1. Week 1
   1. Introduction
      1. Welcome
         * Machine learning grew out of work in Artificial Intelligence
         * new capability for computers
         * ex:
           + database mining

large datasets from growth of automation/web

ex: web click data, medical records, biology, engineering

* + - * + applications can’t be programmed by hand

autonomous helicopter, handwriting recognition, most of natural language processing (NLP), computer vision

* + - * + self customizing programs

amazon, Netflix product recommendations

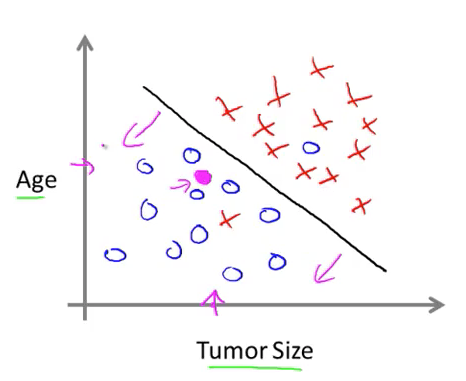
* + - * + understanding learning (brain real AI)
    1. What is Machine Learning?
       - Field of study that gives computers the ability to learn without being explicitly programmed
       - a computer program is said to learn from Experience E with respect to some Task T and some performance measure P, if its performance on T, as measured by P, improves with experience E
       - ex:
         * suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam. What is the task T in this setting?
         * T = classifying emails as spam or not spam
         * E = watching you label emails as spam or not spam
         * P = the number of emails correctly classified as spam/not spam
       - machine learning algorithms
         * supervised learning

teach computer how to do something

* + - * + unsupervised learning

let computer learn by itself

* + - * + reinforcement learning
        + recommender systems
    1. Supervised Learning
       - right answers were given
       - we already gave the actual price of the apartment
       - regression: predict continuous value output (price)
       - classification problem: discrete value output (0 or 1)
         * classifying if someone has cancer or not
         * algorithm will map a straight line to classify the cancer



* + - * infinite number of features so algorithm can use to make the predictions
      * ex:
        + you’re running a company, and you want to develop learning algorithm to address each of two problems

you have a large inventory of identical items. you want to predict how many of these items will sell over the next 3 months

you’d like software to examine individual customer accounts, and for each account decide if it has been hacked/compromised. should you treat these as classifications or as regression problems

problem 1 = regression problem

problem 2 = classification

* + 1. Unsupervised Learning
       - give data set not told what to do with it
       - tell it to find structure in the data
       - clustering algorithm
         * ex:

Google News = cluster news stories that are about the same topic gets grouped together

* + - * examples of clustering

organize computing clusters

social network analysis

see which groups of friends are mutual

market segmentation

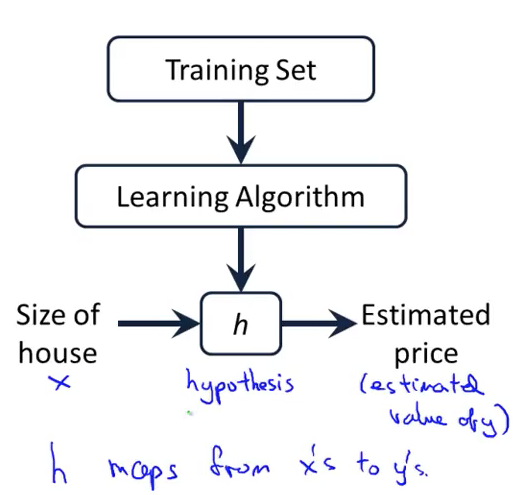
discover market segments

astronomical data analysis

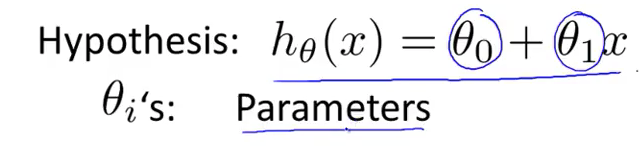
* + - * cocktail party problem
        + there are 2 microphones in a party
        + let’s say there are 2 peoples talking
        + 1 microphone will pick up Person A better than Person B
        + 1 microphone will pick up Person B better than Person A
        + use algorithm to create 2 audio’s each for the person
        + code for this

