Compboost

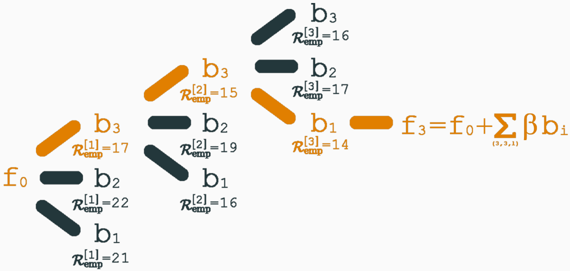
Modular framework for **comp**onent-wise **boost**ing

Daniel Schalk, Janek Thomas, Bernd Bischl
daniel.schalk@stat.uni-muenchen.de

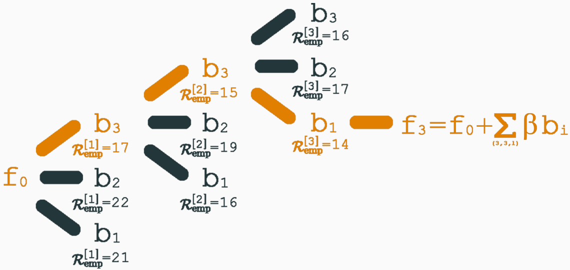
September 28, 2018

LMU Munich
Working Group Computational Statistics

Why Component-Wise Boosting?


f0
b3
R[1]emp=17
b2
R[1]emp=22
b1
R[1]emp=21
b3
R[2]emp=15
b2
R[2]emp=19
b1
R[2]emp=16
b3
R[3]emp=16
b2
R[3]emp=17
b1
R[3]emp=14
f3=f0+∑(3,3,1) β bi

Why Component-Wise Boosting?



- Inherent (unbiased) feature selection ▸ Hofner et al. (2011).
- Resulting model is sparse since important effects are selected first and therefore it is able to learn in high-dimensional feature spaces ($p \gg n$).
- Parameters are updated iteratively. Therefore, the whole trace of how the model evolves is available.

Available R Packages

Most popular package for model-based boosting is `mboost` ▸ Hothorn et al. (2017):

- Large number of available base-learner and losses.
- Extended to more complex problems:
  - Functional data ▸ Brockhaus et al. (2017)
  - GAMLSS models ▸ Mayr et al. (2012)
  - Survival analysis
- Extendible with custom base-learner and losses.

Available R Packages

Most popular package for model-based boosting is `mboost` ▸ Hothorn et al. (2017):

- Large number of available base-learner and losses.
- Extended to more complex problems:
  - Functional data ▸ Brockhaus et al. (2017)
  - GAMLSS models ▸ Mayr et al. (2012)
  - Survival analysis
- Extendible with custom base-learner and losses.

So, why another boosting implementation?

- Main parts of `mboost` are written in `R` and gets slow for large datasets.
- Complex implementation:
  - Nested scopes
  - Mixture of different `R` class systems

About compboost

About compboost

The `compboost` package is a fast and flexible framework for model-based boosting completely written in `C++`:

- With `mboost` as standard, we want to keep the modular principle of defining custom base-learner and losses.
- Completely written in `C++` and exposed by `Rcpp` ▸ Eddelbuettel (2013) ▸ Eddelbuettel and François (2017) to obtain high performance and full memory control.
- `R` API is written in `R6` to provide convenient wrapper.
- Major parts of the `compboost` functionality are unit tested against `mboost` to ensure correctness.

Functionality of compboost

Main components:

- Base-learner and loss classes.
- Logger class for early stopping and logging mechanisms.

Possible extensions:

- Custom `R` or `C++` base-learner.
- Custom `R` or `C++` loss objects.
- Custom logging and stopping rules via custom losses.

Custom classes can be defined without recompiling the whole package, even when using `C++` functions.

