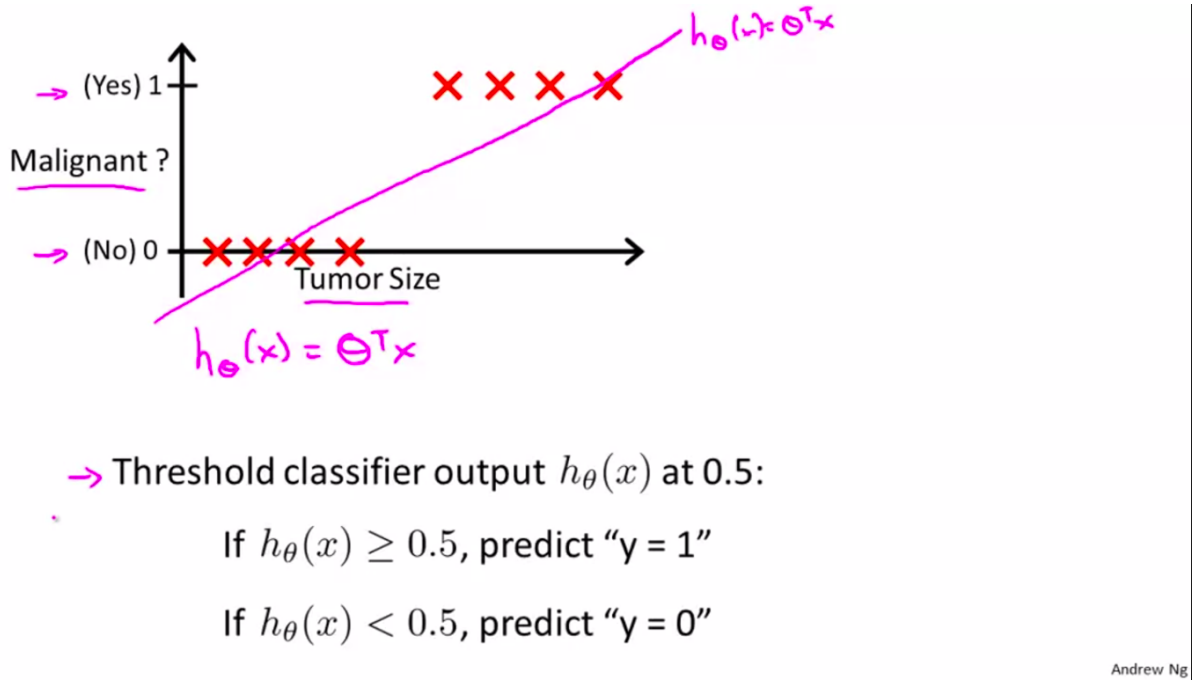
Lecture 2

Classification

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.

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.



However, it is easy to construct examples where this method performs very poorly. One of the problems with this approach is that, even though $y \in \{0, 1\}$, for example, it might happen that our prediction $h_\theta(x) > 1$ or $h_\theta(x) < 0$ or assume any real valued number.

Binary classification problem: y can take on only two values, 0 and 1.

