Working Paper Series
ISSN 1177-777X

RANDOM MODEL TREES: AN EFFECTIVE AND SCALABLE REGRESSION METHOD

Bernhard Pfahringer

Working Paper: 03/2010
June 2010

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

Random model trees: an effective and scalable regression method

Bernhard Pfahringer

University of Waikato, New Zealand,
bernhard@cs.waikato.ac.nz,
http://www.cs.waikato.ac.nz/~bernhard

Abstract. We present and investigate ensembles of randomized model trees as a novel regression method. Such ensembles combine the scalability of tree-based methods with predictive performance rivaling the state of the art in numeric prediction. An extensive empirical investigation shows that Random Model Trees produce predictive performance which is competitive with state-of-the-art methods like Gaussian Processes Regression or Additive Groves of Regression Trees. The training and optimization of Random Model Trees scales better than Gaussian Processes Regression to larger datasets, and enjoys a constant advantage over Additive Groves of the order of one to two orders of magnitude.

Keywords: regression, ensembles, supervised learning, randomization

1 Introduction

Simple linear regression can work very well for arbitrary regression problems, especially when sample sizes are small and when good attributes are provided. Its performance usually breaks down when the relationship between the input and output contains significant non-linearity, and also in linear cases when there is strong collinearity present between pairs of inputs. Non-linear regression algorithms try to overcome these issues of simple linear regression. Still, on small data samples the main effect to be predicted is usually well modeled by a single global linear regression model, even if that effect or relationship in itself is not completely linear. Only when more data becomes available can non-linearity be extracted in a reliable and robust way. Samples sizes of 500 or even several thousand samples might be necessary. Non-linear methods include neural networks, support vector regression, gaussian process regression (GP) [10], and Additive Groves (AG) [12], among others. In this paper we introduce a new tree-based algorithm called Random Model Trees (RMT). We will compare RMTs with linear regression, GPs and AGs. We do not include support vector regression or neural networks in this comparison, as we have found GPs using so-called Radial Basis function kernels to perform as well, and their parameter optimization is simpler involving only two tuning parameters. We also only include AGs and no other tree-based methods like e.g. boosting, as AGs have been shown to perform as least as good as these tree-based alternatives. We will show that RMTs

are competitive in terms of predictive performance to both GPs and AGs, but that they can be an order of magnitude faster than AGs. Furthermore, being tree-based, RMTs scale with $O(NlogN)$, where N is the number of training examples. Therefore they can be applied to much larger problems than GPs.

In the following we will describe the new algorithm in Section 2, in Section 3 we will discuss the algorithms used for comparison, and will focus on parameter optimization. Parameter optimization is important, as the optimal values vary strongly depending on the specific datasets, and therefore no good default values exist. Section 4 will present results for both UCI data [1] as well as some regression problems from an application in Near Infrared Spectroscopy (NIR) [8]. The final section summarizes and present directions for future research.

2 Random Model Trees

Random Model Trees are essentially the combination of two existing algorithms in Machine Learning: single model trees [9] are combined with Random Forest [3] ideas. Model trees are decision trees where every single leaf holds a linear model which is optimised for the local subspace described by this leaf. This works well in practise, as piece-wise linear regression can approximate arbitrary functions as long as the single pieces are small enough. For differentiable functions piece-wise linear regression can also be viewed as a crude one-step Taylor series expansion of such a function. Decision trees split the data into a number of small axis-parallel hypercubes, each of which will have its own local linear model. Issues with learning model trees include high training times searching for optimal splits and optimizing local linear models, potential strong discontinuities in prediction at the borders between hypercubes, and erroneously overshooting extrapolation in sparse areas inside the hypercubes. Smoothing [14] inside a single tree and bagging of multiple trees [2] are standard ways to address these shortcomings.

Trees are also unstable, meaning that small changes in the training data can lead to the construction of trees that differ greatly in structure. While this may be problematic for a single tree, it is possible to take advantage of this effect in an ensemble. Random Forests [3] have shown to improve the performance of single decision trees considerably: tree diversity is generated by two ways of randomization. First the training data is sampled with replacement for each single tree like in Bagging. Secondly, when growing a tree, instead of always computing the best possible split for each node only a random subset of all attributes is considered at every node, and the best split for that subset is computed. Such trees have been used both for classification and for regression, but in the regression setting so far only trees with constant leaf prediction were used, i.e. a regression tree, but a model tree. Random model trees for the first time combine model trees and random forests.

