

Machine Learning

Week 3

Section 1

Logistic Regression

Classification Problems

- Email: spam / not spam
- Online transactions: fraud
- Tumor: malignant / benign

Predict $y \{ 0 / 1 \} \{ \text{negative} / \text{positive} \} \{ \text{absent} / \text{present} \}$

y can also predict multiple values ex $\{ 0 \ 1 \ 2 \ 3 \}$

Logistic regression does not work well with classification problems because classification is not a linear function.

So we will develop logistic regression for a classification algorithm.

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 $x^{(i)}$ 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, $y \in \{0, 1\}$. 0 is also called the negative class, and 1 the positive class, and they are sometimes also denoted by the symbols “-” and “+.” Given $x^{(i)}$, the corresponding $y^{(i)}$ is also called the label for the training example.

Hypothesis Representation

$g(z)$ is the sigmoid function / logistic function (these terms are interchangeable)

$h(x)$ is the estimated probability that $y = 1$ on input x

If for some x , $h(x) = 0.7$, there is a 70% chance of outcome $y = 1$

On the flip side, 30% chance of outcome $y = 0$

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 $h_\theta(x)$ to take values larger than 1 or smaller than 0 when we know that $y \in \{0, 1\}$. To fix this, let's change the form for our hypotheses $h_\theta(x)$ to satisfy $0 \leq h_\theta(x) \leq 1$. This is accomplished by plugging $\theta^T x$ into the Logistic Function.

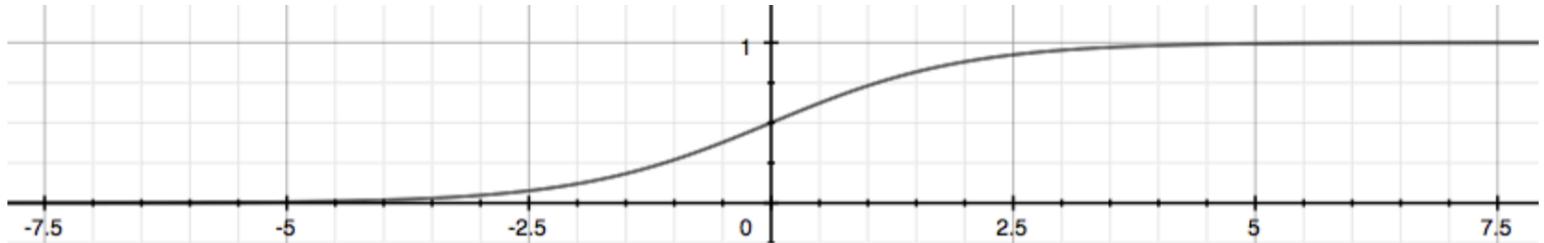
Our new form uses the "Sigmoid Function," also called the "Logistic Function":

