

# Early Exit Optimizations for Additive Machine Learned Ranking Systems

B. Barla Cambazoglu  
Yahoo! Research  
Barcelona, Spain  
barla@yahoo-inc.com

Jiang Chen  
Yahoo! Labs  
Sunnyvale, CA, USA  
jiangc@yahoo-inc.com

Hugo Zaragoza  
Yahoo! Research  
Barcelona, Spain  
hugoz@yahoo-inc.com

Ciya Liao  
Yahoo! Labs  
Sunnyvale, CA, USA  
ciyaliao@yahoo-inc.com

Olivier Chapelle  
Yahoo! Research  
Sunnyvale, CA, USA  
chap@yahoo-inc.com

Zhaohui Zheng  
Yahoo! Labs  
Sunnyvale, CA, USA  
zhaohui@yahoo-inc.com

## ABSTRACT

Some commercial web search engines rely on sophisticated machine learning systems for ranking web documents. Due to very large collection sizes and tight constraints on query response times, online efficiency of these learning systems forms a bottleneck. An important problem in such systems is to speedup the ranking process without sacrificing much from the quality of results. In this paper, we propose optimization strategies that allow short-circuiting score computations in additive learning systems. The strategies are evaluated over a state-of-the-art machine learning system and a large, real-life query log, obtained from Yahoo!. By the proposed strategies, we are able to speedup the score computations by more than four times with almost no loss in result quality.

## Categories and Subject Descriptors

H.3.3 [Information Storage Systems]: Information Retrieval Systems

## General Terms

Algorithms, Performance

## Keywords

Web search, machine learning, early exit, optimization

## 1. INTRODUCTION

A traditional problem in web search is to estimate the relevance of a set of documents to a given user query, rank the documents in decreasing order of their relevance, and present the  $k$  most relevant documents to the user. Due to inadequacy of simple statistical scoring schemes [3] in satisfying user expectations, some major search engines employ

complex scoring functions learned by machine learning systems [7, 9, 28]. Such systems are trained offline by using a training set made up of hundreds of features. Typically, the objective in learning is to minimize a metric that estimates the overall error between editorially assigned document relevance scores and the relevance scores predicted by the machine learning system. In the online phase, given a query, documents are scored using the previously trained system. The top  $k$  documents are then ranked in decreasing order of predicted scores and returned to the user.

The scoring process needs to be fast enough so that query response time constraints can be satisfied (typically, a few hundred milliseconds for web queries). In practice, it is not possible to score every document using expensive machine learned ranking techniques. Therefore, a two-phase scoring scheme is usually employed. In the first phase, a simple but inaccurate scoring technique (e.g., a BM25 variant) is used for selecting a small subset of potentially relevant documents from the entire collection. In the second phase, selected documents are rescored by a complex but accurate machine learned ranking architecture, similar to what we described. The final ranking is determined by the document scores computed in the second phase.

Many machine learned ranking architectures are based on additive ensembles (e.g., boosted decision trees [33]), where many scorers are executed sequentially in a chain and score contributions of individual scorers are accumulated to compute the final document scores. That is, each document goes through the entire chain of scorers receiving a partial contribution to its final predicted score. Predicted ranks of documents are continually refined as more scorers are applied. Since the scoring process is online and the number of scorers is high, efficiency of this process is important. If the process is not optimized, only a limited number of documents can be scored within the allowed time. Any optimization that aims to speed up the scoring process, however, should not lead to an intolerable loss in result quality.

Two features of web search allow short-circuiting the scoring process in additive ensembles. First, document relevance follows a skewed distribution. For most queries, typically, there are very few highly relevant documents, but many irrelevant documents. Second, most users view only the first few result pages (e.g., 10 or 20 results). Hence, it may be possible to terminate scoring of documents that are unlikely to be ranked within the top  $k$  earlier. This way, unneces-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WSDM'10, February 4–6, 2010, New York City, New York, USA.  
Copyright 2010 ACM 978-1-60558-889-6/10/02 ...\$10.00.

sary score computations for many, potentially less relevant documents are avoided, speeding up the ranking process.

The above-mentioned approach requires estimating via predictive functions, which we refer to as early exit functions, the likelihood of a document being ranked in the final top  $k$  documents. In this paper, our focus is on devising such functions for additive machine learning systems. We make a formal definition of the problem and state its differences from the existing vector-space ranking optimizations in traditional IR literature. We propose four different early exit functions and evaluate their performance using a state-of-the-art machine learning system based on gradient boosted decision trees. As the data set, we use a large, real-life query and result log obtained from Yahoo!.

The proposed early exit functions lead to considerable speed improvements under no or little loss in result quality. The performance improvements achieved by this work can be valuable for large-scale search engines in different ways. First, given the same amount of time, more documents can be scored by the second phase ranking, i.e., accurate machine learning systems. Second, more costly but precise ranking systems can now be afforded, potentially improving the result quality [8]. Third, if the quality is not a major concern, response times can be decreased, query throughput can be increased, or hardware costs can be reduced.

The rest of the paper is organized as follows. In Section 2, we formally state the early exit problem. Section 3 surveys the related work in the literature and compares our work with existing early exit optimizations in vector-space ranking. Proposed early exit functions are described in Section 4. The experimental setup is described in Section 5. Section 6 presents various performance results. We discuss further research issues and the limitations of our work in Section 7. The paper is concluded in Section 8.

## 2. RANKING AND EARLY EXITING IN ADDITIVE ENSEMBLES

### 2.1 Ranking in Additive Ensembles

Additive ensembles are used in several of the most successful machine learning algorithms. These include kernel methods and SVMs [26], boosting [15], bagging [4], and generalized additive models [17]. In additive ensembles, the final score  $s(x_i)$  of an object  $x_i$  (e.g., a query-document pair) is computed by a sum over many, simple scorers as  $s(x_i) = \sum_{j=1}^N f_j(x_i)$ , where  $f_j$  is a scorer that belongs to an ensemble  $\mathcal{F} = \langle f_1, \dots, f_N \rangle$  of  $N$  scorers, each executed in a sequence. The scorers in  $\mathcal{F}$  are expected to be sorted in decreasing order of importance before the execution.

In ranking, we have a set  $\mathcal{D} = \{d_1, \dots, d_M\}$  of  $M$  documents that we want to rank according to their relevance to a query  $q$ . For each document  $d_i$ , the ensemble produces a score  $s(d_i, q) = \sum_{j=1}^N f_j(d_i, q)$ , indicating the relevance of  $d_i$  to  $q$ . Then, scores are sorted in decreasing order, and documents with the top  $k$  scores are returned to the user.

If we assume that the cost of computing  $f_j(d_i, q)$  is a constant  $c$  for all  $(d_i, q)$  and all  $f_j$  (this is usually the case), then the cost of computing  $s(d_i, q)$  is  $cN$ . The total cost of scoring all documents in collection  $\mathcal{D}$  against the query then becomes  $C(\mathcal{D}) = cMN$ . For tasks with tight constraints on execution time (e.g., web search), this cost is not affordable if both  $M$  and  $N$  are high (e.g.,  $M > 100,000$  and  $N > 1000$ ).

