Report of the Final Project

Solving the l_1 Regularized Problem Using Different Methods

Haocong Wang

Mingming Pei(mp1636)

1. Introduction

Finding the solution to the least square problem with l_1 regularization is of utmost importance since there are lots of applications including pattern recongnition, feature selection, image processing and bioinformatics.

In this project, we consider and build several methods to solve the l_1 problem:

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

which is a standard LASSO problem without constraints. We set the data as:

$$n=1024, m=512$$
$$A=randn(m,n), u=sprandn(n,1,0.1)$$
$$b=A*u, \mu=10^{-3}, x_0=rand(n,1)$$

In this project we used six different ways to solve the LASSO problem. We first consider two simple ways to solve the problem by calling Mosek and Gurobi in CVX. Then we try to use Gurobi and Mosek directly. Last but not least, we use gradient projection and subgradient method to solve the problem. We compared the execution time and the error rate with using mosek in CVS to find out which method is better at solving the l_1 regularized problem.

2. Use CVX by calling Gurobi and Mosek

We first use CVX to solve this problem by calling different solvers such as Gurobi and Mosek. CVX is a Matlab-based modeling system for convex optimization. CVX turns Matlab into a modeling language, allowing constraints and objectives to be specified using standard Matlab expression syntax [1]. We provide detailed introduction in following subsections.

2.1 Use Mosek in CVX

The code for using Mosek in CVX [2] is shown as following:

```
function [x, cvx_optval]=l1_cvx_mosek(x0, A, b, mu, opts1)
```

```
[n, ~] = size(x0);
cvx_solver mosek
cvx_begin
        variable x(n);
        minimize (mu*norm(x,1) + 0.5*norm(A*x-b))
cvx_end
end
```

2.2 Use Gurobi in CVX

The code for using Gurobi in CVX [2] is shown as following:

```
function [x, cvx_optval]=l1_cvx_gurobi(x0, A, b, mu, opts2)
[n, ~] = size(x0);
cvx_solver gurobi
cvx_begin
        variable x(n)
        minimize (mu*norm(x,1) + 0.5*norm(A*x-b))
cvx_end
end
```

3. Call Mosek and Gurobi directly

The original can be transformed into:

$$\min \frac{1}{2} \parallel Ax - b \parallel_2^2 + \mu 1^T t$$

$$s.t. x \leq t$$

$$-t \leq x$$

When $b = Au$, it can be formulated as:

$$\min \frac{1}{2} \parallel Ay \parallel_2^2 + \mu 1^T t$$

$$s.t. y + u \leq t$$

$$-t \leq y + u$$

Then we can use Mosek and Gurobi directly

3.1 Mosek

The MOSEK Optimization Suite is a powerful software package capable of solving large-scale optimization problems [3]. The code for solving this problem is shown below:

```
function [x, cvx_optval]=l1_mosek(x0, A, b, mu, opts3)
[~,n] = size(A);
q = [0.5*(A'*A), zeros(n,n);zeros(n,n),zeros(n,n)];
c = [zeros(n,1);mu*ones(n,1)];
a = [eye(n),eye(n);-eye(n),eye(n)];
```

```
v = A\b;
blc = [-v;v];
res = mskqpopt(q,c,a,blc,[],[],[],[],'minimize');
x = res.sol.itr.xx(1:1024)+v;
cvx_optval = res.sol.itr.pobjval;
end
```

3.2 Call Gurobi directly

The Gurobi Optimizer [4] is a commercial optimization solver for linear programming (LP), quadratic programming (QP), etc. Our code is shown below:

```
function [x, cvx_optval]=l1_gurobi(x0, A, b, mu, opts4)
[~,n] = size(A);
At = [A -A];
model.Q = 0.5*sparse(At'*At);
c = mu*ones(2*n,1)- (b'*At)';
model.obj =c;
P =eye(2*n);
model.A = sparse(P);
l1 = zeros(2*n,1);
model.rhs =full(l1);
model.sense = '>';
params.method = 2;
res = gurobi(model, params);
x = res.x(1:n,1)-res.x(n+1:2*n,1);
cvx_optval = 0.5 * sum_square(A * x - b) + mu * norm(x, 1);
end
```

