Pattern Analysis and Applications
https://doi.org/10.1007/s10044-019-00856-6

THEORETICAL ADVANCES

# An empirical comparison of random forest-based and other learning-to-rank algorithms

**Muhammad Ibrahim**[1]

Received: 10 July 2018 / Accepted: 18 October 2019


**Abstract**
Random forest (RF)-based pointwise learning-to-rank (LtR) algorithms use surrogate loss functions to minimize the ranking error. In spite of their competitive performance to other state-of-the-art LtR algorithms, these algorithms, unlike other frameworks such as boosting and neural network, have not been thoroughly investigated in the literature so far. In the first part of this study, we aim to better understand and improve the RF-based pointwise LtR algorithms. When working with such an algorithm, currently we need to choose a setting from a number of available options such as (1) classification versus regression setting, (2) using absolute relevance judgements versus mapped labels, (3) the number of features using which a split-point for data is chosen, and (4) using weighted versus un-weighted average of the predictions of multiple base learners (i.e., trees). We conduct a thorough study on these four aspects as well as on a pairwise objective function for RF-based rank-learners. Experimental results on several benchmark LtR datasets demonstrate that performance can be significantly improved by exploring these aspects. In the second part of this paper, we, guided by our investigations performed into RF-based rank-learners, conduct extensive comparison between these and state-of-the-art rank-learning algorithms. This comparison reveals some interesting and insightful findings about LtR algorithms including the finding that RF-based LtR algorithms are among the most robust techniques across datasets with diverse properties.

**Keywords** Learning-to-rank · Random forest · Decision tree · Parameter settings

## 1 Introduction

Information retrieval (IR) systems are used by billions of people today. A user submits a query to an IR system (such as Google[1]) and gets a list of documents ranked by their degrees of relevance with respect to the query. Traditional IR systems accomplish this task by using heuristic functions such as tf-idf score, BM25 score, etc.; such a function takes a query representation and a document representation as input and produces a relevance score for that document.[2] These scores are then used to sort the documents in descending order of their relevance and presented to the user. From the past decade, however, researchers have been emphasizing on the importance of combining a good number of such scores in order to yield a more accurate ranking. To this end, they use the supervised machine learning framework to automatically and effectively combine different relevance signals of documents. In this setting, a training example is a query–document pair, the corresponding relevance judgement for the document with respect to the query is considered to be the ground truth label, and the features are measurements of various base rankers (e.g., tf-idf score)—this is then called the learning-to-rank (LtR) problem. Many supervised learning methods have been used so far with empirical success over conventional scoring functions [2–4]. LtR methods are applicable to a range of information retrieval tasks such as item recommendation [5], document summarization [6], query auto-completion [7], content-based image retrieval [8, 9], etc.

The rest of the article is organized as follows. Section 2 describes the mathematical problem formulation of LtR problem and the random forest (RF) framework. Section 3 discusses why it is important to investigate the RF-based rank-learning algorithms, which is followed by the contributions made in this article. Section 4 introduces the datasets to be used for our experimentation. Section 5 explores four

[1] www.google.com.

[2] Manning et al. [1] nicely explain these models.

✉ Muhammad Ibrahim
ibrahim@cse.du.ac.bd

[1] Department of Computer Science and Engineering, University of Dhaka, Dhaka 1000, Bangladesh

Published online: 28 October 2019

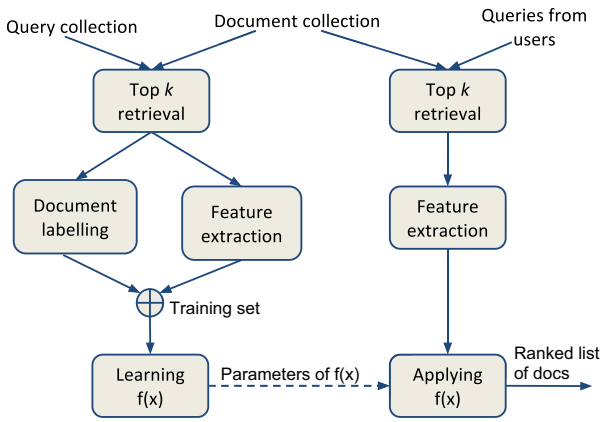


**Fig. 1** An LtR-based IR system

research directions of random forest-based pointwise LtR algorithms. Section 6 conducts extensive experiments using random forest-based and state-of-the-art LtR algorithms. Finally, Sect. 7 concludes the paper.

## 2 Background

In this section, we provide the problem formulation of learning-to-rank and a brief description of the random forest framework.

### 2.1 Problem formulation of learning-to-rank

The task of developing an LtR-based IR system can be viewed as a two-stage process [10, 11]. In the first stage, an initial retrieval approach named *Top-k retrieval* (involving one or more base rankers such as BM25 score) is used to retrieve top-$k$ documents for each query from the whole collection. After that, relevance judgements for the retrieved $k$ documents are collected usually from human judges [12, 13] and features are extracted for each of these query–document pairs. The features are then normalized on a per-query basis. These features along with the relevance labels constitute a training set. In the second stage, an LtR algorithm is employed to learn a ranking function $f(x)$ from the training set. Once the system has been trained, the system is employed in real environment. For a query submitted by the user, the previously mentioned top-$k$ retrieval approach is applied. Features are then extracted for the retrieved documents. A relevance score for each document is generated using the learned model. The documents are then ranked using these scores, and the ranked list is returned to the user. Figure 1 depicts the complete scenario.

Suppose we have a set of queries $\mathcal{Q} = \{q_1, q_2, \ldots, q_{|\mathcal{Q}|}\}$, where each query $q$ is associated with a set of $n_q$ documents $\mathcal{D}_q = \{d_{q,1}, d_{q,2}, \ldots, d_{q,n_q}\}$. Each query–document pair is further associated with a feature vector of length $M$ and a relevance label $l$, such that a dataset $\mathcal{D}$ consists of $N = \sum_q n_q$ feature vectors and corresponding labels. The feature vector contains the scores of other simple rankers, for example the cosine similarity score, BM25 score, etc. These scores are usually normalized at the query level so that the values lie between 0 and 1 [14]. A ranking function $f : \mathbb{R}^M \rightarrow \mathbb{R}$ is learnt on these data that assigns scores to feature vectors, i.e., query–document pairs. Given a query, the vector of scores produced by this function can then be sorted in decreasing order to produce a ranking.

Given a training set, the goal of an LtR algorithm is thus to find the ranking function that minimizes a loss function over the training set. 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 Normalized Discounted Cumulative Gain (NDCG). However, the loss functions based on such IR metrics are difficult to optimize directly mainly because they are almost invariably non-convex. Hence, the researchers have attempted a range of heuristics to define the loss function which have given rise to a large number of LtR algorithms.

For a more rigorous mathematical formulation of the problem of LtR, the readers are encouraged to go through Ibrahim and Carman [15].

### 2.2 Random forest

The basic building block of a random forest [16] is a decision tree. A decision (classification/regression) tree partitions the data recursively so that the data at each of the final partitions (i.e., leaves of the tree) take part in deciding the label of a unseen example. In a classification tree, data partition criterion is chosen so as to minimize the zero-one loss, whereas in a regression tree the mean squared error is minimized during partition. In order to recursively divide the data, a predefined subset of features/attributes is randomly chosen at each node of a tree, and for each attribute, the data are sorted in ascending order, and each of these possible data partitions are evaluated using the objective function, and the split-point that minimizes the objective function is thus chosen. While such a tree has low bias, the downside is its high variance. A random forest is built to retain the benefit of decision trees but at the same time to almost remove their (high) variance. Such a forest aggregates the predictions of a large number of decision trees. Each tree is learnt using a perturbed training data, and sufficiently deep trees are learnt. Each tree can be learnt in parallel. Together, these properties render a conceptually simple, but effective and efficient learning framework. More on random forests will be provided as we wade through the paper.

## 3 Motivation and contributions

In this section, we first discuss the potential benefits of a random forest over many other methods. We then highlight the fact that in spite of having several appealing characteristics, the use of random forests for solving the LtR problem, unlike many other methods, has not been thoroughly investigated so far—hence this area needs further investigation.

### 3.1 Advantages of random forests

After Hastie et. al [17], Breiman [16] and others, some benefits of a random forest over many other state-of-the-art methods are given as follows.

1. It can capture complex interactions among features by learning a nonlinear combination of them.
2. It can work equally well with continuous, discrete or even missing values with no/little modification.
3. It is relatively robust to outliers, i.e., noisy examples, and also to noisy labels.
4. It is completely parallelizable because the trees are independent of each other.
5. Very few parameters need to be tuned, if at all.
6. No normalization of the features is needed.
7. It is less affected by uninformative features.
8. It has an embedded mechanism to select the most useful features.

As for the performance of a random forest in various learning domains, it, despite having conceptual simplicity, is comparable to, or quite often better than state-of-the-art algorithms ([17, Ch. 15], [18–21]). To mention a few comments by notable researchers, Hastie et al. [17, Ch. 15] write: "In our experience random forests do remarkably well, with very little tuning required." According to Biau [22], "random forests have emerged as serious competitors to state-of-the-art methods such as boosting and support vector machines." Similar comments can be found in other studies.

A limitation of the random forests is, unlike a single decision tree, an actionable plan is not directly evident given an instance. That is, the outcome of the model is not directly human-interpretable. Eliciting a human-interpretable solution is useful in different domains such as medical research. However, research is emerging to mitigate this drawback (e.g., [23]).

Since its inception in 2001, the random forest has been found to be very effective, and hence quite popular in a variety of applications, such as bioinformatics [24], digital image processing [20], human computer interaction [25], and geographical data analysis [26].

### 3.2 Need for further research in random forest-based LtR

In a very well-known LtR challenge organized by Yahoo! [27], many of the top-ranked participants used some form of the randomized tree ensemble methods. (Geurts and Louppe [28] and Mohan et al. [29] are two of them.) The organizers [27], after their experience in the challenge, point out that the success of tree ensemble-based algorithms may be attributed to the fact that these models can capture complex nonlinear interactions among the features (which are oftentimes individually weak). A recent report on LtR [30] also reaffirms the superiority of tree ensemble-based LtR methods.

In the previous subsection, we have discussed that the random forest enjoys several benefits over many other methods. We also discussed that the random forest-based (relatively straightforward) LtR algorithms have already exhibited competitive performance to the state-of-the-art methods. In the literature, significant research has been conducted on other state-of-the-art techniques in the context of LtR such as SVM, neural networks and boosting [3], which, however, is not the case with the random forest. The only in-depth work on RF-based LtR is done by Ibrahim and Carman [15] where the authors propose a listwise version of RF-based rank-learning algorithm. However, the work lacks (1) thorough investigation into RF-based pointwise algorithms and (2) extensive comparison between RF-based algorithms and state-of-the-art rank-learning algorithms. Considering these facts, we believe it to be imperative to further investigate the random forest-based LtR algorithms. That is why, this research is undertaken.

In the first part of this article, we conduct four investigations regarding the right setting of an RF-based LtR algorithm. Regarding the recently proposed listwise version of RF-based LtR algorithm [15], the authors admit that their algorithm is computationally too costly to be used for extensive experimentation. Due to computational constraints, we conduct our empirical investigation using the RF-based pointwise algorithms. However, in the comparison with different LtR algorithms performed in the second part of this article, we do evaluate both the RF-based pointwise and listwise systems as well as state-of-the-art rank-learners.

### 3.3 Contributions

The following contributions are made in this article:

1. We compare the efficacy of classification and regression settings with RF-based pointwise algorithms. The existing literature does not offer an explicit guideline as to which one is preferable (if at all).

