Arka Mukherjee Follow
Computer Science and Engineering Undergrad student, interested in Machine Learning
Apr 3 · 7 min read

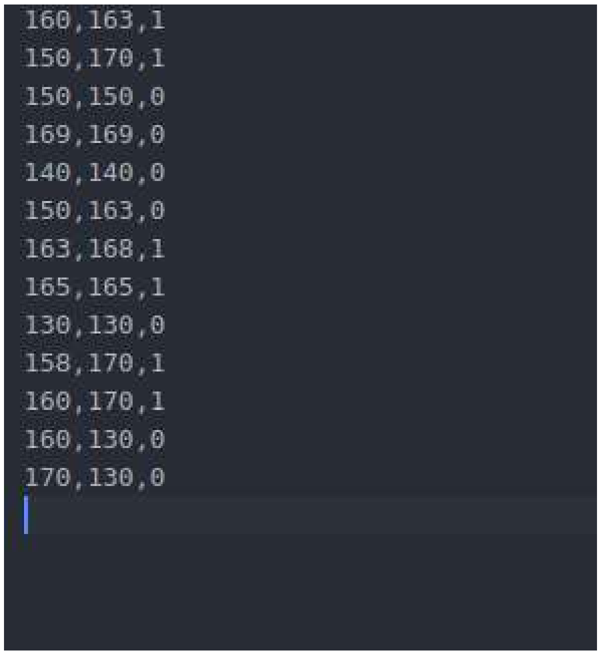
A study of Classification Problems using Logistic Regression and an insight to the admissions problem

Abstract:

In our world, many of the commonly encountered problems are classification problems. We are often confused between definite values or rigid choices of things. In this article, we will discuss about an algorithm used to solve simple classification problems effectively using Machine Learning. Also, we will analyze a hypothetical Binary Class problem involving Grad-School outcomes based on the Entrance Exam Marks and the Undergrad Marks.

Introduction:

Supervised Learning is a machine learning technique in which we associate our inputs with our targets in the given dataset. We already have a definite intuition regarding our final output. We broadly have two types of Supervised Learning problems, “Regression” and “Classification”. We will be discussing about Classification problems in this article. A Classification problem is a problem in which we separate our input data into discrete categories. By discrete, we mean separate classes.
Example: Think of a hypothetical dataset based of Grad School admissions having three columns. The first column being the Entrance Exams Score(**Input X1**), the second column being the UG Exams(**Input X2**), and the third column being the result, admit/reject(**Target Y**). A segment of the dataset used in this article is shown below.



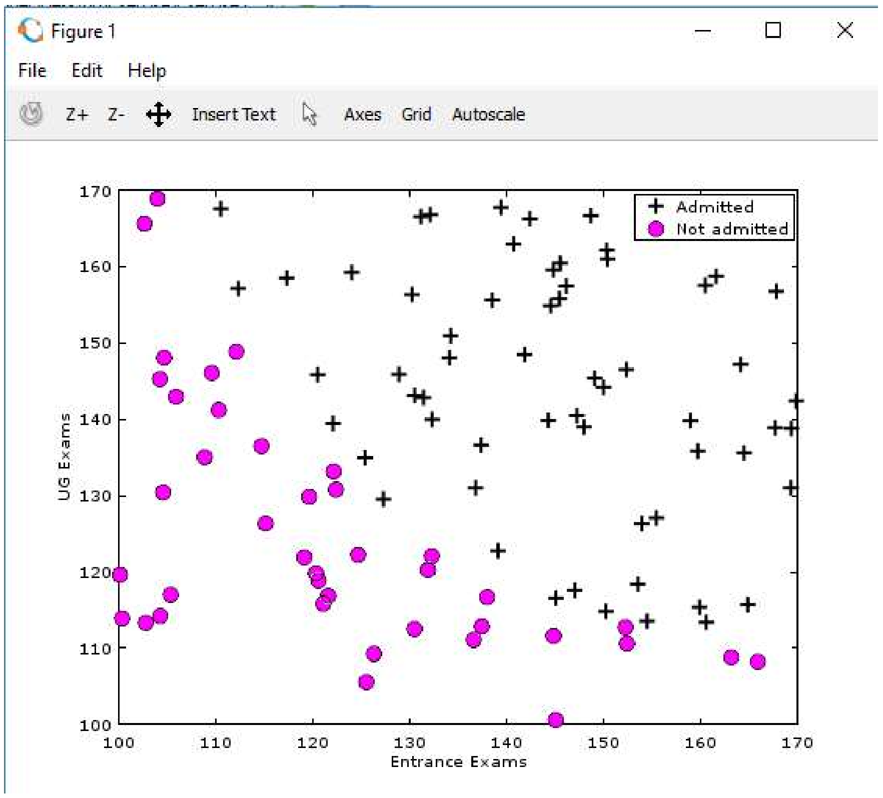
```
160,163,1
150,170,1
150,150,0
169,169,0
140,140,0
150,163,0
163,168,1
165,165,1
130,130,0
158,170,1
160,170,1
160,130,0
170,130,0
```

