Compboost

Modular framework for component-wise boosting

Daniel Schalk

June 23, 2018

LMU Munich
Working Group Computational Statistics

Table of contents

1. What is Component-Wise Boosting
2. Compboost vs Other Implementations
3. About Compboost
4. Small Usecase
5. Next Steps

What is Component-Wise Boosting

Component-Wise Boosting: Terminology

- Loss Function:

$$L : \mathcal{Y} \times \mathcal{X} \to \mathbb{R}$$

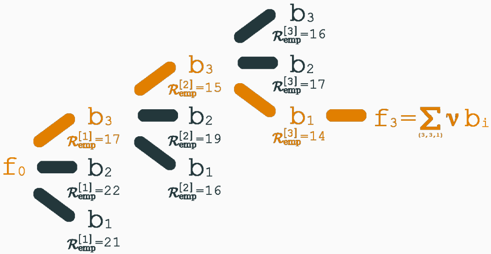
- Empirical Risk:

$$\mathcal{R}_{\text{emp}}(\theta) = \frac{1}{n} \sum_{i=1}^{n} L\left(y^{(i)}, f(x^{(i)})\right)$$

- Estimated model/parameter at iteration m:

$$\hat{f}^{[m]}, \theta^{[m]}$$

Component-Wise Boosting: The Idea



Iteration 1: $\hat{f}^{[1]}(x) = \beta b_3(x_3, \theta^{[1]})$

Iteration 2: $\hat{f}^{[2]}(x) = \beta b_3(x_3, \theta^{[1]}) + \beta b_3(x_3, \theta^{[2]})$

Iteration 2: $\hat{f}^{[3]}(x) = \beta b_3(x_3, \theta^{[1]}) + \beta b_3(x_3, \theta^{[2]}) + \beta b_1(x_1, \theta^{[3]})$

$$\Rightarrow \hat{f}^{[3]}(x) = \beta \left(b_3(x_3, \theta^{[1]} + \theta^{[2]}) + b_1(x_1, \theta^{[3]}) \right)$$

Component-Wise Boosting: The Algorithm

Result: Component-wise boosting model $\hat{f}(x)$

Initialize $\hat{f}^{[0]}(x) = \arg\min_{c \in \mathbb{R}} \mathcal{R}_{\text{emp}}(c)$;

for $m \in \{1, \dots, M\}$ **do**

// Update pseudo residuals:

$$r^{[m](i)} = -\left[\frac{\delta}{\delta f(x^{(i)})} L\left(y^{(i)}, f(x^{(i)})\right)\right]_{f = f^{[m-1]}}, \ \forall i \in \{1, \dots, n\} \ ;$$

// Get index j^* of m-th base-learner from optimizer:

for $j \in \{1, \dots, J\}$ **do**

// Fit each base-learner $b_j^{[m]}$ to the pseudo residuals:

$$\hat{\theta}_j^{[m]} = \arg\min_{\theta_j} \sum_{i=1}^{n} \left(r^{[m](i)} - b_j^{[m]}(x^{(i)}, \theta_j)\right)^2 \ ;$$

// Calculate the SSE of the fitted base-learner:

$$\text{SSE}_j = \sum_{i=1}^{n} \left(r^{[m](i)} - b_j^{[m]}(x^{(i)}, \hat{\theta}_j)\right)^2 \ ;$$

end

// Add selected component to model:

$$\hat{f}^{[m]}(x) = \hat{f}^{[m-1]}(x) + \beta b_{j^*}^{[m]}\left(x, \theta_{j^*}^{[m]}\right)$$

end

Returns: $\hat{f}(x) = \hat{f}^{[m]}(x)$;

Available R Packages

- Tree-based implementations:
    - `xgboost`
    - `catboost`
    - `gbm`
- Model-based implementations:
    - `mboost` (`gamboost`, `gamboostLSS`)

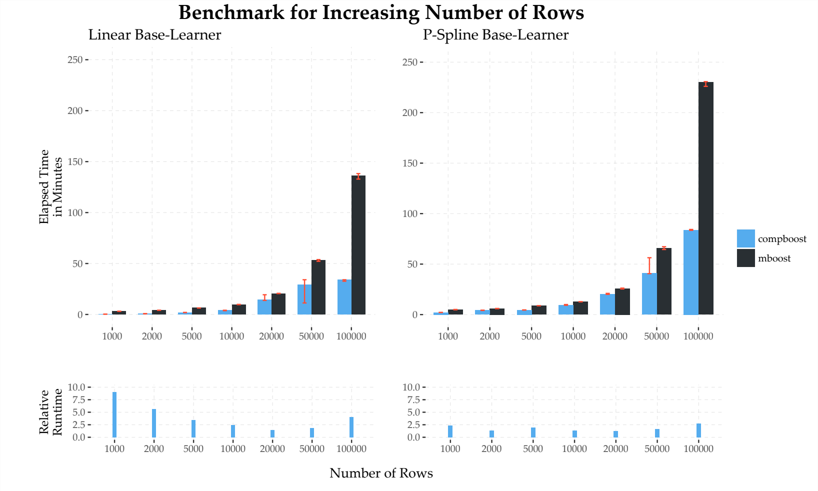
So, why another boosting implementation?

