ANDREW NG on LOGISTIC REGRESSION:

LECTURE 6.1: INTRODUCTION to LOGISTIC REGRESSION

↳ **logistic regression** - one of the most popular algorithms for classification problems

↳ **classification problems** include: spam detection, whether online transactions are fraudulent or not, and whether a tumor is cancerous or not.

↳ In all of these problems, the variable we are **trying to predict is y**, which can be thought of as 0 or 1, spam or not spam, etc.

↳ 0 = negative class, 1 = positive class => This is for **2-class problems, binary**. There can also be problems with multiple classes where y = {0, 1, 2, 3} and so on.

↳ Many classification problems COULD be done with **linear regression**, however outliers can greatly affect model predictions. Using linear regression for classification is generally not good. You could get lucky, but it rarely is very effective or accurate.

↳ Even though regression is in the logistic regression title, it is actually a classification algorithm.

↳ It is more useful because the positive and negative values are between 0 and 1 and do not have a definite threshold in the same way as linear (such as 0.5). It gives a measure of confidence in itself.

LECTURE 6.2: HYPOTHESIS REPRESENTATION

↳ hypothesis representation: we want $0 \leq \text{hypothesis}\Theta(x) \leq 1$

↳ we will use: $\text{hypothesis}\Theta(x) = g(\Theta^T(x))$ ⇦ g is sigmoid

↳ $g(z) = 1 / 1 + e^{-z}$ ⇦ Sigmoid (logistic) function **SO** $h\Theta(x) = 1 / 1 + e^{-\Theta t x}$ (plugged in $\Theta^T x$ for z)

↳ We must fit our parameters, Θ , to our data. Given a training set, must pick a value for Θ

Example: If $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$

$h_\theta(x) = 0.7$ $\quad y=1$

Tell patient that 70% chance of tumor being malignant

↳ $h\Theta(x)$ = estimated probability that y = 1 on input x

i.e. ⇨ **$h\Theta(x) = p(y=1|x; \Theta)$**

The probability that y = 1, given x, parameterized by Θ.

$$P(y=0|x;\theta) + P(y=1|x;\theta) = 1$$
$$P(y=0|x;\theta) = 1 - P(y=1|x;\theta)$$

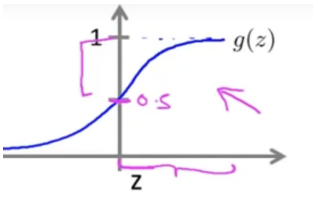
⇨ This shows how the probability of y=0 and the probability of y=1 must be equal to 1

⇨ second half moved right of =. Can now compute the probability that y=0 as well

---

LECTURE 6.3: DECISION BOUNDARY

↳ a closer look at what the logistic regression's hypothesis function is computing, particularly when we have more than one feature.



