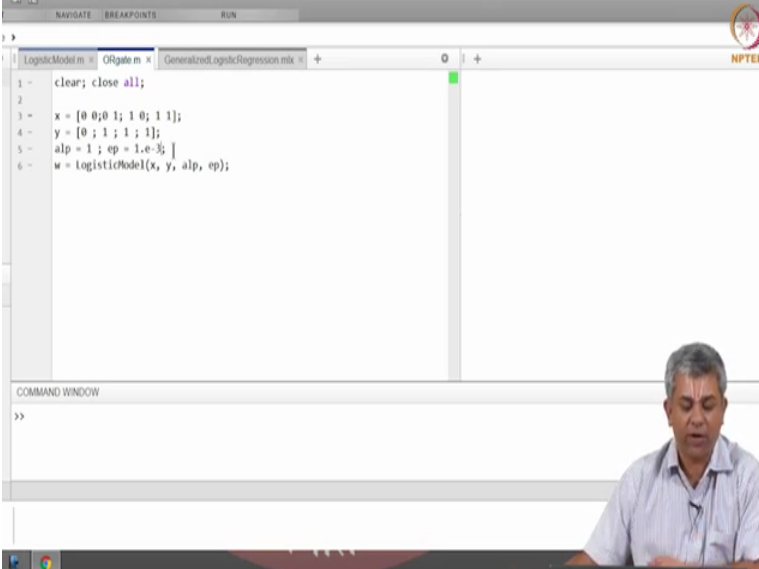


**Machine Learning for Engineering and Science Applications**  
**Professor Dr. Balaji Srinivasan**  
**Department of Mechanical Engineering**  
**Indian Institute of Technology, Madras**  
**Generalized Function for Logistic Regression**

(Refer Slide Time: 0:25)



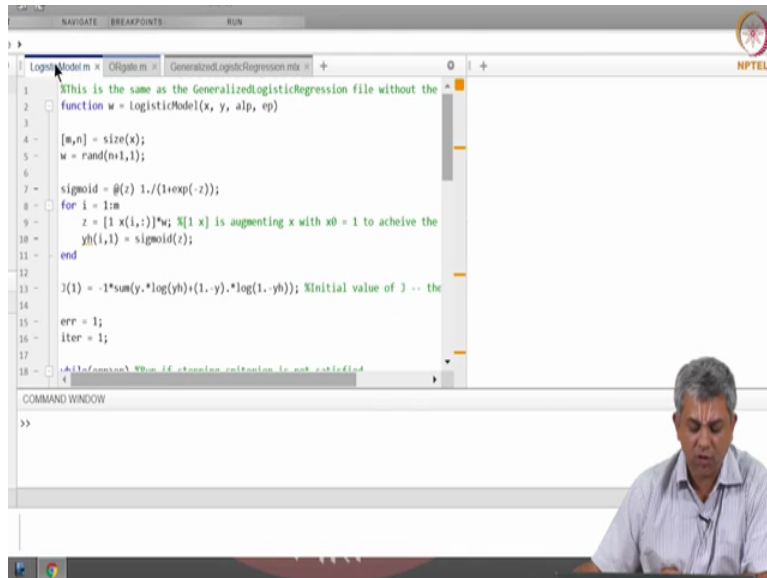
The image shows a MATLAB IDE window with the following code in the editor:

```
1 - clear; close all;
2
3 - x = [0 0; 0 1; 1 0; 1 1];
4 - y = [0 ; 1 ; 1 ; 1];
5 - alp = 1 ; ep = 1.e-3;
6 - w = logisticModel(x, y, alp, ep);
```

Below the editor is a command window with the prompt `>>`. The NPTEL logo is visible in the top right corner of the IDE window.

Welcome back. We will be doing a repeat of the linear regression example except with the logistic regression case and a slightly different example. So what I have written here is a very short code which is an OR gate. So notice the OR gate is now given as four different examples: 00, 01, 10, 11. And I have given the corresponding ground truths here,  $y$  here is simply the ground truth. For 00 it is 0; 01, 1; 10, 1 and 11 also 1. And like we did in the linear regression I have given a learning rate, I have given a reasonably high learning rate of  $w$  equal to 1,  $\alpha$  equal to 1. And epsilon, the stopping criteria, let us make it a little bit smaller, maybe let us make it  $10^{-3}$ , just to sort of see how the whole thing proceeds. This is obviously not good enough in general. So we will see how it works.

