ISC0630E

Assignment 4
IIT2016067: Maanas Vohra

Question Description

Using the data set of two examination results design a predictor using logistic regression for predicting whether a student can get an admission in the institution. Use regularizer to further tune the parameters. Use 70 % data for training and rest 30% data for testing your predictor and calculate the efficiency of the predictor/hypothesis.

This should be done with delta learning rule using Newton's method and compare the results with using gradient descent.

Introduction

This is used for the classification problem. This is just like the regression problem, except that the values y we now want to predict take on only a small number of discrete values. Given $x^{(i)}$ as the training sample features, the corresponding $y^{(i)}$ is also called the label for the training example.

The i^{th} training sample is represented as $(x^{(i)}, y^{(i)})$.

We are given a dataset comprising of marks obtained by various students in two subjects and a binary result of 0 if the student was not admitted based on his marks and 1 if the student was admitted.

Hypothesis Function

hθ(x) = g (XΘ)

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} \\ x_0^{(2)} & x_1^{(2)} \\ x_0^{(3)} & x_1^{(3)} \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

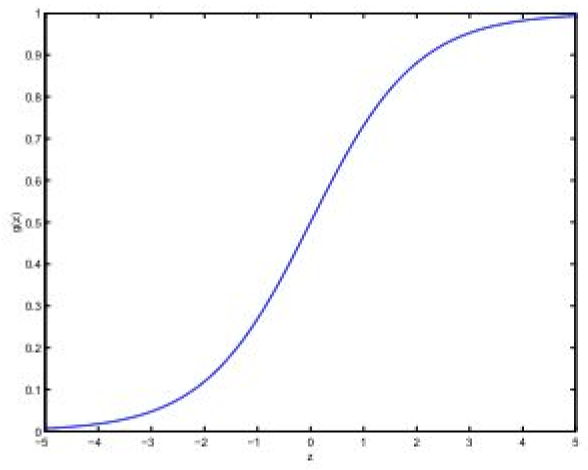
$$g(z) = \frac{1}{1 + e^{-z}}$$

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}. \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}.$$

The hypothesis considers both Θ and x as (n + 1) size vectors.

(1) X = m X (n + 1) matrix, containing the training samples features in the rows, where each $x^{(i)}$ is a (n + 1) X 1 matrix.
(2) m = number of training samples
(3) Θ_i = the parameters
(4) x_0 = 1 for every sample.
(5) n = features count, not including x_0 for every training sample
(6) Y contains all the m target values from the training samples, so it's m X 1 matrix

The hypothesis function here is known as the logistic or the sigmoid function. The plot of the logistic function v/s the input is shown below:



$$h(x) = \begin{cases} > 0.5, & \text{if } \theta^T x > 0 \\ < 0.5, & \text{if } \theta^T x < 0 \end{cases}$$

If the weighted sum of inputs is greater than zero, the predicted class is 1 and vice-versa. So the decision boundary separating both the classes can be found by setting the weighted sum of inputs to 0.

The derivative of the sigmoid function is :

$$\begin{aligned} g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2}\left(e^{-z}\right) \\ &= \frac{1}{(1+e^{-z})}\cdot\left(1-\frac{1}{(1+e^{-z})}\right) \\ &= g(z)(1-g(z)). \end{aligned}$$

Cost Function

The cost function for a single training sample can be found as

$$cost = \begin{cases} -log(h(x), & \text{if } y = 1 \\ -log(1 - h(x)), & \text{if } y = 0 \end{cases}$$

Using this, we get

$$cost(h(x), y) = -ylog(h(x)) - (1 - y)log(1 - h(x))$$

Using regularization and for m samples, each having (n + 1) features we get

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[-y^{(i)} \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2.$$

This represents the required cost function.

Gradient Descent

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta).$$

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \qquad \text{for } j = 0$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j \qquad \text{for } j \geq 1$$

In the vectorized form it becomes,

$$\nabla J(\theta) = \frac{1}{m} \cdot X^T \cdot \left(g(X \cdot \theta) - \vec{y}\right) + \lambda/m * \theta\ (2:n)$$

Thus the gradient descent changes to :

$$\theta := \theta - \alpha \nabla J(\theta)$$

Where α is the learning rate.

The convergence arises when $J_t(\Theta) - J_{t+1}(\Theta) <= \text{ERROR\_THRESHOLD}$

Newton Method

This method is used to find the point when the function has a value of zero, or approximates to zero.

