THE UNIVERSITY OF IOWA, THE DEPARTMENT OF CHEMISTRY

Optimization Techniques (MATH:4820:0001)

Programming Assignment

Sparse Logistic Regression

Zhen Li

May 8, 2015

1 THEORETICAL PREPARATION

According to the text of the assignment, we are supposed to maximize the following

$$f(\gamma,\mathbf{c}) = \prod_{i=1}^{N} p(\mathbf{x}_i)^{y_i}(1-p(\mathbf{x}_i))^{1-y_i} \tag{1.1}$$

Maximizing $f(\gamma,\mathbf{c})$ is equivalent to mizimizing a so called *log likelihood function* $l(\gamma,\mathbf{c})$ as (1.2).

$$l(\gamma,\mathbf{c}) := \sum_{i=1}^{N} y_i ln(1+exp(\gamma+\mathbf{c}^T\mathbf{x}_i)) + (1-y_i)ln(1+exp(-\gamma-\mathbf{c}^T\mathbf{x}_i)) \tag{1.2}$$

The proof to show that maximizing the function $f(\gamma,\mathbf{c})$ is equivalent to minimizing the funtion $l(\gamma,\mathbf{c})$ is as below:

$\because \prod_{i=1}^{N} p(\mathbf{x}) = 1/(1+exp(\gamma+\mathbf{c}^T\mathbf{x}))$

$\therefore p(\mathbf{x}) \in (0,1)$

$\therefore \prod_{i=1}^{N} p(\mathbf{x}_i)^{y_i}(1-p(\mathbf{x}_i))^{1-y_i} \in (0,1)$.

$\because ln(x)$ increases monotonicaly

$\therefore$ maximizing $f(\gamma,\mathbf{c})$ is equivalent to maximizing $ln(f(\gamma,\mathbf{c}))$.

$$\begin{aligned} ln(f(\gamma,\mathbf{c})) &= ln\prod_{i=1}^{N} p(\mathbf{x}_i)^{y_i}(1-p(\mathbf{x}_i))^{1-y_i} \\ &= \sum_{i=1}^{N}(y_i lnp(\mathbf{x}_i)+(1-y_i)ln(1-p(\mathbf{x}_i))) \\ &= \sum_{i=1}^{N}(-y_i ln(1+exp(\gamma+\mathbf{c}^T\mathbf{x}_i))-(1-y_i)ln(1+exp(-\gamma-\mathbf{c}^T\mathbf{x}_i))) \end{aligned} \tag{1.3}$$

Therefore maximizing $lnf(\gamma,\mathbf{c})$ is equivalent to minimizing $-lnf(\gamma,\mathbf{c})$ (which is $l(\gamma,\mathbf{c})$).
For $l(\gamma,\mathbf{c}) = \sum_{i=1}^{N}(y_i ln(1+exp(\gamma+\mathbf{c}^T\mathbf{x}_i))+(1-y_i)ln(1+exp(-\gamma-\mathbf{c}^T\mathbf{x}_i)))$ with $l(\gamma,\mathbf{c})$ called *log likelihood function.*
Next step is to show that $l(\gamma,\mathbf{c})$ is a convex function of $(\gamma,\mathbf{c})$.
As we have shown in previous homework (Homework 1 question 3), suppose that $g(\mathbf{x}):\mathbb{R}^n\to\mathbb{R}$ is convex and $f:\mathbb{R}\to\mathbb{R}$ is convex and increasing, then $f(g(\mathbf{x})):\mathbb{R}^n\to\mathbb{R}$ is convex as well.
Set $g(\gamma,\mathbf{c}) = \gamma+\mathbf{c}^T\mathbf{x}_i$, and it's easy to see $g(\gamma,\mathbf{c})$ is convex. Set $f(x) = ln(1+e^x)$, and $f'(x) = e^x(1+e^x)^{-1} > 0$ and $f''(x) = -e^x(1+e^x)^{-2}e^x + e^x(1+e^x)^{-1} = e^x(1+e^x)^{-2} > 0$. Therefore $f(x)$ is convex and increasing. So combine them together and $ln(1+exp(\gamma+\mathbf{c}^T\mathbf{x}_i)$ is convex (so as $ln(1+exp(-\gamma-\mathbf{c}^T\mathbf{x}_i))$. Because $y_i\in\{0,1\}$, so $y_i\geqslant 0$. Due to the property of convex function (nonnegative multiple), $y_i ln(1+exp(\gamma+\mathbf{c}^T\mathbf{x}_i))$ and $(1-y_i)ln(1+exp(-\gamma-\mathbf{c}^T\mathbf{x}_i)))$ are convex. Due to the affine property of convex function, $l(\gamma,\mathbf{c}) = \sum_{i=1}^{N}(y_i ln(1+exp(\gamma+\mathbf{c}^T\mathbf{x}_i))+(1-y_i)ln(1+exp(-\gamma-\mathbf{c}^T\mathbf{x}_i)))$ is convex as well.