**Table 1** Statistics of the datasets (sorted by # queries)

| Characteristic | TD2004 | HP2004 | NP2004 | Ohsumed |
|---|---|---|---|---|
| Task | Topic distillation | Homepage finding | Named page finding | Medical docs |
| # Queries (overall) | 75 | 75 | 75 | 106 |
| # Queries (train) | 50 | 50 | 50 | 63 |
| # Rel. labels | 2 | 2 | 2 | 3 |
| # Features | 64 | 64 | 64 | 45 |
| # Query-doc pairs (overall) | 75000 | 75000 | 75000 | 16000 |
| # Query-doc pairs (train) | 50000 | 50000 | 50000 | 9684 |
| # Docs per query | 988 | 992 | 984 | 152 |
| # Docs of diff. labels (0/1/2/3/4) per query | 973/15 | 991/1 | 983/1 | 106/24/21 |
| Characteristic | MQ2008 | MQ2007 | MSLR-WEB10K | Yahoo |
| Task | Web search | Web search | Web search | Web search |
| # Queries (overall) | 784 | 1692 | 10000 | 29921 |
| # Queries (train) | 470 | 1015 | 6000 | 19944 |
| # Rel. labels | 3 | 3 | 5 | 5 |
| # Features | 46 | 46 | 136 | 519 |
| # Query-doc pairs (overall) | 15211 | 69623 | 1200192 | 709877 |
| # Query-doc pairs (train) | 9630 | 42158 | 723412 | 473134 |
| # Docs per query | 19 | 41 | 120 | 23 |
| # Docs of diff. labels (0/1/2/3/4) per query | 15/2.5/1 | 30/8/2 | 42/39/16/2/0.9 | 6/8/7/2/0.4 |

In the last row, 973/15 for TD2004 means that there are 973 and 15 documents of label 0 and 1, respectively, and 106/24/21 for Ohsumed means that there are 106, 24 and 21 documents of label, 0, 1 and 2, respectively. Similar notion is used for other datasets

2. We examine the effect of using the absolute relevance labels versus mapped relevance labels. Again, the current literature does not extensively explore this area.
3. We investigate the mechanism of controlling the model complexity using the number of features considered at each node of a tree.
4. Instead of using the average of the predictions of the trees of a random forest, we investigate the performance of a weighted random forest where a weight is assigned to a tree according to its individual ranking performance.
5. We develop an RF-based pairwise algorithm that greedily maximizes correctly ranked pairs of documents.
6. We provide extensive empirical comparison between various random forest-based systems (9 in total) and state-of-the-art rank-learning algorithms (6 in total) using eight publicly available datasets having diverse properties. In this process, the robust LtR algorithms across different domains are identified.

## 4 Datasets and experimental setup

Since we begin discussing our experiments from the next section, the datasets are introduced here.

We choose the LtR datasets that enjoy widespread acceptance in research community. Table 1 shows their various information. The datasets contain both the navigational and informational queries. All the datasets except the Yahoo are pre-divided by the providers into fivefold. The features of all these datasets are mostly used in academia. However, features of the Yahoo dataset (which was published as part of a public challenge [27]) 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 a training, a validation, and a test set. For the sake of better compatibility

**Table 2** Datasets sorted by different properties (MSLR-W stands for MSLR-WEB10K)

| | # Queries | # Query-doc pairs | # Features | # Relevance labels | Class imbalance | # Docs per query |
|---|---|---|---|---|---|---|
| Low | TD2004 | MQ2008 | Ohsumed | TD2004 | Yahoo | MQ2008 |
| | HP2004 | Ohsumed | MQ2008 | HP2004 | MQ2008 | Yahoo |
| | NP2004 | MQ2007 | MQ2007 | NP2004 | MQ2007 | MQ2007 |
| | Ohsumed | TD2004 | TD2004 | Ohsumed | Ohsumed | MSLR-W |
| | MQ2008 | HP2004 | HP2004 | MQ2008 | MSLR-W | Ohsumed |
| | MQ2007 | NP2004 | NP2004 | MQ2007 | TD2004 | TD2004 |
| | MSLR-W | Yahoo | MSLR-W | MSLR-W | HP2004 | HP2004 |
| High | Yahoo | MSLR-W | Yahoo | Yahoo | NP2004 | NP2004 |

with the existing literature, all the algorithms in this paper are trained using these predefined training sets.

Further details about the datasets can be found in Qin et al. [14], in Letor website[3], in Microsoft Research website[4], and in Chapelle et al. [27].

Table 2 arranges the datasets based on their various characteristics.

Regarding the evaluation metrics, we use two widely known measures, namely NDCG@10 and MAP. Since the big datasets are of main concern for many, for these datasets we add another relatively recently proposed metric which is ERR.[5] The ensemble size, i.e., the number of trees in an RF, is 500, and unpruned trees are learnt.

## 5 Determining the right settings of RF-based algorithms

This section conducts four distinct investigations.

### 5.1 RF-pointwise with classification or regression setting?

This section investigates either a classification or a regression setting should be used with RF-based pointwise algorithms.

#### 5.1.1 Classification model

Since it has been proven that the ranking error is bounded by both the classification error [33] and regression error [34], practitioners oftentimes address the LtR problem using a classification or a regression model. In this setting, by treating the relevance judgements of documents as the target variable, a classification (or regression) algorithm learns to predict the relevance label of an individual query–document pair. During evaluation, the documents associated with a query are ranked in decreasing order of their predicted relevance scores. This approach is called *pointwise* in the literature [2] because it treats the instances (i.e., feature vectors corresponding to the query–document pairs) independently from one another even if two documents are associated with the same query. The benefits of this approach include lower computational resource requirement and conceptual simplicity. From now on, we call this algorithm (i.e., where a query–document pair is considered to be a single training example) *RF-point*.

**Table 3** RF-point with classification (RF-p-cla) versus regression (RF-p-regr) settings

| Metric | RF-p-regr | RF-p-cla | RF-p-regr | RF-p-cla |
|---|---|---|---|---|
| MSLR-WEB10K | | | Yahoo | |
| NDCG@10 | ***0.4512*** | 0.4451 | ***0.7554*** | 0.7538 |
| ERR | ***0.3505*** | 0.3434 | **0.4603** | 0.4594 |
| MAP | ***0.3576*** | 0.3544 | **0.6290** | 0.6278 |
| MQ2007 | | | MQ2008 | |
| NDCG@10 | 0.4345 | *0.4360* | 0.2227 | *0.2234* |
| MAP | 0.4515 | *0.4524* | 0.4674 | *0.4693* |
| Ohsumed | | | TD2004 | |
| NDCG@10 | 0.4168 | *0.4306* | 0.3509 | *0.3513* |
| MAP | *0.4265* | 0.4213 | 0.2559 | *0.2574* |
| NP2004 | | | HP2004 | |
| NDCG@10 | 0.7624 | *0.7860* | 0.7578 | *0.8082* |
| MAP | 0.6307 | *0.6647* | 0.6730 | *0.7174* |

For two larger datasets (MSLR-WEB10K and Yahoo), 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. For smaller datasets, an average over five independent runs is reported (and each run is the result of fivefold cross-validation), and the winning value is given in italic font (note that these notions are used in all the subsequent tables of this paper unless otherwise stated)

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

[4] http://research.microsoft.com/en-us/projects/mslr/.

[5] To know details of these metrics, the reader can go through Järvelin and Kekäläinen [31], Chapelle et al. [32] and Ibrahim and Murshed [4].

**Procedure:** $BuildTree(\mathcal{D})$
**Data:** Dataset $\mathcal{D}$ having $M$ features.
**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**
          $\mathcal{D}_{left} \leftarrow \{\boldsymbol{x} \in \mathcal{D}_{node} | x_{feature} < split\}$;
          $\mathcal{D}_{right} \leftarrow \{\boldsymbol{x} \in \mathcal{D}_{node} | x_{feature} \geq split\}$;
          $gain \leftarrow Gain(\mathcal{D}_{node}, \mathcal{D}_{left}, \mathcal{D}_{right})$;
          **if** $gain > best.gain$ **then**
            $best \leftarrow \langle gain, feature, split \rangle$;
          **end**
        **end**
      **end**
      **if** $best.gain > 0$ **then**
        $leftChild \leftarrow createNode(\boldsymbol{x} \in \mathcal{D}_{node} | x_{best.feature} < best.split)$;
        $rightChild \leftarrow createNode(\boldsymbol{x} \in \mathcal{D}_{node} | x_{best.feature} \geq best.split)$;
        $nodeList.add(\{node.leftChild, node.rightChild\})$;
      **end**
    **end**
  **end**
  return $root$;
**end**

**Procedure:** $Gain_{entropy}(\mathcal{D}, \mathcal{D}_{left}, \mathcal{D}_{right})$
**Data:** Data at node and left/right child: $\mathcal{D}, \mathcal{D}_{left}, \mathcal{D}_{right}$
**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 $c$th class) is computed as: $Entropy(\mathcal{D}) = -\sum_{c=0}^{C-1} \frac{|\mathcal{D}_c|}{|\mathcal{D}|} \log \frac{|\mathcal{D}_c|}{|\mathcal{D}|}$;

**Algorithm 1:** Generic tree building procedure for a random forest with entropy-based gain (i.e., classification setting) [16].

Inspired by a successful adaptation by Geurts and Louppe [28], the "classification setting" that we employ in this article can actually be interpreted as a blend of the classification and regression settings.[6] To split a node of a tree, a classification setting (entropy-based gain[7]) is used. To assign a label to a data partition (i.e., during evaluation of a test instance), a regression setting is used as follows: When the test instance lands in a data partition (i.e., a leaf node), unlike classification where the mode of different class labels is assigned as the predicted label, the algorithm assigns as the score to the test instance the average of the relevance labels of all the documents of that data partition. Finally, in order to produce a ranked list, the documents are sorted in decreasing order of the predicted scores. Thus, the training and the testing phase adopts a classification and a regression setting, respectively. The procedure for building a tree is given in Algorithm 1.[8] Each tree is learnt using a bootstrapped (without replacement) sample of the training set, and the bootstrapping is performed on a per-query basis. The pseudo-code for building a forest is given in Algorithm 2.

[6] This algorithm is also used by Ibrahim and Carman [15].

[7] While there exist several functions to be used as splitting criterion, Marko Robnik-Sikonja [35] shows that this choice makes insignificant performance variation, if at all.

[8] While in the literature most of the implementations of a tree uses a depth-first (i.e., recursive) exploration of the nodes, the implementation shown here uses a breadth-first exploration mainly because we think that this represents a more systematic way of exploring the nodes. For an entropy-based objective function, the node exploration strategy does not affect the tree structure, i.e., the data partitions [15].

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

**Algorithm 2:** Procedure for constructing a random forest with query-level sub-sampling [16].

**Procedure:** $Gain_{sumOfSquaredError}(\mathcal{D}, \mathcal{D}_{left}, \mathcal{D}_{right})$
**Data:** Data at node and left/right child: $\mathcal{D}, \mathcal{D}_{left}, \mathcal{D}_{right}$
**Result:** Change in sum of squared error resulting from split.
**begin**
  return $squaredError(\mathcal{D}) - (squaredError(\mathcal{D}_{left}) + squaredError(\mathcal{D}_{right}))$;
**end**

Where $squaredError(\mathcal{D})$ is the sum of squared error for the data given as argument, i.e., $\sum_{x_{q,i} \in \mathcal{D}} (mean(l) - l_{q,i})^2$; $mean(l)$ is the arithmetic average of all the relevance labels of $\mathcal{D}$.;

**Algorithm 3:** Gain in the regression setting.

#### 5.1.2 Regression model

Mohan et al. [29] use a random forest consisting of regression trees in a pointwise fashion to evaluate the MSLR-WEB10K and Yahoo datasets. Geurts and Louppe [28] employ a variant of random forest, namely Extremely Randomized Trees [36], to evaluate the Yahoo dataset, and their findings do not reveal any significant performance difference between the classification and regression settings. None of these works examines performance on other (smaller) LtR datasets. We examine performance of a random forest with both the classification and regression settings, and evaluate on eight datasets. The classification model is introduced in the previous subsection where the gain is computed using entropy function (cf. the gain function of Algorithm 1). The regression setting treats the relevance judgements (e.g., 0–4) as the target variable, and then minimizes the sum of squared error of the data at a node during a split. The objective function of this algorithm is described in Algorithm 3. The rest of the steps of the algorithm are exactly the same as the classification model (Algorithms 1 and 2).

