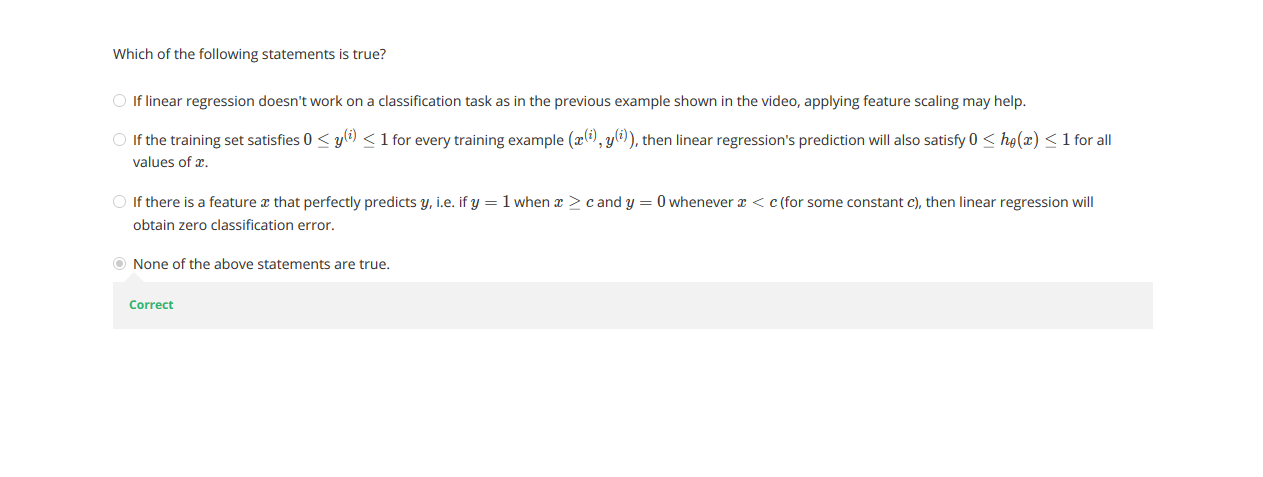
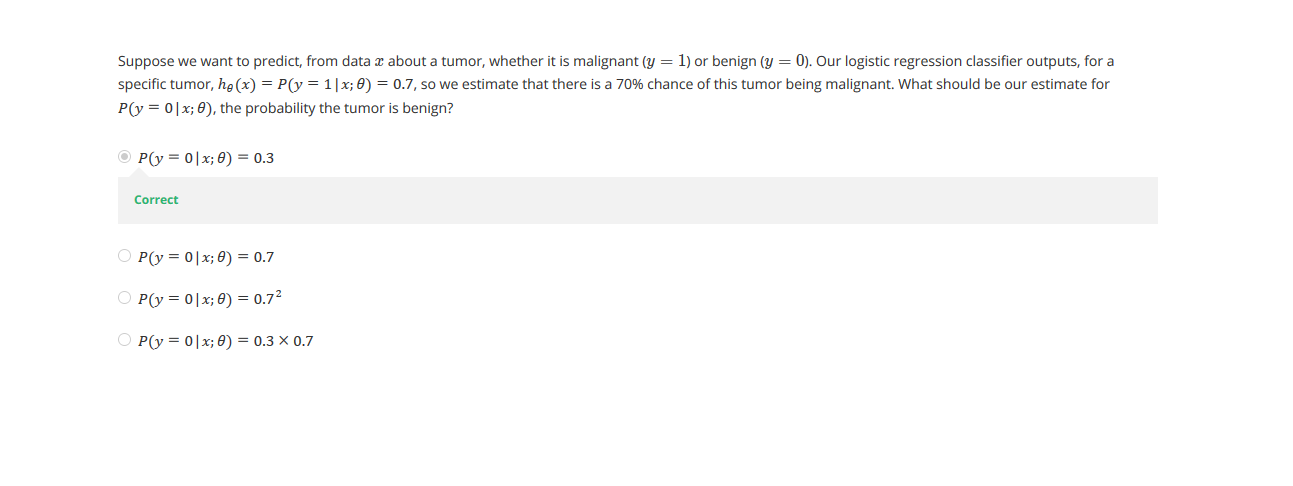
--

**Classification**

To attempt classification, one method is to use linear regression and map all predictions greater than 0.5 as a 1 and all less than 0.5 as a 0. However, this method doesn't work well because classification is not actually a linear function.

