Compboost

Modular framework for component-wise boosting

Daniel Schalk

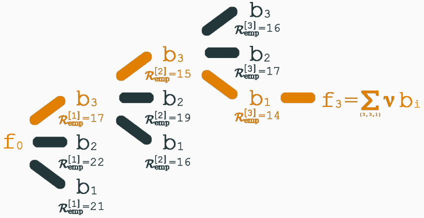
June 26, 2018

LMU Munich
Working Group Computational Statistics

Table of contents

Why Component-Wise Boosting

Why Component-Wise Boosting?



b3
$\mathcal{R}_{emp}^{[3]}=16$
b3
$\mathcal{R}_{emp}^{[2]}=15$
b2
$\mathcal{R}_{emp}^{[3]}=17$
b3
$\mathcal{R}_{emp}^{[1]}=17$
b2
$\mathcal{R}_{emp}^{[2]}=19$
b1
$\mathcal{R}_{emp}^{[3]}=14$
$f_3=\sum_{(3,3,1)} \nu b_i$
f_0
b2
$\mathcal{R}_{emp}^{[1]}=22$
b1
$\mathcal{R}_{emp}^{[2]}=16$
b1
$\mathcal{R}_{emp}^{[1]}=21$

Component-Wise Boosting: The Idea

Available R Packages

Most popular package for model-based boosting is `mboost`:

- Huge functionality: Lots of available base-learner and losses.
- Suited to boost more complex analyses such as survival tasks.
- Nice possibility to extend with own base-learner and losses.

So, why another boosting implementation?

- Major parts of the algorithm are implemented in `R` which is not the best choice for expensive algorithms.
- The implementation is hard to debug due to nested scopes and no standard `R` system like `S3` or `S4`.

About Compboost

About Compboost

- With `mboost` as standard, we want to keep the modular principle of defining custom base-learner and losses.
- Core is written in `C++` and exposed by `Rcpp` to obtain high performance and full memory control.
- `R` API is written with `R6` to provide convenient wrapper.
- Major parts of `compboost` functions are unit tested against `mboost` to ensure correct use.

Compboost: Functionality

Main classes:

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

Custom functionality:

- `R` or `C++` base-learner.
- `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$addBaselearner("age", "spline", PSplineBlearner,
  degree = 3, knots = 10, penalty = 2, differences = 2)
cboost$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())
```

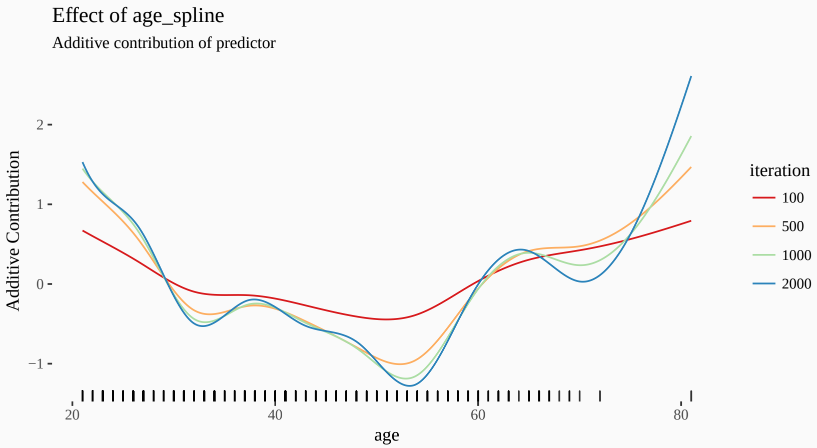
```
##
##  age_spline mass_linear
##        1511         489
```

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

```
## [1] 0.6582 0.6565 0.6549 0.6533 0.6518 0.6503
```

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

Plot Results



Custom Poisson Loss: Definition

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

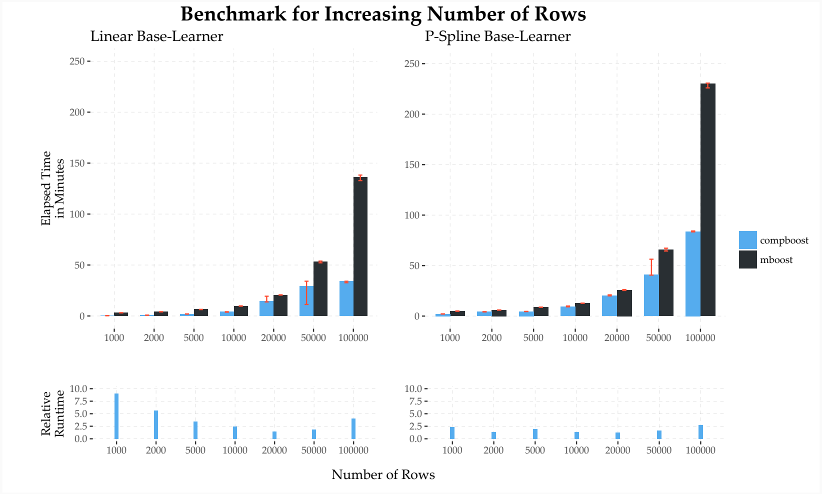
Custom Poisson Loss: Train Model

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

cboost = Compboost$new(VonBort, "deaths", loss = my.poisson.loss)
cboost$addBaselearner("year", "spline", PSplineBlearner,
  degree = 3, knots = 10, penalty = 2, differences = 2)
cboost$train(100, trace = FALSE)

mod = mboost(deaths ~ bbs(year, differences = 2, lambda = 2,
  degree = 3, knots = 10), 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]])))
```

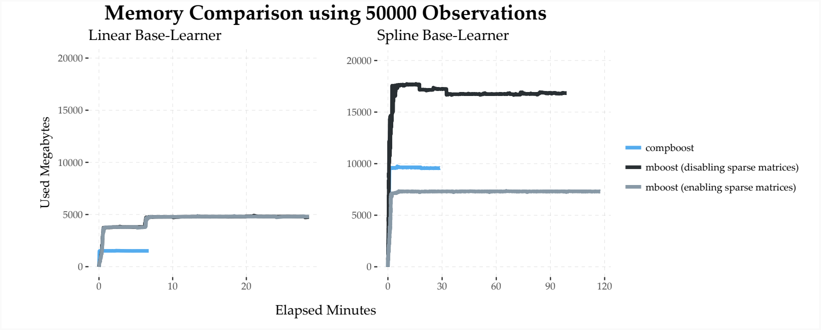
```
##      compboost  mboost
## [1,]   -1.4093 -1.4093
## [2,]   -0.8975 -0.8975
## [3,]   -0.4040 -0.4040
## [4,]    0.0947  0.0947
## [5,]    0.3384  0.3384
## [6,]    0.1531  0.1531
```

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.

Questions?