Accelerated component-wise gradient boosting using efficient data representation and momentum-based optimization

D. Schalk, B. Bischl, D. Rügamer

18th, Dec. 2022

Background

About the project

- Component-wise boosting (CWB; Bühlmann and Yu, 2003; Bühlmann et al., 2007) is a method that ensembles weak base learners to a strong learner.
- Utilizing interpretable statistical models as base learners makes the fitted model interpretable.
- Also, CWB can fit models in high dimensional features spaces ($n \ll p$) and provides an inherent (unbiased) feature selection.
- But, depending on the size of the data, it is infeasible to train the algorithm w.r.t runtime or memory allocation.

$\Rightarrow$ **Goal:** Speed up the fitting process and reduce the memory consumption without loosing predictive or estimation accuracy (Schalk et al., 2022).

CWB basics

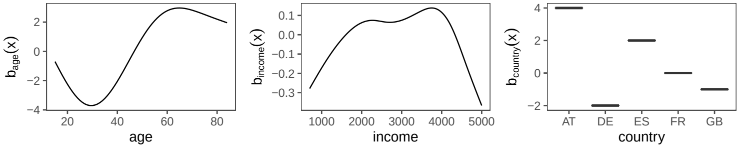
- Feature vector $\boldsymbol{x} = (x_1, \dots, x_p) \in \mathcal{X}$
- CWB fits a model $\hat{f}$ based on an additive structure

$$\mathbb{E}[Y|\boldsymbol{x}] = \hat{f}(\boldsymbol{x}) = f_0 + \sum_{k=1}^{K} b_k(\boldsymbol{x}).$$

  Hence, CWB can be used as fitting engine for GAMs.
- K additive terms represented by base learner $b_k : \mathcal{X} \to \mathbb{R}$. Often, one base learner per feature is used to model univariate effects.

Example

- Three features x_{age}, x_{country}, and x_{income}
- Response Y is a numeric score for "happiness"
- Aim: Model Y with b_{age} a P-spline, b_{income} a P-spline and b_{country} a categorical effect



- Model: $\hat{f}(\boldsymbol{x}) = f_0 + b_{\text{age}}(\boldsymbol{x}) + b_{\text{income}}(\boldsymbol{x}) + b_{\text{country}}(\boldsymbol{x})$

CWB algorithm I

- Given is a data set $\mathcal{D}$ with n observations and p features, a loss function $L(y, \hat{y})$, and a set of base learners $b_1, \dots, b_K$.
- The gaol of fitting CWB is to minimize the empirical risk $\mathcal{R}_{\text{emp}}(\hat{f}|\mathcal{D}) = n^{-1} \sum_{(x,y)\in\mathcal{D}} L(y, \hat{f}(x))$.
- The algorithm is initialized with a loss-optimal constant c that minimizes the empirical risk:

$$f_0 = \hat{f}^{[0]}(\boldsymbol{x}) = \arg\min_{c\in\mathcal{Y}} \mathcal{R}_{\text{emp}}(c|\mathcal{D})$$

- Model updates in iteration $m \in \mathbb{N}$ are calculated by applying functional gradient descent:

$$\hat{f}^{[m+1]} = \hat{f}^{[m]} + \nu \hat{b}^{[m]}$$

CWB algorithm II

- The negative functional gradient is expressed by pseudo residuals:

$$r^{[m](i)} = -\nabla_f L(y^{(i)}, f(\boldsymbol{x}^{(i)}))\big|_{f=\hat{f}^{[m-1]}},\ \ i \in \{1, \dots, n\}$$

- In CWB, each base learner $b_1, \dots, b_K$ is fitted to $\boldsymbol{r}^{[m]}$ and the one with the lowest sum of squared errors (SSE) is chosen as new component $\hat{b}^{[m]}$.
- The last two step are repeated M times.