$$h_{\theta}(x) = g(\theta^T x)$$

$$z = \theta^T x$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

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



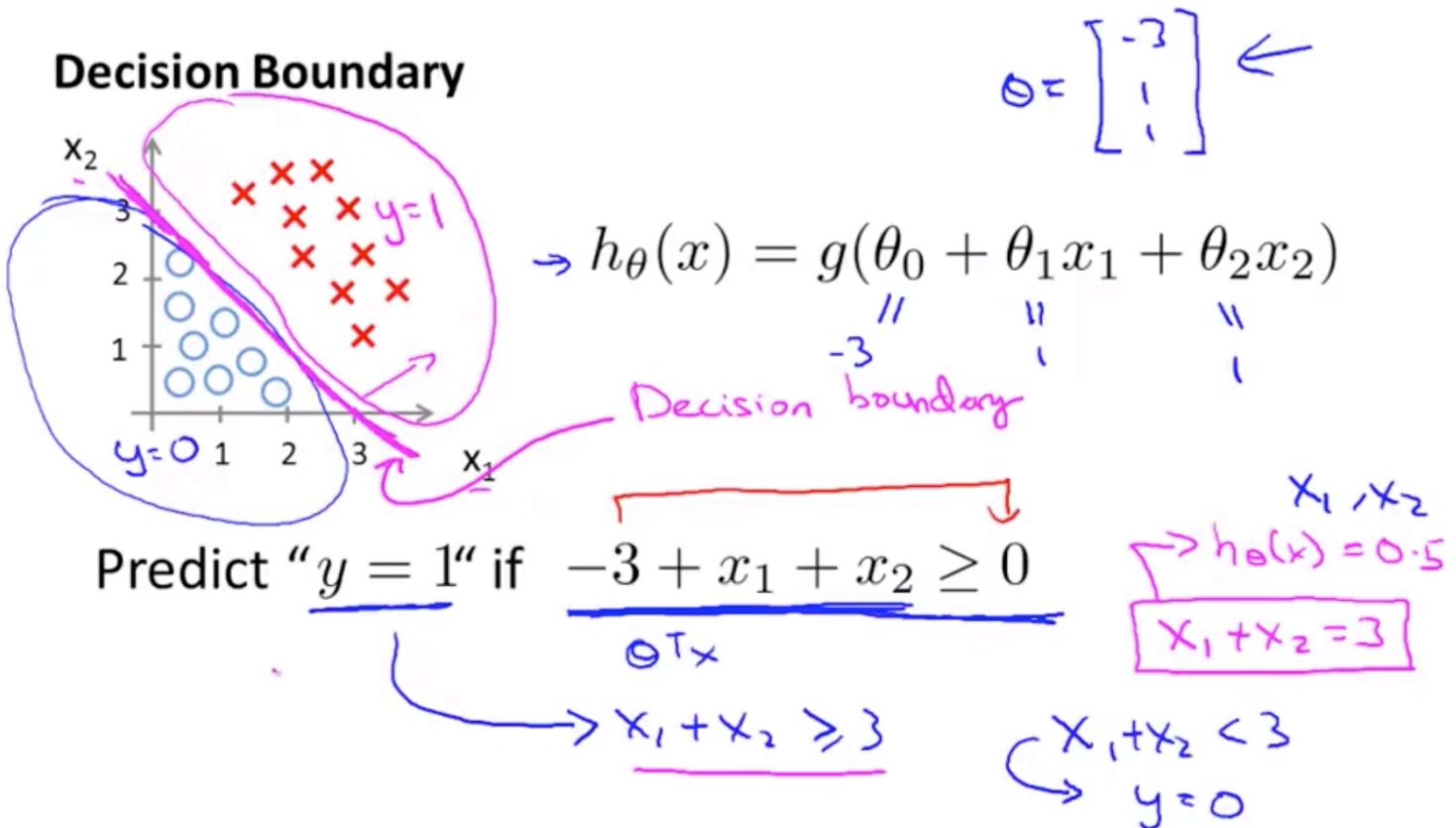
The function $g(z)$, shown here, maps any real number to the $(0, 1)$ interval, making it useful for transforming an arbitrary-valued function into a function better suited for classification.

$h_{\theta}(x)$ will give us the **probability** that our output is 1. For example, $h_{\theta}(x) = 0.7$ 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%).

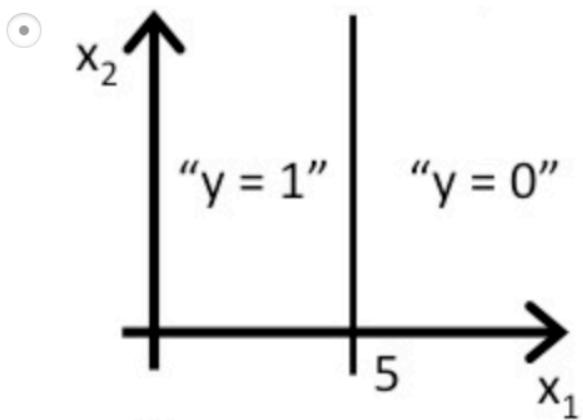
$$h_{\theta}(x) = P(y = 1|x; \theta) = 1 - P(y = 0|x; \theta)$$

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

Decision Boundary



Consider logistic regression with two features x_1 and x_2 . Suppose $\theta_0 = 5, \theta_1 = -1, \theta_2 = 0$, so that $h_\theta(x) = g(5 - x_1)$. Which of these shows the decision boundary of $h_\theta(x)$?



