Efficient Cost-Aware Cascade Ranking in Multi-Stage Retrieval

Ruey-Cheng Chen
RMIT University
Melbourne, Australia

Luke Gallagher
RMIT University
Melbourne, Australia

Roi Blanco
RMIT University
Melbourne, Australia

J. Shane Culpepper
RMIT University
Melbourne, Australia

ABSTRACT

Complex machine learning models are now an integral part of modern, large-scale retrieval systems. However, collection size growth continues to outpace advances in efficiency improvements in the learning models which achieve the highest effectiveness. In this paper, we re-examine the importance of tightly integrating feature costs into multi-stage learning-to-rank (LTR) IR systems. We present a novel approach to optimizing cascaded ranking models which can directly leverage a variety of different state-of-the-art LTR rankers such as LambdaMART and Gradient Boosted Decision Trees. Using our cascade model, we conclusively show that feature costs and the number of documents being re-ranked in each stage of the cascade can be balanced to maximize both efficiency and effectiveness. Finally, we also demonstrate that our cascade model can easily be deployed on commonly used collections to achieve state-of-the-art effectiveness results while only using a subset of the features required by the full model.

1 INTRODUCTION

Learning-to-Rank (LTR) systems are now commonly deployed by major search engine companies and they have been repeatedly shown to be highly effective for a variety of search related problems [6, 15, 26, 30]. There has been a growing body of recent work which focuses on improving the efficiency of multi-stage LTR systems using several different techniques: improving tree traversal [19], cascaded ranking [36], tree pruning [18, 38, 39], and minimizing sample sizes in stages [11, 22].

In this paper we revisit the idea of cascaded ranking in order to provide more control over efficiency and effectiveness trade-offs in large scale search systems. A cascade ranking model [36] is a sequence of learning-to-rank models (called *stages*) chained together to collectively rank a set of documents for a query. The main assumption behind cascaded ranking is that full inspection of the content, which would presumably require generating expensive features is not required for every incoming document as only a small fraction of all documents will be relevant. Therefore, LTR models in a cascade can be deployed in an ascending order of model complexity, and only a fraction of documents in each stage will advance to the next stage. Generally, early-stage rankers are cheaper to run, and usually focus on executing an early-exit strategy, such as filtering out non-relevant documents as quickly as possible. Ranking models in later stages are usually more accurate but require more resources.

When discussing system performance, it is important to consider both ranking effectiveness and system throughput within the same framework. Wang et al. [36] used a modified AdaRank algorithm to incorporate the costs of individual rankers, in terms of execution time for each single-feature weak learner used in the training procedure. This cascade model, however, cannot be used with gradient-boosted tree models, which are now widely believed to be state-of-the art for web search ranking algorithms [25, 30].

Conceptually, the making of a tree-based cascade model can be reasonably separated into two steps, which are *cascade construction* and *model deployment*. In the first step, a learning algorithm takes into account the effectiveness of features and the cost of feature extraction, makes the best tradeoffs by following the direction from cascade designer, and automatically trains a cascade of ranking models. The learned cascade can then be deployed in the second step, focusing on optimizing low-level system performance. In this paper, we develop a new approach to constructing a cost-aware cascade. A considerable amount of recent research effort has been invested in the space of optimizing the run-time performance of gradient-boosted tree models [3, 14, 19, 20], which can be directly leveraged by our new cascading approach.

Research Goals. In this work, we revisit the problem of integrating feature costs into learning-to-rank models. In particular, we focus on how best to balance feature importance and feature costs in multi-stage cascade ranking models. Our overarching goal is to devise a generic framework which can be used with *any* state-of-the-art LTR algorithm, and allows more control over balancing efficiency and effectiveness in the entire re-ranking process. In order to achieve these goals, we focus on two related research problems:

Research Question (RQ1): *When designing multi-stage retrieval systems, what approaches provide the best balance between extraction/runtime costs and feature importance when using cascaded LTR algorithms?*

Research Question (RQ2): *Can we build multi-stage ranking models that require substantially less costs than a full cost-insensitive model, and still achieve overall effectiveness close to the full model?*

SIGIR '17, August 07-11, 2017, Shinjuku, Tokyo, Japan

978-1-4503-5022-8/17/08…$15.00
DOI: http://dx.doi.org/10.1145/3077136.3080819

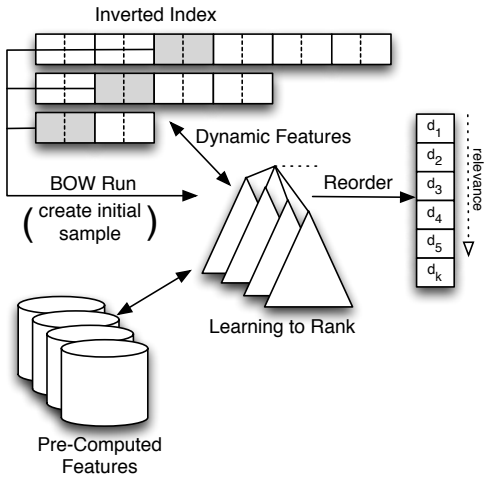


Figure 1: A typical learning-to-rank system configuration is composed of an inverted index which is used to generate an initial candidate set (sample) of s documents. This set of documents is then re-ordered using one or more rounds of machine learning algorithms. The number of documents can be pruned in each round, or iteratively smaller subsets of the highest ranking documents in the initial sample s are re-ordered. A final top-k set of documents are then returned from the system in relevance order.

2 BACKGROUND AND RELATED WORK

Learning to rank. A significant body of prior work exists in the area of learning-to-rank (LTR) [15]. The majority of research advances in LTR have focused on ways to improve the effectiveness of the systems, with several document collections released to test their performance. A recent study by Tax et al. [30] compare 87 learning to rank methods using 20 different test collections.

However, one common problem with these test collections is that the features used by the models are often not fully defined, making it very difficult to implement them using commonly used IR test collections. This in turn prevents easily transferring the advances made into working end-to-end search systems. While many different publicly available search engines [32] are commonly used by researchers and practitioners, only `Terrier 4.x` [23] currently supports end-to-end multi-stage retrieval on commonly used IR document collections with little or no manual intervention. So the chasm between academic research and large search engine companies on provably good system architectures remains relatively wide. Figure 1 shows the architecture of a complete LTR system consisting of at least two stages. Every aspect of this architecture should be considered when building effective and efficient search systems. Macdonald et al. [22] were among the first to consider all of the different angles when building an LTR system for adhoc search.

Improving Efficiency in LTR Algorithms. A critical aspect of LTR must be considered when translating these powerful models into working search engines which must index internet-scale document collections – efficiency. Efficiency concerns may be strictly algorithmic [3, 7, 14, 19, 20], they may explicitly focus on feature costs [1, 6, 23, 35–37, 39], or they may perform post-learning optimizations to reduce the size of the tree ensembles [18, 38, 39].

