Comparing Pointwise and Listwise Objective Functions for Random-Forest-Based Learning-to-Rank

MUHAMMAD IBRAHIM and MARK CARMAN, Monash University, Australia

Current random-forest (RF)-based learning-to-rank (LtR) algorithms use a classification or regression framework to solve the ranking problem in a pointwise manner. The success of this simple yet effective approach coupled with the inherent parallelizability of the learning algorithm makes it a strong candidate for widespread adoption. In this article, we aim to better understand the effectiveness of RF-based rank-learning algorithms with a focus on the comparison between pointwise and listwise approaches.

We introduce what we believe to be the first listwise version of an RF-based LtR algorithm. The algorithm directly optimizes an information retrieval metric of choice (in our case, NDCG) in a greedy manner. Direct optimization of the listwise objective functions is computationally prohibitive for most learning algorithms, but possible in RF since each tree maximizes the objective in a coordinate-wise fashion. Computational complexity of the listwise approach is higher than the pointwise counterpart; hence for larger datasets, we design a hybrid algorithm that combines a listwise objective in the early stages of tree construction and a pointwise objective in the latter stages. We also study the effect of the discount function of NDCG on the listwise algorithm.

Experimental results on several publicly available LtR datasets reveal that the listwise/hybrid algorithm outperforms the pointwise approach on the majority (but not all) of the datasets. We then investigate several aspects of the two algorithms to better understand the inevitable performance tradeoffs. The aspects include examining an RF-based unsupervised LtR algorithm and comparing individual tree strength. Finally, we compare the the investigated RF-based algorithms with several other LtR algorithms.

Categories and Subject Descriptors: H.3.3. [**Information Search and Retrieval**]: Retrieval Models; H.3.4 [**Systems and Software**]: Performance Evaluation (Efficiency and Effectiveness)

General Terms: Algorithms, Design, Experimentation, Performance

Additional Key Words and Phrases: Learning-to-rank, random forest, objective function, splitting criterion, computational complexity, evaluation metrics

ACM Reference Format:
Muhammad Ibrahim and Mark Carman. 2016. Comparing pointwise and listwise objective functions for random-forest-based learning-to-rank. ACM Trans. Inf. Syst. 34, 4, Article 20 (August 2016), 38 pages.
DOI: http://dx.doi.org/10.1145/2866571

1. INTRODUCTION

When a user submits a query, the task of an information retrieval (IR) system is to return a list of documents ordered by the predicted (absolute or marginal) relevance to that query. Traditionally, different scoring methods, ranging from heuristic models (e.g., cosine similarity over tf-idf vectors) to probabilistic models (e.g., BM25, Language Models), have been used for this task. Recently researchers have investigated supervised

Mark Carman acknowledges research funding from NICTA (National ICT Australia).
Authors' addresses: M. Ibrahim, Faculty of Information Technology, VIC 3800, Monash University Australia; email: muhammad.ibrahim@monash.edu; M. Carman, Faculty of Information Technology, VIC 3800, Monash University Australia; email: mark.carman@monash.edu.

DOI: http://dx.doi.org/10.1145/2866571

machine-learning techniques for solving this problem. In this setting, a training example is a query-document pair, the corresponding label is the relevance judgement, and the features are measurements of various base rankers (e.g., cosine similarity)—this is called the learning-to-rank (LtR) problem. Many supervised learning methods have been developed and have found empirical success over conventional methods [Liu 2011].

The LtR algorithms can be broadly categorized into three groups as follows: (1) pointwise: these algorithms minimize a loss function over the training set that assigns a loss to each individual feature vector (and thus treats each feature vector independently of one another); (2) pairwise: here the loss function is calculated over every pair of feature vectors pertaining to the same query; and (3) listwise: here the loss function treats each query (and its associated documents) as a single instance and accordingly computes a single loss for all the documents pertaining to the same query. Details of these approaches can be found in Liu [2011] and Li [2011].

Random forest [Breiman 2001] is a simple but effective learning algorithm that aggregates the outputs of a large number of independent and variant base learners, usually decision trees. Its major benefits over other state-of-the-art methods include inherent parallelizability, ease of tuning, and competitive performance. These benefits attract researchers of various disciplines such as bioinformatics [Qi 2012], computer vision [Criminisi and Shotton 2013], and so forth, where random forest is a very popular choice. But for the LtR task, random forest has not been thoroughly investigated.

For the LtR problem, listwise algorithms have been shown empirically [Li 2011; Liu 2011] to outperform pointwise algorithms in general. Their generalization ability has also been proved [Lan et al. 2009]. However, the listwise algorithms are usually computationally more demanding, and they oftentimes lack conceptual simplicity.

While it is relatively straightforward to design an RF-based pointwise algorithm, we found no existing work on its listwise counterpart. The main barrier to designing a listwise algorithm is that most of the IR metrics are nonsmooth and nonconvex with respect to model parameters, which makes them difficult to directly optimize using many learning algorithms. However, decision-tree-based algorithms optimize an objective function by splitting the data in a coordinatewise fashion (by enumerating all possible cut points along a subset of the dimensions). Thus, it is possible to apply these techniques directly to the optimization of nonconvex and even nonsmooth objective functions (such as those that directly optimize IR metrics like NDCG). RF-based pointwise algorithms have already been shown to be competitive with the state-of-the-art methods [Geurts and Louppe 2011; Mohan et al. 2011]. In this article, we extensively compare the random-forest-based pointwise and listwise LtR algorithms.

The major contributions of this article are as follows:

—We design a random-forest-based listwise LtR algorithm that directly optimizes any rank-based metric—we use NDCG and some of its variations.
—We design a random-forest-based hybrid algorithm that can incorporate different splitting criteria in a single tree. This is especially useful when some splitting criteria are computationally more complex (e.g., NDCG-based ones) than others (e.g., entropy-based ones) because the amount of computationally complex splits can be controlled depending on the availability of computational resources. We empirically show that its performance is at least as good as its pointwise counterpart.
—Our experiments reveal the following:
  (1) For smaller and moderate LtR datasets (in terms of number of queries) with graded relevance labels, the performance of the listwise approaches is better than their pointwise counterparts.
  (2) For large datasets, performances of RF-based pointwise and listwise/hybrid algorithms seem to be dependant on the properties of the dataset.

—We compare the relative performance difference of pointwise and listwise algorithms by investigating the predictive accuracy (also known as strength) of individual trees. This reveals an aspect that is common to both objective functions: both of them tend to isolate similar training instances in the leaves of a tree. Hence, when a tree is sufficiently deep, the difference between the two objective functions diminishes, thereby yielding similar performance in some cases.
—We employ a purely random (unsupervised partitioning)-tree-based ensemble LtR algorithm. While on big datasets its performance is not on par with random-forest-based pointwise/listwise algorithms, on smaller datasets it performs close to RF-point. Importantly, it is better than some state-of-the-art algorithms, namely, AdaRank [Xu and Li 2007], RankSVM [Joachims 2002], RankBoost [Freund et al. 2003], and sometimes Coordinate Ascent [Metzler and Croft 2007]. Given the non-adaptiveness of its objective function, this finding is interesting. This indicates that the nearest-neighbor-based methods may be effective in LtR. Moreover, a completely random-tree-based ensemble has a huge computational advantage over many other LtR algorithms.
—Our experiments reveal that RF-based pointwise/listwise algorithms consistently win over RankSVM (pairwise), RankBoost (pairwise), Coordinate Ascent (listwise), and AdaRank (listwise), and are at least similar to Mart (pointwise) on big datasets. This shows that the RF-based LtR algorithms are indeed competitive to the state-of-the-art methods. Also, the learning curves show that with smaller training sets, RF-based LtR algorithms perform very well (outperform LambdaMart [Wu et al. 2010]), which makes them strong candidates for the domains where getting a large amount of labeled training data is difficult (e.g., domain-specific search and enterprise search).

The rest of the article is organized as follows. Section 2 discusses the random forest framework and the LtR problem formulation. Section 3 reviews some related methods. Section 4 explains an (existing) random-forest-based pointwise algorithm (RF-point) that will be used as the baseline, and then designs its listwise counterpart (RF-list). Here we also discuss the time complexities of the pointwise and listwise algorithms, which then drive us to develop a hybrid algorithm. Section 5 discusses the experimental results of RF-point and RF-list. Section 6 investigates the performance difference of pointwise and listwise objective functions from several perspectives. Although the main focus of this research is to compare RF-based LtR algorithms, in Section 7, we compare them with several state-of-the-art algorithms. Section 8 summarizes the article.

Table I lists the notations used in the article. If the meaning is obvious, we omit the subscripts/superscripts.

2. BACKGROUND

In this section, we describe the random forests and the learning-to-rank problem in more detail.

2.1. Random Forest

Random forest is an (usually bagged) ensemble of recursive partitions over the feature space, where each partition is assigned a particular class label (for classification) or real number (for regression), and where the partitions are randomly chosen from a distribution biased toward those giving the best prediction accuracy. For classification, the majority class within each partition in the training data is usually assigned to the partition, while for regression the average label is usually chosen so as to minimize zero-one loss and mean squared error, respectively.

In order to build the recursive partitions, a fixed-sized subset of the attributes is randomly chosen at each node of a tree; then for each attribute, the training data (at

Table I. Notation

Symbol	Description
N	# instances (query-doc pairs)
M	# features (base rankers)
C	# classes (relevance labels)
$\mathcal{D}$	A dataset
$q \in \mathcal{Q}$	A query q among a set of queries $\mathcal{Q}$
n_q	# docs for query q
$d_{q,i}$	ith document of query q ($1 \leq i \leq n_q$)
$l_{q,i}$	Relevance label for ith doc of query q
$\mathcal{Q}_\mathcal{D}$	Set of queries present in dataset $\mathcal{D}$
$\vec{x}_{q,i} = \vec{\psi}(q, d_{q,i})$	Feature vector corresponding to ith doc of query q
$\psi_i(.)$	ith base ranker ($1 \leq i \leq M$)
$f \in \mathcal{F}$	Retrieval function f among set of models $\mathcal{F}$
$s_f(q, d)$	Score assigned by f to document d for query q
$rank_f(d_{q,i})$	Rank position of doc $d_{q,i}$ in a list produced by f
$\mathcal{L}(\mathcal{D}; f)$	Loss function applied to dataset $\mathcal{D}$ for function f
E	Size of the ensemble (# trees)
K	# features randomly chosen at a node
n_{min}	Minimum # instances needed to split a node
b	# queries (and associated docs) in a subsample per tree
$Leaves$	List of leaves of a tree

ALGORITHM 1: Procedure for Constructing a Random Forest with Query-Level Subsampling

Procedure: *BuildForest($\mathcal{D}$, E, b)*
Data: Training set $\mathcal{D}$, ensemble size E, number of queries per subsample b
Result: Tree ensemble *Trees*
begin
  $Trees \leftarrow \emptyset$;
  for $i \in \{1, \ldots, E\}$ **do**
    $\mathcal{D}_i \leftarrow \emptyset$;
    while $|\mathcal{Q}_{\mathcal{D}_i}| < b$ **do**
      $q \leftarrow chooseRandom(\mathcal{Q}_\mathcal{D} \setminus \mathcal{Q}_{\mathcal{D}_i})$;
      $\mathcal{D}_i.add(\langle \vec{x}_{q,j}, l_{q,j} \rangle_{j=1}^{n_q})$;
    end
    $Trees.add(BuildTree(\mathcal{D}_i))$;
  end
  return *Trees*;
end

Where the function $chooseRandom(A)$ selects an item uniformly at random from the set A.

that node) is sorted according to the attribute, and a list of potential split points (midpoints between consecutive training data points) is enumerated to find the split that minimizes the expected loss over the child nodes. Finally, the attribute and associated split point with the minimal loss is chosen and the process is repeated recursively down the tree. For classification, the entropy or gini function[1] is used to calculate the loss for each split, while for regression, the mean squared error is used. Algorithms 1 and 2

[1]If $p(c|leaf)$ denotes the estimated probability of a class at a leaf, then $\mathcal{L}_{entropy}(leaf) = -\sum_c p(c|leaf)log(p(c|leaf))$ and $\mathcal{L}_{gini}(leaf) = \sum_c p(c|leaf)(1 - p(c|leaf))$.

ALGORITHM 2: Generic Tree Building Procedure for a Random Forest

Procedure: $BuildTree(\mathcal{D})$
Data: Dataset $\mathcal{D}$
Result: A tree *root*
begin
  $root \leftarrow createNode(\mathcal{D})$;
  $nodeList \leftarrow \{root\}$;
  while $|nodeList| > 0$ **do**
    $node \leftarrow nodeList[1]$;
    $nodeList \leftarrow nodeList \setminus node$;
    if $|\mathcal{D}_{node}| > 1$ **then**
      $best \leftarrow \langle 0, null, null \rangle$;
      for $feature \in randomSubset(\{1, .., M\}, logM)$ **do**
        for $split \in midpoints(sort(\mathcal{D}_{node}, feature))$ **do**
          $\mathcal{D}_{left} \leftarrow \{\vec{x} \in \mathcal{D}_{node} | x_{feature} < split\}$;
          $\mathcal{D}_{right} \leftarrow \{\vec{x} \in \mathcal{D}_{node} | x_{feature} \geq split\}$;
          $gain \leftarrow Gain(\mathcal{D}_{node}, \mathcal{D}_{left}, \mathcal{D}_{right})$;
          if $gain > best.gain$ **then**
            $best \leftarrow \langle gain, feature, split \rangle$;
          end
        end
      end
      if $best.gain > 0$ **then**
        $leftChild \leftarrow createNode(\vec{x} \in \mathcal{D}_{node} | x_{best.feature} < best.split)$;
        $rightChild \leftarrow createNode(\vec{x} \in \mathcal{D}_{node} | x_{best.feature} \geq best.split)$;
        $nodeList.add(\{node.leftChild, node.rightChild\})$;
      end
    end
  end
  return *root*;
end

The variation in the *Gain(.)* routine results in different algorithms (eg. pointwise, listwise etc.

show the pseudo-codes for building a forest and a tree, respectively. The subsampling performed in lines 4 through 7 of Algorithm 1 will be explained in Section 4.1.2. The variation in the generic *Gain* routine mentioned in line 9 of Algorithm 2 results in different algorithms (e.g., pointwise, listwise, etc.), which will be discussed in detail as we introduce different algorithms in Section 4.

2.2. Learning to Rank

Suppose we have a set of queries $\mathcal{Q} = \{q_i\}_{i=1}^{|\mathcal{Q}|}$, where each query q is associated with a set of n_q documents $\mathcal{D}_q = \{d_{q,i}\}_{i=1}^{n_q}$. Each query-document pair $\langle q,d_{q,j} \rangle$ is further associated with a feature vector $\vec{x}_{q,j} = \vec{\psi}(q, d_{q,j}) = \langle \psi_1(q, d_{q,j}), \ldots, \psi_M(q, d_{q,j}) \rangle \in \mathbb{R}^M$ and a relevance label $l_{q,j}$, such that a dataset $\mathcal{D}$ consists of $N = \sum_i n_{q_i}$ feature vectors and corresponding labels. The elements of the feature vector (i.e., $\psi_i(.)$) are scores of other simple rankers, for example, the cosine similarity over tf-idf vectors for the particular query-document pair, the BM25 score for the same pair, and so forth. These scores are usually normalized at the query level, such that each feature value lies in the range 0 to 1 [Qin et al. 2010b].

