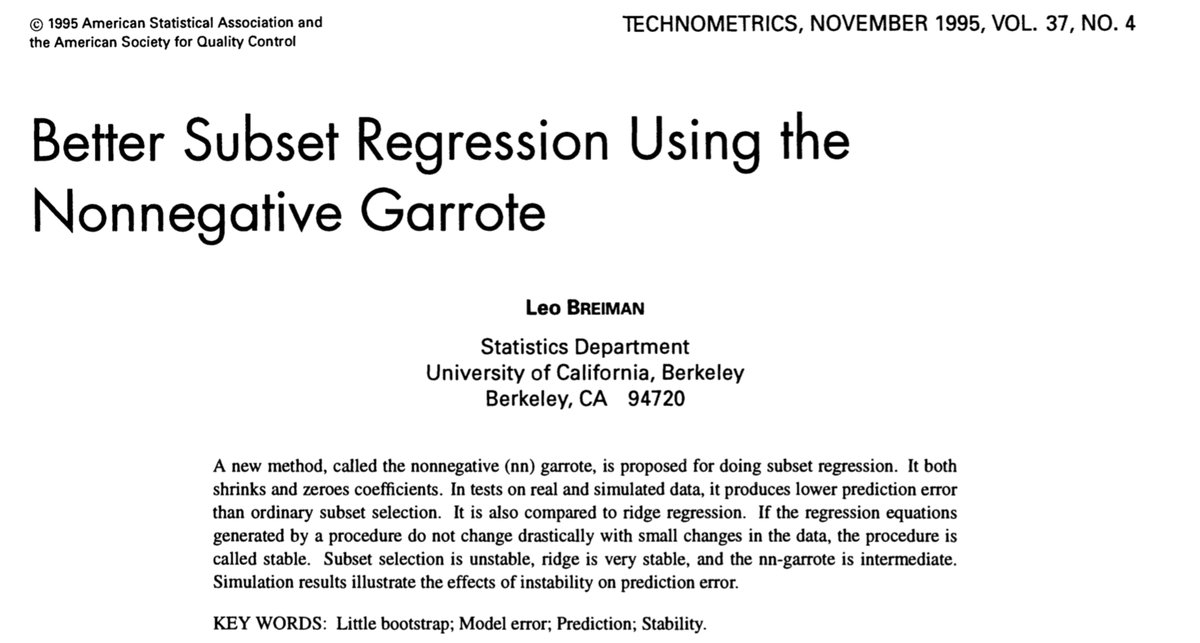
Fifth post of our series on [classification from scratch](https://freakonometrics.hypotheses.org/52731), following the previous post on penalization using the ℓ2\ell\_2ℓ2​ norm (so-called [Ridge regression](https://freakonometrics.hypotheses.org/52773)), this time, we will discuss penalization based on the ℓ1\ell\_1ℓ1​ norm (the so-called Lasso regression).

First of all, one should admit that if the name stands for [least absolute shrinkage and selection operator](https://en.wikipedia.org/wiki/Lasso_(statistics)), that’s actually a very cool name… Funny story, a few years before, Leo Breiman introduce a concept of [garrote technique](https://t.co/THHHYIwgCL)… “The garrote eliminates some variables, shrinks others, and is relatively stable”.



I guess that somehow, the lasso is the extension of the garotte technique

**Normalization of the covariates**

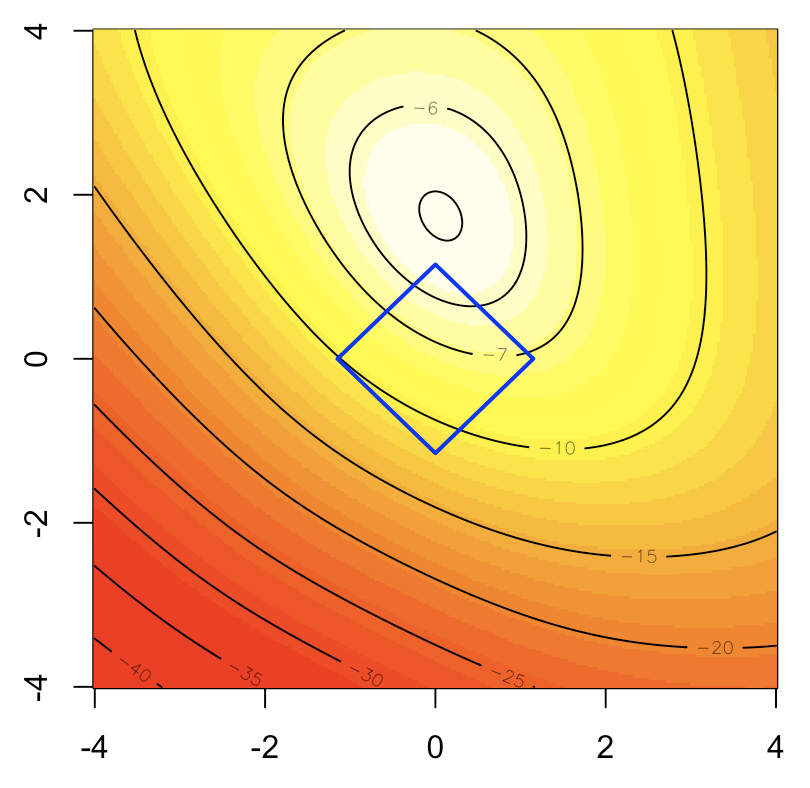
As [previously](https://freakonometrics.hypotheses.org/52773), the first step will be to consider linear transformations of all covariates xjx\_jxj​ to get centered and scaled variables (with unit variance)

|  |
| --- |
| y = myocarde$PRONO  X = myocarde[,1:7]  **for**(j **in** 1:7) X[,j] = (X[,j]-**mean**(X[,j]))/**sd**(X[,j])  X = **as.matrix**(X) |

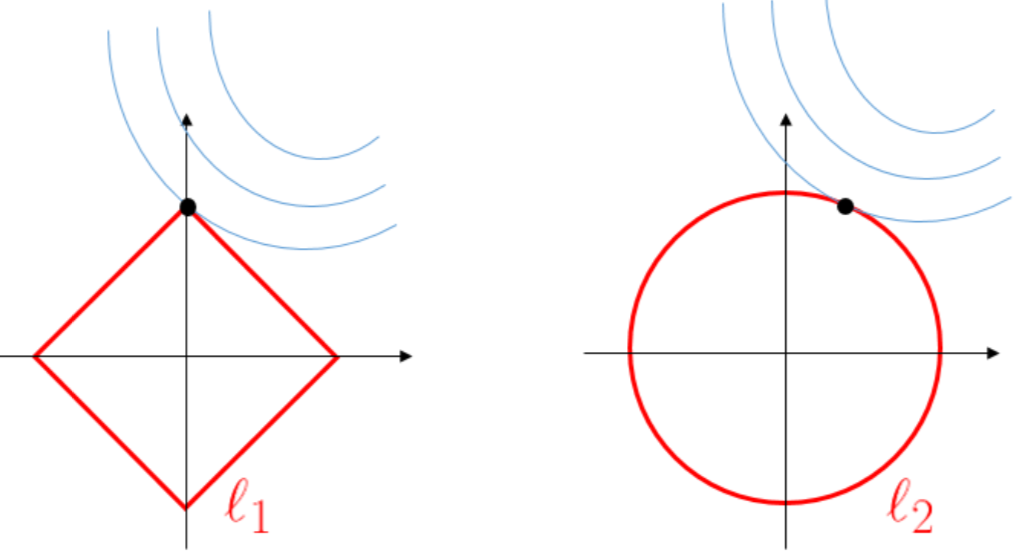
**Ridge Regression (from scratch)**

The heuristics about Lasso regression is the following graph. In the background, we can visualize the (two-dimensional) log-likelihood of the logistic regression, and the blue square is the constraint we have, if we rewite the optimization problem as a contrained optimization problem,

|  |
| --- |
| LogLik = **function**(bbeta){  b0=bbeta[1]  **beta**=bbeta[-1]  **sum**(-y\***log**(1 + **exp**(-(b0+X%\*%**beta**))) -  (1-y)\***log**(1 + **exp**(b0+X%\*%**beta**)))}  u = **seq**(-4,4,**length**=251)  v = **outer**(u,u,**function**(x,y) LogLik(**c**(1,x,y)))  **image**(u,u,v,**col**=**rev**(**heat.colors**(25)))  **contour**(u,u,v,add=TRUE)  **polygon**(**c**(-1,0,1,0),**c**(0,1,0,-1),border="blue") |



The nice thing here is that is works as a variable selection tool, since some components can be null here. That’s the idea behind the following (popular) graph

  
(with lasso on the left, and ridge on the right).

