse, Institut de Recherche en Informatique de Toulouse (IRIT), 118, Route de Narbonne, Toulouse, 31062, France;
emails: romain. deveaud@gmail.com, {Josiane.Mothe, mdzia.ullah}@irit.fr; J.-Y. Nie,

Université de Montréal, Département d'informatique et recherche opérationnelle, C.P. 6128, Succ Centre-Ville, Montréal, Québec, Canada; emails: nie@iro.umontreal.ca.

https://doi.org/10.1145/3231937

1 INTRODUCTION

Modern Information Retrieval (IR) systems involve more and more complex operations, which require setting a large number of parameters. For example, at the very basic preprocessing level, we have to choose among different options of word stemming. Then a retrieval model should be chosen. This latter often involves a set of parameters as well—language models require smoothing parameters and BM25 has another set of parameters. Finally, the pseudo-relevance feedback step requires yet another set of parameters: the number of expansion terms to be added to the query, their weighting scheme, the number of feedback documents to consider, and so on [11, 13]. Over the years, and through evaluation forums such as TREC,[1] CLEF,[2] and NTCIR,[3] the IR community has produced an abundant field of knowledge, however scattered in the literature, on setting the appropriate values of these parameters to optimise the performance of the retrieval systems. For example, we know that the number of pseudo-relevance feedback documents used in IR experiments typically varies between 10 and 50, and the number of expansion terms is in the range of 10 to 20 [16, 36]. BM25 or a language model is often chosen, and they are believed to be effective on most test collections. When a specific retrieval method involves some parameters (e.g., the parameters related to query expansion), one typically tunes them on a set of training queries to maximise the global effectiveness. The typical method for parameter tuning is through grid search [73]: A set of possible values is defined for each parameter, and grid search determines the best value for each parameter to maximise the effectiveness of the retrieval system on a set of training queries.

To be more robust, one also test different settings on several test collections. For example, Reference [80] analysed the influence of the smoothing function in Language Modelling (LM) on several test collections, and some specific range of the smoothing parameter is recommended. Alternatively, it is possible to optimize the studied parameter value using a collection and observe its effects on other collections [35], which is a form of transfer learning.

These methodologies for parameter tuning assume that the same selected parameters would fit all the queries. In practice, even if the selected system configuration is the best for a set of queries, it has been often observed that it behaves differently on different queries: It may excel on some queries while failing miserably on some others [1, 37, 60]. This fact indicates a critical problem in the usual way to set system parameters: It is done once and for all queries. It is desirable that we choose the appropriate parameters for each query at hand, thus avoiding the problem of the one-size-fits-all solution.

There is an abundant literature on the effects of individual system parameters on retrieval results. Indeed, for any new method proposed, it is required that an analysis is made in depth to evaluate the effect of parameter setting [83], e.g., how the method behaves along with the changes of its inherent parameters. However, there have been few studies trying to determine the parameters automatically for a given query.

There are also only a few descriptive analyses of cross parameter effects [9, 23, 29], which examine the results and the effects of various parameter settings. In Reference [23], the authors analysed the influence of indexing and retrieval parameters on retrieval effectiveness; while the authors of Reference [9] analysed an even larger set of parameters. The authors of Reference [29] analysed the correlation between effectiveness measures and system parameters. However, none of these studies attempted to determine automatically the best parameters at the query level for new queries.

[1] Text REtrieval Conference; http://trec.nist.gov.
[2] Conference and Labs of the Evaluation Forum; http://www.clef-initiative.eu.
[3] NII Testbeds and Community for Information Access Research; http://research.nii.ac.jp/ntcir/index-en.html.

The study presented in this article is built on the results and conclusions of the previous descriptive analysis studies but moves a step further by performing a predictive analysis: We investigate how system parameters can be set to fit a given query, i.e., a query-dependent setting of system parameters. We assume that some parameters of the system can be set on the fly at querying time, and a retrieval system allows us to set different values for the parameters easily. This is indeed the case for most IR systems nowadays. Retrieval platforms such as Terrier[4] [61], Lemur[5] [70], or Lucene[6] [53] allow us to set parameters for the retrieval step. For example, one may choose between several retrieval models (e.g., BM25, language models), different query expansion schemes, and so on. We target this group of parameters that can be set at query time. In contrast, we assume that an IR system has already built an index that cannot be changed easily. For example, it would be difficult to choose between different stemmers at query time, unless we construct several indexes using different stemmers. We exclude these parameters that cannot be set at query time in this study.

The problem we tackle in this article is query-dependent parameter setting. A complete set of parameters of a system (that can be set at query time) forms a *system configuration*. In theory, some parameters such as the number of expansion terms can vary in a large range. However, the existing studies provide sufficient evidence that there is a smaller preferred range for them. Therefore, we only allow the parameters to vary in a much smaller range, based on our prior knowledge of the literature to reduce the complexity. Having all the parameters grouped in system configurations, our problem then boils down to selecting the best system configuration, among all the possible ones, for a given query. This problem could be seen as a performance prediction problem: Given a query, we predict the performance of a configuration and we pick up the configuration corresponding to the highest predicted performance. However, the prediction of system performance is known to be hard. Alternatively, the problem can be cast as a configuration ranking problem: We could rank the configurations according to their expected performance on a given query and we choose to use the first configuration for the query. This approach does not require us to predict system performance explicitly. It could be solved using a learning-to-rank [48] approach. This is what we propose in this article. Our utilisation of learning-to-rank (LTR) in this article is different from its traditional utilisation for document ranking, where features are extracted for document–query pairs. In our case, for each query, we aim at ranking system configurations. Thus in our setting, features are extracted from query–configuration pairs. Given the number of parameters and their possible domains, our candidate space is formed of tens of thousands of possible system configurations, each of which sets a specific value for each of the system parameters.

In our approach, the LTR models are trained to rank configurations with respect to a target effectiveness metric (such as nDCG@k), thus emphasising the importance of ranking "good" system configurations higher in the ranked list. Our final choice is the configuration ranked at first. Compared to tuning each parameter at a time, this approach has the advantage of taking all the system parameters into account at the same time, which allows them to influence each other implicitly within the same configuration. However, we do not address the problem of mutual influence among parameters explicitly in this study. Our main focus is put on making a query-dependent setting of the system configuration using LTR. To our knowledge, this problem has not been studied extensively before. In particular, no attempt has been made using LTR approaches.

[4] http://terrier.org/.
[5] https://www.lemurproject.org/indri.php.
[6] https://github.com/lucene4ir/lucene4ir.

The main contributions of this article are as follows:

- We propose a novel approach to set IR system parameters based on the learning-to-rank technique by exploiting query and configuration features,
- we develop a method that adaptively selects the best system parameters according to the query, and show that this approach is effective and feasible in practice,
- we carry out extensive experiments on several standard test collections, which show the superiority of our proposed method over the state-of-the-art methods.

2 RELATED WORK

We aim at solving the problem of automatic parameter selection by using learning-to-rank techniques in an original way. Our work is thus at the intersection of several trending topics in IR including parameter analysis, adaptive IR systems, and learning-to-rank.

2.1 Parameter Analysis

A few pieces of work performed descriptive analysis to understand better the results obtained with various system configurations.

In Reference [6], the authors analysed past TREC results on the TREC-6 collection—the average precision values and systems—on a per query basis. They carried out analysis of variance, cluster analysis, rank correlation, beadplot, multidimensional scaling, and item response analysis. When considering an analysis of variance, they focused on two factors only: topic and system, and one performance measure (average precision). They also used cluster analysis to cluster systems according to AP they obtained on the various queries. However, the authors stated that the results were inconclusive and that none of these methods had yielded any substantial new insights.

Dinçer [27] compared performances of various search strategies by means of principal component analysis (PCA). The authors showed the PCA method can reveal implicit performance relations among retrieval systems across topics. They considered 100 topics from TREC 12 Robust collection, the 44 submitted runs, and showed that PCA can highlight the peculiarities of some runs and of some topics.

Some applications have also been suggested based on the results of such analysis. Mizzaro and Robertson [56] used data analysis to distinguish good systems from bad systems by defining a minimal subset of queries. Based on the analysis, the authors concluded that "easy" queries perform the best in this task. Bigot et al. [8] used the results of the similar type of analysis as Dinçer but using Benzecri's χ^2 correspondence analysis to suggest a system fusion method.

In the above studies, the authors did not analyse system parameters nor query/topic features. In the studies described below, these elements have been analysed.

Compaoré et al. [23] analysed indexing parameters and retrieval models to determine which parameters significantly affect search engine performance. They found that the retrieval model is the most important parameter to be tuned to improve the performance. More interestingly, they showed that the most significant parameters depend on the topic difficulty. While the retrieval model remains the most important parameter for easy queries, the query expansion model is the most important for hard queries. The study in Reference [9] was significantly enlarged with regard to the number of parameters: Eighty thousand different system configurations in total have been tested, covering 4 different stemming algorithms, 21 retrieval models, 7 topic fields in queries, 6 query expansion models, and other query expansion parameters. These configurations have been applied to the 100 topics from TREC7-8 adhoc collections. The authors concluded that the parameters that have the largest impact on system effectiveness across queries are the retrieval model and

the expansion model. However, the best parameters change according to queries. This study provides clear experimental evidence that the parameters should be set in a query-dependent manner.

Also using various system configurations, Reference [29] analysed the correlation among the effectiveness measures of systems configured by combining system parameters including 6 different stop lists, 6 types of stemmers, 7 flavours of n-grams, and 17 IR models. He considered the correlation between different evaluation measures and found that they change depending on the system configuration.

Fusi and Elibol tackled the problem of robustness of system setting: They suggested to use Probabilistic Matrix Factorization to automatically identify high-performing pipelines across a wide range of datasets [34].

All these studies are limited to the descriptive analysis: Given the parameters and results, these studies focused on understanding the impact of parameters on system performance and their correlations. In this article, we focus rather on a predictive analysis: We aim at constructing prediction models from the relationships among query–system–performance triplets. These models can predict the best configuration for a given query. Our study is a natural extension of the previous ones and is built on the latter.

2.2 Adaptive Systems

Many factors affect system performance. These include factors related to the query. It is thus intuitive to design adaptive systems that can fit the system to the characteristics of the query. The characteristics of the user can also impact the specific choice of system parameters. The adaptation to users has been implemented in most search engines. Typically, the past interactions with the user are leveraged to build a user profile so that the retrieved documents or recommended items can better fit the user [71]. In this study, we only consider system adaptation to the query.

Selective query expansion (QE) is probably the most studied topic among the adaptive techniques. The motivation of selective QE stems from the observation that pseudo-relevance feedback QE improves the effectiveness on some queries but deteriorates on some others. In selective QE, the system decides whether QE should be applied or not [25]. This is based on feature analysis and learning models: Queries are characterized by a set of features and the learning is based on a set of examples for which the impact of query expansion decision is known. The system learns to make a binary decision on the use of QE. It is shown that the trained model is able to selectively apply QE to new queries [79].

In Reference [13], adaptive QE is cast as a problem of selecting appropriate expansion terms from pseudo-relevant feedback documents. The features used to make the selection include query-dependent features, making the selection query dependent. More recently, Xu et al. [77] proposed a learning-to-rank based query expansion. It re-ranks a set of potential terms for expanding query by exploiting the top retrieved documents and the collection statistics.

In the studies on selective QE mentioned above, one focused on determining the useful expansion terms rather than a system configuration. While the selection of appropriate expansion terms is shown to be important, we believe that selecting the appropriate expansion parameters is equally important, and it is complementary to the studies mentioned above.

A study that is related to ours is in Reference [9], in which, after a descriptive analysis on how parameters affect effectiveness, the authors presented a method to automatically decide which system configuration should be used to process a query. The meta-system learns the best system configuration to use on a per query basis similar to what we intend to do. This method was developed for the case of repeated queries. In comparison, we investigate the problem of configuration selection for any query, including new queries that have never been seen before. Therefore, our scope is larger, since the model learns from query features rather than from past queries.

Another related work to ours is Reference [59], which proposed a per-parameter learning (PPL) method to predict the best value for a set of system parameters for optimal retrieval. It learns a classification model for each system parameter and uses this classifier to predict the best value of that parameter for an unseen query. However, the separate tuning for each parameter has the potential drawback that it will fail to cope with the possible dependencies among the parameters. In practice, many parameters are dependent. For example, the number of expansion terms to be used may depend on the number of feedback documents considered. In our study, instead of learning for each parameter separately, we learn to rank the complete configurations combining all system parameters. The parameters may interact within the same configuration. This is a way to implicitly cope with the dependencies among parameters.

A key difference between our study and the previous ones is the technique used to make the selection. We use LTR techniques, which have not been used before for this task. Our hypothesis is that configurations can effectively be ranked according to their characteristics facing a given user's query and that we can train a model to rank configurations for any query.

We carried out a preliminary study on learning-to-rank system configurations in a short article [26], in which the principle was laid down, and the first set of experiments were presented. This article presents a substantial extension of our previous article.

2.3 Learning to Rank

Nowadays, commercial live search engines reportedly utilise ranking models that have been learned over thousands of features. Learning-to-rank is a document ranking technique [48] that surfaced when researchers started to use machine-learning algorithms to learn an appropriate combination of features into effective ranking models. It is composed of a training phase and a prediction phase. The training phase aims at learning a model that optimizes the document ranking. The capability of the model to correctly rank documents for new queries is evaluated during the prediction phase on test data. Interested readers can find more details about the techniques of LTR for IR in Reference [49].

In learning-to-rank, the training data consist of query–document pairs represented by feature vectors. These features include those that describe the query (e.g., TF and IDF of query terms, query length), the document (e.g., TF and IDF of document terms, document length, PageRank score), as well as the relationship between them (e.g., BM25 score). A relevance judgement between the query and the document is given as the ground truth. For example, the Letor 3.0 and Letor 4.0 collections [63] provide around one hundred query–document pair features that have been shown to be useful for learning the document ranking function [49]. An exhaustive list of these features can be found at http://www.microsoft.com/en-us/research/project/mslr. However, the set of features can be reduced for this purpose to about 10 features without losing too much accuracy, depending on the collection [44, 45].

Different approaches have been developed to solve the learning-to-rank problem in IR, including *pointwise*, *pairwise*, and *listwise* approaches.

In the *pointwise* approach, each instance is a vector of features x_i, which represents a query–document pair. The ground truth can be either a relevance score $s \in \mathbb{R}$ or a class of relevance (e.g., "not relevant," "partially relevant," and "highly relevant"). In the former case, learning-to-rank can be solved as a regression problem while in the latter case, it is considered as a classification problem or as an ordinal regression problem, depending on whether there is an ordinal relationship between the classes of relevance [45].

In the *pairwise* approach or preference learning [33], each instance is a pair of feature vectors (x_i, x_j) for a given query q. The ground truth is given as a preference $y \in \{-1, 1\}$ between the two documents and is also considered as a classification problem. Various algorithms have been

proposed such as RankNet [12] based on neural networks, RankBoost [31] based on boosting or RankSVM-Primal [18], and RankSVM-Struct [42] based on SVM.

Finally, the *listwise* approach considers the whole ranked list of documents as the instance of the algorithm and employs a listwise loss function to train the model. Several articles have suggested that listwise learning-to-rank algorithms, including ListNet [14], ListMLE [76], and LambdaMart [75] perform better than pairwise approach on the problem of document ranking.

In practice, learning-to-rank is often used to re-rank a list of top-ranked documents for a given query to promote the documents that the ranking model learnt as relevant [52]. The list of documents to be re-ranked, from a few hundred to a few thousand [17, 52, 63], are initially ranked by a standard retrieval model such as BM25 [66] or Language Modelling [41, 62].

A key advantage of LTR models, compared to the traditional models, is its ability to incorporate any type of feature related to query–document pairs. In principle, the same methods can apply to rank any objects, provided that similar features can be extracted. In other words, the framework is flexible enough to be used to rank system configurations for a query. The difference is that, for a given query, we rank a set of system configurations rather than documents. Another difference with the traditional utilisations of LTR is that we are not interested in determining the whole ranking of different system configurations. We are only interested in selecting the first one. It is thus possible that an LTR method that works well for document ranking works poorly for configuration ranking. Indeed, we will observe this phenomenon in our experiments.

The problem of selecting a good system configuration for a query can be considered as a prediction (or regression) problem: Given a candidate configuration, we try to predict its effectiveness on a query. It can also be cast as a ranking problem: We have a large set of choices, and we have to rank them so that the best one can be selected. This forms a larger category of problem, to which the principle of LTR naturally applies. An LTR approach to rank system configuration is more general than the selective QE that decides whether to use QE or not. In our case, we have much more choices than just a binary one. Moreover, the binary decision of selective QE is included in our approach as a special case, because "no expansion" is among the possible choices. Our LTR approach is also more suitable to the problem of selecting system configurations than a selection on the per parameter basis, because we do not assume independence between parameters.

The approach is also feasible in practice, because we can derive a large set of training examples from assessed queries. For each query, it is possible to obtain a system performance measure using each of the system configurations. Of course, the more assessed queries we have, the more we could expect that the trained LTR model will be able to deal with new queries. However, in our experiments, we will show that a small number of assessed queries (100) can already result in highly effective models.

What is unknown is which features are useful for configuration selection. Therefore, in our study, we will extract all features that may deem useful and test their impact on the final results. We also do not know if listwise models will be the best methods as for document ranking. So we will test the three types of LTR models.

3 LEARNING-TO-RANK SYSTEM CONFIGURATIONS

In this section, we describe our proposed approach for ranking system configurations based on the learning-to-rank techniques for IR [48]. We will first formulate the problem in the next subsection. Then the features will be specified. Finally, the training of LTR models will be described.

3.1 Model

We assume that an IR system involves a set $\mathbf{P}$ of parameters. Each parameter $p_i \in \mathbf{P}$ can take a value from its domain D_i. Therefore, we have at most $\prod_i |D_i|$ possible configurations (without

Table 1. Description of the System Parameters That We Considered When Constructing System Configurations

Parameter	Description	Values[7]
Retrieval model	21 different retrieval models	BB2, BM25, DFRee, DirichletLM, HiemstraLM, InB2, InL2, *JsKLs, PL2, DFI0, XSqrAM, DLH13, DLH, DPH, IFB2, TFIDF, InexpB2, DFRBM25, LGD, LemurTFIDF, and InexpC2.*
Expansion model	7 query expansion models	nil[8], KL, Bo1, Bo2, KLCorrect, Information, and KLComplete.
Expansion documents	6 variants of number of documents used for query expansion	2, 5, 10, 20, 50, and 100.
Expansion terms	5 variants of number of expansion terms	2, 5, 10, 15, and 20.
Expansion min-docs	5 variants of minimal number of documents an expansion term should appear in	2, 5, 10, 20, and 50.

Because of the enormous number of configurations, we selected the 7 retrieval models for which the results where the best in average over the topics and collections when using the initial queries and we then combined these 7 retrieval models with the expansion variations. For the other 14 retrieval models (italic font in the table), we only kept the configuration with no expansion.

considering the fact that some configurations are impossible). This number could be very large, given the quite large number of system parameters used in modern IR systems and their possible values. Moreover, the large number of system parameters could be mutually inclusive [30], which makes the problem of choosing a globally optimised set of parameters challenging.

We cast the problem of setting the optimal system parameter value as a configuration ranking problem using a learning-to-rank approach, where, for each query, we rank system configurations to find the most appropriate one. Let us denote by $\mathbf{C}$ the set of all feasible configurations of $\prod_i |D_i|$ system configurations. We also assume that we have a set $\mathbf{Q}$ of queries with corresponding relevance judgements on a document collection, which can be used to generate training examples for LTR models. Given a query $q_k \in \mathbf{Q}$ and a configuration $c_j \in \mathbf{C}$, we can run the IR system to obtain the search result $\mathbf{R}$ and generate a measure of performance such as Average Precision. We hypothesize that a good configuration is generalizable to queries with similar characteristics. Therefore, the LTR model will be able to select a suitable configuration for a query based on the characteristics.

