Improving Scalability and Performance of Random Forest Based Learning-to-Rank Algorithms by Aggressive Subsampling

Muhammad Ibrahim **Mark Carman**

Faculty of Information Technology
Monash University, Australia,
Wellington Road, Clayton VIC 3800.
Email: muhammad.ibrahim@monash.edu, mark.carman@monash.edu

Abstract

Random forest based Learning-to-rank (LtR) algorithms exhibit competitive performance to other state-of-the-art algorithms. Traditionally, each tree of the forest is learnt from a bootstrapped copy (sampled with replacement) of the training set, where approximately 63% examples are unique, although some studies show that sampling without replacement also works well. The goal of using a bootstrapped copy instead of the original training set is to reduce correlation among individual trees, thereby making the prediction of the ensemble more accurate. In this study, we investigate whether we can decrease the correlation of the trees even more without compromising accuracy. Among several potential options, we work with the sub-sample used for learning individual trees. We investigate the performance of a random forest based LtR algorithm as we reduce the size of the sub-samples used for learning individual trees. Experiments on Letor data sets reveal that substantial reduction of training time can be achieved using only small amount of data training data. Not only that, the accuracy is likely to increase while maintaining the same level of performance stability as the baseline. Thus in addition to the existing benefit of being completely parallelizable, this study empirically discovers yet another ingredient of random forest based LtR algorithms for making them one of the top contenders for large scale LtR.

Keywords: Random forest, Learning-to-rank, Scalability, Sub-sampling, Correlation, Bootstrapping.

1 Introduction

When a user submits a query, the task of an information retrieval system is to return a list of documents ordered by the predicted relevance to that query. Traditionally different scoring methods, ranging from simple heuristic models (eg. tf-idf score) to probabilistic models (eg. BM25, Language Models), have been used for this task[1]. Recently researchers have investigated supervised machine learning techniques for solving this problem. In this setting a training example is a query-document pair, the corresponding is the 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 (eg. 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 (Li 2011), (Liu 2011).

Since it has been proven that the ranking error is bounded by both the classification error (Li et al. 2007) and regression error (Cossock & Zhang 2006), people often address the LtR problem using classification or regression framework. In these settings a classification or regression algorithm learns to predict the relevance label (which can also be thought as relevance score) of an individual query-document pair, and then during evaluation time, the documents associated to a query are ranked in decreasing order of these scores. This approach is called *pointwise* in the literature (Li 2011) because it treats the instances (i.e. feature vectors corresponding to 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 and resource requirements as well as simplicity.

1.1 Random Forest

A random forest (Breiman 2001) is a simple, effective, non-parametric learning algorithm which aggregates the outputs of a large number of independent and variant base learners (decision trees in the default setting). Its major benefits over other state-of-the-art methods include inherent parallelizability, ease of tuning and competitive performance. These benefits attract researchers of various disciplines such as Bioinformatics (Qi 2012), Computer Vision (Criminisi & Shotton 2013), and Data Analysis (Ham et al. 2005), where random forest is a very popular choice. For the LtR task, however, its use has not been studied thoroughly. A few LtR algorithms use random forest in a pointwise manner with regression settings such as (Geurts & Louppe 2011) and (Mohan 2010). The authors show that these algorithms, in spite of their relative simplicity as compared to other state-of-the-art techniques, show competitive performance.

1.2 Bootstrapping in Random Forest

Bootstrapping is a well known effective technique in statistical learning, and is an inherent component of random forest. The method builds a *bootstrapped* sample by randomly choosing an instance *with* replacement from the original training set of size the same as the original sample. The benefits of bootstrapping include: (i) sound mathematical properties (Hastie et al. 2009, Ch. 8), (ii) for ensemble learning, proper use of it reduces variance and possibly also bias (Friedman & Hall 2007), (iii) for ensemble learning, during training phase, the out-of-bag data can be used to estimate test error (Breiman 2001).

In the existing random forest based pointwise algorithms, simple bootstrapping is used where sampling is performed without any concept of query-document structure of the data. That is, sampling is performed not on

[1] (Manning et al. 2008) is a useful work to know about these models.

a per query basis, but rather on a par example (query-document pair) basis. We believe that this may not be the most appropriate approach because it results in fewer documents (and thus less information) per query in the sample. So bootstrapping in our methods ensures that once a query is chosen for inclusion in a training sample (also known as *bag*), all of its associated documents (examples) are chosen, i.e., we sample the training data on a per query-basis.

The usual practice among the researchers is to perform bootstrapping with replacement (Breiman 2001), but (Friedman 2002), (Friedman & Hall 2007) show that bootstrapping without replacement also works for supervised learning, and moreover, (Ganjisaffar et al. 2011) show that it works well for the LtR task. Bootstrapping without replacement has an additional potential benefit from the perspective of scalability because it reduces the learning time.

Suppose Q is the set of queries pertaining to a training set $\mathcal{T}$, and a query q has n_q number of associated documents where its ith document is d_i^q. The bootstrapping (on a per query basis) procedure is given in Algorithm 1.

Data: Training set $\mathcal{T}$ containing $|Q|$ queries (and associated documents).
Result: Bootstrap/Sub-Sample $\mathcal{T}_{sample}$
$\mathcal{T}_{sample} \leftarrow \emptyset$;
while *size*($\mathcal{T}_{sample}$) $<$ *threshold* **do**
  $q \leftarrow chooseRandomQuery(Q)$;
  $\mathcal{T}_{sample}.add(\{d_i^q\}_{i=1}^{n_q})$;
  if *sampleWithoutReplacement* **then**
    $Q = Q \setminus q$;
  end
end

Algorithm 1: Bootstrap Sampling / Sub-sampling on a Per Query Basis. The *threshold* is the required size of the sample.

