A Study of Logistic Regression And Its Optimization Techniques Using Octave

Annapoorani Anantharaman
CSE, Jeppiaar Engineering College, India

***Abstract* ---** *A classification problem produces a binary output even when the input values are real numbers. Linear regression cannot be used to solve classification problems. Instead, logistic regression is used. Logistic regression is a supervised learning algorithm which estimates the probability of an outcome for the input given. Logistic regression is useful in many real world problems in many fields and its performance can be improved by some optimization techniques.*

This paper describes logistic regression and its various optimization techniques along with the performance metrics by using the study of graduate school admissions.

Keywords *- Logistic regression, optimisation techniques, Newton Raphson method, BFGS, L-BFGS, Gradient Descent, Conjugate Gradient Descent*

Literary survey:

I. INTRODUCTION

Logistic regression is a supervised learning algorithm. In supervised learning, the inputs are provided with their class labels and the output is determined by using the suitable algorithm. Logistic regression is a classification algorithm which determines the probability of occurrence of an event by using a model. The case study of admission scores as field inputs x1 and x2 and the corresponding result y which predicts if the student is accepted or rejected is used to demonstrate logistic regression. In this paper, let 1 represent success and 0 represent failure. The programming language used is octave.

II. LOGISTIC REGRESSION

Logistic regression can be used to solve a classification problem. The fundamental steps involve structuring data, finding the model, parameter fitting and interpretation of the output. It ensures the generated number is always between 0 and 1.

A. Hypothesis function

Let us consider the binary classification, the ouput is classified into two labels. The hypothesis function[1] is used to predict the value of output y for the given input fields x1,x2….xn where n is the number of columns and m depicts the number of rows in the input data.

$$h_\Theta(x) = g(\Theta^T x) = 1/(1 + e^{-(\Theta Tx)}) \qquad (1)$$

where $h_\Theta(x)$ is the hypothesis function which gives the predicted value of the given input x and x is a matrix of all the input field values. This can also be represented as

$$h_\Theta(x) = g(z) = 1/(1+e^{-z}) \qquad (2)$$

This is the sigmoid function or the logistic regression function.

The range of the hypothesis function is $0 \leq h(X) \leq 1$. The probability that the output y is 1 given x parameterised by Θ is given by

$$h_\Theta(x) = P(y=1 \mid x; \Theta) \qquad (3)$$

So probability y = 0 or y = 1 can be given by

$$P(y=0|x; \Theta) + P(y=1|x; \Theta) = 1 \qquad (4)$$

Or $\quad P(y=0|x; \Theta) = 1 - P(y=1|x; \Theta)$

This is written as $P(y|\ x;\theta)=h\theta(x)^y \,.\, (1-h\theta(x))^{1-y}$

This is the maximum likelihood function.[3]

B. Decision boundary

The decision boundary[1] is given by the sigmoid curve which is a curve drawn based on the hypothesis function with g(z) in the y axis and z along the x axis. The curve intersects the y axis at g(z) = 0.5 and z =0.

The function varies with the values of input variables to produce an output value. The prediction can be made as 1 or 0 based on the threshold value used to classify the function.

Predict y = 1 if g(z) >= 0.5, then z>=0

Therefore, $\Theta^T x >= 0$

Predict y =0 if g(z) < 0.5, then z<0

Therefore, $\Theta^T x < 0$

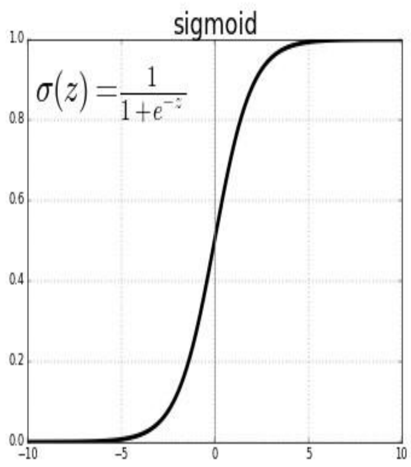

Fig.1 describes the sigmoid function as a function of the hypothesis function in the x axis and the z values in the y axis

C. Cost function

The cost function[7] should ideally be minimum. It is the difference in values of predicted output and the true output values. The cost function is represented by J(Θ). A cost function should be chosen which is convex and ideally has only one local optima, the lowest point in the curve.

