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

MUHAMMAD IBRAHIM, Monash University, Australia
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 paper 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 which combines 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 trade-offs. The aspects include examining an RF-based unsupervised LtR algorithm, and comparing individual tree strength. Finally, we compare the three RF-based algorithms with several other LtR algorithms.

Additional Key Words and Phrases: Learning-to-rank, random forest, objective function, splitting criteria, NDCG

ACM Reference Format:
ACM Trans. Inf. Syst. V, N, Article A (January YYYY), 37 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

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 (eg. cosine similarity over tf-idf vectors) to probabilistic models (eg. BM25, Language Models), have been used for this task. Recently researchers have investigated supervised 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 (eg. cosine similarity) – this is called the learning-to-rank (LtR) problem. Many supervised learning methods have been developed and found empirical success over conventional methods [Liu 2011].

The LtR algorithms can be broadly categorized into three groups as follows: (i) pointwise: these algorithms minimize a loss function over the training set which assigns a loss to each individual feature vector (and thus treats each feature vector independently of one another), (ii) pairwise: where the loss function is calculated over every pair of feature vectors pertaining to the same query, and (iii) listwise: where 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 which 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] etc. where random forest is a very popular choice. But for LtR task, random forest has not been thoroughly investigated.

For 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 lack conceptual simplicity. Also, they are comparatively more prone to overfit as conjectured by Tan et al. [2013].

While it is simple to design an RF-based pointwise algorithm, we found no existing work on its listwise counterpart. The main barrier to design a listwise algorithm is that most of the IR metrics are non-smooth and non-convex with respect to model paprameters 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 coordinate-wise 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 non-convex and even non-smooth objective functions (such as those directly optimizes IR metrics like NDCG). RF-based pointwise algorithms have already been shown to be competitive to the state-of-the-art methods [Geurts and Louppe 2011], [Mohan et al. 2011]. In this paper we extensively compare the random forest based pointwise and listwise LtR algorithms.

The major contributions of this paper are as follows:

— We design a random forest based listwise LtR algorithm which directly optimizes any rank-based metric – we use NDCG and some of its variations.
— We design a random forest based hybrid algorithm which can incorporate different splitting criteria in a single tree. This is especially useful when some splitting criteria are computationally more complex (eg. NDCG-based ones) than others (eg. entropy-based ones) because the amount of computationally complex splitting 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 that:
  (1) For smaller and moderate LtR datasets (in terms of number of queries) with graded relevance labels, the performance of the listwise approaches are better than their pointwise counterparts.
  (2) For larger datasets, there is no clear winner between random forest based pointwise and listwise/hybrid algorithms, i.e., performance seems to be dependant on the properties of the dataset.
— We compare the relative performance difference of pointwise and listwise algorithms by investigating into the predictive accuracy (also known as strength) of individual trees. This shows that although looks different, both objective functions 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 many cases.
— We employ a purely-random(unsupervised partitioning)-tree based ensemble LtR algorithm. While its performance is not on par with random forest based pointwise/listwise algorithms, 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 many nearest-neighbor based methods may be effective for LtR. Moreover, random-

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 amongst 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 amongst 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 sub-sample per tree
$Leaves$	List of leaves of a tree

ized tree ensemble has huge computational advantage over many other LtR algorithms.

— Our experiments reveal that RF-based pointwise/listwise algorithms consistently win over to RankSVM (pairwise), RankBoost (pairwise), Coordinate Ascent (listwise), AdaRank (listwise), and at least similar to Mart (pointwise) on big datasetes. This shows that RF-based LtR algorithms are indeed at least competitive to the state-of-the-art methods. Also, the learning curves reveal 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 labelled training data are usually small (eg. domain-specific search and enterprize search).

The rest of the paper is organized as follows. Section 2 discusses 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), and then designs its listwise counterpart (RF-list). Here we also discuss the time complexities of the pointwise and listwise algorithms which then drives us to develop a hybrid algorithm. Section 5 discusses the experimental results of RF-point and RF-list. Section 6 investigates from several perspectives as to why performance of RF-point and RF-list was broadly similar. Although the main focus of this research is to compare RF-based LtR algorithms, we compare them with several state-of-the-art algorithms in Section 7. Section 8 summarizes the paper.

