CS 584

MACHINE LEARNING

ASSIGNMENT 3

by

Saurabh Katkar

A20320336

Code created using Rstudio version 0.98.1062

Logistic Regression

Introduction

Logistic regression is a type of probabilistic classification model in which the probabilities describing the possible outcomes of a single trial are modeled, as a function of the explanatory (predictor) variables, using a logistic function, which is also known as a **'sigmoid function'**. Logistic regression measures the relationship between a categorical dependent variable and one or more independent variables by using **probability scores** as the predicted values of the dependent variable.

Dataset Used: The Iris Dataset

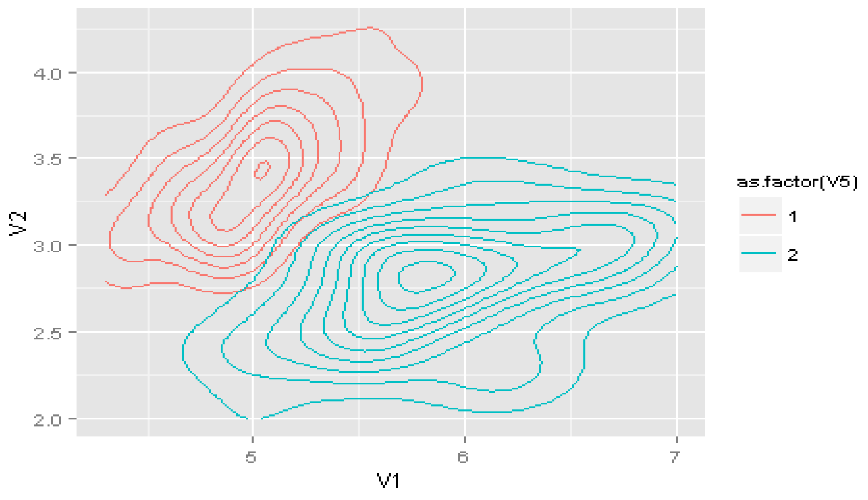
The ***Iris* flower data set** is a multivariate data set consisting of 50 samples from each of three species of *Iris* (*Iris setosa*, *Iris virginica* and *Iris versicolor*). Four features were measured from each sample: the length and the width of the sepals and petals. Using these continuous features classification can be made by developing a generative model of the species each flower in the dataset belongs to.

1. Using a 2 Class Iris Dataset

1.1 Loading the dataset:

The dataset is loaded into **R** using the function **read.table()**. We can subsequently print the contents of the dataset and plot them on the graph using **print()** and **plot()** respectively.
We then split the dataset so as to include only one feature of the dataset mapping two classes.

Thus we obtain the following plot for the dataset;



Performing Logistic Regression on the Iris Data:

We initially set the parameters(theta) to some random values and find out the initial value of the the output(y) variable using the **sigmoid function.**

The sigmoid function is given as –

$$\textit{Sigmoid(x)} = \frac{1}{1+exp(-x)}$$

Using logistic regression we predict the y-value of the data for the purpose of classification. With that into consideration our sigmoid function becomes –

$$H\ \theta\ (x) = \textit{Sigmoid}(\theta^{T}x) = \frac{1}{1+exp\left(-\left((\theta^{T})x\right)\right)}$$

This function is also called as the **logistic function** and it takes on values that are between 0 and 1.

Improving the parameters:

Newton's Method for Optimizing the Parameters:

Initially we assigned random values for our parameters but using the predicted output derived we can estimated the right parameters required. For this we make use of **Newton's method.**

Newton's method, similarly to gradient descent, is a way to search for the minimum of the derivative of the cost function. This method takes an initial guess of the parameters and trains it iteratively to come up with an optimized value of θ. This process continues till it converges to the actual solution.

The Newton method takes into account the values of Gradient, Hessian and the old parameter to come up with an optimized version of the new parameter.