#### 5.1.3 Result analysis

Table 3 shows the results of the eight datasets. We observe two different outcomes, and the difference can be attributed to the cardinality of the datasets.[9] On the two larger datasets, namely MSLR-WEB10K and Yahoo, statistically significant differences are observed with the regression setting being the winner, and the MSLR-WEB10K dataset benefits from use of regression more than the Yahoo. For smaller datasets, however, the classification setting almost always wins over the regression setting. Thus, this investigation suggests that for big datasets the regression setting performs better than

[9] For all of the experiments of this section, for two larger datasets (MSLR-WEB10K and Yahoo), 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. For the smaller datasets, an average over five independent runs is reported (and each run is the result of five-fold cross-validation), and the winning value is given in italic font.

**Table 4** For regression setting, RF-point with standard relevance (RF-p-SR) versus mapped relevance (RF-p-MR) approaches

| Metric | RF-p-SR | RF-p-MR | RF-p-SR | RF-p-MR |
|---|---|---|---|---|
| MSLR-WEB10K | | | Yahoo | |
| NDCG@10 | 0.4512 | 0.4495 | ***0.7554*** | 0.7522 |
| ERR | 0.3505 | ***0.3596*** | 0.4603 | 0.4608 |
| MAP | ***0.3576*** | 0.3508 | ***0.6290*** | 0.6226 |
| MQ2007 | | | MQ2008 | |
| NDCG@10 | 0.4345 | *0.4350* | *0.2227* | 0.2221 |
| MAP | *0.4515* | 0.4514 | *0.4674* | 0.4659 |
| Ohsumed | | | TD2004 | |
| NDCG@10 | 0.4168 | *0.4246* | *0.3509* | 0.3418 |
| MAP | *0.4265* | 0.4172 | *0.2559* | 0.2542 |
| HP2004 | | | NP2004 | |
| NDCG@10 | *0.7578* | 0.7534 | *0.7624* | 0.7592 |
| MAP | *0.6730* | 0.6676 | 0.6307 | *0.6321* |

**Table 5** For classification setting, RF-point with standard relevance (RF-p-SR) versus mapped relevance (RF-p-MR) approaches

| Metric | RF-p-SR | RF-p-MR | RF-p-SR | RF-p-MR |
|---|---|---|---|---|
| MSLR-WEB10K | | | Yahoo | |
| NDCG@10 | 0.4451 | **0.4484** | **0.7551** | 0.7536 |
| ERR | 0.3434 | ***0.3522*** | 0.4602 | 0.4608 |
| MAP | ***0.3544*** | 0.3524 | ***0.6283*** | 0.6246 |
| MQ2007 | | | MQ2008 | |
| NDCG@10 | *0.4360* | 0.4346 | *0.2234* | 0.2222 |
| MAP | *0.4524* | 0.4508 | 0.4693 | 0.4693 |
| Ohsumed | | | TD2004 | |
| NDCG@10 | *0.4306* | 0.4281 | 0.3513 | *0.3522* |
| MAP | *0.4213* | 0.4186 | 0.2574 | *0.2578* |
| HP2004 | | | NP2004 | |
| NDCG@10 | *0.8082* | 0.8057 | *0.7860* | 0.7686 |
| MAP | 0.7174 | *0.7204* | *0.6647* | 0.6499 |

its classification counterpart, whereas for smaller dataset the opposite behavior is expected.

Guided by the findings of this investigation, for the subsequent experiments of this section we use the regression setting for the two larger datasets, and the classification setting for the remaining six datasets.

## 5.2 Ground truth labels: absolute relevance or scaled?

This section examines as to whether better performance could be achieved using the mapped relevance labels instead of absolute relevance.

### 5.2.1 Motivation and methodology

In a pointwise setting—be it a classification or a regression—the relevance labels assigned by an oracle to the query–document pairs are treated as the target variable, and the learning algorithm tries to predict exactly the target variable. However, since from the perspective of IR metrics the relevant documents are of more importance (to be placed in the top part of the returned ranked list) than the non-relevant ones, a natural idea is to stretch out the differences between the highly relevant documents more than in between the less relevant ones. One such approach is to map the relevance labels $l_{q,i}$ to $2^{l_{q,i}} - 1$. For instance, the labels {0, 1, 2, 3, 4} are mapped onto {0, 1, 3, 7, 15}.

The data partitions produced by the sum-of-square-based objective function (Algorithm 3) are affected if the ground truth labels are modified. As for the classification setting (i.e., when using an entropy-based objective function), although there is no effect of stretching out the labels on the data partitions, the assignment of scores to the final data partitions (i.e., to the leaf nodes of a tree) is affected.[10] That is, in the classification setting the mapped relevance acts as a different output encoding scheme. Hence, we experiment with both the classification and regression settings. We henceforth call the standard approach as *standard relevance* (SR), and the approach of stretching out labels as *mapped relevance* (MR).

To this end, we briefly discuss two related works. Li et al. [33], while using a gradient-boosted framework with both the classification and regression settings on proprietary datasets, study the effectiveness of the SR and MR approaches. They conclude that on their datasets the SR approach performs slightly better than the MR approach with the classification setting. The other study we find has already been discussed earlier, which is by Geurts and Louppe [28]; the authors employ a variant of random forest in both the classification and regression settings on the Yahoo dataset, and examine the effect of using the SR and MR approaches on evaluation phase only (i.e., as different output encoding schemes). Thus, the novelty of our investigation here is two-fold: (1) investigating a standard random forest framework and (2) evaluating a range of datasets.

### 5.2.2 Result analysis

The results of the SR and MR approaches with both the regression and classification settings are given in Tables 4 and 5, respectively. On the two larger datasets, the MR approach demonstrates a tendency to win over SR approach

[10] Recall that the usual practice has been to use the average relevance of the instances as the score.

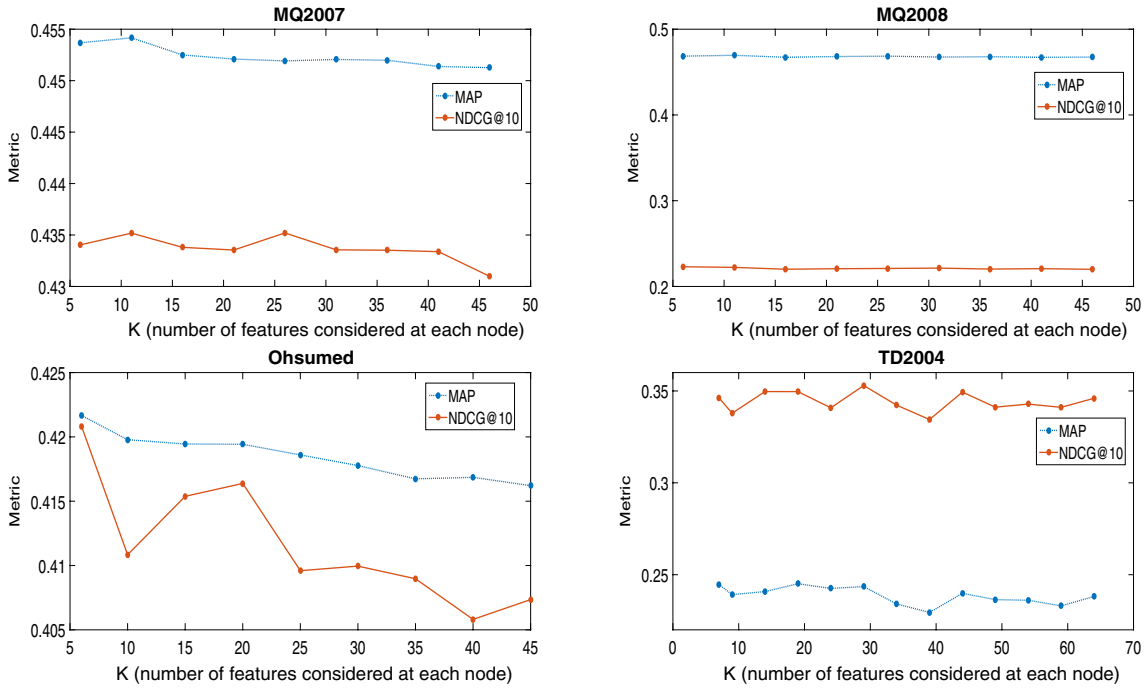


**Fig. 2** Performance variation as $K$ (i.e., number of candidate features at each node) is increased

**Table 6** Varying $K$ with regression setting on validation set, $K \in \{\log(M)+1, \sqrt{(M)}, 1/4M, 1/2M, 3/4M, M\}$

| Metric | $K=8$ | $K=12$ | $K=34$ | $K=68$ | $K=102$ | $K=136$ |
|---|---|---|---|---|---|---|
| Data: MSLR-WEB10K (Fold1) | | | | | | |
| NDCG@10 | 0.4579 | ***0.4637*** | ***0.4695*** | 0.4677 | 0.4668 | † ***0.4618*** |
| ERR | 0.3604 | ***0.3660*** | ***0.3708*** | 0.3707 | 0.3711 | † ***0.3670*** |
| MAP | 0.3665 | ***0.3695*** | ***0.3735*** | 0.3731 | 0.3715 | † ***0.3683*** |
| Metric | $K=10$ | $K=23$ | $K=130$ | $K=260$ | $K=390$ | $K=519$ |
| Data: Yahoo | | | | | | |
| NDCG@10 | 0.7464 | ***0.7496*** | ***0.7527*** | † ***0.7505*** | 0.7506 | 0.7496 |
| ERR | 0.4549 | 0.4559 | **0.4573** | 0.4573 | 0.4565 | 0.4561 |
| MAP | 0.6199 | ***0.6234*** | 0.6246 | 0.6235 | 0.6228 | †**0.6211** |

Significance test result is performed between every two consecutive settings of $K$. A † is used to highlight that performance is statistically poorer

**Table 7** With test sets, the best values of $K$ ($=1/4M$) found from validation sets

| Metric | $K=8$ ($=\log(M)+1$) | $K=34$ ($=1/4M$) | $K=10$ ($=\log(M)+1$) | $K=130$ ($=1/4M$) |
|---|---|---|---|---|
| MSLR-WEB10K | | | Yahoo | |
| NDCG@10 | 0.4510 | ***0.4622*** | 0.7542 | ***0.7606*** |
| ERR | 0.3498 | ***0.3624*** | 0.4602 | ***0.4641*** |
| MAP | 0.3570 | ***0.3636*** | 0.6281 | ***0.6338*** |

in terms of ERR, both for classification and regression, whereas in terms of NDCG@10 the opposite behavior is observed. In terms of MAP, SR approach wins in most of the datasets for both classification and regression. On smaller datasets, the SR approach mostly wins over MR approach—both for classification and regression, and in both NDCG@10 and MAP.

**Table 8** Comparison between RF-point and RF-point-weighted

| Metric | RF-point | RF-point-weighted | RF-point | RF-point-weighted |
|---|---|---|---|---|
| MSLR-WEB10K | | | Yahoo | |
| NDCG@10 | 0.4512 | 0.4513 | 0.7554 | 0.7552 |
| ERR | 0.3505 | 0.3508 | 0.4603 | 0.4603 |
| MAP | 0.3576 | 0.3576 | 0.6290 | 0.6290 |
| MQ2007 | | | MQ2008 | |
| NDCG@10 | 0.4360 | 0.4360 | 0.2234 | 0.2234 |
| MAP | 0.4524 | 0.4524 | 0.4693 | 0.4693 |
| Ohsumed | | | TD2004 | |
| NDCG@10 | *0.4306* | 0.4305 | *0.3513* | 0.3509 |
| MAP | *0.4213* | 0.4212 | *0.2574* | 0.2568 |
| NP2004 | | | HP2004 | |
| NDCG@10 | 0.7860 | *0.7861* | *0.8082* | 0.8076 |
| MAP | 0.6647 | *0.6648* | 0.7174 | *0.7178* |

Since from this experiment we have not observed any clear benefit of using the mapped relevance approach, for the subsequent experiments we stick to the conventional setting, i.e., using the original relevance labels.

