Machine and Me Learning

Aprameyan Desikan

August 2021

Introduction

The task of making sure that Moore's law remains satisfied may have put humanity at a metaphorical crisis. Thankfully, it continues to satisfy with the advent of artificial intelligence. One of the foremost of problems to solve in the world has been to bring order to chaos, and in the scientific industry, it is through advancements in medicine and technology and our understanding of the world and universe. Machine learning has played a vital role in making this possible.
So, the main questions to answer are, what is Machine Learning, why we need it and where it is needed.

What?

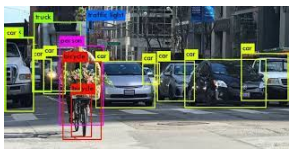
It is the ability to make a computer accomplish a task without explicitly coding for the said task. While doing so, we first train it and provide it with the "experience". Then, we provide it with the "task" at hand, and finally check its "performance" by basing it on how it predicts the task for a new set of data.

Why?

This question is answered by seeing where ML has been hiding in plain sight.

(a) Google Search

(b) Youtube recommendations



(c) Image recognition

ML applications

Where?

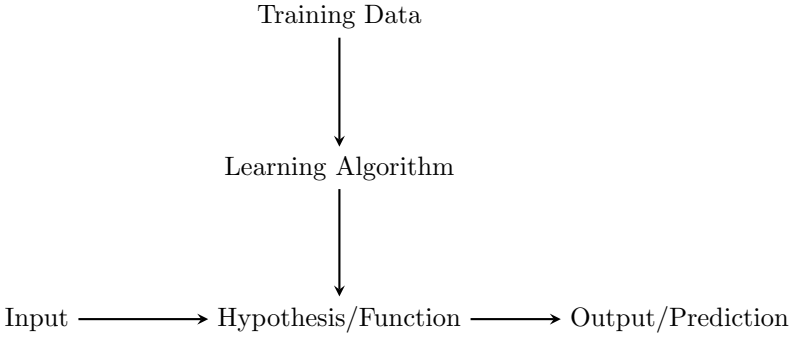
Now this is the question we will mainly be looking at in detail. The topics we will be looking at are part of what we call Supervised Learning, meaning that we are aware of what we want to predict and how we know that the input and output is related.

- Linear and Polynomial Regression
- Classification using Logistic Regression

We will also look into the details regarding the optimisation of the algorithms used.

1 Linear Regression

The basic idea is as follows;



1.1 Linear Regression in 1 variable

In the example that we will be considering, the hypothesis that we will be ending up with is a linear function which will fit the data we have. For eg.; let us consider the prices of books and how they vary with the number of pages. The number of pages, which is our input is what we call a feature. There can be multiple features, but for starters we will stick with one. Since, we are dealing with linear regression, we want to fit the given data in a line. i.e.

$$h(x) = \theta_0 + \theta_1 x$$

θ_0 and θ_1 are called the parameters, which are what we want to find. In order to find them we first determine the Cost function. The idea is to check how much our output values from our line equation varies with the desired output and minimise it. This variation is written in the form of our cost function, $J(\theta_0, \theta_1)$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h(x^i) - y^i)^2$$

'm' is the number of training examples that we provide and (x^i, y^i) is the i'th training input and output. This in a way conveys the total error (squared of course) between our line and the given data. It is for sure that the line will not perfectly pass through each and every point of the training examples, so it makes sense to minimise this error and not reducing it to 0. So to reduce the cost function we use Gradient Descent.

Gradient Descent

This is the method on which our machine learning algorithm would be working.
The way it works is as follows;

- Pick a random θ_0 and θ_1, usually taken to be 0
- Goal is to keep changing the values of the parameters until we reduce the cost function to its least value

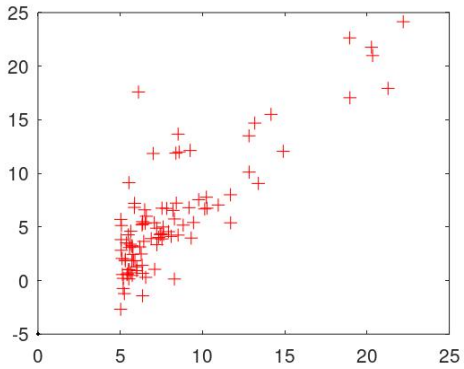
$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_j}$$

