Automatic Gradient Boosting

Janek Thomas JANEK.THOMAS@STAT.UNI-MUENCHEN.DE
Stefan Coors STEFAN.COORS@CAMPUS.LMU.DE
Bernd Bischl BERND.BISCHL@STAT.UNI-MUENCHEN.DE
Department of Statistics, LMU, Ludwigstrasse 33, D80539 Munich

Abstract

Automatic machine learning performs predictive modeling with high performing machine learning tools without human interference. This is achieved by making machine learning applications parameter-free, i.e. only a dataset is provided while the complete model selection and model building process is handled internally through (often meta) optimization. Projects like Auto-WEKA and auto-sklearn aim to solve the Combined Algorithm Selection and Hyperparameter optimization (CASH) problem resulting in huge configuration spaces. However, for most real-world applications, the optimization over only a few different key learning algorithms can not only be sufficient, but also potentially beneficial. The latter becomes apparent when one considers that models have to be validated, explained, deployed and maintained. Here, less complex model are often preferred, for validation or efficiency reasons, or even a strict requirement. Automatic gradient boosting simplifies this idea one step further, using only gradient boosting as a single learning algorithm in combination with model-based hyperparameter tuning, threshold optimization and encoding of categorical features. We introduce this general framework as well as a concrete implementation called autoxgboost. It is compared to current AutoML projects on 16 datasets and despite its simplicity is able to achieve comparable results on about half of the datasets as well as performing best on two.

Keywords: AutoML, Gradient Boosting, Bayesian Optimization, Machine Learning

1. Introduction

Machine Learning, *Predictive Modeling* and *Artificial Intelligence* are ongoing topics in research as well as in industrial applications. While data are gathered everywhere nowadays, many potential insights are often not fully achieved since data science and ML experts are still a rare commodity. While many stages of a data analysis project still need to be done manually by human data scientists, model search and optimization can be done automatically. Automatic machine learning (AutoML) simplifies the workload by making decisions for common predictive modeling tasks like regression or classification. We distinguish between *single-learner* AutoML methods which aim to make single algorithms parameter-free and more general approaches, which combine several learning algorithms into one optimization problem. These *multi-learner* methods solve the *Combined Algorithm Selection and Hyperparameter optimization (CASH)* problem (Thornton et al. (2013)). Modern approaches that include pre- and postpressing methods are referred to as *machine learning pipeline configuration*.

There is a growing number of open source approaches for automating machine learning available for non-professionals. As one of the first frameworks, Auto-WEKA (Thornton et al. (2013)) introduced a system for automatically choosing from a broad variety of learning algorithms implemented in the open source software *WEKA* (Hall et al. (2009)). Hereby, Auto-WEKA simultaneously tunes hyperparameters over all learning algorithms model using the Bayesian optimization framework SMAC (Hutter et al. (2011)). Similar to Auto-WEKA is auto-sklearn (Feurer et al. (2015)), which is based on the scikit-learn toolkit for python and includes all of its learners as well as available preprocessing operations. It stacks multiple models to achieve high predictive performance. Another python-based AutoML tool is called *Tree-based Pipeline Optimization Tool (TPOT)* by Olson et al. (2016) and uses genetic programming instead of Bayesian optimization to tune over a similar space as auto-sklearn.

Only few *single-learner* AutoML methods exist. A lot of services, for example Google's *Cloud AutoML*, focus on specialized application domains like image recognition using deep neural networks. For general machine learning tasks, Probst et al. (2018) introduced the tuneRanger software, which automatically tunes a random forest. Another algorithm approach is called *Parameter-free STOchastic Learning (PiSTOL)* (Orabona (2014)) and directly tries to optimize the generalization performance of a learning algorithm, in a stochastic approximation way.

Our proposed approach reduces the AutoML framework to the construction of an optimal gradient boosting model (Friedman (2001)), which is a strong predictive algorithm, as long as its hyperparameters are adequately tuned. Besides tuning the hyperparameters via Bayesian optimization, categorical feature transformation is performed as a preprocessing step. Moreover, for classification tasks, thresholds are optimized repeatedly. By focusing on a single learning algorithm, hyperparameters can be optimized much more thoroughly and the resulting model can be analyzed and deployed much easier.

