Learning to rank with extremely randomized trees

Pierre Geurts P.GEURTS@ULG.AC.BE
Gilles Louppe G.LOUPPE@ULG.AC.BE
Department of Electrical Engineering and Computer Science & GIGA-R
University of Liège
Institut Montefiore, Sart Tilman B28, B-4000 Liège, Belgium

Editor: Olivier Chapelle, Yi Chang, Tie-Yan Liu

Abstract

In this paper, we report on our experiments on the Yahoo! Labs Learning to Rank challenge organized in the context of the 23rd International Conference of Machine Learning (ICML 2010). We competed in both the learning to rank and the transfer learning tracks of the challenge with several tree-based ensemble methods, including Tree Bagging (Breiman, 1996), Random Forests (Breiman, 2001), and Extremely Randomized Trees (Geurts et al., 2006). Our methods ranked 10th in the first track and 4th in the second track. Although not at the very top of the ranking, our results show that ensembles of randomized trees are quite competitive for the "learning to rank" problem. The paper also analyzes computing times of our algorithms and presents some post-challenge experiments with transfer learning methods.

Keywords: Learning to rank, regression trees, ensemble methods, transfer learning

1. Introduction

In the context of machine learning, "learning to rank" (LTR) is concerned with the automatic induction of ranking models from training data. From an information retrieval perspective, one is given a learning sample of query-item pairs described by a set of input features and by a numerical score measuring the relevance of the item as a response to the query (or its rank among other items that can be associated to the query). The goal is then to learn, from this data, a function of the input features that ranks as well as possible (potentially unseen) items for a given (potentially unseen) query. LTR has been a very active area of research within machine learning in the last decade, mainly driven by its applications in the context of web search engines and recommender systems.

This paper reports on our experiments on the Yahoo! Labs LTR challenge organized in the context of the 23rd International Conference of Machine Learning (ICML 2010) in Haifa, Israël (Chapelle and Chang, 2011). The goal of the challenge was to have researchers and industrial scientists compete with their algorithms on a real-world large-scale dataset. The challenge was divided into two tracks: a standard learning to rank track and a transfer learning track. The goal of the first track was to obtain the best possible performance on a large-scale dataset of query-URL pairs. For the second track, an additional smaller dataset, compiling query-URL pairs from a different country than the first dataset, was provided

Table 1: Description of the challenge datasets

	set1			set2		
	Train	Val	Test	Train	Val	Test
Nb queries	19,944	2,994	6,983	1,266	1,266	3,798
Nb URLs	473,134	71,083	165,660	34,815	34,881	103,174

and the goal was to leverage the first dataset to improve the performance on the second dataset.

Our approach to this challenge is primarily based on the use of ensembles of randomized regression trees, in particular Extremely Randomized Trees (Geurts et al., 2006). Although we experimented with several output encodings, we basically addressed this problem as a standard regression problem trying to predict the relevance score as a function of the input features. On the first track of the challenge, our submission was ranked at the 10th position. Although this result was obtained with a heterogeneous (and not very elegant) ensemble of 16000 randomized trees produced by combining Bagging, Random Forests and Extremely Randomized Trees, our post-challenge experiments show that using Extremely Randomized Trees alone with default parameter settings and only 1000 trees yields exactly the same rank. On the second track, our submission was ranked at the 4th position. We have experimented with various simple transfer learning methods but unfortunately none of these really outperformed a model learned only from the second dataset. Interestingly, post-challenge experiments show however that some of the transfer approaches that we tested would have actually improved performances with respect to a model learned only from the second dataset if the size of this dataset would have been smaller.

The rest of this paper is organized as follows. Section 2 describes the two datasets and the rules of the challenge. In Section 3, we briefly review standard regression trees and ensemble methods and then present several simple adaptations of these methods for the learning to rank problem and for transfer learning. Our experiments on both tracks are then described in Section 4, which ends with an analysis of the computational requirements of our algorithms. We conclude and discuss future work directions in Section 5.