Heuristically, the maths explanation is the following. Consider a simple regression yi=xiβ+εy\_i=x\_i\beta+\varepsilonyi​=xi​β+ε, with ℓ1\ell\_1ℓ1​-penality and a ℓ2\ell\_2ℓ2​-loss fuction. The optimization problem becomesmin⁡{yTy−2yTxβ+βxTxβ+2λ∣β∣}\min\big\{\mathbf{y}^T\mathbf{y}-2\mathbf{y}^T\mathbf{x}\beta+\beta\mathbf{x}^T\mathbf{x}\beta+2\lambda{\color{red}{|}}\beta{\color{red}{|}}\big\}min{yTy−2yTxβ+βxTxβ+2λ∣β∣}The first order condition can be written−2yTx+2xTxβ^±2λ=0-2\mathbf{y}^T\mathbf{x}+2\mathbf{x}^T\mathbf{x}\widehat{\beta}{\color{red}{\pm} }2\lambda=0−2yTx+2xTxβ

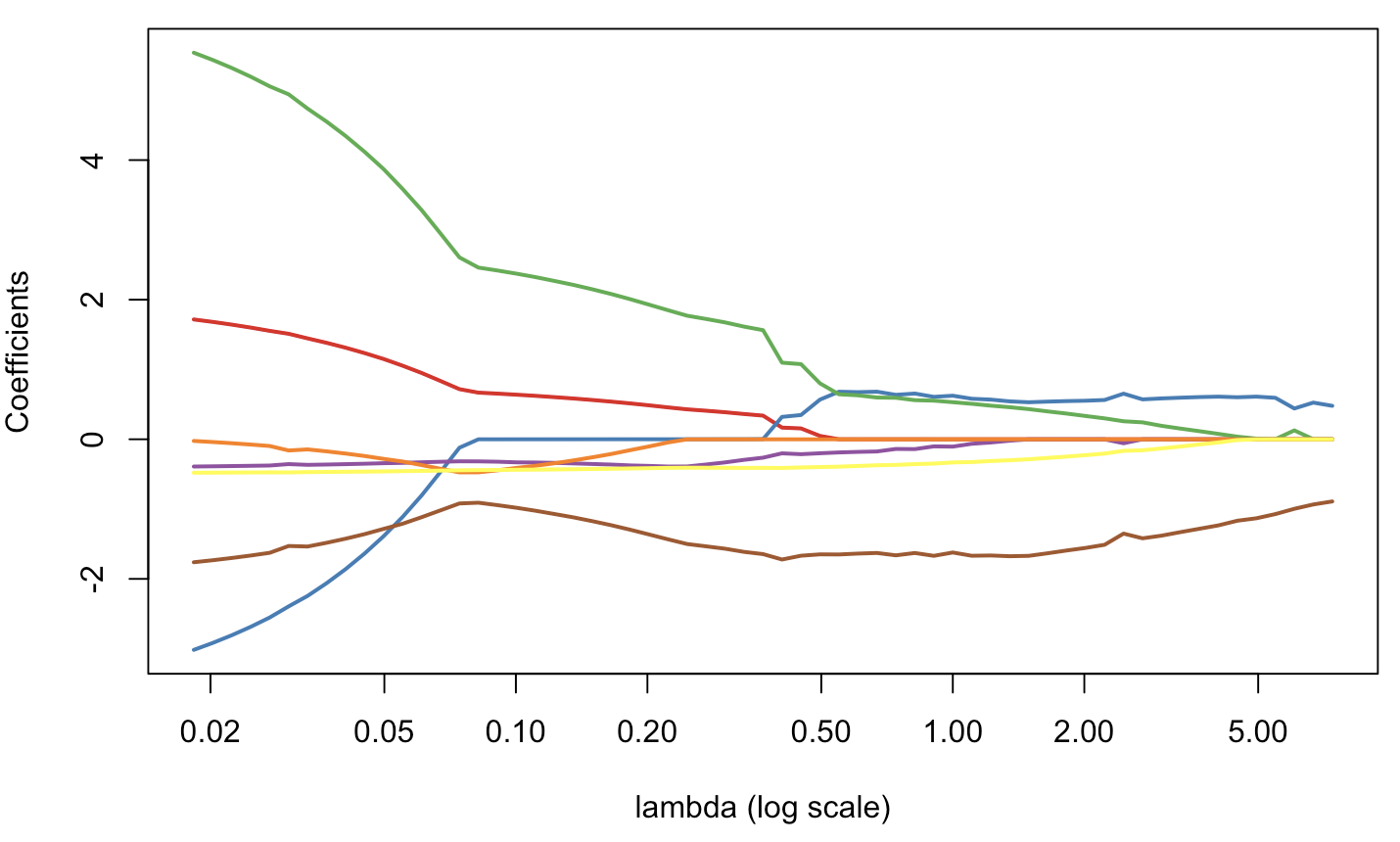
​±2λ=0(the sign in ±{\color{red}{\pm}}± being the sign of β^\widehat{\beta}β​).  
Assume that yTx>0\mathbf{y}^T\mathbf{x}>0yTx>0, then solution is  
β^λlasso=max⁡{yTx−λxTx,0}\widehat{\beta}\_{\lambda}^{lasso}=\max\left\lbrace\frac{\mathbf{y}^T\mathbf{x}-\lambda}{\mathbf{x}^T\mathbf{x}},0\right\rbraceβ

​λlasso​=max{xTxyTx−λ​,0}(we get a corner solution when λ\lambdaλ is large).

**Optimization routine**

As in our previous post, let us start with standard (R) optimization routines, such as [BFGS](https://en.wikipedia.org/wiki/Broyden%E2%80%93Fletcher%E2%80%93Goldfarb%E2%80%93Shanno_algorithm)