2. Method

This section introduces the structure of the automatic gradient boosting framework. The general workflow of the approach can be seen in Figure 1. Automatic gradient boosting uses gradient boosting with trees (GBT) as its only learning algorithm. GBT is popular due to its strong predictive performance and robustness. A large number of machine learning competitions were won by these algorithms, see Chen and Guestrin (2016) for an overview. Furthermore it possesses multiple highly desirable properties for an AutoML system: It is insensitive to outliers, as the trees used in gradient boosting are invariant to monotone transformations of the data, which makes scaling the data obsolete; GBT implementations are usually able to handle missing values in the data directly by learning default split directions for missing values (Chen and Guestrin (2016)); In addition, they are capable of handling high dimensional feature spaces, as features are evaluated separately for each split in a tree, which can be parallelized and does not result in a harder optimization problem for more features. One last important aspect to consider is that boosting can be easily adapted to tasks like ranking (Li et al. (2008)) or survival analysis (Chen et al. (2013)). Modern GBT frameworks like xgboost (Chen and Guestrin (2016)) or lightgbm (Ke et al. (2017)) are highly configurable with a large number of hyperparameters for regularization and

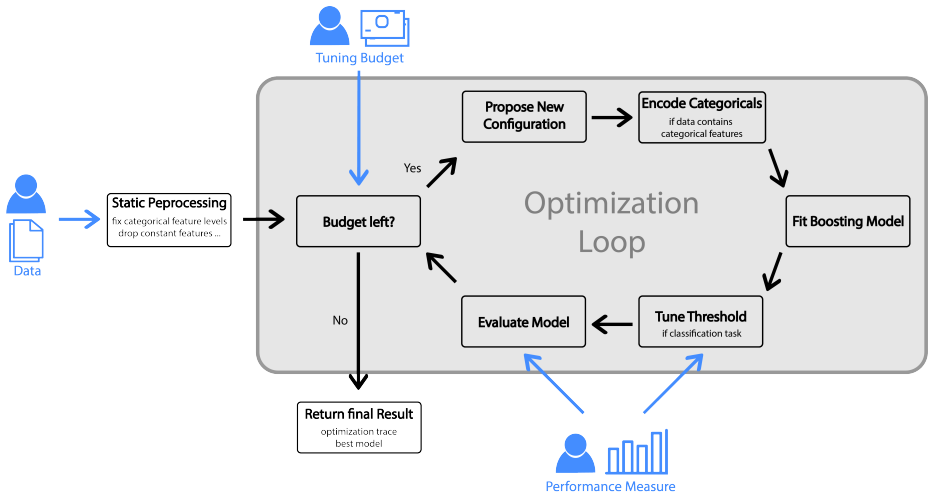

Figure 1: Workflow of the automatic gradient boosting approach. Blue lines indicate input by human.

optimization. With the recent addition of dropout boosting (Rashmi and Gilad-Bachrach (2015)) to these frameworks, it is possible to let the boosting algorithms behave similar, or even identical to random forests by setting hyperparameters accordingly.

The large number of hyperparameters makes tuning for GBT a necessity. Different methods like grid or random search can be used for simple hyperparameter optimization problems, but to achieve more efficient optimization, adaptive strategies should be employed. *Sequential model-based optimization (SMBO)*, also known as *Bayesian optimization* is one of the state-of-the-art adaptive hyperparameter optimization strategies (Snoek et al. (2012)). Depending on the difficulty of the hyperparameter space in terms of categorical and dependent hyperparameters, different surrogate models are used. In Table 1 two different possible hyperparameter spaces are proposed, a fully numeric one (denoted Simple=Y), which can be optimized with Gaussian process surrogate models, and a more complex one (denoted Simple=N), which can be optimized with a random forest surrogate. Arguably the most important hyperparameter in GBT is the number of boosting iterations, which is efficiently found by early-stopping, i.e., measuring the performance on validation data after each iteration. An advantage of combining early stopping with SMBO is that the validation error can be directly returned to the optimizer without the need of an additional holdout set or resampling. No necessity for internal resampling and the parallel implementation of GBT algorithms allow to mitigate the disadvantage of the sequential nature of SMBO without using parallel SMBO variants (see Bischl et al. (2014) for an overview), such that parallel system architectures can be fully utilized.

