Report ASP20 Boost

First Report

Johannes Strauß Levin Wiebelt Sebastian Aristizabal

04 Juni 2020

Contents

1. Introduction:

[S: please format all equations and terms - y, xi with a subscripted i, zi (same), ...]

The course "Advanced Statistical Programming with R" consists in implementing a package to address location-scale regression. The sample dataset consists of a response y, that's expectation is in linear relationship to a set of predictors xi. Its variance, in contrast, is dependent on a different set of linear predictors zi, however transformed by the response function g(x) = exp(x). This ensures positiveness of the variances. The goal approach is to estimate the effects of both: xi on the expectation of y (location) - this is captured by the estimates beta, and zi on the variance of y (scale) - this is captured by the estimates gamma. Our student group `asp20boost` solves this task applying the concept of boosting.

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 the following *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

[S: please format all terms - xi, beta and gamma as greek letters, v also as greek letter ...] **[S: please add reference]**

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 requires 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 boosting concept relies on the idea of iterative estimation. To estimate the location parameters beta 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 algortihm focuses on those calculated residuals. The effect of the predictors xi on these residuals is estimated by least squares and the resulting effect is added to the location-estimate beta, adjusted by some learning rate v. 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. **(compare Kneib et al, p217,218)**

Scale-Booster

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

[Why is that? kurze Erklärung]

3. Implementation

The implementation of the boosting algorithm is accomplished by a function *gradient_boost*, which harmonizes with the R6-Class *LocationScaleRegressionBoost*. This class, in turn, is built upon the *LocationScaleRegression* class of the `asp20model`-package. It follows a strict dependency of the `asp20boost-` on the `asp20model`-package.

In the current version of the `asp20boost`-package, the boosting algorithm is already implemented in order to allow componentwise boosting, which will be explained in part 4. For good explanation and for following our development process in this part we first describe the implementation of ordinary boosting - which however does not appear in our code anymore.

A simple booster for the location parameters **beta** works without extending the *LocationScaleRegression* class. The current residuals may be extrated with the *resid()*-method, then estimated. The location parameter is updated. This causes the *LocationScaleRegression* class to calculate updated state-dependent objects, such as the loglikelihood, gradients and the residuals. This is what's needed to repeat the procedure of estimating residuals and updating location parameters **Beta**.

The simple booster for the scale parameters **gamma**, as already mentioned, consists in the same two repeating steps of first estimating a term, and second updating the scale parameters gamma by adding the estimated effect - adjusted by a learning rate **v**. The estimated term here is the derivative of the loglikelihood 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.

[**Erklärung weiter ausführen**]

Having implemented a boosting algorithm for location and scale, the code-basis for our package was achieved.

4. Functionality: Componentwise Boosting

A usefull functionality of boosting is componentwise boosting. The idea 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 the sense of lowering the loss function. The loss function in our case is the loglikelihood of the normally distributed data. Hence in each iteration only one component of the location-, as well as one component of the scale-parameters are updated.

The implementation in our package works via the extension of the **LocationScaleRegression** class by 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 - seperately for each component, and determine loss functions for a hypothetical update with the respective component. Comparing with the old loss function value, the highest loss-improvement may be 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 optimizing the stopping point of our booster. This optimization of the number of iterations may be done via cross validation. Working out the theoretical concept behind this idea is one of our next milestones.

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. Departing from this point we intend to reduce proneness to input errors.

Performance

- A major performance problem in our code is that small learning rates for the beta-booster cause processing time of the code to increase sharply. This remains to be solved. *A good choice of starting values for beta and gamma may enhance performance. A possible candidate is the mean of the response.

Further

- Allow user-input of the LocationScaleRegressionBoost-model in form of a dataframe, 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 applying roxygen2.

Application

[Johannes: Munich Rent Data? - Identify Outliers]