

SDS 385: Homework 1

G. Paulon

September 7, 2016

Problem 1. Linear regression

(A) Let us write the WLS objective (or cost) function

$$l(\beta) = \sum_{i=1}^N \frac{w_i}{2} (y_i - x_i^T \beta)^2$$

in a matricial form. We get

$$\begin{aligned} l(\beta) &= \frac{1}{2} (y - X\beta)^T W (y - X\beta) \\ &= \frac{1}{2} (y^T - (X\beta)^T) W (y - X\beta) \\ &= \frac{1}{2} (y^T W y - y^T W X \beta - (X\beta)^T W y + (X\beta)^T W X \beta) \end{aligned}$$

Let us remark here that the second and the third term of the sum can be collected in one single term. In fact

$$(y^T W X \beta)^T = (X \beta)^T W y,$$

i.e. the first is the transposed of the second but, since they are real numbers, they are also equal. Hence we have

$$l(\beta) = \frac{1}{2} (y^T W y - 2\beta^T X^T W y + \beta^T X^T W X \beta).$$

Since we are trying to minimize a convex function, we can just find its stationary points. To do this, we first need to calculate the gradient of the WLS objective function equalizing it to 0. Thus,

$$\begin{aligned} \nabla l(\hat{\beta}) &= 0 \\ \Leftrightarrow \frac{1}{2} (-2X^T W y + 2X^T W X \hat{\beta}) &= 0 \\ \Leftrightarrow (X^T W X) \hat{\beta} &= X^T W y, \end{aligned}$$

which is the set of desired equations (also known as normal equations).

(B) Solving the linear system requires the matrix $X^T W X$ to be nonsingular (and therefore invertible). The matrix $X^T W X$ is positive definite, and therefore nonsingular, in case X has full rank (i.e. the columns of X are linearly independent). The normal equations can be easily solved by inverting the matrices, which leads to

$$\hat{\beta} = (X^T W X)^{-1} X^T W y.$$

The classical inversion method, however, is not recommended as its computational complexity grows rapidly as the dimension of the matrices increases. Moreover, the inversion of a matrix is not stable numerically speaking, as small perturbations on the original matrix

can produce high differences in the inverse matrices (i.e. the numerical solution does not approach the exact one).

For these reasons, we present here two other approaches that are used to deal with the solution of linear systems: the QR and the Cholesky factorizations.

- The QR factorization consists in finding two matrices Q (orthogonal) and R (upper triangular) such that a matrix A can be written as their product. In our case, we want to write the QR factorization of $W^{\frac{1}{2}}X$, i.e. $W^{\frac{1}{2}}X = QR$. Therefore we get

$$\begin{aligned}(X^T W X) \hat{\beta} &= X^T W y \\ \Leftrightarrow (W^{\frac{1}{2}} X)^T (W^{\frac{1}{2}} X) \hat{\beta} &= (W^{\frac{1}{2}} X)^T W^{\frac{1}{2}} y \\ \Leftrightarrow (QR)^T QR \hat{\beta} &= (QR)^T W^{\frac{1}{2}} y \\ \Leftrightarrow R^T Q^T QR \hat{\beta} &= R^T Q^T W^{\frac{1}{2}} y \\ \Leftrightarrow R \hat{\beta} &= Q^T W^{\frac{1}{2}} y.\end{aligned}$$

This system can be easily solved because R is upper triangular.

In our case, the pseudocode can be summarized as follows.

Algorithm 1 QR solver

- 1: **procedure** MY_QR(X, y, W)
 - 2: find the factors Q, R , s.t. $W^{\frac{1}{2}}X = QR$
 - 3: compute the right-term: $b \leftarrow Q^T W^{\frac{1}{2}} y$
 - 4: solve the linear system: $\hat{\beta} \leftarrow R^{-1} b$
-

- The Cholesky factorization can be found for symmetric and positive definite matrices, and it consists in factorizing the original matrix as $A = RR^T$, where R is a lower triangular matrix.

In our case, the pseudocode can be summarized as follows.

