資工三 B00902064 宋昊恩

Convex Optimization Hw6

1. Implementation:

For this homework, I have tried multi-times implementations.

In the beginning, I used for loop to do calculate values, including first-order and twice-order gradient of $\mathbf{f}(\mathbf{w})$. It was obvious that the execution of this program was pretty slow, even after I added some parallel mechanisms in it.

```
>> newton
Iter 1: 369104.665132
Iter 2: 149834.719046
Iter 3: 54222.086745
Iter 4: 23704.198046
Iter 5: 13727.869227
Iter 6: 9565.439222
Iter 7: 7384.281827
Iter 8: 6043.823809
Iter 9: 5121.087733
Iter 10: 4464.282826
Iter 11: 3958.705398
Iter 12: 3540.471242
CPU time: 23916.170000 (excluding IO)
Elapsed time: 23523.836630 (excluding IO)
Final step size = 1.000000
>>
```

Figure1: for loop with parallel

```
>> newton
Iter 1: 369104.665132
Iter 2: 113462.519272
Iter 3: 50964.835626
Iter 4: 27134.937526
Iter 5: 16922.381408
Iter 6: 11881.077798
Iter 7: 8945.865786
Iter 8: 7134.741557
Iter 9: 5929.635506
Iter 10: 5066.759351
Iter 11: 4419.419759
Iter 12: 3916.399859
Iter 13: 3528.384628
CPU time: 6535.630000 (excluding IO)
Elapsed time: 6745.332891 (excluding IO)
Final step size = 1.000000
>>
```

Figure2: matrix calculation

After I noticed that all the calculations can fulfilled by matrix manipulation, I rewrote my code, and gained a huge performance improvement.

However, just the day before deadline, I found out that I misused the algorithm on label {0, 1} data. After I rewrote the code again, the number of iterations reduced a lot, and the program behavior was much more acceptable.

2. Experiment:

The above iteration value is the norm of first order gradient. CPU time is calculated by $\mathbf{cputime}$ in $\mathbf{matlab}$. Elapsed time is calculated by $\mathbf{tic}$ and $\mathbf{toc}$ in $\mathbf{matlab}$. Both of them do not include the time for IO reading.

It is clear that their values are different in $\mathbf{Figure\ 1}$ and $\mathbf{Figure\ 2}$. I have reviewed my code several times, so I believe that it is the behavior of numerical mistakes and error propagation.

Then I want to further prove the correctness of my code. I try to check my final $\mathbf{f}(\mathbf{w})$ value with other classmates, and I found that we have slightly different results. My score is $\mathbf{508484.874037}$, however, one of my classmate's value is about $\mathbf{493333}$.

There are two possible reasons; one reason is that my code is just wrong; the second reason may be caused by the error of numerical calculation as above said.

To figure out the real reason, I reduce my $\mathbf{eps}$ from $\mathbf{0.01}$ to $\mathbf{0.005}$, and get the result $\mathbf{492796.846594}$, which is reasonable as second reason.

```
>> newton
Iter 1: 313748.270049
Iter 2: 59681.846887
Iter 3: 22837.994845
Iter 4: 14129.063416
Iter 5: 2475.368497
CPU time: 11187.980000 (excluding IO)
Elapsed time: 12048.773607 (excluding IO)
Final f(w) value = 508484.874037
Final step size = 1.000000
>> Your MATLAB session has timed out.  All license keys have been returned.
```

Figure 3: matrix manipulation, label set of {-1, 1}, eps = 0.01

```
>> newton
Iter 1: 313748.270049
Iter 2: 59681.846887
Iter 3: 22837.994845
Iter 4: 14129.063416
Iter 5: 2475.368497
Iter 6: 1088.123455
CPU time: 23522.490000 (excluding IO)
Elapsed time: 23341.684516 (excluding IO)
Final f(w) value = 492796.846594
Final step size = 1.000000
>>
```

Figure 3: matrix manipulation, label set of {-1, 1}, eps = 0.005

3. Conclusion:

This homework takes me a lot of time to implement and put experiment on it.

It is the first time I realize that the BLAS library work so well than implementation by myself, and I finally understand why teacher always ask us to transform general calculation into matrix-form in our exam.

4. Code implementation:

```
function newton()
addpath('liblinear-1.94/matlab/');

tnFile = 'kddb';
ttFile = 'kdda.t';

eps = 0.01;
C = 0.1;
eta = 0.01;
xi = 0.1;

[label, inst] = libsvmread(tnFile);
[n, m] = size(inst);
label = 2 * label - ones(n, 1);

w = zeros([m, 1]); % weight
e = zeros([n, 1]); % e ^ (-y_i * x_i * w')
f = zeros([n, 1]); % e ^ (-y_i * x_i * w') / (1 + e ^ (-y_i * x_i * w')) ^ 2

wx = zeros([n, 1]); % x * w
sx = zeros([n, 1]); % x * s
rf = 0 + C * n * log(2);

gf = zeros([m, 1]); % first gradient order
r = zeros([m, 1]); % CG var
d = zeros([m, 1]); % CG var

CPUTIME = cputime;
TIME = tic;

curIter = 0;
while true
    curIter = curIter + 1;

    e = exp(-label .* wx);
    f = e ./ (ones(n, 1) + e) .^ 2;

    gf = w + C * (inst' * (label .* (ones(n, 1) ./ (ones(n, 1) + e) - ones(n, 1))));
    r = -gf;
    d = -gf;
    if (curIter == 1)
        gzero = gf;
    end


    % stop condition
    fprintf('Iter %d: %f\n', curIter, norm(gf));
    if (norm(gf) <= eps * norm(gzero))
        break
    end


    % Find s
    s = zeros([m, 1]);
    while norm(r) > xi * norm(gf)
        gs = (d + C * inst' * (f .* (inst * d)));

        alpha = norm(r) ^ 2 / (d' * gs);
        s = s + alpha * d;
        nr = r - alpha * gs;

        beta = norm(nr) ^ 2 / norm(r) ^ 2;
        d = nr + beta * d;
```

```
            r = nr;
        end


        % Find alpha and update
        alpha = 1.0;
        sx = inst * s;
        rs = eta * gf' * s;

        while true
            left = 0.5 * dot(w + alpha * s, w + alpha * s) + C * (ones(1, n) * log(ones(n, 1) + exp(-label .* (wx + alpha * sx))));

            if left <= rf + alpha * rs
                rf = left;
                break;
            else
                alpha = alpha / 2;
            end
        end


        % update w, wx
        w = w + alpha * s;
        wx = wx + alpha * sx;
        galpha = alpha;
    end


    % print statistic
    fprintf('CPU time: %f (excluding IO)\n', cputime - CPUTIME);
    fprintf('Elapsed time: %f (excluding IO)\n', toc(TIME));
    fprintf('Final f(w) value = %f\n', rf);
    fprintf('Final step size = %f\n', galpha);
```