Supervised learning with decision tree-based methods in computational and systems biology

Supplementary material

Pierre Geurts, Alexandre Irrthum, Louis Wehenkel
Department of EE and CS & GIGA-Research, University of Liège, Belgium

The first section of this supplementary material gives an overview of several more or less advanced extensions of decision tree-based methods that have been considered in the literature. The second section provides a few practical guidelines on how to best exploit all these methods. Pointers towards freely available software packages implementing these methods are given in the third section. The document ends with a "hands-on" illustration in the **R** statistical language.

Extensions

There exist many extensions of the basic decision tree learning algorithm described in the paper. We focus here on a few generic ones and refer the interested reader to the references below for further details. More specific improvements for solving computational biology problems are also mentioned in the 'Applications' section of the paper.

Handling of input information. Standard decision trees are designed for simple numerical or symbolic input attributes, for which they propose very simple univariate binary tests. Many works have proposed to extend these candidate tests to handle more complex problems. Extensions can be divided into two families: generic extensions that still apply only to numerical or symbolic attributes and specific extensions to deal with more structured data types.

One popular generic extension is to consider multivariate splits that combine several input attributes at a time. When inputs are numerical, a linear multivariate split may be defined as follows: $[\sum_i \alpha_i X_i + \alpha_0 < 0?]$, where the α_i's are parameters that need to be determined during tree induction. Trees with linear splits are often called oblique trees in the literature. Several heuristics have been proposed to optimize these parameters [7, 25]. Since these splits are not anymore constrained to be axis-parallel, they can more easily approximate a smooth decision boundary. However, the use of multivariate splits significantly increases the computational complexity of the induction algorithm and the resulting trees are also more difficult to interpret than those based on univariate splits. This extension is also less useful in the context of ensemble methods, since these latter are able to smoothly approximate oblique decision boundaries with individual trees based only on univariate splits (see Figure 3 in the paper).

Decision trees have also been extended to handle complex structured input data types such as time series [15], graphs [23], or images [1]. For example in [23], the authors extend decision tree tests for graph classification. They define candidate tests that detect the occurrence of subgraphs in a (labeled) graph and provide an efficient algorithm to find the subgraph that maximizes the score at a given tree node. They exploit this idea in the context of AdaBoost with decision stumps.

Handling of output information. In addition to candidate tests, there exist also several alternatives to the simple node labeling procedures that have been described above.

The first basic extension consists of *regression trees*, which instead of symbolic labels or class probability vectors attached to their terminal nodes, use real numbers in order to approximate a numerical output variable. Such standard (single) regression trees are equivalent to piecewise constant models. One of their extensions consists in labeling the leaves with functions of the inputs, such as for example linear regression functions [28, 14].

While decision trees have been mainly used for predicting a single discrete or numerical output, there exist extensions of these methods to deal with multivariate or even more complex outputs [30, 5, 17]. The main idea of these methods is to adapt both the score measure used to evaluate and select decision tree splits and the way predictions are computed at decision tree leaves to take into account the additional complexity of the output space. These extensions allow one to tackle with tree-based methods problems such as hierarchical multi-label classification [33], ranking [32], image [17] or graph completion [18].

Semantics. Finally, let us mention extensions of decision trees that significantly modify their semantics and accordingly their induction algorithm.

First-order logical decision trees [4] are a modification of the decision tree semantics based on using conjunctions of first order logic predicates at their test nodes.

Fuzzy decision trees, on the other hand, are based on the exploitation of principles from fuzzy logic [26]. In fuzzy trees, objects may be propagated along several paths and reach several terminal nodes with different degrees of membership. The output is then computed by aggregating the labels of all the terminal nodes reached by an object. Similar extensions have also been proposed based upon probabilistic [9] or statistical principles [19].

A decision tree can be converted to a set of (mutually exclusive) rules, each one corresponding to a tree branch. Algorithms have been proposed to learn directly sets of rules (that may not be representable by a tree) [10] or to simplify the set of rules corresponding to a decision tree [27].

