Joint Optimization of Cascade Ranking Models

Luke Gallagher
RMIT University
Melbourne, Australia

Ruey-Cheng Chen
Seek Ltd.
Melbourne, Australia

Roi Blanco*
Amazon Research
Barcelona, Spain

J. Shane Culpepper
RMIT University
Melbourne, Australia

ABSTRACT

Reducing excessive costs in feature acquisition and model evaluation has been a long-standing challenge in learning-to-rank systems. A cascaded ranking architecture turns ranking into a pipeline of multiple stages, and has been shown to be a powerful approach to balancing efficiency and effectiveness trade-offs in large-scale search systems. However, learning a cascade model is often complex, and usually performed stagewise independently across the entire ranking pipeline. In this work we show that learning a cascade ranking model in this manner is often suboptimal in terms of both effectiveness and efficiency. We present a new general framework for learning an end-to-end cascade of rankers using backpropagation. We show that stagewise objectives can be chained together and optimized jointly to achieve significantly better trade-offs globally. This novel approach is generalizable to not only differentiable models but also state-of-the-art tree-based algorithms such as LambdaMART and cost-efficient gradient boosted trees, and it opens up new opportunities for exploring additional efficiency-effectiveness trade-offs in large-scale search systems.

ACM Reference Format:
Luke Gallagher, Ruey-Cheng Chen, Roi Blanco, and J. Shane Culpepper. 2019. Joint Optimization of Cascade Ranking Models. In *The Twelfth ACM International Conference on Web Search and Data Mining (WSDM '19), February 11–15, 2019, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3289600.3290986

1 INTRODUCTION

The complexity and breadth of new ranking models being deployed in search engines continue to improve their effectiveness. The most significant recent advances rely on multiple stages of reranking using a combination of simple bag-of-words queries and machine learning. This process is often referred to as Learning-to-Rank (LTR) [4, 7, 21], and if multiple stages are employed, it is referred to as *cascade ranking* [40]. As the complexity of algorithms being deployed continues to grow, so too do the efficiency costs. Understanding the trade-offs between the two competing goals of effectiveness and efficiency as collection sizes increase remains an interesting and important problem in large scale search applications.

Tree boosting algorithms such as LambdaMART [4] and Gradient Boosted Regression Trees (GBRT) [15] are considered state-of-the-art for many search tasks. Given the importance of these algorithms in nearly every major search engine company, several recent studies have focused on how to improve the scalability of these algorithms. Examples include reducing model complexity via tree pruning [11, 25], traversal optimizations [16, 24], new data structures [23], and cascaded ranking [8, 41]. External to the algorithms themselves, approaches to improve efficiency include empirically constraining the depth of the training data [27] and per-query cutoff prediction during candidate generation [10, 29].

The theme of this paper is to expand upon our conceptual understanding of trade-offs in complex cascade ranking architectures, by observing that scalable document scoring pipelines can be made more efficient while still minimizing effectiveness loss. Our approach relies on the fact that earlier stages will see a full candidate list, but the majority of documents will not be relevant, or even scored in later stages, and therefore a trade-off has to be made between extracting cheap (or cost-efficient) features early on, and maximizing effectiveness in the final stages of the retrieval process. Therefore, different optimization criteria should exist *between* individual stages to achieve the best outcomes. This leads to the fundamental question of how to train a model such that each stage is optimized over the subset of data that it is likely to see, while also achieving a global minimization objective, such as finding the best balance between efficiency and effectiveness across the entire search process.

Based on these insights, we identify two key unresolved issues in prior work: (1) Cost-aware approaches for LTR currently exist, but do not generalize to a Cascade LTR (CLTR) architecture; (2) Each stagewise ranker is trained independently to all of the others, when in fact the decisions made by previous rankers directly impact the performance of all subsequent rankings in the cascade. In this regard, the work of Wang et al. [41] and Chen et al. [8] are closest to our own, but neither address these two key limitations. So, we explore both of these problems in this work, through the following research questions:

Research Question (RQ1): *How can loss be minimized in a CLTR architecture such that both stagewise and global loss are optimized jointly?*

*Work performed while the author was at RMIT University.

WSDM '19, February 11–15, 2019, Melbourne, VIC, Australia

ACM ISBN 978-1-4503-5940-5/19/10…$15.00
https://doi.org/10.1145/3289600.3290986

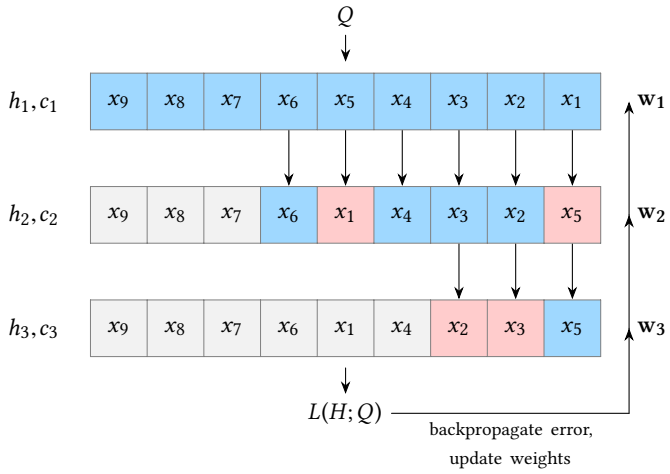

Figure 1: Training phase of a 3-stage cascade. Cutoffs $c_1 = 6, c_2 = 3$ and $c_3 = 0$. The candidate set of documents for a given query Q enters ranker h_1 that extracts features and reranks all documents (9 in this case) and outputs the top c_1 documents. Ranker h_2 receives c_1 documents, performs feature extraction and reranking (swapping x_1 and x_5), and then outputs the top c_2 documents. Ranker h_3 reranks the top c_2 documents (swapping x_2 and x_3) resulting in the final ranking, for which the global cascade loss $L(H; Q)$ is computed. As the final step, the error is backpropagated through the cascade network to update stagewise weights for each ranker.

Research Question (RQ2): *In a CLTR architecture, what is the best way to perform cost-aware optimizations directly in a tree-based learning algorithm during model construction?*

Our Contributions. This paper has three major contributions:

(1) We propose a globally optimized cascade architecture and introduce a novel method to jointly learn ranking models across multiple stages in a cascade using backpropagation. This notifies upstream rankers of the global error observed throughout the entire ranking pipeline. In order to achieve this, we introduce a smoothed approximation of a "gate-keeper" mechanism to model the cascade globally.

(2) Building on insights from a new cost-aware algorithm designed specifically for gradient boosting [33], we develop new methods for learning cascade models that do not require a retraining step, a shortcoming of prior work on cascaded ranking.