4. Use Subgradient Algorithm

The subgradient algorithm [5] is shown below:

$$x^{(k)} = x^{(k-1)} - t_k g^{(k-1)}$$

In the above function, $g^{(k-1)}$ is the subgradient of object function when $x = x^{(k-1)}$. In our code, we use a large μ as the start point to guarentee the convergence. Apart from that, we utilize continuation method for step length α.

```
function [x, cvx_optval]=l1_Subgradient(x0, A, b, mu, opts5)
pseudomu = 100; % 10000*mu
iter = 500;
tol = 1e-7;
x = x0;
% continuation trick
while pseudomu >= mu
    k = 1;
```

```
        alpha = 3e-4;
        while k < iter
                x0 = x;
                g = A' * (A * x - b)    + pseudomu * sign(x);
                if mod(k,50)==0
                        alpha = alpha * 0.9;
                end
                x = x - alpha * g;
                if norm(x0-x) < tol
                        break;
                end
                k = k + 1;
        end
        pseudomu = pseudomu / 10;
end
cvx_optval = 0.5*sum_square(A*x - b) + mu * norm(x,1);
end
```

5. Use Projection Gradient Algorithm

For the projection gradient algorithm [6], the core iteration step is:

$$x^+ = P_C(x - t \bigtriangledown g(x))$$

We divide variable x into two parts:

$$x_1, x_2$$
$$s.t. x_1 \geq 0, x_2 \geq 0$$

We define C as $\mathbb{R}^+{}_{2n}$. Just like the implementation of subgradient algorithm, we use a large μ as the start point to guarentee the convergence.

```
function [x, cvx_optval] = l1_ProjectGradient(x0, A, b, mu, opts5)
x0 = [max(x0,0); max(-x0,0)];
[~,n] = size(A);
pseodumu = mu * 1e5;
tol = 1e-4;
alpha_l = 1e-6;
alpha_u = 1;
alpha = 1e-4;
iter = 200;
Atb = A' * b;
AtA = A' * A;
while pseodumu >= mu
        cur = pseodumu * ones(2*n,1) - [Atb; -Atb];
        AtAx0 = AtA* (x0(1:n)-x0((n+1):2*n));
        g0 = [AtAx0;-AtAx0] + cur;
        x = x0;
        g = g0;
        k = 1;
        while k < iter
                x0 = x;
                x = max(x - alpha*g, 0);
                g0 = g;
                AAz = AtA* (x(1:n)-x((n+1):2*n));
                g = [AAz;-AAz] + cur;
                y = g - g0;
                s = x - x0;
                BB = (s'*s) / (s'*y);
                if s'*y <=0
                        alpha = alpha_u;
                else
                        alpha = max(alpha_l, min(BB, alpha_u));
                end
                k = k + 1;
                if    norm(max(x-g, 0) - x) < tol
                        break
                end
        end
        pseodumu = pseodumu / 10;
end
x =    x(1:n)-x((n+1):2*n);
cvx_optval = 0.5*sum_square(A*x - b) + mu*norm(x,1);
end
```

6. Result and Conclusion

We ran our code using Matlab and compared the time consuming and the result of each algorithms. The result is shown in the following table.

Algorithm	CPU Time(s)	Error with CVX mosek	Optimal Value
CVX call gurobi	0.83	2.30e-10	7.1896382316e-02
CVX call mosek	1.95	0.00e+00	7.1896382289e-02

Gurobi	0.95	9.67e-06	7.1896693312e-02
Mosek	0.83	1.19e-05	7.1897480989e-02
Sub-gradient	0.20	3.25e-06	7.1896295861e-02
Project Gradient	0.06	2.96e-06	7.1896251697e-02

Table1-1 Results of using different methods solving Lasso Problem

From table 1-1, we have several findings. Firstly, calling mosek from CVX took much more time than using Mosek directly. Second, using CVX to call gurobi took almost the same time as using gurobi directly. It is obvious thatproject gradient method is the fastest one among six methods. Though the optimal value we got using different methods are close, the results of using CVX calling optimizer are closer. As far as we are concerned, if we pay more attention to the execution time, we can choose project gradient method to solve the unconstrained l_1 regularized problem. If we are seeking for more precision, using calling gurobi in CVX is a good way.

7. References

[1] Grant, M. C., & Boyd, S. P. (2011). CVX Research, Inc. CVX: Matlab software for disciplined convex programming, version, 2.

[2] Grant, M., Boyd, S., & Ye, Y. (2009). CVX users' guide. online: http://www. stanford. edu/~ boyd/software. html.

[3] ApS, M. (2019). Mosek optimization toolbox for matlab. User's Guide and Reference Manual, version, 4.

[4] Yin, W. (2015). Gurobi mex: A matlab interface for gurobi (2011). convexoptimization. com/wikimization/index. php/gurobi mex.

[5] Boyd, S., Xiao, L., & Mutapcic, A. (2003). Subgradient methods. lecture notes of EE392o, Stanford University, Autumn Quarter, 2004, 2004-2005.

[6] Bernaards, C. A., & Jennrich, R. I. (2005). Gradient projection algorithms and software for arbitrary rotation criteria in factor analysis. Educational and Psychological Measurement, 65(5), 676-696.