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

2. BACKGROUND

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

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

ALGORITHM 1: Procedure for constructing a Random Forest with query-level sub-sampling.

```
Procedure: BuildForest(D, E, b)
Data: Training set $\mathcal{D}$, ensemble size $E$, number of queries per sub-sample $b$
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.

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 minimise zero-one loss and mean squared error respectively.

In order to build the recursive partitions, a fixed-sized subset of the attributes are randomly chosen at each node of a tree, then for each attribute the training data (at that node) is sorted according to the attribute and a list of potential split-points (mid-points between consecutive training datapoints) is enumerated to find the split which minimises 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 show the pseudo-codes for building a forest and a tree respectively. The sub-sampling performed in lines 4-7 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 (eg. pointwise, listwise etc.) which will be discussed in details 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 $\vec{x}_{q,j} = \vec{\psi}(q, d_{q,j}) = \langle \psi_1(q, d_{q,j}), ..., \psi_M(q, d_{q,j}) \rangle \in \mathbb{R}^M$ 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, or the BM25 score for the same pair, etc.. 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].

[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.

A ranking function is then a function $f : \mathbb{R}^M \rightarrow \mathbb{R}$ that assigns scores to feature vectors, i.e., 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:

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

Here $rank_f(d_{q,i}, q)$ returns the position of document $d_{q,i}$ in 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 amongst a set of functions $\mathcal{F}$ that minimises 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 which will be used for evaluation. As such, a typical example of a loss function is one based on Normalised 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 normalised 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, i.e. 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 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 this paper we shall use NDCG to refer to the un-truncated 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 (i) they are non-differentiable with respect to the parameters of the retrieval function (due to the sorting in Equation 1) and (ii) they are almost invariably non-smooth and non-convex. 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 below.

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, i.e., using a pointwise approach. Despite the fact that very simple settings are used, they report competitive performance to other state-of-the-art algorithms. Mohan et al. [2011] also use a pointwise approach with regression settings. The main difference between these two papers 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

There has been some work on direct optimization of ranking measures for rank-learning, which we describe below.

Metzler and Croft [2007] use coordinate ascent to learn a ranking function which 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 learning a grid search over a large parameter space. They report better results than a standard language model.

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

Taylor et al. [2008] address the issue of non-smoothness 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, i.e., if non-integer 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 non-smoothness of the IR metrics. The authors use neural network framework to optimize the objective function. The algorithm is called as SoftRank. A problem 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 simple but effective algorithm called AdaRank which uses the AdaBoost framework [Freund and Schapire 1995] but it maintains 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 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. Thus it is intuitive that the AdaBoost framework produces the weak learners which are specifically built to deliver high NDCG, in contrast to delivering high classification accuracy.

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

[Qin et al. 2010a] develop a general framework which 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 non-continuous and non-differentiable) by the (smooth) logistic function. This surrogate objective function can then be optimized using any technique which is able to handle the local optima problem (the authors use the *random restart* method). A problem 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 of this study is that they use only three small datasets, namely, Ohsumed, TD2003 and TD2004.

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]	-	-

3.3. Motivation for Our Approach

From the previous discussion we see that the objective functions of LtR algorithms which use boosting, neural network and support vector machine have been well investigated in the literature. For example, for gradient boosting, Li et al. [2007] and Quoc and Le [2007] use pointwise and listwise algorithms respectively. For AdaBoost, Freund et al. [2003] and Xu and Li [2007] are pairwise and listwise algorithms respectively. For neural network, Cao et al. [2007] and Taylor et al. [2008] use listwise algorithms. For 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.

However, in spite of random forest's huge success in other disciplines such as bioinformatics [Qi 2012], image processing [Criminisi and Shotton 2013], spectral data processing [Ham et al. 2005] etc., there is very little work on using random forest for solving LtR problem, and importantly, to the best of our knowledge, 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 paper we choose to study its objective functions.

Since the NDCG function is both non-smooth and non-convex with respect to the model parameters, it is difficult to directly optimize the loss function of Equation 3. Instead, practitioners normally use some surrogate loss functions, for example, standard loss functions of 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 study 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 paper. We then discuss the existing RF-based pointwise algorithm. After that we develop its listwise counterpart.

4.1. Common Properties for All RF-based Algorithms

The three properties we discuss are as follows: (1) document score during training, (2) sub-sample 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], Mohan et al. [2011] etc.) over 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 $c \in \{0, 1, ..., C-1\}$ given the $leaf$ of the tree[2].

The score assigned to a test instance (eg. 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 ith tree where the instance has landed (here we have overridden the notation $socre(.)$ as compared to Equation 9).

4.1.2. Sub-sampling Per Tree. As for choosing between document-level and query-level sampling, traditionally bootstrapped samples were 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 which 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, to sample a query along with *all* the documents associated with it.

As for choosing between sampling with replacement and without replacement, 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. According to Ganjisaffar et al. [2011a], it works well for LtR task. Moreover, it reduces training time, especially for large datasets. Very recent works by Scornet et al. [2014] and Wager [2014] show that sampling without replacement allows for better theoretical analysis (including variance estimation) as compared to sampling with replacement. Thus our settings regarding sub-sampling are: (1) using sub-sampling (we note that some existing works use the entire data without any sampling at all, e.g. Geurts and Louppe [2011]), (2) using query-level sub-sampling, and (3) using sub-sampling without replacement, which means 63%[3] queries (and all of their associated documents) are in a sub-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. For stopping further splitting a node of a tree, the prevalent practice in RF literature is to grow a tree as deep as possible, thereby deliberately allowing it to overfit in an attempt to produce a tree with low bias [Breiman 2001]. This is favorable because the role of each individual tree is to sample from the space of possible predictors. The individual (high) variance of each tree does not harm in the end because the prediction of the ensemble is aggregated over the predictions of

[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 with c. The authors (and later Mohan [2010]) empirically examined various $g(.)$ including c, c^2, 2^c with several datasests such as Yahoo, and concluded that $g(c) = c$ works the best.
[3]63 is the average cardinality of a bootstrap sample (without replacement).

