`HDPenReg`: An `R` package for LARS algorithm

Quentin Grimonprez

August 30, 2013

Contents

1 Introduction

Lasso, fused lasso and fusion problems are linear regression problems with l_1 constraints on the estimates.

This package provides a C++ implementation of the LARS which is an algorithm to solve the lasso problem. The storage of the results has been designed for handling the case $n < p$, where n is the number of individuals and p the number of covariates.

In the next section, the lasso problem will be presented as well as the fusion. The LARS algorithm will be detailed in section 3. In section 4, we will present all the functionalities of the package through some examples.

2 Lasso & Fusion

2.1 Lasso

Let $\mathbf{X}$ the matrix of size $n \times p$ of covariates and y, the response vector of length n.

The goal of ordinary least squares (OLS) is to find $\hat{\beta}$ minimizing the total square sum error $S(\hat{\beta}) = \|y - \mathbf{X}\hat{\beta}\|_2^2$.

If we have a lot of covariates, the estimates can be more difficult to interpret, we prefer to have less covariates but the most relevant one. The lasso [Tibshirani, 1994] consists in adding a penalization on the l_1-norm of $\hat{\beta}$ to the OLS problem. The aim of this penalization is to force some coefficients to be exactly zero.

So, the lasso problem is to find $\hat{\beta}$ such that

$$\hat{\beta} = \arg\min_{\beta} \left\{ \|y - \mathbf{X}\beta\|_2^2 \right\} \quad \text{subject to } \|\beta\|_1 \leq t \tag{1}$$

where the bound t is a tuning parameter. For large enough t, we obtain the OLS solution.

We will see later a path algorithm giving solutions for all values of the tuning parameter.

2.2 Fusion

If there is an order in the covariates (e.g. DNA sequence), a penalization can be added to the lasso problem to encourage sparsity in the difference of successive coefficients. This new problem is called the fused lasso[Tibshirani et al., 2005] and is defined by :

$$\hat{\beta} = \arg\min_{\beta} \left\{ \|y - \mathbf{X}\beta\|_2^2 \right\} \quad \text{subject to } \|\beta\|_1 \leq t_1 \text{ and } \sum_{i=2}^{p} |\beta_i - \beta_{i-1}| \leq t_2 \tag{2}$$

where t_1 and t_2 are tuning parameters.

The fused lasso problem is more difficult to solve, but it can be interesting to consider the problem with only the second penalization. This problem is called the fusion.

$$\hat{\beta} = \arg\min_{\beta} \left\{ \|y - \mathbf{X}\beta\|_2^2 \right\} \quad \text{subject to } \sum_{i=2}^{p} |\beta_i - \beta_{i-1}| \leq t_2 \tag{3}$$

The problem can be rewritten as a lasso problem:

Let L a matrix of size $p \times p$

$$L = \begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ -1 & \ddots & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & -1 & 1 & 0 \\ 0 & \cdots & 0 & -1 & 1 \end{pmatrix}.$$

The lasso form of the fusion problem is

$$\hat{\theta} = \arg\min_{\beta} \left\{ \|y - \mathbf{Z}\theta\|_2^2 \right\} \quad \text{subject to } \|\theta\|_1 \leq t_2 \tag{4}$$

with $\theta = L\beta$ and $\mathbf{Z} = \mathbf{X}L^{-1}$.

So, we have just to resolve the lasso problem with $\mathbf{Z}$ and then transform back the estimates to find the path of the fusion problem.

3 LARS algorithm

The LARS (Least Angle Regression) algorithm [Efron et al., 2004] is a stepwise procedure for solving the OLS problem, but with a modification, it can solve the lasso problem.

The principle of the algorithm is to add or drop covariate one-at-a-time in the active set (non zero covariates). At each step, coefficients will be updated in making a step in the equiangular direction of the most correlated covariates until a new covariate is added or dropped.

Algorithm

We want to estimate $\hat{\mu} = X\hat{\beta}$.

0) Let $i = 1$, $\mathcal{A}_0 = \emptyset$ the actual active (non zeros) covariates, $\hat{\mu}_{\mathcal{A}_0} = 0$ and $\hat{c} = X'y$ the current correlation.

