

Fitting Linear Models and Generalized Linear Models with large data sets in R ¹

Marco Enea ²

Dipartimento di Scienze Statistiche e Matematiche “S. Vianelli”, University of Palermo,
e-mail: enea@dssm.unipa.it

Abstract: We present an estimating algorithm to fit linear and generalized linear models not involving the QR decomposition. Some new R functions are presented and discussed. For large data sets, comparisons with respect to the well-known `lm()` and `glm()`, as well as to `biglm()` and `bigglm()` from the package `biglm`, show that the proposed functions speed up computation while preserving numerical stability and accuracy.

Keywords: Regression updating, Methodology and algorithms of statistical computing, Linear regression, Generalized linear regression, Statistical computing, R programming.

1. Introduction

When dealing with large data sets in R, fitting Linear Models (LMs) and Generalized Linear Models (GLMs) via the conventional `lm()` and `glm()` functions may often fail because of the limited availability of memory. One solution is to use the functions `biglm()` and `bigglm()`, from the package `biglm` by Thomas Lumley. They fit LMs and GLMs using an amount of memory which is only $O(p^2)$, where p is the number of covariates. In most cases, fitting a GLM to a large dataset, allocated in R memory, may be slow both with `glm()` and with `bigglm()`. To speed up calculations, one might think of linking R to an optimized BLAS (Basic Linear Algebra Subroutines). Libraries like ATLAS (Whaley *et al.*, 2001) provide an efficient implementation of the standard BLAS, allowing better performance, especially for the multiplication between matrices. However, because in the QR decomposition the multiplication $X'X$ is avoided, the advantage of using an optimized BLAS with `glm()` and `bigglm()` is limited. Thus, we present some functions, `speedlm()`, `update.speedlm()`, `speedglm()` and `shglm()`, wholly written in R, which operate just to fit LMs and GLMs to large data sets by solving the equations $X'X\beta = X'y$. The proposed functions `update.speedlm()` and `shglm()` overcome the known limitation of `biglm()` and `bigglm()` about the presence of factors among the covariates, in which it is required that all their levels must be included in each row-block of data partition. Moreover, the method `summary()` provides some additional information. Besides, we introduce the general algorithm to fit LMs and GLMs to large datasets, then present the functions and finally show some comparisons, also in terms of performance.

¹The present paper is financially supported by the institution for which the author works.

²Address of correspondence: Dipartimento di Scienze Statistiche e Matematiche “S. Vianelli”, Facoltà di Economia, Viale delle Scienze ed. 13 - 90128 Palermo Tel. ++390916626245 - Fax ++39091485726

2. Rationale: updating algorithm

Let $\mathbf{y} = \mathbf{X}\beta + \epsilon$ be the usual regression model, where $\mathbf{X}$ is an $n \times p$ matrix ($p \leq n$), $\mathbf{y}$ a vector of length n and β a vector of length p . In such a model, the least squares estimate of β which minimizes $(\mathbf{y} - \mathbf{X}\beta)'(\mathbf{y} - \mathbf{X}\beta)$ is given by $\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$. However, to estimate β it is computationally preferable to solve directly the equations system $\mathbf{X}'\mathbf{X}\beta = \mathbf{X}'\mathbf{y}$. If $\mathbf{X}$ is too large for the random-access memory a solution to fit a LM is that to use an updating algorithm. Such a regression updating may be done by partitioning the matrix $\mathbf{X}$ in g row-blocks of dimension $n_i \times p$, $i = 1, \dots, g$, with $\sum_{i=1}^g n_i = n$. By indicating with $\mathbf{X}_i$ the i -th row-block, we can write $(\sum_{j=1}^i \mathbf{X}_j'\mathbf{X}_j)\beta_i = \sum_{j=1}^i \mathbf{X}_j'\mathbf{y}_j$, $i = 1, \dots, g$, with $\hat{\beta}_g = \hat{\beta}$. Thus, at the i -th updating, we get an intermediate estimate of the first $\sum_{j=1}^i n_j$ rows of $\mathbf{X}$, keeping in memory an object of size p^2 . But, not to overcome memory limits, $\mathbf{X}_i$ has to be chosen of an opportune size. For GLMs, because in general no explicit solution exists, maximum likelihood estimation can be done by an “iterative weighed least squares” (IWLS) procedure. The general equations system has the form $\mathbf{X}'\mathbf{W}\mathbf{X}\beta^{(m)} = \mathbf{X}'\mathbf{W}\mathbf{z}$, where at the $(m - 1)$ - th iteration the estimate $\hat{\beta}^{(m-1)}$ determines the new values of $\mathbf{W}$ and $\mathbf{z}$ to estimate $\beta^{(m)}$. The equations system has the same form of the normal equations for a linear model but it has to be iteratively solved. As before, the equations system for the IWLS procedure might be written at each iteration as $(\sum_{i=1}^g \mathbf{X}_i'\mathbf{W}_i\mathbf{X}_i)\beta^{(m)} = \sum_{i=1}^g \mathbf{X}_i'\mathbf{W}_i\mathbf{z}_i$. The main difference between the two procedures is that in the last formulation no intermediate estimates are possible and, in addition, $\mathbf{X}_i$ must be re-read at each iteration, with the disadvantage of repeated accesses to the memory (usually the hard disk) that might seriously increase the computation time. Besides, an eigendecomposition of $\mathbf{X}'\mathbf{X}$ may be done to take into account a possible singularity of $\mathbf{X}$.

