Math 467: Project 2

asif zubair

December 1, 2016

1. Basic Simplex Solver

We provide a basic simplex solver for the linear programming problem.

The problem we solve is $\min_{x \epsilon \boldsymbol{R}^n} c^T x$ subject to $Ax = b, x \geqslant 0$. The matrix A is assumed to have the property that there are columns $A_{i_k}, k = 1, \ldots, m$ such that $A_{i_k} = e_k$ and the vector b is assumed to have non-negative components. The function signature is

```
[Solution, BasicVar, Status] = basicsimplex(A, b, c, BasicVar0)
```

Here, `BasicVar0` is a vector that contains the integer index i_k for the initial basic variables. The function outputs `Status` which returns the status of linear programming: `0` for successful, `-1` if no optimal solution exist and the cost can be as low as possible.

We see that the structure of the *augmented matrix* is $\begin{bmatrix} A & b \\ c^T & 0 \end{bmatrix}$. Assume that the first m columns of $\mathbf{A}$ are the basic columns. For the given case, these columns form a square m x m identity matrix $\mathbf{I}$. The non-basic columns of $\mathbf{A}$ form an m x $(n-m)$ matrix $\mathbf{D}$. We partition the cost vector correspondingly as $c^T = [c_B^T, c_D^T]$. Thus, the *augmented matrix* can be rewritten as $\begin{bmatrix} I & D & b \\ c_B^T & c_D^T & 0 \end{bmatrix}$. We convert this into the *cononical tableau corresponding to the basis* $\mathbf{B}$ by applying elementary row operations.

The final *canonical matrix* with *reduced cost coefficients* is given by:

$\begin{bmatrix} I & D & b \\ 0 & c_D^T - c_B^T D & -c_B^T b \end{bmatrix}$

This is the *canonical form* that we start with in the `basicsimplex` implementation.

Example

We provide a working of the solver for an example from the course book (Ex, 16.3, Pg. 359).

Here we consider the linear programming problem:

max $7x_1 + 6x_2$ subject to $2x_1 + x_2 \leq 3$, $x_1 + 4x_2 \leq 4$, $x_1, x_2 \geq 0$.

We first print the *canonical matrix* and then proceed with pivoting based on cost coefficients.

```
**********************
Current tableau:
     2     1     1     0     3
     1     4     0     1     4
    -7    -6     0     0     0

Pivot  is 1
Current basic feasible solution is
    1.5000
         0
```

```
         0
    2.5000

**********************
**********************
Current tableau:
    1.0000    0.5000    0.5000         0    1.5000
         0    3.5000   -0.5000    1.0000    2.5000
         0   -2.5000    3.5000         0   10.5000

Pivot  is 2
Current basic feasible solution is
    1.1429
    0.7143
         0
         0

**********************
**********************
Current tableau:
    1.0000         0    0.5714   -0.1429    1.1429
         0    1.0000   -0.1429    0.2857    0.7143
         0         0    3.1429    0.7143   12.2857

Found feasible solution:
    1.1429
    0.7143
         0
         0

**********************
```

We can see the function correctly outputs the feasible solution as $x_1 = 1.1429$ and $x_2 = 0.7143$.

3. Regression

The L^2 regression has a closed form solution given by the *normal equations* as:

$$\beta = (X^T X)^{-1} X^T y$$

Here, $X = [1, x]$ and β is the estimate of the regression coefficients.

The L^1 regression can be formulated as a linear programming problem as

$$\min_{w \epsilon \boldsymbol{R}^n} \sum_{k=1}^{N} |w_k|$$

subject to $w_k = y_k - a x_k - b$ for $k = 1 \dots N$. However, to use the simplex solver we need to have an additional constraint that the variables w_k, a, b should be non-negative.