$$f(\gamma,\mathbf{c}) := l(\gamma,\mathbf{c})+\alpha\parallel\mathbf{c}\parallel_1 \tag{1.4}$$

where $\parallel\mathbf{c}\parallel_1 = \sum_{i=1}^{N}|c_1|$ and $\alpha>0$. $f(\gamma,\mathbf{c})$ is a convex function. Because $|c_i|$ and $l(\gamma,\mathbf{c})$ convex and the nonnegative multipule and affine property, $f(\gamma,\mathbf{c})$ is convex as well.
In order to make the problem smooth, the following problem has been introduced.

$$\begin{aligned} &\min_{r,\mathbf{c},\mathbf{w}} l(\gamma,\mathbf{c})+\alpha\sum_{i=1}^{n}w_i \\ &\quad\text{subject to} \\ &\quad w_i\geqslant +c_i, i=1,2,\dots,n \\ &\quad w_i\geqslant -c_i, i=1,2,\dots,n \end{aligned} \tag{1.5}$$

The KKT (Karush-Kuhn-Tucher) necessary condition for this problem is as follow:

Let the $(r^*,\mathbf{c}^*,\mathbf{w}^*)$ be a local minimizer.
We add more varibles to $f(\gamma,\mathbf{c})$:

$f(\gamma, \mathbf{c}, \mathbf{w}) = l(\gamma, \mathbf{c}) + \alpha \sum_{i=1}^{n} w_i$
The constraints are:
$w_i - c_i \geqslant 0$
$w_i + c_i \geqslant 0$
Then the Lagrangian is:

$$\begin{aligned} L(\gamma, \mathbf{c}, \mathbf{w}, \boldsymbol{\lambda}^+, \boldsymbol{\lambda}^-) &= f(\gamma, \mathbf{c}, \mathbf{w}) - \sum_{i=1}^{n} \lambda_i^+ (w_i - c_i) - \sum_{i=1}^{n} \lambda_i^- (w_i + c_i) \\ &= l(\gamma, \mathbf{c}) + \alpha \sum_{i=1}^{n} w_i - \sum_{i=1}^{n} \lambda_i^+ (w_i - c_i) - \sum_{i=1}^{n} \lambda_i^- (w_i + c_i) \end{aligned} \tag{1.6}$$

and we have:

$$\begin{aligned} &\frac{\partial L}{\partial \gamma} = \frac{\partial l}{\partial \gamma}(\gamma, \mathbf{c}) = 0 \\ &\frac{\partial L}{\partial c_j} = \frac{\partial l}{\partial c_j}(\gamma, \mathbf{c}) + \lambda_j^+ - \lambda_j^- = 0 \\ &\frac{\partial L}{\partial w_j} = \alpha - \lambda_j^+ - \lambda_j^- = 0 \end{aligned} \tag{1.7}$$

If the inequality constraint is inactive, then the Langrange multiplier will be 0.

$$\begin{aligned} &\lambda_j^{\pm} \geqslant 0 \\ &w_j \mp c_j \geqslant 0 \\ &\lambda_j^+ (w_j - c_j) = 0 \\ &\lambda_j^- (w_j + c_j) = 0 \\ &\text{for } j = 1, 2, \ldots, n \end{aligned} \tag{1.8}$$

Therefore, for c_i there will be three different conditions.

$$\begin{aligned} &\text{if } c_j > 0 \\ &\lambda_j^+ = \alpha \text{ and } \lambda_j^- = 0 \\ &\text{if } c_j < 0 \\ &\lambda_j^- = \alpha \text{ and } \lambda_j^+ = 0 \\ &\text{if } c_j = 0 \\ &\lambda_j^+ + \lambda_j^- = \alpha \\ &|\frac{\partial l}{\partial c_j}(\gamma, \mathbf{c})| \leqslant \alpha \end{aligned} \tag{1.9}$$

If $(r^*, \mathbf{c}^*, \mathbf{w}^*)$ is a constrained local minimizer, then we want:
$\mathbf{d}^T Hess_{\gamma, \mathbf{c}, \mathbf{w}} L(r^*, \mathbf{c}^*, \mathbf{w}^*, \boldsymbol{\lambda}^+, \boldsymbol{\lambda}^-) \mathbf{d} \geqslant 0$
for all $\mathbf{d} \in T_\Omega(r^*, \mathbf{c}^*, \mathbf{w}^*)$

2 ALGORITHM

Active set strategy is going to be used as suggested in the context of the assignment. In general, the strategy is saying that there will be a set of active constaints we put into consideration. The set is not fixed and constraints can be added or removed from this set according to movement we will have. From 1, we can see that the sign of c_i decides the activetion of constraints.

