Optimization Project

Nick Kallfa

4-30-15

1 Introduction

In this project we set out to solve a sparse logistic regression problem. The data was obtained from a study performed on patients admitted to an adult intensive care unit (ICU). The goal is to develop a model to predict the probability of survival of the patients being admitted into the ICU based on various predictors.

We can formulate this as a probalistic model and try to determine the optimal variables that maximize the likelihood of predicting the true outcome using the predictors. Formulating the problem in this way allows us to represent it as a constrained optimization problem for which we can develop algorithms to solve.

In this report we will describe the data being used in the implementation as well as the problem we are trying to solve. We will also explain the algorithm being used and give results of the performance of our algorithm and address any poor performance or problems within the code.

2 The Data

The data was obtained from a study performed at Baystate Medical Center in Springfield, Massachusetts. The data consists of 200 data points where each data point is a vector of length 19. The entire collection of data points is organized into a 200×19 size matrix which we will call the observation matrix. A single data point consists of various measurements or attributes for a single patient admitted to the ICU. These attributes/measurements include the patient's age, blood pressure, heart rate, and gender among others. These are the predictors used in determining whether or not a patient lived or died. See figure 1 for the entire list of predictors.

In addition to the observation matrix, there is also, for each data point, a scalar value y associated to it. That is, for the ith row of the observation matrix there is a $y_i \in \{0, 1\}$ which represents the outcome. If the patient survived, then to the data point representing that patient, we associate a y value of 1. If a patient did not survive the ICU visit, then we associate a y value of 0 to the data point representing the patient. All of the corresponding y values are organized into a column vector $\mathbf{y}$ of length 200 which we will call the outcome vector. The element in the *ith* position of the vector $\mathbf{y}$ tells us whether the patient whose attributes are stored in the *ith* row of the observation matrix survived the ICU.

Figure 1: List of Predictors

1. Age
2. Gender
3. Race
4. Service at ICU Admission
5. Cancer Present
6. History of Chronic Renal Failure
7. Probability of Infection
8. CPR Performed Prior to Admission
9. Systolic Blood Pressure
10. Heart Rate
11. Previous Admission to ICU within 6 Months
12. Type of Admission
13. Long Bone, Multiple, Neck, Single Area, or Hip Fracture
14. PO2 from Initial Blood Gases
15. PH from Initial Blood Gases
16. PC02 from Initial Blood Gases
17. Bicarbonate from Initial Blood Gases
18. Creatinine from Initial Blood Gases
19. Level of Consciousness

Using the data contained in the observation matrix and the known outcomes for each of the 200 patients, our goal is to develop a model that when given a new data point $\mathbf{x} \in \mathbb{R}^n$ we can predict the outcome that occurs.

3 The Problem

The observation matrix is a collection of vectors or data points $\mathbf{x}_i \in \mathbb{R}^n$ for $i = 1, 2, .., 200$. For our particular data set, we have 19 attributes per data point so that $n = 19$. For the *ith* data point we have an associated $y \in \{0, 1\}$ which is stored in the *ith* position of the outcome vector $\mathbf{y}$. If we are given a new data point $\mathbf{x} \in \mathbb{R}^n$ we would like to develop a model that accurately predicts the outcome y. We can formulate this as a probabilistic model and try to maximize the likelihood of an accurate prediction.

For each $\mathbf{x}_i$ we have a probability $p(\mathbf{x}_i)$ that $y = 1$ and a probability $1 - p(\mathbf{x}_i)$ that $y = 0$ so that for each i we have a likelihood function given by:

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

The likelihood for the entire data as is then given by:

$$P := \prod_{i=1}^{N} \Big[p(\mathbf{x}_i)^{y_i} \cdot \Big(1 - p(\mathbf{x}_i)\Big)^{1-y_i}\Big] \tag{1}$$

We assume that the probability function $p(\mathbf{x})$ is given by:

$$p(\mathbf{x}) := \frac{1}{1 + e^{\gamma + \mathbf{c}^T \mathbf{x}_i}}$$

Our goal is to determine the optimal γ and $\mathbf{c}$ that maximizes P as a function of $(\gamma, \mathbf{c})$. As shown in the appendix, maximizing P is equivalent to minimizing the log likelihood function given by $\ell(\gamma, \mathbf{c})$ below.

$$\ell(\gamma, \mathbf{c}) := \sum_{i=1}^{N} \Big[y_i \ln\Big(1 + e^{\gamma + \mathbf{c}^T \mathbf{x}_i}\Big) + (1 - y_i) \ln\Big(1 + e^{-\gamma - \mathbf{c}^T \mathbf{x}_i}\Big)\Big] \tag{2}$$

We would like to find a suitable sparse $\mathbf{c}$ that minimizes ℓ so we must add a penalty term with a parameter $\mu > 0$ which controls how many entries of $\mathbf{c}$ are non-zero. The penalty term we use is the 1-norm of $\mathbf{c}$ denoted by

$$||\mathbf{c}||_1 = \sum_{i=1}^{n} |c_i| = \sum_{i=1}^{n} w_i \text{ where } w_i = |c_i| = s_i c_i \text{ with } s_i \in \{-1, 0, 1\}$$

Note that s_i is chosen to ensure that $w_i \geq 0$. In other words, $\mathbf{s}$ simply represents the signs of each of the element in $\mathbf{c}$. Let our penalized function be denoted by $f(\gamma, \mathbf{c})$ and it is given by:

$$f(\gamma, \mathbf{c}) = \ell(\gamma, \mathbf{c}) + \mu||\mathbf{c}||_1 \tag{3}$$

This function f is a convex function as proved in the appendix. However, this is a non-smooth optimization problem, which we can force to be a smooth constrained optimization problem by introducing some constraints. The entire constrained optimization problem we are trying to solve is stated below:

$$\min_{\gamma, \mathbf{c}, \mathbf{w}} \Big(\ell(\gamma, \mathbf{c}) + \mu \sum_{i=1}^{n} w_i\Big) \qquad \text{subject to} \tag{4}$$

$$\begin{aligned} w_i &\geq +c_i && \text{for } i = 1, 2, .., n \\ w_i &\geq -c_i && \text{for } i = 1, 2, .., n \end{aligned}$$

4 The Algorithm

One iteration of the algorithm involves the following three steps:

1. Newton step for γ and non-zero entries of $\mathbf{c}$
2. Check the entries of $\mathbf{c}$ that are zero to determine if we should increase or decrease the entries of $\mathbf{c}$ that are zero
3. Gradient step for γ and entries of $\mathbf{c}$ satisfying a criterion

Note that in each iteration we are always computing a newton and gradient step for γ, but that we are selectively determining which entries of $\mathbf{c}$ should be updated during both the newton and gradient step. Depending on a given criteria we can update entries of $\mathbf{c}$ with a gradient and/or newton step. We will explain how to update $\mathbf{c}$ in the following subsections.

