Compboost

Modular framework for component-wise boosting

Daniel Schalk, Janek Thomas, Bernd Bischl

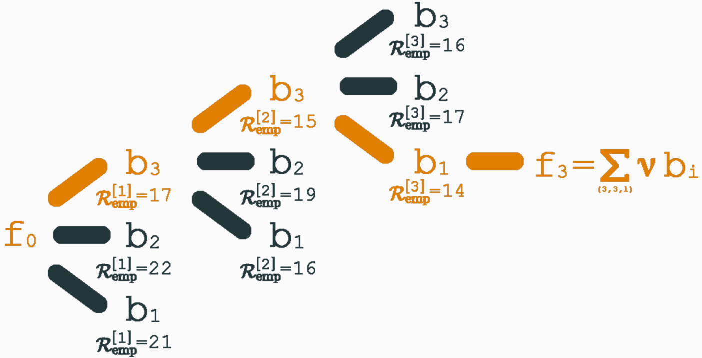
10 July 2018

LMU Munich
Working Group Computational Statistics

Table of Contents

Why Component-Wise Boosting

Why Component-Wise Boosting?



Why Component-Wise Boosting?

- Inherent (unbiased) feature selection [Hofner et al., 2011].
- The resulting model is sparse since important effects are selected first and therefore it is able to learn in high-dimensional feature spaces ($p \gg n$).
- The 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.
- Suited for 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 algorithm is implemented in `R` which is not the best choice for expensive algorithms.
- Complex implementation:
  - Nested scopes
  - Mixture of different `R` class systems.

About Compboost

About Compboost

- 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 with `R6` to provide convenient wrapper.
- Major parts of the `compboost` functionality are unit tested against `mboost` to ensure correct use.

Compboost: Functionality

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.

Small Usecase

Initializing Model

```r
library(compboost)

data(PimaIndiansDiabetes, package = "mlbench")

cboost = Compboost$new(PimaIndiansDiabetes, "diabetes",
  loss = BinomialLoss$new())

cboost$addBaselearner("age", "spline", PSplineBlearner,
  degree = 3, n.knots = 10, penalty = 2, differences = 2)
cboost$addBaselearner("mass", "linear", PolynomialBlearner,
  degree = 1, intercept = TRUE)
```

Initializing Model

```
cboost$train(2000, trace = FALSE)
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
##
## BinomialLoss Loss:
##
##   Loss function: L(y,x) = log(1 + exp(-2yf(x))
##
##
```

Access Results

```
table(cboost$selected())
```

```
##
##  age_spline mass_linear
##        1511         489
```

```
head(cboost$risk())
```

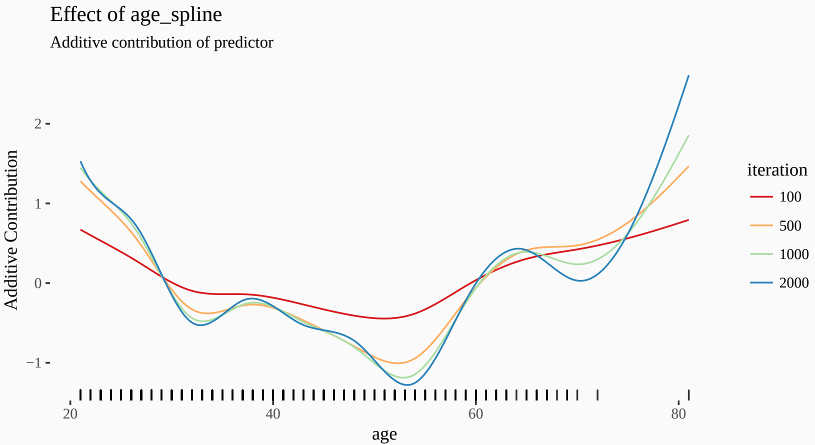
```
## [1] 0.6581551 0.6565063 0.6548970 0.6533263 0.6517930
## [6] 0.6502964
```

```
str(cboost$coef())
```

```
## List of 3
##  $ age_spline : num [1:14, 1] 4.407 0.913 1.124 -1.044 0.126 ...
##  $ mass_linear: num [1:2, 1] 3.0352 -0.0886
##  $ offset     : num 0.312
```

