The Journal of Open Source Software

wbacon: Weighted BACON algorithms for multivariate outlier nomination (detection) and robust linear regression

Tobias Schoch[1]

1 University of Applied Sciences and Arts Northwestern Switzerland, School of Business, Riggenbachstrasse 16, CH-4600 Olten, Switzerland

DOI: 10.21105/joss.03238

Software

- Review
- Repository
- Archive

Editor: Frederick Boehm

Reviewers:

- @msalibian
- @aalfons

Submitted: 22 April 2021
Published: 06 June 2021

Summary

Outlier nomination (detection) and robust regression are computationally hard problems. This is all the more true when the number of variables and observations grow rapidly. Among all candidate methods, the two BACON (blocked adaptive computationally efficient outlier nominators) algorithms of Billor et al. (2000) have favorable computational characteristics as they require only a few model evaluations irrespective of the sample size. This makes them popular algorithms for multivariate outlier nomination/detection and robust linear regression (at the time of writing Google Scholar reports more than 500 citations of the Billor et al. (2000) paper).

`wbacon` is a package for the `R` statistical software (R Core Team, 2021). It is aimed at medium to large data sets that can possibly have (sampling) weights (e.g., data from complex survey samples). The package has a user-friendly `R` interface (with plotting methods, etc.) and is written mainly in the `C` language (with OpenMP support for parallelization; see OpenMP Architecture Review Board (2018)) for performance reasons.

The BACON algorithms

Technically, the BACON algorithms consist of the application of series of simple statistical estimation methods such as coordinate-wise means/medians, covariance matrix, Mahalanobis distances, or least squares regression on subsets of the data. The algorithms start from an initial small subset of non-outlying ("good") data and keep adding those observations to the subset whose distances (or discrepancies in the case of the regression algorithm) are smaller than a predefined threshold value. The algorithms terminate if the subset cannot be increased further. The observations not in the final subset are nominated as outliers. We follow Billor et al. (2000) and use the term "nomination" of outliers instead of "detection" to emphasize that the algorithms should not go beyond nominating observations as potential outliers. It is left to the analyst to finally label outlying observations as such.

The BACON algorithm for multivariate outlier nomination can be initialized in two ways: version "V1" or "V2" (see Billor et al., 2000). In version V2, the algorithm is started from the coordinate-wise median. As a consequence, the resulting estimators of location and scatter are robust, i.e., the breakdown point[1] is approximately 40% (Billor et al., 2000), but the estimators are not affine equivariant estimators of the population location and scatter. However, Billor et al. (2000) show that the estimators are *nearly* affine equivariant.

[1]Intuitively, the breakdown point of an estimator is the proportion of outliers an estimator can handle before giving a arbitrary or meaningless result; see Maronna et al. (2018) for a rigorous definition.

The initialization by version V1 yields estimators that are affine equivariant by design because the algorithm is started from the coordinate-wise mean, but the estimators have a very low breakdown point.

A naive implementation of the BACON algorithms would call the (simple) estimation methods iteratively on a sequence of growing subsets of the data without bothering too much with reusing or updating existing blocks of data. This leads to an excessively large number of copy/modify operations and (unnecessary) recomputations. Overall, we would end up with a computationally inefficient implementation. For small data sets, the inefficiencies would likely go unnoticed. With large amounts of data, however, the situation is quite different.

Statement of need

The two BACON algorithms are available from the package `robustX` (Maechler, Stahel, et al., 2021) for the `R` statistical software. The BACON algorithm for multivariate outlier nomination has been extended to weighted problems (in the context of survey sampling) and incomplete data by Béguin & Hulliger (2008). The extended method is available from the `R` package `modi` (Hulliger & Sterchi, 2020). Both implementations are not explicitly targeted at large data sets. Our package fills this gap.