Again, the three steps listed above represent one iteration of the algorithm. We then repeat this process until the gradient of the Lagrangian function is below a specified tolerance. We use Grad$L$ to represent the gradient of the Lagrangian. The following three subsections explain the details of each of the three steps listed above.

4.1 Newton Step

We first take a newton step for γ and for the non-zero entries of $\mathbf{c}$. Let the Newton direction be denoted by $\mathbf{D}^N = \left[\mathbf{d}^N, \delta^N\right]$ (N stands for Newton). δ^N is the newton direction for γ and $\mathbf{d}^N$ is the newton direction for $\mathbf{c}$. This is given by solving the linear system

$$\text{Hessf}(\gamma, \mathbf{c}) \cdot \mathbf{D}^N = -\text{Gradf}(\gamma, \mathbf{c})$$

for $\mathbf{D}^N$.

We have defined Gradf and Hessf in our implementation as given below:

$$\text{Gradf} := \left[\frac{\partial f}{\partial c_1} + \mu \cdot s_1, \frac{\partial f}{\partial c_2} + \mu \cdot s_2, \ldots, \frac{\partial f}{\partial c_{19}} + \mu \cdot s_{19}, \frac{\partial f}{\partial \gamma}\right] \tag{5}$$

and

$$\text{Hessf} := \begin{bmatrix} \frac{\partial^2 f}{\partial c_1^2} & \frac{\partial^2 f}{\partial c_1 \partial c_2} & \cdots & \cdots & \frac{\partial^2 f}{\partial c_1 \partial c_{19}} & \frac{\partial^2 f}{\partial c_1 \partial \gamma} \\ \frac{\partial^2 f}{\partial c_2 \partial c_1} & \frac{\partial^2 f}{\partial c_2^2} & \cdots & \cdots & \frac{\partial^2 f}{\partial c_2 \partial c_{19}} & \frac{\partial^2 f}{\partial c_2 \partial \gamma} \\ \vdots & \vdots & \ddots & & \vdots & \vdots \\ \frac{\partial^2 f}{\partial c_{18} \partial c_1} & \frac{\partial^2 f}{\partial c_{18} \partial c_2} & \cdots & \frac{\partial^2 f}{\partial c_{18}^2} & \frac{\partial^2 f}{\partial c_{18} \partial c_{19}} & \frac{\partial^2 f}{\partial c_{18} \partial \gamma} \\ \frac{\partial^2 f}{\partial c_{19} \partial c_1} & \frac{\partial^2 f}{\partial c_{19} \partial c_2} & \cdots & \cdots & \frac{\partial^2 f}{\partial c_{19}^2} & \frac{\partial^2 f}{\partial c_{19} \partial \gamma} \\ \frac{\partial^2 f}{\partial \gamma \partial c_1} & \frac{\partial^2 f}{\partial \gamma \partial c_2} & \cdots & \cdots & \frac{\partial^2 f}{\partial \gamma \partial c_{19}} & \frac{\partial^2 f}{\partial \gamma^2} \end{bmatrix} \tag{6}$$

Solving for $\mathbf{D}^N$ gives us a direction to move γ and non-zero entries of $\mathbf{c}$ in.

Since we are only updating non-zero entries of $\mathbf{c}$ then in solving the linear system for the newton direction we can ignore any element of the gradient $\text{Gradf}_i(\gamma, \mathbf{c})$ where $c_i = 0$. That is, if the *ith* element of $\mathbf{c}$ is zero we ignore the *ith* element of the gradient of f. We also ignore the *ith* row and *ith* column of the Hessian matrix of f due to the symmetry of the Hessian matrix. This reduces the number of equations and number of unknowns when solving the linear system.

Once we have computed the newton direction for γ and non-zero elements of $\mathbf{c}$, we then need to determine how far we move in that direction. In other words, we need to perform a line search to determine an appropriate step size in the direction $\mathbf{D}^N$. Let α denote the optimal step length.

We want to take a newton step for entries of $\mathbf{c}$, however we must not move the entries of $\mathbf{c}$ so far that they change sign. In other words, we must limit the line search in that the sign of each element of $\mathbf{c}$ should not change. This can be done by choosing the initial estimate for α, denoted by α_0, satisying

$$\alpha_0 = \min\Big(1, \min_{i:c_i>0, d_i^N<0} \frac{-c_i}{d_i^N}, \min_{i:c_i<0, d_i^N>0} \frac{+c_i}{d_i^N}\Big) \tag{7}$$

Once we have computed α_0 we can then begin the line search to find an appropriate step size α using the any decrease criterion. Upon exiting the line search we have computed an appropriate step size α for which we can then move γ and the non-zero entries of $\mathbf{c}$ in the direction $\mathbf{D}^N$.

The outputs from this step is the new function value, the updated vector $\mathbf{c}$, the updated value for γ, the updated vector of signs $\mathbf{s}$, and the new gradient of f.

4.2 Checking Zero Entries of c

Recall that we only updated non-zero entries of $\mathbf{c}$ in the newton step. We did not change any entries of $\mathbf{c}$ that were zero. Entries of $\mathbf{c}$ that are zero as inputs in Newton's method remain zero when the updated $\mathbf{c}$ is output from Newton's method so if any entry of $\mathbf{c}$ becomes zero during an iteration, it will remain zero during all future iterations. We need a way to determine whether or not we should increase or decrease the zero entries of $\mathbf{c}$ and this is the step of the algorithm where we make that determination.