2. Learning to rank challenge: data and protocol

Contestants of the LTR challenge were provided with two distinct datasets collected at Yahoo! from real-world web search ranking data of two different countries. Both datasets (respectively labeled as `set1` and `set2`) consist of query-URL pairs represented as feature vectors. 415 features are common to the two sets, 104 are specific to `set1` and 181 are specific to `set2`. All of them are numerical features normalized between 0 and 1. For each query-URL pair, a numerical relevance score also indicates how well the URL matches the corresponding query, 0 being irrelevant and 4 being perfectly relevant. The queries, URLs and features semantics have not been revealed.

Both datasets were randomly split into 3 subsets, training, validation and test (see Table 1 for their respective sizes), and relevance scores were given for the training subsets only. In the first track of the challenge, contestants were asked to train a ranking function on

the training subset of `set1` and to predict a ranking of the URLs for each query on the validation and test subsets. In the second track, contestants were asked to do the same for `set2`, except that they were also allowed to leverage `set1`. In both tracks, predictions for the validation and test sets were evaluated and ranked using the Expected Reciprocal Rank (ERR) metric (Chapelle et al., 2009), which measures the expected reciprocal length of time that the user will take to find a relevant document (the higher, the better). After each submission, contestants were given as immediate feedback their score on the validation set but their score on the test set was kept secret in order to establish the final ranking at the end of the challenge. For informational purposes, contestants were also given their score using the Normalized Discounted Cumulative Gain (NDCG) metric (Järvelin and Kekäläinen, 2002). See (Chapelle and Chang, 2011) for more details about these metrics and the challenge data.

3. Method

After a brief description of tree-based ensemble methods in Section 3.1, we discuss in Section 3.2 various output encodings for solving the learning to rank problem and then in Section 3.3 several simple transfer learning strategies.

3.1. Tree-based ensemble methods

In this section, we briefly describe standard supervised learning of classification and regression trees, ensemble methods based on them, and the ensembles of Extremely Randomized Trees that are at the heart of our solution.

Standard classification and regression trees. A classification/regression tree (Breiman et al., 1984) represents an input-output model by a tree whose interior nodes are each labeled with a (typically binary) test based on one input feature. For numerical features, typical tests compare the value of the feature to a discretization threshold. Each terminal node of the tree is labeled with a value of the output (discrete or continuous) and the predicted output for a new instance is determined as the output associated to the leaf reached by the instance when it is propagated through the tree starting at the root node.

A tree is built from a learning sample of input-output pairs by recursively identifying at each node the test that leads to a split of the node sample into two subsamples that are as pure as possible in terms of their output values. In the case of a numerical output (regression trees), the impurity of the output in a subsample is directly measured by the (empirical) output variance. The algorithm then stops developing a branch as soon as some stopping criterion is met. A standard criterion is to avoid splitting a node as soon as either the output is constant in the node subsample or the number of learning sample instances that reach this node is lower than a given threshold n_{min}.

Ensemble methods. Single classification and regression trees typically suffer from high variance, which makes them not competitive in terms of accuracy. One very efficient way to circumvent this problem is to use them in the context of ensemble methods. Mainly two families of ensemble methods have been studied in the literature: Randomization-based methods and boosting methods. Although boosting methods can be very effective and have

been successful in the context of ranking (Li et al., 2007; Wu et al., 2008), we will focus in this paper on randomization methods.

Randomization methods produce different models from a single initial learning sample by introducing random perturbations into the learning procedure. For example, Bagging (Breiman, 1996) builds each tree of the ensemble from a random bootstrap copy of the original data. Another popular tree-based randomization method is the Random Forests algorithm proposed by Breiman (2001), which combines the bootstrap sampling idea of Bagging with a further randomization of the input features that are used as candidates to split an interior node of the tree. Instead of looking for the best split among all features, the algorithm first selects, at each node, a subset of K features at random and then determines the best test over these features to effectively split the node. This method is very effective and has found many successful applications in various fields.