A ranking function is then a function $f : \mathbb{R}^M \rightarrow \mathbb{R}$ that assigns scores to feature vectors, that is, query-document pairs. Given a query q, let $\vec{s}_f(q) = \langle f(\vec{x}_{q,1}), \ldots, f(\vec{x}_{q,n_q}) \rangle$ denote

the vector of scores assigned by f to the documents for that query. The vector can then be sorted in decreasing order to produce a ranking:

$$sort(\vec{s}_f(q)) = \langle rank_f(d_{q,1}, q), \ldots, rank_f(d_{q,n_q}, q)\rangle. \tag{1}$$

Here $rank_f(d_{q,i}, q)$ returns the position of document $d_{q,i}$ in a sorted list based on the scores generated by f.

Given a training set $\mathcal{D}$, the goal of an LtR algorithm is to find the function f among a set of functions $\mathcal{F}$ that minimizes a loss function $\mathcal{L}$ over the training set:

$$f^* = \arg\min_{f\in\mathcal{F}} \mathcal{L}(\mathcal{D}; f). \tag{2}$$

Ideally, the loss function should optimize the metric that will be used for evaluation. As such, a typical example of a loss function is one based on Normalized Discounted Cumulative Gain (NDCG):

$$\mathcal{L}(\mathcal{D}; f) \;=\; 1 - \overline{NDCG}(Q; f) \tag{3}$$

$$\overline{NDCG}(Q; f) \;=\; \frac{1}{|Q|}\sum_{q\in Q} NDCG(q; f). \tag{4}$$

The NDCG for a particular query is defined as the normalized version of the Discounted Cumulative Gain (DCG), as follows:

$$DCG(q; f) \;=\; \sum_{r=1}^{n_q} \frac{gain(l_{q,doc(r,q;f)})}{discount(r)} \tag{5}$$

$$NDCG(q; f) \;=\; \frac{DCG(q; f)}{max_{f'} DCG(q; f')}, \tag{6}$$

where $doc(r, q; f) = rank_f^{-1}(r, q)$ denotes the inverse rank function, that is, the document at rank r in the list retrieved by model f, and thus $l_{q,doc(r,q;f)}$ denotes the relevance judgement for the document in position r for query q.

Sometimes NDCG is truncated at a value $k \leq n_q$ and is denoted by NDCG@k. Throughout the article, we shall use NDCG to refer to the untruncated version.

The gain and discount factors used to weight rank position r are usually defined as

$$gain(l) \;=\; 2^l - 1 \tag{7}$$

$$discount(r) \;=\; \log(r + 1). \tag{8}$$

We note that LtR algorithms mainly differ in defining the loss function $\mathcal{L}(\mathcal{D}; f)$. Equation (3) and other ranking loss functions based on IR metrics are difficult to optimize directly because (1) they are nondifferentiable with respect to the parameters of the retrieval function (due to the sorting in Equation (1)) and (2) they are almost invariably nonsmooth and nonconvex. Hence, the researchers have attempted a range of heuristics to define $\mathcal{L}(\mathcal{D}; f)$, which gave rise to a large number of LtR algorithms.

3. RELATED WORK

Our work is related to two threads of research: rank-learning algorithms based on the random forest, and techniques for direct optimization of ranking measures. These are described next.

3.1. Learning to Rank Using Random Forest

Various machine-learning algorithms have been extensively used to solve the LtR problem, yet the application of random forests has thus far not been thoroughly investigated. Geurts and Louppe [2011] use a variation of random forest called Extremely Randomized Trees (ERT) [Geurts et al. 2006] with both classification and regression settings, that is, using a pointwise approach. Despite the fact that simple settings are used, they report competitive performance on the Yahoo dataset to other state-of-the-art algorithms. Mohan et al. [2011] also use a pointwise approach with the regression setting. The main difference between these two works is that the former one used a variation of original random forest, while the latter one used the original framework.

3.2. Direct Optimization of Ranking Measures

Several methods on direct optimization of ranking measures for rank learning have been proposed. Next we describe some of them.

Metzler and Croft [2007] use the coordinate ascent method to learn a ranking function that finds a local optimum of an objective function that directly uses an IR metric like NDCG or MAP. The coordinate ascent updates one parameter at a time while keeping other parameters fixed, thereby minimizing the scalability issue of conducting a grid search over a large parameter space. The authors report better results than a standard language model.

Wu et al. [2010] propose the LambdaMart algorithm, which blends the sophisticated idea of its predecessor, namely, LambdaRank [Quoc and Le 2007], with the gradient boosting framework [Friedman 2001]. The gradient boosting framework can be used to optimize any loss function for which a gradient is defined at each training point, and since LambdaRank uses the (approximated) gradient of the implicit loss function at each training point, these approximated gradients can be used in the gradient boosting framework (instead of the neural network framework, which was used in LambdaRank).

Taylor et al. [2008] address the issue of nonsmoothness of the IR metrics and point out that although the IR metrics like NDCG are either flat or discontinuous everywhere with respect to the model parameters, the scores are smooth with respect to the model parameters. They argue that if the individual ranks of the instances can be mapped into a smooth distribution instead of discrete ranks, then these ranks will be changed smoothly with changes of the model parameters. NDCG changes only if the ranks of the documents change, but if the ranks themselves are allowed to change smoothly with respect to the model parameters, that is, if noninteger ranks are used, then it is possible to devise a *soft* version of NDCG that changes smoothly with respect to the model parameters, thereby solving the fundamental problem of nonsmoothness of the IR metrics. The authors use the neural network framework to optimize the objective function. The algorithm is called SoftRank. A limitation of this algorithm is that it takes $\mathbf{O}(n_q^3)$ time, where n_q is the number of documents per query.

Xu and Li [2007] propose a conceptually simple but effective algorithm called AdaRank, which uses the AdaBoost framework [Freund and Schapire 1995] while maintaining a listwise approach. The AdaBoost framework assigns weights to the individual (weak) learners of an ensemble according to the classification error of the learner on weighted training instances. The authors modify the weighting scheme so that instead of classification error, ranking error of all documents associated with a query is used. Note that the weights are associated with queries rather than single documents, thereby retaining the query-document structure of the training data. This ensures that the later learners of the ensemble focus on the hard queries (and associated documents) to achieve an overall good NDCG. This way AdaRank produces weak

Table II. LtR Algorithms Using Different Approaches and Frameworks

Method	Pointwise	Pairwise	Listwise
Coordinate ascent	-	-	Metzler and Croft [2007], Tan et al. [2013]
Adaboost	-	Freund et al. [2003]	Xu and Li [2007]
Gradient boosting	Li et al. [2007]	-	Wu et al. [2010]
Neural net	-	Quoc and Le [2007]	Taylor et al. [2008]
SVM	Nallapati [2004]	Joachims [2002]	Yue et al. [2007]
Random forest	Geurts and Louppe [2011]	-	-

learners that are specifically built to deliver high NDCG, in contrast to delivering high classification accuracy.

Like Metzler and Croft [2007] and Tan et al. [2013] also use the coordinate ascent method in a listwise manner. The main difference between the two approaches is that the latter locates the optimal point along a particular coordinate (i.e., feature) using a sophisticated technique, while the former resorts to heuristics to find a good point.

Qin et al. [2010a] develop a general framework that first converts any rank-based metric into a document-based metric and then replaces the position function of the differences of scores between the said document and every other document of the list (which is an indicator function and hence noncontinuous and nondifferentiable) by the (smooth) logistic function. This surrogate objective function can then be optimized using any technique that is able to handle the local optima problem (the authors use the *random restart* method). A drawback of this framework is that the converted document-based metric needs $\mathbf{O}(n_q^2)$ time complexity, whereas original sorting could be done in $\mathbf{O}(n_q \log n_q)$ time. Another drawback is that they use only three small datasets, namely, Ohsumed, TD2003, and TD2004.

3.3. Motivation for Our Approach

From the previous discussion, we see that the objective functions of LtR algorithms that use boosting, neural network, and support vector machine have been well investigated in the literature. For example, with gradient boosting, Li et al. [2007] and Quoc and Le [2007] use pointwise and listwise algorithms, respectively. With AdaBoost, Freund et al. [2003] and Xu and Li [2007] use pairwise and listwise algorithms, respectively. With neural network, Cao et al. [2007] and Taylor et al. [2008] use listwise algorithms. With SVM, Nallapati [2004], Joachims [2002], and Yue et al. [2007] use pointwise, pairwise, and listwise algorithms, respectively. Table II summarizes some of the methods under different approaches.

In spite of achieving much success in other disciplines such as bioinformatics [Qi 2012], image processing [Criminisi and Shotton 2013], and spectral data processing [Ham et al. 2005], there is little work on using random forest for solving the LtR problem, and importantly, to the best of our knowledge, there is no work on using direct optimization of ranking measures. This work aims at filling this gap. In a recent competition arranged by Yahoo! [Chapelle and Chang 2011], a number of top performers used some form of randomized ensembles. That is why we think that more investigation into the utility of random-forest-based LtR algorithms is needed. In this article, we investigate its objective functions.

Since the NDCG function is both nonsmooth and nonconvex with respect to the model parameters, it is difficult to directly optimize the loss function of Equation (3). Instead, LtR practitioners normally use surrogate loss functions, for example, standard loss functions of the classification/regression problem (such as logistic loss, exponential loss, etc.) because perfect classification/regression leads to perfect ranking. In this work, we address the question: how does a random-forest-based LtR algorithm perform

with different objective functions? As such, we investigate the pointwise and listwise objective functions and analyze their similarities and differences.

4. APPROACH

In this section, we first describe some properties common to all of the algorithms studied in this article. We then discuss the existing RF-based pointwise algorithm, which will be used as the baseline. After that we develop its listwise counterpart.

4.1. Common Properties for All RF-Based Algorithms

We discuss three aspects as follows: (1) document score during training, (2) subsampling per tree, and (3) termination criteria.

4.1.1. Document Score During Test Phase. In order to assign a score to a document with respect to a given query, we use the expected relevance (employed by Li et al. [2007], Geurts and Louppe [2011], and Mohan et al. [2011], etc.) over the training data instances at that particular node of the tree. Specifically, every leaf of a tree (partition of the data space) is given a score as follows:

$$score(leaf) = \sum_{c=0}^{C-1} p(c|leaf) * c, \tag{9}$$

where $p(c|leaf)$ denotes the relative frequency of relevance level $c \in \{0, 1, \ldots, C-1\}$ given the $leaf$ of the tree.[2]

The score assigned to a test instance (e.g., jth document of query q) by an ensemble of E trees is calculated as

$$f(\vec{x}_{q,j}) = \frac{1}{E} \sum_{i=1}^{E} score_i(leaf), \tag{10}$$

where the $score_i(leaf)$ is the score of the leaf of the ith tree where the test instance has landed (here we have overridden the notation $socre(.)$ as compared to Equation (9)).

4.1.2. Subsampling Per Tree. As for choosing between document-level and query-level sampling, traditionally bootstrapped samples are used for learning a tree of a random forest. A bootstrapped sample is formed by randomly choosing instances *with* replacement from the original sample, and thus it has equal size to the original sample. In pointwise algorithms, sampling can be performed at the document (instance) level because it assigns a loss to individual query-document pairs, although query-level sampling is also applicable. For a listwise algorithm that aims at optimizing IR metrics such as NDCG directly, NDCGs of individual queries should be as reliable as possible. Hence, we need to sample the training data on a per-query basis, which means sampling a query along with *all* the documents associated with it.

As for choosing between sampling with replacement and without replacement, the usual practice among RF researchers is to perform sampling with replacement. However, Friedman [2002] and Friedman and Hall [2007] show that sampling without replacement also works well for classification and regression. Also, according to Ganjisaffar et al. [2011a], it works well for the LtR task. Moreover, it reduces the training time significantly, especially for large datasets. Recent works by Scornet et al. [2014] and Wager [2014] show that sampling without replacement allows for better theoretical analysis (which includes variance estimation) as compared to sampling

[2]The original equation proposed by Li et al. [2007] was $score(leaf) = \sum_{c=1}^{C} p(c|leaf) * g(c)$, where $g(.)$ is any monotonically increasing function of c. The authors (and later Mohan [2010]) empirically examine various $g(.)$ including c, c^2, and 2^c on several datasests such as Yahoo and conclude that $g(c) = c$ works the best.

with replacement. For these reasons, our settings regarding subsampling are (1) using subsampling (we note that some existing works such as Geurts and Louppe [2011] do not use any sampling at all, i.e., use the entire data per tree), (2) using query-level subsampling, and (3) using subsampling without replacement, which means 63% queries[3] (and all of their associated documents) are in a resultant sample. These settings are maintained unless otherwise noted (when dealing with large datasets, we modify the number of queries to sample as will be discussed in Section 5.2.1).

4.1.3. Termination Criteria. To stop further splitting of a node of a tree, the prevalent practice in RF literature is to grow a tree as deep as possible, thereby deliberately allowing the individual tree to overfit in an attempt to decrease the bias [Breiman 2001]. This is favorable to a random forest because the role of learning each individual tree is to sample from the space of possible predictors. The individual (high) variance of each tree does not deteriorate the performance of the ensemble because the prediction of the ensemble is the average of the predictions of individual trees, thereby reducing the variance of the ensemble. However, later studies such as Segal [2004] empirically discover for the regression problem that marginal improvement may be found by tuning the maximum node size parameter of a tree, and later works on theoretical analysis of RF such as Lin and Jeon [2006], Geurts et al. [2006], Scornet et al. [2014], and Scornet [2014] support this claim by arguing that as the data size grows, the maximum node size should also grow. However, for the sake of clarity and better readability, we do not investigate this aspect in this article (except a short discussion on overfitting in Section 6.2), thereby leaving it to future work. Throughout the article, we maintain the predominant practice of letting the trees grow as deep as possible. Thus, the condition for not splitting a node is if no gain can be found by considering all the split points of the randomly chosen K features. Note that this condition also encompasses the case when the node contains instances from only one label.

4.2. Random-Forest-Based Pointwise Algorithm

It has been shown that both classification and regression errors are upper bounds on the ranking error [Li et al. 2007; Cossock and Zhang 2006]. That is why existing pointwise algorithms for RF-based rank learning yield good performance in general. For our experiments, we choose an algorithm that can be interpreted as a blend of classification and regression settings—the classification setting (with entropy function as the objective) is used to identify the best partitions of the sample space (i.e., during the training phase), while the regression setting is used to assign scores to the final data partitions obtained (i.e., during the evaluation phase). We henceforth refer to this algorithm as *RF-point*.