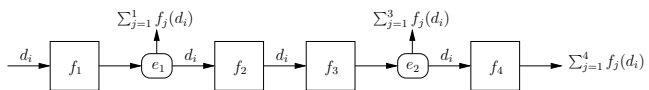


Figure 1: An additive ensemble with early exits.

### 2.2 Early Exits

In ranking, the main objective is to have documents correctly ranked, i.e., scores do not need to be very accurate as long as the final ordering of documents is correct. Moreover, in web search, users are most often interested only in documents at high ranks (i.e., top  $k$ ). It is relatively more important to correctly select these documents. Finally, scorers in additive ensembles are typically sorted in decreasing importance order (or can always be sorted explicitly). These together make it possible to speed up the scoring process in additive ensembles as score computations can be short-circuited, i.e., some documents are not scored by all scorers.

In general, we can early exit the score computation of a pair  $(d_i, q)$  at any position  $\ell < N$ , computing a partial final score  $\hat{s}(d_i, q) = \sum_{j=1}^{\ell} f_j(d_i, q)$  using only the first  $\ell$  scorers. This decision requires estimating the likelihood that  $d_i$  will end up within the top  $k$  documents and accordingly either continuing the score accumulation for  $(d_i, q)$  or exiting at some position  $\ell$ , right after scorer  $f_{\ell}$  is executed. We refer to functions responsible for this decision as early exit functions.

In its most general form, an early exit function  $F_{\ell}(d_i, q, H)$  at position  $\ell$  takes as input the scored (document, query) pair and some history information  $H$  about all previous decisions. Although it is possible to place an exit function at every position  $1 \leq \ell < N$ , for computational efficiency, it makes sense to place the early exit functions selectively and sparsely over a sequence  $e = \langle e_1, \dots, e_L \rangle$  of  $L$  positions such that  $L \ll N$ . Fig. 1 shows an example additive ensemble composed of four scorers and two early exit functions.

Given all these, we can now define an early exit strategy  $\mathcal{E} := (e, \{F_{\ell}\}_{\ell \in e})$  as a sequence of early exit positions and corresponding early exit functions. The cost of ranking a collection  $\mathcal{D}$  by an exit strategy  $\mathcal{E}$  is given by  $C(\mathcal{D}, \mathcal{E}) = c \sum_{i=1}^M p(d_i, q)$ , where  $p(d_i, q)$  is the position at which the score computation of  $(d_i, q)$  is terminated. Note that this equation excludes the cost of executing the exit functions. If the exit functions are costly or there are many positions where we try to exit early, then the cost of exit functions should also be considered in estimating the total cost.

Unfortunately, terminating score computations early may degrade result quality as some documents, now, will not be in their optimal ranks, i.e., there is a compromise between speed and quality. Let us call  $\mathcal{R}_q^*$  the ground-truth result set obtained in case of full score computation and  $\mathcal{R}_q^{\mathcal{E}}$  the result set obtained using the exit strategy  $\mathcal{E}$ . Also, let  $\chi_k(\mathcal{R}_q^*, \mathcal{R}_q^{\mathcal{E}})$  be a function that measures the relevance loss in top  $k$  (e.g., loss in P@ $k$ ) with  $\mathcal{R}_q^{\mathcal{E}}$  relative to  $\mathcal{R}_q^*$ . The early exit problem in additive ensembles can now be stated as finding an exit strategy  $\mathcal{E}$  that minimizes the expected relevance loss  $E_q\{\chi_k(\mathcal{R}_q^*, \mathcal{R}_q^{\mathcal{E}})\}$  while satisfying  $C(\mathcal{D}, \mathcal{E}) \leq \gamma$ , where  $\gamma$  is the maximum allowed execution cost. In other words, the objective is to minimize the relevance loss due to early exits while guaranteeing a given maximum execution cost requirement<sup>1</sup>.

<sup>1</sup>Alternatively, we can minimize  $C(\mathcal{D}, \mathcal{E})$  constraining  $E_q\{\chi_k(\mathcal{R}_q^*, \mathcal{R}_q^{\mathcal{E}})\}$  or minimize both jointly.

### 3. RELATED WORK

#### 3.1 Machine Learned Ranking

Document retrieval has traditionally been based on a manually designed ranking function (e.g., BM25). However, web page ranking is considered as a supervised learning problem, and several machine learning algorithms have already been applied to it. One of the earliest publications in the context of web page ranking is [7], where a neural network is trained on pairwise preferences. Pairwise learning is now one of the most popular techniques: it can be used directly with linear functions [9] or it can be combined within the boosted decision tree framework [33]. Recent work has addressed the possibility of directly optimizing ranking measures such as MAP [32] or NDCG [28]. Finally, it is noteworthy that a simple regression on regression labels [11] turns out to be competitive compared to more advanced techniques [20].

#### 3.2 Early Exit Optimizations in ML

A few early exit optimizations have already been proposed in the machine learning community. One example is the face detection algorithm in [30], which is based on a cascade of simple classifiers. In this algorithm, a classifier is executed and if its prediction is positive (i.e., a face is detected), then another classifier is triggered. Otherwise, the execution is short-circuited. This process continues until all classifiers agree that there is a face. This approach is shown to save a lot of computational power.

In the context of nonlinear support vector machines (a particular case of additive ensembles), two algorithms have been proposed to reduce the online evaluation cost of the decisions [12, 25]. These works are based on the idea that the points that are far away from the decision boundary can be classified very quickly with high confidence. To best of our knowledge, there are no prior works addressing early exit optimization issues in additive ensembles.

#### 3.3 Early Exit Optimizations in IR

There are many works on early exit optimization for query-document similarity computations using inverted indices [3]. These works aim to increase search efficiency without degrading the quality of results, typically, by limiting the number of postings processed and the number of documents that are candidates for top  $k$ . They mainly differ in the way they process postings and early exit criteria they employ.

Buckley and Lewit [6] proposed an algorithm that traverses posting lists in decreasing order of  $idf$  values and quits processing of lists when it is guaranteed that no new document can achieve a score high enough to enter the final top  $k$ . Wong and Lee [31] considered also the maximum  $tf$  value in each list while sorting the lists. Moreover, postings are stored in decreasing order of  $tf$  values, and lists are accessed from disk in units of disk pages, allowing concurrent traversal of lists. They proposed two simple heuristics for predicting the point where the top  $k$  document set is stabilized to exit scoring computations early. Persin [24] used early exit strategies within a posting list. He used frequency-sorted indices, where all postings after the first posting with a  $tf-idf$  score contribution less than a certain update threshold are not processed. His strategy also involved an insertion threshold for limiting the size of the candidate set.

Harman and Candela [16] suggested fully scoring all documents in posting lists of the terms with an  $idf$  value less than

a certain fraction of the maximum  $idf$  of any term in the vocabulary. In processing the remaining lists, only the scores of the previously scored documents are updated. Moffat and Zobel [22] extended this idea by replacing the threshold on lists with a threshold on the number of score accumulators allocated. In their *quit* heuristic, once the number of accumulators reached a prespecified threshold, the computation is halted. Their *continue* heuristic is similar, but after the threshold is reached the remaining lists are continued to be processed, updating only the previously allocated accumulators. Later, Anh et al. [1] proposed a scheme combining the quit mechanism of [22] with impact-sorted lists, where fully computed scores are stored in postings. Anh and Moffat [2] extended the *continue* heuristic and impact-sorted lists with an additional phase, in which the top  $k$  accumulators are further continued to be refined by score updates.

