Accelerated Query Processing Via Similarity Score Prediction

Matthias Petri
The University of Melbourne
Melbourne, Australia

Alistair Moffat
The University of Melbourne
Melbourne, Australia

Joel Mackenzie
RMIT University
Melbourne, Australia

J. Shane Culpepper
RMIT University
Melbourne, Australia

Daniel Beck
The University of Melbourne
Melbourne, Australia

ABSTRACT

Processing top-k bag-of-words queries is critical to many information retrieval applications, including web-scale search. In this work, we consider algorithmic properties associated with dynamic pruning mechanisms. Such algorithms maintain a score threshold (the k th highest similarity score identified so far) so that low-scoring documents can be bypassed, allowing fast top-k retrieval with no loss in effectiveness. In standard pruning algorithms the score threshold is initialized to the lowest possible value. To accelerate processing, we make use of term- and query-dependent features to predict the final value of that threshold, and then employ the predicted value right from the commencement of processing. Because of the asymmetry associated with prediction errors (if the estimated threshold is too high the query will need to be re-executed in order to assure the correct answer), the prediction process must be risk-sensitive. We explore techniques for balancing those factors, and provide detailed experimental results that show the practical usefulness of the new approach.

KEYWORDS

Inverted index; query efficiency; dynamic pruning

ACM Reference Format:
Matthias Petri, Alistair Moffat, Joel Mackenzie, J. Shane Culpepper, and Daniel Beck. 2019. Accelerated Query Processing Via Similarity Score Prediction. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19), July 21–25, 2019, Paris, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3331184.3331207

1 INTRODUCTION

Millions of dollars per day are spent on the electricity and hardware costs associated with web query processing, and hence even relatively small improvements in efficiency translate into substantial monetary (and environmental) savings. Most web search queries are, at least in the first phase of evaluation, scored assuming that term occurrences are independent, and that document scores can be computed as the sum of the term contributions that are determined at indexing time. An answer to a query is then a set of k top-scoring documents, either for immediate display to the user who formulated the query, or as input to a secondary re-ranking phase.

A number of efficient query processing strategies have been devised for this situation, including the MaxScore and WAND approaches. Both seek to avoid unnecessary computation by only scoring documents that have some hope of making it into an evolving top-k set and bypassing the documents that do not. Central to both of these strategies is a non-decreasing value which we refer to here as θ, the k th largest document score encountered to date during the processing. The idea is that as θ increases, a decreasing fraction of the remaining documents need to be considered for inclusion in the top-k set, and an increasing fraction can be bypassed. Once all documents have been considered and all postings processed, θ will have converged to the value $\Theta_k(\mathcal{D},Q)$, the k th largest computed similarity score for query Q on collection $\mathcal{D}$. Figure 1 illustrates these ideas.

In this context we consider the following research questions:

RQ1 Is it possible to estimate $\Theta_k(\mathcal{D},Q)$ in advance, before any processing of postings is undertaken?

RQ2 If $\Theta_k(\mathcal{D},Q)$ can be estimated in advance, to what extent does that knowledge allow accelerated query processing?

RQ3 Does pre-knowledge of $\Theta_k(\mathcal{D},Q)$ allow other modifications to the way that queries are processed using methods such as MaxScore, WAND, and BMW?

2 BACKGROUND

Term-Weighted Similarity Scoring. We suppose that to compute the similarity between an q-term query $Q=\{t_1,t_2,\ldots,t_q\}$ and a document d a formulation $\mathcal{S}(d,Q) = f_1(d) + \sum_{i=1}^{q} f_2(t_i,Q)\cdot C(t_i,d)$ is used, where $f_1(\cdot)$ is a function of d alone and can be pre-computed; where $f_2(\cdot,\cdot)$ is a function of a term and a query, and not of d; and where $C(t,d)$ represents the query-independent relative importance of term t in document d, and can also be pre-computed. That is, we presume that $\mathcal{S}(d,Q)$ can be computed as a transposed linear sum of term-document contributions. Many retrieval formulations have this structure, including the well-known BM25 approach.

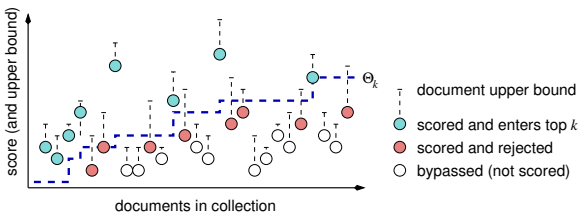
We also suppose that a "top-k" filter will be applied, with the answer set nominally formed by scoring every document in the collection, then decreasing-score ordering that set, and then taking the first k documents. If we denote by $\Theta_k(\mathcal{D},Q)$ the k th largest similarity score that results when Q is evaluated against a collection $\mathcal{D}$ of documents:

$$\Theta_k(\mathcal{D},Q) = \max\{\theta \mid |\{d\in\mathcal{D} \mid \mathcal{S}(d,Q)\geq\theta\}|\geq k\},$$

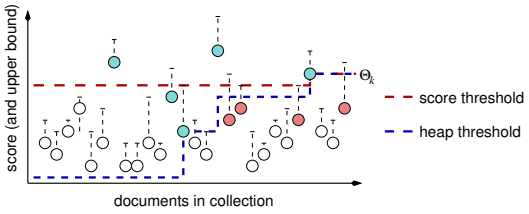
then a top-k query is answered via a set $\mathcal{A}_k(\mathcal{D},Q)$ containing k documents with scores not smaller than $\Theta_k(\mathcal{D},Q)$. If the answer set will be presented immediately to the user, then $k=10$ or $k=100$ might

SIGIR '19, July 21–25, 2019, Paris, France

ACM ISBN 978-1-4503-6172-9/19/07…$15.00
https://doi.org/10.1145/3331184.3331207



(a) Normal pruning, with a single threshold (blue line).



(b) Use of an initial score threshold estimate (dark red line).

Figure 1: Dynamic pruning, with the top $k = 3$ documents to be identified: (a) with the heap threshold (blue dashed line) initialized to a minimum value, and the score threshold equal to the heap threshold throughout; and (b) using an initial non-trivial score threshold (red dashed line) that approximates the final heap threshold. Provided that the initial score threshold estimate is lower than the final k th score, an equivalent answer set is returned, with fewer documents scored and more documents bypassed.

be appropriate. If the set will be supplied as input to a second-stage re-ranker, then $k = 1000$ or even $k = 10{,}000$ might be used. For any given collection $\mathcal{D}$ and query Q, the threshold $\Theta_k(\mathcal{D},Q)$ is non-increasing with k. Note also that where there is no risk of ambiguity we will use Θ_k as shorthand for $\Theta_k(\mathcal{D},Q)$, taking as read the existence of a collection and query. Finally, we use ∇ to represent the lowest score that can be assigned to any document (often zero, or minus infinity).

Query Processing. Operationally, it is usual for each contribution $C(t,d)$ to be either stored as part of the corresponding posting in a document-level inverted file, or to be computable directly from the stored posting information. A range of processing strategies is then possible. In *document-at-a-time* processing, the query terms' postings lists are simultaneously processed, and a final score for each document is computed before any other document is considered [48]. One aspect of document-at-at-time processing that has made it attractive for practical use is a range of *dynamic pruning heuristics*. The WAND mechanism developed by Broder et al. [2] is a good example of these techniques. In the WAND approach, a *maximum contribution*, U_t is associated with each term t, computed at index construction time as $U_t = \max\{C(t,d) \mid d \in \mathcal{D}\}$. A heap of the "seen so far" top-k scores and their corresponding documents is maintained, together with a variable θ that is the smallest amongst those k scores. That heap contains zero items at the commencement of the process, with θ initialized to ∇.