Most boosting implementations cannot natively handle categorical variables and it is necessary to transform such features. The simplest possibility is to encode these features into integers, with the drawback that the optimal order is unknown and a random one is used. The concept of dummy encoding is to create one separate feature for each level,

i.e., a feature x with levels a, b, c is dummy encoded into binary features x_a^*, x_b^* and x_c^*. No information is lost by this encoding but it can be infeasible for high cardinality features with a high number of unique feature levels. A third method of transforming categorical features is *impact encoding* (Micci-Barreca (2001)). Features are encoded by replacing categories with aggregated values of the target in the respective group, e.g., $\bar{y}|_{x=a}$ for regression or $P(y|x = a)$ for classification. We evaluate different combinations of these encodings, mainly based on a threshold, e.g., features with less than k levels are dummy encoded while integer or impact encoding is done for the remaining categorical features. It is also possible to tune this threshold k together with the GBT hyperparameters. For datasets with few categorical features the encoding can also be learned separately for each feature.

Depending on the overall performance metric that should be optimized, it can be difficult to find the best loss function for GBT since not every performance metric can be directly plugged in as loss functions. For binary- and multiclass classification it is often useful to optimize classification thresholds of each class directly with regard to the used performance metric. The threshold is optimized for each iteration of the model-based optimization on the validation set that was already used for early stopping. The resulting internal performance value is hence biased, but since the tuning error of the optimizer is biased anyways, we let this slide, especially since reducing training data size or extra resampling is much less efficient. This design decision should be investigated in more details in future studies, though.

Combining all of the above components, we achieve a fast, scalable and robust AutoML solution that can handle categorical parameters (even with many levels), outliers and missing data, while having a much smaller configuration space compared to existing solutions.

3. Implementation

This section introduces an implementation of the described automatic gradient boosting framework. In general, the introduced framework could use a large number of available boosting library (e.g., xgboost or lightgbm) as well as different SMBO libraries (e.g., SMAC, Spearmint or mlrMBO). We decided to implement it in R using xgboost as a GBT implementation, mlrMBO (Bischl et al. (2017)) for SMBO and mlr (Bischl et al. (2016)) as a general machine learning framework as well as for threshold optimization. For threshold optimization a multi-start linesearch is used for binary classification and for multiclass classification *Generalized Simulated Annealing (GSA)* (Tsallis and Stariolo (1996)) is applied. The software is available via Github[1] and is currently able to handle binary and multiclass classification as well as regression. It can be used in a standalone version or within the mlr framework as a learner. Other than the data itself no further information has to be passed to autoxgboost, but it may be useful to define the performance metric (otherwise a default will be used depending on the type of the data) and the maximum runtime. The result is a reusable machine learning pipeline based on the library mlrCPO[2] that can be deployed or saved for later use. Currently two different hyperparameter spaces are predefined (see Table 1) and the simpler one (where Simple=Y) is used by default.

1. https://github.com/ja-thomas/autoxgboost
2. https://github.com/mlr-org/mlrCPO

4. Benchmark

We compare the performance of autoxgboost to the AutoML solutions Auto-WEKA and auto-sklearn. In order to ensure comparability, we evaluate autoxgboost on a subset[3] of the datasets Auto-WEKA and auto-sklearn used in their respective publications. This includes identical training- and test-data splits and the same performance measure. The chosen datasets are very different regarding the number of numeric and factor features, as well as the number of target class levels and the train and test dataset sizes. Hence, the datasets chosen by Thornton et al. (2013) serve as an adequate heterogeneous base for an initial performance evaluation in different situations. Moreover, like in the paper of Thornton et al. (2013), 25 runs were performed.
The parameter settings of autoxgboost were mostly left at their default values discussed in the previous section. However, at most 160 tuning iterations were allowed with a maximum runtime of 10 hours. The benchmark was run on Intel Xeon E5-2697 v3 processors with 28 cores and 64gb RAM. The hyperparameter ranges corresponded to the ones from Table 1 (simple).