The classification problem is just like the regression problem, except that the values we now want to predict take on only a small number of discrete values. For now, we will focus on the **binary classification** **problem** in which y can take on only two values, 0 and 1. (Most of what we say here will also generalize to the multiple-class case.) For instance, if we are trying to build a spam classifier for email, then x(i)x^{(i)}x(i) may be some features of a piece of email, and y may be 1 if it is a piece of spam mail, and 0 otherwise. Hence, y∈{0,1}. 0 is also called the negative class, and 1 the positive class, and they are sometimes also denoted by the symbols “-” and “+.” Given x(i)x^{(i)}x(i), the corresponding y(i)y^{(i)}y(i) is also called the label for the training example.



**Hypothesis Representation**

We could approach the classification problem ignoring the fact that y is discrete-valued, and use our old linear regression algorithm to try to predict y given x. However, it is easy to construct examples where this method performs very poorly. Intuitively, it also doesn’t make sense for hθ(x)h\_\theta (x)hθ​(x) to take values larger than 1 or smaller than 0 when we know that y ∈ {0, 1}. To fix this, let’s change the form for our hypotheses hθ(x)h\_\theta (x)hθ​(x) to satisfy 0≤hθ(x)≤10 \leq h\_\theta (x) \leq 10≤hθ​(x)≤1. This is accomplished by plugging θTx\theta^TxθTx into the Logistic Function.

Our new form uses the "Sigmoid Function," also called the "Logistic Function":

|  |
| --- |
| hθ(x)=g(θTx)z=θTxg(z)=11+e−z |

The following image shows us what the sigmoid function looks like:



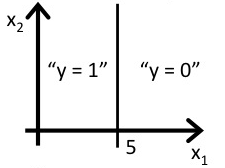
The function g(z), shown here, maps any real number to the (0, 1) interval, making it useful for transforming an arbitrary-valued function into a function better suited for classification.

hθ(x)h\_\theta(x)hθ​(x) will give us the **probability** that our output is 1. For example, hθ(x)=0.7h\_\theta(x)=0.7hθ​(x)=0.7 gives us a probability of 70% that our output is 1. Our probability that our prediction is 0 is just the complement of our probability that it is 1 (e.g. if probability that it is 1 is 70%, then the probability that it is 0 is 30%).

|  |
| --- |
| hθ(x)=P(y=1|x;θ)=1−P(y=0|x;θ)P(y=0|x;θ)+P(y=1|x;θ)=1 |

Consider logistic regression with two features x1x\_1x1​ and x2x\_2x2​. Suppose θ0=5\theta\_0 = 5θ0​=5, θ1=−1\theta\_1 = -1θ1​=−1, θ2=0\theta\_2 = 0θ2​=0, so that hθ(x)=g(5−x1)h\_\theta(x) = g(5 - x\_1)hθ​(x)=g(5−x1​). Which of these shows the decision boundary of hθ(x)h\_\theta(x)hθ​(x)?



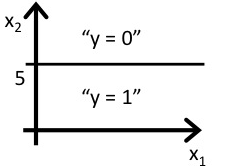
 **(A)**

.

Correct

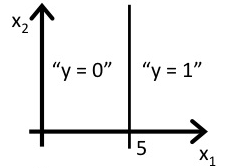
Predict Y = 0 if x1x\_1x1​ is greater than 5.





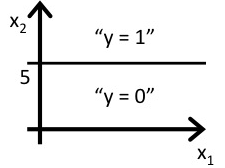
.





.





**Decision Boundary**

In order to get our discrete 0 or 1 classification, we can translate the output of the hypothesis function as follows:

|  |
| --- |
| hθ(x)≥0.5→y=1hθ(x)<0.5→y=0 |

The way our logistic function g behaves is that when its input is greater than or equal to zero, its output is greater than or equal to 0.5:

|  |
| --- |
| g(z)≥0.5whenz≥0 |

Remember.

