The *fertilized forests* Decision Forest Library

Christoph Lassner
University of Augsburg
Universitätsstr. 6a
86159 Augsburg, Germany
Christoph.Lassner@informatik.uni-augsburg.de

Rainer Lienhart
University of Augsburg
Universitätsstr. 6a
86159 Augsburg, Germany
Rainer.Lienhart@informatik.uni-augsburg.de

ABSTRACT

Since the introduction of Random Forests in the 80's they have been a frequently used statistical tool for a variety of machine learning tasks. Many different training algorithms and model adaptions demonstrate the versatility of the forests. This variety resulted in a fragmentation of research and code, since each adaption requires its own algorithms and representations.

In 2011, Criminisi and Shotton developed a unifying Decision Forest model for many tasks. By identifying the reusable parts and specifying clear interfaces, we extend this approach to an object oriented representation and implementation. This has the great advantage that research on specific parts of the Decision Forest model can be done 'locally' by reusing well-tested and high-performance components.

Our *fertilized forests* library is open source and easy to extend. It provides components allowing for parallelization up to node optimization level to exploit modern many core architectures. Additionally, the library provides consistent and easy-to-maintain interfaces to C++, Python and Matlab and offers cross-platform and cross-interface persistence.

Categories and Subject Descriptors

G.4 [**Mathematics of Computing**]: Mathematical Software;
D.2.2 [**Software**]: Software engineering—*Design tools and techniques, Software libraries*;
I.2.6 [**Computing methodologies**]: Artificial intelligence—*Learning*

General Terms

Algorithms, Design, Experimentation

Keywords

Decision Forests; Object Oriented Implementation; Open Source; Parallel Implementation; Machine Learning

MM'15, October 26–30, 2015, Brisbane, Australia.

ACM 978-1-4503-3459-4/15/10...$15.00
DOI: http://dx.doi.org/10.1145/2733373.2807407.

1. INTRODUCTION

The basic idea of Decision Trees, hierarchical splitting of data with the aim of an increasingly informative grouping, is appealingly simple. This leads on the one hand to many possible refinements and specializations. On the other hand, many researchers have created their own, hand-tailored implementation for their specific needs. This is especially true for the "classic" algorithms of the early days of Decision Forests[1], because each of the algorithms was created for a specific data type and purpose. But if not implemented with thorough testing and proper thoughts about efficiency, even a simple algorithm may be erroneous or inefficient.

However, in their work [6], Criminisi and Shotton developed a unifying Decision Forest Model suitable for many tasks and data types, making many of these dedicated algorithms obsolete. Furthermore, this new model provides a solid foundation for an object oriented framework with high reusability of components.

With the recent developments in Decision Forest theory and in computing technology, the benefits of such an object oriented framework compared to hand-tailored problem oriented implementations are becoming increasingly important. Computing platforms with significantly more than 40 CPUs render tree-level parallelism insufficient to exploit their power, when memory intensive models such as Hough Forests [10] with fewer trees than CPUs are trained. On the theoretical side, recent algorithmic extensions such as Alternating Decision Forests [17] or, as mentioned before, Hough Forests, merely use and recombine existing algorithmic parts. We tackle this problem by providing an open source library for Decision Forest training.

With clear interfaces and strong locality, the library classes remain easy to understand and easy to alter and extend. While not going as far as to use SSE optimizations, the library is written in highly templated C++ and optimized thoroughly without destroying code readability. At the same time, OpenMP is used to create nested parallelization on tree and node optimization level, which allows to make use of many cores even when training fewer trees than cores.

The open source library is available under a permissive license with extensive documentation and examples. It provides clean and consistent interfaces to C++, Python and Matlab. With the idea to be easily extendable, it features an interface generator that automatically keeps the interfaces up-to-date with changes.

[1]Similar to [6], we use the phrase "Decision Forest" as a synonym for "Random Forest". The new term emphasizes the reference to the new, but equivalent, theoretical framework.

Name	License	Core lang.	Available bindings	OS ind.	Parallelization	Type aware	Extend.	Setups
fertilized forests	BSD	C++	Python, Matlab	✓	Trees & Nodes	✓	✓	2295
scikit-learn [15]	BSD	Cython	Python	✓	Trees			32
OpenCV [3]	BSD	C++	Python	✓		✓		9
Sherwood [5]	MSR-LA	C++	C#		Nodes		✓	5
ALGLIB [2]	GPL 2+	C	C#, VB.NET, Python	✓				1
WEKA [11]	GPL 2	Java	Python	✓				4

Table 1: Overview over Decision Forest libraries. The column "Type aware" refers to whether a library can treat input data in its native data type. The column "Extend." (extendable) refers to whether it is possible without much overhead to extend the implemented algorithms of the library. "Setups" summarizes how many different training setups can be realized with the library.