Repeat until $|\mathcal{A}| = \min(n, p)$:

1) Compute the greatest absolute current correlation :

$$\hat{C} = \max_j \{|\hat{c}_j|\}.$$

2) Let $\mathcal{A}_i = \left\{ j : |c_j| = \hat{C} \right\}$, the active set.

3) Define

$$X_{\mathcal{A}_i} = (\dots \operatorname{sign}(\hat{c}_j)\mathbf{x}_j \dots)_{j\in\mathcal{A}_i}.$$

We compute

$$\mathcal{G}_{\mathcal{A}_i} = X'_{\mathcal{A}_i} X_{\mathcal{A}_i}$$

and

$$A_{\mathcal{A}_i} = (\mathbf{1}'_{\mathcal{A}_i} \mathcal{G}^{-1}_{\mathcal{A}_i} \mathbf{1}_{\mathcal{A}_i})^{-1/2}$$

where $\mathbf{1}_{\mathcal{A}_i}$is a vector of 1 of length $|\mathcal{A}_i|$.

4) Compute the equiangular vector (unit vector making equal angles with the columns of $X_{\mathcal{A}_i}$)

$$\mathbf{u}_{\mathcal{A}_i} = X_{\mathcal{A}_i}\omega_{\mathcal{A}_i}$$

where $\omega_{\mathcal{A}_i} = A_{\mathcal{A}_i}\mathcal{G}^{-1}_{\mathcal{A}_i}\mathbf{1}_{\mathcal{A}_i}$.

5) Compute $\mathbf{a} = X'\mathbf{u}_{\mathcal{A}_i}$.

6) Update of the coefficient :

$$\hat{\mu}_{\mathcal{A}_{i+1}} = \hat{\mu}_{\mathcal{A}_i} + \gamma\ \mathbf{u}_{\mathcal{A}_i}$$

How to choose the value of γ?

Compute

$$\hat{\gamma} = \min^{+}_{j\in\mathcal{A}^c_i} \left\{ \frac{\hat{C} - \hat{c}_j}{A_{\mathcal{A}_i} - a_j}, \frac{\hat{C} + \hat{c}_j}{A_{\mathcal{A}_i} + a_j} \right\}.$$

$\hat{\gamma}$ is the smallest positive value such that a new index joins the active set.

For the OLS problem, we use $\gamma = \hat{\gamma}$, but a lasso solution has a restriction, each non-zero coefficient of $\hat{\beta}_j$ must have the same sign as that of the current correlation $\hat{c}_j$.

So, we want to find this value called $\tilde{\gamma}$.

Let $\hat{\mathbf{d}} = \operatorname{sign}(\hat{c})_{\mathcal{A}_i}\omega_{\mathcal{A}_i}$.

and

$$\gamma_j = -\frac{\hat{\beta}_j}{\hat{d}_j}.$$

Compute

$$\tilde{\gamma} = \min_{\gamma_j>0} \{\gamma_j\},$$

and let $\tilde{j}$ the index such that $\tilde{\gamma} = \gamma_{\tilde{j}}$.

If $\tilde{\gamma} < \hat{\gamma}$, then

$$\hat{\mu}_{\mathcal{A}_{i+1}} = \hat{\mu}_{\mathcal{A}_i} + \tilde{\gamma}\ \mathbf{u}_{\mathcal{A}_i} \quad \text{and} \quad \mathcal{A}_{i+1} = \mathcal{A}_i - \{\tilde{j}\}$$

and skip the next step 1).

Else

$$\hat{\mu}_{\mathcal{A}_{i+1}} = \hat{\mu}_{\mathcal{A}_i} + \hat{\gamma}\ \mathbf{u}_{\mathcal{A}_i}.$$

7) Compute the current correlation

$$\hat{c} = X'(y - \hat{\mu}_{\mathcal{A}_{i+1}}).$$

and $i = i + 1$.