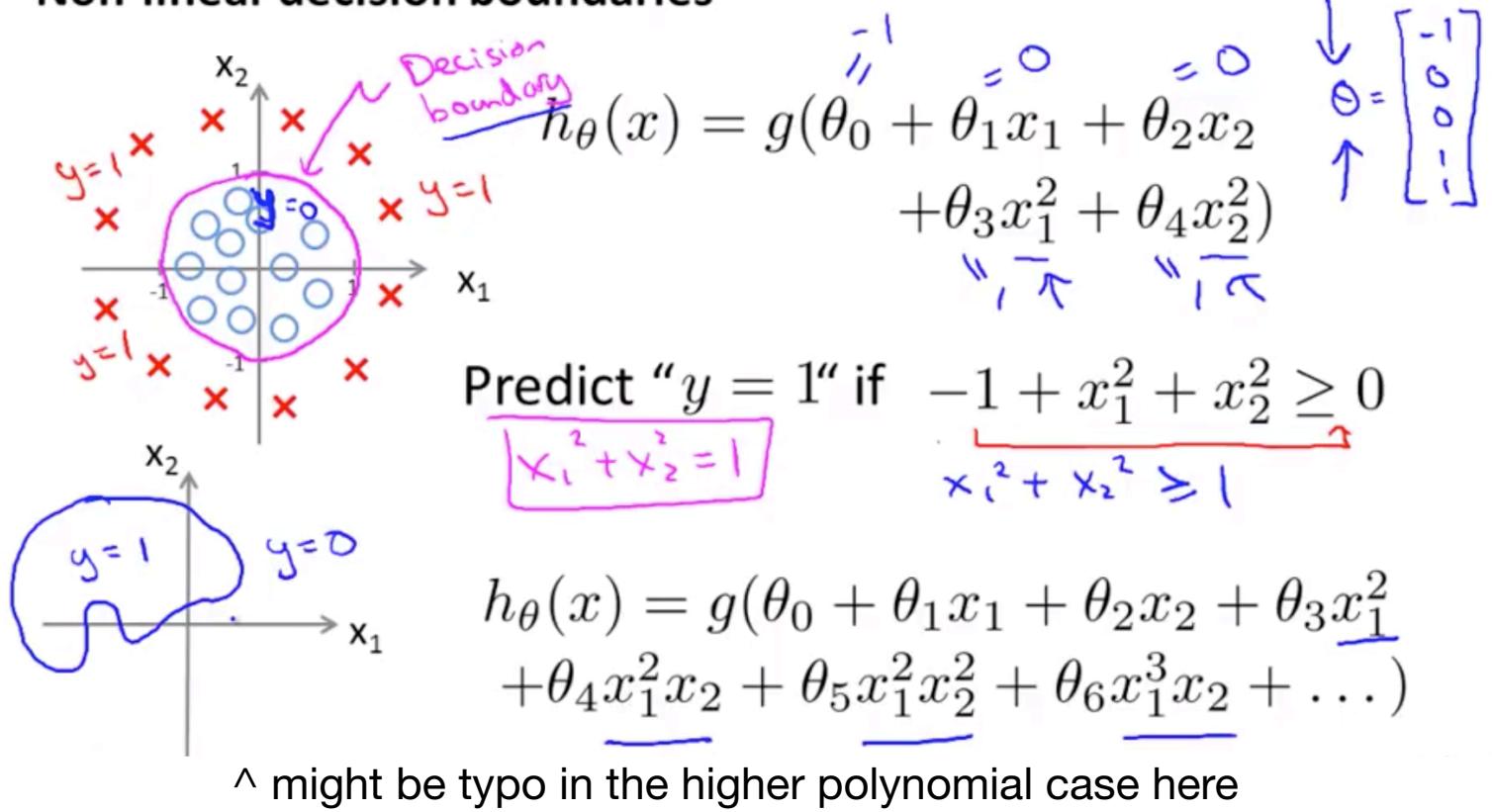
Correct

Predict $Y = 0$ if x_1 is greater than 5.