For evaluation, 100000 bootstrap samples of size 4 were drawn from all 25 runs to simulate 4 parallel runs. Finally, the median of those 100000 mean misclassification error values is returned and presented in Table 2. The bold numbers in each row indicates the best performing algorithm for the specific dataset. We added a simple majority class baseline

3. The subset was selected to reduce computational demand. It was not cherry picked or altered in any way to improve results. A larger benchmark on more datasets is planned. An overview of the datasets can be found at https://github.com/ja-thomas/autoxgboost

Name	**Range**	**Dependency**	$\log_2$ **scale**	**Simple**
eta	$[0.01, 0.2]$		N	Y
gamma	$[-7, 6]$		Y	Y
max_depth	$\{3, 4, \ldots, 20\}$		N	Y
colsample_bytree	$[0.5, 1]$		N	Y
colsample_bylevel	$[0.5, 1]$		N	Y
lambda	$[-10, 10]$		Y	Y
alpha	$[-10, 10]$		Y	Y
subsample	$[0.5, 1]$		N	Y
booster	gbtree, gblinear, dart		N	N
sample_type	uniform weighted	dart	N	N
normalize_type	tree, forest	dart	N	N
rate_drop	$[0, 1]$	dart	N	N
skip_drop	$[0, 1]$	dart	N	N
one_drop	TRUE, FALSE	dart	N	N
grow_policy	depthwise, lossguide		N	N
max_leaves	$\{0, 1, \ldots, 8\}$	lossguide	Y	N
max_bin,	$\{2, 3, \ldots, 9\}$		Y	N

Table 1: Proposed hyperparameter spaces to tune over in autoxgboost. The first 8 parameters are defined as the simple space (default).

Dataset	baseline	autoxgboost	Auto-WEKA	auto-sklearn
Dexter	52.78	12.22	7.22	**5.56**
GermanCredit	32.67	27.67*	28.33	**27.00**
Dorothea	6.09	**5.22**	6.38	5.51
Yeast	68.99	**38.88**	40.45	40.67
Amazon	99.33	26.22	37.56	**16.00**
Secom	**7.87**	**7.87**	**7.87**	**7.87**
Semeion	92.45	8.38	**5.03**	5.24
Car	29,15	1.16	0.58	**0.39**
Madelon	50.26	16.54	21.15	**12.44**
KR-vs-KP	48.96	1.67	**0.31**	0.42
Abalone	84.04	73.75*	**73.02**	73.50
Wine Quality	55.68	**33.70**	**33.70**	33.76
Waveform	68.80	15.40*	**14.40**	14.93
Gisette	50.71	2.48	2.24	**1.62**
Convex	50.00	22.74	22.05	**17.53**
Rot. MNIST + BI	88.88	47.09*	55.84	**46.92**

Table 2: Benchmark results are median percent error across 100 000 bootstrap samples (out of 25 runs) simulating 4 parallel runs. Bold numbers indicate best performing algorithms. Stars indicate a relative difference of less than 5% to auto-sklearn.

as an indicator that all implementations work as they should, i.e., they should significantly outperform this baseline. As we can see easily in Table 2, only for the dataset *Secom*, the baseline achieves the same performance as the AutoML frameworks. On 9 of the 16 datasets, auto-sklearn provides the best results. So does Auto-WEKA on four and autoxgboost on two datasets. autoxgboost and Auto-WEKA slightly perform better than auto-sklearn on the *Wine Quality* dataset.