In order to determine whether or not we should increase or decrease zero entries of $\mathbf{c}$ we need to determine if

$$\left|\frac{\partial \ell}{\partial c_i}(\gamma, \mathbf{c})\right| > \mu \text{ when } c_i = 0 \tag{8}$$

If $\frac{\partial \ell}{\partial c_i}(\gamma, \mathbf{c}) > \mu$ when $c_i = 0$ then we want to decrease c_i by setting $s_i = -1$.

If $\frac{\partial \ell}{\partial c_i}(\gamma, \mathbf{c}) < -\mu$ when $c_i = 0$ then we want to increase c_i by setting $s_i = +1$. Otherwise, we do not change s_i. Recall that $\mu > 0$ is a parameter chosen by the user to control how many entries of $\mathbf{c}$ are non-zero.

The function used in our algorithm checks to see if $\left|\frac{\partial \ell}{\partial c_i}\right| > \mu$ and $c_i = 0$ simultaneously. If it does, then it stores the index of the entry of $\mathbf{c}$ where this occurs in an indexing vector. Otherwise, the indexing vector is simply assigned a value of zero. Essentially, I am storing the indices of $\mathbf{c}$ for which $\left|\frac{\partial \ell}{\partial c_i}\right| > \mu$ and $c_i = 0$ simultaneously. This indexing vector is output from the function and used as input for the gradient step. This will help us determine which entries of $\mathbf{c}$ should be updated in the gradient step and which ones should not be updated.

4.3 Gradient Step

The gradient step is the final portion for a single iteration of the algorithm. Let the gradient direction be denoted by $\mathbf{D}^G = [\mathbf{d}^G, \delta^G]$ where δ^G is the gradient direction for γ and $\mathbf{d}^G$ is the gradient direction for $\mathbf{c}$. Recall that $\mathbf{D}^G = -\text{Gradf}(\gamma, \mathbf{c})$.

We will always take a gradient step for γ. However we will only take gradient steps for entries of $\mathbf{c}$ that are either non-zero or satisfy the condition given in (8).

Determining which entries of $\mathbf{c}$ to take a gradient step for is determined by the indexing vector explained in the previous subsection. If the *ith* element of the indexing vector is non-zero, then we take a gradient step for c_i. Otherwise, if the *ith* element of the indexing vector is zero, then we do not take a gradient step for c_i. As in the newton step we compute an initial estimate of the step size α by (7) to ensure that the entries of $\mathbf{c}$ do not change sign during the line search. A line search is then performed to find an appropriate step size α using the any decrease criterion.

The outputs of this step is the new function value, the updated $\mathbf{c}$, the updated value for γ, the updated vector of signs $\mathbf{s}$, and the new gradient of f.

5 Testing

In order to ensure the code we wrote is working correctly we performed a number of tests for the different functions used in the algorithm. There are three important functions in our code that could be a source of bugs. The function file that computes function, gradient, and hessian evaluations should be tested along with the function for Newton's method and the gradient method. We explain the tests and give the results of these tests for these three important functions below.

5.1 Testing the Function File

Gradient and hessian evaluations are well-known sources of bugs, so we should check to see if the function file is computing function, gradient, and hessian evaluations correctly. Instead of using a 200×19 observation matrix, an outcome vector of length 200 and a vector $\mathbf{c}$ of length 19 into our function, we could use smaller size inputs for the observation matrix, outcome vector, and $\mathbf{c}$.

We could use a scalar value of 1 for every input of our function file and this is something we would be able to verify by hand. Evaluating the function file with every input set to 1 we get the following three outputs for the function, gradient, and hessian:

$$f(1,1) \approx 3.1269 \qquad \text{Grad}f(1,1) \approx [1.8808, 0.8808]$$

$$\text{Hess}f(1,1) \approx \begin{bmatrix} 0.105 & 0.105 \\ \\ 0.105 & 0.105 \end{bmatrix}$$

We can determine the true function, gradient and hessian for $(\gamma, c) = (1, 1)$ and doing so we get that

$$f(1,1) = \ln(1+e^2) + 1 \approx 3.1269 \qquad \text{Grad}f(1,1) = \left[\frac{e^2}{1+e^2} + 1, \frac{e^2}{1+e^2}\right] \approx [1.8808, 0.8808]$$

$$\text{Hess}f(1,1) = \begin{bmatrix} \frac{e^2}{(1+e^2)^2} & \frac{e^2}{(1+e^2)^2} \\ \frac{e^2}{(1+e^2)^2} & \frac{e^2}{(1+e^2)^2} \end{bmatrix} \approx \begin{bmatrix} 0.105 & 0.105 \\ 0.105 & 0.105 \end{bmatrix}$$

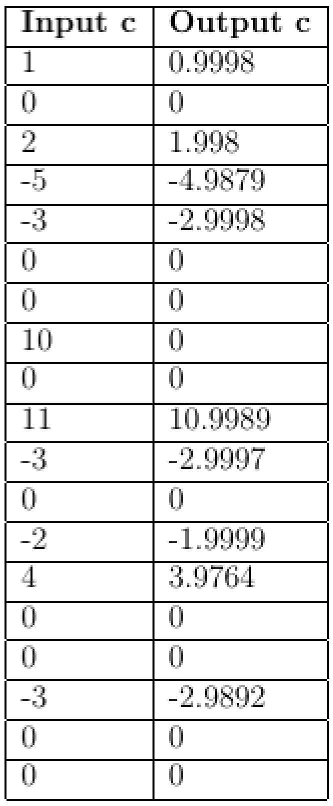
So we see that the hand calculations match with our computations performed on the computer. This gave us some reassurance that our function file for computing function, gradient, and hessian evaluations is performing correctly.