The gradient can also be found out to be;

$$\frac{\partial J(\theta_0, \theta_1)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h(x^i) - y^i) x^i$$

It is important to know that we do not use the equal symbol as the true meaning of := is to assign and update the value using its previous value. α is the learning rate, and its value is crucial in the working of the algorithm. This algorithm works because, the theta value would decrease when the theta as at a place of positive slope and would increase when it is at a place of negative slope, and would in the end reach the minimum. α tells us how fast we want to reach the minimum, although using a large α might also cause the algorithm to overshoot the minimum and end up diverging. Making it too small will cause it to converge very slowly. Now, to the code to understand how ML is implemented. Taking the example of the population and profit for a company in various cities.

```
data=load('ex1data1.txt');
x=data(:,1); % only one feature,i.e. population
y=data(:,2); % Profits
plot(x,y,'r+','MarkerSize',7);
```



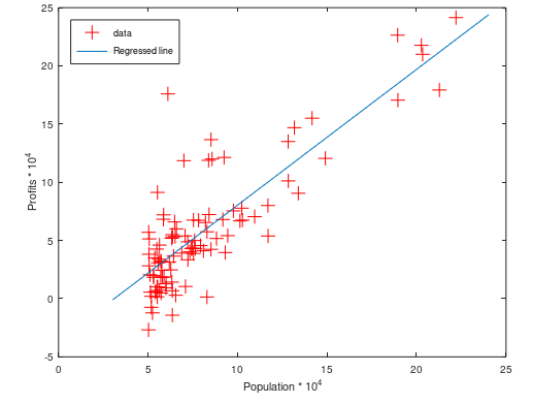
The cost function;

```
function J = computeCost(X, y, theta);
m = length(y); % number of training examples
x=X*theta; % 'x' now stores h(x)
a=x-y; % matrix containing errors
J=sum(a.^2)/(2*m); % cost function
```

Finally, the gradient descent algorithm;

```
function [theta] = gradientDescent(X, y, theta, alpha, iterations)
    m = length(y); % number of training examples
    for iter = 1:iterations
        x=X*theta; % h(x) stored
        delta=X'*(x-y) % Gradients of J wrt each theta element
        theta=theta-alpha*delta/m; % The updates for each theta
    end;
end;
```

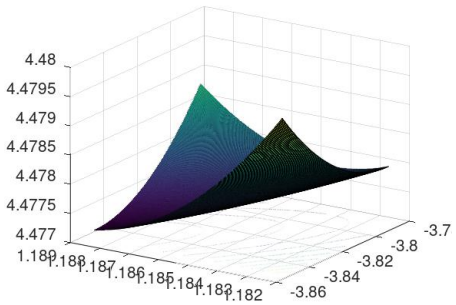
One can also check whether the cost function is decreasing with every iteration. Now, with the values of θ_0 and θ_1 we can plot the line and get;



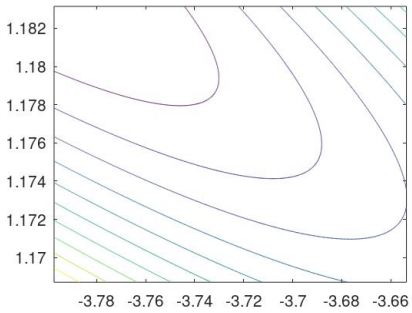
While using the same logic in python;

```
    def f0(t0,t1,x,y):
        k=0
        sum=0
        while k<5:
            sum=sum+(t0+t1*x[k]-y[k])
            k=k+1
    return sum/5
    def f1(t0,t1,x,y):
    k=0
    sum=0
    while k<5:
        sum=sum+(t0+t1*x[k]-y[k])*x[k]
        k=k+1
    return sum/5
n=0
x=[]
y=[]
while n<5:
  i=float(input("Enter input"))
  x.append(i)
  o=float(input("Enter output"))
  y.append(o)
  n=n+1
t0=0
t1=0
a=input("enter learning rate")
a=float(a)
k=0
temp0=0
temp1=0
while(abs(f0(t0,t1,x,y))>0.000001):
  temp0=t0-a*f0(t0,t1,x,y)
  temp1=t1-a*f1(t0,t1,x,y)
  t0=temp0
  t1=temp1
  k=k+1
print("minima occurs at ",t0,t1)
```

One might be curious to see how the cost function also varies the parameters.



