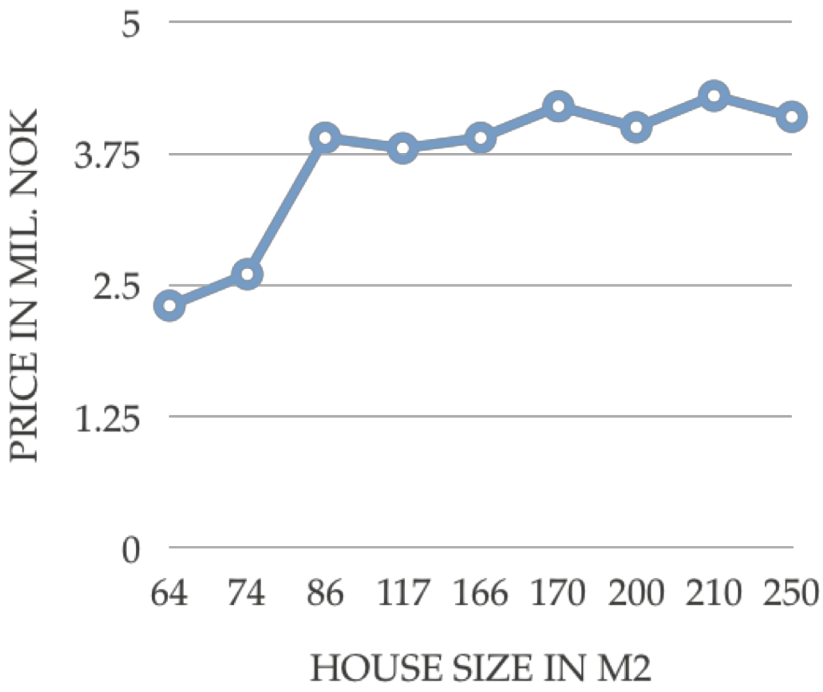
Data Mining

Classification: Logistic Regression

Regression Example

- Housing price data example
  - Supervised learning regression problem



Classification Hypothesis

- Given training data
  - Notation
    - m = number of **training examples**
    - x = input variables / features
    - y = output variable "target" variables
    - (x^i, y^i) - specific example (i^{th} training example)
- Pass into a learning algorithm
  - Algorithm outputs a function (denoted *h*) (h = **hypothesis**)
    - θ_i are **para**

$$h_\theta(x) = \theta_0 + \theta_1 x$$

adient

Linear regression - Parameters

- Choosing values for θ_i (parameters)
  - Different values give you different functions
  - If θ_0 is 1.5 and θ_1 is 0 then we get straight line parallel with X along 1.5 @ y
  - If θ_1 is > 0 then we get a positive slope
- Based on our training set we want to generate parameters which makes the straight line

Linear regression – Cost Function

- A cost function lets us figure out how to fit the best straight line to our data
- We want to solve a **minimization problem**
  - Minimize $(h_\theta(x) - y)^2$
  - i.e. minimize the difference between h(x) and y for each/any/every example
  - Sum this over the training set

$$J(\theta_1, \theta_2) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2$$

- **Hypothesis -** is like your prediction machine, throw in an *x* value, get a putative *y* value
- **Cost -** is a way to, using your training data, determine values for your θ values which make the hypothesis as accurate as possible

$$\text{Minimize } J(\theta_1, \theta_2)$$

  - This cost function is also called the squared error cost functionThis cost function is reasonable choice for most regression functions
  - Probably most commonly used function