In this work, we consider a set of common system parameters often used in IR studies (see Table 1), mainly for query expansion, query difficulty prediction or learning-to-rank. As we mentioned earlier, we only consider the parameters that can be set at query time and exclude those related to indexing, as well as those that are not found to be influential on results [4]. We considered one parameter for retrieval model and four parameters for pseudo-relevance feedback.

For each of the parameters, a set of possible values is defined. These values correspond to those that are often used in IR experiments. The set of possible values can certainly be further extended later; but this is beyond the scope of this study, whose primary goal is to demonstrate that feasibility and effectiveness of a new method to set system parameters. In theory, we could have 22,050 ($21 \times 7 \times 6 \times 5 \times 5$) combinations of parameters, but some of the combinations are invalid. For example, when "nil" is selected for Expansion model, the choices in Expansion documents,

[7]Details can be found at terrier.org/docs/v4.2/javadoc.

[8]*nil* means that no expansion has been used; in that case the other expansion parameters are set to 0.

Expansion terms, and Expansion min-docs become irrelevant. In the end, we have more than 10,000 valid system configurations, which form the set of candidate configurations $\mathbf{C}$.

For a training query q, a run is made according to each configuration $c \in \mathbf{C}$ to obtain the search results. The search results are contrasted with the gold standard to obtain the performance measure (e.g., AP). In this way, we obtain a large set of $< q_k, c_j, performance >$ triples. In our experiments, we used Terrier [61] as the retrieval system.

To train our configuration ranking models, we extracted a set of features from the query q_k and the configuration c_j. Usually, in learning-to-rank approaches, the features are related to the query, the document (and sometimes the document collection), as well as the relationships between them. For example, LETOR associates many scores related to query–document pairs. In our case, we defined features relating to the query q_k and to the configuration c_j. It can be costly to define features on the relationships between the query and the configuration, because this requires us to run a retrieval operation for each system configuration and to extract some features according to the retrieved results. In a realistic setting, this would be prohibitively expensive. Therefore, we do not include these features in this study and leave them to future work. However, we do include some post-retrieval features (e.g., query difficulty) that are extracted from a standard configuration (standard BM25), which can be obtained at a reasonably low cost (only one run per query). These features do not reflect the relationships between a configuration and a query but are part of the features of the query.

The performance obtained for the query using the configuration will be used as labels during the training (see Section 3.3). Training examples are composed of features and a label to learn; we describe both in the next subsections.

Our approach aims at training a ranking function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ that minimises:

$$R_{emp}[f] = \frac{1}{Q} \sum_{q=1}^{Q} \mathcal{L}(\pi(f, C_q), y_q), \tag{1}$$

where $\pi(f, C_q)$ is the ranking of IR system configurations C_q by f for the query q. $\mathcal{L}$ is the loss function, measuring the discrepancy between the predicted ranking $\pi(f, C_q)$ and the actual ranking of configurations y_q. In the experiments of this article, we will test several learning-to-rank algorithms that employ different types of loss functions: pointwise, pairwise, and listwise.

3.2 Features for Each Combination of Query-Configuration

As we explained earlier, it would be difficult to extract features about the relation between configuration and query in a realistic context. Therefore, we only considered query features and configuration features.

The extracted features are further divided into four groups, two associated with queries and two associated with configurations. More specifically, there are 46 features covering the statistical characteristics of the query (QueryStats), 30 features describing the linguistic properties of the query (QueryLing), 1 feature for the retrieval model (RetModel), and 4 features related to query expansion parameters (Expansion). The last two groups of features are the same as shown in Table 1.

3.2.1 Configuration Features. A configuration possesses a set of parameters. The parameters are considered as system configuration features. We considered two types of configuration features: the retrieval model and four parameters related to query expansion (the expansion model, the number of documents used during expansion, the number of terms used to expand the initial query, and the minimum number of documents in which a term should occur to be a candidate for

query expansion). For example, one of the configurations $c_j \in \mathbf{C}$ is as follows: Retrieval Model = BM25, Expansion Model = KLComplete, Expansion documents = 50, Expansion terms = 20, and Minimum docs = 10.

3.2.2 Query Features. Query features have mainly been used in the literature for predicting query difficulty [15]. In that context, these features are often categorized into pre-retrieval and post-retrieval features; the former can be extracted from the query itself prior to any search on the document collection, whereas the latter are both query and collection dependent. We extracted both pre- and post-retrieval query features, where some of them convey statistical properties of the query and others refer to linguistics features. Overall, we extracted a set of 76 query features covering the statistical and linguistic properties of the query. The details about the features are described in Reference [57]. For the sake of completeness, we also describe them in the Appendix in Tables 15 and 16 at the end of this article; however, we describe them as QueryStats and QueryLing features in the following.

We consider features that are extracted from queries by statistical means. In the IR literature, these features take into account term weighting and document score. We group them into a category named QueryStats. These features have often been used in IR for query and document matching for automatic query expansion [16, 28], for learning-to-rank documents, and for query difficulty prediction.

In addition to statistical features, natural language processing techniques have also been frequently used in IR. Some linguistic cues have been proposed for different purposes. For example, semantic relationships between terms in WordNet have been used to expand users' queries [50], for term disambiguation [64], or for query difficulty prediction. We thus consider another group of features named QueryLing that are based on these linguistic cues.

- QueryStats: these features are extracted using either pre- or post-retrieval approaches. As pre-retrieval features, we extracted inverse document frequency (IDF) of the query term and estimated two variants of IDF over the query terms using the mean and standard deviation aggregation functions. In addition, we also extracted the term frequency (TF) of the query term in the corpus and estimated the same two variants over the query terms. Post-retrieval query features are extracted from the query–document pairs after a first retrieval. These features were previously used in both query difficulty prediction [15, 19, 38] and learning-to-rank [48, 52]. For example, the BM25 score has been used as a feature on the query–document pair. The other post-retrieval features that we estimated are term frequency (TF) in the top retrieved documents, TF-IDF, Language Modelling with Dirichlet smoothing (μ = 1,000 and 2,500), Language Modelling with Jelinek-Mercer smoothing (λ = 0.4), and document length (DI). Moreover, post-retrieval features are extracted separately from either the document title, body or whole document. We used the Terrier IR platform[9] to calculate the post-retrieval features using the FAT component,[10] which facilitates to compute many post-retrieval features in a single run [51]. We used standard BM25 with no reformulation although any run implemented in the platform could have been used. To make the post-retrieval features usable as query features, we aggregated them over the retrieved documents for a given query. For example, we calculated the mean of the BM25 scores over the retrieved document list for the considered query. Similarly to pre-retrieval statistical query features, we employed the same two aggregation functions namely mean

[9] http://terrier.org/docs/v4.0/learning.html.

[10] The Terrier FAT component calculates multiple query dependent features. It "is so-called because it 'fattens' the result set from the initial ranking (known as the sample [48]) with the postings of matching terms from the inverted index" [51].

Table 2. The Four Groups of Features for a Query–Configuration Pair: QueryStats, QueryLing, RetModel, and Expansion

Group	Variants	Features
QueryStats	3 Pre-retrieval features with mean and standard deviation variants of IDF	IDF [38, 40], and CLARITY [24].
	40 Letor features with mean and standard deviation variants (0 stands for Title, 1 for Body and 2 for both)	SFM(DL,0/1/2), SFM(TF,0/1/2), SFM(IDF,0/1/2), SFM(SUM_TF,0/1/2), SFM(MEAN_TF,0/1/2), SFM(TF_IDF,0/1/2), SFM(BM25,0/1/2), SFM(LMIR.DIR,0/1/2), SFM(LMIR.JM.λ-C-0.4,0/1/2), Pagerank_prior, Pagerank_rank
	3 Query difficulty predictors	WIG [82], QF [82], and NQC [68].
QueryLing	12 WordNet features with mean and standard deviation variants	SYNONYMS, HYPONYMS, MERONYMS, HOLONYMS, HYPERNYMS, and SISTER- TERMS [57].
	18 Linguistic query features No variant	NBWORDS, INTERR, NP, ACRO, NUM, PREP, CC, PP, VBCONJ, UNKNOWN, AVGSIZE, AVGMORPH, %CONSTR, AVGSYNSETS, SYNTDEPTHAVG, SYNTDEPTHMAX, SYNTDISTANCEAVG, and SYNTDISTANCEMAX [58].
RetModel	1 feature representing retrieval model	Retrieval model such as HiemstraLM, BM25, and so on (see Table 1)
Expansion	4 features for query expansion	Expansion model, number of expansion documents, number of expansion terms, and minimum number of documents (see Table 1).

Mean and standard deviation are used as aggregation functions. LETOR feature notations are consistent with the ones Terrier give and details are provided in References [51, 57, 63].

and standard deviation. Moreover, one of the post-retrieval features is PageRank, which can be calculated for linked documents only. Furthermore, we estimated some state-of-the-art query difficulty predictors, including Query Feedback [82] features using various numbers of feedback documents, WIG [82], and a variant of the Normalized query commitment (NQC), which is based on the standard deviation of retrieved documents scores [69].

- QueryLing: These are collection-independent pre-retrieval query features and focus on modelling the linguistic properties of the query. We extracted some linguistic features by exploiting different types of relationship in WordNet [55], including Synonym, Hyponym, Hypernym, Meronym, Holonym, and Sister-terms. Given a query, we counted the related terms of each relationship type of a query term and employed the same two aggregation functions over the set of terms in a query (mean and standard deviation). Moreover, we also extracted the features defined in Reference [58], such as the number of WordNet synsets for a query term, average number of morphemes per word, and so on (see Table 2).

Note that all the training examples of a given query share the same query features but have different configuration features. Query features aim to inform the learning-to-rank technique about the characteristics of the query, thus allowing us to select different system configurations on a per-query basis.

3.3 Labels for Each Example

For each example, we calculated a label that corresponds to the effectiveness of the configuration $c_j \in \mathbf{C}$ when treating a query, q_k. This label is used for the training. More specifically, various effectiveness measures for each example (including AP, P@10, P@100, nDCG@10, nDCG, and R-Prec) will be used for different experiments. When one of the measures is used, the configurations will be ranked according to it for the given query. This will be further detailed in the experiments.

3.4 Training

As our goal of this study is to investigate if the LTR approach for configuration selection is effective, we use all the reasonable features at our disposal without performing any feature selection, leaving this aspect to future work.

In principle, any LTR approach could be used for our task: It would rank system configurations in such a way that the best configuration will be ranked first. This is the configuration that we want to select. Notice, however, that the relative positions of the elements at lower positions are also important in learning-to-rank models (for document ranking). The optimisation of an LTR model takes into account all the elements in the ranked list to some depth. The ranking of configurations at lower positions seems intuitively less critical for our task. Therefore, our learning objective could be different. Intuitively, for our task, we should use nDCG@1 as the objective function to optimize in the training process, because we are only interested in selecting the first configuration.

However, optimising for a very short result list (1 configuration) may make the model unstable and subject to the variation in the features of the query. Therefore, we also consider nDCG@k ($k > 1$) to keep the best top ranked configurations rather than a single one. Details are provided in the evaluation sections.

3.4.1 Learning-to-Rank Techniques. Since this is the first time LTR is used for selecting the system configuration for a given query, we have no clear idea on which LTR method fits best to the problem. We know that with LTR for document ranking, the pointwise methods perform slightly lower than the pairwise and listwise methods. However, the situation in ranking system configurations could be different. Thus, we consider the three types of methods for this new task. We experimented with a large selection of the existing learning-to-rank techniques made available in the RankLib[11] and the SVM^{rank}[12] toolkits. For linear regression, we used the linear model in scikit-learn.[13] Our goal was to experiment the three types of LTR models, namely, pointwise (of which the standard linear regression can be considered as a special form), pairwise, and listwise. We will see that the performance varies largely among the models. We first performed a preliminary analysis (see Table 4 Section 4.3.1) to make a first selection of the most promising LTR models for further experiments: Standard linear regression, Random Forests [10], Gradient Boosted Regression Trees (GBRT) [32], Coordinate Ascent [54], SVM^{rank} [42], RankNet [12], RankBoost [31], AdaRank [78], ListNet [14], and LambdaMART [75]. From these initial results, in the evaluation section, we kept five of these algorithms. These selected techniques cover all three categories of approaches of learning-to-rank: Linear regression, GBRT, and Random Forests are pointwise techniques, SVM^{rank} is pairwise, and LambdaMART uses both pairwise and listwise criteria.[14]

LTR algorithms use positive and negative examples during training. To help convergence, we used 10% of the best configurations as positive examples and 10% of the least performing configurations as negative examples.

3.4.2 Optimization Criteria. It refers to the criterion that is optimized for a ranked list when training the model. This criterion reflects how well the ranked list ranks the most performing configurations (measured by an evaluation metric) on top. As stated previously, the normalized

[11] sourceforge.net/p/lemur/wiki/RankLib/ as for the label which needs to be integer, we discretise the metric by x 10,000 to make it integer.

[12] www.cs.cornell.edu/people/tj/svm_light/svm_rank.html.

[13] http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression.

[14] The implementation of LambdaMART we used is listwise; https://sourceforge.net/p/lemur/wiki/RankLib%20How%20to%20use/.

Table 3. Statistics of the Collections Used in the Experiments

Collection	Number of documents	Number of topics (Range)
TREC7-8	528,918	100 (350–450)
WT10G	1,692,096	100 (451–550)
GOV2	25,205,179	150 (701–850)
Clueweb12B	52,343,021	100 (201–300)

discounted cumulative gain at the cut-off rank 1 (nDCG@1), which optimizes the top 1 configuration, appears the most intuitive option. However, we will also examine several other choices and consider deeper ranked lists.

In the experiments, we mainly focus on nDCG@1 to train the ranking algorithms but we will also show the results with other alternative optimization criteria: nDCG@k ($k \in \{1, 2, \ldots, 10\}$), expected reciprocal rank at n (err@n), and precision at n (P@n) (see Section 4.4.4 and Figure 8).

The optimization criterion should not be confused with the evaluation metric that measures the retrieval effectiveness.

3.4.3 Evaluation Metric. The evaluation metric is used to measure the retrieval effectiveness for a query with a specific configuration. This measure is used to label the configurations for each training query and to evaluate the retrieval effectiveness one can obtain for test queries using a trained LTR model to select configurations. We considered several common evaluation metrics including mean average precision (MAP), precision at the cut-off "n" (P@n and we chose P@10 and P@100), nDCG@10, nDCG, and precision after "R" documents have been retrieved (RPrec). We chose these measures, since these measures combine both (1) widely used measures in IR and (2) complementary measures in the sense that they are weakly correlated [5].

4 EXPERIMENTS AND EVALUATION

In this section, we describe the collections, experimental setup, and results that we obtained by running our proposed approach on the collections and comparing with the baselines.

4.1 TREC Collections

The experiments carried out in this article are on four Text REtrieval Conference (TREC)[15] test collections listed in Table 3: TREC7-8, WT10G, GOV2, and Clueweb12. The TREC7-8 collection is an aggregation of two ad hoc test collections (TREC7 and TREC8) with approximately 500,000 newspaper articles like the *Financial Times*, the *Federal Register*, the *Foreign Broadcast Information Service*, and the *LA Times* [74]. The WT10G collection is composed of approximately 1.6 million Web/Blog page documents [39]. The GOV2 collection includes 25 million web pages, which is the crawl of the .gov domain [20]. For Clueweb12, we opted to the smaller subset Clueweb12-B13,[16] which contains approximately 50 million web pages [21, 22].

Each of the TREC collections consists of different numbers of representative topics. The "standard" format of a TREC topic statement comprises a topic ID, a title, a description, and a narrative. The title contains two or three words that represent the keywords a user could have used to submit a query to a search engine. In our experiments, a query is composed of the topic title only. In the rest of the article, we will use term "query" to refer to the title part of the topic. TREC7-8 collection is composed of 100 topics (i.e., merged of queries 351–400 for TREC07 and queries 401–450 for TREC-8). Similarly, we have 100 topics for WT10G, 150 topics for GOV2, and 100 topics for

[15] trec.nist.gov.

[16] https://lemurproject.org/clueweb12/index.php.

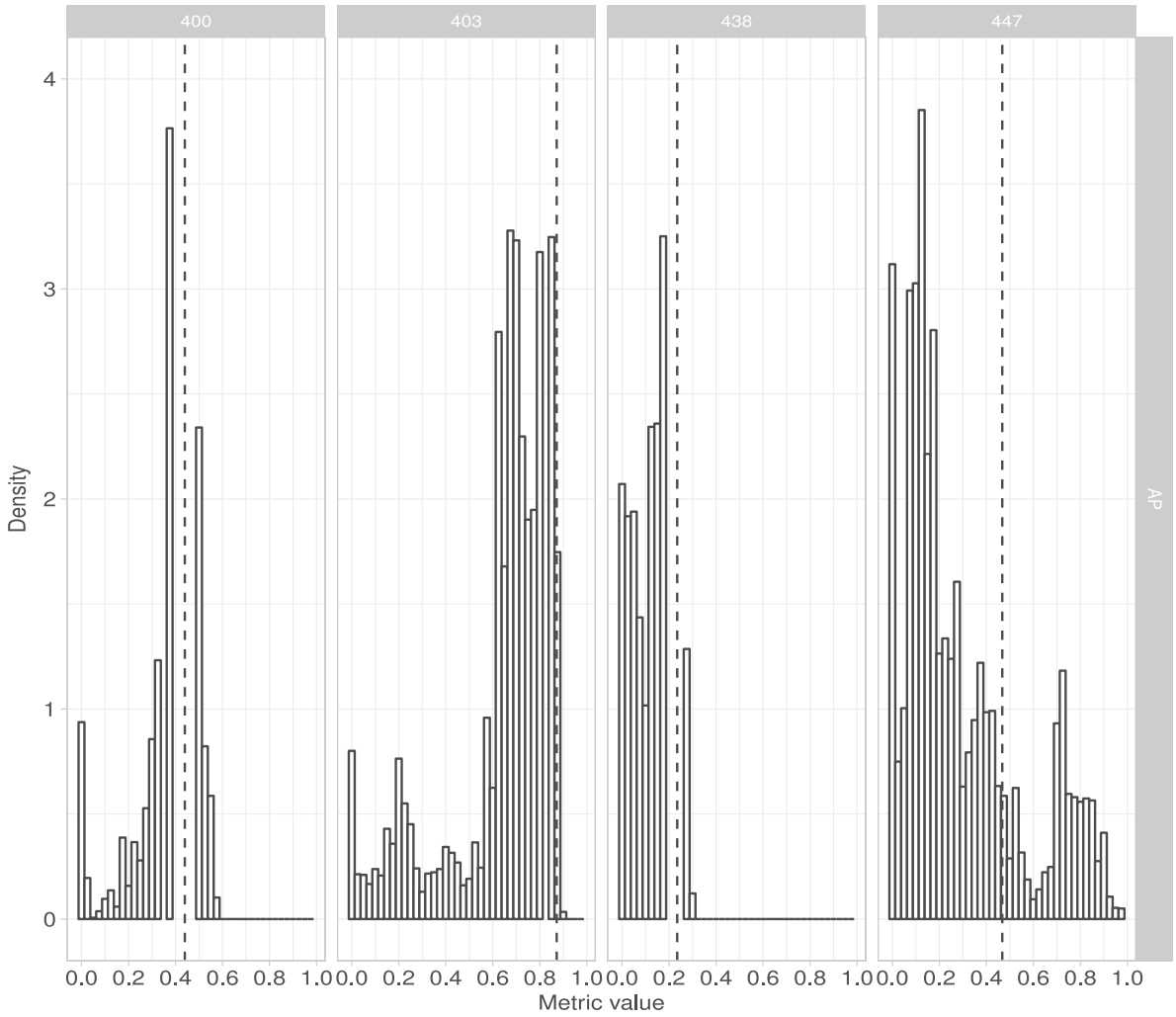


Fig. 1. Density plots of AP for four illustrative queries using the all set of configurations. In each sub-figure, the X-axis corresponds to AP while the Y-axis corresponds to the number of configurations that got the corresponding AP value. From left to right, queries are 400, 403, 438, and 447 from the TREC7-8 collection. Dash lines represent the value of grid search for the considered query.

