Implementation of a classification model in prediction of coronary heart disease

Suraj R Bhute 21M61R03
MTech Medical Imaging and Informatics,
SMST, IIT Kharagpur
Kharagpur, India
surajrbhute123@gmail.com

Priya Sarkar 21MM45003
M.sc. Nuclear Medicine
SMST, IIT Kharagpur
Kharagpur, India
sarkar.priya333@gmail.com

Tushar Dinesh Mahore 21MM61R16
MTech Medical Imaging and Informatics,
SMST, IIT Kharagpur
Kharagpur, India
tusharmahore111@gmail.com

***Abstract*—World Health Organization has estimated 12 million deaths occur worldwide, every year due to Heart diseases. Half the deaths in the United States and other developed countries are due to cardio vascular diseases. The early prognosis of cardiovascular diseases can aid in making decisions on lifestyle changes in high risk patients and in turn reduce the complications. This work intends to pinpoint the most relevant/risk factors of heart disease as well as predict the overall risk using logistic regression. A classification model is build in MATLAB using Logistic Regression and Regularized Logistic Regression to predict 10-year risk of future coronary heart disease in patients from the given classification data set.**

***Index Terms*—logistic regression, regularized logistic regression, coronary heart disease, prediction, gradient descent**

I. Introduction

Logistic Regression is a statistical tool used to solve classification problem. For a model if predictive value is categorical then it is referred as classification problem. For example, like if the match lost or win, customer buys a product or not, student passes or fails in particular test, logistic regression model is used. Each object in this model can be assigned a probability ranging from 0 to 1 with a sum of one. Logistic regression essentially adapts the linear regression formula to allow it to act as a classifier. When more than one variable parameters are used for prediction then, that model is called Multiple Logistic Regression, which is a statistical test used to used to determine the numerical relationship between such a set of variables and predict a single binary variable using one or more other variables. Regularized Logistic Regression reduces or shrinks the coefficients in the resulting regression which reduces the variance in the model which is done to avoid over fitting and improve the generalization performance. Here in this following problem we will be using Multiple Logistic Regression where we have 15 variable features. Each feature is a potential risk factor. There are both demographic, behavioural, and medical risk factors.

A. Logistic Regression Model

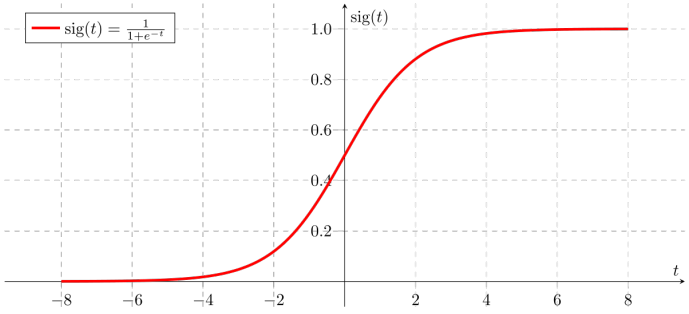
Let our hypothesis function be denoted as $h_\theta(x)$. The logistic function or the sigmoid function is denoted as

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

The cost function of Logistic regression is denoted as

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[-y^i log(h_\theta(x^i) - (1-y^i) log(1-h_\theta(x^i)]$$

As g(z) tends towards 1 as z→∞ and when g(z) tends towards 0, z→ −∞. h(x), is always bounded between 0 and 1.



Plot of g(z)

B. Regularized Logistic Regression

Regularization is used to train models that generalize better on unseen data, by preventing the algorithm from over-fitting the training dataset. A cost function is added to avoid over-fitting in the training dataset and improve generalization. The cost function contains the squared prediction errors, the squared sum of the magnitudes of all the coefficients in the model and tuning parameter that adjusts how large a penalty there will be. The cost function of Regularised Logistic regression is denoted as

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

II. METHODS

A. Data Pre-Processing

Data preprocessing is an integral step as the quality of data and the useful information that can be derived from it is directly affected by the nature of the input data. The most common step involved in data preprocessing is replacing Null / NAN values. Data Imputation can be done using a variety of methods like dropping the rows or columns that contain null values or by replacing them with either the mean/median values. The type of data imputation is highly specific to the

dataset and the problem in hand but the ultimate goal is to reduce significant information loss. For the given dataset, the missing values are replaced by the mean values.