individual trees, thereby reducing the variance of the ensemble. However, later studies such as Segal [2004] empirically discovers that it is better to tune the maximum node size of a tree, and recent theoretical works such as Lin and Jeon [2006], Geurts et al. [2006], Scornet et al. [2014], Scornet [2014] confirm this by showing that as the data size grows, maximum node size should also grow. However, for the sake of clarity and better readability, we do not investigate this aspect in this paper (except a short discussion on overfitting in Section 6.2), thereby leaving it to future work. Throughout this paper we maintain the usual 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

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: $Entropy(\mathcal{D}) = -\sum_{c=0}^{C-1} \frac{|\mathcal{D}_c|}{|\mathcal{D}|} \log \frac{|\mathcal{D}_c|}{|\mathcal{D}|}$;

It has been shown that both classification and regression errors are upper bounds on the ranking error [Li et al. 2007] and [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 which can be interpreted as a blend of classification and regression settings – 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*.

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].

4.3. Random Forest Based Listwise Algorithm

A problem of using surrogate objective function instead of using a true one is, the learning algorithm may spend much effort to correctly classify some instances which have lower contribution in a graded[4] IR metric like NDCG, and even worse, perhaps at the cost of miss-classifying 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 which has direct relationship with NDCG. This is the main motivation behind developing this algorithm. The goal is to learn individual trees in such a way that NDCG (or any other ranking metric) is maximized. The main difference

[4]Graded means it uses a decaying function as discount of each rank as the rank goes down.

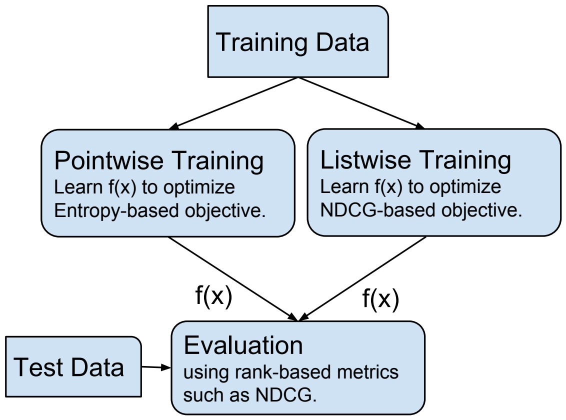

Training Data
Pointwise Training
Learn f(x) to optimize Entropy-based objective.
Listwise Training
Learn f(x) to optimize NDCG-based objective.
f(x)
f(x)
Test Data
Evaluation
using rank-based metrics such as NDCG.

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

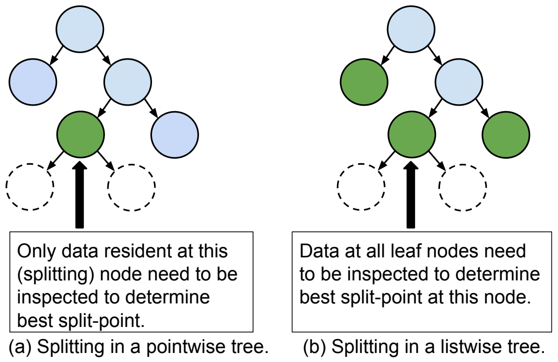

Only data resident at this (splitting) node need to be inspected to determine best split-point.
(a) Splitting in a pointwise tree.
Data at all leaf nodes need to be inspected to determine best split-point at this node.
(b) Splitting in a listwise tree.

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

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 which directly optimizes NDCG (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 of a tree, i.e., 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 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 optimising 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 below.

4.3.1. Document Scores During Training. This issue is not present in RF-point because its splitting criterion is invariant to the partition labels. But in order to compute a listwise objective, RF-list needs to assign a score to each document. To assign score to documents in a given partition during training we use the *expected relevance* mentioned 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 learnt independently from each other which makes it completely parallelizable.

4.3.2. Computing the Objective Function. We now discuss 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 $m\prime$ number of queries (out of m) are present. For a potential split-point, the scores of the documents directed to left and right child are computed according to Section 4.3.1, and then the NDCG values of each of the $m\prime$ queries should be updated according to the new scores of the documents. For the rest of the $m - m\prime$ queries, their current NDCGs will not change after this split because the scores and ranks of their instance documents (sitting in **other** leaves of the tree) will not change. The split-point which 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 entropy 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 initially the tree assigns the same score to all documents. Intuitively, a random order is not a solution. We calculate the expected NDCG over all possible orderings of the tied documents, which can be done efficiently as follows:

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

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 which have all been assigned the same score (by the tree f for query q). The expected gain at rank r can then be

[5]Indeed, the original paper on random forest [Breiman 2001] did mention that two components control the error rate of a random forest: (1) the correlation among the trees – the lower the better, and (2) the strengths of individual trees – the higher the better. In our context, the better the NDCG produced 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

computed as:

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

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

Now we illustrate the above 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 (2nd column) are not necessarily the average label of the documents of the said query residing in that particular leaf. The values of last column are calculated simply as the arithmetic average of the values of 4th column for which the score (2nd 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 biased towards the top few ranks in order to capture user satisfaction [Järvelin and Kekäläinen 2000]. It is likely that this characteristic has an influence in 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 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 in performance of IR systems. Hence in this section we alter the bias 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 with $\alpha > 0$. As α approaches zero, the bias is gradually reduced, and when $\alpha = 0$ there is no discount at all, implying that all the ranks are considered as the same. We also examine values for α that are greater than 1, where the NDCG function is even more biased towards the top ranks than the standard version. We conjecture that this will not work well since it exaggerates the bias toward the top ranks.

$$discount_\alpha(r) = [log(r+1)]^\alpha \quad (14)$$

(2) According to Wang et al. [2013], the discount function given in Equation 15 with $\beta \in (0, 1)$ is appealing from a theoretical perspective. The bias towards top ranks of standard NDCG formula is more than that of this one.

$$discount_\beta(r) = r^\beta \quad (15)$$

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.

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 matter, because the change in the loss function that results from the expansion of a node can be computed locally, based only on the data at the current node. A listwise loss function cannot be decomposed into losses on individual data points, and instead must be computed over the entire set of documents for a particular query. Since the documents for a particular query spread out all over the (current) leaves of a tree, the effect on the loss function of the expansion of a node cannot be calculated locally without knowledge of the scores currently assigned to all other documents in the tree. Thus in listwise approach the node expansion ordering does matter.

Node Expansion Strategies. The recursive implementation, i.e., depth-first exploration is mainly used because of its ease of implementation. But it makes the intermediate structures of a tree highly imbalanced which is not favorable for listwise objective function. Hence we implement 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 the biggest node should get priority to be split in order to reduce the amount of sub-optimal predictions, (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, data size of a node roughly gradually decreases as the depth of the tree increases, thus it follows a systematic way of exploration. Hence throughout the rest of the paper we employ the breadth-first exploration.

4.4. A Hybrid Algorithm to Scale-up RF-list

In this section we first analyze the time complexities of RF-point and RF-list which shows 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 resource is limited, we then discuss some heuristics to scale it up. However, it comes at a cost of sacrifice in performance (of the ideal RF-list). We then design a hybrid algorithm which is hence 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 the two algorithms are as follows, (see Appendix B for their derivation).

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

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

RF-list has a time complexity that is more than quadratic in the number of training examples. For large datasets which is natural for LtR, it may thus be computationally prohibitive. Indeed, as will be shown in Section 5, the largest dataset we managed to use with RF-list was MQ2007 which has around 1000 queries and 45,000 query-document pairs in the training set. For the larger ones, we had to resort to using smaller sub-samples per tree which may result in sub-optimal performance unless the ensemble size is very large – which again may not be supported by the available computational resources. Hence below we discuss some heuristics for reducing 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], Friedman and Hall [2007]).