3. The functions

The updating algorithm is implemented in `update.speedlm()` and `shglm()`. The function `shglm()` can be used for a very large dataset which is usually stored in an external file. The functions `speedlm()` and `speedglm()` are for the fitting of LMs and GLMs to data loaded into the R’s memory, by allowing to speed up the estimation, especially when used jointly with an optimized BLAS. Our functions accomodate themselves when the data matrix is sparse, so the package `Matrix` is also required. Therefore, we describe the arguments of the aforementioned functions.

```
speedlm(formula,data,weights=NULL,offset=NULL,sparse=NULL,set.default=list(),...)
update.speedlm(obj,data,weights=NULL,offset=NULL,sparse=NULL,all.levels=FALSE,
               set.default=list(),...)
speedglm(formula,data,family=gaussian(),weights=NULL,start=NULL,etastart=NULL,
          mustart=NULL,offset=NULL,maxit=25,sparse=NULL,set.default=list(),...)
shglm(formula,data,family=gaussian(),weights.fo=NULL,start=NULL,etastart=NULL,
       mustart=NULL,all.levels=FALSE,chunksize=100,sparse=NULL,offset=NULL,
       maxit=25,set.default=list(),...)
```

Arguments `formula`, `data`, `family`, `start`, `etastart`, `mustart`, `maxit`, `offset`, have the same meaning as in `lm()` and `glm()`. Instead, in `shglm()` the argument `data` must be a function uses connections to a data file, and it will read a number of rows at time equal to `chunksize`, as also shown in the `biglm()`’s help page. If factors are present, and the levels of the factor are the same across all data chunks, it can be set `all.levels` to `TRUE` to speed up the function, otherwise at least two levels are needed for the first

chunk. The argument `sparse` is logical, and it denotes if the data matrix is sparse or not. By default it is `NULL`, so a quick survey for the sample zeroes proportion will be done. Argument `weights` has to be specified as `data$weights`, whereas `weights.fo` must be specified as a formula. Further arguments can be allocated into a list and as arguments to `set.default`. A useful option, which might speed up the computation of $X'X$ for very large matrices, is to set some value to `row.chunk` generally between 75 and 1000, the larger is p the smaller it should be, in `set.default`. That might be useful if an optimized BLAS is not installed. For instance, with my computer, an AMD Athlon 64 X2 Dual-Core processor 3800+ 2 GHz, 1536 MB RAM, Windows XP, R version 2.8.1, on average the time saving is up to 50% with respect to the ordinary `crossprod()` function, .

4. Some example

We report the same example of the `biglm()` help page to show how our function works. Due to space limitations, a part of the summary was deleted.

```
data(trees)
a <- speedlm(log(Volume)~log(Girth)+log(Height),trees[1:11,])
a <- update(a,trees[12:22,]); a <- update(a,trees[23:31,])
summary(a)
```

	coef	se	t	p.value
(Intercept)	-6.631617	0.799790	-8.292	5.057e-09
log(Girth)	1.982650	0.075011	26.432	2.423e-21
log(Height)	1.117123	0.204437	5.464	7.805e-06

```
---
Residual standard error: 0.081386 on 28 degrees of freedom;
observations: 31; R^2: 0.978; adjusted R^2: 0.976;
F-statistic: 613.2 on 2 and 28 df; p-value: 0.
```