In logistic regression, cost function is denoted as

$$\mathrm{Cost}(h_\Theta(x),y) = -\log(h_\Theta(x)) \quad \text{if } y=1$$

$$= -\log(1-h_\Theta(x)) \quad \text{if } y=0$$

It is seen that if $h_\Theta(x) = 1$ and $y = 1$, cost = 0 but as $h_\Theta(x)$ tends to 0, cost tends to infinity. To avoid this problem, a compressed form is used.

$$\mathrm{Cost}(h_\Theta(x),y) = -[y.\log(h_\Theta(x)) + (1-y).\log(1-h_\Theta(x))] \quad (5)$$

The cost function can be given by[1]

$$J(\Theta) = (1/m)* \sum \mathrm{Cost}(h_\Theta(x),y) \quad (6)$$

Therefore, the cost function is,

$$J(\Theta) = -(1/m)* \sum [\, y^i.\log(h_\Theta(x^i)) + (1-y^i).\log(1-h_\Theta(x^i))] \quad (7)$$

D. Fitting function

A fitting function[3] can be drawn by selecting the appropriate fitting parameter theta (Θ). The values of the output vary with the value of Θ. The theta values should be chosen such that optimization is achieved.

There are many optimization techniques. Some are:

- Gradient Descent
- Newton Raphson Method
- BFGS
- L-BFGS
- Conjugate Gradient Descent

E. Multiclass classification:

Logistic regression can be used to classify the outputs into more than just two classes. It can be used to classify into multiple labels using multiclass classification algorithms. To do this, the one-vs-all classification[1] is used. This is also called one versus rest algorithm.

$$h_\Theta^{(i)}(x) = P(y=i \mid x; \Theta)$$

The hypothesis function is calculated by assuming there are only two classes at each stage. One class is treated as one input and all the other classes are assumed to be the other class. Now, binary classification is performed for each individual class. This also used in neural networks in multiclass classification using logistic regression.

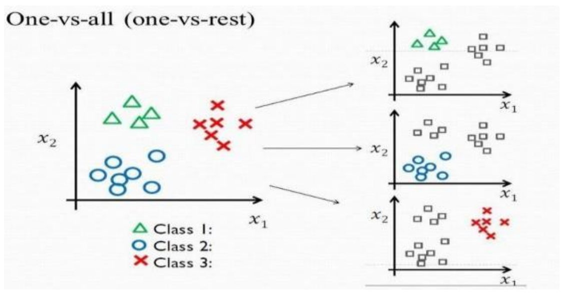

Fig 2.depicts one-vs-all classification algorithm in which each class is considered separately and the logistic regression is applied like binary classification

III. GRADIENT DESCENT

The most basic and vastly used optimisation technique to minimise the cost function is *Gradient Descent.*[4] It is an iterative optimisation algorithm to find the minimum of a function. To find the local minimum using gradient descent, steps proportional to the negative of the gradient of the function at the current point are taken.

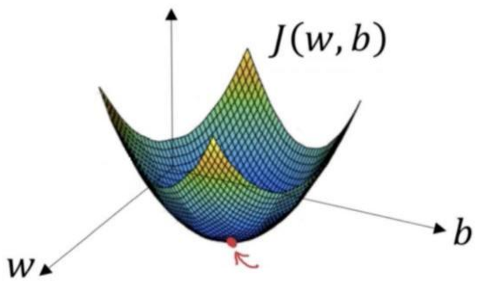

Fig 3. The gradient descent is depicted by the graph where the point in the bottom is the local minima

Gradient descent is used to compute the local minima of the given model such that it converges at that point.

The gradient descent is computed by choosing the appropriate values of theta , the fitting parameter.

Algorithm:

```
Repeat {
        $\Theta_j = \Theta_j - \alpha * \frac{\partial J(\theta)}{\partial \Theta j}$        (8)
} until convergence
```

Where $\frac{\partial J(\theta)}{\partial \Theta j} = (1/m) * \sum (h_\Theta(x^i) - y^i) * x_j^i$

The gradient descent algorithm[1] is best used in order to minimise the cost function by choosing appropriate theta values.

The disadvantage of this algorithm lies in choosing alpha or the learning rate value.

