Wang, M., Zhang, M., Chen, X. & Zhang, H. (2009). Detecting genes and gene–gene interactions for age-related macular degeneration with a forest-based approach. *Stat. Biopharm. Res.*, **1**, 424–430.
Yao, L., Zhong, W., Zhang, Z., Maenner, M. & Engelman, C. (2009). Classification tree for detection of single-nucleotide polymorphism (SNP)-by-SNP interactions related to heart disease: Framingham Heart Study. *BMC Proceedings*, **3**, S83.
Zhang, H. (1997). Multivariate adaptive splines for analysis of longitudinal data. *J. Comput. Graph. Statist.*, **6**, 74–91.
Zhang, H. (2004). Mixed effects multivariate adaptive splines model for the analysis of longitudinal and growth curve data. *Stat. Methods Med. Res.*, **13**, 63–82.
Zhang, H. & Bonney, G. (2000). Use of classification trees for association studies. *Genet. Epidemiol.*, **19**, 323–332.
Zhang, H., Legro, R.S., Zhang, J., Zhang, L., Chen, X., Huang, H., Casson, P.R., Schlaff, W.D., Diamond, M.P., Krawetz, S.A., Coutifaris, C., Brzyski, R.G., Christman, G.M., Santoro, N. & Eisenberg, E. (2010). Decision trees for identifying predictors of treatment effectiveness in clinical trials and its application to ovulation in a study of women with polycystic ovary syndrome. *Human Reproduction*, **25**, 2612–2621.
Zhang, H., Yu, C. & Singer, B. (2003). Cell and tumor classification using gene expression data: Construction of forests. *Proceedings of the National Academy of Sciences*, **100**, 4168–4172.
Zhang, H., Yu, C., Singer, B. & Xiong, M. (2001). Recursive partitioning for tumor classification with gene expression microarray data. *Proceedings of the National Academy of Sciences*, **98**, 6730–6735.

[*Received March 2014, accepted March 2014*]

International Statistical Review (2014), 82, 3, 361–367 doi:10.1111/insr.12062

Thomas Rusch[1] and Achim Zeileis[2]

[1]*WU Vienna, Vienna, Austria*
E-mail: thomas.rusch@wu.ac.at
[2]*Universität Innsbruck, Innsbruck, Austria*
E-mail: Achim.Zeileis@R-Project.org

1 Introduction

We thank Wei-Yin Loh for this review paper. He provides a much-needed guide to tree methods currently available as well as the main ideas behind them, indicative of his experience with and knowledge of this topic. His contribution proves to be very valuable in bringing structure into the vast interdisciplinary field of tree algorithms: We found 83 different tree induction algorithms for different response types listed in his paper, and, along the lines of Loh's disclaimer, this is not even an exhaustive list.

The availability of so many different algorithms for fitting tree-structured models directly relates to the main point of our discussion: The tree literature is highly fragmented. Loh hints at that issue already on the first page, and we gladly take it up for discussion: There are so many recursive partitioning algorithms in the literature that it is nowadays very hard to see the wood for the trees.

In the remainder of our discussion paper, we identify causes for and consequences of this fragmentation, discuss what we perceive to be advantages and disadvantages of the current state of the tree algorithm literature and offer suggestions that might improve the situation in the years ahead by retaining advantages and overcoming disadvantages.

2 The Fragmentation of Tree Algorithms

Currently, there is an abundance of different tree algorithms coming from different communities including statistics, machine learning and other fields. We believe that this fragmentation emerged from various causes and has a number of implications for the development and application of tree models. Some of them are in our opinion good, some are not so good and some are rather unfortunate.

2.1 The Good

The area of tree algorithms is a popular and fruitful field of research in statistics, computer science and beyond. This leads to many people with different backgrounds contributing to the application and development of tree algorithms for various tasks. May this be to derive a set of if-then rules to make decisions, to analyse a large number of data relatively fast, to segment data, to detect or select important variables and interactions or to simply have an interpretable, visualisable, data-driven prediction machine with good performance, which is flexible and can be adapted easily to the problem at hand. An important contributing factor to their popularity is that recursive partitioning algorithms are easily adapted to different situations, as their core principles are easy to understand and intuitive. Most tree algorithms comprise a couple of similar steps, the difference between them entering at some point in the induction stage, where during development a concrete choice must be made – usually related to the loss function or measure of node impurity, split variable selection, split point selection or pruning. Thus by, for example, changing the loss function used to measure node impurity, a new tailored algorithm can be easily invented for a given problem. Owing to this, we now have tree algorithms for many types of problems and variables that we might encounter, say, for online or dynamic data, longitudinal data, big sample sizes, substantive data models, various error structures and so forth. Looking at it this way, the fragmentation reflects in part the diversity of the problems encountered by the community of scholars working in this field, as well as the many ideas they have and different applications they face. Given that there is no free lunch (Wolpert, 1996), a rich diversity in algorithmic solutions is to be welcomed as no single solution will always lead to the best results. Thus, tree models were and are an active field of research and hopefully will remain so in the future.