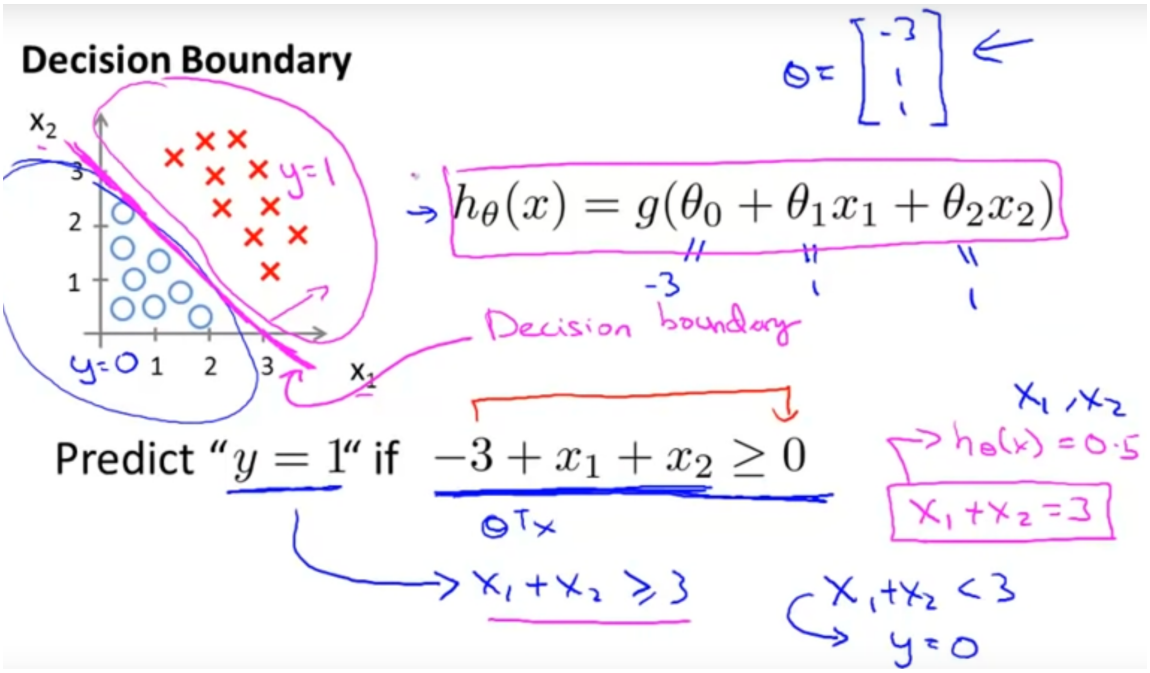
$g(z) \geq 0.5$ when $z \geq 0$

$h_\Theta(x) = g(\Theta^T x) \geq 0.5$ whenever $\Theta^T x \geq 0$

Suppose predict "$y = 1$" if $h_\theta(x) \geq 0.5$

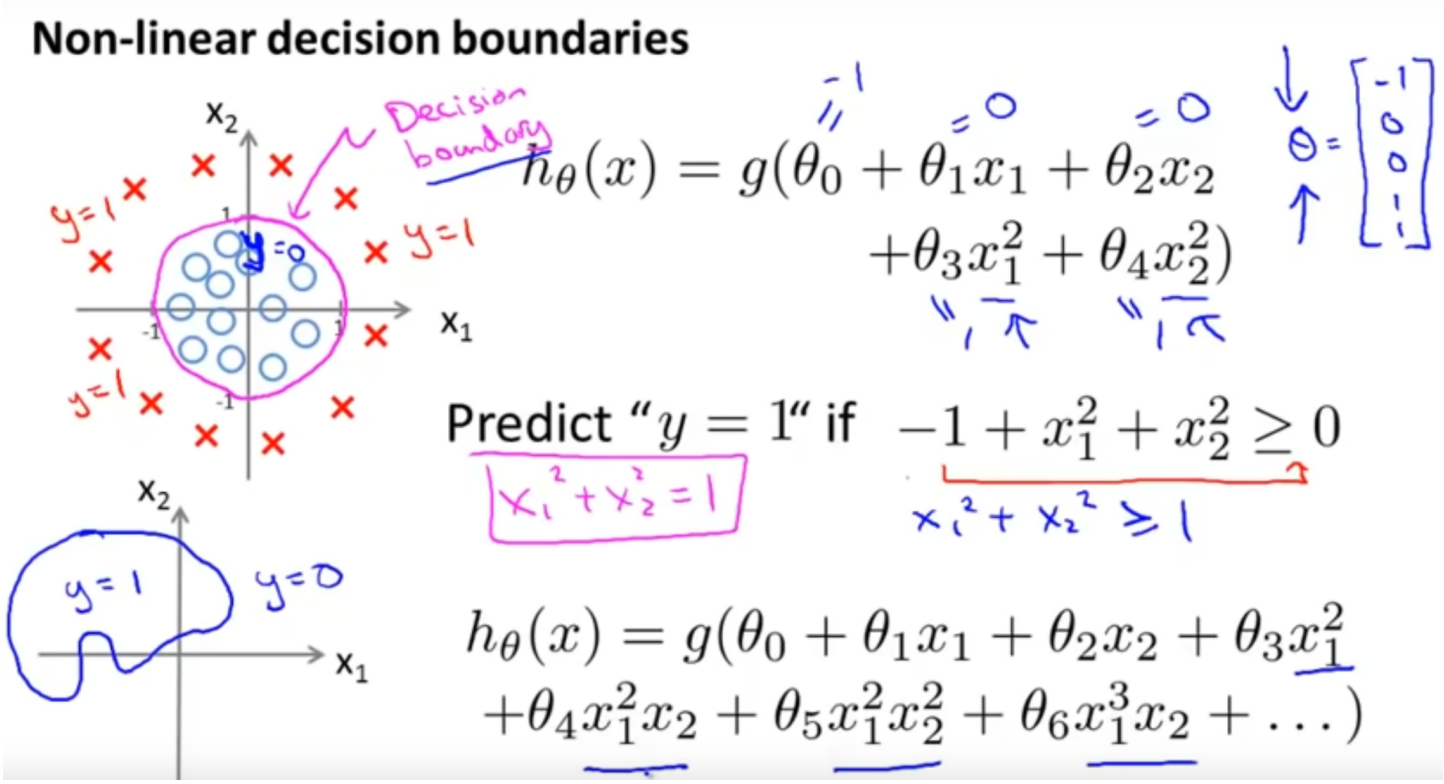
predict "$y = 0$" if $h_\theta(x) < 0.5$