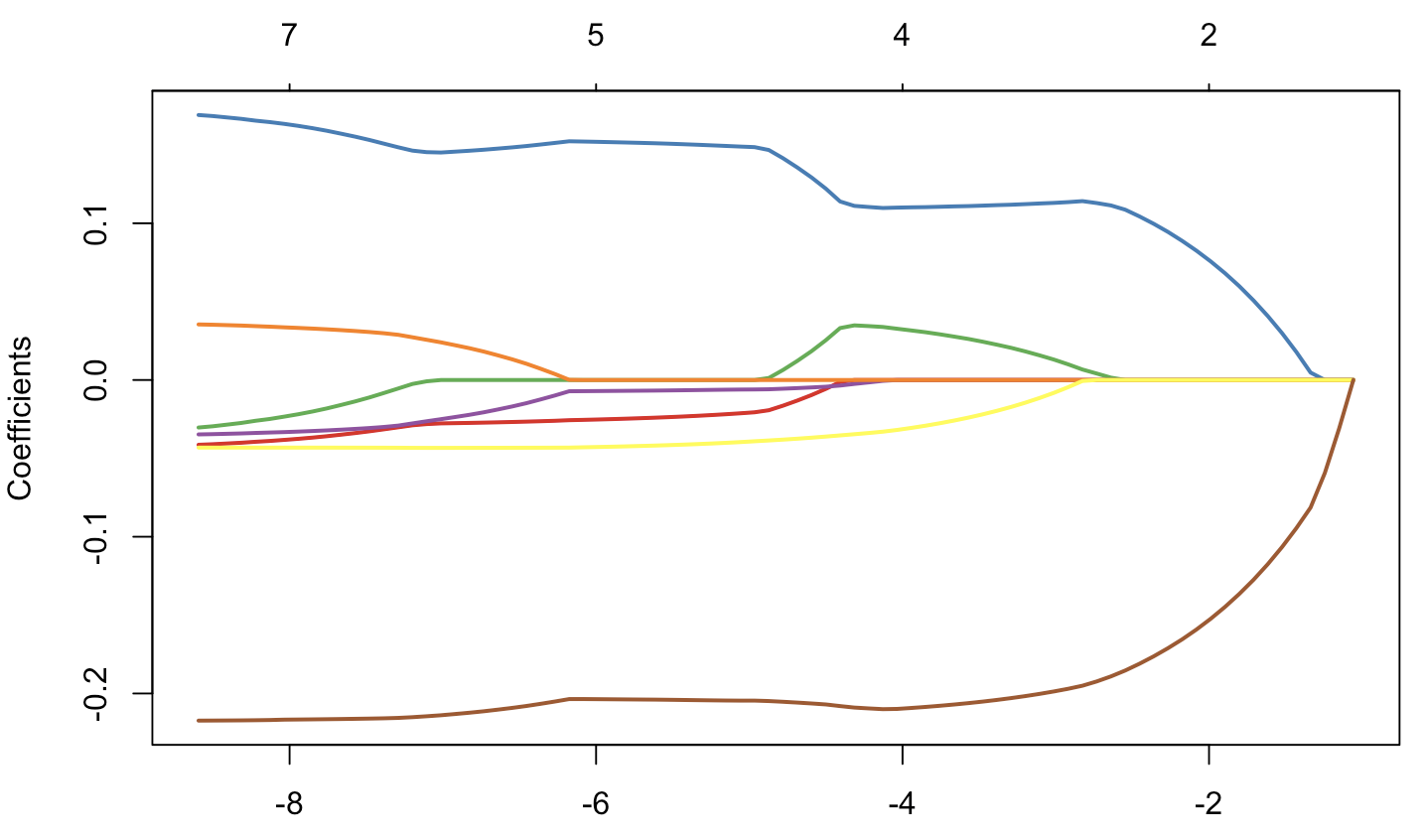
|  |
| --- |
| PennegLogLik = **function**(bbeta,lambda=0){  b0=bbeta[1]  **beta**=bbeta[-1]  -**sum**(-y\***log**(1 + **exp**(-(b0+X%\*%**beta**))) -  (1-y)\***log**(1 + **exp**(b0+X%\*%**beta**)))+lambda\***sum**(**abs**(**beta**))  }  opt\_lasso = **function**(lambda){  beta\_init = **lm**(PRONO~.,**data**=myocarde)$coefficients  logistic\_opt = **optim**(**par** = beta\_init\*0, **function**(x) PennegLogLik(x,lambda),  hessian=TRUE, method = "BFGS", control=**list**(abstol=1e-9))  logistic\_opt$par[-1]  }  v\_lambda=**c**(**exp**(**seq**(-4,2,**length**=61)))  est\_lasso=**Vectorize**(opt\_lasso)(v\_lambda)  **library**("RColorBrewer")  colrs=brewer.pal(7,"Set1")  **plot**(v\_lambda,est\_lasso[1,],**col**=colrs[1],type="l")  **for**(i **in** 2:7) **lines**(v\_lambda,est\_lasso[i,],**col**=colrs[i],lwd=2) |

  
But it is very heratic… or non stable.

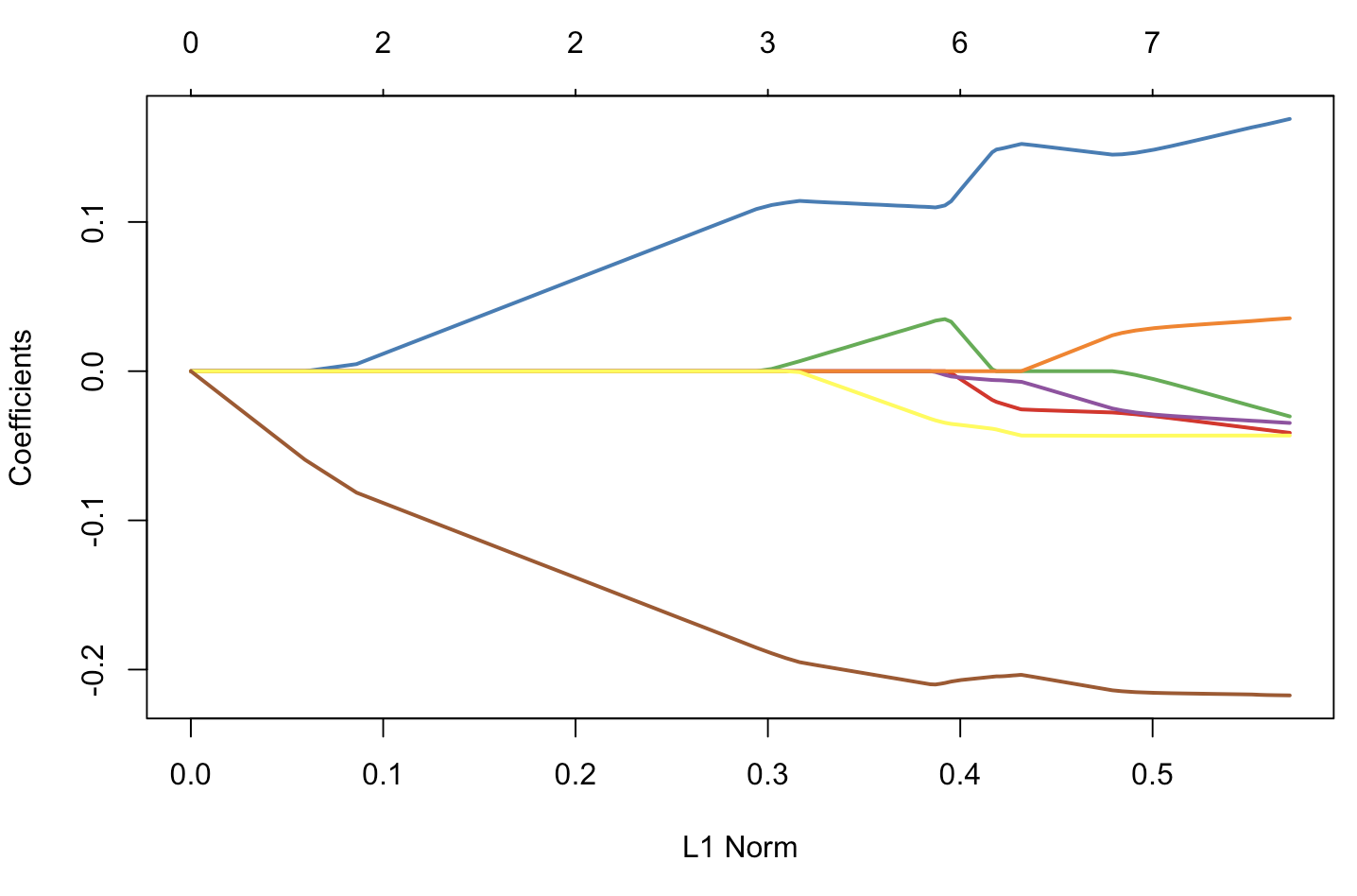
**Using glmnet**

Just to compare, with R routines dedicated to lasso, we get the following

|  |
| --- |
| **library**(glmnet)  glm\_lasso = glmnet(X, y, alpha=1)  **plot**(glm\_lasso,xvar="lambda",**col**=colrs,lwd=2) |



|  |
| --- |
| **plot**(glm\_lasso,**col**=colrs,lwd=2) |



If we look carefully what’s in the ouput, we can see that there is variable selection, in the sense that some β^j,λ=0\widehat{\beta}\_{j,\lambda}=0β

​j,λ​=0, in the sense “really null”

|  |
| --- |
| glmnet(X, y, alpha=1,lambda=**exp**(-4))$beta  7x1 sparse Matrix of **class** "dgCMatrix"  s0  FRCAR .  INCAR 0.11005070  INSYS 0.03231929  PRDIA .  PAPUL .  PVENT -0.03138089  REPUL -0.20962611 |