[W,s,v] = svd( ( repmat( sum( x.\*x, 1 ), size( s, 1 ), 1 ) .\*x ) \*x’ );

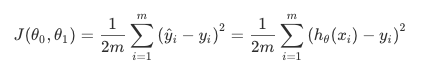
* + - * use octave to prototype because it is very fast
  1. Model and Cost Function
     1. Model Representation
        + notation
          - m = number of training examples
          - x’s = input variable/ features
          - y’s = output variable/ target variable
          - ( x, y ) = one training example
          - ( x^I, y^I ) = ith row training example



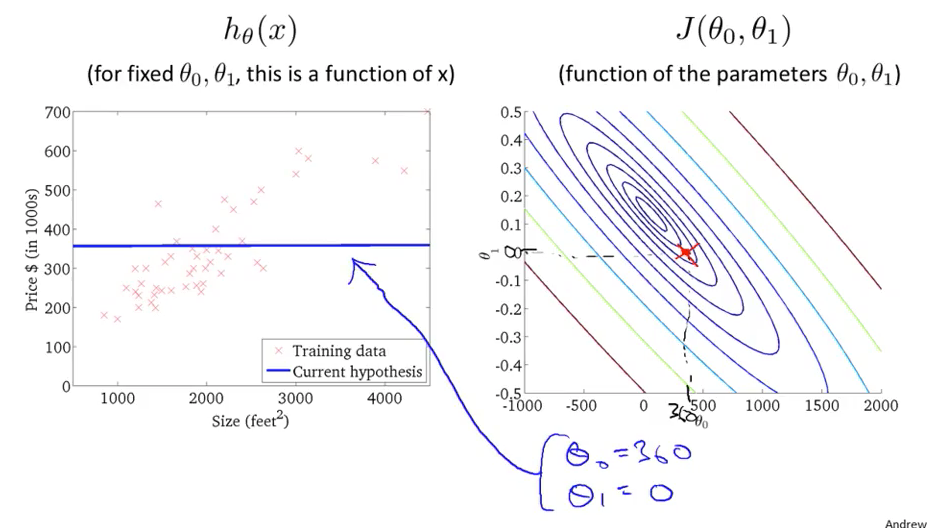
* + 1. Model Representation
    2. Cost Function

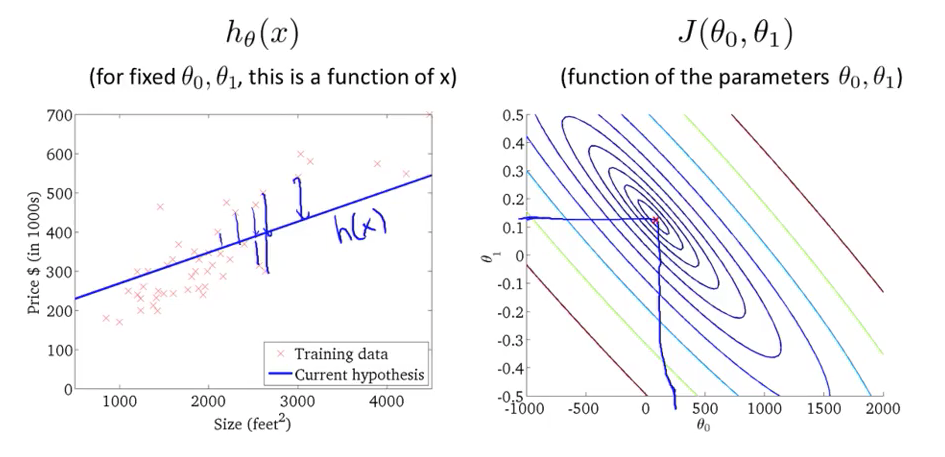


* + - * cost function = takes average diff of all results of the hypothesis with inputs from x and the actual output ys