(3) We show empirically that jointly learning cascade ranking models leads to significant improvements on effectiveness and efficiency trade-offs. Our approach offers new opportunities to balance trade-offs in complex CLTR architectures.

2 BACKGROUND AND RELATED WORK

Learning-to-Rank (LTR) in Information Retrieval is now a mature problem, with many well-known techniques [21]. While there are now many approaches that improve effectiveness in top-k web search (Matveeva et al. [30] being one of the earliest for cascade ranking, and Lucchese et al. [26] being one of the the most recent for LTR), the issues around scalability and efficiency in machine learning algorithms for multi-stage retrieval systems have only recently become a point of emphasis. Among the earliest work drawing attention to this problem was that of Cambazoglu et al. [5]. Other early work soon followed that explored performance trade-offs in single stage reranking by focusing primarily on training and feature extraction costs [1, 2, 27, 28].

A related line of research has focused on *budget-aware* learning [7, 18, 34, 43, 44]. The key intuition being that feature costs should be directly integrated into the learning process. It is often easy to confuse these two lines of research as they are clearly related. But there are subtle distinctions between the two approaches. In this work, we apply the lessons learned from multi-stage retrieval directly to create new approaches to cascaded, budget-aware learning.

Single Stage Cost Reduction. Several recent improvements reduce prediction costs in a "single-stage" architecture, which is the most commonly used configuration in the research literature. DART incorporates the *dropout* technique that is well known for regularizing the learning phase of a neural network [36]. During each iteration, a subset of neurons within the network are deactivated in the current iteration, and the network continues as if these neurons and associated edges do not exist. This has the effect of helping the network to generalize, and not overfit. The design of DART is optimized specifically for learning an ensemble of boosted regression trees. A more recent algorithm X-DART [25] extends the capabilities of DART, and integrates the tree pruning capabilities of CLEAVER [22], which is a post pruning technique that eliminates redundant trees in an existing model.

Another recent algorithm by Peter et al. [33] performs cost-efficient gradient boosting (CEGB), which is a generalization of the Greedy Miser algorithm [44]. CEGB, like Greedy Miser, incorporates both tree evaluation and feature extraction cost as a penalty factor when learning models that include an efficiency component in the objective function. CEGB grows trees in best-first order, allowing trees to expand in any leaf node so that a simultaneous comparison of different leaves/features can be made when deciding the next split. In this work we extend this key idea from CEGB, and exploit it in our CLTR framework to *directly* learn a cascade of tree models.

Quality Versus Efficiency. A number of metrics designed specifically for analyzing the efficiency and effectiveness of systems within the context of an LTR framework exist. Wang et al. [40] proposed MEET – the mean efficiency and effectiveness trade-off metric. MEET can be customized for specific application needs by using a variety of different efficiency and effectiveness measures. Similarly, Capannini et al. [6] proposed a measure called the *area under the quality cost curve* (AuQC) which measures the effectiveness of an LTR algorithm given a time budget. In this work, our cost model depends primarily on feature extraction, and we therefore chose to measure the average cost per document since feature extraction in our current approach is performed at the document level. Further reasoning behind this decision is discussed in Section 4.

Cascade Learning. Prior research also exists on cascaded machine learning for IR and other closely related domains such as medical science, computer vision, and e-commerce [14, 20, 41–43]. These

approaches generally assign different ranking models to each ranking stage in order to collectively achieve a desired trade-off. The earliest work on efficient cascade architectures designed specifically for IR ranking tasks was from Wang et al. [41]. By generalizing the AdaRank algorithm, the authors incorporate a cost model to construct a cascade where each weak learner becomes a single stage within the cascade. Other closely related approaches that build on the cascade paradigm include the work of Xu et al. [43] and Chen et al. [8]. Xu et al. proposed a GBDT-based cascading model that reweights individual trees in a model using a pre-defined cost objective. Chen et al. [8] use weighted $\ell 1$ regularization and a linear model to incorporate feature extraction cost into the process of feature learning, and then optimize individual stages in the cascade using tree-based models derived from the features selected by the linear model. All of the prior approaches apply cost-aware learning and model optimizations stagewise but not globally. Our new approach concurrently optimizes local stage objectives and the global loss, while still accounting for execution costs.

3 APPROACH

A cascade ranking model is a sequence of machine learned rankers chained together to collectively process a pipeline of documents. Each ranking stage assigns scores to all documents entering the stage, and then promotes the highest ranked documents to the next stage. Different sets of features can be used across stages; for instance, early stages may use more features that are not expensive to compute (in time or other resources), whereas later stages may use all of the features available. A cascade filters out documents in the early stages, and the most expensive models are only applied to a handful of documents, making it a cost-efficient strategy for real-time ranking.

Cascade Structures and Ranking. Assume that every document is represented as a d-dimensional vector $x \in \mathbb{R}^d$, and let each stage model $h_j : \mathbb{R}^d \rightarrow \mathbb{R}$ be a function that receives a particular "view" of a document as input, represented as a subset of features whose indexes are $F_j \subset \{1, 2, \ldots, d\}$. Then, a K-stage cascade is represented as follows: $\langle (h_1, c_1), (h_2, c_2), \ldots, (h_K, c_K) \rangle$, with $c_1 > \ldots > c_K = 0$ being the respective rank position cutoffs. To implement early exiting in the cascade, only the c_j top-scoring documents in stage j advance to stage $j+1$ (if any). Note that c_K is set to 0 as no document can practically go beyond the final stage.

In cascade ranking, some documents will receive more than one score estimate as they pass through multiple stages. Generally, when document x enters a new stage j, a new score estimate $h_j(x)$ will be generated. When all documents in stage j have received new estimates, a cutoff threshold κ_j is computed based on the c_j-th top-scoring document:

$$\kappa_j = \langle h_j(x) : h_1(x) \geq \kappa_1, \ldots, h_{j-1}(x) \geq \kappa_{j-1} \rangle_{[c_j]}. \tag{1}$$

Note that $\langle a \rangle_{[c]}$ indicates the c-th element in sequence $\langle a \rangle$ in decreasing order (and $\langle a \rangle_{[0]} = \infty$). Documents with lower scores than κ_j are not promoted to the next stage.