ALGORITHM 3: Pointwise Gain (Using Entropy)

Procedure: $Gain_{entropy}(\mathcal{D}, \mathcal{D}_{left}, \mathcal{D}_{right})$
Result: Change in entropy resulting from split.
begin
  return $Entropy(\mathcal{D}) - (\frac{|\mathcal{D}_{left}|}{|\mathcal{D}|} Entropy(\mathcal{D}_{left}) + \frac{|\mathcal{D}_{right}|}{|\mathcal{D}|} Entropy(\mathcal{D}_{right}))$;
end

Where the entropy of data $\mathcal{D}$ over C class labels (with $\mathcal{D}_c \subseteq \mathcal{D}$ denoting examples of cth class) is computed as: $Entropy(\mathcal{D}) = -\sum_{c=0}^{C-1} \frac{|\mathcal{D}_c|}{|\mathcal{D}|} \log \frac{|\mathcal{D}_c|}{|\mathcal{D}|}$;

The *Gain* routine mentioned in Algorithm 2 is implemented according to Algorithm 3. We follow the formulation of gain given in a notable review by Criminisi [2011].

[3]63 is the average cardinality of a bootstrap sample (without replacement).

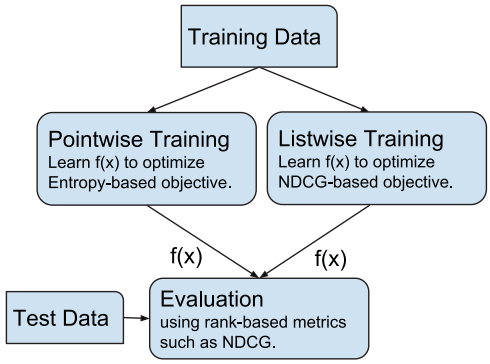

Fig. 1. Relationship between (existing) RF-point and (proposed) RF-list.

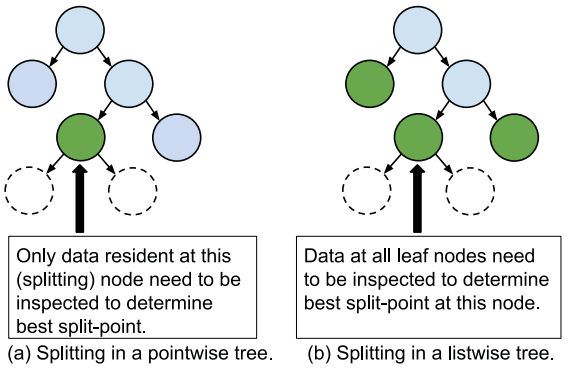

Fig. 2. Local decision in a pointwise tree versus global decision in a listwise tree.

4.3. Random-Forest-Based Listwise Algorithm

A problem of using a surrogate objective function instead of using a true one (e.g., Equation (3)) is that the learning algorithm may spend much effort to correctly classify some instances that have lower contribution in a graded[4] IR metric like NDCG, and even worse, perhaps at the cost of misclassifying some instances from the top of the current ranked list (which are important from the perspective of NDCG). Therefore, a natural choice would be to choose a split point that has a direct relationship with NDCG. This is the motivation behind developing the algorithm of this subsection. The goal is to learn individual trees in such a way that NDCG (or any other ranking metric) is maximized. The primary difference between the existing pointwise algorithm and the proposed listwise algorithm is pictorially explained in Figure 1. We henceforth call this algorithm *RF-list*. The main novelties of RF-list as compared to RF-point are (1) using a loss function that directly optimizes NDCG (i.e., not an approximation to it as adopted by some existing works) and (2) using it in a random forest framework.

In Figure 2, we illustrate the fact that in RF-point (the left figure), the splitting criterion is a local decision because the instances of a single node are involved in calculating the change in entropy that results from splitting the node into two children. But the RF-list (the right figure) involves data of all leaves of the current structure

[4]Graded means a decaying function is used as discount of each rank as the rank goes down to a list.

of a tree, that is, the entire training set. This approach is listwise because, unlike the pointwise approach, a loss is associated with an entire list of documents (pertaining to the same query).

In a nutshell, the basic idea of RF-list is to recursively partition the training data by choosing split points such that for a particular split, the training instances in the two resulting leaf nodes have such scores that maximize NDCG across all the queries of the training set.

The inevitable question arises here regarding the computational difficulty of directly optimizing NDCG or any other rank-based metric. The answer is, for a decision tree, it is not necessary for the objective function to be convex, because the learning algorithm attempts to optimize the objective in a greedy manner.

The subtleties of RF-list are described next.

4.3.1. Document Scores During Training. This issue is not present in RF-point because its splitting criterion is invariant to the predicted labels of partitions. However, in order to compute a listwise objective, RF-list needs to assign a score to each document during training. To assign a score to documents in a given partition during training, we use the *expected relevance* introduced in Equation (9). The motivation is that the more predictive the individual trees are, the better the NDCG.[5] (We cannot use Equation (10) because the trees of a random forest are learned independently from each other, which makes it completely parallelizable.)

4.3.2. Computing the Objective Function. We now explain as to how a split-point at a particular node of a tree is determined. Suppose we need to select a split point for a particular node where documents of m' number of queries (out of m) are present. For a potential split point, the scores of the documents directed to left and right children are computed according to the previous discussion (i.e., Section 4.3.1), and then the NDCG values of each of the m' queries should be updated according to the new scores of the documents. For the rest of the $m - m'$ queries, their current NDCGs will not change due to this split because the scores and ranks of their instance documents (sitting in **other** leaves of the tree) will not change. The split point that gives the maximum average NDCG across all queries should be selected as the best split point for the particular (parent) node in question. Thus, unlike traditional splitting criteria such as the entropy-based one where only the data of a particular node are used to decide the split point, when optimizing a listwise objective function in terms of ranks (such as NDCG), the data at all leaf nodes of the tree (i.e., the entire training data of the tree) must be considered.

Now the key question is, how can we calculate the NDCG of a query when many instances have the same score (since they fall into the same leaf of a particular tree)? While tied document scores rarely occur as the average score in an ensemble due to the fact that different trees split the data differently, they are common for a single tree. Tied scores are especially a problem in early nodes of a tree since a tree, at its root node, assigns the same score to all documents. Intuitively, a random order is not a solution. We, therefore, calculate the Expected NDCG over all possible orderings of the tied documents, which can be done efficiently as follows:

$$\mathbb{E}_{\vec{r} \in rankings(q;f)}[NDCG(q;f)] = \frac{\mathbb{E}_{\vec{r}}[DCG(q;f)]}{\max_{f'} DCG(q;f')}, \tag{11}$$

[5]Indeed, the two components that control the error rate of a random forest are (1) the correlation among the trees—the lower the better, and (2) the strengths of individual trees—the higher the better [Breiman 2001]. In our context, the better the NDCG predicted by a single tree, the higher the strength of that tree.

Table III. An Example of Expected NDCG Computation

Sorted Docs	**Assigned Scores**	**True Label**	**Gain** $(2^{l_{q,doc(r',q;f)}}-1)$	**Expected Gain** (cf. Equation (13))
d_3	1.7	0	0	1.33
d_5	1.7	2	3	1.33
d_7	1.7	1	1	1.33
d_2	0.9	1	1	0.5
d_6	0.9	0	0	0.5
d_1	0.4	1	1	1.0
d_4	0.4	1	1	1.0

where $rankings(q; f)$ denotes the set of equivalent rankings and $\vec{r}$ is one such ranking. Using Equation (5) and the linearity of Expectation, we can write the Expected DCG in terms of the expected gain at each rank position:

$$\mathbb{E}_{\vec{r}}[DCG(q; f)] = \sum_{r=1}^{n_q} \frac{\mathbb{E}_{r' \in ties(r;q,f)}[2^{l_{q,doc(r',q;f)}} - 1]}{log(r+1)}, \tag{12}$$

where $ties(r;q, f)$ denotes the set of ranks around rank r that have all been assigned the same score (by the tree f for query q). The expected gain at rank r can then be computed as

$$\mathop{\mathbb{E}}_{r' \in ties(r)}[2^{l_{q,doc(r',q;f)}}-1] = \frac{1}{\max[ties(r)]-\min[ties(r)]+1} \sum_{j=\min[ties(r)]}^{\max[ties(r)]} 2^{l_{q,doc(j,q;f)}}-1, \tag{13}$$

where $\min[ties(r)]$ and $\max[ties(r)]$ are the top and bottom tied ranks around rank r, respectively.

Now we illustrate the aforementioned concept with an example shown in Table III, which depicts the scenario of seven documents of a query. It is assumed that the seven documents are located in three difference leaves (having 1.7, 0.9, and 0.4 labels). Note that there may be documents of other queries in the same leaf, so the assigned scores (second column) are not necessarily the average label of the documents of the said query residing in that particular leaf. The values of the last column are calculated simply as the arithmetic average of the values of fourth column for which the score (second column) is equal (e.g., 1.33 is the average of 0, 3, and 1).

The *Gain* routine of Algorithm 2 in listwise settings is implemented according to Algorithm 4.

Modifying the Listwise Objective Function. Now we examine some variations of the listwise objective function. The original NDCG formula (cf. Equation (5)) is highly influenced by the top few ranks in order to capture user satisfaction [Järvelin and Kekäläinen 2000]. It is likely that this characteristic has an influence on the performance of RF-list in the sense that a more gentle discount function may cause the learning process to generalize better. It is known to the research community that apart from empirical success, there is no theoretical justification for the decay function used in Equation (5) [Croft et al. 2010]. Moreover, it has been shown by Wang et al. [2013], both theoretically and empirically, that the discount factor of NDCG does have an effect on the performance of IR systems. Hence, in this section, we alter the influence of the discount factor in the following two ways.

(1) We modify the discount of standard NDCG (Equation (8)) by raising the log term to a power as shown in Equation (14). As α approaches zero, the influence of the top few ranks is gradually reduced, and when $\alpha = 0$, there is no discount at all,

ALGORITHM 4: Listwise Gain (Using Expected NDCG)

Procedure: $Gain_{NDCG}(tree, \mathcal{D}, \mathcal{D}_{left}, \mathcal{D}_{right})$
Data: Current *tree*, data at node and left/right child: $\mathcal{D}, \mathcal{D}_{left}, \mathcal{D}_{right}$
Result: Change in NDCG resulting from split.
begin
  $leaves \leftarrow leavesSortedByAverageRelevance(tree)$;
  $leaves.remove(node(\mathcal{D}))$;
  $leaves.insert(node(\mathcal{D}_{left}), node(\mathcal{D}_{right}))$;
  for $q \in \mathcal{Q}_{\mathcal{D}}$ **do**
    $rank[0] \leftarrow 0$;
    $i \leftarrow 0$;
    for $leaf \in leaves$ **do**
      if $q \in Q_{leaf}$ **then**
        i++;
        $counts[i] \leftarrow |\mathcal{D}_{leaf,q}|$;
        $rank[i] \leftarrow rank[i-1] + count[i]$;
        $avgGain[i] \leftarrow \frac{1}{counts[i]} \sum_{d \in \mathcal{D}_{leaf,q}} 2^{l_{q,d}} - 1$;
      end
    end
    $expNDCG[q] \leftarrow \sum_i count[i] * avgGain[i] * \sum_{k=rank[i-1]}^{rank[i]} \frac{1}{\log(k+1)}$;
  end
  return $(\frac{1}{|\mathcal{Q}|} \sum_q expNDCG[q]) - previousAvgExpNDCG$;
end

Note that to reduce the time complexity, the $avgGain$ at each leaf for each query is maintained, and the cumulative discount factor $\sum_{k=rank[i-1]}^{rank[i]} \frac{1}{\log(k+1)}$ is precomputed, so that during gain computation they can be computed in constant time.

implying that all the ranks are considered as the same. We also examine values of α that are greater than 1, implying that the NDCG function is even more influenced by the top few ranks than the standard version—we conjecture that this will not work well as it aggravates the discrepancy:

$$discount_{\alpha}(r) = [log(r+1)]^{\alpha}; \ \alpha > 0. \tag{14}$$

(2) According to Wang et al. [2013], the discount function given in Equation (15) is appealing from a theoretical perspective. The influence of the top few ranks of the standard NDCG formula is more than that of this one:

$$discount_{\beta}(r) = r^{\beta}; \ \beta \in (0,1). \tag{15}$$

4.3.3. Ordering of Leaf Expansion. A tree in a random forest is usually grown in a recursive (i.e., *depth-first*) manner. For pointwise loss functions, the order of expansion of nodes does not influence the data partitions, for the change in the loss function that results from the expansion of a node is computed locally, that is, based only on the data at the current node. A listwise loss function, however, cannot be decomposed into losses on individual data points, and instead must be computed over the entire set of documents associated with a particular query. Since the documents of a particular query are spread out all over the (current) leaves of a tree, the objective function cannot be calculated locally, that is, without knowledge of the scores currently assigned to all other documents in the tree. Thus, in the listwise approach, the node expansion ordering does matter.

Node Expansion Strategies. The recursive implementation (i.e., depth-first exploration) is mostly used in the literature due to its ease of implementation. However,

it makes the intermediate structures of a tree highly imbalanced, which is not favorable for listwise objective function. Hence, we consider the following four exploration strategies: (1) breadth-first: maintains a FIFO queue of leaves; (2) random: selects a random leaf among the available ones; (3) biggest-first: select the biggest leaf—the rationale is that the biggest node should get priority to be split in order to reduce the amount of suboptimal predictions; and (4) most relevant leaf first: as the name implies, selects the node whose score is the highest because relevant documents are of most interest from the perspective of NDCG. We applied all of these strategies on Fold 1 of MSLR-WEB10K and found their results mostly similar to each other (the results are given in Appendix A). In the breadth-first exploration, the data size of a node gradually decreases as the depth of the tree increases; thus, it follows a systematic way of exploration. Hence, throughout the rest of the article, we employ the breadth-first exploration.

4.4. A Hybrid Algorithm to Scale Up RF-Based Listwise Algorithm

Usually the listwise algorithms are computationally more complex, and RF-list is expected to be no exception. In this section, we first analyze the time complexities of RF-point and RF-list, which confirm that RF-list is computationally much slower than RF-point. Since this may prohibit the use of RF-list for very large datasets when computational resources are limited, we then discuss some heuristics to scale it up. Using heuristics comes at a cost of performance degradation (as compared to the original RF-list). We then design a hybrid algorithm, which is a better option for scaling up RF-list.

4.4.1. Time Complexity. Given N training instances over M features and containing Q queries, the training time complexities RF-point and RF-list are as follows (see Appendix B for their derivation):

***RF-point*: $\mathbf{O}(\log(M)N\log^2(N))$**
***RF-list*: $\mathbf{O}(\log(M)N(\log^2(N) + QN))$**

We see that the time complexity of RF-list is greater than quadratic in the number of training examples. For large datasets, which are natural for LtR, it may thus be computationally prohibitive. Indeed, as will be shown in Section 5, the largest dataset on which we managed to apply RF-list was MQ2007, which contains approximately 1,000 queries (45,000 query-document pairs) in the training set. For the larger ones, we had to resort to using smaller subsamples per tree, which may result in suboptimal performance unless the ensemble size is very large—which again may not be supported by the available computational resources. Hence, next, we discuss some heuristics to reduce the learning time of RF-list. We then design a hybrid algorithm to tackle the computational issue.