Another related line of research is to focus on the importance of balancing efficiency and effectiveness in LTR systems, which is directly aligned with our current work. Perhaps the most comprehensive study on this problem is the recent work of Capannini et al. [7]. A less obvious trade-off concern is how to construct the "sample" of documents that are used for training and for scoring at runtime for new queries coming into the system [10, 22]. This issue can have an important impact on both training and runtime scoring in multi-stage systems, and a problem that we revisit in the context of cascaded ranking. Finally, the cost of model training can also be an important problem [2, 22], but is not explored further in this work.

Cascade Ranking. Raykar et al. [28] described an approach to jointly train a cascade of classifiers to support clinical decision making, with the expected cost of feature acquisition taken into account. This approach does not attempt to address the issues of cascade design, such as the number of cascade stages and the design of cutoffs. The closest work to our own are the cascade models previously explored by Wang et al. [36] and Xu et al. [39]. Wang et al. proposed a cascade learning algorithm based on an additive ranking model AdaRank. The algorithm produces a cascade by incrementally incorporating weaker rankers in the ascending order of cost efficiency. In each stage only one weak ranker is incorporated. The document scores are accumulative, so conceptually all the previously selected features are involved in the scoring. However, more recent improvements in GBRT-based LTR algorithms has made this approach less competitive than state-of-the-art learning models.

Xu et al. [39] proposed an algorithm that takes a trained GBDT model and produces a cascade by reweighting the trees in the full model. The effectiveness of the cascade is roughly the same as a full GBDT model. They use a monolithic cost function which accounts for several variables such as: model loss, tree evaluation costs, and feature cost. However, optimization of this loss function is quite complex, and their approach do not address the design issues pointed out in this paper, such as the effect of cascade structures on the final retrieval effectiveness of the cascade model.

3 APPROACH

Stage-wise cascades are flexible models that allow for a number of architectural decisions, such as: the number of stages used, the number of documents forwarded to the next stage, and so on. The choice of features involved in each stage is a critical factor in balancing efficiency and effectiveness in the end-to-end system. This trade-off is further elucidated by the following observations:

- A cascade may choose to defer the use of expensive features to later cascade stages as feature extraction on fewer documents is necessary, and will be more *cost efficient*.
- A cascade may choose to include useful features early on, since features extracted in earlier stages can be re-used in all remaining stages without incurring additional costs. The reusability of key features can make the cascade more *effective*.

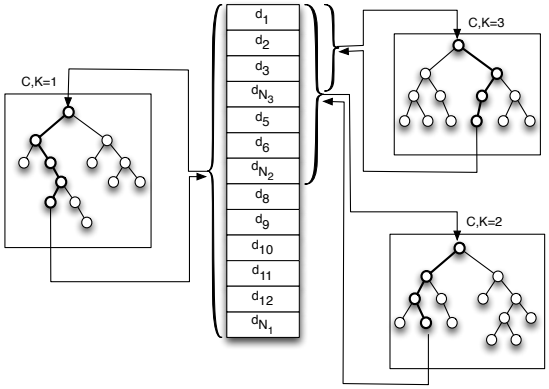


Figure 2: A three level cascade which initially takes d_s documents as the sample input. In Round 1, $C, K = 1$ reorders all d_{N_1} documents. In Round 2, a subset of the d_{N_2} documents are reordered by $C, K = 2$. In the final round, d_{N_3} documents are reordered. Up to d_N documents total can be returned from the final level.

Our general approach to cascade construction is to first assign feature sets to different stages using a set of predefined heuristics (c.f. Sec. 4), and then perform automatic feature selection for every stage of the cascade, while jointly optimizing ranking effectiveness and efficiency. Ideally, the procedure should maintain performance comparable to a complete feature set model while at the same time accounting for feature costs. We now describe a theoretical framework for *model regularization* that reuses well-known solutions in the machine learning field in order to achieve both objectives. Even though the goal of regularization is to minimize the effect of *overfitting*, in this paper we show how it can also be used to produce compact models that are feature extraction cost aware.

3.1 Cost-Aware Feature Selection

Regularization. Supervised machine learning algorithms are exposed to a training set of pairs $\{(\mathbf{x_i}, y_i)\}^n$ with the goal of finding an approximation to the function h, mapping y to $\mathbf{x}$ that minimizes the expected value of a predefined loss function $\mathcal{L}(y, h(\mathbf{x}))$ over the joint distribution of all $(\mathbf{x}, y)$ values:

$$h^* = \underset{h \in H}{\operatorname{argmin}} E_{y,\mathbf{x}}[\mathcal{L}(y, h(\mathbf{x}))] = \underset{h \in H}{\operatorname{argmin}} E_{\mathbf{x}}[E_y[\mathcal{L}(y, h(\mathbf{x}))]|\mathbf{x}]. \quad (1)$$

The choice of loss function depends on the type of problems being learned (classification, regression, pairwise, listwise). It is common practice to incorporate a regularization term $R(h)$ in the loss function to prevent overfitting. Regularization usually leads to improved effectiveness because sparsity is enforced in model training and, as a result, the learned model is less likely to overfit the training set. The most common type of regularizers apply a penalty on the complexity, shrinking the value of the parameters in order to reduce overfitting. For instance, if $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ is a linear model with its parameters represented by a weight vector $w \in R^d$, a widely-used regularizer is the L_2 norm of the weight vector. Given a training set of n instances, the model would minimize the following expression:

$$h^* = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^{n} \mathcal{L}(y, h_{\mathbf{w}}(\mathbf{x})) + \beta\|\mathbf{w}\|_2^2 , \quad (2)$$

where in this case h functionally depends on $\mathbf{w}$. For instance $h(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x})$, where ϕ is a kernel feature mapping.

Cost-Aware L_1 regularization. One problem with Equation 2 is that the learning algorithm is agnostic to feature costs. In order to minimize costs, one would like to reduce the number of features (covariates) that are used by the model, weighted by their cost, and at the same time maximize the performance. This problem is closely related to feature selection, and has a close connection with regularization. In fact, Equation 2 tries to bring down the contribution (weight) of each feature as much as possible while also minimizing the loss. In our case, however, having a non-zero weight for a particular feature implies that we have to pay *the whole* cost of extracting it, no matter how small it is.

Let $\mathbf{c} \in R^d$ be the feature-cost vector, in which each entry represents the normalized cost for extracting the feature. In the case of a linear model, we want to minimize $\mathbf{c}^T\mathcal{I}_{>0}(\mathbf{w})$, where $\mathcal{I}_{>0}$ is the component-wise indicator function, which is 1 if the weight is over zero, and 0 otherwise. This penalty factor would be included in the formulation of Equation 2. In practice, this means we need a procedure for controlling the amount of covariates included in the final model automatically. To do this, we allow the learner to perform automatic feature selection by adding a L_1 penalty to the loss function (Eq. 2). This penalty is the L_1 norm of the weight vector weighted by the feature costs $\mathbf{c}$.

