Lasso 优化算法的探讨和比较

李宗翰，蒋佩禧，袁浩然

2020 年 7 月 10 日

大作业分工

算法讨论实现：李宗翰, 袁浩然, 蒋佩禧
代码编写：李宗翰, 蒋佩禧
PPT 制作：袁浩然, 李宗翰
大作业文档写作：蒋佩禧, 李宗翰
大作业展示：李宗翰

目录

Lasso

ℓ_1 -regularized least squares (LS) problem:

$$\min \left\{ \frac{1}{2} \|\mathcal{A}x - b\|^2 + \gamma \|x\|_1 \right\}$$

CVX

CVX is a modeling system for constructing and solving disciplined convex programs (DCPs).[1]
CVX 作为用多方法进行数值实验的数值标准

[1] en2.

CVX

```
cvx_begin quiet
cvx_precision low
variable x(n)
minimize(0.5*sum_square(A*x - b) + gamma*norm(x,1))
cvx_end
```

Proximal gradient method

$$f(x) = (1/2)\|Ax - b\|_2^2, \quad g(x) = \gamma\|x\|_1$$

Consider the problem

$$\text{minimize} \quad f(x) + g(x)$$

近端算子（Proximal Operator）

$$\text{prox}_h(w) = \arg\min_u \left\{ h(u) + \frac{1}{2}\|u - w\|_1^2 \right\}$$

其中 $prox_h(w)$ 表示变量 w 和函数 $h(.)$ 的近端算子。上面的公式的意义是：对于任意给定的 $w \in R^n$, 我们希望找到使得 $h(u) + \frac{1}{2}\|u - w\|_2^2$ 最小化的解[2]

[2] **en1**.

Proximal gradient method

$$
\begin{aligned}
x^{(k+1)} &= \operatorname{argmin}_{\mathbf{z}}(f(z)+g(z)) \\
&= \operatorname{argmin}_{\mathbf{z}}\left(f\left(x^{(k)}\right)+\nabla f\left(x^{(k)}\right)^T\left(z-x^{(k)}\right)+\frac{1}{2\lambda}\left\|z-x^{(k)}\right\|_1^2+g(z)\right) \\
&= \operatorname{argmin}_{z}\left(\frac{1}{2\lambda}\left\|z-\left(x-\lambda\nabla f\left(x^{(k)}\right)\right)\right\|_1^2+g(z)\right) \\
&= \operatorname{prox}_{g,y}\left(x^{(k)}-\lambda\nabla f\left(x^{(k)}\right)\right) \\
&= \operatorname{prox}_{g,y}\left(x^{(k)}-\lambda A^T\left(Ax^{(k)}-b\right)\right)
\end{aligned}
$$

3

[3] **en1**.

Proximal gradient method

算法 1 近端梯度算法

输入: x^k, λ^{k-1},

parameter $\beta \in (0,1)$

Let $\lambda := \lambda^{k-1}$

repeat

Let $z := \text{prox}_{\lambda_g}\left(x^k - \lambda \nabla f\left(x^k\right)\right)$;

break if $f(z) \leq \hat{f}_\lambda\left(z, x^k\right)$;

Update $\lambda := \beta\lambda$

return $\lambda^k := \lambda, x^{k+1} := z$

加速近端梯度算法

在近端梯度算法的基础上，更新策略如下：

$$
\begin{aligned}
y^{(k+1)} &= x^{(k)} + \tfrac{k}{k+3}\left(x^{(k)} - x^{(k-1)}\right) \\
x^{k+1} &= \operatorname{prox}_{g,y}\left(y^{(k+1)} - \lambda \nabla f\left(y^{(k+1)}\right)\right)
\end{aligned}
$$

加速近端梯度算法

算法 2 加速近端梯度算法

输入: y^k, λ^{k-1},

parameter $\beta \in (0,1)$

Let $\lambda := \lambda^{k-1}$

repeat

Let $z := \text{prox}_{\lambda g}\left(y^k - \lambda \nabla f\left(y^k\right)\right)$;

break if $f(z) \leq \hat{f}_\lambda\left(z, y^k\right)$;

