Optimization and Machine Learning HW6 Report

Tong Yu

R01922141, CSIE, National Taiwan University

1 Summary

In this homework, we implemented a line-search based newton method for L2 Logistic Regression problem.

2 Code interface

In this section, we would briefly introduce the matlab interface in the submitted codes. Please check the comments in the code for more descriptions in detail.

- function [W, f_list] = train_test_nt(file_train, c)

  It is the main function, which includes the main procedure of newton method. It mainly includes a outer loop (newton method). In each iteration, it also has two inner loops (for CG to solve newton linear system and line search to find appropriate step size.)

  file_train is the training data's path and c is the parameter of the model; W is the output model and f_list contains the function values of all iterations.

- function [fun_value] = logis_fun(W, c)

  The function is to calculate the function value of the problem.

  W is the model and c is the parameter of the model; the output is the function value.

  Note that the X_train and Y_train is defined as global, so that we won't need the cost of passing huge training data as parameters every time.

- function [fun_value] = logis_fun_grad(W, c)

  The function is to calculate the gradient of the problem.

  W is the model and c is the parameter of the model; the output is the calculated gradient.

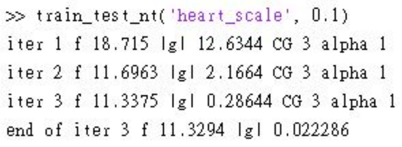
- function [f_h] = appro_hessian(s, W, c)

  The function is to calculate $\nabla^2 f(W)s$. W is the model and c is the parameter of the model. The output is the calculated value.

3 Start with a small dataset heart_scale

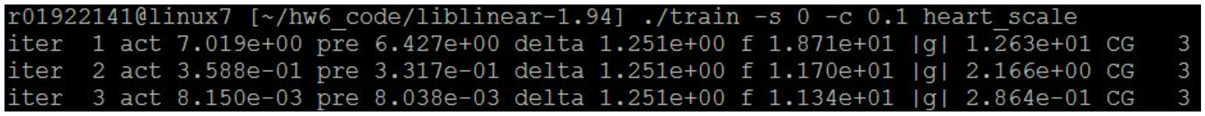
To verify the correctness of our implementation, we start with a small dataset, provided by LIBLINEAR.

- The log of my implementation:



```
>> train_test_nt('heart_scale', 0.1)
iter 1 f 18.715 |g| 12.6344 CG 3 alpha 1
iter 2 f 11.6963 |g| 2.1664 CG 3 alpha 1
iter 3 f 11.3375 |g| 0.28644 CG 3 alpha 1
end of iter 3 f 11.3294 |g| 0.022286
```

- The log of LIBLINEAR TRON implementation:



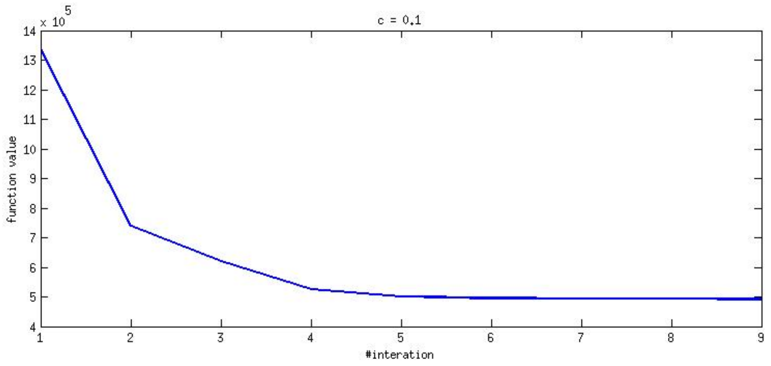
```
r01922141@linux7 [~/hw6_code/liblinear-1.94] ./train -s 0 -c 0.1 heart_scale
iter  1 act 7.019e+00 pre 6.427e+00 delta 1.251e+00 f 1.871e+01 |g| 1.263e+01 CG   3
iter  2 act 3.588e-01 pre 3.317e-01 delta 1.251e+00 f 1.170e+01 |g| 2.166e+00 CG   3
iter  3 act 8.150e-03 pre 8.038e-03 delta 1.251e+00 f 1.134e+01 |g| 2.864e-01 CG   3
```