5. Conclusion

The benchmark results of Section 4 showed that autoxgboost was outperformed on the larger number of datasets by auto-sklearn, but was able to achieve competitive results on some datasets, providing state-of-the-art performance with only a single learning algorithm instead of using a whole library of possibly ensembled algorithms. This is not too surprising as the tuning space is much smaller and on some of the datasets very different learning algorithms might have an edge. Obviously, this is only a small initial benchmark that is not necessarily representative. We plan to evaluate on a larger set of OpenML (Vanschoren et al. (2014)) datasets in the future. One clear advantage of this approach is that the resulting models are boosting models, which can be deployed more easily and allow some form of interpretability for example via feature importance and individualized feature attribution (Lundberg and Lee (2017)). Our AutoML implementation autoxgboost is still in an early state and some of the design decisions are not final and will be evaluated and optimized in the future. Furthermore, we plan to extend the automatic gradient boosting framework to optimize for simultaneously sparse and well performing models using multiobjective SMBO strategies by Horn and Bischl (2016).

References

Bernd Bischl, Simon Wessing, Nadja Bauer, Klaus Friedrichs, and Claus Weihs. MOI-MBO: multiobjective infill for parallel model-based optimization. In *International Conference on Learning and Intelligent Optimization*, pages 173–186. Springer, 2014.

Bernd Bischl, Michel Lang, Lars Kotthoff, Julia Schiffner, Jakob Richter, Erich Studerus, Giuseppe Casalicchio, and Zachary M. Jones. mlr: Machine learning in R. *Journal of Machine Learning Research*, 17(170):1–5, 2016.

Bernd Bischl, Jakob Richter, Jakob Bossek, Daniel Horn, Janek Thomas, and Michel Lang. *mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions*, 2017.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2.

Yifei Chen, Zhenyu Jia, Dan Mercola, and Xiaohui Xie. A gradient boosting algorithm for survival analysis via direct optimization of concordance index. *Computational and mathematical methods in medicine*, 2013, 2013.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232, 10 2001.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11 (1):10–18, November 2009. ISSN 1931-0145.

Daniel Horn and Bernd Bischl. Multi-objective parameter configuration of machine learning algorithms using model-based optimization. In *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*, pages 1–8. IEEE, 2016.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. *Sequential Model-Based Optimization for General Algorithm Configuration*, pages 507–523. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-25566-3.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3149–3157. Curran Associates, Inc., 2017.

Ping Li, Qiang Wu, and Christopher J Burges. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in neural information processing systems*, pages 897–904, 2008.

Scott M Lundberg and Su-In Lee. Consistent feature attribution for tree ensembles. *arXiv preprint arXiv:1706.06060*, 2017.

Daniele Micci-Barreca. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *SIGKDD Explor. Newsl.*, 3(1):27–32, July 2001. ISSN 1931-0145.

Randal S. Olson, Ryan J. Urbanowicz, Peter C. Andrews, Nicole A. Lavender, La Creis Kidd, and Jason H. Moore. Automating biomedical data science through tree-based pipeline optimization. In Giovanni Squillero and Paolo Burelli, editors, *Applications of Evolutionary Computation*, pages 123–137, Cham, 2016. Springer International Publishing. ISBN 978-3-319-31204-0.

Francesco Orabona. Simultaneous model selection and optimization through parameter-free stochastic learning. In *Advances in Neural Information Processing Systems*, pages 1116–1124, 2014.

Philipp Probst, Marvin Wright, and Anne-Laure Boulesteix. Hyperparameters and tuning strategies for random forest. *arXiv preprint arXiv:1804.03515*, 2018.

KV Rashmi and Ran Gilad-Bachrach. Dart: Dropouts meet multiple additive regression trees. In *International Conference on Artificial Intelligence and Statistics*, pages 489–497, 2015.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proc. of KDD-2013*, pages 847–855, 2013.

Constantino Tsallis and Daniel A. Stariolo. Generalized simulated annealing. *Physica A: Statistical Mechanics and its Applications*, 233(1):395 – 406, 1996. ISSN 0378-4371.

Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.