

compboost

Fast and Flexible Way of bla

Daniel Schalk

July 2, 2019

LMU Munich

Working Group Computational Statistics



Use-Case

- We own a small booth at the Christmas market that sells mulled wine.
- As we are very interested in our customers' health, we only sell to customers who we expect to drink less than 15 liters per season.
- To estimate how much a customer drinks, we have collected data from 200 customers in recent years.
- These data include mulled wine consumption (in liter and cups), age, sex, country of origin, weight, body size, and 200 characteristics gained from app usage (that have absolutely no influence).

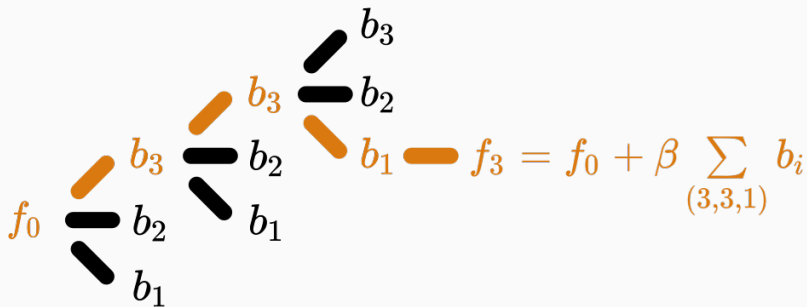
beer_consumption	gender	country	age	weight	height	app_usage1	app_usage2
106.5	m	Seychelles	33	87.17	172.9	0.1680	0.6063
85.5	f	Seychelles	52	89.38	200.4	0.8075	0.9376
116.5	f	Czechia	54	92.03	178.7	0.3849	0.2644
67.0	m	Australia	32	63.53	186.3	0.3277	0.3801
43.0	f	Australia	51	64.73	175.0	0.6021	0.8075
85.0	m	Austria	43	95.74	173.2	0.6044	0.9781
79.0	f	Austria	55	87.65	156.3	0.1246	0.9579
107.0	f	Austria	24	93.17	161.4	0.2946	0.7627
57.0	m	USA	55	76.27	182.5	0.5776	0.5096
89.0	m	USA	16	72.21	203.3	0.6310	0.0645

With these 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?

The General Idea



- 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.

The Idea Behind Compboost

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` 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.

A Short Demonstration

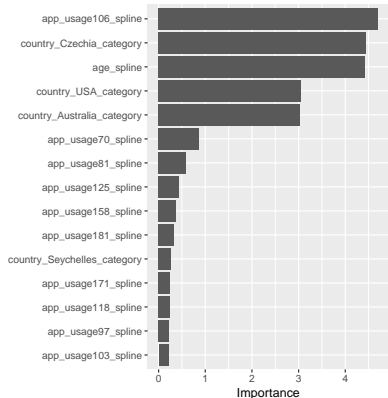
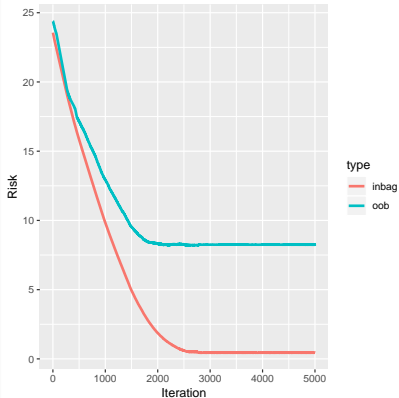
Using Convenience Wrapper

```
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 18 Seconds.
## Final risk based on the train set: 0.44
```

Visualizing Results

```
gg1 = cboost$plotInbagVsOobRisk()  
gg2 = cboost$plotFeatureImportance()  
  
gridExtra::grid.arrange(gg1, gg2, ncol = 2L)
```

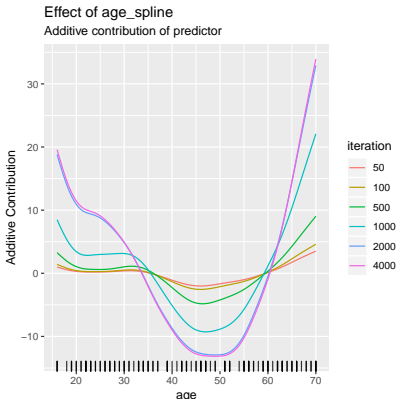
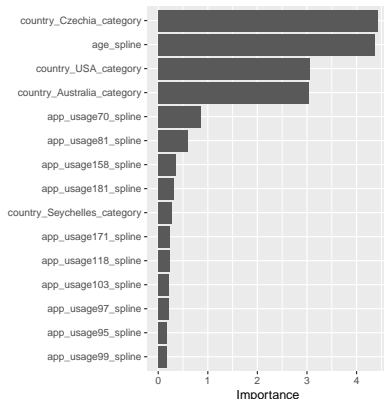


Visualizing Results

```
cboost$train(200L)

gg1 = cboost$plotFeatureImportance()
gg2 = cboost$plot("age_spline", iters = c(50, 100, 500, 1000, 2000, 4000))

gridExtra::grid.arrange(gg1, gg2, ncol = 2L)
```



More Advanced Customizations

- Custom loss function and base-learner
- Advanced stopper for early stopping (e.g. time or performance based stopping)
- Parallelization via openmp is controlled by the optimizer, e.g.
`OptimizerCoordinateDescent$new(4L)`

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 optimizer to select base-learner
- Greater functionality:
 - Functional data structures and loss functions
 - Unbiased feature selection
 - Effect decomposition into constant, linear, and non-linear
- Python API

Questions?