(3) Increasing the minimum number of instances in a leaf or 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], Ishwaran [2013]).
(5) Using a larger gain threshold to split a node.
(6) Using NDCG@5/10 instead of untruncated one in the gain computation.

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

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 computational

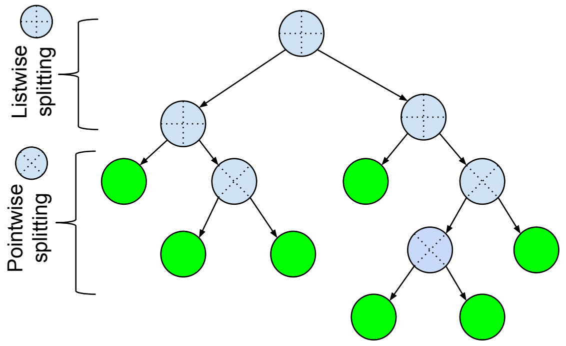


Fig. 3. RF-hybrid-L2: Splits up to level 2 (i.e., 3 splits) are listwise, the rest are pointwise.

time of RF-list is spent when a tree becomes sufficiently deep, i.e., 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 reducing 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 listwise objective function, and then resorting to the pointwise objective function. That is, up to some fixed level of the tree listwise splitting is used, and for the rest of the nodes pointwise splitting is used. We call this algorithms RF-hybrid-Lx where the parameter L is the number of levels of a tree up to which listwise splitting is used. Figure 3 depicts the scenario. (Note that the breadth-first enumeration is a natural choice here too.)

This idea has an intuitive interpretation. Suppose we use listwise splitting for the first two levels (i.e., RF-hybrid-L2 as shown in Figure 3). This means the first three splits (for splitting the root and its two children) are based on 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 learnt solely based on pointwise objective function. When a test instances is to be traversed through the tree, two decisions are taken (root and either left or right child of it) to decide which pointwise sub-tree (out of four) should be used to label this instance. Assuming for the time being that the listwise splitting is better, this hybrid version is supposed 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 deeper the tree will use the listwise splitting.

We note here that it is a well-known practice in classification task to use different types of trees in a randomized ensemble (e.g. [Xu et al. 2012], [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 work which, like us, uses different splitting method 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, it is tempting to think about designing an RF-based pairwise algorithm which can be accomplished . Firstly, we need to create a new training set as follows: for every pair of documents which belong to the same query and which have different relevance judgements, 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 which document has higher relevance. After that, we need to fit a standard random forest with classification setting. During evaluation, every pair of the documents will have a preference label predicted by the model from which is it straightforward to generate a ranking (using a topological sort).

Although pairwise algorithms such as [Freund et al. 2003] were popular in the beginning 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 in the research community. For this reason we do not pursue this direction further in this paper.

5. EXPERIMENTAL RESULTS

We first summarize the experimental settings. We then discuss results for (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 which have been heavily used in the literature. Table IV shows their statistics. Further details can be found in Qin et al. [2010b], in Letor website[6], in Microsoft Research website[7], and in Chapelle and Chang [2011]. All the datasets except Yahoo come pre-divided into 5 folds. The features of all these datasets are mostly used in academia. However, features of Yahoo data (which were 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 pre-divided into training, validation and test sets. All the algorithms in this paper are trained using these pre-defined training sets. We implemented the algorithms on top of *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. 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). We use each query of a test set for the average calculation. If a query does not have any relevant document, we assumed that its NDCG is 0.0.

Parameter Settings: The following settings are maintained for a random forest unless otherwise stated.

— Tree size: as mentioned in Section 4, the only condition which prevents a leaf from being split is if there is no gain from doing so. We do not limit the tree size using maximum height or minimum number of instances.
— Ensemble size: 500 trees are used for each ensemble. (Appendix D shows that this size is a good choice.)
— The number of random features to select at each node, K, is set to $log(\#features)+1$ (as advocated by Breiman [2001]).
— Sub-sampling: as mentioned in Section 4, query-level sub-sampling without replacement is used. Two variations are used: (1) the default case: for each tree 63% queries

[6]http://research.microsoft.com/en-us/um/beijing/projects/letor/
[7]http://research.microsoft.com/en-us/projects/mslr/
[8]http://www.cs.waikato.ac.nz/ml/weka/

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	1692	10000	29921
# Queries (train)	50	50	50	63	470	1015	6000	19944
# Features	64	64	64	45	46	46	136	519
# Rel. Labels	2	2	2	3	3	3	5	5
# Examples (total)	75000	75000	75000	16000	15211	69623	1200192	709877
# Examples (train)	50000	50000	50000	9684	9630	42158	723412	473134
# 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

are sampled without replacement, and (2) for big datasets: in some cases, less than 63% queries are sampled (details will be provided in relevant places).

— For RF-list, breadth-first node exploration strategy has been used.

Statistical Significance Test: Scripts provided by the Letor website are used to perform pairwise significance test. The ***bold and italic*** and **bold** numbers denote that the performance difference is significant with p-value less than 0.01 and 0.05 respectively. † in front of a number indicates that the performance is significantly worse than the baseline.

Table V. Performance comparison among pointwise and listwise approaches. 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 bracket.

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

5.2. Results and Discussion

5.2.1. RF-point and RF-list (with expected NDCG, cf. Equation 12). We first discuss the results for 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 - later we report fold-wise significance test results.

We see from Table V that performance of RF-list is significantly better than RF-point in 6 out of 12 cases (6 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 6 cases when a dataset has (1) comparatively more queries, and (2) more than 2 relevance labels (for Ohsumed, MQ2008 and MQ2007 - (cf. Table IV), and the performance is even better when this type of dataset is comparatively larger (for MQ2007 as will be evident later when we shall report fold-wise p-values). Thus these results suggest that, for comparatively larger datasets having graded relevance, RF-list tends to outperform RF-point.

Another aspect of the results is, 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 or not this aspect has any influence on the varying performance across the two algorithms.

Table VI. Results of RF-point-S5 and RF-list-S5 on big datasets. 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.

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

We further show in Table XVII (in Appendix C) the significance test results on a per-fold basis. Since TD2004, HP2004 and NP2004 have only 15 queries per fold in their test sets, we think that the significance test may not be indicative to consistent performance difference. As for Ohsumed, although it has only 21 queries, because of having graded relevance, we test it. We see that for MQ2007, RF-list strongly wins in folds 3 and 5, whereas it moderately wins in folds 2 and 4. For MQ2008, RF-list strongly wins in only fold 4, and moderately win in folds 1, 2 and 3 (considering some metrics). For Ohsumed, RF-list strongly wins in folds 1 and 3, and moderately wins in folds 4 and 5 (considering some metrics). We conjecture that the better performance for MQ2007 is mainly due to the fact that it has comparatively more training and test queries. Another point in this table is, for only the 2 cases (out of 30 across the three data sets) when RF-point wins, the difference is not significant (the p-values of those 3 cases should have been greater than 0.95 in order to be significant). This further strengthens our conjecture that RF-list is better than RF-point for comparatively larger datasets having graded relevance.

Now we discuss the results for big data. For big datasets such as Microsoft and Yahoo, RF-list takes considerable time to execute. This is because to compute the listwise objective score of a single potential split-point, RF-list needs to enumerate *all* 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 we resort to the 2nd option of the heuristics mentioned in Section 4.4.2 to compare RF-point and RF-list without executing the standard version of 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 for small to moderate-sized datasets (MQ2008, MQ2007 and Ohsumed), sub-sampling 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 (1000 trees were used).

