Automatic Gradient Boosting

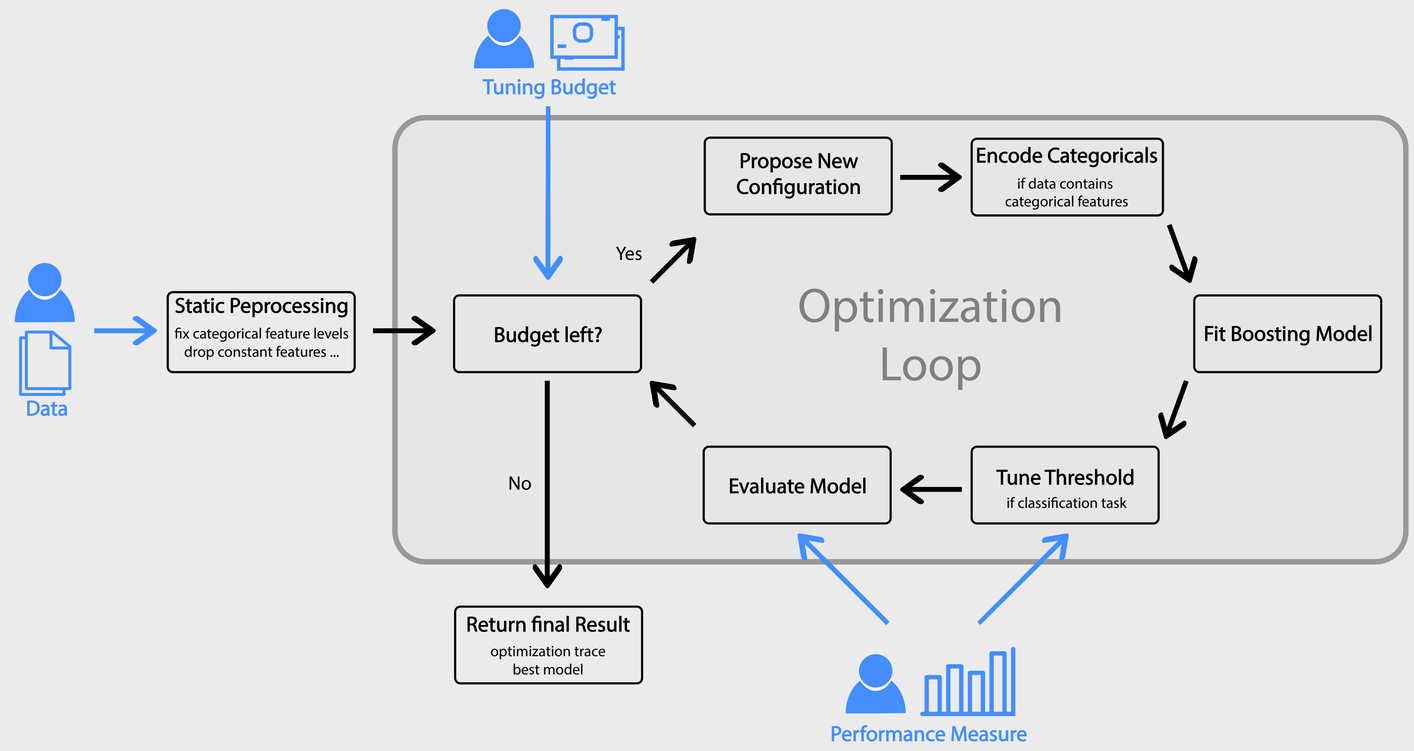
Janek Thomas, Stefan Coors, Bernd Bischl

Department of Statistics, Ludwig Maximilians University
janek.thomas@stat.uni-muenchen.de

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.

Approach



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

Autoxgboost

Project: `www.github.com/ja-thomas/autoxgboost`

Implementation in R using **xgboost** [4], **mlr** [2], **mlrMBO** [3] and **mlrCPO** [1].

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

Example

```
library(OpenML)
library(autoxgboost)
data = getOMLDataSet(31)
GermanCredit = convertOMLDataSetToMlr(data)
autoxgboost(GermanCredit, iterations = 50)
```

```
## Autoxgboost tuning result
##
## Recommended parameters:
##                eta: 0.115
##              gamma: 0.064
##          max_depth: 12
##  colsample_bytree: 0.965
## colsample_bylevel: 0.641
##             lambda: 0.193
##              alpha: 0.020
##          subsample: 0.828
##  scale_pos_weight: 73.283
##           nrounds: 41
##
##
## Preprocessing pipeline:
## (fixfactors >> dummyencode >> dropconst)(fixfactors.drop.unused.levels = TRUE,
## fixfactors.fix.factors.prediction = TRUE, dummyencode.reference.cat = FALSE, dropconst.rel.tol = 1e-08,
## dropconst.abs.tol = 1e-08, dropconst.ignore.na = FALSE)
##
## With tuning result: mmce = 0.195
## Classification Threshold: 0.569
```

Benchmark

						Results			
Name	**Factors**	**Numerics**	**Classes**	**Train instances**	**Test instances**	**baseline**	**autoxgboost**	**Auto-WEKA**	**auto-sklearn**
Dexter	20 000	0	2	420	180	52.78	12.22	7.22	**5.56**
GermanCredit	13	7	2	700	300	32.67	27.67*	28.33	**27.00**
Dorothea	100 000	0	2	805	345	6.09	**5.22**	6.38	5.51
Yeast	0	8	10	1 038	446	68.99	**38.88**	40.45	40.67
Amazon	10 000	0	49	1 050	450	99.33	26.22	37.56	**16.00**
Semeion	256	0	10	1 115	478	92.45	8.38	**5.03**	5.24
Car	6	0	4	1 209	519	29,15	1.16	0.58	**0.39**
Madelon	500	0	2	1 820	780	50.26	16.54	21.15	**12.44**
KR-vs-KP	37	0	2	2 237	959	48.96	1.67	**0.31**	0.42
Abalone	1	7	28	2 923	1 254	84.04	73.75*	**73.02**	73.50
Wine Quality	0	11	11	3 425	1 469	55.68	**33.70**	**33.70**	33.76
Waveform	0	40	3	3 500	1 500	68.80	15.40*	**14.40**	14.93
Gisette	5 000	0	2	4 900	2 100	50.71	2.48	2.24	**1.62**
Convex	0	784	2	8 000	50 000	50.00	22.74	22.05	**17.53**
Rot. MNIST + BI	0	784	10	12 000	50 000	88.88	47.09*	55.84	**46.92**

Median percent error across 100000 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. Results of Auto-WEKA and auto-sklearn are taken from [5].

References

[1] Martin Binder. *mlrCPO: Composable Preprocessing Operators for mlr*, 2018. R package.

[2] 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.

[3] 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.