Programming Assignment
Due Wednesday, May 6th

Sparse (Linear) Logistic Regression

Suppose that we have a collection of vectors $\mathbf{x}_i \in \mathbb{R}^n$, $i = 1, 2, \ldots, N$, and that for each i there is a value $y_i \in \{0, 1\}$. Given a new value $\mathbf{x} \in \mathbb{R}^n$, we want to predict the value y to be expected. One way of dealing with this in a probabilistic way is to suppose that for $\mathbf{x} \in \mathbb{R}^n$ there is a probability $p(\mathbf{x})$ that $y = 1$ and probability $1 - p(\mathbf{x})$ that $y = 0$. For each i there is a likelihood function

$$p(\mathbf{x}_i)^{y_i}(1 - p(\mathbf{x}_i))^{1-y_i}$$

that given $\mathbf{x} = \mathbf{x}_i$, y has the value y_i. Assuming the data points $(\mathbf{x}_i, y_i)$ are obtained independently, the likelihood for the data set as a whole is

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

Assuming that we have a formula $p(\mathbf{x}) = 1/(1 + \exp(\gamma + \mathbf{c}^T\mathbf{x}))$, we want to find γ and $\mathbf{c}$ that maximizes the likelihood as a function of $(\gamma, \mathbf{c})$.

Show that this is equivalent to minimizing

$$\ell(\gamma, \mathbf{c}) := \sum_{i=1}^{N} \left\{ y_i \ln(1 + \exp(\gamma + \mathbf{c}^T\mathbf{x}_i)) + (1 - y_i)\ln(1 + \exp(-\gamma - \mathbf{c}^T\mathbf{x}_i)) \right\}.$$

This function $\ell(\gamma, \mathbf{c})$ is called the *log likelihood function*. Show that $\ell(\gamma, \mathbf{c})$ is a convex function of $(\gamma, \mathbf{c})$.

If $n > N$ then there cannot be unique minimizers, since if $\mathbf{w}$ is perpendicular to all $\mathbf{x}_i$, $i = 1, 2, \ldots, N$, then $\ell(\gamma, \mathbf{c} + \mathbf{w}) = \ell(\gamma, \mathbf{c})$. In this case, it is still appropriate to look for sparse $\mathbf{c}$; that is, we want most entries of $\mathbf{c}$ to be zero. To help us find suitable sparse $\mathbf{c}$ we add a penalty term especially designed to make this happen:

$$f(\gamma, \mathbf{c}) := \ell(\gamma, \mathbf{c}) + \alpha \|\mathbf{c}\|_1$$

where $\|\mathbf{c}\|_1 = \sum_{i=1}^{n} |c_i|$ and $\alpha > 0$ is chosen to control how many entries of $\mathbf{c}$ are non-zero. We now have a non-smooth optimization problem. *Show that* f is a convex function. We can avoid the non-smoothness here by introducing some simple constraints:

$$\begin{aligned} \min_{\gamma, \mathbf{c}, \mathbf{w}} \ell(\gamma, \mathbf{c}) + \alpha \sum_{i=1}^{n} w_i \qquad & \text{subject to} \\ w_i \geq +c_i, \qquad & i = 1, 2, \ldots, n, \\ w_i \geq -c_i, \qquad & i = 1, 2, \ldots, n. \end{aligned}$$

Write out the KKT necessary conditions for this problem. Suppose that we have optimal γ, $\mathbf{c}$, $\mathbf{w}$. *Determine* the appropriate Lagrange multipliers in terms of γ, $\mathbf{c}$, $\mathbf{w}$ so that the KKT conditions can be verified.

We will carry out this minimization using an *active set strategy*: for a given $\mathbf{c}$, for each i we can put $w_i = s_i c_i$ with $s_i = \pm 1$ chosen so that $w_i \geq 0$. We can use the gradient of $\ell(\gamma, \mathbf{c}) + \alpha \sum_{i=1}^{n} s_i c_i$ with respect to $(\gamma, \mathbf{c})$ to determine a direction to move in. However, we should not move so far that c_i changes sign for $i = 1, 2, \ldots, n$; we can allow $c_i = 0$. If $c_i = 0$ and $|\partial \ell / \partial c_i(\mathbf{c})| \leq \alpha$ then we should keep $c_i = 0$. But if $c_i = 0$ and $\partial \ell / \partial c_i(\mathbf{c}) > \alpha$ then we should make c_i negative; if $c_i = 0$ and $\partial \ell / \partial c_i(\mathbf{c}) < -\alpha$ then we should make c_i positive. For each i we keep track of a variable $s_i \in \{-1, 0, +1\}$ so that $s_i = \text{sign}(c_i)$.

Since $\ell(\gamma, \mathbf{c})$ is a convex function of $(\gamma, \mathbf{c})$, Hess $\ell(\gamma, \mathbf{c})$ is at least positive semi-definite for every $(\gamma, \mathbf{c})$.