For big datasets, the variance of individual trees are already comparatively smaller, so we learn 500 trees per ensemble. As for sub-sampling, a random 5% of the training queries is used to learn a tree – this means that 300 and 1000 queries are used for MSLR-WEB10K and Yahoo datasets respectively (cf. Table IV). (More analysis on the relationship between sub-sampling method, ensemble size and time complexity are given in Appendix E.) We call this algorithm as 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[9]. For MSLR-WEB10K dataset, RF-list-S5 wins over its pointwise counterpart in terms of NDCG@10 and ERR (which are heavily biased towards top few ranks). In contrast to the findings of MSLR-WEB10K, on Yahoo dataset RF-list-S5 is outperformed by RF-point-S5. This raises question about consistency of performance of RF-list. Below we give some discrepancies (cf. Table IV) between the two datasets which may have an influence on this performance difference.

(1) # features: 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 much less relevant documents per query than that of Yahoo.

In this paper we do not analyze the above-mentioned aspects. Future research may reveal if these aspects have any role to play in the performance difference between pointwise and listwise algorithms.

In summary, the experiments of this subsection suggest that performance of RF-list and RF-point vary across different datasets.

A concern here could be whether the worse performance of RF-list-S5 on the Yahoo dataset is related to small sub-sample size. That is to say, given the results of RF-point-S5 and RF-list-S5 on Yahoo data, how likely it is that RF-list-S63 will win over RF-point-S63? We postpone this discussion until we conduct in-depth analysis of the hybrid algorithm in Section 5.2.3 which will provide us with a reliable hint as to what would be the case if we were to able to run RF-list with standard setting (i.e., 63% sub-sampling) for 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 setting $\alpha = 0.1$ (when the NDCG is less biased towards the top ranks) is better than standard NDCG. The bad performance of the case $\alpha = 4.0$ indicates an ultra high bias towards the top ranks does not work well, which was expected.

As for the second method (i.e., Equation 15) we choose $\beta \in \{0.01, 0.1\}$. Similar trend is found because $\beta = 0.01$ or 0.1 means that the bias towards top ranks is less than standard NDCG.

Thus this experiment suggests that (1) as correctly conjectured in Section 4.3.2, the amount of bias towards the top ranks of a ranked list is indeed an important factor for RF-list, and (2) a comparatively less biased formula of standard NDCG is better because it probably makes the learning more stable. In general Equation 15 is better than Equation 14 which is also justifiable because the utility of the former is established by theoretical analysis.

Note that for Yahoo dataset, although the listwise objective function with changed discounted NDCG (Equation 15) outperforms the standard NDCG-based one, 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 observations are summarized below:

— For MSLR-WEB10K RF-hybrid-L6 wins in all three metrics. (This affirms the efficacy of using RF-point/list-S5 for measuring relative performance.) The following

[9]As conjectured, RF-point-S5 yields sub-optimal 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. 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.

Metric	Value of α in Equation 14: 0.1	1.0 (Standard NDCG)	4.0	Value of β in Equation 15: 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***

Table VIII. Results of RF-point and RF-list-S5/RF-hybrid-L6. Significance test was 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.

Metric	RF-point-S5	RF-point	RF-list-S5	RF-hybrid-L6
Data: MSLR-WEB10K				
	Sub-sample size per tree		Sub-sample 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**

conclusion can hence be drawn: if there is not sufficient computational resources to run RF-list, the benefit of the listwise objective function (if applicable, e.g., in MSLR-WEB10K) can be injected into the pointwise algorithm using RF-hybrid. The proportion of blending will depend on the availability of the resources – the more resources are available, the deeper of the trees can be grown with listwise splitting. Note that this idea of hybridization is likely to work for any other computationally complex objective function.

— For Yahoo data the RF-hybrid-L6 (i.e., having listwise splitting up to level 6 of a tree) performs slightly worse than its pointwise counterpart. This supports the finding mentioned in Table VI which showed that the listwise objective function is not effective for this dataset, which implies that injecting "listwise nature" into the objective function does not help here. This further corroborates the hypothesis that RF-hybrid is indeed able to incorporate diverse splitting criteria in a single tree. More on this aspect are discussed next.

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 another 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 listwise objective function (e.g., MSLR-WEB10K), we hope to see an 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

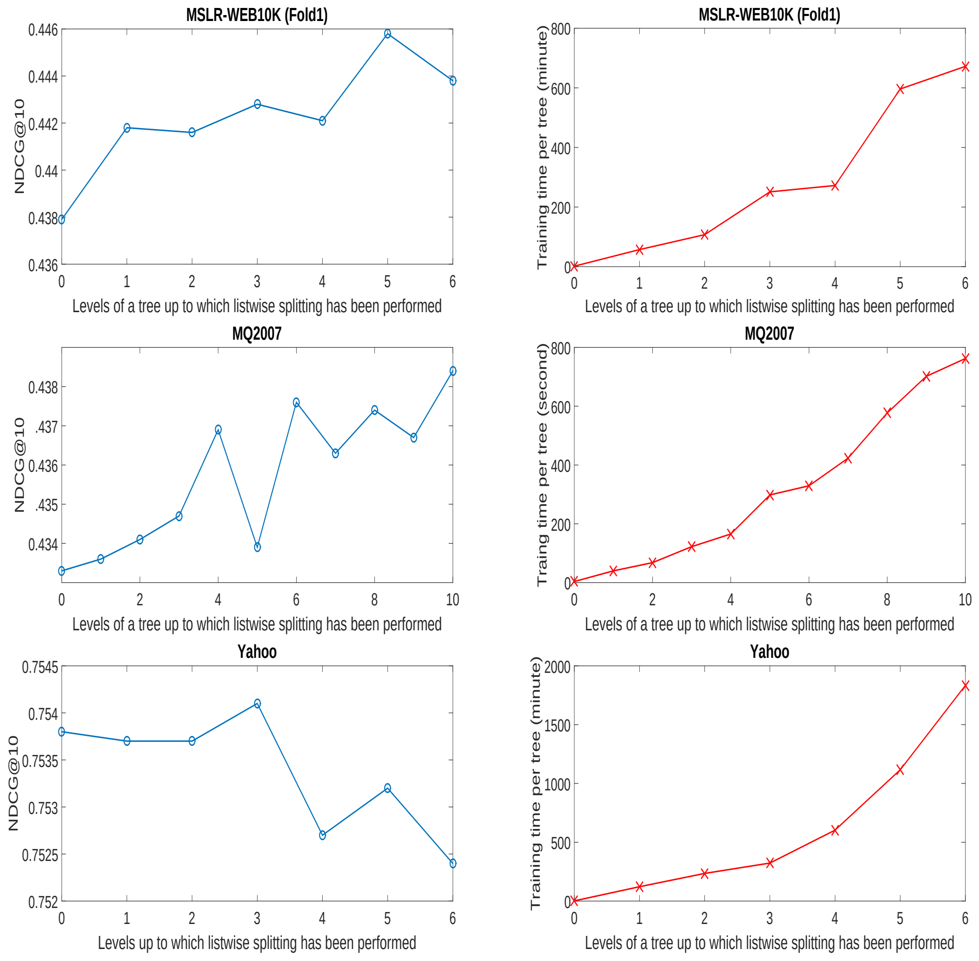
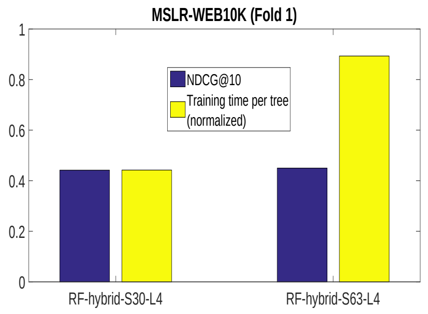


Fig. 4. RF-hybrid with different levels of tree up to which listwise splitting has been performed versus performance and computational time.

learn an entire tree using 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 for MSLR-WEB10K performance roughly steadily increases with increasing level of injection of the listwise splitting in a tree. All the differences between RF-hybrid and RF-point have been found to be statistically significant over RF-point at $p < 0.01$ level. For 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 benefit from using listwise objective function.

For 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, but the opposite behavior was observed for RF-hybrid-L4 and RF-hybrid-L6 at $p < 0.05$ level.) This strongly predicts that had we been able to run the RF-list with standard sub-sampling (i.e., RF-list-S63), performance would not be



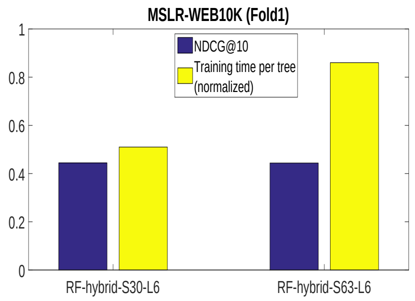


Fig. 5. Effect of sub-sampling method on RF-hybrid-L6 in terms of performance and (normalized) training time. Left figure: RF-hybrid-L4 with 30% and 63% sub-sample (per tree). Right figure: RF-hybrid-L6 with 30% and 63% sub-sample (per tree).

better than RF-point. This analysis diminishes the concern about the effectiveness of sub-sampling method while comparing between RF-point-S5 and RF-list-S5 (raised in Section 5.2.1) – thus we can say that the mediocre performance of RF-list-S5 on Yahoo dataset is less likely to be related to smaller sub-sample size.

Table IX. Training time improvement for RF-hybrid with sub-sampling method.

	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
Time per tree in minute (normalized)	137 (0.4498)	272 (0.8931)	403 (0.5100)	672 (0.8600)

A question may arise here: can we take privilege of sub-sampling 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 per-tree-training-time of RF-hybrid-S30 and RF-hybrid-S63 (i.e., standard sub-sampling) 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 (we did not find the difference 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 sub-sampling method indeed helps to reduce learning time greatly for big datasets, especially when a computationally complex objective function is employed.

6. FURTHER ANALYSIS OF SPLITTING CRITERION

As explained throughout the paper that 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 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 taken from a commercial search engine. In this section we investigate the reason behind this from three perspectives, namely, the importance of splitting criterion, overfitting, and individual tree strength.

6.1. Importance of Splitting Criteria

We aim to understand as to how important the splitting criteria is 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) helps?

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 (miss-classification rate) to optimizing the actual metric (NDCG). Now we design the objective function to be independent of any metric.

A random forest is, in fact, not a single algorithm, rather it can be viewed as a general framework where the parameters can be altered to result in a class of algorithms. A large number of variations of Breiman's random forest (which we henceforth call 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, i.e., where the amount of randomness is large. To this end, we choose an algorithm which has been widely used for classification and regression, both in theoretical (e.g. [Genuer 2012], [Scornet 2014], [Lin and Jeon 2006]) and empirical (e.g. [Geurts et al. 2006], [Fan et al. 2003]) research[11]. Below we give a brief description of the algorithm which we call RF-rand.

In RF-rand, the splitting criterion is completely non-adaptive 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 a score to the final data partitions. To split a node, a feature is randomly selected. Then to select the cut-off value of the available data, there is a number of choices in the literature from which we choose the following: a random value is chosen between minimum and maximum values of that feature [Geurts et al. 2006].