CWB algorithm III

Algorithm 1 Vanilla CWB algorithm

Input Train data $\mathcal{D}$, learning rate ν, number of boosting iterations M, loss function L, base learners $b_1, \dots, b_K$
Output Model $\hat{f} = \hat{f}^{[M]}$

```
1:  procedure CWB($\mathcal{D}, \nu, L, b_1, \dots, b_K$)
2:      Initialize: $f_0 = \hat{f}^{[0]}(\mathbf{x}) = \arg\min_{c \in \mathcal{Y}} \mathcal{R}_{\text{emp}}(c|\mathcal{D})$
3:      while $m \leq M$ do
4:          $r^{[m](i)} = -\left.\frac{\partial L(y^{(i)}, f(\mathbf{x}^{(i)}))}{\partial f(\mathbf{x}^{(i)})}\right|_{f=\hat{f}^{[m-1]}}, \quad \forall i \in \{1, \dots, n\}$
5:          for $k \in \{1, \dots, K\}$ do
6:              $\hat{\boldsymbol{\theta}}_k^{[m]} = \left(\mathbf{Z}_k^{\mathsf{T}} \mathbf{Z}_k + \mathbf{K}_k\right)^{-1} \mathbf{Z}_k^{\mathsf{T}} \mathbf{r}^{[m]}$
7:              $\text{SSE}_k = \sum_{i=1}^{n} (r^{[m](i)} - b_k(\mathbf{x}^{(i)}|\hat{\boldsymbol{\theta}}_k^{[m]}))^2$
8:          $k^{[m]} = \arg\min_{k \in \{1,\dots,K\}} \text{SSE}_k$
9:          $\hat{f}^{[m]}(\mathbf{x}) = \hat{f}^{[m-1]}(\mathbf{x}) + \nu b_{k^{[m]}}(\mathbf{x}|\hat{\boldsymbol{\theta}}_{k^{[m]}}^{[m]})$
10:     return $\hat{f} = \hat{f}^{[M]}$
```

Goal of the project

⇒ Increase CWB's efficiency:

- **Acceleration:** Speed up the fitting process by using Nesterovs momentum.
- **Memory:** Reduce the memory consumption by discretizing numerical features.

Acceleration

Nesterovs momentum

Gradient descent:

Parameter space | Function space

$$\hat{\boldsymbol{\theta}}^{[m+1]} = \hat{\boldsymbol{\theta}}^{[m]} + \nu \nabla_{\boldsymbol{\theta}} \mathcal{R}_{\text{emp}}(\hat{f}(.|\hat{\boldsymbol{\theta}}^{[m]})|\mathcal{D}) \quad \Rightarrow \quad \hat{f}^{[m+1]} = \hat{f}^{[m]} + \nu \hat{b}^{[m]}$$

Nesterov momentum:

Parameter space | Function space

$$\begin{aligned} \boldsymbol{u}^{[m]} &= \nabla_{\boldsymbol{\theta}} \mathcal{R}_{\text{emp}}(\hat{f}(.|\hat{\boldsymbol{\theta}}^{[m]} - \gamma \hat{\boldsymbol{\vartheta}}^{[m-1]})|\mathcal{D}) \\ \hat{\boldsymbol{\vartheta}}^{[m]} &= \gamma \hat{\boldsymbol{\vartheta}}^{[m-1]} + \nu \boldsymbol{u}^{[m]} \\ \hat{\boldsymbol{\theta}}^{[m+1]} &= \hat{\boldsymbol{\theta}}^{[m]} + \hat{\boldsymbol{\vartheta}}^{[m]} \end{aligned} \quad \Rightarrow \quad ???$$

$\Rightarrow$ **Idea:** Use Nesterovs momentum and adjust it for functional updates and CWB.

Nesterovs momentum in function space

- Biau et al. (2019) proposed an adjustment of gradient boosting with Nesterovs momentum as optimizer.
- Lu et al. (2020) lined out that this approach may diverge and proposed an corrected algorithm Accelerated Gradient Boosting Machine (AGBM):

$$
\begin{aligned}
g^{[m]} &= (1 - \theta_m) f^{[m]} + \theta_m h^{[m]} \\
f^{[m+1]} &= g^{[m]} + \eta b^{[m]} \\
h^{[m+1]} &= h^{[m]} + \eta / \theta_m b_{\text{cor}}^{[m]}
\end{aligned}
$$

with $\theta_m = 2/(m+1)$ and $h^{[m]}$ the momentum sequence.

Base learners in AGBM

- $b^{[m]}$ is fit to pseudo residuals $\boldsymbol{r}^{[m]}$ w.r.t. $\hat{g}^{[m-1]}$ instead of $\hat{f}^{[m]}$
- $b_{\text{cor}}^{[m]}$ is fit to error-corrected pseudo residuals:

$$c^{[m](i)} = r^{[m](i)} + \frac{m}{m+1}(c^{[m-1](i)} - \hat{b}_{\text{cor}}^{[m-1]}(\boldsymbol{x}^{(i)})),$$

with $i = 1, \dots, n$, if $m > 1$ and $\boldsymbol{c}^{[m]} = \boldsymbol{r}^{[m]}$ if $m = 0$.