Implement a version of the algorithm described below. Test it on the dataset `icu.dat` (see `icu.txt`) available on ICON. Note that this dataset is actually copyrighted by J. Wiley & Sons. Read it in using the script `read_icu_data_file.m`. All files needed are in `ICU-data.zip` on ICON. Note that column 2 of line i of `data` is the y_i value; the remainder of line i after column 2 is the row vector $\mathbf{x}_i^T$. This can be extracted by the Matlab code

```
yvals = data(:,2); xvals = data(:,3:end);
```

In applying your optimization algorithm, use the initial guess $\mathbf{c} = 0$ and set $\alpha = 0.05$.

Algorithm

The algorithm you will develop is mostly based on Newton's method using the vector $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ of signs ($s_i = 0$ or $s_i = \pm 1$). We first take a Newton step, but keeping $c_j = 0$ if $s_j = 0$, and ignoring the equation $\partial / \partial c_i \left[\ell(\gamma, \mathbf{c}) + \sum_{i=1}^{n} s_i c_i \right] = 0$. This reduces both the number of unknowns and the number of equations to satisfy. Compute the Newton step $(\delta, \mathbf{d})$ where δ is the Newton step for γ and $\mathbf{d}$ the Newton step for $\mathbf{c}$. (To do this we need to solve a linear system for *both* δ and $\mathbf{d}$.) Use an Armijo/backtracking line search *limited by the condition that the sign of each c_i should not change*. That is, if $s_i = +1$ (so $c_i > 0$), we limit α so that $c_i + \alpha d_i \geq 0$, and if $s_i = -1$ we limit α so that $c_i + \alpha d_i \leq 0$. This can be implemented using

$$\alpha_0 \leftarrow \min(1, \min_{i: c_i > 0, d_i < 0} \frac{-c_i}{d_i}, \min_{i: c_i < 0, d_i > 0} \frac{+c_i}{d_i})$$

before starting the line search. If after the step we have $c_i + \alpha d_i = 0$ but $c_i \neq 0$, then when we update c_i, we also need to update $s_i \leftarrow 0$.

If we simply repeated the Newton-type step described above, if $c_i = 0$ for some iteration, it will remain zero for all future iterations. This is not good. So after each Newton-type step, we need to see if $\left| \partial \ell / \partial c_j(\gamma, \mathbf{c}) \right| > \alpha$ where $c_j = 0$. If $\partial \ell / \partial c_j(\gamma, \mathbf{c}) > \alpha$ and

$c_j = 0$ then we want to decrease c_j, and we set $s_j = -1$; if $\partial\ell/\partial c_j(\gamma, \mathbf{c}) < -\alpha$ and $c_j = 0$ then we want to increase c_j, and we set $s_j = +1$. Otherwise we do not change s_j in this step. Then we take a *gradient* step for $\ell(\gamma, \mathbf{c}) + \sum_{i=1}^{n} s_i c_i$. If $c_j = 0$ and $\left|\partial\ell/\partial c_j(\gamma, \mathbf{c})\right| \leq \alpha$ then we do not want to change c_j. So our gradient step direction is $(\delta, \mathbf{d})$ where

$$\begin{aligned} \delta &= -\partial\ell/\partial\gamma(\gamma, \mathbf{c}), \\ d_i &= -\partial\ell/\partial c_i(\gamma, \mathbf{c}) + s_i \qquad \text{if } c_i \neq 0 \text{ or } \left|\partial\ell/\partial c_j(\gamma, \mathbf{c})\right| > \alpha, \\ d_i &= 0 \qquad\qquad\qquad\quad \text{otherwise.} \end{aligned}$$

Again, use an Armijo/backtracking line search for this gradient step.

The overall algorithm involves taking a Newton-type step as described above, checking if $\left|\partial\ell/\partial c_j(\gamma, \mathbf{c})\right| > \alpha$ where $c_j = 0$ for some j; if this is so, then a gradient step is taken. The process then repeats until the KKT conditions are (approximately) satisfied.

Report

There are a number of things you need to include in your report.

1. Description of your algorithm and your implementation. You should give pseudo-code or something similar to describe how your algorithm works. The description of the implementation does not need to include the complete source code, unless it is quite compact. You should submit your code as a separate file, perhaps as a zip file. However, you should include documentation showing the main parts of your implementation, and how they are used.

2. Give evidence of the correctness of your implementation. Describe what tests you do. These tests can (and, in fact, should) include tests of the pieces of your implementation as well as the complete system. Computations of gradients and Jacobian matrices are well-known as sources of bugs and so should be tested before using them. Comment on how to tell if the computed results are locally optimal or a KKT point for the problem.

3. Describe the performance of the algorithm both in terms of the accuracy of the result(s), the number of iterations of your method, and the computer time spent in the calculations. Comment on how any poor performance you identify could be improved by either improving the algorithm or by improving the implementation. Does the algorithm fail completely? If so, why? And what could be done to remedy the problem?

4. Report the results for the above problems. When you do this, you should think carefully about how to represent them so that the reader will be able to extract useful information from your report. Consider combining graphs so that results can be easily compared. Report on the effects of changing the value of α as this is a parameter that is under the control of the user.