## 5.3 Controlling individual tree complexity

This section investigates the optimal number of candidate features at each node of a tree.

### 5.3.1 Motivation and methodology

In a random forest, a crucial source of randomization in a tree is achieved by reducing the number of candidate features at each node (denoted by $K$), which has the default value of $\log(M) + 1$ [16]. Wager et al. [37] point out that the more the training data, the less need for randomization, because the goal of randomization is to decrease correlation, which in turn decreases the variance of the ensemble. If the training data are large, then the individual trees have already relatively low variance, thereby in less need of randomization.

In this section, we investigate as to how performance is affected while we vary individual tree complexity by controlling $K$. On the one hand, the configuration of $K = 1$ produces highly stochastic trees, which greatly de-correlates the trees but causes a high individual tree variance. On the other hand, the configuration of $K = M$, i.e., using all the features, produces comparatively low variance trees, but results in high correlation between the trees. By carefully controlling $K$, we aim to find out the optimal amount of individual tree complexity.

### 5.3.2 Result analysis

We analyze results of the two types of datasets separately.

*Relatively Small Datasets.* In Fig. 2, we plot the performance as $K$ is varied. We observe that for all the four datasets, in general there is no improvement as $K$ is increased from the default value (i.e., $\log(M) + 1$), rather in some cases there is a tendency of slight degradation of performance. This behavior was expected from our previous discussion that the smaller datasets are mostly benefited from the use of smaller $K$. Hence, we conclude that for the smaller datasets the default value of $K$ is among the best choices.[11]

*Big Datasets.* Table 6 shows the result of six different assignments of $K$, namely $\log(M) + 1$, $\sqrt{(M)}$ (this is also practiced by some researchers such as Gislason et al. [26]), $1 / 4M$, $1 / 2M$, $3 / 4M$, and $M$ (this is equivalent to bagging [38]). The results show that among the six settings, for both the datasets the setting of $K = 1/4M$ yields the best performance. Also, the performance improvement is more vivid in the MSLR-WEB10K dataset than in the Yahoo. We conjecture that the cause of performance improvement is as follows: The LtR data contains many individually weak features that, when taken together, help in effective decision making (Nayyar et al. [39] also make a similar observation). Hence, comparatively large number of features are required to generate useful data partitions.

Finally, Table 7 validates the findings emerged from the experiments of validations sets. As expected, the characteristics revealed from the validation set are also observed in test sets.

This investigation thus reveals that while for smaller datasets the default value of $K$ is among the best choices, for larger datasets it would be wise to use a higher $K$—instead

[11] Since the properties of HP2004 and NP2004 datasets are similar to that of TD2004, we do not conduct further experiments.

of the usual practice of $K = \log(M) + 1$ or $K = \sqrt{(M)}$, our experiment finds that $K \approx 1/4M$ gives significantly better performance. In general, the weaker the individual features, the more the possibility that a higher $K$ will work better.

## 5.4 Weighting the trees

This section examines the effect of assigning weights to individual trees.

### 5.4.1 Motivation and methodology

The prevalent approach to using a random forest is to compute an un-weighted average of the predictions of individual trees. It is, however, conceivable that the quality of the predictions of the individual trees is not the same. Hence, we could assign weights to the predictions of the individual trees according to their individual accuracy. Assigning weights to the trees, however, may result in overfitting by increasing the variance of the ensemble prediction. To demystify this trade-off, in this section we investigate a weighted version of the pointwise algorithm that assigns a weight to a tree based on its individual ranking accuracy.

Ibrahim and Carman [15] propose the notion of Expected NDCG. We predict the individual performance of a tree by using the Expected NDCG per tree which is evaluated using the heldout data (i.e., validation set). As such, we assign the weight of $i$th tree as follows:

$$w_i = \frac{1}{|\mathcal{D}_{\text{heldout}}|} \sum_{q \in \mathcal{D}_{\text{heldout}}} \mathbb{E}[NDCG(q; s_i(.))], \tag{1}$$

where $\mathbb{E}[NDCG(.)]$ is the Expected NDCG of query $q$ (cf. Equation 11 of [15]) based on the scores $\{s_i(\boldsymbol{x}_{q,j})\}_{j=1}^{n_q}$ produced by $i$th tree. The prediction of a test instance is then the weighted average of predictions of individual trees. We call this algorithm *RF-point-weighted*.

Regarding the existing works related to our idea, Marko Robnik-Šikonja [35], Winham et al. [40], among others, examine techniques for improving classification/regression accuracy by using various weighting schemes. Another related line of research is to search for a subset of the trees that performs better than the entire ensemble [41–43]. The novelty of our investigation as compared to these existing works is, we assign weights to the individual trees based on their individual performance on a rank-based evaluation metric, namely NDCG (instead of the classification/regression accuracy).

### 5.4.2 Result analysis

Table 8 shows the results for the eight datasets. We observe that the accuracy of the two systems (RF-point and RF-point-weighted) is strikingly similar. This is, we conjecture, due to the fact that in our experiment an ensemble is sufficiently large to offset the negative effect of the relatively poor trees. We could reduce the number of trees to check whether performance really decreases, but since our ensemble has only 500 trees which can be run on any reasonable machine, we think that this line of experiment does not contain much practical value.

Our investigation thus suggests that the standard unweighted RF-based pointwise approach performs similar to its weighted counterpart.

## 5.5 Random forest-based pairwise algorithm

So far, we have investigated RF-based pointwise objective functions, and earlier we mentioned that a listwise objective function for RF-based LtR algorithm has already been proposed [15]. The listwise algorithm greedily optimizes expected NDCG across all queries of the data as follows. For each split, the expected NDCG over all queries is calculated and the best split-point is chosen. We call this algorithm *RF-list*. Since all of the queries are involved in the decision of each split, the computational cost of this listwise algorithm is much higher than its pointwise counterpart, which oftentimes makes it impractical for very big datasets (such as MSLR-WEB10K and Yahoo). To enhance the applicability of this algorithm, the authors then design a hybrid version of their listwise algorithm which performs listwise splitting only for some predefined number of levels of a tree, and the rest of the splits are performed using traditional entropy-based or mean squared error-based gain. We call this algorithm *RF-hybrid-Lx* where $x$ is the level/depth of a tree up to which the listwise splitting criterion is used.

**Table 9** Comparison between RF-point, RF-pair, and RF-list/hybrid algorithms

| Data | Algorithms | RF-point | RF-pair | RF-hybrid-L6 | RF-pair-weighted |
|---|---|---|---|---|---|
| MSLR-WEB10K (Fold 1) | NDCG@10 | 0.4379 | 0.4010 | 0.4442 | 0.4069 |
| | ERR | 0.3405 | 0.3109 | 0.3489 | 0.3173 |
| | MAP | 0.3542 | 0.3309 | 0.3591 | 0.3337 |
| Yahoo | NDCG@10 | 0.7538 | 0.7182 | 0.7525 | 0.7198 |
| | ERR | 0.4594 | 0.4461 | 0.4590 | 0.4468 |
| | MAP | 0.6278 | 0.5948 | 0.6263 | 0.5963 |
| MQ2007 | NDCG@10 | 0.4368 | 0.4433 | 0.4442 | 0.4428 |
| | MAP | 0.4523 | 0.4632 | 0.4606 | 0.4619 |
| MQ2008 | NDCG@10 | 0.2245 | 0.2285 | 0.2326 | 0.2302 |
| | MAP | 0.4706 | 0.4755 | 0.4778 | 0.4777 |
| Ohsumed | NDCG@10 | 0.4187 | 0.4477 | 0.4377 | 0.4404 |
| | MAP | 0.4141 | 0.4426 | 0.4326 | 0.4408 |
| TD2004 | NDCG@10 | 0.3521 | 0.3529 | 0.3421 | N/A |
| | MAP | 0.2551 | 0.2520 | 0.2549 | N/A |
| HP2004 | NDCG@10 | 0.8068 | 0.7877 | 0.8032 | N/A |
| | MAP | 0.7042 | 0.7001 | 0.6910 | N/A |
| NP2004 | NDCG@10 | 0.7955 | 0.7518 | 0.7797 | N/A |
| | MAP | 0.6754 | 0.6265 | 0.6342 | N/A |

**Procedure:** $Gain_{corretlyRankedPairs}(\mathcal{D}, \mathcal{D}_{left}, \mathcal{D}_{right})$
**Data:** Data at node and left/right child: $\mathcal{D}, \mathcal{D}_{left}, \mathcal{D}_{right}$
**Result:** A measure of correctly ranked pairs of $\mathcal{D}$ resulting from split.
**begin**
  $leftDist, rightDist \leftarrow getDocDistributionPerQuery(\mathcal{D}_{left}, \mathcal{D}_{right})$;
  **if** $score(\mathcal{D}_{left}) < score(\mathcal{D}_{right})$ **then**
    $smaller \leftarrow leftDist$;
    $bigger \leftarrow rightDist$;
  **end**
  **else**
    $smaller \leftarrow rightDist$;
    $bigger \leftarrow leftDist$;
  **end**
  $correctlyRankedPairs \leftarrow 0$;
  **for** $q \in \mathcal{Q}_{\mathcal{D}}$ **do**
    $distSmaller \leftarrow smaller[q]$;
    $distBigger \leftarrow bigger[q]$;
    **for** $c1 = 0$ *upto* $|distSmaller|$ **do**
      **for** $c2 = c1 + 1$ *upto* $|distBigger|$ **do**
        $correctlyRankedPairs \leftarrow correctlyRankedPairs + w(c1, c2)distSmaller[c1]disBigger[c2]$;
      **end**
    **end**
  **end**
  return $correctlyRankedPairs$;
**end**

Where the function $getDocDistributionPerQuery(\mathcal{D}_{left}, \mathcal{D}_{right})$ computes the (absolute, i.e., non-normalized) relevance label distribution of each query of the two data partitions. As such, $leftDist$ ($rightDist$) contains a distribution for each query of the left (right) child, and $w(c1, c2)$ is any monotonically increasing function of $|c1 - c2|$.

**Algorithm 4:** Pairwise gain

In the existing literature, the LtR algorithms are oftentimes categorized into three groups based on their objective functions: pointwise, pairwise and listwise [3]. In the literature, we do not find any RF-based pairwise LtR algorithm. To make the investigation of our paper comprehensive, we could think about designing an RF-based pairwise algorithm which can be accomplished as described next. 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 their marginal relevance. After that, we shall fit a standard random forest with classification setting to the modified data. During evaluation, every pair of the documents will have a preference label predicted by the model from which it is, using a topological sort, straightforward to generate a ranking.

Although pairwise algorithms such as RankBoost [44] were popular in early times of LtR research, they essentially operate on quadratic sized training data. Hence, for big datasets it is mostly not a feasible solution, and that is mainly why in recent years they have drawn comparatively less interest of research community. We still design a greedy, yet scalable pairwise objective function as detailed in Algorithm 4. The function computes a gain for a split by counting the number of correctly ranked pairs of documents of each query of the data at the parent node. Thus the gain is, like its pointwise counterpart and unlike its listwise counterpart, computed locally, thereby causing the algorithm to be greedy. A weight can be assigned to a correctly ranked pair depending on the labels of the pair—the higher the difference between the two labels, the higher the weight (i.e., reward) for ranking the pair correctly (in fact, any monotonically increasing function of the difference can be used). We call this algorithm *RF-pair*. Note in our implementation that we avoid exhaustive enumeration (of quadratic order) of all pairs by exploiting the fact that usually the number of distinct relevance labels in LtR data does not exceed five, thereby making it scalable to big datasets.

#### 5.5.1 Result analysis

Table 9 shows results for RF-pair, along with that of the pointwise (with classification setting) and listwise algorithms for ease of comparison. With the un-weighted version, on smaller datasets, in four out of six datasets RF-pair performs slightly better than that of RF-point, and in two of these four cases its performance is less than that of RF-list. For big datasets, accuracy of RF-point is much better than RF-pair—we hypothesize that the mediocre performance of RF-pair is due to the greedy nature of the algorithm. In the weighted version, we assign the algebraic difference between the relevance labels of the two documents at hand as their weight. We see that in general no significant improvement is found by the weighted version. (Note that for datasets with binary relevance labels, namely TD2004, NP2004 and HP2004, the weighted version is not applicable.)

