Semismooth Newton Augmented Lagrangian Method for Solving Lasso Problems

with Implementation in R

MyungWon Lee (s1687781)[1]   Michael Renfrew (s1406083)[2]

[1, 2]University of Edinburgh
School of Mathematics

Final Year Project Presentation

THE UNIVERSITY of EDINBURGH
School of Mathematics

Outline

What is Lasso Regression

- Stands for *least absolute shrinkage and selection operator*. Introduced by Robert Tibshirani in 1996.

What is Lasso Regression

- Stands for *least absolute shrinkage and selection operator*. Introduced by Robert Tibshirani in 1996.
- In a regular least-squares regression with many more variables than number of subjects, there is a need to select features with the strongest effect.

What is Lasso Regression

- Stands for *least absolute shrinkage and selection operator*. Introduced by Robert Tibshirani in 1996.
- In a regular least-squares regression with many more variables than number of subjects, there is a need to select features with the strongest effect.
- This improves model explainability and reduces overfit.

What is Lasso Regression

- Stands for *least absolute shrinkage and selection operator*. Introduced by Robert Tibshirani in 1996.
- In a regular least-squares regression with many more variables than number of subjects, there is a need to select features with the strongest effect.
- This improves model explainability and reduces overfit.
- Lasso introduces a penalty term on the ℓ_1-norm of the parameter vector in addition to the least-squares term:

$$\arg\min_{x} \left\{ \frac{1}{2} \|Ax - b\|^2 + \lambda \|x\|_1 \right\}$$

What is Lasso Regression

- Stands for *least absolute shrinkage and selection operator*. Introduced by Robert Tibshirani in 1996.
- In a regular least-squares regression with many more variables than number of subjects, there is a need to select features with the strongest effect.
- This improves model explainability and reduces overfit.
- Lasso introduces a penalty term on the ℓ_1-norm of the parameter vector in addition to the least-squares term:

$$\arg\min_x \left\{ \frac{1}{2} \|Ax - b\|^2 + \lambda \|x\|_1 \right\}$$

- This *Lagrangian* form is equivalent to a constrained minimisation problem

Lasso Regression (continued)

- Similar to *ridge regression*, which introduces a constraint on the ℓ_2 norm of the parameter vector, but better, since Lasso tends to push parameters to zero, whereas ridge only decreases their absolute value.

Lasso Regression (continued)

- Similar to *ridge regression*, which introduces a constraint on the ℓ_2 norm of the parameter vector, but better, since Lasso tends to push parameters to zero, whereas ridge only decreases their absolute value.
- The optimal case would be *best-subset selection*, when all possible combinations of regressors are tried to produce the one with the lowest sum of squares for a given number of non-zero parameters.

Lasso Regression (continued)

- Similar to *ridge regression*, which introduces a constraint on the ℓ_2 norm of the parameter vector, but better, since Lasso tends to push parameters to zero, whereas ridge only decreases their absolute value.
- The optimal case would be *best-subset selection*, when all possible combinations of regressors are tried to produce the one with the lowest sum of squares for a given number of non-zero parameters.
- This is computationally costly, since it is $O(m!)$, but Lasso gives a very good approximation to it.

Lasso Regression (continued)

- Similar to *ridge regression*, which introduces a constraint on the ℓ_2 norm of the parameter vector, but better, since Lasso tends to push parameters to zero, whereas ridge only decreases their absolute value.
- The optimal case would be *best-subset selection*, when all possible combinations of regressors are tried to produce the one with the lowest sum of squares for a given number of non-zero parameters.
- This is computationally costly, since it is $O(m!)$, but Lasso gives a very good approximation to it.
- Both ridge and Lasso are *convex* problems, and can be solved with the correct numerical, computational methods

Original Paper

- *A highly efficient semismooth Newton augmented Lagrangian method for solving Lasso problems*, Li, Xudong, Sun, Defeng and Toh, Kim-Chuan, *SIAM Journal on Optimization*, Vol. 28(1), pp 433-458, **2018**, *SIAM*

Convex Composite

Primal Problem

$$(\mathbf{P}) \quad \min \left\{ \frac{1}{2}\|Ax - b\|^2 + \lambda\|x\|_1 \right\}$$

where $A : \mathcal{X} \to \mathcal{Y}$, $h : \mathcal{Y} \to \mathbb{R}$, $p : \mathcal{X} \to (-\infty, +\infty]$ and $c \in \mathcal{X}$ is a vector.
$h(\xi) = \frac{1}{2}\|\xi - b\|^2$ and $p(x) = \lambda\|x\|_1$

Convex Composite

Primal Problem

$$(\mathbf{P}) \quad \min \left\{ \frac{1}{2}\|Ax-b\|^2 + \lambda\|x\|_1 \right\}$$

where $A : \mathcal{X} \to \mathcal{Y}$, $h : \mathcal{Y} \to \mathbb{R}$, $p : \mathcal{X} \to (-\infty, +\infty]$ and $c \in \mathcal{X}$ is a vector.
$h(\xi) = \frac{1}{2}\|\xi - b\|^2$ and $p(x) = \lambda\|x\|_1$

Dual Problem

$$(\mathbf{D}) \quad \max \left\{ -\frac{1}{2}\|\xi\|^2 + \langle b, \xi \rangle \mid A^T\xi + u - c = 0, \|u\|_\infty \leq \lambda \right\}$$

where $h^*(\xi) = -\frac{1}{2}\|\xi\|^2 + \langle b, \xi \rangle$ and $p^*(x) = \mathbb{I}_{\|x\|_\infty}$
p^* and h^*: the Fenchel conjugate functions of p and h

Assumptions on the loss function

Assumption 1.