The success and efficiency of Random Model Trees critically depends on some specific engineering features. Determining the best split point for an attribute is expensive: the data must be sorted according to this attribute, and then a linear scan can determine the best split for minimizing the weighted sum squared error

(or a similar numeric loss function). Furthermore best splits are usually not balanced thus leading to potentially very skewed trees, i.e. trees where leaves can have vastly different numbers of examples. This in turn causes issues for the local model generation: to prevent against overfitting some form of regularization is needed. We use ridge regression [7], which like all such regularization methods depends on a user parameter, in this case the ridge value. The problem with large differences in leafsizes is that such regularization parameters strongly depend on the number of training examples. Thus no single good value exists that would work well for such skewed trees. Therefore they would need separate independent optimization at every single leaf, which is expensive.

Random model trees use an alternative approach: trees are approximately balanced by only splitting on the median of some attribute. An approximate procedure for median computation was recently described in [13]. This procedure needs only two linear scans over the data to approximate the median. Random model trees employ this produce for split selection and thus induce reasonably balanced trees where one global setting for the ridge value works across all leaves, thus simplifying the optimization procedure.

Additionally, to prevent against extreme cases of extrapolation, each leaf (or hypercube) records the local minimum and maximum value for the target. Predictions from the local model are then compared to these thresholds and capped, if necessary. This simple procedure has proved very effective, as single extreme values can have a large influence on measures like root mean squared error, even after averaging multiple predictions from an ensemble of model trees.

Finally, as the trees are semi-random and therefore definitely not optimal in isolation, averaging an appropriate number of such trees is essential for good predictive performance. At least 30 trees should always be computed, and computing more (and sometimes a lot more) trees does further improve performance. Of course, due to the random nature of the process, adding more trees to an ensemble will never significantly degrade performance, but as for most ensemble methods any improvements diminish eventually.

Another method for improving a single model is additive regression [4, 6], which iteratively fits the residuals of a predictor or model. As a single model tree usually fits the training data quite well, there is not much scope for additive regression when using model trees. Still, for some datasets a vey small number of additive regression iterations can improve performance. This is usually the case for more shallow trees, where the linear models cover larger areas and thus do not fit every single example well, therefore leaving residuals of reasonable size such that additive regression can extract some more signal from them.

3 Experiments

Random model trees are compared to linear regression, gaussian process regression, and additive groves. This comparison comprises a good number of UCI datasets as well as a number of datasets from an application in Near Infrared (NIR) spectroscopy. We have only included datasets which have at least 950

examples. The names and sizes of the UCI datasets are listed in Table 1, and the info for the NIR datasets is given in Table 2. The NIR data is an interesting addition to the UCI data, as it has substantially different characteristics: first of all the NIR data has more attributes, between 163 and 445 for the processed data we use here, but the raw output of the spectrometers can have up to a thousand attributes. Secondly, as the attributes describe spectrograms which are continuous including peaks but not large discontinuous jumps, attributes are by construction strongly correlated with their neighboring attributes. Such collinearity can be problematic for regression algorithms.

Table 1. Numeric UCI dataset characteristics: the name, the number of numeric attributes, the number of categorical attributes, and the number of examples.

Name	#num	#cat	#examples
stock	9	0	950
quake	3	0	2178
abalone	7	1	4177
delta_ailerons	5	0	7129
bank32nh	32	0	8192
bank8FM	8	0	8192
cpu_act	21	0	8192
cpu_small	12	0	8192
kin8nm	8	0	8192
puma32H	32	0	8192
puma8NH	8	0	8192
delta_elevators	6	0	9517
ailerons	40	0	13750
pol	48	0	15000
elevators	18	0	16599
cal_housing	8	0	20640
house_16H	16	0	22784
house_8L	8	0	22784
2dplanes	10	0	40768
fried	10	0	40768
mv	7	3	40768
layout	31	0	66615
colorhistogram	31	0	68040
colormoments	8	0	68040
cooctexture	15	0	68040
elnino	9	0	178080
census	67	0	2458285

The different algorithms deal differently with either categorical or missing values, therefore all data was preprocessed by replacing categorical values with multiple binary indicator attributes; missing values were imputed using the re-

Table 2. NIR dataset characteristics: the name, the number of (numeric) attributes, and the number of examples.