5.2 Testing Newton's Method

We only update non-zero entries of $\mathbf{c}$ during the newton step. So, if we input a vector with some entries that are zero and other entries that are not zero we would expect the non-zero entries to change and the zero entries would remain zero. This is the test we use to determine if the newton step is performing correctly.

The result of the test are given below in figure 2. Note that input $\mathbf{c}$ refers to the original $mathbfc$ we are using before taking a Newton step and the output $\mathbf{c}$ refers to the updated vector $\mathbf{c}$ after taking a Newton step for the non-zero entries.

Figure 2: Test for Newton's Method



Input c	**Output c**
1	0.9998
0	0
2	1.998
-5	-4.9879
-3	-2.9998
0	0
0	0
10	0
0	0
11	10.9989
-3	-2.9997
0	0
-2	-1.9999
4	3.9764
0	0
0	0
-3	-2.9892
0	0
0	0

Note that the eighth element of the input **c** is 10 whereas the eighth element of the output **c** is 0. All of the other non-zero entries of the input **c** were slightly changed while the eighth entry was significantly changed. This is likely due to a bug which we will discuss later in the report. However, despite this one issue all of the zero entries of the input **c** remained zero entries of the output **c** which is ultimately what we wanted to have happen.

5.3 Testing the Gradient Method

For the gradient method, we will update non-zero entries of **c** and only update the zero entroes of **c** that have

$$\left|\frac{\partial \ell}{\partial c_i}(\gamma, \mathbf{c})\right| > \mu \tag{9}$$

So we performed a test using a random vector as the input **c** and then zeroed some elements of the input **c** manually. If the gradient method was working correctly, then we should see all non-zero elements of the input **c** change as well as the zero entries of the input **c** that satisfy (9). The results are given in figure 3 below.

Figure 3: Test #1 for Gradient Method

Input c	**Gradient Magnitude for Input c**	**Output c**
0.4378	–	0.3311
0	3.674	-0.0547
-2.0442	–	-2.0603
0.8845	–	0.8976
0.2092	–	0.1996
-1.0499	–	-1.0837
-0.3628	–	-0.4681
0	4.9107	-0.0457
3.2603	–	3.2138
0	4.2035	-0.0685
-1.3731	–	-1.3487
2.8091	–	2.5999
0.6138	–	0.5957
-0.1017	–	-0.1238
0	1.0511	-0.0237
-0.1360	–	-0.1523
-0.0408	–	-0.0735
1.0291	–	0.9914
0.1581	–	0

We did not include the gradient magnitude for non-zero entries of the input **c** since they will be updated with a gradient step regardless of the gradient magnitude. For this test we set $\mu = 0.05$ so we note that for each zero entry of the input **c** the gradient magnitude exceeded μ so we would expect the gradient step to update every zero entry of the input **c**. We see from the test that we did in fact update input **c** for all entries that should be updated according to (9).

We also examined what happened if the gradient magnitude does not exceed μ. Keeping the same input **c** as in figure 3, we manually set the gradient magnitude of the second entry of input **c** to be 0.04. Then in this test we would expect the second entry of input **c** to remain zero in the second entry of output **c**. The results are given below in figure 4:

Figure 4: Test #2 for Gradient Method

Input c	**Gradient Magnitude for Input c**	**Output c**
0.4378	–	0.3311
0	0.04	0
-2.0442	–	-2.0603
0.8845	–	0.8976
0.2092	–	0.1996
-1.0499	–	-1.0837
-0.3628	–	-0.4681
0	4.9107	-0.0457
3.2603	–	3.2138
0	4.2035	-0.0685
-1.3731	–	-1.3487
2.8091	–	2.5999
0.6138	–	0.5957
-0.1017	–	-0.1238
0	1.0511	-0.0237
-0.1360	–	-0.1523
-0.0408	–	-0.0735
1.0291	–	0.9914
0.1581	–	0

So zero entries of $\mathbf{c}$ that do not satisfy (9) remain zero as we wanted. This gives us confidence that our algorithm is running as expected.

6 Results of the Test Problem

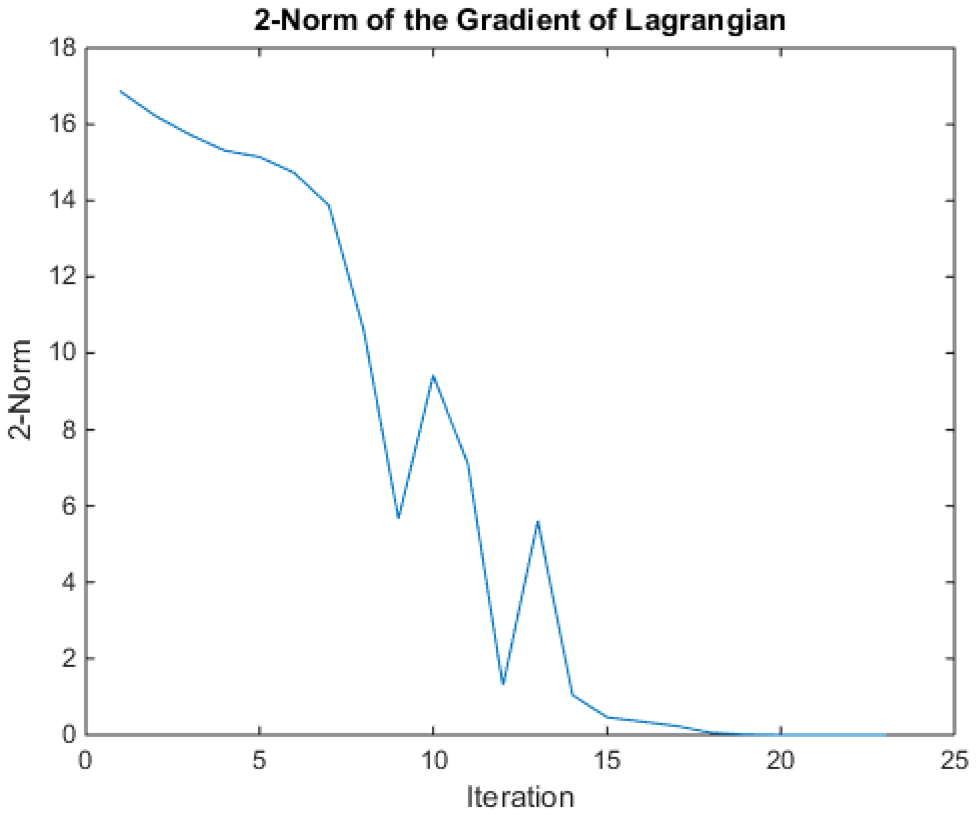
For the test problem we started with initial values of $\mathbf{c} = \mathbf{0}$, $\gamma = 0$, and $\mu = 0.05$. We set a tolerance of 10^{-10} and ended with a final count of 23 iterations to reach a minimum value for f as 66.1360. We provide a list of the function values obtained at the end of each iteration in figure 5.