Given the stagewise scores and cutoff thresholds, document x is said to be *covered* by stage j if it enters stage j but fails to advance to stage $j+1$. This property can be represented as an indicator function $\mathcal{I}[x \in X_j]$, defined as follows:

$$\mathcal{I}[x \in X_j] := \mathcal{I}\left[h_j(x) < \kappa_j\right] \prod_{k=1}^{j-1} \mathcal{I}\left[h_k(x) \geq \kappa_k\right]. \tag{2}$$

To compute the final score $H(x)$, the cascade will make use of all score estimates it has generated for document x. In this paper we explore the following three types of cascade structures, with detailed formulations given in Table 1:

- *Independent-Chaining Cascade (ICC)*, where the final score $H(x)$ is the last score estimate that document x receives. This design assumes that the stage covering document x provides the most accurate relevance estimate.
- *Full-Chaining Cascade (FCC)*, where the final score $H(x)$ is the summation of all score estimates that document x has received, based on the assumption that combining all estimates leads to a more accurate estimation of document relevance.
- *Weak-Chaining Cascade (WCC)*, where the final score $H(x)$ is the maximum from all of the score estimates that document x has received. In this cascade structure, the stagewise models focus more on optimizing a sparse, opportunistic set of documents.

Risk Minimization. We use the principle of *empirical risk minimization* [38] to learn a cascade ranking model. The empirical risk is defined as:

$$R(H) = \sum_{Q} L(H; Q) + \gamma C(H),$$

where $Q = \langle (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \rangle$ indicates a query as a list of (x, y) pairs, with x being the document and y being the relevance label. The ranking loss of $L(H; Q)$ and the feature extraction cost $C(H)$ are combined with the hyperparameter γ. Similar loss functions have been used successfully in previous work [8, 37, 41, 43].

The key idea we pursue here is to use *backpropagation* in order to learn a combined ranking function across multiple stages of a cascade simultaneously. This technique is now widely used when building neural networking models, where it essentially redistributes downstream errors back to upstream ranking models by taking dependencies and relative contributions of individual components into account.

To apply backpropagation in a cascade setting, we must first derive the partial derivative of $R(H)$ with respect to stage model configuration w_j:[1]

$$\sum_{Q} \sum_{x \in D(Q)} \frac{\partial [L(H; Q) + \gamma C(H)]}{\partial H(x)} \frac{\partial H(x)}{\partial h_j(x)} \frac{\partial h_j(x)}{\partial w_j}. \tag{3}$$

In this equation, $D(Q)$ denotes the set of documents in Q. Observe that the ranking loss $L(H; Q)$ is dependent on all document scores and in turn linked to stagewise scores. These document scores are not inter-dependent as κ values are computed on-the-fly based on rank positions, rather than being a variable in the model. This can be contrasted with Wang et al. [41] where score thresholds are

[1] One further assumption is made here – the internal configuration w_j of the stage j model h_j can be updated in a model-specific way using an iterative approach, such as Gradient Descent for linear models, or Gradient Boosting for decision trees.

Table 1: An overview to the three proposed cascade structures

Cascade	Score Function $H(x)$	Partial Derivative $G_j(x) = \partial H(x)/\partial h_j(x)$
ICC	$\sum_{j=1}^{K} \mathcal{I}[x \in X_j]\, h_j(x)$	$\frac{\partial I_j(x)}{\partial h_j(x)} \left(\sum_{j'=j}^{K} \frac{\mathcal{I}[x \in X_{j'}]}{I_j(x) - \mathcal{I}[j'=j]}\, h_{j'}(x) \right) + \mathcal{I}[x \in X_j]$
FCC	$\sum_{j=1}^{K} \mathcal{I}[x \in X_j] \sum_{l=1}^{j} h_l(x)$	$\frac{\partial I_j(x)}{\partial h_j(x)} \left(\sum_{j'=j}^{K} \frac{\mathcal{I}[x \in X_{j'}]}{I_j(x) - \mathcal{I}[j'=j]} \sum_{l=1}^{j'} h_l(x) \right) + \sum_{j'=j}^{K} \mathcal{I}[x \in X_{j'}]$
WCC	$\sum_{j=1}^{K} \mathcal{I}[x \in X_j] \max_{1 \le l \le j} h_l(x)$	$\frac{\partial I_j(x)}{\partial h_j(x)} \left(\sum_{j'=j}^{K} \frac{\mathcal{I}[x \in X_{j'}]}{I_j(x) - \mathcal{I}[j'=j]} \max_{1 \le l \le j'} h_l(x) \right) + \sum_{j'=j}^{K} \mathcal{I}[x \in X_{j'}]\, \mathcal{I}[h_j(x) = \max_{1 \le l \le j'} h_l(x)]$

detached from rank cutoffs, which in turn may not reliably select the correct number of documents entering individual stages in the cascade.

Also observe that some elements within the inner summation already have known formulations. The first component $\partial[L(H;Q) + \gamma C(H)]/\partial H(x)$ is the derivative of the empirical risk with respect to the final score, which is dependent on the type of loss function used (e.g. pointwise, pairwise, or listwise). Analogously, the third component $\partial h_j(x)/\partial w_j$, which is the derivative of the stage j score with respect to its model configuration, is also model specific.

The key addition required to properly induce a cascade ranking model is the second component, $\partial H(x)/\partial h_j(x)$, which is the partial derivative of the cascade score with respect to the stage j score. For each cascade structure, the respective partial derivative $G_j(x) = \partial H(x)/\partial h_j(x)$ is shown in Table 1. Putting all of this together, we have the following descent update in iteration t with respect to stage j in the model:

$$w_j^{(t+1)} = w_j^{(t)} - \eta^{(t)} \sum_{Q} \sum_{x} G_j(x)\, \frac{\partial[L(H;Q) + \gamma C(H)]}{\partial H(x)}\, \frac{\partial h_j(x)}{\partial w_j}, \tag{4}$$

Note that, when evaluating $\partial H(x)/\partial h_j(x)$, one caveat is that when $x = h_j^{-1}(\kappa_j)$, the partial derivative is unbounded. This is due to the discontinuity in the resulting derivative, caused by the Dirac's delta function $\delta[\cdot]$ in the following equation:

$$\frac{\partial}{\partial h_j(x)} \mathcal{I}\left[h_j(x) \ge \kappa_j\right] = \delta\left[h_j(x) - \kappa_j\right]. \tag{5}$$

Removing the discontinuity can mitigate this issue, i.e. zeroing out summations in Table 1 that involve (5). An alternative approach which will be introduced next is to use a smoothed approximation.

Smooth Approximation for Indicator Functions. The stage-wise update (4) suggests that, in each iteration, documents are reassigned to the training sets of individual stage models according to the value $G_j(x)$. But this reassignment is "hard" as $G_j(x)$ involves $\mathcal{I}[x \in X_j]$, and every document belongs to just one subset. As a result, training instances are not shared across stages, and every stage will see fewer documents than it should, which may lead to under-fitting.