Extremely Randomized Trees. Most experiments in this paper will be carried out with a particular randomization method called Extremely Randomized Trees (ET) proposed by Geurts et al. (2006). This method is similar to the Random Forests algorithm in the sense that it is based on selecting at each node a random subset of K features to decide on the split. Unlike in the Random Forests method, each tree is built from the complete learning sample (no bootstrap copying) and, most importantly, for each of the features (randomly selected at each interior node) a discretization threshold (cut-point) is selected at random to define a split, instead of choosing the best cut-point based on the local sample (as in Tree Bagging or in the Random Forests method). As a consequence, when K is fixed to one, the resulting tree structure is actually selected independently of the output labels of the training set. In practice, the algorithm only depends on a single main parameter, K. Good default values of K have been found empirically to be $K = \sqrt{p}$ for classification problems and $K = p$ for regression problems, where p is the number of input features (Geurts et al., 2006).

Experiments in (Geurts et al., 2006) show that this method is most of the time competitive with Random Forests in terms of accuracy, and sometimes superior. Because it removes the need for the optimization of the discretization thresholds, it has also a clear advantage in terms of computing times and ease of implementation.

We refer the reader to (Geurts et al., 2006) for a more formal description of the algorithm and a detailed discussion of its main features.

Summary. The main strength of tree-based randomization methods is the fact that they are (almost) parameter-free while still able to learn non-linear models, hence making them good all-purpose and off-the-shelf methods (Hastie et al., 2001). Their computing times are also very competitive and, through feature importance measures, they can provide some insight about the problem at hand (Hastie et al., 2001). On the downside, their accuracy is sometimes not at the level of the state-of-the-art on some specific classes of problems and their storage requirement might be prohibitive in some applications.

3.2. Output encodings

In most of our experiments, we have adopted a simple pointwise regression approach for solving the ranking problem with ensembles of randomized trees. Each query-URL pair

is treated as an independent training instance described by its input features and labeled with the relevance score of the URL for that query. Ensembles of regression trees are fitted on the training sample to learn a predictive model for the relevance score. The URLs corresponding to a query from the validation or test sample are then ranked according to the relevance score predicted by the regression model.

In addition to the previous regression setting, we also compared several alternative output encodings:

- **Regr-proba**: the same pointwise regression approach but recoding the relevance scores $\{0, 1, 2, 3, 4\}$ into the probabilities of relevance $\{0, 1/16, 3/16, 7/16, 15/16\}$ that are at the basis of the definition of the ERR criterion.

- **Regr-rank**: the same pointwise regression approach but instead of computing the ensemble prediction by averaging the relevance scores predicted by each tree, we computed for each URL its rank among the other URLs as predicted by each tree of the ensemble and then ranked the URLs according to their average rank over all ensemble trees.

- **Clas-avg**: the pointwise classification approach suggested in (Li et al., 2007). The relevance score is considered as a discrete output (with five values) and an ensemble of classification trees is built to predict it. For a given query-URL pair, the classification model outputs class-conditional probability estimates $\hat{P}(y = i|x)$ that are then turned into a relevance score by computing the expected relevance $\sum_{i=0}^{4} i\hat{P}(i|x)$.

- **Clas-moc**: a variant of the previous approach addressing the problem as a multiple ordinal classification problem (Li et al., 2007). Four binary classifiers are trained to learn the probabilities $P(y \leq i|x)$ for $i \in \{0, 1, 2, 3\}$. A score prediction is then obtained from the resulting estimates $\hat{P}(y \leq i|x)$ by computing

$$\sum_{i=1}^{4} i(\hat{P}(y \leq k|x) - \hat{P}(y \leq k-1|x))$$

  (with $\hat{P}(y \leq 4|x) = 1$).

3.3. Transfer learning

In this section, we enumerate several simple strategies for transfer learning that we have exploited on the second track of the challenge. In the description that follows, we will denote by $F1$ (resp. $F2$) the set of features defined only on `set1` (resp. `set2`), by FC the set of common features, and by $M(X, F)$ a model learned from data X with features F. The methods that we have compared are the following:

1. $M(\texttt{set2}, FC + F2)$: A model learned from `set2` as a base case.

2. $M(\texttt{set1}, FC)$: A model learned from `set1` with the common features only.