Clueweb12 (i.e., 50 from TREC web track 2013 [21] and 50 from 2014 [22]). Moreover, each of the test collections provides a *qrels* file, which contains the relevance judgments for each query on a document collection. This qrels file is used by the evaluation program *trec_eval* to calculate the system effectiveness (e.g., MAP, P@100, etc.).

4.2 Insights about Configuration Selection Importance

Before using an LTR model to select a system configuration, we show how important it is to select a good configuration. Figure 1 reports the density of AP for four individual queries from TREC7-8. In each of the sub-figures, the X-axis corresponds to the effectiveness measurement (in bins from 0 to 1), while the Y-axis corresponds to the number of configurations that obtained the corresponding effectiveness. We used the configurations as depicted in Section 3.

The four queries we show have very different distributions, but we can see that for each of them the selection of one configuration has a huge impact on AP. For example, for the first query on the left side, most of the configurations get AP between 0.3 and 0.6, but it is possible to find a configuration for which the AP is as high as 0.8136. Automatically picking up that specific configuration

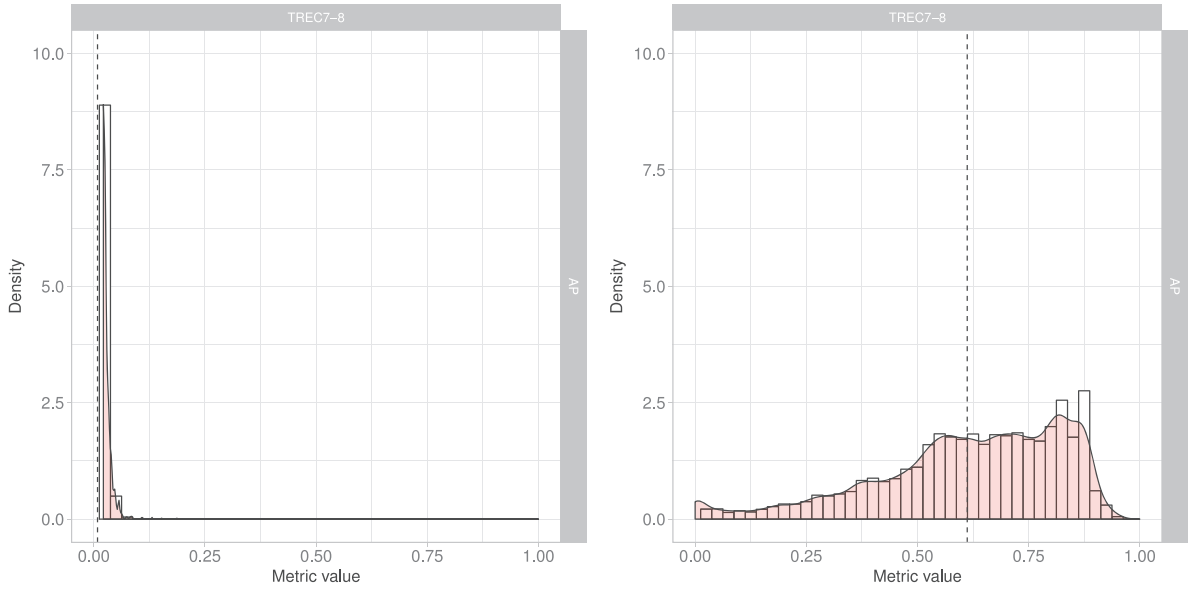


Fig. 2. Density plots of AP for hard and easy queries using the whole set of configurations. In each sub-figure, the X-axis corresponds to AP, while the Y-axis corresponds to the number of configurations that yield the corresponding AP value. Hard queries from the TREC7-8 collection are displayed on the left side of the figure while easy queries from the same collection are displayed on the right side. The dashed line is the mean AP value. Results are from five hard and five easy queries where queries were first sorted in ascending or descending order based on the third quartile values.

would be tremendously interesting. The same holds for the last query, on the right side of the figure: While most of the configurations get low performance, a few get high AP (e.g., the highest AP for the query 447 is 0.9676). The selection is probably easier for query 403 for which most of the configurations get high AP. In this figure, we also report the AP obtained using the grid search method, which selects a configuration for the whole set of queries. Obviously, for some queries, this method fails to generate an effectiveness score close to the best (e.g., 447 on the right side part of the figure).

The above examples show the large impact of selecting a system configuration for different queries. We also know from past evaluation campaigns that there is not a single configuration that is the best for all the queries, and this is indeed the case in Figure 1: The best configuration for the four queries is not the same. Therefore, the selection of system configuration should be query dependent, which is what we intend to do: selecting for each query the best configuration.

To go a step further, we also had a look at two groups of queries for which it is easy or hard to pick a good configuration. Let us consider a set of five queries for which most of the configurations performed well (75% of configurations got AP higher than 0.74 across these queries) and a set of five queries for which most of the configurations performed poorly (75% of configurations got AP lower than 0.01). We can see that (see Figure 2) configurations do not make a big difference in results for hard queries while they can make a huge difference for easy queries. These results are consistent with previous findings [56].

We also had a look at the relationship between query features and configuration features. Figure 3 illustrates this relationship when considering nDCG@10 measure. This figure visualizes a matrix where a column (X-axis) corresponds to a query feature, while a row (Y-axis) corresponds to a configuration. Each value in the matrix corresponds to the canonical correlation when considering nDCG@10 as the measure of the effectiveness for a given configuration for all the queries.

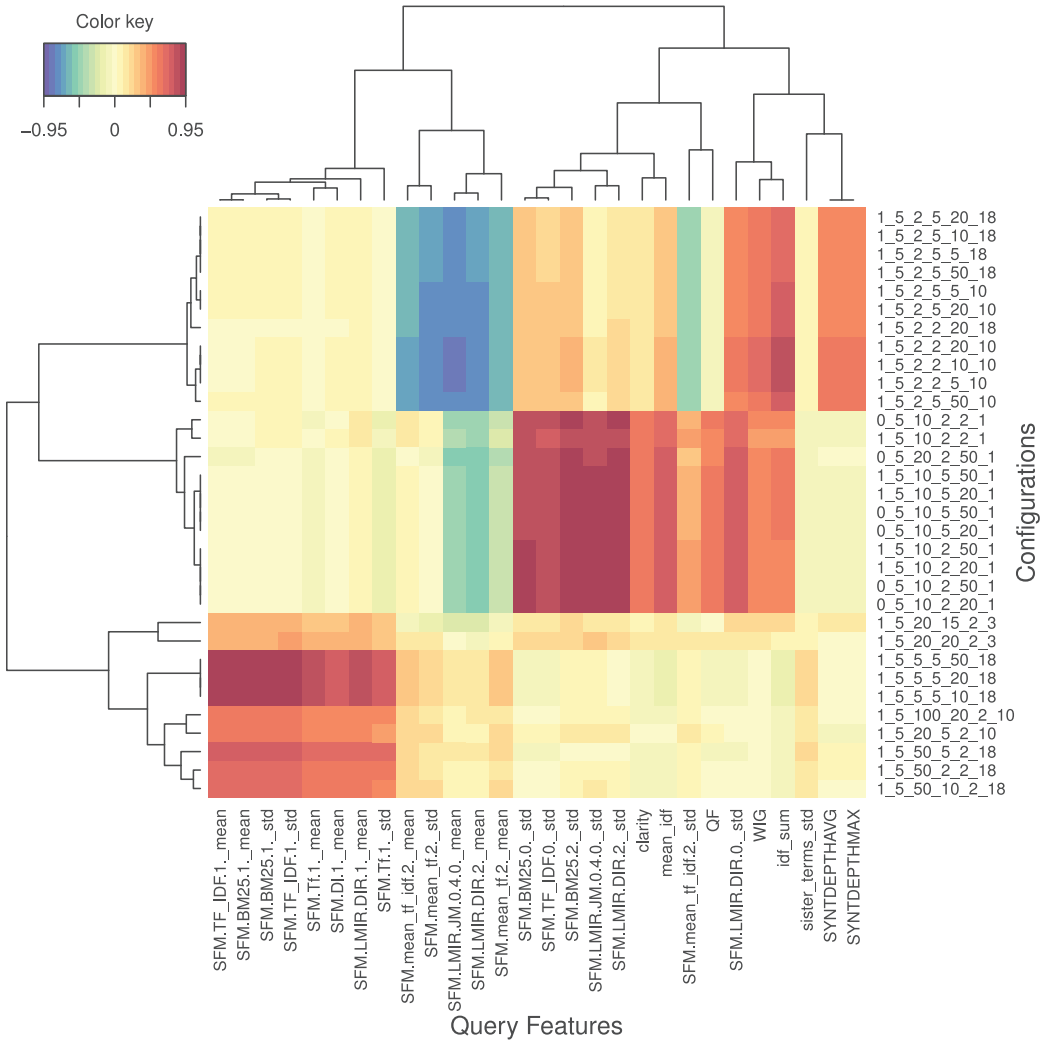


Fig. 3. Visualizing the canonical correlation between the system configurations and query features for evaluation metric nDCG@10 on TREC7-8 collection. The X-axis represents the query features and the Y-axis represents the system configurations. Here, a configuration "1_5_2_5_20_18" refers to "Param free expansion," "Query expansion model," "Number of Expansion documents," "Number of expansion terms," "Minimum number of expansion documents," "Retrieval model," respectively. While we provide the values of the features and settings, this figure is intended to show their overall correlations.

Columns and rows have been ordered to display the strongest correlations. To make the figure more readable, we did not plot all the features but rather a selection of them. The highest correlations (either positive in red or negative in blue) show that some query features are closely related to the effectiveness of the configurations. We also see redundancy either in terms of configurations (similar rows) or in terms of query features (similar columns). The fact that there are complementary query features and complementary configurations which can be clearly seen in different rows or different columns.

4.3 Experimental System Architecture

We depicted the schematic diagram of our proposed approach in Figure 4, which is composed of four main components: pre-processing, training, ranking, and evaluation. Each of the components is described as follows:

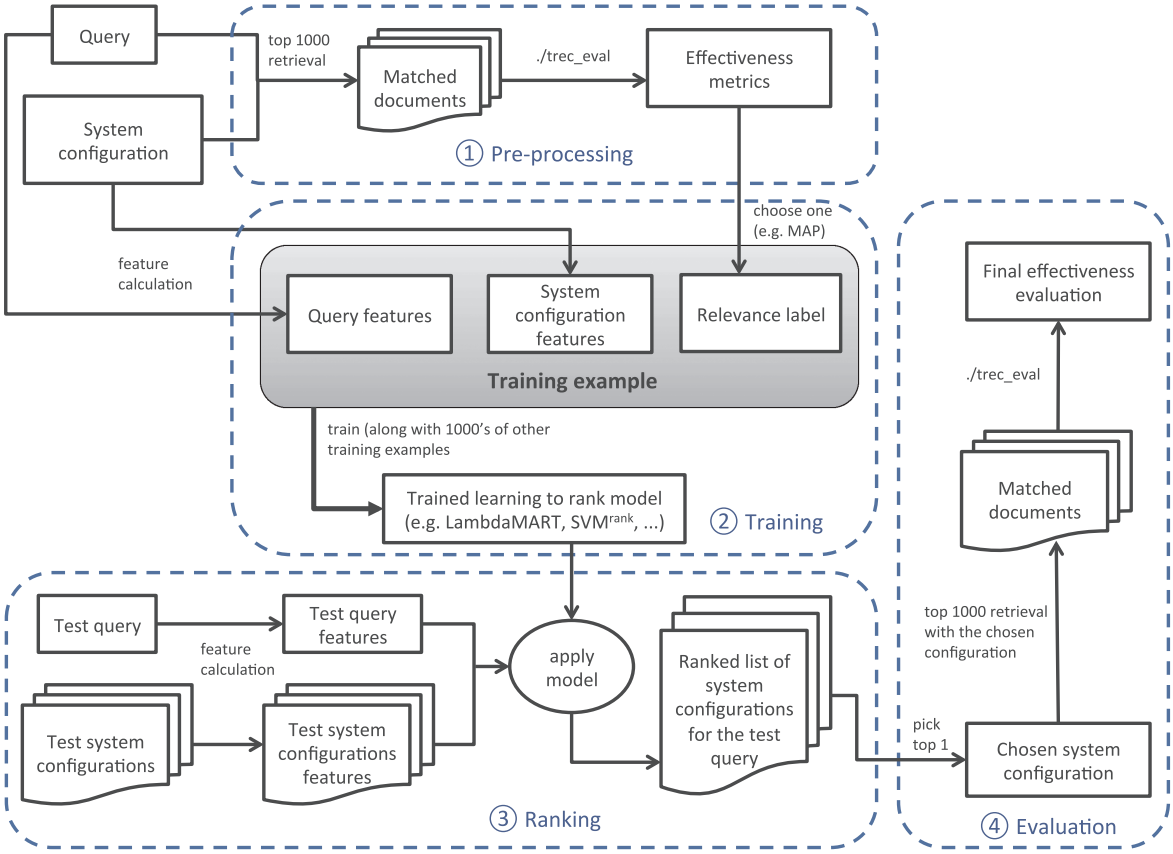


Fig. 4. The four steps of our approach for learning-to-rank system configurations. Steps 1 and 2 are only performed once. Step 1 is covered in this article by Section 3.1; Step 2 is covered by Sections 3.2, 3.3, and 3.4. Steps 3 and 4 are parts of the experiments we detail in the subsequent sections.

- *Pre-processing:* Given a collection, first, we determine all feasible system configurations C. Second, we index the documents corresponding to the collection using a Terrier retrieval system with default parameters including the stop word removal. Third, given the title of a TREC topic, we obtain a run based on each of the system configurations $c_j \in \mathbf{C}$. Finally, we estimate the evaluation metrics using *trec_eval* tool for each run. The metric value is considered as the relevance label of the training example for the underlying run generated by a query and a configuration.
- *Training:* This step makes use of the training examples constructed from the query features, system configuration features, and relevance labels measured by the respective evaluation metric (see the part in grey in Figure 4) and a learning-to-rank algorithm using an optimization criterion (e.g., nDCG@1). Once the training is done, a learned model is generated.
- *Document Ranking:* In the ranking (testing) step, we consider a new query that has not been seen during training and apply our trained model to rank the system configurations for the current query.
- *Evaluation:* The final step is the evaluation of our method in which we calculate the system performance measure by applying the top configuration predicted by the learning-to-rank model for each query using an evaluation metric (e.g., AP).

Following our architecture depicted in Figure 4, we use a fivefold cross validation for all experiments, where each fold has separate training (3/5), validation (1/5), and test sets (1/5). The training

Table 4. Preliminary Results (Nine Different LTR Models of Different Types) Considering nDCG@1 Objective Function Using Six Different Evaluation Measures MAP, P@10, NDCG@10, NDCG, P@100, and RPREC

		MAP	P@10	NDCG@10	NDCG	P@100	RPREC
	Grid search	0.262	0.469	0.498	0.546	0.243	0.299
				nDCG@1			
Pointwise	Linear Regression	0.319$^{\Delta}$	**0.579**$^{\Delta}$	0.612$^{\Delta}$	0.619$^{\Delta}$	0.275$^{\Delta}$	0.349$^{\Delta}$
	Random Forests	**0.335**$^{\Delta}$	0.575$^{\Delta}$	**0.614**$^{\Delta}$	**0.634**$^{\Delta}$	**0.289**$^{\Delta}$	**0.371**$^{\Delta}$
	GBRT	0.334$^{\Delta}$	0.546$^{\Delta}$	0.565$^{\Delta}$	0.616$^{\Delta}$	0.289$^{\Delta}$	0.359$^{\Delta}$
Pairwise	SVM^{rank}	0.315$^{\Delta}$	0.542	0.533	0.606$^{\Delta}$	0.268$^{\Delta}$	0.346$^{\Delta}$
	RankNet	0.311$^{\Delta}$	0.447	0.443	0.479$^{\nabla}$	0.274$^{\Delta}$	0.298
	RankBoost	0.198$^{\nabla}$	0.395	0.401$^{\nabla}$	0.559	0.243	0.195$^{\nabla}$
Listwise	LambdaMART	0.310$^{\Delta}$	0.539	0.338$^{\nabla}$	0.592$^{\Delta}$	0.228	0.329
	Coordasc	0.200$^{\nabla}$	0.316$^{\nabla}$	0.366$^{\nabla}$	0.479	0.269$^{\Delta}$	0.199$^{\nabla}$
	ListNet	0.195$^{\nabla}$	0.386	0.343$^{\nabla}$	0.353$^{\nabla}$	0.110$^{\nabla}$	0.221$^{\nabla}$
	Oracle	0.4117	0.7700	0.8022	0.7073	0.3449	0.4484

TREC7-8 collection is used with fivefold cross-validation.

queries are used to train the learning-to-rank models, the validation queries are used to minimise the over-fitting, and the test queries are used to evaluate the learned models.

Since a quite large number of LTR algorithms exist, we performed a preliminary evaluation on TREC7-8 to select the most interesting models for further investigation.

4.3.1 Preliminary Results. Table 4 presents our preliminary results on TREC7-8 collection with fivefold cross-validation. In this table, we considered different LTR algorithms of each type for a total of nine algorithms, nDCG@1 optimization functions used to learn the model on the training queries, and six different IR system effectiveness measures. These results are obtained for test queries where each query is run using the configuration our trained model predicts.

Table 4 shows that when considering nDCG@1 as the optimization function, thus considering the first ranked configuration, pointwise Random forests algorithm consistently performs the best across performance measures. Linear regression and GBRT, other pointwise algorithms also consistently perform well across performance measures. Also quite close in terms of performance are SVM^{rank} and RankNet for pairwise and LambdaMART for listwise.

Based on these preliminary results, we kept three pointwise (linear regression, Random forests, and GBRT), one pairwise (SVM^{rank}), and one listwise (LambdaMART) LTR algorithms for further evaluations in this article.

These preliminary results also show that our LTR methods can highly improve the results with grid search. These preliminary results need to be confirmed and the next section details the experiment design we set up for that.

4.3.2 Experiment Design. To evaluate our approach to select system configuration, we examine five research questions (**R1**–**R6**):

RQ1: *How effective is our proposed approach using learning-to-rank to select system configuration? Is it able to select the best configuration for a query or a configuration close to it?*

We compare our proposed learning-to-rank system with the BM25 and parameter tuning with the grid search as baselines. The grid search is a well-known technique for optimal parameter

search, which selects the best configuration on a set of training queries and uses it on the test queries. In this experiment, we use all the 81 features related to Linguistic (QueryLing), Statistical (QueryStats), Retrieval (RetModel), and Query expansion (Expansion) aspects. We also report Oracle result where for each query the best configuration is systematically selected. Finally, we go deep into the results to understand better the results.

RQ2: *What LTR model is the most effective for the task?*

We have different classes of LTR approaches: pointwise, pairwise, and listwise approaches. We intend to determine which class of approaches is the most suitable for our task.

RQ3: *Are the different groups of features deem necessary in the learning-to-rank system configuration?*

To evaluate the effect of different groups of features on the trained model, we conducted a feature ablation study. We removed one group of features to see how this affects the system performance.

RQ4: *What is the impact of the optimization criteria at different rank positions?*

Learning to rank algorithms use different optimization criteria while in our case nDCG@1 is used as a natural choice. The optimization criteria may have an impact on the trained model. We studied the impact of the different optimization criteria and at different rank depths.

RQ5: *How influential are the query expansion parameters?*

Query expansion parameters are the most crucial part of system configurations in **C**, because they lead to varying different system performances. We evaluate the influence of expansion features only on the learned model by conducting an ablation study of the expansion parameters. It is thus a more detailed analysis compared to RQ2 that looked at the whole set of expansion features together.

RQ6: *What would be the cost of implementing the method in a commercial search engine?*

Our model considers several query features and several configurations. We discuss the cost of implementing our model on the training phase and on using the trained model or running phase.

4.4 Experimental Results

To address the research questions defined in the previous section, we conducted several experiments and reported the outcome in this section.