Figure 5: Function Values at each Iteration

Iteration	Function Value	Iteration	Function Value
1	107.3025	13	68.8964
2	105.8283	14	66.8811
3	104.51	15	66.6269
4	102.4951	16	66.4554
5	101.558	17	66.1798
6	101.174	18	66.1729
7	100.2922	19	66.1361
8	97.8534	20	66.1360
9	92.0711	21	66.1360
10	81.9429	22	66.1360
11	78.3325	23	66.1360
12	74.3091		

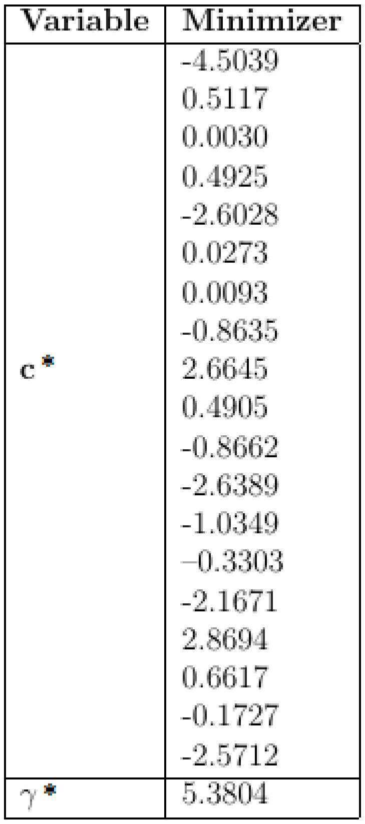
We see that even by the 14*th* iteration the difference between the value at the 14*th* iteration and the final iteration is about 7/10. We are approaching the minimizer rather quickly. The runtime for the algorithm was 0.096780 seconds so it was an immediate computation.

We also provide a plot of the 2-norm of the Lagrangian gradient. This is what was used to control the iterations and was set to stop iterations if the norm of the gradient of the Lagrangian fell below 10^{-10}. This plot is given below:



Note that there is a large drop in the norm between about the *7th* and *8th* iteration and then it jumps up and down a couple times before finally settling down below the threshold. Since this is one of the first order necessary conditions for a constrained optimization problem we would expect the norm of the gradient of the Lagrangian to be 0 at the minimizer. In the figure below we give the minimizer for the constrained optimization problem given in (4).

Figure 6: Minimizer $(\mathbf{c}^*, \gamma^*)$



Variable	Minimizer
	-4.5039
	0.5117
	0.0030
	0.4925
	-2.6028
	0.0273
	0.0093
	-0.8635
$\mathbf{c}^*$	2.6645
	0.4905
	-0.8662
	-2.6389
	-1.0349
	−0.3303
	-2.1671
	2.8694
	0.6617
	-0.1727
	-2.5712
γ^*	5.3804

The first 19 entries are the $\mathbf{c}^*$ that give the minimum value. The γ^* that gives the minimum is $\gamma^* = 5.3804$.

6.1 Changing Parameters

Aside from changing our initial values for $\mathbf{c}$ and γ we also have control over the parameter μ which controls the number of non-zero entries of $\mathbf{c}$. For the test problem given earlier, we set $\mu = 0.05$. We'd like to explore how changing μ affects the results. So we performed a number of tests with different values for μ and give our results below:

Figure 7: Changing μ affects the Minimum Function Value

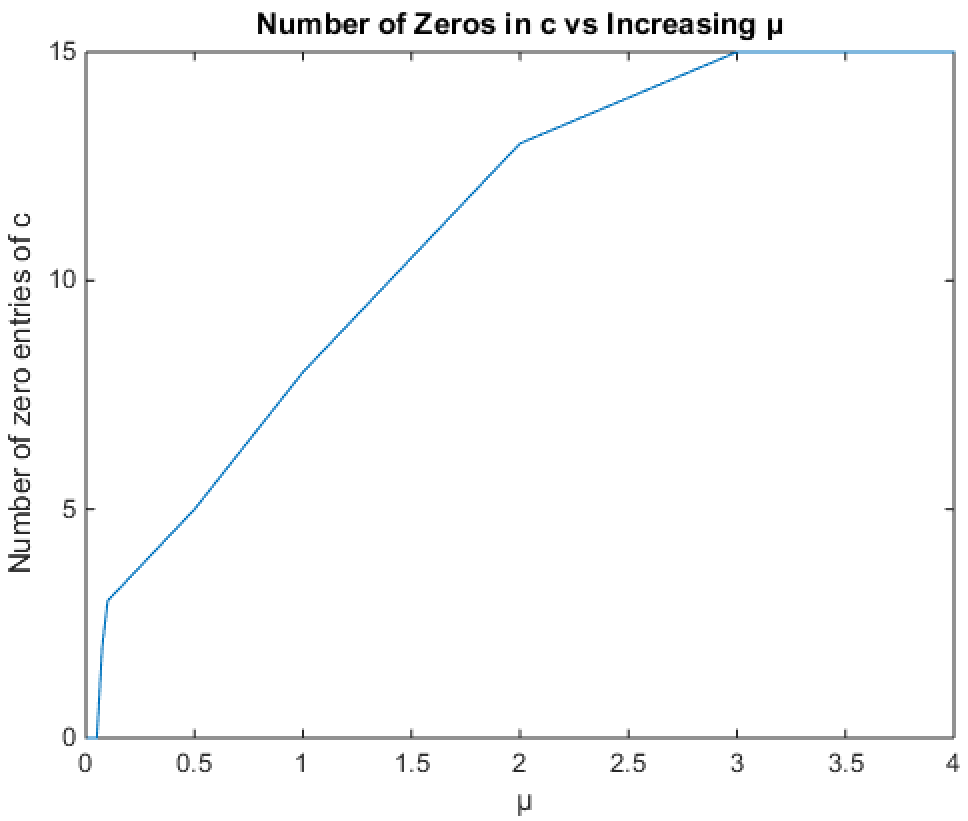
μ	**Minimum Function Value**
0.01	65.0773
0.025	65.4837
0.05	66.1360
0.075	66.7587
0.1	67.3563
0.5	74.5135
1	79.8808
2	85.4782
3	88.5148