|  |
| --- |
| z=0,e0=1⇒g(z)=1/2z→∞,e−∞→0⇒g(z)=1z→−∞,e∞→∞⇒g(z)=0 |

So if our input to g is θTX\theta^T XθTX, then that means:

|  |
| --- |
| hθ(x)=g(θTx)≥0.5whenθTx≥0 |

From these statements we can now say:

|  |
| --- |
| θTx≥0⇒y=1θTx<0⇒y=0 |

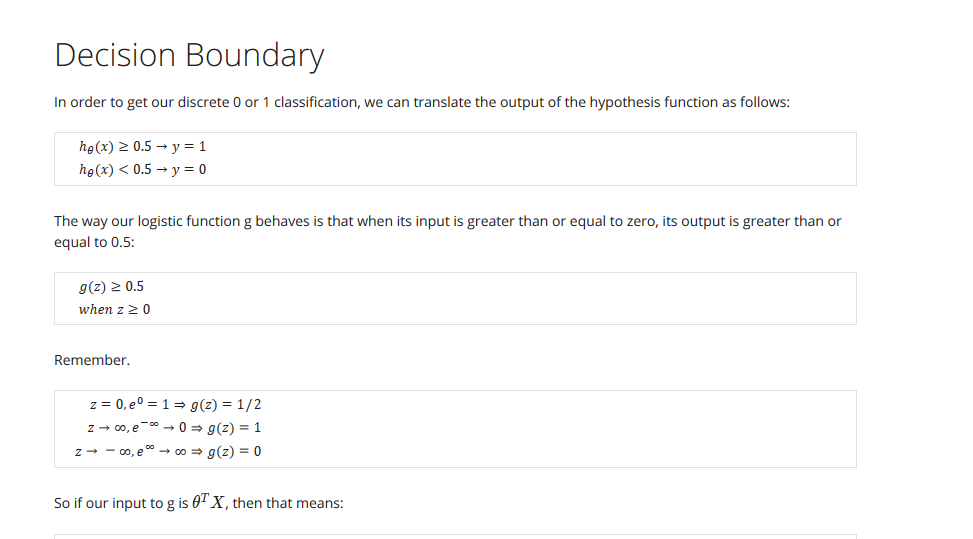
The **decision boundary** is the line that separates the area where y = 0 and where y = 1. It is created by our hypothesis function.

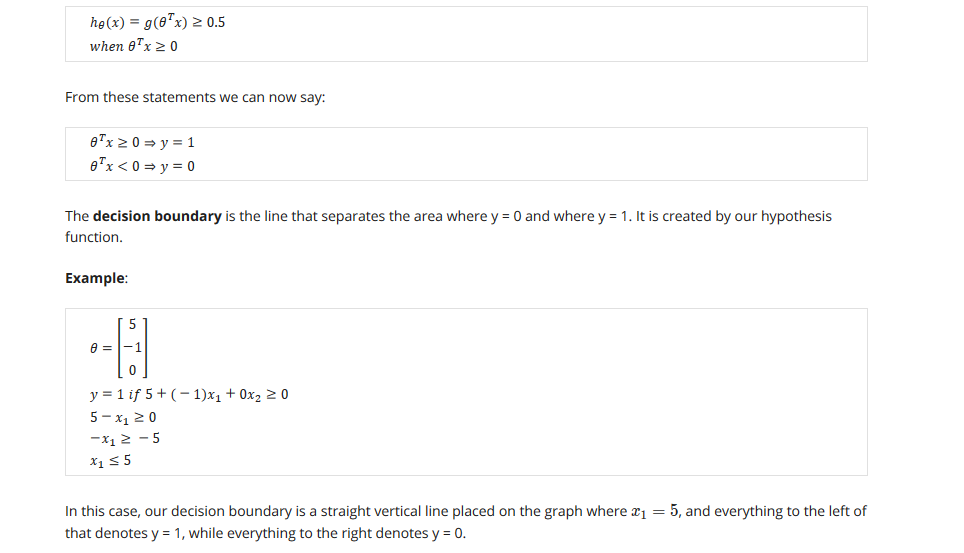
**Example**:

|  |
| --- |
| θ=5−10y=1if5+(−1)x1+0x2≥05−x1≥0−x1≥−5x1≤5 |

In this case, our decision boundary is a straight vertical line placed on the graph where x1=5x\_1 = 5x1​=5, and everything to the left of that denotes y = 1, while everything to the right denotes y = 0.