4.4.1 Effectiveness of the Learning to Rank System Configurations. While in learning-to-rank documents, the pointwise methods usually yield lower effectiveness than the pairwise and listwise methods, we need to compare the three approaches, since we are dealing with a different task. There are two major differences: (1) We are interested in the first ranked system configuration while the standard LTR is interested in the full document list; (2) our labels are continuous variables while document ranking uses discrete ratings. It is thus worth comparing the results from the three categories of LTR methods. We compared the performance of our proposed learning-to-rank system configuration approach to two baselines. One baseline is the grid search method, which selects the best configuration of a set of training queries (we used both the training and the validation queries here) and uses it on the test queries. This corresponds to the common practice in IR for setting multiple parameters. Notice that this configuration is query independent. A second baseline is Random search [7]. It has been recently introduced as an alternative to manual and grid searches for hyper-parameter optimization. Random search has been shown to be effective even when using 64 trials only [7]. In the experiments, we considered 1,000 trials and report the

Table 5. Comparative Performance of Our Proposed Learning-to-Rank System Configuration Approach Using Five Different Learning to Rank Models on TREC7-8 Collection to Three Baselines: BM25, Grid Search, and Random Search

	MAP	nDCG@10	P@10	RPrec
BM25 (Baseline)	0.211	0.465	0.431	0.255
Grid search (Baseline)	0.262	0.498	0.469	0.299
Random search (Iter: 1000)	0.263	0.507	0.441	0.301
Linear Regression	$0.319^{\Delta\uparrow}$	$0.612^{\Delta\uparrow}$	$0.579^{\Delta\uparrow}$	$0.349^{\Delta\uparrow}$
Random Forests	$0.335^{\Delta\uparrow}$	$0.614^{\Delta\uparrow}$	$0.575^{\Delta\uparrow}$	$0.371^{\Delta\uparrow}$
GBRT	$0.334^{\Delta\uparrow}$	0.565^{Δ}	$0.546^{\Delta\uparrow}$	$0.359^{\Delta\uparrow}$
SVM^{rank}	$0.315^{\Delta\uparrow}$	0.533	$0.542^{\uparrow}$	$0.346^{\Delta\uparrow}$
LambdaMART	$0.310^{\Delta\uparrow}$	$0.338^{\nabla\downarrow}$	$0.539^{\uparrow}$	0.329
Oracle performance	0.412	0.802	0.770	0.448

Oracle performance (upper bound) is also reported when the best configuration for any individual query is used. Δ indicates statistically significant improvement over the grid search baseline, whereas $\uparrow$ indicates statistically significant improvement over the Random search baseline, according to a paired t-test ($p < 0.05$). We used nDCG@1 as optimization function and four different effectiveness measures.

Table 6. Comparative Performance of Our Proposed Learning to Rank System Configuration Approach on WT10G Collection

	MAP	nDCG@10	P@10	RPrec
BM25 (Baseline)	0.199	0.364	0.340	0.243
Grid search (Baseline)	0.245	0.396	0.361	0.273
Random search (Iter: 1000)	0.244	0.367	0.384	0.273
Linear Regression	0.260	$\mathbf{0.453}^{\Delta\uparrow}$	$0.416^{\Delta\uparrow}$	0.301
Random Forests	$\mathbf{0.319}^{\Delta\uparrow}$	$0.452^{\Delta\uparrow}$	$\mathbf{0.437}^{\Delta\uparrow}$	$\mathbf{0.325}^{\Delta\uparrow}$
GBRT	$0.303^{\Delta\uparrow}$	0.400	$0.401^{\uparrow}$	$0.214^{\nabla\downarrow}$
SVM^{rank}	0.228	$0.447^{\Delta\uparrow}$	$0.410^{\uparrow}$	0.296
LambdaMART	0.210	0.321^{∇}	0.285^{∇}	$0.200^{\nabla\downarrow}$
Oracle performance	0.406	0.657	0.638	0.443

Legend and settings are identical to Table 5.

result for the most effective configuration. Moreover, for an easy comparison, we also provided the performance of a standard BM25 run (without query expansion), using the default configuration provided by Terrier.

The training step makes use of many query features as presented in Table 2. The models have been learned using the full set of 81 features related to Linguistic (QueryLing), Statistical (QueryStats), Retrieval (RetModel), and Query expansion (Expansion).

Given a test query, we used the system configuration that has been ranked first by the learned models. We report the average performance on the test queries in Tables 5, 6, 7, and 8 for all four collections TREC7-8, WT10G, GOV2, and Clueweb12, respectively. We also report in Tables 5 to 8 the upper bound of our method, which used the best possible system configuration for each query in the test set (i.e., the Oracle performance).

From Tables 5 to 8, we see that all learning-to-rank techniques can reasonably learn to rank system configurations for most evaluation measures and test collections. The results are generally better than with grid search or random search. These results clearly indicate the benefit of using

Table 7. Comparative Performance of Our Proposed Learning to Rank System Configuration Approach on GOV2 Collection

	MAP	nDCG@10	P@10	RPrec
BM25 (Baseline)	0.279	0.477	0.542	0.345
Grid search (Baseline)	0.357	0.535	0.629	0.390
Random search (Iter: 1000)	0.353	0.519	0.624	0.384
Linear Regression	$0.410^{\Delta\uparrow}$	$0.651^{\Delta\uparrow}$	$\mathbf{0.770}^{\Delta\uparrow}$	$0.441^{\Delta\uparrow}$
Random Forests	$\mathbf{0.411}^{\Delta\uparrow}$	$\mathbf{0.659}^{\Delta\uparrow}$	$0.767^{\Delta\uparrow}$	$0.446^{\Delta\uparrow}$
GBRT	$0.396^{\Delta\uparrow}$	$0.642^{\Delta\uparrow}$	$0.760^{\Delta\uparrow}$	$\mathbf{0.448}^{\Delta\uparrow}$
SVM^{rank}	0.363	$0.634^{\Delta\uparrow}$	$0.741^{\Delta\uparrow}$	$0.433^{\Delta\uparrow}$
LambdaMART	0.324^{∇}	$0.312^{\nabla\downarrow}$	0.618	$0.280^{\nabla\downarrow}$
Oracle performance	0.478	0.813	0.910	0.515

Legend and settings are identical to the Table 5.

Table 8. Comparative Performance of Our Proposed Learning to Rank System Configuration Approach on Clueweb12 Collection

	MAP	nDCG@10	P@10	RPrec
BM25 (Baseline)	0.026	0.151	0.210	0.068
Grid search (Baseline)	0.032	0.167	0.259	0.071
Random search (Iter: 1000)	0.031	0.156	0.223	0.066
Linear Regression	$0.042^{\Delta\uparrow}$	$0.232^{\Delta\uparrow}$	$0.313^{\Delta\uparrow}$	$\mathbf{0.080}^{\Delta\uparrow}$
Random Forests	$\mathbf{0.043}^{\Delta\uparrow}$	$\mathbf{0.235}^{\Delta\uparrow}$	$\mathbf{0.336}^{\Delta\uparrow}$	$0.076^{\uparrow}$
GBRT	$0.040^{\Delta\uparrow}$	$0.227^{\Delta\uparrow}$	$0.349^{\Delta\uparrow}$	$0.079^{\Delta\uparrow}$
SVM^{rank}	$0.040^{\Delta\uparrow}$	$0.235^{\Delta\uparrow}$	$0.324^{\Delta\uparrow}$	$0.078^{\Delta\uparrow}$
LambdaMART	0.025^{∇}	0.175	$0.278^{\uparrow}$	0.072
Oracle performance	0.059	0.321	0.420	0.096

Legend and settings are identical to the Table 5.

a learning-to-rank model to select an appropriate system configuration for a query rather than setting a unique configuration globally.

Among the learning-to-rank models, the pointwise models, especially Random Forests, produce the best results. The second best is the standard linear regression, another pointwise algorithm. The pairwise SVM^{rank} follows. The listwise LambdaMART performs the worst. The performances obtained with the configuration selected by LambdaMART are sometimes lower than that determined by grid search.

This observation differs from the traditional use of learning-to-rank models for document ranking, where pairwise and listwise models are found to perform better than pointwise models [48]. A possible explanation of this is in the difference of the optimization process used in learning-to-rank and our final goal. In pairwise and listwise learning-to-rank methods, the optimization takes into account the relative positions of configurations at lower ranks, while this is not important for our task, which only selects the best configuration. Thus, the changes of other configurations at lower positions do not affect our final choice but will impact the objective functions used in these learning-to-rank algorithms.

The learning-to-rank models also compared favourably to the best-performing systems of the TREC-7 and TREC-8 AdHoc Tracks, TREC-9 Web Track (WT10G), TREC-2004, TREC-2005, and TREC-2006 Terabyte Tracks (GOV2).

The best system that uses the title only at TREC-7 ("ok7as") [65] and TREC-8 ("pir9At0") [43] obtained 0.2614 and 0.3063, respectively, while the Random Forests model can produce 0.3121 and 0.3809 in MAP on the two separate sets of queries.

The best performance from the participants of TREC-9 Web Track was 0.2011 in MAP, while the Random Forests model can produce 0.3371, which is significantly better. The best systems at TREC 2004, 2005, and 2006 Terabyte Tracks obtained 0.2840, 0.3885, and 0.3392 in MAP, respectively. Our Random Forests model achieves 0.3786, 0.4464, and 0.4076 in MAP in the respective Tracks, which is statistically significantly better than the best participants' method (the MAP is 0.4109 when AP is averaged over the three sets of queries).

4.4.2 Differences between Good and Bad Configurations. To understand what distinguishes a good configuration from a bad configuration, we analyse the parameter values in the best 10% and worst 10% configurations, which have been used as positive and negative examples to train our LTR models. In this analysis, we focused on GOV2 collection.

Figure 5 shows the distribution of the parameter values used for each query: (a) among the 10% best configurations and (b) among 10% worst configurations for GOV2. The top part of the figure reports the number of configurations that use a specific retrieval model. They are ordered according to the number of times they are in the 10% best configurations for a query. Only the first 7 retrieval models have been used for query expansion, that is why the 14 others have smaller bars. We can see that the first model "DFRee" appears twice more in best configurations than in the worst ones. This is clearly the overall best retrieval model.

With regard to expansion models, "Bo2" is more frequent in the best configurations than the worst while, except "KL complete" and "Information" expansion models. On the number of expansion terms, the number of expansion documents, and the minimal number of documents the expansion terms should appear in, we see more mixed pictures: There is not a clear winner. In fact, these numbers strongly depend on the retrieval model and the expansion model used. It is impossible to select a good number independently from the models.

It is also interesting to compare the parameter values between the selected configurations and the Oracle configurations to see how close the selections are to the best. Figure 6 reports the distribution of the parameters on GOV2 testing set (from the fivefold cross-validation) both for the Oracle and for our trained model when using Random Forests LTR algorithm with nDCG@1 as the objective function and AP as effectiveness measure. We can see that for retrieval model, the distributions of the Oracle and of our model are very close to each other. This suggests that our LTR model is able to select good retrieval models. However, we observe more differences on the other parameters. These results indicate that it is more difficult to determine the good parameters for expansion than the retrieval model.

In the next section, we will carry out further analysis on the impact of each parameter through ablation.

4.4.3 Ablation Analysis of Different Group of Features. Not all the features are of the same importance. Since we defined several groups of features, it is worth knowing which ones are the most important. This may have an impact on costs. To evaluate the effect of each group of features presented in Section 3.2 (QueryStats, QueryLing, RetModel, and Expansion) for selecting the configuration, we performed the feature ablation analysis. We removed one group of features at each time, performed again the training, and testing steps as before. If we observe a large decrease in retrieval effectiveness, then this would indicate that the ablated features are very important for the learner, while an increase in results would suggest that the features confuse the learner.

We first report the main findings in Figure 7 to make a fast and easy read. In that figure, we reported the performance of feature ablation for Random Forests on TREC7-8 collection. The shape

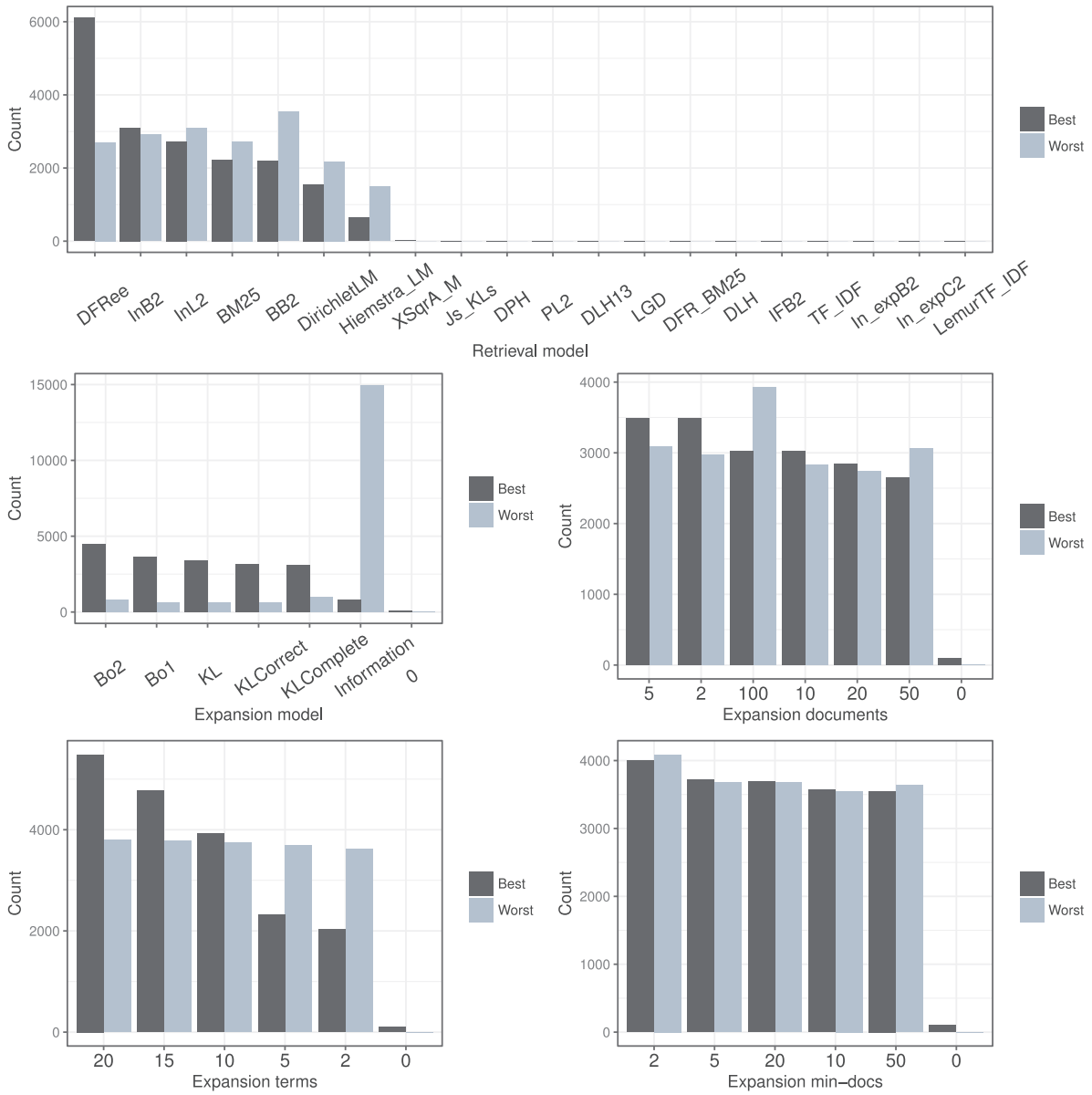


Fig. 5. Distribution of the configurations for a given value of the various parameters of the configurations for GOV2: retrieval model on the top part, then the four expansion parameters when considering the 10% best configurations (black bars) and when considering the 10% worst (grey bars) for each query.

for the other collections is similar. We can clearly see that expansion features are the most important configuration features, since when removing them from the model, the results decrease drastically.

To provide more details, we also report the average performance on the test queries of the features ablation study in Tables 9, 10, 11, and 12 for collections TREC7-8, WT10G, GOV2, and Clueweb12, respectively.

Table 9 should be read as follows: When considering the MAP, for example, the model trained using all the features and Random Forests achieves 0.335. When training the model without the linguistic features associated with the query, MAP increases by 3% to 0.345. Thus linguistic features seem to penalise the effectiveness. Reversely, when training the model without expansion features, MAP decreases of more than 31.9% to 0.228, thus expansion features are very useful in the model.

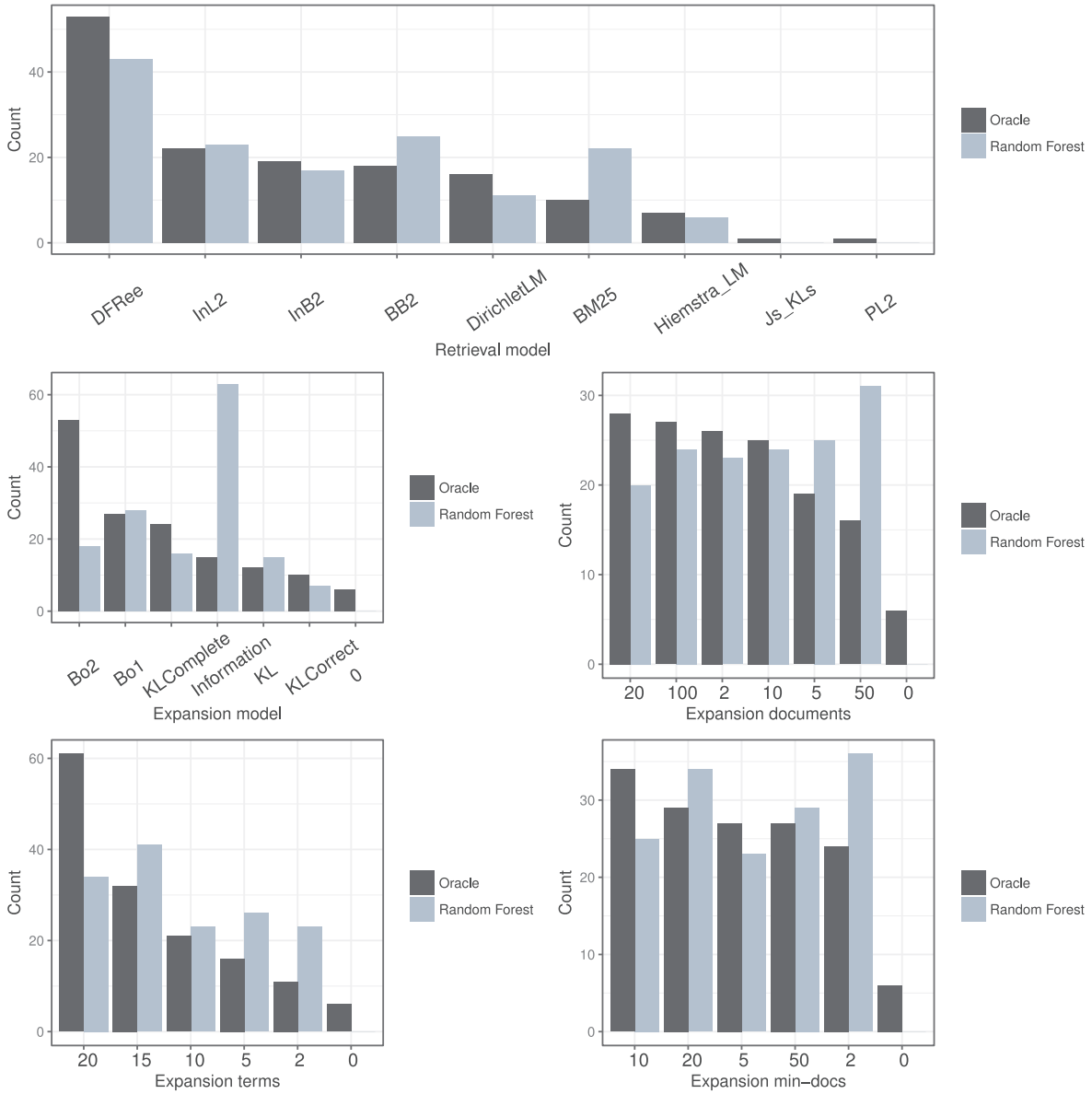


Fig. 6. Distribution of the configurations for a given value of the various parameters of the configurations for GOV2: retrieval model on the top part, then the four expansion parameters; both for the Oracle (black bars) and for our trained model (grey bars) using Random Forests, nDCG@1 as the objective function, and AP as effectiveness measure.