2.2 The Bad

This abundance of tree algorithms also has its dark side. For one, not only is it hard to keep up to date with various developments, but it is even harder to *choose* the 'right' algorithm for a given problem. Take the case of regression trees for explaining and predicting a metric outcome and assume that there is additive Gaussian error. Which algorithm to take? There are, among others, AID, CART, CTree, C4.5, GUIDE and M5 – all having different properties in different settings. There is a lack of guidance as to which algorithm to select. One might narrow the possibilities down by looking at certain additional, desirable properties like unbiasedness in split variable selection but that still leaves one with a number of possibilities to consider. People looking to solve their problems with tree algorithms might easily be intimidated by the large number of possibilities and if different tree algorithms give different answers. Perhaps this contributed to many people inventing a new algorithm for their specific problem rather than work through different properties of existing algorithms and benchmarking them against each other on their data, which in turn perpetuates the fragmentation problem.

The fact that tree methods can be easily adapted to new situations can backfire. First, this sometimes leads to new tree algorithms or changes to old ones that appear *ad hoc*, which was already noted by Murthy (1998). While experimentation is a necessary part of algorithm development, we should nevertheless take care to propose and use well-motivated, well-founded, methodologically sound procedures in the end. Second, modifications or improvements of old algorithms are often considered to be entirely new algorithms. They may be named differently

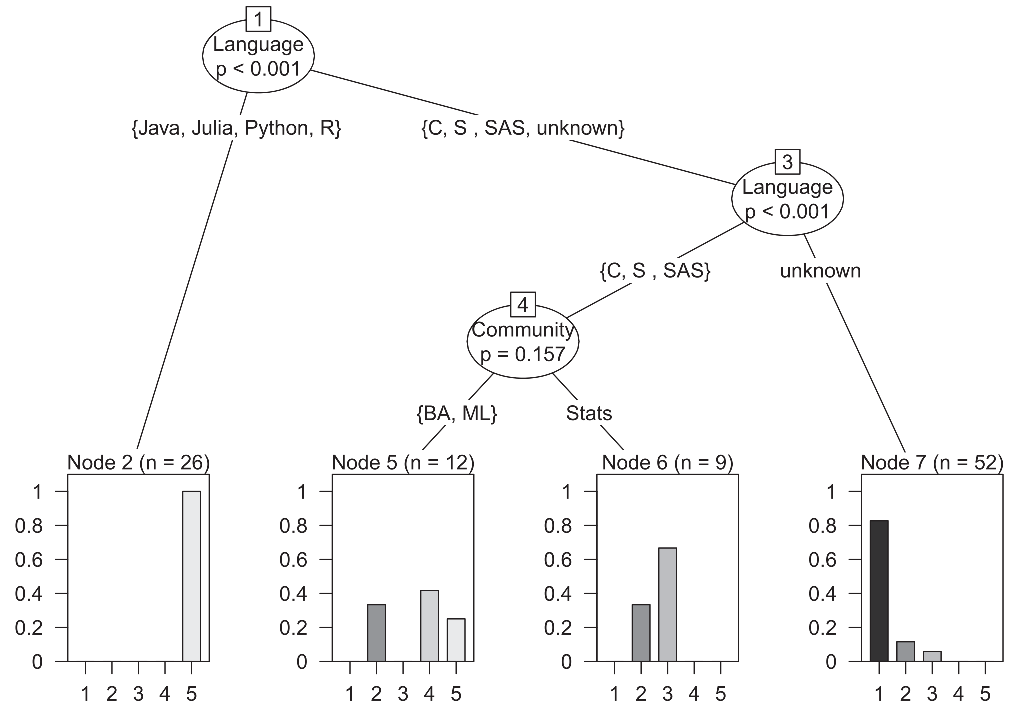

Figure 1. *A classification tree of tree algorithms fitted with CTree (Hothorn, Hornik, Zeileis, 2006). The target variable is the group of implementation as defined in Section 2.3. Predictor variables were the publication 'Year', the programming 'Language', whether it allows fitting of a classification ('CT'), regression ('RT'), or model tree ('MT'), author names ('FirstAuthor', 'SecondAuthor', 'LastAuthor'), 'Community' the algorithm is aimed at (machine learning vs. business analytics vs. statistics) and 'Journal' venue.*