Unlike the above-mentioned term-at-a-time scoring approaches, a separate thread of works investigated document-at-a-time scoring. Brown [5] suggested maintaining the top-weighted  $n$  documents for each document-id-sorted list. Postings in low- $idf$  lists are not considered in score updates if the corresponding document is not in the top document set of the list and does not appear in a high- $idf$  list. Turtle and Flood [29] presented a heuristic for document-at-a-time processing on frequency-sorted lists. Strohan et al. [27] combined the ideas in [5] and [29] under a new algorithm.

Further related work can be found in [13], [14], and [18].

#### 3.4 Comparison with Works in IR

It is possible to make an analogy between machine learning scorers and posting lists as postings provide precomputed partial score information (i.e., weights) for documents. Despite this analogy, scorers and posting lists differ in a number of ways. First, the number of scorers are in the order of thousands [20], whereas the number of lists is equal to the query length, which is typically low. Second, a scorer contributes to every document's score, whereas a list contributes to a subset of documents. Third, scores provided by lists are independent of the query. However, scorers may use features that result in query-dependent scores.

The main idea behind optimizations for both vector-space ranking and machine learned ranking is the same: form the top  $k$  document set as fast as possible, without sacrificing from result quality. However, due to the following reasons, existing optimizations in vector-space ranking are not directly applicable to the early exit problem described in this work. First, it is not possible to employ a heuristic similar to sorting postings within a list in decreasing weight order [1, 24, 31] or keeping additional score information [5, 27]. This is because no information is available about the score contributions generated by the scorers before the execution as scores become available at run-time. Second, the assumption about monotonically increasing scores [6] is not valid since scorers can assign both negative and positive scores. Hence, the documents highly scored by early scorers may end up with low scores (although not very likely). Third, in most IR works, the primary objective is to reduce relatively expensive costs such as disk accesses [6], decompression of postings [22], or sorting the candidate document set [16]. However, features used by the scorers are already in the memory and individual scorers are cheap. Therefore, expensive early exit algorithms cannot be afforded. Also, frequency and placement of early exit functions is an issue.

---

**Algorithm 1** A generic algorithm for EST.

---

**Require:**  $k$ : number of documents requested  
**Require:**  $st[1 \dots N]$ : array of score thresholds  
1:  $D \leftarrow \{1 \dots M\} \triangleright$  set of documents not exited  
2: **for**  $d = 1$  to  $M$  **do**  
3:   **for**  $p = 1$  to  $N$  **do**  
4:      $score[d] \leftarrow score[d] + \text{SCORE}(p, d)$   
5:     **if**  $score[d] < st[p]$  **then**  
6:        $D \leftarrow D - \{d\} \triangleright$  early exit for document  $d$   
7:       **break**  
8: **return** the highest scored  $k$  documents in  $D$

---

## 4. EARLY EXIT ALGORITHMS

### 4.1 Traversal Order

In additive ensembles, two different traversal orders are possible in score computations. Scores can be separately computed for each document by iterating on all scorers, i.e., a document-ordered traversal strategy (DOT), where the outer loop iterates on documents and the inner loop iterates on scorers. Alternatively, scores can be separately computed for each scorer by iterating on all documents, i.e., a scorer-ordered traversal strategy (SOT), where the outer loop iterates on scorers and the inner loop iterates on documents. In DOT, at every iteration of the outer loop, we obtain the complete score information for a partial set of documents. In SOT, on the other hand, we obtain a partial score information for the entire set of documents. Certain early exit functions may require a particular traversal order.

Both techniques have their advantages and disadvantages. In general, DOT provides good cache hit rates in accessing feature vectors. Also, it is memory efficient as the feature vector of a document can be deallocated after the document is scored (i.e., after an iteration of the outer loop). SOT, on the other hand, requires keeping all feature vectors in memory until the last scorer is executed and has low cache utilization. However, the strategies adopting SOT result in better early exit performance as we will discuss.

### 4.2 Algorithms

In this section, we present the proposed early exit functions. All of our functions have a threshold value that needs to be tuned in an offline manner. Early exits are achieved based on score comparisons determined by these fixed thresholds. We evaluate four early exit functions, each with a different type of thresholding: score, capacity, rank, and proximity. We name the functions according to the type of thresholding as EST, ECT, ERT, and EPT, respectively. Among our functions, EST can work under both DOT and SOT schemes. However, ERT and EPT require the SOT scheme, and ECT is only meaningful with the DOT scheme<sup>2</sup>.

During the discussion and in the algorithms provided, for simplicity, we assume that there exists an early exit function between every two scorers (in the experiments, however, we place the exits sparsely). Scores are accumulated in the  $score$  array, and eliminated documents are removed from the initial candidate set  $D$ . Only the documents with a fully computed score, i.e., the ones that are still in  $D$  when all scoring computations terminate can appear in the top  $k$  re-

<sup>2</sup>In fact, ECT under the SOT scheme is equivalent to ERT as it will be clear later.

---

**Algorithm 2** A generic algorithm for ECT.

---

**Require:**  $k$ : number of documents requested  
**Require:**  $ct[1 \dots N]$ : array of heap capacity thresholds  
1:  $D \leftarrow \{1 \dots M\} \triangleright$  set of documents not exited  
2: **for**  $p = 1$  to  $N$  **do**  
3:    $H[p] \leftarrow \emptyset \triangleright$  initialize maximum score heaps  
4: **for**  $d = 1$  to  $M$  **do**  
5:   **for**  $p = 1$  to  $N$  **do**  
6:      $score[d] \leftarrow score[d] + \text{SCORE}(p, d)$   
7:     **if**  $\text{SIZE}(H[p]) < ct[p]$  **then**  
8:        $\text{PUSH}(H[p], score[d])$   
9:     **else**  
10:      **if**  $score[d] < \text{MIN}(H[p])$  **then**  
11:        $D \leftarrow D - \{d\} \triangleright$  early exit for document  $d$   
12:       **break**  
13:      **else**  
14:        $\text{POP}(H[p])$   
15:        $\text{PUSH}(H[p], score[d])$   
16: **return** the highest scored  $k$  documents in  $D$

---

sults. If there are less than  $k$  documents with fully computed scores, then the best (highest) partial scores can be used for filling the remaining slots (not shown in the algorithms).

#### 4.2.1 Early Exits Using Score Thresholds (EST)

A naive approach is to filter the documents based on the scores they have accumulated so far, i.e., we quit scoring the documents with low scores. In EST (Algorithm 1), exits are based on comparisons between accumulated scores and offline-computed score thresholds. That is, at each position  $p$ , we compare the current document score with the lowest (worst) permissible score threshold  $st[p]$ . If the accumulated score is less than  $st[p]$ , we quit scoring the document, i.e., no further score update is performed by remaining scorers.

