CMPE 142 HW2

Jordan Liss

February 16, 2017

1 Question 1: Logistic Regression

(i) Loss Function=

$$L(\hat{y}, y) = -ylog\hat{y} - (1-y)log(1-\hat{y})$$

Prediction Sigmoid Function =

$$\hat{y} = \sigma(\hat{z}) = \frac{1}{1 + e^{(} - \hat{z})}$$

Where,

$$\hat{z} = X^T W$$

The objective is to obtain a gradient of the loss function a iterate until the equation is 0. The gradient of the loss function is:

$$\nabla_w l = X^T(\hat{y} - y)$$

First we will look at the process of taking the derivative of $\hat{y}$ or $\sigma(\hat{z})$. Below is the process:

$$\hat{y} = (1 - e^{(\hat{z})})^{-1}$$

$$\frac{\delta \hat{y}}{\delta z} = -(1 + e^{-\hat{z}})^{-2} - e^{\hat{z}}$$

$$\frac{\delta \hat{y}}{\delta z} = (1 + e^{-\hat{z}})^{-2} e^{\hat{z}}$$

$$\frac{\delta \hat{y}}{\delta z} = \frac{e^{-\hat{z}}}{(1 + e^{-\hat{z}})^2}$$

$$\frac{\delta \hat{y}}{\delta z} = \frac{e^{-\hat{z}}}{(1 + e^{-\hat{z}})} \frac{1}{(1 + e^{-\hat{z}})}$$

$$\frac{\delta \hat{y}}{\delta z} = (1 - \frac{1}{(1 + e^{-\hat{z}})}) \frac{1}{(1 + e^{-\hat{z}})}$$

And if

$$\hat{y} = (1 - e^{(\hat{z})})^{-1}$$

Then,

$$\frac{\delta \hat{y}}{\delta z} = (\hat{y})(1 - (\hat{y}))$$

Now that we have this handy trick, we can walk through taking the gradient of the loss function wrt to w,

$$L(\hat{y}, y) = -ylog\hat{y} - (1-y)log(1-\hat{y})$$

$$\nabla_w(-y \log(\hat{y}) - (1-y) \log(1-\hat{y}))$$

$$\nabla_w(-y \log(\frac{1}{(1 + e^{-\hat{z}})}) - (1-y) \log(1 - \frac{1}{(1 + e^{-\hat{z}})}))$$

$$\nabla_w(-y \log(1) - \log(1 + e^{-\hat{z}})) - (1-y) \log(\frac{e^{-\hat{z}}}{(1 + e^{-\hat{z}})})$$

$$\nabla_w(-y\log(1) - \log(1+e^{-\hat{z}})) - (1-y)(\log(e^{-\hat{z}}) - \log((1+e^{-\hat{z}})))$$

$$\nabla_w(y\log(1+e^{-\hat{z}})) - (1-y)(\hat{z}) - \log((1+e^{-\hat{z}})))$$

$$\nabla_w(-(\hat{z}) - \log((1+e^{-\hat{z}})) + y\hat{z})$$

$$\nabla_w((X^T w) + \log((1+e^{-X^T w})) - yX^T w)$$

$$X^T + \frac{X^T e^{X^T w}}{(1+e^{-X^T w})} - yX^T$$

$$X^T(1 + \frac{e^{X^T w}}{(1+e^{-X^T w})} - y)$$

$$X^T(1 + (1 - \frac{1}{(1+e^{-X^T w})}) - y)$$

$$X^T(\frac{1}{(1+e^{-X^T w})}) - y)$$

$$X^T(\hat{y} - y)$$

(ii) Sigmoid implementation in Matlab

```
function f=sigmoid(z)
        f=1./(1+exp(-z));
end
```

Below is the test of the sigmoid with a vector of values ranging from -1000 to 1000.

