Dynamic Cutoff Prediction in Multi-Stage Retrieval Systems

J. Shane Culpepper
RMIT University
Melbourne, Australia
shane.culpepper@rmit.edu.au

Charles L. A. Clarke
University of Waterloo
Waterloo, Canada
claclarke@cs.uwaterloo.ca

Jimmy Lin
University of Waterloo
Waterloo, Canada
jimmylin@uwaterloo.ca

ABSTRACT

Modern multi-stage retrieval systems are comprised of a candidate generation stage followed by one or more reranking stages. In such an architecture, the quality of the final ranked list may not be sensitive to the quality of the initial candidate pool, especially in terms of early precision. This provides several opportunities to increase retrieval efficiency without significantly sacrificing effectiveness. In this paper, we explore a new approach to dynamically predicting the size of an initial result set in the candidate generation stage, which can directly affect the overall efficiency and effectiveness of the entire system. Previous work exploring this tradeoff has focused on global parameter settings that apply to all queries, even though optimal settings vary across queries. In contrast, we propose a technique that makes a parameter prediction to maximize efficiency within an effectiveness envelope on a *per query* basis, using only static pre-retrieval features. Experimental results show that substantial efficiency gains are achievable. In addition, our framework provides a versatile tool that can be used to estimate the effectiveness-efficiency tradeoffs that are possible *before* selecting and tuning algorithms to make machine-learned predictions.

Keywords

Experimentation; Measurement; Multi-Stage Retrieval; Query Prediction

1. INTRODUCTION

Effectiveness-efficiency tradeoffs have been extensively explored in search engine architectures: Highly-effective ranking models often take advantage of computationally expensive features and hence are slow, while fast ranking algorithms often sacrifice effectiveness. In a modern multi-stage ranking architecture [1, 2, 3, 7, 22, 23, 26], an initial candidate generation stage is followed by one or more rerankers, and the end-to-end effectiveness-efficiency tradeoffs are often a combination of many different component-level tradeoffs. In this work, we focus on the initial candidate generation stage, whose responsibility is to provide an initial set of documents that are then considered in more detail, for example, by machine-learned rankers [5, 6]. Previous work [3] has shown that the quality of the final ranked list is relatively insensitive to the quality of the initial candidate set, especially in terms of early precision. The intuition is as follows: as long as the candidate generation stage can supply a "reasonable" pool of documents, it is likely that later-stage rankers can identify the best documents and place them in high ranking positions, regardless of the original rank scores. If the final ranked list is assessed in terms of, say, NDCG@10, the initial candidate pool only needs to contain ten documents of the highest relevance grade to achieve the best possible score—provided that the later-stage rankers identify these documents (which is largely an orthogonal issue).

In this work, we explore how to dynamically predict the size of the candidate pool k. In a standard document-at-a-time query evaluation algorithm, query evaluation latency increases as a function of k. A large candidate document pool also means greater cost in the feature extraction and reranking stages downstream. Thus, we desire a k as small as possible while remaining within an effectiveness envelope.

Our Contributions. The key contribution of our work is to show that we can achieve substantial savings in candidate generation efficiency in multi-stage ranking *without sacrificing effectiveness*, tuned on a *per query* basis, using only static pre-retrieval features. We accomplish this by building classifier cascades that make binary decisions at several different cutoffs along an effectiveness-efficiency tradeoff curve. In addition, a key feature of our approach, worth emphasizing, is that we are able to train these classifier cascades *without requiring relevance judgments*, which overcomes a limitation with previous studies since relevance judgments restrict the scope of their experiments to at most a few hundred queries (at least in the academic context). In contrast, we are able to run experiments on tens of thousands of queries. This can be achieved by leveraging a recently-introduced evaluation technique called Maximized Effectiveness Difference (MED) [11, 31]. Our experimental results show substantial improvements over selecting a fixed k without lowering effectiveness, which can translate directly into efficiency improvements in later stage reranking.

2. BACKGROUND

We assume a standard formulation of the ranked retrieval problem, where given a user query q, our goal is to return a ranked list that maximizes a particular metric. In the web context, the metric would likely emphasize early precision, e.g., NDCG@10. In this section, we discuss tradeoffs between effectiveness and efficiency in the context of multi-stage ranking.

2.1 Multi-Stage Ranking Efficiency

Multi-stage retrieval systems have become the dominant model for efficient and effective web search [1, 2, 3, 7, 22, 23, 26]. The

ADCS '16, December 05 - 07, 2016, Caulfield, VIC, Australia

ISBN 978-1-4503-4865-2/16/12…$15.00

DOI: http://dx.doi.org/10.1145/3015022.3015026

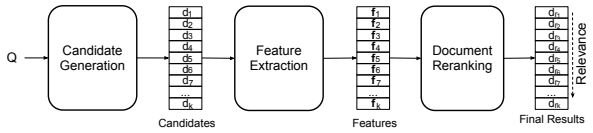

Figure 1: Illustration of a multi-stage retrieval architecture with distinct candidate generation, feature extraction, and document reranking stages.

key idea of this approach is to efficiently generate a set of candidate documents that are likely to be relevant to a query, and then iteratively reorder the documents using a series of more expensive machine learning techniques. As the cost of the later stage reordering can be computationally expensive, minimizing the number of candidate documents in early stage retrieval can yield significant benefits in overall query processing time. Kohavi et al. [17] stated that every 100 ms boost in search speed increases revenue by 0.6% at Bing. Thus, even small gains in overall performance can translate into tangible benefits in commercial search engines.

The simplest example of a multi-stage retrieval architecture is shown in Figure 1, from Asadi and Lin [2]. The input to the candidate generation stage is a query q and the output is a set of k document ids $\{d_1, d_2, \ldots d_k\}$. In principle, the candidates can be treated as a list or a set—the difference is whether subsequent stages take advantage of the document score or ranking. These document ids serve as input to the feature extraction stage, which returns a list of k feature vectors $\{\mathbf{f}_1, \mathbf{f}_2, \ldots \mathbf{f}_k\}$, each corresponding to a candidate document. These serve as input to the document reranking stage, which typically applies a machine-learned model to produce the final ranking. Of course, there can be an arbitrary number of reranking stages. For example, Pedersen [26] describes a four-stage retrieval architecture in Bing, shown in Figure 2. The key take-away message is that increasingly expensive reranking steps may require processing fewer and fewer documents.