We require that the cost function J(Θ) is minimum. This means that the gradient of the function J'(Θ) is zero or approximates to zero. The newton method works this way by changing the parameters Θ as :

$$\theta := \theta - \frac{f(\theta)}{f'(\theta)}.$$

The parameters change till the value of the function f(Θ) converge. By changing f(Θ) to l'(Θ) we obtain:

$$\theta := \theta - \frac{\ell'(\theta)}{\ell''(\theta)}.$$

(Here l'(Θ) = J'(Θ), they both are the same function)

The Hessian of the k-dimensional function is defined as :

$$H_f(\mathbf{x}) := \nabla^2 f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1^2} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_K} \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2^2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_K} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_K \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_K \partial x_2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_K^2} \end{bmatrix}$$

The algorithm is as follows :

```
1: procedure NR($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
2:     $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$                 ▷ Initialize parameters
3:     while not converged do
4:         $\mathbf{g} \leftarrow \nabla J(\boldsymbol{\theta})$             ▷ Compute gradient
5:         $\mathbf{H} \leftarrow \nabla^2 J(\boldsymbol{\theta})$           ▷ Compute Hessian
6:         $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mathbf{H}^{-1}\mathbf{g}$       ▷ Update parameters
7:     return $\boldsymbol{\theta}$
```

The cost function in logistic regression is :

$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{N} y^{(i)} \log \mu^{(i)} + (1 - y^{(i)}) \log(1 - \mu^{(i)})$$

$$\text{where } \mu^{(i)} := h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = 1/(1 + \exp(-\boldsymbol{\theta}^T \mathbf{x}))$$

The gradient g of the cost function J is thus :

$$\mathbf{g} := \nabla J(\boldsymbol{\theta}) = \sum_{i=1}^{N} (\mu^{(i)} - y^{(i)})\mathbf{x}^{(i)}$$
$$= \mathbf{X}^T(\boldsymbol{\mu} - \mathbf{y})$$

The hessian of the matrix can be calculated as follows:

$$\mathbf{H} := \nabla^2 J(\boldsymbol{\theta}) = \sum_{i=1}^{N} \mu^{(i)}(1 - \mu^{(i)})\mathbf{x}^{(i)}(\mathbf{x}^{(i)})^T$$
$$= \mathbf{X}^T\mathbf{S}\mathbf{X}$$
$$\text{where } \mathbf{S} = \text{diag}(\mu^{(i)}(1 - \mu^{(i)}))$$

Newton’s method typically enjoys faster convergence than (batch) gradient descent, and requires many fewer iterations to get very close to the minimum. One iteration of Newton’s can, however, be more expensive than one iteration of gradient descent, since it requires finding and inverting an n-by-n Hessian; but so long as n is not too large, it is usually much faster overall.

Results:

Using λ = 0.03, ERROR_THRESHOLD = 0.0000001, α = 0.01

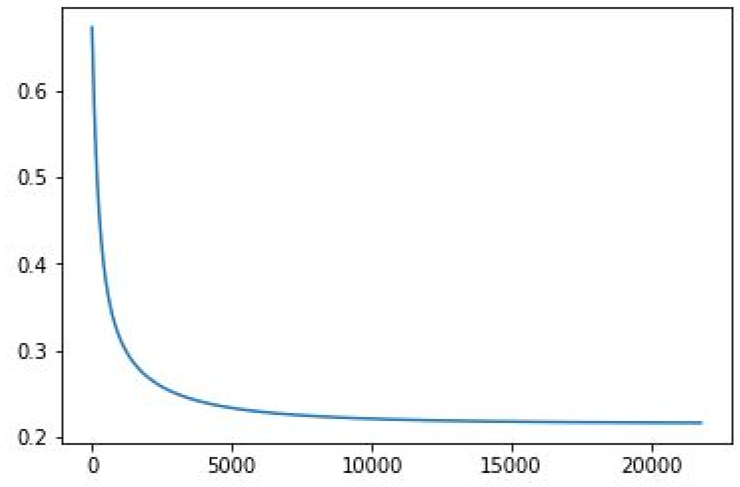
(1) Gradient Descent:

Total iterations taken: 21763

Final error is 0.21610942771516592

Final theta values are [1.69952765 3.45272988 3.18989866]