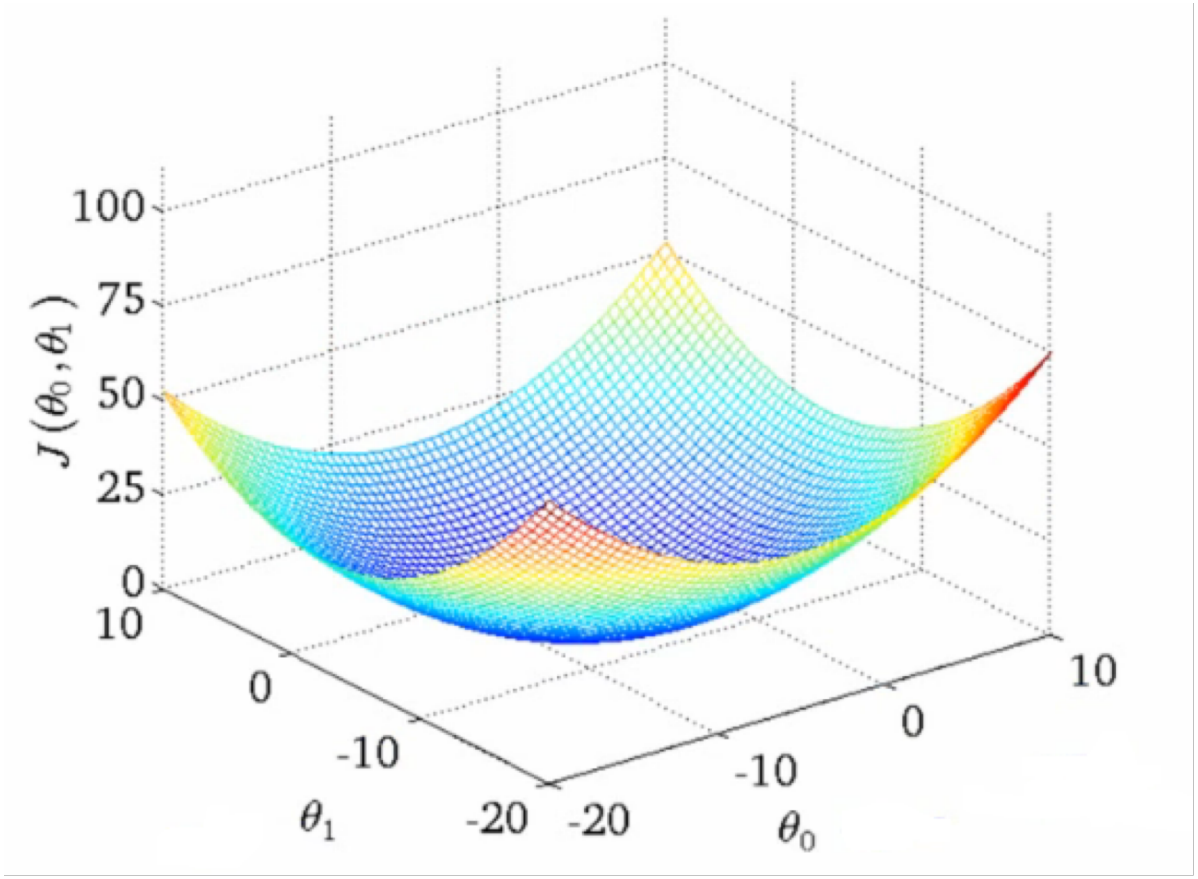
Cost function - a deeper look

- Lets consider some intuition about the cost function and why we want to use it
  - The cost function determines parameters
  - The value associated with the parameters determines how your hypothesis behaves, with different values generate different
- Simplified hypothesis
  - Assumes $\theta_0 = 0$

$$h_\theta(x) = \theta_1 x$$

Cost function - a deeper look

Minimize $J(\theta_1, \theta_2) = \frac{1}{m} \sum_{i=1}^{m} (\theta_1 x^i - y^i)^2$


100
75
50
25
0
$J(\theta_0, \theta_1)$
10
0
-10
-20
θ_1
-20
-10
0
10
θ_0

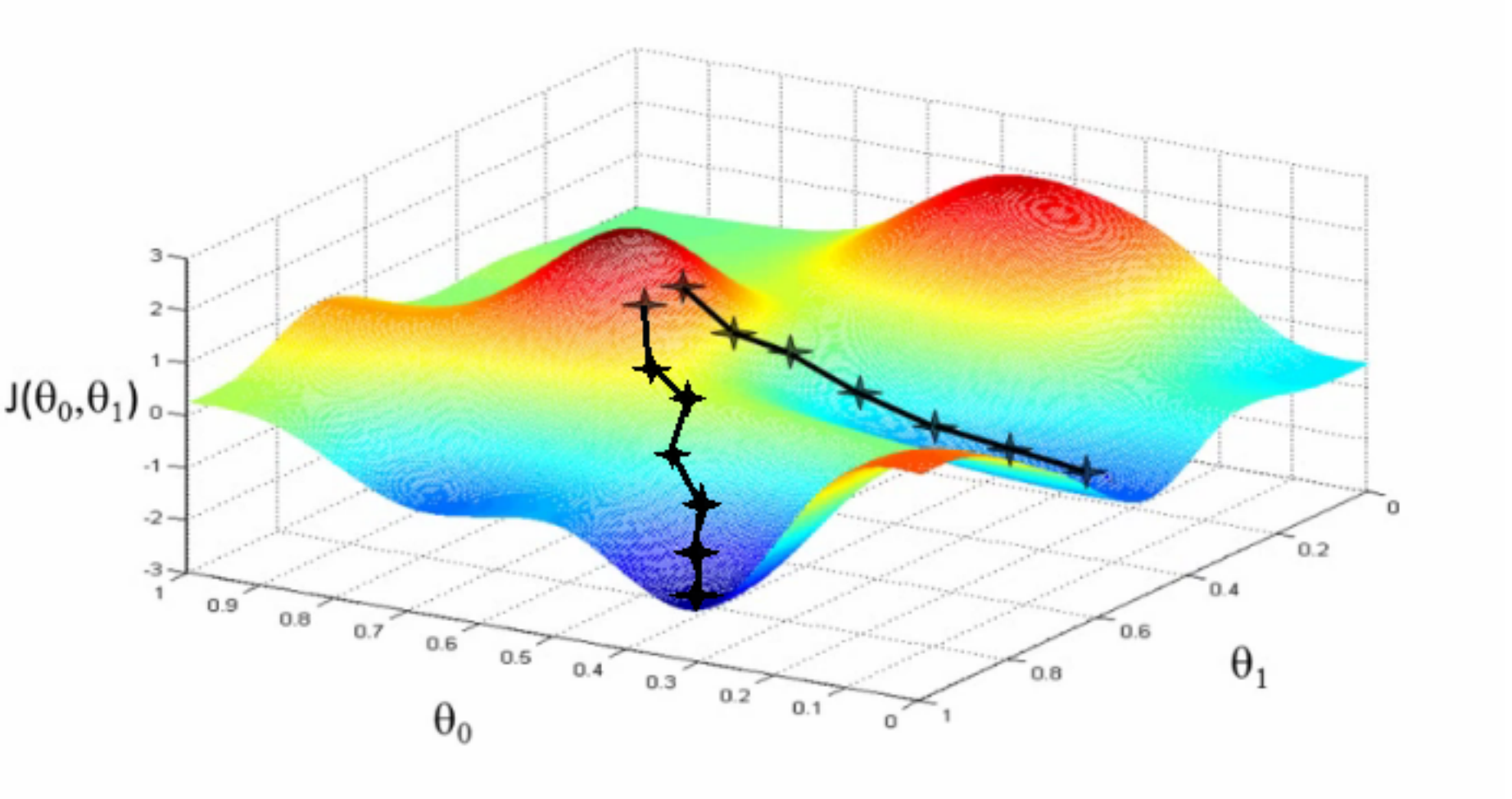
Gradient Descent (GD) algorithm

- Minimize cost function J
- Gradient descent
  - Used widely in machine learning for minimization
- Start by looking at a general J() function
- Problem
  - We have $J(\theta_0, \theta_1)$
  - We want to **minimize $J(\theta_0, \theta_1)$**
- Gradient descent applies to more general functions $J(\theta_0, \theta_1, \theta_2 \ldots \theta_n)$ min $J(\theta_0, \theta_1, \theta_2 \ldots \theta_n)$

How does GD Work?