3. $M(\alpha\texttt{set1} + (1 - \alpha)\texttt{set2}, FC + F1 + F2)$: A model learned from a weighted union of the two datasets with all features. The introduction of the parameter α assumes

that the learning algorithm can deal with weighted instances. It adjusts the relative weight put on the instances of the two sets. In our experiments, α will be optimized in $[0, 1]$ on the validation set.

4. $\alpha M(\texttt{set1}, FC)+(1-\alpha)M(\texttt{set2}, FC+F2)$: a convex combination of the predictions of a model learned from `set1` and a model learned from `set2`. The parameter $\alpha \in [0, 1]$ will be optimized on the validation set.

5. $M(\texttt{set2}, FC+F2+M(\texttt{set1}, FC))$: A model learned from `set2` using as an additional input feature the relevance score predicted by the model learned from `set1`

6. $M(\texttt{set2} - M(\texttt{set1}, FC), FC + F2)$: A model learned from queries in `set2` where the outputs are the residuals with respect to the model learned from `set1` only. Predictions are then obtained by summing the predictions of this model with the predictions of the model $M(\texttt{set1}, FC)$.

Several of these variants have only been added after the challenge ended. Methods 1 and 2 are the two base cases that do not exploit respectively `set1` and `set2` data. Method 3 is straightforward. We included methods 4 and 5 because they were proposed by two other contestants and were found to improve (slightly) over the use of the `set2` data only. Method 6 was inspired by least-square residual fitting (Hastie et al., 2001). Similar approaches were proposed in (Chen et al., 2008) and (Gao et al., 2009).

4. Experiments

Below, we successively report on our experiments on the two tracks of the challenge. The section ends with a discussion of the computing times and storage requirements of our solutions. While the validation of several methods compared below was actually done by cross-validation on the training sample during the challenge period, we have rerun all methods on the complete training sample and evaluated their performance both on the validation and test samples. Our results are thus directly comparable to other contestants' results.

In our discussion, a method is declared significantly better than another if a paired t-test comparing the two methods in terms of ERR returns a p-value lower than 0.05. For readibility reason, standard errors are not reported systematically in our tables but typical values on both tracks and sets are given in Table 2. They have shown to be very stable from one method to another.

4.1. Track 1: learning to rank

Our strategy on this track was first to compare several randomization methods and then to increase as much as possible the number of trees to reach convergence. In a second step, we also compared the alternative output encoding strategies described in Section 3.2.

Comparison of tree-based methods. Table 3 compares Random Forests (RF) and Extremely Randomized Trees (ET) with 500 trees and the two default settings for their main common parameter K. For comparison, we also report in this table the results obtained with a model that would rank all URLs randomly.

Table 2: Typical standard errors for ERR and NCDG on both tracks

`set1` validation		`set1` test	
ERR	NDCG	ERR	NDCG
~ 0.006	~ 0.004	~ 0.004	~ 0.0025

`set2` validation		`set2` test	
ERR	NDCG	ERR	NDCG
~ 0.008	~ 0.005	~ 0.005	~ 0.003

Table 3: Comparison of different tree-based ensemble methods (with 500 trees. In bold: best values in each column)

	`set1` validation		`set1` test	
Method	ERR	NDCG	ERR	NDCG
Random	0.2779	0.5809	0.2830	0.5783
ET, $K = \sqrt{p}$	0.4533	0.7841	0.4595	0.7896
ET, $K = p$	0.**4564**	**0.7907**	0.4620	**0.7948**
RF, $K = \sqrt{p}$	0.4552	0.7884	0.4611	0.7923
RF, $K = p$ (Bagging)	0.4553	0.7866	**0.4629**	0.7934

The two methods are not very sensitive to the value of K, although in both cases, the best result is obtained with $K = p$. Both methods are very close in terms of performance with a slight advantage to ET $K = p$ on the validation sample and a slight advantage to RF $K = p$ (which is equivalent to Bagging in this case) on the test sample. These differences are however not statistically significant.