Usecase

Initialize Model

We are interested in modelling the risk of diabetes of female Pima Indians. Interesting features are age and the `mass`.

```r
library(compboost)

data(PimaIndiansDiabetes, package = "mlbench")

# Defining a new Compboost object:
cboost = Compboost$new(data = PimaIndiansDiabetes, target = "diabetes",
  loss = LossBinomial$new())

# Adding a linear and spline base-learner to the Compboost object:
cboost$addBaselearner(feature = "mass", id = "linear", BaselearnerPolynomial,
  degree = 1, intercept = TRUE)
cboost$addBaselearner(feature = "age", id = "spline", BaselearnerPSpline,
  degree = 3, n.knots = 10, penalty = 2, differences = 2)
```

Initialize Model

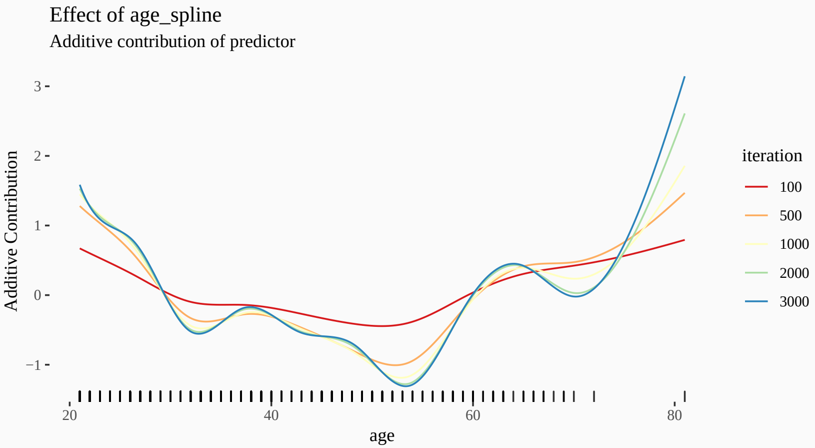
```
cboost$train(2000, trace = 1000)
##    1/2000: risk = 0.66
## 1000/2000: risk = 0.54
## 2000/2000: risk = 0.54
##
##
## Train 2000 iterations in 0 Seconds.
## Final risk based on the train set: 0.54

cboost
## Component-Wise Gradient Boosting
##
## Trained on PimaIndiansDiabetes with target diabetes
## Number of base-learners: 2
## Learning rate: 0.05
## Iterations: 2000
## Positive class: neg
## Offset: 0.3118
##
## LossBinomial Loss:
##
##   Loss function: L(y,x) = log(1 + exp(-2yf(x))
##
##
```

Access Results and Continue Training

```
cboost$train(1000) # Set model to iteration 1000

table(cboost$getSelectedBaselearner()) # Table of vector of selected base-learner
##
##  age_spline mass_linear
##         611         389

cboost$train(3000) # Set model to iteration 3000
##
## You have already trained 2000 iterations.
## Train 1000 additional iterations.

str(cboost$getInbagRisk()) # Get vector of inbag risk
##  num [1:3001] 0.658 0.657 0.655 0.653 0.652 ...

str(cboost$getEstimatedCoef()) # Get list of estimated parameter
## List of 3
##  $ age_spline : num [1:14, 1] 5.581 0.67 1.251 -1.134 0.199 ...
##  $ mass_linear: num [1:2, 1] 3.08 -0.09
##  $ offset     : num 0.312
```

Plot Results

```
cboost$plot("age_spline", iters = c(100, 500, 1000, 2000, 3000))
```



Custom Loss: Definition

As an example we want to define a custom loss corresponding to the Poisson distribution:

```r
lossPoisson = function (truth, response) {
  return (-log(exp(response)^truth * exp(-exp(response)) / gamma(truth + 1)))
}
gradPoisson = function (truth, response) {
  return (exp(response) - truth)
}
constInitPoisson = function (truth) {
  return (log(mean.default(truth)))
}
# Define custom loss:
my.poisson.loss = LossCustom$new(lossPoisson, gradPoisson, constInitPoisson)
```