The alternating decision tree method [11] is a classification algorithm that tries to combine the interpretability of decision trees with the accuracy improvement obtained by boosting.

MARS, for multiple adaptive regression splines [13], can be seen as a modification of the decision tree algorithm in order to better handle high-dimensional regression problems for which piecewise constant models are too rough.

A few practical guidelines

We try to give in this section a few guidelines on how to best exploit these decision tree-based methods. See also the last section of this supplementary material for a **R** script that implements some of these ideas on an illustrative problem.

Pre-processing

Tree-based methods do not require much pre-processing, except that the data must be storable in a standard attribute-object table. Numerical input variables do not need to be normalized and categorical input variables can be treated as such. Although it is always a good idea to remove irrelevant variables, tree-based methods are quite robust to their presence and there is thus usually no need to filter variables prior to building the model. With the goal of obtaining the last percent of accuracy, it is however often possible to get some improvement by applying some feature selection techniques to remove these irrelevant variables (using for example tree-based attribute importance measures themselves) prior to building a predictive model with the tree-based supervised learning methods. Like other machine learning methods, decision trees typically suffer in the presence of unbalanced datasets, ie., datasets which contain many more examples from one class than of the others. Although sophisticated approaches exist to deal with this problem [3], a simple and often appropriate way to handle it is to merely under-sample the majority class so that it contains a comparable amount of examples with respect to the other classes.

Accuracy as a priority

If accuracy is the main concern, single trees can be ruled out in favor of ensemble methods without any risk. Among existing ensemble methods, it's usually difficult to predict which method will be the best. Randomization and boosting methods have different properties and it is a good practice to compare both of them. Among ensemble methods based on randomization, bagging is usually ruled out by more recent approaches such as Random Forests [6] (of which bagging is a special case anyway) or Extremely Randomized Trees [16]. The accuracy is typically a monotonically increasing function of the number of trees in the ensemble[1] and it is sufficient to build a large amount of trees (from 100 to 1000). Besides the number of trees, all methods typically depend on the setting of a (small) number of

[1]Although this is not always true for boosting methods.

parameters. Although default values of these parameters usually yield excellent results, when necessary, some improvement could be gained by tuning these values to the problem at hand, using for example cross-validation.

To fix ideas, a typical trial with Random Forests would be as follows:

- Split the dataset into a learning sample LS and a test sample TS
- Build a first Random Forest model on LS using the default parameterization of the method (in most software packages, this means to leave everything unchanged) and a fairly large amount of trees (typically 500-1000);
- Evaluate the accuracy of this model on the test sample TS.
- If the accuracy results are not acceptable, then play with different values of the parameter K (see paper for its meaning):
  - Split the learning sample LS into two new subsamples S_1 and S_2;
  - Build on S_1 Random Forests ensembles with increasing values of K, ranging from 1 to the total number of attributes;
  - Select the value K^* that leads to best accuracy on S_2.
- Build a model with this optimal setting on LS and validate it on TS to get an estimate Acc_{final} of its accuracy;
- The final model is obtained by rerunning the algorithm on the whole dataset (learning sample plus test sample). An estimation of its error on new data is given by Acc_{final}.

When the dataset or the learning sample is too small, splitting the data should be replaced by cross-validation[2].

Note that when accuracy is the only concern, it is always a good idea to include in the trial process other machine learning methods such as neural networks or support vector machines. Notice however that the test sample accuracy, if it is used to select the best among several methods, will be an optimistically biased estimate of the true accuracy of the finally chosen model. An additional independent subset (or an external round of cross-validation) should thus be used to obtain a less biased estimate of the true error rate.

Interpretability as a priority

If getting new insights about the input-output relationship is the main concern, then one typically starts by looking at a single pruned decision tree, which gives a first (possibly rough) picture of the relationship. When interpreting this tree, one has however to keep in mind the shortcomings of single trees such as their high variance or their limited ability to deal with correlated or redundant features.

A single tree is generally very usefully complemented by the ranking of the variables provided by ensemble methods, which do not suffer from the same problems.