Name	#num	#examples
omd	171	982
rmd	171	3694
na	171	6363
n	171	7500
tc	439	9849
ph	445	16253
phe	445	16253
p5	163	25904

spective attribute's mean value. This preprocessing should ensure that different algorithm-internal procedures do not impact the comparison.

All of the four algorithms have one or more tuning parameters which need careful optimization for good performance. Additive groves need an explicit validation set for tuning parameters, and the other three algorithms can be paired with some optimization procedure based on a validation set. Therefore the experiments reported here were run by splitting each dataset into three folds, where one third would be used for training, one third for validation, and the final third as an independent test set. The results reported in the next section are accuracies on the independent test set, and total runtimes of the respective optimization procedure over the combined training and validation data. Here are more details for every algorithm:

3.1 Linear Regression

We use the Weka [5] implemementation of Linear (Ridge) Regression, do no internal attribute selection, but vary the ridge parameter from 10^{-8} to 10^{10} in exponential steps of 10, selecting the value with the lowest squared error sum (SSE) on the validation set. Note that the complexity of linear regression is $O(N * K^3)$, where N is the number of examples, and K the number of attributes.

3.2 Gaussian Process Regression

For Gaussian Process Regression we have replaced the generic Weka implementation by a specialized version which firstly hard-codes a Radial Basis function kernel, and secondly uses a conjugate gradient descent solver [11]. These changes result in substantial speedups, usually between one and two orders of magnitude when compared to the standard Weka version. The implementation comprises two tuning parameters: the bandwidth of the kernel, and again a ridge value for the regression. Optimization employs a hill-climbing procedure over a grid of possible pairs of values: starting from some initial point all neighboring grid points are explored until a plateau is reached in terms of SSE on the validation

set. The factor defining the grid for both the bandwidth and the ridge parameter is 2.0, i.e. doubles for going up, and halves for going down are used. Note that the complexity of the conjugate gradient descent solver is only $O(K * N^2)$, i.e. quadratic in the number of examples and linear in the number of attributes, as it is limited to atmost 100 iterations. Usually 100 iterations are enough for full convergence, or at least for getting very close to full convergence. The real problem that GP faces is its memory consumption: as the kernel matrix needs to be precomputed, it takes $O(N^2)$ memory. Thus GPs become infeasible for the largest dataset employed, and also need on the order of about 28 gigabytes of memory for the second largest dataset.

3.3 Additive Groves

We use the C++ implementation as supplied by the authors. That implementation supplies both a "fast" and a "slow" training mode, as well as a Python script for iterative training improvements. As the results in the next section show, this implementation is substantially slower than the other three Weka-based algorithms. Therefore only the "fast" training mode was used. Better prediction would be achieved with more training, but is not feasible here. Even "fast" mode could not finish on the largest dataset. [12] do not discuss the theoretical computational complexity of additive groves, but one would expect $O(K * N * logN)$ behavior from a tree-based algorithm. The results further down seem to confirm this hypothesis, but also show a high constant factor when comparing to the new algorithm, random model trees.

