Report ASP20 Boost

First Report

Johannes Strauß Levin Wiebelt Sebastian Aristizabal

05 Juni 2020

Contents

1. Introduction:

The course "Advanced Statistical Programming with R" consists in implementing a package to address location-scale regression by building upon an R6 class (Chang 2019) given to us as part of the `asp20model` repository (Riebl 2020). The sample dataset consists of a response y, i.e, the expectation is in linear relationship to a set of predictors x_i. Its variance, in contrast, is dependent on a different set of linear predictors z_i, however transformed by the response function $g(x) = exp(x)$. This ensures positiveness of the variances. The goal is to estimate the effects of both: x_i on the expectation of y (location) - this is captured by the estimates β, and z_i on the variance of y (scale) - this is captured by the estimates γ. Our student group `asp20boost` solves this task applying the concept of boosting.

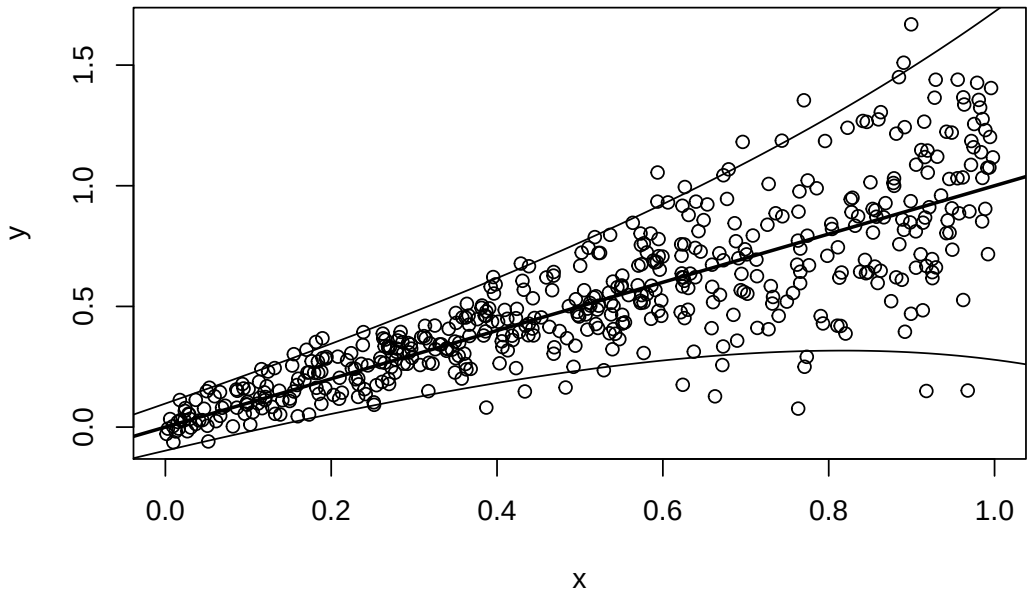

Figure 1: A visual example of Heteroskedasticity. Source: asp20model

In *part 2* of this report we explain the concept of boosting and elaborate on how it suites to address the problem of location-scale regression. In *part 3* we describe our implementation milestones, whereby *part 4* especially elaborates on componentwise boosting. In *part 5* we present our thoughts on upcoming challenges and critical aspects of the package development.

2. Concept: Boosting

The first step in our progress was to understand the theoretical concept of boosting and which types of problems it is able to solve. To explain the concept and follow our process of understanding we first explain boosting for location parameters. Boosting scale parameters required another way of thinking about it and constituted a first milestone in our understanding and implementation. In the following, the term "booster" refers to a boosting algorithm.

Location-Booster

The following description of simple boosting algorithm is mainly based on Fahrmeir et al. (2013), 217-219.

The boosting concept relies on the idea of iterative estimation. To estimate the location parameters β via boosting, one starts with initial estimates, which may be far from optimal. However, with estimates and the response vector at hand, residuals can be calculated. The boosting algorithm focuses on those calculated residuals. The effect of the predictors x_i on these residuals is estimated using least squares and the resulting effect is added to the location-estimate β - adjusted by the learning rate ν. This yields a better fit of the location-model, which results in smaller residuals. These new residuals are estimated again in the next iteration, yielding smaller effect sizes, and hence a convergence of the boosting algorithm towards the OLS-estimate.

A simple booster stops after a fixed number of iterations is reached.

$$u = y - X\hat{\beta}^{(t-1)} \tag{1}$$

Scale-Booster

A boosting algorithm for scale parameters γ consists in the same two repeating steps of first estimating a term, and second updating γ by adding the estimated effect - adjusted by a learning rate. The term in the estimating step is, however, not the residual. For boosting γ this term equals the score function of the model i.e the derivative of the log likelihood of the normal distribution.

$$ll = \frac{1}{2\pi \exp(Z\Gamma)^2} \times \exp(-\frac{u^2}{2 \exp(Z\Gamma)^2}) \tag{2}$$

3. Implementation

To implement a boosting algorithm we constructed the function `gradient_boost` as a part of our R6-Class `LocationScaleRegressionBoost`, which directly builds upon the `LocationScaleRegression` R6 class of the `asp20model`package set at our disposal for this course and this purpose specifically (Riebl 2020).