Adapting AGBM for CWB

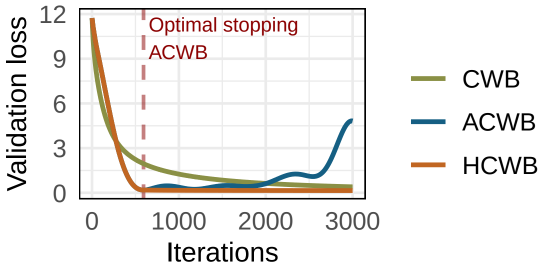
In Schalk et al. (2022), we proposed an accelerated CWB (ACWB) version by incorporating these adaptions to CWB, therefore:

- Both base learners, $b^{[m]}$ and $b_{\text{cor}}^{[m]}$, are the result of a selection process that chooses one of $b_1, \dots, b_K$ w.r.t. to the minimal SSE on the respective pseudo residuals $\boldsymbol{r}^{[m]}$ and $\boldsymbol{c}^{[m]}$.
- Update the estimated parameters accordingly to allow the estimation of partial feature effects.

Considering these points allows to maintain all advantages of CWB in ACWB. We refer to the publication for details about the algorithms.

Hybrid CWB

- It was proven by Lu et al. (2020), that ACWB can overfit if not stopped early.
- Therefore, we combined ACWB with CWB to accelerate the fitting in the beginning and fine-tune the model using CWB:

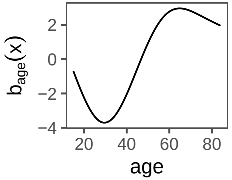


Memory

Base learner design matrix

- Each base learner $b_1, \dots, b_K$ requires to build a design matrix $\boldsymbol{Z}_k \in \mathbb{R}^{n \times d_k}$ based on the feature vector $\boldsymbol{x}_k$.
- For example:

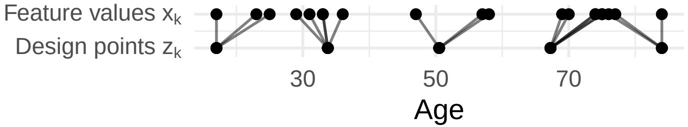
$$\underbrace{\begin{pmatrix} 80 \\ 31 \\ \vdots \\ 28 \\ 49 \end{pmatrix}}_{=\boldsymbol{x}_{\text{age}}} \Rightarrow \underbrace{\begin{pmatrix} 0 & 0 & 0 & 0.04 & 0.56 & 0.39 & 0.01 \\ 0.02 & 0.45 & 0.51 & 0.03 & 0 & 0 & 0 \\ & & & \vdots & & & \\ 0.13 & 0.66 & 0.21 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.03 & 0.53 & 0.43 & 0.01 & 0 \end{pmatrix}}_{=\boldsymbol{Z}_{\text{age}} \text{ (B-spline basis)}} \Rightarrow$$



$\Rightarrow$ If n is large, the memory gets filled very fast (especially if p is also large).

Binning

- To reduce the memory consumption, we applied binning to operate on a reduced representation of $\boldsymbol{Z}_k$.
- Binning is a technique that allows to represent the n values $x_k^{(1)}, \dots, x_k^{(n)}$ of $\boldsymbol{x}_k$ by $n^* < n$ design points $\boldsymbol{z}_k = (z_k^{(1)}, \dots, z_k^{(n^*)})$.
- The idea is to assign each $x_k^{(i)}$ to the closest design point $z_k^{(i)}$ and store the assignment in a map $\text{ind}_k^{(i)}$: $x_k^{(i)} \approx z_k^{(\text{ind}_k^{(i)})}$



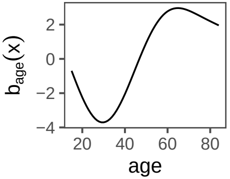
Binning in GLMs

- Lang et al. (2014) used binning to discretize feature vectors to increase the efficiency of multilevel structured additive regression.
- Wood et al. (2017) applied binning to fit GAMs to gigadata and argue that the best approximation is achieved by setting $n^* = \sqrt{n}$.
- Li and Wood (2020) presented optimized cross-product operations of binned design matrices to also speed up the fitting.