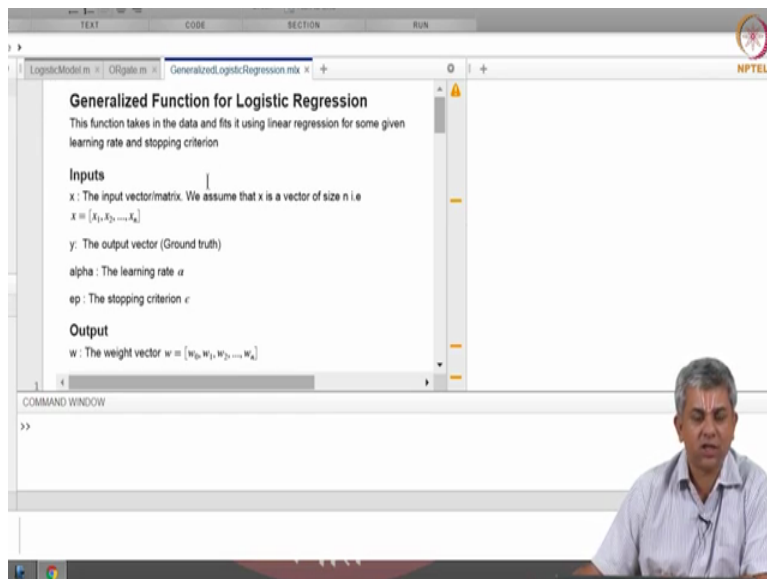
(Refer Slide Time: 1:16)



```
1 %This is the same as the GeneralizedLogisticRegression file without the
2 function w = logisticModel(x, y, alp, ep)
3
4 [m,n] = size(x);
5 w = rand(n+1,1);
6
7 sigmoid = @(z) 1./(1+exp(-z));
8 for i = 1:m
9     z = [1 x(i,:)]*w; % [1 x] is augmenting x with x0 = 1 to achieve the
10    yh(i,1) = sigmoid(z);
11 end
12
13 J(1) = -1*sum(y.*log(yh)+(1.-y).*log(1.-yh)); %Initial value of J -- the
14
15 err = 1;
16 iter = 1;
17
18 %while(err>ep) %When if stopping criterion is not satisfied
```

Now I will show you the code with the comments. But the one I will be running is here, the one without the comments.

(Refer Slide Time: 1:19)



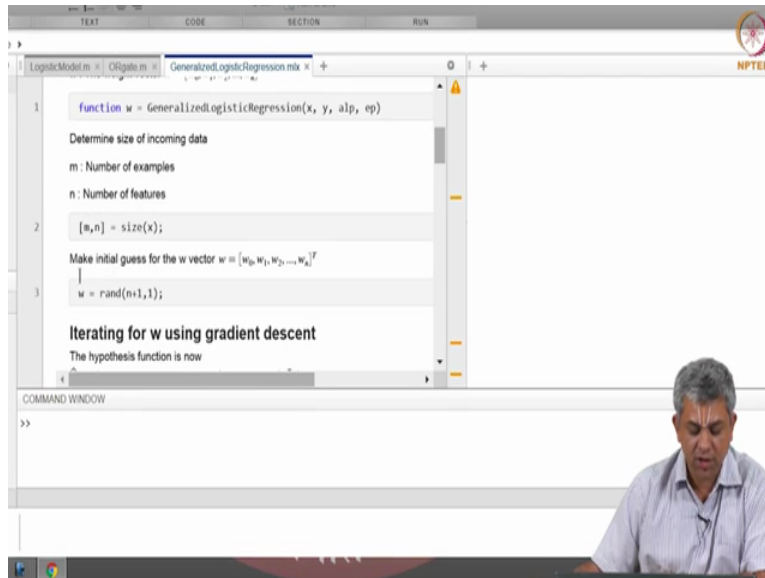
```
Generalized Function for Logistic Regression
This function takes in the data and fits it using linear regression for some given
learning rate and stopping criterion

Inputs
x: The input vector/matrix. We assume that x is a vector of size n i.e
x = [x1, x2, ..., xn]
y: The output vector (Ground truth)
alpha: The learning rate alpha
ep: The stopping criterion epsilon

Output
w: The weight vector w = [w0, w1, w2, ..., wn]
```

So the general logistic regression code you will notice this is there in the week 5 material on the NPTEL website. You can copy it from there; it is more or less identical to the linear regression code.