In the current version of the `asp20boost`-package, the boosting algorithm is already implemented in order to allow for componentwise boosting, which will be explained in part 4.

To further provide clarity, accountability and understanding during to our development process we describe briefly our first implementation of an ordinary boosting algorithm. This isn't visible in our code anymore.

A simple booster for the location parameters β works without extending the `asp20model`'s `LocationScaleRegression` class. The current residuals may be extracted with the `resid()` command and then estimated. The location parameter is updated. This causes the `LocationScaleRegression` class to calculate updated state-dependent objects, such as the log likelihood, the gradients and the residuals. These objects are again used to repeatedly estimate residuals and update the location parameter β.

The simple booster for the scale parameters γ, as already mentioned, consists in repeating the same two steps of first estimating a term and then updating the scale parameters γ by adding the estimated effect - adjusted by the learning rate ν. The estimated term here is the derivative of the log likelihood of the normal distribution. To calculate this, our code makes use of the `resid()`-function, but this time passing the argument "deviance", which results in residuals adjusted by the fitted scale estimates.

This simple implementation of a boosting algorithm for location and scale served the code-basis or "skeleton" for our package and can be thought of as the first milestone of this project.

4. Functionality: Componentwise Boosting

A useful extension of our simple boosting algorithm regards *componentwise boosting*. The key idea here is to not update a whole parameter vector, but only one entry of it. The entry chosen for the update at this juncture is the one yielding the best improvement in terms of minimization of the loss function. The loss function in our case is the deviance calculated using γ and our β.

Hence, in each iteration only one component of the location- and one component of the scale-parameters are updated.

1. Beta

$$\begin{aligned} \beta_j^{(0)} &= 0, \ \text{for} j = 1, .., k. \\ u_i &= y - X\widehat{\beta}^{(t-1)} \\ \widehat{b}_j &= ((\vec{x}^j)^T \vec{x}^j)^{-1}((\vec{x}^j)^T \vec{u} \end{aligned} \tag{3}$$

2. Gamma

$$\begin{aligned} \gamma_j^{(0)} &= 0, \ \text{for} j = 1, .., k. \\ u_i &= \text{deviance} \\ \widehat{b}_j &= U^2 Z^2 e^{-2(Z\Gamma)} \end{aligned} \tag{4}$$

3. Same procedure.

Fit separate models for all covariates i.e obtain

$$\widehat{b}_j = ((\vec{x}^j)^T \vec{x}^j)^{-1}((\vec{x}^j)^T \vec{u}, \ \text{for} j = 1, .., k. \tag{5}$$

And determine the best fitting variable via

$$j^* = \arg \min_{j=0,..,k} \sum_{i=1}^{n} (u_i - x_{ij}\widehat{b}_j)^2. \tag{6}$$

4. Update Coefficients.

$$\begin{aligned} \widehat{\beta}_{j^*}^{(1)} &= \widehat{\beta}_{j^*}^{(0)} + \nu \widehat{b}_{j^*} \\ \widehat{\beta}_{j}^{(1)} &= \widehat{\beta}_{j}^{(0)}, j \neq j^*. \end{aligned} \tag{7}$$

We implement this in our package by extending the `LocationScaleRegression` class to include the two active fields `bestFittingVariableBeta` and `bestFittingVariableGamma`.

These functions partition the design matrix X- respectively Z - into its single columns and then estimate the residuals - respectively the scores - separately for each component, and determine loss functions for a hypothetical update with the respective component. Respective to the old loss function value, the highest loss-improvement is determined and the respective component is used to update the parameter-vector.

We are in the process of reconsidering the design of this implementation. Other design possibilities are the following:

- Create public fields for the heavily used score-function values.
- Move the calculation of componentwise losses to an external function.
- Harmonize boosting and componentwise boosting into one external function, determining the mode of operation by an argument `componentwise = TRUE`.

Another conceptual questions that comes up is if the best loss-improvement may be indicated by the already existing gradients, and hence there is no need for extra calculation.

5. Prospects

Further Functionalities

The most important functionality we intend to implement in our package is the optimization of the number of iterations via cross validation. To work out the theoretical concept behind this idea is, as well as to implementing it are our current next goals. One further, rather loose, idea is to find ways to optimize the learning rate in our boosting algorithm.

Stability

We intend to implement further automated unit tests. This will enable us to assess quickly the stability of our code given differing inputs. We also intend to implement measures for proneness reduction to input errors.

Performance

- A major performance problem in our code is that small learning rates for the β-booster lead to a sharp increase of our code's processing time. This remains to be solved.
- A good choice of starting values for β and γ may enhance performance. A possible candidate is the mean of the response.

Further Aspects

- Allow user-input of the LocationScaleRegressionBoost-model in form of a data frame, for example by allowing to pass an optional "data"-argument as known from the `lm-call`. This may reduce proneness to input-errors.
- Visualize the results of the boosted estimates
- Document the extension of the R6-Class properly and inherit documentation of the `LocationScaleRegression-Class` using roxygen2.

6. References

Chang, Winston. 2019. *R6: Encapsulated Classes with Reference Semantics*. https://CRAN.R-project.org/package=R6.

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*.