4.4.2. Heuristics to Reduce Time Complexity of RF-List. The following heuristics may be applied to reduce the training time of RF-list:

(1) Using a random subset of the available queries at a node for gain computation
(2) Using a smaller fraction of data (instead of traditional 63% data) to learn a tree (similar idea is used by Ibrahim and Carman [2014] and Friedman and Hall [2007])
(3) Increasing the minimum number of instances in a leaf (which is tantamount to decreasing the height of a tree)
(4) Considering a fraction of possible split values for a feature instead of enumerating all of them (similar idea is used by Denil et al. [2013], Geurts and Louppe [2011], Lin and Jeon [2006], and Ishwaran [2013])
(5) Using a larger gain threshold to split a node

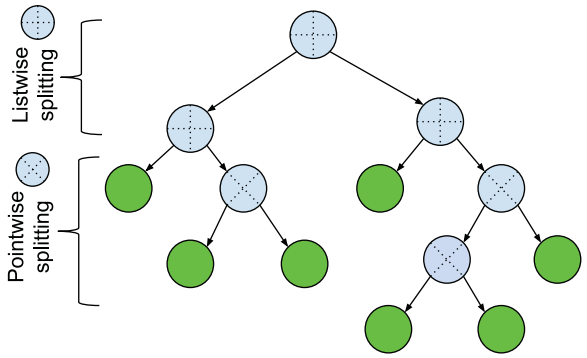

Fig. 3. RF-hybrid-L2: Splits up to level 2 (i.e., three splits) are listwise; the rest are pointwise.

(6) Using NDCG@k with a very small k instead of the untruncated one in the gain computation

We conjecture that each one of these will increase the bias of individual trees because each of them harnesses the explorative ability of a tree. Specifically, these heuristics will reduce the tree size—either due to lack of sufficient training data (idea 2); due to explicitly not splitting a node further (ideas 3 and 5); due to exploring a limited sample space, which results in no further gain (idea 1); or due to limiting the number of potential split points that may not see further gain (idea 4)—thereby making the individual trees less strong, that is, with less predictive power. That said, after extensive experiments using the validation set, an effective setting of these parameters may be found, which we leave to a future work (except some limited experiments with idea 2 as will be detailed in Section 5). Instead, we explore yet another interesting idea as detailed next.

4.4.3. RF-Hybrid Algorithm: A Combination of Listwise and Pointwise Splitting. After implementing the algorithms, our experience is that the major portion of learning time of a tree in RF-list is spent when a tree becomes sufficiently deep, that is, when the number of leaves is large, because for every possible split point, it is computationally expensive to traverse all the leaves. In this section, we aim at minimizing the learning time without using any heuristics mentioned in the previous discussion (which may increase bias). We achieve this by splitting the early nodes of a tree using the listwise objective function, and afterward resorting to the pointwise objective. That is, up to some predefined level of a tree, the listwise splitting is used, and for the rest of the nodes, the pointwise splitting is used. We call this algorithm *RF-hybrid-Lx*, where the parameter L is the number of levels of a tree up to which the listwise splitting is used. Figure 3 depicts the scenario. (Note that the breadth-first enumeration is a natural choice here too.)

The aforementioned idea has an intuitive interpretation. Suppose we use the listwise splitting for the first two levels (i.e., RF-hybrid-L2 as shown in Figure 3). This means that the first three splits (namely, the splits of the root node and then of its two children) are based on the listwise objective function. We can view this as if we have four (sub)trees (of left and right children of the root node), which themselves are learned solely based on the pointwise objective function. We note, however, that the final partitioning produced by the hybrid algorithm is strongly informed by the listwise objective, since that is used to determine the initial and most important cuts in the space. Thus, the hybrid algorithm is very different from a pointwise approach. When a test instance is to be traversed through the tree, two decisions are taken (root and either

Table IV. Statistics of the Datasets (Sorted by # Queries)

Dataset	TD2004	HP2004	NP2004	Ohsumed	MQ2008	MQ2007	MSLR-WEB10K	Yahoo
Task	Topic Distillation	Homepage Finding	Named Page Finding	Medical Docs	Web Search	Web Search	Web Search	Web Search
# Queries (total)	75	75	75	106	784	1,692	10,000	29,921
# Queries (train)	50	50	50	63	470	1,015	6,000	19,944
# Features	64	64	64	45	46	46	136	519
# Rel. Labels	2	2	2	3	3	3	5	5
# Examples (total)	75,000	75,000	75,000	16,000	15,211	69,623	1,200,192	709,877
# Examples (train)	50,000	50,000	50,000	9,684	9,630	42,158	723,412	473,134
# Docs per Query	988	992	984	152	19	41	120	23
# Docs with Rel. Label: 0/1/2/3/4 per query	973/15	991/1	983/1	106/24/21	15/2.5/1	30/8/2	42/39/16/2/0.9	6/8/7/2/0.4

left or right child of it) to decide which pointwise subtree (out of four) should be used to predict the label of this instance. Assuming that the listwise splitting is better, this hybrid algorithm is thus expected to be able to inject the amount of "goodness" of the listwise objective function depending on the computational resource requirement—the higher the availability of resources, the more listwise splitting will be used.

We note here that it is a well-known practice in classification tasks to use different types of trees in a randomized ensemble (e.g., Xu et al. [2012] and Robnik-Šikonja [2004]) in order to get the benefit of various splitting functions and at the same time to further reduce the correlation among the trees. We have not found, however, any research that, like us, uses different splitting methods in a single tree.

4.5. Random-Forest-Based Pairwise Algorithm

We have so far discussed RF-based pointwise and listwise objective functions. To complement our investigation, we could think about designing an RF-based pairwise algorithm, which can be accomplished as described next. First, we need to create a new training set as follows: for every pair of documents that belongs to the same query and that has different relevance judgments, we generate a new feature vector by subtracting one from the other and assign a new ground-truth label, either $+1$ or -1 depending on the pairwise relevance order. After that, we need to fit a standard random forest with classification setting. During evaluation, every pair of documents will have a preference label predicted by the model from which it is straightforward to generate a ranking (using a topological sort).

Although pairwise algorithms such as Freund et al. [2003] were popular in the early times of LtR research, they essentially operate on quadratic-sized training data. Hence, for big datasets, it is mostly not a feasible solution, and that is why in recent years they have drawn comparatively less interest from the research community. For this reason, we do not pursue this direction further in this article.

5. EXPERIMENTAL RESULTS

We first specify the experimental settings. We then analyze results of (1) RF-point and RF-list, (2) RF-list with modified NDCG-discount, and (3) RF-hybrid.

5.1. Methodology

Datasets. We use several publicly available datasets of LtR that have been heavily used in the literature. Table IV shows their statistics. Further details can be found in Qin et al. [2010b], on the Letor website,[6] on the Microsoft Research website,[7] and

[6]http://research.microsoft.com/en-us/um/beijing/projects/letor/.
[7]http://research.microsoft.com/en-us/projects/mslr/.

in Chapelle and Chang [2011]. All the datasets except Yahoo are released with a predivision of five folds. The features of all these datasets are mostly used in academia. However, features of the Yahoo dataset (which was published as part of a public challenge [Chapelle and Chang 2011]) are not disclosed as these are used in a commercial search engine. This one comes, unlike the others, in a single fold but predivided into a training, a validation, and a test set. All the algorithms in this article are trained using these predefined training sets. We implemented the algorithms on top of the *Weka* machine-learning toolkit.[8]

Evaluation Metrics. For small to moderate-sized data (HP2004, TD2004, NP2004, Ohsumed, MQ2008, and MQ2007), we employ two widely used rank-based metrics, namely, NDCG@10 and MAP. For details of these metrics, please see Järvelin and Kekäläinen [2000]. For the big two ones, we add another metric, namely, ERR [Chapelle et al. 2009], as these two datasets are of our main interest.

For the sake of better compatibility with the existing LtR literature, we use the evaluation scripts provided by the Letor website for datasets TD2004, HP2004, NP2004, Ohsumed, MQ2008, and MQ2007. The Yahoo dataset does not come with any script, so we implemented the formulae given by the releaser of the data [Chapelle and Chang 2011] and used them for both the big datasets (MSLR-WEB10K and Yahoo). The reported NDCG@10 for a dataset is the average over all queries of a test set. If a query does not have any relevant document, we assume that its NDCG is 0.0.

Parameter Settings. Throughout the experiments, the following settings are maintained for a random forest unless otherwise stated:

—Tree size: as mentioned in Section 4, the only condition that prevents a leaf from being split is if no gain is found from doing so. We do not limit the tree size by a maximum height or minimum number of instances.
—Ensemble size: 500 trees are used for each ensemble. (Appendix D explains that this size is a good choice.)
—The number of random features to select at each node (denoted by K) is set to $log(\#features) + 1$ (as advocated by Breiman [2001]).
—Subsampling: as mentioned in Section 4, query-level subsampling without replacement is used. Two variations are used: (1) the default case: for each tree, 63% queries are sampled (uniformly at random) without replacement, and (2) for big datasets: in some cases, less than 63% queries are sampled (details will be provided in relevant places).
—When listwise splitting is used, nodes are explored using breadth-first enumeration.

Statistical Significance Test. Scripts provided by the Letor website are used to perform the pairwise significance test. Values of the metrics in ***bold and italic*** and **bold** fonts denote that the performance difference (between the two systems at hand) is significant with p-values less than 0.01 and 0.05, respectively. Sometimes a † sign in front of a value indicates that the performance is significantly worse than the baseline.

5.2. Results and Discussion

5.2.1. RF-Point and RF-List (with Expected NDCG, cf. Equation (11)). We first discuss the results of small to moderate-sized datasets (TD2004, HP2004, NP2004, Ohsumed, MQ2008, and MQ2007). The average results across the five folds are shown in Table V. Here a significance test is performed on all the test queries of all the five folds, not on individual folds—in Appendix C we report foldwise significance test results and a brief analysis.

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

Table V. Performance Comparison Among Pointwise and Listwise Approaches

Dataset	Metric	RF-Point	RF-List
Ohsumed	NDCG@10	0.4187 (0.0579)	**0.4377** (0.0524)
	MAP	0.4141 (0.0664)	***0.4326*** (0.0640)
MQ2008	NDCG@10	0.2245 (0.0422)	***0.2326*** (0.0444)
	MAP	0.4706 (0.0357)	***0.4778*** (0.0367)
MQ2007	NDCG@10	0.4368 (0.0221)	***0.4442*** (0.0161)
	MAP	0.4523 (0.0159)	***0.4606*** (0.0136)
TD2004	NDCG@10	0.3521 (0.0317)	0.3421 (0.0377)
	MAP	0.2551 (0.0217)	0.2549 (0.0242)
HP2004	NDCG@10	0.8068 (0.0535)	0.8032 (0.0578)
	MAP	0.7042 (0.0857)	0.6910 (0.1054)
NP2004	NDCG@10	0.7955 (0.0845)	0.7797 (0.0624)
	MAP	0.6749 (0.0425)	0.6342 (0.0480)

The ***bold and italic*** and **bold** figures denote that the best performance is significant with p-value less than 0.01 and 0.05, respectively. Standard deviation across the five folds is in brackets.

We see from Table V that performance of RF-list is significantly better than RF-point in six out of 12 cases (six cases for each of NDCG@10 and MAP), whereas in the rest of the cases the difference is not significant. Importantly, RF-list wins in all six cases when a dataset has (1) comparatively more queries and (2) graded relevance labels (for Ohsumed, MQ2008, and MQ2007, cf. Table IV), and the performance is even better when this type of dataset is comparatively larger (on MQ2007 as will be evident later when we shall report foldwise p-values). Thus, these results suggest that, for comparatively larger datasets having graded relevance, RF-list tends to outperform RF-point.

Another aspect regarding these results is that TD2004, HP2004, and NP2004 datasets are highly imbalanced in terms of the ratio of relevant to irrelevant documents (cf. Table IV). Future research may reveal whether this aspect has any influence on varying performance across the two algorithms.

Now we discuss the results of big data. For the two big datasets, namely, MSLR-WEB10K and Yahoo, RF-list takes a considerable time to execute. This is because to compute the listwise objective of a split point, RF-list needs to enumerate all of the current leaves of a tree (as opposed to the case of RF-point, where only the current (parent) leaf is involved in deciding the split point), which involves their sorting and traversing a large number of documents. Hence, instead of executing the original version of RF-list, we resort to the second idea of the heuristics mentioned in Section 4.4.2 to compare RF-point and RF-list.

This idea is advocated by Ibrahim and Carman [2014], which replaces bootstrap samples with much smaller-sized (sub)samples for learning a tree of an ensemble in the context of LtR. The authors there show that on small to moderate-sized datasets (MQ2008, MQ2007, and Ohsumed), using smaller subsamples per tree not only greatly reduces the computational time but also results in a small yet reliable improvement in overall performance of the ensemble (due to a reduction in correlation between the trees), provided the ensemble size is sufficiently large (1,000 trees were used).

For big datasets, the variances of individual trees are already comparatively smaller, so we stick to our setting of using 500 trees per ensemble. As for the subsampling, a random 5% of the training queries are used to learn a tree—this means that 300 and 1,000 queries are used for the MSLR-WEB10K and Yahoo datasets, respectively (cf. Table IV). (More analysis on the relationship between subsample size, performance, and time complexity are given in Appendix E.) We call this algorithm *RF-point(/list)-Sy*, where *y* indicates the percentage of training queries used to learn a tree. As such, the RF-point(/list) with the default sampling is denoted by RF-point(/list)-S63. When we use only RF-point(/list), it implies RF-point(/list)-S63.

Table VI shows the results. For the MSLR-WEB10K dataset, RF-list-S5 wins over its pointwise counterpart in terms of NDCG@10 and ERR (which are heavily influenced by the top few ranks). In contrast to the findings of MSLR-WEB10K, on the Yahoo dataset,

Table VI. Results of RF-Point-S5 and RF-List-S5 on Big Datasets

Dataset	Metric	RF-Point-S5	RF-List-S5
MSLR-WEB10K	NDCG@10	0.4256	***0.4344***
	ERR	0.3247	***0.3421***
	MAP	0.3432	0.3417
Yahoo	NDCG@10	***0.7373***	0.7283
	ERR	***0.4532***	0.4507
	MAP	***0.6121***	0.6029

The ***bold and italic*** and **bold** figures denote that the best performance is significant with p-values less than 0.01 and 0.05, respectively.

RF-list-S5 is outperformed by RF-point-S5.[9] This raises the question about consistency of performance of RF-list. To follow we give some discrepancies (cf. Table IV) between the two datasets that may have an influence on this performance difference:

(1) # features (M): MSLR-WEB10K: 136, Yahoo: 519
(2) # missing values (replaced by 0.0 as per the standard practice of other researchers [Liu 2011]): MSLR-WEB10K: 37%, Yahoo: 57%
(3) # docs per query: MSLR-WEB10K: 120, Yahoo: 23
(4) # relevant docs per query: MSLR-WEB10K has many fewer relevant documents per query than that of Yahoo

We leave the analysis of the aforementioned aspects to a future work (apart from a brief analysis on M conducted in Section 7.1) that may reveal if these aspects have any role to play in the performance difference between pointwise and listwise algorithms.

In summary, the findings that emerged from this subsection suggest that performance of RF-list and RF-point varies across different datasets.