(d) Surface plot



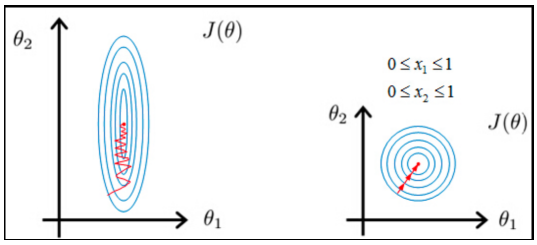
(e) Contour plot

Cost function, surface and contour plots

Similarly, one can also use polynomial regression and multiple features in order to better represent the plot as it is highly biased to assume that a given data would vary linearly.
In order to do so, we just introduce more features in the form of squares or cubes or higher degrees of the same feature.The same code can be used although graphing it might be out of scope for more than 1 feature.

Feature Scaling and Mean Normalisation

These ideas are to make your gradient function algorithm work much better.
Feature Scaling: To bring all your variable/features to the same scale.
eg. say $0 < x^{(1)} < 100$ and $0 < x^{(2)} < 10$. To bring them down to the same scale, we divide $x^{(1)}$ by 100 and $x^{(2)}$ by 10 to bring them to the range of 0 to 1. Preferably we expect x^i to be in the range of -1 to 1.



Source: Stack Overflow

Mean Normalisation: To bring your feature values to have zero mean.
This is done along with feature scaling in order to make gradient descent be done faster. We basically substitute $x^i = \frac{(x^i - \mu^i)}{range/std.dev.}$

Normal Equation

Apart from the gradient descent method we have an algebraic way of using just matrices to find the set of parameters, i.e. theta values. So, with just two matrices, one containing the set of inputs and the other with the outputs, we calculate the optimised theta values.
Consider, we have 'n' number of features with a total of 'm' examples. We have the design matrix 'X' (m×(n+1)) and the ouputs 'y' (m×1) ;

$$X = \begin{bmatrix} 1 & x_1^{(1)} & . & x_j^{(1)} & . & . & x_n^{(1)} \\ 1 & x_1^{(2)} & . & x_j^{(2)} & . & . & x_n^{(2)} \\ 1 & . & . & . & . & . & . \\ 1 & . & . & . & . & . & . \\ 1 & x_1^{(m)} & . & x_j^{(m)} & . & . & x_n^{(m)} \end{bmatrix} \qquad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ . \\ . \\ y_{(m)} \end{bmatrix}$$

$$\theta = (X^T X)^{-1} X^T y$$

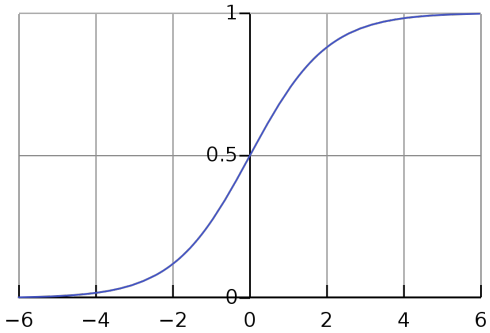
Note: There is no need for feature scaling in this method and just needs to solve the matrix multiplication. However, for large number of training examples, computing might take time. Gradient descent on the other hand can compute more efficiently, but with its comebacks over choosing the precise alpha and having to iterate multiple times.