Algorithm 2 Cholesky solver

- 1: **procedure** MY_CHOL(X, y, W)
 - 2: find the factor R , s.t. $X^T W X = RR^T$
 - 3: solve the first linear system: $q \leftarrow R^{-1} X^T W y$
 - 4: solve the second linear system: $\hat{\beta} \leftarrow R^{-T} q$
-

(C) We here compare the three methods used to solve the linear system of the normal equations. Let us point out that in each of the three algorithms we exploited the fact that W is a diagonal matrix, optimizing the matricial product $X^T W$ by simply multiplying the first element of the diagonal of W by the first column of X^T , and so on. Moreover, since the matrix $X^T W X$ is

symmetric, one can exploit this information to calculate it by using R's `crossprod(W1/2X)` function.

The performances of the three algorithms are displayed in Figure 1. Four test cases were analysed: the values of N and P range from a minimum of $(N, P) = (100, 10)$ to a maximum of $(N, P) = (3000, 1000)$. As one can see, Cholesky method is the fastest one. The inversion is slower than Cholesky method and it is also more unstable than every other method, and therefore it has to be avoided. The QR factorization, despite being the slowest method, is preferable when numerical issues can occur. In fact, this method is the most robust one.

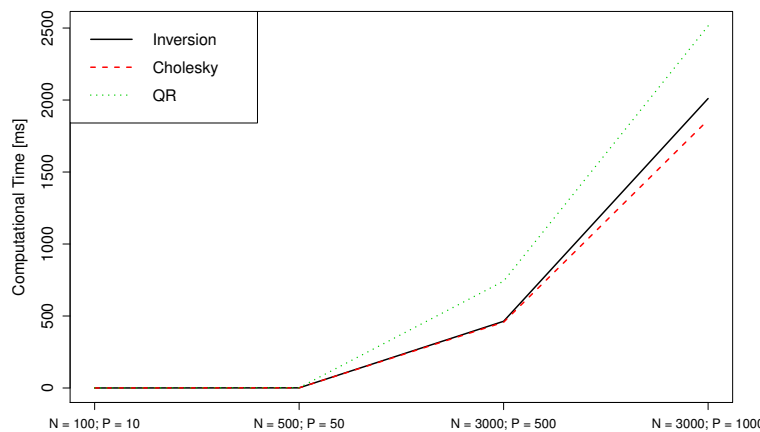


Figure 1: Comparison between the performances of the three algorithms for different dimensions of the data and of their features.

- (D) In the following we use Cholesky method instead of the QR factorization, as the first is faster and now the primary interest is the performance of the algorithms. If X is a highly sparse matrix, one can exploit appropriate routines in R language in order to make the computation faster.

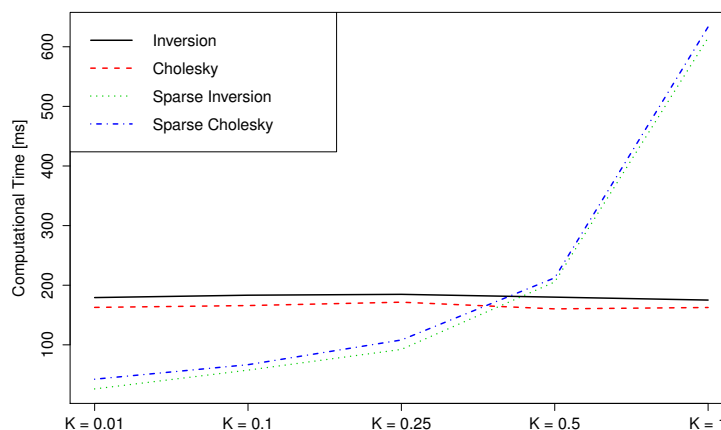


Figure 2: Comparison between the performances of the three algorithms for different sparsity levels K .

Let us take, as an example, $(N, P) = (1000, 500)$ and let us compare the results for different levels of sparsity K of the matrix X . As one can see in Figure 2, when the features matrix is dense, both the Cholesky and the inversion methods outperform the sparse methods, as these latter involve a first stage in which X is converted in a sparse format. However, as K increases, the sparse methods become significantly faster than the others. This is due to the fact that matricial products are more efficient in a sparse format and a lot of useless computations is avoided.