Note that, when dealing with interpretability only, single trees and ensembles should be built using the full dataset to exploit every bit of information that is available. Looking at the accuracy beforehand is however a good practice since it gives information about the quality of the models; one should not pay attention to an attribute ranking if the classification accuracy of the corresponding model is not better than a random guess for example. Although the best methods in terms of accuracy is not ensured to give the most relevant ranking [22], it still makes sense to use it in priority in an interpretability analysis.

Software packages

There exist many software implementations of decision tree-based supervised learning. Here we present some free, well maintained and well documented implementations for which the source code is available.

[2]In the particular case of Random Forests, an accuracy estimate can be obtained at no cost by exploiting for each tree the examples that are not part of the corresponding bootstrap sample. The resulting estimate is called the "out-of-bag" estimate.

Weka[3] is a toolkit implemented in the Java programming language. It implements algorithms related to decision trees, such as a C4.5 variant, regression trees and Random Forests. The algorithms can be accessed programmatically in Java, or with a graphical user interface that allows the comparison of the performances of algorithms and their parameters, and visualization of their results.
Orange[4] is a comprehensive collection of machine learning algorithms. It includes implementations of C4.5 decision trees, regression trees and Random Forests. The algorithms can be accessed programmatically in the C++ programming language or with the Python programming language. Analysis pipelines can also be constructed in a graphical user interface by combining visual components.
Libraries for the *R statistical programming language*[5] exist for various tree algorithms. The *rpart* package implements decision and regression trees, the *randomForest* package implements Random Forests, and the *boost* package implements various boosting algorithms. *Party* is another package that implements various decision tree-based methods including single trees and forests [21]. These libraries are accessible programmatically in **R**. An advantage of using **R** is the compatibility with the rich bioinformatics libraries implemented as part of the Bioconductor project[6].
Random Forests(tm)[7] are the original implementations, by their inventor Leo Breiman, of Random Forests for classification, regression, and survival analysis, in the programming language FORTRAN 77. The web site also provides RAFT (RAndom Forest Tool), a Java-based visualization tool for the interpretation of Random Forests.
JBoost[8] is a library in the Java programming language that focuses on boosting algorithms. It provides extensions of algorithms to multi-class and multi-label problems, alternating decision trees and visualization tools.

Illustrative application in R

In this section, we illustrate the use of tree-based methods in the **R** statistical language [29]. We choose this language for its popularity in bioinformatics and the availability of several packages related to tree-based methods. We assume that the reader is familiar with the general semantics of **R**.

For single trees, Random Forests, and boosting, we use respectively the rpart [31], randomForests [24], and RWeka [20] packages (the latter giving access to all machine learning methods available in Weka [34], including another implementation of single trees, called J48, and Random Forests). This choice is arbitrary and there are other packages in **R** that implement these methods, such as those mentioned in the previous section about Software packages. Our goal here is not to give a full overview of these methods in **R**, but merely to illustrate the typical steps that are involved in the application of tree-based methods. We refer the reader to the documentation of the **R** language [29] and these specific packages for more detailed information and further illustrations.

The following commands install the required packages and make them available in the current environment.

```
> install.packages("rpart")
> install.packages("randomForest")
> install.packages("RWeka")

> library(rpart)
> library(randomForest)
> library(RWeka)
```

As in illustrative problem, we use the Splice (or DNA) dataset from the UCI machine learning repository [2], available in a suitable format in the supplementary file splice_data.txt. The goal of this problem is to predict boundaries between exons and introns in gene sequences of primates. The dataset contains 3186 examples corresponding each to a primate DNA segment of 60 nucleotides. Each segment is classified into one of three classes: *ie* when the segment is centered at the junction between an intron

[3]`http://www.cs.waikato.ac.nz/ml/weka`
[4]`http://www.ailab.si/orange`
[5]`http://www.r-project.org`
[6]`http://www.bioconductor.org`
[7]`http://www.stat.berkeley.edu/~breiman/RandomForests`
[8]`http://jboost.sourceforge.net`

and an exon, *ei* when the segment is centered at the junction between an exon and an intron, and *neither* when the segment is not centered at a particular junction.