Plot Results

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



Custom Poisson Loss: Definition

```
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 = CustomLoss$new(lossPoisson, gradPoisson, constInitPoisson)
```

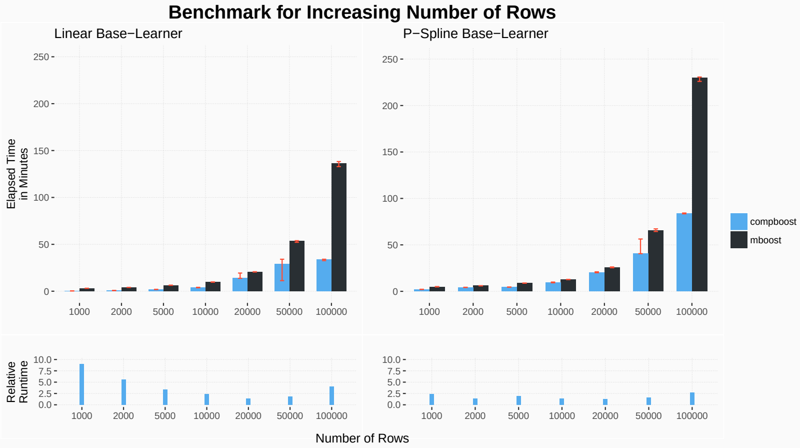
Custom Poisson Loss: Train Model

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

cboost = Compboost$new(VonBort, "deaths", loss = my.poisson.loss)
cboost$addBaselearner("year", "spline", PSplineBlearner)
cboost$train(100, trace = FALSE)

mod = mboost(deaths ~ bbs(year, lambda = 2), data = VonBort, family = Poisson(),
  control = boost_control(mstop = 100, nu = 0.05))

head(cbind(
  compboost = as.numeric(cboost$coef()[[1]]),
  mboost = as.numeric(coef(mod)[[1]])))
```

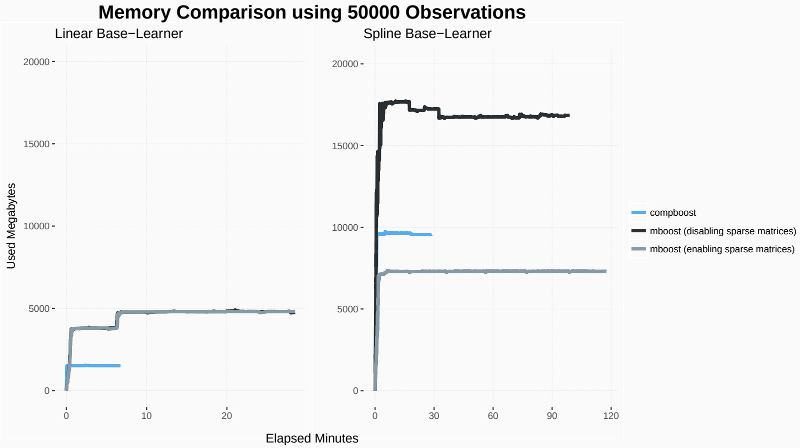
```
##        compboost      mboost
## [1,] -1.22328271 -1.22328271
## [2,] -0.91327247 -0.91327247
## [3,] -0.60784297 -0.60784297
## [4,] -0.33885367 -0.33885367
## [5,] -0.18094044 -0.18094044
## [6,] -0.07880335 -0.07880335
```

Small Benchmark

Runtime Comparison Against Mboost



Memory Comparison Against Mboost



Next Steps

Next Steps

- Implementing more base-learner and loss functions.
- Support for multiclass classification.
- Parallel computations.
- Using sparse matrices to store, e.g., spline data matrices.

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!

Questions?

References i

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.

References ii

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.