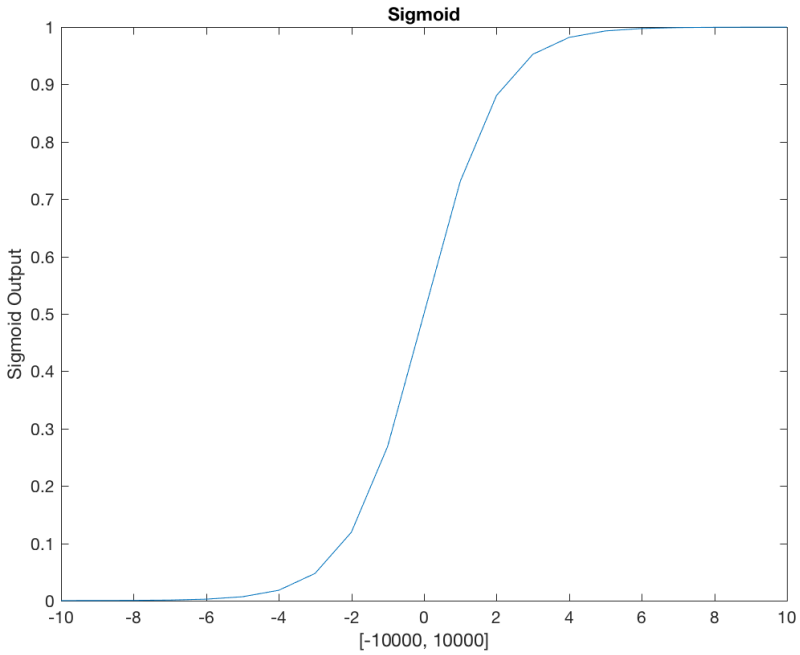

Figure 1: Part 1 Sigmoid Function Test

(iii) Cross entropy implementation in Matlab

```
function [f,G]=cross_entropy(X,w,y)
        z=sigmoid(X*w);
        G=X'*(z-y);
        [xlen,ylen]=size(y);
        for i=1:xlen
```

```
                l(i)=-y(i)*log(z(i))-(1-y(i))*log(1-z(i));
        end
f=sum(l);
end
```

(iv) L2 Regularizer implementation in Matlab

```
function [f,G]=l2_reg(w)
        f=norm(w)^2;
        G=2*w;
end
```

(v) Identical to the homework handout

(vi) Identical to the homework handout

(vii) Identical to the homework handout

(viii) Impementation of classify and accuracy function

```
function [yhat, phat] = classify(X, w)
phat=sigmoid(X*w);
for i=1:length(phat)
        if phat(i) > 0.9
                yhat(i)=1;
        else
                yhat(i)=0;
        end
end
% X is a matrix of training / test data and w is the learned weights
% Returns prediction yhat in {0, 1} and the probability of being amalignant case phat
end
```

The approach is to create a boundary to determine if the prediction is correct or not. %.

The Compute Accuracy Function was a bit more simple. Took the amount of correct predictions over the total number of predictions that can be made.

```
function accuracy = compute_accuracy(y, yhat)
        accuracytemp=abs(yhat-y');
        numofhits=find(accuracytemp == 0);
        accuracy=length(numofhits)/length(accuracytemp)
        % Accuracy is the fraction of correctly classified examples
end
```

After 100 repetitions with a tolerance of 0.01 of my predictions the accuracy of my trained model to the test data given is 87.6 % accurate. After 100 repetitions with a tolerance of 0.01 of my predictions the accuracy of my trained model to the trained data given is 90.7 % accurate.

2 Question 2: Gradient Descent

(i) Implementation of Simple Quadratic Function The implementation of the Simple Quadratic Function starts with taking the gradient of the loss function:

$$G = \begin{bmatrix} 18x - 54 \\ 2x - 8 \end{bmatrix}$$

```
function [f , G] = simple_quadratic (x)
        f=(3.*x(1,:)-9).^2+(x(2,:)-4).^2;
        G=[6.*(3.*x(1,:)-9);2.*(x(2,:)-4)];
        % Compute the value of f and G based on input x.
end

function [xvals, fvals] = gradient_decent(func, x0, options)
        % xvals -> row i + 1 is the value of x at iteration i
        % fvals -> row i + 1 is the value of func(x) at iteration i
        [ftemp,G]=func(x0);
        xvals=x0';
        for i=1:options.NumIterations
                xvals(i+1,:)=xvals(i,:)-options.Stepsize.*G';
                fvals(i+1)=ftemp;
                [ftemp,G]=func(xvals(i+1,:)');
        end
end
```

(iii) Running the gradient descent one can see the plots of the iterations below: 50 iterations with 0.1 stepsize Its clear the converging values are $x_1 = 3$ and $x_2 = 4$.