At each WAND scoring cycle, the query terms t_1 to t_q are assumed to be ordered according to the values d_1 to d_q, the ordinal identifiers of the next unprocessed document in each corresponding postings list. A *pivot* term t_p is then determined as the smallest p such that the sum of the U_i bounds for the terms matching d_1 to d_p exceeds θ. At this point it is known that no further documents prior to d_p can attain a score greater than θ, and so they can all be bypassed. But if t_1 through t_p all indicate the same document d_p, then it needs to be fully evaluated, since its score $\mathcal{S}(d_p,Q)$ could conceivably exceed the current threshold θ. As a result, either d_p will indeed attain a score greater than θ and enter the top-k structure (perhaps causing another document to be ejected and θ increased), or it will have been found to be a *false hope*, with $\mathcal{S}(d_p,Q) < \theta$, meaning that d_p is not in the final answer set $\mathcal{A}_k(\mathcal{D},Q)$. The postings list cursors for terms t_1 to t_p are then advanced to the next document number greater than d_p, the terms reordered according to the new d_1 to d_q values, and the next scoring cycle commenced. Petri et al. [45] provide detailed pseudo-code for this process.

In a refinement of the WAND process, Ding and Suel [16] note that if the upper-bound values U_t are stored once per block of postings rather than on a whole-of-list basis, more precise decision-making is possible and more documents can be bypassed, albeit at an increased cost in terms of the logic required to determine if any given document should be evaluated. Dimopoulos et al. [15] and Petri et al. [45] have also studied this block-max WAND approach. Most recently, Mallia et al. [37, 38] proposed an enhanced version of block-max WAND that leverages variable-sized blocks, allowing for even tighter localized upper-bounds. We refer to this latest approach as BMW, and assume the use of variable-sized blocks. The earlier MaxScore process described by Turtle and Flood [51] similarly makes use of an increasing threshold θ, evolving towards the same final value $\Theta_k(\mathcal{D},Q)$, but employs different processing logic in terms of determining which documents may need to be scored, and which can be bypassed.

All three approaches make use of document-at-a-time scoring, a "k th best so far" threshold θ, and (for any given retrieval similarity formulation) converge to the same final score value $\Theta_k(\mathcal{D},Q)$; and all three of them may thus be amenable to the effects explored in the three research questions listed in Section 1. Note also that all three pruning options are *safe*, in that they generate exactly the same answer set $\mathcal{A}(\mathcal{D},Q)$ as does the underlying exhaustive computation.

Fontoura et al. [17] propose a hybrid query processing approach, where Q is split into two parts: terms with short postings, and terms with long postings. The sub-query with the short postings is processed first. Then, the partial score of the k th element in the heap is used to set the value of θ. In a somewhat similar approach, de Carvalho et al. [13], store the values of the k th highest scores in each postings list for common values of k such as 10, 100, and 1,000. At run-time, θ is initialized to the maximum of the k th highest scores for the candidate postings lists. This idea was also explored by Kane and Tompa [25]. Note that this approach requires a strictly additive similarity (that is, with $f_1(\cdot) = 0$ and $f_2(\cdot,\cdot)$ constant), which then provides the necessary assurance that Θ_k cannot be smaller than the largest (across the query's terms) of the k th largest contribution $C(t,d)$. For any particular query Q, this quantity, denoted Q_k, can be calculated from per-term values stored in the index, and then used as a risk-free initialization for θ. One of our goals in this paper is to generate better estimates of Θ_k.

Figure 1(b) illustrates the interaction between actual document scores (the dots), document upper bound scores $U\_bound(d)$ (the whiskers above the dots), the score computation threshold θ and a

"safe" initialization of it, the heap entry threshold *h_threshold*, and the final top-k cutoff score Θ_k.

In their description of BMW, Ding and Suel [16] note that priming of the threshold might reduced execution time and posed the prediction question that is considered here, and other authors have also considered the drag imposed by the startup cost of the threshold, as it advances upward from its low initial value. For example, Kim et al. [26] consider WAND as it applies to selective search; and Clarke et al. [7] deliberately adjust the threshold so as to allow faster (but non-safe) query execution.

Machine Learning for Efficiency. Most efficiency-based machine learning research work has focused on Learning-to-Rank (LTR) retrieval stages that occur after candidate generation. Cambazoglu et al. [4] were among the first to explore efficiency concerns in an LTR framework. In production search engines, LambdaMART [3] and Gradient Boosted Regression Trees (GBRT) [18] are still widely considered to be state-of-the-art, and as a result, several optimizations to improve performance through tree representation and traversal [24, 30, 31], model pruning [12, 32], and budget-aware learning algorithms [44, 55] have been proposed. Other recent studies have focused on balancing feature costs (efficiency) and system effectiveness across multiple re-ranking stages using cascading loss functions [5, 53].

Another approach to improving efficiency in multi-stage retrieval is to minimize the number of documents that must be re-ranked for each query [9, 35, 50]. These approaches are particularly relevant to our current work as the cost of the ML prediction is part of the candidate generation stage. As dynamic pruning algorithms such as BMW are often used for candidate generation and are heavily optimized for efficiency, the use of ML in early-stage retrieval is still relatively rare outside of query rewriting [20, 33, 56] as the cost of the prediction can easily outweigh the benefits.

Recent LTR research has primarily focused on Neural IR models [41]. Despite consistent progress in improving search effectiveness using these models, few published studies have carefully explored the efficiency costs of deep learning (DL) models in the IR domain. Zamani et al. [57] explore the possibility of learning a latent sparse representation which can be used for efficient end-to-end retrieval without requiring the use of an initial candidate generation stage; but their choice of baseline implementation leaves it unclear whether they are competitive in terms of efficiency.

Kraska et al. [28] show that DL models can in fact be optimized for both efficient and effective prediction, with learned B-Tree operations for 200 million records requiring less than one microsecond on average. So efficient neural models for IR appear to be within reach, albeit mostly unexplored at this time.

Other recent work has turned to Bayesian reasoning to provide more principled arguments for compression and efficiency in deep learning models [29]. While we do not approach our problem from a Bayesian deep learning [19] perspective, Section 4 does explore both neural and Bayesian models for prediction effectiveness.

3 EXPLORING POTENTIAL GAINS

Oracle Evaluation. To demonstrate proof-of-concept, we consider first an *oracle* estimator that "magically" knows the value $\Theta_k(\mathcal{D},Q)$ prior to commencement of evaluation, and is thus able to evaluate each query using the best possible knowledge. Table 1 lists query execution times for MaxScore, WAND and BMW evaluation, measured in milliseconds across a set of 5,000 queries and using the Gov2 collection. Exact knowledge of the final score threshold allows substantially reduced query execution times, across the distribution of query durations from the median (50% quantile) through to the tail (99% quantile). Based on the final rows of mean times, and compared to the "θ initialized to ∇" versions, savings of up to approximately 30% when $k=10$ and up to approximately 50% when $k=1000$ can be realized by a perfect predictor of Θ_k, with the "up to" components of those claims dependent on the execution-time cost of generating the corresponding predicted thresholds. The potential savings relative to the initial θ represented by Q_k (the two middle columns) are smaller, but nevertheless, represent a possible opportunity for improvement. Note that BMW demonstrates its usual advantage over MaxScore and WAND in these experiments. (A detailed description of the hardware, software and data resources used in this evaluation environment used for these initial results is provided in Section 5.)