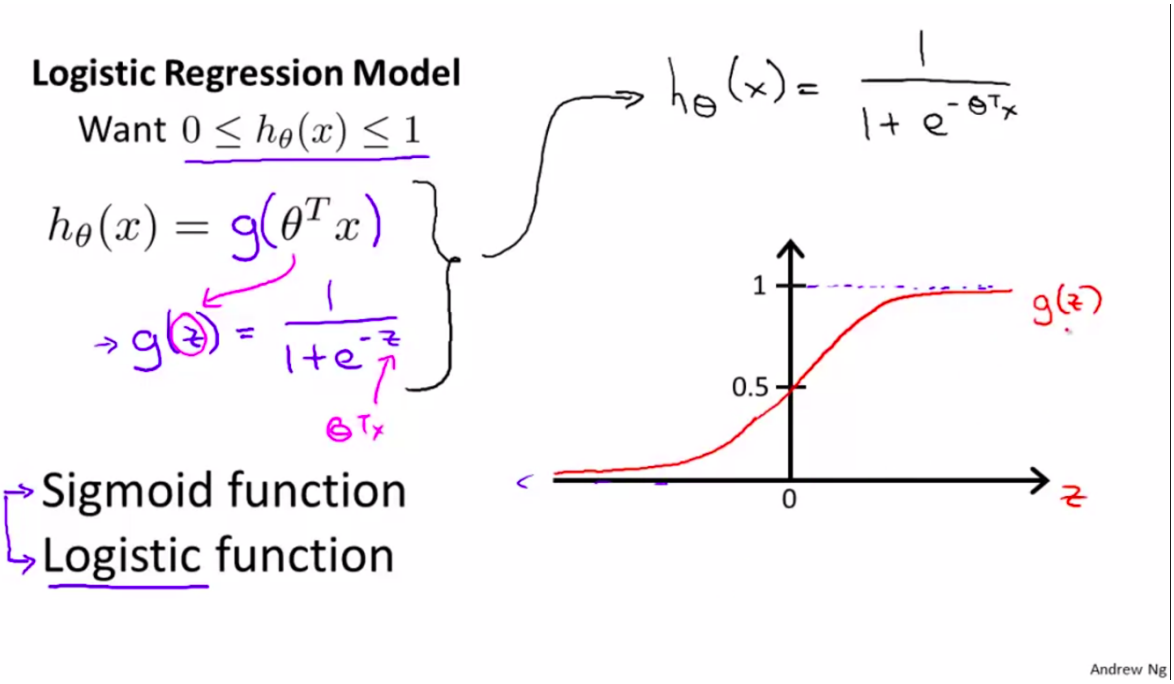
> **Examples:**
>
> - Email (spam or not spam?)
> - Tumor (malignant or benign?)
> - Online transactions (fraudulent? yes or no)
>
> $y \in \{0, 1\}$, where 0 is the "negative class" (benign tumor) and 1 is the "positive class" (malignant tumor)

Hypothesis Representation

As mentioned previously, approaching the classification problem by ignoring the fact that y is discrete-valued, and using linear regression algorithm to try to predict y given x is problematic in many ways. For example, in **binary classification problems**, we know that either $y = 0$ or $y = 1$, so it doesn't make sense that $h_\theta(x) > 1$ or $h_\theta(x) < 0$.

To correct this, let's change the form of our hypothesis $h_\theta(x)$ to satisfy $0 \leq h_\theta \leq 1$. This is accomplished by pluging $\theta^T x$ into the **Logistic (Sigmoid) Function**.

$$
\begin{aligned}
h_\theta(x) &= g(\theta^T x) \\
z &= \theta^T x \\
g(z) &= \frac{1}{1 + e^{-z}}
\end{aligned}
\tag{1}
$$



The function $g(z)$ maps any real number to the interval $(0, 1)$.

Interpretation of hypothesis output: $h_\theta(x)$ is the estimated probability that $y = 1$ **given** the input x.

$$
h_\theta(x) = \mathcal{P}(y = 1 \mid x; \theta) \tag{2}
$$

p.s.: remember that

$$
\mathcal{P}(y = 1 \mid x; \theta) = 1 - \mathcal{P}(y = 0 \mid x; \theta) \tag{3}
$$

> Example:
> If $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ tumorSize \end{bmatrix}$, then $h_\theta(x) = 0.7$ indicates that the patient has 70% chance of tumor being malignant.