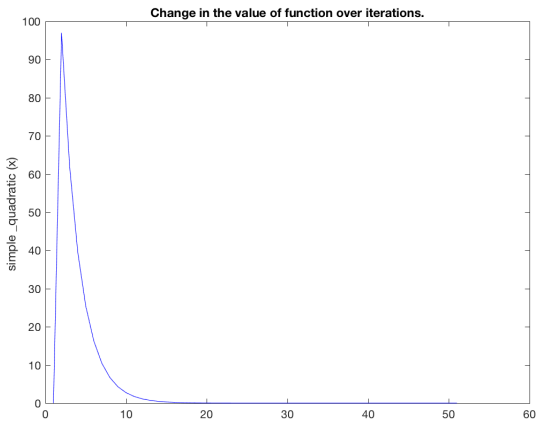

Figure 2: Part 2 First 10 iterations of the Gradient Descent

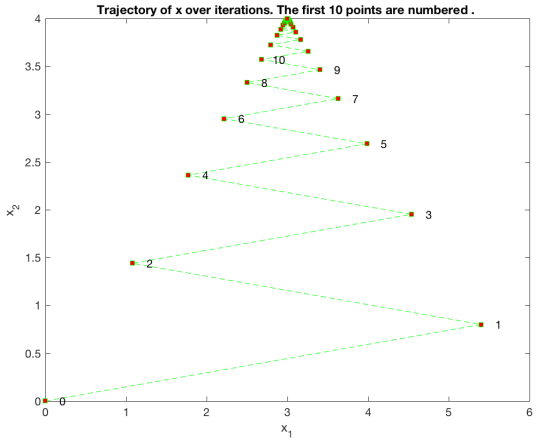

Figure 3: Part 2 Full Gradient Descent Iterations

(vi) 15 iterations with 0.12 stepsize

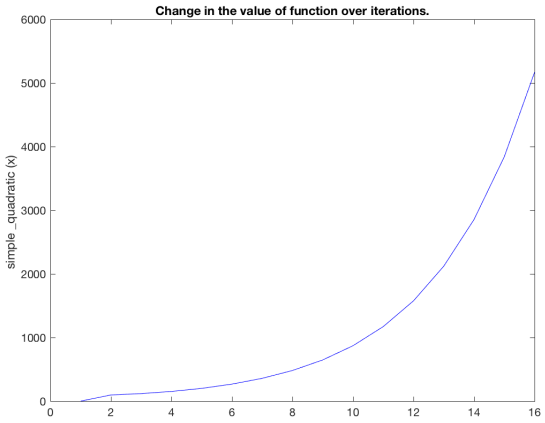

Figure 4: Part 2 First 10 iterations of the Gradient Descent

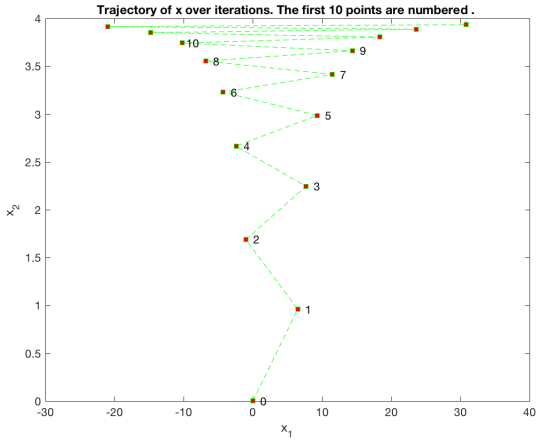

Figure 5: Part 2 Full Gradient Descent Iterations

Difficult to determine if the values are converging, because the function goes unstable. When the stepsize is just larger than 0.1 the gradient descent goes unstable. In figure 4, the output values of f increase exponentially and never plateau to a value, in otherwords never converges to a value.

vii JUST FOR FUN 100 iterations with 0.02

Just for fun I ran the same iteration using a stepsize that is much smaller than 0.1 and the system converges to $x = [3, 4]$ progressively quicker than when the stepsize $= 0.1$. The desired values are still $x = [3, 4]$. Below the system with a stepsize $= 0.02$.

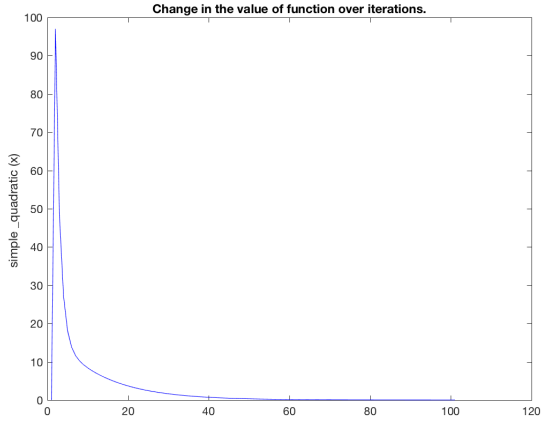

Figure 6: Part 2 First 10 iterations of the Gradient Descent

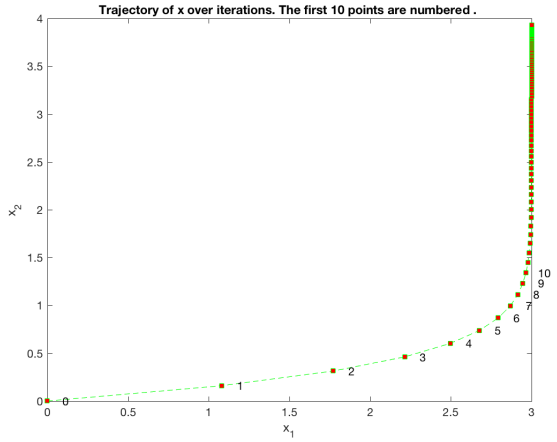

Figure 7: Part 2 Full Gradient Descent Iterations