When analysing Table 9, we can make the following observations.

First, we observe that ablating the Expansion group of features always significantly decreases the performance of the learned models, hinting the huge importance of these features for learning an effective model. This observation is in agreement with what we observed in the previous section (Section 4.4.2): The best values for query expansion parameters vary a lot across queries. So, when the expansion features are removed, the final selection will pick a random value among them, leading to large variations in retrieval effectiveness.

The ablation of the other groups of features has less marked impact. When we remove the RetModel feature, the performances can increase or decrease slightly. This may seem surprising. However, comparing with Figure 5, this observation can be explained by the following facts: (1) The

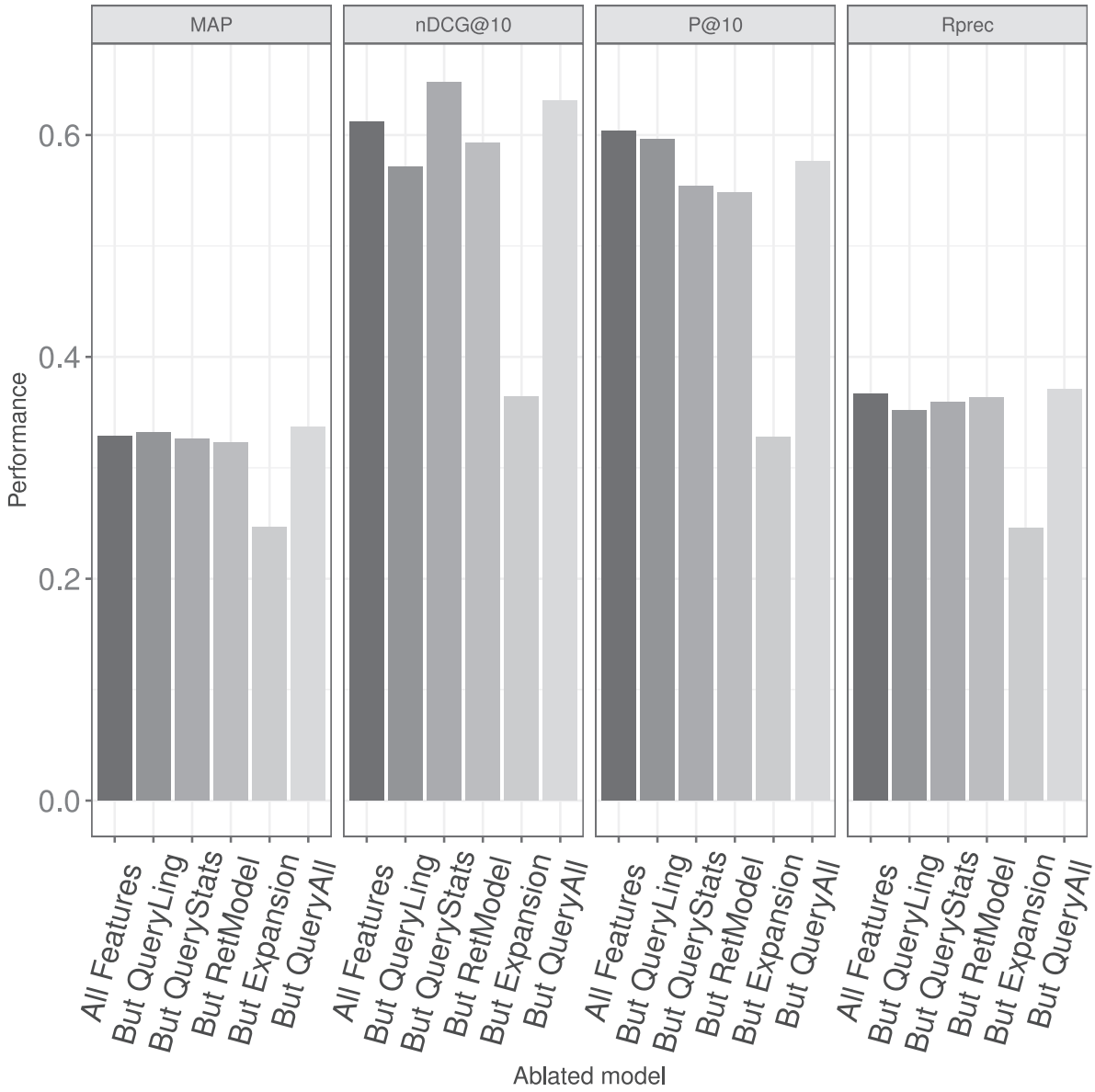


Fig. 7. The performance of feature ablation for Random Forests on TREC7-8 collection. We can observe degradation of the results when removing expansion features and increase when removing QueryStat features.

seven models that are used in combination with expansion parameters (thus can be selected) are all good models that can produce quite high retrieval effectiveness; therefore, any of them could be a reasonable choice, and (2) it is possible that the other features may provide some information to determine the appropriate retrieval model. To confirm this, it would be necessary to examine the correlation between different features. We leave this to future work.

We also observe that the ablation of query-dependent features, namely QueryStats and QueryLing (QueryAll means both QueryStats and QueryLing), produces variable effects. In some cases, we can observe that removing some groups of features leads to even better performance than using the full set of 81 features (first line named (All) for each LTR model). Therefore, it would be desirable to perform a feature selection to keep a subset of useful features. This will be part of our future work.

Table 9. Results with the Five Different Learning-to-Rank Models and Feature Ablations on TREC7-8

	MAP		nDCG@10		P@10		RPrec	
Grid search (Baseline)	0.262		0.498		0.469		0.299	
Random search (Iter: 1000)	0.263		0.507		0.441		0.301	
Linear Regression (All)	$0.319^{\triangle\uparrow}$		$0.612^{\triangle\uparrow}$		$0.579^{\triangle\uparrow}$		$0.349^{\triangle\uparrow}$	
- QueryLing	$0.321^{\triangle\uparrow}$	+0.6%	$0.612^{\triangle\uparrow}$	0.0%	$0.560^{\triangle\uparrow}$	−3.3%	$0.349^{\triangle\uparrow}$	0.0%
- QueryStats	$0.312^{\triangle\uparrow}$	−2.2%	$0.622^{\triangle\uparrow}$	+1.6%	$0.581^{\triangle\uparrow}$	+0.3%	$0.334^{\triangle}$	−4.3%
- RetModel	$0.308^{\triangle\uparrow}$	−3.5%	$0.511^{\blacktriangledown}$	−16.5%	$0.538^{\uparrow}$	−7.1%	$0.340^{\triangle\uparrow}$	−2.6%
- Expansion	$0.152^{\blacktriangledown\triangledown\downarrow}$	−52.4%	$0.289^{\blacktriangledown\triangledown\downarrow}$	−52.8%	$0.303^{\blacktriangledown\triangledown\downarrow}$	−47.7%	$0.200^{\blacktriangledown\triangledown\downarrow}$	−42.7%
- QueryAll	$0.338^{\triangle\uparrow}$	+6.0%	$0.632^{\triangle\uparrow}$	+3.3%	$0.554^{\triangle\uparrow}$	−4.3%	$0.345^{\triangle\uparrow}$	−1.1%
Random Forests (All)	$0.335^{\triangle\uparrow}$		$0.614^{\triangle\uparrow}$		$0.575^{\triangle\uparrow}$		$0.371^{\triangle\uparrow}$	
- QueryLing	$0.345^{\triangle\uparrow}$	+3.0%	$0.595^{\triangle\uparrow}$	−3.1%	$0.585^{\triangle\uparrow}$	+1.7%	$0.357^{\triangle\uparrow}$	−3.8%
- QueryStats	$0.325^{\triangle\uparrow}$	−3.0%	$0.599^{\triangle\uparrow}$	−2.4%	$0.572^{\triangle\uparrow}$	−0.5%	$0.379^{\triangle\uparrow}$	+2.2%
- RetModel	$0.326^{\triangle\uparrow}$	−2.7%	0.561	−8.6%	$0.564^{\triangle\uparrow}$	−1.9%	$0.350^{\triangle\uparrow}$	−5.7%
- Expansion	$0.228^{\blacktriangledown}$	−31.9%	$0.407^{\blacktriangledown\triangledown\downarrow}$	−33.7%	$0.364^{\blacktriangledown\triangledown}$	−36.7%	$0.223^{\blacktriangledown\triangledown\downarrow}$	−39.9%
- QueryAll	$0.330^{\triangle\uparrow}$	−1.5%	$0.611^{\triangle\uparrow}$	−0.5%	$0.600^{\triangle\uparrow}$	+4.3%	$0.358^{\triangle\uparrow}$	−3.5%
GBRT (All)	$0.334^{\triangle\uparrow}$		$0.565^{\triangle}$		$0.546^{\triangle\uparrow}$		$0.359^{\triangle\uparrow}$	
- QueryLing	$0.321^{\triangle\uparrow}$	−3.9%	$0.596^{\triangle\uparrow}$	+5.5%	$0.564^{\triangle\uparrow}$	+3.3%	$0.353^{\triangle\uparrow}$	−1.7%
- QueryStats	$0.311^{\triangle\uparrow}$	−6.9%	$0.601^{\triangle\uparrow}$	+6.4%	$0.561^{\triangle\uparrow}$	+2.8%	$0.345^{\triangle\uparrow}$	−3.9%
- RetModel	$0.316^{\triangle\uparrow}$	−5.4%	$0.577^{\triangle}$	+2.1%	$0.559^{\triangle\uparrow}$	+2.4%	$0.333^{\blacktriangledown\triangle\uparrow}$	−7.2%
- Expansion	$0.223^{\blacktriangledown}$	−33.2%	$0.330^{\blacktriangledown\triangledown\downarrow}$	−41.6%	$0.348^{\blacktriangledown\triangledown\downarrow}$	−36.3%	$0.212^{\blacktriangledown\triangledown\downarrow}$	−41.0%
- QueryAll	$0.320^{\triangle\uparrow}$	−4.2%	$0.618^{\triangle\uparrow}$	+9.4%	$0.616^{\blacktriangle\triangle\uparrow}$	+12.8%	$0.361^{\triangle\uparrow}$	+0.6%
SVM^{rank} (All)	$0.315^{\triangle\uparrow}$		0.533		$0.542^{\uparrow}$		$0.346^{\triangle\uparrow}$	
- QueryLing	$0.314^{\triangle\uparrow}$	−0.3%	$0.582^{\triangle\uparrow}$	+9.2%	$0.545^{\uparrow}$	+0.6%	$0.344^{\triangle\uparrow}$	−0.6%
- QueryStats	$0.308^{\triangle\uparrow}$	−2.2%	0.524	−1.7%	$0.533^{\triangle\uparrow}$	−1.7%	0.322	−6.9%
- RetModel	$0.313^{\triangle\uparrow}$	−0.6%	0.563	+5.6%	$0.557^{\triangle\uparrow}$	+2.8%	0.332	−4.0%
- Expansion	$0.152^{\blacktriangledown\triangledown\downarrow}$	−51.8%	$0.272^{\blacktriangledown\triangledown\downarrow}$	−49.0%	$0.280^{\blacktriangledown\triangledown\downarrow}$	−48.3%	$0.200^{\blacktriangledown\triangledown\downarrow}$	−42.2%
- QueryAll	$0.299^{\triangle\uparrow}$	−5.1%	$0.577^{\triangle}$	+8.3%	$0.537^{\uparrow}$	−0.9%	0.329	−4.9%
LambdaMART (All)	$0.310^{\triangle\uparrow}$		$0.338^{\triangledown\downarrow}$		$0.539^{\uparrow}$		0.329	
- QueryLing	$0.333^{\triangle\uparrow}$	+7.4%	$0.383^{\triangledown\downarrow}$	+13.3%	0.498	−7.6%	$0.371^{\blacktriangle\triangle\uparrow}$	+12.8%
- QueryStats	0.271	−12.6%	$0.344^{\triangledown\downarrow}$	+1.8%	$0.372^{\blacktriangledown\triangledown}$	−31.0%	0.316	−4.0%
- RetModel	$0.257^{\blacktriangledown}$	−17.1%	$0.363^{\triangledown\downarrow}$	+7.4%	0.470	−12.8%	$0.338^{\triangle\uparrow}$	+2.7%
- Expansion	$0.152^{\blacktriangledown\triangledown\downarrow}$	−51.0%	$0.340^{\triangledown\downarrow}$	+0.6%	$0.305^{\blacktriangledown\triangledown\downarrow}$	−43.4%	$0.196^{\blacktriangledown\triangledown\downarrow}$	−40.4%
- QueryAll	$0.279^{\blacktriangledown}$	−10.0%	$0.381^{\triangledown\downarrow}$	+12.7%	0.488	−9.5%	$0.245^{\blacktriangledown\triangledown\downarrow}$	−25.5%
Oracle performance	0.410		0.801		0.775		0.449	

nDCG@1 as optimization function (for all but SVM, which uses Kendall's τ as optimization function).
$\triangle$ and $\uparrow$ indicate statistically significant improvements over the Grid search and Random search baselines respectively, according to a paired t-test ($p < 0.05$).
$\blacktriangledown$ indicates statistically significant decreases induced by a feature ablation with respect to the corresponding (All) models.

4.4.4 Impact of the Optimisation Criteria in Learning to Rank Technique. Learning to rank algorithms use different optimization criteria such as ERR@n, nDCG@n, and so on, to train a model. On the intuition that optimisation criteria may have an impact on the trained model, we experimented with different optimization criteria at the different rank position in training the learning-to-rank models.

In the first set of experiments that we carried in Section 4.4.1 and 4.4.3, the models learned by Random Forests, LambdaMART, and GBRT all used nDCG@1 optimisation criterion. This choice

Table 10. Results with the Five Different Learning-to-Rank Models and Feature Ablations on WT10G

	MAP		nDCG@10		P@10		RPrec	
Grid search (Baseline)	0.245		0.396		0.361		0.273	
Random search (Iter: 1000)	0.244		0.367		0.348		0.273	
Linear Regression (All)	0.260		$0.453^{\Delta\uparrow}$		$0.416^{\Delta\uparrow}$		0.301	
- QueryLing	0.272	+4.6%	$0.453^{\Delta\uparrow}$	0.0%	$0.409^{\uparrow}$	−1.7%	$0.310^{\Delta\uparrow}$	+3.0%
- QueryStats	$0.280^{\Delta\uparrow}$	+7.7%	$0.453^{\Delta\uparrow}$	0.0%	$0.411^{\Delta\uparrow}$	−1.2%	$0.303^{\uparrow}$	+0.7%
- RetModel	0.250	−3.9%	$0.377^{\blacktriangledown}$	−16.8%	0.354	−14.9%	$0.255^{\blacktriangledown}$	−15.3%
- Expansion	$0.159^{\blacktriangledown\nabla\downarrow}$	−38.9%	$0.355^{\blacktriangledown}$	−21.6%	$0.321^{\blacktriangledown}$	−22.8%	$0.207^{\blacktriangledown\nabla\downarrow}$	−31.2%
- QueryAll	0.252	−3.1%	0.441	−2.6%	$0.410^{\uparrow}$	−1.4%	$0.302^{\Delta\uparrow}$	+0.3%
Random Forests (All)	$0.319^{\Delta\uparrow}$		$0.452^{\Delta\uparrow}$		$0.437^{\Delta\uparrow}$		$0.325^{\Delta\uparrow}$	
- QueryLing	$0.308^{\Delta\uparrow}$	−3.5%	$0.477^{\Delta\uparrow}$	+5.5%	0.399	−8.7%	$0.336^{\Delta\uparrow}$	+3.4%
- QueryStats	$0.301^{\blacktriangledown\Delta\uparrow}$	−5.6%	0.438	−3.1%	$0.421^{\Delta\uparrow}$	−3.7%	$0.316^{\Delta\uparrow}$	−2.8%
- RetModel	$0.282^{\blacktriangledown\Delta\uparrow}$	−11.6%	$0.338^{\blacktriangledown}$	−25.2%	$0.345^{\blacktriangledown}$	−21.1%	$0.264^{\blacktriangledown}$	−18.8%
- Expansion	$0.133^{\blacktriangledown\nabla\downarrow}$	−58.3%	$0.356^{\blacktriangledown}$	−21.2%	$0.291^{\blacktriangledown\nabla}$	−33.4%	$0.216^{\blacktriangledown\nabla\downarrow}$	−33.5%
- QueryAll	$0.295^{\blacktriangledown\Delta\uparrow}$	−7.5%	0.421	−6.9%	$0.431^{\Delta\uparrow}$	−1.4%	$0.320^{\Delta\uparrow}$	−1.5%
GBRT (All)	$0.303^{\Delta\uparrow}$		0.400		$0.401^{\uparrow}$		$0.214^{\nabla\downarrow}$	
- QueryLing	$0.285^{\Delta\uparrow}$	−5.9%	0.420	+5.0%	$0.436^{\Delta\uparrow}$	+8.7%	$0.286^{\blacktriangle}$	+33.6%
- QueryStats	$0.247^{\blacktriangledown}$	−18.5%	$0.451^{\Delta\uparrow}$	+12.8%	0.397	−1.0%	0.264	+23.4%
- RetModel	$0.252^{\blacktriangledown}$	−16.8%	0.329^{∇}	−17.8%	0.359	−10.5%	$0.209^{\nabla\downarrow}$	−2.3%
- Expansion	$0.158^{\blacktriangledown\nabla\downarrow}$	−47.9%	0.371	−7.2%	$0.302^{\blacktriangledown}$	−24.7%	$0.210^{\nabla\downarrow}$	−1.9%
- QueryAll	$0.290^{\Delta\uparrow}$	−4.3%	$0.468^{\blacktriangle\Delta\uparrow}$	+17.0%	$0.429^{\Delta\uparrow}$	+7.0%	$0.314^{\blacktriangle\Delta\uparrow}$	+46.7%
SVM^{rank} (All	0.228		$0.447^{\Delta\uparrow}$		$0.410^{\uparrow}$		0.296	
- QueryLing	0.237	+4.0%	0.438	−2.0%	$0.415^{\Delta\uparrow}$	+1.2%	0.285	−3.7%
- QueryStats	0.253	+11.0%	0.440	−1.6%	$0.412^{\uparrow}$	+0.5%	0.293	−1.0%
- RetModel	0.247	+8.3%	$0.381^{\blacktriangledown}$	−14.8%	0.372	−9.3%	$0.248^{\blacktriangledown}$	−16.2%
- Expansion	$0.159^{\blacktriangledown\nabla\downarrow}$	−30.3%	$0.355^{\blacktriangledown}$	−20.6%	$0.321^{\blacktriangledown}$	−21.7%	$0.207^{\blacktriangledown\nabla\downarrow}$	−30.1%
- QueryAll	0.237	+4.0%	0.413	−7.6%	$0.437^{\Delta\uparrow}$	+6.6%	$0.304^{\Delta\uparrow}$	+2.7%
LambdaMART (All)	0.210		0.321^{∇}		0.285^{∇}		$0.200^{\nabla\downarrow}$	
- QueryLing	$0.147^{\blacktriangledown\nabla\downarrow}$	−30.0%	0.364	+13.4%	0.334	+17.2%	0.229	+14.5%
- QueryStats	0.200	−4.8%	$0.263^{\nabla\downarrow}$	−18.1%	0.296	+3.9%	$0.173^{\nabla\downarrow}$	−13.5%
- RetModel	0.230	+9.5%	$0.228^{\blacktriangledown\nabla\downarrow}$	−29.0%	0.311	+9.1%	0.232	+16.0%
- Expansion	$0.135^{\blacktriangledown\nabla\downarrow}$	−35.7%	$0.241^{\nabla\downarrow}$	−24.9%	0.347	+21.8%	$0.200^{\nabla\downarrow}$	0.0%
- QueryAll	$0.296^{\blacktriangle\Delta\uparrow}$	+41.0%	0.310^{∇}	−3.4%	0.300	+5.3%	$0.202^{\nabla\downarrow}$	+1.0%
Oracle performance	0.406		0.657		0.638		0.443	

Legend and settings are identical to Table 9.

makes sense, since we always choose the top-ranked system configuration. However, we also experimented with several other optimisation criteria to see whether they impact the quality of the learned models. In addition to nDCG with different depths, we also experimented with expected reciprocal rank (ERR) and Precision and varied the different cut-off ranks from 1 to 10. Apart from this change, the experimental set-up is identical to what we described in Section 4.4.1, and the results are reported in Figure 8 (SVM is excluded from this analysis, since it uses Kendall's tau as optimisation criteria or loss function).