IV. NEWTON RAPHSON METHOD

The maximum likelihood function[3] is given by

$$P(y|\, x;\theta) = h_\Theta(x)^y \,.\, (1-h_\Theta(x))^{1-y}$$

We ideally need to maximise the right hand side in order to obtain the concave curve. The cumulative likelihood is found by multiplying all the likelihood values together.

$$L(\theta) = \prod i=1\, np(y_i|x_i;\theta) \qquad (9)$$

The product of all values produces a very small decimal number. To avoid this, the log likelihood[4] is taken.

$$\ell(\theta) = \log L(\theta) \qquad (10)$$

$$\ell(\theta) = \sum_{i=1} ny_i \log(h_\Theta(x_i)) + (1-y_i)\log(1-h_\Theta(x_i)) \qquad (11)$$

Newton and Raphson[7] discovered an iterative method to find the roots of a polynomial.

$x_{n+1} = x_n - [f(x_n) / f'(x_n)]$ (12) $y_{n+1} = f(x_{n+1})$ (13)

If $y_{n+1} - y_n \approx 0$ convergence is reached.

Else update point (x_n, y_n) and repeat the above steps until converged.

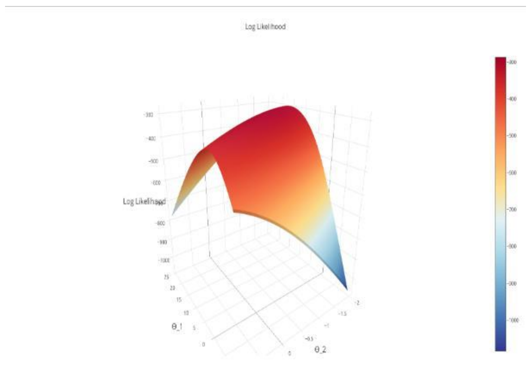

Fig 4. gives the log likelihood with theta1 and theta2 in the x and y axes and the log likelihood value in z axis

To maximize the log likelihood function, we need to find the partial derivatives of ℓ(θ), and set them equal to 0, and solve for θ1 and θ2 to find thecritical pointof the partials. This critical point will be the max of our log-likelihood. This is the concave point.

On substituting $f(x_n)$ with the gradient, $\nabla\ell(\theta)$, from the newton equation into the maximum log likelihood function. The Hessian is then calculated to obtain the denominator. It is a second order differential equation and is represented as a square matrix of order n.

Thus, the equation becomes

$$\Theta n+1 = \theta n + Hl(\Theta\hat{})-1 \,.\nabla\ell(\theta) \qquad (14)$$

The Hessian $H_{l(\Theta\hat{})}$ is given by a 2x2 matrix of values:[4]

$$\sum i = 1nh\theta(xi)(1 - h\theta(xi))\theta 1\theta 1$$

$$\sum i = 1nh\theta(xi)(1 - h\theta(xi))\theta 1$$

$$\sum i = 1nh\theta(xi)(1 - h\theta(xi))\theta 1$$

$$\sum i = 1nh\theta(xi)(1 - h\theta(xi))$$

These are the values (0,0),(0,1),(1,0) and (1,1) in the hessian matrix.

Thus optimization is achieved.

V. BFGS METHOD

The BFGS[5] method is used in optimization and is a type of quasi-Newton method. In quasi newton method, the Hessian matrix need not be computed. The Hessian matrix may be expensive to compute everytime. Instead, it is a generalisation of the secant method. The Hessian is updated by analysing successive gradient vectors instead.

The BFGS method is Broyden-Fletcher-GoldfarbShannon method. It is an iterative method which is used for solving non-linear optimization problems. It is a hill-climbing algorithm which seeks a

stationary point. The necessary condition is that the gradient should be zero.