2 Motivation

In this paper, our goal is to increase scalability while maintaining or improving the level of accuracy.

A random forest has two components which affect its error rate (Breiman 2001): (1) the correlation between the trees: the lower, the better, and (2) the strength of individual trees: the higher, the better. Usually there is a trade-off between the two.

If each tree learns from the entire training set, then there will be high correlation between them, and so their outputs will tend to be comparatively similar, thereby reducing the advantage of aggregation of those outputs. The goal of bootstrapping in random forest is to decrease correlation (i.e., increase variability) between individual trees so that the ensemble is less sensitive to peculiarities of (or perturbations in) the training set. This is achieved by learning individual trees using perturbed training data. At the same time, each tree should be strong enough to have good prediction on unseen data, so intuitively, using a very small training sample for each tree is not likely to work. This raises the question: is there any specific benefit of using 63% of the (distinct) training data examples (as is the case for bootstrapping) in the sample? That is, can we decrease correlation by sub-sampling less than 63% of the data? Put differently, what is the optimal sub-sample size in the context of LtR? To the best of our knowledge, no study has tried to address this question before. If we find that we can use much smaller training set than traditional bootstrapping while maintaining the same level of accuracy, then it will help in reducing training time which is an important factor in large scale LtR as mentioned in (Li 2011, Ch. 7), (Liu 2011, Ch. 20), (Chapelle et al. 2011). While doing so, if we find better performance (because of making individual trees even less correlated), that would add to this benefit. This motivates us to work here.

Another fact increases our motivation which is as follows. Random forests are known to perform well compared to other algorithms with small datasets ((Qi 2012), for example.). For example, some disciplines such as Bioinformtics must work with comparatively smaller samples (Kosorok et al. 2007). Still very good performance is achieved for these domains using random forest based algorithms (Qi 2012). LtR training sets are generally larger than some other disciplines and some other tasks[2]. Hence we investigate the use of smaller sub-samples – smaller than bootstrapped sample – for learning each tree.

The main contribution of this paper is to demonstrate that much smaller sub-samples (for some datasets, as little as 1.6% of the original training set) not only works as well, but indeed increases the accuracy while maintaining similar level of stability of the model in terms of variance across multiple runs.

The rest of the paper is organized as follows. Section 3 discusses some related works. Sections 4 and 5 describe our approach. Section 6 analyzes the experimental results. Section 7 summarizes some important insights from this study. Section 8 concludes the paper.

3 Related Work

Our work is mainly related to two domains. They are described below.

On reducing the correlation among trees. Some options to reduce correlation have been suggested in some works such as the following. (Robnik-Šikonja 2004) use five different gain functions instead of the Gini index[3] to decrease correlation among the trees for classification and regression tasks. With each of the five functions, their method learns one-fifth of the trees of an ensemble, and yields minor improvement. (Geurts et al. 2006) inject more randomness in a standard random forest in selection of the best *split-point*[4]. They consider only one randomly chosen value for an attribute, but the entire training set is used to learn each tree thereby mitigating the possible benefit of scalability. Later (Geurts & Louppe 2011) apply this algorithm to LtR problem.

Another supervised learning framework which can use bootstrapping is gradient boosting (Friedman 2001). (Ganjisaffar et al. 2011) show that bootstrapping without replacement works well in their gradient boosting based LtR algorithm. As for other supervised learning tasks such as classification and regression, (Friedman 2002) shows in the context of regression that the individual trees of a gradient boosted ensemble can learn from a bootstrapped sample (without replacement) instead of the usual practice of using the original sample. (Friedman & Hall 2007) demonstrate both theoretically and empirically that sampling around 50% of the original training set is approximately equivalent to bagging (in terms of bias/variance considerations) for regression problem.

On reducing the training set size. Another line of research which is somewhat related to the topic of our paper is on preparing a comparatively smaller, labelled training sample from a larger unlabelled corpus. (Aslam et al. 2009) investigate different methods for sampling from a large corpus. That is, they studied techniques for generating a good training set from a large document collection. Some other works such as (Macdonald et al. 2012),

[2]For example, as compared to most of the data sets in well-known UCI Repository for Classification (`https://archive.ics.uci.edu/ml/datasets.html`).

[3]Gini index is typically used in a random forest, please see (Breiman 2001) for details.

[4]A split-point is a pair of an attribute and a value for that attribute.

(Long et al. 2010), etc. are concerned with the size of the training set. Some other works such as (Donmez & Carbonell 2008), (Yu 2005) etc. also try to find the examples which, if added to a training set, increase the quality of the learned ranking function. All these works try to improve the quality of the training set while limiting its size so that only *informative* examples are included. Although this line of work is not directly related to our topic, findings from our experiments are applicable *after* these methods are applied.

Thus to the best of our knowledge, there is no study on the optimal size of sub-sample to be used for leaning each tree of a random forest based LtR algorithm. This study tries to address this question.

4 Approach

In our approach, each tree of an ensemble is learnt from a training set which contains q randomly chosen queries (and associated documents) from the original training set. We start with $q = 1$, then gradually increase it (as explained in Experiments section) until the sub-sample contains all the queries of the original training set (i.e., with no bootstrapping at all). In order to get reliable values of evaluation metrics, we run 10 independent runs of each experiment, and report the average value of the metrics.

Although random forest is known to perform well with smaller training sets, if q is too small, then there may be a concern about the stability of the model. Hence for the experiments we, besides measuring average performance, analyze the variance of the metrics across 10 runs. We note however that the variance (instability) is not a major concern for random forest because the larger the number of trees an ensemble has, the lower the variance will be (Breiman 2001). In spite of this, in some cases the lack of sufficient computational resources may limit maximum allowable size of an ensemble, making it important to also analyze the variance.

