Methods for Solving LASSO

Weijie Chen*
Department of Physics
Peking University
1500011335
1500011335@pku.edu.cn

Abstract

In statistics and machine learning, lasso (least absolute shrinkage and selection operator; also Lasso or LASSO) is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces. In this report, I will show main ideas or algorithms of a series of methods for solving LASSO.

1 Description to LASSO

$$\min_x \quad \frac{1}{2}\|Ax-b\|_2^2+\mu\|x\|_1 \tag{1}$$

where $A\in\mathbb{R}^{m\times n}$, $b\in\mathbb{R}^m$ and $\mu>0$ are given.

1.1 Dual problem of Lasso

From the formalism of dual problem, we can derive the dual form of lasso as following.

$$\begin{aligned}&\min -b^Ty+\frac{1}{2}||y||_2^2\\&s.t.||A^Ty||_\infty\le\mu\end{aligned} \tag{2}$$

1.2 Parameters

$n=1024 \quad m=512$

A is an m-by-n matrix of normally distributed random numbers.

u is an sparse n-by-1 vector of 10% nonzero normally distributed random numbers.

$b=Au$

$\mu=1e-3$

2 Using Mosek and Gurobi

2.1 calling Mosek and Gurobi by CVX

Because the primal problem is convex problem, we can using CVX to solve LASSO directly without any reformulation.

*Pre-admission 2019 PKU AAIS

Course: Convex Optimization (2018 Fall), Peking University, Beijing, China.

2.2 calling Mosek and Gurobi directly

Reformulating the primal problem as a quadratic program with box constraints

$$\begin{aligned} \min \quad & \frac{1}{2}\|y\|_2^2 + \mu 1^T(x^+, x^-) \\ s.t. \quad & A(x^+ - x^-) - y = b \\ & 0 \preceq x^- \\ & 0 \preceq x^+ \end{aligned} \tag{3}$$

Then we can call $mskqpopt$ to solve this quadractic program above.

For Gurobi, we reformulate the primal problem as following.

$$\begin{aligned} \min \quad & \frac{1}{2}\|A(x^+ - x^-) - b\|_2^2 + \mu 1^T(x^+, x^-) \\ s.t. \quad & 0 \preceq x^- \\ & 0 \preceq x^+ \end{aligned} \tag{4}$$

In the same way, we can call $gurobi$ to solve this quadratic program.

3 Widely used algorithms

All of these methods contain the continuation strategy when implementing on MATLAB.

The main idea of continuation strategy is to gradually decrease μ from a large value to the target value, which can accelerate the convergent rate from initial value to optimal value.

3.1 Projection Gradient Method with box constraints

Reformulating the primal problem as a quadratic program with box constraints4, then the gradient g of primal problem has simple form and the projector $P(x)$ is $\max(x, 0)$.

Algorithm 1: Projection Gradient Method

Input: Initial vector x_0,tolerance error ϵ, Max iteration number N and learning rate α,α_{max},α_{min}
Output: solution x

calculate g_0 **while** $\|P(x_k - g_k) - x_k\| > \epsilon$ *and* $k < N$ **do**
    $x_{k+1} := P(x_k - \alpha g_k)$
    $s_k := x_{k+1} - x_k$
    $y_k := g_{k+1} - g_k$
    if $s_k^T y_k \leq 0 \quad \alpha := \alpha_{max}$
    else $\alpha := \max(\min(\frac{s_k^T s_k}{s_k^T y_k}, \alpha_{max}), \alpha_{min})$
    $k := k + 1$
end
return $x = x_k$;

3.2 Subgradient Method

From the definition of subgradient, we can generalize gradient descent method to subgradient method.

g is a subgradient of a convex function f at $x \in \operatorname{dom} f$ if

$$f(y) \geq f(x) + g^\top (y - x) \quad \forall y \in dom f \tag{5}$$

For LASSO, the subgradient as follow

$$g = A^T(Ax - b) + \mu sign(x) \tag{6}$$

Algorithm 2: Subgradient Method

Input: Initial vector x_0,tolerance error ϵ,decay rate δ, decay frequency M and learning rate α_0
Output: solution x

while $\|x_{k+1} - x_k\| > \epsilon$ *and* $k < N$ **do**
    calculate subgradient g_k
    $\alpha_k = \alpha_0(1 + 2/\sqrt{k})$ where α_k is diminishing step size.
    $x_{k+1} := x_k - \alpha_k g_k$
    if $k \equiv 0 \pmod{M}$ $\alpha_0 := \delta\alpha_0$
    $k := k + 1$
end
return $x = x_k$;

3.3 Smoothed Primal Problem

As is known to all, norm-1 function isn't differentiable. Thus,we can only use its subgradient when solve it by gradient method.

