compboost

Fast and Flexible Component-Wise Boosting Framework

Daniel Schalk, Janek Thomas, and Bernd Bischl

July 12, 2019

LMU Munich
Working Group Computational Statistics
Use-Case
The Situation

- We own a small booth at the city center that sells beer.
- As we are very interested in our customers’ health, we only sell to customers who we expect to drink less than 110 liters per year.
- To estimate how much a customer drinks, we have collected data from 200 customers in recent years.
- The data includes the beer consumption (in liter), age, sex, country of origin, weight, body size, and 200 characteristics gained from app usage (that have absolutely no influence).
### Overview of the Data

<table>
<thead>
<tr>
<th>beer_consumption</th>
<th>gender</th>
<th>country</th>
<th>age</th>
<th>weight</th>
<th>height</th>
<th>app_usage1</th>
<th>...</th>
<th>app_usage200</th>
</tr>
</thead>
<tbody>
<tr>
<td>106.5</td>
<td>m</td>
<td>Seychelles</td>
<td>33</td>
<td>87.17</td>
<td>172.9</td>
<td>0.1680</td>
<td>...</td>
<td>0.1313</td>
</tr>
<tr>
<td>85.5</td>
<td>f</td>
<td>Seychelles</td>
<td>52</td>
<td>89.38</td>
<td>200.4</td>
<td>0.8075</td>
<td>...</td>
<td>0.6087</td>
</tr>
<tr>
<td>116.5</td>
<td>f</td>
<td>Czechia</td>
<td>54</td>
<td>92.03</td>
<td>178.7</td>
<td>0.3849</td>
<td>...</td>
<td>0.5786</td>
</tr>
<tr>
<td>67.0</td>
<td>m</td>
<td>Australia</td>
<td>32</td>
<td>63.53</td>
<td>186.3</td>
<td>0.3277</td>
<td>...</td>
<td>0.3594</td>
</tr>
<tr>
<td>43.0</td>
<td>f</td>
<td>Australia</td>
<td>51</td>
<td>64.73</td>
<td>175.0</td>
<td>0.6021</td>
<td>...</td>
<td>0.7406</td>
</tr>
<tr>
<td>85.0</td>
<td>m</td>
<td>Austria</td>
<td>43</td>
<td>95.74</td>
<td>173.2</td>
<td>0.6044</td>
<td>...</td>
<td>0.4181</td>
</tr>
<tr>
<td>79.0</td>
<td>f</td>
<td>Austria</td>
<td>55</td>
<td>87.65</td>
<td>156.3</td>
<td>0.1246</td>
<td>...</td>
<td>0.4398</td>
</tr>
<tr>
<td>107.0</td>
<td>f</td>
<td>Austria</td>
<td>24</td>
<td>93.17</td>
<td>161.4</td>
<td>0.2946</td>
<td>...</td>
<td>0.6130</td>
</tr>
<tr>
<td>57.0</td>
<td>m</td>
<td>USA</td>
<td>55</td>
<td>76.27</td>
<td>182.5</td>
<td>0.5776</td>
<td>...</td>
<td>0.4927</td>
</tr>
<tr>
<td>89.0</td>
<td>m</td>
<td>USA</td>
<td>16</td>
<td>72.21</td>
<td>203.3</td>
<td>0.6310</td>
<td>...</td>
<td>0.0735</td>
</tr>
</tbody>
</table>
Our Goals

With this data we want to answer the following questions:

- Which of the customers’ characteristics are important to be able to determine the consumption?
- How does the effect of important features look like?
- How does the model behave on unseen data?
What is Component-Wise Boosting?
General Idea

- Sequential fitting of the base-learner $b_1, b_2, b_3$ on the error / pseudo-residuals of the current ensemble.
- The base-learner with the best fit on the error (measured as mean squared error) is added to the ensemble.
- Results in a weighted sum / additive model over base-learners.
Advantages of Component-Wise Boosting

- Inherent (unbiased) feature selection.
- 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.
Base-Learner Paths

What is Component-Wise Boosting?

Daniel Schalk, Janek Thomas, and Bernd Bischl
About Compboost
Most popular package for model-based boosting is \texttt{mboost}:

- Large number of available base-learner and losses.
- Extended to more complex problems:
  - Functional data
  - GAMLSS models
  - Survival analysis
- Extendible with custom base-learner and losses.

So, why another boosting implementation?

- Main parts of \texttt{mboost} are written in \texttt{R} and gets slow for large datasets.
- Complex implementation:
  - Nested scopes
  - Mixture of different \texttt{R} class systems
Fast and flexible framework for model-based boosting:

- With \texttt{mboost} as standard, we want to keep the modular principle of defining custom base-learner and losses.
- Completely written in C++ and exposed by \texttt{Rcpp} to obtain high performance and full memory control.
- \texttt{R} API is written in \texttt{R6} to provide convenient wrapper.
- Major parts of the \texttt{compboost} functionality are unit tested against \texttt{mboost} to ensure correctness.
Small Demonstration
boostLinear() and boostSplines() automatically add univariate linear models or a GAM for all features.

```r
set.seed(618)
cboost = boostSplines(data = beer_data, target = "beer_consumption",
                      loss = LossAbsolute$new(), learning_rate = 0.1, iterations = 5000L,
                      penalty = 10, oob_fraction = 0.3, trace = 2500L)

## 1/5000 risk = 24 oob_risk = 24
## 2500/5000 risk = 0.6 oob_risk = 8.3
## 5000/5000 risk = 0.44 oob_risk = 8.3
##
## Train 5000 iterations in 11 Seconds.
## Final risk based on the train set: 0.44
```
Visualizing the Results

```r
gg1 = cboost$plotInbagVsOobRisk()

gg2 = cboost$plotFeatureImportance()
```
Visualizing the Results

cboost$\texttt{train}(2000L)$

gg1 = cboost$\texttt{plotFeatureImportance}()$

gg2 = cboost$\texttt{plot}("age\_spline", \texttt{iters} = c(50, 100, 500, 1000, 2000, 4000))$

