Concurrent Learning of Large-Scale Random Forests

Henrik Boström
Dept. of Computer and Systems Sciences
Stockholm University
Forum 100, 164 40, Kista, Sweden
henrik.bostrom@dsv.su.se
www.dsv.su.se/∼henke

Abstract. The random forest algorithm belongs to the class of ensemble learning methods that are *embarassingly parallel*, i.e., the learning task can be straightforwardly divided into subtasks that can be solved independently by concurrent processes. A parallel version of the random forest algorithm has been implemented in Erlang, a concurrent programming language originally developed for telecommunication applications. The implementation can be used for generating very large forests, or handling very large datasets, in a reasonable time frame. This allows for investigating potential gains in predictive performance from generating large-scale forests. An empirical investigation on 34 datasets from the UCI repository shows that forests of 1000 trees significantly outperform forests of 100 trees with respect to accuracy, area under ROC curve (AUC) and Brier score. However, increasing the forest sizes to 10 000 or 100 000 trees does not give any further significant performance gains.

Keywords. random forests, concurrent learning, Erlang

1. Introduction

The random forest algorithm [6] has become one of the most popular machine learning methods. Apart from its wide distribution through well-known platforms, such as WEKA [17] and R [25,19], the popularity can be explained by its relatively low computational cost and ability to often reach state-of-the-art predictive performance [9].

The random forest algorithm belongs to the class of ensemble learning methods that are *embarassingly parallel*, i.e., the learning task can be straightforwardly divided into subtasks that can be solved independently by concurrent processes, since each tree in the forest depends only on the given dataset and not on the other trees in the forest. This contrasts to other ensemble learning methods, such as boosting [15], where each member of the ensemble is dependent on previously generated members. The random forest algorithm hence lends itself to straightforward parallellization.

As multi-core computers are becoming mainstream and there is a clear trend in increasing the number of cores on each platform, machine learning implementations, as most other software, would clearly benefit from being made concurrent. Besides being able to solve the same tasks as earlier, but much faster, this also opens up for running experiments that with a standard sequential implementation would require prohibitively long time. For the random forest algorithm, parallellization means that much larger forests or datasets can be considered than what would otherwise be possible.

Experiments with random forests rarely consider forests that are larger than a few hundred trees, some exceptions being [12,27]. The main research question of this work is to what extent there are any gains from generating large-scale forests compared to generating more moderately sized forests.

To this aim, we have implemented a parallell version of the random forest algorithm, using Erlang[1], which is a concurrent language that was originally developed for telecommunication applications. The language has during recent years reached something of a hype and has been used for implementing everything from database applications at Amazon[2] to the chat system in Facebook.

In the next section, we present the random forest algorithm and discuss some implementation options, regarding both its parallellization and the underlying tree learning algorithm. In Sect. 3, we outline the experimental setup and present results from comparing forests of different sizes. Finally, conclusions and pointers to future work are given in Sect. 4.

2. Methods

We first provide a formulation of the random forest algorithm and discuss some options for the underlying tree learning algorithm. We then discuss the parallellization of the former.

2.1. Random forests

The random forest algorithm [6] generates a set of classification trees [7], while incorporating randomness both in the selection of training examples and in the selection of features to consider when generating each individual tree. The former is done by employing bootstrap aggregating, or bagging [5], which works by randomly selecting n examples with replacement from the initial set of n training examples. Furthermore, when generating each tree in the forest, only a small randomly selected subset of all available input features are considered at each node in the tree. This method is summarized in Algorithm 1, which in turn employs Algorithm 2 for learning each individual tree.