1.2 Logistic Regression

The idea is the same but the task varies. Our outputs are discreet and the intention is to classify them into these discreet sets. For eg. the classification of emails as spam and not spam or items in a grocery list as vegetables, fruits, etc.
Logistic regression is used in this case as linear regression does not work very well. For starters, let's take a binary classification, meaning, the output is either a 0 or a 1. Now we will also be taking a hypothesis which would provide us with values between 0 and 1, and the logic being to consider any output above 0.5 as 1 and anything below as 0. The way we achieve this is by using the sigmoidal function to out initial linear hypothesis (starting with two feature).
After getting the optimised hypothesis, we graph the line which separates the inputs according to the outputs they give,i.e. 0 or 1. The line that separates the data is called the decision boundary.
Sigmoidal Function: It is a function that takes in any value and gives a value between 0 and 1.

$$h(x) := \frac{1}{1+e^{-h(x)}}$$

Note that, when h(x)≥0,y=1 and h(x)<0,y=0;



Source: Wikipedia

Cost Function: The use of sum of error squared leads to a lot of local minima, which is not favourable in finding the most optimised set of theta values. Therefore another function is used.

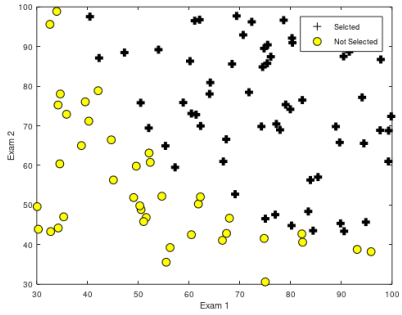
$$J(\theta_0, \theta_1) = -\frac{1}{m}\sum_{i=1}^{m}[y^i(log(h(x^i)) + (1-y^i)log(1-h(x^i))]$$

$$h(x) = sigmoid(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

Alternate Optimisation

Apart from gradient descent, the use of a function in Octave can help speed-en up the process of finding the optimised parameters. We shall see them by understanding the code.
Taking the example of taking the marks in two exams as your features and classifying whether the person is selected or not

```
data=load('ex2data1.txt');
X=[ones(length(data),1) data(:,1:2)];
y=data(:,3);
function plotData(X, y)
    figure; hold on;
    pos=find(y==1);neg=find(y==0);
    plot(X(pos, 2), X(pos, 3), 'k+','LineWidth', 2,'MarkerSize', 5);
    plot(X(neg, 2), X(neg, 3), 'ko', 'MarkerFaceColor', 'y','MarkerSize', 5);
    xlabel('Exam 1');
    ylabel('Exam 2');
end;
plotData(X,y);
```



The sigmoid function;

```
function g = sigmoid(z)
    g=z;
    z=1./(1+exp(-z));
end;
```

The cost function with the gradient;

```
function [J, grad] = costFunction(theta, X, y)
    m = length(y); % number of training examples
    x=X*theta; % storing h(x)
    a=sigmoid(x);
    J=(-y'*log(a)-(1-y)'*log(1-a))/m; % Calulating cost function
    grad=X'*(a-y)/m; % Calculating gradient simultaneously
end;
```

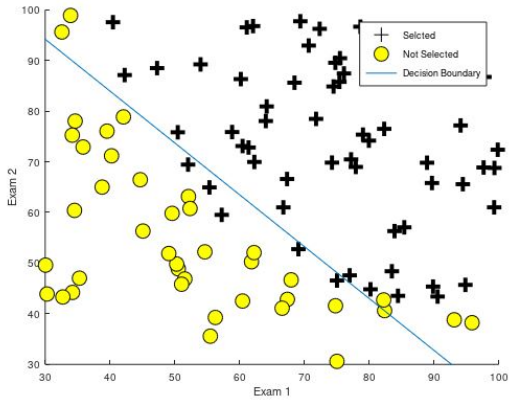
Now, the optimisation algorithm;

```
options = optimset('GradObj', 'on', 'MaxIter', 400);
[theta, cost] = fminunc(@(t)(costFunction(t, X, y)), initial_theta, options);
```

This algorithm is very simple to execute and finds the parameters as part of the matrix 'theta'. Now, we graph the decision boundary, which is basically the $h(x) = \theta_0 + \theta_1 x = 0$ line;