and are often only viewed 'as a whole' rather than emphasising which steps in the tree induction are similar and which are different (e.g. the loss function might change, but the split variable selection is the same or vice versa). Hence, fragmentation is increased more than necessary, and common properties are obscured. This seems to tie in with a third bad effect: Many authors who propose or apply tree algorithms either are not aware of – or choose to ignore – similar work in that area. It happens that even recent papers do not refer to work carried out from 2000 onwards, therefore ignoring more than a decade of active development that may be highly relevant. On the one hand, this leads to reinventing the wheel, loss of time and resources and again more fragmentation of the literature. On the other hand, this also makes it hard for new algorithms to be noticed in an evergrowing, dense wood – an unfortunate situation for both developers and users alike. Perhaps all these points explain why, while there has been numerous new developments and improvements in tree modelling since the seminal work on C4.5 and CART in the 1980s, both remain the most popular tree algorithms for classification and regression (Wu *et al.*, 2008).

2.3 The Ugly

As algorithmic models, classification and regression trees are very closely tied to their respective implementations. With few exceptions, tree models usually *are* their specific implementation. Hence, they can be characterised by the specific combination of the generic computational steps that are adopted and by the way they are turned into a specific implementation – with both aspects being highly intertwined. While this algorithm–implementation dualism is rather natural for algorithmic models, there is also an ugly side to it: the potential lack of free (or any) implementations of new algorithms. This potential lack of access to the core of the actual tree model makes understanding, using, assessing and extending it much more difficult.

To illustrate this empirically, we consider 99 algorithms – including the 83 mentioned by Loh plus 16 very recent or less known ones. For 43 of these, we were not able to find an implementation[1] (see also Figure 2). More specifically, we placed all 99 algorithms into one of five broad classes pertaining to the availability of (free) software:

1. *Algorithms without an existing implementation:* These are algorithms for which a theoretical description was published, but no implementation seems to exist. Often, these are old algorithms or algorithms that were developed for a specific problem. Examples are AID or SUPPORT.
2. *Algorithms with a closed-source, for-profit implementation:* These are implementations of particular algorithms that are sold by a company. The code and specific implementation are kept a proprietary secret. Typical examples are M5, CHAID or CART.
3. *Algorithms with a closed-source, free-of-charge implementation:* These are implementations of particular algorithms that can be obtained free of charge, usually in an executable binary format. The code and specific implementation, however, is still kept a proprietary secret. Examples include GUIDE, CTMBR and HTL.
4. *Algorithms with an open-source, free-of-charge implementation:* These are implementations of particular algorithms that can be obtained free of charge and whose source code is open. However, these implementations either restrict or do not explicitly allow copying, adaptation and distribution of the source code. Examples currently include SECRET and C4.5.
5. *Algorithms with a free and open-source implementation:* These algorithms have implementations that follow the ideas of free software (FLOSS; free, libre open-source software, see Free Software Foundation, Inc., 2013). They are open source and give the user extensive rights with respect to copying, modification and distribution. Examples are most algorithms developed for FLOSS software packages like `Weka` or R, including re-implementations of closed-source algorithms, e.g. RPart, M5′, LMT and CTree and also C5.

We find that most algorithms belong to group 1 (43), followed by group 5 (29). The group of open-source algorithms/implementations (classes 4 and 5) only comprises 34% of all algorithms. To take a closer look at how this availability of (free) software depends on other characteristics of the algorithms; we naturally employ a classification tree (Figure 2, built by CTree). We see that an implementation in R, Python, Java (primarily in the packages `Weka`, `KNIME`, `RapidMiner`) and Julia is predictive of belonging to group 5, whereas other languages are either predictive for group 1 (if we do not know the language), of groups 2 and 3 respectively if suggested to the statistics community, and for groups 4 and 5 for implementations directed at the machine learning and business analytics communities. At any rate, it shows that unfortunately FLOSS is far from being the standard for tree modelling software.