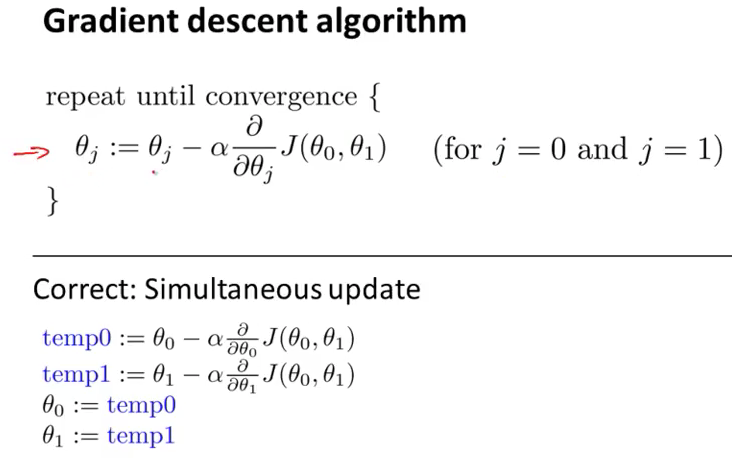


* + 1. Cost Function – Intuition 1
       - basically have to get linear function that minimizes the diff between predicted y and actual y
    2. Cost Function – Intuition 2
       - contour plots
         * each plot on the same line has the same J(o)

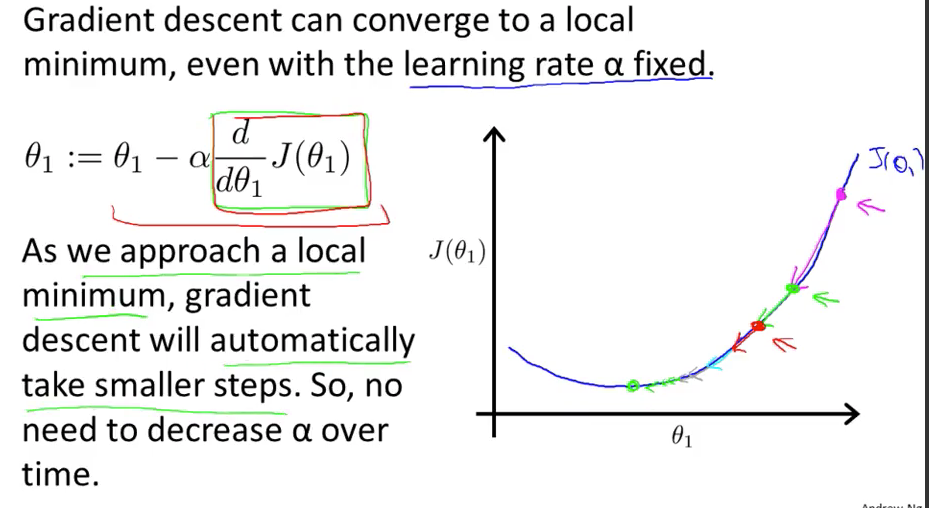




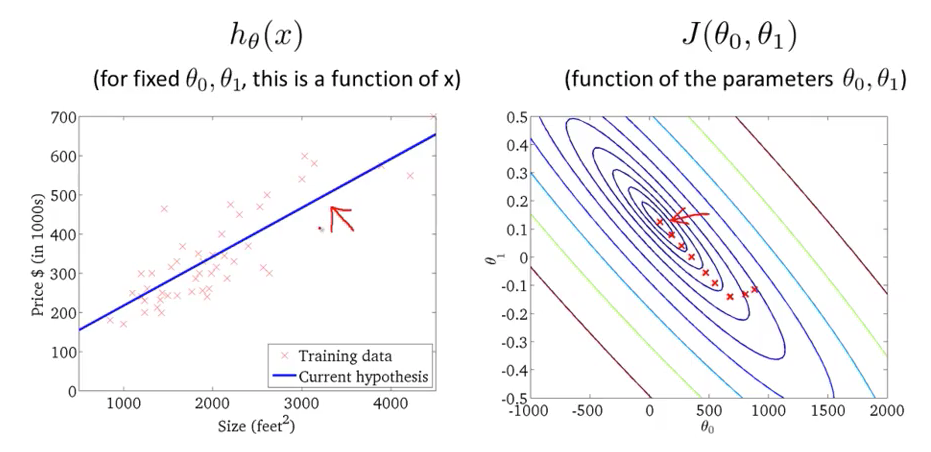
* 1. Parameter Learning
     1. Gradient Descent
        + used to minimize
        + can use to save general functions
        + keep changing theta 0 and theta 1 to reduce J()