The decision boundary is a property of the hypothesis function, not of the data:



NON-LINEAR DECISION BOUNDARIES: Using polynomial terms (x_1^2 and x_2^2)

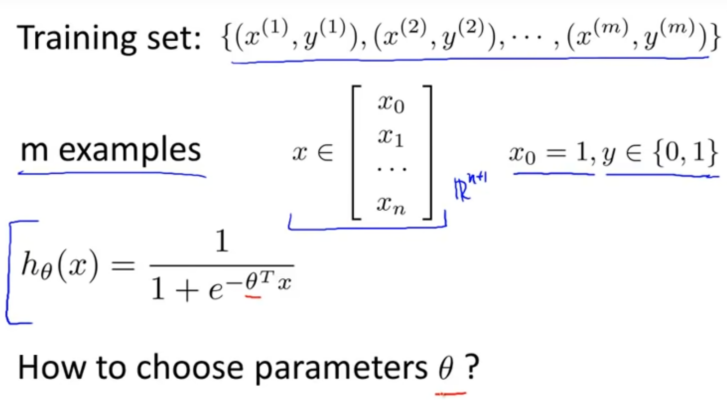
- Xs register as 1, and Os register as 0.
- 2 more features added, x_1^2 and x_2^2
- Now, 5 parameters, Θ_0 to Θ_4
- $x_1^2 + x_2^2 = 1$ is an equation for a circle with a radius of 1 centered on the origin.
- With higher order (more complex) polynomial values, the more complex the decision boundary can be, resulting in a variety of complex shapes

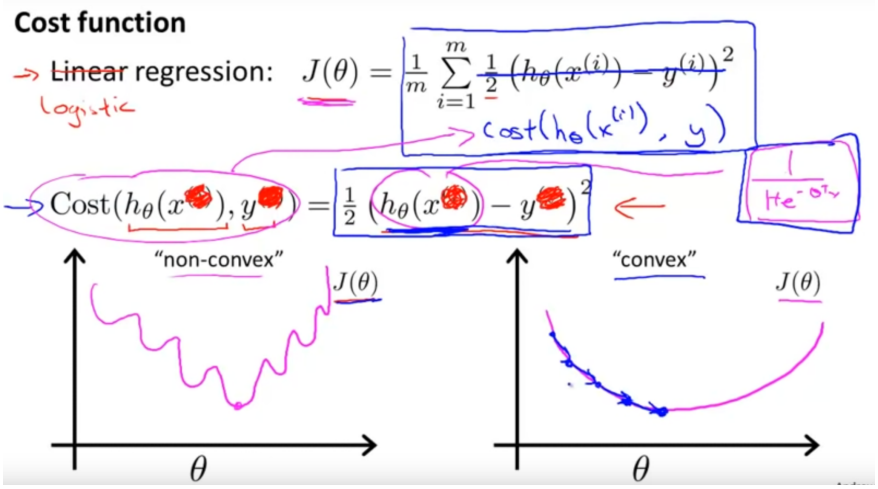


LECTURE 6.4: COST FUNCTION and how to automatically choose the values for Θ for logistic regression and defining the optimization objective.