- Start with initial guessesStart at 0,0 (or any other random value)
- Keeping changing θ_0 and θ_1 a little bit to try and reduce $J(\theta_0,\theta_1)$
- Each time you change the parameters, you select the gradient which reduces $J(\theta_0,\theta_1)$ the most
- Repeat
- Do so until you converge to a local minimum
- Has an interesting property
  - Where you start can determine which minimum you end up

GD Example



Formal Definition of GD

- Do the following until convergence

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \text{(for } j = 0 \text{ and } j = 1\text{)}$$

- Update θ_j by setting it to (θ_j - α) times the partial derivative of the cost function with respect to θ_j
- α (alpha) Is a number called the **learning rate**
- Controls how big a step you take
  - If α is big have an aggressive gradient descent
  - If α is small take tiny steps

GD Update

- For j = 0 and j = 1 means we **simultaneously** update both

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\theta_1 := \text{temp1}$$

- non-simultaneous update it's not gradient descent, and will behave weirdly
  - But it might look sort of right - so it's important to remember this!

Partial Derivatives

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_2) = \frac{\partial}{\partial \theta_0} \frac{1}{2m} \sum_{i=1}^{m} (\theta_1 x^i + \theta_0 - y^i)^2$$

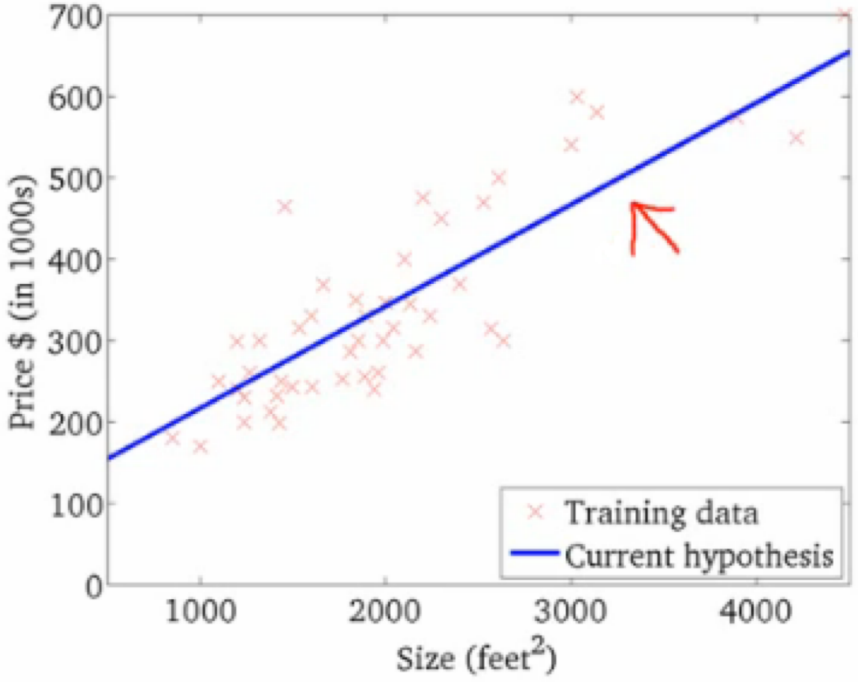
$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_1} \frac{1}{2m} \sum_{i=1}^{m} (\theta_1 x^i + \theta_0 - y^i)^2$$

- How does it workRisk of meeting different local optimum
- The linear regression cost function is always a **convex function -** always has a single minimum
  - Bowl shaped
  - One global optima
  - So gradient descent will always converge to global optima

GD Example

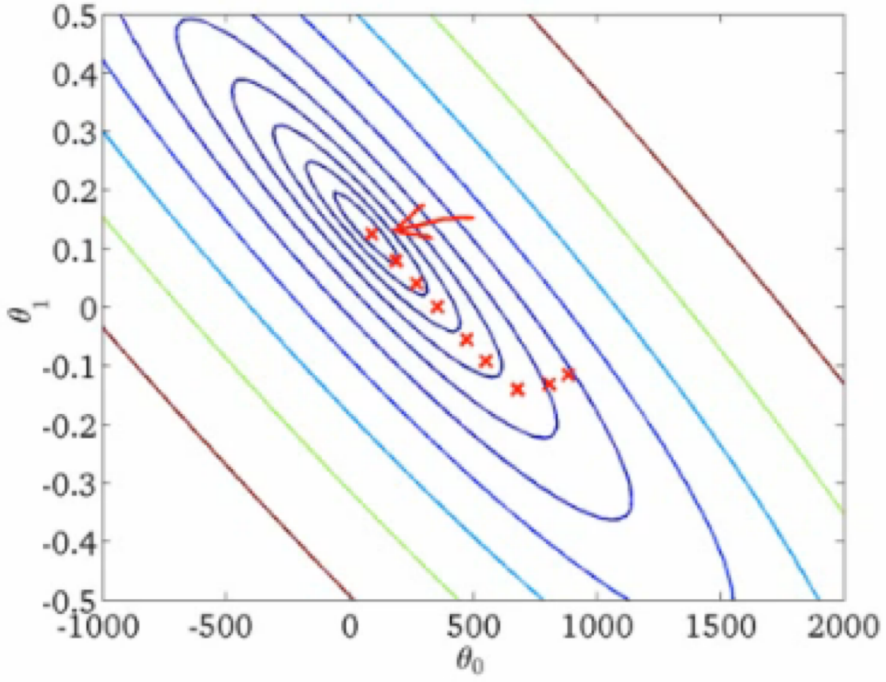
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