As for the data per tree, most of its versions learn a tree from the entire training set instead of 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 we report the results of bootstrapping (without replacement).

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] which uses one of those variations (not our selected one) for LtR task (with much success). As such, the experiment of this section is not a reproduction of an existing technique, rather it does present 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 report smaller ones first. Table X shows that for small to moderate sized datasets the performances of RF-point and RF-rand are comparable. As for the results of big datasets shown in Table XI, we also report a random ranking to examine the effective-

[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 used mainly for the following two reasons. (1) Standard random forest, despite its colossal empirical success across 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 for very close version of it.) It is comparatively easier to theoretically analyze (for example, to prove consistency of) this variation of random forest as done by [Scornet 2014] and the references therein, [Biau 2012], [Denil et al. 2013], [Biau et al. 2008] etc. (2) This version is computationally much faster.

Table X. RF-point Vs 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

ness of RF-rand (later in Section 7 we shall compare it with some other algorithms). We see that for big datasets, the performance difference between RF-point and RF-rand is stark (with $p < 0.01$).

From this experiment we conclude the following:

— For the large LtR datasets, the splitting criterion is important. However, for the smaller datasets the opposite behaviour 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 random splitting method is generally on a par with standard random forests. This similarity between their finding and ours is likely to be due to the fact that the smaller LtR datasets used in our work are similar to those of the classification and regression datasets used in their paper.
— The difference between RF-point and RF-rand is large as compared to the difference between RF-point-S5 and RF-list-S5 or between RF-point and RF-hybrid-L6 (cf. Table VIII). This hints that the listwise objective function cannot drastically change the learnt patterns. More on this will be discussed in later sub-sections.
— Since there is too much randomness in the trees of RF-rand, performance is likely 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.

6.2. Overfitting

When using a listwise algorithm which directly optimizes an NDCG-based objective function, a natural concern is whether growing trees with unlimited depth overfits. The core motivation of random forest was that the individual learners need to have low bias. 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, 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

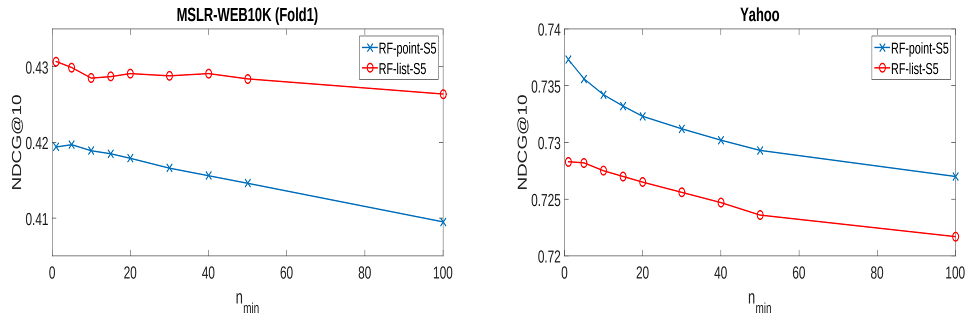


Fig. 6. NDCG@10 of RF-point-S5 and RF-point-S5 as n_{min} varies.

not statistically significant. Yahoo data is more sensitive to this change (we performed significance test for every two consecutive n_{min} values, and for 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 as it is grown, we compute the (untruncated) expected NDCG value of a tree after every split. To get reliable estimate we average this value over a number of trees. (If a tree terminates earlier than others, then we use the final value to compute the average.)

Since the sequence of strength values (as a tree grows) depends on order of splits, to ensure fairness in comparison between RF-list (breadth-first) and RF-point, we implement a breadth-first version of RF-point (as explained in the discussion of node exploration strategies that performance of RF-point does not depend on the order of exploration). For each dataset, we train 20 trees and compute average progressive strength for train and test sets. Figure 7 shows the plots for all of the datasets. Along the x-axis is the order of nodes as being expanded in a tree, and the average strength are along the y-axis. For example, 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 follow similar pattern across all the datasets which is as follows. In the early splits, NDCGs of RF-list are higher, but RF-point catches up after some splits. In the end, both have similar training NDCG. 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* (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 NDCGs to be higher.

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

Thus these plots reveal that the two algorithms, although 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

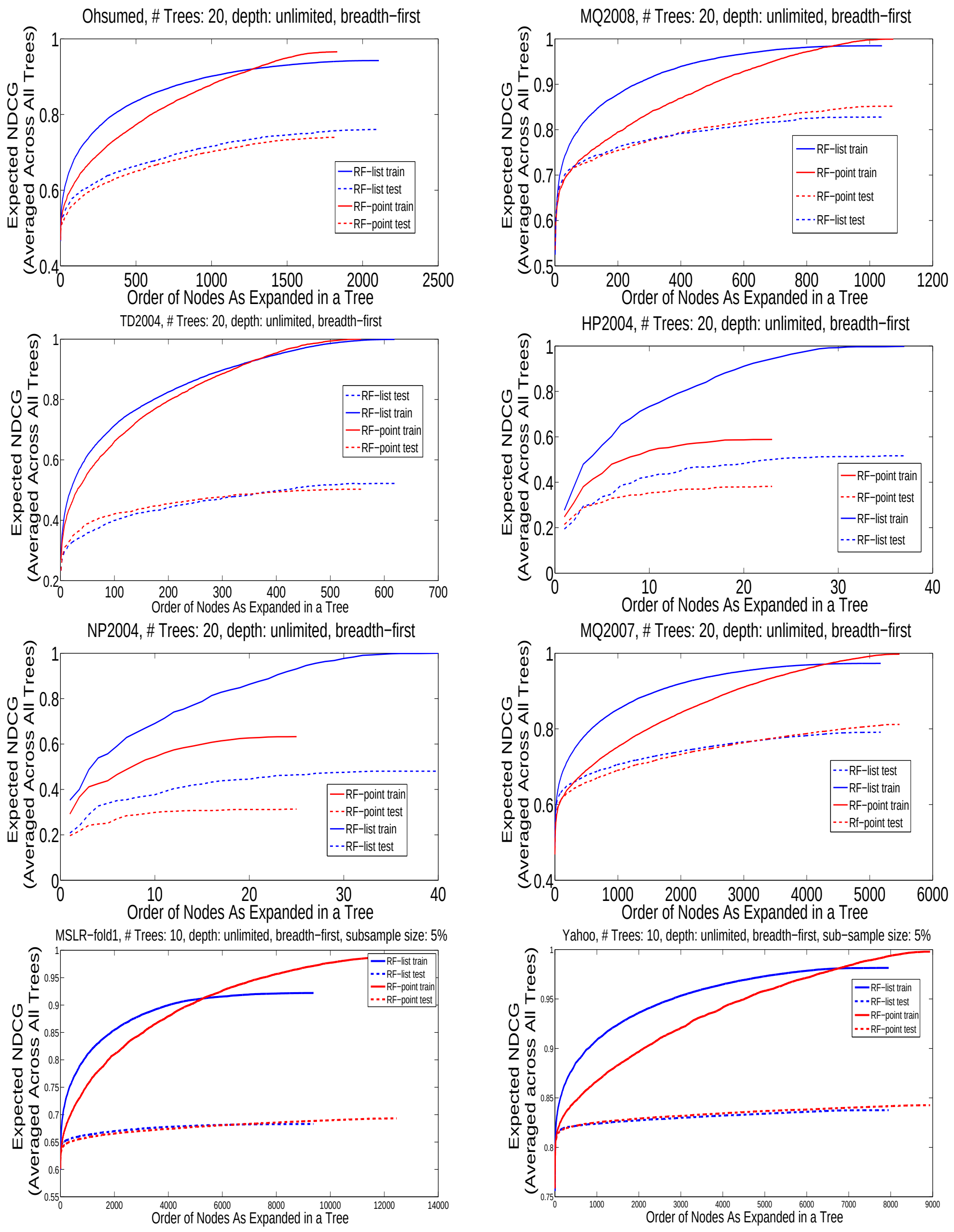


Fig. 7. Plots of progressive strength of individual trees (in terms of training and test expected NDCGs) Vs order of nodes expanded. The blue and red curves refer to RF-point and RF-list (both with breadth-first enumeration) respectively. The solid and dashed lines are for train and test curves respectively.

become similarly *strong*. This explains to some extent as to why the performances of the two algorithms does not differ by large margin.

It can be noted that for some of the datasets such as MQ2007 the strength of RF-point (i.e, the y-value at approximately 5500th point along x-axis) on test set are bet-

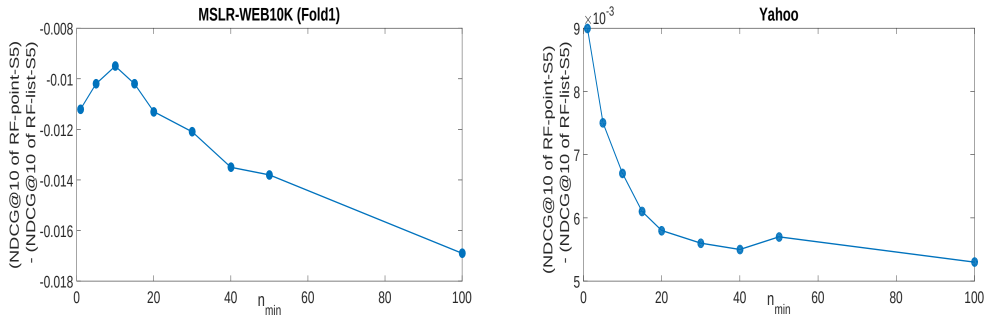


Fig. 8. Performance difference between RF-point-S5 and RF-point-S5 as n_{min} varies.

ter than that of RF-list, but performance of RF-point is still worse than RF-list (cf. Table XVII). The reason is that strength is one of the two components (the other is correlation among the trees) which control the error rate of a random forest [Breiman 2001]. This means that analyzing 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 such as reducing variance (e.g. by reducing correlation amongst 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 the curves of RF-list-breadth goes 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, below we briefly discuss two points in support of our conjecture made above that the deeper the trees, the less distinction between the effect of pointwise and listwise objective functions would be.

(1) *Adding a Termination Criterion.* Figure 7 suggested that the shorter the trees of an ensemble, the more beneficial the listwise algorithm would be. In other words, if listwise algorithm wins over its pointwise counterpart (eg. for MSLR-WEB10K), then the performance difference between pointwise and listwise algorithms is likely to increase as the trees are grown shorter (i.e., n_{min} is increased). The opposite behavior is expected to witness for Yahoo dataset. 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.
(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 MSLR-WEB10K (Fold 1) and Yahoo datasets (the performance of these settings were given in Table VI). These figures suggest that RF-point needs comparatively deeper trees to compete (be it either when it wins (Yahoo) or looses (MSLR-WEB10K)) with RF-list.

6.4. Discussion

The entropy-based objective function tries to isolate the similar class instances (i.e., of the same relevance level) in the leaves. As for the objective function of RF-list, it also achieves a better value if the nodes of a tree which have predominantly relevant documents also have less number of irrelevant documents (and vice versa), because, in

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 performance 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	11770	6190	3.09	6.36	0.925	1.003
Yahoo	8597	6366	2.76	3.81	1.359	1.399

doing so, the scores of those nodes will be higher, and hence the NDCG values will be higher. In addition, using the same reasoning, the listwise objective function prefers to create a child node containing, for example, lots of instances of labels 2 and some instances of label 1 to lots of instances of labels 2 and some instances of label 0 – simply because the latter drags the score for the *relevant leaf* to be lower than it should be.

In essence, 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 our work: any nearest-neighbor based algorithm, if the score of a region is given as the average relevance labels of the instances of that region, will work well for 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 k-nearest neighbor method.

The above discussion may explain the effect of deep trees. When a tree is sufficiently deep, 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 purer, and moreover, with an eye on their ordinal relationship as explained above.

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 to 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 paper 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 was a popular baseline for many years.

Table XIII. Performance of various algorithms on big datasets. The algorithms are: RF-rand, RF-point, RF-hybrid-L6 (RF-hyb), Mart, RankSVM (rSVM), AdaRank (AdaRa), CoorAsc, RankBoost (RBoost), LambdaMart (LmMart), and BM25 score.

Data: MSLR-WEB10K

Metric	RF-rand	RF-point	RF-hyb	Mart	rSVM	AdaRa	CoorAsc	RBoost	LmMart	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

Metric	RF-rand	RF-point	RF-hyb	Mart	rSVM	AdaRa	CoorAsc	RBoost	LmMart	BM25
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 ✓ means 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			✓✓