Another approach would be to make use of all queries for building each tree, but to sample the documents per query, starting initially with a small number of documents, and then gradually increasing the number of documents chosen. In this setting, each tree in the ensemble would have the same queries (all the queries) but different subsets of documents – the size of the subset is the same across all trees. We think, however, that there is a subtle issue in this setting which is as follows. As explained earlier, in order to achieve good accuracy, the trees of a random forest, given they are "strong" enough, are supposed to be as uncorrelated as possible with one another. Now (Yilmaz & Robertson 2009) show that for the LtR task, given a large pool of queries and documents, if we are to select a training set from it, then sampling more queries with less documents is better than sampling less queries with more documents – better for a rank-learner to learn a ranking function. This means that in the former case, i.e., sampling more queries and less documents, the characteristics of the original training set (i.e., the collection of documents mentioned above) are largely retained in the sampled set. Now back to our setting: for every tree, if we sample all queries and a subset of documents per query, then this is not likely to reduce correlation (which is one of our objectives) because the sampled training set for individual trees, according to the above-mentioned work, is likely to contain *enough information* to be representative of the original training set. Hence we avoid this setting.

As our work may be seen as an experiment to tune a parameter where the parameter is the size of sub-sample to be used for learning a tree, we find the best parameter using a validation set (that is, not used for training), and then we apply our model with the learnt parameter to a separate test set to validate our findings.

5 Model

The LtR algorithm we use can be thought of as a hybrid of the classification and regression settings. For splitting a node, the classification settings (entropy-based gain) are used. For assigning a label to a test instance, regression settings are used, i.e., when a test instance lands on a leaf of a tree, then instead of predicting the majority class label, the algorithm assigns a real score to the test instance which is the average of the labels of all the documents of that leaf. Finally, in order to calculate an IR metric for evaluation, the documents are sorted in decreasing order using these scores. Thus the training and the testing phases adopt a classification and a regression setting respectively. The procedure for building a tree is given in Algorithm 2. In our experiments, each configuration uses a sample of fixed size b (the number of queries), for learning each tree, and varying b constitutes different configurations (cf. the 1st line of Algorithm 2). We omit the pseudo-code of some routines when the meaning is obvious (*chooseRandomFeature(.), Split(.) and entropy(.)*).

Data: Entire (non-bagged) data $\mathcal{T}$ having N features, number of queries to sample b
Result: A tree of the ensemble
$\mathcal{T}_{subsampled} \leftarrow getSubsampledData(\mathcal{T}, b)$; //(cf. Alg. 1)
$BuildNode(\mathcal{T}_{subsampled})$;

Algorithm 2: RandomTree

Data: (Sub-sampled) Data $\mathcal{T}$
if $|\mathcal{T}| < gainThreshold$ **then**
  Store the class distribution of $\mathcal{T}$ in this leaf;
  return;
else
  $K \leftarrow 0$;
  while $K < \sqrt{(N)}$ **do**
    $f \leftarrow chooseRandomFeature(N)$;
    $\langle gain, bestSplitVal \rangle \leftarrow$
    $getMaxGain(\mathcal{T}, f)$;
    if $gain > bestGain$ **then**
      $gain \leftarrow bestGain$;
      $bestSplitPoint \leftarrow \langle f, bestSplitVal \rangle$;
    end
    $K = K + 1$;
  end
  if $bestGain > gainThreshold$ **then**
    $\langle \mathcal{T}_{left}, \mathcal{T}_{right} \rangle \leftarrow$
    $Split(\mathcal{T}, bestSplitPoint)$;
    $BuildNode(\mathcal{T}_{left})$;
    $BuildNode(\mathcal{T}_{right})$;
  else
    Store the class distribution of $\mathcal{T}$ in this leaf;
    return;
  end
end

Algorithm 3: BuildNode

6 Experiments

We now discuss the experiments we have performed to understand the effect of sub-sample size on performance of random forest based LtR algorithms.

Data: Data $\mathcal{T}$, feature f
Result: Best split value for this feature
for $splitVal \in possibleSplits(\mathcal{T}, f)$ **do**
  $\mathcal{T}_{Left}, \mathcal{T}_{Right} \leftarrow Split(\mathcal{T}, splitVal)$;
  $gain \leftarrow entropyGain(\mathcal{T}, \mathcal{T}_{Left}, \mathcal{T}_{Right})$;
  if $gain > bestGain$ **then**
    $bestSplitVal \leftarrow splitVal$;
    $bestGain \leftarrow gain$;
  end
end
return $\langle bestGain, bestSplitVal \rangle$;

Algorithm 4: MaxGain

Data: parentData $\mathcal{T}$, leftChildData $\mathcal{T}_L$, rightChildData $\mathcal{T}_R$
Result: Gain for this split.
return $entropy(\mathcal{T}) - (\frac{|\mathcal{T}_L|}{|\mathcal{T}|}.entropy(\mathcal{T}_L) + \frac{|\mathcal{T}_R|}{|\mathcal{T}|}.entropy(\mathcal{T}_R))$

Algorithm 5: EntropyGain

6.1 Setup

We start with $q = 1$, then gradually increase q such that for a data set we have nearly 25 experiments, ending with the experiment with the original training set (i.e., with no bootstrapping at all).

The *Weka* machine learning toolkit[5] is modified and extended to run the experiments.

Table 1 shows statistics of some publicly available datasets from Letor repository[6] which have been heavily used for the LtR task. In the Letor repository, each dataset is divided into 5 predefined folds for fairness in comparison between different algorithms, and each fold is further divided into a training, a validation and a test set having 3/5ths, 1/5th and 1/5th of the entire data respectively. We run the experiments using these 5 folds, and report the average results. Two widely used rank-based metrics are measured, namely, NDCG@10 and MAP. There are 1000 trees in each ensemble. The terminating conditions to stop further building a tree to check whether there is no gain in terms of entropy (cf. Algorithm 5) for any potential split of that node. We do not impose any restriction on maximum height of a tree or minimum required instances of a node.