What the package offers

The implementation of the `wbacon` package is targeted at medium to large data sets and is mainly implemented in the `C` language. The code depends heavily on the subroutines in the libraries `BLAS` (Blackford et al., 2002) and `LAPACK` (Anderson et al., 1999). If computation time is of great importance, we recommend replacing the reference implementation of the `BLAS` library that ships with `R` by a version that has been adapted to the user's hardware (see e.g., OpenBLAS). The non-`BLAS` components of `wbacon` use multiple threads (if the compiler supports `OpenMP`; see OpenMP Architecture Review Board (2018)) for the computations over the p variables/columns because the computational time complexity is dominated by p. For instance, the time complexity of the BACON algorithm for multivariate outlier nomination is dominated by the complexity of the covariance matrix computation, which is of order $\mathcal{O}(np^2)$, where n denotes the number of observations. The major improvements of `wbacon` (in terms of computation time) over the naive implementation, however, are achieved by using partial sorting (in place of a full sort), reusing computed estimates, and employing an up-/downdating scheme for the Cholesky and QR factorizations. The computational costs of the up-/downdating schemes are far less than recomputing the entire decomposition repeatedly.

Diagnostic tools

The BACON algorithms assume that the underlying model is an appropriate description of the non-outlying observations. More precisely (Billor et al., 2000),

- the outlier nomination method assumes that the "good" data have (roughly) an elliptically contoured distribution (this includes the Gaussian distribution as a special case);
- the regression method assumes that the non-outlying ("good") data are described by a linear (homoscedastic) regression model and that the independent variables (having removed the regression intercept/constant, if there is a constant) follow (roughly) an elliptically contoured distribution.

We recommend that the users examine the data structure of the "good" observations to verify that the assumptions hold. The following quote from the authors of the BACON algorithms should be noted.

> "Although the algorithms will often do something reasonable even when these assumptions are violated, it is hard to say what the results mean." (Billor et al., 2000, p. 289)

The wbacon package provides the analyst with tools to identify potentially outlying observations. For multivariate outlier nomination, the package implements several diagnostic plots. Worth mentioning is the graph which plots the robust (Mahalanobis) distances against the univariate projection of the data that maximizes the separation criterion of Qiu & Joe (2006). This kind of diagnostic graph attempts to separate outlying from non-outlying observations as much as possible; see Willems et al. (2009). It is particularly helpful when the outliers are clustered or show patterns. For robust linear regression, the package offers the standard plotting methods that are available for objects of the class lm. In addition, it implements the plot of the robust distances of the (non-constant) design variables against the standardized residuals. This diagnostic plot been proposed by Rousseeuw & Zomeren (1990). All plotting methods can be displayed as hexagonally binned scatter plots, using the functionality of the hexbin (Carr et al., 2021) package. This option is recommended for large data sets.

Illustration

In this section, we illustrate the use of the BACON algorithm for robust linear regression. Our data are on education expenditures in 50 US states in 1975 (Chatterjee & Hadi, 2006, Chap. 5.7). The data can be loaded from the robustbase (Maechler, Rousseeuw, et al., 2021) package.

```r
library(wbacon)
data(education, package = "robustbase")

names(education)[3:6] <- c("RES", "INC", "YOUNG", "EXP")
head(education)
```

The variables are:

- State: State
- Region: group variable with outcomes: 1=Northeastern, 2=North central, 3=Southern, and 4=Western
- RES: Number of residents per thousand residing in urban areas in 1970
- INC: Per capita personal income in 1973 ($US)
- YOUNG: Number of residents per thousand under 18 years of age in 1974
- EXP: Per capita expenditure on public education in a state ($US), projected for 1975

Our goal is to regress education expenditures (EXP) on the variables RES, INC, and YOUNG. For the BACON robust linear regression algorithm, we have

```r
reg <- wBACON_reg(EXP ~ RES + INC + YOUNG, data = education)

reg
```

```
#> Call:
#> wBACON_reg(formula = EXP ~ RES + INC + YOUNG, data = education)
#>
#> Regression on the subset of 49 out of 50 observations (98%)
#>
#> Coefficients:
#> (Intercept)          RES          INC        YOUNG
#>  -277.57731      0.06679      0.04829      0.88693
```

By default, wBACON_reg() uses the parametrization alpha = 0.05, collect = 4, and version = "V2". These parameters are used to call the wBACON() multivariate outlier nomination/detection algorithm on the design matrix. Then, the same parameters are used to compute the robust linear regression.