Of course, with out optimization routine, we cannot expect to have null values

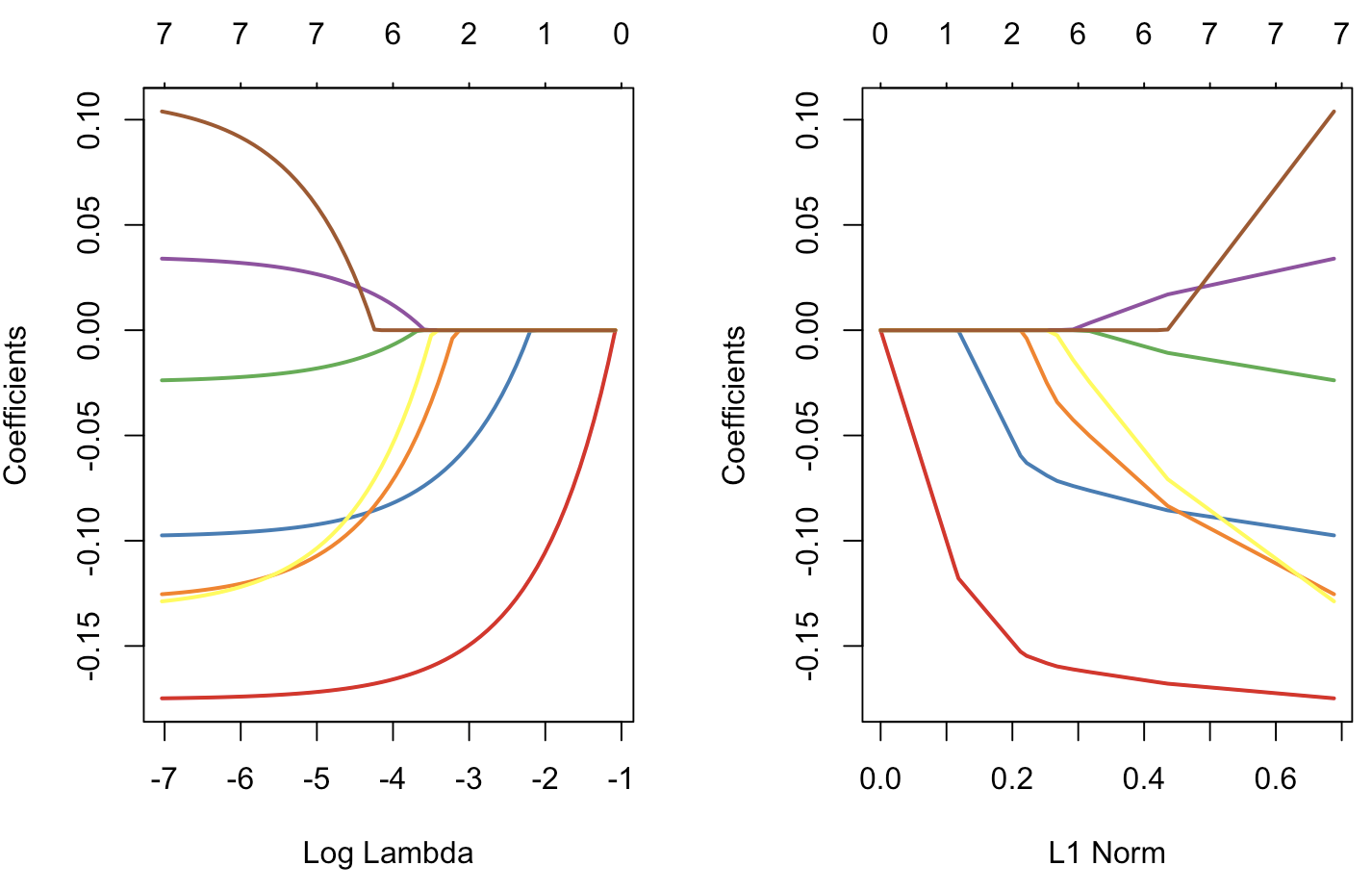
|  |
| --- |
| opt\_lasso(.2)  FRCAR INCAR INSYS PRDIA  0.4810999782 0.0002813658 1.9117847987 -0.3873926427  PAPUL PVENT REPUL  -0.0863050787 -0.4144139379 -1.3849264055 |

So clearly, it will be necessary to spend more time today, to understand how it works…

**Orthogonal covariates**

Before getting into the maths, observe that when covariates are orthogonal, there is some very clear “variable” selection process,

|  |
| --- |
| **library**(factoextra)  pca = **princomp**(X)  pca\_X = get\_pca\_ind(pca)$coord  glm\_lasso = glmnet(pca\_X, y, alpha=1)  **plot**(glm\_lasso,xvar="lambda",**col**=colrs)  **plot**(glm\_lasso,**col**=colrs) |



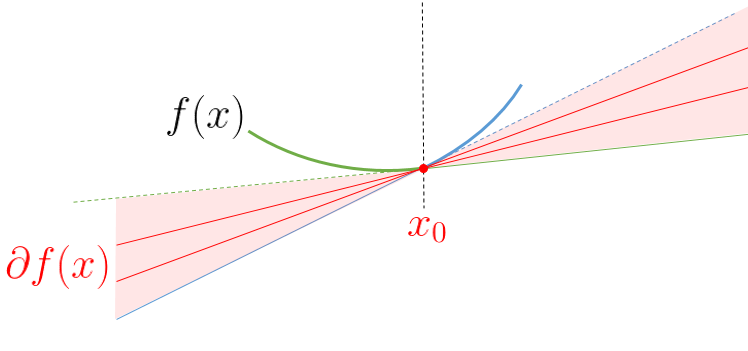
**Interior Point approach**

The penalty is now expressed using the ℓ1\ell\_1ℓ1​ so intuitively, it should be possible to consider algorithms related to linear programming. That was actually suggested in [Koh, Kim & Boyd (2007)](http://jmlr.csail.mit.edu/papers/volume8/koh07a/koh07a.pdf), with some implementation in matlab, see <http://web.stanford.edu/~boyd/l1_logreg/>. If I can find some time, later one, maybe I will try to recode it. But actually, it is not the technique used in most R functions.

Now, o be honest, we face a double challenge today: the first one is to understand how lasso works for the “standard” (least square) problem, the second one is to see how to adapt it to the logistic case.

**Standard lasso (with weights)**

If we get back to the original Lasso approach, the goal was to solvemin⁡{12n∑i=1n[yi−(β0+xiTβ)]2+λ∑j∣βj∣}\min\left\lbrace\frac{1}{2n}\sum\_{i=1}^n [y\_i-(\beta\_0+\mathbf{x}\_i^T\mathbf{\beta})]^2+\lambda \sum\_j |\beta\_j|\right\rbracemin{2n1​i=1∑n​[yi​−(β0​+xiT​β)]2+λj∑​∣βj​∣}(with standard notions, as in [wikipedia](https://en.wikipedia.org/wiki/Lasso_(statistics)) or [Jocelyn Chi’s post](http://jocelynchi.com/a-coordinate-descent-algorithm-for-the-lasso-problem) – most of the code in this section is inspired by Jocelyn’s great post).

Observe that the intercept is not subject to the penalty. The first order condition is then∂∂β0∥y−Xβ−β01∥2=(Xβ−y)T1+β0∥1∥2=0\frac{\partial}{\partial\beta\_0}\|\mathbf{y}-\mathbf{X}\mathbf{\beta}-\beta\_0\mathbf{1}\|^2=(\mathbf{X}\mathbf{\beta}-\mathbf{y})^T\mathbf{1}+\beta\_0\|\mathbf{1}\|^2=0∂β0​∂​∥y−Xβ−β0​1∥2=(Xβ−y)T1+β0​∥1∥2=0i.e.β0=1n2(Xβ−y)T1\beta\_0=\frac{1}{n^2}(\mathbf{X}\mathbf{\beta}-\mathbf{y})^T\mathbf{1}β0​=n21​(Xβ−y)T1Assume now that [KKT conditions](https://en.wikipedia.org/wiki/Karush%E2%80%93Kuhn%E2%80%93Tucker_conditions) are satisfied, since we cannot differentiate (to find points where the gradient is 0\mathbf{0}0), we can check if 0\mathbf{0}0 contains the subdifferential at the minimum.  