(3) AdaRank [Xu and Li 2007]: a listwise algorithm based on 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 random forest can also be thought as optimizing the objective function in a coordinate-wise fashion.
(5) RankBoost [Freund et al. 2003]: a pairwise algorithm based on 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 which were not discussed in Section 3) are given in Appendix F. We also show results of BM25 scorer which is very popular in the IR community as a single feature[12].

7.1. Absolute Performance

Tables 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 rest of the algorithms perform quite poorly as compared to these four.

For MSLR-WEB10K dataset, LambdaMart is the winner by considerable margin from its nearest competitor which is RF-hybrid-L6. RF-point and Mart perform similarly (i.e., there is no statistically difference between them). RankSVM, RankBoost and AdaRank are the lowest three performers here. Interestingly, in spite of partitioning

[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.

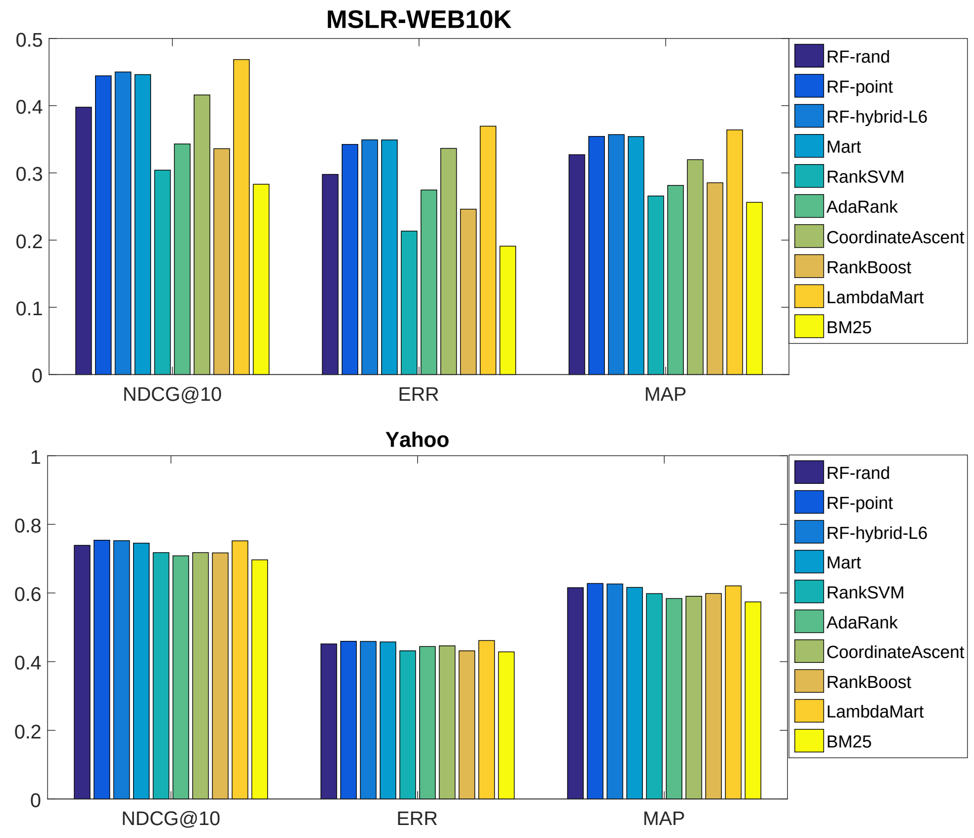


Fig. 9. Performance of different algorithms.

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. However, its performance is significantly worse than RF-point on both datasets, indicating that an objective-based splitting criterion (in this case the entropy-based one) does help in an RF framework. Given 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. CoorAsc is, in spite of being a listwise algorithm, outperformed by RF-point by 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[13].

Both the RF-hybrid-L6 and LambdaMart perform better on MSLR-WEB10K than on Yahoo – this is revealed when we compare RF-point with these algorithms across the two datasets. This raises the questions as to whether the characteristics of Yahoo dataset cause the tree ensemble based listwise algorithms to perform compara-

[13]We note here that improved performance could possibly be achieved for some of these algorithms through extensive parameter tuning. The comparison made here is nonetheless fair because we have not performed parameter tuning on any of the algorithms.

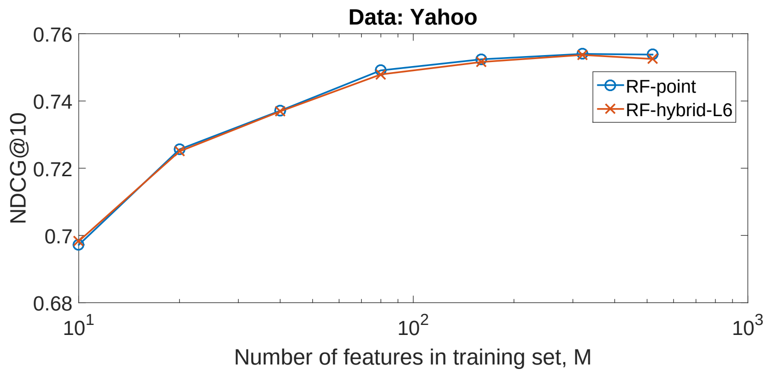


Fig. 10. With various M (in log scale), performance of RF-point and RF-hybrid-L6 on Yahoo dataset. The rightmost point corresponds to the original training set (i.e., $M = 519$)

tively less effectively. As we mentioned earlier that 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 versus 519 in Yahoo), the number of documents per query (120 in MSLR-WEB10K versus 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 vs. variance reduction strategies on the performance of rank-learning algorithms.

Further investigation into performance trade-off between pointwise and listwise algorithms on Yahoo: We now extend our analysis on relative performance of RF-point and RF-hybrid on 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 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 0.01 level. This experiment thus indicates that the larger number of features in Yahoo dataset (with respect to the MSLR dataset) is probably not the reason for the reduced effectiveness of the RF-hybrid over the pointwise one. Thus the question as to why RF-point performs relatively well on Yahoo dataset still remains open.

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

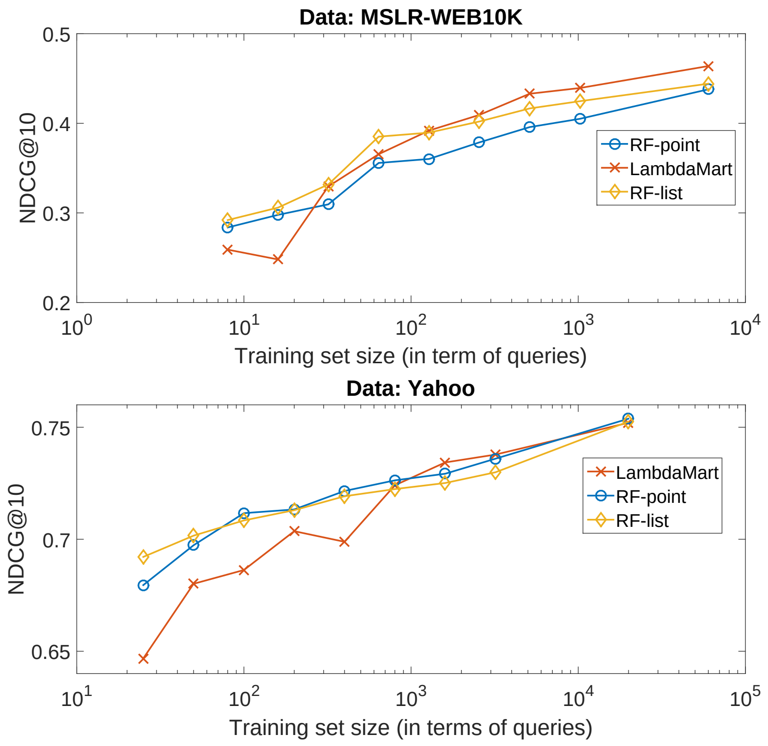


Fig. 11. Effect of training sample (in log scale) on performance of RF-point, RF-list and LambdaMart. Recall that RF-list is computationally not feasible to learn from the entire training data (which is the last point of a curve), so we show performance of RF-hybrid-L6 for that point.

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 Yahoo dataset RF-list shows an interesting characteristics: it is the best with very small training sets, but eventually becomes the worst (among the three) with larger training sets.

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 paper and also that of LambdaMart. On MQ2007 dataset which is the largest (in terms of number of queries) among the six shown in the table, LambdaMart performs slightly better than its closest competitor 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 sub-section 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 labelled 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 dataset.

Table XV. Performance comparison between various RF-based algorithms and LambdaMart on small datasets. For the RF-based algorithms (which are non-deterministic), each metric is the average of five independent runs (and each run is averaged over five folds).

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 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 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 to RankSVM (pairwise), RankBoost (pairwise), Coordinate Ascent (listwise), AdaRank (listwise), and marginally better than Mart (pointwise) on big datasetes. 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.), non-linear models tend to perform better for large scale LtR. This raises a question: should more focus be devoted on 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 (LtR) problem. While these algorithms have demonstrated competitive performance in comparison with other (oftentimes more complicated) techniques, their performance trade-offs are still largely unknown. An attempt has been made in this paper 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 which can directly optimize an IR evaluation metric of choice. Experimental results have shown the listwise approach to outperform the pointwise one across most medium sized datasets. For large datasets, a listwise algorithm can be computationally prohibitive, so we developed a hybrid algorithm which uses listwise splitting in 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.

We then investigated further the relative generalisation performance of the listwise and pointwise approaches. We first investigated an unsupervised partitioning strategy (using random splitting) in order to quantify the importance of splitting criterion. Results on the smaller datasets indicated that the splitting criteria may not be most important aspect for controlling the error rate, while for the larger datasets the splitting criteria does appear to have a significant effect on performance. Secondly, we analyzed the predictive accuracy (strength) of individual trees learnt by the pointwise and listwise algorithms, and noted that the predictive power of the trees became more similar the deeper they were grown. Finally, we investigated the effect of modifying the discount factor used in Normalized Discounted Cummulative 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 generaliazation performance of the algorithm.

We have compared RF-based LtR algorithms with several state-of-the-art methods from pointwise, pairwise and listwise categories. This 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, we have found that RF-based algorithms perform very well with very small training sets.

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

— Analyzing different characteristics of the datasets such as relevance label distribution, 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 when growing a tree.

REFERENCES

Gérard Biau. 2012. Analysis of a random forests model. *The Journal of Machine Learning Research* 98888 (2012), 1063–1095.

Gérard Biau, Luc Devroye, and Gábor Lugosi. 2008. Consistency of random forests and other averaging classifiers. *The 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 *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*. IEEE, 51–58.

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

Yoav Freund and Robert E Schapire. 1995. A desicion-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). *Ann. Statist* 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 Third 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. *Geoscience and Remote Sensing, IEEE Transactions on* 43, 3 (2005), 492–501.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. The Elements of Statistical Learning. (2009).

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, In Press.

Hemant Ishwaran. 2013. The effect of splitting on random forests. *Machine Learning* (2013), 1–44.

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 eighth 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).

Yi Lin and Yongho Jeon. 2006. Random forests and adaptive nearest neighbors. *J. Amer. Statist. Assoc.* 101, 474 (2006), 578–590.

Tie-Yan Liu. 2011. *Learning to rank for information retrieval*. Springerverlag Berlin Heidelberg.

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. (2010).

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. (2004).

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 JC 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.

Online Appendix to: Comparing Pointwise and Listwise Objective Functions for Random Forest based Learning-to-Rank

MUHAMMAD IBRAHIM, Monash University, Australia
MARK CARMAN, Monash University, Australia

A. RESULTS OF DIFFERENT NODE EXPLORATION STRATEGIES

Table XVI. Results of RF-list-S5 with different node exploration strategies on Fold 1 of MSLR-WEB10K Dataset.

Traversal Strategy	NDCG@10	MAP