Conventionally, L_1 regularized regression models with a least square loss function are also known as LASSO (least absolute shrinkage and selection operator) and were originally designed to perform covariate selection, and help to make the model more interpretable. Lasso is able to achieve this by forcing the sum of the absolute value of the regression coefficients to be less than a fixed value, which in practice forces certain coefficients to be set to zero, effectively choosing a simpler model that does not include those coefficients [31]. In our case, we will exploit this property to generate less expensive models in terms of feature extraction time.

To sum up, the ranker would minimize the expression:

$$h^* = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^{n} \mathcal{L}(y, h_{\mathbf{w}}(\mathbf{x})) + \beta\|\mathbf{w}\|_2^2 + \lambda\|\mathbf{c} \circ \mathbf{w}\| , \quad (3)$$

where β and λ are parameters that control the trade-off between the loss and regularization penalty, and $\circ$ is the component-wise product. Therefore, the main idea is to learn a model using Equation 3, and then select the features that have a $\mathbf{w_i} > 0$, either directly in a linear model (we would use $h_{\mathbf{w}}$ for ranking), or as an input to other LTR methods (which would learn a model using only the subset of parameters selected).

There are several options for learning the parameters $\mathbf{w}$ of such a model. An effective method is to use stochastic gradient descent and update $\mathbf{w}$ one example at a time; in this case the training update for a sample $(\mathbf{x_j}, y_j)$ is as follows:

$$\mathbf{w}^{t+1} = \mathbf{w^t} - \eta^t \frac{\partial}{\partial \mathbf{w}} \left(\mathcal{L}(y, h_{\mathbf{w}}(\mathbf{x})) + \frac{\lambda}{n} \sum_i \mathbf{c}_i |\mathbf{w}_i| \right) , \quad (4)$$

where η^t is the learning rate, which may depend on t, the number of iterations so far. Note that the L_2 regularizer in Eq (3) is omitted here for clarity.

Strictly speaking, the L_1 norm is not differentiable (at $\mathbf{w} = 0$). However, methods that rely on subgradients can be used to solve minimization problems that involve L_1 regularized objective functions, which in this particular case, boils down to replacing the partial derivative of the regularizer with its sign, which for each feature i results in:

$$\mathbf{w}_i^{t+1} = \mathbf{w}_i^t - \eta^t \frac{\partial \mathcal{L}(y, h_{\mathbf{w}}(\mathbf{x}))}{\partial w_i} - \eta^t \frac{\lambda}{n} c_i \operatorname{sign}(\mathbf{w}_i^k) \tag{5}$$

One drawback of this formulation is that it may not produce a compact model, because the weight of a feature does not become zero unless it happens to be exactly zero, which is rare in practice. To overcome this limitation, we use a variant of a proximal method proposed by Tsuruoka et al. [33] which works well with the Stochastic Gradient Descent (SGD) optimization procedure, and has been shown empirically to produce very compact models. The main motivation is to smooth out the fluctuation of the gradients through multiple iterations, which can be high when using SGD as it approximates the true gradient, and is computed using the whole sample, one example at a time. The original method, named SGD-cumulative, approximates the loss gradient using the following update rules:

$$\hat{\mathbf{w}}_\mathbf{i}{}^{t+1} = \mathbf{w}_\mathbf{i}{}^t - \eta^t \frac{\partial \mathcal{L}(y, h_{\mathbf{w}}(\mathbf{x}))}{\partial \mathbf{w}_\mathbf{i}}\Big|_{\mathbf{w}=\mathbf{w}^t}, \tag{6}$$

$$u^t = \frac{\lambda}{n} \sum_{j=1}^{t} \eta^j, \tag{7}$$

$$q_i^t = \sum_{j=1}^{t} \left(\mathbf{w}_\mathbf{i}{}^{j+1} - \hat{\mathbf{w}}_\mathbf{i}{}^{j+1} \right), \tag{8}$$

and finally

$$\mathbf{w}_\mathbf{i}{}^{t+1} = \begin{cases} \max(0, \hat{\mathbf{w}}_i{}^{t+1} - (u^t + q^{t-1})), & \hat{\mathbf{w}}_i{}^{t+1} > 0 \\ \min(0, \hat{\mathbf{w}}_i{}^{t+1} + (u^t - q^{t-1})), & \hat{\mathbf{w}}_i{}^{t+1} \leq 0 \end{cases} \tag{9}$$

To introduce the per-feature cost $\mathbf{c}_i$, we create one u_i^t variable per feature as $u_i^t = \mathbf{c}_i \frac{\lambda}{n} \sum_{j=1}^{t} \eta^j$ which is then used to update $\mathbf{w}_\mathbf{i}{}^{t+1}$. It is important to note that the method is able to select a subset of features that can be used to further retrain any arbitrary model, and thus it can be used in combination with state of the art non-linear rankers such as the ones commonly used in production systems (GBRT or LambdaMART for example). Henceforth, we will use a hinge loss function h as in binary classification using Linear SVM:

$$\mathcal{L}(y, h_{\mathbf{w}}(\mathbf{x})) = \max \left\{ 0, 1 - \operatorname{sign}(2y - 1) \circ \operatorname{sign}(\mathbf{w}^T \mathbf{x}) \right\} \tag{10}$$

This proved to be empirically effective in our setup, while also converging quickly in fewer epochs.

There are several alternative feature selection methods, that in general are based on an optimality criteria metrics such as Bayesian information criterion, or Minimum Description Length. In this work, we also make further use of GBRT's feature importance [12],[1] as it intrinsically captures interdependencies between covariates. In short, the process learns a set of decision trees, where each node splits the data using one feature. With each split, the tree outputs are modified, and the training squared loss varies. Then, once an ensemble is learned, the non-terminal nodes of the trees can be iterated through to compute the reduction of squared loss for every feature, and the results aggregated for different feature splits. Lastly, the final importance is computed as the average over all of the trees.

[1] An equivalent process exists for the case of multi-class classification.

3.2 Cascade Construction

Constructing a cascade model involves setting a number of parameters, including the number of cascade stages K, the cutoff thresholds $\langle c_1, c_2, \ldots, c_K \rangle$, and the features sets used in each stage $\langle F_1, \ldots F_K \rangle$. As one might expect, the design space of a cascade model is humongous. The complexity of exploring the entire space of all possible parameter combinations and feature allocations is prohibitively large, and interdependencies between features can affect both the effectiveness and computational costs significantly.

Randomized Search. To tackle this problem, randomized search [4] is performed in this study to select the cascade configuration. This is done by randomly sampling a large number of cascade configurations from this space, followed by a seletion step that maximizes the cascade effectiveness on validation data. Ideally, this approach can explore any search space fairly efficiently within a relatively small number of rounds, but when feature allocation is involved many feature combinations it explores will not be effective. Randomized search does not work well when good configurations are difficult to reach.