#### 4.2.2 Early Exits Using Capacity Thresholds (ECT)

Statically computed score thresholds in EST may lead to poor exit decisions for many documents as distribution of scores vary depending on individual queries. A solution is to adjust the lowest permissible score threshold dynamically, during the score computation, i.e., based on previously observed document scores for the query. In ECT, at every exit position, a maximum score heap is maintained in order to record the best partial document scores observed so far at the current exit position. The capacity of the heap at position  $p$  is determined by threshold  $ct[p]$ .

During score computations (Algorithm 2), the first  $ct[p]$  document scores are unconditionally inserted into the heap. This is a warm-up period for collecting information about the score distribution. Afterwards, documents are eliminated based on comparisons between their current scores and the current worst score in the heap, i.e., if the document score is lower than the minimum score in the heap, we stop scoring the document. Otherwise, we pop the minimum score from the heap and push the current document score.

In this strategy, the order in which the documents are scored is very important. A good scoring order could be the decreasing order of document relevance as this scoring order increases the likelihood of early exits. However, since this order is not available (in fact, this is our objective), scores assigned priori by a cheaper scoring function can be used to obtain a reasonable scoring order for documents.

**Table 1: Comparison of early exit functions**

	Exit functions			
	EST	ECT	ERT	EPT
Threshold used	score	capacity	rank	proximity
Traversal order	DOT or SOT	DOT	SOT	SOT
Time complexity at position $p$	$O(M)$	$O(M \log ct[p])$	$O(M)$	$O(M)$
Total space complexity	$O(F)$	$O(F + \sum_{p=1}^N ct[p])$	$O(MF)$	$O(MF)$

---

**Algorithm 3** A generic algorithm for ERT.

---

**Require:**  $k$ : number of documents requested  
**Require:**  $rt[1 \dots N]$ : array of rank thresholds  
1:  $D \leftarrow \{1 \dots M\} \triangleright$  set of documents not exited  
2: **for**  $p = 1$  to  $N$  **do**  
3:   **for**  $d = 1$  to  $M$  **do**  
4:     **if**  $d \in D$  **then**  
5:        $score[d] \leftarrow score[d] + SCORE(p, d)$   
6:      $d' \leftarrow SELECT(D, rt[p])$   
7:     **for**  $d = 1$  to  $M$  **do**  
8:       **if**  $d \in D$  **then**  
9:         **if**  $score[d] < score[d']$  **then**  
10:          $D \leftarrow D - \{d\} \triangleright$  early exit for document  $d$   
11: **return** the highest scored  $k$  documents in  $D$

---

#### 4.2.3 Early Exits Using Rank Thresholds (ERT)

The weakness of ECT is that the exit decisions are based on only a partial set of previously seen document scores. Only a partial ranking information can be utilized due to the DOT scheme employed. In fact, having the complete rank information for documents is quite valuable as eventually getting the final document ranks right is the main objective. ERT, like any exit function using the SOT scheme, has the advantage of utilizing the global ranking information.

Early exits decisions in ERT (Algorithm 3) are based on comparisons between the current ranks of documents (computed over all documents) and offline-computed rank thresholds. At position  $p$ , the documents that survived previous exit tests are ranked in decreasing order of their scores. The documents having a rank better than an offline-computed rank threshold  $rt[p]$  are allowed for further scoring. The rest of the documents are eliminated. Algorithm 3 shows an efficient version which uses the linear-time selection algorithm [10] (SELECT) for finding the document at rank  $rt[p]$  (instead of sorting the documents) and prunes the documents based on this pivot document’s score.

#### 4.2.4 Early Exits Using Proximity Thresholds (EPT)

EPT (Algorithm 4) is similar to ERT in that document ranks are utilized. In EPT, however, the pivot document rank is always fixed to  $k$ . At an exit position  $p$ , the decision of eliminating a document is made based on an offline-computed score proximity threshold  $pt[p]$ . The idea is to keep scoring the documents that have a score close enough to the score of the document at the  $k$ th rank. Therefore, only the documents that are within the first  $k$  ranks as well as documents that are within a score proximity of the  $k$ th document’s score are continued to be scored by later scorers. The rest of the documents are early exited. In a sense, EPT is a hybrid scheme that combines the currently available rank and score information.

---

**Algorithm 4** A generic algorithm for EPT.

---

**Require:**  $k$ : number of documents requested  
**Require:**  $pt[1 \dots N]$ : array of difference thresholds  
1:  $D \leftarrow \{1 \dots M\} \triangleright$  set of documents not exited  
2: **for**  $p = 1$  to  $N$  **do**  
3:   **for**  $d = 1$  to  $M$  **do**  
4:     **if**  $d \in D$  **then**  
5:        $score[d] \leftarrow score[d] + SCORE(p, d)$   
6:      $d' \leftarrow SELECT(D, k)$   
7:     **for**  $d = 1$  to  $M$  **do**  
8:       **if**  $d \in D$  **then**  
9:         **if**  $score[d] < score[d'] - pt[p]$  **then**  
10:          $D \leftarrow D - \{d\} \triangleright$  early exit for document  $d$   
11: **return** the highest scored  $k$  documents in  $D$

---

### 4.3 Comparison of Functions

Early exit functions differ in several aspects, which are summarized in Table 1. Although the worst-case time complexities of ERT and EPT are equal to that of EST, in practice, EST is much faster as it involves only a single score comparison per document, whereas ERT and EPT require many comparisons per document during the SELECT operation. Note that, in early exit functions using SOT, feature vectors cannot be deallocated throughout the execution, and this may become an issue if  $F$  (the size of feature vectors) and  $M$  are both high (especially, if query processing is multi-threaded and many queries are concurrently processed).

### 4.4 Rank-Preserving Early Exit Algorithms

In this work, we prefer not to discuss algorithms that guarantee correctness of the top  $k$  results (with respect to ranking without optimization). This is because we aim to keep the algorithms as general as possible, instead of tuning for a specific type of scorers. Depending on properties of scorers, it may be possible to devise algorithms that preserve ranking. For example, in the framework of gradient boosted decision trees, which we consider in Section 6, the minimum and maximum score contributions of each scorer are known priori. Thus, a simple rank-preserving algorithm can be developed as follows. For each scorer  $p$ , we compute (offline) the sum of the maximum possible score contributions of succeeding scorers, i.e., we compute  $m_p = \sum_{q=p+1}^N f_q^{\max}$ , where  $f_q^{\max}$  is the maximum contribution of scorer  $q$ . In ranking, a document  $d$  can then be eliminated at scorer  $p$  whenever  $score[d] + m_p < \text{MIN}(H[N])$  holds. Unfortunately, this algorithm does not perform well in the framework we adopt as the difference between the minimum and maximum score contributions is high and scores are not skewed across the scorers (unlike posting lists, discussed in Section 3). Hence, we prefer not to report results for this algorithm.