Method	Time	$k=10$			$k=1{,}000$		
		∇	Q_k	Θ_k	∇	Q_k	Θ_k
MaxScore	P_{50}	3.8	2.9	2.0	13.3	8.0	7.3
	P_{95}	35.3	34.4	24.9	66.6	60.5	49.7
	P_{99}	70.0	68.1	47.5	118.9	107.9	88.3
	Mean	9.0	8.1	6.0	21.3	16.9	14.4
WAND	P_{50}	2.8	2.2	1.7	13.1	7.2	6.1
	P_{95}	26.9	24.8	20.8	59.5	48.8	41.2
	P_{99}	64.5	57.2	51.7	102.6	90.6	83.9
	Mean	7.1	6.1	5.1	19.7	14.0	11.8
BMW	P_{50}	1.0	0.9	0.7	5.9	3.4	2.7
	P_{95}	7.4	7.0	5.2	27.8	23.2	16.2
	P_{99}	15.7	14.8	10.6	48.2	44.1	30.8
	Mean	2.3	2.1	1.6	9.7	6.7	5.0

Table 1: Per-query execution times (milliseconds) for MaxScore processing (top), WAND processing (middle), and BMW processing (bottom) on the Gov2 collection. Two different values of k are explored, and three different initializations for θ: standard initialization, with θ initialized to ∇; with θ initialized to Q_k, the maximum value of the k th contribution across the q terms in Q [13, 25]; and an oracle in which θ is initialized to $\Theta_k(\mathcal{D},Q)$ for each query Q. The four rows in each group record the median, the 95 th and 99 th percentiles, and the mean.

We also explored the distribution of Θ_k scores across the same set of queries, and verified that they show a broadly Gaussian distribution, with a mean value of approximately 19 for $k=10$ and of approximately 13 for $k=1{,}000$. Note that in the BM25 computation all scores are positive, and hence $\nabla=0$ may be used.

Sensitivity. Another important factor that has the potential to affect the viability of any proposed prediction approach is the sensitivity of query execution time to imprecise thresholds. Suppose, for example, that prediction $\hat{\Theta}_k$ arises from some modeling mechanism, where $\hat{\Theta}_k < \Theta_k$. In this situation executing the WAND mechanism

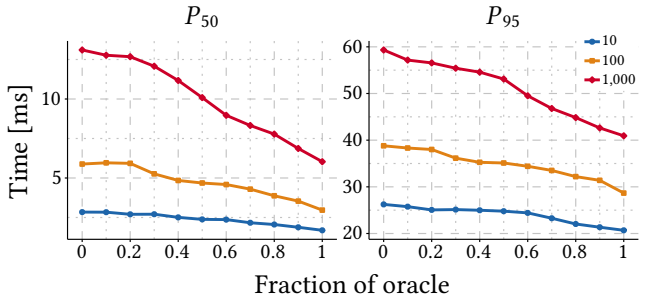


Figure 2: Sensitivity of the median (P_{50}) and 95 th percentile tail latency (P_{95}) for WAND query processing, plotted as a function of the fraction of Θ_k used as the initial estimate, for three values of k. Similar trends were observed for MaxScore and for BMW.

with θ_0 initialized to $\hat{\Theta}_k$ generates the correct outcome, since a superset of the final answers $\mathcal{A}(\mathcal{D},\mathcal{Q})$ are examined. But the execution time may increase, since more documents will have been processed.

Figure 2 explores this issue, using the 5,000 queries and Gov2 collection. Both plots show the same behavior – moderate savings (only) in execution time when the initial threshold θ is around 75% or less of the oracle value Θ_k, and with $0.9\Theta_k$ or more required if anything approaching the full potential savings is to be captured. Similar patterns arise for other retrieval algorithms.

Failure Detection. If the estimate $\hat{\Theta}_k$ used to initialize θ_0 is *greater* than Θ_k, then the pruning computation is no longer safe, and might produce a different answer set than the underlying exhaustive computation. Fortunately, it is always possible to determine if this situation has arisen. One scenario that might occur is if the computed answer set contains fewer than k documents; in this case it is clear that θ was initialized too high, and that the query needs to be re-executed with a more generous initial value. A second more subtle situation might also occur: the computed answer set might contain k documents, because k or more documents were identified for full scoring and k of them remain in the heap at the completion of processing, but the smallest of those k scores is less than the estimate $\hat{\Theta}_k$. In this case the k th largest value in the heap can be used as an initial estimate θ_0 for a re-execution, since we now know that there are at least k documents in $\mathcal{D}$ that have scores greater than or equal to that value. Note that in this second situation the answer set might – by luck – be correct, and the re-computation unnecessary. But there is no simple way of testing whether that is the case, hence the need to re-execute the query. Section 6 explores an approach that under some situations allows the re-execution to be avoided, and then uses on that observation to describe an alternative mechanism for patching.

Query Processing Revisited. Based on these various considerations, Algorithm 1 shows the proposed process, with the iterator implied in the "for" loop at step 5 hiding the details of the corresponding pruning mechanism. The per-document score upper bound $U\_bound(d)$ is derived in the usual manner from the U_t values stored in the postings lists, and the similarity formulation in use.

In terms of risks, estimated thresholds $\hat{\Theta}_k$ that are lower than Θ_k cause additional documents to be scored, and hence reduce query throughput (Figure 2); whereas estimated thresholds $\hat{\Theta}_k$ that are higher than Θ_k result in the whole query being re-executed as a result of the test at step 15. That is, both kinds of estimation error incur a penalty, but over-estimation is a more costly mistake. That tension is explored further in the next section.

Algorithm 1 Computation of $\mathcal{A}(\mathcal{D},\mathcal{Q})$ based on a estimated score threshold. The test at step 15 ensures that the answer returned at step 18 is safe, even if the score threshold estimated at step 1 is incorrect.

```
1:  develop an estimate Θ̂_k of the final k th largest score
2:  set θ ← Θ̂_k                                  ▷ scoring threshold
3:  set h_threshold ← ∇                           ▷ heap entry threshold
4:  initialize an empty min-heap
5:  for each d ∈ D with U_bound(d) ≥ θ do
6:      calculate score ← S(d,Q)
7:      if score > h_threshold then
8:          add (d,score) to the heap
9:          if the heap has more than k items then
10:             eject the least-weight document in the heap
11:             adjust h_threshold to reflect the modified heap
12:             if h_threshold > θ then
13:                 set θ ← h_threshold
14: // now for the post-processing check
15: if h_threshold < Θ̂_k then                     ▷ cannot guarantee answer
16:     set θ ← h_threshold
17:     execute steps 3 to 13 again
18: return the k documents in the heap
```

4 THRESHOLD PREDICTION MODELS

The primary requirements for a pruning threshold prediction model – estimation speed and model accuracy – are in tension, as more complex models might produce better predictions, but may also require more computation. Efficient query processing schemes execute top-k queries in around 5–20 milliseconds (Table 1), and so any model must generate predictions in a fraction of that time if the potential gains are to be realized. We note that transferring data to the GPU is expensive. Therefore we investigate relatively simple models, aiming to produce predictions that requires less than one millisecond of CPU time.

Algorithmic Predictor. As was noted earlier, the maximum of the k th largest contributions ($\mathcal{Q}_k$) from the query terms can be used as a better-than-∇ lower bound on Θ_k [13, 25], and we employ this deterministic predictor as a second baseline in our experiments.

Probabilistic Predictor. Our first approach is a probabilistic model that generates *distributions* over the predicted $\hat{\Theta}_k$. The main advantage of this approach is that it decouples modeling from the decision-making procedure, important in our setting because (as was noted in Section 3) overestimating the thresholds $\hat{\Theta}_k$ incurs a higher penalty than underestimating them. Figure 3 shows an example of this approach in practice: the predictor outputs a Gaussian distribution over $\hat{\Theta}_k$. While the median of this distribution provides the most probable value for the threshold, it is actually higher than the true value, which can incur in a heavy penalty. Instead, a more conservative prediction can be favored by taking a lower quantile as the predicted value.