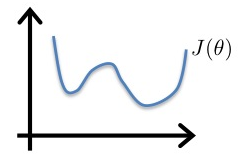
Again, the input to the sigmoid function g(z) (e.g. θTX\theta^T XθTX) doesn't need to be linear, and could be a function that describes a circle (e.g. z=θ0+θ1x12+θ2x22z = \theta\_0 + \theta\_1 x\_1^2 +\theta\_2 x\_2^2z=θ0​+θ1​x12​+θ2​x22​) or any shape to fit our data.





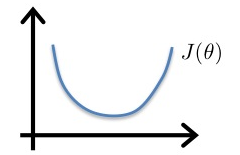
Consider minimizing a cost function J(θ)J(\theta)J(θ). Which one of these functions is convex?





.

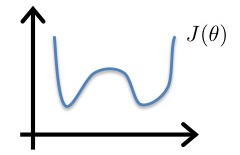


 **(A)**

.

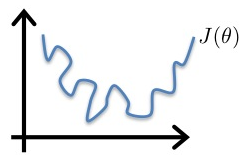
Correct

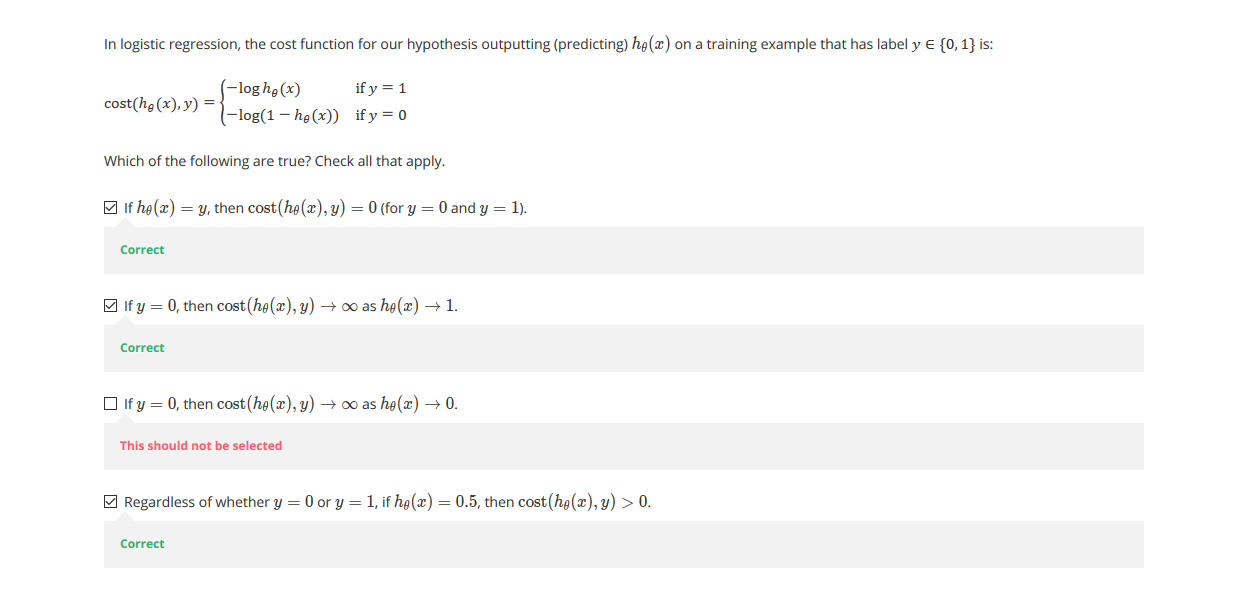




.







**Cost Function**

We cannot use the same cost function that we use for linear regression because the Logistic Function will cause the output to be wavy, causing many local optima. In other words, it will not be a convex function.

Instead, our cost function for logistic regression looks like:

|  |
| --- |
|  |

When y = 1, we get the following plot for J(θ)J(\theta)J(θ) vs hθ(x)h\_\theta (x)hθ​(x):



Similarly, when y = 0, we get the following plot for J(θ)J(\theta)J(θ) vs hθ(x)h\_\theta (x)hθ​(x):