We also include the minimizer $[\mathbf{c}^*, \gamma^*]$ for when $\mu = 3$.

Figure 7: Minimizer $(\mathbf{c}^*, \gamma^*)$ for $\mu = 3$

Variable	**Minimizer**
$\mathbf{c}^*$	0
	0
	0
	0.0811
	0
	0
	-0.3041
	0
	0
	0
	0
	-0.7453
	0
	0
	0
	0
	0
	0
	-1.3657
γ^*	2.3167

Note here that a larger choice of μ forced more entries of $\mathbf{c}^*$ to be zero. This was a pattern for the other values of μ that were chosen. As we increased μ more entries of $\mathbf{c}^*$ were zero. The next plot shows how the number of zero entries in $\mathbf{c}$ increased as we changed μ.



Again, μ is a parameter used to control the number of entries of **c** that are non-zero so we knew a priori that this should occur. This plot is simply a confirmation that our algorithm is running correctly.

6.2 Sources of Problems

During the testing of our implementation we identified a couple of problems with the algorithm as we have implemented it. During the newton step, we found that an entry of **c** that was originally non-zero became 0 after taking the newton step. Moreover, this entry was not even close to 0 originally. We concluded that this was not due to taking a large newton step for that entry, but instead must have occurred due to an error in the code implementing newton's method.

We were not able to find the source of this problem, but it is likely due to the fact that the sizes of the gradient vector and hessian matrix during each newton step changes depending on the number of zero entries in **c**. Recall that if the *ith* element of **c** is zero we ignore the *ith* element of the gradient of f. We also ignore the *ith* row and *ith* column of the Hessian matrix of f due to the symmetry of the Hessian matrix. As a result of this, the size of the newton direction will be different during each iteration.

Because of the way we implemented the algorithm, we needed the newton direction to be a vector of length 19. If during one of the iterations the newton direction has a length less than 19, we had to rearrange the entries of the newton direction in a vector of length 19 while placing zeros in the entries that were created. This is where we believe that a 0 was introduced in one of the entries of the updated **c** after taking the newton step.

Another problem we recognized concerned the data that was given to us. We noticed that if we used the observation matrix as is, then the function, gradient, and hessian evaluations blew up and became NaN. This was due to the first, ninth, and tenth columns of the observation matrix. The values in each of these columns were large relative to the other values in the matrix.

To fix this problem we normalized the first, ninth, and tenth columns by dividing each entry of the columns by the infinity norm of that column. Normalizing the data allowed us to compute function, gradient, and hessian evaluations without any values becoming too large for Matlab to handle.

Another problem we noticed is that the algorithm performed poorly when we used the sufficient decrease criterion in the line search. If this was used, then the algorithm never even converged and function values even increased. Only when we used the any decrease criterion did we obtain results that converged to a minimum value. We do not know why this occurred and this is something that we could try to fix if given more time.

7 Conclusion

In conclusion, we would like to make a final remark on a few improvements we could have made if given more time. First, we would have liked to use the data without having to normalize. If given more time, we might have been able to find a way to compute function, gradient, and hessian evaluations that did not blow up. Unfortunately, we did not have enough time and had to slightly change the data in order to actually get finite values.

We were also surprised that using the sufficient decrease criterion performed so poorly. Function values should not increase when using this criteria and they did. If we had more time to look at the implementation we would have tried to determine why this was occurring and use the sufficient decrease criteria in the line search.

8 Appendix

8.1 Finding the Maximum of (1) is Equivalent to Finding the Minimum of (2)

We have defined the likelihood of the entire data set as

$$P := \prod_{i=1}^{N} \left[p(\mathbf{x}_i)^{y_i} \cdot \Big(1 - p(\mathbf{x}_i)\Big)^{1-y_i} \right] \qquad \text{(A)}$$

where

$$p(\mathbf{x}) := \frac{1}{1 + e^{\gamma + \mathbf{c}^T \mathbf{x}_i}}$$

We want to $\max\limits_{\gamma,\mathbf{c}} \Big(P(\gamma, \mathbf{c})\Big)$, but our goal is to show this is equivalent to solving for $\min\limits_{\gamma,\mathbf{c}} \ell(\gamma, \mathbf{c})$ where ℓ is the log-likelihood function defined to be

$$\ell(\gamma, \mathbf{c}) := \sum_{i=1}^{N} \Big[y_i \ln\Big(1 + e^{\gamma + \mathbf{c}^T \mathbf{x}_i}\Big) + (1 - y_i) \ln\Big(1 + e^{-\gamma - \mathbf{c}^T \mathbf{x}_i}\Big) \Big]$$