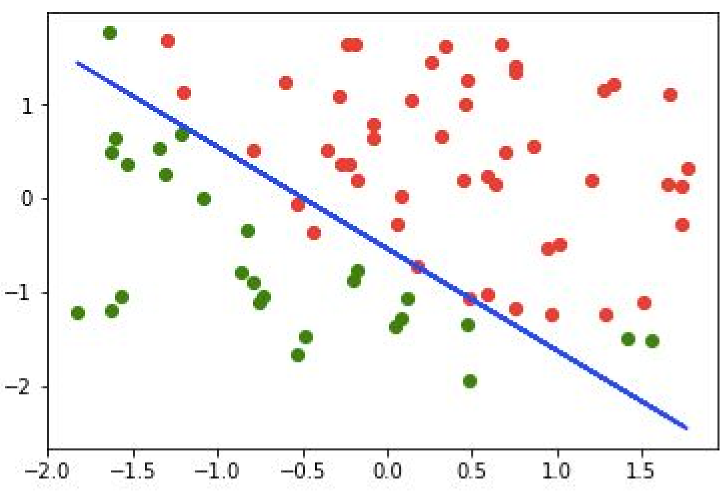
Cost v/s Iterations:



Decision Boundary:

The red dots represent the exam values which are accepted.

The green dots represent the exam values which are not accepted.



Accuracy for training data : 90.66666666666666

Accuracy for testing data : 84.0

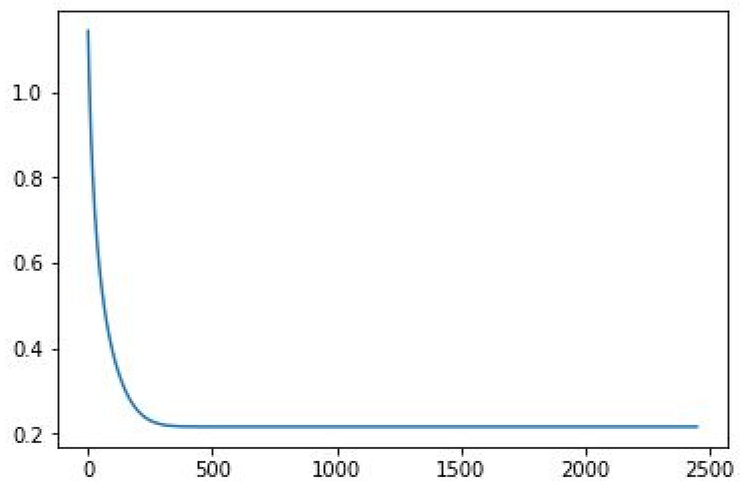
(2) Delta learning rule using Newton's method:

Total iterations taken: 2447

Final error : 0.21551369889295843

Final theta values : [1.84111811 3.71246667 3.44135682]

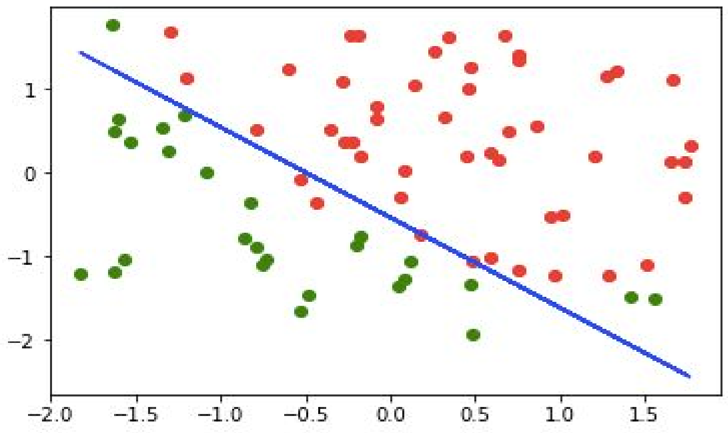
Cost v/s iterations:



Plotting decision boundary :

The red dots represent the exam values which are accepted.

The green dots represent the exam values which are not accepted.



Training Accuracy : 90.66666666666666

Testing Accuracy: 84.0

Comparison : Both the methods have almost same values of theta, final error and same values of test and training accuracy. The only difference here is that newton method takes quite less number of iterations as compared to gradient descent. One iteration of Newton's can, however, be more expensive than one iteration of gradient descent, since it requires finding and inverting an n-by-n Hessian; but so long as n is not too large, it is usually much faster overall.