It is important to emphasize that the size of the candidate pool of documents k is independent of the size of the final ranked list (with only the hard constraint that the final size cannot be greater than k). So, what is the proper setting of k? The work of Macdonald et al. [22] suggests several different answers: "tens of thousands" (Chappelle and Chang [10]), $5{,}000$ (Craswell et al. [12]), $1{,}000$ (Qin et al. [29]), or smaller samples such as 200 (Zhang et al. [34]) or even 20 (Cambazoglu et al. [7]). Of course, the larger the k, the slower the system, in two respects: First, in standard document-at-a-time query evaluation algorithms that would provide the initial candidate documents (e.g., WAND), k has a direct impact on query latency, since a larger heap needs to be maintained, providing fewer opportunities for early exits, depending on document score distributions. Second, for every document in the candidate pool, we need to run the feature extractors to serve as input to the subsequent reranking stages (see Figure 1). Thus, from an efficiency perspective, it is clear that we desire the smallest possible k that allows end-to-end effectiveness to remain within some bounded envelope (see below for more details).

Note that our work focuses on the size of the candidate pool for the purposes of ranking at query time. In contrast, Macdonald et al. [22] focused on the importance of candidate pool size for the *training* of learning-to-rank systems. In particular, they looked at how the size of the candidate set affects the final results, arguing that the relationship is dependent on the type of information need. They show that as few as 10-20 documents may be needed for TREC 2009 and 2010 Web Track queries, but as many as $1{,}500$ may be needed for other queries in the same corpus. Since we are primarily concerned with the application of machine-learned models at query time and are not directly concerned with model training, the work of Macdonald et al. [22] is orthogonal to our study.

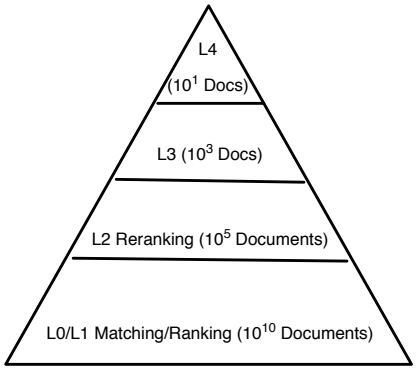

Figure 2: A four-stage retrieval architecture originally described by Pedersen [26].

The closest related work to ours is that of Tonellotto et al. [32], who also attempt to tune effectiveness-efficiency tradeoffs on a per query basis using query difficulty and query efficiency prediction techniques. However, their choice of settings is rather coarse grained: they only select between two configurations, whereas our classifier cascades are able to consider many more settings. Furthermore, their work exhibits the same limitation as most previous studies in requiring relevance judgments for training, and hence they are only able to experiment on 150 queries from TREC 2009–2011. In contrast, since our approach does not require any relevance judgments, we can tune our techniques on tens of thousands of queries, as we will discuss next.

2.2 Multi-Stage Ranking Effectiveness

Tuning ranking parameters requires substantial amounts of training data to measure effectiveness. Fortunately, in the case of tuning candidate generation for a second-stage ranker (see Figure 1), we have this training data readily available, since the second stage itself may be enlisted to provide it. To create this training data we first run the second-stage ranker over a very large candidate set, much larger than time might allow for interactive search. Conceptually this candidate set might be the entire collection, but practically it will be limited to a subset retrieved from query keyword matches and other simple features. Ideally, this set would contain all relevant documents, but mixed together with many non-relevant documents.

The second-stage ranker then ranks this set, producing a ranked list A. Given the potential size of the set, producing this ranking may take substantial time. However, while this time may be far greater than would be tolerable for interactive searching, when creating training data, time is not a problem.

Now, suppose we have a more efficient candidate generation algorithm, designed to feed this second-stage ranker. It produces a much smaller set, which can be more efficiently ranked by the second stage to produce a ranked list B. We measure the effectiveness of the candidate generation algorithm according to its ability to *supply the documents that the second stage needs* in the absence of efficiency constraints (A in this case). More specifically, we compute a rank correlation coefficient or rank similarity measure between A and B, using its value $S(A, B)$ as our effectiveness metric.

Naturally, the similarity measure must be suitable for this purpose [33]. In particular, a rank similarity measure for search results must be appropriately *top-weighed*, placing greater emphasis on

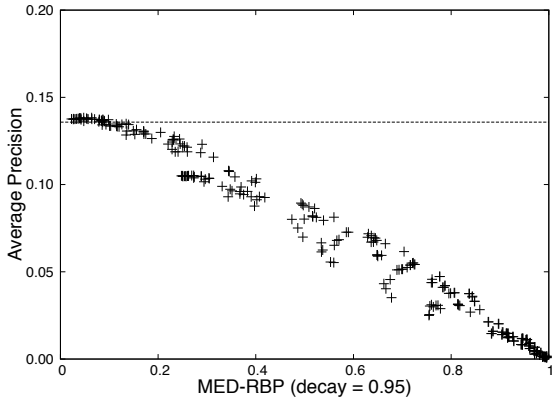

Figure 3: Correlation between $\mathsf{MED}_{\mathsf{RBP}}$ and measured average precision using the 50 queries of the TREC 2010 Web Track adhoc collection and the `IvoryL2Rb` experimental run as the second stage. Each of the 279 points represents one of 31 distinct first stages, across a range of parameter settings. The dashed line indicates the effectiveness of the second stage with no early-stage filter. Reproduced from Tan and Clarke [31].

earlier ranks than on later ranks. If the top document in A is missing from B, the impact on the user will be much greater than if the 100th document is missing.