Using Newton's method, we optimize the parameters using **Iterative Reweighted Least Squares (IRLS)** as follows –

$$w_{t+1} = w_t - H^{-1}g$$

where –

w_{t+1} is the new derived parameter

w_t is the old parameter

g is the **Gradient**

Gradient is calculated by taking the product of the transpose of the X-matrix and the difference between the actual y-label and the predicted y-variable.

$$\nabla = \Sigma(X^T(h\theta(x) - y))$$

H is the **Hessian**

Hessian is calculated by taking the product of the X matrix, transpose of the X-matrix, and a diagonal matrix **B** consisting of the product of the predicted y-values.

$$\nabla^2 = \Sigma(X^T B X)$$

Decision Boundary:

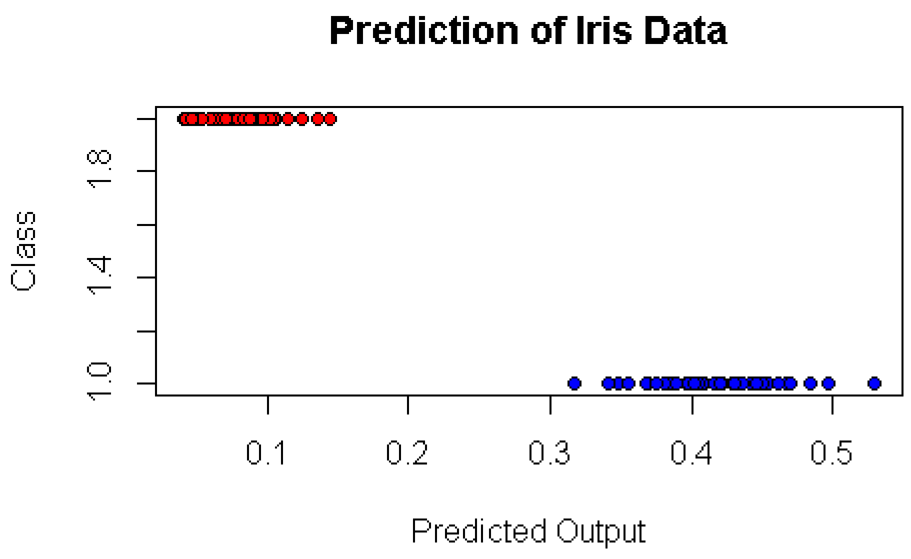
For the logistic regression function

$$y = 1,\ h_{theta}(x) >= 0.5$$

$$y = 0,\ h_{theta}(x) < 0.5$$

Thus using the above equations, we derive the predicted value of the label using logistic regression.

The plot for the data classification based on our prediction of classes is as follows:



Use non-linear combinations of inputs to increase the capacity of the classifier

For this purpose, we employ a polynomial function for multivariate data to create more features based on the input data.

Thus we increase the number of features and map data from low dimension feature space to high dimension feature space.

Performance Evaluation:

We evaluate, with respect to time, **the performance of our logistic regression function** that calculates the **predicted labels** and also performs **parameter optimization** using Newton's method in an iterative manner. This is done using the **system.time()** function of R.

As such the performance measure is as follows:

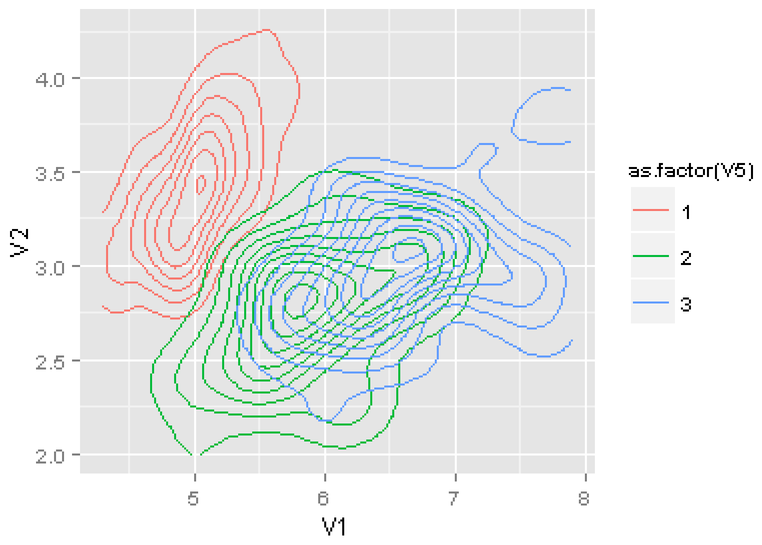
user	system	elapsed
4.64	0.03	4.6

2. Using a 3 Class Iris Dataset

2.1 Loading the dataset:

The dataset is loaded into **R** using the function **read.table()**. We can subsequently print the contents of the dataset and plot them on the graph using **print()** and **plot()** respectively.
We then split the dataset so as to include only one feature of the dataset mapping two classes.