GD Summary

- This is actually **Batch Gradient Descent**
- Refers to the fact that over each step you look at all the training data
  - Each step compute over m training examples
- Sometimes non-batch versions exist, which look at small data subsets
  - We'll look at other forms of gradient descent (to use when m is too large) later in the course

Multiple Variables

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

GD for multiple variables

- Fitting parameters for the hypothesis with gradient descent
  - Parameters are θ_0 to θ_n
  - Instead of thinking about this as n separate values, think about the parameters as a single vector (θ)
    - Where θ is n+1 dimensional
- Our cost function is

$$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Multivariate GD Update

- Similarly, instead of thinking of J as a function of the n+1 numbers,
  - J() is just a function of the parameter vector

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$$

}

(simultaneously update for every $j = 0, \ldots, n$)

GD Steps

Repeat $\{$

$$\theta_0 := \theta_0 - \alpha \underbrace{\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial \theta_0} J(\theta)}$$

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

(simultaneously update θ_0, θ_1) $\}$

Parameter Updates

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

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

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

Multivariate GD update step

New algorithm $(n \geq 1)$:

Repeat $\{$

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

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

(simultaneously update θ_j for $j = 0, \ldots, n$) $\}$

Explanation

- In non-calculus words, this means that we do
  - Learning rate
  - Times 1/m (makes the maths easier)
  - Times the sum of
    - The hypothesis taking in the variable vector, minus the actual value, times the j-th value in that variable vector for EACH example
- It's important to remember that

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

Gradient Descent in Practice

- Feature scaling
  - If you have a problem with multiple features
  - Means gradient descent will converge more quickly
  - e.g.
    - x1 = size (0 - 2000 feet)
    - x2 = number of bedrooms (1-5)
    - Means the contours generated if we plot θ_1 vs. θ_2 give a very tall and thin shape due to the huge range difference
- Running gradient descent on this kind of cost function can take a long time to find the global minimum

Feature Scaling

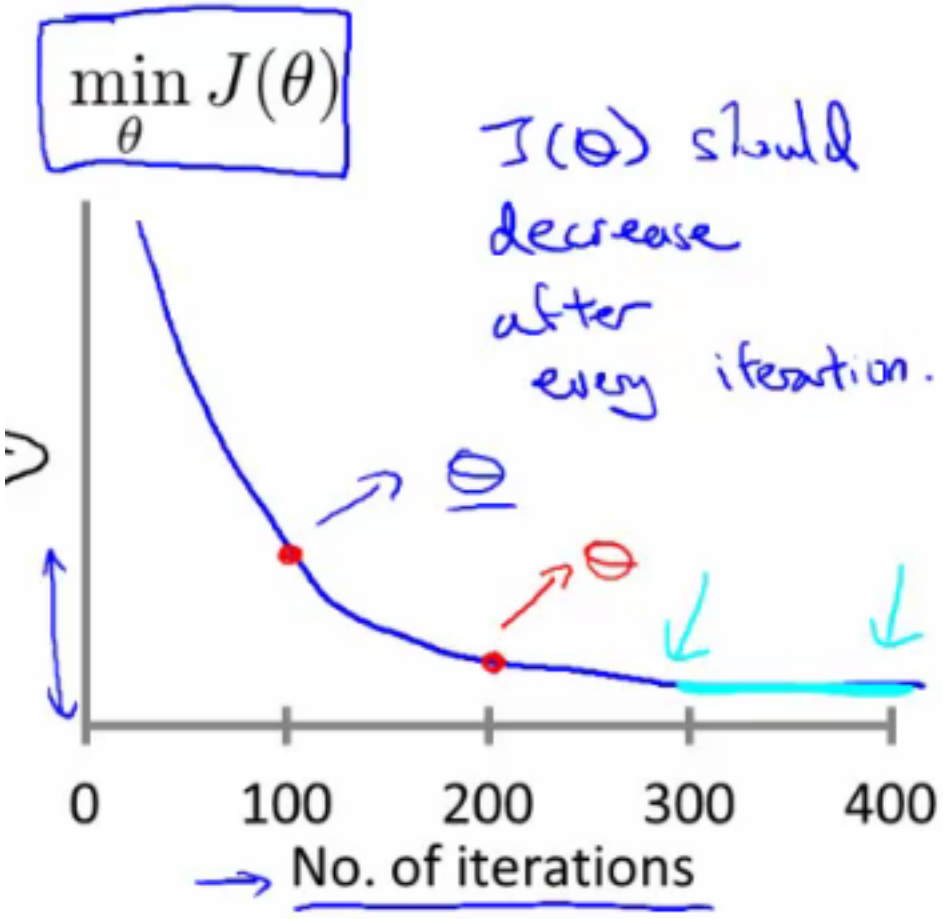
- Pathological input to gradient descent
  - So we need to rescale this input so it's more effective
  - So, if you define each value from x1 and x2 by dividing by the max for each feature
  - Contours become more like circles (as scaled between 0 and 1)