Algorithm 1 RandomForest(examples $\langle e_1, \ldots, e_n \rangle$, features F, no. of trees t)

```
1: for $i = 1 \ldots t$ do
2:     for $j = 1 \ldots n$ do
3:         $e'_j \leftarrow e_{rand(1,n)}$
4:     $T_i \leftarrow RandomTree(\langle e'_1, \ldots, e'_n \rangle, F)$
5: return $T_1, \ldots, T_t$
```

[1]http://www.erlang.org

[2]http://aws.amazon.com/simpledb

Algorithm 2 RandomTree(examples $\langle e_1, \ldots, e_n \rangle$, features F)

```
1:  if $TerminalNode(\langle e_1, \ldots, e_n \rangle)$ then
2:      $T \leftarrow MakeLeaf(\langle e_1, \ldots, e_n \rangle)$
3:  else
4:      $F' \leftarrow RandomSubset(F)$
5:      $\{c_1, \ldots, c_p\} \leftarrow BestSplit(F', \langle e_1, \ldots, e_n \rangle)$
6:      $\{E_1, \ldots, E_p\} \leftarrow Distribute(\langle e_1, \ldots, e_n \rangle, \{c_1, \ldots, c_p\})$
7:      if there is some $E_i \in \{E_1, \ldots, E_p\}$ such that $E_i = \langle e_1, \ldots, e_n \rangle$ then
8:          $T \leftarrow MakeLeaf(\langle e_1, \ldots, e_n \rangle)$
9:      else
10:         for $i = 1 \ldots p$ do
11:             $T_i \leftarrow RandomTree(E_i, F)$
12:         $T \leftarrow \{(c_1, T_1), \ldots, (c_p, T_p)\}$
13: return $T$
```

Both algorithms assume as input a set of features $F = \{f_1, \ldots, f_m\}$, where each feature f_i is the name of an independent (or input) variable. Furthermore, they require an ordered (possibly empty) set of (training) examples $\langle e_1, \ldots, e_n \rangle$, where each example e_i is a pair $\langle \bar{x}_i, y_i \rangle$ such that $\bar{x}_i = \langle x_{i1}, \ldots, x_{im} \rangle$, where each x_{ij} is a value for feature f_j, and where $y_i \in \{l_1, \ldots, l_k\}$, where $l_1, \ldots, l_k$ are the possible class labels.

In Algorithm 1, the function $rand(a, b)$ is employed, which returns a random integer in the interval $[a, b]$.

In Algorithm 2, the boolean function $TerminalNode(\langle e_1, \ldots, e_n \rangle)$ returns $TRUE$ if certain conditions are met. These may vary between implementations, but a common condition, which is also adopted here, is that all examples share the same class label. Note that this also covers the special case of having an empty example set. The function $MakeLeaf(\langle e_1, \ldots, e_n \rangle)$ returns a representation of a leaf in the generated tree. For a classification tree, this is simply the class label that occurs most frequently among the examples $\langle e_1, \ldots, e_n \rangle$. For a probability estimation tree (PET) [22], this is instead a class probability distribution P, such that $0 \leq P(l_i) \leq 1$ for all $l_i \in \{l_1, \ldots, l_k\}$, and $\sum_{i=1,\ldots,k} P(l_i) = 1$. Different ways of forming such distributions may be considered. In [3], it was shown that for random forests, relative frequencies are to be preferred over using Laplace correction and the m-estimate [10]. In this work, we consider forests of PETs where the probabilities are obtained from relative class frequencies.

The function $RandomSubset(F)$ returns a randomly selected subset of the given features. A few alternatives for this have been proposed [6], including functions returning a fixed number of features and a function returning a subset of size $\lfloor log_2|F| + 1 \rfloor$, the latter being adopted here.

The function $BestSplit(F', \langle e_1, \ldots, e_n \rangle)$ partitions the examples $\langle e_1, \ldots, e_n \rangle$ by using each feature in F' and returns the conditions $\{c_1, \ldots, c_p\}$ that result in the best partition according to some criterion. The conditions $\{c_1, \ldots, c_p\}$ that are used for forming a partition refer to a single feature f only, and they are required to be mutually exclusive and exhaustive, i.e., for each possible example e with a non-missing value on f, there is exactly one condition c_i that holds for e. In this study, we limit ourselves to numeric and nominal features and furthermore only consider conditions resulting in binary partitions, i.e., they are on the form $\{f_{num} \leq v, f_{num} > v\}$ for a numeric feature f_{num}