One possible solution is to soften the document assignment by approximating the indicator $\mathcal{I}[x \in X_j]$ with a smoothing function. For simplicity, we define $I_j(x) = \mathcal{I}[h_j(x) \ge \kappa_j]$ and rewrite $\mathcal{I}[x \in X_j]$ as:

$$\mathcal{I}[x \in X_j] := \left(\prod_{k=1}^{j-1} I_k(x) \right) \left(1 - I_j(x)\right).$$

The indicator $I_j(x)$ is essentially a step function with respect to $h(x)$, and the approximation has a similar behavior without a sudden leap from one extreme to another. This can be seen as assigning a membership value to document x, indicating its strength during gradient updates.

In this paper, we advocate approximating $I_j(x)$ with non-linear activation functions widely used for neural networks, such as the logistic function [19] and the ReLU function [32]:

$$I_j^{(\text{logistic})}(x) = \left(1 + \exp\left(-\frac{h_j(x) - \kappa_j}{\sigma}\right)\right)^{-1}, \tag{6}$$

$$I_j^{(\text{relu})}(x) = \frac{1}{2}\left(1 + \min\left\{1, \max\left\{-1, \frac{h_j(x) - \kappa_j}{\delta}\right\}\right\}\right), \tag{7}$$

where σ is a scale parameter and δ controls how far off the "ramp" is with respect to $h_j(x) = \kappa_j$. Note that, in (7), the ReLU function is two-sided and its range is normalized between 0 and 1. Making either approximation, the partial derivative $\partial H(x)/\partial h_j(x)$ would no longer be a constant 0 or 1 but some value determined by how far off the score $h_j(x)$ is w.r.t. κ_j. Written as $\tilde{G}_j(x)$, this approximated partial derivative will replace $G_j(x)$ in (4), allowing the update to incorporate the loss generated across all documents, with each document-wise loss discounted differently according to the approximated indicator value.

The main result (4) directly applies to models where all components in the gradient (or the subgradient) can be explicitly constructed. This includes differentiable models such as logistic regression [37], and even ones with non-differentiable losses (e.g. Hinge loss [8]). We omit these derivations due to space limitations.

Tree Cascades. Some further approximations are needed to apply our result on tree-based models. In gradient boosted trees [9, 17, 33], a descent update for stage model T_j (represented by a set of regression trees) is commonly written as:

$$\begin{aligned} F^{(t)} &= \arg\min_{F} \sum_{Q} \left(L(T_j^{(t)} + F; Q) + \gamma C(T_j^{(t)} + F) \right). \\ T_j^{(t+1)} &= T_j^{(t)} + \eta^{(t)} F^{(t)}, \end{aligned} \tag{8}$$

Following Burges [4] and prior work [9, 33], we first derive the Newton step for the first component w.r.t. the node regions $R_1, \ldots, R_n$, while leaving the second component intact. Here, the instance weight $\tilde{G}_j(x)$ is taken into account in the computation of gradients and the Hessian:

$$F^{(t)}(x) = \sum_{i} \left(\sum_{Q} \frac{\sum_{x' \in R_i} \tilde{G}_j(x)\, g_{Q,x'}}{\sum_{x' \in R_i} \tilde{G}_j(x)\, h_{Q,x'}} \right) \mathcal{I}(x \in R_i) \tag{9}$$

Algorithm 1 Gradient boosting with a CEGB model [33].

1: **for** $j = 1, 2, \ldots, K$ **do**
2: Make predictions using $T_j^{(t)}$ to compute $\tilde{G}_j(\cdot)$.
3: Perform boosting update (8) and (9) to obtain $T_j^{(t+1)}$.
4: Inform stages $j+1, \ldots, K$ about new features in $T_j^{(t+1)}$.
5: **end for**

where R_i denotes the region covered by the i-th leaf node, $g_{Q,x} = \partial L(T_j^{(t)}; Q)/\partial T_j^{(t)}(x)$ is the gradient, and therefore $h_{Q,x} = \partial^2 L(T_j^{(t)}; Q)/\partial T_j^{(t)}(x)^2$ is the Hessian of the first component. This formulation is compatible with pointwise and pairwise loss functions, including LambdaRank [4]. Note that when spawning a new tree (i.e. deciding $R_1, \ldots, R_n$), the gradients, Hessian, and the cost of introducing new features, $C(T_j^{(t)} + F)$ - $C(T_j^{(t)})$, are all taken into account, as indicated by Eq (12) in Peter et al. [33].

A revised stagewise gradient boosting procedure implementing the final treatment is given in Algorithm 1. This procedure is used in our experiments to optimize a set of cost-efficient gradient boosting (CEGB) models [33]. When implementing the procedure one needs to be aware of cost model updates hidden by the tree construction process. In CEGB, the later stages must be informed about any change in feature extraction cost in order to properly function. Chaining together the stagewise cost models is a crucial implementation detail for tree-based cascades.

4 EXPERIMENTS

We now examine the efficiency and effectiveness of the CLTR architecture described in Section 3. We first detail the datasets and evaluation methods for measuring both efficiency and effectiveness, and then provide the details of the approach taken to assign feature extraction costs for the datasets. An overview of the learning algorithms employed follows next, along with configuration information for the baseline systems, and then we describe our new cascade configurations. The remainder of the discussion centers on the exploration of the CLTR architecture, and its benefits and limitations relative to the baseline systems.

4.1 Experimental Context

Datasets. The Yahoo! LTR and MSLR-WEB10K collections are used to conduct our reranking experiments [7, 35]. These datasets are designed specifically for LTR tasks and a summary of these collections is presented in Table 2. Set 1 of the Yahoo! LTR dataset is used since it is the only official test collection which provides feature extraction cost information [7]. However, the average retrieval depth of 23.73 (across train, valid and test sets) does not reflect a cascade configured for a production environment. The MSLR-WEB10K collection has an average query depth that is much deeper than Yahoo! S1, allowing for a more realistic simulation of cascade learning, and arguably closer to a more realistic multi-stage reranking environment.

Evaluation. To measure effectiveness we compute the early precision metrics ERR and NDCG to depths 1, 3 and 5 using the `gdeval` tool[2]. RBP is computed using `rbp_eval`[3] with a persistence $p = 0.5$ also targeting early precision. These metrics and depths were chosen due to the nature of the datasets used, and the assumed operational scenario our research system might be deployed in – large scale search tasks where only a few of the highest ranking results are required. All significance testing is done using a paired t-test with Bonferroni correction. Note that RBP was configured to use graded relevance, and the residuals are omitted since all documents in the two collections are judged.

Table 2: Overview of the three datasets used for conducting the experiments. The rightmost column shows the average query depth for each collection.