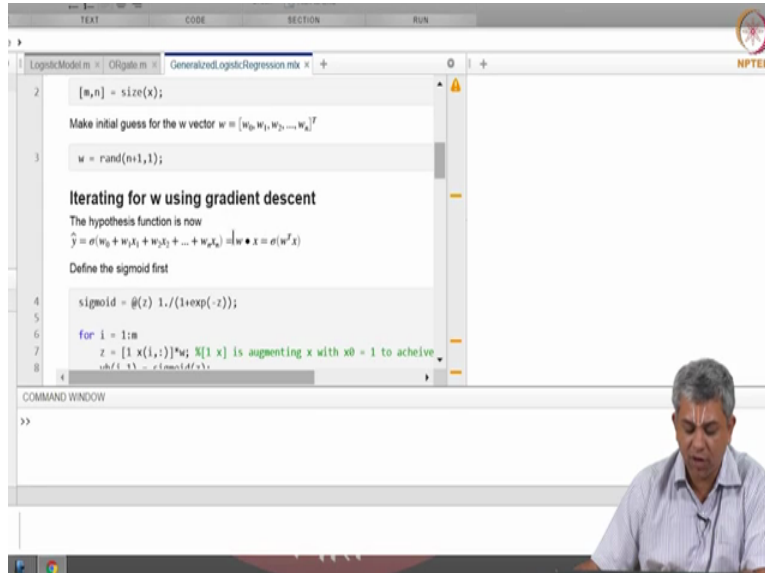
(Refer Slide Time: 1:32)



```
function w = GeneralizedLogisticRegression(x, y, alp, ep)
Determine size of incoming data
m : Number of examples
n : Number of features
[m,n] = size(x);
Make initial guess for the w vector w = [w_0, w_1, w_2, ..., w_n]^T
w = rand(n+1,1);
Iterating for w using gradient descent
The hypothesis function is now
```

So it is remarkably identical, you will see this term is the same, this term is the same.

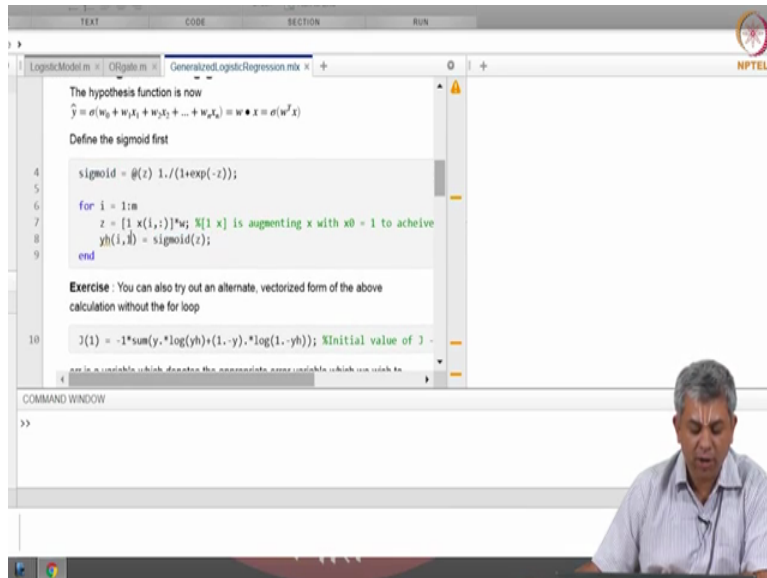
(Refer Slide Time: 1:36)



```
[m,n] = size(x);
Make initial guess for the w vector w = [w_0, w_1, w_2, ..., w_n]^T
w = rand(n+1,1);
Iterating for w using gradient descent
The hypothesis function is now
ŷ = σ(w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n) = w • x = σ(w^T x)
Define the sigmoid first
sigmoid = @(z) 1./(1+exp(-z));
for i = 1:m
    z = [1 x(1,:)]; % [1 x] is augmenting x with x_0 = 1 to achieve
    % z = [1 x_1 x_2 ... x_n]
```

The only place we start differing is here. Earlier in linear regression we had only this term,  $w \cdot x$  up to  $w \cdot x_n$  which was simply  $w$  dot  $x$ . There is a small typo here, please ignore that. So we have sigmoid of  $w$  dot  $x$  which is sitting here.