To do this we start by taking the natural logarithm of both sides of (A) to obtain

$$\begin{aligned}
\ln(P) &= \ln\Big[\prod_{i=1}^{N} p(\mathbf{x}_i)^{y_i} \cdot \Big(1 - p(\mathbf{x}_i)\Big)^{1-y_i} \Big] \\
&= \sum_{i=1}^{N} \ln\Big[p(\mathbf{x}_i)^{y_i} \cdot \Big(1 - p(\mathbf{x}_i)\Big)^{1-y_i} \Big] \\
&= \sum_{i=1}^{N} \ln\Big[p(\mathbf{x}_i)^{y_i} \Big] + \ln\Big[\Big(1 - p(\mathbf{x}_i)\Big)^{1-y_i} \Big] \\
&= \sum_{i=1}^{N} y_i \ln\Big[\frac{1}{1 + e^{\gamma + \mathbf{c}^T \mathbf{x}_i}} \Big] + (1 - y_i) \ln\Big[1 - \frac{1}{1 + e^{\gamma + \mathbf{c}^T \mathbf{x}_i}} \Big] \\
&= \sum_{i=1}^{N} y_i \Big(\ln(1) - \ln(1 + e^{\gamma + \mathbf{c}^T \mathbf{x}_i}) \Big) + (1 - y_i) \ln\Big[\frac{e^{\gamma + \mathbf{c}^T \mathbf{x}_i}}{1 + e^{\gamma + \mathbf{c}^T \mathbf{x}_i}} \Big] \\
&= \sum_{i=1}^{N} y_i \Big(0 - \ln(1 + e^{\gamma + \mathbf{c}^T \mathbf{x}_i}) \Big) + (1 - y_i) \ln\Big[\Big(\frac{1 + e^{\gamma + \mathbf{c}^T \mathbf{x}_i}}{e^{\gamma + \mathbf{c}^T \mathbf{x}_i}} \Big)^{-1} \Big] \\
&= \sum_{i=1}^{N} -y_i \ln\Big(1 + e^{\gamma + \mathbf{c}^T \mathbf{x}_i}\Big) - (1 - y_i) \ln\Big[1 + \frac{1}{e^{\gamma + \mathbf{c}^T \mathbf{x}_i}} \Big]
\end{aligned}$$

$$\begin{aligned}
&= \sum_{i=1}^{N} -y_i \ln\Big(1+e^{\gamma+\mathbf{c}^T\mathbf{x}_i}\Big) - (1-y_i)\ln\Big(1+e^{-\gamma-\mathbf{c}^T\mathbf{x}_i}\Big) \\
&= \sum_{i=1}^{N} -\Big[y_i \ln\Big(1+e^{\gamma+\mathbf{c}^T\mathbf{x}_i}\Big) + (1-y_i)\ln\Big(1+e^{-\gamma-\mathbf{c}^T\mathbf{x}_i}\Big)\Big] \\
&= \sum_{i=1}^{N} -\ell(\gamma,\mathbf{c})
\end{aligned}$$

So from this string of equalities we see that

$$\begin{aligned}
\max_{\gamma,\mathbf{c}} L(\gamma,\mathbf{c}) &\iff \max_{\gamma,\mathbf{c}} \ln\Big(L(\gamma,\mathbf{c})\Big) \\
&\iff \min_{\gamma,\mathbf{c}} -\ln\Big(L(\gamma,\mathbf{c})\Big) \\
&\iff \min_{\gamma,\mathbf{c}} \ell(\gamma,\mathbf{c})
\end{aligned}$$

so we have expressed our maximization problem as an equivalent minimization problem.

8.2 Showing the Log-Likelihood Function is Convex

The log-likelihood function $\ell(\gamma,\mathbf{c})$ is defined to be

$$\ell(\gamma,\mathbf{c}) := \sum_{i=1}^{N} \Big[y_i \ln\Big(1+e^{\gamma+\mathbf{c}^T\mathbf{x}_i}\Big) + (1-y_i)\ln\Big(1+e^{-\gamma-\mathbf{c}^T\mathbf{x}_i}\Big)\Big]$$

First, note that for $\alpha \in [0,1]$ we have that $f(\mathbf{c}) = \mathbf{c}^T\mathbf{x}_i$ is convex since

$$\begin{aligned}
f\Big(\alpha\mathbf{c}_1 + (1-\alpha)\mathbf{c}_2\Big) &= \Big(\alpha\mathbf{c}_1 + (1-\alpha)\mathbf{c}_2\Big)^T\mathbf{x}_i \\
&= \Big(\alpha\mathbf{c}_1^T + (1-\alpha)\mathbf{c}_2^T\Big)\mathbf{x}_i \\
&= \alpha\mathbf{c}_1^T\mathbf{x}_i + (1-\alpha)\mathbf{c}_2^T\mathbf{x}_i \\
&= \alpha f(\mathbf{c}_1) + (1-\alpha)f(\mathbf{c}_2)
\end{aligned}$$

So $f(\mathbf{c}) = \mathbf{c}^T\mathbf{x}_i$ is convex for any $\mathbf{c}$ and since γ is a constant function of itself then we know that γ is convex being a constant. This gives us that $\gamma + \mathbf{c}^T\mathbf{x}_i$ is convex being the sum of two convex functions. Similarly, we can show that $f(\mathbf{c}) = -\mathbf{c}^T\mathbf{x}_i$ is convex and use this to show that $-\gamma - \mathbf{c}^T\mathbf{x}_i$ is convex.

Now since the exponential function is convex and monotonically increasing we know both of the functions given by

$$e^{\gamma+\mathbf{c}^T\mathbf{x}_i}$$

and

$$e^{-\gamma-\mathbf{c}^T\mathbf{x}_i}$$

are convex since in one of the homeworks we showed that if f and g are convex with g non-decreasing then the composition $h = f \circ g$ is convex. This then implies that the two functions given by

$$1 + e^{\gamma+\mathbf{c}^T\mathbf{x}_i}$$

and

$$1 + e^{-\gamma-\mathbf{c}^T\mathbf{x}_i}$$

are convex since they are both the sum of two convex functions. Now if $\ln\left(1+e^{\gamma+\mathbf{c}^T\mathbf{x}_i}\right)$ and $\ln\left(1+e^{-\gamma-\mathbf{c}^T\mathbf{x}_i}\right)$ are convex then we are essentially done since $y_i \in \{0,1\}$ for all i so that $y_i \ln\left(1+e^{\gamma+\mathbf{c}^T\mathbf{x}_i}\right) + (1-y_i)\ln\left(1+e^{-\gamma-\mathbf{c}^T\mathbf{x}_i}\right)$ would be convex being the sum of two convex functions and we are summing this all from $i = 1, .., N$ so since the sum of a finite number of convex functions is convex we would have that $\ell(\gamma, \mathbf{c})$ is convex. So it only remains to show that $\ln\left(1+e^{\gamma+\mathbf{c}^T\mathbf{x}_i}\right)$ and $\ln\left(1+e^{-\gamma-\mathbf{c}^T\mathbf{x}_i}\right)$ are convex and we are done. Let $u = \gamma + \mathbf{c}^T\mathbf{x}_i$ for notational purposes. We just want to show that $f(u) = \ln(1+e^u)$ is convex.