and some $v \in \mathbb{R}$, and $\{f_{nom} = v, f_{nom} \neq v\}$ for a nominal feature f_{nom} and some v in the domain of f_{nom}. Several methods for finding threshold values, i.e., v above, for the numeric conditions have been proposed, including the standard method proposed in [14]. However, in order to avoid the cost of sorting the numerical values, as required by the method in [14], and also to increase diversity of the trees, we here instead consider only a single threshold value that is obtained from the average of two examples that are randomly selected from two different classes. In a similar way, only one randomly selected value for each considered nominal feature is evaluated at each node in the tree.

There has been a plentitude of criteria suggested for evaluating the resulting partitions. Two well-known candidates for this are Gini impurity [7] and information gain [23]. The latter criterion is adopted in this study. It should be noted that since we are only considering binary partitions, its bias in favor of choosing partitions with many members can be ignored here.

The function $Distribute(\langle e_1, \ldots, e_n \rangle, \{c_1, \ldots, c_p\})$ results in a partition $\{E_1, \ldots, E_p\}$ of $\langle e_1, \ldots, e_n \rangle$ such that condition c_i holds for all examples in E_i. Particular care has to be taken to examples having missing values on the feature that the conditions are referring to. The standard way to handle this is by first partitioning all examples with known values on the feature, and then distributing fractions of the remaining examples according to the relative size of each group in the partition [24]. In order to increase diversity of the trees in the forest, we here instead randomly assign each example with a missing value to any of the subgroups, where the probability of a subgroup being selected is the same as its relative size.

It should be noted that the tree growing algorithm returns a leaf node whenever it fails to partition the examples over multiple subsets. If the partition contains more than one non-empty subset, each subset is used to recursively generate a subtree.

When making a prediction with a forest of PETs, a class probability distribution is formed by averaging the class probability distributions that are output by each individual tree in the forest.

2.2. Parallellizing the random forest algorithm

As the number of trees in a forest normally can be expected to be larger than the number of available cores, one straightforward approach of parallellizing the random forest algorithm is to assign to each sub-process the task of generating a subset of the forest. In Erlang, a sub-process is started by the spawn-function, leaving the decision of how to distribute the work of multiple processes over the available cores to the built-in task scheduler. The sub-processes communicate through message passing, where each process has a separate queue of messages to be handled. In principle, a separate process can be started for generating each individual tree, but for very large forests, this may result in that the message queue for the main process that keeps track of all sub-processes, grows out of bounds. Instead, we here adopt the approach of spawning as many sub-processes as there are available cores, requesting each sub-process to generate an approximately equal number of trees. In order to utilize all available cores also during classification, each sub-process retains the generated sub-forest in its local memory for making predictions, where the latter are collected and formed into collective votes by the main process. This has demonstrated to be significantly faster than having the main process first collecting all sub-forests and then applying the trees sequentially.

3. Empirical investigation

In this section, we first present the three evaluation criteria that are considered in the empirical investigation. We then present the experimental setup and formulate the test hypotheses. Finally, we present and analyze the outcome of the experiment.

3.1. Evaluation criteria

The most common way of evaluating predictive performance of classifiers is by measuring the accuracy (i.e., the percentage of correctly classified examples). During recent years, there has however been a growing interest in also the ranking performance, which can be evaluated by measuring the area under the ROC curve (AUC) [13]. The AUC can be interpreted as the probability of ranking a true positive example ahead of a false positive when ordering examples according to decreasing likelihood of being positive [4]. When using the output of classifiers to calculate the expected utility of different alternatives in decision situations, the correctness of predicted class probabilities may be of crucial importance, something which also has attracted increased attention during recent years [20,18]. One commonly employed measure of correctness of predicted class probabilities, which also is adopted in this study, is the squared error or *Brier score* [8].

It should be noted that the three evaluation criteria are not completely correlated, and a model that is less accurate than another, may very well result in a higher AUC or Brier score (and vice versa).

