

# ASP20 Boost

## Second Report

Johannes Strauß

Levin Wiebelt

Sebastian Aristizabal

24 Juli 2020

# Contents

<b>1. Introduction</b>	<b>2</b>
<b>2. Code Updates</b>	<b>3</b>
Cholesky decomposition . . . . .	3
Automated Unit Tests . . . . .	3
API design . . . . .	3
<b>3. Simulation Studies</b>	<b>4</b>
Two-dimensional case . . . . .	4
Multidimensional case . . . . .	4
Conclusions . . . . .	5
<b>4. Further Tasks</b>	<b>6</b>
Ensure functionality of gradient-boost algorithm . . . . .	6
Choice of initial parameter values . . . . .	6
<b>5. References</b>	<b>7</b>

# 1. Introduction

Following up on the first report from June 5th 2020, this document describes the further progress of developing the R-package **asp20boost**. In part 2, we describe the changes in design and new features since completion of the first report, these include amongst others automated unit tests and the improvement of matrix calculations via Cholesky-decomposition. To assess the performance of the gradient-boost algorithm, we conducted several simulation studies. The results are described in part 3. Finally, in part 4, we outline remaining challenges.

## 2. Code Updates

### Cholesky decomposition

Following the issue raised by Hannes Riebl in our Gitlab-repository, we implemented a Cholesky decomposition in the matrix calculations of the `LocationScaleRegressionBoost` class. As described in the first report the idea of boosting consists in the iterative estimation of

- Residuals in the case of location-estimation
- The gradient (wrt. scale parameters) of the loss-function in the case of scale estimation.

In both cases this is done by the least squares procedure  $(X'X)^{-1}X'u$  (Fahrmeir et al. 2013, 226). If repeatedly executed, this calculation may be improved by decomposing the symmetric positive definite matrix  $X'X$  via Cholesky's procedure. The inverse then is calculated rapidly by the R-command `chol2inv()`.

### Automated Unit Tests

To automate the testing process of our package we began with the implementation of unit tests. Up to now, we automated the check for correct model class creation. In order to test the boosting algorithm with simulated data, external R-scripts seem more adequate.

### API design

Componentwise boosting demands selection of the component, which is to be updated. Both this selection and the ensuing calculation of the update are done by the fields `bestFittingVariablebeta` and `bestFittingVariableGamma`. We decided to move these calculations from active fields to public fields of the `LocationScaleRegressionBoost` class. This is motivated by the fact that a substantial amount of calculations happen in these fields, while active fields should be of low computational cost. Hence public fields are more accurate.

## 3. Simulation Studies

In order to assess the functionality of the `gradient_boost` function, we conducted several simulation studies which may be categorized into two sets. The first set is implemented in the file *simulation\_plot\_4\_params.R*. It sets up a simple location-scale model with two parameters  $\beta$  for location, as well as two parameters  $\gamma$  for scale. The algorithm does not yet work properly for all possible parameter combinations of  $\beta$  and  $\gamma$ . The second set is implemented in the file *simulation\_variable\_selection.R*. It aims to check if variable selection works correctly for multidimensional ( $\dim > 2$ )  $\beta$  or  $\gamma$ . Results also depend on the concrete parameter configuration.

### Two-dimensional case

Setting up the model identically to the example in the `asp20model` repository leads to algorithm convergence towards the true values  $\beta = (0, 1)'$  and  $\gamma = (-3, 2)'$  (Riebl 2020).

Let the true model be characterized by the following parameter configuration:  $\beta = (1, 1)'$  and  $\gamma = (1, 1)'$ . After three iterations, the algorithm yields estimates of  $\hat{\beta} = (0.01, 0.05)'$  and  $\hat{\gamma} = (12.30, 0.00)'$ . The algorithm detects no further change in gradients and stops prematurely. By lowering the `abstol` parameter functionality may be restored. After 3412 iterations estimates are  $\hat{\beta} = (0.80, 1.44)'$  and  $\hat{\gamma} = (0.96, 1.12)'$ . These values are not yet good estimates but reasonably close to conclude basic functionality of the boosting algorithm.