1. $h^*(\cdot)$ is essentially smooth where h^* is differentiable on $\text{int}(\text{dom } h^*) \neq \emptyset$ and $\lim_{k\to\infty}\|h^*(\xi^k)\| = +\infty$ whenever $\{\xi^k\}$ is a sequence in $\text{int}(\text{dom } h^*)$ converging to a boundary point ξ of $\text{int}(\text{dom } h^*)$.
2. $p^*(\cdot)$ is either an indicator function $\delta_p(\cdot)$ or a support function δ_p^* for some nonempty polyhedral convex set $P \subseteq \mathcal{X}$.

Assumption 2.

$h : \mathcal{Y} \to \mathbb{R}$ is a convex differentiable function whose gradient is $\frac{1}{\alpha_h}$ *Lipschitz continuous* i.e.

$$\|\nabla h(\xi') - \nabla h(\xi)\| \leq \frac{1}{\alpha_h}\|\xi' - \xi\|, \quad \forall \xi', \xi \in \mathcal{Y}$$

($h^*(\cdot)$ is strongly convex with modulus α_k)

Assumptions on the loss function

Assumption 3.

h is essentially locally strongly convex, i.e. for any compact and convex set $K \subset dom\ \partial h$, there exists $\beta_K > 0$ s.t.

$$(1-\lambda)h(\xi') + \lambda h(\xi) \le h((1-\lambda)\xi' + \lambda\xi) + \frac{1}{2}\beta_K \lambda(1-\lambda)\|\xi' - \xi\|^2, \quad \forall \xi', \xi \in K$$

($\nabla h^*(\cdot)$ is locally *Lipschitz continuous* and directionally differentiable on int(dom h^*))

Assumption 4.

The Karush Kuhn Tucker (**KKT**) system is nonempty and its solution is denoted as $\bar{\xi}, \bar{u}$ and $\bar{x}$.

An augmented Lagrangian method for (D)

- The Lagrangian function of (**D**) is,

$$\ell(\xi, u; x) = h^*(\xi) + p^*(u) - \langle x, A^*\xi + u - c\rangle, \quad \forall (\xi, u; x) \in \mathcal{Y} \times \mathcal{X} \times \mathcal{X}$$

- Given $\sigma > 0$, the augmented Lagrangian function of (**D**) is,

$$\mathcal{L}_\sigma(\bar{\xi}, \bar{u}; \bar{x}) := \ell(\xi, u; x) + \frac{\sigma}{2}\|A^*\xi + u - c\|^2, \ \forall (\xi, u; x) \in \mathcal{Y} \times \mathcal{X} \times \mathcal{X}$$

- $\mathrm{Prox}_{\sigma\lambda\|\cdot\|_1}(x) = \arg\min_u\{\frac{1}{2}\|u - x\|^2 + \sigma\lambda\|u\|_1\}$ and $\mathrm{Prox}_{\sigma p}(x) = \mathrm{sgn}(x) \circ \max\{|x| - \sigma\lambda, 0\}$.

Algorithm 1

Algorithm Inexact Augmented Lagrangian Method

Let $\sigma_0 > 0$ be a given penalty parameter.
Choose $(\xi^0, u^0, x^0) \in \mathrm{int}(\mathrm{dom}\, h^*) \times \mathrm{dom}\, p^* \times \mathcal{X}$.

for $i = 0, 1, \ldots, \infty$ **do**

(1) **Compute**

$$(\xi^{k+1}, u^{k+1}) \approx \arg\min\{\Psi_k(\xi, u) := \mathcal{L}_{\sigma_k}(\xi, u; x^k)\} \tag{1}$$

(2) **Compute**

$$x^{k+1} = x^k - \sigma_k(A^*\xi^{k+1} + u^{k+1} - c) \ \& \ \textbf{update} \quad \sigma_{k+1} \uparrow \sigma_\infty \leq \infty$$

Global and Local Convergence

From [Rockafellar, 1976a[1] and 1976b[2]],

Convergence Criterion for inner subproblem (1)

$$(A) \quad \Psi_k(\xi^{k+1}, u^{k+1}) - \inf \Psi_k \leq \frac{\varepsilon_k^2}{2\sigma_k}, \sum_{k=0}^{\infty} \varepsilon_k < \infty$$

$$(B1) \quad \Psi_k(\xi^{k+1}, u^{k+1}) - \inf \Psi_k \leq \left(\frac{\delta_k^2}{2\sigma_k}\right) \|x^{k+1} - x^k\|^2, \ \delta_k \geq 0, \ \sum_{k=0}^{\infty} \delta_k < \infty$$

$$(B2) \quad \mathrm{dist}(0, \partial \Psi_k(\xi^{k+1}, u^{k+1})) \leq \left(\frac{\delta_k'}{\sigma_k}\right) \|x^{k+1} - x^k\|, \ 0 \leq \delta_k' \to 0$$

where, $\mathrm{dist}(x, \mathcal{C}) := \inf_{x' \in \mathcal{C}} \|x - x'\|$ for any $x \in \mathcal{X}$ and any $\mathcal{C} \subset \mathcal{X}$

[1] Rockafellar, R.T., 1976. Monotone operators and the proximal point algorithm. SIAM journal on control and optimization, 14(5), pp.877-898.

[2] Rockafellar, R.T., 1976. Augmented Lagrangians and applications of the proximal point algorithm in convex programming. Mathematics of operations research, 1(2), pp.97-116.