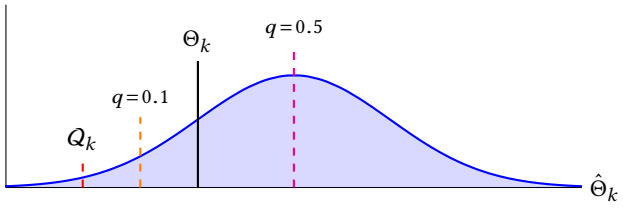


Figure 3: Pictorial representation of the probabilistic approach. The predictor outputs a Gaussian distribution over $\hat{\Theta}_k$, where the median over-predicts. A lower quantile provides a more conservative estimate while still giving improvements over the algorithmic predictor Q_k.

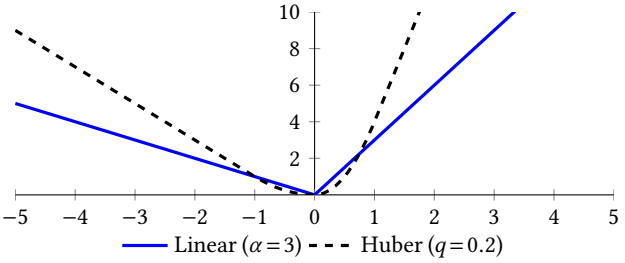


Figure 4: Two asymmetric loss functions: asymmetric linear loss, and quantile Huber loss [10], providing different asymptotic loss penalties for positive and negative losses of the same magnitude.

The concept of penalizing overestimates and underestimates differently is formalized using an *asymmetric loss function*. Let $u = \Theta_k - \hat{\Theta}_k$ be the difference between the true threshold and the predicted value. The asymmetric linear loss for the probabilistic predictor can be defined as:

$$L(\hat{\Theta}_k, \Theta_k) = \begin{cases} u & \text{if } u \geq 0 \\ \alpha \cdot |u| & \text{if } u \leq 0, \end{cases}$$

where $\alpha > 0$ is a parameter that captures the penalty of an overestimate as compared to an underestimate. When $\alpha > 1$ overestimates are penalized disproportionately; conversely, when $0 < \alpha < 1$, underestimates are more heavily penalized. This loss is a generalization of the absolute error, or L1 loss, which is recovered when $\alpha = 1$. Figure 4 shows an instance of this loss (blue solid line).

The relationship between α and distribution quantiles can be formalized using Minimum Bayes Risk (MBR). Given a predictive distribution, MBR aims at obtaining the optimal value given a loss function,

$$\hat{\Theta}_k = \underset{\hat{\Theta}_k \in \mathbb{R}}{\operatorname{argmin}} \mathbb{E}\left[L(\bar{\Theta}_k, \Theta_k)\right] = \int_{\bar{\Theta}_k} L(\bar{\Theta}_k, \Theta_k) \cdot p(\bar{\Theta}_k \mid Q)\, d\bar{\Theta}_k,$$

where $\bar{\Theta}_k$ is a random variable modeling the desired threshold, and conditioned on the input query Q. When using an asymmetric linear loss, it can be proven that the optimal result is the $1/(\alpha+1)$ quantile of the distribution [6]. This easily translates to applying a fixed cost. For example, if overestimates are three times more costly than underestimates, the 0.25 quantile should be used.

Uncoupling the loss from the model has a series of advantages. For instance, at test time, one can obtain multiple values for $\hat{\Theta}_k$, either sequentially or in parallel, without having to train multiple models or run the predictor multiple times. Another advantage is that query-specific losses can be applied.

The main drawback of this approach is that it restricts the class of models that can be applied if prediction cost is critical. In this work, a Bayesian Linear Regression model (BLR) is used since it provides closed form solutions for the predictive distributions [34, 42], and avoids traditional Bayesian inference techniques such as sampling, which can be prohibitively slow. We use the Scikit-learn 0.20 toolkit implementation with default hyperparameters and train the model by optimizing the marginal likelihood [34].

Loss-Based Predictor. An alternative is to incorporate the desired asymmetric loss into the model definition. Such models do not generate distributions but instead try to directly predict a value given the loss. The main advantage of this approach is that it allows the exploration of more complex models beyond simple linear regression, while still maintaining acceptable prediction efficiency. In particular, if the loss is differentiable, neural-based models can be trained using backpropagation, which we explore in this work.

The asymmetric linear loss is differentiable everywhere except when $u = 0$. According to Dabney et al. [10], this can make training in neural models unstable, and they instead propose an alternative approach based on the Huber loss [22],

$$L(\hat{\Theta}_k, \Theta_k, q) = \begin{cases} q \cdot L_{Huber}(\hat{\Theta}_k, \Theta_k) & \text{if } u \geq 0 \\ (1-q) \cdot L_{Huber}(\hat{\Theta}_k, \Theta_k) & \text{if } u \leq 0, \end{cases}$$

$$L_{Huber}(\hat{\Theta}_k, \Theta_k) = \begin{cases} \frac{1}{2}u^2 & \text{if } |u| \leq \kappa \\ \kappa(|u| - \kappa/2) & \text{if } |u| \geq \kappa, \end{cases}$$

where q is the desired quantile to be optimized and κ is a parameter that splits the loss into a linear and a squared region. The loss is linear when the absolute error is larger than κ but falls back to a squared error loss when it is below that value. In this work, we follow Dabney et al. [10] and use this loss with $\kappa = 1$ to train our models. Figure 4 shows an example of an asymmetric Huber loss (black dashed line).

The specific neural architectures we employ are multi-layer perceptrons with two (MLP-L2) or four (MLP-L4) hidden layers which can be evaluated efficiently on a single CPU. Training is done via standard techniques using the Adam [27] optimizer, with an initial learning rate of 0.001, and early stopping on a development set. The architecture is regularized by applying batch normalization [23] and a dropout rate of 0.25 between each layer [47]. Hidden layer sizes are the same as the input layer.

When compared to the probabilistic approach, the main drawback of loss-based predictors is that the loss needs to be known at training time. This vastly reduces the flexibility of any decision making procedure: different losses require multiple models and any changes to the loss after deployment require model retraining. In this work, we compare the probabilistic and loss-based predictors empirically but these key differences should be considered when applying this approach in real world scenarios.

Data, Features and Costs. To train the models we use 8,073,821 queries drawn a 2006 Microsoft Query log released as part of a WSDM workshop [8]. We use an additional 1000 queries as a development set and 1000 queries drawn from the same collection (ensuring no overlap with train/dev) to evaluate the performance of our predictors.

Type	#	Description
Score	14	Histogram of block max scores such that 2^i th highest block max score is stored, for $i \in [0,14]$
Score	3	k th highest for $k \in \{10,100,100\}$
Weight	1	IDF component of BM25 for a given term
Frequency	4	Mean, median, min and maximum f_t value
Frequency	9	Histogram of f_t storing the number of f_t values $\geq 2^i$ for $i \in [0,8]$
Doc len	4	Mean, median, min, and max document length

Table 2: The 35 features that must be added to the index so that they can be used for threshold score prediction.

For each query we extract the relevant features from the inverted index, and also determine $\Theta_k(\mathcal{D},\mathcal{Q})$ for $k \in \{10,100,1000\}$. For the intrinsic evaluation of the different predictors we utilize the standard Gov2 collection described in Section 5.

We extract and store several features from the inverted index, listed in Table 2; and hence add a constant per-term overhead to the cost of storing it. These features are easily extracted from the inverted index and add 7.5 bits per posting to the index of Gov2, and 6.6 bits per posting to the ClueWeb09B index without any compression of the features. Note that in practice, many these features are likely to be stored for use in the re-ranking stage of query processing.

The cost of storing the prediction models is small. The BLR model requires 1.0 MiB and the MLP models require 1.7 MiB (MLP-L4) and 0.9 MiB (MLP-L2). All of the models are static arrays of floating-point numbers and are retained in memory during query processing. Training is equally fast, and requires less than 30 minutes per model.