Problem 2. Generalized linear models

(A) The negative log-likelihood for the problem of logistic regression can be written as

$$\begin{aligned} l(\beta) &= -\log \left\{ \prod_{i=1}^N p(y_i | \beta) \right\} \\ &= -\sum_{i=1}^N \left\{ \log \binom{m_i}{y_i} + y_i \log(w_i) + (m_i - y_i) \log(1 - w_i) \right\}. \end{aligned}$$

The gradient of this latter expression is

$$\begin{aligned} \nabla l(\beta) &= -\sum_{i=1}^N \left\{ y_i \frac{1}{w_i} \frac{x_i e^{-x_i^T \beta}}{(1 + e^{-x_i^T \beta})^2} - (m_i - y_i) \frac{1}{1 - w_i} \frac{x_i e^{-x_i^T \beta}}{(1 + e^{-x_i^T \beta})^2} \right\} \\ &= -\sum_{i=1}^N \left\{ y_i \frac{x_i e^{-x_i^T \beta}}{1 + e^{-x_i^T \beta}} - m_i \frac{x_i}{1 + e^{-x_i^T \beta}} + y_i \frac{x_i}{1 + e^{-x_i^T \beta}} \right\} \\ &= -\sum_{i=1}^N \left\{ y_i x_i - m_i \frac{x_i}{1 + e^{-x_i^T \beta}} \right\} \\ &= -\sum_{i=1}^N \{ x_i (y_i - m_i w_i) \} \\ &= -X^T (y - mw). \end{aligned}$$

(B) The gradient descent method has been implemented for a fixed step size $\alpha = 0.01$, which is a sufficiently small value to ensure convergence. As a stopping criterion, we used the relative decrement of the negative log-likelihood. In other terms, we defined the quantity

$$\delta = \frac{l(\beta^{(k)}) - l(\beta^{(k+1)})}{|l(\beta^{(k)}) + \varepsilon|}$$

and the algorithm stops when δ is smaller than a certain threshold (in our case set to 10^{-10}).

We tested the validity of this method on a real dataset containing the classification of a breast cell. We dispose of 10 features for X , and of $n = 569$ data points. In Figure 3 the values of the negative log-likelihood are displayed. The gradient descent converges, in this case, after around 60000 iterations.

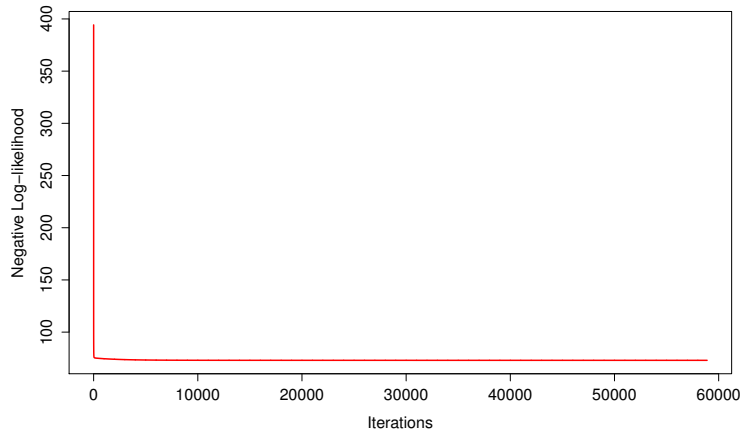


Figure 3: Negative log-likelihood of the data as the gradient descent converges.

- (C) In order to write the second-order Taylor approximation, we first need to compute the Hessian of the negative log-likelihood, that is

$$\begin{aligned}\nabla^2 l(\beta) &= \nabla \left(- \sum_{i=1}^N x_i (y_i - m_i w_i) \right) \\ &= \left(\sum_{i=1}^N \nabla(m_i x_{i1} w_i); \dots; \sum_{i=1}^N \nabla(m_i x_{ip} w_i) \right).\end{aligned}$$

By exploiting the fact that $\frac{\partial w_i}{\partial \beta_j} = x_{ij} w_i (1 - w_i)$, we obtain,

$$\begin{aligned}\nabla^2 l(\beta) &= \begin{pmatrix} \sum_{i=1}^N m_i x_{i1} x_{i1} w_i (1 - w_i) & \dots & \sum_{i=1}^N m_i x_{ip} x_{i1} w_i (1 - w_i) \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^N m_i x_{i1} x_{ip} w_i (1 - w_i) & \dots & \sum_{i=1}^N m_i x_{ip} x_{ip} w_i (1 - w_i) \end{pmatrix} \\ &= X^T W X,\end{aligned}$$

where the matrix W is defined as $W = \text{diag}\{m_1 w_1 (1 - w_1); \dots; m_N w_N (1 - w_N)\}$.