To ensure a high breakdown point, version = "V2" should not be changed to "V1" unless you have good reasons to do so. The main "turning knob" to tune the regression algorithm is alpha, which defines the $(1-\alpha)$ quantile $t_{\alpha,\nu}$ of the Student t-distribution with ν degrees of freedom. In fact, the quantile $t_{\alpha/(2r+2),r-p}$ is used as the cutoff value (see Billor et al., 2000), where r and p denote, respectively, the number observations in the set of "good" observations and the number of variables. All observations whose discrepancies (defined as the scaled residuals on the set of "good" observations and the scaled prediction error on the set of "bad" observations) are smaller (in absolute value) than the cutoff value are selected into the subset of "good" data [see document methods.pdf in the folder inst/doc of the source package]. By choosing larger values for alpha (e.g., 0.2), more observations are selected (ceteris paribus) into the subset of "good" data (and vice versa).

The parameter collect specifies the size of the initial subset, which is defined as $m = p \cdot collect$. It should be chosen such that m is considerably smaller than the number of observations n. Otherwise we are at risk of selecting too many "bad" observations into the initial subset, which will eventually bias the regression estimates.

The instance reg is an object of the class wbaconlm. The printed output of wBACON_reg() is identical with the one of the stats::lm() function. In addition, we are told the size of the subset on which the final regression has been computed. The observations not in the subset are considered potential outliers (here 1 out of 50 observations). The summary() method can be used to summarize the estimated model.

```
summary(reg)
```

```
#> Call:
#> wBACON_reg(formula = EXP ~ RES + INC + YOUNG, data = education)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -81.128 -22.154  -7.542  22.542  80.890
#>
#> Coefficients:
#>               Estimate Std. Error t value Pr(>|t|)
#> (Intercept) -277.57731  132.42286  -2.096 0.041724 *
#> RES            0.06679    0.04934   1.354 0.182591
#> INC            0.04829    0.01215   3.976 0.000252 ***
#> YOUNG          0.88693    0.33114   2.678 0.010291 *
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 35.81 on 45 degrees of freedom
#> Multiple R-squared:  0.4967,    Adjusted R-squared:  0.4631
#> F-statistic:  14.8 on 3 and 45 DF,  p-value: 7.653e-07
```

The methods `coef()`, `vcov()`, and `predict()` work exactly the same as their `lm()` counterparts. This is also true for the first three `plot()` types, that is

- `which = 1`: Residuals vs Fitted,
- `which = 2`: Normal Q-Q,
- `which = 3`: Scale-Location.

The plot types `4:6` of `plot.lm()` are not implemented for objects of the class `wbaconlm` because it is not sensible to study the standard regression influence diagnostics in the presence of outliers in the model's design space (leverage observations). Instead, type four (`which = 4`) plots the robust Mahalanobis distances with respect to the non-constant design variables against the standardized residual. This plot has been proposed by Rousseeuw & Zomeren (1990). This plot method is also available in the package `robustbase` (Maechler, Rousseeuw, et al., 2021) for robust regression estimators of the class `lmrob`.

See vignette to learn more about the package.

Benchmarking

We compare our implementation with `robustX::BACON()` in terms of computational time. First, we consider estimating a robust linear regression for a Gaussian mixture distribution, where a proportion of $1-\epsilon$ of the observations on the p independent variables is generated by the Gaussian model, while a proportion of ϵ (the outliers) is generated by a shifted Gaussian distribution. For the outlying observations (i.e., ϵ proportion of the data), the response variable is generated by a regression coefficient which is 10 times larger than the coefficient of the non-outlying observations. We choose $\epsilon = 0.05$; see Appendix for more details on the setup. The number of variables (p) and the number of observations (n) are varied.