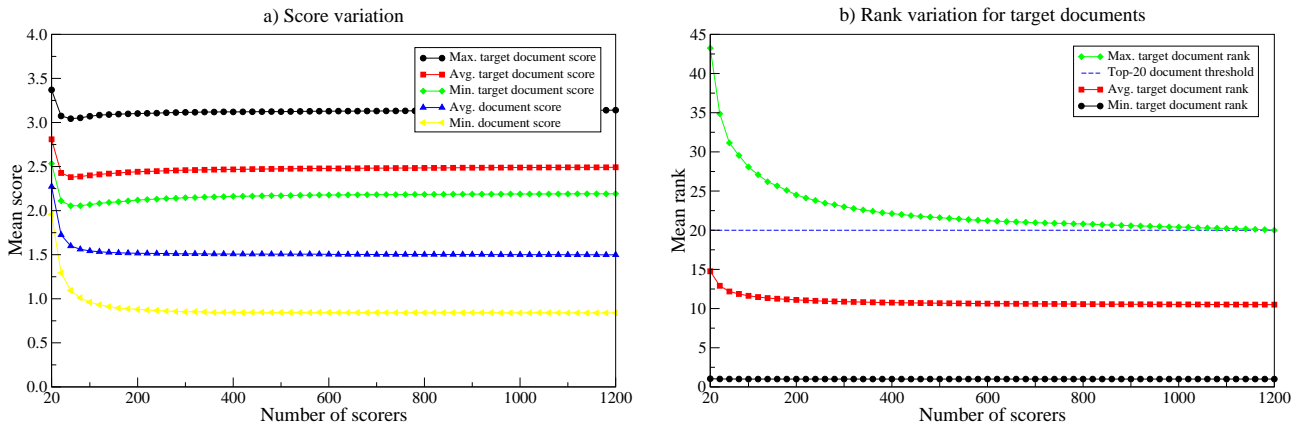


Figure 2: Score and rank variation.

## 5. EXPERIMENTAL FRAMEWORK

We use 7400 web queries, randomly and uniformly sampled from query logs of Yahoo!. All queries are evaluated over the entire web index of the same search engine. For each query, we obtain the highest-ranked 20 documents, i.e., the documents listed in the first two result pages. Since 75.2% of page views are for the first two result pages [19], this is a reasonable number (at the moment, this percentage must be even higher due to continuous improvements in web search relevance). We also obtain 200 additional good documents, sampled from the same web index, using a simple ranking function based on a linear combination of a BM25 variant with a link analysis metric (a very good predictor according to [23]). We then mix the two document sets as an effort towards simulating a scenario where all top documents fall into the same node in a large search cluster. Hence, result qualities provided by our experiments will be stricter bounds for cases where top documents are more evenly distributed.

Documents are evaluated using a learning system based on gradient boosted decision trees [33], composed of  $N = 1200$  scorers (decision trees). The system is trained by hundreds of proprietary features<sup>3</sup>. The number of scorers is chosen depending on the point where the error rate observed over a large training data set saturates. Note that offline pruning of scorers is not feasible as almost all scorers are needed to refine the scores of very highly ranked documents.

Our decision of using gradient boosted decision trees is based on the following. First, they are currently the state-of-the-art in machine learning with very high accuracy rates in regression and classification. Second, they are very well suited to the ranking problem in web search as they can be trained by large training sets. Third, they are fast enough to be used in a time-constrained problem like query processing.

To form a baseline for our experiments, we rank all 220 documents using our system (i.e., full score computation without any early exits) and select the top 20 documents for every query. We refer to these documents as “target documents”. In our experiments, we evaluate the quality of different early exit functions by counting the number of target documents missing in the resulting top 20 documents (effectively, we compute the value  $20 \times (1 - P@20)$ ). In all plots,

<sup>3</sup>Due to their commercial value, we refrain from disclosing the training features. Moreover, the techniques proposed in this work are independent of the feature set used. Interested reader may refer to [21] for a representative set of features.

reported values are averaged over all queries. We should note that, in the experiments conducted, missing target documents are mostly borderline documents near the 20th rank. Therefore, we do not report precision values at low  $k$  values (e.g.,  $P@5$ ) or  $DCG@20$  values as there is no or insignificant amount of relevance loss with respect to these metrics.

Figs. 2a and 2b display variation of average, minimum, and maximum document scores and ranks with increasing number of scorers, respectively. As seen in Fig. 2a, there is high score variation at very early scorers. This is because the first scorer tries to assign an average score to documents as an effort to minimize the estimated error. But, the following scorers quickly differentiate between more relevant and less relevant documents. As more scorers are executed, average, minimum, and maximum scores stabilize very quickly for both target documents and the documents in the entire input set. Similarly, according to Fig. 2b, the average document rank stabilizes very quickly for the target document set. However, the maximum rank is always higher than 20, i.e., there are target documents not within the final top 20 until the latest scorers execute. Note that the plots show only the average behavior over all queries. Both score and rank variation can be quite different for individual queries.

## 6. PERFORMANCE

All of the early exit functions we described require a threshold to be set in an offline manner. In our experiments, we refrain from using highly tuned thresholds. Instead, we are more interested in observing the general behavior of exit functions with varying thresholds. This allows us to understand the impact of threshold selection on the performance. Moreover, the problem of determining the optimum thresholds does not have a trivial solution (but, the thresholds can be tuned empirically). We discuss this issue in Section 7.

### 6.1 Experiments with Single Exit Position

As the first experiment, in order to understand the general performance behavior, we place a single exit function at every possible exit position and observe the performance with different thresholds. In other words, we try all possible (threshold, exit position) combinations via brute force evaluation. Thresholds are empirically selected from a wide range that let us explore the quality-performance trade-off and trends in detail. Selected thresholds are as follows:

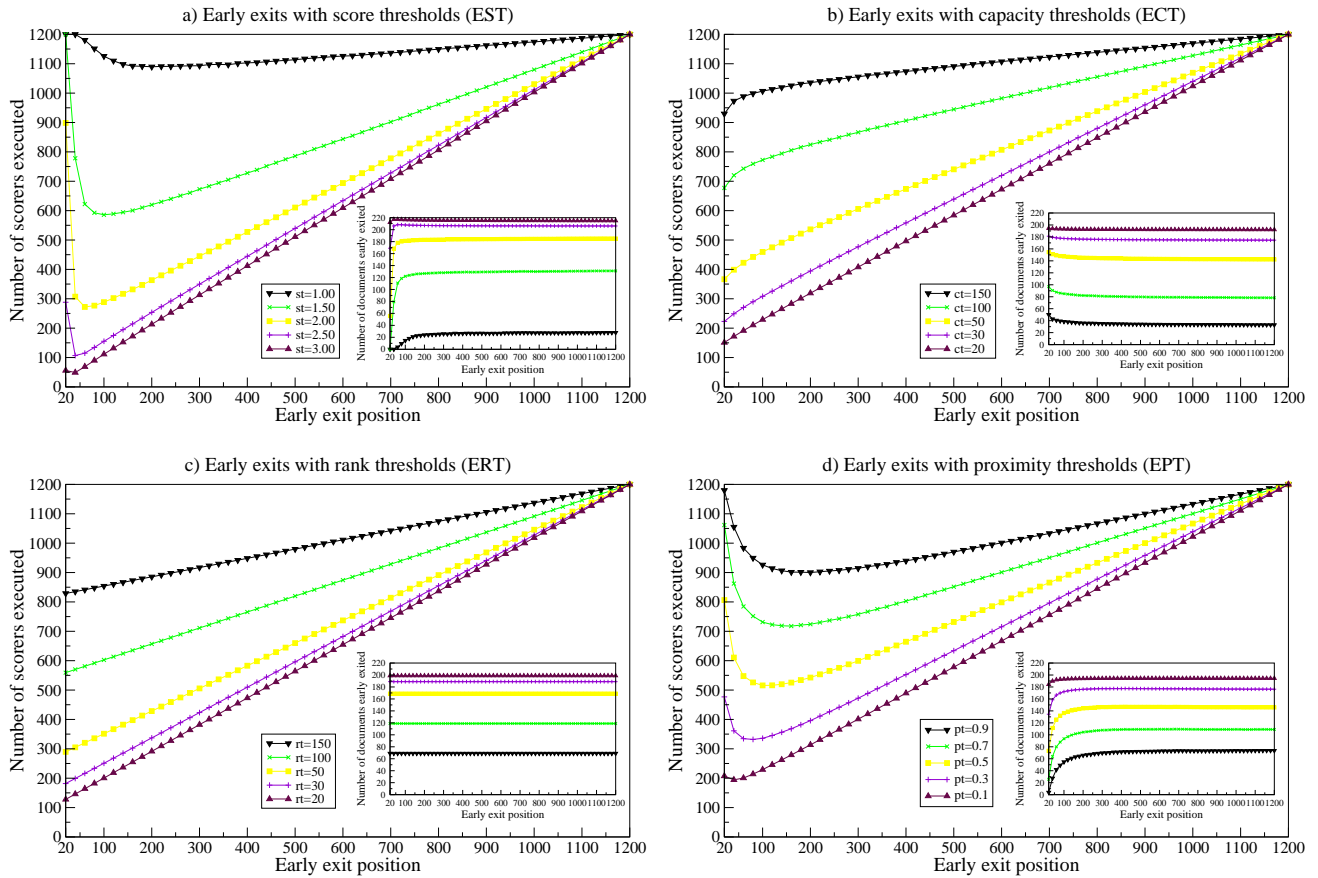


Figure 3: Number of documents early exited (inner plot) and number of scorers executed (outer plot).

- EST:  $st \in \{1.0, 1.5, 2.0, 2.5, 3.0\}$ ,
- ECT:  $ct \in \{150, 100, 50, 30, 20\}$ ,
- ERT:  $rt \in \{150, 100, 50, 30, 20\}$ ,
- EPT:  $pt \in \{0.9, 0.7, 0.5, 0.3, 0.1\}$ .

For each exit position, Fig. 3 (inner plots) shows the number of documents eliminated. Interestingly, the behavior is very similar for EST and EPT as there is a sharp incline at early positions. This can be explained by the initial decline in average scores (Fig. 2a), which render documents more likely to be subject to elimination. A slight initial decline is observed for ECT. For ERT, the number of eliminated documents is fixed as it is independent of the query (exactly  $M - rt[p]$  documents are discarded at each position  $p$ ).

Fig. 3 (outer plots) shows the average number of scorers executed. For example, for EST with  $ct = 1.5$ , if we place an exit function at position 100, then about 600 scorers are executed on average. Data values in these plots are calculated using corresponding values in inner plots: if  $e[p]$  documents are exited at position  $p$ , we can compute the average number of scorers executed ( $N_{\text{avg}}$ ) by the following formula

$$N_{\text{avg}} = \frac{p \times e[p] + N \times (M - e[p])}{M}.$$

In our case, execution costs of individual scorers are comparable, and the overhead of early exit functions is negligible. Hence, the average number of scorers executed provides a good estimate of the total execution time.

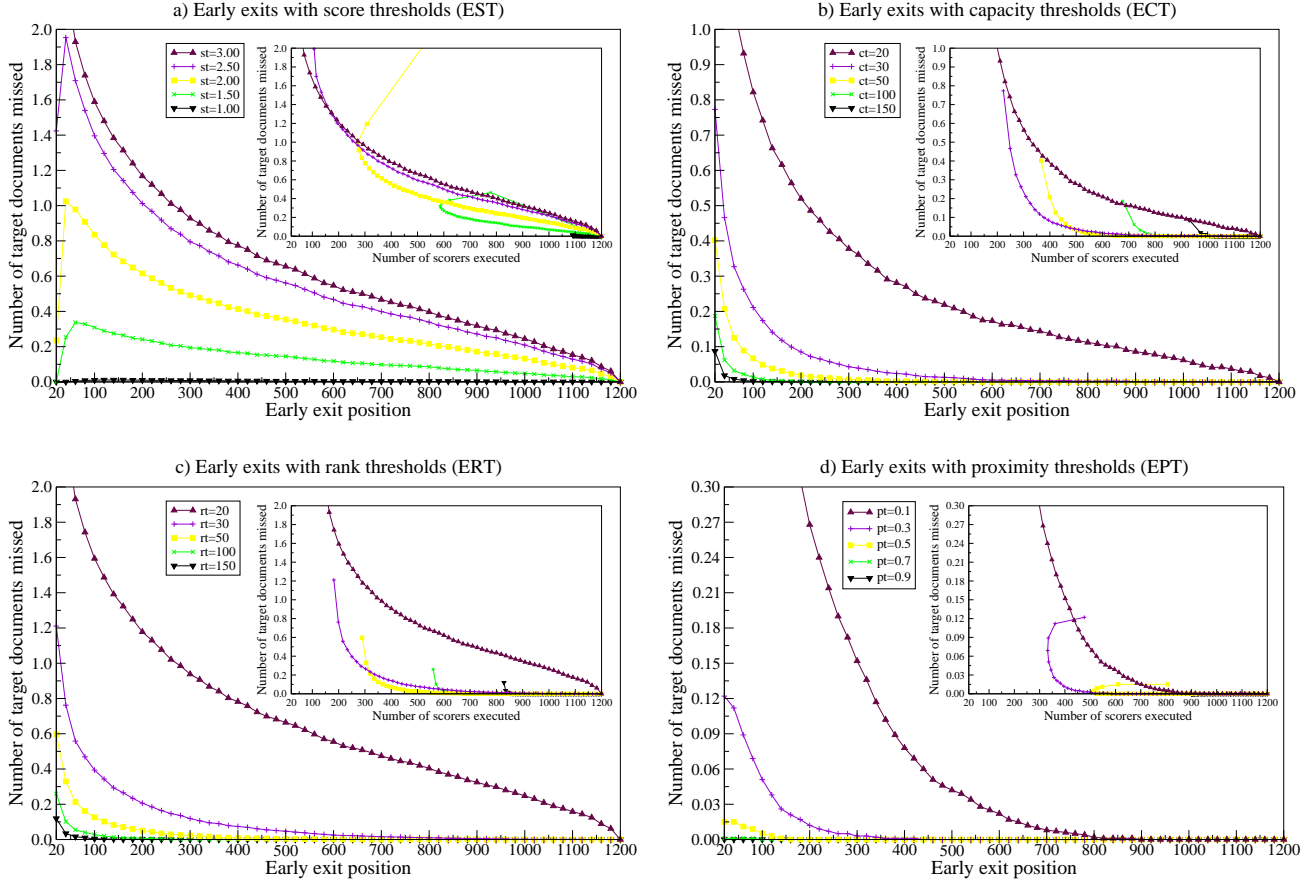
Fig. 4 (outer plots) provides the number of target docu-

ments missing in the final top 20 documents due to early exits. We observe that EPT results in very little loss of target documents relative to other exit functions, among which EST performs the worst. However, the number of missing target documents is not indicative on its own as the number of scorers executed should also be considered. Hence, we also display the trade-off between the two, in Fig. 4 (inner plots), where each data value represents, for every (threshold, exit position) pair tested, the number of scorers executed and the corresponding number of missing target documents. As seen from the figure, if thresholds are set too tight, the number of missed documents can grow quickly with little reduction in the number of scorers executed. If they are set too loose, on the other hand, because many documents will not be eliminated, desired efficiency improvements may not be achieved.