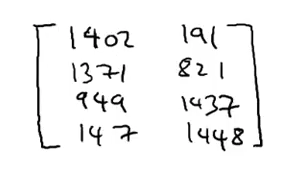
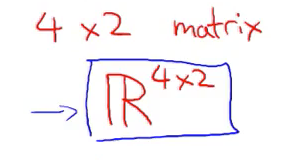
* + - * a := b this will assign a to have the value of b
      * a = b (truth assertion, Boolean condition )
      * alpha = is the learning rate
        + controls how big the step we take down hill
        + larger alpha = more aggressive
      * want to simultaneously update theta 0 and theta 1
    1. Gradient Descent Intuition
       - if alpha is too small, gradient descent can be slow
       - if alpha is too large, gradient descent can overshoot the min, it may fail to converge or even diverge
       - if at local min, it will not move
       - gradient decent can converge to a local min, even when the learning rate is fixed
       - as we approach a local min, gradient descent will auto take smaller steps
       - so no need to decrease alpha over time



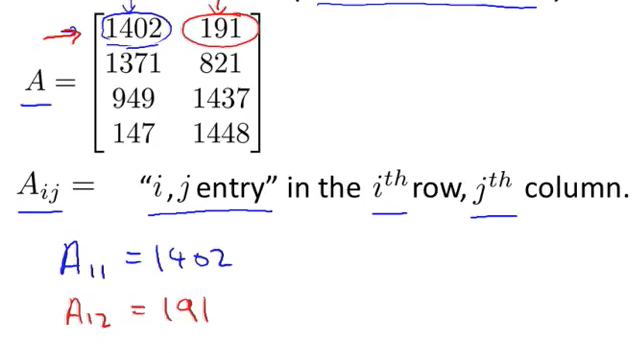
* + 1. Gradient Descent For Linear Regression



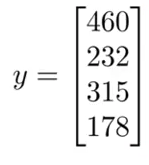
* + - * batch gradient descent
        + each step of gradient descent uses all the training examples
  1. Linear Algebra Review
     1. Matrices and Vectors
        + matrix = rectangular array of numbers

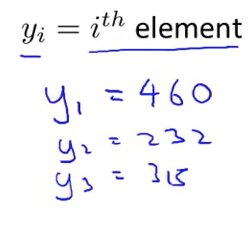
* + - * dimension of matrix = number of rows x number of columns
      * matrix elements (entries of matrix)



* + - * Vector = An n x 1 Matrix

Screen%20Shot%202018-05-15%20at%2010.33.54%20AM.png

* + - * + this is a 4 dimensional vector



* + - * + 1-indexed = start with y1
        + 0-indexed = start with y0
        + use 1-indexed unless specified otherwise
        + machine learning applications usually use 0-indexed vectors
        + use capital letters to refer to matrix
        + code

// the ; denotes we are going back to a new row

A = [1,2,3; 4,5,6; 7,8,9; 10,11,12]

// initialize new vector

v = [1;2;3]

// get dimension of matrix A

// m = rows, c = columns

[m,n] = size(A)

// can do it this way too

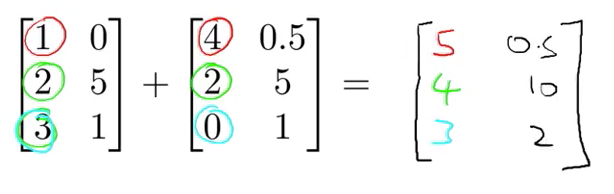
dim\_A = size(A)

dim\_v = size(v)

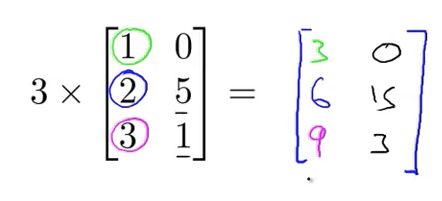
// indexing matrix

A\_23 = A(2,3)