While the library is significantly more flexible than others with these features, it stays competitive in terms of speed and learning performance (an evaluation is given in Section 4). With a strong focus on computer vision applications, it features last years winner of the ACM Multimedia Open-Source Software Competition, CAFFE [13], as part of the library that can be used for feature extraction. For making this possible, we contributed a platform independent build system to the project, making CAFFE available on Windows, and integrated parts of it into our own library.

2. FEATURES

The library tackles many of the software engineering challenges in its domain by using an **object oriented model** of Decision Forests. This has several advantages:

Local parameterization Instead of having few, overparameterized factory functions, parameters are local to the objects they are related to. This means, that new parameterized objects can easily be created, without having to bloat parameter lists with default values.

Recombination It is natural to recombine objects to create new, meaningful forest types. E.g., Hough Forests can be created by reusing regression and classification threshold optimizers and just adding two new objects.

Code encapsulation Semantically close code is automatically grouped together. This makes understanding and extending the code of a class easier, since only the class's interface must be understood to enhance it.

Inheritance Minor modifications to existing algorithms can be created by inheriting from their defining classes. Not the entire functionality must be rewritten, only the most relevant parts.

The library currently contains classes for all classification and regression concepts described in [6], but has been extended with a state-of-the art implementation of Hough Forests, two variants of boosting, several strategies for split optimization and many entropy functions, including the recently described induced entropies [14].

Deterministic, nested parallelism To be able to fully exploit modern many core architectures, the library supports deterministic, nested parallelism down to node optimization level. While coarse-grained tree-level parallelism is insufficient on modern machines, an additional parallelization step during node optimization offers a second level of fine-grained parallelism. When carefully implemented, race conditions can be avoided while maintaining good parallelization behavior. This guarantees the same, deterministic results independent of the number of used threads.

Templated classes The library has been created with multimedia and computer vision applications in mind. In the age of big data, the sampled signals are getting so large that inflating the data only to adjust its data type becomes permissively wasteful, e.g., increasing the amount of image data by four only to convert its data type from `unsigned byte` to `float`. To avoid this, all library classes are templated with the relevant types of the data they process. To take the hassle away of re-typing the template parameters frequently and to reflect this concept in the non-templated languages MATLAB and Python, a factory object is used that receives the template parameters once and then creates all library objects correctly templated for the user's convenience.

OS independence Being compatible to the gcc, Intel and Microsoft Visual C++ compilers, the library can be used on all major platforms. The build system is realized with SCons[2] and can hence be used on all platforms where a Python distribution is available.

Interfaces The library offers completely consistent interfaces to C++, MATLAB and Python. Convenient updating and complete consistency is ensured by using an easy to use interface generator written in Python that comes with the library.

OS and interface independent persistence By using Boost serialization text archives for persistence across all interfaces, the library objects can be serialized and deserialized across all possible platforms and interfaces. This allows training trees on a large HPC Linux cluster and then performing analyses, e.g., in Python on Windows.

2.1 Comparison with existing libraries

There are countless libraries for Decision Forest training available (an overview over the most relevant ones is given in Table 1). However, most of them are outdated concerning the supported algorithms and none of them offers a comparably versatile combination of open source code, OS availability, cross platform serialization and number and completeness of interfaces.

Interestingly, there is no other Decision Forest library with support for MATLAB, but its built-in Decision Forest implementation is outdated and slow (more than two orders of magnitude slower than our implementation). Thus, our library could be a good alternative for computer vision researchers in this environment. It is, however, necessary to note, that while a lot of care has been taken to copy as few data as possible in the general library design, a copy of parameters from and to MATLAB is necessary due to its column major storage order concept.

[2] http://www.scons.org

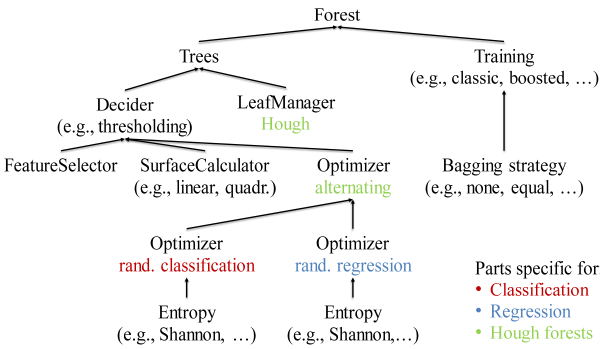

Figure 1: Hough Forest object structure. Arrows represent 'is-using' relations.