3.4 Random Model Trees

A Weka-based Java implementation is used here. Unfortunately Random Model Trees comprise too many possible tuning parameters. Therefore, in the interest of speed, only a crucial subset of these parameter is optimised, all other paramters used reasonable defaults. The number of randomly chosen attributes to evaluate is set to 10%, but thresholded to be at least 2 and at most 5. Allowing a value of 1 would lead to completely random trees, which are fast to generate, but which also perform worse than semi-random trees. Limiting the maximum ensures diversity of the trees in the ensemble. A low limit of 5 would probably cause break down of tree performance when a large percentage of the attributes is irrelevant or very noisy, but even in these cases the trees would still be reasonably balanced, and the linear leaf models should be able to extract some signal from the useful subset of attributes. Ensemble size is fixed to 100 trees for parameter optimization search. The final ensemble then comprises 300 trees, which usually slightly increases performance again. The two parameters that need optimizing are tree-depth and the ridge value. Optimization uses hill-climbing along one dimension, and switches between dimensions after reaching a plateau in one dimension. Tree depth is modified by $+1$ or -1, the ridge is multiplied by either 0.5 or 2.0, similar to the GP search parameters described above. Finally, when a global plateau for both tree depth and ridge is found, these parameters are fixed to their best

value, and additive regression is tried for an increasing number of models as long as SSE on the validation set decreases. In all experiments reported below optimization chose either no additive regression, or at most two iterations. The complexity of building random model trees is $O(N*logN+N*K^3)$. The first term accounts for tree construction, which is independent of the number of attributes, as only a constant number of attributes will be considered for each split. The second term accounts for training of the local linear models and indicates a potential weakness of random model trees: large number of attributes can slow done training considerably. The results further down clearly confirm this.

4 Results

In this section we compare all four algorithm both with respect to predictive performance as well as to efficiency, measured as build time. The results are split into two parts each, one for UCI data, and one for NIR data, as these datasets have quite different characteristics, which in turn cause qualitative differences of the results.

4.1 Correlation

Correlation is used here as a measure of accuracy, as it is upper-bounded by 1.0 and therefore can be compared in a meaningful way across different datasets. To be precise, r^2 is used, which also allows for direct estimation of the amount of variance in the data that is captured by each model or algorithm.

Figure 1 shows correlation for all four algorithms over the UCI datasets. The datasets have been sorted by the correlation value of linear regression from low to high to facilitate comparison. All three non-linear algorithms usually improve over linear regression or are at least as good. This is absolutely true for Gaussian Processes regression, but both tree-based methods show the occasional catastrophic failure, random model trees on the "elevator" dataset, and additive groves on the "elnino" and the "colorhistogram" data. We have investigated the first of these failures, random model trees on "elevator": basically the optimization procedure goes astray in this case. A more complete grid search is able to find a set of parameters leading to $R^2 = 0.9$, which would be competitive with GP and AG on that dataset. We suppose that using more complete search (using the "slow" setting or the iterative method) for AG on both "elnino" and "colorhistogram" would yield similar improvements. Also noteworthy in this figure is the spectacular success of AG on "puma32H".

The one dataset missing from Figure 1 is the largest set "census". Both GPs and AGs cannot process this data within reasonable amounts of resources. Table 3 displays the results on this dataset for linear regression and for random model trees. RMT provides a small improvement over linear regression, and does so within about 5.5 hours. Given a partial AG training run it is estimated that AG in "fast mode" would need about 25 days to complete. Note however, that AG also supports some parallelism, which was not used here. Similarly, search

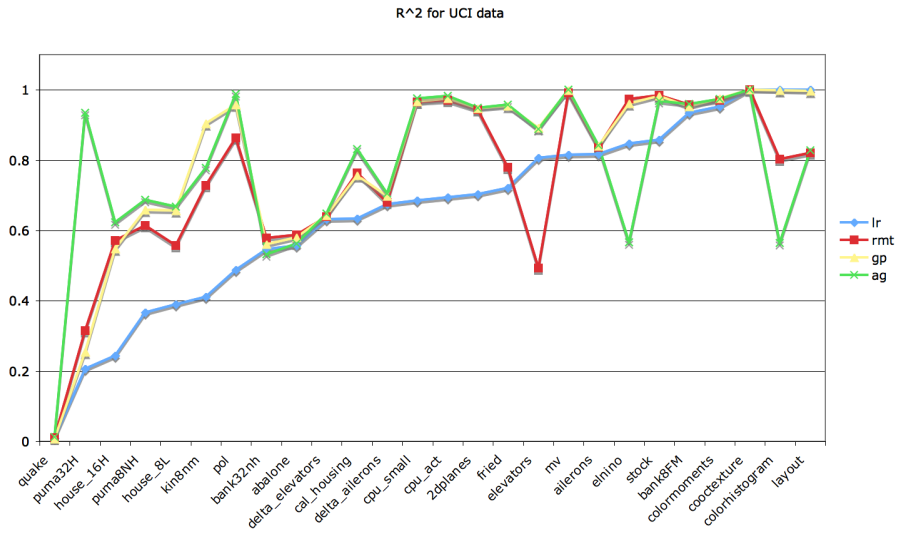

Fig. 1. R^2 for UCI datasets, sorted by the linear regression result.