Effect of the ensemble size. Table 4 shows the effect of the number of trees with ET and $K = p$ on ERR and NDCG on the validation and test samples. The last column gives the rank that each entry would have obtained at the challenge. As expected, both the ERR and the NDCG monotonically increase as the size of the ensemble increases. ERR however saturates at 1000 trees. Although slight, the improvement from 500 to 1000 trees is statistically significant (p-value=0.0009) and it makes the method climbs from the 17th rank to the 10th rank.

Output encoding. The five different output encoding methods are compared in Table 5, in each case with ET, $K = p$ and 500 trees. Regr-proba is the best approach on the validation set but it is less good than Regr on the test set. On the test set, only Clas-moc is better than the straightforward regression encoding and the difference is significant (p-value=0.03). However, because it builds four binary classifiers, this variant uses four times more trees than the other methods and as shown in Table 4, the regression approach reaches the same test ERR with 1000 trees or more. Unlike what was found in Li et al. (2007), the two classification encodings are thus not interesting in our context.

Table 4: Effect of the number of trees (with ET, $K = p$)

	`set1` validation		`set1` test		
Nb trees	ERR	NDCG	ERR	NDCG	Rank
100	0.4548	0.7875	0.4609	0.7909	> 20
500	0.4564	0.7907	0.4620	0.7948	17
1000	0.4570	0.7906	0.4632	0.7957	10
5000	0.4571	0.7917	0.4632	0.7957	10
10000	0.4568	0.7919	0.4633	0.7961	10

Table 5: Comparison of different output encodings (with ET, $K = p$, 500 trees. In bold: best values in each column)

	`set1` validation		`set1` test	
Method	ERR	NDCG	ERR	NDCG
Regr	0.4564	0.7907	0.4620	0.7948
Regr-proba	**0.4570**	0.7877	0.4614	0.7898
Regr-rank	0.4551	0.7874	0.4601	0.7895
Clas-avg	0.4547	0.7862	0.4605	0.7900
Clas-moc	0.4565	**0.7912**	**0.4632**	**0.7961**

Final submission. In a last attempt to get further improvement, our final submission to track 1 was an aggregation of several of our intermediate submissions (by simply averaging relevance scores). The resulting ensemble contains in total 16000 trees: 1000 bagged trees, 10000 ET with $K = p$, and 5000 ET with $K = \sqrt{p}$. The ensemble is compared in Table 6 with an ensemble of 1000 ET with $K = p$ and with the best performer's scores. The heterogeneous ensemble yields an improvement both on the validation and test samples with respect to the pure ET ensemble. The improvement is significant on the test sample (p-value=0.007) but not on the validation sample (p-value=0.19). Both methods reaching the exact same rank at the challenge, the ET ensemble could be preferred because of its simplicity and lower size. Our final ERR score on the test set is about 1% smaller than the ERR score of the best performer of the challenge. Yet, this difference turns out to be very statistically significant (see Chapelle and Chang, 2011).

Table 6: Final submission to track 1

	`set1` validation		`set1` test		
Method	ERR	NDCG	ERR	NDCG	Rank
ET, $K = p$, 1000 trees	0.4570	0.7906	0.4632	0.7957	10
Mix of 16000 trees	0.4579	0.7923	0.4642	0.7967	10
Best performer	0.4611	0.7995	0.4686	0.8041	1

Table 7: Comparison of various approaches for transfer learning (with ET, $K = p$, 1000 trees. In bold: best values in each column)

		set2 validation		set2 test	
Method		ERR	NDCG	ERR	NDCG
Random		0.2567	0.5299	0.2597	0.5251
1	$M(\texttt{set2}, FC + F2)$	**0.4514**	**0.7859**	0.4611	0.7810
2	$M(\texttt{set1}, FC)$	0.4353	0.7441	0.4472	0.7409
3	$M(\alpha\texttt{set1} + (1-\alpha)\texttt{set2}, FC + F1 + F2)$	0.4514	0.7859	0.4611	0.7810
4	$\alpha M(\texttt{set1}, FC) + (1-\alpha)M(\texttt{set2}, FC + F2)$	0.4514	0.7859	0.4611	0.7810
5	$M(\texttt{set2}, FC + F2 + M(\texttt{set1}, FC))$	0.4497	0.7853	**0.4625**	**0.7830**
6	$M(\texttt{set2} - M(\texttt{set1}, FC), FC + F2)$	0.4487	0.7811	0.4594	0.7784

4.2. Track 2: transfer learning

For our experiments on the second track, we focused on the pointwise regression approach with the ET algorithm and investigated several ways to combine the two datasets.