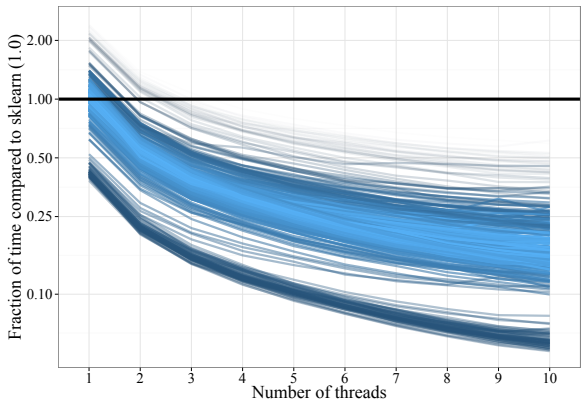

Figure 2: Runtime comparison with scikit-learn.

While most of the other software listed as representative selection in Table 1 is competitive in the historically relevant areas such as single-threaded runtime and platform availability, few are data type aware, and no other supports nested parallelism and is nearly as versatile and easy to extend. Our aim is to excel in this area by creating a library that is well suited for the dynamic process of research.

3. LIBRARY DESIGN

The library's design is general enough to enable efficient recombination of components and to enable benefits by inheritance, but still groups related code together closely. Figure 1 shows an overview of the objects necessary to represent a Hough Forest. This specific scenario has been selected, because it illustrates many of the benefits well.

By exchanging the *LeafManager*, which controls the information stored at leafs, as well as the *Optimizer*, which optimizes the thresholds according to the data annotations, it is possible already to change the objective function and the resulting model. This can, e.g., be used to create a classification or regression forest. By adding two new classes, a *HoughLeafManager* and an *AlternatingOptimizer*, and by reusing existing classes, the completely different concept of a Hough Forest can be defined.

Parallelization is implemented in the *Training* and *Decider* objects. All objects being located lower in the hierarchy automatically benefit of this parallelization without explicitly implementing it.

4. EVALUATION

To give the reader an impression of the performance of the library, we did some experiments with the competing library 'scikit-learn'. We selected that library as competitor, since (a) we appreciate its impact on the machine learning community and its high popularity, (b) used it in some ways as inspiration for our project, and (c) the parameterization of this library is similar enough to ours to allow a fair comparison.

Runtime It is not straightforward to set up an experiment resulting in a reliable statement about library speeds: specifics of the data can bias results in favor of each candidate, or a specific parameter setting can as well strongly influence the results removing its generality.

Therefore, we came up with the idea to use 100 runs with randomly (but equivalently) sampled parameter settings on each of three large scale, computer vision datasets (chars74k [9], MNIST [1] and USPS [12]). This results in a large set of overall runs under diverse conditions, and we hope that it provides a better impression of performance.

While parallelization over trees trivially has a good parallelization behavior, this is not necessarily true for deterministic parallelization over the node optimization. To visualize the behavior of our library in this specific case, we varied the number of threads for node optimization.

The results are visualized in Figure 2. For each run, the corresponding runtime of scikit-learn was used as normalization (hence the straight line at height 1.0). All other lines show the traces of our library. The brightness of the color is higher for traces closer to the mean trace.

The first important results are the performance values for one thread. There are a few runs for certain parameter settings where the runtime of our library goes up to more than twice of the runtime of scikit-learn. The mean, however, is at about 1.0, with the start of a solid trace at little more than a third of the runtime of scikit-learn (the y-axis is logarithmic). For an increasing number of threads, the library shows good parallelization behavior peaking out in the best cases at substantially smaller values than 0.1, even though only 10 threads have been used.

We identified the ability to parallelize beyond tree level to be of critical importance. Dantone et al. report a training time of 3 hours on a 700 CPU cluster for their Hough Forest human pose estimation implementation [8]. Since our university does not have a comparably sized cluster, we exploited our local infrastructure using our library. It comprises of only 64 CPUs, of which two times twenty are part of server systems. Even though Dantone's original model only uses 10 trees, we could fully exploit our computing power and reduced the training time by more than an order of magnitude to 2.5 hours on our infrastructure.

Classification performance In Figure 3, we provide a comparison of F1-scores with scikit-learn on five large scale computer vision classification datasets (parameters were again randomly sampled 100 times). The x-axis shows the various datasets, where the left of each pair of columns shows the fertilized forests performance, and the right the scikit-learn performance. The datasets, from left to right, are chars74k [9], g50c [4], letter [1], MNIST [1] and USPS [12].