Update $\lambda := \beta\lambda$

return $\lambda^k := \lambda, x^{k+1} := z$

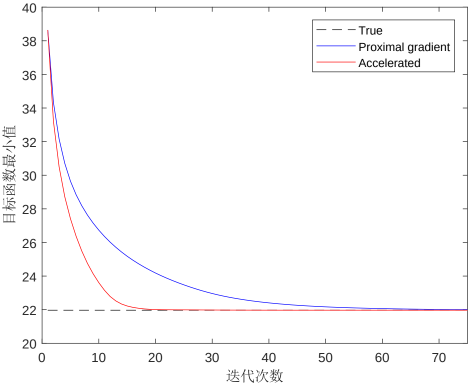

图: 近端梯度算法 vs 加速近端梯度算法

ADMM

$$\text{minimize} \quad (1/2)\|Ax - b\|^2 + \gamma\|x\|_1$$

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & x - z = 0 \end{array}$$

$$f(x) = (1/2)\|Ax - b\|^2 \text{ and } g(z) = \gamma\|z\|_1$$

ADMM

$$
\begin{aligned}
x^{k+1} &:= \left(I+\lambda A^T A\right)^{-1}\left(z^k-u^k-\lambda A^T b\right) \\
z^{k+1} &:= \operatorname{prox}_{\lambda\gamma\|\cdot\|_1}\left(x^{k+1}+u^k\right) \\
u^{k+1} &:= u^k+x^{k+1}-z^{k+1}
\end{aligned}
$$

4

[4] **en3**.

ADMM

算法 3 ADMM

输入: A, x^k, u^k, z^k

1: parameter λ, ρ

2: **repeat** $\mathrm{x}^{k+1} := \left(I + \lambda A^T A\right)^{-1}\left(z^k - u^k - \lambda A^T b\right)$;

5

[5] **en4**.

数值实验

ℓ_1 -regularized least squares (LS) problem:

$$\min\left\{\frac{1}{2}\|\mathcal{A}x-b\|^2+\gamma\|x\|_1\right\}$$

我们比较了近端迭代，加速近端迭代，ADMM，CVX 来解决这个问题

数值实验

参数设置：

$$A \in \mathbf{R}^{m \times n}, A_{ij} \sim \mathcal{N}(0,1)$$

$$x^{\text{true}} \in \mathbf{R}^{n}, b = Ax^{\text{true}} + v, v \in 0.01 * (1, m)$$

$$\gamma = 0.1\gamma_{\max}, \gamma_{\max} = \left\|A^{T}b\right\|_{\infty}$$ [6]

近端迭代的参数设置：$\lambda = 1$
$\epsilon = 10^{-4}$ 为迭代停止标准
计算平台：IntelCore i5-8265 CPU @1.60GHz 8.00GB RAM Windows10

[6] **en1**.

参数设置：A=500*2500,b=A*x0+v,
其中 x0 为 n*1 的稀疏矩阵，密度 (在矩阵中非零元素所占的比例):0.05,v 为 m*1 的随机生成的数组

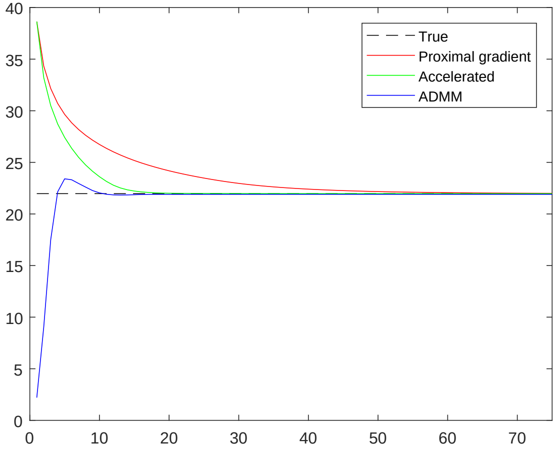

图 3: Proximal gradient,Accelerated,ADMM

Method	Iterations	Time (s)	$p^\star$	Error (abs)
CVX	15	26.53	16.5822	–
Proximal gradient	127	0.72	16.5835	0.09
Accelerated	23	0.15	16.6006	0.26
ADMM	20	0.07	16.6011	0.18

在这个计算中，近端梯度算法迭代较慢，但迭代足够多的次数后，准确性较高，ADMM 和加速近端梯度算法虽然迭代次数较少，但得出的值高于真实目标函数的值

参数设置：A=1000*3500,b=A*x0+v,
其中 x0 为 n*1 的稀疏矩阵，密度 (在矩阵中非零元素所占的比例):0.05,v 为 m*1 的随机生成的数组