Semismooth Newton method for inner problems

Definition (Semismoothness) [Miffin, 1977][3]

Suppose that $M : \mathcal{X} \rightrightarrows \mathcal{L}(\mathcal{X}, \mathcal{Y})$ & $F : \mathcal{X} \to \mathcal{Y}$, is a locally *Lipschitz continuous* function. F is said to be semismooth at $x \in \mathcal{X}$ if F is directionally differentiable at x and for any $G \in \partial F(x + \Delta x) = M(x + \Delta x)$ & $\Delta x \to 0$.

$$F(x + \Delta x) - F(x) - G(\Delta x) = o(\|\Delta x\|)$$

F is said to be strongly semismooth at $x \in \mathcal{X}$ if,

$$F(x + \Delta x) - F(x) - G(\Delta x) = O(\|\Delta x\|^2)$$

Then F is said to be semismooth (strongly semismooth) function on $\mathcal{X}$ if it is semismooth (strongly semismooth) everywhere in $\mathcal{X}$

[3] Mifflin, Robert. "Semismooth and semiconvex functions in constrained optimization". In: SIAM Journal on Control and Optimization 15.6 (1977),pp. 959–972

Semismooth Newton method for inner problems

- While fixing $\sigma > 0$, we newly denote for $\xi \in \text{int}(\text{dom}\, h^*)$,

$$\begin{aligned}\psi(\xi) &:= \inf_u \Psi(\xi, u) = \inf \mathcal{L}_\sigma(\xi, u; x) \\ &= h^*(\xi) + p^*(\text{Prox}_{p^*/\sigma})(\bar{x}/\sigma - A^*\xi + c) \\ &+ \frac{1}{2\sigma}\|\text{Prox}_{\sigma p}(\bar{x} - \sigma(A^*\xi - c))\|^2 - \frac{1}{2\sigma}\|\bar{x}\|^2\end{aligned}$$

- We compute the 1^{st} and 2^{nd} derivatives

$$\begin{aligned}\nabla\psi(\xi) &= \nabla h^*(\xi) - A\,\text{Prox}_{\sigma p}(\bar{x} - \sigma(A^*\xi - c)),\ \forall\xi \in \text{int}(\text{dom}\, h^*) \\ \hat{\partial}^2\psi(\xi) &= \nabla^2 h^*(\xi) + \sigma A\,\text{Prox}_{\sigma p}(\bar{x} - \sigma(A^*\xi - c))A^*\end{aligned}$$

- where $\partial^2 h^*(\xi)$ is the Clarke subdifferential of $\nabla h^*(\cdot)$ at ξ and $\partial\,\text{Prox}_{\sigma p}(\bar{x} - \sigma(A^*\xi - c))$ is the Clarke subdifferential of the *Lipschitz continuous* mapping and Jacobian of $\text{Prox}_{\sigma p}(\cdot)$ at $\bar{x} - \sigma(A^*\xi - c)$.

Semismooth Newton method for inner problems

- Then solve the semismooth equation,

$$\nabla \psi(\xi) = 0, \quad \xi \in \mathrm{int}(\mathrm{dom}\, h^*)$$

- Then we define the generalised Hessian of ψ at ξ,

$$V := H + \sigma A P A^*$$

with $H \in \partial^2 h^*(\xi)$ and $P \in \partial\,\mathrm{Prox}_{\sigma p}(\bar{x} - \sigma(A^*\xi - c))$ where for the case of our paper, $h(\xi) = \frac{1}{2}\|\xi\|^2 + b^T\xi$, $\nabla h^*(\xi) = \xi + b$, $\nabla^2 h^*(\xi) = \mathbb{I}_m$. $h^*(\cdot)$ is twice differentiable and $\mathrm{Prox}_{\lambda\|\cdot\|}$ is a piecewise linear function which are strongly semismooth.

Algorithm 2

Algorithm Semismooth Newton Method

Given the hyperparameter, $\mu \in (0, \frac{1}{2}), \tilde{\eta} \in (0,1), \tau \in (0,1]$ & $\delta \in (0,1)$.
Choose $\xi^0 \in \text{int}(\text{dom}\, h^*)$ and **for** $j = 0, 1, \ldots, \infty$ **do**
(1) **Choose** $H_j \in \partial^2(\nabla h^*)(\xi^j)$ & $P_j \in \partial\, \text{Prox}_{\sigma p}(\tilde{x} - \sigma(A^*\xi^j - c))$. **Let** $V_j := H_j + \sigma A P_j A^*$. **Solve** the following Linear System exactly or by the conjugate gradient algorithm to find d^j

$$V_j d + \nabla\psi(\xi^j) = 0, \text{where } d = \Delta\xi$$
$$\text{s.t. } \|V_j d^j + \nabla\psi(\xi^j)\| \leq \min(\tilde{\eta}, \|\nabla\psi(\xi^j)\|)^{1+\tau}$$

(2) **Set** $\alpha = \delta^{m_j}$ where m_j is the first nonnegative integer m for which,

$$\xi^j + \delta^m d^j \in \text{int}(\text{dom}\, h^*) \text{ and } \psi(\xi^j + \delta^m d^j) \leq \psi(\xi^j) + \mu\delta^m\langle\nabla\psi(\xi^j), d^j\rangle$$

(3) **Set** $\xi^{j+1} = \xi^j + \alpha_j d^j$

Convergence of semismooth Newton method

Let the sequence $\{\xi^j\}$ be generated by **Algorithm 2**,

Theorem

Assume that $\nabla h^*(\cdot)$ and $\text{Prox}_{\sigma p}(\cdot)$ are strongly semismooth on $\int(\text{dom}\, h^*)$ and $\mathcal{X}$ then ξ^j converges to the unique optimal solution $\bar{\xi} \in \text{int}(\text{dom}\, h^*)$ and $\bar{u} = \bar{x} - \sigma(A^T\bar{\xi} - c)$ at least superlinearly, i.e.,

$$\|\xi^{j+1} - \bar{\xi}\| = O(\|\xi^j - \bar{\xi}\|^{1+\tau}), \text{ for any } j \geq 0, V_j \in \partial^2\psi(\xi^j)$$