The cost-aware L_1 regularization algorithm, as described in Sec. 3.1, was developed to mitigate this issue and simplify the search. It turns the search problem in a combinatorial space (that covers all possible ways of feature allocation) into a simple line search, making it possible to "sift through" the feature allocation space efficiently by tweaking λ. In our formulation, the coefficient λ controls the desired level of effectiveness-efficiency tradeoff, so when a different tradeoff is given a different subset of features that reflects this change will be selected. Practically speaking, a small λ leads to a gently reduced feature set with slightly decreased effectiveness compared to the full model; a large λ, on the other hand, will prune the feature set fairly aggressively and result in a compact model that uses only couples of features, which is ideal as an early stage model. Using this algorithm, a cascade can then be constructed by feeding in a sequence of decreasing λ values (from early to late) to generate cascade stages.

More details about the use of randomized search will be described in later sections. In the first two experiments, we use a set of predefined cascade configurations to simplify the experimental setup and serve as the experimental control. Further investigations on fine-tuning cascade configurations is carried out using GOV2 with the best-performing cascade methods discussed in Sec. 4.3.

Feature Availability. We also experimented with a number of feature availability settings, and assume that the availability of a feature may change across cascade stages. In a production search system, some features might arrive much later than the others for various reasons, such as that they are expensive to run or their generation being deferred due to the design of the feature extraction procedure. To simulate this effect, our approach is to have certain models subdivide the full feature set into K equal-sized

partitions and assign feature partitions to the respective cascade stages. The rationale behind this approach is that, by presenting a limited choice of features, which is $1/K$ of the full set, a feature extraction pipeline can be simulated to work in parallel with the ranking models, serving features in an order based on a pre-defined criteria. Each cascade stage has access to all features extracted in the previous stages without incurring additional costs. In this paper, we explored three different feature availability settings:

(1) **Cost-biased allocation (C)**: Features are first sorted in ascending order of unit cost and partitioned into K stages. This setting is a close approximation to the scenario where cheap features are available to the ranking model earlier than expensive ones.
(2) **Cost-efficiency-biased allocation (E)**: The features are first sorted in descending order of cost efficiency and partitioned into K stages. The cost efficiency of a feature is defined as the importance score divided by its unit cost, where importance is computed from a ground-truth tree model as described in Sec. 3.1. This setting simulates having a dedicated extraction pipeline for more cost-efficient features.
(3) **Full allocation (F)**: All features are accessible from individual cascade stages. This setting represents the scenario where the choice of extracted feature is unrestricted, providing the greatest flexibility to the underlying cost-aware feature selection algorithm.

After applying one of these settings, cost-aware L_1 regularization is performed to each cascade stage separately with a sequence of decreasing λ values. The algorithm (Sec. 3.1) will reach the desired level of feature size in 10–20 epochs. Running this procedure for more iterations does not change the results. We also set a constant decaying learning rate $\eta = 0.1$ across the board.

4 EXPERIMENTS

We now evaluate the impact of our approaches on reducing costs in cascade learners in two different settings, one large but shallow LTR dataset, and a standard TREC benchmark with 150 queries but a large number of documents to be ranked per query.

Experimental Setup. All experiments were executed on a 24-core Intel Xeon E5-2630 with 256 GB of RAM hosting RedHat RHEL v7.2, and baselines generated using Indri[2], Krovetz stemming, and dependency models generated using Metzler's MRF configuration[3]. All LTR algorithms were implemented in Python using `scikit-learn`[4] `0.18.1` and `xgboost` [5] `0.6a2`. Source code, configuration files, and detailed explanations for all experiments can be found in the GitHub repository for this paper[6].

Two different test collections were used for the experiments. The first collection is the C14 Webscope Yahoo Learning To Rank dataset [7] [9]. The dataset contains two subsets designed for different purposes and used different feature sets. We use only Set 1 (Y!S1) which contains 519 features (out of 700 in total) with an associated feature cost, and has 19,944 training queries, 1,266 validation queries, and 3,798 test queries. The original cost estimates included with the data were used without modification in our experiments. All features used in Set 1 have extraction costs between 1 and 200. Our second collection is the TREC GOV2 test collection (GOV2) using queries 701–850 in a 5-fold cross validated configuration. We created 425 features for this collection as described next in Section 4.2. For all queries, we created the initial sample by running BM25 with $k_1 = 0.9$ and $b = 0.4$ to an initial depth of 5,000. A summary of the two benchmark collections are shown in Table 1.

Table 1: Summary of the key properties of the two benchmark collections used in this study.

	# Queries	# Total Docs	# Features
Y!S1	6,983	165,660	519
GOV2	150	1,500,000	425

Learning Algorithms. Table 2 summarizes all of the baselines, as well as all of the new cascade model configurations tested on the two collections. We used a broad range of different learning algorithms in our experiments. These ranking models can be divided into the following three categories:

(1) **Ground Truth Models**: We compare with ranking models executed in a non-cascade setting, where the full set of features is used in training and prediction. Three ranking models are employed: Gradient-Boosted Decision Trees (GBDT) [12], Gradient-Boosted Regression Trees (GBRT) [12], and LambdaMART [5]. Our implementations of these ranking models are based on `xgboost`.
(2) **Baselines**: We use several baseline methods, such as QL [40], BM25 [29], and SDM [24], on GOV2 to verify the gain in effectiveness relative to a standard retrieval setting. These baseline methods are however not available for Y!S1.
(3) **Cascade Baselines**: We implemented the cascade ranking algorithm described in Wang et al. [36], using the suggested setting $\gamma = 0.1$. Note that setting a smaller γ does not improve its effectiveness. We also implemented early stopping on training effectiveness to avoid explicitly setting the number of cascade iterations.

Model hyperparameters (number of trees, depth, learning rate) were trained with the provided independent validation set for Y!S1, and using 5-fold cross-validation on GOV2. For ease of experimentation, some cascade parameters, such as the number of stages K, and cutoff thresholds $\langle c_1, c_2, \ldots, c_K \rangle$, were fixed in the first two experiments. Other parameters, such as λ, were tuned on the validation data using randomized search. Tree cascade parameters were tuned differently on the two datasets, as previous parameters for the ground truth models did not always generalize well on the learned cascades. We also empirically found that the linear cascades work better with the L_2 regularization turned off (i.e., $\beta = 0$), making the SGD optimizations more stable. Further experimental details are described in Sections 4.1 and 4.2.

Evaluation Metrics. For retrieval effectiveness, we used standard early precision evaluation metrics: Expected Reciprocal Rank (ERR),

[2] http://www.lemurproject.org/indri.php
[3] http://ciir.cs.umass.edu/~metzler/dm.pl
[4] http://scikit-learn.org/stable/
[5] https://github.com/dmlc/xgboost
[6] https://github.com/rmit-ir/LTR_Cascade
[7] https://webscope.sandbox.yahoo.com/catalog.php?datatype=c

Table 2: Summary of the baselines and new cascading methods used.