With respect to `biglm()`, the report of the `speedlm()`'s `summary()` is more complete and very similar to that of `lm()`. The computation of this additional information could make `speedlm()` a little slower than `biglm()`. With `shglm()` it is not required that all factor levels are present at each chunk, but at least two and only for the first chunk. The file `data1.txt` contains a dataframe of dimension 100×4 with a factor with four levels. We may use the function `make.data()` reported in the `bigglm()`'s help page to open it.

```
da<-make.data("http://dssm.unipa.it/enea/data1.txt", chunksize=30,
  col.names=c("y", "fat1", "x1", "x2"))
b1<-shglm(y~factor(fat1)+x1,weights=~I(x2^2),data=da,family=Gamma(log))
summary(b1)
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	0.2496108	0.1516744	1.6457	0.1031
factor(fat1)2	0.0434504	0.2350394	0.1849	0.8537
factor(fat1)3	-0.4858701	0.2451410	-1.9820	0.0504
factor(fat1)4	0.0683798	0.2264690	0.3019	0.7634
x1	-0.1914939	0.0922088	-2.0767	0.0405

```
---
null df: 99; null deviance: 124.23; residuals df: 95; residuals deviance: 113.36;
# obs.: 100; # non-zero weighted obs.: 100; AIC: 294.1343; log Likelihood: -141.0671;
RSS: 79.7; dispersion: 0.83872; iterations: 10;rank: 5; max tolerance: 6.26e-05.
```

The summary of `shglm()` is more complete than the `bigglm()` one, also reporting the values of AIC and log likelihood. **Table 1** shows a comparison in terms of computation size and mean times among the aforementioned functions. Comparisons among the functions `lm()`, `speedlm()` and `biglm()` for a LM, and `glm()`, `speedglm()` and

`bigglm()` for a GLM, are made over a 100000×50 covariates matrix $\mathbf{X}$. The comparison between the functions `shglm()` and `bigglm()` is made over a 500000×50 matrix. Each time is the mean time of 10 computations. For the LM, a normal response is assumed. For the GLM, a Poisson response is assumed.

Table 1: *Computation mean times and size of LMs and GLMs for some R functions.*

Function	n	p	Size (Mb)		Ordinary BLAS			Optimized BLAS		
			X	object	user	system	elapsed	user	system	elapsed
lm	100000	50	38.15	90.86	6.67	0.57	7.25	6.32	0.51	6.83
speedlm	100000	50	38.15	0.08	3.08	0.12	3.21	2.28	0.15	2.46
speedlm*	100000	50	38.15	0.08	2.63	0.21	2.86	2.46	0.18	2.63
biglm	100000	50	38.15	0.04	2.77	0.14	2.91	2.61	0.210	2.82
glm	100000	50	38.15	146.60	24.42	0.82	25.25	23.29	0.80	24.16
speedglm	100000	50	38.15	0.06	9.27	0.06	9.48	4.57	0.18	4.77
speedglm*	100000	50	38.15	0.06	6.50	0.22	6.75	5.78	0.51	6.29
bigglm	100000	50	38.15	0.05	15.38	0.02	15.46	15.44	0.004	15.49
bigglm	500000	50	190.73	0.05	341.19	1.67	343.11	303.87	1.69	308.87
shglm	500000	50	190.73	0.05	359.49	1.69	361.72	300.82	1.69	302.71

**By setting `row.chunk=1000` in `set.default`.*

It is important to stress that the memory allocation for the output of our functions is of the same order as the `biglm()` and `bigglm()` one. When the optimized BLAS ATLAS is used, the computation time of `speedlm()` is a little lower than the `biglm()` one and more than two times lower than `lm()`, the computation time of `speedglm()` is halved, whereas neither `glm()` nor `bigglm()` are able to gain a better performance by using ATLAS. By setting `row.chunk` to 1000, interesting performances are also obtained when ATLAS is not used, especially for `speedglm()`. Note that `shglm()` is slower than `bigglm()` if an optimized BLAS is not used, but the former function offers some advantages in terms of functionality, such as the use of factors without particular care on levels and moreover, when there is singularity, `bigglm()` doesn't work whereas `shglm()` does.

In conclusion, for small datasets, functions `lm()` and `glm()` are preferable for known reasons. For large datasets, the functions `speedlm()` and `speedglm()` compete with `biglm()` and `bigglm()`. For GLMs, one has to distinguish two cases: 1) the dataset is large and entirely loaded in R memory or, 2) the dataset is stored into an external file because too large for R memory. In the first case, the function `speedglm()` speeds up computations. In the second case, `shglm()` solves some problems of `bigglm()`. In addition, a more complete summary is provided for all the functions. Among the advantages of `biglm()` and `bigglm()`, there is the implementation of a robust estimation method of standard errors and the greater numerical stability of QR decomposition.

References

- R Development Core Team (2008) *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0.
- Whaley R.C., Petitet A. and Dongarra J.J. (2001) Automated empirical optimization of software and the ATLAS project, *Parallel Computing*, 27, 1–2, 3–35.