Decision Boundary

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

$$h_\theta(x) \geq 0.5 \quad \Rightarrow \quad y = 1$$
$$h_\theta(x) < 0.5 \quad \Rightarrow \quad y = 0$$

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:

$$g(z) \geq 0.5, \quad whenz \geq 0.$$

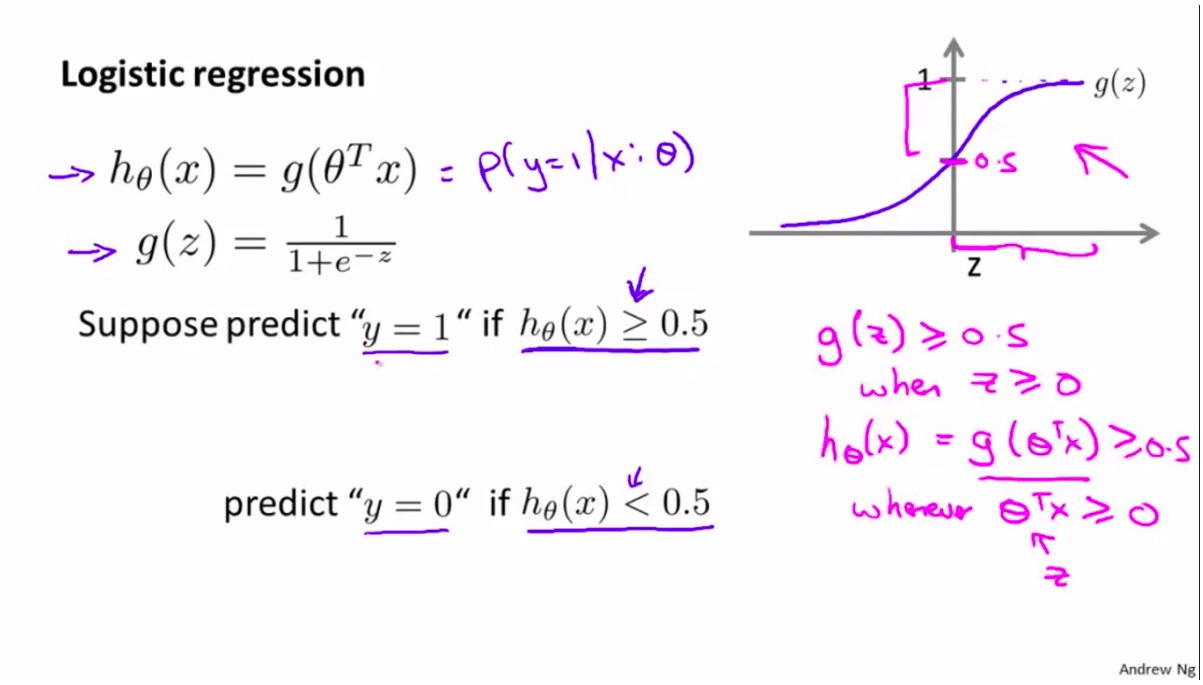
> **Note:**
> $z = 0,\ e^0 = 1 \ \Rightarrow \ g(z) = 1/2$
> $z \to \infty,\ e^{-\infty} \to 0 \ \Rightarrow \ g(z) = 1$
> $z \to -\infty,\ e^{\infty} \to \infty \ \Rightarrow \ g(z) = 0$

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

$$h_\theta(x) = g(\theta^T X) \geq 0.5, \quad \text{when} \quad \theta^T x \geq 0.$$

From these statements we can now say:

$$\theta^T x \geq 0 \quad \Rightarrow \quad y = 1$$
$$\theta^T x < 0 \quad \Rightarrow \quad y = 0$$



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:

$$\theta = \begin{bmatrix} 5 \\ -1 \\ 0 \end{bmatrix}$$

So, $y = 1$ if $\theta^T x = 5 - x_1 + 0x_2 \geq 0 \Rightarrow 5 - x_1 \geq 0 \Rightarrow x_1 \leq 5.$

Note: Again, the input to the sigmoid function g(z) (e.g. $\theta^T X$) doesn't need to be linear, and could be a function that describes a circle (e.g. $z = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2$) or any shape to fit our data.