In general, RF-pair has been found to be (1) not considerably better than RF-list on most of the smaller datasets, and (2) much poorer than RF-point on big datasets. For these reasons, we do not further investigate RF-pair algorithm.

### 5.6 Discussion

The findings of this section are summarized as follows:

- Using the eight datasets, we compare between RF-point with classification and RF-point with regression. We find that for smaller datasets, the classification setting is more robust across different datasets than the regression setting. However, for big datasets the opposite behavior is observed.
- We compare between the use of absolute relevance (i.e., $l_{q,i}$) and mapped relevance (i.e., $2^{l_{q,i}} - 1$)—for both classification and regression settings. We find that (1) on the larger datasets there is no clear winning approach, and (2) on the smaller datasets the conventional practice of using the absolute relevance labels marginally wins over the mapped relevance approach.
- We examine the effect of varying tree complexity by controlling $K$. For smaller datasets, we do not find any reliable change in performance. For big datasets, however, $K \approx 1/4M$ yields significantly better performance than the default value (i.e., $\log(M) + 1$). As such, even a non-extensive tuning of $K$ has been found to improve the performance significantly.
- We investigate a weighted RF-based pointwise algorithm where each tree is assigned a weight based on its individual ranking performance. The investigation suggests that RF-point performs similar to its weighted counterpart.
- We devise an RF-based pairwise algorithm which greedily maximizes correctly ranked pairs of documents. However, in our experiments it was not found to be better than its pointwise counterpart.

## 6 Comparison with other algorithms

Although the focus of this paper is RF-based LtR algorithms, to get a global picture it is imperative to compare these algorithms with the state-of-the-art rank-learners, especially the tree ensemble-based ones. This section is devoted to such a comparison. As mentioned earlier, Ibrahim and Carman [15] conduct limited experiments on a small number of random forest-based rank-learners and state-of-the-art algorithms. We maintain their experimental settings

**Table 10** Performance of various RF-based algorithms on smaller datasets

| Data | Metric | RFR | RFPC | RFPCS | RFPR | RFPRS | RFL | RFLS |
|---|---|---|---|---|---|---|---|---|
| MQ2007 | NDCG@10 | 0.4314 | 0.4360 | 0.4439 | 0.4345 | 0.4423 | 0.4433 | **0.4442** |
| | MAP | 0.4493 | 0.4524 | 0.4604 | 0.4515 | 0.4588 | **0.4613** | 0.4611 |
| MQ2008 | NDCG@10 | 0.2228 | 0.2234 | 0.2286 | 0.2227 | 0.2269 | 0.2313 | **0.2320** |
| | MAP | 0.4700 | 0.4693 | 0.4735 | 0.4674 | 0.4719 | 0.4774 | **0.4777** |
| Ohsumed | NDCG@10 | 0.4351 | 0.4306 | **0.4499** | 0.4168 | 0.4468 | 0.4443 | 0.4490 |
| | MAP | 0.4311 | 0.4213 | **0.4450** | 0.4265 | 0.4421 | 0.4332 | 0.4430 |
| TD2004 | NDCG@10 | 0.3299 | 0.3513 | **0.3692** | 0.3509 | 0.3627 | 0.3558 | 0.3622 |
| | MAP | 0.2275 | 0.2574 | **0.2675** | 0.2559 | 0.2629 | 0.2586 | 0.2627 |
| HP2004 | NDCG@10 | 0.8048 | 0.8082 | **0.8203** | 0.7578 | 0.7957 | 0.7935 | 0.8035 |
| | MAP | 0.7040 | 0.7174 | **0.7196** | 0.6730 | 0.7117 | 0.7031 | 0.7159 |
| NP2004 | NDCG@10 | 0.7480 | 0.7860 | 0.7963 | 0.7624 | **0.7993** | 0.7675 | 0.7991 |
| | MAP | 0.6309 | 0.6647 | 0.6602 | 0.6307 | 0.6625 | 0.6317 | **0.6673** |

Each metric is the average over five independent runs, and each run is the average over fivefold. For RF-point(/list)-S%, the sub-sample (per tree) percentage and size in terms of queries are (for the six datasets, in the order of top to bottom): 10, 10, 6, 7, 9, and 4. The algorithms are: RF-rand (rfr), RF-point-classification (rfpc), RF-point-classification-S% (rfpcs), RF-point-regression (rfpr), RF-point-regression-S% (rfprs), RF-list (rfl), RF-list-S% (rfls)

**Table 11** Lambdamart (LMart), coordinate ascent (CoorAsc), AdaRank (AdaR), mart, RankBoost (RankB), RankSVM (rSVM) on small to moderate-sized datasets

| Data | Metric | LMart | CoorAsc | AdaR | Mart | RankB | rSVM |
|---|---|---|---|---|---|---|---|
| MQ2007 | NDCG@10 | **0.4487** | 0.4408 | 0.4324 | 0.4422 | 0.4330 | 0.4436 |
| | MAP | **0.4678** | 0.4614 | 0.4530 | 0.4608 | 0.4573 | 0.4659 |
| MQ2008 | NDCG@10 | 0.2302 | 0.2292 | 0.2220 | 0.2267 | 0.2263 | **0.2309** |
| | MAP | 0.4751 | **0.4788** | 0.4700 | 0.4730 | 0.4767 | 0.4744 |
| Ohsumed | NDCG@10 | 0.4367 | 0.4433 | 0.4476 | 0.4217 | 0.4362 | **0.4504** |
| | MAP | 0.4173 | **0.4461** | 0.4458 | 0.4322 | 0.4452 | 0.4447 |
| TD2004 | NDCG@10 | **0.3292** | 0.3074 | 0.2812 | 0.2592 | 0.3076 | 0.2913 |
| | MAP | **0.2378** | 0.2159 | 0.1899 | 0.1877 | 0.2235 | 0.2061 |
| HP2004 | NDCG@10 | 0.6398 | 0.7492 | 0.7682 | 0.6012 | 0.7160 | **0.7720** |
| | MAP | 0.5577 | 0.6433 | 0.6626 | 0.4771 | 0.6257 | **0.6720** |
| NP2004 | NDCG@10 | 0.6221 | **0.7976** | 0.6974 | 0.5866 | 0.7004 | 0.7950 |
| | MAP | 0.5006 | 0.6529 | 0.5830 | 0.4433 | 0.5591 | **0.6755** |

but extend the coverage—both in terms of algorithms and datasets—here lies the novelty of this comparison.

In terms of similar comparisons to that of ours, Tax et al. [45] collect results of different LtR papers available in the Web—the authors themselves, however, do not run any experiment at all. While this paper might be useful as a reference point due to the fact that previously experimental results on different datasets were largely scattered in different papers, we do not find any core research contribution to the LtR literature offered by this paper as explained above. Considering this fact, in what follows, we offer a rigorous and thorough comparison where we ourselves run the mostly used LtR algorithms on different datasets and compare them with the RF-based algorithms investigated in our paper.

## 6.1 Baselines

We use the following six baselines:

1. RankSVM [46]: an SVM-based pairwise algorithm which is heavily used as a benchmark algorithm in the LtR literature.
2. RankBoost [44]: a pairwise algorithm based on AdaBoost method which is a tree ensemble method.
3. AdaRank [47]: a listwise algorithm based on AdaBoost method.
4. Mart [33]: a pointwise algorithm based on gradient boosting which is also a tree ensemble method.
5. LambdaMart [48]: a listwise algorithm based on gradient boosting.
6. CoorAsc [49]: a listwise algorithm based on coordinate ascent method. Since a random forest can be interpreted

**Table 12** Using the data from Tables 10 and 11, the ranks and aggregate ranks of all algorithms (in terms of NDCG@10) across the six datasets

| Algorithm | MQ7 | MQ8 | Ohsu | TD | HP | NP | Aggregate rank |
|---|---|---|---|---|---|---|---|
| RF-rand | 9 | 8 | 9 | 6 | 3 | 8 | 42 |
| RF-point-cla | 6 | 7 | 10 | 4 | 1 | 5 | 33 |
| RF-point-cla-S% | 0 | 1 | 1 | 0 | 0 | 3 | 5 |
| RF-point-regr | 7 | 7 | 12 | 5 | 8 | 7 | 46 |
| RF-point-regr-S% | 0 | 1 | 4 | 1 | 4 | 0 | 10 |
| RF-list | 3 | 0 | 5 | 4 | 5 | 6 | 22 |
| RF-list-S% | 0 | 0 | 2 | 2 | 3 | 1 | 8 |
| LambaMart | 1 | 0 | 7 | 7 | 11 | 11 | 37 |
| CoordAscent | 1 | 1 | 6 | 9 | 9 | 2 | 28 |
| AdaRank | 8 | 7 | 3 | 11 | 7 | 10 | 47 |
| Mart | 1 | 2 | 11 | 12 | 12 | 12 | 50 |
| RankBoost | 7 | 3 | 8 | 8 | 10 | 11 | 47 |
| RankSVM | 0 | 0 | 0 | 10 | 6 | 4 | 20 |

The lower the rank, the better

as an optimization of the objective function in a coordinate-wise fashion, we include the CoorAsc algorithm in our experiments.

The parameter settings of these algorithms are given in Appendix. In addition to the results of these algorithms, we also show results of BM25 scorer which is very popular in the IR community as a single feature.[12]

Since we intend to make the empirical comparison of RF-based LtR algorithms extensive, we include a number of variations of them as detailed next. We use the RF-pointwise algorithm with both classification and regression settings as in Sect. 5.1. We did not find any clear winner among them, Ibrahim [50, 51] show that for RF-based pointwise algorithms, using a smaller sub-sample to learn a tree helps to achieve scalability and often better performance. Hence, we include their idea in our experiments which we call the *sub-sampled approach*. Besides the listwise and hybrid versions of RF-based LtR algorithms proposed by Ibrahim and Carman [15], they use yet another version of RF-based rank-learner which splits the feature space randomly (i.e., without any gain computation) which we include in our experiments. Since we discover in Sect. 5.3 that the value of the parameter $K$ does have an impact in performance, we include this dimension for experiments with big datasets.

To summarize, we use the following RF-based algorithms:

- Pointwise algorithm with classification setting (cf. Sect. 5.1, RF-point-cla).
- Pointwise algorithm with regression setting (cf. Sect. 5.1, RF-point-regr).
- Pointwise algorithm with random splitting [15] (RF-rand).
- Listwise algorithm [15] (RF-list).
- Hybrid algorithm [15] (RF-hybrid-L$x$; the value of $x$ will be mentioned as needed).
- Pointwise/listwise algorithm with subsampling [52] (the amount of data used to learn a tree will be mentioned as needed).
- Pointwise/hybrid algorithm with varying $K$ (the value of $K$ will be mentioned as needed) (cf. Sect. 5.3).

Before delving into the analysis of the experimental results, the following three points need to be discussed:

- *Selecting the right number of documents per query (k) in the top-k retrieval phase.* As discussed in Macdonald et al. [11], the choice of $k$ in the top-$k$ retrieval phase—the phase when $k$ documents are retrieved (per query) using a base ranker to prepare the training set—is likely to vary the effectiveness of the learning phase [11]. Now the question for us is, are the values for $k$ values for each of the datasets, as decided by the dataset creators (and shown in Table 1), appropriate? In other words, will the chosen values for $k$ provide for reliable performance? One of the major findings of the said work of Macdonald et al. [11] is that the navigational queries need a comparatively larger $k$ to allow us to draw a reliable conclusion (from the experimental results) about the LtR algorithms.[13] Among our eight data-

[12] Although the features are not disclosed for the Yahoo dataset, a particular feature index corresponding to BM25 is mentioned.