Entrance Exam Score, UG Score, Outcome(Binary)

We define our target class Y as a binary class. Thus we define the set as, **Y={0,1}** where the set elements are:

0 implies Reject(Negative Class),
1 implies Accept(Positive Class)

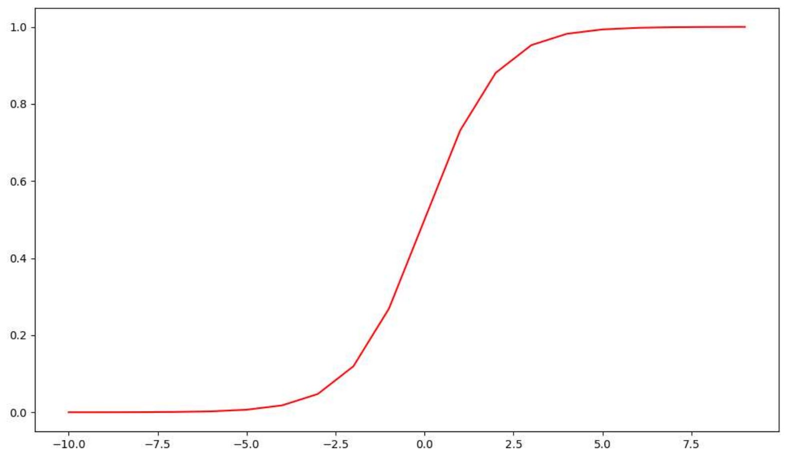
Thus, this is a Classification problem. We can divide the dataset into two categories based on the outcome of the admission process. The algorithm which generally resolves a dataset into discrete categories is defined as a *classifier*. In this example, the dependent variable is *dichotomous*. In such cases, we prefer Logistic Regression. We can plot a scatter-plot of our Dataset in MATLAB/Octave and observe the following graph:



Scatter plot of our dataset

Activation Function and Mapping Function involved:

First of all, consider the function that maps our output and input and is hence used for computing predictions while calculating the Squared Error Function. Let us call that function a Mapping Function or a Hypothesis Function, **h(X)**. The term "Hypothesis" was coined for various historical reasons. Also, the term "Logistic" is a synonym for Sigmoid. Thus, we use a Sigmoid activation function with an "S Shaped curve" in this algorithm. A curve generated via our programmed function is depicted below.



Sigmoid Activation Function

Mathematically, we define our function as:

$$S(t) = \frac{1}{1 + e^{-t}}.$$

The Sigmoid function also has several other mathematical properties. For example, we can represent the derivative of the function by the function itself.

$$S'(t) = S(t)(1 - S(t)).$$

This, the Sigmoid/Logistic function simplifies the mathematics involved during optimization and is an ideal choice for a small scale classification problem. Using the Sigmoid function, we can effectively limit our range so that it represents probability effectively. We can represent our Hypothesis/Mapping function mathematically as:

$$h_\theta(x) = g(\theta^T x)$$

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

This is essentially a very basic linear cost function, just that it's coupled with a sigmoid function.

Our Hypothesis function gives us the numerical value for the probability any given event happening. We can define the range of the mapping function as: **0≤h(X)≤1**.

Cost Function Intuition:

A cost function is a mathematical function to estimate how well the hypothesis suits the input set and target set. Graphically, we know that a better cost function will fit our given data better. It is generally denoted as a function of our parameters. In this case, we refer to it as **J(θ)**.
In this algorithm, we use a **logarithmic cost function** which is derived from the principle of *Maximum Likelihood Estimation(M.S.E)* to ensure that the function we get as an output is a **Convex function.**

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

A brief intuition is to think that the logarithm involved in the cost function roughly counter-acts the **exp** involved.