## 6.2 Experiments with Multiple Exit Positions

In practice, multiple functions can be placed at different positions. However, selection of the optimum number of exit positions and positions (also the thresholds) is not a trivial problem. In the mean time, selection of these parameters is a relatively less important problem. In practice, thresholds and exit positions can be tuned by brute force search over some training data. Even if this approach can be computationally expensive, it is still feasible as the tuning is performed infrequently (in fact, only when the ensemble is modified) and in an offline manner.

In this work, we are not interested in tuning parameters. Instead, we observe the effect of exit positions and thresholds



**Figure 4: Number of target documents missed versus scorers executed (inner plot) and number of target documents missed as exit position increases (outer plot).**

on performance by conducting experiments with multiple, heuristically selected exit positions and thresholds. These experiments give us a good idea about the relative performance of our early exit functions.

For selecting the early exit positions, we follow a simple heuristic that models the gap between two exit positions by a polynomial function. For example, for the constant function  $x^0$ , the exit positions are placed at equal distances. As the degree of the polynomial increases, placement becomes more skewed towards the early scorers, i.e., exit functions are more densely placed at earlier positions. The following list shows the placement functions used and the resulting exit positions (we place four early exit functions and positions are rounded to the nearest multiple of 20):

- $x^0 : p_1[1 \dots 4] = \{40, 340, 620, 920\}$ ,
- $x^1 : p_2[1 \dots 4] = \{40, 160, 400, 740\}$ ,
- $x^2 : p_3[1 \dots 4] = \{40, 80, 240, 600\}$ ,
- $x^3 : p_4[1 \dots 4] = \{40, 60, 160, 460\}$ .

Figs. 5a and 5b compare the exit functions in terms of the number of scorers executed and target documents missed for a variety of thresholds and placement functions. Each curve in the figures is composed of four data points, which correspond to the above-mentioned placement functions. Horizontally, first data points in all curves correspond to placements by the polynomial  $x^3$ , the second data points correspond to  $x^2$ , and so on. As an example, the leftmost data

point in Fig. 5a is obtained by running EST with parameters  $p_4$  and  $st$  while the rightmost data point is obtained by running EPT with parameters  $p_1$  and  $pt$ . Two different sets of thresholds (loose and tight) are tried (displayed in the legends of the figures). Tight thresholds favor result quality while loose thresholds are better for improving efficiency.

According to Fig. 5, we observe that EPT, ERT, and ECT all perform considerably better than EST. Among them, EPT performs slightly better than ERT and ECT, with almost no loss in result quality and reducing the number of scorers by about four times. For example, by executing only 300 scorers on average, EPT results in almost no target document loss (Fig. 5a). However, a point that needs to be considered is the caching effect mentioned in Section 4.1. Since ECT employs DOT, in practice, it has the potential to outperform EPT when real execution times are considered.

### 6.3 Distribution of Queries

All results reported so far are averages over all queries. It is also interesting to observe the distribution of queries according to the number of target documents missed. Fig. 6 shows in what percentage of queries the same number of target documents is missed. Since queries for which more than three documents are missed are relatively rare, their percentages are summed and displayed as a single number. The results displayed in the figure are obtained using parameter  $p_3$  and loose thresholds (see Fig. 5a).

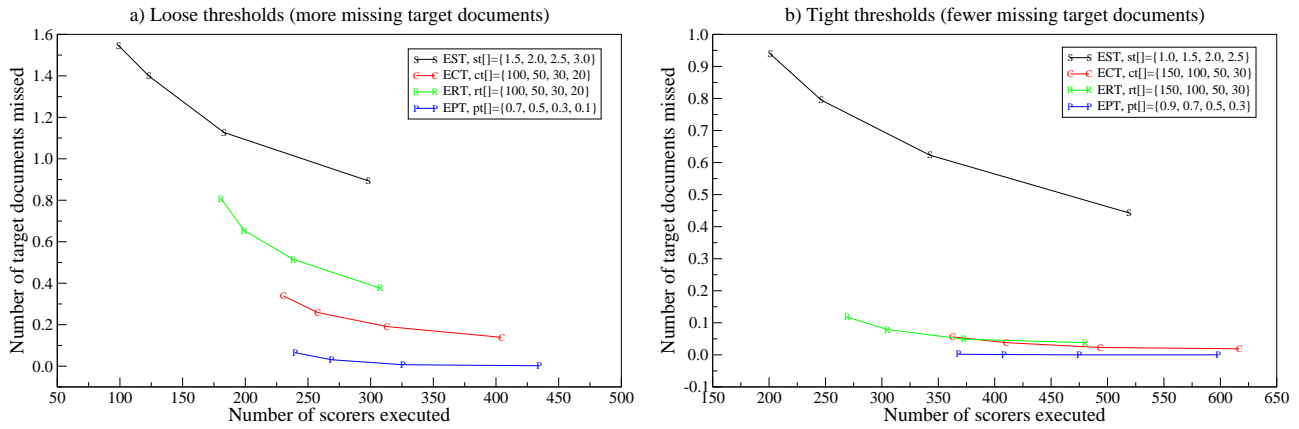


Figure 5: Performance of early exit functions with varying threshold values and placement strategies.

According to Fig. 6, EPT achieves results identical to the case with no early exit optimization for 94% of queries. It misses more than two target documents in only six queries. The poor performance of EST is more clearly visible in this figure. In the worst case of EST, we observe a query with only one correctly selected target document. Although their distributions are relatively similar, ECT outperforms ERT, conforming with Fig. 5a. However, in the worst case, ECT is close to EST with 17 missed target documents.

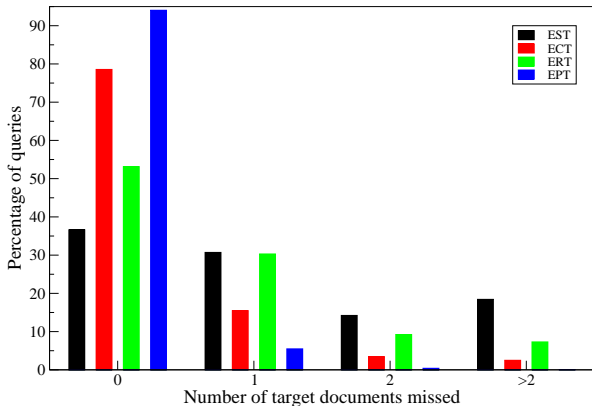


Figure 6: Percent dissection of queries according to the number of missed target documents.