Prediction Speed. The simplicity of the models being used means that prediction time is negligible. The mean query prediction time for is 0.6 ms for MLP-L2 and 0.9 ms for MLP-L4. The Bayesian linear regression model takes 0.6 ms to obtain the predictive distribution for a single query and 0.1 ms to get a quantile value. This distinction is important because (as explained above), new quantiles can be obtained without incurring an additional prediction overhead.

Python 3.7, Pytorch 1.0, Scipy 1.1 and Scikit-learn 0.20 were used to implement the predictors. Thus, a substantial overhead exists in the running times reported above as compared to an optimized C/C++ version which would be deployed in a production environment. That is, prediction costs are negligible for both of the models used. Even so, prediction costs are included in the experiments in Section 5.

Intrinsic Evaluation. We assess predictive performance using three metrics. The first one is Pearson's ρ correlation, a common metric to evaluate regression models, and serves as a sanity check for our approach. The other metrics are *Mean Underprediction Fraction* (MUF), expressed as the mean fraction $\hat{\Theta}_k/\Theta_k$ when $\hat{\Theta}_k < \Theta_k$, and *Overprediction Rate* (O%), which illustrates the percentage of predictions when $\hat{\Theta}_k > \Theta_k$. These two are directly related to our task: an ideal model would have a MUF of 1.0, while keeping O% close to 0.

Table 3 shows the results for $k \in \{10,1000\}$ on the Gov2 dataset for several quantiles. First, we note that all models obtain a strong ρ correlation (> 0.7) under symmetric conditions ($q = 0.5$), showing that our models are able to learn the relation between the features and

Quant.	Q_k		MLP-L2			MLP-L4			BLR		
	ρ	MUF	ρ	MUF	O%	ρ	MUF	O%	ρ	MUF	O%
$k = 10$											
0.50	0.47	0.70	0.81	0.88	58	0.81	0.89	62	0.82	0.90	57
0.30	-	-	-	0.87	44	-	0.87	47	-	0.87	28
0.10	-	-	-	0.85	25	-	0.85	23	-	0.78	7
0.05	-	-	-	0.83	16	-	0.84	18	-	0.72	3
0.01	-	-	-	0.78	3	-	0.78	3	-	0.61	1
$k = 1{,}000$											
0.50	0.53	0.81	0.85	0.91	57	0.86	0.91	65	0.88	0.91	57
0.30	-	-	-	0.90	48	-	0.90	48	-	0.89	25
0.10	-	-	-	0.88	28	-	0.89	29	-	0.80	5
0.05	-	-	-	0.87	18	-	0.86	17	-	0.76	2
0.01	-	-	-	0.82	4	-	0.83	10	-	0.66	0

Table 3: Mean under-prediction fraction (MUF) expressed as the mean fraction $\hat{\Theta}_k/\Theta_k$ when $\hat{\Theta}_k < \Theta_k$, Pearson's ρ of the predicted threshold $\hat{\Theta}_k$ and the true value Θ_k, and percentage rate of over-predictions (O%) where $\hat{\Theta}_k > \Theta_k$; for four modeling approaches and $k \in \{10,1000\}$. The Q_k estimator never over-predicts.

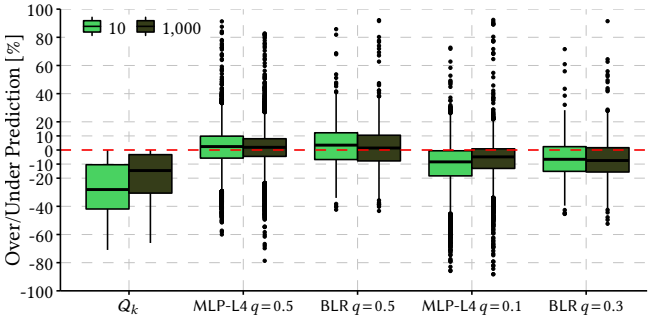


Figure 5: Relative prediction error using MLP-L4 (two quantiles), BLR (two quantiles), and Q_k, for $k \in \{10,1000\}$.

the thresholds. These values are also higher than the ones obtained by Q_k, showing that is room for improvement over the baseline.

In terms of MUF and O%, BLR tends to be more conservative than the MLP models, and when $q = 0.01$ has no overpredictions observed. However, in that scenario MUF is lower than Q_k, which is undesirable since Q_k also never overpredicts. Clearly, in order to obtain performance gains, a higher quantile must be used. The MLP models tend to be more optimistic, so lower quantiles are viable. The remaining question is whether favorable tradeoffs between MUF and O% translate to performance improvements, addressed in the next section.

Figure 5 shows relative prediction errors for MLP-L4, and the effect the asymmetric loss function has on prediction accuracy. Estimations using $q = 0.5$ tend to have a high percentage of over predictions (positive relative errors), whereas the $q = 0.1$ and $q = 0.3$ models skew towards a smaller fraction of positive relative errors, but decrease the mean under-prediction fraction (MUF). Even so, a substantial fraction of the prediction errors are less than the 10% target suggested by Figure 2. The next section provides end-to-end experiments that measure the tradeoffs involved. Compared to the naive deterministic predictor Q_k, the two models make substantially better predictions.

Feature Analysis. The weighted summation in the Linear Bayes model means that it is possible to explore the relative importance of the features, creating an equal footing by standardizing each column of scores, and with missing features set to a z-score of zero, equivalent to the column mean. However, interdependencies between features meant that the weights were not always straightforward to interpret. The most important features were the k th largest contribution $C(t,d)$ of the two rarest terms in the query. The predictor also correctly identifies that the length of the rarest term is inversely correlated with Θ_k. This is consistent with the intuition that a query consisting of only frequent terms is very likely to have a small Θ_k. Further ablation studies are required to gain a deeper understanding into the performance of the different predictors.

5 EXPERIMENTS

Data Resources. We use two document collections: Gov2, 25 million documents from the .gov domain; and ClueWeb09B, 50 million web documents crawled in 2009. Queries were taken from the 2006 Microsoft log (see Craswell et al. [8]), and Krovetz-stemmed, stopped, and then shuffled into a random order. A set of 5,000 distinct queries were sampled for early exploration; a second set of 1,000 for initial training, ensuring no overlap with the first set; a third set of 10,000 were reserved for final measurement, again making sure that there was no overlap; and the de-duplicated balance (around five million distinct queries) was used for model training. Note that Gov2 was used for the preliminary analyses in the previous sections. We now use ClueWeb09B to validate our preliminary analysis, as it is more representative of a real-world web search scenario.

Software Resources. The collections were first indexed via Indri, then the indexes extracted and reordered using recursive graph bisection [14, 36], and used as input to the PISA experimentation platform[1] [38, 39, 43]. Where BMW is used, we apply variable-sized blocks with a mean block size of 40 elements.

Hardware Platform. Except where otherwise noted, the results here use programs implemented in C++17 and compiled with GCC 7.2.0 using the highest optimization settings. Experiments are performed on a machine with two Intel Xeon Gold 6144 CPUs (3.50GHz), 512 GiB of RAM, running Linux 4.13.0. The CPU is based on the Skylake micro-architecture, which supports the AVX-512 instruction set, though we did not optimize for such instructions. Each CPU has L1, L2, and L3 cache sizes of 32 kiB, 1024 kiB, and 24.75 MiB, respectively. All timing experiments were carried out on an otherwise idle system, using a single thread.