Convergence of semismooth Newton method

Let the sequence $\{\xi^j\}$ be generated by **Algorithm 2**,

Theorem

Assume that $\nabla h^*(\cdot)$ and $\mathrm{Prox}_{\sigma p}(\cdot)$ are strongly semismooth on $\int(\mathrm{dom}\, h^*)$ and $\mathcal{X}$ then ξ^j converges to the unique optimal solution $\bar{\xi} \in \mathrm{int}(\mathrm{dom}\, h^*)$ and $\bar{u} = \bar{x} - \sigma(A^T\bar{\xi} - c)$ at least superlinearly, i.e.,

$$\|\xi^{j+1} - \bar{\xi}\| = O(\|\xi^j - \bar{\xi}\|^{1+\tau}), \text{ for any } j \geq 0, V_j \in \partial^2\psi(\xi^j)$$

Then, the implementable stopping criterion from the stopping criteria (A), (B1) and (B2)

$$(A') \quad \|\psi_k(\xi^{k+1})\| \leq \frac{\varepsilon_k}{\sqrt{\sigma_k/\alpha_h}}$$

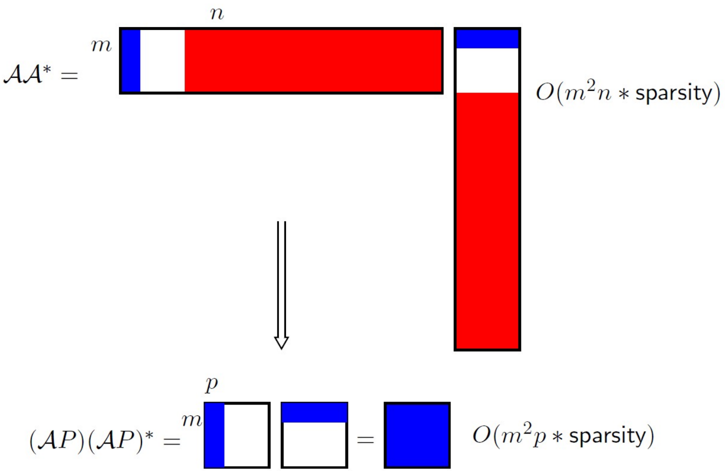
$$(B1') \quad \|\nabla\psi_k(\xi^{k+1})\| \leq \sqrt{\alpha_h\sigma_k}\delta_k\|A^*\xi^{k+1} + u^{k+1} - c\|$$

$$(B2') \quad \|\nabla\psi_k(\xi^{k+1})\| \leq \delta'\|A^*\xi^{k+1} + u^{k+1} - c\|, \ 0 \leq \delta'_k \to 0$$

where $\sum_{k=0}^{\infty}\varepsilon_k < \infty$ and $\sum_{k=0}^{\infty}\delta_k < \infty$ and $\|\nabla\psi_k(\xi^{k+1})\|$ is sufficiently small.

Semismooth Newton method

We can exploit the second order sparsity,



Semismooth Newton method

- We solve the linear system,

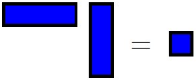
$$(\mathbb{I}_m + \sigma \mathcal{A} P \mathcal{A}^T)d = -\nabla \psi(\xi)$$

- We let $\mathcal{J} := \{j \mid |x_j| > \sigma\lambda, \ j = 1, ..., n\}$ and $|\mathcal{J}| = r$, the cardinality of $\mathcal{J}$. Then we have

$$\mathcal{A} P \mathcal{A}^T = (\mathcal{A}P)(\mathcal{A}P)^T = \mathcal{A}_{\mathcal{J}} \mathcal{A}_{\mathcal{J}}^T$$

- For the case when $p << m$, instead of factorising an $m \times m$ matrix, we can invert a much smaller, $p \times p$, matrix by using the Sherman-Morrison-Wood formula,

$$(\mathbb{I}_m + \sigma \mathcal{A} P \mathcal{A}^T)^{-1} = (\mathbb{I}_m + \sigma \mathcal{A}_{\mathcal{J}} \mathcal{A}_{\mathcal{J}}^T)^{-1} = \mathbb{I}_m - \mathcal{A}_{\mathcal{J}}(\sigma^{-1}\mathbb{I}_r + \mathcal{A}_{\mathcal{J}}^T \mathcal{A}_{\mathcal{J}})^{-1} \mathcal{A}_{\mathcal{J}}^T$$

$(\mathcal{A}P)^*(\mathcal{A}P) =$  $O(p^2 m * \text{sparsity})$

Semismooth Newton method

The relative KKT residual:

$$\eta = \frac{\|\bar{x} - \mathrm{Prox}_{\lambda\|\cdot\|}(\bar{x} - A^*(A\bar{x} - b))\|}{1 + \|\bar{x}\| + \|A\bar{x} - b\|} < \varepsilon$$

Number of nonzeros: $\mathrm{nnz} := \min\left\{ k \mid \sum_{t=1}^{k} |\hat{x}_t| \geq 0.999\|x\|_1 \right\}$

Datasets

Data	Dimension(m; n)	$\lambda_{max}(AA^*)$
pyrim5	74; 201376	1.22e+06
triazines4	186; 635376	2.07e+07
abalone7	4177; 6435	5.21e+05
bodyfat7	252; 116280	5.29e+04
housing7	506; 77520	3.28e+05
mpg7	392; 3432	1.28e+04
space_ga9	3107; 5005	4.01e+03

Table: UCI and Statlib Testing Instances

- Methylation data - 450k CpG methylation sites for 656 patients, and their ages

Implementation details - overview

How was the algorithm implemented and tested in code?