Mean Normalization

- May want to get everything into -1 to +1 range (approximately)
  - Want to avoid large ranges, small ranges or very different ranges from one another
  - Rule a thumb regarding acceptable ranges
    - -3 to +3 is generally fine - any bigger bad
    - -1/3 to +1/3 is ok - any smaller bad
- **mean normalization**
  - Take a feature x_i
  - Replace it by $(x_i - \text{mean})/(\text{max}-\text{min})$
  - So your values all have an average of about 0

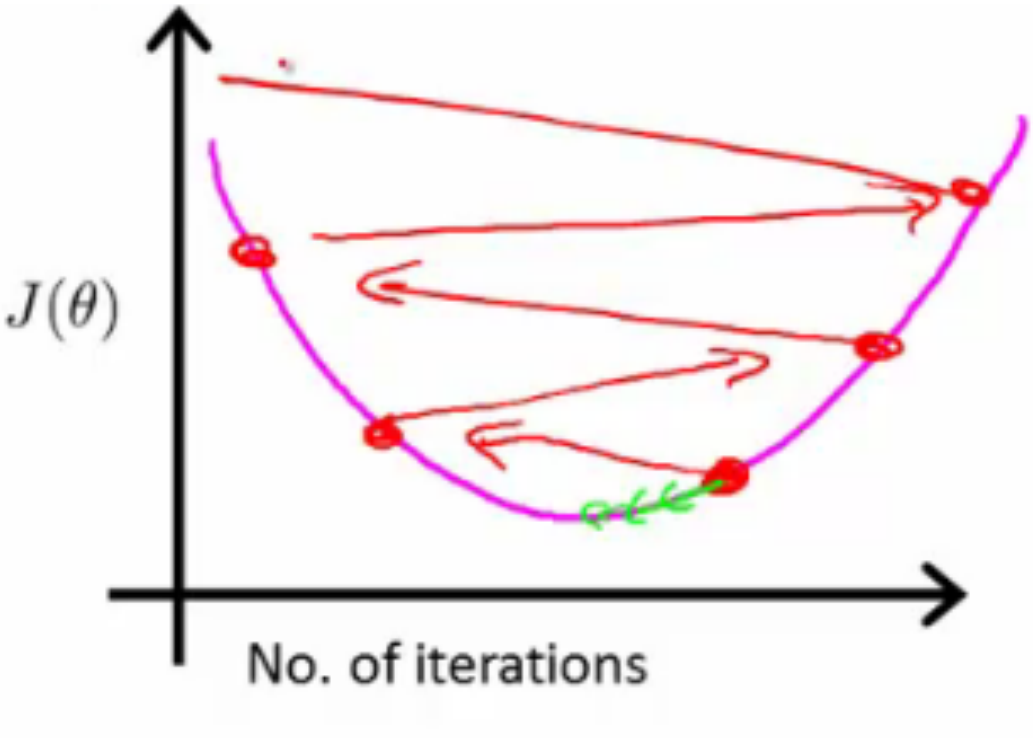
Practical Tips

- Plot min J(θ) vs. no of iterations
  - (i.e. plotting J(θ) over the course of gradient descent
- If gradient descent is working then J(θ) should decrease after every iteration
- Can also show if you're not making huge gains after a certain number



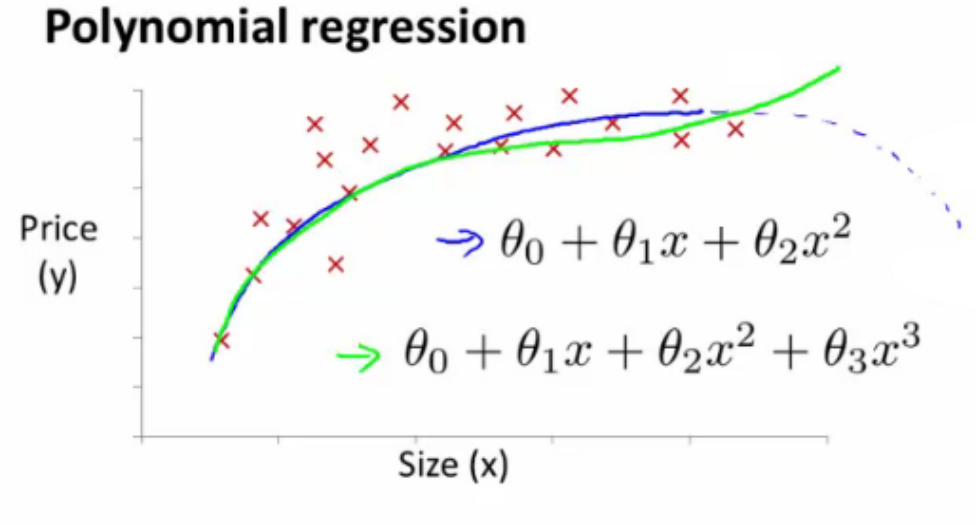
Learning Rate

- reduce learning rate so you actually reach the minimum (green line)
- use a smaller α
- f α is too small then rate is too slow
  - Slow convergence



•Go for roughly threefold increases0.001, 0.003, 0.01, 0.03. 0.1, 0.3

Polynomial regression



- May fit the data better
- $\theta_0 + \theta_1 x + \theta_2 x^2$ e.g. here we have a quadratic function
- For housing data could use a quadratic function
  - But may not fit the data so well
  - So instead must use a cubic function