Thus we obtain the following plot for the dataset;



Performing Logistic Regression on the Iris Data:

As before, we initially set the parameters(theta) to some random values and find out the initial value of the the output(y) variable using the **sigmoid function.**

Using logistic regression we predict the y-value of the data for the purpose of classification. With that into consideration our sigmoid function becomes –

$$H\,\theta\,(x) = Sigmoid(\theta^{T}x) = \frac{1}{1+exp\left(-\left((\theta^{T})x\right)\right)}$$

One vs All approach for multiclass classification:

The simplest approach is to reduce the problem of classifying among **K classes into K binary problems**, where each problem discriminates a given class from the other K − 1 classes.

For this approach, we require N = K binary classifiers, where the k^th classifier is trained with positive examples belonging to class k and negative examples belonging to the other K − 1 classes. When testing an unknown example, the classifier producing the **maximum output** is considered the winner, and this class label is assigned to that example.

This approach, although simple, provides performance that is comparable to other more complicated approaches when the binary classifier is tuned well.

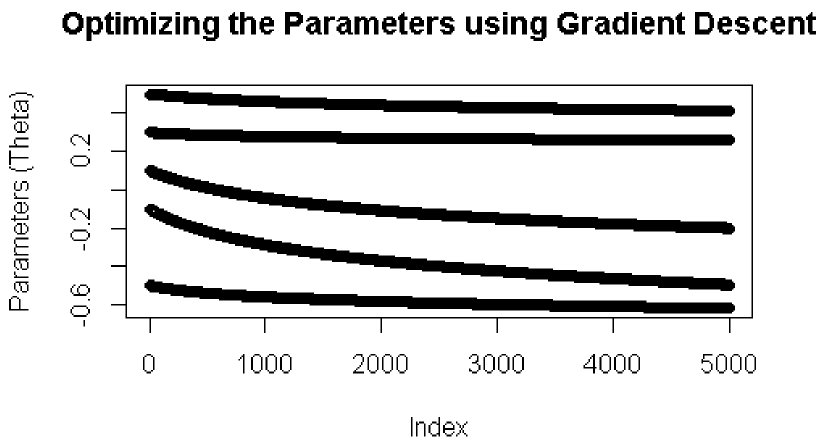
Gradient Descent Algorithm for optimizing the parameters:

In Logistic regression, gradient descent algorithm is used to minimize functions, subsequently minimizing parameters. Given a function defined by a set of parameters, gradient descent starts with an initial set of parameter values and iteratively moves toward a set of parameter values that minimize the function. This iterative minimization is achieved using calculus, taking steps in the negative direction of the function gradient.

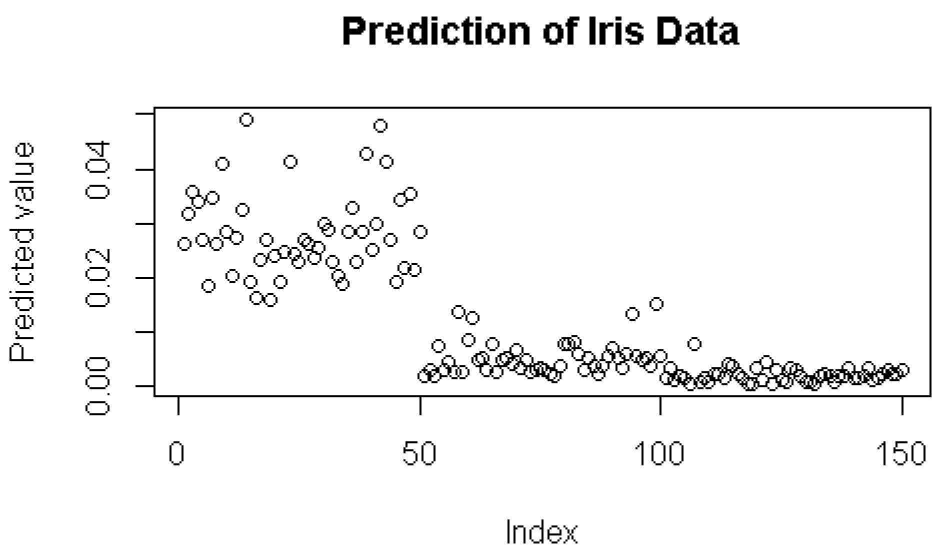
As such the equation for gradient descent in Logistic regression is given as –