6.2 Result Analysis

Figures 1 and 2 show the performance on validation sets in terms of NDCG@10 and MAP (respectively) of the 10 independent runs of each experiment. In both figures the standard deviation is also shown in the right. In the left-hand side plots of these figures, each point is the NDCG@10 (MAP) of a single run (out of 10) of an experiment where a tree of the ensemble at hand is learnt from a $p\%$ of random queries of the original training set – p is indicated by the x-axis. The procedure for computing p is as follows: we start with $b = 1$ for all the three datasets[7], then we increase it by 3, 20 and 50 for Ohsumed, MQ2008, and MQ2007 respectively until $b = |Q|$ (i.e. unless all the queries are used), thereby making the total number of distinct configurations for Ohsumed, MQ2008 and MQ2007 as 22, 25 and 22 respectively. There are 10 points for each p value which show the NDCG@10 (MAP) value for 10 independent runs. For example, for the experiment where each tree of the ensemble uses 60% random queries as its training set, the 10 NDCG@10 (MAP) values are indicated along the vertical line corresponding to p = 60%. A smooth-spline is fitted (with parameter 0.01) in order to capture the trend of a plot (as shown by the black curve). The bar plots in the right-hand side indicate the standard deviations of the NDCG@10 (MAPs) across the 10 runs of each experiment. The values of x-axes of the right-hand side plots indicate the same things as that of the left-hand side plots.

The plots show that the performance of the initial configuration is low (except for Ohsumed) which is anticipated because using only one query is not likely to yield good accuracy. Then performance increases, but after using a certain percentage of the queries, it starts to decrease, and this trend of reduction continues until the end. The reduction of accuracy for the configurations beyond the point $p = 63\%$ is already known to the research community as it causes the trees to be more correlated[8], but our experiments suggest that using standard bootstrapping is not the best choice for the LtR problem.

For Ohsumed dataset, using only one query is one of the best configurations in terms of performance. We conjecture that the reason is that Ohsumed has fewer queries and more documents per query than MQ2008 and MQ2007 (cf. Table 1), so one query is sometimes enough to learn a good ranking function, and because of the large number of base learners used in an ensemble, the overall accuracy is high. Another conjecture is, there are only 63 ($\approx 106.(3/5)$) queries per fold in the training set of Ohsumed dataset which is comparatively smaller than the ensemble size (i.e. 1000). So each query is used for learning approximately 16 ($\approx 1000/63$) trees, whereas this number stands to only 2 ($\approx 1000/((784).(3/5))$) and 1 ($\approx 1000/((1692).(3/5))$) for MQ2008 and MQ2007 respectively. Hence we conjecture that as we grow the ensemble size, similar trend (of witnessing better performance using fewer queries per tree) can be found in MQ2008 and MQ2007 as well. We leave this to future work.

As for variance analysis, we focus more on the MAP since it is a comparatively more stable metric than NDCG@10, but we do not omit the NDCG@10 altogether from the discussion. Accordingly, for MQ2008, the variances of the models (represented by standard deviations) after the first configuration seem to be random in the sense that there is no clear trend in the bar plots after the 1st configuration. For MQ2007, there seems to be a consistent trend of small decrease as the sub-sample size increases – this may be due to the fact that the MQ2007 has comparatively large number of queries, so as the number of queries increases, so does the stability of the models. As for Ohsumed, for both MAP and NDCG@10, no clear trend is seen, and the differences seem to be due to relatively small number of independent experiments (10 only). We conjecture that a reason for insignificant variation of variances across different models[9] is that the large number of base learners used in an ensemble mitigates the possible instability in output.

Figure 3 shows that the training time steadily increases with the increasing percentage of the queries.

Now we need to select the *best* setting (i.e., the number/percentage of queries) from these experiments, and then to apply this setting to a separate test set in order to corroborate the findings. To select the best setting, we consider only absolute performance since the variances of different sub-sample sizes don't vary greatly. For Ohsumed, MQ2008 and MQ2007, the best configurations

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

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

[7] Recall that b is the number of queries to sample.

[8] In fact, the motivation of injecting randomization in the trees came from this insight (Breiman 2001).

[9] Please note that the y-axis in the right-hand side bar plots are quite enlarged, so the differences of variances are actually not very big.

Table 1: Statistics of the Three Datasets Used in Our Experiments.

Data Set	Task	# Queries	# Features	# Relevance Labels	# Query-doc Pairs	Avg. # Docs per Query	# Relevant Docs per Query
Ohsumed	Medical Docs	106	45	3	16000	152	46
MQ2008	Web Search	784	46	3	15000	19	5
MQ2007	Web Search	1692	46	3	69000	41	12

are: 1st (with 1 queries (1.6%)), 2nd (with 21 queries (9%)), and 2nd and 3rd (with 51 and 101 queries (5% and 10%)) respectively – for MQ2007 since only this dataset has a slight trend of decreasing variance as sub-sample size increases, we select two best settings here. Table 2 summarizes the improvements on validation sets of the selected best settings. Across different datasets, only 1.6-10% of the queries are needed to get even better accuracy with similar model variance, and the learning time is reduced by 10-29 times.

Now using the best settings found with the validation sets, we run the experiments on separate test sets. Suggested by Table 2, the number of queries we chose for Ohsumed, MQ2008 and MQ2007 are 1, 41 and 51/101 respectively. Table 3 shows the improvement of accuracy and reduction of learning time of these settings over the baselines (i.e., over 63% sub-sampling).