4 HDPenReg package

4.1 LarsPath class

The `HDlars` and `HDfusion` functions return a `LarsPath` S4 object. This object contains:

variable	A list of size step+1 of vectors. The i-th element contains the vector of indices of active covariates at the step $i-1$.
coefficient	A list of size step+1 of vectors. The i-th element contains the vector of the estimates for active covariates at the step $i-1$.
mu	Intercept.
meanX	Mean of columns of X.
lambda	Vector containing the penalty parameter at each step.
l1norm	Vector containing $\|\hat{\beta}\|_1$ at each step.
addIndex	Vector of length "step" containing the index of the dropped variable at the each step, 0 means no variable has been dropped at this step.
dropIndex	Vector of length "step" containing the index of the added variable at each step, , 0 means no variable has been added at this step.
nbStep	Number of steps of the algorithm.
ignored	A vector containing indices of ignored covariates during the algorithm.
p	Total number of covariates.
fusion	`TRUE` if you have run the `HDfusion`.
error	Error message from lars.

With `LarsPath` objects, you can use following methods :

- `plot` To see the path of the LARS algorithm.
- `plotCoefficient` To plot the values of estimates at a given step.
- `coeff` To extract in a vector the value of all estimates at a given step.

4.2 `simul`

This function simulates copy-number signals in genomic but it's not based on any biological assumptions.

This function takes in following arguments:

n	Number of individuals.
nbSNP	Number of covariates.
probCas	Probability for an individual to be a case.
nbSeg	Number of causal segments.
meanSegmentsize	Mean of segments.
prob	A 2×2 matrix containing probabilities for covariates to create an anomaly. prob= $\begin{pmatrix} prob_{11} & prob_{12} \\ prob_{21} & prob_{22} \end{pmatrix}$ $prob_{11} = \mathbb{P}[$ anomaly $\mid$ causal covariates and control individual] $prob_{21} = \mathbb{P}[$ anomaly $\mid$ causal covariates and case individual] $prob_{12} = \mathbb{P}[$ anomaly $\mid$ no causal covariates and control individual] $prob_{22} = \mathbb{P}[$ anomaly $\mid$ no causal covariates and case individual]
alpha	Parameter of the beta law $\mathcal{B}eta(alpha, alpha)$. This parameter allows to control the variance of the signal.

This function will generate n signals composed of noised segments. A "normal" segment will have a mean of 2, whereas a segment with an anomaly will have a mean of 1 or 3.

The simulation process can be defined as follows:
First, we simulate the response y_i, $i = 1, \ldots, n$, $y_i \sim \mathcal{B}(probCas)$ and randomly choose the centers of causal segments and their mean values (1 or 3 corresponding to a loss or a gain).

Then for an individual i, at each covariate, we use a binomial law with parameters p_{jk} ($j = 1$ for control individual, and $j = 2$ for case individual, $k = 1$ for causal covariates and $k = 2$ for non-causal covariate) to determine if there is an anomaly at the covariate.

Then, for each anomaly, we generate its size λ, $\lambda \sim \mathcal{P}(meanSegmentSize)$.

Finally, we simulate the signal with a beta law $m + \mathcal{B}eta(alpha, alpha)$ where m is a constant representing the status of the segment (gain, normal or loss).

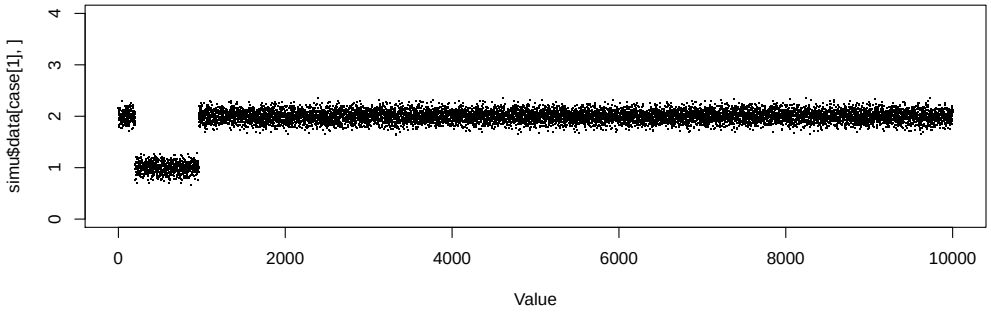


Figure 1: Example of a simulated signal.