A concern here could be whether the worse performance of RF-list-S5 on the Yahoo dataset is related to small subsample size. That is to say, given the results of RF-point-S5 and RF-list-S5 on the Yahoo data, how likely is it that RF-list-S63 will win over RF-point-S63? We postpone this discussion until we conduct an in-depth analysis of the hybrid algorithm in Section 5.2.3, which will provide us with a reliable indication as to what would be the case if we were to be able to run RF-list with standard settings (i.e., 63% subsampling) for the Yahoo dataset.

5.2.2. RF-List with Modified NDCG Discount (cf. Equations (14) and (15)). When using Equation (14), we choose $\alpha \in \{0.1, 1.0, 4.0\}$. Results are given in Table VII. We see that the setting with $\alpha = 0.1$ (when the influence of top few ranks is reduced) yields better performance than that of standard NDCG. The poor performance of the setting with $\alpha = 4.0$ indicates that giving (unnecessary) high importance to the top few ranks does not work well, which was conjectured in Section 4.3.2.

As for the second method (i.e., Equation (15)), we choose $\beta \in \{0.01, 0.1\}$. A similar trend to the previous results is found because $\beta = 0.01$ or 0.1 means that the influence of the top few ranks is less than that of standard NDCG.

To summarize, this experiment discovers that (1) as correctly conjectured in Section 4.3.2, the amount of influence of the top ranks of a ranked list is indeed an important factor for RF-list, and (2) reducing the influence in a controlled manner yields slightly better performance because it probably causes the learning to be more stable. In general, Equation (15) has been found to be better than Equation (14), which is also justifiable as the utility of the former equation is established by theoretical analysis.

[9]As conjectured, RF-point-S5 yields suboptimal performance as compared to the results of RF-point-S63 (as will be shown later in Table XI when we report more results). Nonetheless, it can fairly be compared to RF-point-S5.

Table VII. Results of RF-List-S5 with Different Discount Functions

	Value of α in Equation (14)			Value of β in Equation (15)	
Metric	**0.1**	**1.0 (Standard NDCG)**	**4.0**	**0.01**	**0.1**
Data: MSLR-WEB10K (Fold 1)					
NDCG@10	**0.4333**	0.4305	†***0.4203***	0.4322	0.4319
ERR	0.3435	0.3423	†***0.3351***	0.3419	0.3406
MAP	***0.3457***	0.3424	†***0.3350***	***0.3478***	***0.3465***
Data: Yahoo					
NDCG@10	0.7294	0.7290	†***0.7257***	***0.7325***	***0.7323***
ERR	0.4511	0.4510	0.4503	***0.4523***	**0.4520**
MAP	0.6032	0.6028	†***0.5989***	***0.6063***	***0.6061***

Pairwise significance test is performed treating the standard NDCG (with discount 1.0) as baseline. The ***bold and italic*** and **bold** figures denote that the performance difference is significant with p-value less than 0.01 and 0.05, respectively. † means significantly worse than baseline.

Table VIII. Results of RF-Point and RF-List-S5/RF-hybrid-L6

Metric	**RF-Point-S5**	**RF-Point**	**RF-List-S5**	**RF-Hybrid-L6**
Data: MSLR-WEB10K				
	Subsample Size Per Tree		**Subsample Size Per Tree**	
	5%	***63%***	***5%***	***63%***
NDCG@10	0.4256	0.4445	***0.4344***	***0.4502***
ERR	0.3247	0.3424	***0.3421***	***0.3492***
MAP	0.3432	0.3543	0.3417	***0.3570***
Data: Yahoo				
NDCG@10	0.7373	0.7538	†***0.7283***	†**0.7525**
ERR	0.4532	0.4594	†***0.4507***	0.4590
MAP	0.6121	0.6278	†***0.6029***	†**0.6263**

Significance test is conducted between (1) RF-point-S5 and RF-list-S5, and (2) RF-point and RF- hybrid-L6. The ***bold and italic*** and **bold** figures denote that the performance difference is significant with p-value less than 0.01 and 0.05, respectively. † means significantly worse.

We note that for the Yahoo dataset, although the listwise objective function with modified discounted NDCG (Equation (15)) outperforms the listwise objective with standard NDCG, it is still statistically worse than the result of its pointwise counterpart mentioned in Table VI.

5.2.3. RF-Hybrid. Table VIII shows the results for RF-hybrid. The following observations are made:

—For the MSLR-WEB10K dataset, RF-hybrid-L6 (i.e., having listwise splitting up to level 6 of a tree) wins in all three metrics. (This affirms the efficacy of using RF-point/list-S5 for measuring relative performance.) We can, therefore, conclude that if sufficient computational resources are not available to apply RF-list, RF-hybrid is a viable option to incorporate the benefit of the listwise objective function (if applicable, e.g., in MSLR-WEB10K). The amount of listwise splits will depend on the availability of the resources—the more resources are available, the deeper of the trees can be learned with listwise splitting. Note that this idea of hybridization is likely to work for any other computationally complex objective function.
—For the Yahoo data, the RF-hybrid-L6 performs slightly worse than its pointwise counterpart. This, again, supports the finding mentioned in Table VI, which showed that the listwise objective function is not effective for this dataset, which implies that guiding the data partitions by listwise splits (during the early nodes) does not help here. This further corroborates the hypothesis that RF-hybrid is indeed able to incorporate diverse splitting criteria in a single tree.

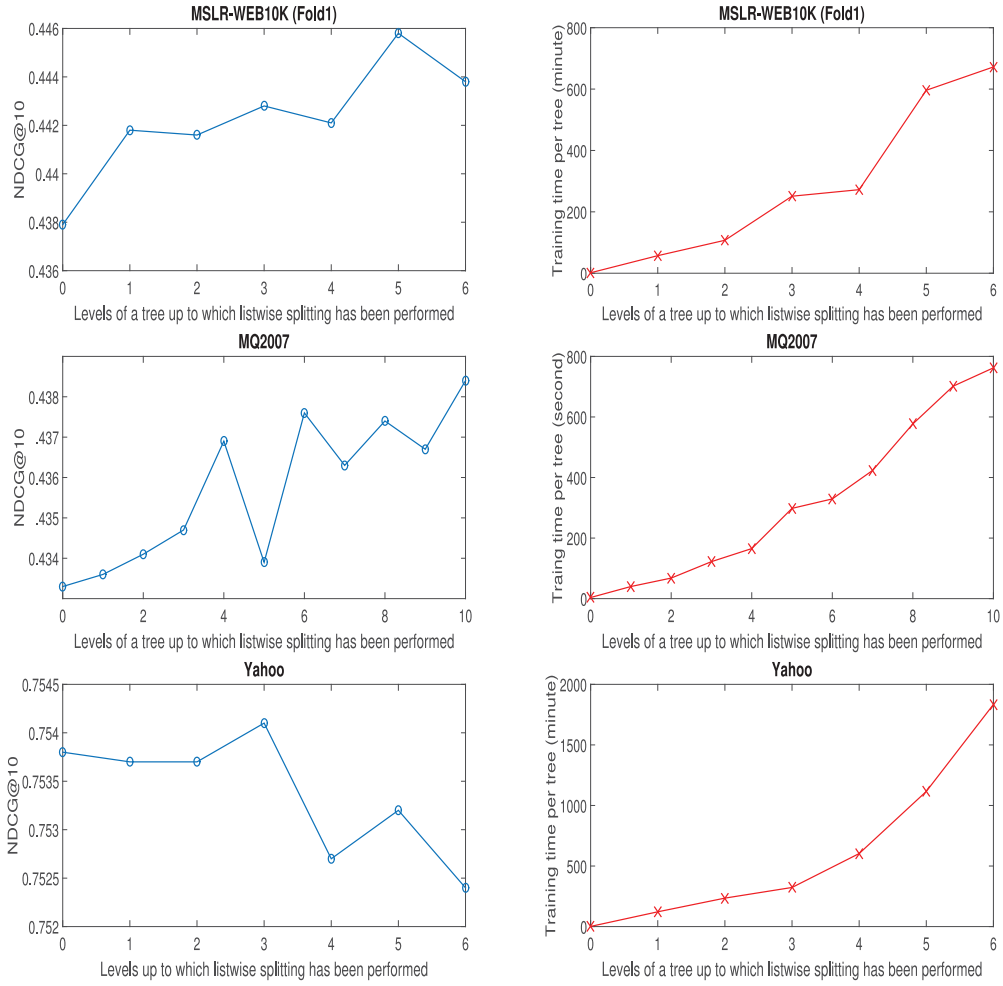

Fig. 4. RF-hybrid with different levels of tree up to which listwise splitting has been performed versus performance (left figures) and computational time (right figures).

In order to be certain as to whether RF-hybrid can indeed incorporate the benefit (if applicable) of the listwise objective function into a tree, we now conduct the following experiment. We gradually increase L (i.e., the level of a tree up to which listwise splitting is performed) and examine the performance trend. If a dataset benefits from a listwise objective function (e.g., MSLR-WEB10K), we hope to see a gradually increasing curve for performance.

In Figure 4, we show plots of performance and training time for three datasets, namely, MSLR-WEB10K, MQ2007, and Yahoo. For the big datasets, since we cannot learn an entire tree using the listwise objective function with the computational resources available to us, we stop at level 6 (out of approximately 17 and 16 for MSLR-WEB10K and Yahoo, respectively).

From the plots of Figure 4, we see that on the MSLR-WEB10K data, performance gradually increases (ignoring the minor fluctuations) with increasing level of injection of the listwise splitting in a tree. All of these differences between RF-hybrid and RF-point have been found to be statistically significant over RF-point at the $p < 0.01$ level.

Table IX. Training Time Improvement for RF-Hybrid with Subsampling Method

Metric	Level 4 (RF-Hybrid-L4)		Level 6 (RF-Hybrid-L6)	
	30% Queries	63% Queries	30% Queries	63% Queries
NDCG@10	0.4417	0.4421	0.4442	0.4438
ERR	0.3457	0.3458	0.3489	0.3474
MAP	0.3565	0.3562	0.3591	0.3562
Training time per tree (minute)	137	272	403	672

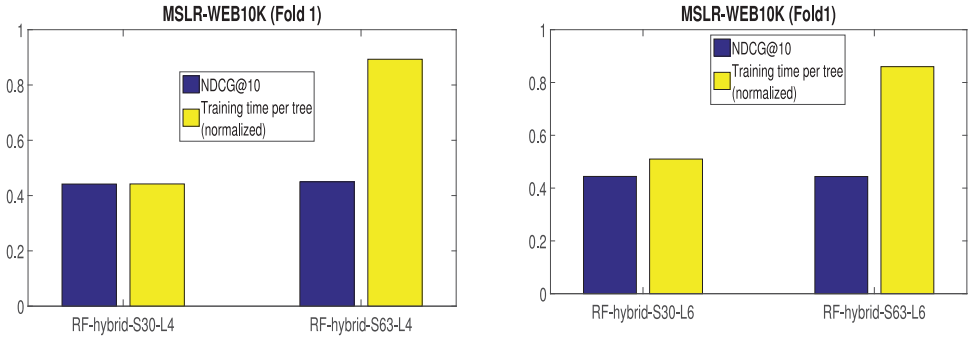

Fig. 5. Effect of subsampling method on RF-hybrid-L6 in terms of performance and (normalized) training time. Left figure: RF-hybrid-L4 with 30% and 63% subsample (per tree). Right figure: RF-hybrid-L6 with 30% and 63% subsample (per tree).

For the MQ2007 dataset, although there is more fluctuation because of its comparatively smaller size, the increasing trend is clearly visible (note that in all three cases, the ticks of the y-axis are very close to each other). This behavior was expected for these two datasets as both of them are already shown to have benefitted from the listwise objective function (cf. Tables V and VIII).

On the Yahoo dataset, the performance remains similar or becomes slightly worse as we increase the amount of the listwise splitting. (Here no RF-hybrid significantly wins over RF-point.) This strongly predicts that had we been able to run the RF-list with standard subsampling (i.e., RF-list-S63), performance would not be better than RF-point. This analysis diminishes the concern (raised in Section 5.2.1) about the effectiveness of the subsampling method while comparing between RF-point-S5 and RF-list-S5, and thus we can conclude that the mediocre performance of RF-list-S5 on the Yahoo dataset is less likely to be related to the smaller subsample size.

A question may arise here: can we take the privilege of the subsampling method for RF-hybrid? That is to say, can we reduce training time by using smaller data per tree without compromising performance? Table IX and Figure 5 show performance and training time (per tree) of RF-hybrid-S30 and RF-hybrid-S63 (i.e., standard subsampling) for two levels, namely, 4 and 6 (as such, the notation becomes RF-hybrid-S30(/63)-L4(/6)). We see that for both the levels, RF-hybrid-S30 produces similar NDCG@10 to that of RF-hybrid-S63 (the difference was not found to be statistically significant), but the learning time is greatly reduced, namely, by a factor of 2 and 1.7, respectively. Thus, it is evident that the subsampling approach indeed helps to reduce learning time greatly for big datasets without compromising performance, especially when a computationally complex objective function is employed.

6. FURTHER ANALYSIS OF SPLITTING CRITERION

As explained throughout the article, the objective function of RF-list is tailored to reduce the ranking error rate, as opposed to that of RF-point, where the goal is to minimize the misclassification rate. However, we see from the experimental results that although in most of the datasets (namely, Ohsumed, MQ2008, MQ2007, and MSLR-WEB10K), its performance has been found to be better than its pointwise counterpart, the margin is not that large, and importantly, it did not win in the Yahoo dataset, which is a crucial one as it has been a part of a commercial search engine. In this section, we investigate the reason behind this from three perspectives, namely, the importance of splitting criteria, overfitting, and individual tree strength.

6.1. Importance of Splitting Criteria

We aim to better understand how important the splitting criteria are in developing effective partitions of feature space for LtR. That is, does making it more adaptive to the evaluation metric (as done in RF-list) help?

6.1.1. LtR with Completely Randomized Trees. Here we employ the opposite motivation of RF-list. The idea of RF-list was to move from optimizing a surrogate metric (misclassification rate) to optimizing the actual metric of interest (NDCG). Now we design an objective function that is independent from any metric.

A random forest is, in fact, not a single algorithm; rather, it can be viewed as a general framework where some dimensions can be altered, which results in a class of algorithms. A large number of variations of Breiman's random forest (which we sometimes call the standard random forest) have been investigated in the literature.[10] For our purpose, we need to select a version where the splitting criterion is less adaptive to the training data, that is, where the amount of randomness is high. To this end, we choose an algorithm that has been widely used for classification and regression, both in theoretical (e.g., Genuer [2012], Scornet [2014] and Lin and Jeon [2006]) and empirical (e.g., Geurts et al. [2006] and Fan et al. [2003]) research.[11] Here we give a brief description of the algorithm that we call *RF-rand*.

In RF-rand, the splitting criterion is completely nonadaptive to the training data in the sense that the training phase does not depend on the labels of the training instances—the labels are only used to assign scores to the final data partitions. To split a node, a feature is randomly selected. Then, to select the cutoff value (along the selected feature) of the available data (at that node), there are a number of choices in the literature from which we choose the following: a random value is chosen between minimum and maximum values of the selected feature [Geurts et al. 2006].