Namely0∈∂(12∥y−Xβ∥2+λ∥β∥ℓ1)=12∇∥y−Xβ∥2+∂(λ∥β∥ℓ1)\mathbf{0}\in\partial \left(\frac{1}{2}\|\mathbf{y}-\mathbf{X}\mathbf{\beta}\|^2+\lambda\|\mathbf{\beta}\|\_{\ell\_1}\right)=\frac{1}{2}\nabla\|\mathbf{y}-\mathbf{X}\mathbf{\beta}\|^2+\partial(\lambda\|\mathbf{\beta}\|\_{\ell\_1})0∈∂(21​∥y−Xβ∥2+λ∥β∥ℓ1​​)=21​∇∥y−Xβ∥2+∂(λ∥β∥ℓ1​​)  
For the term on the left, we recognize 12∇∥y−Xβ∥2=−XT(y−Xβ)=−g\frac{1}{2}\nabla\|\mathbf{y}-\mathbf{X}\mathbf{\beta}\|^2=-\mathbf{X}^T(\mathbf{y}-\mathbf{X}\mathbf{\beta})=-\mathbf{g}21​∇∥y−Xβ∥2=−XT(y−Xβ)=−gso that the previous equation can be writengk∈∂(λ∣βk∣)={{+λ} if βk>0{−λ} if βk<0(−λ,+λ) if βk=0g\_k\in\partial(\lambda|\beta\_k|)=\begin{cases}\{+\lambda\}\text{ if }\beta\_k>0 \\ \{-\lambda\}\text{ if }\beta\_k<0 \\ (-\lambda,+\lambda)\text{ if }\beta\_k=0\end{cases}gk​∈∂(λ∣βk​∣)=⎩⎪⎪⎨⎪⎪⎧​{+λ} if βk​>0{−λ} if βk​<0(−λ,+λ) if βk​=0​i.e. if βk≠0\beta\_k\neq 0βk​​=0, then gk=sign(βk)⋅λg\_k = \text{sign}(\beta\_k)\cdot\lambdagk​=sign(βk​)⋅λ.

Then we write the [KKT conditions](https://en.wikipedia.org/wiki/Karush%E2%80%93Kuhn%E2%80%93Tucker_conditions) for this formulation and simplify them to produce a set of rules for checking our solution

We can split βj\beta\_jβj​ into a sum of its positive and negative parts by replacing βj\beta\_jβj​ with βj+−βj−\beta\_j^+-\beta\_j^-βj+​−βj−​ where βj+,βj−≥0\beta\_j^+,\beta\_j^-\geq0βj+​,βj−​≥0. Then the Lasso problem becomes−log⁡L(β)+λ∑j(βj+−βj−)-\log\mathcal{L}(\mathbf{\beta})+\lambda\sum\_j(\beta\_j^+-\beta\_j^-)−logL(β)+λj∑​(βj+​−βj−​)with constraints βj+−βj−\beta\_j^+-\beta\_j^-βj+​−βj−​.

Let αj+,αj−\alpha\_j^+,\alpha\_j^-αj+​,αj−​ denote the Lagrange multipliers for βj+,βj−\beta\_j^+,\beta\_j^-βj+​,βj−​, respectively.

L(β)+λ∑j(βj+−βj−)−∑jαj+βj+−∑jαj−βj−.L({\mathbf{\beta}}) + \lambda \sum\_{j} (\beta\_{j}^{+} - \beta\_{j}^{-}) - \sum\_{j}\alpha\_{j}^{+}\beta\_{j}^{+} - \sum\_{j} \alpha\_{j}^{-}\beta\_{j}^{-}.L(β)+λj∑​(βj+​−βj−​)−j∑​αj+​βj+​−j∑​αj−​βj−​.To satisfy the stationarity condition, we take the gradient of the Lagrangian with respect to βj+\beta\_{j}^{+}βj+​ and set it to zero to obtain∇L(β)j+λ−αj+=0\nabla L({\mathbf{\beta}})\_{j} + \lambda - \alpha\_{j}^{+} = 0∇L(β)j​+λ−αj+​=0We do the same with respect to βj−\beta\_{j}^{-}βj−​ to obtain−∇L(β)j+λ−αj−=0-\nabla L({\mathbf{\beta}})\_{j}+\lambda-\alpha\_{j}^{-} = 0−∇L(β)j​+λ−αj−​=0

As discussed in [Jocelyn Chi’s post](http://jocelynchi.com/a-coordinate-descent-algorithm-for-the-lasso-problem), primal feasibility requires that the primal constraints be satisfied so this gives us βj+≥0\beta\_{j}^{+} \ge 0βj+​≥0 and βj−≥0\beta\_{j}^{-} \ge 0βj−​≥0. Then dual feasibility requires non-negativity of the Lagrange multipliers so we get αj+≥0\alpha\_{j}^{+} \ge 0αj+​≥0 and αj−≥0\alpha\_{j}^{-} \ge 0αj−​≥0. And finally, complementary slackness requires that αj+βj+=0\alpha\_{j}^{+}\beta\_{j}^{+} = 0αj+​βj+​=0 and αj−βj−=0\alpha\_{j}^{-}\beta\_{j}^{-} = 0αj−​βj−​=0. We can simplify these conditions to obtain a simple set of rules for checking whether or not our solution is a minimum. The following is inspired by [Jocelyn Chi’s post](http://jocelynchi.com/a-coordinate-descent-algorithm-for-the-lasso-problem).

From ∇L(β)j+λ−αj+=0\nabla L(\beta)\_{j} + \lambda - \alpha\_{j}^{+} = 0∇L(β)j​+λ−αj+​=0, we have ∇L(β)j+λ=αj+≥0\nabla L(\beta)\_{j} + \lambda= \alpha\_{j}^{+} \ge 0∇L(β)j​+λ=αj+​≥0. This gives us ∇L(β)j≥−λ\nabla L(\beta)\_{j} \ge -\lambda∇L(β)j​≥−λ. From −∇L(β)j+λ−αj−=0-\nabla L(\beta)\_{j} + \lambda - \alpha\_{j}^{-} = 0−∇L(β)j​+λ−αj−​=0, we have −∇L(β)j+λ=αj−≥0-\nabla L(\beta)\_{j} + \lambda = \alpha\_{j}^{-} \ge 0−∇L(β)j​+λ=αj−​≥0. This gives us −∇L(β)j≥−λ-\nabla L(\beta)\_{j} \ge -\lambda−∇L(β)j​≥−λ, which gives us ∇L(β)j≤λ\nabla L(\beta)\_{j} \le \lambda∇L(β)j​≤λ. Hence, ∣∇L(β)j∣≤λ  ∀j\lvert \nabla L(\beta)\_{j} \rvert \le \lambda \; \forall j∣∇L(β)j​∣≤λ∀j

When βj+>0,λ>0\beta\_{j}^{+} > 0, \lambda > 0βj+​>0,λ>0, complementary slackness requires αj+=0\alpha\_{j}^{+} = 0αj+​=0. So ∇L(β)j+λ=αj+=0\nabla L(\beta)\_{j} + \lambda = \alpha\_{j}^{+} = 0∇L(β)j​+λ=αj+​=0. Hence, ∇L(β)j=−λ<0\nabla L(\beta)\_{j} = -\lambda < 0∇L(β)j​=−λ<0 since λ>0\lambda > 0λ>0. At the same time, −∇L(β)j+λ=αj−≥0-\nabla L(\beta)\_{j} + \lambda = \alpha\_{j}^{-} \ge 0−∇L(β)j​+λ=αj−​≥0 so 2λ=αj−>02 \lambda = \alpha\_{j}^{-} > 02λ=αj−​>0 since λ>0\lambda > 0λ>0. Then complementary slackness requires βj−=0\beta\_{j}^{-} = 0βj−​=0. Hence, when βj+>0\beta\_{j}^{+} > 0βj+​>0, we have βj−=0\beta\_{j}^{-}=0βj−​=0 and ∇L(β)j=−λ\nabla L(\beta)\_{j} = -\lambda∇L(β)j​=−λ