3.2. Experimental setup

We consider random forests, as described in Sect. 2, of four different sizes: 100 trees, 1000 trees, 10 000 trees, and 100 000 trees. The proposed forest sizes are expected to be sufficient for giving an indication on whether or not one can expect gains from generating very large forests.

The forests are compared w.r.t. accuracy, AUC and Brier score using stratified tenfold cross-validation on 34 data sets from the UCI Repository [2]. The names of the data sets are listed in Table 1. The average scores obtained for the ten folds are reported. For data sets with more than two classes, the weighted AUC is reported, i.e., the AUC for each class is weighted in proportion to the relative frequency of the class [13].

There are actually a number of hypotheses to be tested. The null hypotheses can be formulated as there is no difference in predictive performance, i.e., as measured by accuracy, AUC and Brier score, between forests with 100, 1000, 10 000 and 100 000 trees, respectively.

The algorithms were implemented in Erlang OTP R14B and executed on a Dell PowerEdge R815 with 48 cores (2.2 GHz AMD Opteron) and 64 GB primary memory, running Debian Linux. Time measurements for selected datasets are given at the end of the next section.

3.3. Experimental results

For each of the four considered forest sizes, i.e., 100, 1000, 10 000 and 100 000 trees, the accuracy, AUC and Brier score are listed for all 34 datasets in Tables 1, 2, and 3.

According to a Friedman test [11], the observed differences in average ranks deviate significantly ($p < 0.05$) from what can be expected under the null hypothesis for each of the three criteria. Hence, one can safely conclude that all three criteria are affected by the forest sizes.

Table 1. Accuracy

Dataset	**100 trees**	**1000 trees**	**10 000 trees**	**100 000 trees**
audiology	77.00 (4)	78.50 (1)	77.50 (2.5)	77.50 (2.5)
balance-scale	86.89 (4)	87.68 (2.5)	88.00 (1)	87.68 (2.5)
breast-cancer	72.43 (1)	72.40 (3)	72.40 (3)	72.40 (3)
breast-canc-wisc	96.86 (2.5)	96.86 (2.5)	96.86 (2.5)	96.86 (2.5)
car	94.91 (3.5)	95.02 (1)	94.97 (2)	94.91 (3.5)
clev-heart-disease	55.75 (2)	55.74 (3.5)	56.06 (1)	55.74 (3.5)
crx	87.08 (4)	87.51 (1.5)	87.51 (1.5)	87.37 (3)
cylinder-bands	77.41 (4)	79.81 (3)	80.00 (2)	80.19 (1)
dermatology	98.09 (1)	98.08 (2.5)	98.08 (2.5)	97.81 (4)
ecoli	87.20 (4)	87.50 (2)	87.50 (2)	87.50 (2)
glass	81.28 (3)	81.69 (2)	81.23 (4)	81.71 (1)
hepatitis	86.38 (1.5)	85.79 (3)	85.71 (4)	86.38 (1.5)
house-votes	95.88 (3)	96.11 (1)	95.88 (3)	95.88 (3)
image-segmentation	92.86 (4)	94.29 (1)	93.33 (3)	93.81 (2)
ionosphere	94.59 (1)	94.02 (2)	93.75 (3.5)	93.75 (3.5)
iris	95.33 (2.5)	95.33 (2.5)	95.33 (2.5)	95.33 (2.5)
kr-vs-kp	98.37 (4)	98.62 (2)	98.62 (2)	98.62 (2)
lung-cancer	42.50 (4)	55.83 (2)	55.83 (2)	55.83 (2)
lymphography	81.62 (4)	85.05 (2)	85.05 (2)	85.05 (2)
mushroom	100.00 (2.5)	100.00 (2.5)	100.00 (2.5)	100.00 (2.5)
new-thyroid	98.12 (1)	97.21 (2.5)	96.73 (4)	97.21 (2.5)
pima-indians-diab	76.04 (3)	75.65 (4)	76.43 (1)	76.17 (2)
post-op-patients	65.56 (2.5)	65.56 (2.5)	65.56 (2.5)	65.56 (2.5)
primary-tumor	42.50 (4)	43.39 (1)	42.80 (3)	43.09 (2)
promoters	92.45 (4)	93.36 (2.5)	94.36 (1)	93.36 (2.5)
sick-euthyroid	96.84 (2)	96.90 (1)	96.78 (3.5)	96.78 (3.5)
soybean-large	89.54 (4)	90.52 (1)	89.86 (3)	90.18 (2)
spambase	95.50 (4)	95.65 (2)	95.61 (3)	95.67 (1)
spectf	92.84 (4)	93.99 (1)	93.13 (2.5)	93.13 (2.5)
splice	96.24 (4)	96.80 (1.5)	96.80 (1.5)	96.74 (3)
tae	58.29 (4)	58.92 (3)	58.96 (2)	59.63 (1)
tic-tac-toe	98.96 (4)	99.06 (2.5)	99.06 (2.5)	99.17 (1)
wine	97.75 (4)	98.33 (2)	98.33 (2)	98.33 (2)
yeast	61.67 (2)	61.80 (1)	61.60 (3)	61.53 (4)
Average rank:	3.12	2.06	2.44	2.38