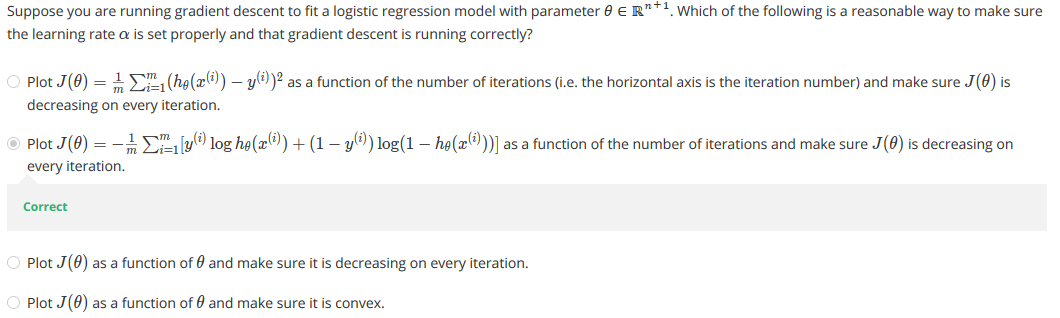


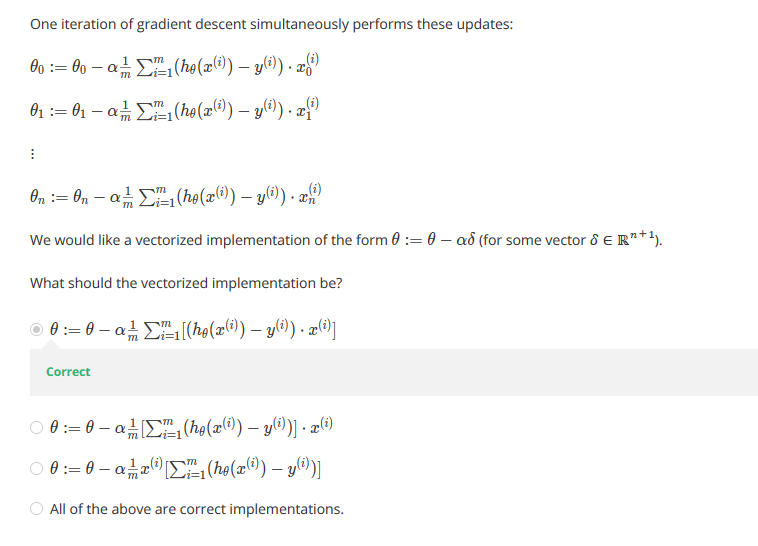
|  |
| --- |
|  |

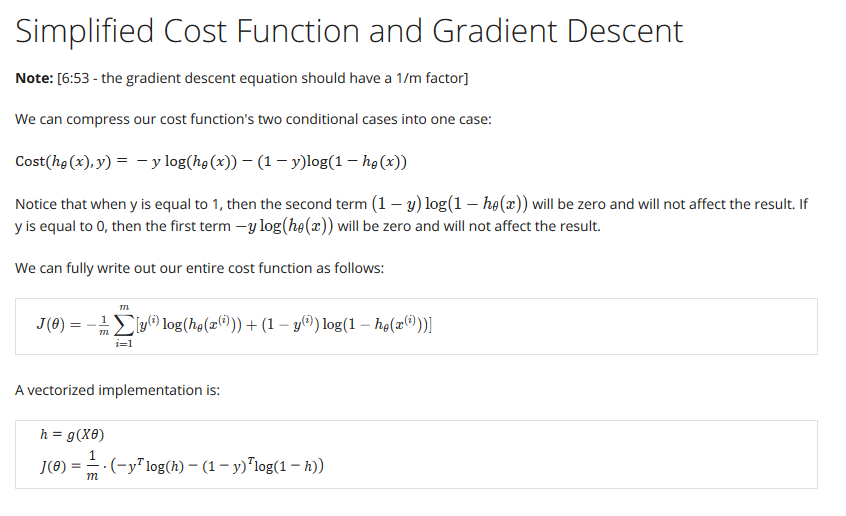
If our correct answer 'y' is 0, then the cost function will be 0 if our hypothesis function also outputs 0. If our hypothesis approaches 1, then the cost function will approach infinity.