	Queries	Total Docs	Features	Avg. Depth
Yahoo! S1	6,983	165,660	519	23.73
MSLR-WEB10K	10,000	6,000,960	136	120.02

Feature efficiency is computed as the average cost per document throughout the entire pipeline. For single stage systems this is the sum of the extraction costs of all features used in the model. For cascade systems the feature extraction cost depends on the number of documents entering each stage, and the features used in that stage. Henceforth, the efficiency of a (cascaded) system is calculated as:

$$C = \frac{1}{N} \sum_{j=1}^{K} \left[\sum_{f \in F_j \setminus (F_1 \cup \ldots \cup F_{j-1})} \text{cost}(f) \right] n_j, \tag{10}$$

where N is the total number of documents that enter the system, $\text{cost}(\cdot)$ is the cost for extracting feature f from feature set F_j in cascade stage j, and n_j is the true number of documents scored in cascade stage j (note the distinction from cutoff c_j). The calculation of cost follows the assumption that once a feature is extracted, any later stage that uses the same feature is able to do so with no additional cost.

Cost Model. We follow Chen et al. [8] in that we do not account for the cost of prediction execution for the learned models. Only the feature extraction cost is included as the prediction cost is negligible. If required by future approaches, it would be relatively straightforward to include this cost using an efficient GBRT implementation [23, 24], which have per document prediction costs on the order of microseconds.

Cost Labels. Labeled feature costs are supplied as part of the Yahoo! Set1 dataset. The cost labels for the 519 features are: {1, 5, 10, 20, 50, 100, 150, 200}. The MSLR-WEB10K does not provide feature extraction cost information, although each of the 136 features is documented[4]. Using this information and our own domain expertise, we assign reasonable cost labels[5] to the MSLR-WEB10K features by reusing the same set of cost labels from Yahoo! S1. The goal of this process is to have feature extraction costs assigned in a relative manner. Full details of the cost labels can be found in the

[2] github.com/trec-web/trec-web-2013
[3] people.eng.unimelb.edu.au/ammoffat/rbp_eval-0.2.tar.gz
[4] microsoft.com/en-us/research/project/mslr
[5] github.com/rmit-ir/joint-cascade-ranking

Table 3: Descriptions of the experimental and baseline systems. The new systems ICC, FCC and WCC correspond to the scoring functions in Table 1. LambdaRank loss is used across all tree-boosted systems for tree construction.

System	Description
LGBM-BL	Ke et al. – Cost-insensitive single stage baseline [17].
CEGB-BL	Peter et al. – Cost-aware single stage baseline [33].
LMartC3-BL	Chen et al. – Cost-aware cascade with independent stagewise optimization [8].
BM25F-SD	Chapelle and Chang – Yahoo! S1 ranking of test data by feature 637 [7].
BM25	Qin and Liu – MSLR-WEB10K ranking of test data by feature 110 [35].
ICC	Independent-chaining cascade with joint stagewise optimization.
FCC	Full-chaining cascade with joint stagewise optimization.
WCC	Weak-chaining cascade with joint stagewise optimization.

GitHub project repository along with all of the necessary source code to reproduce the results shown in this paper, and can be used for future experiments on this problem.

Baseline Configuration. The top half of Table 3 describes the baseline systems used for our experiments. LGBM-BL is a cost-insensitive single stage model, and provides an upper performance bound for a system that optimizes purely for effectiveness. Model hyperparameters (trees, leaves, learning rate) were selected using the provided validation set for Yahoo! S1 and via 5-fold cross-validation for MSLR-WEB10K. Common parameters unless otherwise noted for trees and learning rate selection are 2000 and $\{0.05, 0.1\}$ respectively. Subsampling was set to 0.5, as this is a common choice for the test collections used [6]. Early stopping was used to help reduce model complexity and over-fitting. LGBM-BL was configured to use 31 leaves.

CEGB-BL is a cost-aware single stage system, where the balance between efficiency and effectiveness is controlled using a hyper-parameter λ. Our default baseline configuration of CEGB-BL uses 15 leaves as this was the recommendation by the authors for these datasets in the original paper. The trade-off parameter for CEGB-BL was set to $\lambda = 10^{-6}$ [33][6]. For convenience, we also show results when CEGB-BL uses 31 leaves, which was found to be the best configuration for LGBM-BL. In general this custom configuration is more effective, but also much less efficient, resulting in costs that are generally quite close to the full LGBM-BL models in our experiments.

LMartC3-BL is a 3 stage cost-aware cascade, and represents a state-of-the-art cascade system. We configure LMartC3-BL as was described by Chen et al. [8]. Note that we do not include results from Wang et al. [40] as their system is never better than LMartC3-BL. We verified this claim using the reference implementations provided by Chen et al. [8].

Cascade Configuration. The remaining entries in Table 3 describe our cascade systems. Three modes for document scoring are detailed in Section 3: *independent*, *full* and *weak* corresponding to systems ICC, FCC and WCC respectively. We configure the cascade models using the same parameter choices as the baseline configurations, with the following notable differences that aim to articulate the intuitive benefits of a cascade system.

The first stage is weighted to focus on efficiency by using $\lambda = 10^{-5}$, and the number of leaves used in each stage is 15, except the final stage which has 31. All reported cascades use the logistic activation function with σ tuned over values $\{0.1, 0.2, \ldots, 0.5\}$. In the interest of space, we omit results using the ReLU activation function which were less effective.

Cascade hyperparameters such as the number of reranking stages, documents promoted per stage, and the expressive power of stagewise ensembles are important details. Simplifying assumptions were made on a per dataset basis to fix the number of cascade stages and per stage cutoffs. For Yahoo! S1 3 stages with cutoffs $\langle 10, 5, \perp \rangle$ were used, and for MSLR-WEB10K 4 stages with cutoffs $\langle 40, 20, 10, \perp \rangle$ were used, where $\perp$ signifies the absence of a cutoff. In practice the final stage only scores the documents that it "sees" to attain the final ranking. A per-query cutoff selection mechanism similar to the one described by Mackenzie et al. [29], Mohammad et al. [31], could yield a CLTR that is better able to optimize efficiency and effectiveness during feature extraction. We leave this exploration as future work.

4.2 Collection Comparisons

Yahoo! LTR Experiments. The Yahoo! S1 dataset contains a large number of shallow queries which somewhat limit its ability to fully benefit from a cascading system. However, it is currently the only LTR dataset (that we are aware of) to provide feature extraction costs, making it a valuable resource for our experiments. Results for Yahoo! S1 can be found in Table 4. Included for reference is a ranking of the test data by BM25F-SD (feature 637) [3, 7]. Unsurprisingly, the cost-insensitive baseline LGBM-BL is the overall winner in terms of effectiveness with CEGB-BL (31) a close second, but the total cost of the model is still quite high. CEGB-BL offers a cost reduction of around 70% compared to LGBM-BL, exhibits a good trade-off between efficiency and effectiveness, and is significantly more effective than LMartC3-BL, while using the fewest number of features out of the complex ranker baselines.