A post-hoc Nemenyi test [11] is employed to investigate if the difference in average ranks for two methods deviate significantly from what can be expected under the null hypothesis. The critical difference, i.e., the minimum difference in average ranks between two methods for which the difference can be considered significant, when comparing 4 methods on 34 datasets, is 0.804 at the $p = 0.05$ level.

The Nemenyi test shows that when comparing the average ranks in Table 1, one can conclude that forests of 1000 trees significantly outperform forests of 100 trees with respect to accuracy. Forests of 10 000 and 100 000 trees suprisingly appear to perform slightly worse than forests of 1000 trees, but the Nemenyi test does not allow to rule out these differences as due to chance.

Table 2. Area under ROC curve

Dataset	**100 trees**	**1000 trees**	**10 000 trees**	**100 000 trees**
audiology	96.94 (1)	96.92 (2)	96.86 (3.5)	96.86 (3.5)
balance-scale	94.60 (4)	94.73 (3)	94.78 (2)	94.79 (1)
breast-cancer	67.43 (4)	67.70 (3)	67.89 (2)	68.01 (1)
breast-canc-wisc	99.29 (4)	99.34 (2.5)	99.34 (2.5)	99.36 (1)
car	99.58 (4)	99.67 (3)	99.68 (1.5)	99.68 (1.5)
clev-heart-disease	81.66 (4)	82.36 (3)	82.50 (2)	82.75 (1)
crx	92.85 (4)	93.29 (1)	93.26 (2)	93.25 (3)
cylinder-bands	88.16 (4)	89.32 (3)	89.65 (1)	89.61 (2)
dermatology	99.95 (1.5)	99.94 (3.5)	99.95 (1.5)	99.94 (3.5)
ecoli	97.01 (4)	97.37 (3)	97.56 (1)	97.53 (2)
glass	94.95 (4)	95.51 (2)	95.61 (1)	95.39 (3)
hepatitis	86.84 (4)	88.09 (2)	87.35 (3)	88.11 (1)
house-votes	99.04 (4)	99.21 (3)	99.25 (1)	99.23 (2)
image-segmentation	99.03 (4)	99.23 (1)	99.21 (2)	99.18 (3)
ionosphere	98.57 (2.5)	98.61 (1)	98.54 (4)	98.57 (2.5)
iris	100.00 (1.5)	100.00 (1.5)	99.87 (3.5)	99.87 (3.5)
kr-vs-kp	99.90 (4)	99.92 (2)	99.92 (2)	99.92 (2)
lung-cancer	70.00 (4)	76.25 (2)	77.50 (1)	74.58 (3)
lymphography	92.52 (4)	92.92 (3)	93.18 (2)	93.35 (1)
mushroom	100.00 (2.5)	100.00 (2.5)	100.00 (2.5)	100.00 (2.5)
new-thyroid	100.00 (1)	99.91 (3)	99.91 (3)	99.91 (3)
pima-indians-diab	82.62 (4)	82.69 (3)	82.75 (1)	82.74 (2)
post-op-patients	36.10 (4)	41.08 (1)	38.92 (2)	37.65 (3)
primary-tumor	82.49 (4)	83.24 (1)	83.09 (3)	83.13 (2)
promoters	98.27 (3)	98.67 (1)	98.27 (3)	98.27 (3)
sick-euthyroid	98.15 (4)	98.22 (1)	98.19 (3)	98.20 (2)
soybean-large	99.23 (4)	99.32 (2.5)	99.32 (2.5)	99.34 (1)
spambase	98.88 (4)	98.93 (2.5)	98.93 (2.5)	98.94 (1)
spectf	97.62 (4)	97.64 (3)	97.80 (1)	97.72 (2)
splice	99.48 (4)	99.50 (2)	99.50 (2)	99.50 (2)
tae	76.05 (4)	76.35 (1)	76.09 (3)	76.23 (2)
tic-tac-toe	99.97 (4)	99.99 (2)	99.99 (2)	99.99 (2)
wine	100.00 (2.5)	100.00 (2.5)	100.00 (2.5)	100.00 (2.5)
yeast	84.99 (4)	85.33 (3)	85.37 (2)	85.40 (1)
Average rank:	3.51	2.22	2.16	2.10