Method Name	Parameters	Description
GBDT-BL	Y!S1/GOV2: 1,000/525 trees, 16/16 nodes	GBDT [12] (`xgboost`), $\eta = 0.05$, subsample rate 0.8.
GBRT-BL	Y!S1/GOV2: 1,000/525 trees, 16/16 nodes	GBRT[12] (`xgboost`), $\eta = 0.05$, subsample rate 0.8.
LambdaMART-BL	Y!S1/GOV2: 1,000/525 trees, 16/32 nodes	LambdaMART [5] (`xgboost`), $\eta = 0.05$, subsample rate 0.8.
QL-BL, BM25-BL, SDM-BL	Default Indri settings	Commonly used single-pass retrieval runs to depth 1,000 using Query likelihood with Dirichlet priors smoothing, BM25, and a Sequential Dependency Model (SDM). Note that while SDM is a strong effectiveness baseline, it has well-known efficiency limitations when used on large document collection [17].
WLM-BL	Y!S1: $\gamma = 0.1$, GOV2: $\gamma = 0.1$	Reimplementation of the linear cascade model by Wang et al. [36], with early stopping on training NDCG.
LM-C3-*X*		Three level cascade using linear model under the elected feature availability setting *X*, trained using Stochastic Gradient Descent (SGD) with batch size set to 50 and $\eta = 0.1$. The setting *X* could be C/E/F.
GBDT-C3-*X*	Y!S1: 1,000 trees, 16 nodes GOV2: adaptive (Sec. 4.2)	Three level cascade using GBDT under the elected feature availability setting *X*, using the same SGD configuration as LM-C3-*X*.
GBRT-C3-*X*	Y!S1: 1,000 trees, 16 nodes GOV2: adaptive (Sec. 4.2)	Three level cascade using GBRT under the elected feature availability setting *X*, using the same SGD configuration as LM-C3-*X*.
LambdaMART-C3-*X*	Y!S1: 1,000 trees, 16 nodes GOV2: adaptive (Sec. 4.2)	Three level cascade using LambdaMART under the elected feature availability setting *X*, using the same SGD configuration as LM-C3-*X*.

Normalized Discounted Cumulative Gain (NDCG), and Precision (P), with three cutoffs (5, 10, and 20). We use `gdeval`[8] to compute ERR and NDCG, and `trec_eval`[9] to compute the precision to ensure that reported numbers are easily reproducible.

In this work, we focus on early precision improvements only, but if deeper metrics are desirable, our cascade approach can be tuned to support it. The cost of a cascade is given by the following formula:

$$\frac{1}{N}\sum_{i=1}^{K}\sum_{f\in F_i} N_i\, C(f),$$

where $C(f)$ denotes the unit cost of feature f, N_i denotes the number of documents that enter cascade level i, and N denotes the total number of documents that enter the cascade.

4.1 Experiments on the Y!S1 Collection

In the first experiment, we tested the effectiveness of the proposed cascade ranking algorithm on the Y!S1 collection. As a significant number of queries in this data have less than 40 retrieved documents, there is relatively little flexibility in the design of cascade stages and cutoffs. In our initial investigation, we chose to utilize a fixed configuration to simplify the experimental design. The cascade is configured to contain only 3 stages, with fixed cutoffs $\langle 20, 10\rangle$ between stages.

The λ values for the linear cascade models were derived using randomized search and NDCG on the validation data (cf. Table 3). For simplicity, all of the tree cascades in this experiment were trained with the same parameter setting as their ground truth counterparts. Note that tuning the number of trees/nodes in the tree cascades can further improve the performance, and this approach is explored further in Sec. 4.2.

[8] http://trec.nist.gov/data/web/10/gdeval.pl
[9] http://trec.nist.gov/trec_eval/

Main Results. The main results for the Y!S1 experiments are presented in Table 3. In the table, the results are divided into three sections. From top to bottom they are: ground truth models, the cascade baseline, and the proposed cascade ranking models. Ground truth models, such as GBDT-BL or GBRT-BL, provide the best effectiveness in general, but the feature extraction costs are also significantly higher. Interestingly, these models already perform their own kind of feature selection as some of the input features are never used in the final trees, and therefore incur different costs.

When comparing cascading models, the cascade baseline WLM-BL spends far less (0.62% of the cost incurred by GBDT-BL) on feature extraction than full models, at the cost of degraded effectiveness. Cascade models LM-C3-C, LM-C3-E, and LM-C3-F performed relatively poorly in terms of ERR@k and NDCG@k with respect to ground truth models, but in general their effectiveness is better than the WLM-BL baseline. The tree-based cascade models are more competitive than their linear model counterparts. Among all cascading models, LambdaMART-based cascades appear to provide the best tradeoff. The best-performing cascade LambdaMART-C3-F significantly outperformed WLM-BL on all 9 tested metrics, but is still less efficient that all three ground truth models, leaving a noticeable gap of 0.01–0.02 in ERR@k, 0.04–0.05 in NDCG@k, and 0.01–0.03 in P@k.

4.2 Experiments on the GOV2 Collection

In the second experiment, we investigate the use of the cascade ranking models on a commonly used web test collection, GOV2, where documents and features are to be processed and extracted by ourselves. To prepare the data for the cascade ranking experiment, for each query we retrieved 5,000 documents using BM25, and for each retrieved document 425 query or non-query features were extracted. All 425 of the features implemented depend on either: the query; the query and term statistics from the indexed postings; the query, document and bigram statistics from ephemeral postings;

Table 3: Main results on Yahoo! Learning-to-Rank Challenge data. For the proposed cascade models, significant improvements over WLM-BL are indicated by * for $p < 0.05$ and ** for $p < 0.01$ in a paired t-test.