The new cascade models ICC and FCC significantly improve effectiveness for all metrics except NDCG@1 when compared against the cost competitive baseline CEGB-BL, with FCC having a higher final cost. This partially supports RQ1, and indicates that locally and globally optimizing document rankings in the cascade leads to higher early precision effectiveness. LMartC3-BL is significantly less effective across all metrics and is even marginally more expensive in some cases. Among the new models, ICC is the most effective cascade system overall, and also has a slightly lower cost than CEGB-BL. FCC has a similar effectiveness profile, but is more expensive.

[6] Also based on recommendations from the authors

Table 4: Yahoo! S1 (519 features). Entries marked △, ▲ correspond to statistical significance using a paired t-test with Bonferroni correction at 95% and 99% confidence intervals respectively. Comparisons are relative to CEGB-BL.

	RBP	ERR@k			NDCG@k						
System	$p = 0.5$	@1	@3	@5	@1	@3	@5	# Stages	# Feat. Stage	# Feat. Total	Cost
Ground Truth Models											
LGBM-BL	0.472▲	0.362▲	0.440▲	0.461▲	0.721▲	0.722▲	0.742▲	1	501	501	16,517
CEGB-BL (31)	0.473▲	0.360▲	0.438▲	0.459▲	0.715▲	0.716▲	0.736▲	1	459	459	14,445
CEGB-BL	0.466	0.356	0.433	0.454	0.703	0.701	0.721	1	187	187	4,793
LMartC3-BL	0.448▼	0.336▼	0.413▼	0.435▼	0.663▼	0.664▼	0.685▼	3	162/274/356	356	4,951
BM25F-SD	0.432▼	0.317▼	0.399▼	0.422▼	0.629▼	0.641▼	0.667▼	1	1	1	150
Experimental Cascade Models											
ICC	0.469▲	0.360▲	0.437▲	0.458▲	0.711	0.710▲	0.728▲	3	153/246/405	416	4,751
FCC	0.470▲	0.360▲	0.436▲	0.457▲	0.710	0.709▲	0.728▲	3	124/353/412	435	5,476
WCC	0.468▲	0.357	0.434△	0.455△	0.706	0.706▲	0.723	3	152/218/385	395	3,961

WCC has the worst performance of the new models, but it is also the most efficient. Overall, backpropogation and joint learning appear to provide greater control over the trade-offs in cost-aware tree-based models, making it a plausible solution for RQ2.

MSLR-WEB10K Experiments. The MSLR-WEB10K collection contains 10,000 queries and 136 features that are publicly documented from a retired version of the Bing search engine. Results are shown in Table 5. BM25 is used as a lower bound reference (feature 110) [35]. Again, LGBM-BL is the most effective system with the highest cost as expected. The efficiency gains for CEGB-BL are not as pronounced when compared to those seen on Yahoo! S1 (possibly a result of fewer features and pseudo cost labels). For ERR the new cascade systems ICC and FCC significantly improve effectiveness when compared to CEGB-BL, while reducing efficiency costs around 32% and 47% respectively. NDCG also exhibits some positive gains, albeit less consistently.

The MSLR-WEB10K dataset reveals a very different trade-off profile than Yahoo! S1 for the systems being compared. The deeper document depth provides cascaded systems additional opportunities to discover efficiency improvements that do not degrade effectiveness. This suggests that even more favourable trade-offs could be uncovered by including additional documents and features in the cascade.

Detailed Trade-off Comparisons. While the aggregate comparisons provide some evidence of the benefits of our proposed approaches, they do not clearly demonstrate the full potential of cascade modelling. Figure 2 shows the Pareto frontier of the trade-off performance for ICC, FCC, WCC, and the cost-sensitive baseline CEGB-BL using the early precision metric ERR@3. As the model cost increases, the advantages of using a cascade model on both collections become much clearer. Similar performance profiles can be shown for all of the metrics in Table 4 and Table 5, but are omitted here due to space limitations. For both collections, our new cascading approaches can produce greater effectiveness gains at a lower cost, with ICC consistently being the best approach overall. Any of these models could be further tuned to choose a more favourable trade-off stopping point based on the requirements of the system. The ability to more easily control the optimization of this trade-off is a key strength of the cascade models presented in this paper.

Table 6: Yahoo! S1. Risk sensitivity of systems when compared against the BM25F-SD baseline.

System	BM25F-SD − T_{risk} $\alpha = 2$		
	RBP@0.5	ERR@3	NDCG@5
LGBM-BL	**3.687**	1.899	-1.516
CEGB-BL	-1.252	-0.687	**-3.882**
ICC	0.686	1.368	**-2.607**
FCC	1.885	1.852	-1.094
WCC	0.288	0.480	**-3.883**

Risk Sensitivity. However, more control over model optimization can also have unexpected consequences. One such danger is the risk of *over-optimization*, which has become an increasingly common problem with the widespread adoption of machine learning models in computer science. One useful sanity check for new LTR models is to measure the risk sensitivity using a metric such as T_{risk} [13]. Table 6 shows the corresponding T_{risk} profile for Yahoo! S1 systems. Each system is compared against the BM25F-SD baseline, and we omit LMartC3-BL since it is not as competitive as the other systems. A T_{risk} score greater than +2 or less than −2 (shown in boldface) is regarded as significant for upside reward (no risk) or downside risk (high risk) respectively. LGBM-BL exhibits the least risk overall, and has significant upside reward for RBP, however the cost objective is unbounded. CEGB-BL poses a significant downside risk for NDCG, as do ICC and WCC. Interestingly, FCC has the best risk profile among the three new cascading models despite having the worst overall performance in terms of cost. This might be attributed to the model using more costly features, which produces a risk profile closer to the full LGBM-BL model.

In order to gain further insight into the potential risks of cost-aware modeling, we compare our best performing model ICC

Table 5: MSLR-WEB10K (136 features), results averaged over the 5 folds. Entries marked △, ▲ correspond to statistical significance using a paired t-test with Bonferroni correction at 95% and 99% confidence intervals respectively. Comparisons are relative to CEGB-BL.