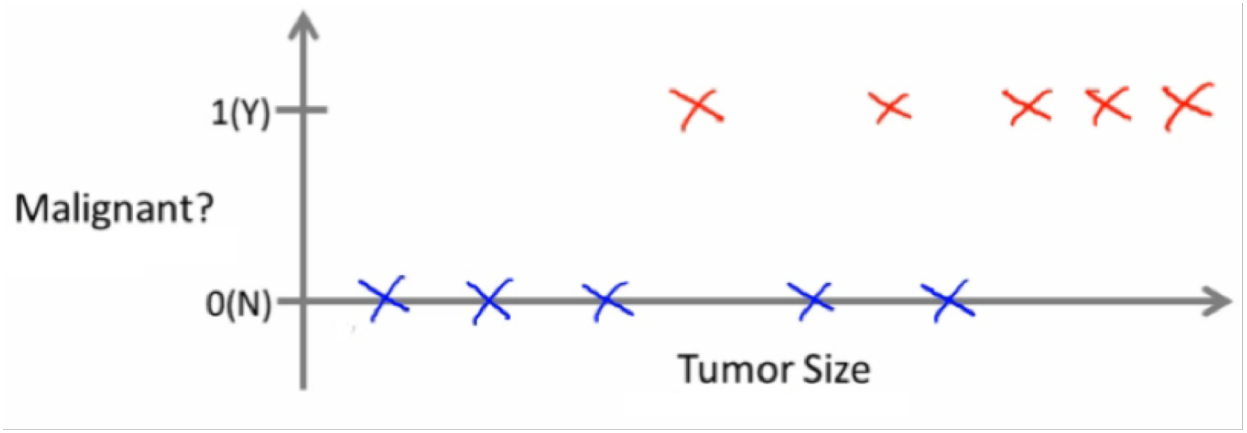
Logistic Regression

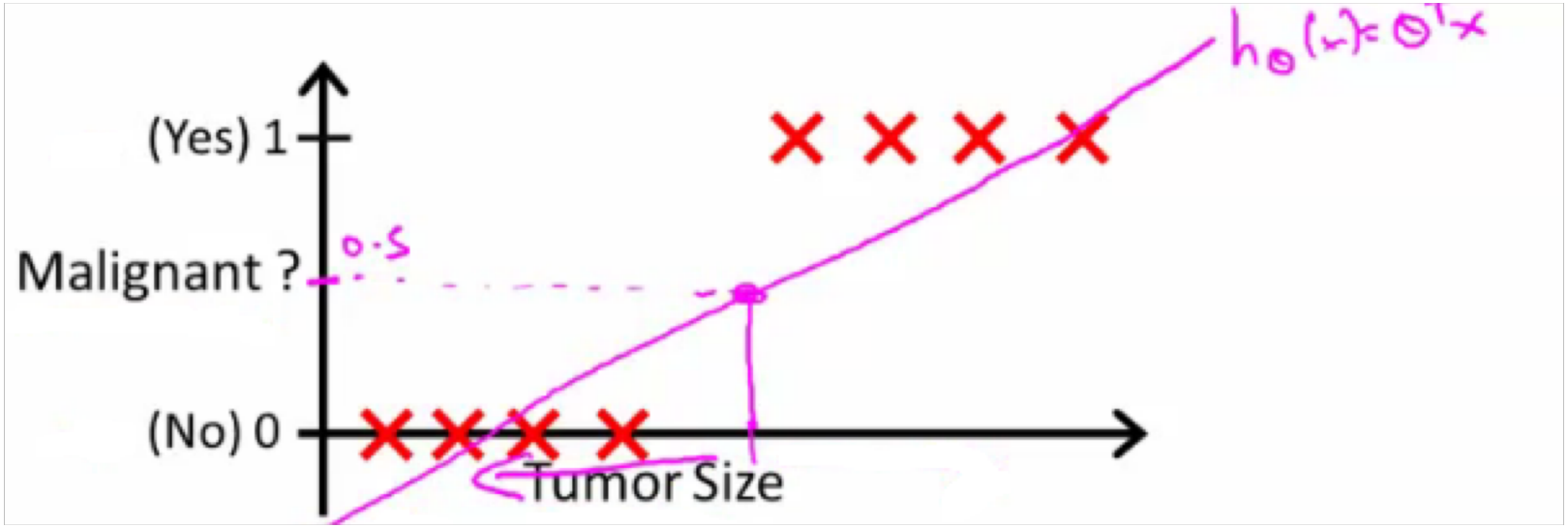
Binary Classificaiton Problem

- Where y is a discrete value
- Examples:
  - Email -> spam/not spam?
  - Online transactions -> fraudulent?
  - Tumor -> Malignant/benign
- Variable in these problems is YY is either 0 or 1
  - 0 = negative class (absence of something)
  - 1 = positive class (presence of something)

Can we use Linear Regression?

- Can we use linear regression for binar classificaiton?