For the regression exercise, our setup is intentionally *limited to single-threaded* computations (`n_threads = 1`; no OpenMP parallelization support). It is clear that when the number of variables is large, the parallelized computations are (usually) much faster. Table 1 shows the ratio of average computation time of the two implementation for some configurations of n and p. A ratio > 1.0 (< 1.0) implies that `wBACON_reg()` is faster (slower) than `robustX::BACON()`. The average ratio refers to computation time averaged over repeated benchmarks using the R package `microbenchmark` (Mersmann et al., 2019). It is evident from the results in Table 1 that `wBACON_reg()` is considerably faster than its competitor, e.g., `wBACON_reg()` is on average 4.4 times faster for the setup $p = 5$ and $n = 100$. More importantly, the differences become larger as we increase n or p. The differences in computation time are mainly due to the fact that `wBACON_reg()` updates the regression estimates as the subset of non-outlying observations grows, while `robustX::BACON()` recomputes the estimates at each iteration. When thread-level parallelism is enabled in `wBACON_reg()`, the differences become even larger (for $p \geq 20$ and $n \geq 10^3$).

	$p=5$	$p=10$	$p=20$
$n=10^2$	4.4	5.3	7.1
$n=10^3$	43.9	52.5	55.0

Table 1. Robust linear regression (single thread): Ratio of average computation time. A ratio > 1.0 implies that `wBACON_reg()` is faster than `robustX::BACON()`

In the second benchmark, we study the ratio of average computation time of `wBACON()` vs. `robustX::BACON()` for multivariate outlier nomination; see Table 2. A ratio > 1.0 (< 1.0) implies that `wBACON()` is faster (slower) than `robustX::BACON()`. In this benchmark, `wBACON()` is set up with thread-level and instruction-level parallelization: OpenBLAS in place of the standard `BLAS` library and full OpenMP (OpenMP Architecture Review Board, 2018) support. For ease of simplicity, we use a "plain vanilla" parallelization mode which spawns all available cores/threads.

	$p = 5$	$p = 10$	$p = 20$	$p = 30$	$p = 40$	$p = 50$	$p = 100$	$p = 200$
$n = 10^3$	0.5	1.0	1.2	1.4	2.3	2.7	4.7	9.0
$n = 10^4$	0.9	1.1	1.5	1.7	2.6	3.0	5.0	8.3
$n = 10^5$	0.1	0.6	0.8	1.3	2.0	2.3	4.7	9.9
$n = 10^6$	0.9	1.5	2.2	2.8	4.0	4.1	7.7	13.7

Table 2. Multivariate outlier nomination/detection (multithreading): Ratio of average computation time. A ratio > 1.0 (< 1.0) implies that `wBACON()` is faster (slower) than `robustX::BACON()`

For very small data sets (e.g., $n = 10^3$ and $p = 5$), `wBACON()` is slower because parallelization leads to computation overhead that dominates computation time. Clearly, it would be more efficient to specify only 1 or 2 threads for such small data sets. However, the differences in computation time are hardly noticeable to the user (0.08 vs. 0.14 seconds). For larger data sets (in terms of number of variables and observations), `wBACON()` outperforms `robustX::BACON()`; see Table 2. For instance, `wBACON()` is 13.7 times faster for the setup $p = 200$ and $n = 10^6$. The differences in computation time between the two implementations become larger as we increase n or p.

Acknowledgements

I would like to acknowledge many fruitful discussions with Beat Hulliger. This research did not receive any special grant from funding agencies in the public, commercial, or not-for-profit sectors.

Appendix

Consider the Gaussian mixture distribution $G = (1 - \epsilon) \cdot N(0 \cdot 1_p, I_p) + \epsilon \cdot N(4 \cdot 1_p, I_p)$, where $\epsilon = 0.05$ (amount of contamination), N is the cumulative distribution function of the p-variate Gaussian distribution, I_p and 1_p are, respectively, the $(p \times p)$ identity matrix and the p-vector of ones. We generate the $(\lfloor \epsilon n \rfloor \times p)$ matrix X_{good} of "good" observations from the $N(0 \cdot 1_p, I_p)$ distribution, where n denotes the sample size. The matrix X_{bad} consisting of $\lceil (1 - \epsilon)n \rceil$ "bad" observations is generated from the $N(4 \cdot 1_p, I_p)$ distribution.