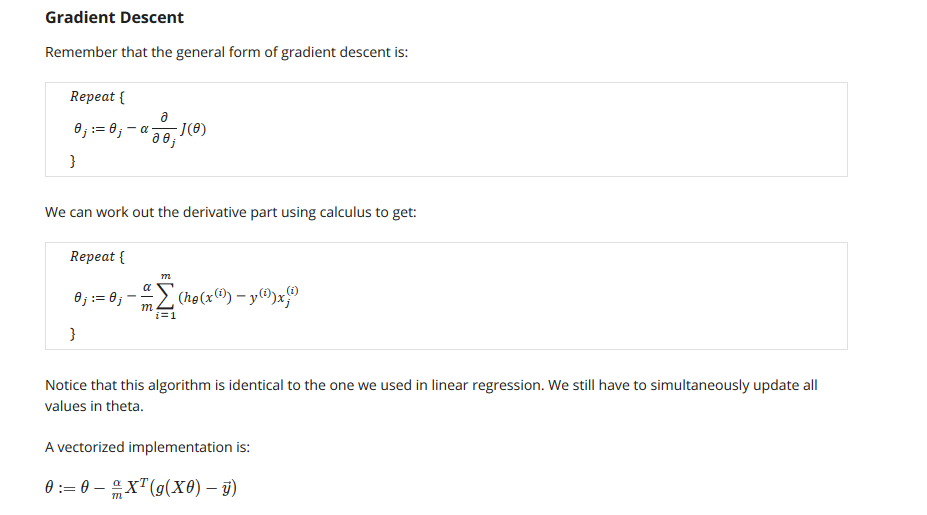
If our correct answer 'y' is 1, then the cost function will be 0 if our hypothesis function outputs 1. If our hypothesis approaches 0, then the cost function will approach infinity.

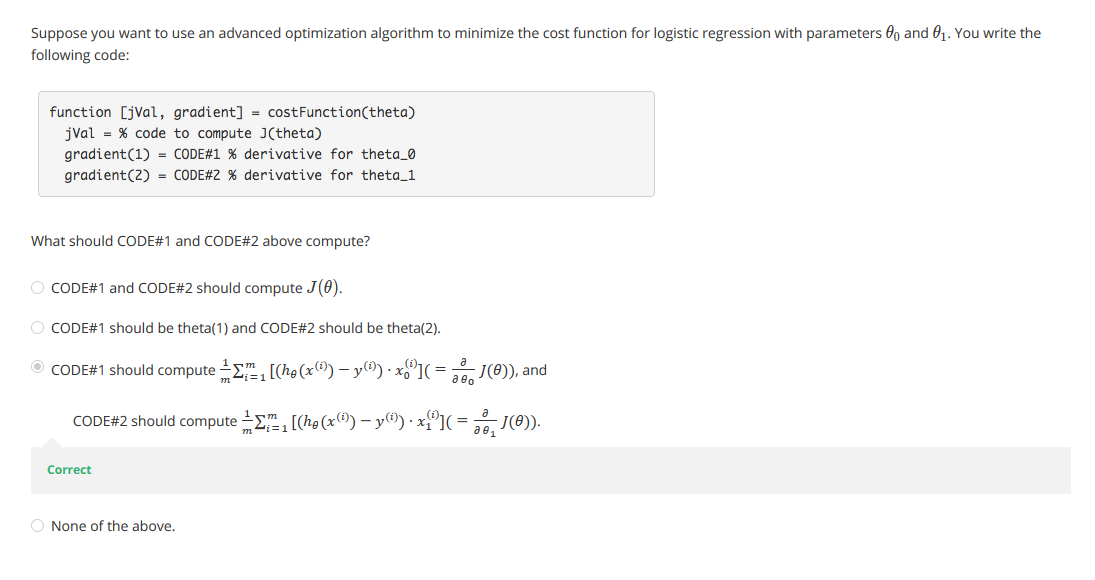
Note that writing the cost function in this way guarantees that J(θ) is convex for logistic regression.