When analysing the Figure 8, we observe different patterns for different learning-to-rank techniques. In Figure 8(a), we see very little variation in the performance of Random Forests, for all

Table 11. Results with the Five Different Learning to Rank Models and Feature Ablations on GOV2

	MAP		nDCG@10		P@10		RPrec	
Grid search (Baseline)	0.357		0.535		0.629		0.390	
Random search (Iter: 1000)	0.353		0.519		0.624		0.384	
Linear Regression (All)	$0.410^{\Delta\uparrow}$		$0.651^{\Delta\uparrow}$		$0.770^{\Delta\uparrow}$		$0.441^{\Delta\uparrow}$	
- QueryLing	$0.390^{\Delta\uparrow}$	−4.9%	$0.627^{\Delta\uparrow}$	−3.7%	$0.779^{\Delta\uparrow}$	+1.2%	$0.427^{\Delta\uparrow}$	−3.2%
- QueryStats	$0.384^{\blacktriangledown\Delta\uparrow}$	−6.3%	$0.650^{\Delta\uparrow}$	−0.1%	$0.759^{\Delta\uparrow}$	−1.4%	$0.433^{\Delta\uparrow}$	−1.8%
- RetModel	$0.382^{\blacktriangledown\Delta\uparrow}$	−6.8%	$0.512^{\blacktriangledown}$	−21.4%	$0.613^{\blacktriangledown}$	−20.4%	$0.374^{\blacktriangledown}$	−15.2%
- Expansion	$0.127^{\blacktriangledown\triangledown\downarrow}$	−69.0%	$0.378^{\blacktriangledown\triangledown\downarrow}$	−41.9%	$0.528^{\blacktriangledown\triangledown\downarrow}$	−31.4%	$0.212^{\blacktriangledown\triangledown\downarrow}$	−51.9%
- QueryAll	$0.398^{\Delta\uparrow}$	−2.9%	$0.626^{\Delta\uparrow}$	−3.8%	$0.762^{\Delta\uparrow}$	−1.0%	$0.423^{\Delta\uparrow}$	−4.1%
Random Forests (All)	$0.411^{\Delta\uparrow}$		$0.659^{\Delta\uparrow}$		$0.767^{\Delta\uparrow}$		$0.446^{\Delta\uparrow}$	
- QueryLing	$0.418^{\Delta\uparrow}$	+1.7%	$0.650^{\Delta\uparrow}$	−1.4%	$0.799^{\blacktriangle\Delta\uparrow}$	+4.2%	$0.449^{\Delta\uparrow}$	+0.7%
- QueryStats	$0.407^{\Delta\uparrow}$	−1.0%	$0.653^{\Delta\uparrow}$	−0.9%	$0.766^{\Delta\uparrow}$	−0.1%	$0.447^{\Delta\uparrow}$	+0.2%
- RetModel	$0.356^{\blacktriangledown}$	−13.4%	$0.493^{\blacktriangledown}$	−25.2%	$0.581^{\blacktriangledown}$	−24.2%	$0.382^{\blacktriangledown}$	−14.3%
- Expansion	$0.213^{\blacktriangledown\triangledown\downarrow}$	−48.2%	$0.528^{\blacktriangledown}$	−19.9%	$0.578^{\blacktriangledown}$	−24.6%	$0.247^{\blacktriangledown\triangledown\downarrow}$	−44.6%
- QueryAll	$0.411^{\Delta\uparrow}$	0.0%	$0.649^{\Delta\uparrow}$	−1.5%	$0.771^{\Delta\uparrow}$	+0.5%	$0.447^{\Delta\uparrow}$	+0.2%
GBRT All)	$0.396^{\Delta\uparrow}$		$0.642^{\Delta\uparrow}$		$0.760^{\Delta\uparrow}$		$0.448^{\Delta\uparrow}$	
- QueryLing	$0.412^{\Delta\uparrow}$	+4.0%	$0.627^{\Delta\uparrow}$	−2.3%	$0.777^{\Delta\uparrow}$	+2.2%	$0.439^{\Delta\uparrow}$	−2.0%
- QueryStats	$0.396^{\Delta\uparrow}$	0.0%	$0.629^{\Delta\uparrow}$	−2.0%	$0.758^{\Delta\uparrow}$	−0.3%	$0.440^{\Delta\uparrow}$	−1.8%
- RetModel	$0.352^{\blacktriangledown}$	−11.1%	$0.493^{\blacktriangledown}$	−23.2%	$0.573^{\blacktriangledown}$	−24.6%	$0.357^{\blacktriangledown\triangledown}$	−20.3%
- Expansion	$0.180^{\blacktriangledown\triangledown\downarrow}$	−54.5%	$0.456^{\blacktriangledown\triangledown}$	−29.0%	$0.580^{\blacktriangledown}$	−23.7%	$0.271^{\blacktriangledown\triangledown\downarrow}$	−39.5%
- QueryAll	$0.409^{\Delta\uparrow}$	+3.3%	$0.658^{\Delta\uparrow}$	+2.5%	$0.788^{\Delta\uparrow}$	+3.7%	$0.447^{\Delta\uparrow}$	−0.2%
SVM^{rank} (All)	0.363		$0.634^{\Delta\uparrow}$		$0.741^{\Delta\uparrow}$		$0.433^{\Delta\uparrow}$	
- QueryLing	0.345	−5.0%	$0.617^{\Delta\uparrow}$	−2.7%	$0.762^{\Delta\uparrow}$	+2.8%	$0.414^{\Delta\uparrow}$	−4.4%
- QueryStats	0.336	−7.4%	$0.626^{\Delta\uparrow}$	−1.3%	$0.756^{\Delta\uparrow}$	+2.0%	0.405	−6.5%
- RetModel	0.375	+3.3%	$0.516^{\blacktriangledown}$	−18.6%	$0.610^{\blacktriangledown}$	−17.7%	$0.372^{\blacktriangledown}$	−14.1%
- Expansion	$0.127^{\blacktriangledown\triangledown\downarrow}$	−65.0%	$0.378^{\blacktriangledown\triangledown\downarrow}$	−40.4%	$0.528^{\blacktriangledown\triangledown\downarrow}$	−28.7%	$0.212^{\blacktriangledown\triangledown\downarrow}$	−51.0%
- QueryAll	0.342	−5.8%	$0.599^{\Delta\uparrow}$	−5.5%	$0.746^{\Delta\uparrow}$	+0.7%	$0.393^{\blacktriangledown}$	−9.2%
LambdaMART (All)	$0.324^{\triangledown}$		$0.312^{\triangledown\downarrow}$		0.618		$0.280^{\triangledown\downarrow}$	
- QueryLing	0.362	+11.7%	$0.365^{\triangledown\downarrow}$	+17.0%	0.573	−7.3%	$0.296^{\triangledown\downarrow}$	+5.7%
- QueryStats	$0.300^{\triangledown\downarrow}$	−7.4%	$0.394^{\blacktriangle\triangledown\downarrow}$	+26.3%	$0.384^{\blacktriangledown\triangledown\downarrow}$	−37.9%	$0.269^{\triangledown\downarrow}$	−3.9%
- RetModel	$0.362^{\blacktriangle}$	+11.7%	$0.445^{\blacktriangle\triangledown\downarrow}$	+42.6%	0.577	−6.6%	$0.358^{\blacktriangle}$	+27.9%
- Expansion	$0.149^{\blacktriangledown\triangledown\downarrow}$	−54.0%	$0.408^{\blacktriangle\triangledown\downarrow}$	+30.8%	$0.444^{\blacktriangledown\triangledown\downarrow}$	−28.2%	$0.205^{\blacktriangledown\triangledown\downarrow}$	−26.8%
- QueryAll	$0.376^{\blacktriangle}$	+16.1%	$0.389^{\triangledown\downarrow}$	+24.7%	$0.485^{\blacktriangledown\triangledown\downarrow}$	−21.5%	$0.407^{\blacktriangle}$	+45.4%
Oracle performance	0.478		0.813		0.910		0.515	

Legend and settings are identical to Table 9.

collections and all optimisation criteria, which suggests that this LTR model is very stable and can achieve strong results regardless of the optimisation method. These results also make sense in the light of the results of Section 4.4.1 and 4.4.3, which show that Random Forests achieves the best results overall. These strong performances seem to be linked to the adaptability of this learner.

When analysing the Figure 8(b), we observe the same stability for GBRT, except on the WT10G collection for the nDCG@n and P@n metrics. Finally, we see on Figure 8(c) that the performance fluctuates more for LambdaMART for both nDCG@n and P@n criteria, suggesting that this LTR model is unstable and might not be the most suitable for our specific task. These observations confirm the results in Section 4.4.1 and 4.4.3, where LambdaMART achieved the lowest performance

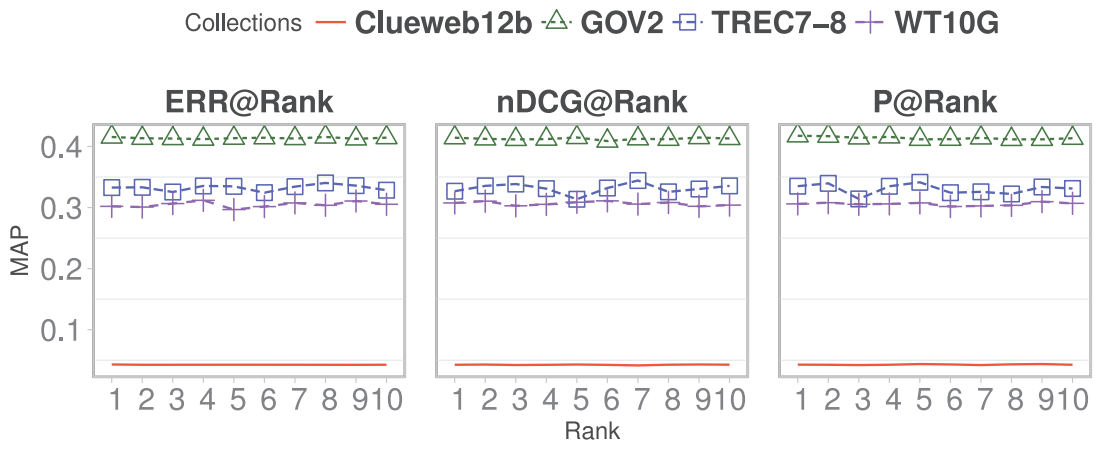
Table 12. Results with the Five Different Learning to Rank Models and Feature Ablations on Clueweb12 Collection

	MAP		nDCG@10		P@10		RPrec	
Grid search (Baseline)	0.032		0.167		0.259		0.071	
Random search (Iter: 1000)	0.031		0.156		**0.223**		0.066	
Linear Regression (All)	$0.042^{\Delta\uparrow}$		$0.232^{\Delta\uparrow}$		$0.313^{\Delta\uparrow}$		$0.080^{\Delta\uparrow}$	
- QueryLing	$0.044^{\Delta\uparrow}$	+4.8%	$0.229^{\Delta\uparrow}$	−1.3%	$0.318^{\Delta\uparrow}$	+1.6%	$0.079^{\Delta\uparrow}$	−1.2%
- QueryStats	$0.043^{\Delta\uparrow}$	+2.4%	$0.214^{\blacktriangledown\Delta\uparrow}$	−7.8%	$0.319^{\Delta\uparrow}$	+1.9%	$0.079^{\Delta\uparrow}$	−1.2%
- RetModel	$0.041^{\Delta\uparrow}$	−2.4%	$0.157^{\blacktriangledown}$	−32.3%	$0.214^{\blacktriangledown\triangledown}$	−31.6%	$0.071^{\blacktriangledown}$	−11.2%
- Expansion	$0.020^{\blacktriangledown\triangledown\downarrow}$	−52.4%	$0.190^{\blacktriangledown}$	−18.1%	$0.249^{\blacktriangledown}$	−20.4%	$0.041^{\blacktriangledown\triangledown\downarrow}$	−48.8%
- QueryAll	$0.044^{\Delta\uparrow}$	+4.8%	$0.210^{\blacktriangledown\Delta\uparrow}$	−9.5%	$0.301^{\Delta\uparrow}$	−3.8%	$0.081^{\Delta\uparrow}$	+1.2%
Random Forests (All)	$0.043^{\Delta\uparrow}$		$0.235^{\Delta\uparrow}$		$0.336^{\Delta\uparrow}$		$0.076^{\uparrow}$	
- QueryLing	$0.042^{\Delta\uparrow}$	−2.3%	$0.235^{\Delta\uparrow}$	0.0%	$0.325^{\Delta\uparrow}$	−3.3%	$0.080^{\Delta\uparrow}$	+5.3%
- QueryStats	$0.043^{\Delta\uparrow}$	0.0%	$0.239^{\Delta\uparrow}$	+1.7%	$0.317^{\Delta\uparrow}$	−5.7%	$0.078^{\uparrow}$	+2.6%
- RetModel	$0.038^{\Delta\uparrow}$	−11.6%	$0.162^{\blacktriangledown}$	−31.1%	$0.216^{\blacktriangledown}$	−35.7%	$0.059^{\blacktriangledown\triangledown}$	−22.4%
- Expansion	$0.025^{\blacktriangledown\triangledown\downarrow}$	−41.9%	$0.203^{\blacktriangledown\Delta\uparrow}$	−13.6%	$0.278^{\blacktriangledown\uparrow}$	−17.3%	$0.061^{\blacktriangledown}$	−19.7%
- QueryAll	$0.043^{\Delta\uparrow}$	0.0%	$0.234^{\Delta\uparrow}$	−0.4%	$0.337^{\Delta\uparrow}$	+0.3%	$0.079^{\Delta\uparrow}$	+4.0%
GBRT (All)	$0.040^{\Delta\uparrow}$		$0.227^{\Delta\uparrow}$		$0.349^{\Delta\uparrow}$		$0.079^{\Delta\uparrow}$	
- QueryLing	$0.041^{\Delta\uparrow}$	+2.5%	$0.232^{\Delta\uparrow}$	+2.2%	$0.322^{\blacktriangledown\Delta\uparrow}$	−7.7%	0.073	−7.6%
- QueryStats	$0.044^{\blacktriangle\Delta\uparrow}$	+10.0%	$0.231^{\Delta\uparrow}$	+1.8%	$0.316^{\blacktriangledown\Delta\uparrow}$	−9.5%	$0.076^{\uparrow}$	−3.8%
- RetModel	$0.040^{\Delta\uparrow}$	0.0%	$0.149^{\blacktriangledown}$	−34.4%	$0.216^{\blacktriangledown}$	−38.1%	$0.061^{\blacktriangledown\triangledown}$	−22.8%
- Expansion	$0.032^{\blacktriangledown}$	−20.0%	$0.203^{\Delta\uparrow}$	−10.6%	$0.292^{\blacktriangledown\uparrow}$	−16.3%	$0.051^{\blacktriangledown\triangledown\downarrow}$	−35.4%
- QueryAll	$0.042^{\Delta\uparrow}$	+5.0%	$0.233^{\Delta\uparrow}$	+2.6%	$0.331^{\blacktriangledown\Delta\uparrow}$	−5.2%	$0.080^{\Delta\uparrow}$	+1.3%
SVM^{rank} (All)	$0.040^{\Delta\uparrow}$		$0.235^{\Delta\uparrow}$		$0.324^{\Delta\uparrow}$		$0.078^{\Delta\uparrow}$	
- QueryLing	$0.042^{\Delta\uparrow}$	+5.0%	$0.234^{\Delta\uparrow}$	−0.4%	$0.326^{\Delta\uparrow}$	+0.6%	$0.079^{\Delta\uparrow}$	+1.3%
- QueryStats	$0.042^{\Delta\uparrow}$	+5.0%	$0.231^{\Delta\uparrow}$	−1.7%	$0.308^{\Delta\uparrow}$	−4.9%	$0.076^{\uparrow}$	−2.6%
- RetModel	$0.038^{\Delta\uparrow}$	−5.0%	$0.155^{\blacktriangledown}$	−34.0%	$0.227^{\blacktriangledown}$	−29.9%	0.073	−6.4%
- Expansion	$0.010^{\blacktriangledown\triangledown\downarrow}$	−75.0%	$0.190^{\blacktriangledown}$	−19.1%	$0.249^{\blacktriangledown}$	−23.1%	$0.041^{\blacktriangledown\triangledown\downarrow}$	−47.4%
- QueryAll	$0.043^{\Delta\uparrow}$	+7.5%	$0.227^{\Delta\uparrow}$	−3.4%	$0.321^{\Delta\uparrow}$	−0.9%	$0.078^{\uparrow}$	0.0%
LambdaMART (All)	$0.025^{\triangledown}$		0.175		$0.278^{\uparrow}$		0.072	
- QueryLing	0.026	+4.0%	0.181	+3.4%	0.246	−11.5%	$0.055^{\blacktriangledown\triangledown}$	−23.6%
- QueryStats	$0.038^{\blacktriangle\Delta\uparrow}$	+52.0%	$0.189^{\uparrow}$	+8.0%	$0.280^{\uparrow}$	+0.7%	$0.051^{\blacktriangledown\triangledown\downarrow}$	−29.2%
- RetModel	$0.039^{\blacktriangle\Delta\uparrow}$	+56.0%	0.143	−18.3%	$0.159^{\blacktriangledown\triangledown\downarrow}$	−42.8%	0.063	−12.5%
- Expansion	$0.021^{\triangledown\downarrow}$	−16.0%	0.177	+1.1%	0.246	−11.5%	$0.047^{\blacktriangledown\triangledown\downarrow}$	−34.7%
- QueryAll	0.026	+4.0%	$0.210^{\blacktriangle\Delta\uparrow}$	+20.0%	$0.274^{\uparrow}$	−1.4%	0.067	−6.9%
Oracle performance	0.059		0.321		0.420		0.096	

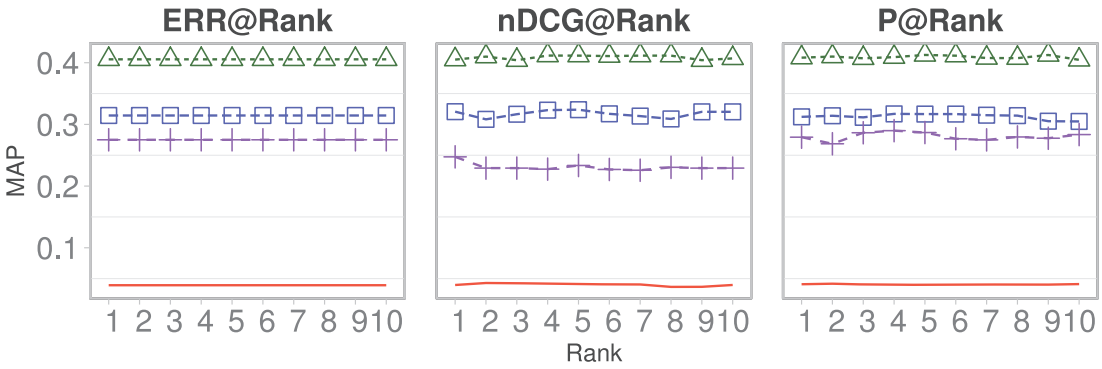
Legend and settings are identical to Table 9.

in comparison to the other LTR models. The fluctuation of the performance with different depths can again explain by the fact that LambdaMART takes into account the whole ranked list to determine the quality of the list. The depth of the ranked list will have some impact on the model. In particular, it will take into account the relative positions of configurations at lower ranks, which are not important for our task.