and ensemble construction of random model trees could be parallelized as well, so parallelism is really an orthogonal issue, which can safely be ignored in this comparison. GPs, or at least our implementation, cannot be employed for this dataset, as it would need about five terabytes of main memory to store the kernel matrix for the 800000 examples training set.

Table 3. Partial results for the UCI Census dataset, 2458285 examples in total, therefore about 800000 in the training fold.

	LR	RMT	GP	AG
Time (secs)	1205	19811	(need 5 Terabyte of RAM) ?	25 days
R^2	0.912	0.932	?	?

Figure 2 shows results for the NIR data, again sorted by the performance of linear regression. The number of datasets is smaller here, so conclusions might be more questionable. Still, there are definite differences present when comparing to the UCI data. First of all there is one dataset where all the non-linear methods are outperformed by simple linear regression, namely the "p5" dataset. Currently we have no good explanation for this behavior. Also, GPs and RMTs seem to perform very similarly, with the exception of the "phe" dataset. AG on the other hand shows more variance, being the best algorithm twice, but also being the worst four times. This is surprising as one would not expect regression trees to

be affected by collinearity in attributes, but that may be a consequence of the strong additive component in AGs, which due its similarity to boosting might overfit correlated attributes.

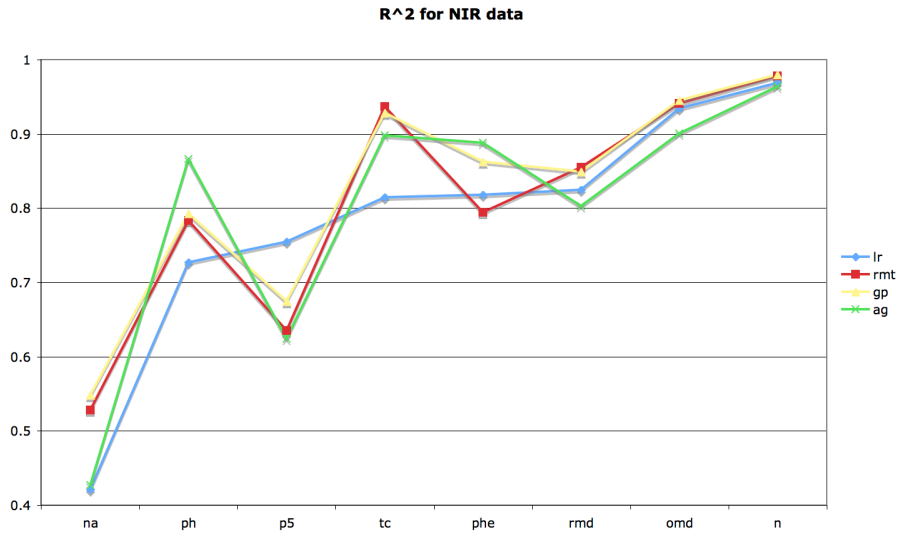

Fig. 2. R^2 for NIR datasets, sorted by the linear regression result.

4.2 Efficiency

While the previous subsection investigated predictive performance, and did not produce a clear winner at all, this subsection looks at build times as the most important resource factor. This aspect produces a very clear ordering of the competing algorithms. Again results are separated into two groups, one for UCI data, and one for NIR data.