The following command loads this dataset in the R environment:

```
> DNA <- read.csv("splice_data.txt", header=T)
```

Getting the most accurate predictions. In order to simulate a real problem, we will divide the data into two subsets: one will be the learning sample for which we know the output and the other will be the test sample for which we are interested in making the best possible predictions.

```
> l <- length(DNA[,1])
> ls <- sample(1:l,2*l/3)
> DNA.ls <- DNA[ls,]
> DNA.ts <- DNA[-ls,]
```

In order to select the best model among several, we will further split the learning sample into two samples: S_1 used for learning the models and S_2 used to estimate their accuracy and select the best among them.

```
> lls <- dim(DNA.ls)[1]
> s1 <- sample(1:lls,2*lls/3)
> DNA.ls.s1 <- DNA.ls[s1,]
> DNA.ls.s2 <- DNA.ls[-s1,]
```

We will compare single trees, Random Forests, and adaboost.

The following code builds a pruned decision tree on S_1 and estimates its accuracy on S_2.

```
# build a fully grown tree
> model.dt <- rpart(Class ~ .,method="class", data=DNA.ls.s1)
# prune it using 10-fold cross-validation
> model.prunedddt <- prune(model.dt,
+                   cp=model.dt$cptable[which.min(model.dt$cptable[,"xerror"]),"CP"])
# compute predictions on S2
> pred.pruneddt <- predict(model.pruneddt,newdata=DNA.ls.s2,type="class")
# estimate accuracy
> acc.pruneddt <- 100*sum(pred.pruneddt==DNA.ls.s2$Class)/dim(DNA.ls.s2)[1]
> acc.pruneddt
[1] 94.051
```

Please note that, because of the random splitting of the dataset and the stochastic nature of the algorithms, the results that you could obtain may differ slightly from those given here.

The following code builds a random forest model on S_1 and tests it on S_2.

```
# Build a random forest model
> model.rf <- randomForest(Class~.,ntreeTry=500,data=DNA.ls.s1)
# compute predictions on s2
> pred.rf <- predict(model.rf,newdata=DNA.ls.s2,type="class")
# estimate accuracy
> acc.rf <- 100*sum(pred.rf==DNA.ls.s2$Class)/dim(DNA.ls.s2)[1]
>acc.rf
[1] 96.17564
```

As expected, the accuracy is higher than that of the single pruned tree. Let us now try to optimize the parameter K of Random Forests using the tuneRF function provided in the randomForests package (that exploits out-of-bag estimates).

```
# tune the parameter K using out-of-bag estimates
> model.rftuned <- tuneRF(DNA.ls.s1[,1:60],DNA.ls.s1$Class,ntreeTry=500,
+                         improve=0.001,stepFactor=1.5,plot=TRUE,doBest=TRUE)
> pred.rftuned <- predict(model.rftuned,newdata=DNA.ls.s2,type="class")
```

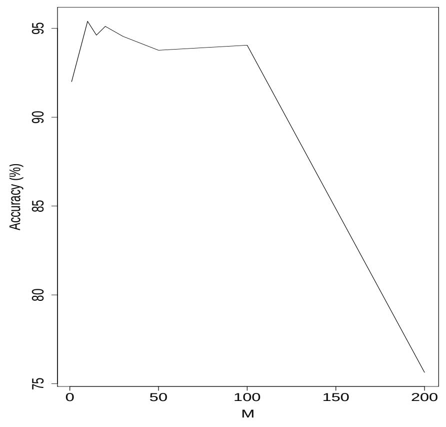

Figure 1: Evolution of the accuracy of AdaBoost with the complexity parameter M.

```
# estimate accuracy
> acc.rftuned <- 100*sum(pred.rftuned==DNA.ls.s2$Class)/dim(DNA.ls.s2)[1]
> acc.rftuned
[1] 96.60057
```

We observe that a slight improvement is achieved when we optimize K. In our case, the analysis shows that the best accuracy (maximum of the out-of-bag accuracy) is achieved for $K = 10$.