From the experiments on the test sets, we see that in addition to the increased scalability, on average the accuracy of the models run with our selected best settings is higher than the baselines. This improvement of accuracy is not likely to be a random gain because we have conducted 10 distinct runs of each experiment, and the plots of the Figures 1 and 2 clearly show the average trend of improvement of our configurations over the baselines. We note that in information retrieval, improvement of accuracy by as small amount as 0.01-0.02 (1-2%) in the commonly used metrics such as NDCG and MAP is considered to be considerably better performance (Chapelle et al. 2007), (Xu & Croft 2000).

So far we have not compared performance of random forest based pointwise LtR algorithm with other algorithms. Table 4 shows the comparison between other algorithms whose performance are listed in the Letor website; in addition we run experiments for another state-of-the-art algorithm called LambdaMart[10] (Wu et al. 2010) because it is also a tree ensemble method using the gradient boosting framework[11]. The top performances of a metric of a dataset (i.e. of a row) is shown in **bold face**. The results of the two random forest based algorithms (RF-bt stands for 63% sub-sampling and RF-sb stands for reduced sub-sampling) are copied from Table 3. We see that although very simple as compared to most of the baseline algorithms, in most of the cases the performance of RF-sb (which is the contribution of this paper) is very close to the best value. Let us not forget one inherent benefit of a random forest based algorithm over all other ones listed in Table 4: it is embarrassingly parallelizable as each tree can be leant in parallel. Table 4 also unveils one important aspect: no single algorithm performs consistently better than others across all data sets. This shows that LtR task is per se a complex task.

7 Discussion

We now summarize the observations from the experiments discussed above.

- Our experiments have empirically discovered that for the datasets we have used, the training time can be reduced by 10-29 times thereby increasing scalability of a random forest based pointwise LtR algorithm while achieving better prediction accuracy and similar level of stability (in terms of standard deviation of results across multiple runs). This is because the plots of Figures 1 and 2 show that for an experiment with the best settings, there is a high probability that the performance will be higher than the baseline.
- The findings depend on the datasets, but the optimal % of queries to be used in each tree definitely lies far below the 63% level as currently being used in the LtR literature and even the 50% which is reported by (Friedman & Hall 2007) for classification and regression tasks. is the optimal value to be used in each tree. Here lies the main contribution of our work. Further research can be done in order to find the aspects, if exist, of a training set which influence the optimum percentage. Some aspects to investigate are: number of queries, average number of documents per query, and average ratio of relevant to irrelevant documents per query.
- By reducing the sub-sample size per tree, we decrease the correlation among the trees which is supposed to cause the error rate of the ensemble to decrease. However, as mentioned in the original paper of random forest (Breiman 2001), there is another aspect which controls the accuracy of the ensemble – the strength of the individual trees. By reducing sub-sample size, the strength is likely to decrease. In spite of this, we still have achieved slightly better performance – this means that the positive effect of decreasing correlation *compensates* for any negative effect of reducing strength. Inspired by this conjecture, we are currently working on as to how to retain/increase strength of individual trees in spite of reducing sub-sample size. One idea is to increase the parameter K (randomly chosen features at each split). However, since this will increase the training time linearly, we shall probably loose the benefit of scalability which is currently present in our experiments. So there may be some optimal choice of sub-sample size and K which will both increase accuracy of ensemble and decrease the training time as compared to the baseline (where the baseline uses a bootstrapped sample and $K = \sqrt{\#features}$).
- The findings discovered in this work are in fact somewhat pessimistic. For example, for MQ2008 and MQ2007 datasets, we have showed that the 2nd configurations (having 21 and 51 queries respectively) have the best accuracy. So here the granularity of our experiments could be improved in order to identify the exact number of queries that results in the best performance. Work along this line is in progress.
- There may be a relation between the ensemble size, T and number of queries used to learn each tree, b. For example, if $b = 1$, then each query appears in approximately $\frac{T}{b}$ number of trees. We are working to elicit some sort of relationship between these two parameters.

[10] The source code has been taken from `https://code.google.com/p/jforests/`).

[11] The parameters of LambdaMart are set as follows: number of trees = 1000, number of leaves per tree = 7, learning rate = 0.5.