↳ training set of m training examples, and each example is represented by a feature vector that is n+1 dimensional

↳ first feature, x_0 is 1 (first feature is always equal to 1), and since this is a classification problem, every value for y will be either 0 or 1

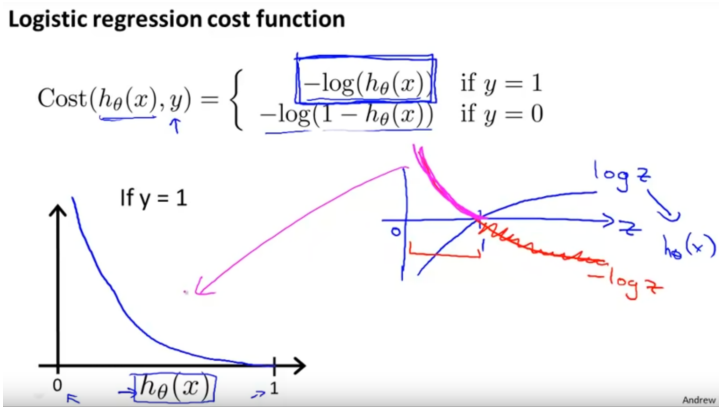




↳ In linear regression model, the cost function is a 1/m times the sum over the training set

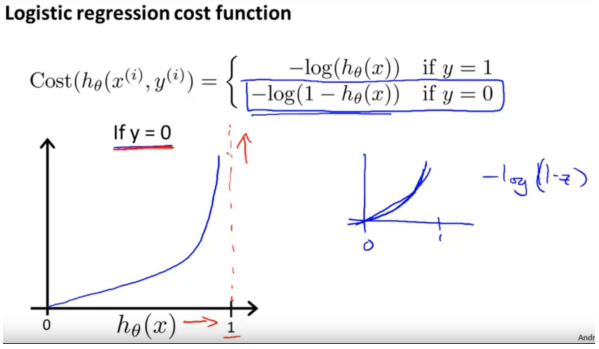
↳ Cost(hΘ(x, y) = ½ (h Θ(x - y)^2 (equals ½ of the error^2), but **this would be a non-convex function** of the parameter's data. If you run gradient descent on this, it is not guaranteed to converge to the global minimum

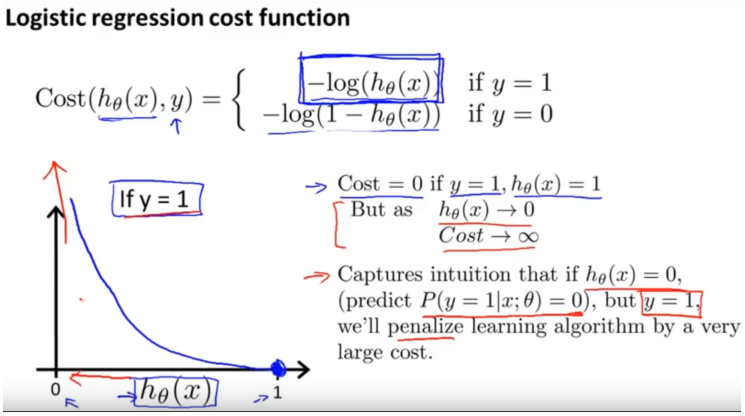
↳ We want a **convex function** so we can easily find the global minimum loss



↳ The cost goes up so high if the y turns out to equal 1, and our weights (Θ) will be affected greatly. For example, if we tell a patient that there is 100% certainty that their tumor is benign (y=0) but it turns out that it is malignant (y=1), our cost function penalizes our model greatly.