As trees are inherently algorithmic, we view the implementation as an integral part of each algorithm. In our opinion, restrictions to viewing, modifying and sharing tree implementations are one of the main reasons for the bad sides of fragmentation discussed earlier. Depending on which group an algorithm belongs to this has different implications. For example, for algorithms belonging to groups 1 through 4, this leads to the need for authors proposing improvements to existing algorithms to implement the improved algorithm from scratch. Often, this adapted implementation is then again not FLOSS, and the problem perpetuates. Not being able to rely on existing code also applies to using an algorithm on different platforms. Another example is that the restriction of distribution and modification effectively prohibits to change the specific implementation (say, with regard to adapt it to parallel computing) or to improve it (say, with regard to speed). This also restricts the possibilities of combining the algorithms with other methods to form a pipeline of methods and distribute the bundle. For algorithms from groups 1 through 3, a further consequence is that specific steps that may not be well documented can be hard to reproduce [as was the case with M5 (Quinlan, 1993), which prompted M5′ (Wang & Witten, 1997) as a 'a rational reconstruction'].

We strongly believe that these and other implications of a lack of free implementations led to many synergy potentials having been lost over the years, partly because conceptual similarities and differences of various tree algorithms were not obvious enough and partly because the lack of reusable computational tools slowed down the pace development. Furthermore, there seems to be some confusion among practitioners as to which algorithms perform well (or even best) for their particular problems which often leads to suboptimal algorithms being used. This view appears to be shared increasingly by other researchers (e.g. Vukićević *et al.*, 2012).

3 A Possible Remedy

We have a suggestion as to what we think will improve or even solve the problems discussed in Sections 2.2 and 2.3 while retaining the advantages mentioned in Section 2.1: (Academic) publications of tree algorithms should be accompanied by free implementations (in the FLOSS sense). This means opening the source code of past and future implementations, giving users permission to modify, adapt and distribute it with an appropriate free software license and making the code/implementations easily publicly available, preferably with a low adoption threshold (e.g. in a popular language such as Python, C, Java or R).

Specifically, we think that the following six steps should be undertaken to reduce the bad and ugly aspects to a minimum while retaining the good:

- Every newly suggested algorithm or larger improvement should come with a FLOSS implementation.
- The source code of currently existing implementations should be opened, and they should be licensed with a FLOSS license.
- For algorithms for which there is no up-to-date or even existing software, any trademark should be relinquished, and the original source code should be made freely available or re-implemented as FLOSS (as has already happened with, e.g. J4.8 or RPart).
- All implementations should be made available on a public repository or archive, the author's homepages or as freely accessible supplementary material to articles.
- When a software is used, extended, modified and so on, the software (and not only the underlying algorithm) should be referred to and cited.
- When reviewing or editing papers or algorithms, we should point out the above and demand this as a new standard.

FLOSS software licenses should be used so that the copyright holders grant the rights to inspect, modify and (re)distribute the software. Most FLOSS licenses preserve the original copyright in such modified/extended versions and in an academic context citation of the software (and not only the underlying algorithm) is appropriate.

Then, improvements to algorithms do not need to be their own algorithm but can be suggested or submitted as patches or adaptations. Code building blocks (e.g. for split point selection, or predictions or tree visualisations) can be reused and recombined or be made computationally more efficient. Assessment and comparison of algorithms are facilitated, both for evaluating newly suggested methods and for choosing a particular model in practice. Hence, not only users of FLOSS implementations will profit but also the authors because their work is easier to understand, use and ultimately cite.

Finally, the aforementioned steps can also reduce the fragmentation and possibly achieve a certain degree of standardisation through developing and reusing computational 'tree toolkits'. Some effort in this vein has been made already. For example, there are two R packages providing standardised frameworks: `partykit` (Hothorn & Zeileis, 2014) for representing, summarising and plotting of various tree models from different free software sources, and `caret` (Kuhn, 2008) for training, tuning and benchmarking of various tree algorithms (among many other methods). Other efforts of providing such standardisation exist as well (Vukićević *et al.*, 2012).

4 Conclusion

In this discussion, we follow up on Loh's review paper and take a closer look at the fragmented field of tree models. While indicative of a creative, active, and diverse community of researchers, this fragmentation also leads to undesirable side effects making it hard to understand, assess, use and compare tree algorithms. Hence, a common language for describing tree models, both conceptually and (perhaps more importantly) computationally, is crucial to reduce the fragmentation. In particular, making FLOSS should be an integral part of the communication about classification and regression trees. We argue that this can alleviate many of the problems caused by or following from the fragmentation while retaining the good that comes from having a bright and vibrant community.