Figure 3 plots logarithmic build time in seconds for all algorithms over the UCI datasets being sorted by number of instances, as this is the main complexity factor for most of the algorithms. Linear regression is the fastest for two reasons: all these datasets have 67 or fewer attributes, so the dominating factor is still the number of instances, and its influence is only linear; furthermore optimization concerns only one tuning parameter, looking at only 19 different values. Both tree-based methods RMT and AG show the expected $O(NlogN)$ behavior, but RMTs are consistently one to two order of magnitude faster. The variations visible are explainable by two factors: different numbers of attributes, but more importantly different number of steps in the hill-climbing search, which at times terminates very quickly, and at other times explores a lot more parameter combinations. GPs start very fast for the smallest datasets, but their quadratic

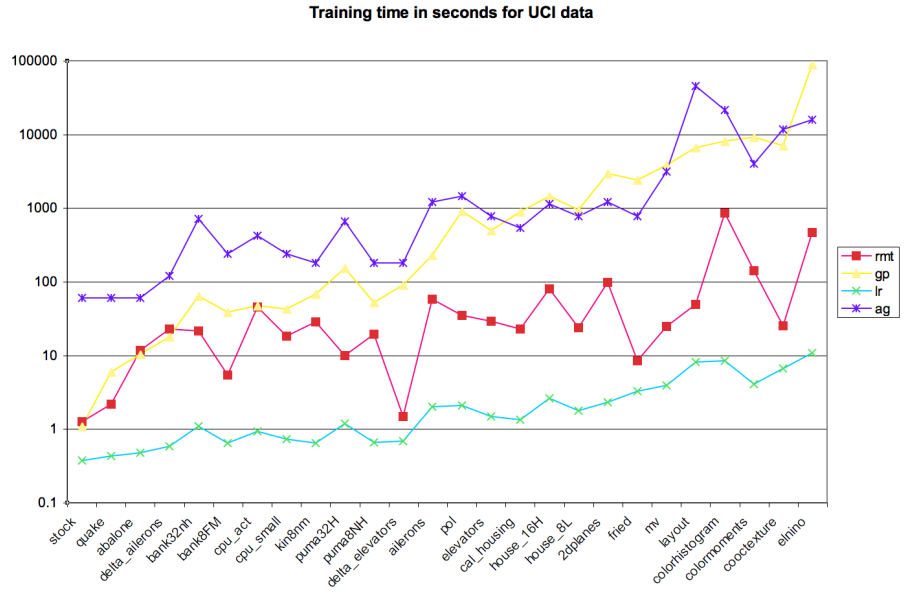

Fig. 3. Training time in seconds for UCI datasets, sorted by the number of instances in each dataset; note the use of a logarithmic y-scale.

complexity is very apparent for larger datasets. For the "elnino" dataset a GP is four orders of magnitude slower than linear regression.

Figure 4 finally present runtimes for the NIR data. Again we see clear relationships, but a reversal when compared to UCI data: GPs are now consistently faster than both tree-based methods, and the gap between the tree-based methods has also narrowed. Again this nicely reflects the theoretical computational complexity of each algorithm: the largest dataset has only about 25000 examples, but all datasets have at least 163 attributes. So RMTs, where the number of attributes has a much larger influence than the number of examples, fall behind GPs, which are linear in the number of attributes.

5 Conclusions

We have introduced a new general regression method that combines model trees with random forests and some engineering details in a novel way. A comparison to linear regression and to two other state-of-the-art regression algorithms over a substantial set of datasets of a wide range of properties has shown that the new algorithm can be competitive to the state of the art regarding predictive performance, but that it is considerably more efficient on datasets with relatively few attributes, and that it can scale reasonably to datasets of hundreds of thousands of examples. Still, when outmost predictive performance is needed in