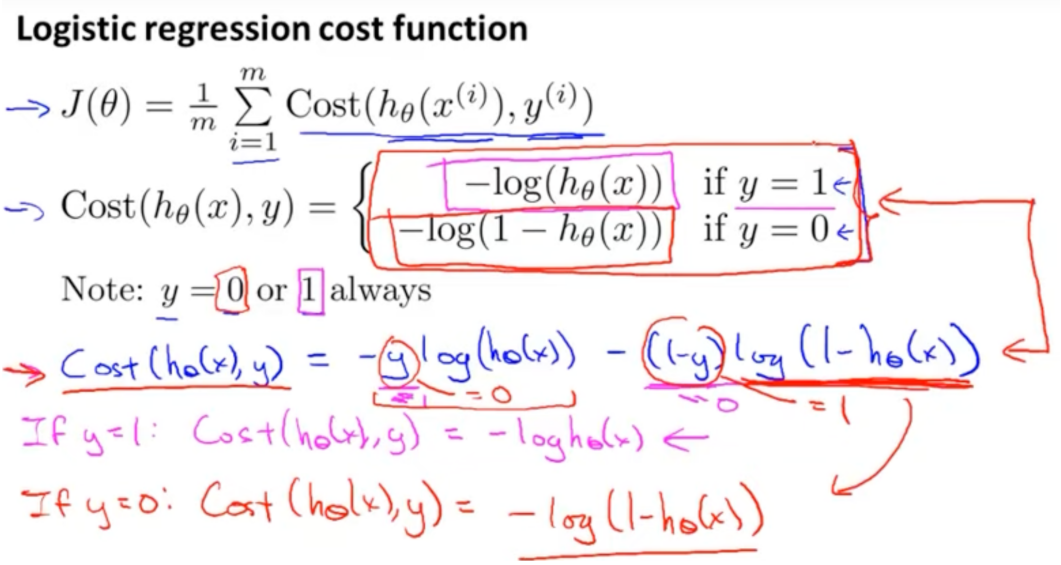
↳ what if y=0? Likewise, we end up paying a very large cost if our prediction is not correct, as hΘ(x) approaches 1





↳ Conversely if the prediction is right on, cost is low.

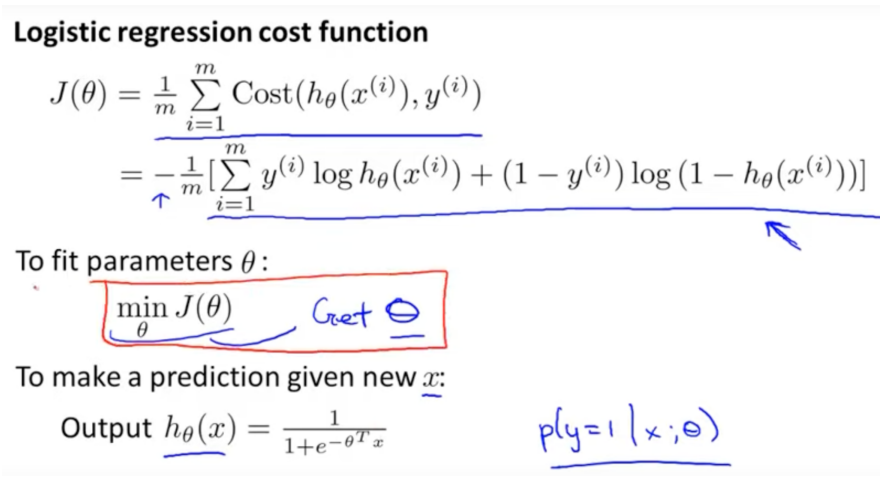
LECTURE 6.5: SIMPLIFIED COST FUNCTION and GRADIENT DESCENT



↳ The equations can be combined into one line, because when you apply either 0 or 1 to the equation, it will cancel out the half of the equation that does not apply.

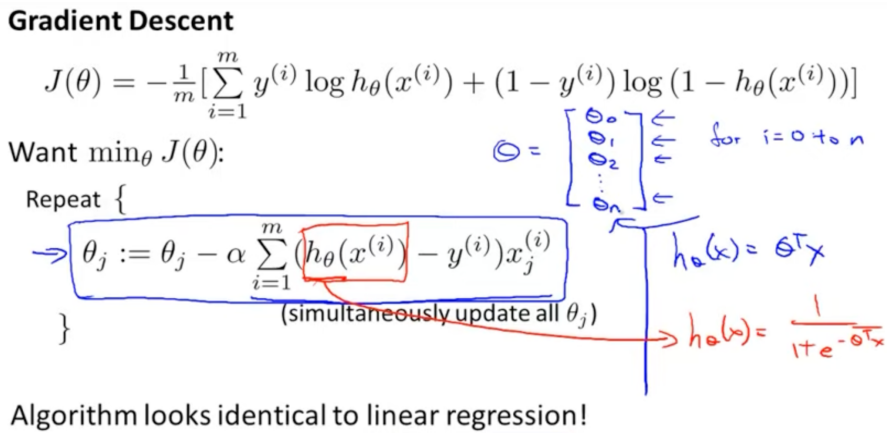
↳ The one liner cost function can now be plugged into a cost function previously used for a linear regression algorithm.