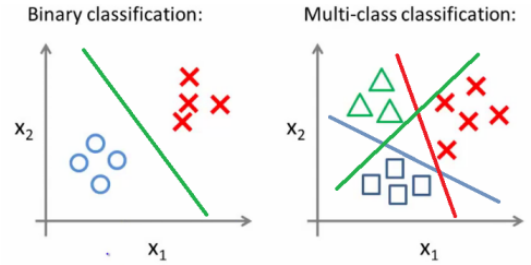
```
function plotDecisionBoundary(theta, X, y)
    plotData(X(:,2:3), y);
    hold on

    if size(X, 2) <= 3
        plot_x = [min(X(:,2))-2,  max(X(:,2))+2];
        plot_y = (-1./theta(3)).*(theta(2).*plot_x + theta(1)); % h(x)=0
        plot(plot_x, plot_y)
        legend('Selected', 'Not Selected', 'Decision Boundary')
    else %for non linear functions
        u = linspace(-1, 1.5, 50);
        v = linspace(-1, 1.5, 50);

        z = zeros(length(u), length(v));

```

```
            for i = 1:length(u)
                for j = 1:length(v)
                    z(i,j) = mapFeature(u(i), v(j))*theta;
                end
            end
            z = z'; % to transpose z before calling contour
        % Plot z = 0
        % Notice you need to specify the range [0, 0]
        contour(u, v, z, [0, 0], 'LineWidth', 2)
        end
    end % courtesy: Machine Learning course by Stanford (Coursera)
```



Multiclass Classification

In this case, we divide different set of classification and treat them as independent binary systems. i.e. If there are 'n' classes, we would have 'n' binary sets which we would use to compare simultaneously and take the value from h(x) which gives the maximum value;



Source:Medium

In the picture above, there are three lines which specify the decision boundary for each set of binary classifications we consider. The value which we take as part of our probability in terms of where to classify a new data, we should consider the hypothesis which gives the largest number.

1.3 Regularisation

Sometimes, in need for trying to fit every point into our hypothesis perfectly without any error, we might over-fit the data. This is not favourable because it loses the generality of being able to predict the classification of new data. So, in order to prevent over-fitting, we use regularisation, which is to reduce the weight of the features(i.e. the parameters). Regularisation is achieved by having a sum of the parameter squared in the cost function which would also be simultaneously minimised.Regularisation also helps in preventing the non inversibility of the XX^T matrix, need it occur.

Linear Regression

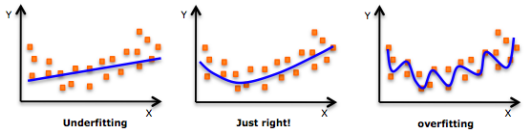
$$J(\theta) = \frac{1}{2m}[\sum_{i=1}^{m}(h(x^i) - y^i)^2 + \lambda \sum_{j=1}^{n} {\theta_j}^2]$$

Logistic Regression

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

Normal Equation

$$\theta = (XX^T + \lambda(I_m - E_{1,1}))^{-1}X^T y;\ E_{1,1} : \text{m}\times\text{m matrix with 1 in (1,1) position}$$

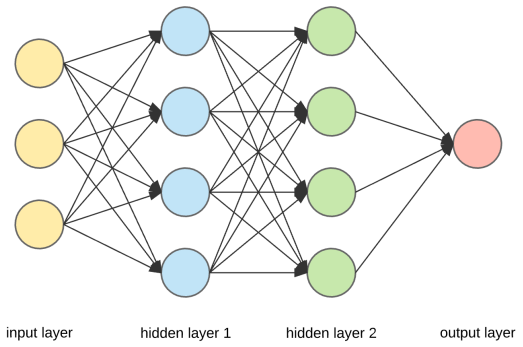


Source:Analytics Vidhya

Thus, we have looked at the major basic algorithms and methods of machine learning.

2 Neural Networks

The brain is most clearly the most intricate organ in the body, and it's function being one of the most important. Many of the inspirations and motivations of engineering has been to mimic the human body. The concepts of robots is well sought and the goal has been to make them more and more "human". The most difficult aspect of the human body has been and will continue to be the brain.



Source:Towards Data Science

The Neural Network that computer scientists came up with or "Artificial Neural Network" (ANN), works in a similar fashion to the actual neural network of the brain, it gets information, it learns, it predicts. But it's structure as an algorithm is also made similar to the neural network, with axons, nodes and neural connections. The neural network is useful as it keeps makes changes and corrections over each layer through which the information travels.