Tan and Clarke [31] describe a family of rank similarity measures specifically intended for comparing ranked lists produced by search engines. Given a traditional effectiveness metric — such as MAP [30], RBP [25], DCG [15], or ERR [9] — Tan and Clarke define a distance measure between two ranked lists in terms of that metric, as follows: "Given two ranked lists, A and B, what is the maximum difference in their effectiveness scores possible under [that metric]."

They call this family of distance measures *maximized effectiveness difference* (MED(A, B)) and develop variants corresponding to several standard effectiveness metrics — including $\mathsf{MED}_{\mathsf{MAP}}$, $\mathsf{MED}_{\mathsf{RBP}}$, $\mathsf{MED}_{\mathsf{DCG}}$, and $\mathsf{MED}_{\mathsf{ERR}}$. They explore MED as a method for quantifying changes to ranking algorithms without the need for human relevance judgments. For example, MED allows a search to identify queries for which a proposed change causes the greatest impact. An open-source implementation is publicly available online,[1] which can be used to compute MED for various effectiveness measures, and is used in this paper.

Building on this work, we have recently applied MED to measure effectiveness of the initial stages in multi-stage rankers [11]. That work follows the procedure outlined above, using a second-stage ranking as a gold standard to measure first-stage effectiveness, validating this procedure. Unlike previous explorations of effectiveness-efficiency tradeoffs, the absence of any requirement for human relevance judgments allows the procedure to be easily applied across tens of thousands of queries.

For illustration purposes, Figure 3 is reproduced from that paper. The figure shows the performance of a number of first-stage rankers, operating over a range of parameter settings, supplying a high-quality second-stage ranker. The horizontal dashed line indicates the effectiveness of the second-stage ranker without first-stage filtering. Values of $\mathsf{MED}_{\mathsf{RBP}}$ below 0.05 produce no practical loss in measured effectiveness.

In this range, MED is measuring shortcomings in the first-stage that are not necessarily reflected in the evaluation measures. A first-stage ranker that fails to return the top document required by the second stage will receive a lower MED score than a first-stage ranker that fails to return the sixth document. While other relevant documents might move up to replace the lost documents, leaving both with the same measured effectiveness, losing the top-ranked document is penalized by the metric more than losing lower-ranked documents.

Previous work explored effectiveness-efficiency tradeoffs of first-stage algorithms and their parameter settings as applied uniformly across all queries [11]. However, optimal algorithms and settings vary across queries. In this paper, we explore a technique for optimizing effectiveness-efficiency tradeoffs on a per-query basis, selecting the optimal algorithm and setting for each using static, pre-retrieval features. While we focus specifically on the two-stage architecture in Figure 1, our methods should generalize to larger multi-stage architectures, such as the one shown in Figure 2, with the effectiveness of each stage measured in terms of the next.

3. APPROACH

Feature Selection. Simple term features have been used successfully in a variety of different learning-to-rank scenarios [18, 20, 22, 23] and in query difficulty prediction [8, 16, 21]. Across all of these studies one general theme has emerged — a mixture of similarity measures and query-specific score aggregation techniques yield the most benefit. Inspired by previous work, we adopt this philosophy in our feature choices as well. We use three simple similarity measures in this work: BM25, TF·IDF, and query likelihood.

These similarity formulations were used since each can easily be precomputed for all term–document combinations and treated as independent term-specific features. In addition to the three similarity scoring regimes, we also adopt several different score aggregation techniques, and compute a variety of static statistical features for each term posting: maximum score, arithmetic mean of scores, harmonic mean of scores, median of scores, variance of scores, first quartile score, third quartile score, and the number of documents containing the term. Additional query-specific features are also incorporated into the model including query length, minimum and maximum feature score across all terms in the query, and means (arithmetic and harmonic) of the query-specific term scores. Table 2 provides a comprehensive breakdown of the term-specific features, each of which can be computed at index time. Table 3 shows how each of the term-specific features are combined into the final feature set used by the classifier. A total of 70 features are used in our work.

Labeling Instances. We now turn our attention to how the training collection was created. One of the key ideas of this work is to use MED to determine a minimal candidate set that also maximizes the possible effectiveness in the final reranking stage. In order to achieve this, we have created a gold standard set using 40,000 queries from the 2009 TREC Million Query Track. For each query, $\mathsf{MED}_{\mathsf{RBP}}$, $\mathsf{MED}_{\mathsf{ERR}}$, and $\mathsf{MED}_{\mathsf{DCG}}$ are computed for the k values of 20, 50, 100, 200, 500, 1,000, 2,000, 5,000, and 10,000. Our gold standard run for tuning k was the `uogTRMQdph40` run, as it represents one of the top-scoring systems that returned results for all 40,000 of the MQ2009 queries (when measured over the small subset of the queries that were evaluated).

[1] https://github.com/claclark/MED

Topic	k								
	20	50	100	200	500	1,000	2,000	5,000	10,000
20001	0.544	0.346	0.104	0.056	**0.010**	0.002	0.001	0.000	0.000
20002	0.536	0.142	0.053	**0.016**	0.002	0.000	0.000	0.000	0.000
20003	0.865	0.856	0.810	0.773	0.706	0.684	0.582	0.122	**0.000**
20004	0.999	0.944	0.132	**0.070**	0.018	0.008	0.008	0.000	0.000

Table 1: The $\mathsf{MED}_{\mathsf{RBP}}$ scores for the first four topics in the TREC MQ2009 collection at 9 different cutoffs for k. If a maximum loss of 0.05 is the target cutoff, then the boldface values represent the k "class" for each topic.

Term Statistics
1. Number of documents containing term t (f_t).
2. Maximum Similarity Score
3. First Quartile Similarity Score
4. Third Quartile Similarity Score
5. Arithmetic Mean of Similarity Scores
6. Harmonic Mean of Similarity Scores
7. Median of Similarity Scores
8. Variance of Similarity Scores

Table 2: Query-independent term features used by the classifier. Each feature can be precomputed and stored with the postings list.