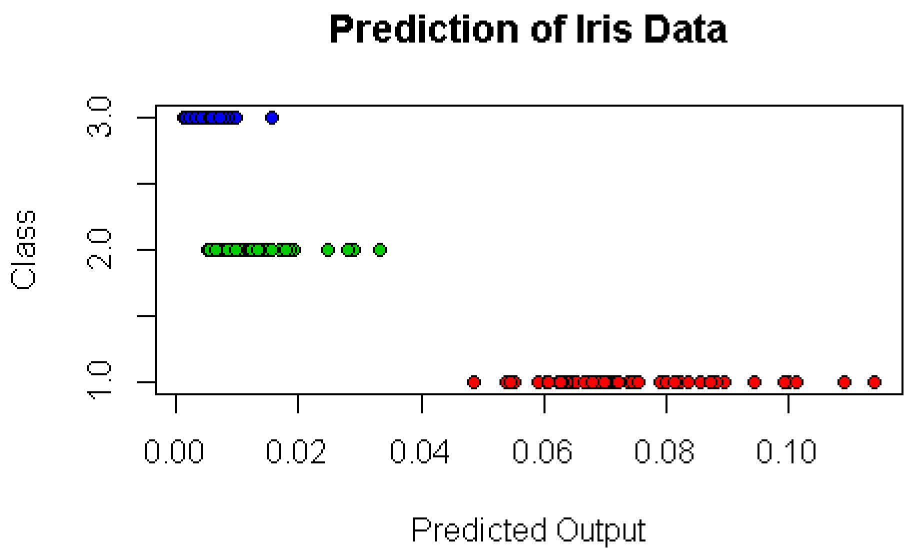
$$\textit{Gradient Descent} = \left(\{\theta_j = \theta_j - n\Sigma\left(\frac{1}{m}\right)\left(h_\theta x^j - y^j\right)x^j\}\right)$$

The plot for the optimized value of the parameters using gradient descent is as follows;



The plot for classifying the data based on our prediction is as follows:





Performance Evaluation:

We evaluate, with respect to time, the performance of our logistic regression function that calculates the predicted labels and also performs gradient descent in an iterative manner. This is done using the **system.time()** function of R.

We take three instances of the function for the three output lists that we have derived for the one vs all algorithm.

As such the performance measure is as follows:

	user	**system**	**elapsed**
Output_1	74.27	2.81	79.44
Output_2	64.96	2.48	68.19
Output_3	60.29	2.23	62.66

<u>Multiplayer Perceptron:</u>

- Assume a two layer feedforward MLP with a single output where all the elements use sigmoid activation (including the output unit). Assume a training dataset $\{x^{(i)}, y^{(i)}\}^{m}_{i=1}$ where $x^{(i)} \in \mathbb{R}^n$ and $y^{(i)} \in \{0, 1\}$. Derive the backpropagation update equations for the weights of the output by minimizing the following error function:

$$1/2\sum^{m}_{i=1}(y^{(i)} - \hat{y}^{(i)})^2.$$

  Compare the result you obtain to that obtained in class using maximum likelihood estimation.

  Solution:

  Please refer **Q2_1_soln.pdf** that is provided.

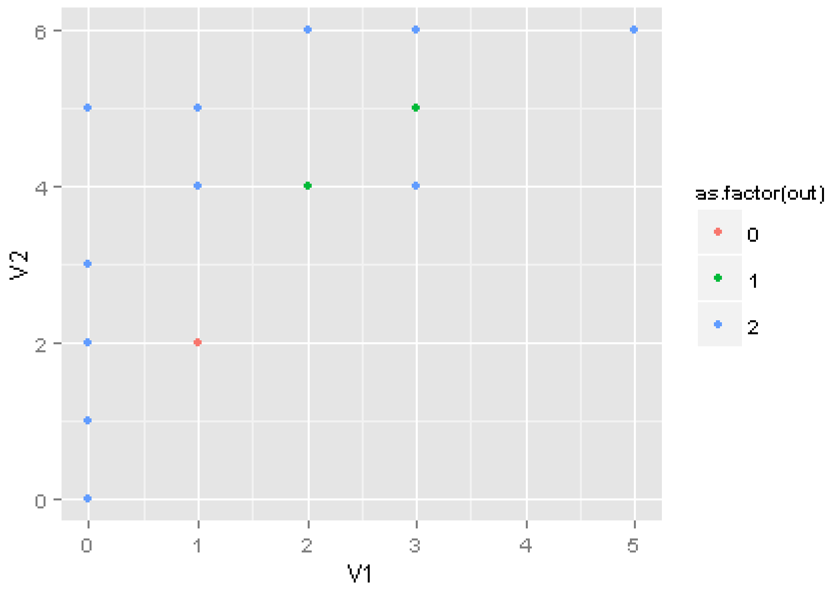
Two layer Feedforward MLP for Three class-classification