Custom Loss: Train Model

```
data(VonBort, package = "vcd")

# Run compboost with custom loss:
cboost = Compboost$new(VonBort, "deaths", loss = my.poisson.loss)
cboost$addBaselearner("year", "spline", BaselearnerPSpline)
cboost$train(100, trace = 0)
## Train 100 iterations in 0 Seconds.
## Final risk based on the train set: 1.1

# Run mboost with pre-defined Poisson family:
mod = mboost(deaths ~ bbs(year, lambda = 2), data = VonBort, family = Poisson(),
  control = boost_control(mstop = 100, nu = 0.05))

head(data.frame(
  compboost = cboost$getEstimatedCoef()[["year_spline"]],
  mboost = coef(mod)[["bbs(year, lambda = 2)"]]))
##   compboost  mboost
## 1   -1.2233 -1.2233
## 2   -0.9133 -0.9133
## 3   -0.6078 -0.6078
## 4   -0.3389 -0.3389
## 5   -0.1809 -0.1809
## 6   -0.0788 -0.0788
```

Using Wrapper Functions

Instead of defining all base-learner individually we can use `boostLinear()` or `boostSplines()` to boost just linear or spline base-learner on all features:

```
# Run wrapper again with custom loss:
cboost = boostSplines(data = VonBort, target = "deaths", loss = my.poisson.loss,
  trace = 50, learning.rate = 0.01, iterations = 200)
##   1/200: risk = 1.1
##  50/200: risk = 1.1
## 100/200: risk = 1.1
## 150/200: risk = 1.1
## 200/200: risk = 1.1
##
##
## Train 200 iterations in 0 Seconds.
## Final risk based on the train set: 1.1

head(cboost$getBaselearnerNames())
## [1] "year_spline"        "corps_G_category"
## [3] "corps_I_category"   "corps_II_category"
## [5] "corps_III_category" "corps_IV_category"
```

Benchmark

Runtime Comparison With mboost

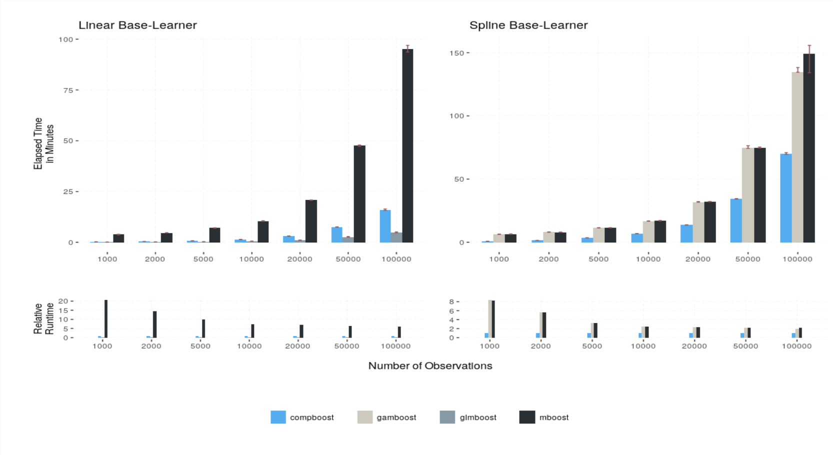


Figure 1: Runtime benchmark on simulated data with 2000 iterations and 2000 observations.