Comparing the average ranks of the forests with respect to AUC, as shown in Table 2, one can by the Nemeniy test conclude that all the three largest forest sizes result in higher AUC than forests of 100 trees, since all three differences in average ranks are above the critical difference (0.804). The differences in AUC between the three largest sizes of forests are however not significant.

Table 3. Brier score

Dataset	**100 trees**	**1000 trees**	**10 000 trees**	**100 000 trees**
audiology	0.4254 (4)	0.4196 (3)	0.4194 (1.5)	0.4194 (1.5)
balance-scale	0.2258 (4)	0.2257 (3)	0.2245 (1.5)	0.2245 (1.5)
breast-cancer	0.3845 (4)	0.3814 (1)	0.3817 (2.5)	0.3817 (2.5)
breast-canc-wisc	0.0499 (2.5)	0.0497 (1)	0.05 (4)	0.0499 (2.5)
car	0.1221 (4)	0.12 (3)	0.1194 (1.5)	0.1194 (1.5)
clev-heart-disease	0.5129 (4)	0.5062 (3)	0.5048 (2)	0.5046 (1)
crx	0.1986 (4)	0.194 (3)	0.1937 (2)	0.1936 (1)
cylinder-bands	0.3008 (4)	0.2976 (3)	0.2965 (1)	0.2968 (2)
dermatology	0.0523 (4)	0.0507 (2.5)	0.0506 (1)	0.0507 (2.5)
ecoli	0.2277 (4)	0.225 (1)	0.2251 (2)	0.2254 (3)
glass	0.3168 (4)	0.3148 (3)	0.3135 (1)	0.3141 (2)
hepatitis	0.2105 (1)	0.214 (4)	0.2137 (3)	0.2135 (2)
house-votes	0.0615 (4)	0.0612 (1.5)	0.0613 (3)	0.0612 (1.5)
image-segmentation	0.1397 (4)	0.1361 (3)	0.136 (2)	0.1359 (1)
ionosphere	0.1117 (4)	0.1109 (1)	0.1116 (3)	0.1112 (2)
iris	0.0603 (1)	0.0621 (2)	0.0627 (3)	0.0628 (4)
kr-vs-kp	0.0415 (4)	0.0404 (1.5)	0.0404 (1.5)	0.0405 (3)
lung-cancer	0.6077 (4)	0.5746 (1.5)	0.5746 (1.5)	0.575 (3)
lymphography	0.2236 (1)	0.2262 (4)	0.2245 (3)	0.2242 (2)
mushroom	0.0005 (3)	0.0004 (1)	0.0005 (3)	0.0005 (3)
new-thyroid	0.0446 (1.5)	0.0446 (1.5)	0.0452 (4)	0.045 (3)
pima-indians-diab	0.3211 (4)	0.3201 (3)	0.3197 (1.5)	0.3197 (1.5)
post-op-patients	0.5159 (4)	0.5092 (1)	0.5098 (2)	0.5104 (3)
primary-tumor	0.7047 (4)	0.7005 (1)	0.7024 (3)	0.7022 (2)
promoters	0.2577 (1.5)	0.2577 (1.5)	0.2603 (3)	0.2606 (4)
sick-euthyroid	0.0561 (4)	0.0552 (1)	0.0553 (2)	0.0554 (3)
soybean-large	0.1916 (4)	0.1855 (3)	0.1839 (2)	0.1838 (1)
spambase	0.076 (4)	0.0745 (2.5)	0.0745 (2.5)	0.0744 (1)
spectf	0.1274 (1)	0.1289 (2)	0.1294 (3.5)	0.1294 (3.5)
splice	0.1443 (4)	0.1415 (2.5)	0.1414 (1)	0.1415 (2.5)
tae	0.5514 (3)	0.5516 (4)	0.5512 (2)	0.5506 (1)
tic-tac-toe	0.0958 (4)	0.0915 (2.5)	0.0914 (1)	0.0915 (2.5)
wine	0.0592 (4)	0.0559 (1)	0.0563 (3)	0.056 (2)
yeast	0.5178 (4)	0.5152 (1)	0.5156 (3)	0.5154 (2)
Average rank:	3.40	2.16	2.25	2.19