↳ There can be other cost functions, but this one is derived from statistics using the **principle of maximum likelihood estimation**. This is an idea in statistics for how to efficiently find parameters data for different models and it also is convex, which is very important.



↳ This is the one most often use for fitting logistic regression models.

WIKIPEDIA: *In statistics,* ***maximum likelihood estimation*** *(**MLE**) is a method of estimating the parameters of an assumed probability distribution, given some observed data. This is achieved by maximizing a likelihood function so that, under the assumed statistical model, the observed data is most probable. The point in the parameter space that maximizes the likelihood function is called the maximum likelihood estimate*



↳ We use this to minimize J, the loss, or difference between the predicted and the actual data.

↳ Using the cost function above, we continually apply it to the output of our model, updating the weights and bias based on the difference between the result of the predictions and the actual data in the training set, using an alpha learning rate value.

↳ The updating function here is identical to that of linear regression. The difference is that the definition for the hypothesis has changed. For linear it is $h\Theta(x) = \Theta^T x$, it is now $h\Theta(x) = 1 / 1 + e^{-\Theta x}$ (using a sigmoid activation function).

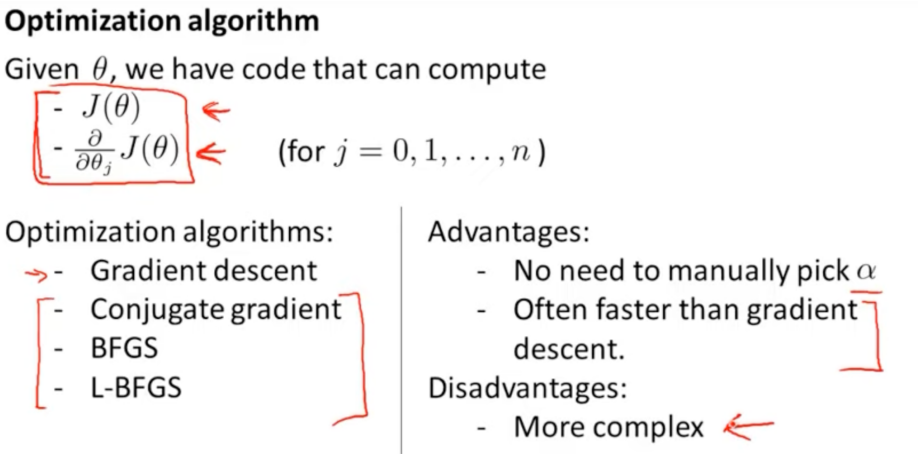
↳ For gradient descent, it is good to monitor it to make sure it is converging. (See Ng's linear regression videos for details on this and figure out how to apply to logistic regression.)

↳ Updating would ideally be done using a vectorized implementation, which would update all the Θ parameters in one step (rather than a for-loop, etc.)

↳ Feature scaling can help gradient descent converge faster in both linear and logistic regression.

LECTURE 6.6: ADVANCED OPTIMIZATION

↳ Gradient descent repeatedly updates the weight parameters based on the partial derivatives $\alpha/\alpha\Theta_j (J\Theta)$.



↳ Technically, you do not actually need code to compute the function $J\Theta$, but only the code to compute the derivative terms, but in order to also monitor convergence of gradient descent as well, then writing code to compute both is essential.