Query Features (Score Dependent)
1. Arithmetic Mean of t_f
2. Harmonic Mean of Maximum Scores
3. Arithmetic Mean of Maximum Scores
4. Arithmetic Mean of Median Score
5. Arithmetic Mean of Mean Scores
6. Arithmetic Mean of Score Variances
7. Arithmetic Mean of Score Interquartile Ranges
8. Minimum Score of terms in the query for each feature in Table 2.
9. Maximum Score of terms in the query for each feature in Table 2.
Query Features (Score Independent)
1. Query Length

Table 3: Query-specific features used by the classifier. All score-dependent features can be computed on the fly for all three similarity metrics at query time using prestored values in Table 2.

Thus, in total we computed MED using three different metrics at 9 distinct cutoffs for k. To label the instances, we now select a sufficiently low value of a given metric, say $\mathsf{MED}_{\mathsf{RBP}} \leq 0.05$, and choose the minimal cutoff that satisfies this constraint — this is what we have previously referred to as the "effectiveness envelope" we would like to maintain.

For example, consider the $\mathsf{MED}_{\mathsf{RBP}}$ computations for the first four topics shown in Table 1. If the minimal acceptable score is $\mathsf{MED}_{\mathsf{RBP}} \leq 0.05$, then for Topic 20001, the nominal class assigned would be $k = 500$, whereas Topic 20002 can achieve a similar MED score with $k = 200$.

Multilabel Classification and Regression. The most obvious solution from a machine learning perspective is to train a multilabel classifier or to use regression. We have explored both possibilities in our early empirical analysis but found that neither approach was reliably better than using a fixed cutoff baseline.

Algorithm 1 LRCASCADE

Input: A query q, a minimum confidence threshold t, and a set of $c-1$ binary classifiers $\mathcal{C}$
Output: A cutoff prediction between 1 and c.

```
for i = 1 to c − 1 do
    p ← PREDICT(C_i, q)
    if p = 0 and Pr(p) > t then
        Return i
    end if
end for
Return c
```

After careful examination of the preliminary results, a clear constraint emerged in producing good results in our classifier: any under-prediction can significantly hurt overall effectiveness, and thus should be avoided. A standard approach to reweight classification is to use a cost-sensitive classifier [14] such as MetaCost [13]. Our experiments with a cost-sensitive classifier that penalized the classifier for under-predicting were more promising than regression or standard multilabel classification, but still not better than using a fixed baseline.

There has also been recent work on building cost-sensitive regression algorithms [35], but this is still an active area of research and beyond the scope of our work. One limitation of regression is that the MED values must be computed for every value of k, and out-of-the-box loss functions do not produce competitive results. Clearly, regression is a promising alternative to the approach we have developed, but we leave this to future work as it will require a custom loss function in order to achieve similar results to the cascade classifier that we describe shortly. Instead, we extend another common technique in regression — choosing a fixed threshold and creating a binary classifier. However, we found that a single threshold was not sufficient for our needs, and that the approach could be extended to make a series of binary predictions to find the best cutoffs. We explain the mechanics of this technique next.

Cascaded Classification. Our approach to prediction relies on a cascade of binary classifiers. Since classes are ordinal and should be treated as such, a series of binary predictions can be used to find the minimum cutoff for each query that also maximizes the overall effectiveness in the final document reordering stage. In this work, a random forest classifier [4] is trained and used for predictions at each stage of the cascade. Before building the classifier, training sets can be created from a multilabeled class set. The number of binary classifiers required is $c-1$, where c is the maximum ordinal label. Labels are monotonically increasing from 1 to c.

Once the binary classifiers are constructed, it is relatively simple to make a prediction for any query q. A feature set can easily be constructed at query parsing time which is then used by Algorithm 1 to assign a cutoff for the query. A left-to-right cascade serves two important purposes. First, the model implicitly mini-

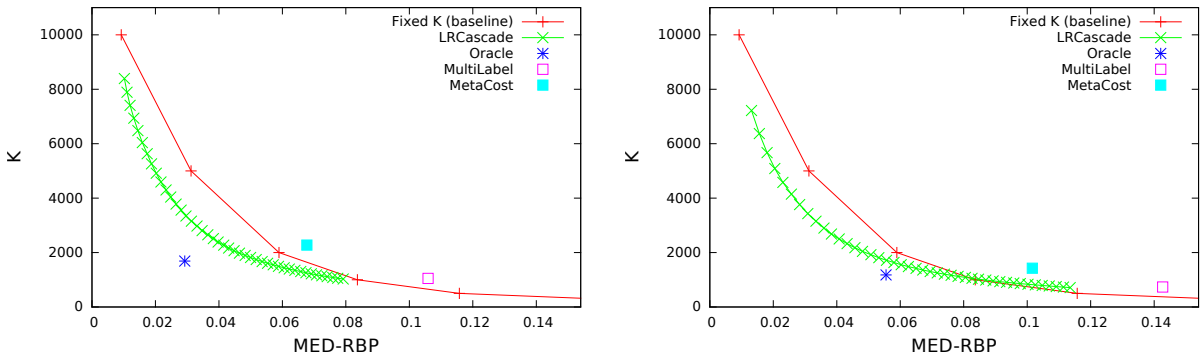

Figure 5: $\mathsf{MED}_{\mathsf{RBP}}$ versus k using a training threshold cutoff of $\mathsf{MED}_{\mathsf{RBP}} \leq 0.05$ (left panel) and $\mathsf{MED}_{\mathsf{RBP}} \leq 0.10$ (right panel) for the MQ2009 queries on ClueWeb09B. The blue star represents the best possible result achievable with a perfect classifier, the green line is the result using our LR Binary Cascade Model, and the red line represents the tradeoff horizon based on using a fixed k (baseline) for all queries.