B. Data Normalization

Data Normalization is done to scale the data to a smaller range. Normalization is required when different features in the dataset are of different scale which leads to poor data models when normalization is not performed. For the given dataset, the values in a particular column (features) are replaced using the given formula:

$$D_{new} = \frac{D_{old} - \mu}{\sigma}$$

where μ- mean of the feature and σ- standard deviation of the feature

C. Logistic Regression Implementation

Logistic Regression hypothesis function is defined as:

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

where,

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

Similarly, cost function in logistic regression is given as:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} [-y^i log(h_\theta(x^i) - (1 - y^i) log(1 - h_\theta(x^i)]$$

and the gradient of the cost is a vector of the same length as θ where the j^{th} element (for j = 0, 1, . . . , n) is defined as:

$$\frac{\delta J(\theta)}{\delta \theta_j} = \frac{1}{m} \sum_{i=1}^{m} h_\theta(x^i - y^i)(x_j)^i$$

The initial gradient value is set to random using the rand function in MATLAB which gives better accuracy than fixing the value to preset. The number of iterations are varied so as to asses the performance and accuracy of the model. The model generates 15x1 values of the gradient which is then used against the input dataset using a custom prediction function which uses the sigmoid function to generate binary decision boundary. Accuracy is calculated by comparing the obtained results with the given output data.

D. Regularized Logistic Regression Implementation

For the implementation of Regularized Logistic Regression, the gradient is changed as follows:

$$\frac{\delta J(w)}{\delta w_i} = \frac{1}{m} \sum_{i=1}^{m} h_\theta(x^j - y^j)(x_i)^j + \lambda w_i$$

The regularization term will heavily penalize large w_i. The effect will be less on smaller w_i's. As such, the growth of w is controlled. The h(x) obtained with these controlled parameters will be more generalizable.Larger value λ of will make w_i shrink closer to 0, which might lead to underfitting.$\lambda = 0$, will have no regularization effect. Accuracy and model validation is done similarly by changing the number of iterations and for different λ values.

Finally to change the θ values the following formula is used in the gradient descent approach:

$$\theta_{new} = \theta_{old} - \alpha * \frac{\delta J(\theta)}{\delta \theta}$$

E. Gradient descent Method

Gradient descent is an iterative first-order optimisation algorithm used to find a local minimum/maximum of a given function. This method is commonly used in machine learning and deep learning to minimise a cost/loss function. There are two specific requirements. A function has to be: Differentiable Convex Gradient Descent Algorithm iteratively calculates the next point using gradient at the current position, then scales it by a learning rate and subtracts obtained value from the current position. It subtracts the value because we want to minimise the function (to maximise it would be adding). This process mathematically shown as:

There's an important parameter which scales the gradient and thus controls the step size. In machine learning, it is called learning rate and have a strong influence on performance.

1) The smaller learning rate the longer gradient descent converges, or may reach maximum iteration before reaching the optimum point.

2) If learning rate is too big the algorithm may not converge to the optimal point or even to diverge completely. Gradient Descent method's steps are:

Step1-choose a starting point (initialisation)

Step2-calculate gradient at this point.

Step3-make a scaled step in the opposite direction to the gradient (objective: minimise).

Step4-repeat points 2 and 3 until one of the criteria is met: maximum number of iterations reached step size is smaller than the tolerance.

III. RESULTS

Logistic Regression Results Table

S.NO	ITERATIONS	α	ACCURACY (%)
1	1000	0.1	65.66
		0.01	63.10
		0.001	58.28
2	5,000	0.1	71.71
		0.01	64.90
		0.001	55.16
3	10,000	0.1	75.78
		0.01	67.08
		0.001	64.75