The function returns a list of 3 objects:

data	Matrix of size $n \times p$. Each row contains the signal for an individual.
response	Vector of length n with the status (1 or 0) for each individual.
causalSNP	Vector containing the indices of the centers of segments.

4.3 HDlars

The function `HDlars` performs the LARS algorithm, here is its usage in `R`:

```
> # HDlars(X, y, maxSteps,intercept,eps)
```

X	Matrix of size $n \times p$ of covariates.
y	Vector of length n with the responses.
maxSteps	Maximal number of steps for the LARS algorithm.
	The default value is `3*min(n,p)`.
intercept	If TRUE, there is an intercept in the model.
eps	Tolerance of the algorithm.
	The default value is `.Machine$double.eps^0.5`.

This function returns a `LarsPath` object (see section 4.1).

First, we try the function on a basic example.

We simulate a data set with the function `simul`.

```
> data1=simul(50,10000,0.4,2,1000,matrix(c(0,1,0,0),nrow=2),5)
```

With these parameters, only causal SNPs create an anomaly for the case individual, whereas for the control there is no anomaly through the signal.

We launch the LARS algorithm on the simulated data `data1`.

```
> reslars=HDlars(data1$data, data1$response)
```

We can plot the path of the LARS algorithm with the `plot` function.

```
> plot(reslars)
```

In abscissa, there are the values of $\|\hat{\beta}\|_1$ at each step, and in ordinate, the value of coefficients. A colored curve represents the evolution of the value of one coefficient of $\hat{\beta}$ during the algorithm. A vertical blue line indicates that a variable is added or dropped to the active set.

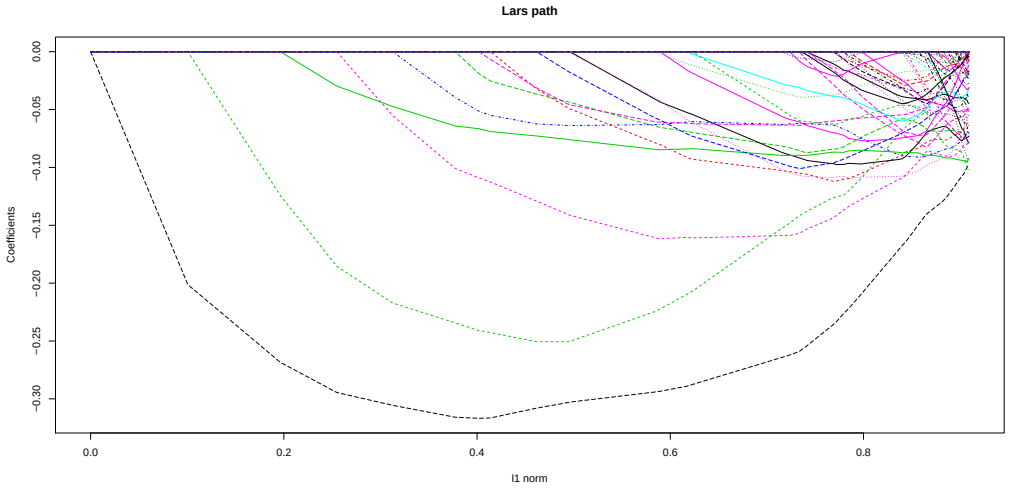


Figure 2: Path of the LARS algorithm.

Have a look on the center of the causal segments and the selected covariates at step 15.

```
> sort(data1$causalSNP)

[1] 4280 6823

> sort(reslars@variable[[16]])

 [1] 3885 3994 4232 4242 4413 4441 4587 4631 6625 6828 6900 7026 7041 7090 7107
```

We find SNPs that are geographically closed to the centers.

4.4 HDcvlars

In order to help select the most appropriate coefficients, a cross-validation function is available. This function will split the data into k folds and performs lars on data from $k-1$ folds and then use the remaining fold to predict the response at given index and compute the prediction error.