Dataset Used: The Multi-Feature (MFEAT) digit dataset

This dataset consists of features of handwritten numerals (`0'--`9') extracted from a collection of Dutch utility maps. 200 patterns per class (for a total of 2,000 patterns) have been digitized in binary images.

In each file the 2000 patterns are stored in ASCI on 2000 lines. The first 200 patterns are of class `0', followed by sets of 200 patterns for each of the classes `1' - `9'. Corresponding patterns in different feature sets (files) correspond to the same original character.

We get the following plot for the digit dataset:



Training the Neural Network:

Picking a network architecture:

We use –

An input layer:

This input layer consists of 600 different values consisting of 240 features that give us the dependent variable.

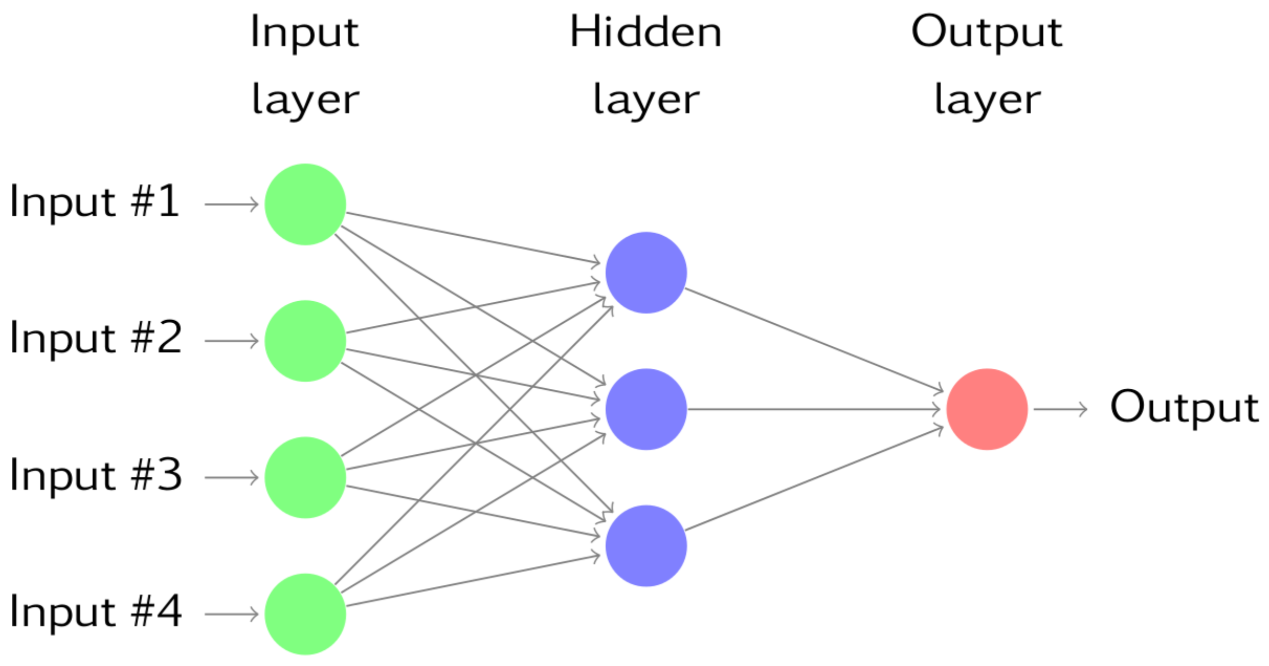
A hidden layer:

The output computed using the inputs and the parameters of the input layer are fed to this hidden layer which computes the final predicted output. In our case we make use of **3 activation units** in our hidden layer.

Output layer:

The output layer takes in the output of the hidden layer as its input along with the parameters to compute the final predicted output.

Example of a two layer neural network:



The Feed forward algorithm:

Using the feed forward algorithm, we train the dataset to compute the output for the hidden layer using a sigmoid activation function which in turn will derive the final output derived from the weighted sum of the parameters.