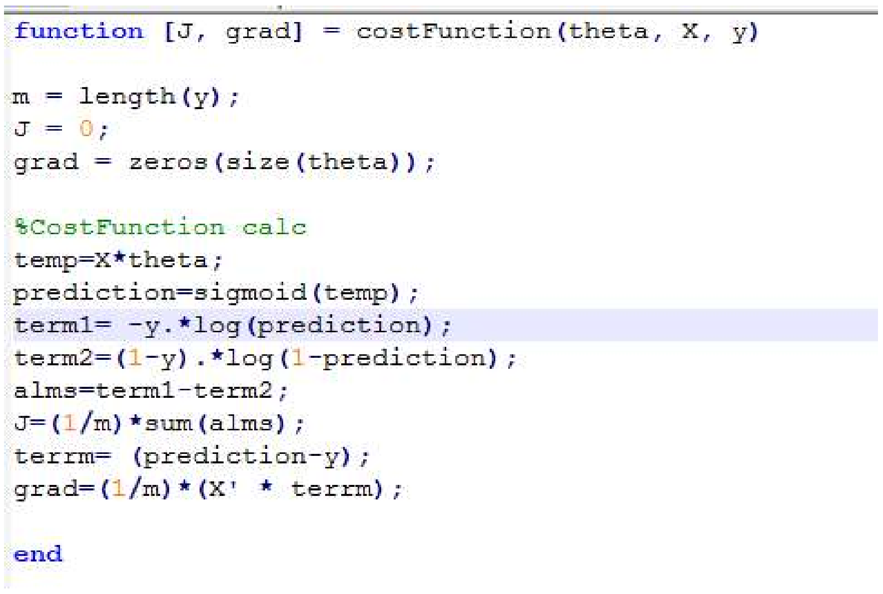
As Christopher M. Bishop stated in his work,

> *"When we assume that our positive and negative training samples come from two different Gaussian clusters (different location but same covariance) then we can develop a perfect classifier."*

Optimization is easier in a Convex Function as it is easier to reach a minima in a curve and get an optimal value for the parameter, θ. However, we observe that we get the same Grad function as a Linear Regression method after we differentiate J(θ).

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

However, the Grad value is not the same thing as we possess a different mapping function for Linear and Logistic Regression.



```
function [J, grad] = costFunction(theta, X, y)

m = length(y);
J = 0;
grad = zeros(size(theta));

%CostFunction calc
temp=X*theta;
prediction=sigmoid(temp);
term1= -y.*log(prediction);
term2=(1-y).*log(1-prediction);
alms=term1-term2;
J=(1/m)*sum(alms);
terrm= (prediction-y);
grad=(1/m)*(X' * terrm);

end
```

Prototype code for a Cost Function

Therefore using the parameter vector θ and the Cost Function J(θ), we can apply any given optimization algorithm to implement a simple decision boundary.

Optimizing the parameters:

Now if we want to fit the parameters(In this case, θ), we would want to apply minimization(Optimization) algorithms.
We could take simple mini-steps*(**limit(θ)->o**)* to our given minima which is usually called **Gradient Descent**, or go for advanced optimization algorithms such as ***BFGS*** or ***Limited-Memory BFGS***.

The advanced optimization methods are a part of the ***Quasi Newton*** methods where we compute an optimized value of the **Hessian Matrix**. L-BFGS is very similar to BFGS, other than the fact that it has better memory optimization and is a more scalable algorithm. Also when juxtaposed with Gradient Descent, the Advanced Algorithms are less prone to trial and error conditions as there is no need to pick an arbitrary value of *alpha(*Learning Rate) to get more optimal results. The mathematical notation for the Hessian Matrix is shown below:

$$\mathbf{H} = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \dfrac{\partial^2 f}{\partial x_2\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 f}{\partial x_n\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