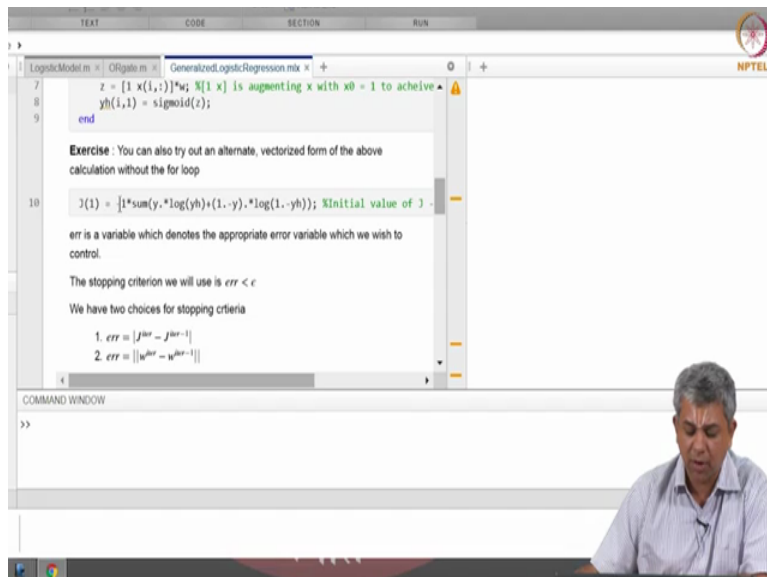
(Refer Slide Time: 1:52)



```
LogisticModel.m | Ofgate.m | Generalized Logistic Regression.m | +
The hypothesis function is now
 $\hat{y} = \sigma(w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n) = w \cdot x = \sigma(w^T x)$ 
Define the sigmoid first
4 sigmoid = @(z) 1./(1+exp(-z));
5
6 for i = 1:m
7     z = [1 x(i,:)]*w; % [1 x] is augmenting x with x0 = 1 to achieve
8     yh(i,1) = sigmoid(z);
9 end
Exercise : You can also try out an alternate, vectorized form of the above
calculation without the for loop
10 J(1) = -1*sum(y.*log(yh)+(1.-y).*log(1.-yh)); %Initial value of J
err is a variable which denotes the appropriate error variable which we wish to
COMMAND WINDOW
>>
```

I have defined an inline function for sigmoid and the only thing that changes from linear regression is instead of having just z which was the output for linear regression, you have sigmoid of z as the new hypothesis function.

(Refer Slide Time: 2:09)



```
LogisticModel.m | Ofgate.m | Generalized Logistic Regression.m | +
7     z = [1 x(i,:)]*w; % [1 x] is augmenting x with x0 = 1 to achieve
8     yh(i,1) = sigmoid(z);
9 end
Exercise : You can also try out an alternate, vectorized form of the above
calculation without the for loop
10 J(1) = -1*sum(y.*log(yh)+(1.-y).*log(1.-yh)); %Initial value of J
err is a variable which denotes the appropriate error variable which we wish to
control.
The stopping criterion we will use is  $err < \epsilon$ 
We have two choices for stopping criteria
1.  $err = |J^{iter} - J^{iter-1}|$ 
2.  $err = ||w^{iter} - w^{iter-1}||$ 
COMMAND WINDOW
>>
```

The other change is in calculating the loss function. The loss function is now the binary entropy loss function.

(Refer Slide Time: 2:22)

The screenshot shows a MATLAB editor window with the following content:

The gradient has (n+1) components. That is  $V_{w_j}$  is a (n+1)x1 vector

$$\frac{\partial J}{\partial w_0} = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})$$
$$\frac{\partial J}{\partial w_1} = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)}) x_1^{(i)}$$

...

$$\frac{\partial J}{\partial w_j} = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)}) x_j^{(i)}$$

...

$$\frac{\partial J}{\partial w_n} = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)}) x_n^{(i)}$$

16 `pred_err = yh - y;`

COMMAND WINDOW  
>>

Otherwise the error terms et cetera remain exactly the same. In fact even the gradient terms remain the same. You may remove, include or you may include or you may remove the 1 by m term.