From an initial guess x_0 and an approximate Hessian matrix, the following is repeated until it converges to the solution:

1. Obtain a direction p_k by solving $B_k p_{k\,=}$ delta $f(x_k)$.
2. Find $\alpha_k$in the direction found in the first step, so $\alpha_{k=}$argmin $f(x_k + \alpha p_k)$.
3. Set $s_k = \alpha_k p_k$ and update $x_{k+1} = x_k + s_k$.
4. Find y_k= difference of differentiation in $f(x_{k+1})$ and $f(x_k)$.
5. Bk+1 = Bk + ykykT/ ykTsk - Bksk (Bksk)T/skTBksk

Where $f_k$denotes the objective function to be minimized. The first step of the algorithm is carried out using the inverse of the matrix , which can be obtained efficiently by applying theSherman–Morrison formulato the step 5 of the algorithm, giving

Bk+1-1 = (I - skykT/ ykTsk) Bk-1(I- ykskT /ykTsk) + skskT/ ykTsk

This can be computed efficiently without temporary matrices, recognizing that is symmetric, and that and are scalars.

The BFGS method is a more optimal method but is difficult to implement. Thus predefined libraries are available for its implementation.

VI. L-BFGS METHOD

LBFGS[5] is Limited BFGS method which approximates the BFGS method within limited memory. This is a parameter estimation technique in machine learning and is used to minimise f(x) value. L-BFGS uses an estimation to the inverseHessian matrixto steer its search through variable space, but where BFGS stores a dense n x n approximation to the inverse Hessian (*n* being the number of variables in the problem), L-BFGS stores only a few vectors that represent the approximation implicitly. Due to its resulting linear memory requirement, the LBFGS method is particularly well suited for optimization problems with a large number of variables. Instead of the inverse Hessian H_k, LBFGS maintains a history of the past *m* updates of the position x and gradient $\nabla f(x)$, where generally the history size *m* can be small. These updates are used to implicitly do operations requiring the $H_k$vector product.

The algorithm starts with an initial estimate of the optimal value x_o , and proceeds iteratively to refine that estimate with a sequence of better estimates.

Sk= xk+1 – xk (15)

Yk= gk+1 – gk (16)

Where $g_k$is the gradient descent of $f(x_k)$.

Let $\rho_k = 1/y^T s_k$

Hk+1 = (I - ρkskykT)Hk(I - ρkykskT) + ρkskskT (17)

The maximisation of the problem is determined by

$z = z + s_i(\alpha_i - \beta_i)$ (18)

Where $z = H_k g_k$

For minimisation problems, -z is taken.

The L-BFGS method is more complex and thus optimisation libraries are available for its implementation. [7]

VII. CONJUGATE GRADIENT DESCENT

This is one of the most optimal methods for solving linear equations in an iterative manner[6]. It is useful for sparse equations to solve equations of the form

$Ax = b$ (19)

Where x is unknown vector, b is a known vector and A is a sparse positive definite matrix. If it is dense, it is made sparse by back substitution.

Let the sub span space be D_i composed of $\{r_1, r_2 \ldots\ldots r_{i-1}\}$

$r_i^{(T)} r_j = 0$ (20)

where r is the residual. Each residual is a previous combination of the residuals.

βi =riTri/ ri-1Tri-1 (21) let $d_0 = r_0 = b - Ax$
αi = riTri/ diTAdi (22)

Then,

xi+1 = xi + αidi(23) ri+1 = ri- αiAdi (24) βi+1 = ri+1Tri+1/ riTri(25) di+1 = ri+1 + βi+1 di (26)

The above steps are repeated over and over iteratively until a minima is obtained. This is one of the best optimisation methods but it is complex. It uses the principle of steepest hill descent.

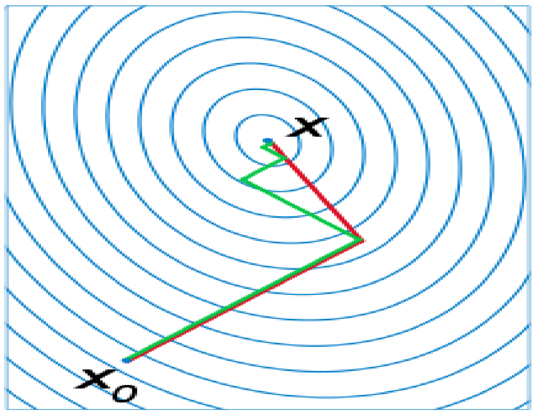

Fig 5. Gives the conjugate gradient descent

VIII. STUDY: ADMISSIONS OF GRADUATES BASED ON EXAM-SCORE USING OCTAVE

A dataset[1]with the input fields as the marks of the entrance exam scores and the UG scores are given represented by X1 and X2 respectively and the output or y feature is a set of known admitted or not admitted values represented by 1 or 0. A sample of the data set is given below:

TABLE 1
Dataset for UG admission

X1	X2	y
34.62366	78.02469	0.00000
30.28671	43.89500	0.00000
35.84741	72.90220	0.00000
60.18260	86.30855	1.00000
79.03274	75.34438	1.00000
45.08328	56.31637	0.00000
61.10666	96.51143	1.00000
75.02475	46.55401	1.00000
76.09879	87.42057	1.00000

The programming language used is Octave and the editor is Ocave GNU CLI. It can also be implemented in matlab.

The dataset can be plotted as a graph to obtain the following:

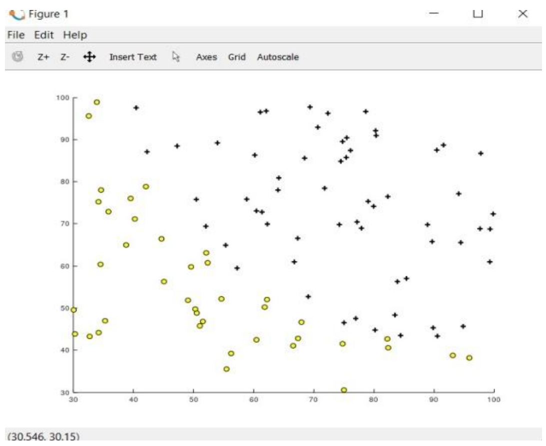

Fig 6. Shows the distribution of the data set points X and y where the black points represent y=1 and yellow are y=0 in octave GNU

Where the + denotes admitted or y=1 and the yellow dots represent not admitted or y=0.

The cost function can be calculated in Octave GNU by the below code

J=(1/(2*m))* sum((theta(1)*X(:,1)+ theta(2)*X(:,2) - y).^2) (27)

The theta values can be updated by gradient descent or by the other optimisation methods. The other methods are implemented as a library in Octave as fminunc.[2] % Set options for fminunc options = optimset('GradObj', 'on', 'MaxIter', 400); % This function will return theta and the cost

[theta, cost] = ... fminunc(@(t)(costFunction(t, X, y)), initial theta, options)

It is seen that the libraries are easier and compute the theta values *0.68 seconds* faster than gradient descent.

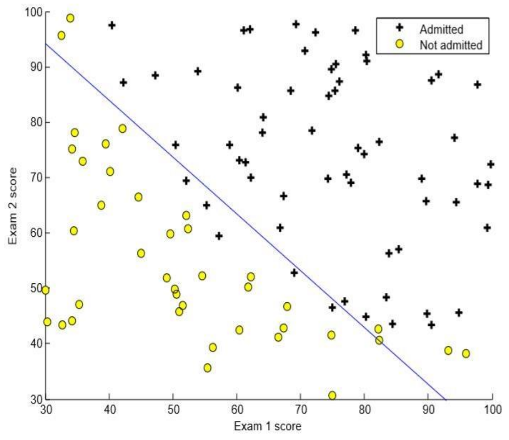

Fig 7. Gives the plot of the values after logistic regression is performed in Octave GNU

IX. CONCLUSION

Thus, Logistic regression is a very useful technique to predict the values of a given input and classify it as binary values. The accuracy of prediction is an important factor. It depends on the actual values and the predicted values. The parameter values can be determined by various techniques and optimization methods which help in increasing the parameters. The efficiency of logistic regression and optimisation techniques can be seen from our example above. Logistic regression is used in many real world problems and is a quintessential machine learning technique for solving classification problems.

REFERENCE

[1] Machine Learning course – Andrew Ng
[2] Revisit of Logistic Regression: Efficient Optimization and Kernel Extension- Takumi Kobayashi, Nobuyuki Otsu, Kenji Watanabe
[3] Logistic Regression — Gradient Descent Optimization – Abhinav Mazumdar
[4] A study of Classification Problems using Logistic Regression – Arka Mukherjee
[5] Wikipedia – BFGS
[6] An Introduction to the Conjugate Gradient Method Without the Agonizing Pain Edition by Jonathan Richard Shewchuk
[7] PadhAi Labs – Deep Learning Course