More complex case:

Non-linear decision boundaries

Non-linear decision boundaries



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

$$\begin{aligned} h_{\theta}(x) \geq 0.5 &\rightarrow y = 1 \\ h_{\theta}(x) < 0.5 &\rightarrow y = 0 \end{aligned}$$

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:

$$\begin{aligned} g(z) \geq 0.5 \\ \text{when } z \geq 0 \end{aligned}$$

Remember.

$$\begin{aligned} z = 0, e^0 = 1 &\Rightarrow g(z) = 1/2 \\ z \rightarrow \infty, e^{-\infty} \rightarrow 0 &\Rightarrow g(z) = 1 \\ z \rightarrow -\infty, e^{\infty} \rightarrow \infty &\Rightarrow g(z) = 0 \end{aligned}$$

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

$$\begin{aligned} h_{\theta}(x) = g(\theta^T x) \geq 0.5 \\ \text{when } \theta^T x \geq 0 \end{aligned}$$

From these statements we can now say:

$$\begin{aligned} \theta^T x \geq 0 &\Rightarrow y = 1 \\ \theta^T x < 0 &\Rightarrow y = 0 \end{aligned}$$

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:

$$\begin{aligned} \theta &= \begin{bmatrix} 5 \\ -1 \\ 0 \end{bmatrix} \\ y = 1 \text{ if } 5 + (-1)x_1 + 0x_2 &\geq 0 \\ 5 - x_1 &\geq 0 \\ -x_1 &\geq -5 \\ x_1 &\leq 5 \end{aligned}$$

In this case, our decision boundary is a straight vertical line placed on the graph where $x_1 = 5$, and everything to the left of that denotes $y = 1$, while everything to the right denotes $y = 0$.

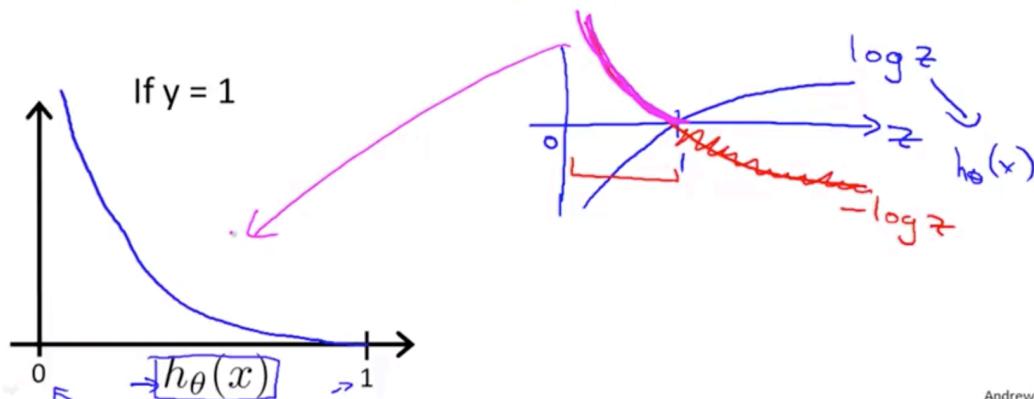
Again, the input to the sigmoid function $g(z)$ (e.g. $\theta^T X$) doesn't need to be linear, and could be a function that describes a circle (e.g. $z = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2$) or any shape to fit our data.

Cost Function

We need a new cost function for logistic regression

Logistic regression cost function

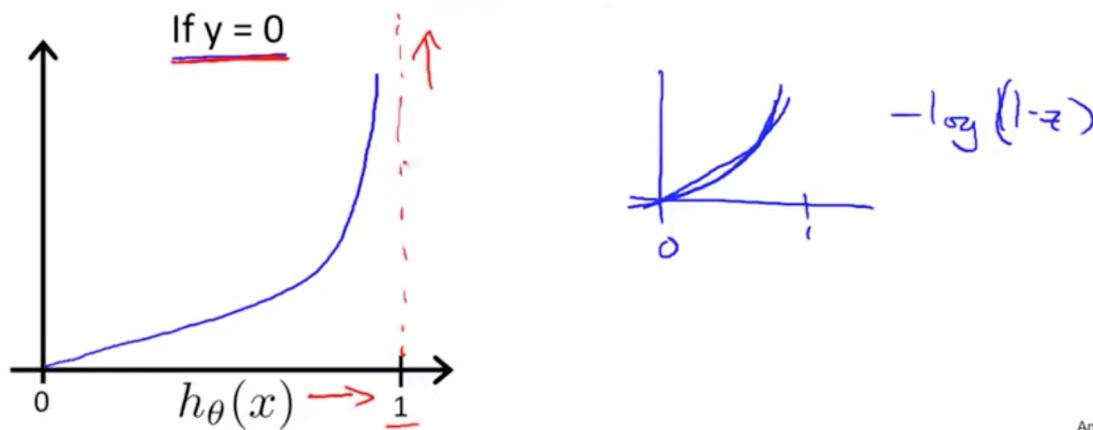
$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