	ERR@k			NDCG@k			P@k			
System	@5	@10	@20	@5	@10	@20	@5	@10	@20	Cost
Ground Truth Models										
GBDT-BL	**0.4605**	**0.4751**	**0.4789**	**0.7448**	**0.7872**	**0.8279**	0.8323	**0.7577**	**0.5967**	15988
GBRT-BL	0.4598	0.4744	0.4782	0.7420	0.7852	0.8264	0.8322	0.7562	0.5962	15876
LambdaMART-BL	0.4526	0.4674	0.4712	0.7314	0.7768	0.8203	**0.8330**	0.7564	0.5964	15856
Cascade Models (including Baseline) [a]										
WLM-BL	0.3679	0.3876	0.3933	0.5886	0.6506	0.7088	0.7832	0.7171	0.5673	99
LM-C3-C	0.3950	0.4127	0.4175	0.6461	0.7067	0.7638	0.8086**	0.7364**	0.5856**	1871
LM-C3-E	0.3871	0.4039	0.4089	0.6503	0.7033	0.7618	0.8192**	0.7413**	0.5885**	1580
LM-C3-F	0.3876	0.4047	0.4093	0.6541	0.7113	0.7666	**0.8226****	**0.7483****	**0.5915****	5278
GBDT-C3-C	0.4191	0.4357	0.4405	0.6535	0.7100	0.7631	0.7878*	0.7245**	0.5781**	1760
GBDT-C3-E	0.4264	0.4419	0.4466	0.6721	0.7180	0.7703	0.7942**	0.7241**	0.5778**	1535
GBDT-C3-F	0.4178	0.4350	0.4395	0.6554	0.7163	0.7672	0.7866	0.7310**	0.5819**	4953
GBRT-C3-C	0.4025	0.4203	0.4254	0.6304	0.6931	0.7488	0.7743	0.7168	0.5737**	1760
GBRT-C3-E	0.4100	0.4260	0.4313	0.6380	0.6867	0.7431	0.7697	0.7009	0.5637**	1535
GBRT-C3-F	0.4158	0.4332	0.4378	0.6479	0.7094	0.7612	0.7862	0.7294**	0.5802**	4949
LambdaMART-C3-C	0.4163	0.4332	0.4379	0.6577	0.7145	0.7673	0.7994**	0.7328**	0.5820**	1760
LambdaMART-C3-E	0.4183	0.4346	0.4394	0.6629	0.7133	0.7671	0.7968**	0.7268**	0.5786**	1535
LambdaMART-C3-F	**0.4353**	**0.4513**	**0.4557**	**0.6847**	**0.7354**	**0.7851**	0.8060**	0.7379**	0.5847**	4929

[a] All -C models set λ values $\langle 100000, 30000, 500\rangle$, -E models use $\langle 8000, 8000, 3000\rangle$, and -F models use $\langle 5000, 800, 300\rangle$.

Table 5: Summary of all features used in this work.

Description	Unit Cost	# Features
Pre-Retrieval Features		
Query Dependent (Unigram)	1	159
Query Dependent (Bigram)	100	147
Document Dependent Features		
Stage 0 Score	1	1
Static Document Priors	500	9
Score (Unigram)	2,000	107
Score (Bigram)	8,000	2
Total		425

the query and the document; or, the document. Table 5 shows a summary of these features. The majority of these features were derived from prior work within the LTR literature [15, 22, 23].

LTR Features. For all experiments, with GOV2, a total of 425 features were used. For each feature, several timing experiments were ran to compute the *relative* feature costs. We then normalized the costs based on the cheapest and most expensive features used in the experiments. Table 5 shows the complete feature breakdown based on the two main categories of features used.

The first set of features are a large collection of pre-retrieval features commonly used for predicting query difficulty [8], and more recently within LTR [21, 34] were gathered. These features draw on statistical information contained within the query alone or on simple scoring methods that require postings list access. As such they are reasonably efficient to compute on-the-fly at query time. The most important point about these features is that they are *query specific*, but must only be computed once using pre-computed unigram scores. This makes it relatively difficult to properly account for their true costs as LTR systems use SVM formatted input files, which implicitly have a *per document* feature cost in the model. Therefore, we divide these one-off pre-retrieval feature costs by the number of documents produced in the initial retrieval stage, resulting in an amortized unit cost of 1. All other costs are computed relative to this cost.

The second set of features are *per document* costs. While the cost of a single Document Prior lookup is very fast in practice, it must be done for every document in the current stage, and therefore more expensive than the one-off cost of the aggregate pre-retrieval feature scores. Likewise, all models incorporating bigrams are more expensive than their unigram counterparts. The bigram costs include a one-off cost to generate an ephemeral posting for the bigram [16], that can be reused to compute all of the bigram pre-retrieval features, and also used on the fly for per document bigram scoring. This amortized cost is reflected in the final unit costs used for our experiments. Alternative indexing approaches [13, 27] have been proposed to improve the efficiency of n-gram scoring in recent years, but feature-specific performance enhancements are beyond the scope of this work.

Table 4: Main results on the GOV2 collection using 5-fold cross validation. For the proposed cascade models, significant improvements over QL-BL/WLM-BL are indicated by $^{*}/^{\dagger}$ for $p < 0.05$ ($^{**}/^{\ddagger}$ for $p < 0.01$) in a paired t-test.

	ERR@k			NDCG@k			P@k			
System	@5	@10	@20	@5	@10	@20	@5	@10	@20	Cost
Baseline Bag-of-Words and Term Dependency Models										
QL-BL	0.3937	0.4131	0.4218	0.3839	0.3826	0.3950	0.5275	0.5074	0.5000	–
BM25-BL	0.3781	0.3980	0.4062	0.3796	0.3806	0.3814	0.5114	0.4893	0.4705	–
SDM-BL	0.4453	0.4632	0.4702	0.4396	0.4346	0.4345	0.6013	0.5711	0.5443	(High)
Ground Truth LTR Models										
GBDT-BL	0.4361	0.4590	0.4652	0.4441	0.4473	0.4411	0.6255	0.6027	0.5487	213683
GBRT-BL	0.4501	0.4678	0.4745	0.4546	0.4446	0.4345	0.6161	0.5805	0.5305	211640
LambdaMART-BL	**0.4590**	**0.4802**	**0.4849**	**0.4684**	**0.4692**	**0.4593**	**0.6470**	**0.6215**	**0.5641**	213482
Cascade Models (cost ≤ 5000)										
WLM-BL	0.4221	0.4422	0.4485	0.4204	0.4177	0.4132	0.5919*	0.5664**	0.5242	1249
LM-C3-C	0.4297*	0.4454*	0.4537*	**0.4453****	0.4328**	0.4314	0.5933*	0.5624	0.5312	4013
LM-C3-E	0.4298**	0.4465**	0.4545**	0.4418**	0.4315**	0.4294**	0.5946**	0.5624**	0.5285*	11
LM-C3-F	0.4366*	0.4537*	0.4608*	0.4435**	0.4440**	**0.4509****‡	0.6161**	0.5779**	0.5601**‡	4717
Cascade Models (cost ∼ 1/2 full model cost)										
LM-C3-F	0.4332*	0.4508*	0.4566*	0.4419**	0.4452**	0.4442**‡	0.6174**	0.5872**	0.5517**†	145693
LambdaMART-C3-F [a]	**0.4396***	**0.4578***	**0.4647***	0.4373*	0.4333**	0.4208	0.6094**	0.5732**	0.5181	129529
LM-C3-F, adaptive [b]	0.4295*	0.4469*	0.4530*	0.4435**	**0.4492****†	0.4501**‡	**0.6242****	**0.5926****	**0.5611****‡	110473

[a] With 650 trees and 32 nodes [b] With λ values $\langle 800, 0.1, 0.05\rangle$ and cutoffs $\langle 2500, 700\rangle$

Due to space constraints, we cannot describe all of the features or costs. A detailed description for all of the features as well as how costs (both estimated and real) can be found in the GitHub repository for the paper. The main point we want to make is that Table 5 provides realistic relative costs for both one-off and per-document features, and takes into account the relative complexity of each.