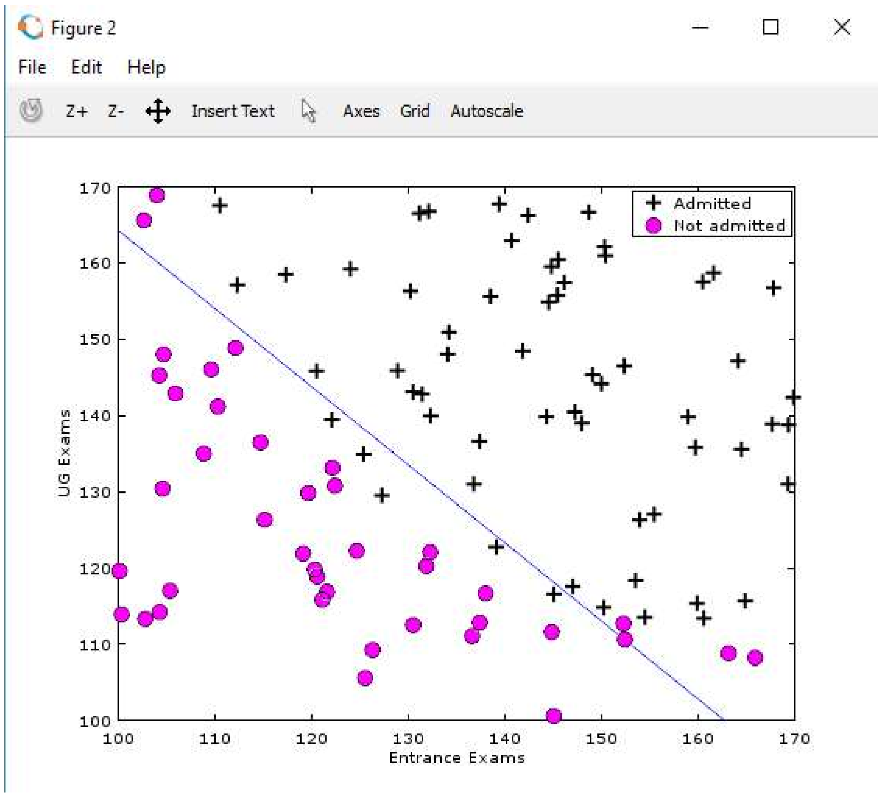
or, component-wise:

$$\mathbf{H}_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}.$$

Generally, we apply advanced optimization via libraries. One particular library in C++ used for the applying advanced optimization algorithms is the Eigen C++ Library.

However, different libraries might have different time complexities. Not choosing a library wisely might end up degrading our algorithm's overall performance and having an abysmal effect on our overall program.

Thus, we can plot a decision boundary after effectively minimizing our parameters using an optimization method. Thus, we use MATLAB's generic *"fminunc"* to apply advanced optimization. After applying our function with θ and the Grad value as the parameters, we can plot our Decision Boundary as follows:



Decision Boundary separating the data. In this case, a linear boundary fits better.

The concept of Overfitting a curve:

Overfitting is basically a condition in which our learned hypothesis somehow manages to fit all the points in the dataset and generates little or no error but has a major flaw. Our algorithm **fails to predict** the discrete class of **new testing examples.**
The solution to this issue:
We can penalize certain parameters(θ)to fix this. We add a regulation parameter(λ) to our optimization algorithm which can optimize a curve. This technique is called *Regularization*. Thus, we modify our cost function and Optimization algorithms to penalize the parameters and obtain an optimal fit.

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

Goodness of a Fit:

In **Goodness of Fit(G.O.F)**, we basically check whether a given model is correctly specified and how well it fits the statistical observations. There are usually two commonly used methods to estimate the Goodness of Fit. One test is the famous Pearson Chi-Square Test, and the other test is the *Hosmer and Lemeshow Test*. Straightforward **Pearson Chi-Square** test is a simpler technique and hence we'll discuss the intuition of the test.

$$\frac{(\text{observed} - \text{expected})^2}{\text{expected}}$$

The Chi-Square(χ2) has a good deal of mathematical accuracy for computing statistical parameters from a given dataset. Also, this test is asymptotically true. Its main role is to calculate frequencies for every observation in a dataset.
Thus, we can calculate the statistical p-value by comparing the value to the value of the distribution.
We could also use a variety of other tests such as the **Stukel Test or Information Matrix** Test to measure the fit of Logistic Regression effectively.

Since this is my first article on Machine Learning, I discussed about an elementary concept in the field. In the next article, I will write about Alpha Beta Pruning using a U64 Bitboard based Chess-Engine.

References:

1. W.S Lee and B Liu,"Learning with positive and unlabeled examples using weighted logistic regression"-ICML, 2003—vvvvw.aaai.org

2. DC Liu and J Nocedal, "On the limited memory BFGS method for large scale optimization"-Mathematical programming, 1989—Springer

3. MA Babyak, "What you see may not be what you get: a brief, nontechnical introduction to overfitting in regression-type models",-Psychosomatic medicine, 2004—journals.lww.com