1. First, we installed MATLAB, cloned the SuiteLasso package.

Implementation details - overview

How was the algorithm implemented and tested in code?

1. First, we installed MATLAB, cloned the SuiteLasso package.
2. Ensured it reproduced the results of the [Li et al, 2018] paper on UCI and Statlib data sets

Implementation details - overview

How was the algorithm implemented and tested in code?

1. First, we installed MATLAB, cloned the SuiteLasso package.
2. Ensured it reproduced the results of the [Li et al, 2018] paper on UCI and Statlib data sets
3. Then, we made drafts in R of the original MATLAB code with reference to the pseudocode in [Li et al, 2018]

Implementation details - overview

How was the algorithm implemented and tested in code?

1. First, we installed MATLAB, cloned the SuiteLasso package.
2. Ensured it reproduced the results of the [Li et al, 2018] paper on UCI and Statlib data sets
3. Then, we made drafts in R of the original MATLAB code with reference to the pseudocode in [Li et al, 2018]
4. Changed norms, matrix-vector algebra functions, preconditioning functions, workflow and parameterisation as necessary.

Implementation details - debugging

To transform the drafts into working code, the packages were debugged side-by-side as follows:

- Breakpoints were introduced in both packages after every major event.

Implementation details - debugging

To transform the drafts into working code, the packages were debugged side-by-side as follows:

- Breakpoints were introduced in both packages after every major event.
- The numerical equality of key values was ensured
  - Primal objective
  - Dual objective
  - Relative difference
  - Parameter vectors x and x_i
  - σ for matrix scaling, etc...

Implementation details - debugging

To transform the drafts into working code, the packages were debugged side-by-side as follows:

- Breakpoints were introduced in both packages after every major event.
- The numerical equality of key values was ensured
  - Primal objective
  - Dual objective
  - Relative difference
  - Parameter vectors x and x_i
  - σ for matrix scaling, etc...
- Where we wanted to test the equality of a vector, equality of a suitable norm was taken as a substitute.

Implementation details - debugging

To transform the drafts into working code, the packages were debugged side-by-side as follows:

- Breakpoints were introduced in both packages after every major event.
- The numerical equality of key values was ensured
    - Primal objective
    - Dual objective
    - Relative difference
    - Parameter vectors x and x_i
    - σ for matrix scaling, etc...
- Where we wanted to test the equality of a vector, equality of a suitable norm was taken as a substitute.
- This revealed easily-fixed bugs in the R code when they occurred.

Implementation details - debugging

To transform the drafts into working code, the packages were debugged side-by-side as follows:

- Breakpoints were introduced in both packages after every major event.
- The numerical equality of key values was ensured
    - Primal objective
    - Dual objective
    - Relative difference
    - Parameter vectors x and x_i
    - σ for matrix scaling, etc...
- Where we wanted to test the equality of a vector, equality of a suitable norm was taken as a substitute.
- This revealed easily-fixed bugs in the R code when they occurred.
- To begin, this was only performed on *abalone7* data at one particular value of λ

Implementation details

1. For the solution of the subproblem, the logical switch on how to invert the sparse Hessian was taken from [Li et al, 2018] and implemented in R, with the same debugging process as above.

Implementation details

1. For the solution of the subproblem, the logical switch on how to invert the sparse Hessian was taken from [Li et al, 2018] and implemented in R, with the same debugging process as above.
2. For the first several iterations on *abalone7* data, the R code was able to generate the same results on key variables as the MATLAB code.

Implementation details

1. For the solution of the subproblem, the logical switch on how to invert the sparse Hessian was taken from [Li et al, 2018] and implemented in R, with the same debugging process as above.
2. For the first several iterations on *abalone7* data, the R code was able to generate the same results on key variables as the MATLAB code.
3. Towards the end of the algorithm, very slight differences (on the order of 10^{-5}, and relatively small also) were observed

Implementation details

1. For the solution of the subproblem, the logical switch on how to invert the sparse Hessian was taken from [Li et al, 2018] and implemented in R, with the same debugging process as above.
2. For the first several iterations on *abalone7* data, the R code was able to generate the same results on key variables as the MATLAB code.
3. Towards the end of the algorithm, very slight differences (on the order of 10^{-5}, and relatively small also) were observed
4. We attributed to different handling of floating point extrema.

Implementation details

1. For the solution of the subproblem, the logical switch on how to invert the sparse Hessian was taken from [Li et al, 2018] and implemented in R, with the same debugging process as above.
2. For the first several iterations on *abalone7* data, the R code was able to generate the same results on key variables as the MATLAB code.
3. Towards the end of the algorithm, very slight differences (on the order of 10^{-5}, and relatively small also) were observed
4. We attributed to different handling of floating point extrema.
5. However, this did lead in general to the R code taking a few more iterations than MATLAB.