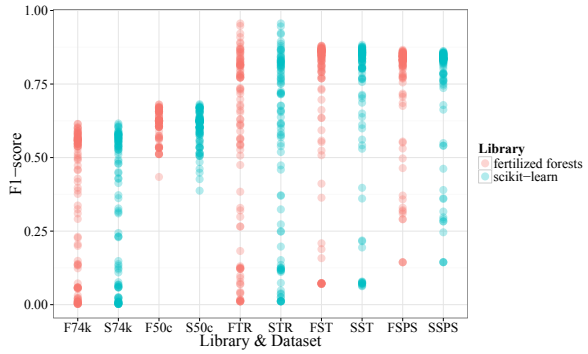

Figure 3: F1-score comparison with scikit-learn.

Figure 4: Human pose estimation with Decision Forests using the fertilized forests library (c.t. [14]). The samples are part of the FashionPose dataset [7].

The y-axis shows the F1-score[3], since some of the datasets have a high number of classes and are not perfectly balanced. The authors of scikit-learn have implemented various heuristics in addition to the traditional Decision Forest algorithm. We went through the source code and added them to our library, as well as one new heuristic. This gives the fertilized forests sometimes a slight edge over the scikit-learn implementation (visible, e.g., for the g50c dataset).

5. APPLICATIONS

The library is well-tested and has been applied in many scenarios. It is in use for research at the Multimedia Computing and Computer Vision Lab and the Institute for Software Engineering at the University of Augsburg, the Information Processing Lab at the University of Washington and the Max Planck Institute for Intelligent Systems in Tübingen.

The first two research projects were a work on uncertainty sampling for model abstraction [16] and on induced entropies for Decision Forests [14]. Whereas the first one mainly used the probabilistic regression features of the library, the second used nearly all features of the library and shows how the simplicity of replacing algorithmic building blocks of Decision Forests can support research in this field. Figure 4 shows some samples of the implemented human pose estimation algorithm and improvements (marked with a yellow dot in the bottom row) while using a different entropy measure.

[3] $F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$.

6. SOURCE & ACKNOWLEDGEMENTS

The main source of information around the library, including API documentation, examples, FAQ page and news section, is the website http://www.fertilized-forests.org. The source code is available at https://github.com/ChrisIS/fertilized-forests.

We thank the students Moritz Einfalt, Philipp Harzig and Christian Diller for their contributions. Furthermore, we thank Matthias Dantone and Jürgen Gall for the permission to use their Hough Forest feature extraction code.

7. REFERENCES

[1] K. Bache and M. Lichman. UCI machine learning repository. http://archive.ics.uci.edu/ml.

[2] S. Bochkanov. Alglib. http://www.alglib.net.

[3] G. Bradski. Opencv. http://www.opencv.org.

[4] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, 2006.

[5] A. Criminisi and J. Shotton, editors. *Decision Forests for Computer Vision and Medical Image Analysis*. Springer-Verlag London, 2013.

[6] A. Criminisi, J. Shotton, and E. Konukoglu. Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning. Technical Report MSR-TR-2011-114, MS Res., 2011.

[7] M. Dantone, J. Gall, C. Leistner, and L. V. Gool. Human Pose Estimation using Body Parts Dependent Joint Regressors. In *Proc. of the IEEE CVPR*, 2013.

[8] M. Dantone, J. Gall, C. Leistner, and L. V. Gool. Body Parts Dependent Joint Regressors for Human Pose Estimation in Still Images. *IEEE TPAMI*, 36(11):2131–2143, November 2014.

[9] T. E. de Campos, B. R. Babu, and M. Varma. Character recognition in natural images. In *Proc. of the VISAPP*, February 2009.

[10] J. Gall, A. Yao, N. Razavi, L. V. Gool, and V. Lempitsky. Hough Forests for Object Detection, Tracking and Action Recognition. *IEEE TPAMI*, 33(11):2188 – 2202, 2011.

[11] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Expl.*, 11, 2009.

[12] J. Hull. A database for handwritten text recognition research. *IEEE TPAMI*, 16(5):550–554, May 1994.

[13] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *ACM MM Open Source Comp.*, 2014.

[14] C. Lassner and R. Lienhart. Norm-induced entropies for decision forests. In *Proc.of the IEEE WACV*, 2015.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *JMLR*, 12:2825–2830, 2011.

[16] A. Schiendorfer, C. Lassner, G. Anders, W. Reif, and R. Lienhart. Active learning for abstract models of collectives. In *Proc. of the SAOS workshop*, 2015.

[17] S. Schulter, P. Wohlhart, C. Leistner, A. Saffari, P. M. Roth, and H. Bischof. Alternating Decision Forests. In *Proc. of the IEEE CVPR*, 2013.