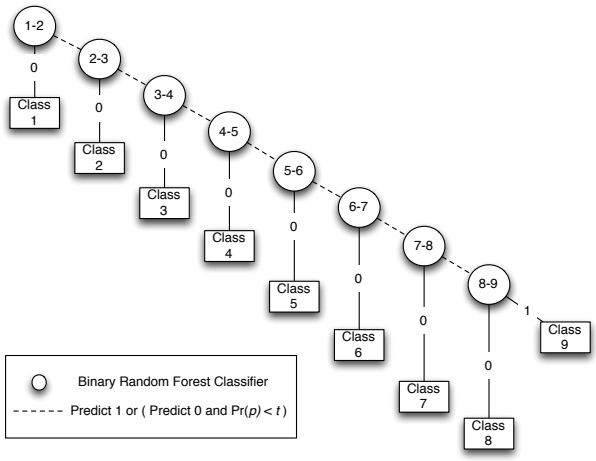

Figure 4: A left-to-right nine class cascade. Each node is a binary, random forest classifier. If the classifier predicts 0 with a probability $Pr(p) > t$, the current node ID is written as the class. Otherwise, the instance is passed to the next classifier in the chain.

mizes the likelihood of a false positive as assignments are made smallest to largest, and exits only occur for high probability predictions. Second, each prediction has a small cost. In a left-to-right cascade, queries with the smallest cutoff incur the least amount of processing time. If a larger cutoff is required, the cost of extra predictions is small relative to the cost of more expensive reordering stages of large candidate sets later in the scoring process.

Figure 4 shows an example of a left-to-right nine class cascade of binary classifiers. Each node in the tree is a binary random forest classifier pre-trained using one of $\mathcal{B}$ training sets. By increasing the cutoff threshold t, the percentage of under-predictions is decreased at the cost of increasing the percentage of over-predictions. However, some level of over-prediction is acceptable as this always results in a gradual increase in overall effectiveness.

4. EXPERIMENTS

Experimental Configuration. For all experiments, 40,000 queries from the 2009 Million Query Track (MQ2009) were used with a stopped and unpruned ClueWeb 2009 category B index (CW09B). The `uogTRMQdph40` system is used as the gold standard, as it represents one of the top-scoring systems that returned results for all 40,000 MQ2009 queries (measured over the small subset of the queries that were evaluated). Specifically, this is the highest scoring system that submitted results for all of the queries in MQ2009, making it the best choice as the gold standard in our work.

To generate the bag-of-words candidate run, a BM25 implementation [24, 27, 28] using the same formulation and parameterization as described in Section 3 was ran for all 40,000 MQ2009 queries.[2] The stopwords list and Krovetz stemmer were derived directly from the Indri[3] search engine. A total of 50,220,423 documents were indexed from the CW09B collection, and all queries were ran to a depth of 10,000.

The classifier was constructed as described in Section 3. For k, the values used were 20, 50, 100, 200, 500, 1,000, 2,000, 5,000, and 10,000. For each bucket, three different MED variants were computed: $\mathsf{MED}_{\mathsf{RBP}}$, $\mathsf{MED}_{\mathsf{ERR}}$, and $\mathsf{MED}_{\mathsf{DCG}}$. Standard ten fold cross-validation was used to generate the predictions. Note that before generating the final folds, we removed all queries for which we had any judgments (687 topics) for further validation purposes.

Dynamic Selection of k. Our first set of experiments were designed to test the hypothesis that a best k value can be determined on a query-by-query basis which *minimizes* effectiveness loss and *maximizes* efficiency. In other words, finding the smallest acceptable k for a target MED value can minimize the amount of work later stage rerankers must do and also minimizes the cost of using a safe-to-k candidate generation algorithm such as WAND. In order to test this hypothesis, we created several different datasets to train our predictor. We experimented with $\mathsf{MED}_{\mathsf{RBP}}$ using the cutoffs 0.02, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30, and 0.50. We also experimented with $\mathsf{MED}_{\mathsf{DCG}}$ using the cutoffs 0.2, 0.3, 0.5, 0.7, 1.00, 1.20, and 1.50. Experiments were also performed with $\mathsf{MED}_{\mathsf{ERR}}$ using the cutoffs 0.05, 0.10, 0.15, 0.20, 0.25, 0.30, and 0.50, and achieve similar results.

Figure 5 shows the tradeoff achievable between the candidate set size k and $\mathsf{MED}_{\mathsf{RBP}}$. The left pane is a summary of results when using a target of $\mathsf{MED}_{\mathsf{RBP}} \leq 0.05$, and the right pane summarizes performance for $\mathsf{MED}_{\mathsf{RBP}} \leq 0.10$. In both graphs, the red line represents the tradeoff horizon in efficiency and effectiveness when using a fixed cutoff for all queries, as is generally done in current system configurations. This is the baseline. The blue star

[2]http://github.com/jsc/WANDbl
[3]http://www.lemurproject.org/indri.php

Method	Interpolated MED_{RBP}				Interpolated k			
	Predicted MED_{RBP}	Predicted k	Fixed k	Difference in k	Predicted k	Predicted MED_{RBP}	Fixed MED_{RBP}	Difference in MED_{RBP}
Oracle	0.029	1,688	5,459	+223%	1,688	0.029	0.067	+128%
MultiLabel	0.106	1,053	653	– 38%	1,053	0.106	0.082	– 22%
MetaCost	0.068	2,277	1,644	– 28%	2,277	0.068	0.056	– 16%
LRCascade, $t = 0.75$	0.045	2,071	3,535	+ 71%	2,071	0.045	0.058	+ 30%
LRCascade, $t = 0.80$	0.036	2,656	4,432	+ 67%	2,656	0.036	0.053	+ 45%
LRCascade, $t = 0.85$	0.028	3,561	5,715	+ 61%	3,561	0.028	0.044	+ 59%

Table 4: Interpolated k and MED_{RBP} when training at $\text{MED}_{\text{RBP}} \leq 0.05$. The relative gain or loss for k and MED_{RBP} is shown when compared to using a fixed cutoff for all queries. The Oracle method represents the best possible result given a perfect classifier.