We can find that at each iteration, the function value f and the norm of gradient $|g|$ are quite similar. (when cg doesn't reach trust region boundary)

4 Explore the kddb dataset

4.1 The function values

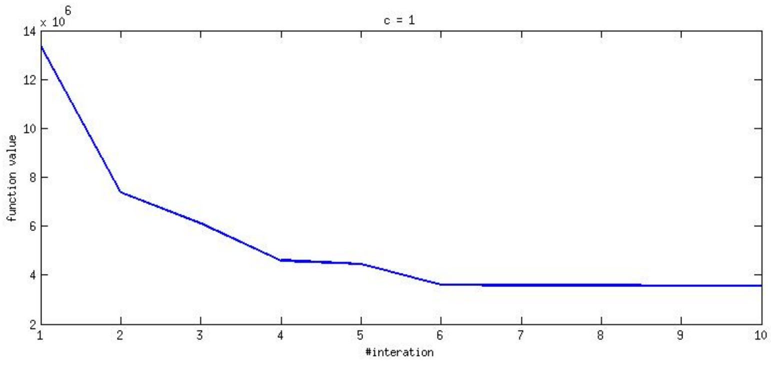
With parameter $C = 0.1$, and $epsilon = 0.001$ in stopping criterion, we have the following results:



The function value at the last iteration is 4.925e+05. To verify the correctness, the function value at the last iteration of LIBLINEAR is 4.944e+05, which is quite close. The test accuracy in kddb.t is 90.00%. (LIBLINEAR's result is 89.9858%)

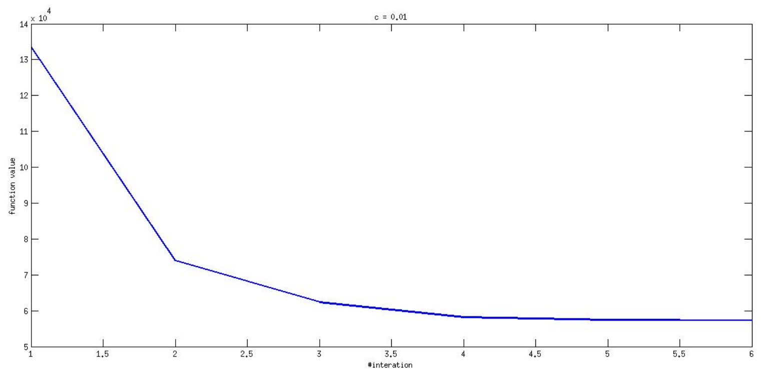
4.2 Different regularization parameter

$C = 0.1$ may not the best parameter for this problem, so we also tried $C = 1$ and $epsilon = 0.001$. The function values during the experiment is as the following:



The function value at the last iteration is 3.552e+06. And the function value at the last iteration of LIBLINEAR is 3.628e+06. The test accuracy in kddb.t is 89.67%. (LIBLINEAR's result is 89.724%)

As the homework slides describes, when $C = 0.01$ and $epsilon = 0.001$, we can observe step size $a = 1$ (which won't happen when $C = 0.1$) at the final iteration. The function value is as the following:

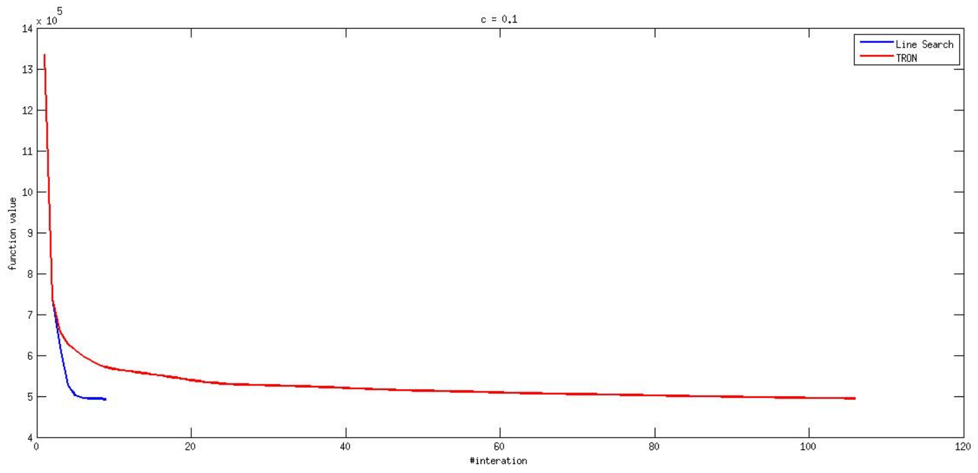


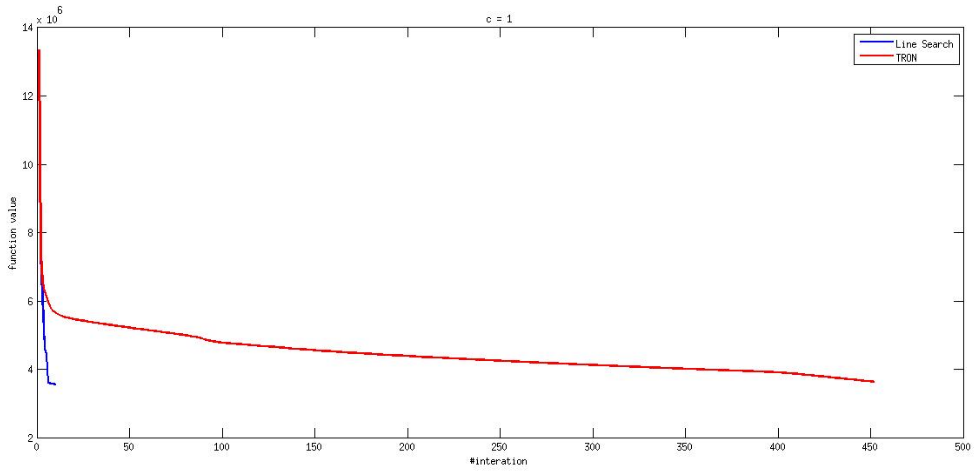
The function value at the last iteration is 5.733e+04. And the function value at the last iteration of LIBLINEAR is 5.736e+04. The test accuracy in kddb.t is 89.64%. (LIBLINEAR's result is 89.6225%)

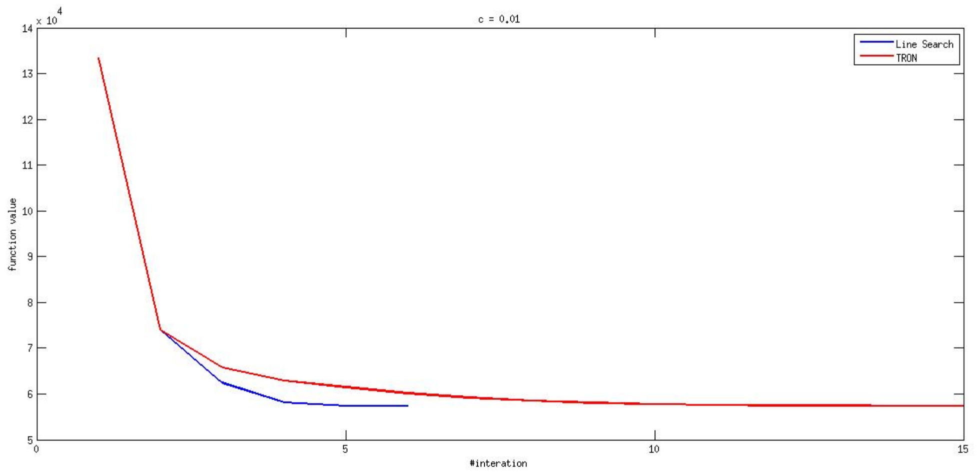
4.3 A comparison with TRON in LIBLINEAR

The first difference is programming language (matlab versus C). However, in my implementation, the calculation of function value and gradient doesn't use for loop but use matrix operation. About the outer and inner loop in newton method, as their number of iterations is usually not that big. So, the matlab implementation is comparable to the C implementation in efficiency.

As for line search versus TRON, in experiments we find TRON usually need much more iterations to achieve a low function value. The detailed comparison of function values between line search and TRON are as the followings ($C = 0.1$ and 1 and 0.01 respectively; considering possible time difference between matlab and C, #iteration is used as x-axis.)







As C becomes smaller, we find the gap between line search and TRON in kddb becomes smaller.