In the case of parameter configuration :  $\beta = (2, 2)'$  and  $\gamma = (2, 2)'$  early stopping remains as a problem. Functionality can not be achieved by lowering the tolerance threshold `abstol` of the break condition. The break condition is a comparison of gradients of successive iterations.

### Multidimensional case

In the simulation studies for multidimensional models, a total of six parameters for  $\beta$  as well as six for  $\gamma$  may be specified.

#### Simple Case

A simple case consists in setting intercept and one slope for  $\beta$  and  $\gamma$  to one while the remaining  $2 \times 4$  parameters are zero:  $\beta = (1, 1, 0, 0, 0, 0)'$  and  $\gamma = (1, 1, 0, 0, 0, 0)'$ . The algorithm should depict the  $2 \times 2$  parameters of interest and estimate them close to one while the others should remain zero. The resulting estimates displayed below indicate proper estimates for the scale parameter vector  $\hat{\gamma}$  but not for the location estimates  $\hat{\beta}$ .

```
model$beta
[1] 0.15122097 1.72916260 0.61363927 -0.14620695 0.02756381 0.28560006
```

```

> model$gamma
[1] 0.95706770 1.15196564 -0.02756374 0.00000000 -0.06839582 0.00000000

model$beta
[1] 0.15122097 1.72916260 0.61363927 -0.14620695 0.02756381 0.28560006
model$gamma
[1] 0.95706770 1.15196564 -0.02756374 0.00000000 -0.06839582 0.00000000

```

## Example Case

Setting up the example model from the `asp20model` repository, the algorithm, in turn, finds correct estimates being however somewhat inaccurate for  $\hat{\gamma}_1$  (Riebl 2020):

```

model$beta
[1] -0.0148539032 1.0130172152 0.0108580428 -0.0026268607 0.0001592711
[6] 0.0049089286
model$gamma
[1] -3.103392229 1.258857920 0.017474511 -0.005902031 -0.092450456
[6] 0.000000000

```

## 1-0-Shifted Case

Another model setup  $\beta = (1, 0, 1, 0, 1, 0)'$  and  $\gamma = (0, 1, 0, 1, 0, 1)'$  yields again correct estimates for  $\gamma$ , but not for  $\beta$ :

```

model$beta
[1] 0.69898406 0.53163733 1.64776999 -0.92116479 1.10914562 -0.02877867
model$gamma
[1] 0.01368480 1.17157405 -0.03057068 0.94892465 -0.03256148 0.89579817

```

## Conclusions

From these findings we conclude that the evolution of gradients during iterations needs to be checked. We need to find out why gradients converge without yielding sufficiently adequate parameter estimates. At the moment component-wise boosting is hardwired into our code. A useful implementation would be to allow for non-componentwise boosting again. Being able to switch between overall and componentwise boosting may facilitate locating weak points in our code.

## 4. Further Tasks

### Ensure functionality of gradient-boost algorithm

As depicted in part 3, functionality of our algorithm needs to be ensured independent of the concrete data-generating process.

### Choice of initial parameter values

The performance of the boosting algorithm may be enhanced if initial estimates are chosen sensibly. For the location parameters  $\beta$  this may be the OLS-estimate or the mean response. For the scale parameters  $\gamma$  this may be the (homoskedastic) sample variance. Another possibility is to ask other student groups for straight-forward and fast numerical procedures to determine good starting values. However, this may be costly in terms of computation time and hence slow down overall performance of the algorithm. Trade-offs need to be considered.

## 5. References

- Fahrmeir, Ludwig, Thomas. Kneib, Stefan Lang, and Brian Marx. 2013. *Regression: Models, Methods and Applications*. New York: Springer.
- Riebl, Hannes. 2020. *Asp20model: An R6 Class for Location-Scale Regression Models*.