The function `HDcvlars` performs a cross-validation, here is its usage in `R`:

```
> # HDcvlars(X, y, nbFolds, index, mode, maxSteps, partition, intercept, eps)
```

X	Matrix of size say $n \times p$ of covariates.
y	Vector of length n with the responses.
nbFolds	Number of folds for the cross-validation. The default value is 10.
index	Values at which prediction error should be computed. When mode = "fraction", this is the fraction of the saturated |beta|. When mode="lambda", this is values of lambda.
mode	Either "fraction" or "lambda". Type of values containing in partition.
maxSteps	Maximal number of steps for the LARS algorithm. The default value is `3*min(n,p)`.
partition	partition in nbFolds folds of y.
intercept	If TRUE, there is an intercept in the model.
eps	Tolerance of the algorithm. The default value is `.Machine$double.eps^0.5`.

This function returns a list containing :

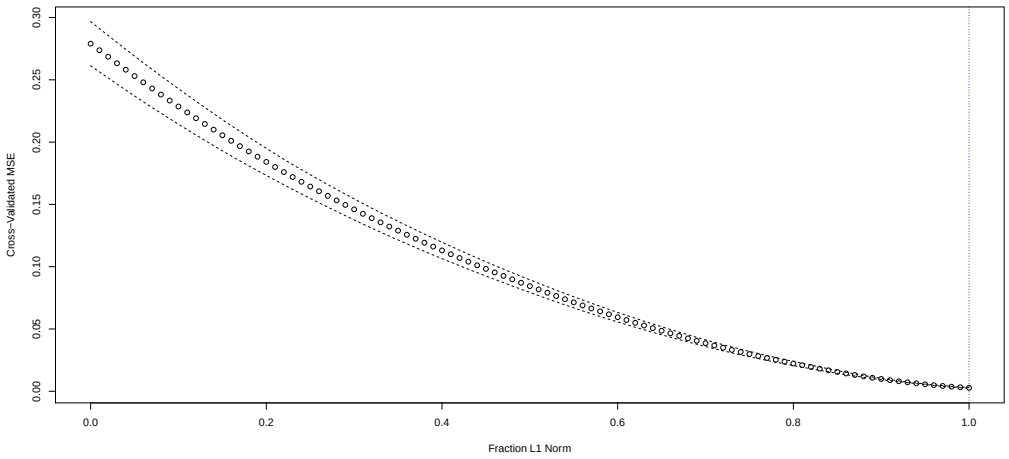
cv	Vector containing the mean squared error (MSE) for each index.
cvError	Vector containing the standard error of the MSE for each index.
minCv	Littlest mean squared error.
minIndex	Value of index for which the cv criterion is minimal.
index	l_1 norm ratio at which the mean squared error of prediction was computed.
maxSteps	Maximal number of steps for the LARS algorithm.

We perform a 5-folds cross-validation on the previously simulated data set.

```
> rescvlars=HDcvlars(data1$data, data1$response,5)
```

Then, we can plot the MSE curve

```
> plot(rescvlars)
```



The L_1-norm fraction with the littlest MSE is 1.

We can now compute the coefficients associated to this fraction with the `computeCoefficients` function.

```
> coefficients=computeCoefficients(reslars, rescvlars$minIndex, mode = "lambda")
```

We obtain a list containing two vectors : variable and coefficient.

If we have a new data set without the response, we can use the `predict` function to compute the associated response.

```
> # yPred=predict(reslars, rescvlars$fraction, mode = "fraction")
```

4.5 HDfusion

The `HDlars` function performs the LARS algorithm, here is its usage in `R`:

```
> # HDfusion(X, y, maxSteps,intercept,eps)
```

This function takes in arguments which are the same parameters as `HDlars`.

