## Classification and Representation

## Logistic Regression Model

- ▶ Cost Function — 10 min
- ▤ Cost Function — 3 min
- ▶ Simplified Cost Function and Gradient Descent — 10 min
- ▤ Simplified Cost Function and Gradient Descent — 3 min
- ▶ Advanced Optimization — 14 min
- ▤ Advanced Optimization — 3 min

## Multiclass Classification

## Review

## Solving the Problem of Overfitting

## Review

# 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:

$$J(\theta)$$

$$\frac{\partial}{\partial \theta_j} J(\theta)$$

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

```
1  function [jVal, gradient] = costFunction(theta)
2    jVal = [...code to compute J(theta)...];
3    gradient = [...code to compute derivative of J(theta)...];
4  end
```

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)

```
1  options = optimset('GradObj', 'on', 'MaxIter', 100);
2  initialTheta = zeros(2,1);
3    [optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta,
         options);
4
```

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

✓ Completado