Binning in CWB

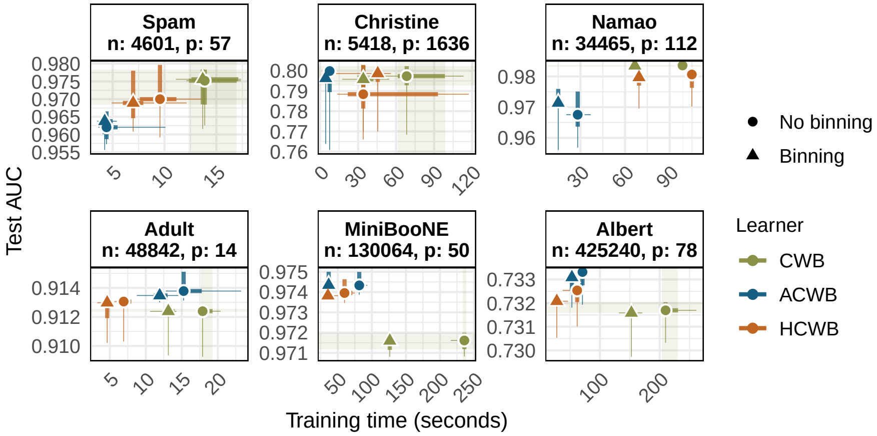
- Represent numerical features $\boldsymbol{x}_k$ by n^* design points $\boldsymbol{z}_k$.
- Build the design matrix $\boldsymbol{Z}_k^*$ based on $\boldsymbol{z}_k$ which requires to store $n^* d_k$ values instead of nd_k.
- Use optimized cross-product operations to estimate the parameters $\hat{\boldsymbol{\theta}}_k^{[m]}$ of base learner b_k to also speed up the fitting.

$$\underbrace{\begin{pmatrix} 15 \\ 32 \\ 50 \\ 67 \\ 84 \end{pmatrix}}_{=\boldsymbol{z}_{\text{age}}} \Rightarrow \underbrace{\begin{pmatrix} 0.17 & 0.67 & 0.17 & 0 & 0 & 0 & 0 \\ 0 & 0.34 & 0.59 & 0.06 & 0 & 0 & 0 \\ 0 & 0 & 0.01 & 0.43 & 0.53 & 0.01 & 0 \\ 0 & 0 & 0 & 0.06 & 0.59 & 0.34 & 0 \\ 0 & 0 & 0 & 0 & 0.17 & 0.67 & 0.17 \end{pmatrix}}_{=\boldsymbol{Z}^*_{\text{age}} \text{ (B-spline basis)}} \Rightarrow$$



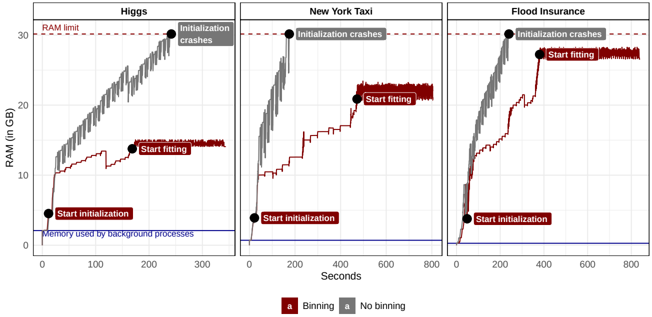
Application to data

Benchmark


Spam
n: 4601, p: 57
Christine
n: 5418, p: 1636
Namao
n: 34465, p: 112
Adult
n: 48842, p: 14
MiniBooNE
n: 130064, p: 50
Albert
n: 425240, p: 78
Test AUC
Training time (seconds)
● No binning
▲ Binning
Learner
CWB
ACWB
HCWB
0.980
0.975
0.970
0.965
0.960
0.955
0.80
0.79
0.78
0.77
0.76
0.98
0.97
0.96
0.914
0.912
0.910
0.975
0.974
0.973
0.972
0.971
0.733
0.732
0.731
0.730
5
10
15
0
30
60
90
120
30
60
90
5
10
15
20
50
100
150
200
250
100
200

Binning with big data

- HIGGS: 11×10^6 observations, 29 features, 2.4 GB
- NYC Yellow Taxi Trip: 24.3×10^6 observations, 22 features, 3.3 GB
- FEMA's National Flood Insurance Policy Database: 14.5×10^6 observations, 50 features, 3.4 GB