As for the data per tree, most of the variants of RF-rand learn a tree from the entire training set instead of a bootstrapped sample (e.g., Geurts et al. [2006]) in order to reduce bias and variance of individual trees. We, however, think that in our case, it is sufficient to use bootstrapped sampling (without replacement) because the datasets in our case are already sufficiently large, as opposed to those works. Moreover, we did perform some experiments with full samples and found the results to be similar. So

[10]The list of references is too long to mention here; readers can look into Criminisi [2011] and the references therein.

[11]This version of random forest has been widely studied mainly for the following two reasons: (1) Standard random forest, despite its colossal empirical success over a range of tasks, to this date lacks the theoretical evidence of its consistency [Denil et al. 2013]. (Although very recent works by Wager [2014] and Scornet et al. [2014] prove consistency of a very close version of it.) It is comparatively easier to theoretically analyze (e.g., to prove consistency of) this variation of random forest as done by Scornet [2014] and the references therein, Biau [2012], Denil et al. [2013], and Biau et al. [2008]. (2) This version is computationally much faster.

Table X. RF-Point Versus RF-Rand on Small to Moderate Datasets

Dataset	Metric	RF-Point	RF-Rand
Ohsumed	NDCG@10	0.4187	0.4332
	MAP	0.4141	0.4312
MQ2008	NDCG@10	0.2245	0.2232
	MAP	0.4706	0.4742
MQ2007	NDCG@10	0.4368	0.4293
	MAP	0.4523	0.4480

Table XI. Big Datasets with RF-Point, RF-Rand, and a Complete Random Ranking

Dataset	Metric	RF-Point	RF-Rand	Random
MSLR-WEB10K	NDCG@10	0.4445	0.3978	0.1893
	ERR	0.3424	0.2978	0.1588
	MAP	0.3543	0.3271	0.1796
Yahoo	NDCG@10	0.7538	0.7389	0.5411
	ERR	0.4594	0.4517	0.2810
	MAP	0.6278	0.6156	0.4635

we report the results of bootstrapping (without replacement and, as before, at query level).

A point here to note is, although many variations of standard random forest are available in the literature, we have found only one work [Geurts and Louppe 2011] that uses one of those variations (not our selected one) for the LtR task (with much success). As such, the experiment of this section is not a reproduction of an existing technique; rather, it presents undiscovered results in the context of LtR, as will be evident next.

6.1.2. Results and Discussion. Although our main interest here is the big datasets, we first report results of some smaller datasets in Table X. We see that for the Ohsumed, MQ2008, and MQ2007 datasets, in general the performances of RF-point and RF-rand are comparable to one another. As for the results of big datasets shown in Table XI (where we also report a random ranking to examine the effectiveness of RF-rand, and later in Section 7 we shall compare RF-rand with several state-of-the-art algorithms), we observe that the performance difference between RF-point and RF-rand is stark (with $p < 0.01$).

From this investigation we conclude the following:

—On large LtR datasets, the splitting criterion is important. On smaller datasets, however, the opposite behavior was observed. This agrees with the findings of the work by Geurts et al. [2006], who use classification and regression datasets—which are much smaller than the larger LtR datasets we have used—to show that the random splitting method is generally on par with standard random forests.
—On large datasets, the numerical value of improvement of the pointwise objective over the random splitting method is much better than the improvement of the listwise objective over the pointwise objective (if applicable) (cf. Table VIII). This indicates that the listwise objective function cannot drastically improve (if applicable) the learned patterns over the pointwise objective. More on this will be discussed in Sections 6.3 and 6.4.
—Since there is too much randomness in the individual trees of RF-rand, performance is expected to be comparatively more dependent on the ensemble size. The larger the ensemble, the narrower is likely to be the difference between RF-point and RF-rand.

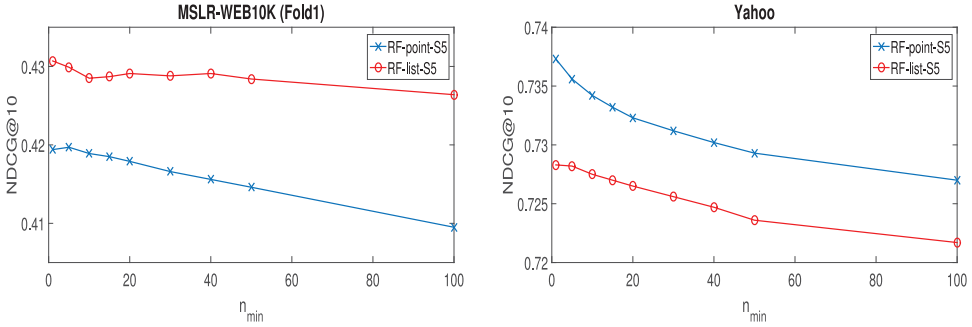

Fig. 6. NDCG@10 of RF-point-S5 and RF-point-S5 as n_{min} varies.

6.2. Overfitting

When using a listwise algorithm that directly optimizes an NDCG-based objective function, a natural concern is whether growing trees with unlimited depth overfit. (A similar concern is expressed by Tan et al. [2013].) The core motivation of random forest was that the individual learners need to have low bias, and one way to achieve this is to grow unlimited trees. While it is empirically found that the fully grown trees work very well, studies such as Segal [2004] and Lin and Jeon [2006] suggest that the minimum number of instances required to consider a split (n_{min}) can be tuned to gain slight performance improvement in classification and regression, especially for big datasets. Hence, in this section, we investigate this aspect.

Figure 6 shows performance as we increase n_{min}. In general, reducing n_{min} does not improve performance for both pointwise and listwise algorithms for both the datasets. The only exception we found is RF-point-S5 on MSLR-WEB10K, where a minor numerical improvement is seen when n_{min} is increased from 1 to 5, but this improvement is not statistically significant. The Yahoo data is more sensitive to this change (we performed a significance test for every two consecutive n_{min} values, and for the Yahoo dataset all these differences were found to be significant). Thus, this experiments suggest that unlimited trees do not overfit.

6.3. Strength of Individual Trees

In order to better understand the generalization performance of RF-point and RF-list, in this section we compare the strength of individual trees grown in a pointwise and listwise manner. The term "strength" was coined by Breiman [2001] to indicate the predictive power of the individual classification trees. In our context, we measure this predictive power in terms of NDCG.

To measure the strength of a single tree while it is growing, we compute the (untruncated) Expected NDCG of a tree after every split. To get a reliable estimate, we take the average of a particular value over 20 trees of an ensemble.

Since the sequence of these strength values (while a tree is growing) depends on the order of splits, to ensure fairness in comparison between RF-list (with breadth-first node enumeration) and RF-point, we implement a breadth-first version of RF-point (as explained in Section 4.3.3 that data partitions of RF-point do not depend on the order of exploration). Figure 7 shows the plots of all eight datasets using both training and test sets. Along the x-axis is the order of nodes as being expanded in a tree, and the average strength is along the y-axis. For instance, the first point of a curve is the Expected NDCG (averaged over 20 trees) after the root of a tree is split.

It can be seen from the plots of Figure 7 that the strength follows a similar pattern across all the datasets, which is as follows. In the early splits, Expected NDCGs of a

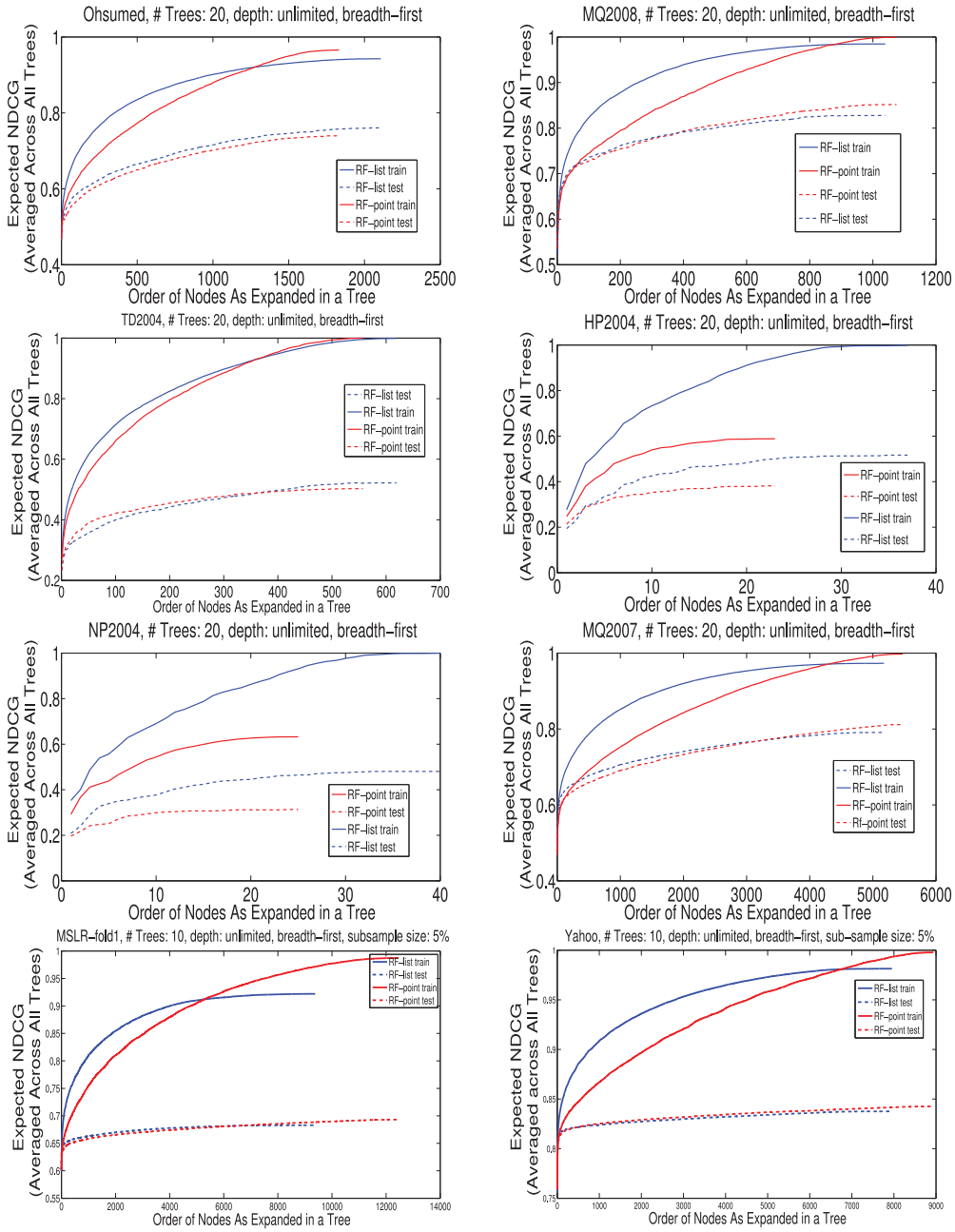

Fig. 7. Plots of progressive strength of individual trees (in terms of training and test Expected NDCGs) against the order of nodes expanded. The blue and red curves refer to RF-point and RF-list (both with breadth-first enumeration), respectively. The solid and dashed lines are for train and test curves, respectively.

tree of RF-list are higher, but the curve of RF-point catches up to that of RF-list after some splits (which varies from dataset to dataset). The reason for this pattern is due to the fact that a tree in RF-list directly maximizes NDCG. However, in RF-point, when the nodes are sufficiently *pure*, that is, contain instances of mostly one label (which happens after a tree has grown sufficiently deep), the relevant documents are likely to be in such partitions where they are mostly surrounded by other relevant documents, thereby getting higher scores and, in turn, causing the Expected NDCGs to be higher.

Let us not be confused here that the procedure for computation of NDCG values is not the same as those calculated *after* building a complete ensemble.

Thus, these plots reveal that the individual trees of the two algorithms, although they differ in the form of objective functions, essentially learn similar patterns of the training data given that sufficiently deep trees are grown—at that point individual trees of the two algorithms become similarly *strong*. This explains to some extent why the performances of the two algorithms do not differ by a large margin.

It can be noted that for some of the datasets such as MQ2007, the strength of the trees of RF-point on the test set is better than that of RF-list (the y-value at approximately the 5500th point along x-axis), but performance of RF-point is still worse than RF-list (cf. Table V). A possible explanation for this phenomenon is that the strength is one of the two components (the other is correlation among the trees) that control the error rate of a random forest [Breiman 2001]. This means that analyzing the mere strength of the trees will not be sufficient to predict the error rate of the ensemble. Therefore, instead of concentrating on increasing strength, future research may focus on other aspects of randomized-tree-ensemble-based rank learners such as reducing variance (e.g., by reducing correlation among the trees) and bias.

As a side note, we have also compared the strength of trees of RF-list with breadth-first and depth-first enumeration and found, as expected, that the curves of RF-list-breadth go up quite early, but as a tree grows deeper, RF-list-depth catches up. This further strengthens our conjecture that breadth-first exploration is a natural choice for RF-list. For brevity, we do not show those plots here.

Before concluding this section, next we briefly discuss two points in support of our conjecture made earlier that the deeper the trees, the less distinction between the effect of pointwise and listwise objective functions is observed.

(1) *Adding a Termination Criterion.* Figure 7 suggested that the shorter the trees of an ensemble, the more beneficial the listwise algorithm would be (as compared to the pointwise algorithm). In other words, if the listwise algorithm wins over its pointwise counterpart (e.g., on MSLR-WEB10K), then the performance difference between pointwise and listwise algorithms is expected to increase as shorter trees are learned. The opposite behavior is expected to be witnessed for the Yahoo dataset where listwise algorithms could not improve performance. To validate our conjecture, we exploit the parameter n_{min}, which is the minimum number of instances to split a node. This parameter has been widely used in the theoretical studies such as Lin and Jeon [2006] as a termination criterion. In Figure 8, we plot the performance difference between RF-point-S5 and RF-list-S5, and it shows that our conjecture holds mostly true (note the negative y-axis of the left plot).
(2) *Statistics of Individual Trees.* Table XII examines the (1) average number of leaves of a tree, (2) average leaf size, and (3) average leaf score of the MSLR-WEB10K (Fold 1) and Yahoo datasets (the performances of these settings were given in Table VI). These values suggest that RF-point needs comparatively deeper trees to compete (be it either when it wins (Yahoo) or loses (MSLR-WEB10K)) with RF-list.

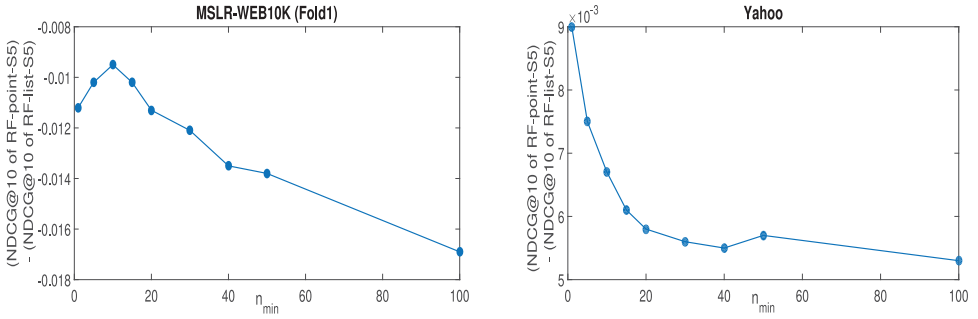

Fig. 8. Performance difference between RF-point-S5 and RF-point-S5 as n_{min} varies.