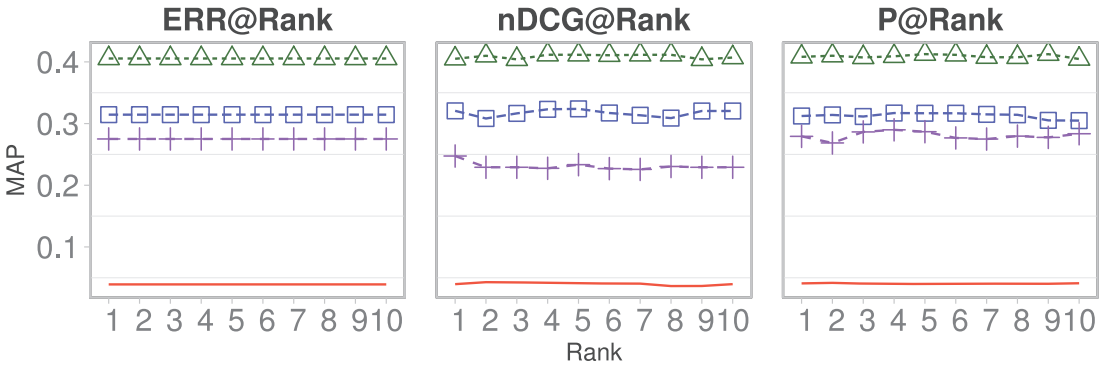
4.4.5 Influence of Query Expansion Features. Since the query expansion parameters exhibit the largest impact on the retrieval effectiveness, we conducted a more detailed feature ablation



(a) Random Forests



(b) GBRT



(c) LambdaMART

Fig. 8. MAP results when varying the optimisation criteria of Random Forests, GBRT, and LambdaMART. The X-axis represents different ranks of the optimisation criterium (i.e., ERR@1, ERR@2, etc.). S VM^{Rank} is not included in this analysis, since the implementation we used does not allow such settings.

experiment on some individual features of this group, with the aim of identifying the single features that have the highest impact on the quality of the learned model.

In this experiment, we compared the performance of the models that have been learned with all features to those of the models that have been learned after removing each of the four Expansion features (see Table 1) individually.

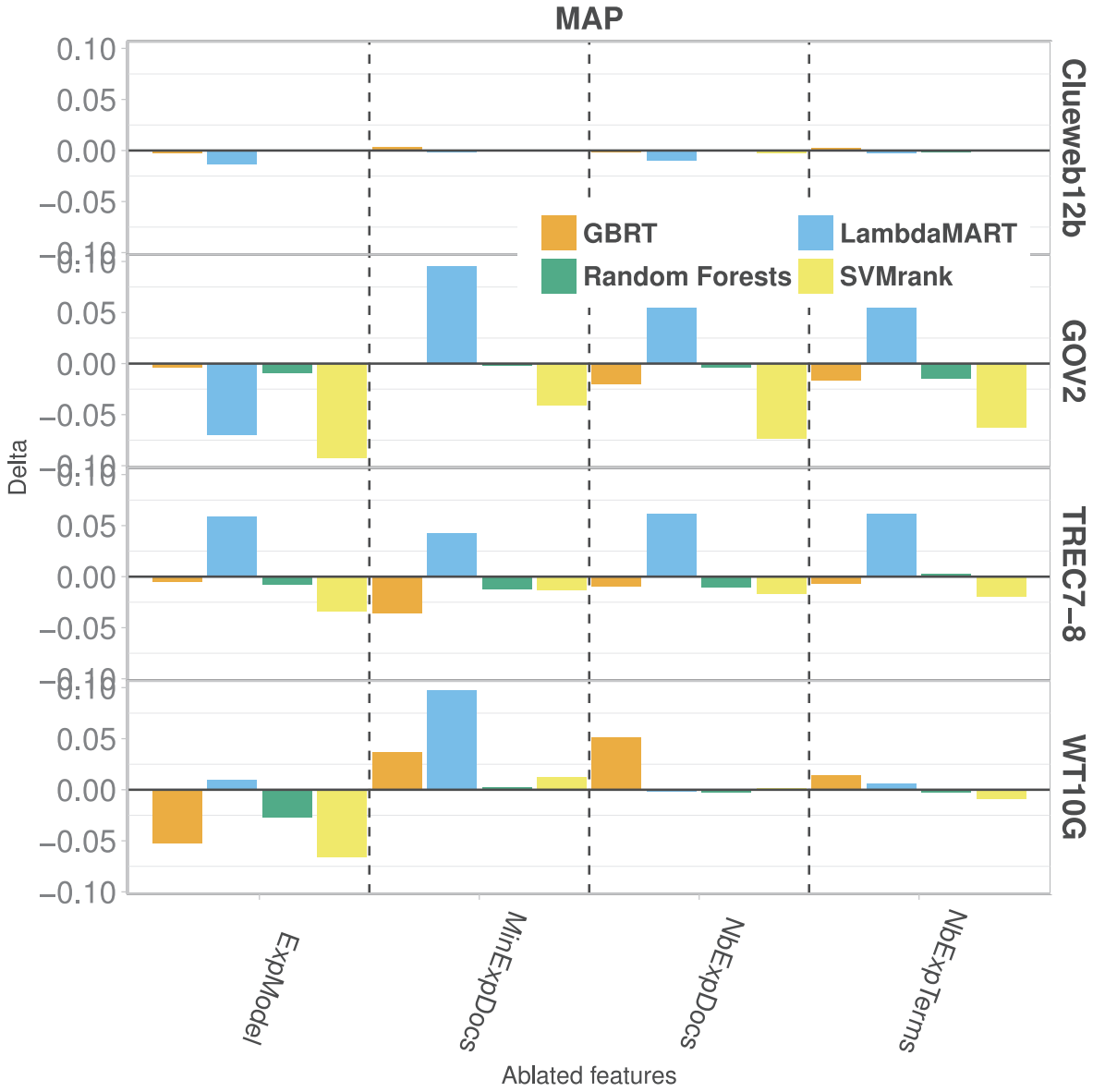


Fig. 9. Absolute changes in MAP when removing individual Expansion features, with respect to models that use all features. This plot is best viewed in colour. Bars above 0.00 line mean that results decreased.

More specifically, we calculate the Δ_μ, where $\mu \in \{MAP, P@10, nDGC@10, RPrec\}$, which represents the difference between the μ metric after and before ablation. Hence, a positive Δ_μ means that removing the feature has improved the quality of the learned model (i.e., the feature is harmful) and a negative Δ_μ means that removing the feature harms performance (i.e., the feature is beneficial).

We reported the results of $\mu = MAP$ in Figure 9. The results with other evaluation metrics are similar. When analysing this figure, we see that the largest changes in results occur for LambdaMART (highest bars either with a positive or a negative change), which confirms our observations in previous experiments that this LTR model appears to be more sensitive to features than others on this specific task. However, the SVM^{rank}, GBRT, and Random Forests techniques are less sensitive to the variation, which highlights their stability and their effectiveness for this task.

Furthermore, we see that the expansion model (ExpModel) appears to be a strong feature for SVM^{rank}, GBRT, and Random Forests on the four collections. When this feature is removed, the effectiveness generally decreases (except LambdaMART). This observation sounds intuitive: Without specifying the expansion model, the choice of other expansion parameters may not be meaningful.

However, the other features like the number of expansion documents or the number of expansion terms seem to be less important and have more variable effects, as seen on Figure 9. We see that removing the number of expansion documents from the features can even lead to improvements on WT10G and GOV2 for LambdaMART and GBRT, respectively.

4.4.6 Cost Aspect. Two main factors influence the cost of the proposed method: (1) the number of candidate configurations and (2) the number of features. Moreover, cost should be distinguished between off-line and on-line. Off-line cost is for training while on-line is for processing a new user's query. The latter is more critical. Most of the costs of our method are off-line costs as we detail below.

Off-line cost: Since the trained system can predict the best system configuration for the given query without going through all the configurations, the number of configurations only affects the cost of the training phase.

The more configurations the meta system can choose among, the higher the probability to have a configuration that is effective for a given query, since we have an enlarged configuration space. The off-line cost can, in turn, be decomposed into two parts: pre-processing and training. In the pre-processing step, we have to design the set of configurations, run the queries with each configuration, evaluate the runs to generate effectiveness metrics, and, finally, prepare the features for the training stage. Therefore, the more configurations we have in pre-processing stage, the higher the cost to build the runs.

Suppose that we have to build runs in C configuration settings for each query, evaluate the effectiveness, and prepare the features for the training stage. Given a collection with a set Q of queries, the cost to build all the runs is $C \times cost\ (run, Q, c_i)$, where $cost\ (run, Q, c_i)$ is a function that returns the cost of retrieving a batch run for the set of Q queries under a particular configuration $c_i \in C$. At training stage, as we explore a large number of configurations (which could probably be reduced) with varying degree of effectiveness, we used a sampling process to reduce it to 20%, which helps to train a LTR model quickly and leads to a set of globally effective configurations.

During the training process of a LTR model (e.g., Random Forests), the time complexity is $O\ (Nt \times Nf \times Nc \times \log(Nc))$, where Nt is the number of trees, Nf is the number of features at each node, and Nc is the number of training samples. In our case with default parameters, $Nc = C \times 20\%$ and $Nt = 300$, $Nf = 23$ (since the feature sampling rate is 30% and the total number of features is 78). Thus, in the worst-case scenario, the cost to train a Random Forest would be $O\ (300 \times 23 \times \text{Nc} \times \log(Nc))$. This is a training cost which only occurs at an off-line stage.

On-line cost: The cost that matters the most at the on-line stage is related to the selection of a configuration, which has a complexity of $O\ (Nt \times Nf \times \log(Nf)) \approx O\ (Nf)$ (since $Nf \gg Nt$ and $Nf \gg \log(Nf)$), a linear time complexity. We can further lower the cost of training and prediction by sampling the training examples (i.e., a smaller Nf) while maintaining a good predictive performance.

However, the number of query features affects both the training phase (off-line) and the running phase (on-line). In principle, the more features we have, the better the trained model could be. However, experimentally we found that ablating some features helps the training and improves

Table 13. Availability of the Off-the-Self Parameters in the Open-Source IR Systems Such as LEMUR and TERRIER

Parameter	LEMUR	TERRIER
Retrieval model	BM25, TF· IDF, VSM, Language model (Dirichlet, Jelinek-mercer, Two-stage), KL-divergence, and so on.	BB2, BM25, DFRee, DirichletLM, HiemstraLM, XSqrAM, DPH, IFB2, TFIDF, and InexpC2, and so on.[17]
Pseudo relevance feedback	Yes	Yes
Expansion model	RMExpander, PonteExpander, TFIDFExpander	KL, Bo1, Bo2, KLCorrect, Information, and KLComplete[18]
Expansion documents	Yes	Yes
Expansion terms	Yes	Yes
Minimum documents	No	Yes

the performance of the model. In this article, we only ablated groups of features, a finer study would be needed to detect which are the most important features of each group and which ones are common across collections.

Notice also that the cost of feature extraction is not the same for all features. It depends on the type of feature: pre-retrieval features are less costly than post-retrieval features, since the latter need to process the query to be calculated. For example, the cost of extracting idf features is fairly low, as the system only needs to retrieve statistics from the inverted-index for each query term and to aggregate the values. However, the post-retrieval QF (query feedback) feature, which is the percentage of overlap at some rank between the documents retrieved for the original query and the expanded query, requires a costly two-stage processing. Some other post-retrieval features and query performance predictors (QPP) such as those used in Letor, e.g., BM25 or NQC, are obtained from the documents retrieved at the first stage. They require less computation than QF. It is possible to make the feature extraction faster. For example, we could replace QF predictors by another QPP such as the weighted product model (WPM) [67], which was shown to be effective and requires only the documents from the first retrieval stage. In addition, given that the average length of the queries is small, 4.1 words [3], the cost of estimating the considered query features is not prohibitively high in practice.

We see in the above analysis that the cost depends on the number of features we use. As we showed earlier in Figure 3, the features could be redundant (i.e., rows or columns that are similar). We could limit the number of features without degrading the overall performance of our method. We will leave investigation on this aspect to future studies.

The additional cost of implementing a strategy as we proposed in this article in the existing Lemur and Terrier platforms is relatively small if we consider a limited number of features and a limited number of configurations. With regard to query features, Terrier has already implemented in its FAT [51] component a number of post-retrieval features that could be easily used. Lemur provides the clarity feature.

With regard to configuration parameters as stated in Table 13, Lemur has several configurations of language modelling and non-language modelling based retrieval models, with various strategies of reformulation made possible by the λ parameter [46]. In Terrier, configuration parameters could be easily set, since it has already implemented an extensive list of retrieval models, expansion models, and so on. Therefore, our method can be directly implemented on top of the existing platforms.

Table 14. Transfer Learning Using Random Forests

Training on WT10G and GOV2 collections; testing on TREC7-8 collection				
	MAP	nDCG@10	P@10	RPrec
BM25	0.211	0.464	0.431	0.255
Random search	0.174	0.406	0.391	0.234
RF 5-fold CV	**0.329**$^{\uparrow}$	**0.612**$^{\uparrow}$	**0.604**$^{\uparrow}$	**0.367**$^{\uparrow}$
RF transfer	0.249$^{\uparrow}$	0.492	0.474	0.298$^{\uparrow}$
Oracle	0.412	0.802	0.770	0.448
Training on TREC7-8 and GOV2; testing on WT10G collection				
	MAP	nDCG@10	P@10	RPrec
BM25	0.199	0.363	0.340	0.243
Random search	0.227	0.341	0.342	0.251
RF 5-fold CV	**0.307**$^{\uparrow}$	**0.443**$^{\uparrow}$	**0.427**$^{\uparrow}$	**0.333**$^{\uparrow}$
RF transfer	0.269	0.441$^{\uparrow}$	0.420	0.265
Oracle	0.406	0.657	0.638	0.443
Training on TREC7-8 and WT10G; testing on GOV2 collection				
	MAP	nDCG@10	P@10	RPrec
BM25	0.279	0.476	0.542	0.345
Random search	0.321	0.456	0.540	0.353
RF 5-fold CV	**0.411**$^{\uparrow}$	**0.652**$^{\uparrow}$	**0.788**$^{\uparrow}$	**0.452**$^{\uparrow}$
RF transfer	0.366	0.601$^{\uparrow}$	0.722$^{\uparrow}$	0.390
Oracle	0.478	0.813	0.909	0.515

We compare the initial results when learning is made on the same collection as testing (using fivefold cross-validation on queries) and when using transfer learning from two collections to the third one. BM25 and Random search, as well as Oracle performance (upper bound) are also reported. ↑ indicates statistically significant improvement over the Random search baseline, according to a paired t-test ($p < 0.05$). We used nDCG@1 as optimization function and four different effectiveness measures.

4.5 Transfer Learning

Transfer learning refers to the principle of training a model on one dataset and using it on other datasets.

In our case, transfer learning can be applied as follows: learning on one collection and testing on the others. Since the collections differ both in terms of queries and in terms of document collection, the result of transfer learning will give a good cue on how generalisable the model is. If transfer learning works well, then it would mean that the model does not need to be trained again for new document collections (or when new documents are added), and thus there is no training cost apart from the initial one. Reversely, if the model should be trained for any new document collection before being applicable, then its implies additional off-line cost for training.

In Table 14, we report the results for Random Forests, since it performed best in our previous experiments. We consider three of the four collections, because these three collections shared exactly the same features once we removed page rank features. We train on two collections and tested on the third one. For each combination, we report the BM25 and Random search baselines as well as Oracle which uses the best configuration, as in the previous experiments. We then report the initial results using Random Forests when the model is trained on the current collection using fivefold cross-validation and when using transfer learning.

In Table 14, we can see that transfer learning is better than the Random search baseline, which is the best of our three baselines. However, cross-validation on the same collection works better than transfer learning.

This result shows that while transfer learning could help choosing better configurations compared to Random search, the document and query collections are very different in nature so that a model learned from one collection could be hardly transferred to another collection. A better solution is to train a model using a set of training queries on the same collection. This result indicates that transfer learning for choosing system configurations are difficult because of the large differences between collections.

5 CONCLUSIONS AND DISCUSSIONS

Main results. In this article, we proposed a new approach to set system configuration using learning-to-rank methods. This work is motivated by the intuition that selecting a good system configuration boils down to rank the candidate configurations. Thus, LTR methods can be used. In this article, we showed that this approach is feasible, and it can produce performance superior to the state of the art, specifically to the traditional grid search method and to the best results submitted to TREC. Our approach can be seen as a generalisation of selective query expansion (SQE) that implements a binary decision: A model decides whether the query should be expanded or not [2, 25, 81]. In SQE, the state-of-the-art models use quite simple features and methods to make the decision.

The method we proposed in this article uses much more complex features and learning algorithms to make the decision. In addition, our decision is also more complex: We have to decide not only the number of feedback documents and terms to use but also several other system setting parameters.

Globally, our study proposes to see a search system no longer as a system with pre-set parameters, but as a system with parameters settable according to the query. Thus, this study paves the way to a new research direction—IR as an adaptive service. Depending on the characteristics of the query, different configurations can be used.

Among the three families of LTR approaches, the best and most stable (across collections and measures) are the pointwise approaches, in particular Random Forests. The two other families of approaches tend to be less effective. The worst ones are the listwise methods. This observation is different from that on document ranking, where listwise approaches produce the best results. This difference could be explained by the difference between our task and that of document ranking: In our case, we are interested in selecting one best configuration (and indeed we optimized the model on ndcg@1), while in document ranking, one aims to rank a list of documents. In the latter case, the relative positions of documents at lower ranks matter, but not in our case of configuration selection. Therefore, a pointwise approach (or, similarly, a regression approach) is more suitable.

Our experiments also showed the importance and benefits to make query-dependent configuration setting. Using any LTR method, we were able to select a better configuration than the Grid or Random search methods, which make query-independent selections.

The feature ablation analysis demonstrated various impacts of different features. In particular, the features of query exhibited lower impact than we initially thought. More investigations are needed to fully understand the reasons.

[17] http://terrier.org/docs/v4.2/javadoc/org/terrier/matching/models/package-summary.html.
[18] http://terrier.org/docs/v4.2/javadoc/org/terrier/matching/models/queryexpansion/package-summary.html.

Selection and costs. This is the first study on utilising LTR to select system configurations and we have limited its scope to several aspects. For example, we did not make a selection of the features to be used and simply used all the features proposed in the literature that sound relevant. However, we observed that the relevance of some of the features to our task may be low. The features can also be redundant, providing similar or even contradictory information. We think that we should apply some feature selection to help the algorithms focusing on the most important features. It is also useful to carry out an analysis of the correlation between features to understand their interactions. This aspect will be very important for real computational cost. Indeed, feature extraction may incur a significant cost that should not be neglected for concrete use of our method in a search engine. A more extensive investigation is needed to select the most influential features of each type and the ones that are common across collections. The tradeoff between effectiveness and feature calculation needs to be investigated in future studies.

Impact of LTR algorithms. The main application of learning-to-rank in IR is to rank documents. The literature of the domain has shown that listwise algorithms are the best [47, 72]. In this article, however, our observation is different. Overall, pointwise approaches, and specifically, Random Forests, performs the best. This result is based on global performance on the set of queries, which as usual hide some epiphenomenon or specific behaviour. It is necessary to carry out a detailed analysis to understand why there is such a difference between leaning to rank system configurations and learning-to-rank documents.

The focus in optimising system performance. The need for a careful selection of configuration is different from one query to another.

As we reported in this article, queries that have a broad range of effectiveness values require more a careful selection than queries that have a limited range of performance values. Indeed, when there is a limited range of effectiveness values, any configuration can be picked up, even randomly, without decreasing user's satisfaction; while with a large range of effectiveness values, the selection is more difficult and should be done more carefully.