Comparison of transfer learning approaches. All methods discussed in Section 3.3 are compared in Table 7, in all cases with ET, $K = p$, and 1000 trees. For methods 3 and 4, the value of α was optimized on the validation set. For Method 3, α was selected in $\{0.0, 0.05, 0.125, 0.25, 0.5, 1.0\}$. For Method 4 that does not require to relearn a model for each new α value, we screened all values of α in $[0, 1]$ with a stepsize of 0.01.

On the validation set, no method is able to improve with respect to the use of the `set2` data. The model learned from `set1` is clearly inferior but it is nevertheless quite good compared to the random model. For both methods 3 and 4, the optimal value of α is actually 0, meaning that these two models are equivalent to the model learned from `set2`.[1] Method 5, which introduces the prediction of the model learned on `set1` among the features, shows the best performances overall on the test set. However, the improvement over the base model is only slight and actually not statistically significant.

Experiments with less `set2` data. One potential reason for these disappointing results could be that, given the size of the `set2` learning sample, the limit of what can be obtained with our model family on this problem is already reached. As a consequence, `set1` data is not bringing any useful additional information. To check whether the transfer methods could still bring some improvement in the presence of less `set2` data, we repeated the same experiment but reducing this time by 95% the size of the `set2` learning sample (i.e., by randomly sampling 5% of the queries). Average results over five random samplings are reported in Table 8, with all transfer methods. With only 5% of the `set2` data (about 65 queries), the model learned on `set2` data is now comparable with the model learned on `set1` data. Transfer methods 3, 4, and 5 improve with respect to both the model learned from

1. Note that the value of α that optimizes ERR on the test set is 0.22. It corresponds to an ERR of 0.4622, which is slightly better than ERR of the model learned on `set2` alone.

Table 8: Transfer learning with only 5% of the `set2` data (with ET, $K = p$, 1000 trees, average over 5 random samplings).

		`set2` validation		`set2` test	
Method		ERR	NDCG	ERR	NDCG
1	$M(\texttt{set2/5\%}, FC + F2)$	0.4357	0.7434	0.4472	0.7407
2	$M(\texttt{set1}, FC)$	0.4353	0.7441	0.4472	0.7409
3	$M(\alpha\texttt{set1} + (1-\alpha)\texttt{set2/5\%}, FC + F1 + F2)$	**0.4419**	0.7552	0.4522	0.7519
4	$\alpha M(\texttt{set1}, FC) + (1-\alpha) M(\texttt{set2/5\%}, FC + F2)$	0.4418	**0.7580**	**0.4527**	**0.7529**
5	$M(\texttt{set2/5\%}, FC + F2 + M(\texttt{set1}, FC))$	0.4361	0.7450	0.4486	0.7433
6	$M(\texttt{set2/5\%} - M(\texttt{set1}, FC), FC + F2)$	0.4356	0.7453	0.4466	0.7435

Table 9: Final submission to track 2

	`set2` validation		`set2` test		
Method	ERR	NDCG	ERR	NDCG	Rank
ET, 10000 trees, $K = p$	**0.4513**	0.7867	0.4617	0.7819	4
Best performer	0.4506	**0.7875**	**0.4635**	**0.7863**	1

`set2/5%` and the model learned from `set1`. The improvement is significant[2] for Method 3 (p-value=0.0173) and Method 4 (p-value=0.0014), but not for Method 5. For methods 3 and 4, the (average) optimal α values are respectively 0.5 and 0.6, which now gives more weight to the `set1` data. Although the improvement is not impressive, this experiment thus suggests that transfer methods could be effective in the presence of less `set2` data.