Andrew ♪

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

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



Andr

Side note: We could use convexity analysis but that is beyond the scope of this course

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:

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

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

When $y = 1$, we get the following plot for $J(\theta)$ vs $h_\theta(x)$:

See the plots above

$$\begin{aligned}\text{Cost}(h_\theta(x), y) &= 0 \text{ if } h_\theta(x) = y \\ \text{Cost}(h_\theta(x), y) &\rightarrow \infty \text{ if } y = 0 \text{ and } h_\theta(x) \rightarrow 1 \\ \text{Cost}(h_\theta(x), y) &\rightarrow \infty \text{ if } y = 1 \text{ and } h_\theta(x) \rightarrow 0\end{aligned}$$

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 $J(\theta)$ is convex for logistic regression.

Simplified Cost Function and Gradient Descent

Logistic regression cost function

$$\begin{aligned}\rightarrow J(\theta) &= \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_\theta(x^{(i)}), y^{(i)}) \\ \rightarrow \text{Cost}(h_\theta(x), y) &= \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}\end{aligned}$$

Note: $y = 0$ or 1 always

We can rewrite the cost function as follows:

$$\begin{aligned}\rightarrow \text{Cost}(h_\theta(x), y) &= -\underbrace{(y \log(h_\theta(x)) + (1-y) \log(1-h_\theta(x)))}_{=0 \text{ if } y=0 \text{ or } 1} \\ \text{If } y=1: \text{Cost}(h_\theta(x), y) &= -\log(h_\theta(x)) \\ \text{If } y=0: \text{Cost}(h_\theta(x), y) &= -\log(1-h_\theta(x))\end{aligned}$$

So now our Logistic Regression Cost Function looks like this:

$$\begin{aligned}J(\theta) &= \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_\theta(x^{(i)}), y^{(i)}) \\ &= -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)})) \right]\end{aligned}$$

Same as before, now we want to find / fit the parameters theta to minimize J to make new predictions

We're going to use gradient descent again

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

Want $\min_{\theta} J(\theta)$:

Repeat {

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

}

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update all θ_j)

There is a typo below, there should be a $1/m$ term

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

Want $\min_{\theta} J(\theta)$:

$$\Theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \quad \text{for } i=0 \text{ to } n$$

Repeat {

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

}

(simultaneously update all θ_j)

$$h_\theta(x) = \theta^T x$$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Algorithm looks identical to linear regression!

One iteration of gradient descent simultaneously performs these updates:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)}$$

⋮

$$\theta_n := \theta_n - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_n^{(i)}$$

We would like a vectorized implementation of the form $\theta := \theta - \alpha \delta$ (for some vector $\delta \in \mathbb{R}^{n+1}$).

What should the vectorized implementation be?

Ⓐ $\theta := \theta - \alpha \frac{1}{m} \sum_{i=1}^m [(h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}]$

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

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

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:

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

A vectorized implementation is:

$$\begin{aligned} h &= g(X\theta) \\ J(\theta) &= \frac{1}{m} \cdot (-y^T \log(h) - (1 - y)^T \log(1 - h)) \end{aligned}$$

Gradient Descent

Remember that the general form of gradient descent is:

$$\begin{aligned} \text{Repeat } \{ \\ \theta_j &:= \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \\ \} \end{aligned}$$

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

$$\begin{aligned} \text{Repeat } \{ \\ \theta_j &:= \theta_j - \frac{\alpha}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \\ \} \end{aligned}$$

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:

$$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - \vec{y})$$

Advanced Optimization

This will allow us to get logistic regression run faster using gradient descent and be more efficient for larger datasets with more features

Gradient descent is one choice of optimization algorithm,
others include:

- Gradient Descent
- Conjugate Gradient
- BFGS
- L-BFGS

These are more sophisticated optimization algorithms, but are beyond the scope of this course

Some advantages: No need to pick learning rate alpha and is often faster, but more complex.