However, on the other hand, we can substitute norm-1 to another differentiable functions. After that, we can using gradient method dircetly for solving the smoothed problem.

The most direct substitute function is element-wise norm-2 function (Huber Penalty) in the neighbourhood of $x = 0$.

$$\phi_\lambda(x) = \begin{cases} x^2/(2\lambda) & |x| \le \lambda \\ |x| - \lambda/2 & |x| > \lambda \end{cases} \tag{7}$$

Using this smooth penalty, we can easily write down the gradient of smoothed primal problem.

$$g_s = A^T(Ax - b) + \mu \begin{cases} x/\lambda & |x| \le \lambda \\ sign(x) & |x| > \lambda \end{cases} \tag{8}$$

3.4 Proximal Gradient Method

Proximity operator (prox-operator) of convex funtion h is defined as following.

$$prox_h(x) = argmin_u \left(\frac{1}{2} \|u - x\|_2^2 + h(u) \right) \tag{9}$$

which is well-defined because of the strict convexity of the 2-norm. The proximity operator can be seen as a generalization of a projection. For some special cases, we can write down the exact form of proximity operator.

- $h(x) \equiv 0, \quad prox_h(x) = x$
- $h(x) = \|x\|_1, \quad prox_h(x)_i = \begin{cases} x_i - 1 & x_i > 1 \\ 0 & -1 \le x_i \le 1 \\ x_i + 1 & x_i < -1 \end{cases}$

For general unconstrained optimization with objective split in two components

$$\min_x f(x) = g(x) + h(x) \tag{10}$$

where g convex and differentiable, but h is a convex non-differentiable function with inexpensive prox-operator. We see that the proximity operator is important because x^* is a minimizer to the problem $\min_x g(x) + h(x)$ if and only if (where $\gamma > 0$ is any positive real number)

$$x^* = prox_{\gamma h}(x^* - \gamma \nabla g(x^*)) \tag{11}$$

That means x^* is a fixed point of this equation. So, we can easily introduce to a iterative method called **Proximal Gradient Method**.

$$x_k = prox_{t_k h}\left(x_{k-1} - t_k \nabla g(x_{k-1})\right) \tag{12}$$

3.5 Fast Proximal Gradient Method (FISTA)

Fast Proximal Gradient Method, also called FISTA, whose acronym stands for **Fast Iterative Shrinkage-Thresholding Algorithm**, is an accelerated first-order optimization algorithm.

Algorithm 3: Fast Proximal Gradient Method

Input: Initial vector x_0,tolerance error ϵ and initial learning rate α_0
Output: solution x

initializing velocity $v_0 := x_0$
while $\|x_{k+1} - x_k\| > \epsilon$ *and* $k < N$ **do**
  $\theta_k = \frac{2}{k+1}$
  $y = (1-\theta_k)x_{k-1} + \theta_k v_{k-1}$
  $x_k := prox_{\alpha_k h}\left(x_{k-1} - \alpha_k \nabla g(x_{k-1})\right)$
  $v_k = x_{k-1} + \frac{1}{\theta_k}(x_k - x_{k-1})$
  $k := k + 1$
end
return $x = x_k$;

If $h \equiv 0$, fast proximal gradient method can degenerate to fast gradient method. It can be used to accelerate the smoothed primal problem.

3.6 Augmented Lagrangian Method for dual problem

For inducing augmented lagrangian method, we can reformulate the dual problem via spliting.

$$\begin{aligned} &\min f(y) = -b^T y + \frac{1}{2}||y||_2^2 + \mathrm{I}_{\{||z||_\infty \le \mu\}} \\ &s.t.\ z = A^T y \end{aligned} \tag{13}$$

where I_C is an indicator function of range C.