---

country\_Czechia\_category

age\_spline

country\_USA\_category

country\_Australia\_category

app\_usage70\_spline

appp\_usage81\_spline

app\_usage158\_spline

app\_usage181\_spline

country\_Seychelles\_category

app\_usage171\_spline

app\_usage118\_spline

app\_usage103\_spline

app\_usage97\_spline

app\_usage95\_spline

app\_usage99\_spline

---

Effect of age\_spline

Additive contribution of predictor

<table>
<thead>
<tr>
<th>iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>500</td>
</tr>
<tr>
<td>1000</td>
</tr>
<tr>
<td>2000</td>
</tr>
<tr>
<td>4000</td>
</tr>
</tbody>
</table>

---

Small Demonstration

Daniel Schalk, Janek Thomas, and Bernd Bischl
Using the R6 Interface

cboost = Compboost$new(data = beer_data, target = "beer_consumption",
  loss = LossQuantile$new(0.9), learning_rate = 0.1, oob_fraction = 0.3)

cboost$addBaselearner("age", "spline", BaselearnerPSpline)
cboost$addBaselearner("country", "category", BaselearnerPolynomial)

cboost$addLogger(logger = LoggerTime, use_as_stopper = TRUE, logger_id = "time",
  max_time = 2e5, time_unit = "microseconds")

cboost$train(10000, trace = 500)

## 1/10000 risk = 11 oob_risk = 10 time = 0
## 500/10000 risk = 7.9 oob_risk = 8.2 time = 22107
## 1000/10000 risk = 6.3 oob_risk = 6.6 time = 46764
## 1500/10000 risk = 5.1 oob_risk = 5.4 time = 76091
## 2000/10000 risk = 4.2 oob_risk = 4.5 time = 112149
## 2500/10000 risk = 3.5 oob_risk = 3.8 time = 154647
##
## Train 2978 iterations in 0 Seconds.
## Final risk based on the train set: 3.2
Overview of the Functionality

- **Base-learner**: BaselearnerPolynomial, BaselearnerSpline, BaselearnerCustom, and BaselearnerCustomCpp
- **Loss functions**: LossQuadratic, LossAbsolute, LossQuantile, LossHuber, LossBinomial, LossCustom, and LossCustomCpp
- **Logger/Stopper**: LoggerIteration, LoggerInbagRisk, LoggerOobRisk, and LoggerTime

  → Performance-based early stopping can be applied using the LoggerOobRisk and specifying the relative improvement that should be reached (e.g. 0 for stopping when out of bag risk starts to increase).
Performance Considerations
- Optimizer are parallelized via openmp:

- Take advantage of the matrix structure to speed up the algorithm by reducing the number of repetitive or too expensive calculations.

- Matrices are stored (if possible) as a sparse matrix.
Small Comparison With Mboost

- **Runtime (in minutes):**

<table>
<thead>
<tr>
<th>nrows / ncols</th>
<th>mboost</th>
<th>compboost</th>
<th>compboost (16 threads)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20000 / 200</td>
<td>21.10 (1)</td>
<td>10.47 (2.02)</td>
<td>0.95 (22.21)</td>
</tr>
<tr>
<td>20000 / 2000</td>
<td>216.70 (1)</td>
<td>83.95 (2.58)</td>
<td>8.15 (26.59)</td>
</tr>
</tbody>
</table>

- **Memory (in GB):**

<table>
<thead>
<tr>
<th>nrows / ncols</th>
<th>mboost</th>
<th>compboost</th>
<th>compboost (16 threads)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20000 / 200</td>
<td>1.04 (1)</td>
<td>0.28 (3.71)</td>
<td>0.30 (3.47)</td>
</tr>
<tr>
<td>20000 / 2000</td>
<td>8.70 (1)</td>
<td>2.60 (3.35)</td>
<td>2.98 (2.92)</td>
</tr>
</tbody>
</table>

(Comparison was made by just using spline base-learner with 20 knots and 5000 iterations. The numbers in the brackets are the relative values compared to mboost.)
What’s Next?
What’s Next?

- Research on computational aspects of the algorithm:
  - More stable base-learner selection process via resampling
  - Base-learner selection for arbitrary performance measures
  - Smarter and faster optimizers
- Greater functionality:
  - Functional data structures and loss functions
  - Unbiased feature selection
  - Effect decomposition into constant, linear, and non-linear
- Reducing the memory load by applying binning on numerical features.
- Adding hyperparameter tuning by providing a mlr (mlr3) learner API.
- Exposing C++ classes to python.
• Slides are available at:

  www.github.com/schalkdaniel/talk_compboost_useR

• Actively developed on GitHub:

  www.github.com/schalkdaniel/compboost

• Project page:

  www.compboost.org

• JOSS DOI:

  10.21105/joss.00967