## 7. DISCUSSION

Limitations and over-simplifications of this work needs to be pointed out. First, herein, we assumed that execution costs of scorers are similar. Although the costs of scorers (i.e., basically, the average depths of decision trees) are similar in our case, this may not be true for other additive systems. Hence, costs of scorers need to be taken into account. One simple heuristic could be to reorder the scorers (offline) in decreasing order of importance/cost ratios, rather than solely based on importance. Going further, we could even try to modify the learning algorithm in such a way that it purposely places cheaper scorers at early positions.

Second, the behavior of early exits on a large search cluster could be interesting to investigate. In our work, as an effort towards simulating a worst-case scenario, where all top 20

documents are stored in the same child node in a large search cluster, we mixed two result sets of different qualities. In practice, relevant documents will be distributed on many search nodes, each having only a few relevant documents. This may further improve the performance gains achieved by early exit optimizations.

Third, instead of tuning the exit functions, we tried to demonstrate the general behavior via empirically selected thresholds and exit positions. One possible research direction would be to automate the tuning process by an algorithm that estimates the (threshold, position) pair that achieves the maximum benefit in early exiting.

Fourth, we believe that conditional ensembles can also benefit from early exit optimizations. Further research is needed to understand whether the proposed techniques are directly applicable to such ensembles and how they perform.

Fifth, in this work, we have not considered any constraints on execution time. In practice, however, there may be upper bounds on the processing time of queries, and the bounds may vary depending on the query. It would be interesting to extend these algorithms to encapsulate such constraints.

Sixth, the query set we used in experiments are randomly and uniformly sampled. In a real-life search engine setting, however, most frequently submitted queries are caught by the result cache and can be answered without any need for scoring. Ideally, our query set should have been sampled from queries that cannot be answered by the result cache.

## 8. CONCLUSIONS

We presented and evaluated four different early exit optimization strategies for improving performance of additive machine learning ensembles, which are used by some search engines for ranking purposes. The proposed strategies are found to achieve considerable speedups in online execution times of additive ensembles (with the EPT approach, the improvement is up to four times with almost no loss in quality). These results show that early exit optimizations have the potential to be of great value for large-scale search engines as we have mentioned in Section 1. We plan to continue this research with the issues given in Section 7.

## 9. ADDITIONAL AUTHORS

Additional authors: Jon Degenhardt (Yahoo! Labs, email: jondeg@yahoo-inc.com).

## 10. REFERENCES

- [1] V. N. Anh, O. Kretser, and A. Moffat. Vector-space ranking with effective early termination. In *Proc. 24th Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 35–42, 2001.
- [2] V. N. Anh and A. Moffat. Pruned query evaluation using pre-computed impacts. In *Proc. 29th Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 372–379, 2006.
- [3] R. Baeza-Yates and Ribeiro-Neto. *Modern information retrieval*. Addison-Wesley, 1999.
- [4] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
- [5] E. W. Brown. Fast evaluation of structured queries for information retrieval. In *Proc. 18th Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 30–38, 1995.
- [6] C. Buckley and A. Lewit. Optimizations of inverted vector searches. In *Proc. 8th Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 97–110, 1985.
- [7] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proc. 22nd Int'l Conf. on Machine learning*, 2005.
- [8] B. B. Cambazoglu, V. Plachouras, and R. Baeza-Yates. Quantifying performance and quality gains in distributed web search engines. In *Proc. 32nd Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 411–418, 2009.
- [9] Y. Cao, J. Xu, T. Liu, H. Li, Y. Huang, and H. Hon. Adapting ranking SVM to document retrieval. In *Proc. 29th Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 186–193, 2006.
- [10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms (2nd ed.)*. MIT Press, 2001.
- [11] D. Cossock and T. Zhang. Subset ranking using regression. In *Proc. Conf. on Learning Theory*, 2006.
- [12] D. DeCoste. Anytime interval-valued outputs for kernel machines: fast support vector machine classification via distance geometry. In *19th Int'l Conf. on Machine Learning*, 2002.
- [13] R. Fagin. Combining fuzzy information from multiple systems. *Journal of Computer and System Sciences*, 58(1):83–99, 1999.
- [14] R. Fagin. Combining fuzzy information: an overview. *ACM SIGMOD Record*, 31(2):109–118, 2002.
- [15] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proc. Int'l Conf. on Machine Learning*, pages 148–146, 1996.
- [16] D. Harman and G. Candela. Retrieving records from a gigabyte of text on a minicomputer using statistical ranking. *Journal of the American Society for Information Science*, 41(8):581–589, 1990.
- [17] T. Hastie and R. Tibshirani. *Generalized Additive Models*. Chapman & Hall/CRC, 1990.
- [18] R. Kumar, K. Punera, T. Suel, and S. Vassilvitskii. Top-k aggregation using intersections of ranked inputs. In *Proc. of the Second ACM Int'l Conf. on Web Search and Data Mining*, pages 222–231, 2009.
- [19] R. Lempel and S. Moran. Predictive caching and prefetching of query results in search engines. In *Proc. 12th Int'l World Wide Web Conf.*, pages 19–28, 2003.
- [20] P. Li, C. Burges, and Q. Wu. Mcrank: Learning to rank using multiple classification and gradient boosting. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 21*, pages 897–904. MIT Press, Cambridge, MA, 2008.
- [21] T.-Y. Liu, J. Xu, T. Qin, and W. X. and Hang Li. Letor: benchmark dataset for research on learning to rank for information retrieval. In *SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, 2007.
- [22] A. Moffat and J. Zobel. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems*, 14(4):349–379, 1996.
- [23] M. Najork, H. Zaragoza, and M. J. Taylor. Hits on the web: how does it compare? In *Proc. 30th Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 471–478, 2007.
- [24] M. Persin. Document filtering for fast ranking. In *Proc. 17th Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 339–348, 1994.
- [25] S. Romdhani, P. Torr, B. Schölkopf, and A. Blake. Fast face detection, using a sequential reduced support vector evaluation. In *Proc. Int'l Conf. on Computer Vision*, 2001.
- [26] B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, 2002.
- [27] T. Strohman, H. Turtle, and W. B. Croft. Optimization strategies for complex queries. In *Proc. 28th Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 219–225, 2005.
- [28] M. J. Taylor, H. Zaragoza, N. Craswell, S. Robertson, and C. Burges. Optimisation methods for ranking functions with multiple parameters. In *Proc. 15th Int'l Conf. on Information and Knowledge Management*, pages 585–593, 2006.
- [29] H. Turtle and J. Flood. Query evaluation: strategies and optimizations. *Information Processing & Management*, 31(6):831–850, 1995.
- [30] P. A. Viola and M. J. Jones. Robust real-time face detection. *Int'l Journal of Computer Vision*, 57(2):137–154, 2004.
- [31] W. Y. P. Wong and D. K. Lee. Implementations of partial document ranking using inverted files. *Information Processing & Management*, 29(5):647–669, 1993.
- [32] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *Proc. 30th Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 271–278, 2007.
- [33] Z. Zheng, H. Zha, T. Zhang, O. Chapelle, K. Chen, and G. Sun. A general boosting method and its application to learning ranking functions for web search. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1697–1704. 2008.