	RBP	ERR@k			NDCG@k						
System	$p = 0.5$	@1	@3	@5	@1	@3	@5	# Stages	# Feat. Stage	# Feat. Total	Cost
Ground Truth Models											
LGBM-BL	0.377▲	0.249▲	0.330▲	0.354▲	0.477△	0.465▲	0.471▲	1	132	132	3,382
CEGB-BL (31)	0.375▲	0.248▲	0.329▲	0.353▲	0.476	0.463▲	0.468▲	1	130	130	3,330
CEGB-BL	0.372	0.245	0.325	0.349	0.470	0.459	0.464	1	127	127	3,238
LMartC3-BL	0.226▼	0.115▼	0.172▼	0.193▼	0.242▼	0.251▼	0.262▼	3	99/113/122	122	2,951
BM25	0.206▼	0.074▼	0.129▼	0.153▼	0.212▼	0.238▼	0.257▼	1	1	1	100
Experimental Cascade Models											
ICC	0.374	0.251▲	0.331▲	0.354▲	0.478	0.463△	0.465	4	70/105/106/127	128	1,728
FCC	0.374△	0.250▲	0.330▲	0.353▲	0.477△	0.462▲	0.467	4	94/119/117/128	129	2,802
WCC	0.370	0.249	0.327	0.350	0.474	0.458	0.460	4	85/116/113/126	129	1,873

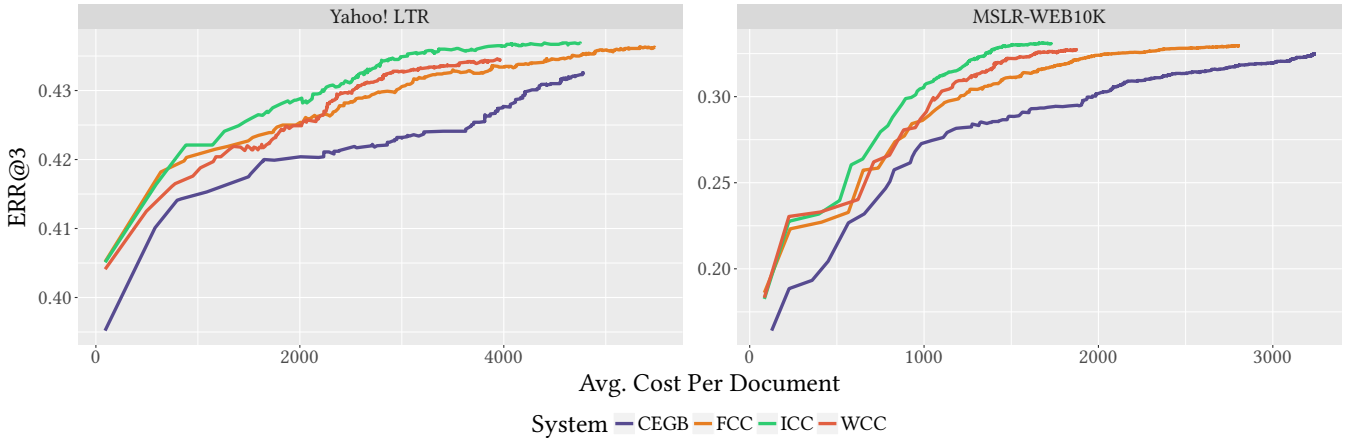

Figure 2: Yahoo! S1 (left), MSLR-WEB10K (right). Quality versus cost trade-off for the three backprop cascades (ICC, FCC, WCC), and CEGB-BL baseline. The curves depict the per tree evaluation within the ensemble for each system.

against the CEGB-BL baseline on Yahoo! S1 relative to the BM25F-SD scores, which would represent an initial Stage0 ordering of candidate documents. Figure 3 is the resulting scatterplot for NDCG@5 (the metric that has the highest risk in Table 6 for all of the methods), showing wins and losses greater than 10%. While both systems are significantly more effective than the baseline overall, there are quite a few queries where performance is measurably worse, indicating care should be taken when optimizing cost-aware models as they are susceptible to over-tuning.

For cost-aware systems the trade-off between user model, cost and risk is crucial, and understanding how to train models for risk sensitivity [12, 39], efficiency, and effectiveness simultaneously is an interesting area of future work worth pursuing.

5 CONCLUSION

This paper explores approaches to address two open research questions in cost-aware cascaded learning. We develop a general framework for jointly learning a cascade of ranking models composed of state-of-the-art tree boosting algorithms using back-propagation (RQ1). We extend this approach using a state-of-the-art, cost-aware, learning algorithm CEGB-BL, and show how to construct tree-based, cascade models (RQ2). We demonstrate that globally optimized, cascade rankers can achieve much better trade-offs between efficiency and effectiveness than previous approaches to cascade ranking, and have additional advantages over the best known cost-aware, single-stage, tree-based ranking models. Our results open up new opportunities for further investigation into the merits of combining complexity reduction techniques and cost constrained learning in unified cascade ranking frameworks.

Acknowledgments. This work was supported by the Australian Research Council's *Discovery Projects* Scheme (DP170102231), and an Australian Government Research Training Program Scholarship.

REFERENCES

[1] N. Asadi and J. Lin. 2013. Document Vector Representations for Feature Extraction in Multi-Stage Document Ranking. *Inf. Retr.* 16, 6 (2013), 747–768.

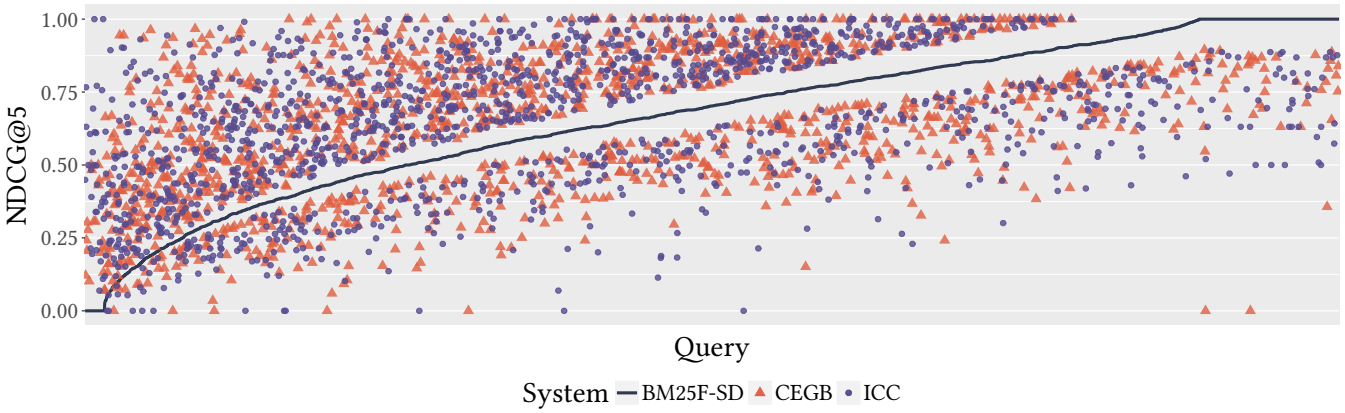

Figure 3: Yahoo! S1. Per query wins and losses shown for systems CEGB-BL and ICC compared to the BM25F-SD baseline. A win or loss is shown when it is 10% greater or 10% lower than BM25F-SD and is the overall winner or loser for a given query. Wins/losses for CEGB-BL and ICC are 921/542 and 1120/510 respectively.