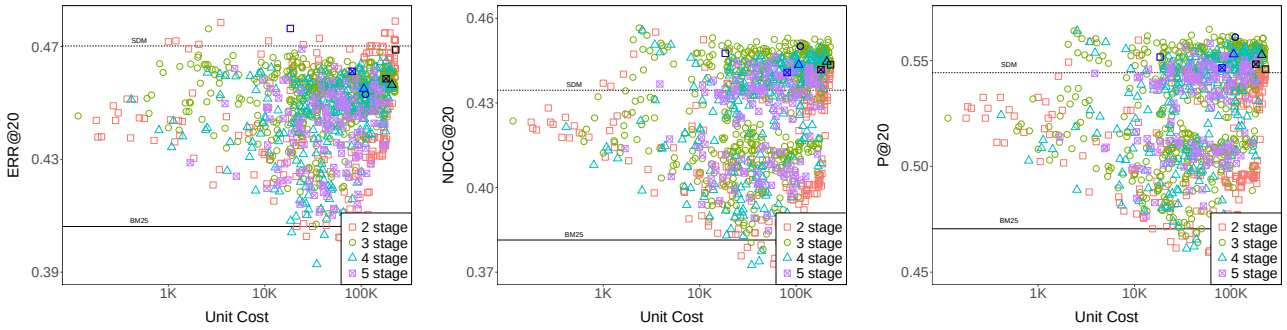
Main Results. In our initial investigation, the cascade is configured to 3 stages with cutoffs $\langle 1000, 100\rangle$. In this basic configuration, the cutoff thresholds are selected from widely used cutoff values in adhoc retrieval experiments. The experiment is conducted in a 5-fold cross-validated setting, so fixing the cascade configuration can considerably speed up the search, with the caveat of achieving limited improvement on retrieval effectiveness. This issue is briefly investigated in this experiment by including a run that also optimizes the cutoff thresholds. In Sec 4.3, we explore various cascade configurations and investigate the effect of cascade parameters on retrieval effectiveness.

Using a randomized search-based approach, the cascade models LM-C3-C, LM-C3-E, and LM-C3-F are selected by maximizing the unbounded NDCG score on the validation folds. In a 5-fold cross validated setting, this metric is averaged across 5 folds on the respective validation sets. The unbounded NDCG is not specific to any cutoff threshold, so essentially it can be used to optimize any cascade stage. Similarly behaved recall-oriented metrics (such as Mean Average Precision) could also be used.

The main results for GOV2 are presented in Table 4. In contrast to the Y!S1 collection in the previous experiment, more than 72% of the features used on GOV2 are query dependent pre-retrieval features. The presence of query specific features poses a serious challenge to all cascade models. Query features are usually cheaper to compute, and more likely to be selected (by cost-biased strategy, for example) in early cascade stages. GBDT and LambdaMART can effectively use these query features, but for other ranking models the query features are not as useful. As a result, cascading models that do not effectively utilize query features often see reduced effectiveness in the early cascade stages.

Ground truth models, as expected, give the best effectiveness among all baselines but also incurred the most feature extraction cost, around 210,000–220,000 unit cost. When compared with GBDT-BL and GBRT-BL, LambdaMART-BL achieves the best effectiveness. The cascade baseline WLM-BL spends far less on feature extraction, requiring only 0.58% of the full model cost, but at the cost of effectiveness.

A range of cascade models that spend less than 1/20 of the full model cost are first selected using the LM-C3-C, LM-C3-E, and LM-C3-F approaches. All three linear cascades outperform the WLM-BL baseline in nearly all metrics with the exception of P@10. Compared to WLM-BL, LM-C3-F significantly improves NDCG@20 by 0.037, and P@20 by 0.035, spending three times more on feature extraction. All three selected models behave differently than in the previous experiment. Both LM-C3-F and LM-C3-C are of comparable costs roughly in the range of 4,000–4,800. LM-C3-E tends to select



(a) ERR@20 (b) NDCG@20 (c) P@20

Figure 3: Effectiveness versus Cascade Cost in the GOV2 collection using the LM-C*-F models. The solid line at the bottom represents the effectiveness of a BM25 BOW run, the dotted line is a Sequential Dependency Model run which represents a competitive baseline on the collection, and the dots represent different LTR configurations and their respective trade-offs. Based on the validation data, the highlighted dots in black signify the most effective runs overall, while the highlighted dots in blue are the best cost-effective runs.

extremely compact feature sets and results in a greatly reduced cascade model that uses only the cheapest features. In general, the order of the three models in terms of effectiveness (in descending order) is LM-C3-F, LM-C3-E, and LM-C3-C, despite the fact that LM-C3-E actually costs much less than LM-C3-C. Other cascading models which require 1/2 of the full model cost are also shown in the table. None of the configurations from LM-C3-C and LM-C3-E fall into this range. The best-scoring LM-C3-F model (in terms of validation set NDCG) achieves comparable performance to the same model selected in the previous group, but requires much more feature extraction resources. A LambdaMART-C3-F model trained by fitting the selected feature sets in LM-C3-F does slightly better on ERR@k but sees degraded performance on NDCG@k and P@k. Note that, unlike the Y!S1 experiment, the parameters used in training LambdaMART-BL do not generalize over LambdaMART-C3-F. Another round of randomized search is needed to find the configuration that maximizes the tradeoff.

Finally, a LM-C3-F run that simultaneously optimizes the λ values and cutoff thresholds is also presented. This model is generally the most effective cascade model in terms of NDCG@k and P@k. It significantly outperforms the WLM-BL model on P@10 by 0.03, on P@20 by 0.035, and on NDCG@20 by 0.035. This result suggests that jointly optimizing feature allocation and cascade configuration can lead to further improvements. This issue is investigated further in the next experiment.

4.3 Effect of Cascade Configuration

In the third experiment, we investigate the effect of cascade configurations on retrieval effectiveness and cascade cost. We relax two variables that were held fixed in the previous experiments – the number of cascade stages, K, and the cutoffs $\langle c_1, c_2, \ldots, c_K \rangle$ – and jointly optimize these parameters together with λ values in a combined random search-based framework. As these configurations are more expensive to tune, the exploration was deferred until the influence of other variables was better understood. This experiment was carried out using the GOV2 collection. Our exploration starts by executing a full-range randomized search over the entire cascade design space.

We used predefined grids of each variable to ensure that the explored data points were not too densely packed [10]. We then iterated from $K = 2$ to 5, which indicates the number of cascade stages, and for each setting of K, sampled a set of feasible cutoffs and λ values from the aforementioned settings. In the experiment, each setting of K produced more than 200 configurations.