**Advanced Optimization**

**Note:** [7:35 - '100' should be 100 instead. The value provided should be an integer and not a character string.]

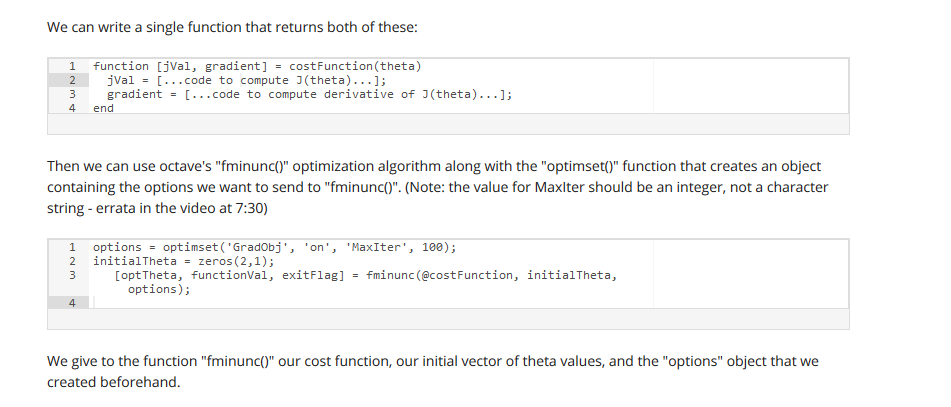
"Conjugate gradient", "BFGS", and "L-BFGS" are more sophisticated, faster ways to optimize θ that can be used instead of gradient descent. We suggest that you should not write these more sophisticated algorithms yourself (unless you are an expert in numerical computing) but use the libraries instead, as they're already tested and highly optimized. Octave provides them.

We first need to provide a function that evaluates the following two functions for a given input value θ:

J(θ)∂∂θjJ(θ)

We can write a single function that returns both of these:

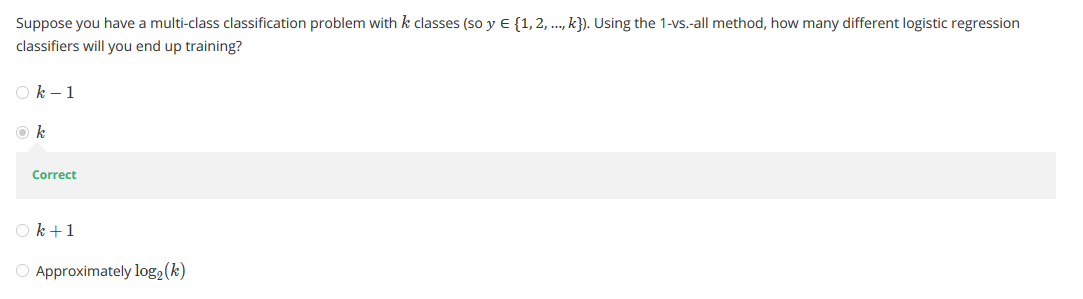


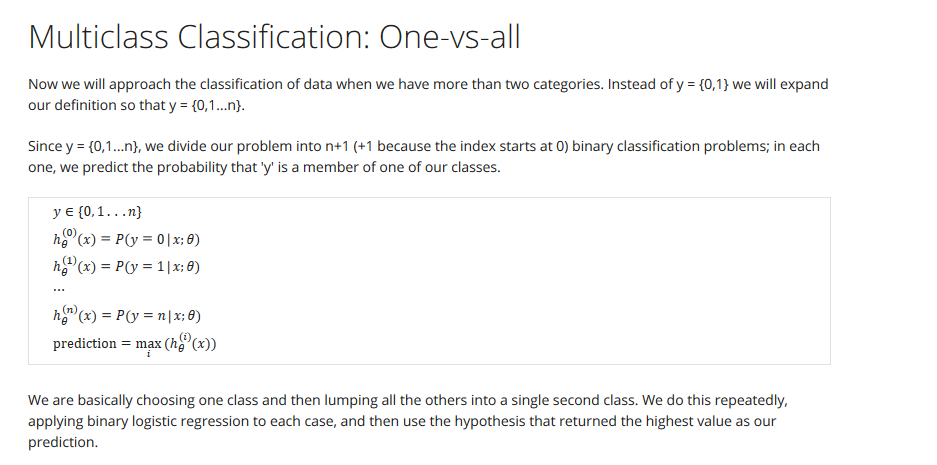


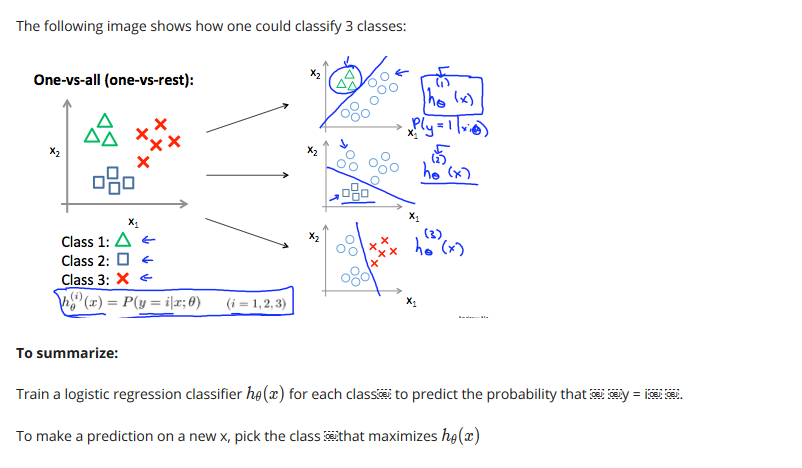
Then we can use octave's "fminunc()" optimization algorithm along with the "optimset()" function that creates an object containing the options we want to send to "fminunc()". (Note: the value for MaxIter should be an integer, not a character string - errata in the video at 7:30)

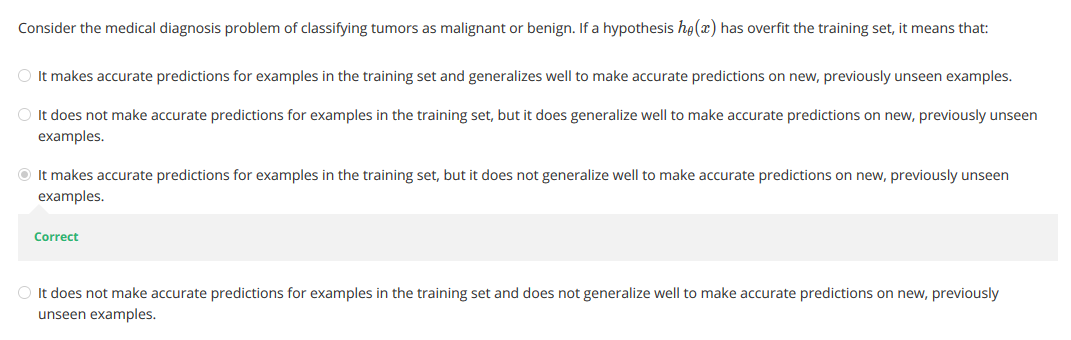


We give to the function "fminunc()" our cost function, our initial vector of theta values, and the "options" object that we created beforehand.



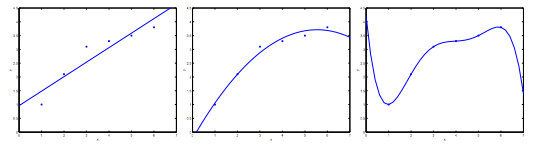






# The Problem of Overfitting

Consider the problem of predicting y from x ∈ R. The leftmost figure below shows the result of fitting a y = θ0+θ1x to a dataset. We see that the data doesn’t really lie on straight line, and so the fit is not very good.



Instead, if we had added an extra feature x2x^2x2 , and fit y=θ0+θ1x+θ2x2y = \theta\_0 + \theta\_1x + \theta\_2x^2y=θ0​+θ1​x+θ2​x2 , then we obtain a slightly better fit to the data (See middle figure). Naively, it might seem that the more features we add, the better. However, there is also a danger in adding too many features: The rightmost figure is the result of fitting a 5th5^{th}5th order polynomial y=∑j=05θjxjy = \sum\_{j=0} ^5 \theta\_j x^jy=∑j=05​θj​xj. We see that even though the fitted curve passes through the data perfectly, we would not expect this to be a very good predictor of, say, housing prices (y) for different living areas (x). Without formally defining what these terms mean, we’ll say the figure on the left shows an instance of **underfitting**—in which the data clearly shows structure not captured by the model—and the figure on the right is an example of **overfitting**.

Underfitting, or high bias, is when the form of our hypothesis function h maps poorly to the trend of the data. It is usually caused by a function that is too simple or uses too few features. At the other extreme, overfitting, or high variance, is caused by a hypothesis function that fits the available data but does not generalize well to predict new data. It is usually caused by a complicated function that creates a lot of unnecessary curves and angles unrelated to the data.

This terminology is applied to both linear and logistic regression. There are two main options to address the issue of overfitting:

1) Reduce the number of features:

* Manually select which features to keep.
* Use a model selection algorithm (studied later in the course).

2) Regularization

* Keep all the features, but reduce the magnitude of parameters θj\theta\_jθj​.
* Regularization works well when we have a lot of slightly useful features.

