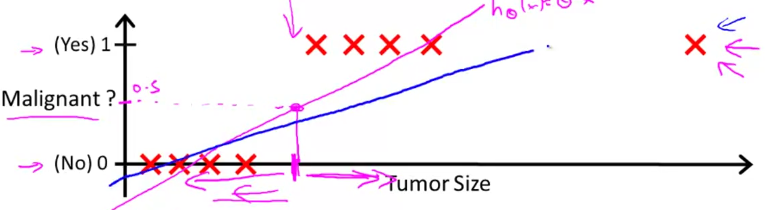
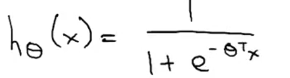
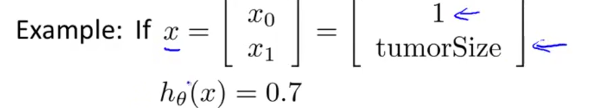
* **Binary Classification**  🡪 the variable that you want to predict, y, is valued via **logistic regression**
* Ex: email spam classification, classifying online transactions as fraudulent/not (someone is using a stolen credit card or stolen user password), classifying tumors as malignant or benign.
* In all of these problems the variable we're trying to predict takes on TWO values 🡪 0/1, spam/not spam, fraudulent/not fraudulent, malignant/benign.
*  🡪 0 = the **negative class** (benign), 1 = **positive class** (malignant)
* Assignment of the 2 classes is somewhat arbitrary
* Intuition = negative class conveys absence of something, positive class conveys presence of
* **Multiclass classification problem** **🡪** values of 1-4, etc.
* Ex: Training set for classifying tumor as malignant or benign where malignancy takes on 2 values, 0/1
* 1 thing we could do, given this training set = apply the Linear regression algorithm that we already know to this data set + try to fit a straight line to the data to get a hypothesis, h(x)
* 
* To make predictions, try to **threshold** the classifier outputs at 0.5 🡪 if hypothesis outputs a value >= 0.5, say y = 1, if < 0.5 say y = 0.
* For this example, it looks like linear regression is actually doing something reasonable, even though this is classification
* But now change the problem a bit w/ 1 training example very out to the right (outlier + if you run linear regression now, we instead get a worse straight line fit to the data + a worse hypothesis.
* 
* So, applying linear regression to a classification problem often isn't a great idea.
* In the 1st example, linear regression was just getting lucky + got us a hypothesis that worked well
* For classification we know that y = 0/1, but if using linear regression, h(x) can output values much larger than 1 or less than 0, even if all training examples have labels y = 0 or 1.
* Even though we know the labels should be 0/1, the algorithm can output values much larger than 1 or much smaller than0
* **Logistic regression** has the property that the output/predictions are always between zero and one
* Want our classifier to output values between 0 and 1, so we want a hypothesis that satisfies this property w/ predictions between 0 and 1.
* When using linear regression, h(x) = theta(transpose)\*x.
* For logistic regression, we modify this a little bit + make h(x) = g(theta(t)\*x), + define g as:
* G(z) = 1 / (1 + E^-z) = the **sigmoid function/the logistic function**
* Put these 2 together 🡪 h(x) = 1 / (1 – E^-(theta(t)\*x))



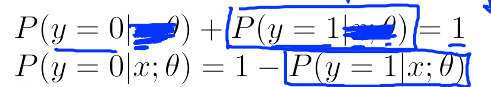
* Plotted, the sigmoid function, g(z), starts off near 0 + rises until it crosses 0.5 at the origin + then flattens out again near 1 = **asymptotes** at one + zero
* So b/c g(z) values are between zero and one, we also have that h(x) must be between zero and one.
* Finally, given this hypothesis representation + given a training set, we need to fit the parameters theta to our data.
* To interpret, when h(x) outputs some number, treat it as the **estimated probability** that y = one on a new input, x.



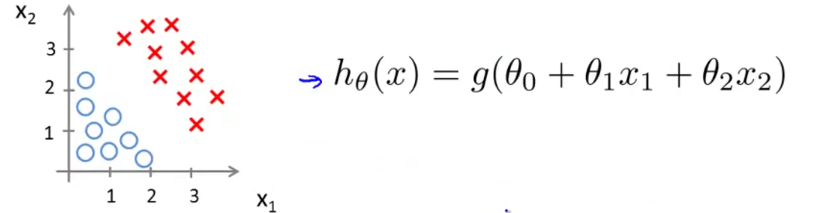
* Patient has a 70% chance, or a 0.7 chance of being malignant.
* *More formally, h(x) outputs the probability* ***P*** *of y = 1 given x, parameterized by theta.*



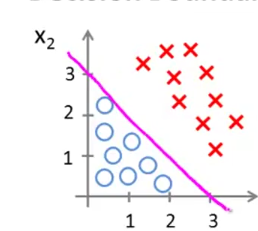
* Since this is a classification task, we know y must be 0 or 1, so given h(x), we can therefore compute probability of y = 0 as well via (1 – h(x)) b/c probability of y = 0 + probability of y = 1 must = 1.