Cost Function

Note: 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:

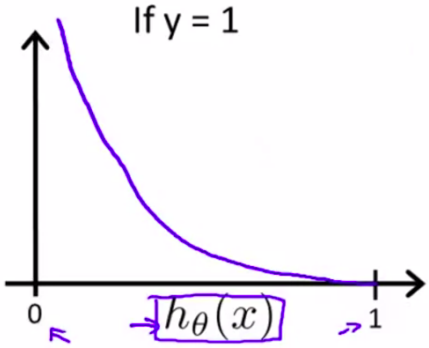
$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} Cost(h_\theta(x), y)$$

where

$$Cost(h_\theta(x), y) = \begin{cases} \log(h_\theta(x)), \text{ if } y = 1 \\ -\log(1 - h_\theta(x)), \text{ if } y = 0 \end{cases}$$

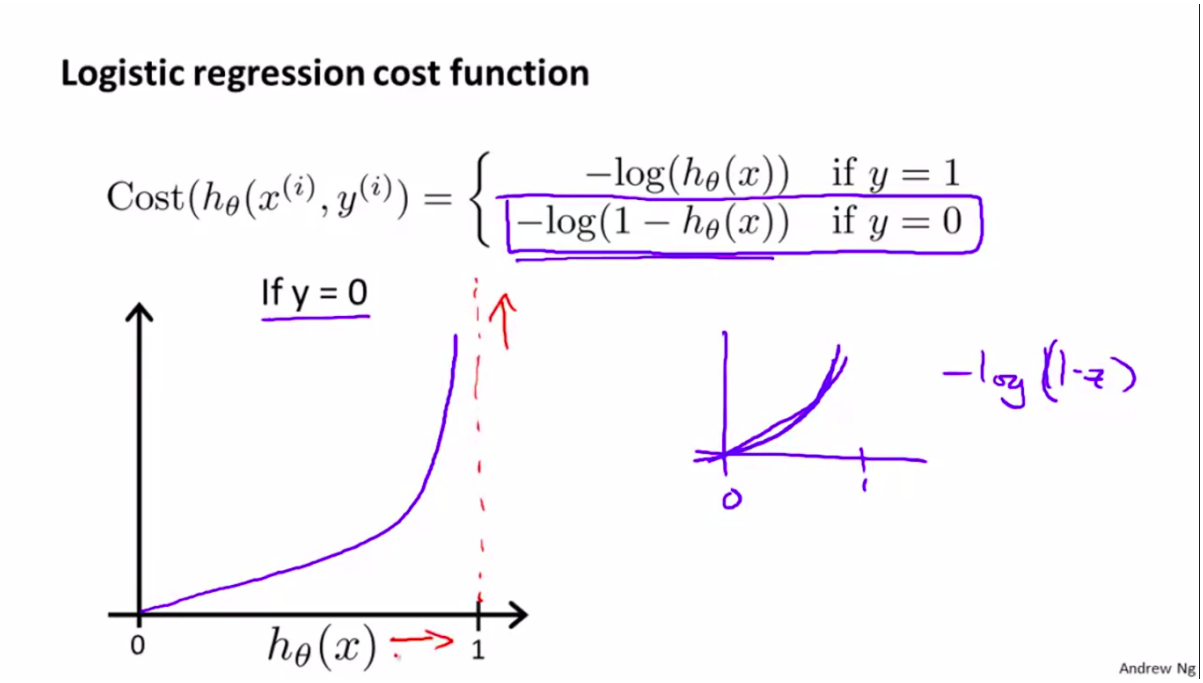
Logistic regression cost function

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



Cost $= 0$ if $y = 1, h_\theta(x) = 1$
But as $h_\theta(x) \to 0$
$Cost \to \infty$

Captures intuition that if $h_\theta(x) = 0$, (predict $P(y = 1|x; \theta) = 0$), but $y = 1$, we'll penalize learning algorithm by a very large cost.