For the regression analysis, we generate the vectors of the response variable $y_{good} = X_{good}1_p + e$ and $y_{bad} = X_{bad}(10 \cdot 1_p) + e$, where e is a random error with standard Gaussian distribution. In the simulation, y is regressed on X, where $y = (y_{good}^T, y_{bad}^T)^T$ and $X = (X_{good}^T, X_{bad}^T)^T$.

Computing environment: R version 3.6.3 (x86_64-pc-linux-gnu, 64 bit, Ubuntu 20.04.2 LTS), Intel Core i7-10700K CPU (8 cores, 16 threads), 3.80 GHz base clock.

References

Anderson, E., Bai, Z., Bischof, C., Blackford, L. S., Demmel, J., Dongarra, J., Croz, J. D., Greenhaum, A., Hammarling, S., McKenney, A., & Sorensen, D. (1999). *LAPACK users' guide* (3rd ed.). Society for Industrial; Applied Mathematics (SIAM). https://doi.org/10.1137/1.9780898719604

Béguin, C., & Hulliger, B. (2008). The BACON-EEM algorithm for multivariate outlier detection in incomplete survey data. *Survey Methodology*, *34*, 91–103.

Billor, N., Hadi, A. S., & Vellemann, P. F. (2000). BACON: Blocked adaptive computationally-efficient outlier nominators. *Computational Statistics and Data Analysis*, *34*, 279–298. https://doi.org/10.1016/S0167-9473(99)00101-2

Blackford, L. S., Petitet, A., Pozo, R., Remington, K., Whaley, R. C., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., Heroux, M., Kaufman, L., & Lumsdaine, A. (2002). An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions on Mathematical Software*, *28*, 135–151. https://doi.org/10.1145/567806.567807

Carr, D., Lewin-Koh, N., & Maechler, M. (2021). *Hexbin: Hexagonal binning routines*. R package version 1.28.2. (The package contains copies of lattice functions written by Deepayan Sarkar). https://CRAN.R-project.org/package=hexbin

Chatterjee, S., & Hadi, A. H. (2006). *Regression analysis by example* (4th ed.). John Wiley; Sons. https://doi.org/10.1002/0470055464

Hulliger, B., & Sterchi, M. (2020). *Modi: Multivariate outlier detection and imputation for incomplete survey data*. R package version 0.1-0. https://CRAN.R-project.org/package=modi

Maechler, M., Rousseeuw, P., Croux, C., Todorov, V., Ruckstuhl, A., Salibian-Barrera, M., Verbeke, T., Koller, M., Conceicao, E. L. T., & Palma, M. A. di. (2021). *Robustbase: Basic robust statistics*. R package version 0.93-7. https://CRAN.R-project.org/package=robustbase

Maechler, M., Stahel, W. A., Turner, R., Oetliker, U., & Schoch, T. (2021). *robustX: 'eXtra' / 'eXperimental' functionality for robust statistics*. R package version 1.2.5. https://CRAN.R-project.org/package=robustX

Maronna, R. A., Martin, R. D., Yohai, V. J., & Salibián-Barrera, M. (2018). *Robust statistics: Theory and methods (with R)* (2nd ed.). John Wiley; Sons. https://doi.org/10.1002/9781119214656

Mersmann, O., Beleites, C., Hurling, R., Friedman, A., & Ulrich, J. M. (2019). *Microbenchmark: Accurate timing functions*. R package version 1.4-7. https://CRAN.R-project.org/package=microbenchmark

OpenMP Architecture Review Board. (2018). *OpenMP application program interface*. https://www.openmp.org

Qiu, Q., & Joe, H. (2006). Separation index and partial membership for clustering. *Computational Statistics and Data Analysis*, *50*, 585–603. https://doi.org/10.1016/j.csda.2004.09.009

R Core Team. (2021). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Rousseeuw, P. J., & Zomeren, K. van. (1990). Unmasking multivariate outliers and leverage points. *Journal of the American Statistical Association*, *411*, 633–639. https://doi.org/10.1080/01621459.1990.10474920

Willems, G., Joe, H., & Zamar, R. (2009). Diagnosing multivariate outliers detected by robust estimators. *Journal of Computational and Graphical Statistics*, *18*, 73–91. https://doi.org/10.1198/jcgs.2009.0005