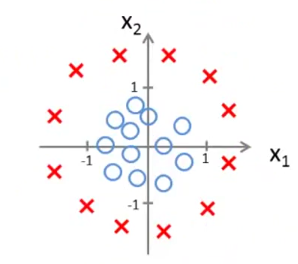
* So h(x) = g(theta(t)\*x) = 1 / (1 – E^-(theta(t)\*x)) , which slowly increases from 0 to 1.
* H(x) is outputting estimates of the probability that y = one, given x and parameterized by theta.
* We can assume that if h(x) >= 0.5, its more likely to be y = 1 than y equals 0, so we predict y = 1, and vice versa for h(x) < 0.5
* Looking at the sigmoid function, g(z) >= 0.5 whenever z >= 0.
* Since the hypothesis for logistic regression 1 / (1 – E^-(theta(t)\*x)), this h(x) is therefore going to be >= 0.5 whenever theta(t)\*x (b/c this is z) >= 0.
* So a hypothesis predicts y = 1 whenever theta(t)\*x >= 0.
* By similar argument, h(x) < 0.5 whenever g(z) < 0.5 because the range of values of z that cause g(z) to take on values < 0.5 (z = negative)
* So when g(z) < 0.5, a hypothesis will predict y = 0 + h(x) will predict y = 0 whenever theta(t)\*x < 0.
* To summarize if we decide to predict y=1 or y=0 depending on whether an estimated probability h(x) >=0.5 or < 0.5, its the same as saying we predict y=1 whenever theta(t)\*x >= 0 and predict y is = 0 whenever theta(t)\*x < 0



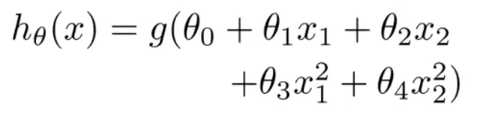
* Suppose we have a training set + a h(x) above + suppose that via a procedure-to-be-specified, we end up choose parameter value theta 0 = -3, theta 1 = 1, theta 2 = 1.
* parameter vector = 3\*1 = [-3, 1, 1]
* Given this choice of hypothesis parameters, try to figure out where a hypothesis would end up predicting y = 1 or y = 0.
* We know the probability y = 1 is more likely (>= 0.5) when theta(t)\*x > 0, + w/ out theta values, this is: *y = 1 if -3 + x1 + x2 >= 0 🡺 x1 + x2 >= 3*
* X1 + X2 = 3 defines the equation of a straight line which passes through 3 on the x1 + the x2 axis.



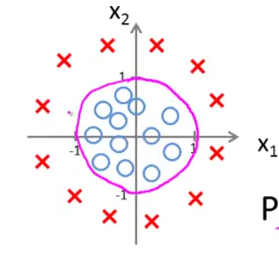
* So the part of the X1-X2 plane that corresponds to when X1 + X2 >= 3 is everything to the upper right portion of the magenta line 🡪 the region where our hypothesis predicts y = 1
* The region where x1 + x2 < 3 corresponds to the region below the line = where our hypothesis predicts y = 0.
* This magenta line = **the decision boundary.**
* X1 + X2 = 3 corresponds to the set of points/region where h(X) = 0.5 exactly = the decision boundary straight line = separates the region where h(x) predicts Y = 1 from the region where it predicts y = 0.
* The decision boundary is a *property of the hypothesis*, including the parameters theta0, theta1, theta2, not of the data set.
* 9:22
* Later we will fit the parameters + use the training set data to determine the value of the parameters
* More complex example 🡪 non-linear



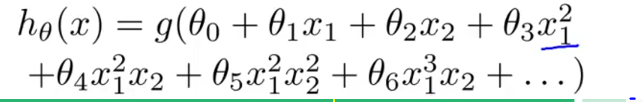
* Could add extra higher-order polynomial terms to our features, like in polynomial regression in linear regression
* Add 2 extra features, x1^2 and x2^2 to the features = 5 parameters, theta0 through theta4.



* Assume via a future procedure to be specified, we choose theta0 = -1, theta1 = 0, theta2 = 0, theta3 = 1 + theta4 = one, so our parameter vector = 5\*1 🡪 [-1, 0, 0, 1, 1]
* So h(x) predicts y = 1 when -1 + x1^2 + x2^2 >= 0 = whenever theta(t)\*my features >= 0 = So h(x) predicts y = 1 when x1^2 + x2^2 >= 1
* Now h(x) is the equation for circle of radius 1 centered around the origin = the decision boundary.



* Everything outside the circle 🡪 predict y = 1, inside the circle 🡪 y = 0.
* So by adding more complex polynomial terms to my features, I can get more complex decision boundaries that don't just try to separate the positive and negative examples in a straight line
* Once again, the decision boundary is a *property*, NOT of the *training set*, but of the *hypothesis* under the parameters.
* the training set is NOT what we use to define the decision boundary.
* The training set *may* be used to *fit the parameters theta.*
* But, once you have the parameters theta, *that* is what defines the decisions boundary.
* More complex example.
* Can we come up w/ even more complex decision boundaries then this if I have even higher polynomial terms?



* Now we can find decision boundaries that may be an ellipse or some funny shape