[2] N. Asadi and J. Lin. 2013. Effectiveness/Efficiency Tradeoffs for Candidate Generation in Multi-Stage Retrieval Architectures. In *Proc. SIGIR*. 997–1000.

[3] A. Broder, E. Gabrilovich, V. Josifovski, G. Mavromatis, D. Metzler, and J. Wang. 2010. Exploiting Site-level Information to Improve Web Search. In *Proc. CIKM*. 1393–1396.

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

[5] B. B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt. 2010. Early Exit Optimizations for Additive Machine Learned Ranking Systems.. In *Proc. WSDM*. 411–420.

[6] G. Capannini, C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, and N. Tonellotto. 2016. Quality versus efficiency in document scoring with learning-to-rank models. *Inf. Proc. & Man.* 52, 6 (2016), 1161–1177.

[7] O. Chapelle and Y. Chang. 2011. Yahoo! Learning to Rank Challenge Overview. *J. Mach. Learn. Res.* 14 (2011), 1–24.

[8] R-C. Chen, L. Gallagher, R. Blanco, and J. S. Culpepper. 2017. Efficient Cost-Aware Cascade Ranking in Multi-Stage Retrieval. In *Proc. SIGIR*. 445–454.

[9] T. Chen and C. Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proc. KDD*. 785–794.

[10] J. S. Culpepper, C. L. A. Clarke, and J. Lin. 2016. Dynamic Cutoff Prediction in Multi-Stage Retrieval Systems. In *Proc. ADCS*. 17–24.

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

[12] B. T. Dinçer, C. Macdonald, and I. Ounis. 2016. Risk-sensitive evaluation and learning to rank using multiple baselines. In *Proc. SIGIR*. 483–492.

[13] B. T. Dinçer, C. Macdonald, and I. Ounis. 2014. Hypothesis testing for the risk-sensitive evaluation of retrieval systems.. In *Proc. SIGIR*. 23–32.

[14] M. M. Dundar and J. Bi. 2007. Joint Optimization of Cascaded Classifiers for Computer Aided Detection. In *IEEE Conf. on Comp. Vis. and Pat. Recog.* 1–8.

[15] J. Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[16] X. Jin, T. Yang, and X. Tang. 2016. A Comparison of Cache Blocking Methods for Fast Execution of Ensemble-based Score Computation. In *Proc. SIGIR*. 629–638.

[17] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T-Y. Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Proc. NeurIPS*. 3149–3157.

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

[19] Y. LeCun, L. Bottou, G. B. Orr, and K-R. Müller. 1998. Efficient backprop. In *Neural networks: Tricks of the trade*. Springer, 9–50.

[20] S. Liu, F. Xiao, W. Ou, and L. Si. 2017. Cascade Ranking for Operational E-commerce Search. In *Proc. KDD*. 1557–1565.

[21] T.-Y. Liu. 2009. Learning to Rank for Information Retrieval. *Foundations and Trends in Information Retrieval* 3, 3 (2009), 225–331.

[22] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, F. Silvestri, and S. Trani. 2016. Post-learning optimization of tree ensembles for efficient ranking. In *Proc. SIGIR*. 949–952.

[23] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. 2015. QuickScorer: A Fast Algorithm to Rank Documents with Additive Ensembles of Regression Trees. In *Proc. SIGIR*. 73–82.

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

[25] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, and S. Trani. 2017. X-DART: Blending Dropout and Pruning for Efficient Learning to Rank. In *Proc. SIGIR*. 1077–1080.

[26] C. Lucchese, F. M. Nardini, R. Perego, S. Orlando, and S. Trani. 2018. Selective Gradient Boosting for Effective Learning to Rank. In *Proc. SIGIR*. 155–164.

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

[28] C. Macdonald, R. L. T. Santos, I. Ounis, and B. He. 2013. About learning models with multiple query-dependent features. *ACM Trans. Information Systems* 31, 3 (2013), 11:1–11:39.

[29] J. Mackenzie, J. S. Culpepper, R. Blanco, M. Crane, C. L. A. Clarke, and J. Lin. 2018. Query Driven Algorithm Selection in Early Stage Retrieval. In *Proc. WSDM*. 396–404.

[30] I. Matveeva, C. Burges, T. Burkard, A. Laucius, and L. Wong. 2006. High Accuracy Retrieval with Multiple Nested Ranker. In *Proc. SIGIR*. 437–444.

[31] H. R. Mohammad, K. Xu, J. Callan, and J. S. Culpepper. 2018. Dynamic Shard Cutoff Prediction for Selective Search. In *Proc. SIGIR*. 85–94.

[32] V. Nair and G. E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proc. ICML*. 807–814.

[33] S. Peter, F. Diego, F. A. Hamprecht, and B. Nadler. 2017. Cost Efficient Gradient Boosting. In *Proc. NeurIPS*. 1550–1560.

[34] C. Pölitz and R. Schenkel. 2011. Learning to rank under tight budget constraints. In *Proc. SIGIR*. 1173–1174.

[35] T. Qin and T-Y. Liu. 2013. Introducing LETOR 4.0 Datasets. *CoRR* (2013). http://arxiv.org/abs/1306.2597

[36] K. V. Rashmi and R. Gilad-Bachrach. 2015. DART: Dropouts meet Multiple Additive Regression Trees. *J. Mach. Learn. Res.* 38 (2015), 489–497.

[37] V. C. Raykar, B. Krishnapuram, and S. Yu. 2010. Designing efficient cascaded classifiers: tradeoff between accuracy and cost. In *Proc. KDD*. 853–860.

[38] V. N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA.

[39] L. Wang, P. N. Bennett, and K. Collins-Thompson. 2012. Robust ranking models via risk-sensitive optimization. In *Proc. SIGIR*. 761–770.

[40] L. Wang, J. Lin, and D. Metzler. 2010. Learning to efficiently rank. In *Proc. SIGIR*. 138–145.

[41] L. Wang, J. Lin, and D. Metzler. 2011. A Cascade Ranking Model for Efficient Ranked Retrieval. In *Proc. SIGIR*. 105–114.

[42] Z. Xu, M. J. Kusner, K. Q. Weinberger, and M. Chen. 2013. Cost-Sensitive Tree of Classifiers.. In *Proc. ICML*. 133–141.

[43] Z. Xu, M. J. Kusner, K. Q. Weinberger, M. Chen, and O. Chapelle. 2014. Classifier Cascades and Trees for Minimizing Feature Evaluation Cost. *J. Mach. Learn. Res.* 15 (2014), 2113–2144.

[44] Z. Xu, K. Q. Weinberger, and O. Chapelle. 2012. The Greedy Miser: Learning Under Test-time Budgets. In *Proc. ICML*. 1175–1182.