We can achieve this by reformulating the problem with $w_k = (w_k)_+ - (w_k)_-$, $a = a_+ - a_-$, $b = b_+ - b_-$. Now, we can impose the non-negative constraints on the variables $(w_k)_+, (w_k)_-, a_+, a_-, b_+, b_-$

Thus, the final optimization problem we solve is

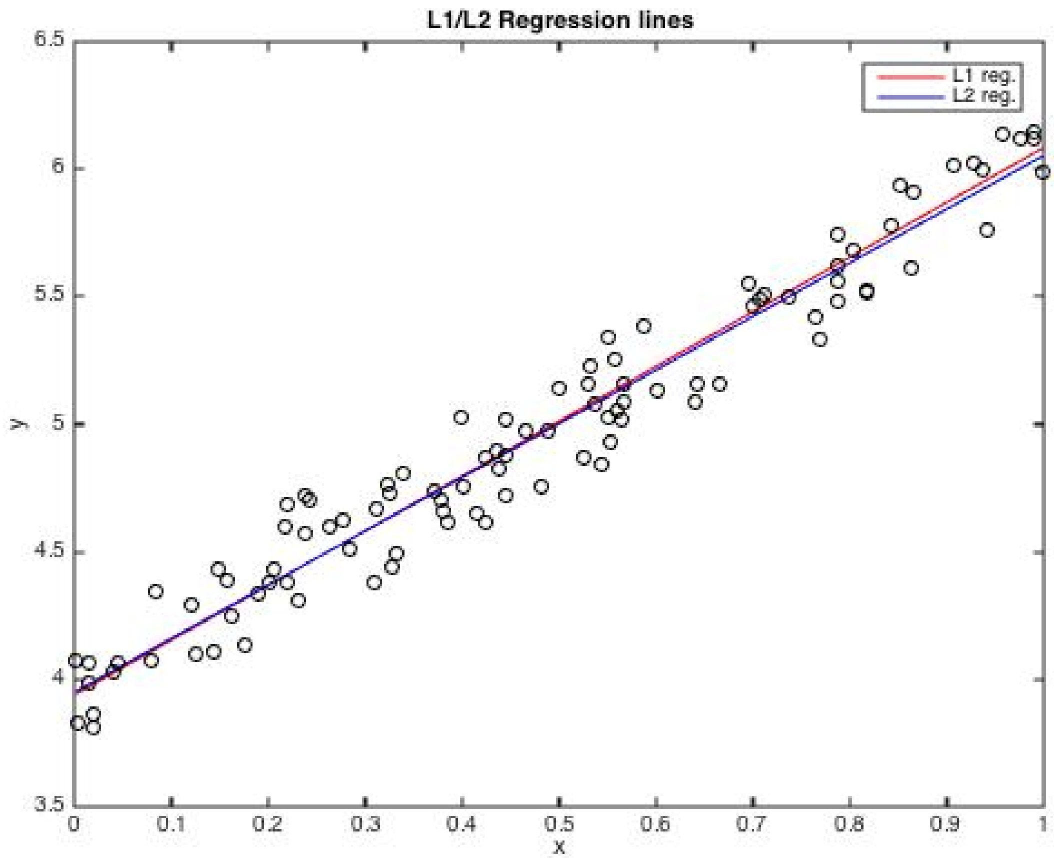
$\min_{w \epsilon \boldsymbol{R}^2 n} \sum_{k=1}^{N}((w_k)_+ + (w_k)_-)$ subject to $(w_k)_+ - (w_k)_- + (a_+ - a_-)x_k + b_+ - b_- = y_k$ for $k = 1 \dots N$

Hence, we have $2n + 4$ variables with n constraints.

We provide two examples of correlations - positive and negative. The data was generated by sampling from uniform distribution x and then generating y from $ax + b$ with added uniform noise sampled uniformly on $(-0.25, 0.25)$