Let us now try the AdaBoost algorithm [12]. As mentioned in the paper, AdaBoost combines weak models. In this experiment, we will use "pruned" decision trees obtained by changing the parameter M that controls the minimum number of objects required for a leaf to be splitted. The following code selects the optimal value of this parameter by using 5-fold cross-validation on S_1, then build a model on S_1 using this optimal value and test it on S_2.

```
# List of values of M to consider (from smaller to larger trees)
> lm <- c(200,100,50,30,20,15,10,1)
> acc <- NA
# cycle through all values
> for (i in 1:length(lm)) {
+  m.adaboost <- AdaBoostM1(Class~., data=DNA.ls.s1,
+                     control=Weka_control(I=500,W=list(J48,M=lm[i],U=TRUE)))
+  eval <- evaluate_Weka_classifier(m.adaboost,numFolds=5,seed=123)
+  acc[i] <- eval$details[1]
+ }
# plot evolution of the 5-fold CV accuracy with respect to M
> plot(lm,acc,type="l",xlab="M",ylab="Accuracy (%)")
# build a new model with the optimal setting
> optm <- lm[which.max(acc)]
> model.adaboost <- AdaBoostM1(Class~., data=DNA.ls.s1,
+                     control=Weka_control(I=500,W=list(J48,M=optm,U=TRUE)))
# Evaluate its accuracy
> pred.adaboost <- predict(model.adaboost,newdata=DNA.ls.s2,type="class")
> acc.adaboost <- 100*sum(pred.adaboost==DNA.ls.s2$Class)/dim(DNA.ls.s2)[1]
> acc.adaboost
[1] 95.46742
```

The obtained accuracy is slightly lower than that obtained with Random Forests. Figure 1 shows the evolution of the 5-fold cross-validation accuracy with respect to M. We observe that for this case, the best accuracy is obtained when $M = 10$

Globally, the best model turns out to be the Random Forest model with $K = 10$, whose accuracy estimated on S_2 is 96.6%. To get the final predictions on the test sample, we then have to learn a new model on the whole learning sample.

```
> finalmodel <- randomForest(Class~.,ntreeTry=500,data=DNA.ls,mtry=10)
> finalpred <- predict(finalmodel,newdata=DNA.ts,type="class")
```

Let us now evaluate the accuracy of this model on the test sample:

```
> finalacc <- 100*sum(finalpred==DNA.ts$Class)/dim(DNA.ts)[1]
> finalacc
[1] 97.16714
```

This estimate is close to the estimate on S_2 that guided the choice of the Random Rorests model.

Getting new insights. A first picture of the input-output relationship can be obtained by looking at a pruned decision tree grown on the whole available dataset (ie. the learning sample in our hypothetical experiment):

```
# grow and prune the decision tree
> model.dt <- rpart(Class ~ .,method="class", data=DNA.ls)
> model.pruneddt <- prune(model.dt,
+                    cp=model.dt$cptable[which.min(model.dt$cptable[,"xerror"]),"CP"])
# plot it
> par(mfrow=c(1,2), xpd=NA)
> plot(model.pruneddt, uniform=TRUE, main="Pruned Tree for DNA")
> text(model.pruneddt, use.n=TRUE, cex=.8, pretty=0)
```

The decision tree is shown in Figure 2. It uses only 7 variables among the 60 available. These variables correspond to nucleotides at positions 28, 29, 30, 31, 32, 33 and 35, i.e. the three nucleotides immediately before and the three nucleotides immediately after the potential splice junction, plus one nucleotide a bit farther. A finer analysis can be obtained by looking at particular tree leaves. For example, the leftmost leaf contains a majority of examples of the *ei* class and corresponds to the motif $G|GT--G$ (where $|$ symbolizes the junction and $-$ symbolizes an unspecified nucleotide). This motif corresponds to the consensus 5́-splice site motif described in the literature [8].