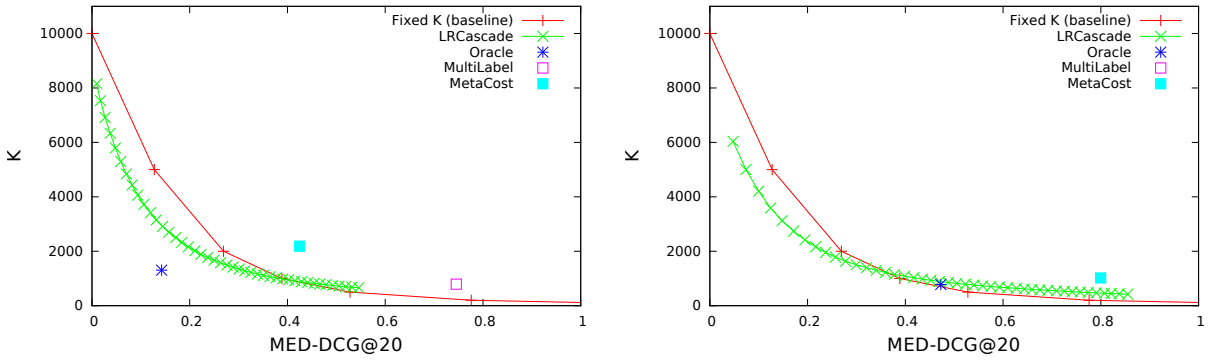

Figure 6: MED_{DCG} versus k using a training threshold cutoff of $\text{MED}_{\text{DCG}} \leq 0.50$ (left panel) and $\text{MED}_{\text{DCG}} \leq 1.0$ (right panel) for the MQ2009 queries on ClueWeb09B. The blue star represents the best possible result with a perfect classifier, the green line is the result using our LR Binary Cascade Model, and the red line represents the tradeoff horizon based on using a fixed k for all queries (the baseline).

represents the gold standard result that would be achievable with a perfect classifier. The two squares represent results using standard machine learning approaches: MultiLabel, a multilabel random forest classifier, and MetaCost [13], a cost-sensitive classifier.[4] When compared to the fixed baseline, we see that traditional approaches to classification do not provide any real benefit.

In contrast, the LRCascade approach (green line) shows clear improvements over both multilabel and fixed cutoff baseline approaches. Note that each point on the green line represents a fixed value of t between 0.50 and 0.99 in increments of 0.01. So, a value of 0.5 represents normal binary classification. As the value of t increases, greater confidence in the prediction is required before exiting the cascade. We found that in practice values of $t = 0.75$ up to 0.85 produce the best tradeoff.

The lower the choice of MED, the less likely there is any loss in effectiveness. Our experiments suggest that targeting low MED values is most likely to reap the most rewards. That is, minimizing the likelihood of any effectiveness loss provides further benefits over a variable cutoff approach.

Table 4 shows the breakdown for using a fixed k and predicted k interpolation when using a training set targeted at $\text{MED}_{\text{RBP}} \leq 0.05$. Columns 2–5 show the relative gains in terms of k, and columns 6–9 show the relative gains in terms of MED_{RBP}. The first row shows the gold standard Oracle result, which represents the best result that is achievable using this parameter – metric – target threshold combination. Changing any one of these three constraints will change the gain (or loss) possible. Computing the Oracle result is in itself interesting, as it provides a *bound* on how much benefit the three constraint combination could provide.

[4]http://www.cs.waikato.ac.nz/ml/weka/

The interpretation of the data in columns 2–5 is as follows: given a particular setting, how far *below* the interpolated fixed k curve (red) are we? That is, if we accept a particular level of MED_{RBP} effectiveness, how much efficiency can we gain over simply just adopting a fixed k cutoff for all queries (specifically, the k cutoff that would achieve the same level of MED_{RBP})? The interpretation of the data in columns 6–9 is as follows: given a particular setting, how far *left* of the interpolated fixed k curve (red) are we? That is, how much more effective (in terms of MED_{RBP}) can we make our results over simply setting a fixed k? In designing actual search architectures, the first interpretation is more intuitive, since we want to optimize efficiency without sacrificing effectiveness, but the alternative perspective is interesting as well in quantifying the benefits of our technique.

We see that both MultiLabel and MetaCost are marginally worse than a fixed cutoff, with MetaCost being the slightly better choice. The LRCascade method is the clear winner across a wide range of t. The exact value of t can be set depending on which direction a user wishes to bias the tradeoff. Choosing a lower t decreases the average k while increasing the average MED_{RBP}. Choosing a higher t biases the tradeoff in the effectiveness direction.

Figure 6 shows the same experiment when using $\text{MED}_{\text{DCG}} \leq 0.50$ and $\text{MED}_{\text{DCG}} \leq 1.00$. Changing the underlying evaluation metric does not change the general trends for all methods tested. Multilabel classifiers do not outperform fixed cutoffs, while the LRCascade is the superior tradeoff. We also ran a similar set of experiments using MED_{ERR} and obtained similar results and trends.

Figure 7 shows the percentage of queries which obtain a bound of $\text{MED}_{\text{RBP}} \leq 0.10$ or $\text{MED}_{\text{DCG}} \leq 0.50$. We can see that the LRCascade approach is clearly predicting cutoffs that have a lower