Implementation details

- Nevertheless, after de-scaling and transforming the output vectors, we obtained the exact same results as the MATLAB code in terms of objective values, minimum and maximum parameter estimates and number of non-zeros calculated according to:
- nnz $:= \min \left\{ k \mid \sum_{t=1}^{k} |\hat{x}_t| \geq 0.999 \|x\|_1 \right\}$

Overall results

Data	λ_c	Time (s)	Objective Value
		Matlab — R-SSNAL — *glmnet*	Matlab — R-SSNAL — *glmnet*
pyrim5	10^{-3}	2.16 — 9.92 — 0.40	0.07511 — 0.07511 — 0.0795
(74;201376)	10^{-4}	2.63 — 27.82 — 0.26	0.0109 — 0.0108 — 0.0260
triazines4	10^{-3}	13.79 — 170.92 — 3.3	0.5452 — 0.5452 — 0.5548
(186;635376)	10^{-4}	27.90 — 1580.88 — 5.48	0.1156 — 0.1156 — 0.1524
abalone7	10^{-3}	1.95 — 6.84 — 1.04	11407 — 11407 — 12158
(4177;6435)	10^{-4}	3.47 — 18.14 — 2.48	9289 — 9289 — 9716
bodyfat7	10^{-3}	1.64 — 5.00 — 0.30	0.2925 — 0.2925 — 1.334
(252;116280)	10^{-4}	2.27 — 7.52 — 0.98	0.03031 — 0.03031 — 0.2372
housing7	10^{-3}	2.92 — 13.88 — 0.60	2775 — 2775 — 2819
(506;77520)	10^{-4}	2.27 — 7.52 — 0.98	920.3 — 920.3 — 987.1
mpg7	10^{-3}	0.32 — 1.02 — 0.04	1669 — 1669 — 2076
(392;3432)	10^{-4}	0.37 — 2.90 — 0.16	890 — 890 — 985.57
space_ga9	10^{-3}	0.81 — 2.34 — 0.10	31.9 — 31.9 — 62.08
(3107;5005)	10^{-4}	2.27 — 7.52 — 0.98	19.88 — 19.88 — 31.63

Table: Performance comparisons of SSNAL in Matlab and R and *glmnet*

Overall results

Data	λ_c	min	max	NNZ
		Matlab — R-SSNAL — *glmnet*	Matlab — R-SSNAL — *glmnet*	Matlab — R-SSNAL — *glmnet*
pyrim5	10^{-3}	-0.0422 — -0.0422 — -0.1732	0.165 — 0.166 — 0.1067	70 — 70 — 166
(74;201376)	10^{-4}	-0.0897 — -0.0896 — -0.63454	0.172 — 0.172 — 0.064	77 — 78 — 1643
triazines4	10^{-3}	-0.163 — -0.163 — -0.163	0.161 — 0.161 — 0.182	565 — 572 — 292
(186;635376)	10^{-4}	-0.458 — -0.458 — -0.4525	0.300 — 0.296 — 0.2465	261 — 261 — 1573
abalone7	10^{-3}	-8.13 — -8.13 — -13.49	11.7 — 11.7 — 11.7	24 — 24 — 21
(4177;6435)	10^{-4}	-13.3 — -13.3 — -12.97	16.1 — 16.1 — 7.96	59 — 59 — 129
bodyfat7	10^{-3}	-0.0465 — -0.0465 — -0.8133	1.05 — 1.05 — 1.202	2 — 2 — 17
(252;116280)	10^{-4}	-0.0526 — -0.0526 — -1.06	1.05 — 1.045 — 1.314	3 — 3 — 49
housing7	10^{-3}	-7.37 — -7.37 — -8.02	3.25 — 3.25 — 4.114	158 — 158 — 163
(506;77520)	10^{-4}	-13.1 — -13.1 — -19.7	11.3 — 11.27 — 8.39	281 — 281 — 484
mpg7	10^{-3}	-5.08 — -5.08 — -23.68	17 — 16.98 — 14.99	47 — 47 — 46
(392;3432)	10^{-4}	-11.8 — -11.8 — -18.93	15.3 — 15.3 — 16.38	128 — 128 — 172
space_ga9	10^{-3}	-1.14 — -1.14 — -3.68	0.978 — 0.978 — 3.77	14 — 14 — 19
(3107;5005)	10^{-4}	-3.56 — -3.56 — -4.59	2.64 — 2.64 — 4.71	38 — 38 — 58

Table: Performance comparisons of **SSNAL** in **Matlab** and **R** and *glmnet*

Implementation details

1. The algorithm was profiled.

Implementation details

1. The algorithm was profiled.
2. Slow steps required either a reworking of the matrix algebra (matrix-free methods).

Implementation details

1. The algorithm was profiled.
2. Slow steps required either a reworking of the matrix algebra (matrix-free methods).
3. For very large problems, we implemented fast algorithms suggested on *StackOverflow* in RCpp, which can execute fast C++ code as an R function.

Implementation details

1. The algorithm was profiled.
2. Slow steps required either a reworking of the matrix algebra (matrix-free methods).
3. For very large problems, we implemented fast algorithms suggested on *StackOverflow* in RCpp, which can execute fast C++ code as an R function.
4. We then exported the (polynomial basis expanded) data from MATLAB to R and compared it to the glmnet package using a manual 10-fold cross-validation for the seven UCI data sets.

Implementation details

1. The algorithm was profiled.
2. Slow steps required either a reworking of the matrix algebra (matrix-free methods).
3. For very large problems, we implemented fast algorithms suggested on *StackOverflow* in RCpp, which can execute fast C++ code as an R function.
4. We then exported the (polynomial basis expanded) data from MATLAB to R and compared it to the glmnet package using a manual 10-fold cross-validation for the seven UCI data sets.
5. The objective function for *glmnet* was suitably transformed.