Runtime Comparison With mboost

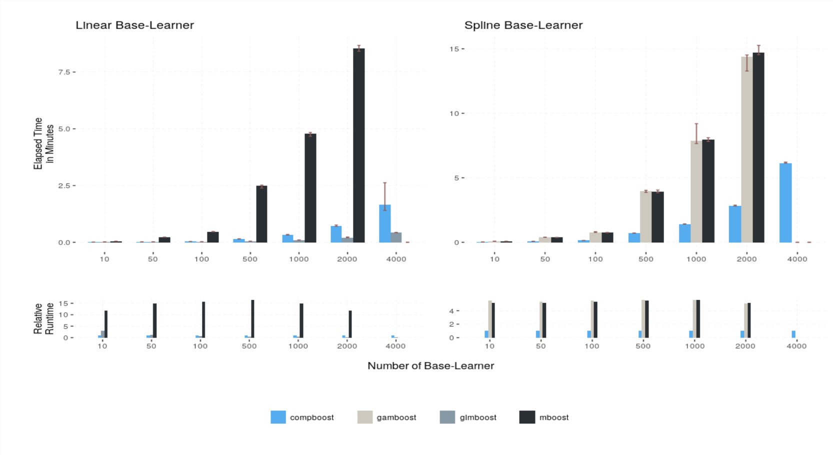


Figure 2: Runtime benchmark on simulated data with 2000 iterations and 1000 base-learner.

Memory Comparison With mboost

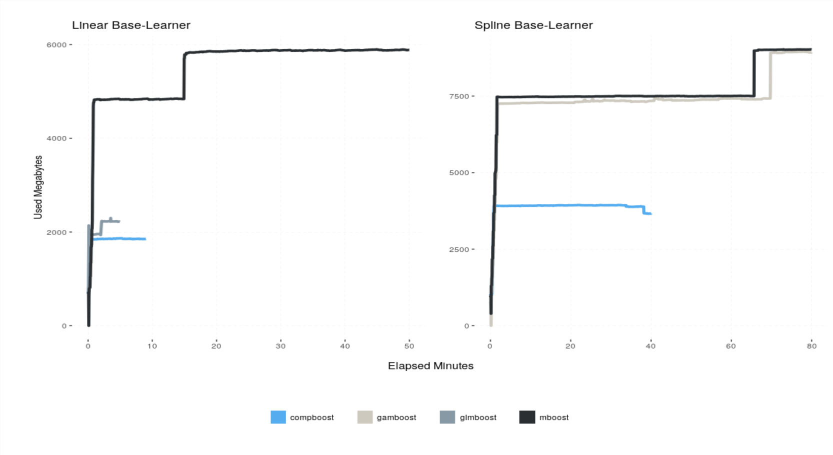


Figure 3: Memory benchmark on simulated data with 1000 iterations, 50000 observations, and 1000 base-learner.

Next Steps

Next Steps

- Implementing more base-learner and loss functions.
- Adding support for multiclass classification.
- Running the algorithm in parallel.
- Releasing the package on *CRAN*.

Where to Find

- Developed on *GitHub*:

  www.github.com/schalkdaniel/compboost

- Additional resources on the project page:

  www.compboost.org

- Bug reports via the issue tracker.

Contributions are highly welcome!

References

Brockhaus, S., Rügamer, D., and Greven, S. (2017). Boosting functional regression models with fdboost. *arXiv preprint arXiv:1705.10662*.

Eddelbuettel, D. (2013). *Seamless R and C++ Integration with Rcpp*. Springer, New York. ISBN 978-1-4614-6867-7.

Eddelbuettel, D. and François, R. (2017). Exposing C++ functions and classes with rcpp modules. *Vignette included in R package Rcpp, URL http://CRAN.R-Project.org/package=Rcpp*.

Hofner, B., Hothorn, T., Kneib, T., and Schmid, M. (2011). A framework for unbiased model selection based on boosting. *Journal of Computational and Graphical Statistics*, 20(4):956–971.

Hothorn, T., Buehlmann, P., Kneib, T., Schmid, M., and Hofner, B. (2017). *mboost: Model-Based Boosting*. R package version 2.9-0.

Mayr, A., Fenske, N., Hofner, B., Kneib, T., and Schmid, M. (2012). Generalized additive models for location, scale and shape for high dimensional data – a flexible approach based on boosting. *Journal of the Royal Statistical Society: Series C – Applied Statistics*, 61(3):403–427.

Sanderson, C. and Curtin, R. (2016). Armadillo: a template-based c++ library for linear algebra. *Journal of Open Source Software*, 1(2):26.