Final submission. Given that we were unsuccesful at exploiting `set1` data, our final submission to the challenge is an ET ensemble learned on `set2` data only (with $K = p$). Since the more trees the better, we increased the number of trees to 10000. Their performances on the validation and test samples are reported in Table 9, as well as the result of the best perfomer of this track. Our submission ranked at the fourth position of the challenge. On this track, the differences between the top four contestants were however not very significant (see Chapelle and Chang, 2011).

This result was surprising for two reasons. First, we were only ranked at the sixth position on the validation set. This means that other contestants have probably overfitted the validation set, which is not surprising given that we observed discrepancy between these two sets in the results in Table 7. As a further clue of this discrepancy, note also that we are actually performing better on the validation set than the overall track winner (see Table 9). Second, our final solution does not exploit any data from `set1`, at all. As a matter of fact, it turns out that other contestants were not much more successful than us at transfering information from `set1` to `set2` (Chapelle and Chang, 2011).

2. Using a paired t-test to compare the ERR values found by both methods over the 5 random samplings.

Table 10: Computing times and storage requirements of different methods

Method	Learning[1]	Testing[2]	Nb nodes[3]	Storage[4]	ERR `set1` test[5]
ET, $K = p$	296s	492ms	280K	1.3MB	0.4620
RF, $K = p$	618s	469ms	140K	663kB	0.4629
ET, $K = \sqrt{p}$	20s	543ms	400K	1.8MB	0.4595
RF, $K = \sqrt{p}$	28s	507ms	205K	959kB	0.4552
ET, $K = p$,$n_{min} = 50$	268s	336ms	47K	215kB	0.4626
ET, $K = p$,$n_{min} = 100$	254s	260ms	25K	115kB	0.4613

[1]Computing times in seconds for learning one tree on `set1` learning sample

[2]Computing times in milliseconds for testing one tree on the validation and test samples of `set1`

[3]Total number of nodes in one tree built on `set1` learning sample

[4]Storage requirement for one tree

[5]ERR obtained with 500 trees on the `set1` test sample

4.3. Computational considerations

To conclude our experiments, we report in this section on the computational requirements of our algorithms. Table 10 shows training and testing times, as well as storage requirements of different tree-based methods all computed on `set1`.[3] In addition to the methods discussed above, we also analysed in this table the effect of the stop splitting parameter n_{min} that reduces the complexity of the trees. These statistics have been obtained with an unoptimized code intended only for research purpose[4] and should thus be taken with some caution.

ET are twice as fast as RF for $K = p$ and 30% faster for $K = \sqrt{p}$. Methods differ greatly in terms of the size of the models (from 25K nodes to 400K nodes) but prediction times, which is mainly related to the tree depth, only vary by 50% from the slowest to the fastest method. Given that all methods are very close in terms of accuracy, retrospectively, ET with $K = p$ and $n_{min} = 100$ seems to be the best method as it produces much smaller model with no apparent degradation in terms of ERR score. If one can sacrifice some accuracy to reduce computing times, then ET with $K = \sqrt{p}$ is also an interesting tradeoff. 500 models are built in less than 3 hours (on a single core) and its ERR score is only 2% lower than the ERR score of the best performing method of the challenge. Note that of course, better tradeoffs for a given application could be achieved by adopting other parameter settings (e.g., less trees in the ensemble or other values of K or n_{min}). Given that trees in the ensemble are also built independently of each other, the parallelization of the algorithm is trivial.

3. The software has been implemented in C with a MATLAB (The MathWorks, Inc.) interface. Computing times excludes loading times and assumes that all data are stored in main memory. They have been computed on a Mac Pro Quad-core Intel Xeon Nehalem 2,66 GHz with 16Go of memory. Storage requirements have been derived from binary mat files generated with MATLAB assuming that all numbers (i.e., tested attributes, node indices and discretization thresholds) are stored in single precision.
4. Software can be downloaded from http://www.montefiore.ulg.ac.be/~geurts/Software.html.