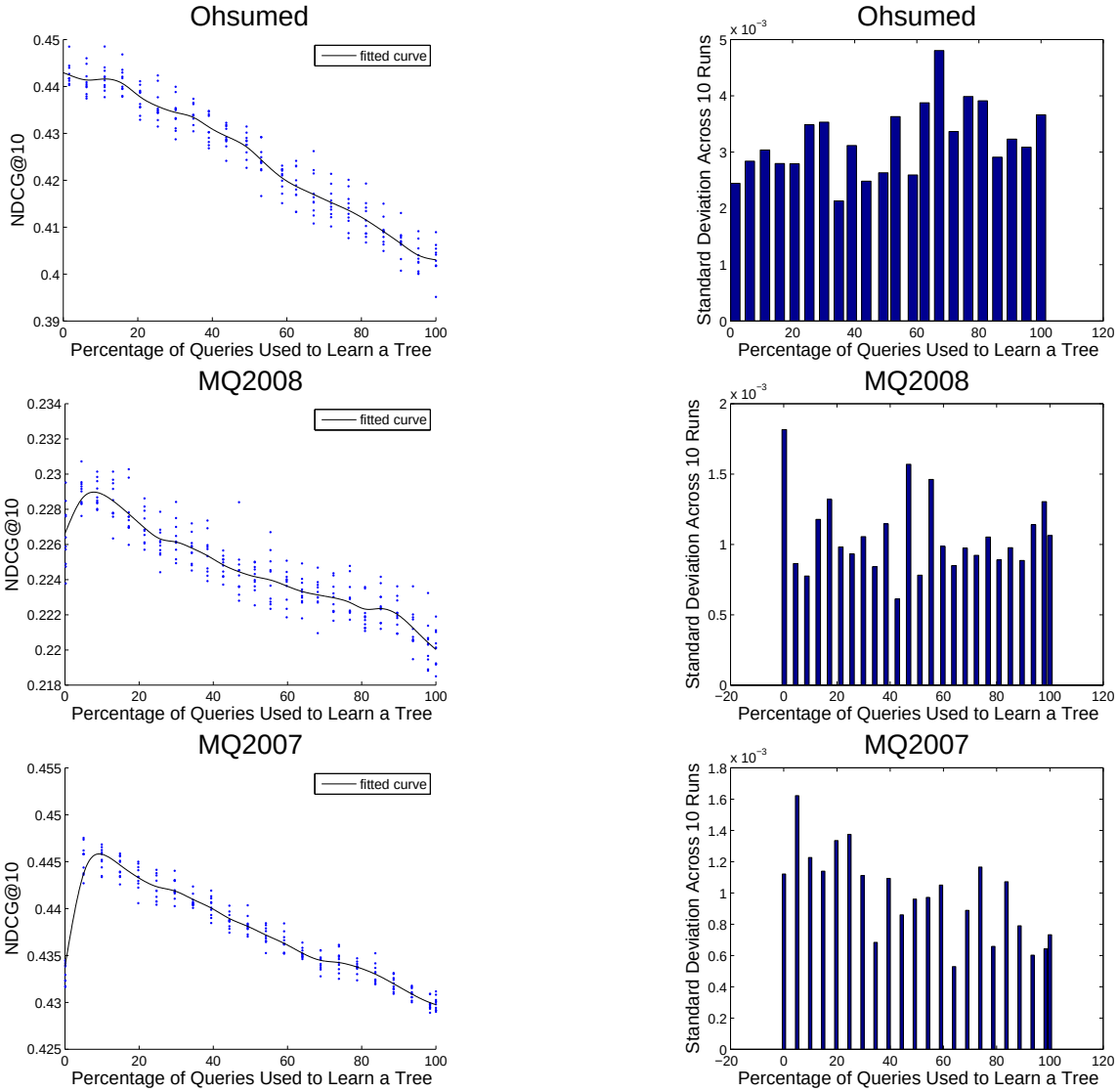

Figure 1: Performance (NDCG@10) and Standard Deviation of Various Datasets for 1000 trees in an ensemble (With Validation Sets): 1st, 2nd and 3rd rows are for Ohsumed, MQ2008 and MQ2007 respectively. In the left plots, each point is the NDCG@10 of a single run (out of 10) of an experiment (using the % of queries per tree indicated by the x-axis). There are 10 points for a single value in the x-axis implying the NDCG@10 for 10 independent runs. A smooth-spline is fitted (with parameter 0.01). The bar plots in the right-hand side indicate the standard deviations of the NDCG@10s across the 10 runs.

8 Conclusions and Future Work

Random forests has two components which control its accuracy, namely, (1) the correlation between the trees, and (2) the strength of the individual trees. In this work, we have investigated into the former of these two in the context of LtR task. We have used reduced sub-sample per tree to decrease correlation. Experiments show that we can achieve better performance while achieving substantial reduction in the training time – training time is an important factor for large scale LtR task. The stability of the models is also roughly of similar level of the baseline.

We are now working on several directions as mentioned in the Discussion section such as investigating other options for reducing correlation (as well as the training time) and on the better understanding the relationship between the ensemble size and number of queries.

References

Aslam, J. A., Kanoulas, E., Pavlu, V., Savev, S. & Yilmaz, E. (2009), Document selection methodologies for efficient and effective learning-to-rank, *in* 'Proc. of the 32nd international ACM SIGIR conf. on Research and development in information retrieval', ACM, pp. 468–475.

Breiman, L. (2001), 'Random forests', *Machine learning* **45**(1), 5–32.

Chapelle, O., Chang, Y. & Liu, T.-Y. (2011), Future directions in learning to rank, *in* 'JMLR Workshop and Conference Proceedings', Vol. 14, pp. 91–100.

Chapelle, O., Le, Q. & Smola, A. (2007), Large margin optimization of ranking measures, *in* 'NIPS Workshop: Machine Learning for Web Search'.

Cossock, D. & Zhang, T. (2006), 'Subset ranking using regression', *Learning theory* pp. 605–619.