Note:
$Cost(h_\theta(x), y) = 0, \text{ if } h_\theta(x) = y$
$Cost(h_\theta(x), y) \to \infty \text{ if } y = 0 \text{ and } h_\theta(x) \to 1$
$Cost(h_\theta(x), y) \to \infty \text{ if } y = 1 \text{ and } h_\theta(x) \to 0$

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

We can compress our cost function's two conditional cases into one case:

$$Cost(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x)) \tag{4}$$

Notice that when y is equal to 1, then the second term $(1 - y) \log(1 - h_\theta(x))$ will be zero and will not affect the result. If y is equal to 0, then the first term $-y \log(h_\theta(x))$ will be zero and will not affect the result.

We can fully write out our entire cost function as follows:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] \tag{5}$$

A vectorized implementation is:

$$h = g(X\theta) \tag{6}$$
$$J(\theta) = \frac{1}{m} \cdot \left(-y^T \log(h) - (1 - y)^T \log(1 - h) \right)$$

Gradient Descent

Remember that the general form of gradient descent is:

$Repeat\ \{$

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$

$\}$

We can work out the derivative part using calculus to get:

$Repeat\ \{$

$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$

$\}$

Notice that this algorithm is identical to the one we used in linear regression. We still have to simultaneously update all values in theta.

A vectorized implementation is:

$$\theta := \theta - \frac{\alpha}{m} X^T \left(g(X\theta) - y \right) \tag{7}$$

Advanced Optimization

"Conjugate gradient", "BFGS", and "L-BFGS" are more sophisticated, faster ways to optimize θ that can be used instead of gradient descent.

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

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

Indeed, we can write a single function that returns both of these:

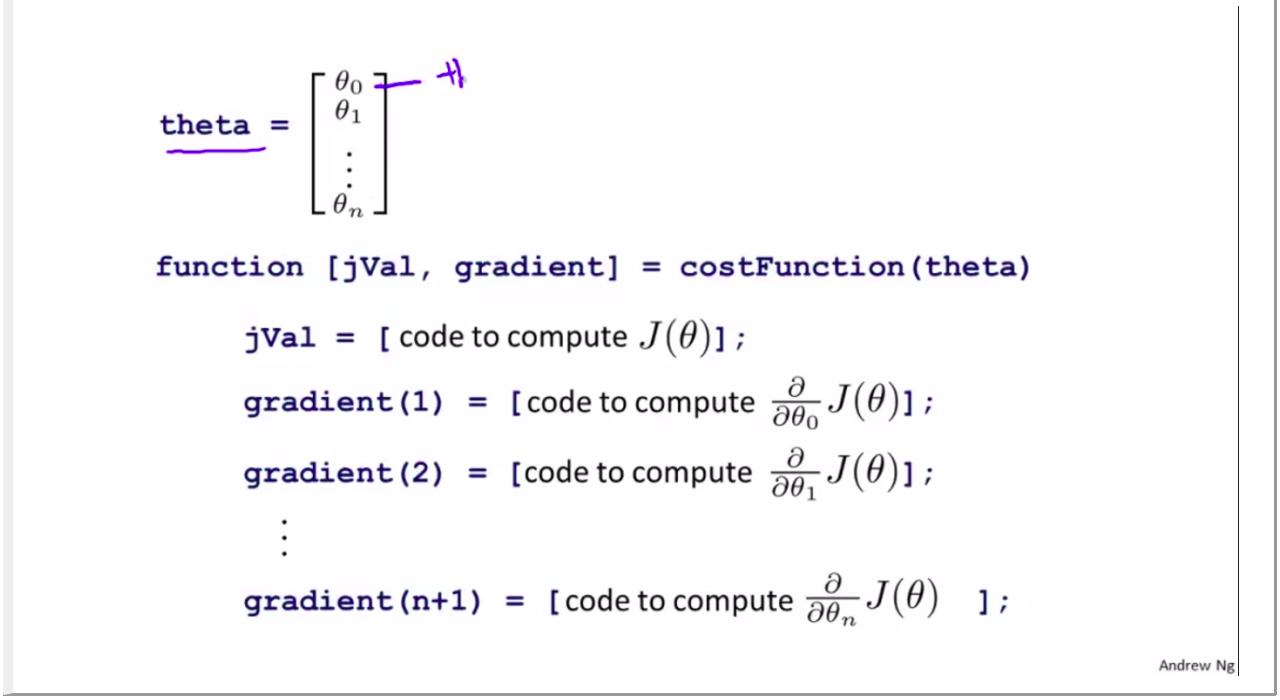
```
function [jVal, gradient] = costFunction(theta)
  jVal = [...code to compute J(theta)...];
  gradient = [...code to compute derivative of J(theta)...];
end
```