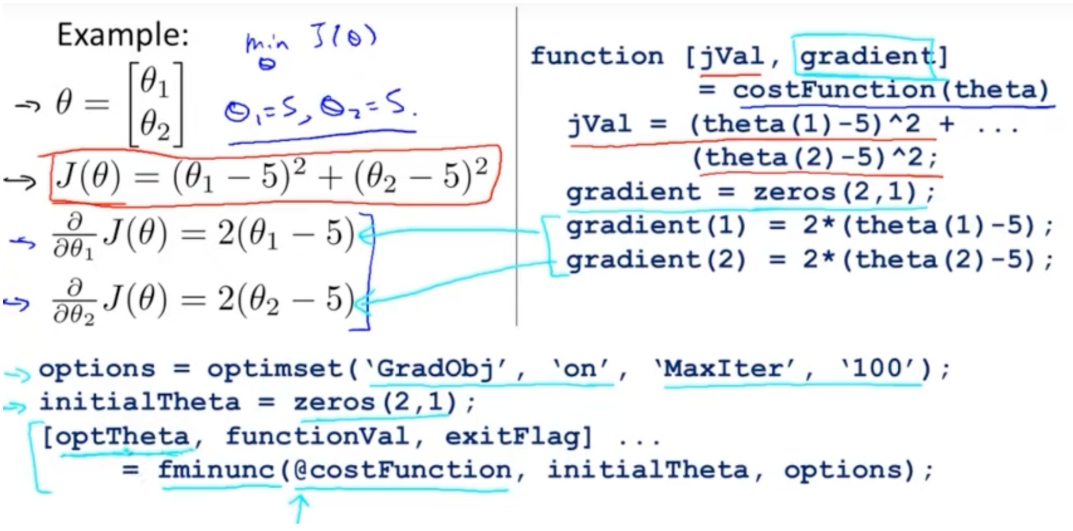
↳ There are more advanced and sophisticated algorithms than gradient descent for optimization. The algorithms listed here are examples of such:

↳ They have a clever inner loop called a line-search algorithm that automatically tries out different values for the learning rate α and will update it with every iteration as well.

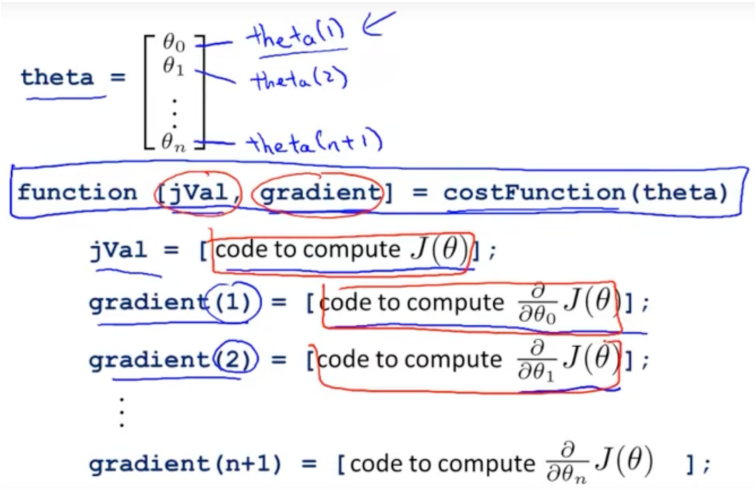
↳ Ng suggests not using these algorithms unless you are advanced in numerical computing, but rather use libraries that have been created for the purpose, such as **Octave** (matlab variant).

↳ Example of how to use these optimization algorithms: given two parameters (weights)

↳ (implemented here using derivatives that Ng calculated, the α functions)



↳ The Octave library function returns the value for J, it takes two arguments, the jVal, applies the cost function as shown, and the



gradient in this case is a 2 by 1 vector, the two elements of which are the two partial derivatives.

↳ Then you call the advanced optimization function, fminunc() (function minimization unconstrained), using options, a data structure that stores the options you want such as GradObj, the gradient, which is set to 'on' and the maximum number of iterations.

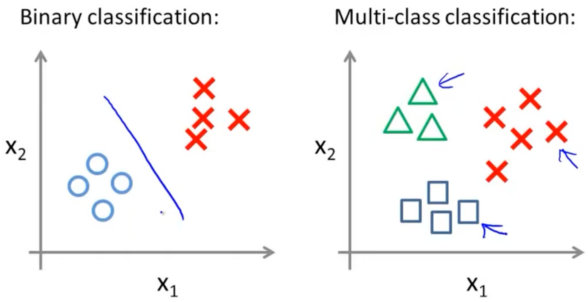
↳ You also give an initial guess for Θ, must be at least a 2 by 1 vector in this case.