Similarly, when βj−>0,λ>0\beta\_{j}^{-} > 0, \lambda > 0βj−​>0,λ>0, complementary slackness requires αj−=0\alpha\_{j}^{-}=0αj−​=0. So −∇L(β)j+λ=αj−=0-\nabla L(\beta)\_{j} + \lambda = \alpha\_{j}^{-} = 0−∇L(β)j​+λ=αj−​=0 and ∇L(β)j=λ>0\nabla L(\beta)\_{j}=\lambda>0∇L(β)j​=λ>0 since λ>0\lambda > 0λ>0. Then from ∇L(β)j+λ=αj+≥0\nabla L(\beta)\_{j} + \lambda = \alpha\_{j}^{+} \ge 0∇L(β)j​+λ=αj+​≥0 and the above, we get 2λ=αj+>02 \lambda = \alpha\_{j}^{+} > 02λ=αj+​>0. Then complementary slackness requires βj+=0\beta\_{j}^{+} = 0βj+​=0. Hence, when βj−>0\beta\_{j}^{-} > 0βj−​>0, we have βj+=0\beta\_{j}^{+}=0βj+​=0 and ∇L(β)j=λ\nabla L(\beta)\_{j} = \lambda∇L(β)j​=λ.

Since βj=βj+−βj−\beta\_{j} = \beta\_{j}^{+} - \beta\_{j}^{-}βj​=βj+​−βj−​, this means that when βj>0\beta\_{j} > 0βj​>0, ∇L(β)j=−λ\nabla L(\beta)\_{j} = -\lambda∇L(β)j​=−λ. And when βj<0\beta\_{j} <0βj​<0, ∇L(β)j=λ\nabla L(\beta)\_{j} = \lambda∇L(β)j​=λ. Combining this with ∣∇L(β)j∣≤λ  ∀j\lvert \nabla L(\beta)\_{j} \rvert \le \lambda \; \forall j∣∇L(β)j​∣≤λ∀j, we arrive at the same convergence requirements that we obtained before using subdifferential calculus.

For conveniency, introduce the soft-thresholding functionS(z,γ)=sign(z)⋅(∣z∣−γ)+={z−γ if γ>∣z∣ and z<0z+γ if γ<∣z∣ and z<00 if γ≥∣z∣S(z,\gamma)=\text{sign}(z)\cdot(|z|-\gamma)\_+=\begin{cases}z-\gamma&\text{ if }\gamma>|z|\text{ and }z<0\\z+\gamma&\text{ if }\gamma<|z|\text{ and }z<0 \\0&\text{ if }\gamma\geq|z|\end{cases}S(z,γ)=sign(z)⋅(∣z∣−γ)+​=⎩⎪⎪⎨⎪⎪⎧​z−γz+γ0​ if γ>∣z∣ and z<0 if γ<∣z∣ and z<0 if γ≥∣z∣​  
Noticing that the optimization problem 12∥y−Xβ∥ℓ22+λ∥β∥ℓ1\frac{1}{2}\|\mathbf{y}-\mathbf{X}\mathbf{\beta}\|\_{\ell\_2}^2+\lambda\|\mathbf{\beta}\|\_{\ell\_1}21​∥y−Xβ∥ℓ2​2​+λ∥β∥ℓ1​​can also be written  
min⁡{∑j=1p−β^jols⋅βj+12βj2+λ∣βj∣}\min\left\lbrace\sum\_{j=1}^p -\widehat{\beta}\_j^{ols}\cdot\beta\_j+\frac{1}{2}\beta\_j^2+\lambda|\beta\_j|\right\rbracemin{j=1∑p​−β

​jols​⋅βj​+21​βj2​+λ∣βj​∣}observe thatβ^j,λ=S(β^jols,λ)\widehat{\beta}\_{j,\lambda}=S(\widehat{\beta}\_j^{ols},\lambda)β​j,λ​=S(β

​jols​,λ)which is a coordinate-wise update.

Now, if we consider a (slightly) more general problem, with weights in the first partmin⁡{12n∑i=1nωi[yi−(β0+xiTβ)]2+λ∑j∣βj∣}\min\left\lbrace\frac{1}{2n}\sum\_{i=1}^n{\color{red}{\omega\_i}} [y\_i-(\beta\_0+\mathbf{x}\_i^T\mathbf{\beta})]^2+\lambda \sum\_j |\beta\_j|\right\rbracemin{2n1​i=1∑n​ωi​[yi​−(β0​+xiT​β)]2+λj∑​∣βj​∣}the coordinate-wise update becomes  
β^j,λ,ω=S(β^jω−ols,λ)\widehat{\beta}\_{j,\lambda,{\color{red}{\omega}}}=S(\widehat{\beta}\_j^{{\color{red}{\omega-}}ols},\lambda)β

​j,λ,ω​=S(β​jω−ols​,λ)  
An alternative is to setrj=y−(β01+∑k≠jβkxk)=y−y^(j)\mathbf{r}\_j=\mathbf{y} - \left(\beta\_0\mathbf{1}+\sum\_{k\neq j}\beta\_k\mathbf{x}\_k\right)=\mathbf{y}-\widehat{\mathbf{y}}^{(j)}rj​=y−⎝⎜⎛​β0​1+k​=j∑​βk​xk​⎠⎟⎞​=y−y​(j)  
so that the optimization problem can be written, equivalently  
min⁡{12n∑j=1p[rj−βjxj]2+λ∣βj∣}\min\left\lbrace\frac{1}{2n}\sum\_{j=1}^p [\mathbf{r}\_j-\beta\_j\mathbf{x}\_j]^2+\lambda |\beta\_j|\right\rbracemin{2n1​j=1∑p​[rj​−βj​xj​]2+λ∣βj​∣}  
hencemin⁡{12n∑j=1pβj2∥xj∥−2βjrjTxj+λ∣βj∣}\min\left\lbrace\frac{1}{2n}\sum\_{j=1}^p \beta\_j^2\|\mathbf{x}\_j\|-2\beta\_j\mathbf{r}\_j^T\mathbf{x}\_j+\lambda |\beta\_j|\right\rbracemin{2n1​j=1∑p​βj2​∥xj​∥−2βj​rjT​xj​+λ∣βj​∣}  
and one gets  
βj,λ=1∥xj∥2S(rjTxj,nλ)\beta\_{j,\lambda} = \frac{1}{\|\mathbf{x}\_j\|^2}S(\mathbf{r}\_j^T\mathbf{x}\_j,n\lambda)βj,λ​=∥xj​∥21​S(rjT​xj​,nλ)  
or, if we develop  
βj,λ=1∑ixij2S(∑ixi,j[yi−y^i(j)],nλ)\beta\_{j,\lambda} = \frac{1}{\sum\_i x\_{ij}^2}S\left(\sum\_ix\_{i,j}[y\_i-\widehat{y}\_i^{(j)}],n\lambda\right)βj,λ​=∑i​xij2​1​S(i∑​xi,j​[yi​−y​i(j)​],nλ)  
Again, if there are weights ω=(ωi)\mathbf{\omega}=(\omega\_i)ω=(ωi​), the coordinate-wise update becomes  
βj,λ,ω=1∑iωixij2S(∑iωixi,j[yi−y^i(j)],nλ)\beta\_{j,\lambda,{\color{red}{\omega}}} = \frac{1}{\sum\_i {\color{red}{\omega\_i}}x\_{ij}^2}S\left(\sum\_i{\color{red}{\omega\_i}}x\_{i,j}[y\_i-\widehat{y}\_i^{(j)}],n\lambda\right)βj,λ,ω​=∑i​ωi​xij2​1​S(i∑​ωi​xi,j​[yi​−y