5. Discussion

In this paper, we have reported on our experiments in the context of the 2010 Yahoo! Labs LTR challenge. Our solution based on ensembles of randomized regression trees has reached rank 10 on the first track and rank 4 on the second track. These results suggest that tree-based randomization methods can be competitive on ranking problems. The default parameter settings and output encoding turned out to be optimal in terms of accuracy, which highlights the fact that these methods are essentially parameter-free. Computing times of these methods remain also competitive on these very large datasets, both for training the model and for making predictions.

It is interesting to analyse our solution in the light of the other participants' solutions. Remarkably, it actually turns out that all top-four contestants on both tracks have been using regression tree-based methods. Unlike us, however, they all exploited some form of (gradient) boosting, most of the time combined with randomization. The superiority of these methods over our pure randomization-based approaches is clear from the fact that we were ranked lower at the challenge. We believe however that one advantage of pure randomization methods is that they require less parameter tuning than boosting methods. Actually, we have experimented on the challenge data with gradient boosting, combined with several kinds of randomization, but we were not able to improve with respect to randomization methods. Chapelle and Chang (2011) also experimented on `set1` data with gradient boosting and obtained an ERR score on the test sample of 0.46201, which is lower than our best score in Table 6 (but the difference is probably not significant). This suggests that other participant's best performances are probably due either to better parameter tuning, specific combination of randomization and boosting approaches, or clever pre- or post-processing (e.g, feature selection). A comparison of all these methods from the point of view of computing times would also be interesting. Boosting methods have the advantage of typically requiring smaller trees but their parallelization, although it has been studied (Ye et al., 2009), is not as trivial as that of randomization methods. Training times should also account for parameter tuning.

In terms of future works, we would like to pursue experiments on the transfer learning approaches. In particular, it would be interesting to evaluate our simple strategies on other tasks and also to investigate more sophisticated approaches on the challenge datasets (Pan and Yang, 2010). This would help us understand why we were not successful at exploiting `set1` dataset when learning from the complete `set2` dataset.

Acknowledgments

The authors would like to thank Yahoo! and the workshop organizers for setting up the challenge and the prize, Raphaël Marée for useful discussions, Louis Wehenkel for reading the manuscript, and Olivier Chapelle and the anonymous reviewers for useful suggestions. PG and GL are respectively research associate and research fellow of the FNRS, Belgium. This work is partially supported by the Interuniversity Attraction Poles Programme (IAP P6/25 BIOMAGNET), initiated by the Belgian State, Science Policy Office and by the European Network of Excellence, PASCAL2. Experiments were all carried out on the NIC3 supercomputer of the University of Liège.

References

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

L. Brei-man, J. H. Friedman, R. A. Olsen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth International (California), 1984.

O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. *JMLR Workshop and Conference Proceedings*, 14:1–24, 2011.

O. Chapelle, D. Metlzer, Y. Zhang, and P. Grinspan. Expected reciprocal rank for graded relevance. In *Proceeding of the 18th ACM conference on Information and knowledge management*, pages 621–630. ACM, 2009.

K. Chen, R. Lu, C. K. Wong, G. Sun, L. Heck, and B. Tseng. Trada: tree based ranking function adaptation. In *Proceeding of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 1143–1152, New York, NY, USA, 2008. ACM.

J. Gao, Q. Wu, C. Burges, K. Svore, Y. Su, N. Khan, S. Shah, and H. Zhou. Model adaptation via model interpolation and boosting for web search ranking. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2 - Volume 2*, EMNLP '09, pages 505–513, Morristown, NJ, USA, 2009. Association for Computational Linguistics.

P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 36 (1):3–42, 2006.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):446, 2002.

P. Li, C. Burges, and Q. Wu. Learning to rank using classification and gradient boosting. *Proceedings of the International Conference on Advances in Neural Information Processing Systems (NIPS)*, 2007.

S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, October 2010. ISSN 1041-4347.

Q. Wu, C.J.C. Burges, K.M. Svore, and J. Gao. Ranking, boosting, and model adaptation. *Tecnical Report, MSR-TR-2008-109*, 2008.

J. Ye, J.-H. Chow, J. Chen, and Z. Zheng. Stochastic gradient boosted distributed decision trees. In *Proceeding of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 2061–2064, New York, NY, USA, 2009. ACM.