

# 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.

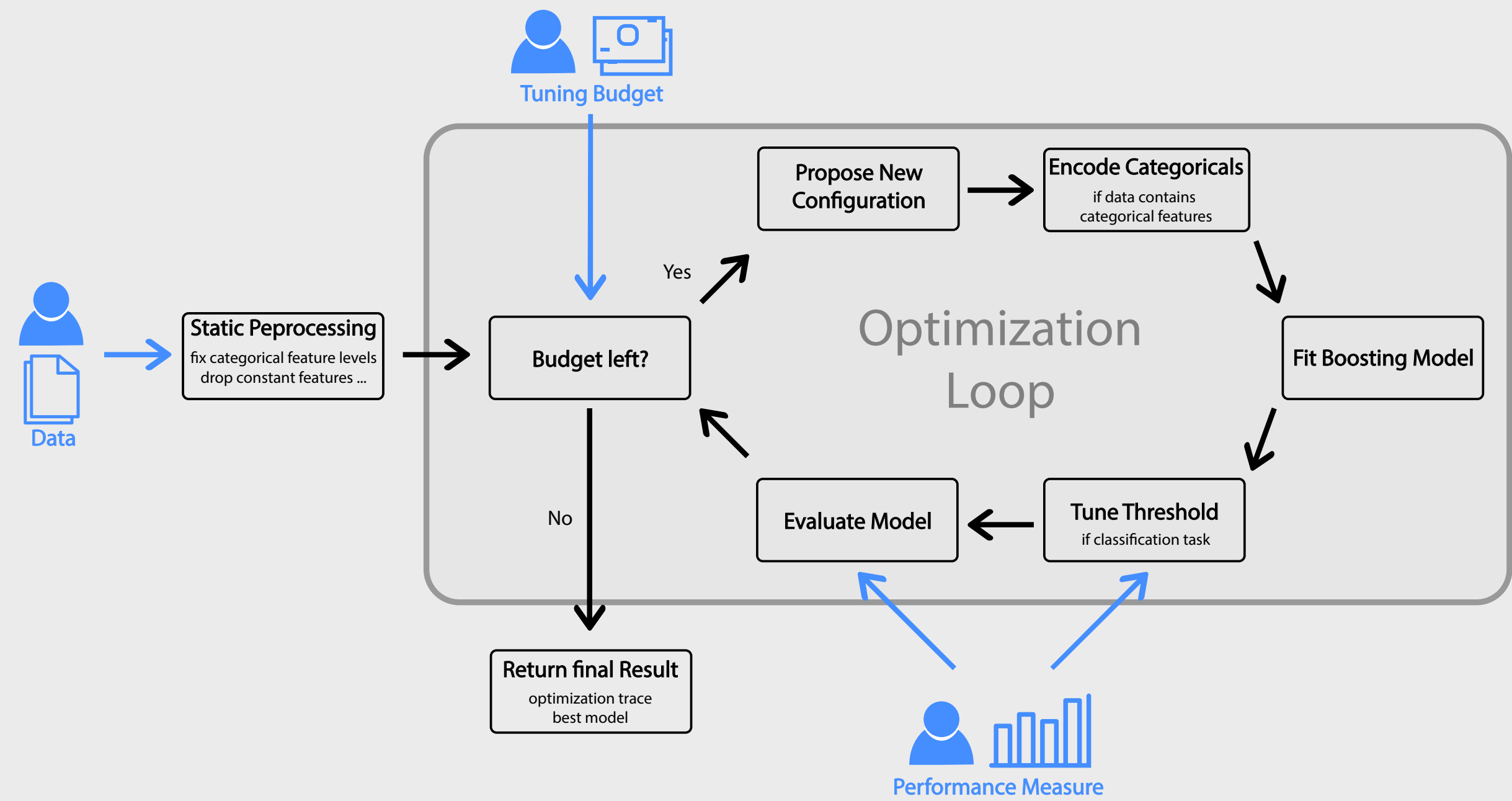
## Autoxgboost

**Project:** [www.github.com/ja-thomas/autoxgboost](https://www.github.com/ja-thomas/autoxgboost)

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

| Name                     | Range                  | Dependency | log <sub>2</sub> scale | Simple |
|--------------------------|------------------------|------------|------------------------|--------|
| <i>eta</i>               | [0.01, 0.2]            |            | N                      | Y      |
| <i>gamma</i>             | [−7, 6]                |            | Y                      | Y      |
| <i>max_depth</i>         | {3, 4, . . . , 20}     |            | N                      | Y      |
| <i>colsample_bytree</i>  | [0.5, 1]               |            | N                      | Y      |
| <i>colsample_bylevel</i> | [0.5, 1]               |            | N                      | Y      |
| <i>lambda</i>            | [−10, 10]              |            | Y                      | Y      |
| <i>alpha</i>             | [−10, 10]              |            | Y                      | Y      |
| <i>subsample</i>         | [0.5, 1]               |            | N                      | Y      |
| <i>booster</i>           | gbtree, gblinear, dart |            | N                      | N      |
| <i>sample_type</i>       | uniform weighted       | dart       | N                      | N      |
| <i>normalize_type</i>    | tree, forest           | dart       | N                      | N      |
| <i>rate_drop</i>         | [0, 1]                 | dart       | N                      | N      |
| <i>skip_drop</i>         | [0, 1]                 | dart       | N                      | N      |
| <i>one_drop</i>          | TRUE, FALSE            | dart       | N                      | N      |
| <i>grow_policy</i>       | depthwise, lossguide   |            | N                      | N      |
| <i>max_leaves</i>        | {0, 1, . . . , 8}      | lossguide  | Y                      | N      |
| <i>max_bin</i>           | {2, 3, . . . , 9}      |            | Y                      | N      |

## Approach



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

## 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

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

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].