Model Distribution Trade-Offs. Section 4 describes several possible trade-offs between the modeling approaches in terms of the measures used: Pearson's ρ, MUF, and O%. In particular there is a complex trade-off between the high cost of sometimes over-predicting and the smaller, but more pervasive, cost of under-predicting. Table 4 provides execution times for a range of combinations of predictive approach and pruning technique, focusing on retrieving the top $k = 1{,}000$ results on the ClueWeb09B test collection, using the set of 10,000 held-out test queries. All three algorithms show performance benefits when compared with Q_k, and with the BLR predictor consistently outperforming MLP, presumably because of the tighter prediction range from BLR (Figure 5). For example, MaxScore is generally the least efficient pruning method, and hence also benefits the most with accurate threshold prediction. The algorithm favors the most aggressive quantile as over-predictions incur a significant penalty. Overall, MaxScore improves by approximately 15%. It is also noteworthy that BMW also benefits for the new approach. For BLR with $q = 0.1$, there is a 4% improvement overall, including the cost of the prediction, which is approximately 0.5 milliseconds per query (as noted, unoptimized). As the BMW implementation used is already heavily optimized, even minor further performance improvements are notable.

Finally, WAND benefits the least of the three algorithms, a consequence of the interaction between under/over-prediction accuracy and the global upper bounds computation, explored further in the next section. Taking all trade-offs into consideration, the BLR algorithm appears to be the best choice overall, with $q = 0.1$.

6 REFINEMENTS

Over-Prediction Risk. The process described in Algorithm 1 requires that the query be re-evaluated (step 15) if the final heap threshold score (variable *h_threshold*) is still below the score threshold θ, which (in this scenario) will itself still be equal to $\hat{\Theta}_k$. The re-computation is required in order to be certain that the answer set returned is indeed the top-k for the query. However, the test is pessimistic, since it may be the case that every one of the top-k documents has had its score computed and been added to the heap, that *h_threshold* $= \Theta_k$, and that the uncertainty has arisen solely because of an over-prediction, with $\hat{\Theta}_k > \Theta_k$. To explore the likelihood of such situations, the three pruning methods were tested with controlled over-prediction, setting $\hat{\Theta}_k$ to increasing multiples of Θ_k. In each case the fraction of queries for which the heap nevertheless contained a correct top-k answer set (including allowing for document swaps caused by tie-breaking issues) for the query was recorded at each over-prediction level.

The results of this experiment are presented in Figure 6 on the ClueWeb09B collection for $k = 1{,}000$. The low resilience displayed by the BMW pruning mechanism compared to the other two options is a direct consequence of its ability to bypass a higher fraction of the documents. On the other hand, the propensity of WAND to score more documents than the bound θ would suggest, a result of its more generous calculation of *U_bound*(d), means that it is more likely to have generated a correct answer. The same trends were observed for smaller answer sets ($k = 10$ and $k = 100$) on both collections.

Reducing the Re-Computation. To exploit this pattern of behavior, Algorithm 2 describes an enhancement that can be employed in situations when document similarity is based on a scoring regime in which $f_2(t_i, Q)$ is constant, and WAND pruning is being used. If those conditions hold, then a q term query Q has at most $2^q - 1$ different values of *U_bound*, one for each non-empty combination of query terms, referred to here as being a *subquery*. Some of the subquery *U_bound* scores will already be greater than $\hat{\Theta}_k$, meaning that all documents containing those term combinations will have been fully scored during the first phase. Other subquery *U_bound* scores will be below *h_threshold*; those combinations can be safely ignored. In between these two bounds, the subqueries need to be explored, by

[1] https://github.com/pisa-engine/pisa

Method	Time	∇	Q_k	MLP					BLR				
				0.5	0.3	0.1	0.05	0.01	0.5	0.3	0.1	0.05	0.01
MaxScore	P_{50}	45.2	35.4	34.3	38.4	32.4	31.8	33.0	33.8	31.0	34.0	31.9	31.1
	P_{95}	167.2	155.7	197.6	218.9	189.0	175.9	166.6	182.4	155.1	148.6	141.5	132.7
	P_{99}	260.5	249.6	419.9	339.9	325.6	297.5	286.3	302.2	264.3	242.6	237.2	213.5
	Mean	61.6	51.8	60.2	66.6	55.9	52.7	51.5	56.6	49.0	49.7	47.1	45.0
WAND	P_{50}	33.0	26.9	28.2	27.8	26.0	26.0	25.3	26.6	24.7	25.9	25.4	27.6
	P_{95}	153.3	147.4	194.4	196.6	186.8	184.1	171.9	177.6	151.9	147.8	146.6	148.2
	P_{99}	272.4	269.4	427.7	428.7	414.5	393.1	386.3	379.9	287.6	268.2	267.6	266.4
	Mean	51.2	45.1	56.5	56.7	52.5	51.5	49.3	51.8	44.7	44.5	44.6	45.5
BMW	P_{50}	16.8	13.6	14.4	13.8	13.2	13.1	13.2	14.1	13.2	13.0	13.4	13.6
	P_{95}	82.4	82.5	98.8	97.9	88.6	89.3	80.0	94.5	83.1	78.4	79.4	79.2
	P_{99}	153.0	151.9	186.8	186.3	180.1	178.2	159.9	184.3	165.0	152.1	158.2	148.2
	Mean	27.1	24.9	28.5	28.0	25.8	25.1	24.4	27.1	24.7	23.9	24.4	24.3

Table 4: Algorithm 1 query execution times, retrieving $k = 1{,}000$ results from the ClueWeb09B collection, predicting $\hat{\Theta}_k$ using two baseline methods (∇ and Q_k) and five quantiles q for the MLP and BLR predictors. The best value in each row is in red. All times are in milliseconds.

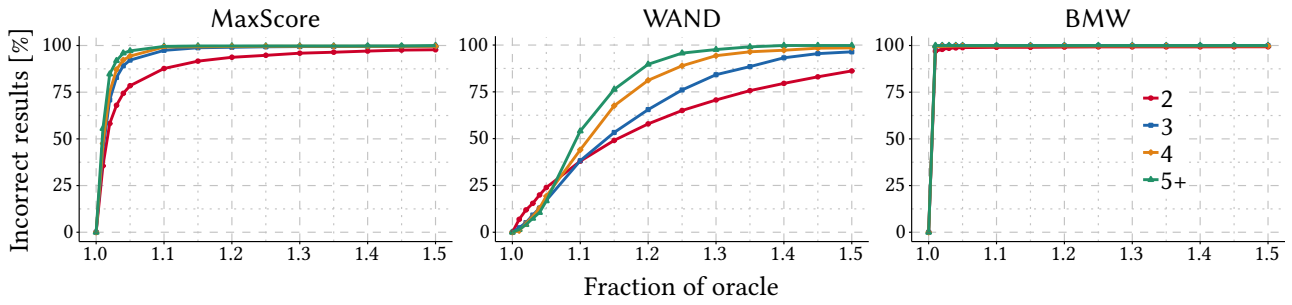


Figure 6: Percentage of instances in which an incorrect answer set is generated when the estimated threshold $\hat{\Theta}_k$ is greater than the true threshold Θ_k for queries of varying lengths. Note the vulnerability of the BMW implementation to this issue.

determining the documents that match the subquery as a Boolean conjunction, and then scoring those documents by summing their $C(t,d)$ contributions. Theobald et al. [49] and Mamoulis et al. [40] have made similar observations.