Regularized Logistic Regression Results Table

S.NO	ITERATIONS	α	λ	ACCURACY (%)
1	1000	0.1	1	65.37
			10	63.86
			100	61.78
		0.01	1	62.06
			10	63.01
			100	61.97
		0.001	1	57.33
			10	58.56
			100	58.28
2	5,000	0.1	1	70.29
			10	64.52
			100	61.78
		0.01	1	65.85
			10	63.58
			100	61.68
		0.001	1	58.94
			10	59.04
			100	56.48
3	10,000	0.1	1	72.09
			10	64.52
			100	61.78
		0.01	1	65.75
			10	64.52
			100	61.78
		0.001	1	61.49
			10	63.01
			100	62.72

IV. Results and Conclusion

- In Logistic regression as the number of iteration goes on increasing the Accuracy is also increasing. within the particular iteration as learning rate is decreasing accuracy is also going down.
- In Regularized Logistic regression also, as the number of iteration goes on increasing the Accuracy is also increasing. within the particular iteration as learning rate is decreasing accuracy is also going down.
- Compared to Regularized Logistic regression, Logistic regression is performed well on given data as we can see in result table.

Appendix

Step 1: Reading the data from dataG2 xlsx file

```
clc; clear; close all;
data=readtable("DataG2.xlsx");

data=table2array(data);
dataset=[];
```

Step 2: Replacing NaN value with the mean value of that column

```
for j=1:(size(data,2)-1)
    tmp=data(:,j);
    average=nanmean(tmp);
    x=tmp;
    x(isnan(tmp))=average;
    dataset=[dataset,x];
end
clear tmp j average;
```

Step 3: Normalization of the data

```
[m,n]=size(dataset);
average=mean(dataset,1);
stand_dev=std(dataset,1);
for j=1:n
dataset(:,j)=(dataset(:,j)-average(1,j))/stand_dev(1,j);
end
```

Step 4: Calling the function Logistic Regression

```
ip=dataset;
op=data(:,end);

init_theta=rand(size(ip,2),1);
iteration=10000;
alpha=0.001;
theta_1=Logi_Regression(ip,op,init_theta,iteration,alpha);
predicted_1=predict(ip,theta_1);
fprintf("Accuracy is: %.2f", mean(double(predicted_1==op)) * 100);
```

```
Accuracy is: 63.10
```

Step 5: Calling the function Regularized Logistic Regression

```
theta_2=Regul_Logi_Regression(ip,op,init_theta,iteration,alpha,10);
predicted_2=predict(ip,theta_2);
fprintf("Accuracy is: %.2f %%", mean(double(predicted_2==op)) * 100);
```

```
Accuracy is: 63.01 %
```

Function for logistic regression with gradient descent approach

```
function theta_new = Logi_Regression(X, Y, theta_old,maxIter,alpha)
    [nSamples, nFeature] = size(X);
    theta_new = theta_old;
    for j = 1:maxIter
        temp = zeros(nFeature,1);
        for k = 1:nSamples
            temp = temp + (1 / (1 + exp(-(X(k,:) * theta_new)))- Y(k)) * X(k,:)';
        end
        temp=temp/nSamples;
       theta_new =theta_new - alpha* temp;
    end
end
```

Function for
Regularized logistic regression with gradient descent approach

```
function theta_new = Regul_Logi_Regression(X, Y, theta_old,maxIter,alpha,lambda)
    [nSamples, nFeature] = size(X);
    theta_new = theta_old;
    for j = 1:maxIter
        temp = zeros(nFeature,1);
        for k = 1:nSamples
            temp = temp + (1 / (1 + exp(-(X(k,:) * theta_new)))- Y(k)) * X(k,:)';
        end
        for iter = 1:nFeature
            temp1=temp(iter,1)+lambda*theta_new(iter,1);
            temp1=temp1/nSamples;
            theta_new(iter,1) =theta_new(iter,1) - alpha* temp1;
            temp1=0;
        end
    end
end
```

Sigmoid function

```
function out=sigmoid(X,theta_new)
out=1/(1+exp(-sum))
end
```

Function for Prediction

```
function res=predict(X,theta_new)
[nSamples, nFeature] = size(X);
res = zeros(nSamples,1);
   for i = 1:nSamples
       sigm = 1 / (1 + exp(-(X(i,:) * theta_new)));
       if sigm >= 0.5
            res(i) = 1;
        else
            res(i) = 0;
        end
   end
end
```