[13] This is intuitive since navigational queries have very few relevant documents, setting a higher value for $k$ facilitates the base ranker(s) put those few relevant documents into the training set of rank-learning phase.

sets, HP2004 and NP2004 contain navigational queries, and from Table 1, we see that around 1000 documents per query were retrieved, which is termed as "sufficient" by the said work [11]. For informational queries, the same work of Macdonald et al. finds that much smaller-sized (as small as 10–20) $k$ works well. So we can conclude that the datasets used in this paper fulfill the condition for being eligible for LtR experimentation. We note that these datasets have been used for LtR experimentations for a long time.

– *Choosing the right metric for evaluation given a particular user information need.* MAP may not be considered as a very effective choice for evaluation for navigational information need, i.e., the queries oftentimes having only one relevant document [1, Sec. 8.4]. NDCG is, however, considered to be a reasonable choice in different scenarios. We still report MAP (along with NDCG@10) on HP2004 and NP2004 datasets (which contain navigational queries) mainly for the sake of comparison with the existing works since a large number of existing works report MAP on these datasets.
– *Optimizing the right metric during learning in listwise algorithms.* In general, the idea of choosing the training metric carefully in a listwise algorithm sounds appealing. He et al. [53] and Robertson [54] speculate that the training metric that is optimized during learning need not to be the same as the evaluation metric. While Donmez et al. [55] contradict this speculation by showing that the training metric should be the same as the test metric given "sufficient" training data are available, Yilmaz and Robertson [56] later put their claim into question by arguing that sufficient training data are not always available in practice. More recently, Macdonald et al. [11] provide further insight into this ongoing academic debate by conducting extensive experiments, and reveal that optimizing the very test metric during training does not necessarily translate into a better effectiveness during evaluation. In particular, they (Macdonald et al.) discover that optimizing ERR cannot improve the effectiveness over the setting of optimizing NDCG. From this discussion, we conclude that NDCG has been rightfully a standard choice for optimization among the researchers for a range of user information needs.

## 6.2 Smaller datasets

We begin our analysis with the smaller datasets. Results of RF-rand (cf. [15]), RF-point (with both classification and regression) (cf. Sect. 5.1), RF-point-S%, i.e., RF-point with reduced sub-sample (cf. [52]), RF-list (cf. [15]), and RF-list-S% are given in Table 10. For RF-point(/list)-S%, the sizes of sub-samples are taken from the best values found in [52].[14] Since the RF-based algorithms are stochastic, we report the average metrics over five independent runs, and recall that each run contains a metric averaged over fivefold of the datasets. Performances of the six state-of-the-art algorithms are listed in Table 11.

**Table 13** BM25 and language model (LM) performance on the six datasets along with the best and worst LtR performers among the algorithms we have investigated (cf. Tables 10, 11)

| Data | Metric | BM25 | LM | Best LtR | Worst LtR |
|---|---|---|---|---|---|
| MQ2007 | NDCG@10 | 0.2956 | 0.2723 | 0.4487 (+ 152%) | 0.4324 |
| | MAP | 0.3384 | 0.3274 | 0.4678 (+ 138%) | 0.4530 |
| MQ2008 | NDCG@10 | 0.1623 | 0.1307 | 0.2320 (+ 143%) | 0.2228 |
| | MAP | 0.3588 | 0.3137 | 0.4777 (+ 133%) | 0.4700 |
| Ohsumed | NDCG@10 | 0.3992 | 0.3902 | 0.4504 (+ 113%) | 0.4217 |
| | MAP | 0.4274 | 0.4260 | 0.4447 (+ 104%) | 0.4322 |
| TD2004 | NDCG@10 | 0.1976 | 0.0854 | 0.3692 (+ 187%) | 0.2592 |
| | MAP | 0.1502 | 0.1076 | 0.2675 (+ 179%) | 0.1877 |
| HP2004 | NDCG@10 | 0.6069 | 0.3570 | 0.8203 (+ 135%) | 0.6012 |
| | MAP | 0.4984 | 0.2629 | 0.7196 (+ 144%) | 0.4771 |
| NP2004 | NDCG@10 | 0.5977 | 0.5654 | 0.7991 (+ 134%) | 0.5866 |
| | MAP | 0.5140 | 0.4741 | 0.6673 (+ 130%) | 0.4433 |

The percentage increase (or decrease) in performance of the best LtR method over BM25 is also reported in bracket

The following observations are made regarding the RF-based algorithms:

– In general, RF-rand performs better than anticipated which is interesting because completely random splitting is used here. In Ohsumed dataset, it performs even better than RF-point, while in MQ2007, MQ2008 and HP2004 its performance is close to that of RF-point.
– RF-list wins over RF-point in MQ2007, MQ2008 and Ohsumed, whereas in TD2004 it marginally wins, and in HP2004 and NP2004 it looses. This indicates that both of these algorithms should be considered for a new domain.

[14] Ibrahim and Carman [15] report results of only RF-rand, RF-point with classification and RF-list.

**Table 14** Performance of various RF-based algorithms on big datasets

| Metrics | Rand | P-cla | P-reg | P-reg-K34 | H-cla-L6 | H-reg-L6 | H-reg-L4-K34 |
|---|---|---|---|---|---|---|---|
| Data: MSLR-WEB10K | | | | | | | |
| NDCG@10 | 0.3978 | 0.4445 | 0.4510 | 0.4622 | 0.4502 | 0.4535 | 0.4656 |
| ERR | 0.2978 | 0.3424 | 0.3498 | 0.3624 | 0.3492 | 0.3531 | 0.3660 |
| MAP | 0.3271 | 0.3543 | 0.3570 | 0.3636 | 0.3570 | 0.3588 | 0.3650 |
| Metrics | Rand | P-cla | P-reg | P-reg-K130 | H-cla-L6 | H-reg-L6 | |
| Data: Yahoo | | | | | | | |
| NDCG@10 | 0.7389 | 0.7538 | 0.7554 | 0.7606 | 0.7525 | 0.7543 | |
| ERR | 0.4517 | 0.4594 | 0.4603 | 0.4641 | 0.4590 | 0.4598 | |
| MAP | 0.6156 | 0.6278 | 0.6290 | 0.6338 | 0.6263 | 0.6281 | |

*P* RF-point, *H* RF-hybrid, *cla* classification, *reg* regression, *rand* RF-rand

**Table 15** Performance of various algorithms on big datasets

| Metrics | Mart | rSVM | AdaRa | CoorAsc | RBoost | LmMart | BM25 |
|---|---|---|---|---|---|---|---|
| Data: MSLR-WEB10K | | | | | | | |
| NDCG@10 | 0.4463 | 0.3041 | 0.3431 | 0.4160 | 0.3360 | 0.4686 | 0.2831 |
| ERR | 0.3491 | 0.2134 | 0.2746 | 0.3365 | 0.2460 | 0.3695 | 0.1910 |
| MAP | 0.3541 | 0.2657 | 0.2814 | 0.3197 | 0.2853 | 0.3640 | 0.2562 |
| Data: Yahoo | | | | | | | |
| NDCG@10 | 0.7453 | 0.7177 | 0.7083 | 0.7177 | 0.7168 | 0.7520 | 0.6966 |
| ERR | 0.4575 | 0.4316 | 0.4441 | 0.4460 | 0.4315 | 0.4615 | 0.4285 |
| MAP | 0.6163 | 0.5983 | 0.5839 | 0.5906 | 0.5985 | 0.6208 | 0.5741 |

The algorithms are: Mart, RankSVM (rSVM), AdaRank (AdaRa), CoorAsc, RankBoost (RBoost), LambdaMart (LmMart), and BM25 score

**Table 16** Significance test results for comparison of different algorithms from Tables 14 and 15

| *p* value | cl, rg | cl, hc | rg, rk | rg, hr | rk, hrk | rk, lm | hrk, lm |
|---|---|---|---|---|---|---|---|
| Data: MSLR-WEB10K, Metric: NDCG@10 | | | | | | | |
| < 0.01 | rg | hc | rk | | | lm | |
| < 0.05 | | | | rg | hrk | | lm |
| *p* value | cl, rg | cl, hc | rg, rk | rg, hr | lm, cl | lm, rg | lm, rk |
| Data: Yahoo, Metric: NDCG@10 | | | | | | | |
| < 0.01 | rg | | rk | | | rg | rk |
| < 0.05 | | cl | | rg | | | |

A pairwise test is performed, and the winning algorithm (among the two given in the heading of a column) is mentioned in a cell (along with the corresponding *p* value). The examined algorithms are: RF-point-cla (cl), RF-point-reg (rg), RF-point-reg-K (rk), RF-hybrid-L6-cla (hc), RF-hybrid-L6-reg (hr), LambdaMart (lm), and RF-hybrid-L4-reg-K (hrk)

– Both RF-point and RF-list leverage from use of the smaller sub-sample approach (with an exception of HP2004). However, RF-point benefits more than RF-list—the increase in performance from RF-point to RF-point-S% is in general more than that from RF-list to RF-list-S%. The sub-sampled version reduces the ensemble variance, thereby improving the accuracy. Performance of RF-list already tends to be better than RF-point, and hence imposing further randomness on RF-list through the use of smaller sub-samples has a relatively mild (positive) effect than that of RF-point.

The following points are observed in the comparison between RF-based algorithms and other state-of-the-art algorithms:

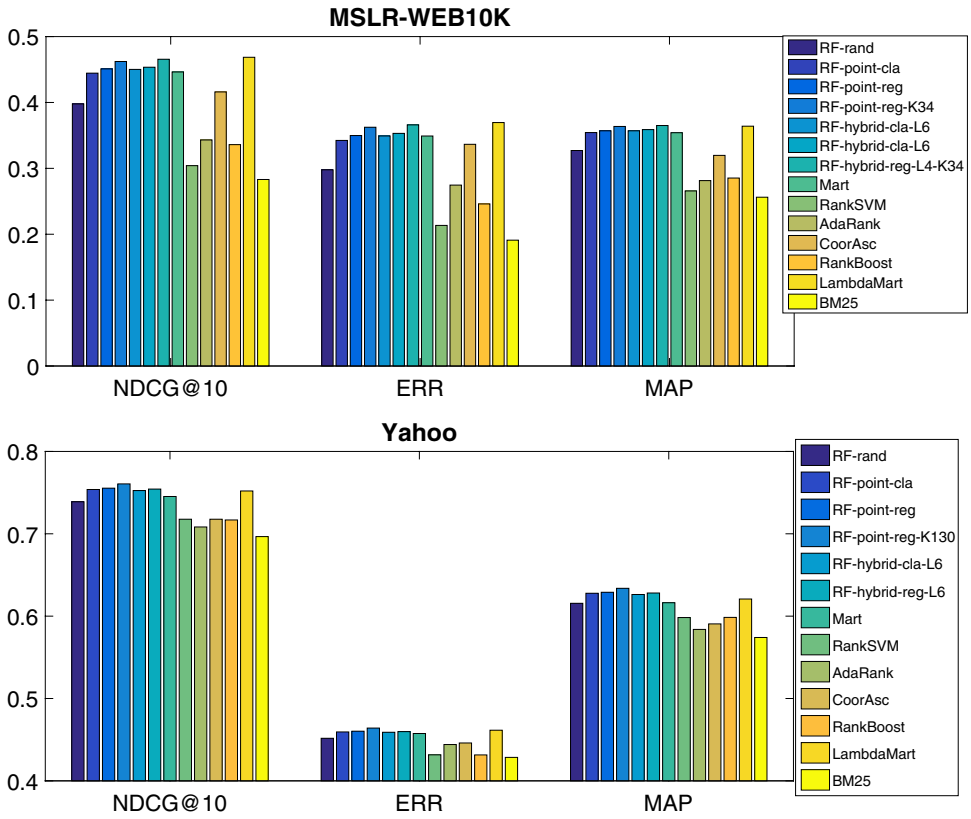


**Fig. 3** Performance of different algorithms

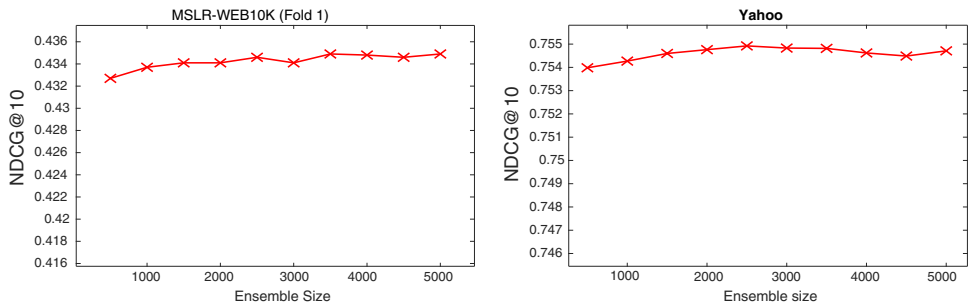


**Fig. 4** Effect of ensemble size on performance of performance on RF-point

– There is no consistent winner across different datasets. (This underlines the importance of investigating new algorithms despite the large number of algorithms in the literature.)
– In terms of robustness across different datasets, there is no sharp decrease in performance of any of the RF-based algorithms. In contrast, all other algorithms suffer heavily in one or more datasets (with varying degrees), in particular, LambdaMart in HP2004 and NP2004, CoorAsc in TD2004 and HP2004, AdaRank in TD2004 and NP2004, Mart in TD2004, NP2004 and HP2004, RankBoost in TD2004, HP2004 and NP2004, and RankSVM in TD2004.
– Although LambdaMart is considered by the LtR community to be one of the top algorithms, in our experiments it suffers from robustness across different datasets. Its performance in HP2004 and NP2004 is not near to the best. (However, on big datasets it is indeed one of the top performers as will be discussed shortly.)
– Since no consistent winning algorithm has been found across different datasets, it is difficult to draw an overall conclusion for all the algorithms on all six datasets. Hence, in Table 12 we show the aggregate rank of each algorithm across all six datasets which is calculated by counting the number of times an algorithm is beaten by some others. Concretely, assuming $A$ is the set of algorithms and $D$ is set of datasets, and $m(A_i, D_j)$ is the score of a metric of choice of $i$th algorithm on $j$th dataset, $aggregated\_rank(A_i) = \sum_{j=1}^{|D|} \sum_{k=1}^{|A|} \mathbb{1}(m(A_i, D_j) < m(A_k, D_j))$ where $\mathbb{1}(.)$ is an indicator function; the lower the aggre-

gated rank of an algorithm, the better.[15] This table demonstrates that RF-point-cla-S% is the most accurate (and also robust) algorithm across different datasets, closely followed by RF-list-S%.[16]

This investigation thus reveals that on the smaller datasets the state-of-the-art algorithms perform differently on different datasets, and that the RF-based algorithms are among the top candidates—both in terms of accuracy and robustness across different datasets. Also, on the highly imbalanced datasets where many otherwise top performers find it difficult to perform well (e.g., LambdaMart), RF-based algorithms are found to be the best. The most unbalanced datasets, namely HP2004 (homepage finding task) and NP2004 (named-page finding task), correspond to the search task of navigational queries, whereas TD2004, MQ2007 and MQ2008 datasets correspond to the search task of informational queries, and the Ohsumed dataset corresponds to a domain-specific search task. As discussed next, the search tasks of domain-specific search and enterprise search usually differ from that of conventional general-purpose search engines. Data of smaller enterprizes may possess different properties than that of web search, and their search requirements may vary from commercial search engines [57–60]. Similarly, the search requirements of domain-specific search (also known as vertical search) may not necessarily match with that of traditional general-purpose web search [57, 61]. Our findings of this subsection reveal that RF-based algorithms are likely to perform very well in these domains where getting a large amount of labeled data is comparatively difficult.

Before concluding the discussion, we show the efficacy of LtR methods as compared to two individually highly informative features, namely BM25 and Language Model in Table 13. This table shows that LtR approaches are indeed effective as compared to a conventional ranker.

## 6.3 Big datasets

While Ibrahim and Carman [15] do not run extensive experiment on smaller datasets, they conduct reasonable experiments on big datasets. Since we have investigated some RF-based new algorithms, to get a comprehensive picture of the scenario it is imperative that we put all these algorithms into a single comparison. For big datasets, the RF-based new algorithms investigated in this paper are: RF-point with regression setting, RF-point (regression) with varying $K$ values, and RF-hybrid (regression) with varying $K$ values. In our experiments, we maintain the experimental settings of the said paper.

### 6.3.1 Absolute performance

Tables 14 and 15 show the results of RF-based and other algorithms, respectively, on the MSLR-WEB10K and Yahoo datasets. Regarding the statistical significance among the differences in performance, Table 16 shows the pairwise significance test results conducted on several pairs of algorithms from Tables 14 and 15—we avoid exhaustive pairwise comparison since most of the algorithms perform relatively poorly as compared to the top performers. Using the data from Tables 14 and 15, Fig. 3 graphically shows performance of various algorithms.

In both the datasets, RF-based algorithms and LambdaMart dominate the winning list, followed by Mart; the rest of the algorithms perform comparatively poorly. Efficacy of RF-hybrid algorithm is observed in both classification (RF-point-classification) and regression (RF-point-regression) settings (on MSLR-WEB10K dataset), but the classification setting gets comparatively more benefit from it. With $K = 1/4M$, application of RF-hybrid becomes restrained by the computational resources available. On the Yahoo dataset, since the listwise objective was not found to be effective as compared to its pointwise counterpart, we do not report performance of RF-hybrid-K130 on it.[17] For the MSLR-WEB10K dataset, LambdaMart marginally wins over its nearest competitor which is RF-hybrid-regression-L4-K34. RF-point performs similar to or better than Mart.

The mediocre performance of AdaRank algorithm which adopts a listwise approach can be attributed to the fact that it combines the features linearly, and linear models may not capture the patterns of LtR data very well, especially if the dataset is large. CoorAsc is, in spite of being a listwise algorithm, outperformed by RF-point by large margin. In the Yahoo dataset, the RF-based pointwise and listwise algorithms dominate the winning list that also includes LambdaMart and Mart. Interestingly, here RF-point-K130 with regression setting wins over LambdaMart.

[15] We perform a pairwise significance test on the comparatively larger (in terms of number of queries) MQ2007 and MQ2008 datasets. Since for the rest of the datasets the number of queries is small, the significance test results may not be reliable.

[16] As explained earlier, since HP2004 and NP2004 datasets contain navigational queries, MAP may not be considered to be a very effective choice for evaluation of this type of information need [1, Sec. 8.4]. That is why, in Table 12 we chose NDCG@10 for overall comparison.

[17] We, however, did run a pilot experiment on comparison between RF-point and RF-hybrid with $K \in \{23, 50, 80, 130\}$ (the value 23 is used as $\sqrt{(M)}$), and did not observe improvement in performance of RF-hybrid over RF-point for any of the settings. We thus conclude that the $K$ is not likely to play a significant role in the relative performance of these two systems for the Yahoo dataset.

Experiments on the MSLR-WEB10K and Yahoo datasets reveal somewhat contrasting findings. Both the RF-hybrid and LambdaMart perform relatively better on MSLR-WEB10K data than on Yahoo data. So the next question to consider is whether the characteristics of the Yahoo dataset somehow lower the performance of the tree ensemble-based listwise algorithms (than the MSLR-WEB10K dataset). The following discrepancies between the two datasets (cf. Table 1) could further be investigated to get an answer to the above-mentioned issue: the number of features, sparsity in data (i.e., number of zeros which include missing values—the missing values are replaced by zero as per the standard practice [3]): MSLR-WEB10K: 37%, Yahoo: 57%, the number of documents per query, the number of relevant documents per query.

As discussed in Sects. 2.2 and 3.1, one of the core benefits of a random forest is that it is built not to overfit—one can increase the number of trees in an ensemble without any reservation. The size of the ensemble will depend on the availability of computational resources; the more these resources, the more trees should be used. Yet, to be certain about this property, we execute a pilot experiment with RF-point on the effect of ensemble size using the two large datasets, namely MSLR-WEB10K and Yahoo. The plots are given in Fig. 4. We see that as we increase the ensemble size, performance of both datasets either improve or remain similar, which is expected from the theoretical discussion mentioned above. The minor fluctuations in the plots are most likely to be due to the inherent randomness of the algorithm.

### 6.4 Discussion

Some key findings emerged from the experiments of this section regarding the RF-based algorithms are summarized as follows:

On smaller datasets:

- RF-based LtR algorithms with reduced sampling per tree helps to improve performance in both the pointwise and listwise algorithms.
- The random splitting-based algorithm performs slightly poorer or as good as the pointwise splitting-based algorithm.
- On the datasets with graded relevance, the listwise algorithm performs better than its pointwise counterpart.

On larger datasets:

- Random splitting is not as effective as it was found in smaller datasets.
- Performance of a listwise splitting criterion is likely to be dependent on the properties of the dataset in that it may or may not improve performance over its pointwise counterpart.
- Hybrid counterparts of the both classification and regression algorithms improve performance (when applicable, for example, in MSLR-WEB10K dataset).

In general, the following findings emerged from our experiments:

- Performances of various algorithms depend heavily on the properties of the datasets. Key such properties include: data size and relevance label distribution.
- RF-based pointwise/listwise algorithms consistently outperforms RankSVM (pairwise), RankBoost (pairwise), Coordinate Ascent (listwise), AdaRank (listwise), and is marginally better than Mart (pointwise) on big datasets. This shows that RF-based LtR algorithms are competitive with the state-of-the-art methods, and that is why research should put more emphasis on them.
- We have found that the nonlinear models tend to perform better for large-scale LtR. This indicates that the model complexity (e.g., in terms of bias-variance trade-off) of LtR algorithms should probably be given more emphasis in the literature which has not been the case so far. This is an interesting research direction that we address in a future work.

An important finding of our experiments regarding the smaller datasets (of diverse properties) is as follows. The RF-based LtR algorithms are found here to be robust and performing quite well, which makes them strong candidates for the domains where getting large amount of labelled training data is difficult (e.g., domain-specific search and enterprize search [62]).

Before concluding our analysis, we reiterate the fact that the random forest-based LtR algorithms are inherently completely parallelizable, and require very little tuning of their parameters. We note that different algorithms such as boosting have recently been shown to be eligible for some form of parallelization [63]. However, these techniques are not as obvious as the parallelism of random forests is, and moreover, these techniques are still subject to scrutiny in terms of performance.[18] Nonetheless, comparing the fastest LtR algorithms is still an interesting direction for future work.

Note that the BM25 scorer on its own performs poorer than all LtR systems in each of the cases which reaffirms the effectiveness of rank-learning algorithms.

[18] The inventors of the said technique [63] admit that there is a concern of degradation of accuracy, slightly though.

## 7 Conclusion

While the learning-to-rank algorithms are being used for quite some time by the practitioners, the random forest-based ones are not thoroughly investigated in the literature so far. In this article, we aimed at minimizing this gap. Firstly, we have investigated a few dimensions of random forest-based algorithms. Guided by this investigation, in the second part of the article, we have extensively compared various RF-based LtR algorithms (9 in total) and other relevant state-of-the-art algorithms (6 in total) using eight LtR benchmark datasets. Experimental results reveal a number of interesting findings which include the robust nature of RF-based algorithms across different datasets and domains.

In our experiments, the main competitor of RF-based LtR algorithms has been found to be the boosting-based ones. Since the former of these two powerful algorithms is seen as a variance reduction mechanism while the latter is seen as a bias reduction technique, a compelling direction for future research is to investigate the bias and variance of LtR algorithms in general, and RF-based algorithms in particular.

## Appendix: Implementations and parameter settings of baseline algorithms

The description of different baseline algorithms along with their parameter settings is given as follows.

RankSVM [46] is an SVM-inspired pairwise LtR algorithm which has been used as a baseline in a large number of works on LtR. In our experiments, we use a publicly available implementation of it.[19]

RankBoost [44] is an Adaboost-inspired [64] pairwise algorithm which, instead of using standard exponential loss, uses a pairwise loss. RankLib[20] is a popular package of a number of LtR algorithms from which we use RankBoost's implementation. We set the number of trees of the ensemble to 500.