Table XII. Tree Size, Leaf Size, and Leaf Score of the Trees of RF-Point-S5 and RF-List-S5, Averaged over the Trees of the Ensemble (the Performances of These Settings Were given in Table VI)

	# Leaves		# Instances in a Leaf		Leaf Score	
Dataset	**RF-Point-S5**	**RF-List-S5**	**RF-Point-S5**	**RF-List-S5**	**RF-Point-S5**	**RF-List-S5**
MSLR-WEB10K	11,770	6,190	3.09	6.36	0.925	1.003
Yahoo	8,597	6,366	2.76	3.81	1.359	1.399

6.4. Discussion

The entropy-based objective function tries to isolate the similar class instances (i.e., of the same relevance label) in the leaf nodes of a tree. As for the NDCG-based objective, it also achieves a better gain if the leaf nodes of a tree that have predominantly relevant documents also have fewer irrelevant documents (and vice versa), because, in such cases, the scores of these nodes will be higher, and hence the resultant NDCG will be higher. In addition, using the same reasoning, the listwise objective function prefers to create a child node containing, for example, a lot of instances of label 2 and a few instances of label 1 to a lot of instances of label 2 and a few instances of label 0—simply because the latter partition drags the score of the *relevant leaf* to be lower than it should be.

In essence, the listwise splitting criterion, like its pointwise counterpart, attempts to improve the purity of the nodes. In addition, it attempts to capture the ordinal relationship between the relevance labels. Hence, the following conjecture emerges from this research: any nearest-neighbor-based algorithm, if the average of the relevance labels of the instances of a region (i.e., data partition) is assigned as the score of that region, will work well for the LtR task. This may be an interesting direction for future work. That is why RF-rand yielded reasonable performance despite its random splitting (cf. Section 6.1.1). In fact, random forest has already been interpreted by Lin and Jeon [2006] as a special case of the k-nearest-neighbor method.

The previous discussion may explain the effect of deep trees. When a tree is sufficiently deep, the entropy-based objective function by then manages to create such partitions that the relevant documents are isolated from the irrelevant ones to a reasonable extent. As compared to entropy-based splitting, an NDCG-based splitting probably takes a *harsher* decision in the early partitions with an aim to quickly make the relevant nodes more pure, and moreover, with an emphasis on their ordinal relationship as explained earlier.

Finally, we quote a comment made by Chapelle and Chang [2011] after their experience in the Yahoo LtR Challenge:

Most learning to rank papers consider a linear function space for the sake of simplicity. This space of functions is probably too limited and the above reasoning explains that substantial gains can be obtained by designing a loss function specifically tuned for ranking. But with ensemble of decision trees, the modeling complexity is large enough and squared loss optimization is sufficient.

This comment agrees with our explanations as to why a simple ensemble of randomized trees such as RF-point works quite well in practice for learning a good ranking function. Hence, for randomized-tree-ensemble-based algorithms, the improvement of the splitting criteria may not have a very big impact on performance (for big data).

7. COMPARISON WITH OTHER ALGORITHMS

In this article, our primary goal is to investigate random-forest-based LtR algorithms. That said, it is tempting to compare these algorithms with other state-of-the-art algorithms, especially the tree-ensemble-based ones. This section performs a comparison on the big datasets from two perspectives: absolute performance and the learning curve.

We use the following six baselines:

(1) Mart [Li et al. 2007]: a pointwise algorithm based on gradient boosting (i.e., tree ensemble based)
(2) RankSVM [Joachims 2002]: a pairwise algorithm based on SVM. This has been a popular baseline for many years
(3) AdaRank [Xu and Li 2007]: a listwise algorithm based on the AdaBoost framework (i.e., ideally a tree-ensemble-based one, although the prevalent implementation uses a single feature instead of a tree)
(4) CoorAsc [Metzler and Croft 2007]: a listwise algorithm based on coordinate ascent method; we choose this one because a random forest can also be interpreted as an optimization of the objective function in a coordinate-wise fashion
(5) RankBoost [Freund et al. 2003]: a pairwise algorithm based on the AdaBoost framework (i.e., tree ensemble based)
(6) LambdaMart [Wu et al. 2010]: a listwise algorithm based on gradient boosting (i.e., tree ensemble based)

The parameter setting of these algorithms (and a brief description of the ones that were not discussed in Section 3) are given in Appendix F. We also show results of the BM25 scorer, which is very popular in the IR community as a single feature.[12]

7.1. Absolute Performance

Table XIII shows the results of various algorithms on the big datasets. As for the statistical significance among the differences in performance, Table XIV shows pairwise significance test results for the four winning algorithms from Table XIII—only the top four are used because the rest of the algorithms perform quite poorly as compared to these four. Using the data from Tables XIII and XIV, Figure 9 graphically shows performance of various algorithms.

For the MSLR-WEB10K dataset, LambdaMart is the winner by a considerable margin from its nearest competitor, which is RF-hybrid-L6. RF-point and Mart perform similarly (i.e., there is no statistical difference between them). RankSVM, RankBoost, and AdaRank are the lowest three performers here. Interestingly, in spite of partitioning the data space randomly, RF-rand performs much better than RankSVM, RankBoost, and AdaRank in both the datasets and, in addition, is better than CoorAsc on Yahoo data. However, its performance is significantly worse than RF-point on both

[12]While the researchers who released the Yahoo dataset did not disclose information regarding the formulas used to compute the feature values, they did mention that a particular feature index corresponded to BM25.

Table XIII. Performance of Various Algorithms on Big Datasets: the Algorithms Are RF-rand (RF-ra), RF-point (RF-po), RF-hybrid-L6 (RF-hy), Mart, RankSVM (rSVM), AdaRank (AdaR), CoorAsc (CoAs), RankBoost (RBst), LambdaMart (LMart), and BM25 Score

Data: MSLR-WEB10K										
Metric	**RF-ra**	**RF-po**	**RF-hy**	**Mart**	**rSVM**	**AdaR**	**CoAs**	**RBst**	**LMart**	**BM25**
NDCG@10	0.3978	0.4445	0.4502	0.4463	0.3041	0.3431	0.4160	0.3360	0.4686	0.2831
ERR	0.2978	0.3424	0.3492	0.3491	0.2134	0.2746	0.3365	0.2460	0.3695	0.1910
MAP	0.3271	0.3543	0.3570	0.3541	0.2657	0.2814	0.3197	0.2853	0.3640	0.2562
Data: Yahoo										
NDCG@10	0.7389	0.7538	0.7525	0.7453	0.7177	0.7083	0.7177	0.7168	0.7520	0.6966
ERR	0.4517	0.4594	0.4590	0.4575	0.4316	0.4441	0.4460	0.4315	0.4615	0.4285
MAP	0.6156	0.6278	0.6263	0.6163	0.5983	0.5839	0.5906	0.5985	0.6208	0.5741

Table XIV. Significance Test Results for Comparison of Four Winning Algorithms of Table XIII. ✓✓ and ✓ Mean That the Difference Is Significant at p-value < 0.01 and 0.05, Respectively, and − Represents Otherwise. Empty Cell Means Not Applicable

Data: MSLR-WEB10K, Metric: NDCG@10

	RF-Hybrid-L6	Mart	LambdaMart
RF-point	✓✓	-	✓✓
RF-hybrid-L6		✓✓	✓✓
Mart			✓✓

Data: Yahoo, Metric: NDCG@10

	RF-Hybrid-L6	Mart	LambdaMart
RF-point	✓	✓✓	-
RF-hybrid-L6		✓✓	-
Mart			✓✓

datasets, indicating that an objective-based splitting criterion (in this case the entropy-based one) does help in an RF framework. Given that AdaRank is a listwise algorithm, its mediocre performance is somewhat surprising. This is probably due to the fact that it combines the features linearly, and we have already discussed in the last part of Section 6.4 that linear models may not capture the patterns of LtR data very well, especially if the dataset is large. CoorAsc is, in spite of being a listwise algorithm, outperformed by RF-point by a large margin. In the Yahoo dataset, among the top four (i.e., RF-point, RF-hybrid-L6, LambdaMart, and Mart), Mart is the worst performer, and interestingly, RF-point and LambdaMart perform similarly.

Both RF-hybrid-L6 and LambdaMart perform relatively better on MSLR-WEB10K data than on Yahoo data—this is revealed when we compare RF-point with these algorithms across the two datasets. This raises the question as to whether the characteristics of the Yahoo dataset cause the tree-ensemble-based listwise algorithms to perform comparatively less effectively. As we mentioned earlier, the aspects to investigate, among others, are the amount of sparsity in the data (37% of the feature values of MSLR-WEB10K are zeros as compared to 57% in Yahoo), the number of features (136 in MSLR-WEB10K vs. 519 in Yahoo), the number of documents per query (120 in MSLR-WEB10K vs. 23 in Yahoo), and the proportion of different labels (cf. Table IV).

Another aspect to ponder is that the boosting framework is mainly seen as a bias reduction technique (as individual trees are high-bias estimators), whereas a randomized tree ensemble (like random forest) is used to reduce variance (as individual trees are high-variance estimators). Hence, an interesting research direction can be to investigate the relative effects of bias versus variance reduction strategies on the performance of rank-learning algorithms.

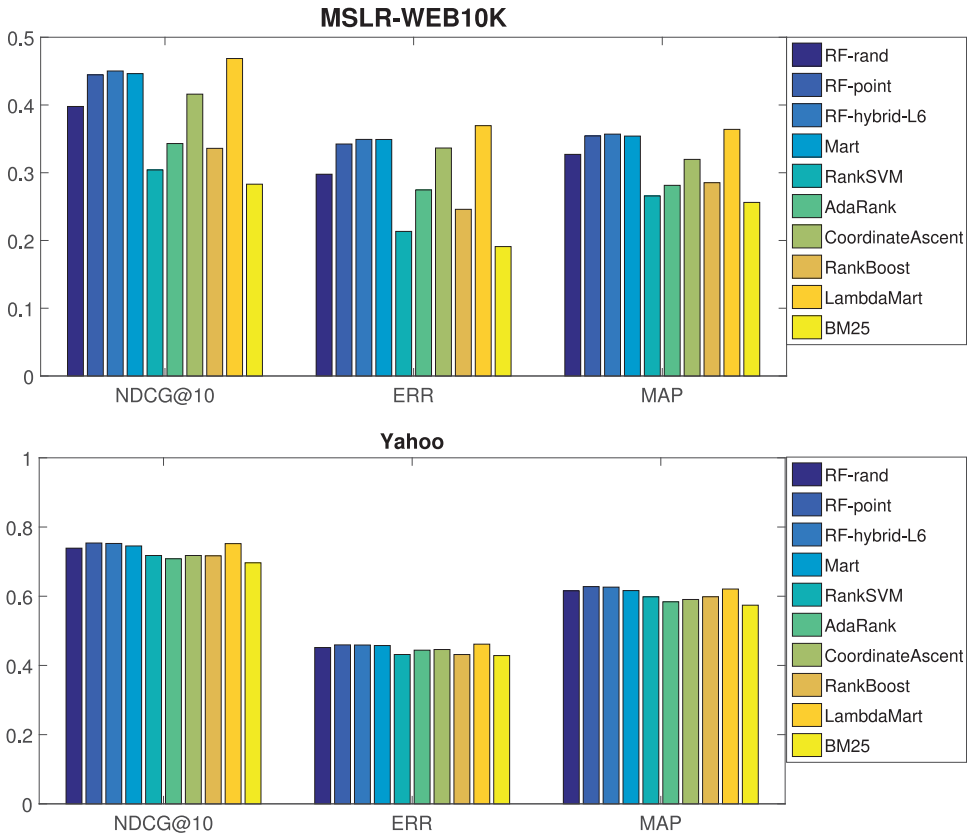

Fig. 9. Performance of different algorithms on big datasets.

Further Investigation into Performance Tradeoff Between Pointwise and Listwise Objective Functions on Yahoo Dataset. We now extend our analysis on relative performance of RF-point and RF-hybrid on the Yahoo dataset as RF-hybrid was unable to outperform RF-point on this dataset. We hypothesize that the increased number of features in the Yahoo data enables the pointwise algorithm to learn a complex hypothesis, thereby mitigating the surmised benefit of the listwise algorithm. The setting of a decisive experiment would be to compare the performance of Yahoo and MSLR-WEB10K using modified training sets where we retain only the common features of both. This is, however, not possible as the formulas used to compute the features on the Yahoo dataset were not disclosed. In this investigation, we reduce the number of features of the Yahoo dataset (as it is our main focus here) from 519 down to 10, 20, 40, 80, 160, and 320. We then train both RF-point and RF-hybrid-L6 using the modified training sets. The goal is to examine if RF-hybrid-L6 performs better with smaller features. Figure 10 shows that although RF-hybrid numerically marginally wins over RF-point as the number of features is reduced from 519 down to 10, none of the differences are statistically significant at the 0.01 level. This experiment thus indicates that the larger number of features in the Yahoo dataset (with respect to MSLR-WEB10K) is probably not the reason for the reduced effectiveness of the RF-hybrid over the pointwise one. Thus, the question as to what is the specific reason that RF-hybrid could not outperform RF-point on the Yahoo dataset still remains open.

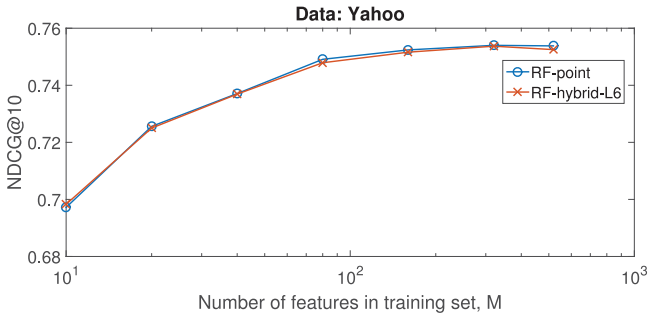

Fig. 10. With various M (in log scale), performance of RF-point and RF-hybrid-L6 on Yahoo dataset. The rightmost point corresponds to the original training set (i.e., $M = 519$).

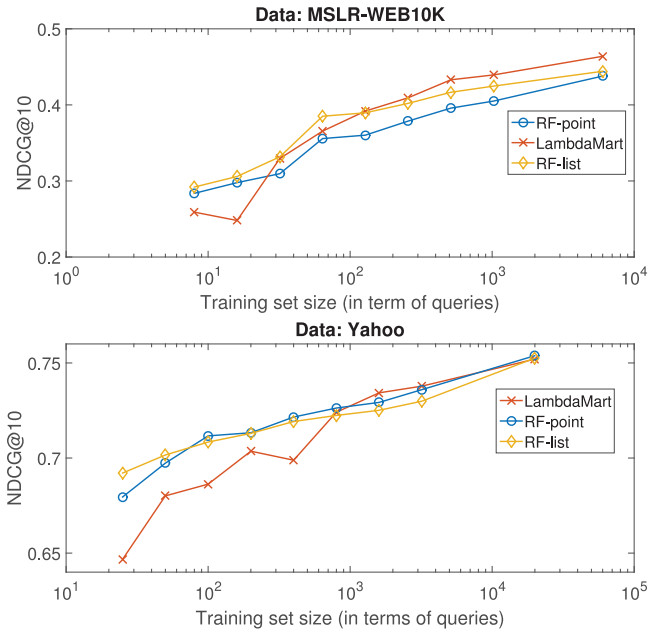

Fig. 11. Effect of training sample (in log scale) on performance of RF-point, RF-list, and LambdaMart. Recall that RF-list is computationally not feasible to learn from the entire training data (which is the last point of a curve), so we show performance of RF-hybrid-L6 for that point.