The second order Taylor expansion is then

$$\begin{aligned}\tilde{l}(\beta) &= l(\beta_0) + \nabla l(\beta_0)^T (\beta - \beta_0) + \frac{1}{2} (\beta - \beta_0)^T \nabla^2 l(\beta_0) (\beta - \beta_0) \\ &= l(\beta_0) - (y - mw)^T X (\beta - \beta_0) + \frac{1}{2} (\beta - \beta_0)^T X^T W X (\beta - \beta_0) \\ &= l(\beta_0) - (y - mw)^T X \beta + (y - mw)^T X \beta_0 \\ &\quad + \frac{1}{2} \beta^T X^T W X \beta + \frac{1}{2} \beta_0^T X^T W X \beta_0 - \beta_0^T X^T W X \beta \\ &= l(\beta_0) + (y - mw)^T X \beta_0 + \frac{1}{2} \beta_0^T X^T W X \beta_0 \\ &\quad - (y - mw + W X \beta_0)^T X \beta + \frac{1}{2} (X \beta)^T W (X \beta) \\ &= \frac{1}{2} (z - X \beta)^T W (z - X \beta) + c,\end{aligned}$$

where

$$z = -W^{-1}(y - mw + WX\beta_0) = W^{-1}(mw - y) - X\beta_0;$$

$$c = l(\beta_0) + (y - mw)^T X\beta_0 + \frac{1}{2}\beta_0^T X^T W X\beta_0$$

$$- \frac{1}{2}(y - mw + WX\beta_0)^T W^{-1}(y - mw + WX\beta_0).$$

(D) Newton's method has successfully been implemented. As a stopping criterion, we used the same as before.

We tested the validity of this method on the same real dataset as before. In Figure 4 the values of the negative log-likelihood are displayed. The gradient descent converges, in this case, after 11 iterations. In Listing 1 we can see that the estimates using R's built-in `glm` function and the two methods we implemented are the same.

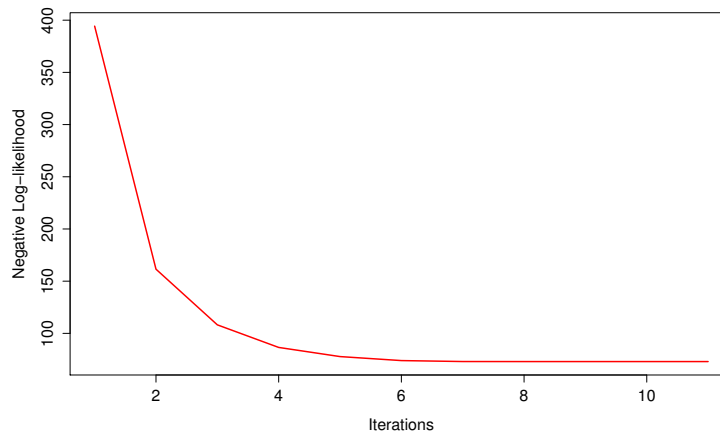


Figure 4: Negative log-likelihood of the data as Newton's method converges.

1		Beta0	Beta1	Beta2	Beta3	Beta4	Beta5	Beta6	Beta7
2	Glm	0.4870168	-7.221851	1.654756	-1.737630	14.00485	1.074953	-0.07723455	0.6751231
3	Gradient	0.4747997	-6.597393	1.655226	-2.311252	13.90645	1.069264	-0.04381625	0.6801968
4	Newton	0.4870168	-7.221851	1.654756	-1.737630	14.00485	1.074953	-0.07723455	0.6751231
5									
6		Beta8	Beta9	Beta10					
7	Glm	2.592874	0.4462563	-0.4824842					
8	Gradient	2.600741	0.4454228	-0.4847644					
9	Newton	2.592874	0.4462563	-0.4824842					

Listing 1: Comparison of the estimates using three different algorithms

(E) As we have seen, Newton's method is significantly faster than gradient descent in terms of the total number of iterations that are needed to reach convergence. However, be aware that each iteration of Newton's method is computationally more intense, as it involves the computation of the Hessian matrix and, moreover, the resolution of a linear system. Overall, Newton's method converges more rapidly as it is able to fully exploit the information

provided by the second derivatives of the function we are trying to minimize, that is, the information concerning the curvature of the objective function. In fact, choosing the descent direction to be simply the vector orthogonal to the level curves can often lead to a slower convergence. The inverse of the Hessian matrix has the role to correct the gradient direction. Newton's method is reliable when the difference between the true function and its quadratic approximation is not too large.