The patching computation is then one of conjunctive processing only, with no further element of pruning or WAND-style bypass, nor any re-sorting of candidates. As a result, it can be very efficient. Note also that if the subqueries that have *U_bound* scores that fall within the possible zone between *h_threshold* and $\hat{\Theta}_k$ are processed in decreasing order of *U_bound* (requiring a small sorting step not explicitly shown in Algorithm 2), then only the terms actually contained in each subquery need to be processed, and it is not necessary to locate and include score components for terms not part of the subquery. This restriction is correct since any documents that contain terms outside the current subquery would have already been scored either as part of the first phase, or as part of an earlier conjunctive subquery. All that is required is that a bit-vector be maintained through the first phase that records, for each document $d \in \mathcal{D}$, whether or not d has yet been scored, as is assumed at step 13 in Algorithm 2. Use of that bit-vector prevents documents being considered more than once for entry to the heap, and means that as each subquery is processed, it is exactly the subquery's subset of the terms that need to have their

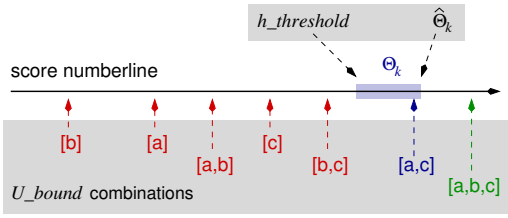


Figure 7: Example situation for three-term query in which $\hat{\Theta}_k$ is an over-estimate of Θ_k. The final *h_threshold* and $\hat{\Theta}_k$ bound Θ_k and hence one more term conjunct needs to be processed, covering documents containing both "a" and "c".

score contributions $C(t,d)$ summed for matching documents d. Note that *h_threshold* continues to be dynamic, and that as subqueries of decreasing *U_bound* are processed, *h_threshold* increases whenever high-scoring documents are identified. The patching process can terminate as soon as *h_threshold* reaches $\hat{\Theta}_k$, or if there are no more subquery *U_bound* values in the interval $[h\_threshold, \hat{\Theta}_k]$.

Figure 7 provides an example of the revised WAND-specific patching process, with $\hat{\Theta}_k$ greater than Θ_k, and, at the completion of

Algorithm 2 Computation of $\mathcal{A}(\mathcal{D},Q)$ based on a estimated score threshold and the knowledge that there are only $2^q - 1$ distinct values possible for $U\_bound(d)$.

```
execute steps 1 to 13 of Algorithm 1, recording the documents
    that get scored via a bit-vector
if h_threshold < Θ̂_k then                          ▷ cannot guarantee answer
    for each of the 2^q − 1 different subqueries of Q do
        set U ← U_bound[the subquery]
        if U > Θ̂_k then
            // the subquery has already been fully considered
        else if U < h_threshold then
            // the subquery cannot contribute to the top k
        else
            // check the subquery as a Boolean conjunction
            for each d ∈ D that matches the subquery do
                if d has not already been scored then
                    execute steps 6 to 13 of Algorithm 1,
                        scoring only for terms in the subquery
                    note in the bit-vector that d has been scored
return the k documents in the heap
```

phase one processing, the heap threshold still below $\hat{\Theta}_k$. Of the $2^q - 1$ combinations of query terms, only a single U_bound values is in the critical range, and only a single second-phase query conjunct needs to be evaluated, that for "a" ∧ "c", and any matching documents properly scored. Wu and Fang [54] and Daoud et al. [11] have also observed that documents can be scored by considering the upper bounds associated with discrete combinations of terms, and that seeking out high U_bound combinations early in the computation may allow small U_bound ones to be avoided.

Overestimation Refinements. Our final experiment aims to quantify the efficiency of Algorithm 2, so we compare it directly to Algorithm 1 which does not implement the enhanced bounds checking. Table 5 shows the efficiency of each algorithm. Note that, since both Algorithms employ the same WAND traversal in the first stage, the first stage computation for both algorithms is equivalent. The enhanced Algorithm 2 clearly outperforms the corresponding Algorithm 1 across the more aggressive quantiles, with a much smaller performance gap as the quantile decreases. With very small quantiles, where overprediction is rare, the enhancement is not advantageous. However, with $q = 0.50$ the advantages become quite clear – the efficiency is best for this configuration for all measures except for the P_{99} tail latency, and mean performance gains over Algorithm 1 of up to 22% are observed. Table 6 provides further evidence that the new patching algorithm greatly lowers the overprediction costs. Clearly, the enhanced bounds checking allows early-termination more often, improving efficiency (since a second-pass is not required in these cases). Interestingly, the enhanced algorithm scores fewer documents on average in the patching phase as compared to our initial approach, which indicates that the ranked conjunction is a suitable and efficient patching mechanism.

7 CONCLUSION

We have shown that heap threshold scores can be predicted (RQ1), that they improve the performance of dynamic pruning algorithms

Method	Time	BLR				
		0.5	0.3	0.1	0.05	0.01
Alg-1	P_{50}	26.6	24.7	25.9	25.4	27.6
Alg-2		21.2	22.2	24.9	26.3	27.2
Alg-1	P_{95}	177.6	151.9	147.8	146.7	148.2
Alg-2		135.8	137.8	143.3	147.4	148.9
Alg-1	P_{99}	379.9	287.6	268.2	267.6	266.4
Alg-2		285.9	272.2	267.1	270.2	272.6
Alg-1	Mean	51.8	44.7	44.5	44.6	45.5
Alg-2		40.3	40.8	43.1	44.9	45.9

Table 5: Algorithm 1 and Algorithm 2 compared (WAND processing only), using $k = 1{,}000$, ClueWeb09B, and the BLR approach.

Measure		BLR				
		0.5	0.3	0.1	0.05	0.01
Docs scored patching	Alg-1	233.2	70.5	11.4	3.7	0.7
	Alg-2	155.9	44.6	6.5	1.6	0.3
% requiring patching	Alg-1	56.9	27.6	6.1	2.5	0.4
	Alg-2	24.1	11.1	2.5	1.1	0.2

Table 6: Algorithm 1 and Algorithm 2 statistics (WAND processing only), $k = 1{,}000$, ClueWeb09B, and the BLR approach.

(RQ2), and that algorithmic variations are also possible (RQ3). We demonstrate that the key bottleneck in predicting the initial threshold is the risk of over-prediction. In order to address this problem, we provide an additional algorithmic modification that reduces the costs of second pass processing. By iteratively applying ranked conjunction, the number of additional documents scored is dramatically reduced in the WAND algorithm, and overall efficiency improved. Our investigation also provides evidence that further work on prediction algorithms and dynamic score patching is warranted. The iterative patching algorithm might be useful in other dynamic pruning scenarios as well, potentially leading to new query processing paradigms.

From the modeling perspective, one avenue for improvement is to devise non-linear probabilistic predictors. It is possible to combine the neural approach with the Bayesian formulation in order to obtain predictive distributions, but this can be prohibitively slow due to the lack of a fast, closed-form solution for inference. A compelling alternative is Gaussian Processes (GPs) [46], which enable non-linear models with exact inference for regression. Their predictions can be combined with MBR in similar ways, as explored by Beck et al. [1]. While standard GPs can be difficult to scale, recently proposed scalable alternatives [21] could provide a way to enable efficient probabilistic non-linear predictions. Another avenue is to explore different asymmetric loss functions, such as the linear exponential [52], which is commonly used in econometrics. The problem of predicting which quantile is suitable on a per-query basis is a promising avenue for even further efficiency gains.

A final area for exploration is reducing (or compressing) the model features. The large index footprint of our proposals renders them of

marginal value for practical deployment, and a smaller model would help make these techniques more attractive to system developers.

Acknowledgment. The third author was supported by an Australian Research Training Program Scholarship. The fifth author was suppported by the Australian Research Council (DP160102686).

REFERENCES

[1] D. Beck, L. Specia, and T. Cohn. Exploring prediction uncertainty in machine translation quality estimation. In *Proc. CoNLL*, pages 208–218, 2016.

[2] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proc. CIKM*, pages 426–434, 2003.

[3] C. Burges. From RankNet to LambdaRank to LambdaMart: An overview. *Learning*, 11(23-581):81, 2010.

[4] 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.

[5] R.-C. Chen, L. Gallagher, R. Blanco, and J. S. Culpepper. Efficient cost-aware cascade ranking in multi-stage retrieval. In *Proc. SIGIR*, pages 445–454, 2017.

[6] P. F. Christoffersen and F. X. Diebold. Optimal prediction under asymmetric loss. *Econometric Theory*, 13(06):808–817, 1997.

[7] 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.