** Octave and Matlab have decent implementations one can use.

We can ask these programs to create a gradient and jVal and optimize the functions as follows:

Example: $\min_{\theta} J(\theta)$
 $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$ $\theta_1 = 5, \theta_2 = 5$.

$$\rightarrow J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$$

$$\rightarrow \frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$$

$$\rightarrow \frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$$

```
function [jVal, gradient] = costFunction(theta)
    jVal = (theta(1)-5)^2 + ...
            (theta(2)-5)^2;
    gradient = zeros(2,1);
    gradient(1) = 2*(theta(1)-5);
    gradient(2) = 2*(theta(2)-5);
```

options = optimset('GradObj', 'on', 'MaxIter', '100');

initialTheta = zeros(2,1);
[optTheta, functionVal, exitFlag] ...
= fminunc(@costFunction, initialTheta, options);

** initialTheta is an initial 'guess' (also needs to be 2-dimensional or greater)

How this might look in octave: [9:00]

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.

Multi-class Classification: One vs All

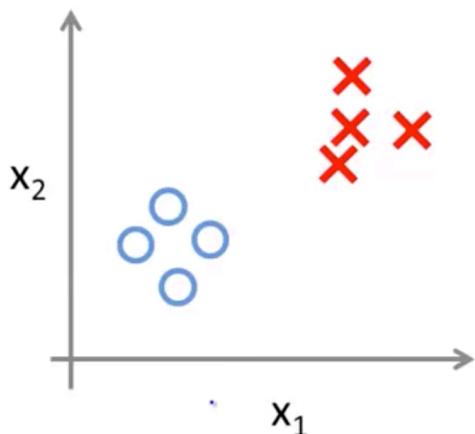
Also sometimes called One vs Rest

Examples of multi class classification

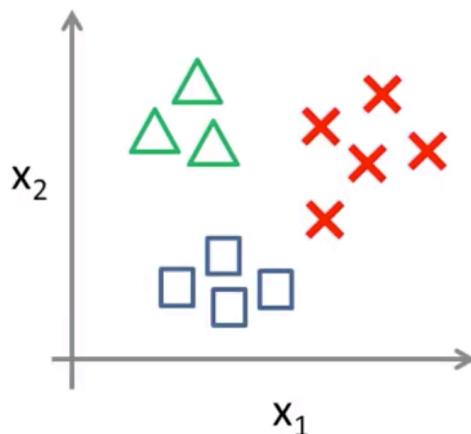
- Email tagging: work, friends, family, hobby
- Medical diagnosis: Not ill, cold, flu
- Weather: sunny, cloudy, rain, snow

I.e. y can take on a few discrete values $y = \{1, 2, 3, 4\}$

Binary classification:



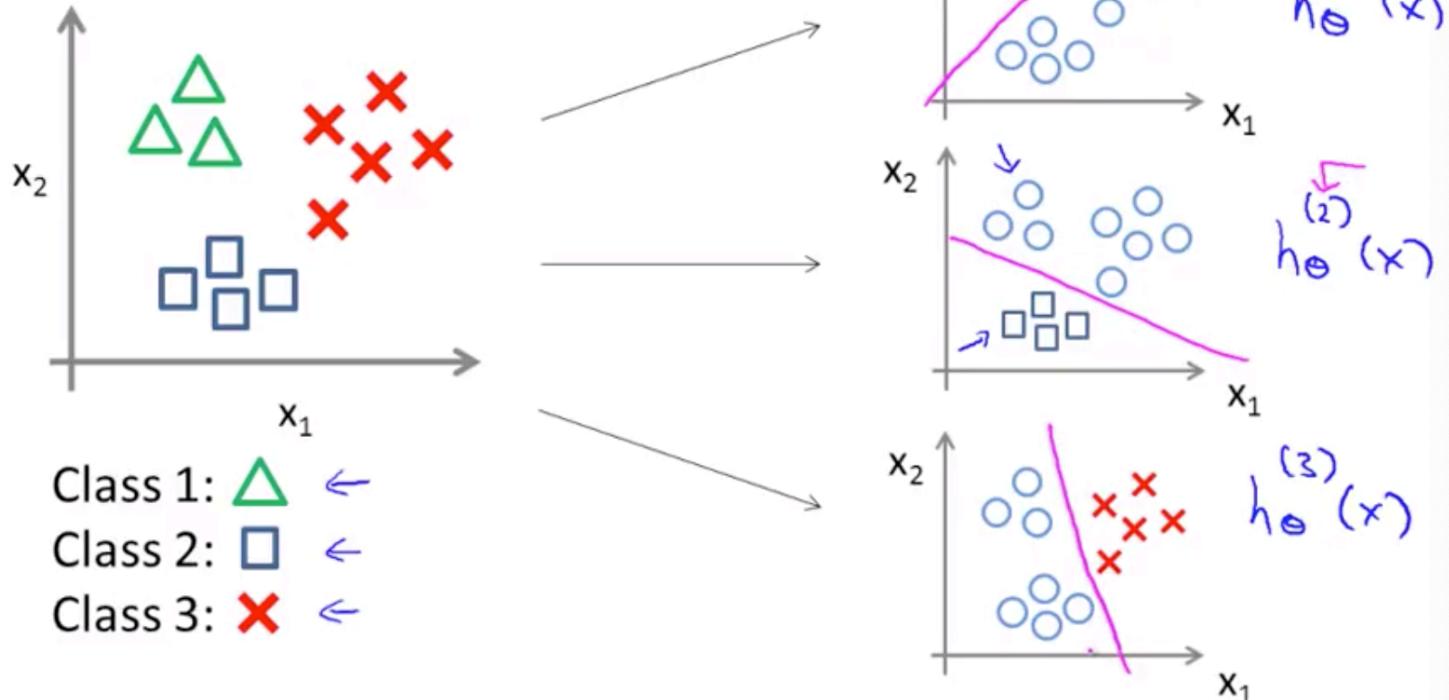
Multi-class classification:



We break it into three smaller problems:

Train a logistic regression classifier ' h ' for each class ' i ' to predict the probability that $y = i$

One-vs-all (one-vs-rest):



On a new input x , to make a prediction, pick the class ' i ' that maximizes h

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 $y = \{0,1\dots n\}$.

Since $y = \{0,1\dots n\}$, 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.

$$y \in \{0, 1\dots n\}$$

$$h_{\theta}^{(0)}(x) = P(y = 0|x; \theta)$$

$$h_{\theta}^{(1)}(x) = P(y = 1|x; \theta)$$

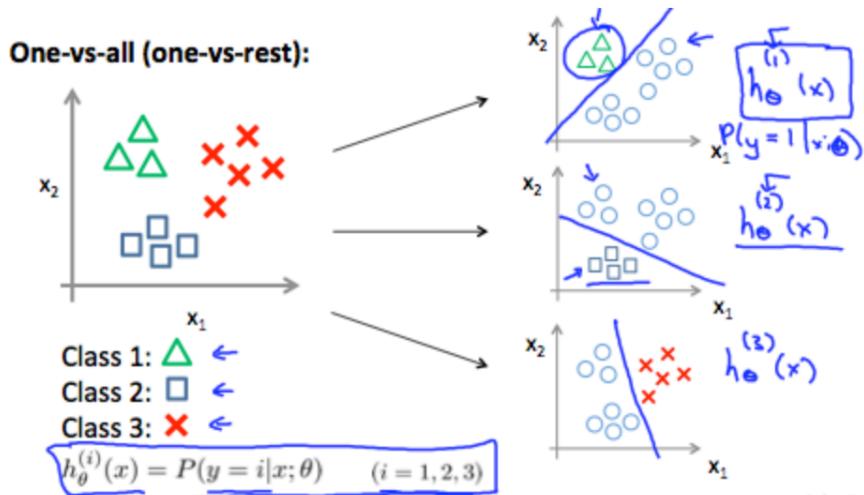
$$\dots$$

$$h_{\theta}^{(n)}(x) = P(y = n|x; \theta)$$

$$\text{prediction} = \max_i(h_{\theta}^{(i)}(x))$$

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 summarize:

Train a logistic regression classifier $h_{\theta}(x)$ for each class to predict the probability that $y = i$.

To make a prediction on a new x , pick the class that maximizes $h_{\theta}(x)$