Implementation details

1. The algorithm was profiled.
2. Slow steps required either a reworking of the matrix algebra (matrix-free methods).
3. For very large problems, we implemented fast algorithms suggested on *StackOverflow* in RCpp, which can execute fast C++ code as an R function.
4. We then exported the (polynomial basis expanded) data from MATLAB to R and compared it to the glmnet package using a manual 10-fold cross-validation for the seven UCI data sets.
5. The objective function for *glmnet* was suitably transformed.
6. Methylation data could not undergo a cross-validation because the algorithm did not converge in sensible time, but we did show better objective values around the optimal lambda obtained by *cv.glmnet*

Implementation details

1. The algorithm was profiled.
2. Slow steps required either a reworking of the matrix algebra (matrix-free methods).
3. For very large problems, we implemented fast algorithms suggested on *StackOverflow* in RCpp, which can execute fast C++ code as an R function.
4. We then exported the (polynomial basis expanded) data from MATLAB to R and compared it to the glmnet package using a manual 10-fold cross-validation for the seven UCI data sets.
5. The objective function for *glmnet* was suitably transformed.
6. Methylation data could not undergo a cross-validation because the algorithm did not converge in sensible time, but we did show better objective values around the optimal lambda obtained by *cv.glmnet*
7. Altogether, around 1,800 lines of code.

Cross-validation

- Split the data into equal portions, i.e. 10 parts

Cross-validation

- Split the data into equal portions, i.e. 10 parts
- Use $\frac{9}{10}$ as a train set

Cross-validation

- Split the data into equal portions, i.e. 10 parts
- Use $\frac{9}{10}$ as a train set
- Make predictions for the remaining $\frac{1}{10}$ and compute mean-squared error for that *fold*

Cross-validation

- Split the data into equal portions, i.e. 10 parts
- Use $\frac{9}{10}$ as a train set
- Make predictions for the remaining $\frac{1}{10}$ and compute mean-squared error for that *fold*
- $\sum(\hat{y} - y_i)^2$

Cross-validation: UCI and Statlib

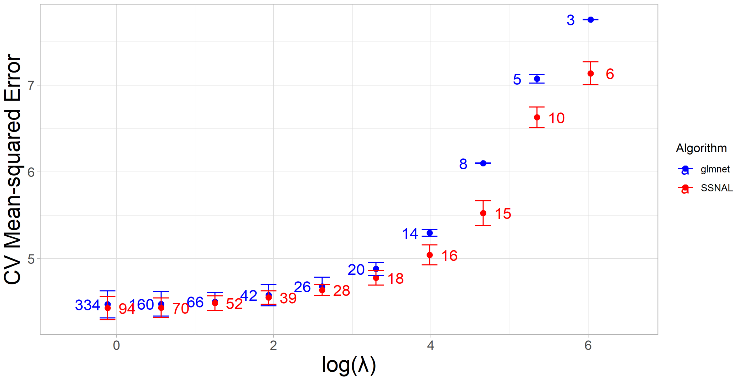
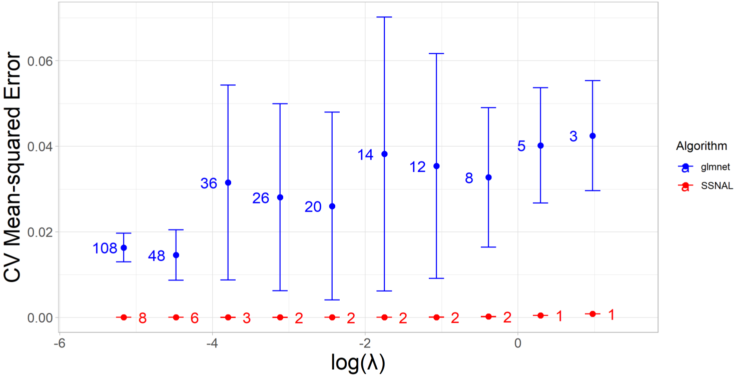