Note that $f'(u) = \dfrac{e^u}{1+e^u} > 0$ and $f''(u) = \dfrac{e^u}{(1+e^u)^2} > 0$ so that $f(u)$ is both increasing and concave up so that it is convex. Similarly, by taking derivatives we can show that $\ln(1+e^{-u})$ is convex since the first derivative is always negative and the second derivative is always positive so the function is decreasing and concave up so it is convex. So $\ell(\gamma, \mathbf{c})$ is convex and we are finally done.

8.3 f is a Convex Function

We want to show that f as given by

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

is a convex function with $\mu > 0$ where $\ell(\gamma, \mathbf{c})$ is defined by

$$\ell(\gamma, \mathbf{c}) := \sum_{i=1}^{N} \Big[y_i \ln\Big(1 + e^{\gamma + \mathbf{c}^T \mathbf{x}_i}\Big) + (1 - y_i) \ln\Big(1 + e^{-\gamma - \mathbf{c}^T \mathbf{x}_i}\Big)$$

Since we already showed that $\ell(\gamma, \mathbf{c})$ is a convex function, then we just need to show that $\mu||\mathbf{c}||_1$ is a convex function since the sum of any two convex functions is also convex. This is easy though since any norm function is convex and since $\mu > 0$ we know that any positive scalar times a convex function is still a convex function so that $\mu||\mathbf{c}||_1$ is convex.

Thus $f(\gamma, \mathbf{c})$ is convex and we are done.

8.4 KKT Conditions

Let L denote the Lagrangian function for the function

$$f(\gamma, \mathbf{c}, \mathbf{w}) := \ell(\gamma, \mathbf{c}) + \mu \sum_{i=1}^{n} w_i$$

where $\ell(\gamma, \mathbf{c})$ is the log-likelihood function given by

$$\ell(\gamma, \mathbf{c}) := \sum_{i=1}^{N} \Big[y_i \ln\Big(1 + e^{\gamma + \mathbf{c}^T \mathbf{x}_i}\Big) + (1 - y_i) \ln\Big(1 + e^{-\gamma - \mathbf{c}^T \mathbf{x}_i}\Big)$$

subject to the constraints given in (4). Then the Lagrangian function L is defined to be

$$L(\gamma, \mathbf{c}, \mathbf{w}, \lambda) := \ell(\gamma, \mathbf{c}) + \mu \sum_{i=1}^{n} w_i - \sum_{i=1}^{n} \lambda_{1,i}(w_i - c_i) - \sum_{i=1}^{n} \lambda_{2,i}(w_i + c_i)$$

Note that λ is a vector of length $2n$ where $\lambda = (\lambda_1, \lambda_2)$ with $\lambda_1, \lambda_2 \in \mathbb{R}^n$. So $\lambda_{1,i}$ simply refers to the *ith* element of λ_1.

The gradient of L with respect to $(\gamma, \mathbf{c}, \mathbf{w})$ is then given by

$$\text{Grad}_{\gamma,\mathbf{c},\mathbf{w}} L = \begin{bmatrix} \dfrac{\partial \ell}{\partial \gamma}(\gamma, \mathbf{c}) \\ \\ \dfrac{\partial \ell}{\partial c_1}(\gamma, \mathbf{c}) + \lambda_{1,1} - \lambda_{2,1} \\ \\ \vdots \\ \\ \dfrac{\partial \ell}{\partial c_n}(\gamma, \mathbf{c}) + \lambda_{1,n} - \lambda_{2,n} \\ \\ \mu - \lambda_{1,1} - \lambda_{2,1} \\ \\ \vdots \\ \\ \mu - \lambda_{1,n} - \lambda_{2,n} \end{bmatrix} \tag{B}$$

Note that $\text{Grad}_{\gamma,\mathbf{c},\mathbf{w}} L$ is a $2n+1$ length vector. Setting this gradient vector for L equal to $\mathbf{0}$ we can solve for the optimal Lagrange Multipliers, $\lambda^* = (\lambda_1^*, \lambda_2^*)$, in terms of $\mu, \mathbf{c}$, and $\mathbf{w}$. We get that

$$\lambda_{1,i}^* = \frac{1}{2}\Big(\mu - \frac{\partial \ell}{\partial \gamma}(\gamma, \mathbf{c})\Big) \qquad \text{for } i = 1, .., n$$

and

$$\lambda_{2,i}^* = \frac{1}{2}\Big(\mu + \frac{\partial \ell}{\partial \gamma}(\gamma, \mathbf{c})\Big) \qquad \text{for } i = 1, .., n$$

So the KKT Necessary Conditions are given by

1. $\text{Grad}_{\gamma,\mathbf{c},\mathbf{w}} L = \mathbf{0}$ where $\text{Grad}_{\gamma,\mathbf{c},\mathbf{w}} L$ is given in (B)
2. $(w_i - c_i) \geq 0$ for $i = 1, .., n$
3. $(w_i + c_i) \geq 0$ for $i = 1, .., n$
4. $\lambda_{1,i}^* \geq 0$ for $i = 1, .., n$
5. $\lambda_{2,i}^* \geq 0$ for $i = 1, .., n$
6. $\lambda_{1,i}^*(w_i - c_i) = 0$ for $i = 1, .., n$
7. $\lambda_{2,i}^*(w_i + c_i) = 0$ for $i = 1, .., n$