Especially in light of open research areas that Loh mentions in his conclusions, FLOSS implementations are an effective means for reducing fragmentation in the future and tackling open hard problems in tree algorithm research faster than it was possible before. The good news is that the tree community has already started to move into this direction, and an increase in FLOSS implementations can already be observed. With free platforms for statistical computing such as R or Python as well as initiatives like the 'Foundation for Open Access Statistics' (FOAS, http://www.foastat.org/), the conditions are now better than ever before. We should use this momentum. Rather than not seeing the wood for the trees, the whole community can grow a healthy, open and light forest of trees within which we can all walk with intimate familiarity.

Notes

[1]We searched with Google for all permutations of author names and algorithm names combined with the words "software" and "implementation", as well as on the main author's homepages.

Acknowledgements

We want to thank Wei-Yin Loh for his stimulating article and Probal Chaudhuri for inviting us to join this discussion.

References

Free Software Foundation, Inc. (2013). What is free software? Version 2013-06-18 05:16:52. http://www.fsf.org/about/what-is-free-software.

Hothorn, T. & Zeileis, A. (2014). `partykit`: A modular toolkit for recursive partytioning in R. *Working Paper 2014-10*, Working Papers in Economics and Statistics, Research Platform Empirical and Experimental Economics, Universität Innsbruck. http://econpapers.repec.org/paper/innwpaper/2014-10.htm.

Kuhn, M. (2008). Building predictive models in R using the `caret` package. *J. Stat. Software*, **28**(5), 1–26.
Vukićević, M., Jovanović, M., Delibašić, B., Išljamović, S. & Suknović, M. (2012). Reusable component-based architecture for decision tree algorithm design. *Int. J. Artif. Intell. Tools*, **21**(05).
Wolpert, D.H. (1996). The lack of a priori distinctions between learning algorithms. *Neural Comput.*, **8**(7), 1341–1390.
Wu, X., Kumar, V., Quinlan, R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G., Ng, A., Liu, B., Philip, Y., Zhou, Z.-H., Steinbach, M., Hand, D. & Steinberg, D. (2008). Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, **14**(1), 1–37.

[*Received April 2014, accepted April 2014*]

International Statistical Review (2014), 82, 3, 367–370 doi:10.1111/insr.12057

Rejoinder

Wei-Yin Loh

Department of Statistics, University of Wisconsin, Madison, WI 53706, USA
E-mail: loh@stat.wisc.edu

I thank the discussants for their thoughtful comments, which helped to fill in some gaps and expand the scope of the review. I will address each one below.

Carolin Strobl wonders when it is helpful to create separate nodes for missing values. CHAID seems to be the only algorithm to do this, but it has a procedure to merge some of the nodes before they are split further. Its effectiveness has not been studied. GUIDE treats missing values in a categorical variable as a separate category but does not assign them to a separate node. If there are missing values at a split on an ordered variable, GUIDE sends them to the same left or right child node, depending on which split yields the greater decrease in node impurity. Ding & Simonoff (2010) studied a simpler version of this technique, where missing ordered values are mapped to infinity and hence are always sent to the right child node. Using only binary-valued variables with training and test sets having missing values where missingness in a predictor variable depends on the values of the response variable (MAR), they found that this technique is better than case deletion, variable deletion, grand mean/mode imputation, surrogate splits (as used in RPART) and fractional weights (as used in C4.5). Case deletion and grand mean/mode imputation tend to be worst, a finding supported by Twala (2009), who considered missingness dependent on other predictor variables (MAR) and missingness due to truncation (MNAR) but not missingness dependent on the response variable. He found that the best method was an ensemble of classification trees constructed by multiple imputation of the missing values with the expectation–maximization algorithm (Dempster *et al.*, 1977; Rubin, 1987). It is not clear whether this is either due to multiple imputation or ensemble averaging. Note that because both studies employed C4.5 and RPART exclusively as the base classifiers, it is unknown if the conclusions extend to other methods. Further, some other missing value techniques, such as nodewise mean and mode imputation (FACT and QUEST) and alternative surrogate split methods (CRUISE), were not included.

Strobl wonders whether ignorance is the reason that biassed recursive partitioning methods continue to be used frequently. Many people still associate the term ‘classification and regression trees’ with CART and its software. Commercial software publishers perpetuate this misconception by largely basing their offerings on CART. The availability of RPART also