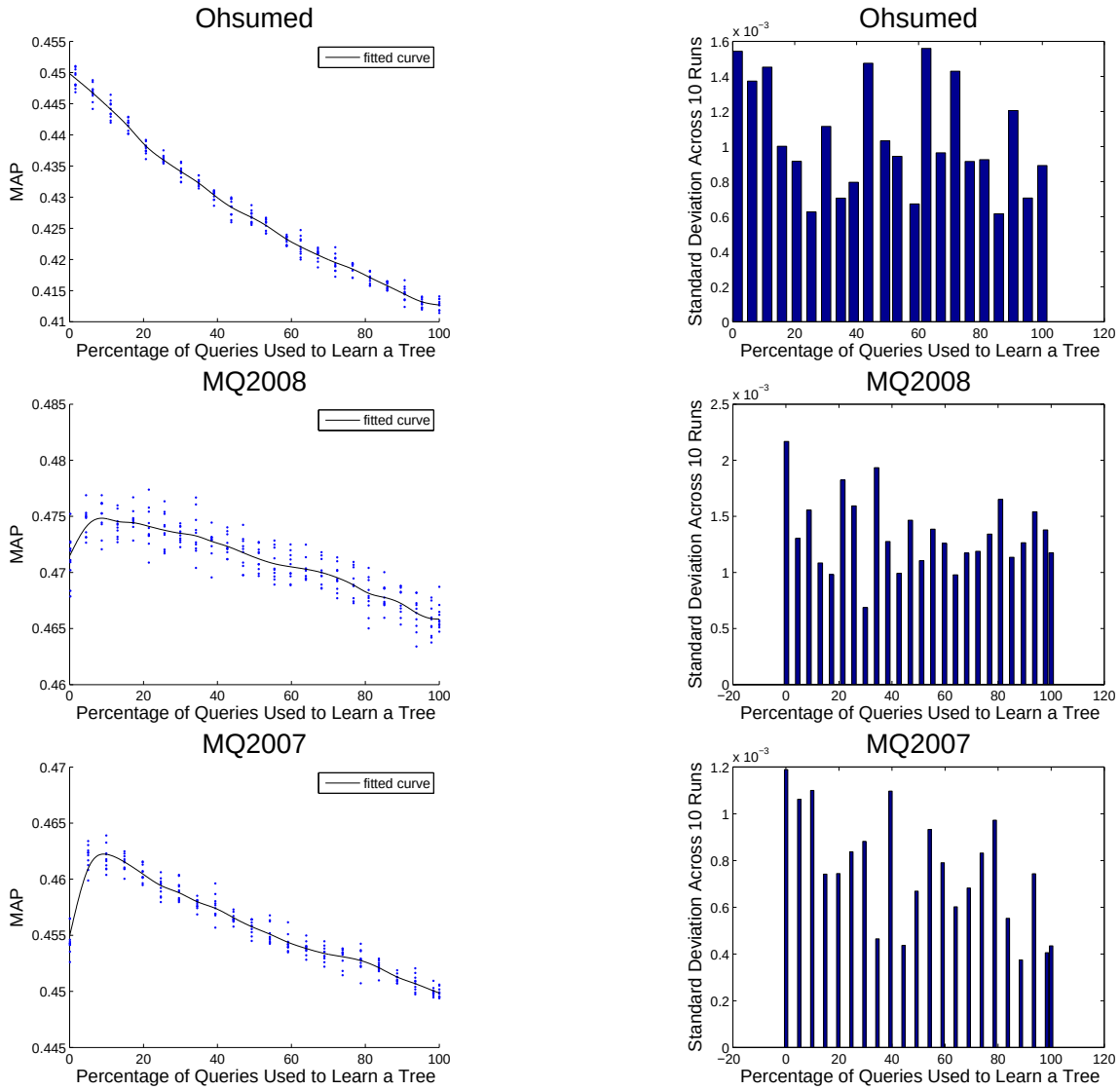

Figure 2: Performance (MAP) and Standard Deviation of Various Datasets for 1000 trees in an ensemble (With Validation Sets): Please See the Caption of Figure 1 for Description.

Criminisi, A. & Shotton, J. (2013), *Decision forests for computer vision and medical image analysis*, Springer.

Donmez, P. & Carbonell, J. G. (2008), Optimizing estimated loss reduction for active sampling in rank learning, *in* 'Proc. of 25th international conf. on Machine learning', ACM, pp. 248–255.

Friedman, J. H. (2001), 'Greedy function approximation: a gradient boosting machine.(english summary)', *Ann. Statist* **29**(5), 1189–1232.

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

Friedman, J. H. & Hall, P. (2007), 'On bagging and nonlinear estimation', *Journal of statistical planning and inference* **137**(3), 669–683.

Ganjisaffar, Y., Caruana, R. & Lopes, C. V. (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.

Geurts, P., Ernst, D. & Wehenkel, L. (2006), 'Extremely randomized trees', *Machine learning* **63**(1), 3–42.

Geurts, P. & Louppe, G. (2011), Learning to rank with extremely randomized trees, *in* 'JMLR: Workshop and Conference Proceedings', Vol. 14.

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

Hastie, T., Tibshirani, R. & Friedman, J. (2009), 'The elements of statistical learning'.

Kosorok, M. R., Ma, S. et al. (2007), 'Marginal asymptotics for the large p, small n paradigm: with applications to microarray data', *The Annals of Statistics* **35**(4), 1456–1486.

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

Li, P., Burges, C. & Wu, Q. (2007), 'Learning to rank using classification and gradient boosting', *Advances in neural information processing systems* **19**.

Liu, T.-Y. (2011), *Learning to rank for information retrieval*, Springerverlag Berlin Heidelberg.

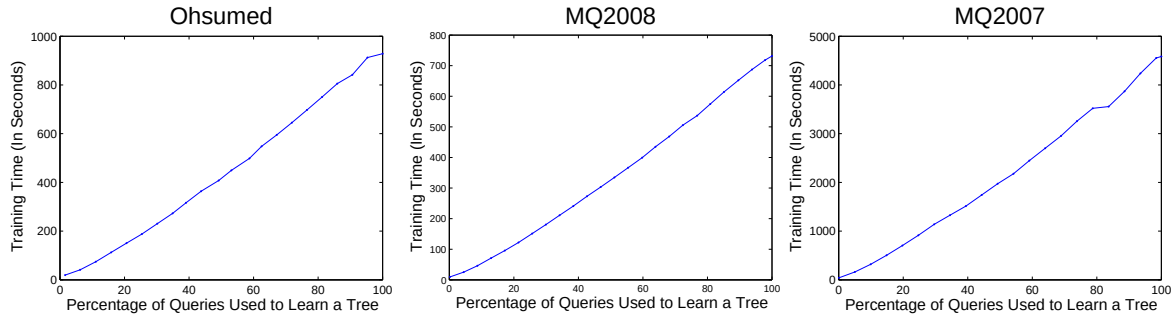

Figure 3: Training Time (to Learn an Ensemble of 1000 Trees) Vs % of Queries Used to Learn Each Tree (With Validation Sets): (a) Ohsumed (b) MQ2008 (c) MQ2007.

Table 2: Summary of Reduction in Training Time and Improvement of Performance for Validation Set.