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()"

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

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

Example: (Logistic Regression)



Multiclass Classification: One-vs-All

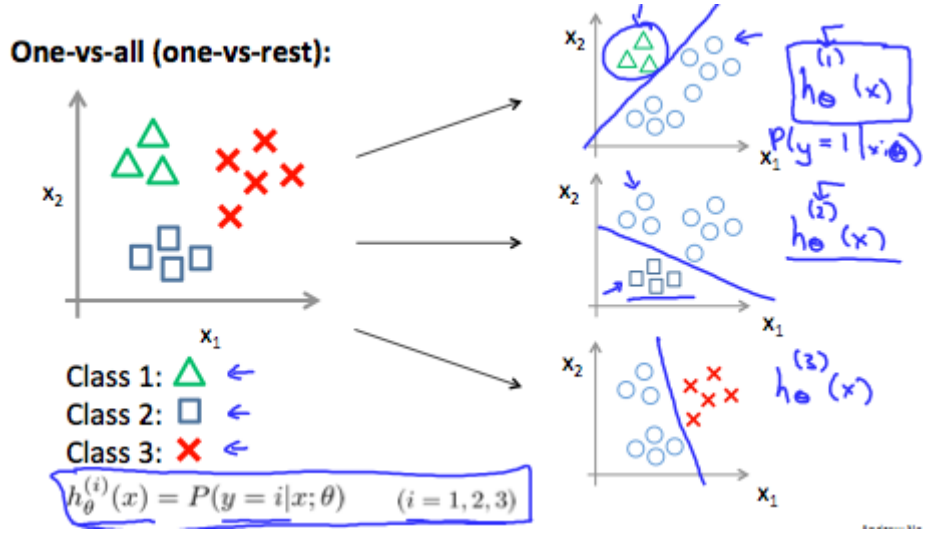
Now we will approach the classification of data when we have more than two categories. Instead of $y = \{0, 1\}$ we will expand our definition so that $y = \{0, 1...n\}$.

Since $y = \{0, 1...n\}$, we divide our problem into $n + 1$ ($+1$ because the index starts at 0) binary classification problems; in each one, we predict the probability that 'y' is a member of one of our classes.

$$
\begin{aligned}
&y \in \{0, 1, \dots, n\} \\
&h_\theta^{(0)}(x) = \mathcal{P}(y = 0 \mid x; \theta) \\
&h_\theta^{(1)}(x) = \mathcal{P}(y = 1 \mid x; \theta) \\
&\vdots \\
&h_\theta^{(n)}(x) = \mathcal{P}(y = n \mid x; \theta) \\
&\text{prediction} = \max_i \left(h_\theta^{(i)}(x) \right)
\end{aligned}
$$

We are basically choosing one class and then lumping all the others into a single second class. We do this repeatedly, applying binary logistic regression to each case, and then use the hypothesis that returned the highest value as our prediction.

The following image shows how one could classify 3 classes:



To sum up...
Train a logistic regression classifier $h_\theta(x)$ for each class j to predict the probability that $y = j$
To make a prediction on a new x, pick the class j that maximizes $h_\theta(x)$

References

[1] Machine Learning - Stanford University (https://www.coursera.org/learn/machine-learning)