Due to the hint from the assignment, we shouldn't move to far to change the sign of c_i because this sign change refers to the activation and deactivation of constraints. But this doesn't mean that we can not change sign. The change of sign of c_i is actually very necessary since we need to make the active set dynamic to find local minimier. As said in the assignment, if $c_i = 0$ and $|\partial l / \partial c_i(\mathbf{c})| \leqslant \alpha$ then, we keep $c_i = 0$; if $c_i = 0$ and $\partial l / \partial c_i(\mathbf{c}) > \alpha$, then we make c_i negative, which means that $w_i \geqslant -c_i$ is activated and $w_i \geqslant c_i$ is inactive); if $c_i = 0$ and $\partial l / \partial c_i(\mathbf{c}) < -\alpha$ then we make c_i positive.

The *pseudocode* would be like following:

```
$\mathbf{c} \leftarrow \mathbf{0}$
$\alpha \leftarrow 0.05$
$A = 10^{-4}$                                              ▷ Armijo/Backtracking constant
while 1 do
    $s_i = \text{sign}(c_i)$                                        ▷ for $i = 1, 2, \dots, n$
    $\delta = -\nabla^2_\gamma \, l(\gamma, \mathbf{c})^{-1} \frac{\partial l(\gamma, \mathbf{c})}{\partial \gamma}$
    $d_i = -\nabla^2_{c_i} \, l(\gamma, \mathbf{c})^{-1} (\frac{\partial l(\gamma, \mathbf{c})}{\partial c_i} + \alpha s_i)$          ▷ for $i = 1, 2, \dots, n$
                                                                ▷ Newton step
                                             ▷ Armijo/Backtracking line search
    $\beta_\delta = 1$                                         ▷ $\beta$ refers to the step lenght
    while $l(\gamma + \beta_\delta \delta, \mathbf{c}) > l(\gamma, \mathbf{c}) + A\beta_\delta \frac{\partial l(\gamma, \mathbf{c})}{\partial \gamma} \delta$ do
        $\beta_\delta \leftarrow \frac{1}{2} \beta_\delta$
    end while
    $\beta \leftarrow \min_{i=1,2,\dots,n} (1, \frac{-c_i}{d_i})$
    while $l(\gamma, \mathbf{c} + \beta \mathbf{d}) + \alpha \beta \mathbf{d}^T \mathbf{s} > l(\gamma, \mathrm{c}) + A\beta \mathbf{d}^T (\frac{\partial l(\gamma, \mathbf{c})}{\partial \mathbf{c}} + \alpha \mathbf{s})$ do
        $\beta \leftarrow \frac{1}{2} \beta$
    end while
                                                          ▷ Take a Newton step
    $\gamma \leftarrow \gamma + \beta_\delta \delta$
    $\mathbf{c} \leftarrow \mathbf{c} + \beta \mathbf{d}$

    for $i = 1, 2, \dots, n$ do
        if $c_i + \beta d_i = 0$ && $c_i != 0$ then
            $s_i = 0$
    end if
end for
                                                            ▷ Update c
for $i = 1,2,\ldots,n$ do
    if $c_i == 0$ && $\frac{\partial l(\gamma,\mathbf{c})}{\partial \mathbf{c}} > \alpha$ then
        $c_i = -0.05$
        $s_i = -1$
    else if $c_i == 0$ && $\frac{\partial l(\gamma,\mathbf{c})}{\partial \mathbf{c}} < -\alpha$ then
        $c_i = 0.05$
        $s_i = 1$
    end if
end for
                                                            ▷ Gradient step
$\delta = -\frac{\partial l(\gamma,\mathbf{c})}{\partial \gamma}$
$\beta_\delta = 1$
while $l(\gamma + \beta_\delta \delta, \mathbf{c}) > l(\gamma,\mathbf{c}) + A\beta_\delta \frac{\partial l(\gamma,\mathbf{c})}{\partial \gamma}\delta$ do
    $\beta_\delta \leftarrow \frac{1}{2}\beta_\delta$
end while
for $i = 1,2,\ldots,n$ do
    if $c_i != 0 \,||\, |\frac{\partial l}{\partial c_i}(\gamma,\mathbf{c})| > \alpha$ then
        $d_i = -\frac{\partial l}{\partial c_i}(\gamma,\mathbf{c}) - \alpha s_i$
    else
        $d_i = 0$
    end if
end for
$\beta \leftarrow \min_{i=1,2,\ldots,n}(1, \frac{-c_i}{d_i})$
while $l(\gamma, \mathbf{c} + \beta\mathbf{d}) + \alpha\beta\mathbf{d}^T\mathbf{s} > l(\mathrm{c}) + A\beta\mathbf{d}^T(\frac{\partial l(\gamma,\mathbf{c})}{\partial \mathbf{c}} + \alpha\mathbf{s})$ do
    $\beta \leftarrow \frac{1}{2}\beta$
end while
                                                            ▷ Take a Gradient step
$\gamma \leftarrow \gamma + \beta_\delta \delta$
$\mathbf{c} \leftarrow \mathbf{c} + \beta\mathbf{d}$

$KKT = 0$
if $\frac{\partial l}{\partial \gamma}(\gamma,\mathbf{c}) > 10^{-6}$ then                  ▷ KKT condition check 1
    $\mathrm{KKT} = \mathrm{KKT} + 1$
end if
```

```
    for $i = 1, 2, \dots, n$ do                                              ▷ KKT condition check 2
        if $\frac{\partial l}{\partial c_i}(\gamma, \mathbf{c}) + \alpha > 10^{-6} \,||\, \frac{\partial l}{\partial c_i}(\gamma, \mathbf{c}) - \alpha > 10^{-6}$ then
            $KKT = \mathrm{KKT} + 1$
        end if
    end for

    if $KKT == 0$ then                                                        ▷ Final KKT check
        return 0;
    end if
end while
```