​i(j)​],nλ)  
The code to compute this componentwise descent is

|  |
| --- |
| soft\_thresholding = **function**(x,a){  result = **numeric**(**length**(x))  result[**which**(x &gt; a)] a)] - a  result[**which**(x &lt; -a)] &lt;- x[**which**(x &lt; -a)] + a  **return**(result)  } |

and the code

|  |
| --- |
| lasso\_coord\_desc = **function**(X,y,**beta**,lambda,tol=1e-6,maxiter=1000){  **beta** = **as.matrix**(**beta**)  X = **as.matrix**(X)  omega = **rep**(1/**length**(y),**length**(y))  obj = **numeric**(**length**=(maxiter+1))  betalist = **list**(**length**(maxiter+1))  betalist[[1]] = **beta**  beta0list = **numeric**(**length**(maxiter+1))  beta0 = **sum**(y-X%\*%**beta**)/(**length**(y))  beta0list[1] = beta0  **for** (j **in** 1:maxiter){  **for** (k **in** 1:**length**(**beta**)){  r = y - X[,-k]%\*%**beta**[-k] - beta0\***rep**(1,**length**(y))  **beta**[k] = (1/**sum**(omega\*X[,k]^2))\*soft\_thresholding(**t**(omega\*r)%\*%X[,k],**length**(y)\*lambda)  }  beta0 = **sum**(y-X%\*%**beta**)/(**length**(y))  beta0list[j+1] = beta0  betalist[[j+1]] = **beta**  obj[j] = (1/2)\*(1/**length**(y))\*norm(omega\*(y - X%\*%**beta** -  beta0\***rep**(1,**length**(y))),'F')^2 + lambda\***sum**(**abs**(**beta**))  **if** (norm(**rbind**(beta0list[j],betalist[[j]]) - **rbind**(beta0,**beta**),'F') &lt; tol) { **break** }  }  **return**(**list**(obj=obj[1:j],**beta**=**beta**,intercept=beta0)) } |

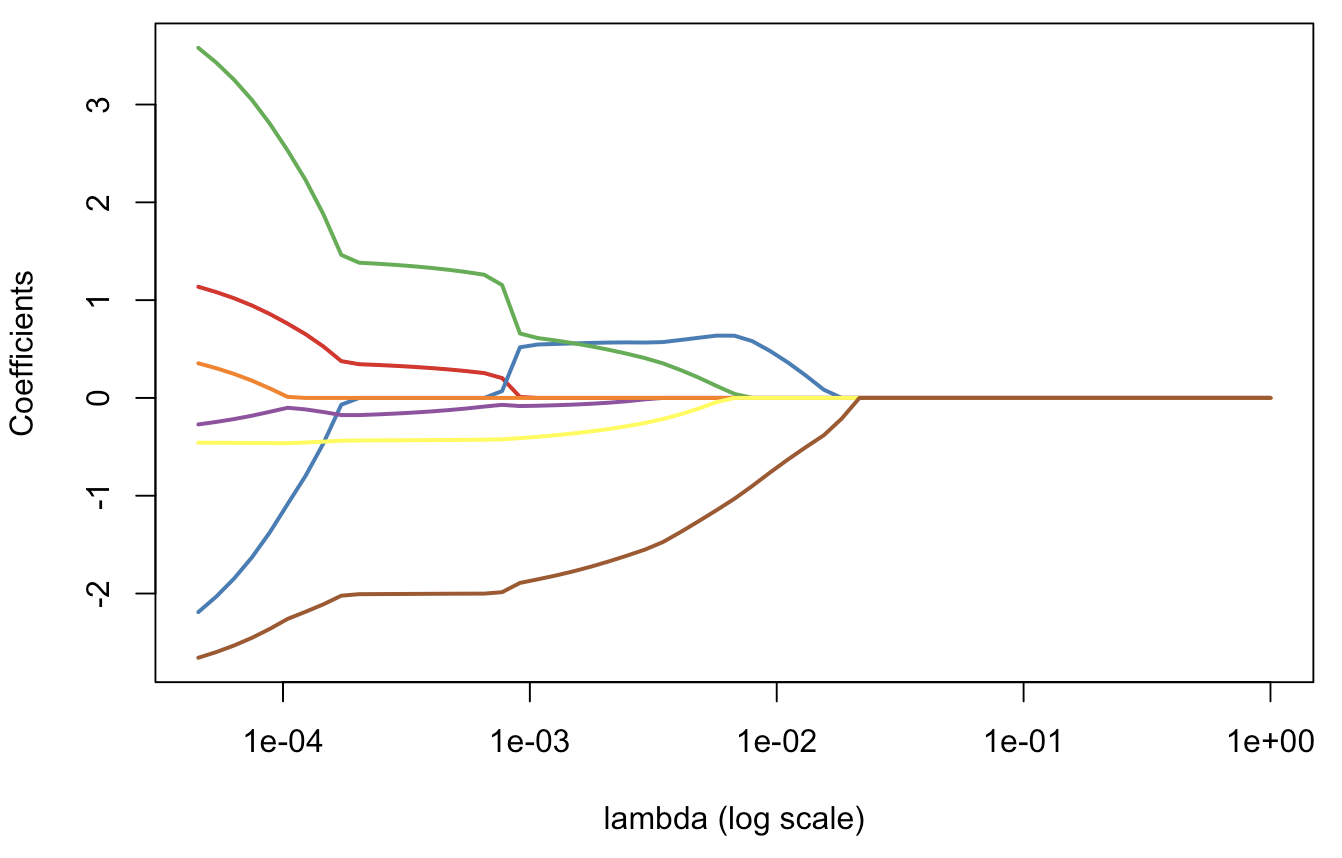
Let’s keep that one warm, and let’s get back to our initial problem.

**The lasso logistic regression**

The trick here is that the logistic problem can be formulated as a quadratic programming problem. Recall that the log-likelihood is here log⁡L=1n∑i=1nyi⋅(β0+xiTβ)−log⁡[1+exp⁡(β0+xiTβ)]\log\mathcal{L}=\frac{1}{n}\sum\_{i=1}^n y\_i\cdot(\beta\_0+\mathbf{x}\_i^T\mathbf{\beta})-\log[1+\exp(\beta\_0+\mathbf{x}\_i^T\mathbf{\beta})]logL=n1​i=1∑n​yi​⋅(β0​+xiT​β)−log[1+exp(β0​+xiT​β)]  
which is a concave function of the parameters. Hence, one can use a quadratic approximation of the log-likelihood – using Taylor expansion,log⁡L≈log⁡L′=1n∑i=1nωi⋅[zi−(β0+xiTβ)]2\log\mathcal{L}\approx\log\mathcal{L}'=\frac{1}{n}\sum\_{i=1}^n \omega\_i\cdot[z\_i-(\beta\_0+\mathbf{x}\_i^T\mathbf{\beta})]^2logL≈logL′=n1​i=1∑n​ωi​⋅[zi​−(β0​+xiT​β)]2  
where ziz\_izi​ is the working response  
zi=(β0+xiTβ)+yi−pipi[1−pi]z\_i=(\beta\_0+\mathbf{x}\_i^T\mathbf{\beta})+\frac{y\_i-p\_i}{p\_i[1-p\_i]}zi​=(β0​+xiT​β)+pi​[1−pi​]yi​−pi​​  
pip\_ipi​ is the predictionpi=exp⁡[β0+xiTβ]1+exp⁡[β0+xiTβ]p\_i = \frac{\exp[\beta\_0+\mathbf{x}\_i^T\mathbf{\beta}]}{1+\exp[\beta\_0+\mathbf{x}\_i^T\mathbf{\beta}]}pi​=1+exp[β0​+xiT​β]exp[β0​+xiT​β]​and ωi\omega\_iωi​ are weights ωi=pi[1−pi]\omega\_i = p\_i[1-p\_i]ωi​=pi​[1−pi​].