(Refer Slide Time: 2:33)

The screenshot shows a MATLAB editor window with the following content:

$$\frac{\partial J}{\partial w_j} = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)}) x_j^{(i)}$$

...

$$\frac{\partial J}{\partial w_n} = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)}) x_n^{(i)}$$

16 `pred_err = yh - y;`  
17 `DJ(1,1) = sum(pred_err)/m;`  
18 `for j = 2:n+1`  
19 `DJ(j,1) = sum(pred_err.*x(:,j-1))/m;`  
20 `end`  
21

$$w_j = w_j - \alpha \frac{\partial J}{\partial w_j}$$

COMMAND WINDOW  
>>

I have chosen to include it just so that I could use the same code, this is just the (02:36) solution. If you wish, you can remove the m.

(Refer Slide Time: 2:42)

The image shows a MATLAB code editor window with the following code:

```
w = w - alp*D;

The hypothesis function is  $\hat{y} = w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n = w \cdot x = w^T x$ 

for i = 1:m
    z = [1 x(i,:)]*w; % [1 x] is augmenting x with x0 = 1 to ach
    %h(i,1) = sigmoid(z);
end
iter = iter + 1;

J = -sum((y./m*(y^2) + (1 - y^2)/m(1 - y^2)))

J(iter) = -1*sum(y.*log(yh)+(1-y).*log(1-yh));
err = abs(J(iter)-J(iter-1));
%err = norm(-1e-10);
```

The COMMAND WINDOW shows the prompt >>.

Otherwise the gradient term is calculated in exactly the same way and finally we calculate the loss once again based on the sigmoid.

(Refer Slide Time: 2:52)

The image shows a MATLAB code editor window with the following code:

```
clear; close all;
x = [0 0; 0 1; 1 0; 1 1];
y = [0 ; 1 ; 1 ; 1];
alp = 1 ; ep = 1.e-3;
w = logisticModel(x, y, alp, ep);
```

The COMMAND WINDOW shows the prompt >> ORgate.

Figure 1 shows two plots:

- The top plot shows the data points (blue circles) and the classifying line (red line) in the  $x_1$ - $x_2$  plane. The data points are at (0,0), (0,1), (1,0), and (1,1). The classifying line is a straight line with a negative slope.
- The bottom plot shows the cost function  $J$  versus iterations. The cost function starts at approximately 1.5 and decreases rapidly, reaching a value near 0.5 after about 100 iterations and continuing to decrease slowly towards 0.25 by 700 iterations.

So let us run this code for the OR gate and see what it gives. You will notice these four points. These are our original data points and the classifying line actually moves from an incorrect classification slowly up and technically speaking this is already correctly classified but we have set a certain stopping criteria so it will move till that stopping criteria is met. Notice also that the

loss function  $j$  is continuously decreasing. So please notice this. You will notice that as the loss function decreases you can hardly notice any change in the classifying line.

You will also see that the classifying line is not quite the one that we had given theoretically which had gone through 0.5. As I mentioned in the previous video this actually depends on what your initial conditions are. So if your initial conditions are different, you might get a slightly different final classifying line. The whole point of this exercise, so this is obviously a very simple exercise with the OR gate. You can try it with AND gate if you wish and you will get a slightly different classifying line.

So if you try with different initial conditions, you will get different classifying lines. The advantage of logistic regression is it will give you an answer. Of course, it will give you some local minima, it might not, it might be good or it might be bad. With four data points it turns out that it can be reasonably good with this. Now if I wanted to do the AND gate example, all I would need to do is change this. If I change this to 0, this to 0 and this to 1, play with this which I would recommend for you, please change all your data points and check what kind of gates it classified. In fact I would even recommend that you try and do the XOR gate which is simply this one change to 0 and see what it does.

You will see that in the case of the XOR gate that  $j$  actually saturates very early, it does not keep on decreasing and it gives an incorrect classification because as we had discussed in the XOR video, XOR is not linearly classifiable. So it is not linearly separable data. OR gate happens to be linearly separable. So this is a simple code. I would encourage you to play with it, look through the code and kind of compare this with the linear regression code and see what you can notice overall. Thank you.