Random Forests can be used to get a quantitative ranking of variables:

```
# grow a random forest model with variable importance computation
> model.rf <- randomForest(Class~.,ntreeTry=500,data=DNA.ls,importance=TRUE,mtry=10)
# plot variable importances (without sorting)
> varImpPlot(model.rf,type=1,sort=FALSE,n.var=60,main="Variable Importance for DNA")
```

The variable importance plot is shown in Figure 2. The importance of a variable was obtained by computing the mean decrease of accuracy on the out-of-bag examples when permuting this variable. Variables are sorted on the y-axis according to their position in the DNA sequence. From this picture, it is clear that nucleotides close to the junction are the most useful to make predictions, which was expected. The top five most important variables (positions) are N30, N29, N32, N31, N35, in this order. This is very similar to the results obtained with the pruned decision tree, but depending on the problem some differences are expected. 2.

References

[1] Y Amit and D Geman. Shape quantization and recognition with randomized trees. *Neural computation*, 9(7):1545–1588, 1997.

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

[3] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *Sigkdd Explorations*, 6:2004, 2004.

[4] H. Blockeel and L. de Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.

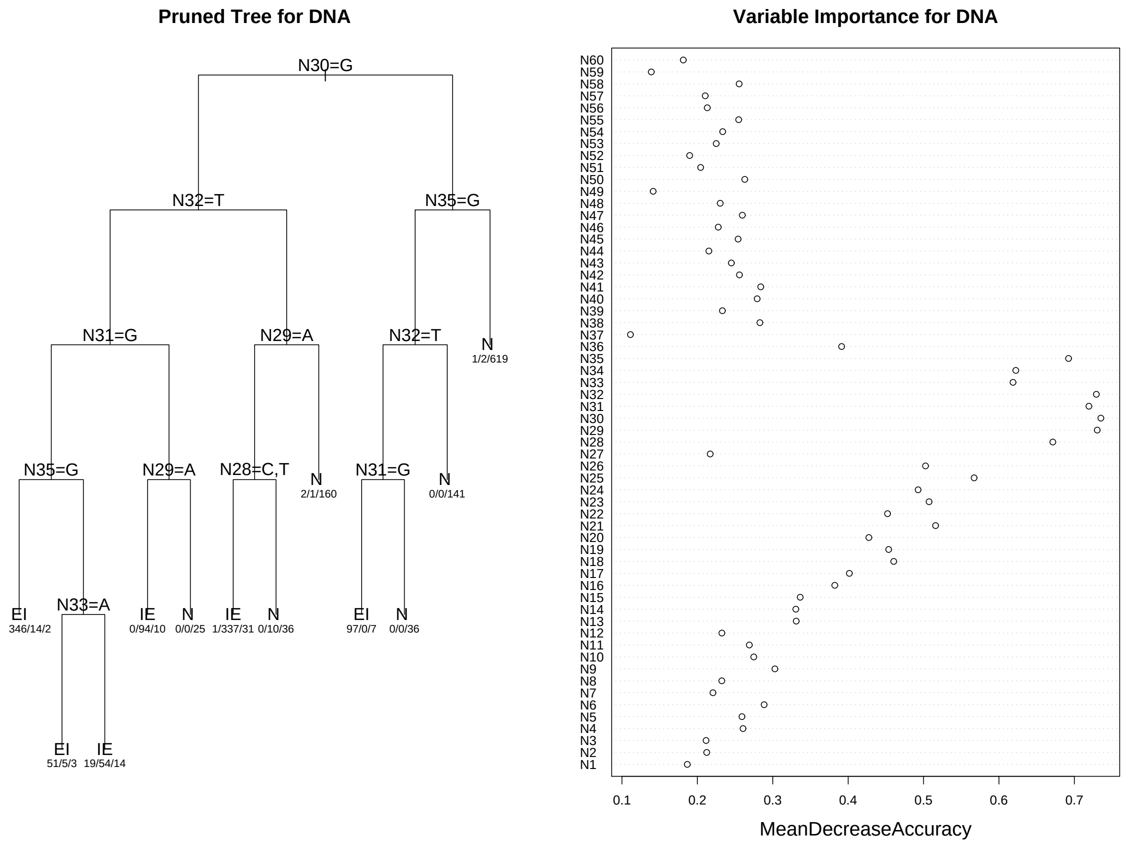

Figure 2: Left, the single pruned tree built on the DNA dataset: the left branch of each test node corresponds to the test being true, the right branch to the test being false; leaf nodes are labeled by the majority class and the distribution of examples among classes (from left to right, *ei*, *ie*, and *neither*). Right, variable importance plot obtained with Random Forests. Variables on the y-axis are ordered according to their position on the nucleotide segment

[5] H. Blockeel, L. De Raedt, and J. Ramon. Top-down induction of clustering trees. In *Proceedings of ICML 1998*, pages 55–63, 1998.