Figure: abalone

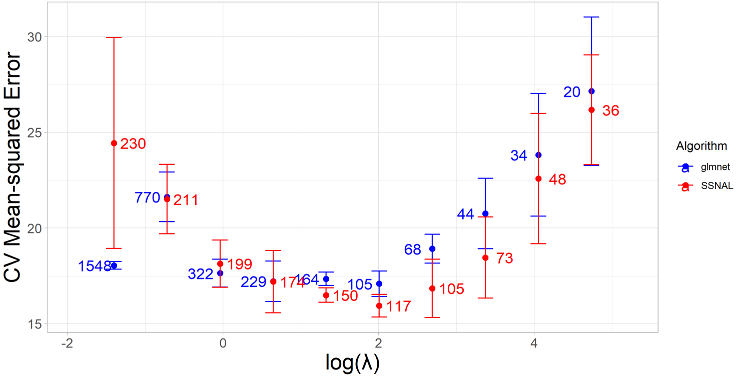
Cross-validation: UCI and Statlib

Figure: bodyfat



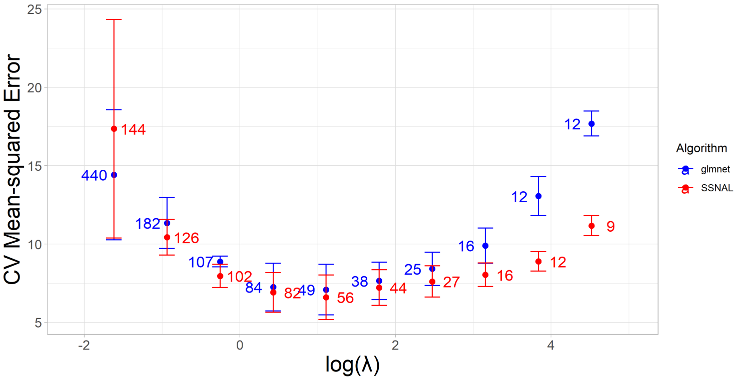
Cross-validation: UCI and Statlib

Figure: housing



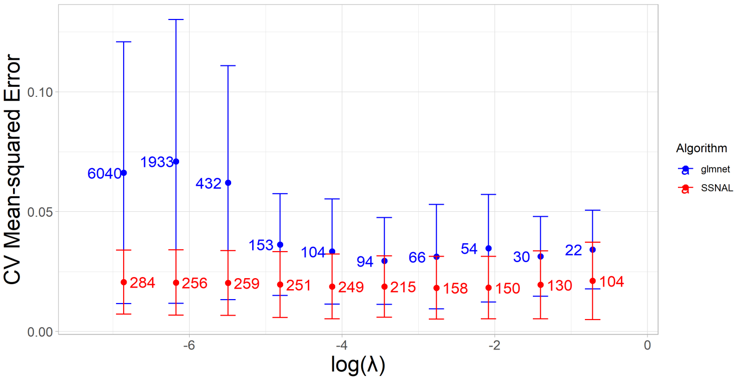
Cross-validation: UCI and Statlib

Figure: mpg



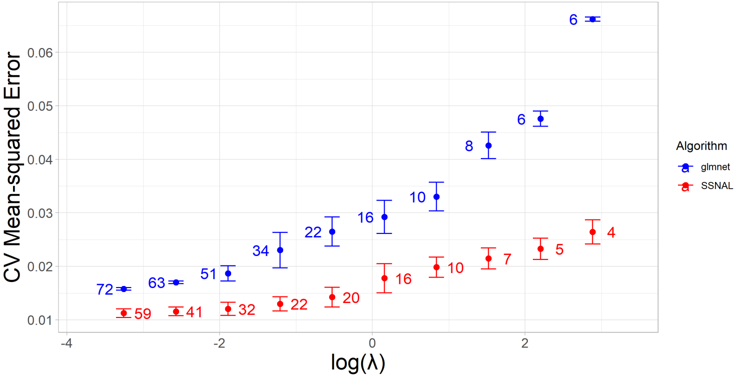
Cross-validation: UCI and Statlib

Figure: pyrim



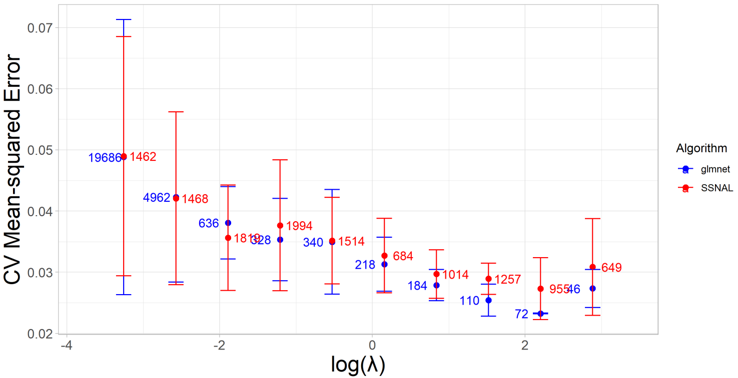
Cross-validation: UCI and Statlib

Figure: space_ga



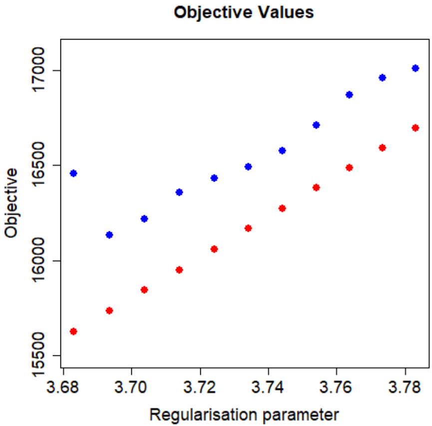
Cross-validation: UCI and Statlib

Figure: triazines



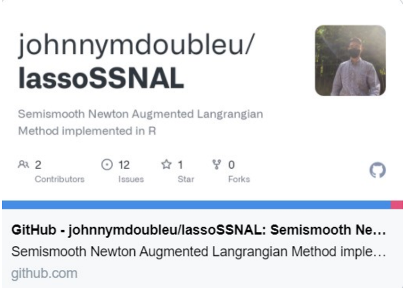
Objective values: methylation data

Figure: Methylation data



Code

▸ Github Repository



Conclusion

1. SSNAL : fast and highly efficient two-phase Semismooth Newton augmented Lagrangian methods for solving large scale Lasso problems.

Conclusion

1. SSNAL : fast and highly efficient two-phase Semismooth Newton augmented Lagrangian methods for solving large scale Lasso problems.
2. SSNAL : reproduced and presented the algorithm in R package with C++. (Currently, aim to publish on CRAN)

Conclusion

1. SSNAL : fast and highly efficient two-phase Semismooth Newton augmented Lagrangian methods for solving large scale Lasso problems.
2. SSNAL : reproduced and presented the algorithm in R package with C++. (Currently, aim to publish on CRAN)
3. Experiment: UCI, Statlib and Methylation dataset and obtained a promising result with additional comparison against *glmnet*

Conclusion

1. SSNAL : fast and highly efficient two-phase Semismooth Newton augmented Lagrangian methods for solving large scale Lasso problems.
2. SSNAL : reproduced and presented the algorithm in R package with C++. (Currently, aim to publish on CRAN)
3. Experiment: UCI, Statlib and Methylation dataset and obtained a promising result with additional comparison against *glmnet*
4. Future Work: on developing a more generalised SSNAL method to apply on Elastic Net, fused Lasso and many more.

Thank you for your attention!