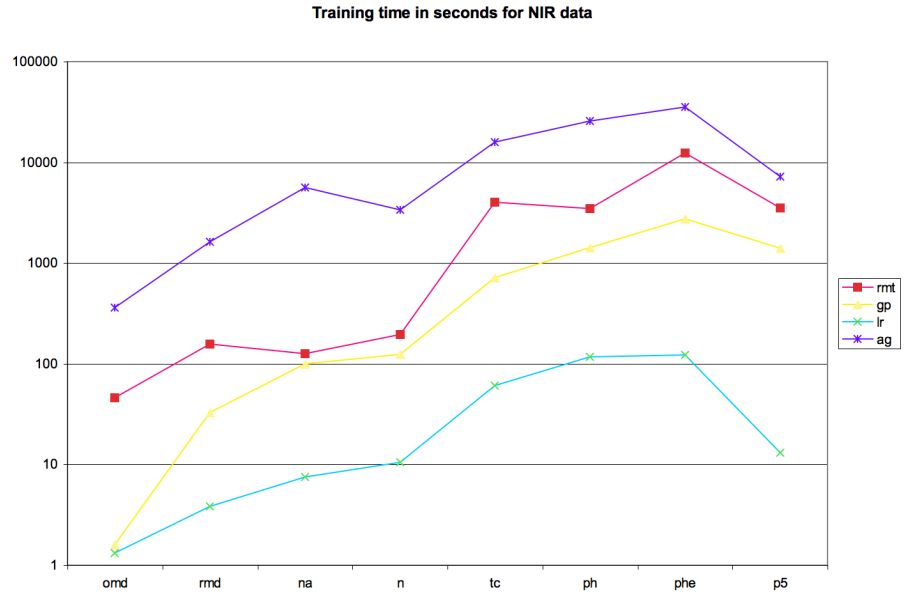

Fig. 4. Training time in seconds for NIR datasets, sorted by the number of instances in each dataset; note the use of a logarithmic y-scale.

an application, an ensemble of well-tuned GPs, AGs, and RMTs would be the method of choice, provided enough computing resources are available.

There are a number of promising directions for future research. The most important one for random model trees is the issue of its complexity in the number of attributes. Either some form of local feature space reduction at each leaf in isolation, or some more global form of feature space reduction either per single tree or for the full ensemble can be explored. Local feature space reduction will have to be very careful with regard to runtime, but also potential loss of information. Another interesting direction will be investigating the possibility of a hybrid of the random model trees and additive groves ideas. And last but not least investigating efficient gaussian process regression for large datasets is a very challenging endeavour. Sparsification and gradient descent methods are potential candidates, but in regression settings they seem to trade off too much of the GP's predictive power for speed and memory savings. Again maybe a hybrid between Gaussian process regression and random model trees might provide a viable alternative.

References

1. Asuncion, A., Newman, D.J.: UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science (2007).

2. Breiman L.: Bagging predictors. Machine Learning 24 (2): 123-140 (1996).
3. Breiman L.: Random Forests. Machine Learning 45 (1): 5-32 (2001).
4. Friedman, J.H.: Multivariate Adaptive Regression Splines. Annals of Statistics 19, 1 (1991).
5. Hall M., Frank E., Holmes G., Pfahringer B., Reutemann P., Witten I.H.: The WEKA Data Mining Software: An Update. SIGKDD Explorations, Volume 11, Issue 1 (2009).
6. Hastie, T., Tibshirani, R., Friedman, J.H.: Elements of Statistical Learning. Springer (2001).
7. Hoerl A.E., Kennard R.W.: Ridge regression: Biased estimation for nonorthogonal problems. Technometrics, 12(3), 55-67, (1970).
8. Pfahringer B., Fletcher D., Bouckaert R., Holmes G.: Random model trees: a competitive off-the-shelf technology for NIRS. The 14th International Conference on Near Infrared Spectroscopy, NIR2009; 78-78 (2009).
9. Quinlan J.R.: Learning with continuous classes. Proceedings Australian Joint Conference on Artificial Intelligence, 343-348 (1992).
10. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press, 2006.
11. Saad Y.: Iterative Methods for Sparse Linear Systems. Society for Industrial and Applied Mathematics (2003).
12. Sorokina D., Caruana R., Riedewald M.: Additive Groves of Regression Trees. Proc European Conference on Machine Learning, 323-334, (2007).
13. Tibshirani R.J.: Fast Computation of the Median by Successive Binning. Unpublished manuscript, http://stat.stanford.edu/~ryantibs/median/ (2008).
14. Wang Y., Witten I.H.: Induction of model trees for predicting continuous classes. Proc European Conference on Machine Learning Poster Papers, 128-137, (1997).