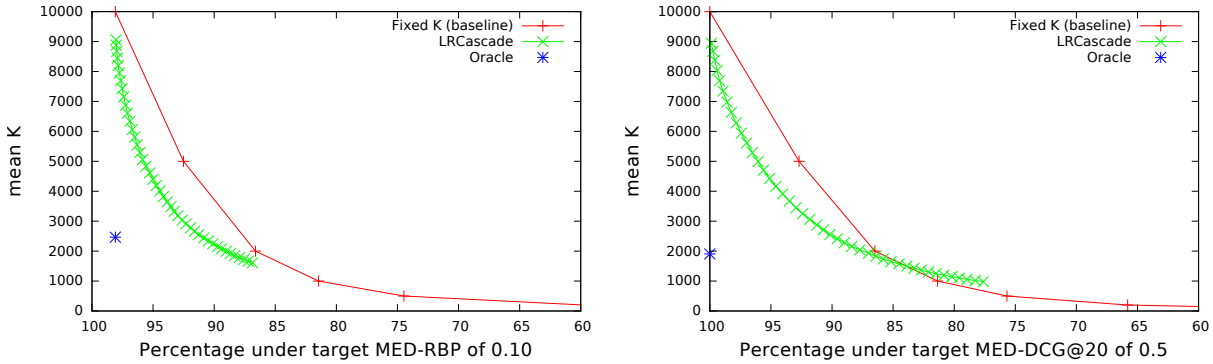

Figure 7: Average k versus the percentage of queries achieving a MED score less than a training target. The left panel shows the comparison of the LRCascade (green line) and a fixed cutoff baseline (red line) for $\text{MED}_{\text{RBP}} \leq 0.10$. The right panel shows the same comparison when using $\text{MED}_{\text{DCG}} \leq 0.50$.

Method	Interpolated MED_{ERR}				Interpolated k			
	Predicted MED_{ERR}	Predicted k	Fixed k	Difference in k	Predicted k	Predicted MED_{ERR}	Fixed MED_{ERR}	Difference in MED_{ERR}
Oracle	0.017	1,752	7,344	+319%	1,752	0.017	0.067	+292%
MultiLabel	0.117	1,159	452	– 61%	1,159	0.117	0.082	– 30%
MetaCost	0.060	3,222	2,214	– 31%	3,222	0.060	0.050	– 16%
LRCascade, $t = 0.75$	0.047	2,206	3,465	+ 57%	2,206	0.047	0.060	+ 26%
LRCascade, $t = 0.80$	0.035	3,013	4,705	+ 56%	3,013	0.035	0.052	+ 47%
LRCascade, $t = 0.85$	0.024	4,191	6,351	+ 52%	4,191	0.024	0.040	+ 70%

Table 5: Interpolated k and MED_{ERR} when training at $\text{MED}_{\text{ERR}} \leq 0.05$. The relative gain or loss for k and MED_{ERR} is shown when compared to using a fixed cutoff for all queries. The Oracle method represents the best possible result given a perfect classifier.

mean k and a higher percentage of queries under the target MED, translating into higher effectiveness. We note that even the gold standard does not achieve 100% under MED_{RBP}. For a portion of the topics, our first stage returns less than the target k documents due to a lack of documents containing any of the query terms. Since MED_{RBP}, like RBP, is conceptually evaluated to infinite depth, this deficiency is reflected by positive scores, some of which fall above the target value. On the other hand, MED_{DCG} is evaluated to fixed depth (depth 20 in this case) and the gold standard achieves 100%.

Finally, Table 5 shows the breakdown for several fixed k and predicted k interpolations when building the training set to target $\text{MED}_{\text{ERR}} \leq 0.05$. The trends remain consistent as when using MED_{RBP} or MED_{DCG}. The most interesting aspect of this table is to note the subtle difference in potential improvements possible for the gold standard Oracle result. Potential gains are +319% and +292% respectively. This is a little better than the Oracle result for MED_{RBP} shown in Table 4, or that of $\text{MED}_{\text{DCG}} \leq 0.50$, which shows potential gains of +259% for k and +147% for MED. Potential gains are sensitive to both training cutoff and metric, which should come as no surprise.

Note that further improvements could be realized by tuning a number of different configuration options such as the number of class cutoffs, using variable cutoff thresholds t at different nodes in the cascade, changing the classifier algorithm (perhaps even using different classifiers) at different nodes in the cascade, or even developing an entirely new approach to cascaded regression / classification. Initial efforts towards variable cutoff thresholds show promising results. The problem might also be cast as a regression problem with a customized loss function. However, any of the gains achievable are independent to all of the classification / regression decisions. In fact, the precise gain can be computed based on the creation of the Oracle run before investing any time and effort into engineering a feature set and prediction scheme. We believe this in itself is an important tool for solving a wide variety of query effectiveness prediction problems in the future.

Method	NDCG@10	ERR	k
Oracle	0.356	0.434	2,386
LRCascade, $t = 0.75$	0.359	0.435	3,422
LRCascade, $t = 0.80$	0.359	0.435	4,062
LRCascade, $t = 0.85$	0.358	0.435	5,130
Fixed, $k = 10,000$	0.358	0.434	10,000

Table 6: Measured performance over 50 held out TREC 2009 Web Track adhoc queries.

Validation. As a final step, we confirmed our past experience (as illustrated by Figure 3) that low MED_{RBP} values produce minimal loss in measured effectiveness. For this purpose we employed the 50 queries of the TREC 2009 Web Track adhoc task, which were held out from the training and test sets of other experiments reported in this section. These 50 queries were pooled to depth 12 for judging, and so should be suitable for computing early-precision effectiveness measures, including NDCG@10 and ERR [19].

Table 6 shows the results. Over these queries, our cascade classifier produces no measurable loss in effectiveness when compared to a fixed k of 10,000. In fact, the classifier achieves a tiny (but not significant) gain in effectiveness in the third decimal place of some measures, reflecting a change of one or two documents across this small query set. On the other hand, there are substantial reductions in average k, reflecting expected efficiency improvements.

5. CONCLUSION

In this work, we have presented a novel query-specific approach to dynamically predict the best parameter cutoffs that maximize both efficiency and effectiveness. To achieve this, we use Maximized Effectiveness Difference (MED) [11, 31] as the basis for evaluating the quality of a candidate set relative to a more expensive gold standard reranking step. By extending this methodology, we are able to create large test corpora and train robust classifiers that require *no* relevance judgments. Our approach to binary cascaded classification is able to achieve up to a 50% improvement in average k. Our approach can be easily generalized to effectively tune a wide variety of other parameters dynamically in multi-stage retrieval systems, and can be used to reliably estimate potential gains achievable with any parameter — metric — target threshold combination. In future work, we will look at creating more appropriate rank-sensitive loss functions and explore how to use regression techniques to further improve the generalizability of our approach.