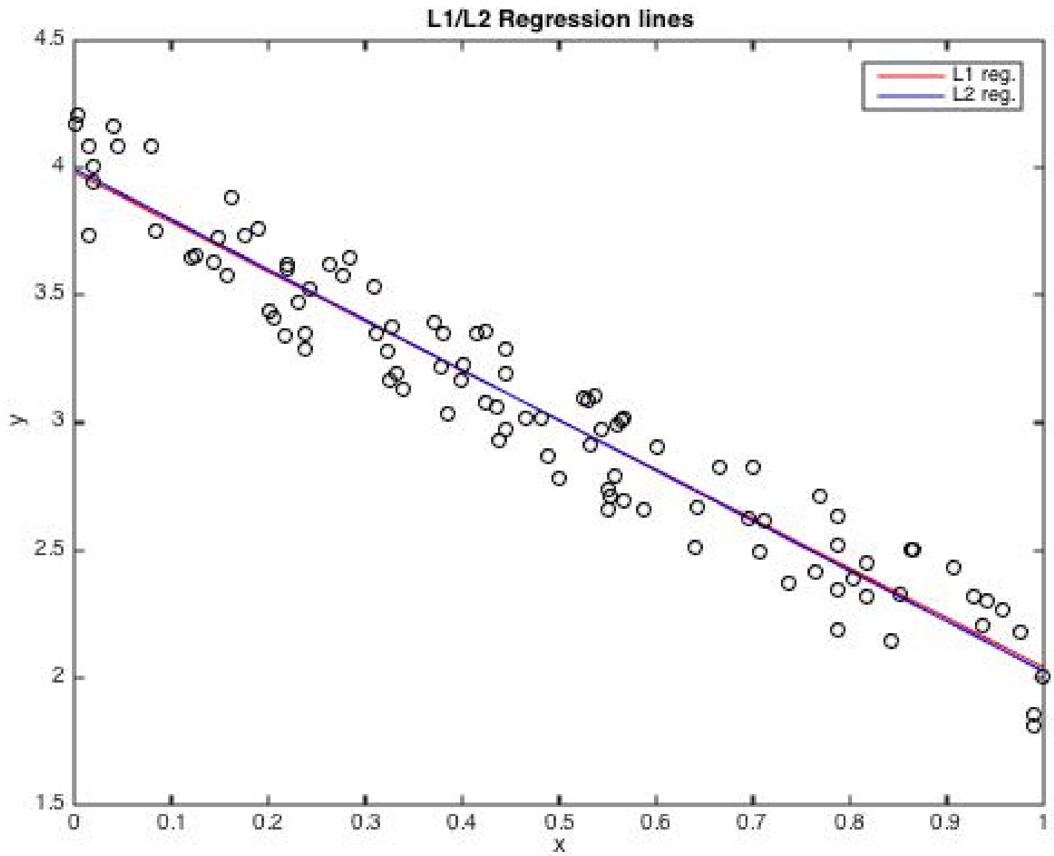
Example 1

```
Coeff.: b = 4.000, a = 2.000
L1 reg.: b = 4.033, a = 1.946
L2 reg.: b = 4.015, a = 1.963
```



Example 2

```
Coeff.: b = 4.000, a = -2.000
L1 reg.: b = 4.020, a = -2.041
L2 reg.: b = 4.018, a = -2.044
```



The L^1 regression is more robust to outliers because L^2 regression can give too much weightage to large residuals. However, when outliers are not present, we see that the two estimators follow each other closely (as shown above).