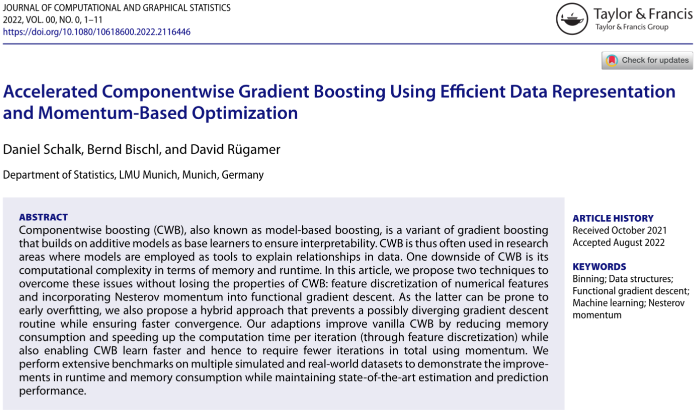
Summary

Summary

- It may be infeasible to train CWB to large amounts of data.
- Two adaptions aim to increase the efficiency without sacrificing performance:
  - **Optimizer:** Incorporating Nesterovs momentum speed up the fitting.
  - **Memory:** Building the design matrix of a base learner on binned data points reduces the memory load.

Summary

- More details about the theory, simulation studies, and benchmarks can be found in our paper:



JOURNAL OF COMPUTATIONAL AND GRAPHICAL STATISTICS
2022, VOL. 00, NO. 0, 1–11
https://doi.org/10.1080/10618600.2022.2116446

Taylor & Francis
Taylor & Francis Group

Check for updates

Accelerated Componentwise Gradient Boosting Using Efficient Data Representation and Momentum-Based Optimization

Daniel Schalk, Bernd Bischl, and David Rügamer

Department of Statistics, LMU Munich, Munich, Germany

ABSTRACT
Componentwise boosting (CWB), also known as model-based boosting, is a variant of gradient boosting that builds on additive models as base learners to ensure interpretability. CWB is thus often used in research areas where models are employed as tools to explain relationships in data. One downside of CWB is its computational complexity in terms of memory and runtime. In this article, we propose two techniques to overcome these issues without losing the properties of CWB: feature discretization of numerical features and incorporating Nesterov momentum into functional gradient descent. As the latter can be prone to early overfitting, we also propose a hybrid approach that prevents a possibly diverging gradient descent routine while ensuring faster convergence. Our adaptions improve vanilla CWB by reducing memory consumption and speeding up the computation time per iteration (through feature discretization) while also enabling CWB learn faster and hence to require fewer iterations in total using momentum. We perform extensive benchmarks on multiple simulated and real-world datasets to demonstrate the improvements in runtime and memory consumption while maintaining state-of-the-art estimation and prediction performance.

ARTICLE HISTORY
Received October 2021
Accepted August 2022

KEYWORDS
Binning; Data structures; Functional gradient descent; Machine learning; Nesterov momentum

Thanks for your attention!

Questions?

References

Biau, G., Cadre, B., and Rouvìère, L. (2019). Accelerated gradient boosting. *Machine Learning*, 108(6):971–992.

Bühlmann, P., Hothorn, T., et al. (2007). Boosting algorithms: Regularization, prediction and model fitting. *Statistical science*, 22(4):477–505.

Bühlmann, P. and Yu, B. (2003). Boosting with the L2 loss: regression and classification. *Journal of the American Statistical Association*, 98(462):324–339.

Lang, S., Umlauf, N., Wechselberger, P., Harttgen, K., and Kneib, T. (2014). Multilevel structured additive regression. *Statistics and Computing*, 24(2):223–238.

Li, Z. and Wood, S. N. (2020). Faster model matrix crossproducts for large generalized linear models with discretized covariates. *Statistics and Computing*, 30(1):19–25.

Lu, H., Karimireddy, S. P., Ponomareva, N., and Mirrokni, V. (2020). Accelerating gradient boosting machines. In *International Conference on Artificial Intelligence and Statistics*, pages 516–526. PMLR.

Schalk, D., Bischl, B., and Rügamer, D. (2022). Accelerated componentwise gradient boosting using efficient data representation and momentum-based optimization. *Journal of Computational and Graphical Statistics*, 0(ja):1–27.

Wood, S. N., Li, Z., Shaddick, G., and Augustin, N. H. (2017). Generalized additive models for gigadata: Modeling the u.k. black smoke network daily data. *Journal of the American Statistical Association*, 112(519):1199–1210.