Then, we can directly write down the augmented lagrangian of dual problem and the iterative method.

$$\begin{aligned} &\mathcal{L}(y,z,x) = -b^T y + \frac{1}{2}||y||_2^2 + \frac{\beta}{2}||(A^T y + \frac{x}{\beta}) - z||_2^2 + \mathrm{I}_{\{||z||_\infty \le \mu\}} \\ &(y^+, z^+) = \underset{y,z}{\operatorname{argmin}}\ \mathcal{L}(y,z,x) \end{aligned} \tag{14}$$

Algorithm 4: Augmented Lagrangian method for the dual problem

Input: hyperparameter γ, β, optimization variables y, x, z, maximum iteration N
Output: solution $f(y)$

initializing $k = 1$
while $k < N$ **do**
  $y := \underset{y}{\operatorname{argmin}}\{-b^T y + \frac{1}{2}||y||_2^2 + \frac{\beta}{2}||(A^T y + \frac{x}{\beta}) - \phi(A^T y + \frac{x}{\beta})||_2^2\}$
  $z := \phi(A^T y + \frac{x}{\beta})$
  $x := x - \gamma(A^T y - z)$
  $k := k + 1$
end
return x

where $\phi(u)_i = \begin{cases} \mu & u_i > \mu \\ u_i & |u_i| \le \mu \\ -\mu & u_i < -\mu \end{cases}$ is the projection on a ∞-norm ball.
It's easy to see that $\psi(u) = u - \phi(u)$ is soft-threshholding.

3.7 Alternating direction method of multipliers (ADMM) for dual problem

From the above algorithm, we notice that the joint minimum value of the augmented lagrangian isn't always explicit and inexpensive. However, coordinate minimizing is usually inexpensive when another variables fixed. That is the intuition of **alternating direction method of multipliers**.

In each iteration:

$$\begin{aligned} y^+ &= \underset{y}{\operatorname{argmin}} - b^T y + \frac{1}{2}||y||_2^2 + \mathrm{I}_{\{||z||_\infty \le \mu\}} + \frac{\beta}{2}||(A^T y + \frac{x}{\beta}) - z||_2^2 \\ &= (I + \beta A A^T)^{-1}(\beta A z + b - A x) \\ z^+ &= \underset{z}{\operatorname{argmin}} - b^T y^+ + \frac{1}{2}||y^+||_2^2 + \mathrm{I}_{\{||z||_\infty \le \mu\}} + \frac{\beta}{2}||(A^T y^+ + \frac{x}{\beta}) - z||_2^2 \\ &= \phi(A^T y^+ + \frac{x}{\beta}) \\ x^+ &= x + \gamma(A^T y^+ - z^+) \end{aligned} \tag{15}$$

Algorithm 5: Alternating direction method of multipliers for the dual problem

Input: hyperparameter γ, β, optimization variables y, x, z, maximum iteration N
Output: solution $f(y)$

initializing $k = 1$
while $k < N$ **do**
    $y := (I + \beta A A^T)^{-1}(\beta A z + b - A x)$
    $z := \phi(A^T y + \frac{x}{\beta})$
    $x := x + \gamma(A^T y - z)$
    $k := k + 1$
end
return x

3.8 Alternating direction method of multipliers (ADMM) with linearization for the primal problem

In order to apply ADMM for the primal problem, we need to reformulate lasso via spliting:

$$\begin{aligned} &\min \frac{1}{2}\|Ax - b\|_2^2 + \mu\|z\|_1 = f(x) + g(z) \\ &s.t.\ x - z = 0 \end{aligned} \tag{16}$$

The augmented lagrangian is :

$$\frac{1}{2}\|Ax - b\|_2^2 + \mu\|z\|_1 + \frac{\beta}{2}||x - z + u||_2^2 \tag{17}$$

So the ADMM iterative algorithm is:

$$\begin{aligned} x &\leftarrow (A^T A + \beta I)^{-1}(A^T b + \beta(z - u)) \\ z &\leftarrow prox_{g/\beta}(x + u) \\ u &\leftarrow u + \gamma(x - z) \end{aligned} \tag{18}$$

If we adopt linearization (single gradient-descent step), the update for x is changed to:

$$x \leftarrow x - c\left(\nabla f\left(x\right) + \beta\left(x - z + u\right)\right) \tag{19}$$

4 Some extra algorithms

4.1 Adagrad

In order to improve gradient method, it's natural to tune step size for each iteration. Adagrad is an algorithm for gradient-based optimization that does just this: It adapts the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features.

Algorithm 6: Adaptive gradient Method

Input: Initial vector x_0, tolerance error ϵ, learning rate α and constant δ
Output: solution x

while $\|x_{k+1} - x_k\| > \epsilon$ *and* $k < N$ **do**
    $g \in \partial f(x_k)$
    $r := r + g \odot g$
    $x_{k+1} := x_k - \left(\frac{\alpha}{\delta + \sqrt{r}} \odot g\right)$ where δ prevents denominator to be 0.
    $k := k + 1$
end
return $x = x_k$;

4.2 Momentum

Momentum is a method that helps accelerate Gradient Descent (especially, Stochastic GD) in the relevant direction and dampens oscillations.