The effectiveness versus cascade cost for each explored combination were then plotted and shown in Figure 3, in which points from different settings of K are plotted in different colors and shapes. For each K setting, the best configuration (with cost $<$ 1/2 full model cost) found by using NDCG validation is plotted as a black dot. These "best" configurations are summarized in Table 6.

Figure 3 shows that a wide range of low cost but effective models can be found regardless of the choice of K. For ERR@20, two stage cascades are often quite effective, but can also be among the most expensive. For NDCG@20 and P@20, three stage cascades consistently provided the most effective configurations. Among all settings of K, three level cascades consistently provided the best trade-off between effectiveness and efficiency. We intend to investigate these trade-offs further in future work.

5 CONCLUSION

In this work, we have presented a new approach to cascaded ranking which can be used with any commonly used LTR algorithms. We make direct comparisons to several state-of-the-art approaches, and conclusively show that our approach can consistently achieve better trade-offs than other cascade ranking systems such as WLM-BL. In the experiments, we have presented several effective feature allocation strategies that have not previously been explored, and are the first to directly explore the relationship between the number of cascades stages and document sample sizes on performance trade-offs.

[10] The range of λ searched was {0.01, 0.03, 0.05, 0.08, 0.1, 0.3, 0.5, 0.8, 1, 3, 5, 8, 10, 30, 50, 80, 100, 300, 500, 800 }; the range of the cutoff threshold is the union of the three sets: {20, 30, …, 100}, {100, 200, …, 1000}, and {2000, 2500, 3000, …, 5000}.

Table 6: The best configuration for the K-stage LM-C3-F cascade (for $K = 2, 3, 4, 5$) found by maximizing the unbounded NDCG on the validation data. Significant improvements over WLM-BL are indicated by */** for $p < 0.05/p < 0.01$ in a paired t-test.

System (λ values; cutoffs)	NDCG@k			P@k			Cost
	@5	@10	@20	@5	@10	@20	
WLM-BL	0.4204	0.4177	0.4132	0.5919	0.5664	0.5242	1249
$\langle 800, 0.01\rangle; \langle 400\rangle$	**0.4529**	**0.4511***	0.4476**	0.6094	0.5866	0.5517*	18297
$\langle 800, 0.1, 0.05\rangle; \langle 2500, 700\rangle$	0.4435	0.4492*	**0.4501****	**0.6242**	**0.5926**	**0.5611***	110472
$\langle 500, 10, 0.03, 0.01\rangle; \langle 3000, 2000, 700\rangle$	0.4446	0.4446	0.4435**	0.6161	0.5913	0.5530*	106465
$\langle 800, 0.5, 0.1, 0.08, 0.05\rangle; \langle 2000, 800, 500, 80\rangle$	0.4343	0.4340	0.4408*	0.6040	0.5691	0.5466	80612

In future work we wish to more closely explore the relationship between feature costs and feature importance weighting at different levels of the cascade. Our current approach to parameter selection is largely empirical, and is quite costly when using hundreds of features and large scale document collections, resulting in several strong Linear Models, which are currently needed before generalizing to gradient boosted tree models. Therefore, an appealing next step in this work is to find more principled approaches to dynamically select the best cascade configuration on a per-query basis, and to further explore the best configurations for a wider variety of LTR ranking algorithms

Funding Statement. This work was supported by the Australian Research Council's *Discovery Projects* Scheme (DP140101587 and DP170102231).

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.

[2] N. Asadi and J. Lin. 2013. Training efficient tree-based models for document ranking. In *Proc. ECIR*. 146–157.

[3] N. Asadi, J. Lin, and A. P. De Vries. 2014. Runtime optimizations for tree-based machine learning models. *Trans. on Know. and Data Eng.* 26, 9 (2014), 2281–2292.

[4] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, Feb (2012), 281–305.

[5] C. Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.

[6] 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.

[7] 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.

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

[9] O. Chapelle and Y. Chang. 2011. Yahoo! Learning to Rank Challenge Overview. 14 (2011), 1–24.

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

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

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

[13] S. Huston, J. S. Culpepper, and W. B. Croft. 2014. Indexing Word-Sequences for Ranked Retrieval. *ACM Trans. Information Systems* 32, 1 (2014), 3.1–3.26.

[14] 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.

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

[16] X. Lu, A. Moffat, and J. S. Culpepper. 2015. On the Cost of Extracting Proximity Features for Term-Dependency Models. In *Proc. CIKM*. 293–302.

[17] X. Lu, A. Moffat, and J. S. Culpepper. 2016. Efficient and Effective Higher Order Proximity Modeling. In *Proc. ICTIR*. 21–30.

[18] 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.

[19] 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.

[20] 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.

[21] C. Macdonald, R. L. T. Santos, and I. Ounis. 2012. On the Usefulness of Query Features for Learning to Rank. In *Proc. CIKM*. 2559–2562.

[22] 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.

[23] 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.

[24] D. Metzler and W. B. Croft. 2005. A Markov random field model for term dependencies.. In *Proc. SIGIR*. 472–479.

[25] A. Mohan, Z. Chen, and K. Q. Weinberger. 2011. Web-Search Ranking with Initialized Gradient Boosted Regression Trees. *Journal of Machine Learning Research* 14 (2011), 77–89.

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

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

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

[29] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. 1994. Okapi at TREC-3.. In *Proc. TREC-3*.

[30] N. Tax, S. Bockting, and D. Hiemstra. 2015. A cross-benchmark comparison of 87 learning to rank methods. *Inf. Proc. & Man.* 51, 6 (2015), 757–772.

[31] R. Tibshirani. 1994. Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society, Series B* 58 (1994), 267–288.

[32] A. Trotman, C. L. A. Clarke, I. Ounis, J. S. Culpepper, M.-A. Cartright, and S. Geva. 2012. Open source information retrieval: a report on the SIGIR 2012 workshop. *SIGIR Forum* 46, 2 (2012), 95–101.

[33] Y. Tsuruoka, J. Tsujii, and S. Ananiadou. 2009. Stochastic Gradient Descent Training for L1-regularized Log-linear Models with Cumulative Penalty. In *Proc. ACL*. 477–485.

[34] S. Tyree, K. Q. Weinberger, K. Agrawal, and J. Paykin. 2011. Parallel Boosted Regression Trees for Web Search Ranking. In *Proc. WWW*. 387–396.

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

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

[37] L. Wang, J. Lin, D. Metzler, and J. Han. 2014. Learning to efficiently rank on big data. In *Proc. WWW (Companion Volume)*. 209–210.

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

[39] Z. Xu, M. J. Kusner, K. Q. Weinberger, M. Chen, and O. Chapelle. 2014. Classifier Cascades and Trees for Minimizing Feature Evaluation Cost. *Journal of Machine Learning Research* 15 (2014), 2113–2144.

[40] C. Zhai and J. Lafferty. 2004. A Study of Smoothing Methods for Language Models Applied to Information Retrieval. *ACM Trans. Information Systems* 22, 2 (April 2004), 179–214.