Data Set	# (and %) of Queries Used in Reduced Subsample Per Tree	# (and %) of Queries Used in Subsample Per Tree in Baseline	Relative Training Time (Speed-up over Baseline)	NDCG@10 of Reduced Sample (Avg of 10 Runs and Std. Dev.)	NDCG@10 of Baseline (Avg of 10 Runs and Std. Dev.)	MAP of Reduced Sample (Avg of 10 Runs and Std. Dev.)	MAP of Baseline (Avg of 10 Runs and Std. Dev.)
Ohsumed	1 (1.6%)	40 (63%)	29 times	0.4427 (0.0024)	0.4187 (0.0039)	0.4490 (0.0015)	0.4221 (0.0016)
MQ2008	21 (4.5%)	301 (64%)	17 times	0.2290 (0.0009)	0.2233 (0.0085)	0.4745 (0.0013)	0.4703 (0.0098)
MQ2007	51 (5%)	651 (64%)	17 times	0.4453 (0.0016)	0.4353 (0.0005)	0.4619 (0.0011)	0.4539 (0.0006)
	101 (10%)	651 (64%)	9 times	0.4453 (0.0012)	0.4353 (0.0005)	0.4619 (0.0011)	0.4539 (0.0006)

Long, B., Chapelle, O., Zhang, Y., Chang, Y., Zheng, Z. & Tseng, B. (2010), Active learning for ranking through expected loss optimization, *in* 'Proceedings of the 33rd international ACM SIGIR conf. on Research and development in information retrieval', ACM, pp. 267–274.

Macdonald, C., Santos, R. L. & Ounis, I. (2012), 'The whens and hows of learning to rank for web search', *Information Retrieval* pp. 1–45.

Manning, C. D., Raghavan, P. & Schütze, H. (2008), *Introduction to information retrieval*, Vol. 1, Cambridge University Press Cambridge.

Mohan, A. (2010), 'An empirical analysis on point-wise machine learning techniques using regression trees for web-search ranking'.

Qi, Y. (2012), Random forest for bioinformatics, *in* 'Ensemble Machine Learning', Springer, pp. 307–323.

Robnik-Šikonja, M. (2004), Improving random forests, *in* 'Machine Learning: ECML 2004', Springer, pp. 359–370.

Wu, Q., Burges, C. J., Svore, K. M. & Gao, J. (2010), 'Adapting boosting for information retrieval measures', *Information Retrieval* **13**(3), 254–270.

Xu, J. & Croft, W. B. (2000), 'Improving the effectiveness of information retrieval with local context analysis', *ACM Transactions on Information Systems (TOIS)* **18**(1), 79–112.

Yilmaz, E. & Robertson, S. (2009), Deep versus shallow judgments in learning to rank, *in* 'Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval', ACM, pp. 662–663.

Yu, H. (2005), Svm selective sampling for ranking with application to data retrieval, *in* 'Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining', ACM, pp. 354–363.

Table 3: Summary of Reduction in Training Time and Improvement of Performance for **Test Set** (Using Best Settings from Validation Sets).

Data Set	**# (and %) of Queries Used in Reduced Subsample Per Tree**	**# (and %) of Queries Used in Subsample Per Tree in Baseline**	**Relative Training Time (Speed-up over Baseline)**	**NDCG@10 of Reduced Sample (Avg of 10 Runs and Std. Dev.)**	**NDCG@10 of Baseline (Avg of 10 Runs and Std. Dev.)**	**MAP of Reduced Sample (Avg of 10 Runs and Std. Dev.)**	**MAP of Baseline (Avg of 10 Runs and Std. Dev.)**
Ohsumed	1 (1.6%)	40 (63%)	29 times	0.4443 (0.0022)	0.4317 (0.0033)	0.4473 (0.0009)	0.4221 (0.0013)
MQ2008	21 (4.5%)	301 (64%)	17 times	0.2292 (0.0008)	0.2238 (0.0011)	0.4736 (0.0016)	0.4701 (0.0012)
MQ2007	51 (5%)	651 (64%)	17 times	0.4447 (0.0013)	0.4363 (0.0010)	0.4619 (0.0012)	0.4533 (0.0004)
	101 (10%)	651 (64%)	9 times	0.4453 (0.0016)	0.4363 (0.0010)	0.4624 (0.0004)	0.4533 (0.0004)

Table 4: Comparison Between Random Forest Based Pointwise Algorithms and Various Algorithms. The top performance is shown in **boldface**. The Algorithms are: ListNet (LN), AdaRank-MAP (AM), AdaRank-NDCG (AN), RankBoost (RB), RankSVM-Primal (RSP), RankSVM-Struct (RSS), LambdaMart (LM), RF-bootstrapped (RF-bt), and RF-reduced-subsampled (RF-sb). The last two algorithms' (RF-bt and RF-sb) results are copied from Table 3.

Data	**Metrics**	LN	AM	AN	RB	RSP	RSS	LM	RF-bt	RF-sb
Ohsumed	N@10	0.441	0.4429	0.4496	0.4302	0.4504	**0.4523**	0.4367	0.4317	0.4443
	MAP	0.4457	0.4487	**0.4498**	0.4411	0.4446	0.4478	0.4173	0.4221	0.4473
MQ2007	N@10	0.4440	0.4335	0.4369	0.4464	0.4436	0.4439	**0.4484**	0.4363	0.4453
	MAP	0.4652	0.4577	0.4602	0.4662	0.4659	0.4644	**0.4675**	0.4533	0.4624
MQ2008	N@10	0.2303	0.2288	**0.2307**	0.2255	0.2279	0.2309	0.2302	0.2238	0.2292
	MAP	0.4775	0.4764	**0.4824**	0.4775	0.4744	0.4696	0.4751	0.4701	0.4736