Matlab Implementation

Simplex Solver

```
function [Solution,BasicVar,Status]=basicsimplex(A,b,c,BasicVar0,verbose)
%% Matlab function that implement the basic simplex algorithm to solve a linear programming problem

    if nargin < 5
        verbose = false;
    end

    %% Basic Preprocessing - Define variables, check conditions etc.
    [m,n] = size(A);
    B = BasicVar0;
    N = setdiff(1:n, B);
    if any(b < 0)
        error('vector b should have non-negative entries.')
    end
```

```
if any(any(A(:,B) ~= eye(m)))
    error('BasicVar0 do not define a natural basis on R^m.')
end

%% simplex setup
table = zeros(m+1, n+1);
table(1:m,1:n) = A;
table(m+1,N) = c(N)' - c(B)'*A(:,N);
table(1:m,end) = b(:);
table(m+1,n+1) = -c(B)'*b;

%% algorithm
increment = true;
Status = 0;
while increment
    if verbose
        fprintf('**********************\n');
        fprintf('Current tableau:\n');
        disp(table);
    end

    if any(table(end,1:n)<0)%check if there is negative cost coeff.
    [~,index] = min(table(end,1:n));
        % check if corresponding column is unbounded
        if all(table(1:m,index)<=0)
            Solution = [];
            BasicVar = [];
            Status = -1;
            error('Problem is not bounded. All entries <= 0 in column %d',index);
        else
            pivot = 0;
            min_found = inf;
            for i = 1:m
                if table(i,index)>0
                    tmp = table(i,end)/table(i,index);
                    if tmp < min_found
                        min_found = tmp;
                        pivot = i;
                    end
                end
            end
            if verbose
                fprintf('Pivot  is %d\n',pivot);
            end
        %normalize
        table(pivot,:) = table(pivot,:)/table(pivot,index);
            % Make all entries in index column zero.
            for i=1:m+1
                if i ~= pivot
                     table(i,:)=table(i,:)-sign(table(i,index))*...
                        abs(table(i,index))*table(pivot,:);
                end
            end
```

```
            end
            if verbose  %print current basic feasible solution
                fprintf('Current basic feasible solution is\n');
                [curr_x, ~] = current_x();
                disp(curr_x);
                fprintf('***********************\n');
            end
        else
            increment = false;
        end
    end

    %internal function, finds current basis vector
    function [curr_x, bv] = current_x()
         curr_x = zeros(n,1);
         bv = [];
         for j=1:n
             if length(find(table(:,j)==0)) == m
                 idx = table(:,j) == 1;
                 try
                     curr_x(j) = table(idx,end);
                 catch ME
                    continue
                 end
                 bv = [bv,j];
             end
         end
    end
    [curr_x, bv] = current_x();
    Solution = curr_x;
    BasicVar = bv;
    if verbose  %print current basic feasible solution
        fprintf('Found feasible solution: \n');
        [curr_x, ~] = current_x();
        disp(curr_x);
        fprintf('**********************\n');
    end
end
```

L1 Regression

```
function [a,b] = l1_reg(x, y)
    %% L1 Regression using Linear Programming
    n = length(x);
    c = [0 0 0 0 ones(1,2*n)]';
    A = [ones(n,1), -ones(n,1), x, -x, eye(n,n), -eye(n,n)];
    b = y;
    [S,bv] = basicsimplex(A, b, c, 5:n+4);
    a = S(1) - S(2);
    b = S(3) - S(4);
end
```

L2 Regression

```
function [a,b] = l2_reg(x,y)
    %% L2 Regression
    m = length(x);
    X = [ones(m,1) x];
    beta = (X'*X) \ X' * y;
    a = beta(1);
    b = beta(2);
end
```

Driver Program

```
x = sort(rand(100,1));
b = 4;
a = 2;
y = b+a*x + rand(size(x))-.5;
fprintf('Coeff.: b = %3.3f, a = %3.3f\n', a, b)

[a,b] = l1_reg(x, y);
fprintf('L1 reg.: b = %3.3f, a = %3.3f\n', a, b)
plot(x,a + b*x,'r-')
title 'L1/L2 Regression lines'
xlabel 'x'
ylabel 'y'
hold on

[a,b] = l2_reg(x, y);
fprintf('L2 reg.: b = %3.3f, a = %3.3f\n', a, b)
plot(x,a + b*x,'b-')
plot(x, y, 'ko')
legend('L1 reg.', 'L2 reg.');
hold off

b = 4;
a = -2;
y = b+a*x + rand(size(x))-.5;
fprintf('Coeff.: b = %3.3f, a = %3.3f\n', a, b)

[a,b] = l1_reg(x, y);
fprintf('L1 reg.: b = %3.3f, a = %3.3f\n', a, b)
plot(x,a + b*x,'r-')
title 'L1/L2 Regression lines'
xlabel 'x'
ylabel 'y'
hold on

[a,b] = l2_reg(x, y);
fprintf('L2 reg.: b = %3.3f, a = %3.3f\n', a, b)
plot(x,a + b*x,'b-')
plot(x, y, 'ko')
legend('L1 reg.', 'L2 reg.');
hold off
```