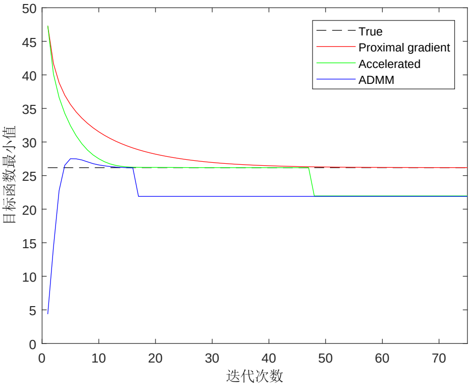

图: Proximal gradient,Accelerated,ADMM

Method	Iterations	Time (s)	$p^\star$	Error (abs)
CVX	15	144.0169	23.1788	–
Proximal gradient	70	0.9906	23.5835	0.05
Accelerated	50	0.15	26.1053	3.16
ADMM	27	0.07	26.5721	3.115

在这个计算中，近端梯度算法迭代较慢，但迭代足够多的次数后，准确性较高，ADMM 和加速近端梯度算法虽然迭代次数较少，但得出的低于真实目标函数的值

参数设置：A=600*2000,b=A*x0+v,
其中 x0 为 n*1 的稀疏矩阵，密度 (在矩阵中非零元素所占的比例):0.05,v 为 m*1 的随机生成的数组

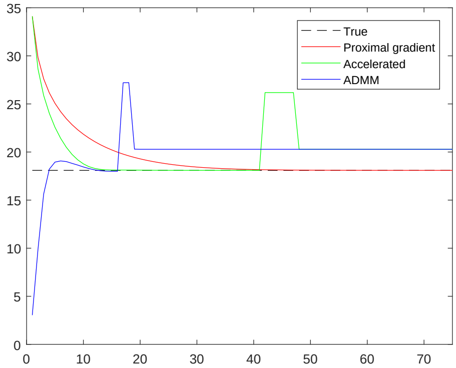

图: Proximal gradient,Accelerated,ADMM

Method	Iterations	Time (s)	$p^\star$	Error (abs)
CVX	15	27.5314	18.5822	$--$
Proximal gradient	100	0.2774	18.5835	0.07
Accelerated	59	0.1559	20.6006	2.26
ADMM	49	0.0589	20.6011	2.18

在这个计算中，近端梯度算法迭代较慢，但迭代足够多的次数后，准确性较高，ADMM 和加速近端梯度算法虽然迭代次数较少，但得出的值高于真实目标函数的值

总结

本文通过三种方式计算 lasso 问题，并对这三种算法进行介绍，且着重介绍了近端梯度算法及其加速，我们利用数值实验计算来比较三个算法的优缺点。
近端梯度算法迭代较慢，但迭代足够多的次数后，准确性较高，ADMM 和加速近端梯度算法虽然迭代次数较少，但显示出了较大的误差
将三种算法集为一体并进行比较是本文的一大优点，但是由于自身知识面所限，难免会有所错漏，有待改进

小作业分工

二分法:
代码编写：李宗翰, 袁浩然, 文档写作：蒋佩禧
牛顿迭代法:
代码编写：蒋佩禧, 李宗翰, 袁浩然, 文档写作：李宗翰

迭代法的通用算法

$$f(x) = x^2 - 1$$

```
f=sym('x^2-1');
[c,E,fc]=newton1(f,0,3,0.005,20);
```

Iter.	Aprox.	Error.
	3.0000	3.0000
1	1.6667	1.3333
2	1.1333	0.5333
3	1.0078	0.1255
4	1.0000	0.0078
5	1.0000	0.0000

牛顿法 vs 二分法

$$12 - 3x + 2\cos x = 0$$

牛顿法:

迭代次数	区间值: b	区间值: a
1	3.43828213866291	3.31995568160492
2	3.31995568160492	3.34836329704004
3	3.34836329704004	3.34741272048233
4	3.34741272048233	3.34740283960879
5	3.34741272048233	3.34740283960879

二分法

迭代次数	区间值: a	区间值: b
1	3.00000000000000	3.50000000000000
2	3.25000000000000	3.50000000000000
3	3.25000000000000	3.37500000000000
4	3.31250000000000	3.37500000000000
$5-13$	...	...
14	3.34735107421875	3.34741210937500
15	3.34738159179688	3.34741210937500
16	3.34739685058594	3.34741210937500
17	3.34739685058594	3.34740447998047
18	3.34739685058594	3.34740447998047

参考文献