6. ACKNOWLEDGMENTS

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC). Shane Culpepper is the recipient of an Australian Research Council DECRA Research Fellowship (DE140100275).

References

[1] N. Asadi and J. Lin. Fast candidate generation for two-phase document ranking: Postings list intersection with Bloom filters. In *Proc. CIKM*, pages 2419–2422, 2012.

[2] N. Asadi and J. Lin. Document vector representations for feature extraction in multi-stage document ranking. *Inf. Retr.*, 16(6):747–768, 2013.

[3] N. Asadi and J. Lin. Effectiveness/efficiency tradeoffs for candidate generation in multi-stage retrieval architectures. In *Proc. SIGIR*, pages 997–1000, 2013.

[4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[5] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proc. ICML*, pages 89–96, 2005.

[6] C. J. C. Burges. From RankNet to LambdaRank to LambdaMART: An overview. Technical Report MSR-TR-2010-82, Microsoft Research, 2010.

[7] B. B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt. Early exit optimizations for additive machine learned ranking systems. In *Proc. WSDM*, pages 411–420, 2010.

[8] D. Carmel and E. Yom-Tov. *Estimating the Query Difficulty for Information Retrieval.* Morgan & Claypool, 2010.

[9] O. Chapelle, D. Metzler, Y. Zhang, and P. Grinspan. Expected reciprocal rank for graded relevance. In *Proc. CIKM*, pages 621–630, 2009.

[10] O. Chappelle and Y. Chang. Yahoo! Learning to Rank Challenge overview. *JMLR W&CP*, 14:1–24, 2009.

[11] C. L. A. Clarke, J. S. Culpepper, and A. Moffat. Assessing efficiency–effectiveness tradeoffs in multi-stage retrieval systems without using relevance judgments. *Inf. Retr.*, 19(4):351–377, 2016.

[12] N. Craswell, D. Fetterly, M. Najork, S. Robertson, and E. Yilmaz. Microsoft Research at TREC-2009: Web and relevance feedback tracks. In *Proc. TREC*, 2009.

[13] P. Domingos. MetaCost: A general method for making classifiers cost-sensitive. In *Proc. KDD*, pages 155–164, 1999.

[14] C. Elkan. The foundations of cost-sensitive learning. In *Proc. IJCAI*, pages 973–978, 2001.

[15] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Information Systems*, 20(4):422–446, 2002.

[16] S. Kim, Y. He, S.-W. Hwang, S. Elnikety, and S. Choi. Delayed-Dynamic-Selective (DDS) prediction for reducing extreme tail latency in web search. In *Proc. WSDM*, pages 7–16, 2015.

[17] R. Kohavi, A. Deng, B. Frasca, T. Walker, Y. Xu, and N. Pohlmann. Online controlled experiments at large scale. In *Proc. KDD*, pages 1168–1176, 2013.

[18] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[19] X. Lu, A. Moffat, and J. S. Culpepper. The effect of pooling and evaluation depth on IR metrics. *Inf. Retr.*, 19(4):416–445, 2016.

[20] C. Macdonald, R. L. T. Santos, and I. Ounis. On the usefulness of query features for learning to rank. In *Proc. CIKM*, pages 2559–2562, 2012.

[21] C. Macdonald, N. Tonellotto, and I. Ounis. Learning to predict response times for online query scheduling. In *Proc. SIGIR*, pages 621–630, 2012.

[22] C. Macdonald, R. L. T. Santos, and I. Ounis. The whens and hows of learning to rank for web search. *Inf. Retr.*, 16(5):584–628, 2013.

[23] C. Macdonald, R. L. T. Santos, I. Ounis, and B. He. About learning models with multiple query-dependent features. *ACM Trans. Information Systems*, 31(3):11, 2013.

[24] J. Mackenzie, F. M. Choudhury, and J. S. Culpepper. Efficient location-aware web search. In *ADCS*, pages 4.1–4.8, 2015.

[25] A. Moffat and J. Zobel. Rank-biased precision for measurement of retrieval effectiveness. *ACM Trans. Information Systems*, 27(1):2.1–2.27, 2008.

[26] J. Pedersen. Query understanding at Bing. *Invited talk, SIGIR*, 2010.

[27] M. Petri, J. S. Culpepper, and A. Moffat. Exploring the magic of WAND. In *Proc. ADCS*, pages 58–65, 2013.

[28] M. Petri, A. Moffat, and J. S. Culpepper. Score-safe term dependency processing with hybrid indexes. In *Proc. SIGIR*, pages 899–902, 2014.

[29] T. Qin, T.-Y. Liu, J. Xu, and H. Li. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Inf. Retr.*, 13 (4):347–374, 2009.

[30] S. Robertson. On GMAP: And other transformations. In *Proc. CIKM*, pages 78–83, 2006.

[31] L. Tan and C. L. A. Clarke. A family of rank similarity measures based on maximized effectiveness difference. *IEEE Trans. Knowledge and Data Engineering*, 27(11):2865–2877, 2015.

[32] N. Tonellotto, C. Macdonald, and I. Ounis. Efficient and effective retrieval using selective pruning. In *Proc. WSDM*, pages 63–72, 2013.

[33] W. Webber, A. Moffat, and J. Zobel. A similarity measure for indefinite rankings. *ACM Trans. Information Systems*, 28(4):20.1–20.38, Nov. 2010.

[34] M. Zhang, D. Kuang, G. Hua, Y. Liu, and S. Ma. Is learning to rank effective for web search? In *Proc. SIGIR Workshop LR4IR*, pages 641–647, 2009.

[35] H. Zhao, A. P. Sinha, and G. Bansal. An extended tuning method for cost-sensitive regression and forecasting. *Decision Support Systems*, 51(3):372–383, 2011.