Compboost vs Other Implementations

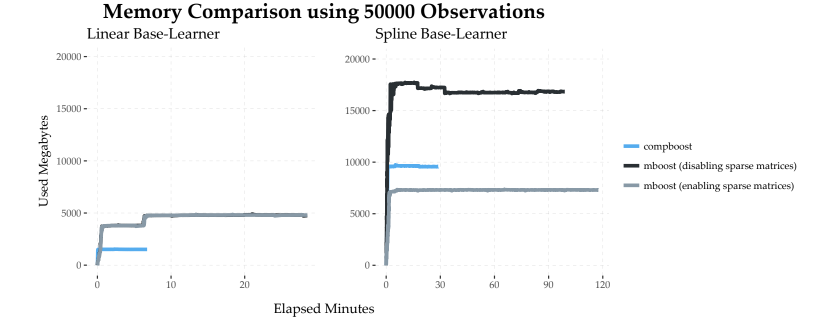
Compboost vs Other Implementations

- Compboost is not designed to have huge predictive power such as `xgboost`.
- Hence, the only competitor is `mboost`:
    - `compboost` is not as comprehensive as `mboost` (just two base-learners and three pre defined losses).
    - `compboost` is not able to use sparse matrices at the moment.

Runtime Comparison



Memory Comparison



About Compboost

About Compboost

- Installation

```
devtools::install_github("schalkdaniel/compboost")
library(compboost)
```

Compboost Members and Member Functions

- **Member Functions**:
  - `addBaselearner()`
  - `addLogger()`
  - `train()`
  - `coef()`
  - `predict()`
  - `risk()`
  - `selected()`
  - `plot()`
  - ...
- **Public Members**:
  - `model`
  - `bl.factory.list`
  - `loss`
  - `optimizer`
  - ...

Small Usecase

Initializng Model

```
mtcars$mpg_cat = ifelse(mtcars$mpg > 15, "A", "B")
cboost = Compboost$new(mtcars, "mpg", loss = QuadraticLoss$new())

cboost$addBaselearner("wt", "spline", PSplineBlearnerFactory,
        degree = 3, knots = 10, penalty = 2, differences = 2)
cboost$addBaselearner("mpg_cat", "linear", PolynomialBlearnerFactory,
        degree = 1, intercept = FALSE)

cboost$train(2000, trace=FALSE)
cboost


## Componentwise Gradient Boosting
##
## Trained on mtcars with target mpg
## Number of base-learners: 3
## Learning rate: 0.05
## Iterations: 2000
## Offset:20.090625
##
## QuadraticLoss Loss:
##
##   Loss function: y = (y - f(x))^2
##
##
```

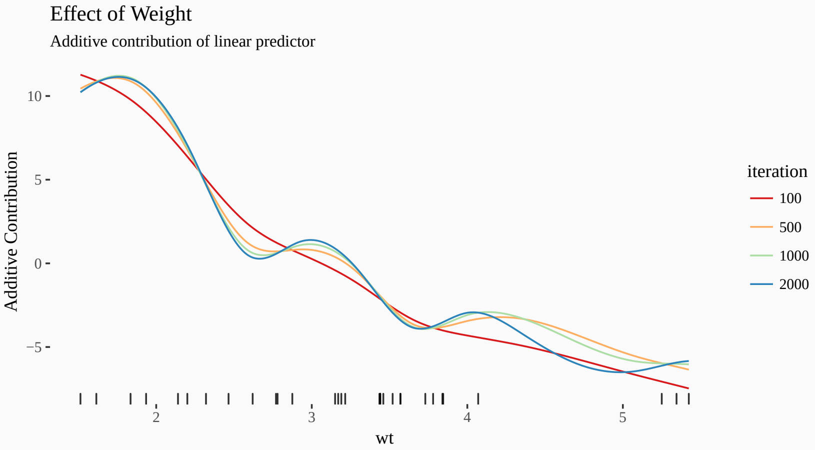
Plot Results

With `plot()` it is possible to illustrate a specific effect. Additionaly, we can specify which iterations we want to visualize. The returned object is an ordinary `ggplot` object:

```
library(ggplot2)
library(ggthemes)

cboost$plot("wt_spline", iters = c(100, 500, 1000, 2000)) +
        labs(title = "Effect of Weight",
                subtitle = "Additive contribution of linear predictor") +
        theme_tufte() +
        scale_color_brewer(palette = "Spectral")
```

Plot Results



Using a Custom Logger

1. Define a custom loss which returns the AUC as "loss":

```r
# Define custom "loss function":
aucLoss = function (truth, response) {
  # Convert response on f basis to probs using sigmoid:
        probs = 1 / (1 + exp(-response))

  #  Calculate AUC:
        mlr:::measureAUC(probabilities = probs, truth = truth,
          negative = -1, positive = 1)
}

# Define also gradient and constant initalization since they are
# required by the custom constructors:
gradDummy = function (truth, response) { return (NA) }
constInitDummy = function (truth, response) { return (NA) }

# Define loss:
auc.loss = CustomLoss$new(aucLoss, gradDummy, constInitDummy)
```

Using a Custom Logger

2. Register a new out of bag risk logger with the custom loss:

```
cboost$addLogger(logger = OobRiskLogger, use.as.stopper = FALSE,
  logger.id = "auc_oob", auc.loss, 0.01, cboost$prepareData(mtcars[idx.test, ]),
  mtcars[idx.test, "mpg_bin"])
cboost$addLogger(logger = TimeLogger, use.as.stopper = FALSE,
  logger.id = "time", max.time = 0, time.unit = "microseconds")

cboost$train(1000)

      Iteration |  Out of Bag Risk |     microseconds |
-------------------------------------------------------
           1/10 |             0.91 |                1 |
           2/10 |             0.91 |             2000 |
           3/10 |             0.91 |             3490 |
           4/10 |             0.91 |             5218 |
           5/10 |             0.91 |             6293 |
           6/10 |             0.91 |             7152 |
```

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?