Try the function on the same data set as previously.

```
> resfusion=HDfusion(data1$data, data1$response)
```

We plot the path of the algorithm.

```
> plot(resfusion)
```

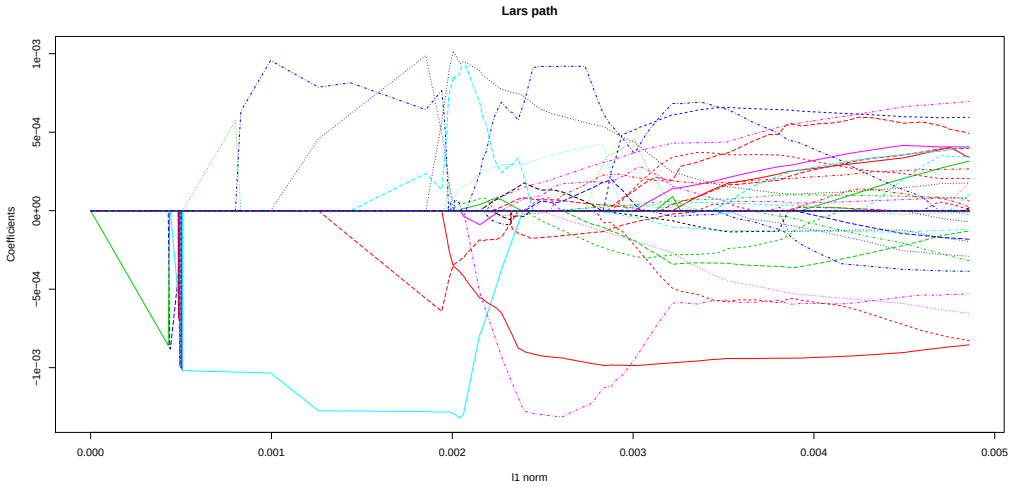


Figure 3: Path of the LARS algorithm on Z.

It is the path of the LARS version of the fusion (see equation (4)), a non-zero coefficient means a breakpoint in the solution with a jump of the value of the coefficient.

The `LarsPath` object contains the solution $\hat{\beta}$ of (4) (for the lasso version of the fusion problem).
To obtain the value of the estimate for the fusion problem (3) at a given step, we can use the `coefficient` function.
Coefficients are organized in segments of the same value. Here, it's more difficult to analyze due to few number of non-zero coefficient.

References

[Efron et al., 2004] Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004). Least angle regression. *Annals of Statistics*, 32:407–499.

[Tibshirani, 1994] Tibshirani, R. (1994). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288.

[Tibshirani et al., 2005] Tibshirani, R., Saunders, M., Rosset, S., Zhu, J., and Knight, K. (2005). Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society Series B*, pages 91–108.

```
> plotCoefficient(resfusion,20)
```

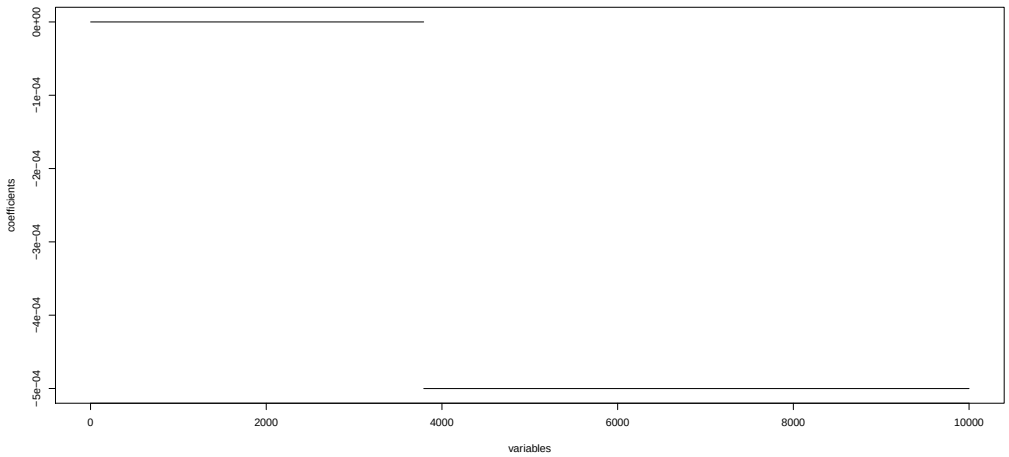


Figure 4: $\hat{\beta}$ coefficients at the step 20.