This observation suggests that in trying to optimise system performance, we should focus on queries for which the range of effectiveness value is large for which it is crucial to select the appropriate configuration. This is different from the previous studies which suggested to focus on "hard" queries [23] that have low effectiveness values.

Other research directions. Although we included the most relevant features available, the feature set can be further enlarged. In particular, the features reflecting the relationships between a query and a specific configuration could be very informative. However, we need to find a tractable way to extract such features. Considering the number of configurations we used, these specific features would have been costly to obtain.

We created a quite large set of candidate configurations based on the results of previous studies. It may be more appropriate to design a way to determine the values of parameters dynamically rather than choosing among the fixed candidate set.

Another open question is the effectiveness of our method when considering diversity as for Clueweb12 TREC task, for example. In that specific task, the idea is that the relevance of documents may depend on the aspects or sub-queries underlying the general query. Intuitively, different configurations could answer different sub-queries and thus their combination could be helpful in diversity task.

APPENDIX

A DETAILS ON THE QUERY FEATURES

Table 15. QueryStats Features

From Reference [63], calculated using Terrier module [51]	
Name	Detail
SFM(DL,0) or 1 or 2	The score for the language model with Dirichlet smoothing for the query and the document title or body or both.
SFM(TF,0) or 1 or 2	The value of the TF for the query and the document title or body or both.
SFM(TF_IDF,0) or 1 or 2	The value of the TF· IDF score for the query and the document title or body or both.
sum_tf_idf_full mean_tf_idf_full	The sum or mean of TF· IDF values for the query terms
sum_tf_full mean_tf_full	The sum or mean of TF values for the query terms
JM.colλ0.4.docλ0.0 and .1	The score value for the language model with Jelinek-Mercer smoothing, with a collection lambda of 0.4.
SFM(BM25,0) or 1 or 2	The value of BM25 model for the query and the document title or body or both.
From Reference [63], calculated using Lemur	
pagerank_rank and pagerank_prior	The the pagerank score and log probability of it (Inlink count within the retrieved document set).
Query difficulty predictors	
Name	Detail
IDF	Query terms IDF (inverse document frequency).
Clarity [24]	The entropy between a query and the collection language models.
WIG [82]	Weighted information gain
QF [82]	Query feedback
NQC [69]	Normalized query commitment

Table 16. QueryLing Features

Name	Detail (from Reference [58])
NBWORDS	Number of words (terms) in the query
INTERR	Number of interrogative words (terms) in the query
PN	Number of proper nouns (according to CoreNLP's POS tagger)
ACRO	Number of acronyms
NUM	Number of numeral values (dates, quantities, and so on.)
PREP	Number of prepositions (idem)
CC	
PP	Number of personal pronouns (idem)
CONJ	Number of conjunctions (according to CoreNLP's POS tagger)
UNKNOWN	Number of unknown tokens (based on WordNet)
AVGSIZE	Word length in number of characters
AVGMORPH	Average number of morphemes per word (according to CELEX)
%CONSTR	
AVGSYNSETS	Average number of query term sense in WordNet
SYNTDEPTH	Syntactic depth (max depth of the syntactic tree, according to CoreNLP parser)
SYNTDIST	Syntactic links span (average distance between words linked by a dependency syntactic relation)
Name	**Detail (from Reference [57])**
SYNONYMS	Number of terms denoted as synonyms (same sense) to the query terms in WordNet
HYPONYMS	Number of terms denoted as hyponyms (generic relationship) to the query terms in WordNet
MERONYM	Number of terms denoted as meronyms (part-whole relationship) to the query terms in WordNet
SISTER-TERMS	Number of terms denoted as sister-term in WordNet (share the same hypernym) for the query terms

ACKNOWLEDGMENT

The authors thank the anonymous reviewers for their valuable comments that helped in improving the quality of this article.

REFERENCES

[1] Nega Alemayehu. 2003. Analysis of performance variation using query expansion. *J. Assoc. Inf. Sci. Technol.* 54, 5 (2003), 379–391.

[2] Giambattista Amati, Claudio Carpineto, and Giovanni Romano. 2004. Query difficulty, robustness, and selective application of query expansion. In *European Conference on Information Retrieval*. Springer, 127–137.

[3] Avi Arampatzis and Jaap Kamps. 2008. A study of query length. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'08)*. ACM, New York, NY, 811–812. DOI: https://doi.org/10.1145/1390334.1390517

[4] Julie Ayter, Adrian Chifu, Sébastien Déjean, Cecile Desclaux, and Josiane Mothe. 2015. Statistical analysis to establish the importance of information retrieval parameters. *J. Univ. Comput. Sci. Inf. Retriev. Recommend.* 21, 13 (2015) (2015), 1767–1789.

[5] Alain Baccini, Sébastien Déjean, Laetitia Lafage, and Josiane Mothe. 2012. How many performance measures to evaluate information retrieval systems? *Knowl. Inf. Syst.* 30, 3 (2012).

[6] David Banks, Paul Over, and Nien-Fan Zhang. 1999. Blind men and elephants: Six approaches to TREC data. *Inf. Retriev.* 1, 1–2 (May 1999), 7–34. DOI: https://doi.org/10.1023/A:1009984519381

[7] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* 13 (Feb. 2012), 281–305.

[8] Anthony Bigot, Claude Chrisment, Taoufiq Dkaki, Gilles Hubert, and Josiane Mothe. 2011. Fusing different information retrieval systems according to query-topics: A study based on correlation in information retrieval systems and TREC topics. *Inf. Retriev.* 14, 6 (2011), 617.

[9] Anthony Bigot, Sébastien Déjean, and Josiane Mothe. 2015. Learning to choose the best system configuration in information retrieval: The case of repeated queries. *J. Univ. Comput. Sci.* 21, 13 (2015), 1726–1745.

[10] Leo Breiman. 2001. Random forests. *Mach. Learn.* 45, 1 (Oct. 2001), 5–32.

[11] Chris Buckley and Gerard Salton. 1995. Optimization of relevance feedback weights. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, 351–357.

[12] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*. ACM, New York, NY, 89–96.

[13] Guihong Cao, Jian-Yun Nie, Jianfeng Gao, and Stephen Robertson. 2008. Selecting good expansion terms for pseudo-relevance feedback. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'08)*. ACM, New York, NY, 243–250. DOI: https://doi.org/10.1145/1390334.1390377

[14] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*. ACM, New York, NY, 129–136.

[15] David Carmel and Elad Yom-Tov. 2010. Estimating the query difficulty for information retrieval. *Synth. Lect. Inf. Concepts Retriev. Serv.* 2, 1 (2010), 1–89.

[16] Claudio Carpineto and Giovanni Romano. 2012. A survey of automatic query expansion in information retrieval. *ACM Comput. Surv.* 44, 1 (2012), 1.

[17] Olivier Chapelle, Yi Chang, and Tie-Yan Liu. 2011. Future directions in learning to rank. In *Yahoo! Learning to Rank Challenge (Proceedings of Machine Learning Research)*, Olivier Chapelle, Yi Chang, and Tie-Yan Liu (Eds.). Vol. 14. PMLR, Haifa, Israel, 91–100.

[18] Olivier Chapelle and S. Sathiya Keerthi. 2010. Efficient algorithms for ranking with SVMs. *Inf. Retriev.* 13, 3 (2010), 201–215.

[19] Adrian Chifu, Lá Laporte, Josiane Mothe, and Zia Ullah Md. 2018. Query performance prediction focused on summarized letor features. In *Proceedings of the 41th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'18)*. ACM.

[20] Charles L. A. Clarke, Nick Craswell, and Ian Soboroff. 2004. Overview of the TREC 2004 terabyte track. In *Proceedings of the Text REtrieval Conference (TREC'04)*, Vol. 4. 74.

[21] Kevyn Collins-Thompson, Paul Bennett, Charles L. A. Clarke, Fernando Diaz, and Ellen M. Voorhees. 2014. *TREC 2013 Web Track Overview*. Technical Report. University of Michingan at Ann Arbor.

[22] Kevyn Collins-Thompson, Craig Macdonald, Paul Bennett, Fernando Diaz, and Ellen M. Voorhees. 2015. *TREC 2014 Web Track Overview*. Technical Report. University of Michigan at Ann Arbor.

[23] Jonathan Compaoré, Sébastien Déjean, Adji Maŕram Gueye, Josiane Mothe, and Joelson Randriamparany. 2011. Mining information retrieval results: Significant IR parameters. *Adv. Inf. Min. Manage.* (Oct. 2011).

[24] Steve Cronen-Townsend, Yun Zhou, and W. Bruce Croft. 2002. Predicting query performance. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'02)*. ACM, New York, NY, 299–306. DOI: https://doi.org/10.1145/564376.564429

[25] Steve Cronen-Townsend, Yun Zhou, and W. Bruce Croft. 2004. A framework for selective query expansion. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management*. ACM, New York, NY, 236–237.

[26] Romain Deveaud, Josiane Mothe, and Jian-Yun Nie. 2016. Learning to rank system configurations. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, New York, NY, 2001–2004.

[27] Bekir Taner Dinçer. 2007. Statistical principal components analysis for retrieval experiments: Research articles. *J. Assoc. Inf. Sci. Technol.* 58, 4 (Feb. 2007), 560–574. DOI: https://doi.org/10.1002/asi.v58:4

[28] Francis C. Fernández-Reyes, Jorge Hermosillo-Valadez, and Manuel Montes-y Gómez. 2018. A prospect-guided global query expansion strategy using word embeddings. *Inf. Process. Manage.* 54, 1 (2018), 1–13.

[29] Nicola Ferro. 2017. What does affect the correlation among evaluation measures? *ACM Trans. Inf. Syst.* 36, 2 (2017), 19:1–19:40. DOI: https://doi.org/10.1145/3106371

[30] Nicola Ferro and Gianmaria Silvello. 2016. The CLEF monolingual grid of points. In *International Conference of the Cross-Language Evaluation Forum for European Languages*. Springer, 16–27.

[31] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. 2003. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.* 4 (Nov. 2003), 933–969.

[32] Jerome H. Friedman. 2000. Greedy function approximation: A gradient boosting machine. *Ann. Stat.* 29 (2000), 1189–1232.
[33] Johannes Fürnkranz and Eyke Hüllermeier. 2011. *Preference Learning*. Springer.
[34] Nicolo Fusi and Huseyn Melih Elibol. 2017. Probabilistic matrix factorization for automated machine learning. *arXiv preprint arXiv:1705.05355* (2017).
[35] Parantapa Goswami, Eric Gaussier, and Massih-Reza Amini. 2017. Exploring the space of information retrieval term scoring functions. *Inf. Process. Manage.* 53, 2 (2017), 454–472.
[36] Donna Harman. 1992. Relevance feedback revisited. In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, 1–10.
[37] Donna Harman and Chris Buckley. 2009. Overview of the reliable information access workshop. *Inf. Retriev.* 12, 6 (18 Jul. 2009), 615. DOI: https://doi.org/10.1007/s10791-009-9101-4
[38] Claudia Hauff, Djoerd Hiemstra, and Franciska de Jong. 2008. A survey of pre-retrieval query performance predictors. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM'08)*. ACM, New York, NY, 1419–1420. DOI: https://doi.org/10.1145/1458082.1458311
[39] David Hawking. 2000. Overview of the TREC-9 web track. In *Proceedings of the Text REtrieval Conference (TREC'00)*. National Institute of Standards and Technology.
[40] Ben He and Iadh Ounis. 2006. Query performance prediction. *Inf. Syst.* 31, 7 (2006), 585–594.
[41] Djoerd Hiemstra. 2001. *Using Language Models for Information Retrieval*. Univ. Twente. I–VIII, 1–163 pages.
[42] Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 133–142.
[43] K. L. Kwok, Laszlo Grunfeld, and M. Chan. 1999. TREC-8 Ad-Hoc, query and filtering track experiments using PIRCS. In *Proceedings of the Text REtrieval Conference (TREC'99)*.
[44] Hanjiang Lai, Yan Pan, Cong Liu, Liang Lin, and Jie Wu. 2013. Sparse learning-to-rank via an efficient primal-dual algorithm. *IEEE Trans. Comput.* 62, 6 (2013), 1221–1233.
[45] Léa Laporte, Rémi Flamary, Stéphane Canu, Sébastien Déjean, and Josiane Mothe. 2014. Nonconvex regularizations for feature selection in ranking with sparse svm. *IEEE Trans. Neur. Netw. Learn. Syst.* 25, 6 (2014), 1118–1130.
[46] Victor Lavrenko and W. Bruce Croft. 2001. Relevance based language models. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'01)*. ACM, New York, NY, 120–127. DOI: https://doi.org/10.1145/383952.383972
[47] Yuan Lin, Jiajin Wu, Bo Xu, Kan Xu, and Hongfei Lin. 2017. Learning to rank using multiple loss functions. *Int. J. Mach. Learn. Cybernet.* (2017), 1–10. DOI: https://doi.org/10.1007/s13042-017-0730-4
[48] Tie-Yan Liu. 2009. Learning to rank for information retrieval. *Found. Trends Inf. Retriev.* 3, 3 (2009), 225–331.
[49] Tie-Yan Liu. 2011. *Learning to Rank for Information Retrieval*. Springer Science & Business Media.
[50] Meili Lu, Xiaobing Sun, Shaowei Wang, David Lo, and Yucong Duan. 2015. Query expansion via wordnet for effective code search. In *Proceedings of the 2015 IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER'15)*. IEEE, 545–549.
[51] Craig Macdonald, Rodrygo L. T. Santos, Iadh Ounis, and Ben He. 2013. About learning models with multiple query-dependent features. *ACM Trans. Inf. Syst.* 31, 3 (2013), 11.
[52] Craig Macdonald, Rodrygo L. T. Santos, and Iadh Ounis. 2013. The whens and hows of learning to rank for web search. *Inf. Retriev.* 16, 5 (2013), 584–628.
[53] Michael McCandless, Erik Hatcher, and Otis Gospodnetic. 2010. *Lucene in Action: Covers Apache Lucene 3.0*. Manning Publications Co.
[54] Donald Metzler and W. Bruce Croft. 2007. Linear feature-based models for information retrieval. *Inf. Retriev.* 10, 3 (2007), 257–274.
[55] George A. Miller. 1995. WordNet: A lexical database for english. *Commun. ACM* 38, 11 (1995), 39–41.
[56] Stefano Mizzaro and Stephen Robertson. 2007. Hits hits TREC: Exploring IR evaluation results with network analysis. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 479–486. DOI: http://doi.acm.org/10.1145/1277741.1277824
[57] Serge Molina, Josiane Mothe, Dorian Roques, Ludovic Tanguy, and Md Zia Ullah. 2017. IRIT-QFR: IRIT query feature resource. In *International Conference of the Cross-Language Evaluation Forum for European Languages*. Springer, 69–81.
[58] Josiane Mothe and Ludovic Tanguy. 2005. Linguistic features to predict query difficulty. In *Proceedings of the Workshop in ACM SIGIR Conference on Research and Development in Information Retrieval*.
[59] Josiane Mothe and Mahdi Washha. 2017. Predicting the best system parameter configuration: The (per parameter learning) PPL method. In *Proceedings of the 21st International Conference on Knowledge-Based and Intelligent Information & Engineering Systems*.
[60] Paul Ogilvie, Ellen Voorhees, and Jamie Callan. 2009. On the number of terms used in automatic query expansion. *Inf. Retriev.* 12, 6 (26 Jul. 2009), 666. DOI: https://doi.org/10.1007/s10791-009-9104-1

[61] Iadh Ounis, Giambattista Amati, Vassilis Plachouras, B. He, C. Macdonald, and C. Lioma. 2006. Terrier: A high performance and scalable information retrieval platform. *SIGR Workshop on Open Source Information Retrieval.*

[62] Jay M. Ponte and W. Bruce Croft. 1998. A language modeling approach to information retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98).* ACM, New York, NY, 275–281.

[63] Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. 2010. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Inf. Retriev.* 13, 4 (2010), 346–374.

[64] Alessandro Raganato, Jose Camacho-Collados, and Roberto Navigli. 2017. Word sense disambiguation: A unified evaluation framework and empirical comparison. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, Vol. 1. 99–110.

[65] Stephen E. Robertson, Steve Walker, Micheline Beaulieu, and Peter Willett. 1999. Okapi at TREC-7: Automatic ad hoc, filtering, VLC and interactive track. *NIST Special Publication SP*500 (1999), 253–264.

[66] Stephen E. Robertson, Steve Walker, Micheline Hancock-Beaulieu, Aarron Gull, and Marianna Lau. 1992. Okapi at TREC. In *Proceedings of The 1st Text REtrieval Conference (TREC 1992)*, Donna K. Harman (Ed.). 500-207. National Institute of Standards and Technology, 21–30.

[67] Haggai Roitman, Shai Erera, Oren Sar Shalom, and Bar Weiner. 2017. Enhanced mean retrieval score estimation for query performance prediction. In *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval (ICTIR'17).* 35–42. DOI: https://doi.org/10.1145/3121050.3121051

[68] Anna Shtok, Oren Kurland, and David Carmel. 2009. Predicting query performance by query-drift estimation. In *Proceedings of the Conference on the Theory of Information Retrieval.* Springer, 305–312.

[69] Anna Shtok, Oren Kurland, David Carmel, Fiana Raiber, and Gad Markovits. 2012. Predicting query performance by query-drift estimation. *ACM Trans. Inf. Syst.* 30, 2, Article 11 (2012), 35 pages.

[70] Trevor Strohman, Donald Metzler, Howard Turtle, and W. Bruce Croft. 2005. Indri: A language model-based search engine for complex queries. In *Proceedings of the International Conference on Intelligent Analysis*, Vol. 2. 2–6.

[71] Kazunari Sugiyama, Kenji Hatano, and Masatoshi Yoshikawa. 2004. Adaptive web search based on user profile constructed without any effort from users. In *Proceedings of the 13th International Conference on World Wide Web.* ACM, 675–684.

[72] Niek Tax, Sander Bockting, and Djoerd Hiemstra. 2015. A cross-benchmark comparison of 87 learning to rank methods. *Inf. Process. Manage.* 51, 6 (2015), 757–772.

[73] Michael Taylor, Hugo Zaragoza, Nick Craswell, Stephen Robertson, and Chris Burges. 2006. Optimisation methods for ranking functions with multiple parameters. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management.* ACM, 585–593.

[74] Ellen M. Voorhees. 2005. The TREC robust retrieval track. In *ACM SIGIR Forum*, Vol. 39. ACM, 11–20.

[75] Qiang Wu, Christopher J. Burges, Krysta M. Svore, and Jianfeng Gao. 2010. Adapting boosting for information retrieval measures. *Inf. Retriev.* 13, 3 (2010), 254–270.

[76] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise approach to learning to rank: Theory and algorithm. In *Proceedings of the 25th International Conference on Machine Learning.* ACM, 1192–1199.

[77] Bo Xu, Hongfei Lin, and Yuan Lin. 2016. Assessment of learning to rank methods for query expansion. *J. Assoc. Inf. Sci. Technol.* 67, 6 (2016), 1345–1357.

[78] Jun Xu and Hang Li. 2007. Adarank: A boosting algorithm for information retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.* ACM, 391–398.

[79] Elad Yom-Tov, Shai Fine, David Carmel, and Adam Darlow. 2005. Learning to estimate query difficulty: Including applications to missing content detection and distributed information retrieval. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'05).* ACM, New York, NY, 512–519. DOI: https://doi.org/10.1145/1076034.1076121

[80] Chengxiang Zhai and John Lafferty. 2001. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.* ACM, 334–342.

[81] Le Zhao and Jamie Callan. 2012. Automatic term mismatch diagnosis for selective query expansion. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval.* ACM, 515–524.

[82] Yun Zhou and W. Bruce Croft. 2007. Query performance prediction in web search environments. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'07).* ACM, New York, NY, USA, 543–550. DOI: https://doi.org/10.1145/1277741.1277835

[83] Justin Zobel. 2004. Writing up. In *Writing for Computer Science.* Springer, 137–156.