[6] L. Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[7] L. Breiman, J.H. Friedman, R.A. Olsen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth International (California), 1984.

[8] Terrence A. Brown. *Genomes*. Wiley-Liss, New York, 2nd edition, 2002.

[9] Wray Buntine. Learning classification trees. *Statistics and Computing*, 2(2):63–73, 1992.

[10] P. Clark and T. Niblett. The cn2 induction algorithm. *Machine learning*, 3(4):261–283, 1989.

[11] Y. Freund and L. Mason. The alternating decision tree algorithm. In *Proceedings of the 16th International Conference on Machine Learning*, pages 124–133, 1999.

[12] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the second European Conference on Computational Learning Theory*, pages 23–27, 1995.

[13] J.H. Friedman. Multivariate adaptive regression splines. *Annals of statistics*, 19:1–141, 1991.

[14] J. Gama. Functional trees. *Machine Learning*, 55:219–250, 2004.

[15] P Geurts. Pattern extraction for time series classification. *Proceedings of the European Conference on Machine Learning*, 2001.

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

[17] P. Geurts, L. Wehenkel, and F. d'Alché Buc. Kernelizing the output of tree-based methods. In *Proc. of ICML*, pages 345–352, 2006.

[18] Pierre Geurts, Nizar Touleimat, Marie Dutreix, and Florence d'Alché Buc. Inferring biological networks with output kernel trees. *BMC Bioinformatics*, 8 Suppl 2:S4, 2007.

[19] Pierre Geurts and Louis Wehenkel. Closed-form dual perturb and combine for tree-based models. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 233–240, New York, NY, USA, 2005. ACM.

[20] Kurt Hornik, Achim Zeileis, Torsten Hothorn, and Christian Buchta. *RWeka: An R Interface to Weka*, 2009. R package version 0.3-16.

[21] T. Hothorn, K. Hornik, and A. Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, 15(3):651–674, 2006.

[22] V.A Huynh-Thu, L Wehenkel, and P Geurts. Exploiting tree-based variable importances to selectively identify relevant variables. *JMLR Workshop and Conference proceedings*, 4:60–73, 2008.

[23] T Kudo, E Maeda, and Y Matsumoto. An application of boosting to graph classification. *Advances in Neural Information Processing Systems*, 17:729–736, 2005.

[24] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

[25] S.K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 1994.

[26] Cristina Olaru and Louis Wehenkel. A complete fuzzy decision tree technique. *Fuzzy Sets and Systems*, 138:221–254, 2003.

[27] J.R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann (San Mateo), 1986.

[28] R. Quinlan. Learning with continuous classes. In *Proceedings of artificial intelligence'92*, pages 343–348. World Scientific, 1992.

[29] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[30] M. Segal. Tree structured methods for longitudinal data. *Journal of the American Statistical Association*, 87:407–418, 1992.

[31] Terry M Therneau and Beth Atkinson. R port by Brian Ripley. *rpart: Recursive Partitioning*, 2009. R package version 3.1-43.

[32] L. Todorovski, H. Blockeel, , and S. Dzeroski. Ranking with predictive clustering trees. In *Proc. of the 13th European Conference on Machine Learning*, volume LNAI 2430, pages 444–456, 2002.

[33] Celine Vens, Jan Struyf, Leander Schietgat, S. Dzeroski, and H. Blockeel. Decision trees for hierarchical multi-label classification. *Machine Learning*, 73(2):185–214, 2008.

[34] Ian H. Witten and Frank Eibe. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2005.