

# Compboost

Modular framework for **component-wise boosting**

---

Daniel Schalk, Janek Thomas, Bernd Bischl

[daniel.schalk@stat.uni-muenchen.de](mailto:daniel.schalk@stat.uni-muenchen.de)

10 July 2018

LMU Munich

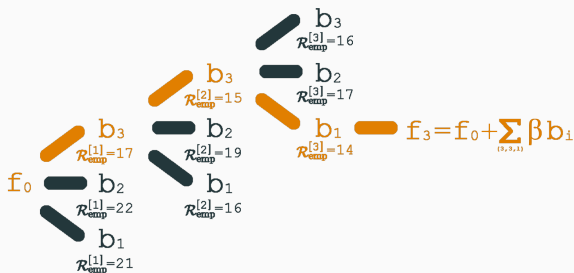
Working Group Computational Statistics



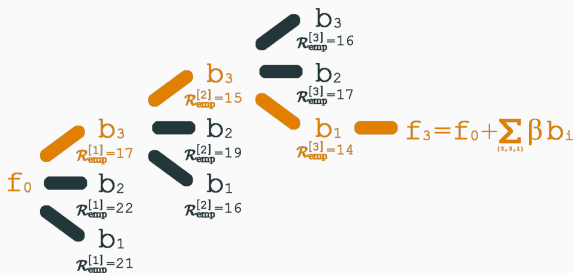
# Why Component-Wise Boosting?

---

# Why Component-Wise Boosting?



# 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.

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.

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 Comboost

---

# 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 [▶ Edelbuettel \(2013\)](#)  
[▶ Edelbuettel 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.



# Comboost: 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.

# Usecase

---

# Initialize Model

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

```
library(compboost)

data(PimaIndiansDiabetes, package = "mlbench")

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

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

# Initialize 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 and Continue Training

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

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

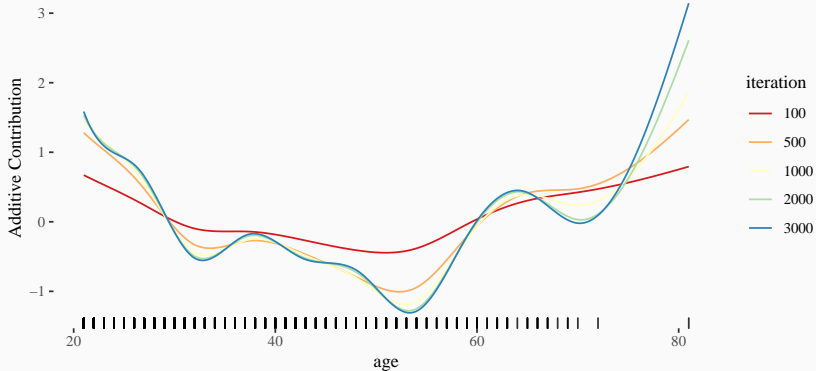
str(cboost$coef()) # Get list of estimated parameter
## List of 3
##  $ age_spline : num [1:14, 1] 5.581 0.67 1.251 -1.134 0.199 ...
##  $ mass_linear: num [1:2, 1] 3.08 -0.09
##  $ offset      : num 0.312
```

# Plot Results

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

## Effect of age\_spline

Additive contribution of predictor



# Custom Loss: Definition

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

```
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 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", PSplineBlearner)
cboost$train(100, trace = FALSE)

# 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))

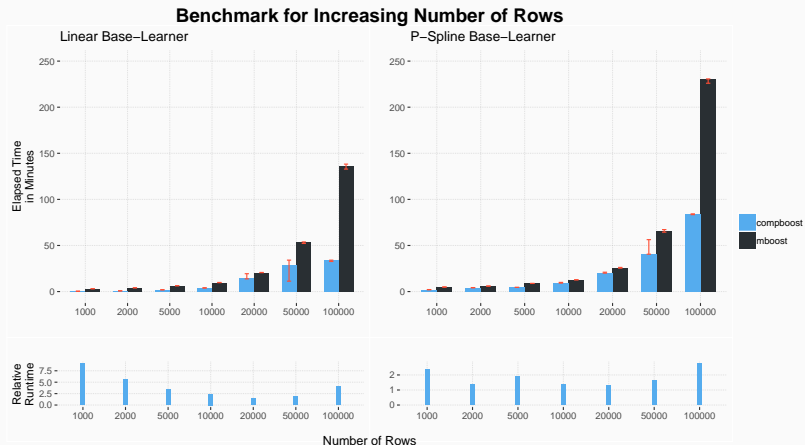
head(data.frame(
  compboost = cboost$coef()[["year_spline"]],
  mboost = coef(mod)[["bbs(year, lambda = 2)"]]))
##   compboost  mboost
## 1   -1.2233 -1.2233
## 2   -0.9133 -0.9133
## 3   -0.6078 -0.6078
## 4   -0.3389 -0.3389
## 5   -0.1809 -0.1809
## 6   -0.0788 -0.0788
```



# Benchmark

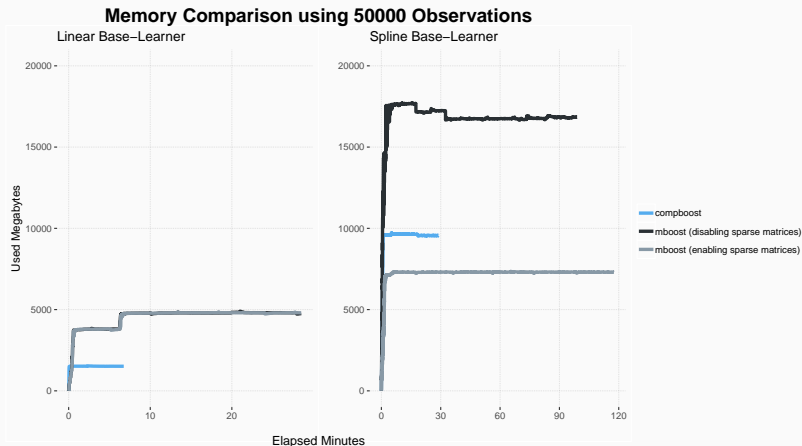
---

# Runtime Comparison With Mboost



**Figure 1:** Runtime benchmark on simulated data with 2000 iterations and 1000 base-learner.

# Memory Comparison With Mboost



**Figure 2:** 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.
- Using sparse matrices to store, e.g., spline data matrices.
- Releasing the package on *CRAN*.

# Where to Find

- Developed on *GitHub*:

[www.github.com/schalkdaniel/compboost](https://www.github.com/schalkdaniel/compboost)

- Additional resources on the project page:

[www.compboost.org](https://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.