[8] N. Craswell, R. Jones, G. Dupret, and E. Viegas, editors. *Proc. 2009 Workshop on Web Search Click Data: WSCD@WSDM*. ACM, 2009.

[9] J. S. Culpepper, C. L. A. Clarke, and J. Lin. Dynamic cutoff prediction in multi-stage retrieval systems. In *Proc. Aust. Doc. Comp. Symp.*, pages 17–24, 2016.

[10] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos. Distributional reinforcement learning with quantile regression. In *Proc. AAAI*, pages 2892–2901, 2018.

[11] C. M. Daoud, E. S. de Moura, D. Fernandes, A. S. da Silva, C. Rossi, and A. Carvalho. Waves: A fast multi-tier top-k query processing algorithm. *Inf. Retr.*, 20(3): 292–316, 2017.

[12] D. Dato, C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. Fast ranking with additive ensembles of oblivious and non-oblivious regression trees. *ACM Trans. Inf. Sys.*, 35(2):15.1–15.31, 2016.

[13] L. L. S. de Carvalho, E. S. de Moura, C. M. Daoud, and A. S. da Silva. Heuristics to improve the BMW method and its variants. *J. Data Inf. Qual.*, 6:178–191, 2015.

[14] L. Dhulipala, I. Kabiljo, B. Karrer, G. Ottaviano, S. Pupyrev, and A. Shalita. Compressing graphs and indexes with recursive graph bisection. In *Proc. KDD*, pages 1535–1544, 2016.

[15] C. Dimopoulos, S. Nepomnyachiy, and T. Suel. Optimizing top-k document retrieval strategies for block-max indexes. In *Proc. WSDM*, pages 113–122, 2013.

[16] S. Ding and T. Suel. Faster top-k document retrieval using block-max indexes. In *Proc. SIGIR*, pages 993–1002, 2011.

[17] M. Fontoura, V. Josifovski, J. Liu, S. Venkatesan, X. Zhu, and J. Zien. Evaluation strategies for top-k queries over memory-resident inverted indexes. *Proc. VLDB*, 4(12):1213–1224, 2011.

[18] J. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

[19] Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *Proc. ICML*, pages 1050–1059, 2016.

[20] Y. He, J. Tang, H. Ouyang, C. Kang, D. Yin, and Y. Chang. Learning to rewrite queries. In *Proc. CIKM*, pages 1443–1452, 2016.

[21] J. Hensman, N. Fusi, and N. D. Lawrence. Gaussian processes for big data. In *Proc. UAI*, pages 282–290, 2013.

[22] P. J. Huber. Robust estimation of a location parameter. *Ann. Math. Statist.*, 35(1): 73–101, 1964.

[23] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. ICML*, pages 448–456, 2015.

[24] X. Jin, T. Yang, and X. Tang. A comparison of cache blocking methods for fast execution of ensemble-based score computation. In *Proc. SIGIR*, pages 629–638, 2016.

[25] A. Kane and F. W. Tompa. Split-lists and initial thresholds for WAND-based search. In *Proc. SIGIR*, pages 877–880, 2018.

[26] Y. Kim, J. Callan, J. S. Culpepper, and A. Moffat. Does selective search benefit from WAND optimization? In *Proc. ECIR*, pages 145–158, 2016.

[27] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proc. ICLR*, pages 1–15, 2015.

[28] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *Proc. SIGMOD*, pages 489–504, 2018.

[29] C. Louizos, K. Ullrich, and M. Welling. Bayesian compression for deep learning. In *Proc. NeurIPS*, pages 3288–3298, 2017.

[30] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. Quickscorer: A fast algorithm to rank documents with additive ensembles of regression trees. In *Proc. SIGIR*, pages 73–82, 2015.

[31] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. Exploiting CPU SIMD extensions to speed-up document scoring with tree ensembles. In *Proc. SIGIR*, pages 833–836, 2016.

[32] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, and S. Trani. X-DART: Blending dropout and pruning for efficient learning to rank. In *Proc. SIGIR*, pages 1077–1080, 2017.

[33] C. Macdonald, N. Tonellotto, and I. Ounis. Efficient and effective selective query rewriting with efficiency predictions. In *Proc. SIGIR*, pages 495–504, 2017.

[34] D. J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1992.

[35] J. Mackenzie, J. S. Culpepper, R. Blanco, M. Crane, C. L. A. Clarke, and J. Lin. Query driven algorithm selection in early stage retrieval. In *Proc. SIGIR*, pages 396–404, 2018.

[36] J. Mackenzie, A. Mallia, M. Petri, J. S. Culpepper, and T. Suel. Compressing inverted indexes with recursive graph bisection: A reproducibility study. In *Proc. ECIR*, pages 339–352, 2019.

[37] A. Mallia and E. Porciani. Faster BlockMax WAND with longer skipping. In *Proc. ECIR*, pages 771–778, 2019.

[38] A. Mallia, G. Ottaviano, E. Porciani, N. Tonellotto, and R. Venturini. Faster BlockMax WAND with variable-sized blocks. In *Proc. SIGIR*, pages 625–634, 2017.

[39] A. Mallia, M. Siedlaczek, and T. Suel. An experimental study of index compression and DAAT query processing methods. In *Proc. ECIR*, pages 353–368, 2019.

[40] N. Mamoulis, M. L. Yiu, K. H. Cheng, and D. W. Cheung. Efficient top-k aggregation of ranked inputs. *ACM Trans. Data. Sys.*, 32(3):19, 2007.

[41] B. Mitra and N. Craswell. An introduction to neural information retrieval. *Found. Trnd. Inf. Retr.*, 13(1):1–126, 2018.

[42] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[43] G. Ottaviano and R. Venturini. Partitioned Elias-Fano indexes. In *Proc. SIGIR*, pages 273–282, 2014.

[44] S. Peter, F. Diego, F. A. Hamprecht, and B. Nadler. Cost efficient gradient boosting. In *Proc. NeurIPS*, pages 1550–1560, 2017.

[45] M. Petri, J. S. Culpepper, and A. Moffat. Exploring the magic of WAND. In *Proc. Aust. Doc. Comp. Symp.*, pages 58–65, 2013.

[46] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[47] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15:1929–1958, 2014.

[48] T. Strohman, H. R. Turtle, and W. B. Croft. Optimization strategies for complex queries. In *Proc. SIGIR*, pages 219–225, 2005.

[49] M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In *Proc. VLDB*, pages 648–659, 2004.

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

[51] H. R. Turtle and J. Flood. Query evaluation: Strategies and optimizations. *Inf. Proc. & Man.*, 31(6):831–850, 1995.

[52] H. Varian. A Bayesian approach to real estate assessment. In S. E. Fienberg and A. Zellner, editors, *Studies in Bayesian Econometrics and Statistics in Honor of Leonard J. Savage*, pages 195–208. 1975.

[53] L. Wang, J. Lin, and D. Metzler. A cascade ranking model for efficient ranked retrieval. In *Proc. SIGIR*, pages 105–114, 2011.

[54] H. Wu and H. Fang. Document prioritization for scalable query processing. In *Proc. CIKM*, pages 1609–1618, 2014.

[55] Z. Xu, M. J. Kusner, K. Q. Weinberger, M. Chen, and O. Chapelle. Classifier cascades and trees for minimizing feature evaluation cost. *J. Mach. Learn. Res.*, 15:2113–2144, 2014.

[56] D. Yin, Y. Hu, J. Tang, T. Daly, M. Zhou, H. Ouyang, J. Chen, C. Kang, H. Deng, C. Nobata, J.-M. Langlois, and Y. Chang. Ranking relevance in Yahoo search. In *Proc. KDD*, pages 323–332, 2016.

[57] H. Zamani, M. Dehghani, W. B. Croft, E. Learned-Miller, and J. Kamps. From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing. In *Proc. CIKM*, pages 497–506, 2018.