Finally, the Nemeniy test shows that forests of 1000, 10 000 and 100 000 trees, all significantly outperform forests of 100 trees with respect to Brier score, as given by the average ranks shown in Table 3. Again, there are no significant differences in performance when comparing the top three methods.

In Table 4, some selected time measurements are shown to give an indication of the computational cost that is required for generating and applying the large-scale forests.[3] It should be noted that if a sequential implementation would have been employed, rather than the parallell version running on a 48-core computer, the experiment would have taken several days to complete.

[3]The times reported are the averages from 10 folds, including both training and testing time. In order to get an estimate for the total time for each experiment, the presented numbers hence have to be multiplied by 10.

Table 4. Timings for some selected datasets (average CPU seconds for each fold)

Dataset	No. ex.	No. feat.	**100 trees**	**1000 trees**	**10 000 trees**	**100 000 trees**
car	1728	6	0.18	0.97	9.10	77.86
crx	689	15	0.09	0.43	4.76	42.45
ecoli	336	7	0.06	0.29	3.88	29.97
glass	214	9	0.04	0.22	2.76	23.28
house-votes	435	16	0.05	0.20	1.76	17.77
kr-vs-kp	3196	36	0.41	2.43	22.95	225.11
mushroom	8124	22	0.74	3.38	26.79	229.40
post-op-patients	90	8	0.02	0.07	0.64	7.84
soybean-large	307	34	0.11	0.66	7.46	60.74
spambase	4601	57	0.77	5.00	48.64	488.01
splice	3190	60	0.54	4.05	34.03	343.12

4. Concluding remarks

A parallell version of the random forest algorithm has been implemented, allowing investigating whether or not performance gains can be obtained from generating very large forests, compared to more moderately sized forests. An empirical investigation on 34 datasets from the UCI repository showed that forests of 1000 trees significantly outperform forests of 100 trees with respect to accuracy, area under ROC curve and Brier score. However, further increasing the sizes up to 10 000 and 100 000 trees did not result in any significant improvements.

Parallell implementations open up possibilities for other experiments with large-scale random forests. One direction for future research is to explore alternative approaches to increasing diversity of the trees, in particular such approaches that may be detrimental for small or moderately sized forests, but which may be beneficial for very large forests only. Such approaches include evaluating fewer features at each node and considering smaller subsets of the training examples.