(Yes) 1
Malignant ?
0.5
(No) 0
Tumor Size
$h_\theta(x) = \theta^T x$

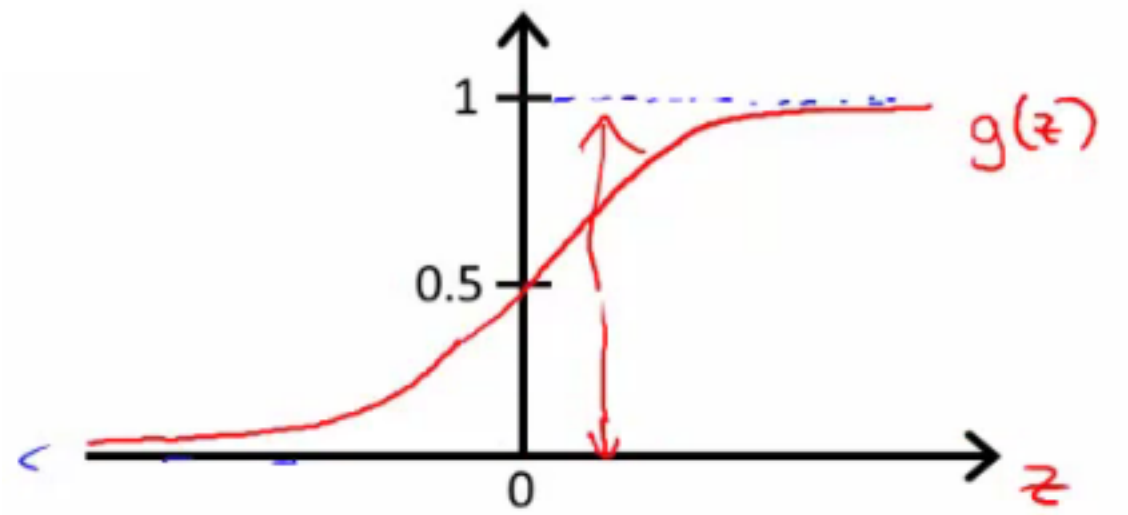
Hypothesis Representation

- What function is used to represent our hypothesis in classification
- We want our classifier to output values between 0 and 1
- When using linear regression we did $h_\theta(x) = (\theta^T x)$
- For classification hypothesis representation we do $h_\theta(x) = g((\theta^T x))$
- $g(z) = 1/(1 + e^{-z})$
- This is the **sigmoid function**, or the **logistic function**

Sigmoid Function

- Given this we need to fit θ to our data

$$\frac{1}{1+e^{\theta^T X}}$$



Hypothesis to Probability

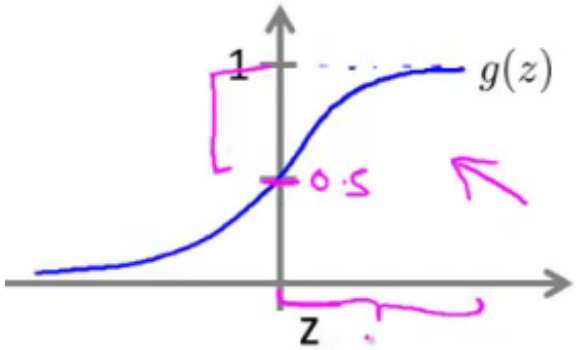
- When our hypothesis ($h_\theta(x)$) outputs a number, we treat that value as the estimated probability that y=1 on input x
- Since this is a binary classification task we know y = 0 or 1So the following must be true
  - $P(y=1|x;\theta) + P(y=0|x;\theta) = 1$
  - $P(y=0|x;\theta) = 1 - P(y=1|x;\theta)$

Decision boundary

- Gives a better sense of what the hypothesis function is computing
  - Better understand of what the hypothesis function looks like
  - One way of using the sigmoid function is;
  - When the probability of y being 1 is greater than 0.5 then we can predict y = 1
  - Else we predict y = 0
- When is it exactly that $h_\theta(x)$ is greater than 0.5?
  - Look at sigmoid function
    - g(z) is greater than or equal to 0.5 when z is greater than or equal to 0

Decision Boundary of Sigmoid

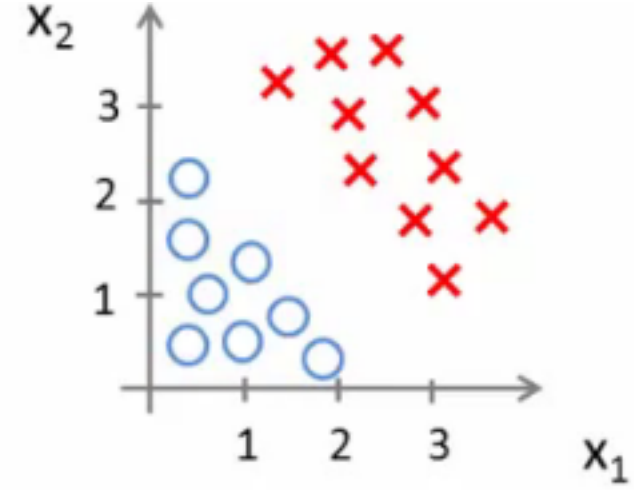
- If z is positive, g(z) is greater than 0.5
  - $z = (\theta^T x)$
  - So when $\theta^T x >= 0$
    - Then $h_\theta >= 0.5$



- So what we've shown is that the hypothesis predicts y = 1 when $\theta^T x >= 0$ The corollary of that when $\theta^T x <= 0$ then the hypothesis predicts y = 0
- Let's use this to better understand how the hypothesis makes its predictions

Decision Boundary Example

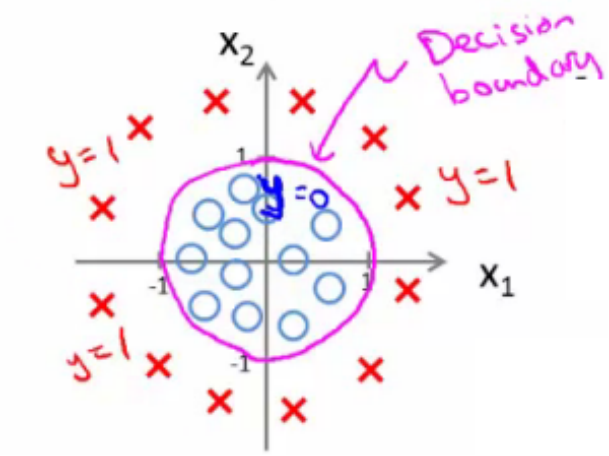
- Given $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
- Say $\theta_0 = -3, \theta_1 = 1, \theta_2 = 1$
- The z here becomes $\theta^T x$
- We predict "y = 1" if
  - $-3x_0 + 1x_1 + 1x_2 >= 0$
  - $-3 + x_1 + x_2 >= 0$