breadth-first	0.4225	0.3411
random-first	0.4243	0.3430
relevant-first	0.4238	0.3443
biggest-first	0.4230	0.3425

B. DERIVATION OF TIME COMPLEXITIES

Here we derive the training time complexities for the RF-point and RF-list.

Assuming that the recursive partitioning procedure results in a balanced tree of depth $\log N$ (where N is the total number of training instances), then the cost of computing the tree can be written as:

$$cost_{tree} = \sum_{h=0}^{\lfloor \log N \rfloor} 2^h cost_{node}(h) \tag{16}$$

where $cost_{node}(h)$ denotes the cost of computing a split-point for a node at depth h in the tree. (Note that in a balanced tree there will be 2^h nodes at depth h.)

B.1. RF-point

For the pointwise algorithm, at each node a set of $\log M$ candidate features are investigated in order to determine the best split point. Each node at depth h will contain $\frac{N}{2^h}$ data points on average. For each candidate feature this data will need to be both sorted (an $\mathbf{O}(n \log(n))$ operation) and iterated over to determine the best split point (an $\mathbf{O}(n)$ operation – using running sums for computing entropies). Thus the cost per node can be calculated as follows:

$$cost_{node}(h) = \mathbf{O}(\log(M)(\frac{N}{2^h}\log(\frac{N}{2^h}) + \frac{N}{2^h})) = \mathbf{O}(\log(M)\frac{N}{2^h}\log(\frac{N}{2^h})) \tag{17}$$

And the total cost to build a tree in a pointwise manner is given by:

$$cost_{tree} = \sum_{h=0}^{\lfloor \log N \rfloor} 2^h \mathbf{O}(\log(M)\frac{N}{2^h}\log\frac{N}{2^h}) = \mathbf{O}(\sum_{h=0}^{\lfloor \log N \rfloor} \log(M) N \log\frac{N}{2^h})$$

DOI: http://dx.doi.org/10.1145/0000000.0000000

$$\begin{aligned} &= \mathbf{O}(\log(M)N \sum_{h=0}^{\lfloor \log N \rfloor} \log(N) - h) \\ &= \mathbf{O}(\log(M)N \log^2(N)) \end{aligned} \tag{18}$$

B.2. RF-list

For the listwise algorithm, we again, for each of the $\log M$ candidate features, need to sort and iterate over $\frac{N}{2^h}$ data points at each node. Evaluating the objective function for each possible split-point, however, now requires iterating over all 2^h leaves of the tree and computing a DCG value for each of Q queries. If we use an appropriate data structure, where the leaves are maintained in sorted order of average relevance value (such that insertion/deletion is logarithmic in the number of leaves) and the relevance label distribution of documents for each query is maintained at each leaf node (such that Equation 12 can be used to update the DCG value of a query independently of its number of documents at the leaf), the cost of recomputing NDCG (across all queries) at each split point is given by:

$$cost_{NDCG}(h) = \mathbf{O}(\log(2^h) + 2^h Q) = \mathbf{O}(2^h Q) \tag{19}$$

Thus the cost to identify the best split point at each node becomes:

$$\begin{aligned} cost_{node}(h) &= \mathbf{O}(\log(M)(\frac{N}{2^h}\log(\frac{N}{2^h}) + \frac{N}{2^h} cost_{NDCG}(h)) \\ &= \mathbf{O}(\log(M)(\frac{N}{2^h}\log(\frac{N}{2^h}) + NQ)) \end{aligned} \tag{20}$$

And the total cost to build a tree in a listwise manner is given by:

$$\begin{aligned} cost_{tree} &= \sum_{h=0}^{\lfloor \log N \rfloor} 2^h \mathbf{O}(\log(M)(\frac{N}{2^h}\log(\frac{N}{2^h}) + NQ)) \\ &= \mathbf{O}(\log(M)N(\sum_{h=0}^{\lfloor \log N \rfloor} \log(N) - h + 2^h Q)) \\ &= \mathbf{O}(\log(M)N(\log^2(N) - \log(N) + Q2^{\log(N)})) \\ &= \mathbf{O}(\log(M)N(\log^2(N) + QN)) \end{aligned} \tag{21}$$

C. FOLD-WISE RESULTS FOR RF-POINT & RF-LIST ON SMALLER DATASETS

Table XVII shows detailed results of RF-point and RF-list on three datasets, namely, Ohsumed, MQ2008 and MQ2007 along with the p-values.

D. PERFORMANCE VERSUS ENSEMBLE SIZE

This section discusses the effect of ensemble size on performance.

As mentioned in Section 5 that throughout the paper we have used 500 trees per ensemble. This is a fairly a standard sized ensemble as compared to the literature of random forest where a few hundreds is a typically considered to be a large value. For big datasets, increasing the ensemble size becomes even less important because the goal to increase the ensemble size is to reduce individual tree variances, and for big datasets the individual tree variance is already comparatively smaller. That said, there is no harm to increase the number of trees as random forest does not overfit in terms

Table XVII. Statistical significance test results for MQ2007, MQ2008 and Ohsumed datasets. The best value is in **bold** if p-value $\leq$ 0.05.

# Training Queries	# Test Queries	Fold	NDCG@10 RF-point	NDCG@10 RF-list	NDCG@10 p-value	MAP RF-point	MAP RF-list	MAP p-value
Data: MQ2007								
1017	339	1	0.4733	0.4652	0.844	0.4781	0.4790	0.442
		2	0.4251	0.4331	0.064	0.4501	**0.4575**	0.032
		3	0.4386	**0.4527**	0.008	0.4479	**0.4577**	0.028
		4	0.4154	0.4241	0.056	0.4346	0.4421	0.058
		5	0.4317	**0.4457**	0.013	0.4506	**0.4668**	0.0003
Data: MQ2008								
468	156	1	0.2170	0.2239	0.119	0.4546	**0.4645**	0.044
		2	0.1627	0.1709	0.055	0.4307	0.4318	0.431
		3	0.2477	0.2522	0.229	0.4516	0.4622	0.070
		4	0.2761	**0.2922**	0.024	0.5015	**0.5205**	0.0071
		5	0.2189	0.2236	0.098	0.5147	0.5098	0.754
Data: Ohsumed								
63	21	1	0.3201	0.3474	0.077	0.3079	**0.3354**	0.008
		2	0.4415	0.4420	0.494	0.3982	0.4158	0.208
		3	0.4154	**0.4536**	0.024	0.4381	**0.4533**	0.034
		4	0.4553	0.4800	0.186	0.4823	**0.5082**	0.022
		5	0.4612	0.4656	0.374	0.4437	0.4506	0.057

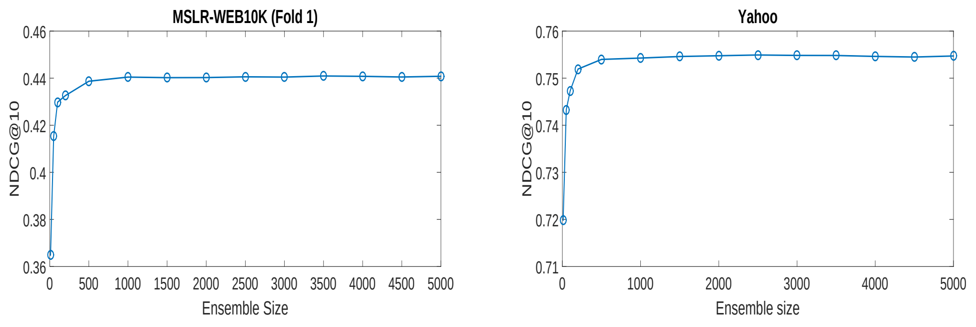


Fig. 12. Effect of ensemble size on performance of RF-point.

of ensemble size. Figure 12 shows plot for performance of RF-point as we increase ensemble size. We see that for both the datasets up to 500 there is some improvement. Then for MSLR-WEB10K a very small improvement is visible when ensemble size is increased to 1000 from 500, but for Yahoo dataset the improvement is not visibly clear. Further increase in ensemble size does not result in visibly considerable improvement. To be certain about the significance of the differences, we performed significance test for both datasets between every two pairs after the size of 500, and found that for MSLR-WEB10K there is very small improvement (at $p < 0.05$ level) if ensemble size is increased beyond 500, but no significant improvement was found for ensembles larger than 1000. Even when we reach 5000 from 500 directly, the improvement is equivalent to 1000. For Yahoo dataset the improvement is even smaller – no significant improvement was found beyond 500. Thus this experiment suggests that an ensemble of size 500 is reasonably sufficient.

E. PERFORMANCE AND TRAINING TIME VERSUS SUB-SAMPLE SIZE

This section shows how training time is affected as the sub-sample per tree of an ensemble is varied.

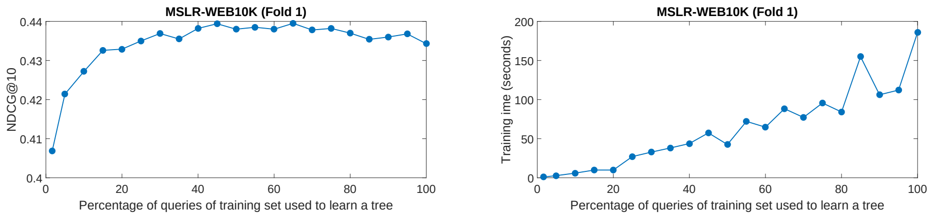


Fig. 13. Performance and training time MSLR-WEB10K (Fold 1) as sub-sample size per tree varies.

Figure 13 shows the training time per tree as we increase sub-sample size per tree. The fluctuations in training time are probably due to the fact that the machine where we ran these experiments was also running some other experiments simultaneously. This, however, is not a problem for our analysis as the trend of the plots can be vividly seen.

For MSLR-WEB10K, the relative speed-up of one of the best configurations with smaller samples (which is 30%) is approximately 2 times over the baseline (i.e., using 63% queries). As for the model size, the ratio of tree size is approximately 2. Thus we see that using smaller sub-sample to learn a tree reduces learning time significantly.

F. IMPLEMENTATIONS USED AND PARAMETER SETTINGS FOR BASELINE ALGORITHMS

Mart [Li et al. 2007] is based on gradient boosted regression tree ensemble [Friedman 2001] and is a pointwise algorithm. We use a publicly available implementation in RankLib[14], setting its parameters as follows: number of trees = 500, number of leaves for each tree = 7 (according to [Hastie et al. 2009, Ch. 10], any value between 4 and 8 will work well). The rest of the parameters are kept unchanged.

RankSVM [Joachims 2002] is a pairwise linear LtR algorithm based on SVM, and has been used as a standard method for many years. We use the publicly available source code[15].

RankBoost [Freund et al. 2003] is a pairwise algorithm which uses AdaBoost framework. Instead of using standard exponential loss, it uses a pairwise loss. We use the implementation in the RankLib package with the number of trees set to 500.

AdaRank [Xu and Li 2007] was discussed in Section 3. We use the implementation in RankLib with the number of base learners set to 500.

CoorAsc [Metzler and Croft 2007] was discussed in Section 3. We again use the implementation in RankLib.

LambdaMart [Wu et al. 2010] was discussed in Section 3. A variation of this algorithm won the Yahoo LtR Challenge [Chapelle and Chang 2011]. We use an open-source implementation of it mentioned in [Ganjisaffar et al. 2011a][16]. The parameter settings are as follows: number of trees = 500, number of leaves for each tree = 31 ([Ganjisaffar et al. 2011b] report that value close to this has been found to be worked well for MSLR-WEB10K). The rest of the parameters are kept unchanged.

[14]https://people.cs.umass.edu/~vdang/ranklib.html
[15]http://research.microsoft.com/en-us/um/beijing/projects/letor//Baselines/RankSVM-Primal.html
[16]https://code.google.com/p/jforests/