Algorithm 7: Momentum Method

Input: Initial vector x_0, initial momentum v, tolerance error ϵ, learning rate β, α
Output: solution x

while $\|x_{k+1} - x_k\| > \epsilon$ *and* $k < N$ **do**
    $g \in \partial f(x_k)$
    $v = \alpha v - \beta g$
    $x_{k+1} := x_k + v$
    $k := k + 1$
end
return $x = x_k$;

4.3 RMSProp

RMSprop developed stemming from the need to resolve Adagrad's radically diminishing learning rates.

Algorithm 8: RMSProp Method

Input: Initial vector x_0, initial momentum v, tolerance error ϵ, learning rate α, constant δ and hyperparameter ρ
Output: solution x

while $\|x_{k+1} - x_k\| > \epsilon$ *and* $k < N$ **do**
    $g \in \partial f(x_k)$
    $r = \rho r + (1 - \rho) g \odot g$
    $x_{k+1} := x_k - \frac{\alpha}{\delta + \sqrt{r}} \odot g$
    $k := k + 1$
end
return $x = x_k$;

4.4 Adam

Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients r like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients s , similar to momentum. Whereas momentum can be seen as a ball running down a slope, Adam behaves like a heavy ball with friction, which thus prefers flat minima in the error surface. We compute the decaying averages of past and past squared gradients s and r respectively as follows:

Algorithm 9: Adaptive Moment Estimation Method

Input: Initial vector x_0,tolerance error ϵ,decay rate α, constant δ and hyperparameter ρ_1, ρ_2
Output: solution x

while $\|x_{k+1} - x_k\| > \epsilon$ *and* $k < N$ **do**
  $g \in \partial f(x_k)$
  $s = \rho_1 s + (1 - \rho_1)g$
  $r = \rho_2 r + (1 - \rho_2)g \odot g$
  $\hat{s} = \frac{s}{1-\rho_1^k}$
  $\hat{r} = \frac{r}{1-\rho_2^k}$
  $x_{k+1} = x - \frac{\alpha \hat{s}}{\delta + \sqrt{\hat{r}}}$
  $k := k + 1$
end
return $x = x_k$;

5 Numerical result and Interpretation

The cpu time, error with cvx-mosek and optimal value of different methods were shown below. The implemention of these methods had been shown in each *.m file.

Method	cpu time/s	error with cvx-mosek	optimal value
cvx-call-mosek	4.26	0.00e+00	0.072844 269 1955
cvx-call-gurobi	8.89	6.02e-08	0.072844 271 0095
call-mosek	1.41	4.19e-06	0.072845 159 0901
call-gurobi	2.98	9.92e-06	0.072844 886 1733
projection gradient	0.40	5.96e-08	0.072844 266 1115
subgradient	3.40	1.97e-07	0.072844 268 1299
smoothgradient	3.78	6.24e-08	0.072844 269 7707
fastsmoothgradient	3.21	6.26e-08	0.072844 267 9946
proxgradient	1.56	4.02e-07	0.072844 267 2131
fastproxgradient	0.23	2.29e-07	0.072844 266 5441
ALM	1.44	9.03e-08	0.072844 266 5079
ADMM dual	0.40	2.56e-07	0.072844 266 8990
ADMM primal	3.66	5.77e-08	0.072844 267 0477
AdaGrad	1.96	8.20e-07	0.072844 382 2341
Adam	1.03	6.96e-07	0.072844 340 4748
RMSProp	1.57	1.30e-06	0.072844 372 1195
Momentum	1.89	8.89e-07	0.072844 389 6123

Table 1: Numerical results of different methods

We can easily find the efficency and accuracy of projection gradient method are both the best. After tuning parameters of subgradient method and introduce diminishing learning rate, its efficency and accuracy are also greater than calling Mosek by CVX.

The best performance of projection gradient method is reasonable, because it's just like unlimit optimization before constrainting into the domain. Besides, using BB step size is nearly a second order descent method.

References

[1] Boyd, S. & Vandenberghe, L. (2004). Convex Optimization. Cambridge: Cambridge University Press.

[2] Ian Goodfellow, Yoshua Bengio , Aaron Courville, (2016). Deep Learning. The MIT Press.