Non-linear decision boundaries

- Get logistic regression to fit a complex non-linear data set
  - Like polynomial regression add higher order terms
  - $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_3 x_1^2 + \theta_4 x_2^2)$
  - We take the transpose of the θ vector times the input vector
    - Say θ^T was [-1,0,0,1,1] then we say;
    - Predict that "y = 1" *if*
      - $-1 + x_1^2 + x_2^2 >= 0$

        or
      - $x_1^2 + x_2^2 >= 1$
    - If we plot $x_1^2 + x_2^2 = 1$
      - This gives us a circle with a radius of 1 around 0



Cost function for logistic regressior

- Fit θ parameters
- Define the optimization object for the cost function we use the fit the parameters
  - Training set of *m* training examples
    - Each example has is n+1 length column vector

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$

m examples $x \in \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix}$ $x_0 = 1, y \in \{0, 1\}$

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

Linear Regression Cost Function

- Linear regression uses the following function to determine θ

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

- Lets simplify $cost(h_\theta(x^i), y) = 1/2(h_\theta(x^i) - y^i)^2$
- We can **redefine J(θ) as**

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

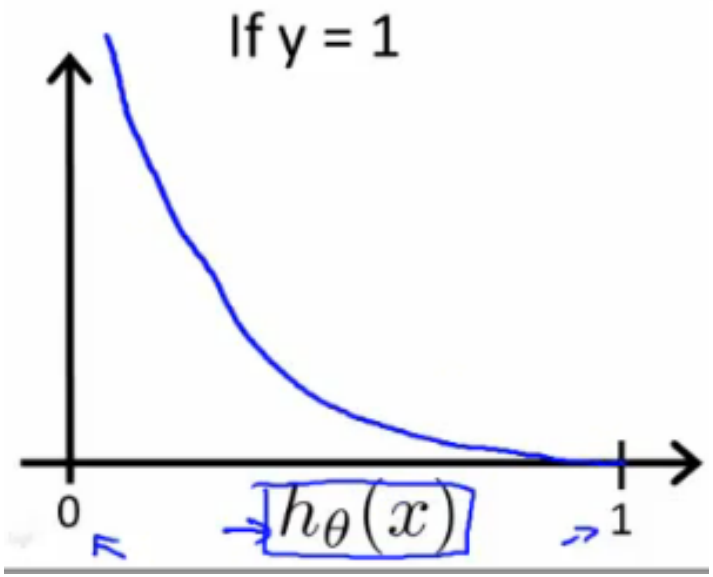
Problem with LR cost function

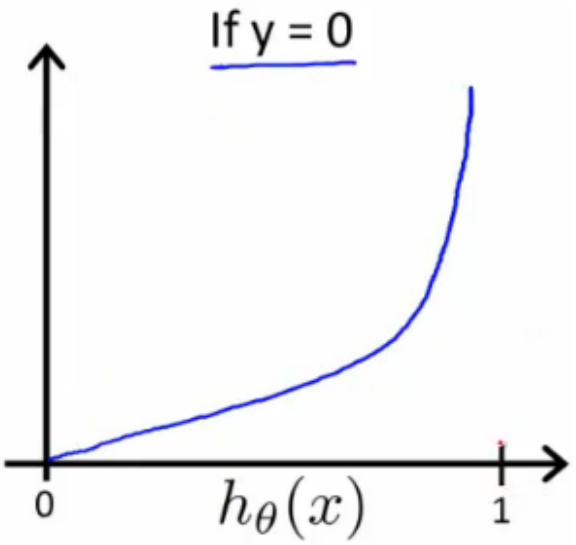
- This is a **non-convex function** for parameter optimization
- Lots of local minima mean gradient descent may not find the global optimum - may get stuck in a global minimum
- We would like a convex function so if you run gradient descent you converge to a global minimum

A convex cost function

- This is our logistic regression cost function

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





Simplified cost function

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

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

Note: $y = 0$ or 1 always

- **cost(h_θ(x),y) = -ylog(h_θ(x)) - (1-y)log(1-h_θ(x))**

Simplified cost function

- y = 1 Then our equation simplifies to
  - $-\log(h_\theta(x)) - (0)\log(1 - h_\theta(x))$
    - $-\log(h_\theta(x))$
    - Which is what we had before when y = 1
- y = 0 Then our equation simplifies to

  - $-(0)\log(h_\theta(x)) - (1)\log(1 - h_\theta(x))$
  - $= -\log(1- h_\theta(x))$
  - Which is what we had before when y = 0

Simplified cost function

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

- Why do we chose this function when other cost functions exist?
- This cost function can be derived from statistics using the principle of **maximum likelihood estimation**
- Also has the nice property that it's convex

Gradient Descent for Logistic Regression

- Use gradient descent as before
- Repeatedly update each parameter using a learning rate

Repeat {

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

(simultaneously update all θ_j)

}

Literature

- An Introduction to Data Mining, 1st edition, Pang-Ning Tan, Michael Steinbach, Vipin Kumar, Anuj Karpatne, Pearson Appendix D
- Machine learning by Andrew Ng
  - https://www.youtube.com/playlist?list=PLLssT5z_DsK-h9vYZkQkYNWcItqhlRJLN
  - Chapters 4 and 6