AdaRank [47] also uses the AdaBoost framework but unlike RankBoost it adopts a listwise approach. We use the implementation in RankLib with the number of base learners set to 500 and NDCG@10 as the optimization metric for learning.

CoorAsc [49] algorithm uses the coordinate ascent method in a listwise manner. Its RankLib implementation is used.

A popular gradient-boosted regression tree ensemble [65]-based pointwise LtR algorithm is Mart [33]. We use its implementation of RankLib with the following change in its default parameter settings: number of trees = 500, number of leaves for each tree = 7 (according to [17, Ch. 10], any value between 4 and 8 is likely to work well).

One of the most popular LtR algorithms is LambdaMart [48]. It blends the ingenuine idea of approximated gradient of its predecessor, namely LambdaRank [66] with the gradient boosting framework [65]. In the Yahoo LtR Challenge [27], a variant of LambdaMart topped the winning list. We use an open-source implementation of it mentioned in [67].[21] The parameter settings we maintain are as follows: number of trees = 500, number of leaves for each tree = 31 (Ganjisaffar et al. [68] report that value close to this has been found to work well for MSLR-WEB10K). On smaller datasets, in order to mitigate overfitting we set it, like Mart, to 7. The rest of the parameters are kept unchanged. We note that during training, all the six baselines make use of validation sets.

## References


1. Manning CD, Raghavan P, Schütze H (2008) Introduction to information retrieval, vol 1. Cambridge University Press, Cambridge
2. Li H (2011) Learning to rank for information retrieval and natural language processing. Synth Lect Hum Lang Technol 4(1):1–113
3. Liu TY (2011) Learning to rank for information retrieval. Springer, Berlin
4. Ibrahim M, Murshed M (2016) From tf-idf to learning-to-rank: an overview. In: Handbook of research on innovations in information retrieval, analysis, and management. IGI Global, USA, pp 62–109
5. Karatzoglou A, Baltrunas L, Shi Y (2013) Learning to rank for recommender systems. In: Proceedings of the 7th ACM conference on recommender systems, ACM, pp 493–494
6. Ouyang Y, Li W, Li S, Lu Q (2011) Applying regression models to query-focused multi-document summarization. Inf Process Manag 47(2):227–237
7. Santos RL, Macdonald C, Ounis I (2013) Learning to rank query suggestions for adhoc and diversity search. Inf Retr 16(4):429–451
8. Li Z, Tang J, Mei T (2019) Deep collaborative embedding for social image understanding. IEEE Trans Pattern Anal Mach Intell 41(9):2070–2083
9. Li Z, Tang J, He X (2018) Robust structured nonnegative matrix factorization for image representation. IEEE Trans Neural Netw Learn Syst 29(5):1947–1960
10. Dang V, Bendersky M, Croft WB (2013) Two-stage learning to rank for information retrieval. In: Advances in information retrieval. Springer, pp 423–434
11. Macdonald C, Santos RL, Ounis I (2013) The whens and hows of learning to rank for web search. Inf Retr 16(5):584–628
12. Aslam JA, Kanoulas E, Pavlu V, Savev S, Yilmaz E (2009) Document selection methodologies for efficient and effective learning-to-rank. In: Proceedings of the 32nd international ACM SIGIR


[19] http://research.microsoft.com/en-us/um/beijing/projects/letor//Baselines/RankSVM-Primal.html.

[20] https://people.cs.umass.edu/~vdang/ranklib.html.

[21] https://code.google.com/p/jforests/.

conference on research and development in information retrieval. ACM, pp 468–475
13. Pavlu V (2008) Large scale ir evaluation. ProQuest LLC, Ann Arbor
14. Qin T, Liu TY, Xu J, Li H (2010) Letor: a benchmark collection for research on learning to rank for information retrieval. Inf Retr 13(4):346–374
15. Ibrahim M, Carman M (2016) Comparing pointwise and listwise objective functions for random-forest-based learning-to-rank. ACM TOIS 34(4):20
16. Breiman L (2001) Random forests. Mach Learn 45(1):5–32
17. Hastie T, Tibshirani R, Friedman J (2009) The elements of statistical learning, 2nd edn. Springer, New York. https://doi.org/10.1007/978-0-387-84858-7
18. Banfield RE, Hall LO, Bowyer KW, Kegelmeyer WP (2007) A comparison of decision tree ensemble creation techniques. IEEE Trans Pattern Anal Mach Intell 29(1):173–180
19. Caruana R, Niculescu-Mizil A (2006) An empirical comparison of supervised learning algorithms. In: Proceedings of the 23rd international conference on machine learning. ACM, pp 161–168
20. Criminisi A (2011) Decision forests: a unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. Found Trends Comput Graph Vis 7(2–3):81–227
21. Fernández-Delgado M, Cernadas E, Barro S, Amorim D (2014) Do we need hundreds of classifiers to solve real world classification problems? J Mach Learn Res 15(1):3133–3181
22. Biau G (2012) Analysis of a random forests model. J Mach Learn Res 98888:1063–1095
23. Cui Z, Chen W, He Y, Chen Y (2015) Optimal action extraction for random forests and boosted trees. In: Proceedings of the 21st ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 179–188
24. Díaz-Uriarte R, De Andres SA (2006) Gene selection and classification of microarray data using random forest. BMC Bioinform 7(1):3
25. Dong Y, Zhang Y, Yue J, Hu Z (2016) Comparison of random forest, random ferns and support vector machine for eye state classification. Multimed Tools Appl 75(19):11763–11783
26. Gislason PO, Benediktsson JA, Sveinsson JR (2006) Random forests for land cover classification. Pattern Recogn Lett 27(4):294–300
27. Chapelle O, Chang Y (2011) Yahoo! learning to rank challenge overview. J Mach Learn Res Proc Track 14:1–24
28. Geurts P, Louppe G (2011) Learning to rank with extremely randomized trees. In: JMLR: workshop and conference proceedings, vol 14
29. Mohan A, Chen Z, Weinberger KQ (2011) Web-search ranking with initialized gradient boosted regression trees. J Mach Learn Res Proc Track 14:77–89
30. Han X, Lei S (2018) Feature selection and model comparison on microsoft learning-to-rank data sets. arXiv preprint arXiv:1803.05127
31. Järvelin K, Kekäläinen J (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, pp 41–48
32. Chapelle O, Metlzer D, Zhang Y, Grinspan P (2009) Expected reciprocal rank for graded relevance. In: Proceedings of the 18th ACM conference on information and knowledge management (CIKM). ACM, pp 621–630
33. Li P, Wu Q, Burges C (2007) McRank: learning to rank using classification and gradient boosting. Adv Neural Inf Process Syst 20:897–904
34. Cossock D, Zhang T (2006) Subset ranking using regression. Learning Theory, pp 605–619
35. Robnik-Šikonja M (2004) Improving random forests. In: Machine learning: ECML 2004. Springer, pp 359–370
36. Geurts P, Ernst D, Wehenkel L (2006) Extremely randomized trees. Mach Learn 63(1):3–42
37. Wager S, Hastie T, Efron B (2014) Confidence intervals for random forests: the jackknife and the infinitesimal jackknife. J Mach Learn Res 15(1):1625–1651
38. Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140
39. Zaidi N, Webb G, Carman M, Petitjean F (2015) Deep broad learning—big models for big data. arXiv preprint arXiv:1509.01346
40. Winham SJ, Freimuth RR, Biernacka JM (2013) A weighted random forests approach to improve predictive performance. Stat Anal Data Min ASA Data Sci J 6(6):496–505
41. Bernard S, Heutte L, Adam S (2009) On the selection of decision trees in random forests. In: International joint conference on neural networks, 2009. IJCNN 2009. IEEE, pp 302–307
42. Li HB, Wang W, Ding HW, Dong J (2010) Trees weighting random forest method for classifying high-dimensional noisy data. In: 2010 IEEE 7th international conference on e-business engineering (ICEBE). IEEE, pp 160–163
43. Oshiro TM, Perez PS, Baranauskas JA (2012) How many trees in a random forest? In: MLDM. Springer, pp 154–168
44. Freund Y, Iyer R, Schapire RE, Singer Y (2003) An efficient boosting algorithm for combining preferences. J Mach Learn Res 4:933–969
45. Tax N, Bockting S, Hiemstra D (2015) A cross-benchmark comparison of 87 learning to rank methods. Inf Process Manag 51(6):757–772
46. Joachims T (2002) Optimizing search engines using clickthrough data. In: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 133–142
47. Xu J, Li H (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, pp 391–398
48. Wu Q, Burges CJ, Svore KM, Gao J (2010) Adapting boosting for information retrieval measures. Inf Retr 13(3):254–270
49. Metzler D, Croft WB (2007) Linear feature-based models for information retrieval. Inf Retr 10(3):257–274
50. Ibrahim M (2019) Sampling non-relevant documents of training sets for learning-to-rank algorithms. Int J Mach Learn Comput 10(2) **(to appear)**
51. Ibrahim M (2019) Reducing correlation of random forest-based learning-to-rank algorithms using subsample size. Comput Intell 35(2):1–25
52. Ibrahim M, Carman M (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, pp 91–99
53. He B, Macdonald C, Ounis I (2008) Retrieval sensitivity under training using different measures. In: Proceedings of the 31st annual international ACM SIGIR conference on research and development in information retrieval. ACM, pp 67–74
54. Robertson S (2008) On the optimisation of evaluation metrics. In: Keynote, SIGIR 2008 workshop learning to rank for information retrieval (LR4IR)
55. Donmez P, Svore KM, Burges CJ (2009) On the local optimality of lambdarank. In: Proceedings of the 32nd international ACM SIGIR conference on research and development in information retrieval. ACM, pp 460–467
56. Yilmaz E, Robertson S (2010) On the choice of effectiveness measures for learning to rank. Inf Retr 13(3):271–290
57. Hanbury A, Lupu M (2013) Toward a model of domain-specific search. In: Proceedings of the 10th conference on open research

areas in information retrieval, Le Centre De Hautes Etudes Internationales D'Informatique Documentaire, pp 33–36
58. Hawking D (2004) Challenges in enterprise search. In: Proceedings of the 15th Australasian database conference, vol 27. Australian Computer Society, Inc., pp 15–24
59. McCallum A, Nigam K, Rennie J, Seymore K (1999) A machine learning approach to building domain-specific search engines. In: IJCAI, vol 99. Citeseer, pp 662–667
60. Owens L, Brown M, Poore K, Nicolson N (2008) The forrester wave: enterprise search, q2 2008. For information and knowledge management professionals
61. Yan X, Lau RY, Song D, Li X, Ma J (2011) Toward a semantic granularity model for domain-specific information retrieval. ACM TOIS 29(3):15
62. Szummer M, Yilmaz E (2011) Semi-supervised learning to rank with preference regularization. In: Proceedings of the 20th ACM international conference on information and knowledge management (CIKM). ACM, pp 269–278
63. Tyree S, Weinberger KQ, Agrawal K, Paykin J (2011) Parallel boosted regression trees for web search ranking. In: Proceedings of the 20th international conference on world wide web. ACM, pp 387–396
64. Freund Y, Schapire RE (1995) A decision-theoretic generalization of on-line learning and an application to boosting. In: Computational learning theory. Springer, pp 23–37
65. Friedman JH (2001) Greedy function approximation: a gradient boosting machine. Ann Stat 29(5):1189–1232 (English summary)
66. Quoc C, Le V (2007) Learning to rank with nonsmooth cost functions. Proc Adv Neural Inf Process Syst 19:193–200
67. Ganjisaffar Y, Caruana R, Lopes CV (2011) 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, pp 85–94
68. Ganjisaffar Y, Debeauvais T, Javanmardi S, Caruana R, Lopes CV (2011) Distributed tuning of machine learning algorithms using mapreduce clusters. In: Proceedings of the third workshop on large scale data mining: theory and applications. ACM, p 2