The KKT condition check is enough because the function that needs to be minimized is a convex and coercive function. Due to the properties of convex functions and coercive functions, the first order condition is sufficient.
The gradients and Hessian matrices are all written out manually and then write the functions out explicitly. Therefore, the gradients and Hessian matrices won't consume too much resource since they are explicit functions. Due to the numerical procedure of MATLAB in calculating gradients and Hessian matrices, if we apply implemented functions to do the calculation, then the accuracy and efficiency of the functions need to be taken care of. And test functions are helpful to verify the robust of the methods. Rosenbrock function is always a good way to test.

3 Results

3.1 Accuracy

The accuracy of the code depends on the accuracy of the ability to do numerical analysis. At the same time, some varibles could be very small or large where accuracy could be low since the difference compared to the numbers is not very significant.
On the other hand, the KKT exit condition is crucial to the accuracy of the method as well because the KKT condition decides when to halt the program and how close the final result is to that KKT condition. Since it's expensive to satisfy the first order KKT condition (the gradient with respect to γ is zero and the gradient with respect to c is zero), then an approximation is made in the code. As long as the gradients are smaller enough, then we can halt the program. Therefore, the accuracy of the method is decided by the approximation. The smaller the tolerence, the more accurate the method is.
Due to the fact that all the gradient and Hessian expression are hand-prooved, therefore there shouldn't be any inaccuracy introduceted from the evaluation of the expression.
According to the fact that the function we are about to minimize is a convex and coercive function, the method should be robust enough in terms of accuracy. However, the efficiency is always some thing that we could improve. As said above, the KKT condition is related to the accuracy, and it's also directly related to the efficiency of the algorithm.

3.2 Efficiency

There are many factors that can affect the efficiency of a method.
First of all is the α that we choose. Since it takes very long time for the MATLAB code to run if we really want to hit the KKT condition, I used 100 iterations to make the comparision. In this way, we are not able to test the efficiency of the program to finish the whole program, but we can see how this important factor affect our evaluations and Armijo/Backtracking line search. This α is chosen to modify the $\mathbf{c}$ to affect the efficiency.
From figures 1, 2, and 3, we could see that the choice of α doesn't affect the program too much on evaluating gradient and Hessian in fixed steps (100). This is reasonable because the both gradient and Hessian have been written explicitly. But the choice of α will affect the efficiency of our program if we let the program fit KKT condition instead of fixing number of iterations. As shown in the figures, the function value as been calculated many times for Armijo/Backtracking. Since the function is defined as a *For* loop, the evaluation of function is relatively slow compared to *sum* function. If the code can be written as sum function to calculate the function value, it will dramatically improve the efficiency of the function because the evaluation of function is heavily called.
Second, the program does a relatively good job on memory since the program is overwriting varibles all the time, which saves memory efficiently.
Third, the program checks the Hessian to see if it's singular and change the diagonal of the Hessian a little if necessary. This will affect the efficiency since the Hessian is mainly used to decide the direction of the step. Step direction is the key to the efficient and the speed of convergence.
At the end, the algorithm can be improved as well. The program uses the most basic Backtracking line search with bisection to find a proper Step length. This Backtracking method might lead to smaller steps, which makes the program very slow. An alternative to this Backtracking is wolfe condition, which includes curvature condition to make sure that the step length is fairly large.
The algorithm uses Newton method and steepest descent. Newton method is used to take a step and steepest descent is mainly used to make a step after the update of active sets. The Newton method can be replaced by Quasi-Newton method since the Hessian might be ill-conditioned and singular. This is just a guess. More test needs to be done before. A Markov chain Monte Carlo method might be able to help solve this problem as well.