The current Erlang implementation needs to be further improved in order to scale well when large number of cores are available. The current implementation utilizes a single global data structure (ets table) for all examples, resulting in locks when multiple processes try to access the same data. Distributing data over multiple tables, or using process local data structures, is expected to give closer to linear scaling with large number of cores.

Another direction for future research is to investigate alternative approaches to parallellize the random forest algorithm, including alternative strategies for distributing the tasks of generating subsets of the trees over multiple cores. In case the number of available cores is larger than the trees to be generated, one may even consider parallellizing the tree generation algorithm, something which has been studied earlier in e.g. [1,21]. The latter type of more fine-grained parallellism may also be required for other ensemble approaches for which subsequently generated models are dependent on the ones previously generated, e.g., boosting. Future work also includes comparing the Erlang implementation to parallel implementations on other platforms, including implementations within the MapReduce framework [21], in Fortran 90 [28], in C++ [26] and one that is under development for GPUs [16].

References

[1] N. Amado, J. Gama, and F. Silva. Parallel implementation of decision tree learning algorithms. In P. Brazdil and A. Jorge, editors, *Progress in Artificial Intelligence*, volume 2258 of *Lecture Notes in Computer Science*, pages 34–52. Springer Berlin / Heidelberg, 2001.

[2] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[3] H. Boström. Estimating class probabilities in random forests. In *Proc. of the International Conference on Machine Learning and Applications*, pages 211–216. IEEE Computer Society, 2007.

[4] A. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(6):1145–1159, 1997.

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

[6] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[7] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[8] G. W. Brier. Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78:1–3, 1950.

[9] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proc. of the 23rd International Conference on Machine Learning*, pages 161–168, 2006.

[10] B. Cestnik. Estimating probabilities: a crucial task in machine learning. In *Proc. of the 9th European Conference on Artifical Intelligence*, volume 2157, pages 147–149. Pitman, 1990.

[11] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[12] R. Diaz-Uriarte and S. Alvarez de Andres. Gene selection and classification of microarray data using random forest. *BMC Bioinformatics*, 7(3), 2006.

[13] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[14] U. M. Fayyad and K. B. Irani. On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, 8:87–102, 1992.

[15] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proc. of the International Conference on Machine Learning*, pages 148–156, 1996.

[16] H. Grahn, N. Lavesson, M. Hellborg Lapajne, and D. Slat. A CUDA implementation of random forests - early results. In *Proc. Third Workshop on Multi-core Computing*, pages 33–38. Chalmers Institute of Technology, 2010.

[17] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1), 2009.

[18] H. Liang, Y. Yan, and H. Zhang. Learning decision trees with log conditional likelihood. *International Journal of Pattern Recognition and Artificial Intelligence*, 24(1):117–151, 2010.

[19] A. Liaw and M. Wiener. Classification and regression by randomForest. *R News*, 2(3):18–22, 2002.

[20] A. Niculescu-Mizil and R. Caruana. Predicting good probabilities with supervised learning. In *Proc. of the 22nd international conference on Machine learning*, pages 625–632, New York, NY, USA, 2005. ACM.

[21] B. Panda, J. Herbach, S. Basu, and R. Bayardo. PLANET: massively parallel learning of tree ensembles with MapReduce. *Proc. VLDB Endow.*, 2:1426–1437, 2009.

[22] F. Provost and P. Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3), 2003.

[23] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[24] J. R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann, 1993.

[25] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0.

[26] D. Schwarz, I. König, and A. Ziegler. On safari to random jungle: a fast implementation of random forests for high-dimensional data. *Bioinformatics*, 26(14):1752–1758, 2010.

[27] A. Statnikov, L. Wang, and C. Aliferis. A comprehensive comparison of random forests and support vector machines for microarray-based cancer classification. *BMC Bioinformatics*, 9(319), 2008.

[28] G. Topić, T. Šmuc, Z. Šojat, and K. Skala. Reimplementation of the random forest algorithm. In *Parallel Numerics, Theory and Applications*, pages 119–128. University of Salzburg, 2005.