7.2. Learning Curve

Now we examine the learning curves for the two algorithms of our main interest, namely, RF-point and RF-list, and the top performer of Table XIII, namely, LambdaMart. Figure 11 shows the learning curve plots for the MSLR-WEB10K and Yahoo datasets. Ignoring some minor fluctuations, the trend in performance across the different algorithms is clear. We see that both the random-forest-based algorithms perform better than LambdaMart with very small datasets. Also, RF-list performs consistently better than RF-point on MSLR-WEB10K, and on Yahoo this trend is observed when the training data are small. On the Yahoo dataset, RF-list shows an interesting characteristic: it is the best with very small training sets, but eventually becomes the poorest (among the three) with larger training sets.

Table XV. Performance Comparison Between Various RF-Based Algorithms and LambdaMart on Small Datasets

Dataset	Metric	RF-Rand	RF-Point	RF-List	LambdaMart
MQ2007	NDCG@10	0.4314	0.4360	0.4433	0.4487
	MAP	0.4493	0.4524	0.4613	0.4678
MQ2008	NDCG@10	0.2228	0.2234	0.2313	0.2302
	MAP	0.4700	0.4693	0.4774	0.4751
Ohsumed	NDCG@10	0.4351	0.4306	0.4443	0.4367
	MAP	0.4311	0.4213	0.4332	0.4173
TD2004	NDCG@10	0.3299	0.3513	0.3558	0.3292
	MAP	0.2275	0.2574	0.2586	0.2378
HP2004	NDCG@10	0.8048	0.8082	0.7935	0.6398
	MAP	0.7040	0.7174	0.7031	0.5577
NP2004	NDCG@10	0.7480	0.7860	0.7675	0.6221
	MAP	0.6309	0.6647	0.6317	0.5006

For the RF-based algorithms (which are nondeterministic), each metric is the average of five independent runs (and each run is averaged over five folds).

In order to validate our conjecture that RF-based algorithms perform well with smaller datasets, in Table XV, we show the performance of the three RF-based algorithms investigated in this article and also that of LambdaMart. On the MQ2007 dataset, which is the largest (in terms of number of queries) among the six shown in the table, LambdaMart performs better than its closest competitor, namely, RF-list. In the rest of the cases, LambdaMart is outperformed by some RF-based algorithm(s). Interestingly, RF-rand again performs quite well even though its splitting criterion is completely random.

This subsection thus discovers that RF-based methods perform well on small datasets, which makes them a strong candidate for the domains where getting large amounts of labeled training data is comparatively more costly, for example, in enterprise search and domain-specific search.

To summarize, the following findings emerge from this section:

—Performance of the various algorithms is heavily dependent on properties of the dataset.
—For the Yahoo dataset, adapting the objective function to a rank-based metric appears to help less than it did in the case of MSLR-WEB10K (considering the relative performance degradation of RF-hybrid and LambdaMart on Yahoo as compared to MSLR-WEB10K).
—In terms of data dependency, LambdaMart is a relatively robust algorithm for big datasets.
—RF-point/list consistently win over RankSVM (pairwise), RankBoost (pairwise), Coordinate Ascent (listwise), and AdaRank (listwise), and are marginally better than Mart (pointwise) on big datasets. This shows that RF-based LtR algorithms are indeed competitive with the state-of-the-art methods, and hence these algorithms need further attention from the research community.
—BM25 on its own performs worse than all LtR systems in all cases, demonstrating the efficacy of the rank learners.
—Irrespective of the approaches (i.e., pointwise, etc.), nonlinear models tend to perform better for large-scale LtR. This raises a question: should more focus be devoted to understanding the model complexity (e.g., bias-variance tradeoff) rather than designing sophisticated LtR algorithms? This may be an interesting future research direction.

Before concluding this section, we highlight the fact that random-forest-based LtR algorithms are inherently completely parallelizable and require little or no tuning of their (very few) parameters.

8. CONCLUSION

Random-forest-based algorithms provide an inherently parallelizable solution to the learning-to-rank problem. While these algorithms have demonstrated competitive performance in comparison with other (oftentimes more complicated) techniques, their performance tradeoffs are still largely unknown. An attempt has been made in this article to rectify this, by examining the effect of pointwise and listwise splitting criteria in the context of learning-to-rank.

We have developed a random-forest-based listwise algorithm that can directly optimize an IR evaluation metric of choice. Experimental results have shown the listwise approach to outperform the pointwise one across most of the comparatively smaller benchmark datasets. For large datasets, a listwise algorithm can be computationally prohibitive, so we developed a hybrid algorithm that uses listwise splitting in the earlier stages of tree construction and pointwise splitting in the latter stages. The hybrid approach resulted in significant performance improvement in one of the two large datasets investigated. In addition, we investigated the effect of modifying the discount factor used in Normalized Discounted Cumulative Gain (NDCG) when employing it as a listwise objective function. By reducing the amount of discounting of lower-ranked documents, we were able to improve the generalization performance of the algorithm.

We then investigated further the relative generalization performance of the listwise and pointwise approaches. First, we examined the efficacy of an unsupervised partitioning strategy (using random splitting) in order to quantify the importance of the splitting criterion. Results on the smaller datasets indicated that the splitting criteria may not be the most important aspect for controlling the error rate, while for the larger datasets the splitting criteria do appear to have a significant effect on performance. Second, we investigated the overfitting issue in both the pointwise and listwise algorithms. Third, we analyzed the predictive accuracy (strength) of individual trees learned by the pointwise and listwise algorithms and noted that the predictive power of the individual trees became more similar the deeper they were grown.

We have compared RF-based LtR algorithms with several state-of-the-art methods from pointwise, pairwise, and listwise categories. This comparison has revealed that RF-based algorithms oftentimes perform better than several state-of-the-art algorithms on big datasets, thereby warranting further research on the RF-based algorithms. Also, by analyzing the learning curves of the big datasets and by comparing their performance on smaller datasets with one of the best LtR algorithms (namely, LambdaMart), we have found that RF-based algorithms perform quite well on smaller training sets, which makes them a strong candidate for the domains where getting a large amount of labeled training data is difficult (e.g., domain-specific search and enterprise search).

Several interesting directions for future research have emerged out of this work:

—Analyzing different characteristics of the datasets such as relevance label distribution and sparsity of the data to understand their impact on the performance difference of pointwise and listwise algorithms
—Investigating nearest-neighbor-based classification/regression algorithms for LtR as discussed in Section 6.4
—Investigating alternative methods for building a hybrid pointwise-listwise objective function
—Improving the treatment of missing values in a training set

—Investigating performance difference between RF-based algorithms and gradient-boosting-based algorithms from a bias-variance perspective as noted in Section 7.1
—Tuning the parameter for controlling the number of candidate features considered at each node of a tree

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments, which led to substantial improvements to the article.

REFERENCES

Gérard Biau. 2012. Analysis of a random forests model. *Journal of Machine Learning Research* 13, 1 (2012), 1063–1095.

Gérard Biau, Luc Devroye, and Gábor Lugosi. 2008. Consistency of random forests and other averaging classifiers. *Journal of Machine Learning Research* 9 (2008), 2015–2033.

Leo Breiman. 2001. Random forests. *Machine Learning* 45, 1 (2001), 5–32.

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*. ACM, 129–136.

Olivier Chapelle and Yi Chang. 2011. Yahoo! learning to rank challenge overview. *Journal of Machine Learning Research-Proceedings Track* 14 (2011), 1–24.

Olivier Chapelle, Donald Metlzer, Ya Zhang, and Pierre Grinspan. 2009. Expected reciprocal rank for graded relevance. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*. ACM, 621–630.

David Cossock and Tong Zhang. 2006. Subset ranking using regression. *Learning Theory* (2006), 605–619.

Antonio Criminisi. 2011. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends® in Computer Graphics and Vision* 7, 2–3 (2011), 81–227.

Antonio Criminisi and Jamie Shotton. 2013. *Decision Forests for Computer Vision and Medical Image Analysis*. Springer.

W. Bruce Croft, Donald Metzler, and Trevor Strohman. 2010. *Search Engines: Information Retrieval in Practice*. Addison-Wesley Reading.

Misha Denil, David Matheson, and Nando De Freitas. 2013. Narrowing the gap: Random forests in theory and in practice. *arXiv Preprint arXiv:1310.1415* (2013).

Wei Fan, Haixun Wang, Philip S. Yu, and Sheng Ma. 2003. Is random model better? On its accuracy and efficiency. In *Proceedings of the 3rd IEEE International Conference on Data Mining, 2003 (ICDM'03)*. IEEE, 51–58.

Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. 2003. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research* 4 (2003), 933–969.

Yoav Freund and Robert E. Schapire. 1995. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory*. Springer, 23–37.

Jerome H. Friedman. 2001. Greedy function approximation: A gradient boosting machine (English summary). *Annals of Statistics* 29, 5 (2001), 1189–1232.

Jerome H. Friedman. 2002. Stochastic gradient boosting. *Computational Statistics & Data Analysis* 38, 4 (2002), 367–378.

Jerome H. Friedman and Peter Hall. 2007. On bagging and nonlinear estimation. *Journal of Statistical Planning and Inference* 137, 3 (2007), 669–683.

Yasser Ganjisaffar, Rich Caruana, and Cristina Videira Lopes. 2011a. Bagging gradient-boosted trees for high precision, low variance ranking models. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 85–94.

Yasser Ganjisaffar, Thomas Debeauvais, Sara Javanmardi, Rich Caruana, and Cristina Videira Lopes. 2011b. Distributed tuning of machine learning algorithms using MapReduce clusters. In *Proceedings of the 3rd Workshop on Large Scale Data Mining: Theory and Applications*. ACM, 2.

Robin Genuer. 2012. Variance reduction in purely random forests. *Journal of Nonparametric Statistics* 24, 3 (2012), 543–562.

Pierre Geurts, Damien Ernst, and Louis Wehenkel. 2006. Extremely randomized trees. *Machine Learning* 63, 1 (2006), 3–42.

Pierre Geurts and Gilles Louppe. 2011. Learning to rank with extremely randomized trees. In *JMLR: Workshop and Conference Proceedings*, Vol. 14.

Jisoo Ham, Yangchi Chen, Melba M. Crawford, and Joydeep Ghosh. 2005. Investigation of the random forest framework for classification of hyperspectral data. *IEEE Transactions on Geoscience and Remote Sensing* 43, 3 (2005), 492–501.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference and Prediction.* Springer, 2nd Edition.

Muhammad Ibrahim and Mark Carman. 2014. Improving scalability and performance of random forest based learning-to-rank algorithms by aggressive subsampling. In *Proceedings of the 12th Australasian Data Mining Conference*. Australian Computer Society, 91–99.

Hemant Ishwaran. 2013. The effect of splitting on random forests. *Machine Learning* 99, 1 (2013), 75–118.

Kalervo Järvelin and Jaana Kekäläinen. 2000. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 41–48.

Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 133–142.

Yanyan Lan, Tie-Yan Liu, Zhiming Ma, and Hang Li. 2009. Generalization analysis of listwise learning-to-rank algorithms. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 577–584.

Hang Li. 2011. Learning to rank for information retrieval and natural language processing. *Synthesis Lectures on Human Language Technologies* 4, 1 (2011), 1–113.

Ping Li, C. Burges, and Qiang Wu. 2007. Learning to rank using classification and gradient boosting. *Advances in Neural Information Processing Systems* 19 (2007), 897–904.

Yi Lin and Yongho Jeon. 2006. Random forests and adaptive nearest neighbors. *Journal of the American Statistical Association* 101, 474 (2006), 578–590.

Tie-Yan Liu. 2011. *Learning to Rank for Information Retrieval*. Springerverlag, Berlin.

Donald Metzler and W. Bruce Croft. 2007. Linear feature-based models for information retrieval. *Information Retrieval* 10, 3 (2007), 257–274.

Ananth Mohan. 2010. *An Empirical Analysis on Point-Wise Machine Learning Techniques Using Regression Trees for Web-Search Ranking.* Master's thesis. Washington University in St. Louis.

Ananth Mohan, Zheng Chen, and Kilian Q Weinberger. 2011. Web-search ranking with initialized gradient boosted regression trees. *Journal of Machine Learning Research-Proceedings Track* 14 (2011), 77–89.

Ramesh Nallapati. 2004. Discriminative models for information retrieval. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 64–71.

Yanjun Qi. 2012. Random forest for bioinformatics. In *Ensemble Machine Learning*. Springer, 307–323.

Tao Qin, Tie-Yan Liu, and Hang Li. 2010a. A general approximation framework for direct optimization of information retrieval measures. *Information Retrieval* 13, 4 (2010), 375–397.

Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. 2010b. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval* 13, 4 (2010), 346–374.

C. Quoc and Viet Le. 2007. Learning to rank with nonsmooth cost functions. *Proceedings of the Advances in Neural Information Processing Systems* 19 (2007), 193–200.

Marko Robnik-Šikonja. 2004. Improving random forests. In *Machine Learning: ECML 2004*. Springer, 359–370.

Erwan Scornet. 2014. On the asymptotics of random forests. *arXiv Preprint arXiv:1409.2090* (2014).

Erwan Scornet, Gérard Biau, and Jean-Philippe Vert. 2014. Consistency of random forests. *arXiv Preprint arXiv:1405.2881* (2014).

Mark R. Segal. 2004. *Machine Learning Benchmarks and Random Forest Regression.* Technical Report. Center for Bioinformatics & Molecular Biostatistics, University of California, San Francisco, CA.

Ming Tan, Tian Xia, Lily Guo, and Shaojun Wang. 2013. Direct optimization of ranking measures for learning to rank models. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*. ACM, 856–864.

Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. 2008. Softrank: Optimizing non-smooth rank metrics. In *Proceedings of the International Conference on Web Search and Web Data Mining*. ACM, 77–86.

Stefan Wager. 2014. Asymptotic theory for random forests. *arXiv Preprint arXiv:1405.0352* (2014).

Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Tie-Yan Liu, and Wei Chen. 2013. A theoretical analysis of ndcg type ranking measures. *arXiv Preprint arXiv:1304.6480* (2013).

Qiang Wu, Christopher J. C. Burges, Krysta M. Svore, and Jianfeng Gao. 2010. Adapting boosting for information retrieval measures. *Information Retrieval* 13, 3 (2010), 254–270.

Baoxun Xu, Joshua Zhexue Huang, Graham Williams, Mark Junjie Li, and Yunming Ye. 2012. Hybrid random forests: Advantages of mixed trees in classifying text data. In *Advances in Knowledge Discovery and Data Mining*. Springer, 147–158.

Jun Xu and Hang Li. 2007. Adarank: A boosting algorithm for information retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 391–398.

Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. 2007. A support vector method for optimizing average precision. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 271–278.

Received May 2015; revised December 2015; accepted December 2015