Initially we do not know the value of the parameters, for that purpose train the algorithm using an estimation of the parameter values.

The algorithm is as follows:

- Start with an initial guess of the parameters. W_i
- Use this to compute the output of the hidden layer $Z^{(i)}$
- Update the parameters using $Z^{(i)}$

We use the **chain rule** for this algorithm using which we get input for the hidden layer.

Thus the equation for calculating $Z^{(i)}$ would be –

$$\boldsymbol{Z} = \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{exp}\left(-\left((\boldsymbol{W^T})\boldsymbol{x}\right)\right)}$$

And the equation for updating W_j would be –

$$\left(\boldsymbol{W_j} = \boldsymbol{W_j} - \boldsymbol{n\Sigma} \left(\boldsymbol{h_\theta x^j - y^j}\right) \boldsymbol{x^j}\right)$$

Backpropagation algorithm:

Using backpropagation algorithm, after updating the parameter of the ith layer, we update the parameter of the (i-1)th layer, thus minimizing the cost function after each iteration.

This results in an optimized value of the parameter using which we can train our dataset and compute the output.

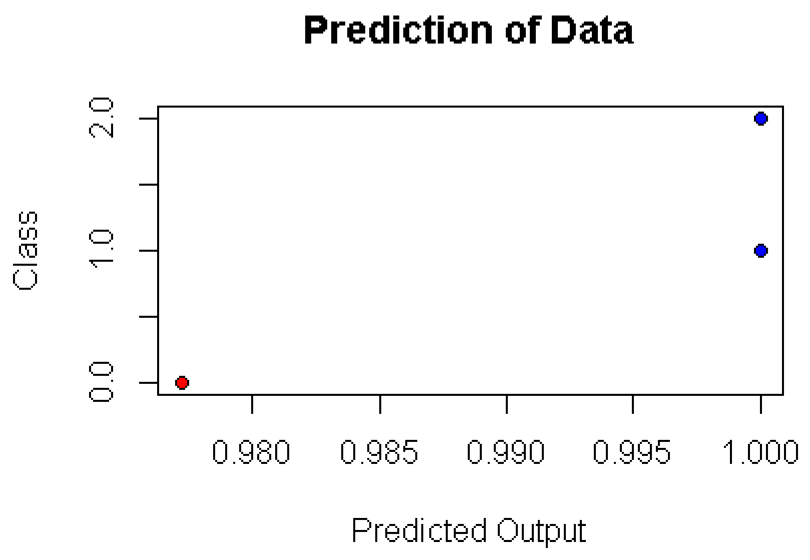
Update rule for V:

$$\left(\boldsymbol{V} = \boldsymbol{V} - \boldsymbol{n\Sigma} \left(\boldsymbol{h_\theta x^j - y^j}\right) \boldsymbol{Z^j}\right)$$

Update rule for W:

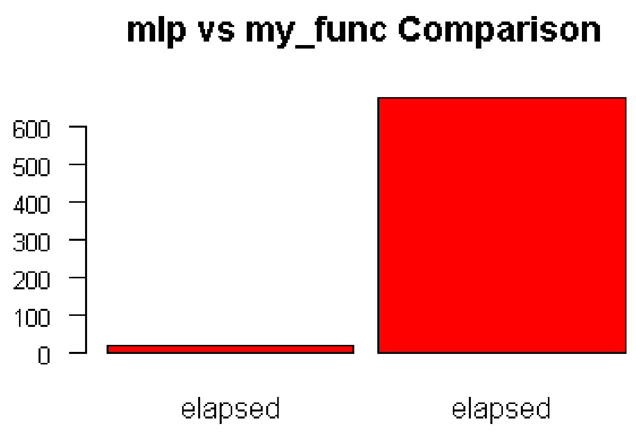
$$\left(\boldsymbol{W_j} = \boldsymbol{W_j} - \boldsymbol{n\Sigma} \left(\boldsymbol{h_\theta x^j - y^j}\right) \boldsymbol{V_j Z_j}(\mathbf{1} - \boldsymbol{Z_j^i})\boldsymbol{X^i}$$

The predicted output obtained is as follows:



The difference in performance with respect to time between the built-in R function and the user function is as follows:

user	system	elapsed	
19.61	0.02	19.63	mlp
669.8	0.19	675.56	my_func


mlp vs my_func Comparison
elapsed
elapsed