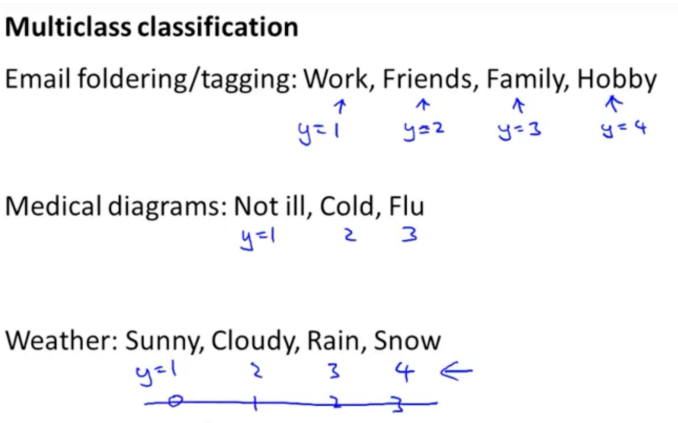
↳ then the command calls fminfunc(), which utilizes the cost function expressed above it

↳ Octave indexes starting at [1]

LECTURE 6.7: MULTI-CLASS CLASSIFICATION (One Versus All classification)

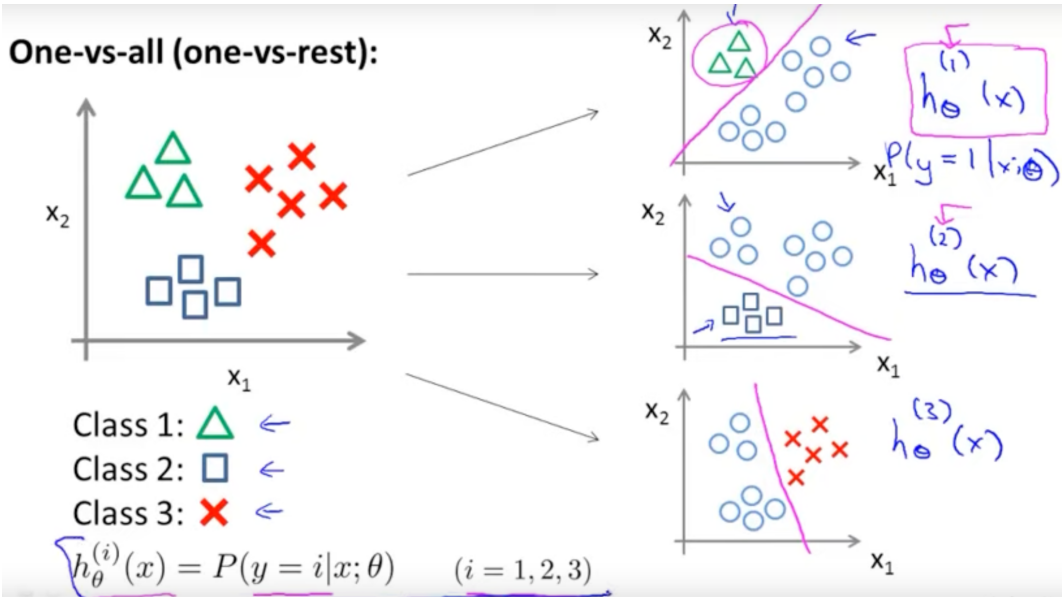
↳ multiple classes, ex email classification, 4 classes in this case, and so on





↳ When working with more than two classes, in an example like this where we have three classes, we will turn it into a set of three different two-class classification problems.

↳ With the first example, we will create a "fake" training set where classes two and three get assigned to the negative set and class 1 gets assigned to the positive set. Then we fit a classifier where triangles are 1 and circles are 0. Then repeat with class 2 as the positive, and the



other classes as negative. The superscript above the h_Θ signifies which class we are running as the positive class currently. Repeat with the third class.

↳ The equation shows how we apply the function, the *i* variable represents which combination of classes are being considered.

↳ This equation says that out of all the classes that could be considered as the positive class currently, i.e. y = 1, we want to choose the max, the one that has the highest probability.