Thus, we obtain a penalized least-square problem. And we can use what was done previously

|  |
| --- |
| lasso\_coord\_desc = **function**(X,y,**beta**,lambda,tol=1e-6,maxiter=1000){  **beta** = **as.matrix**(**beta**)  X = **as.matrix**(X)  obj = **numeric**(**length**=(maxiter+1))  betalist = **list**(**length**(maxiter+1))  betalist[[1]] = **beta**  beta0 = **sum**(y-X%\*%**beta**)/(**length**(y))  p = **exp**(beta0\***rep**(1,**length**(y)) + X%\*%**beta**)/(1+**exp**(beta0\***rep**(1,**length**(y)) + X%\*%**beta**))  z = beta0\***rep**(1,**length**(y)) + X%\*%**beta** + (y-p)/(p\*(1-p))  omega = p\*(1-p)/(**sum**((p\*(1-p))))  beta0list = **numeric**(**length**(maxiter+1))  beta0 = **sum**(y-X%\*%**beta**)/(**length**(y))  beta0list[1] = beta0  **for** (j **in** 1:maxiter){  **for** (k **in** 1:**length**(**beta**)){  r = z - X[,-k]%\*%**beta**[-k] - beta0\***rep**(1,**length**(y))  **beta**[k] = (1/**sum**(omega\*X[,k]^2))\*soft\_thresholding(**t**(omega\*r)%\*%X[,k],**length**(y)\*lambda)  }  beta0 = **sum**(y-X%\*%**beta**)/(**length**(y))  beta0list[j+1] = beta0  betalist[[j+1]] = **beta**  obj[j] = (1/2)\*(1/**length**(y))\*norm(omega\*(z - X%\*%**beta** -  beta0\***rep**(1,**length**(y))),'F')^2 + lambda\***sum**(**abs**(**beta**))  p = **exp**(beta0\***rep**(1,**length**(y)) + X%\*%**beta**)/(1+**exp**(beta0\***rep**(1,**length**(y)) + X%\*%**beta**))  z = beta0\***rep**(1,**length**(y)) + X%\*%**beta** + (y-p)/(p\*(1-p))  omega = p\*(1-p)/(**sum**((p\*(1-p))))  **if** (norm(**rbind**(beta0list[j],betalist[[j]]) -  **rbind**(beta0,**beta**),'F') &lt; tol) { **break** }  }  **return**(**list**(obj=obj[1:j],**beta**=**beta**,intercept=beta0)) } |



It looks like what can get when calling glmnet… and here, we do have null components for some λ\lambdaλ large enough ! Really null… and that’s cool actually.

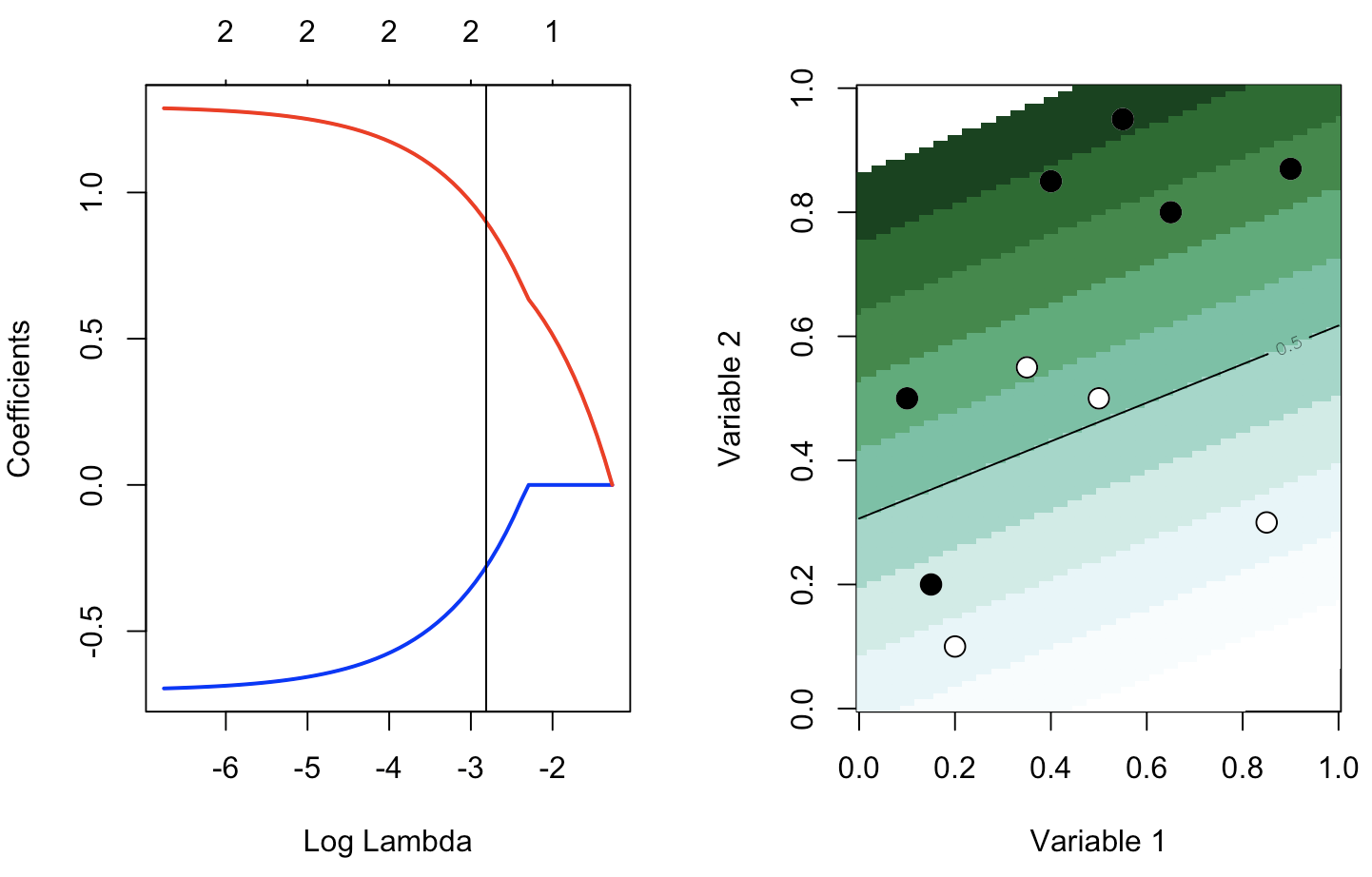
**Application on our second dataset**

Consider now the second dataset, with two covariates. The code to get lasso estimates is

|  |
| --- |
| df0 = **df**  df0$y = **as.numeric**(**df**$y)-1  plot\_lambda = **function**(lambda){  m = **apply**(df0,2,**mean**)  s = **apply**(df0,2,**sd**)  **for**(j **in** 1:2) df0[,j] &lt;- (df0[,j]-m[j])/s[j]  reg = glmnet(**cbind**(df0$x1,df0$x2), df0$y==1, alpha=1,lambda=lambda)  u = **seq**(0,1,**length**=101)  p = **function**(x,y){  xt = (x-m[1])/s[1]  yt = (y-m[2])/s[2]  **predict**(reg,newx=**cbind**(x1=xt,x2=yt),type="response")}  v = **outer**(u,u,p)  **image**(u,u,v,**col**=clr10,breaks=(0:10)/10)  **points**(**df**$x1,**df**$x2,pch=19,cex=1.5,**col**="white")  **points**(**df**$x1,**df**$x2,pch=**c**(1,19)[1+z],cex=1.5)  **contour**(u,u,v,**levels** = .5,add=TRUE)} |

Consider some small values, for [\lambda], so that we only have some sort of shrinkage of parameters,

|  |
| --- |
| reg = glmnet(**cbind**(df0$x1,df0$x2), df0$y==1, alpha=1)  **par**(mfrow=**c**(1,2))  **plot**(reg,xvar="lambda",**col**=**c**("blue","red"),lwd=2)  **abline**(v=**exp**(-2.8))  plot\_lambda(**exp**(-2.8)) |

  
But with a larger λ\lambdaλ, there is variable selection: here β^1,λ=0\widehat{\beta}\_{1,\lambda}=0β

​1,λ​=0

|  |
| --- |
| reg = glmnet(**cbind**(df0$x1,df0$x2), df0$y==1, alpha=1)  **par**(mfrow=**c**(1,2))  **plot**(reg,xvar="lambda",**col**=**c**("blue","red"),lwd=2)  **abline**(v=**exp**(-2.1))  plot\_lambda(**exp**(-2.1)) |