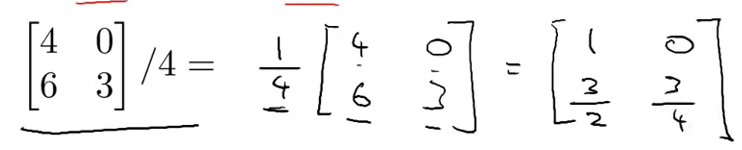
* + 1. Addition and Scalar Multiplication
       - Matrix Addition
         * add up element one at a time
         * can only add matrix of same dimension



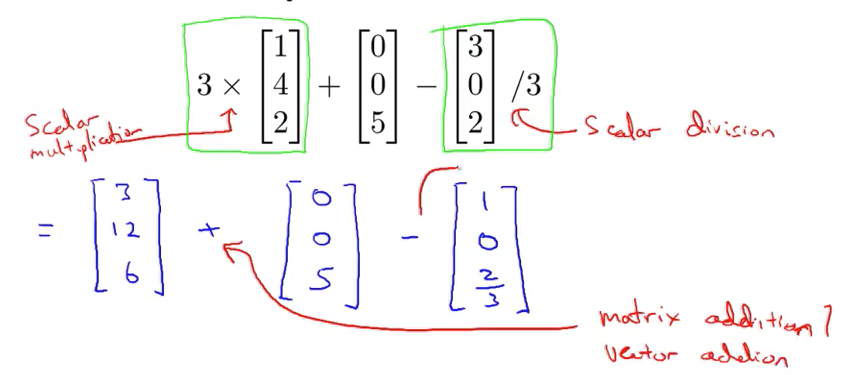
* + - * Scalar Multiplication
        + multiple each element by the scalar



* + - * + Divide by a number
        + same as multiplying



* + - * Combination of Operands
        + evaluate multiply and division first
        + then evaluate addition and subtraction



* + - * code

A = [1,2,4;5,3,2]

B = [1,3,4;1,1,1]

s = 2

add\_AB = A + B

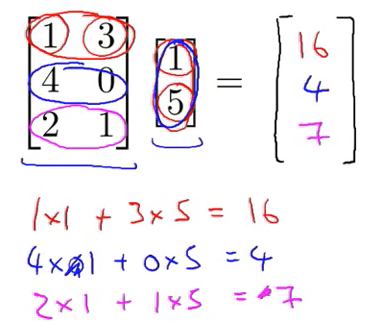
sub\_AB = A – B

mult\_As = A \* s

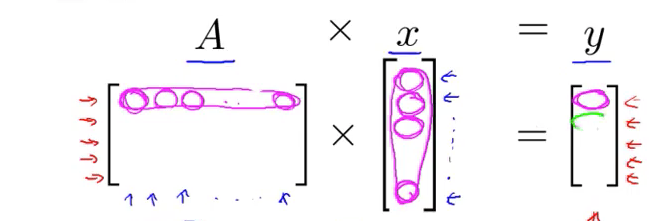
div\_As = A / s

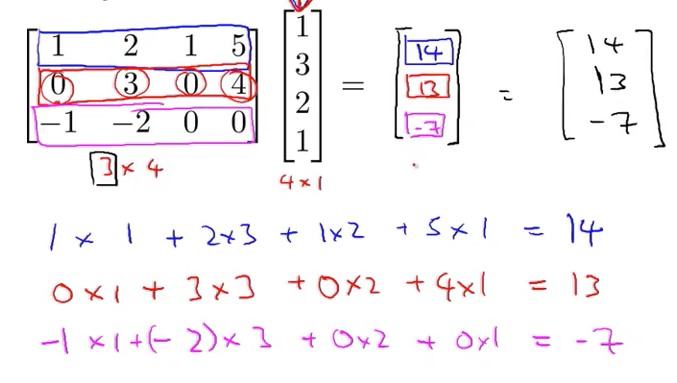
add\_As = A + s

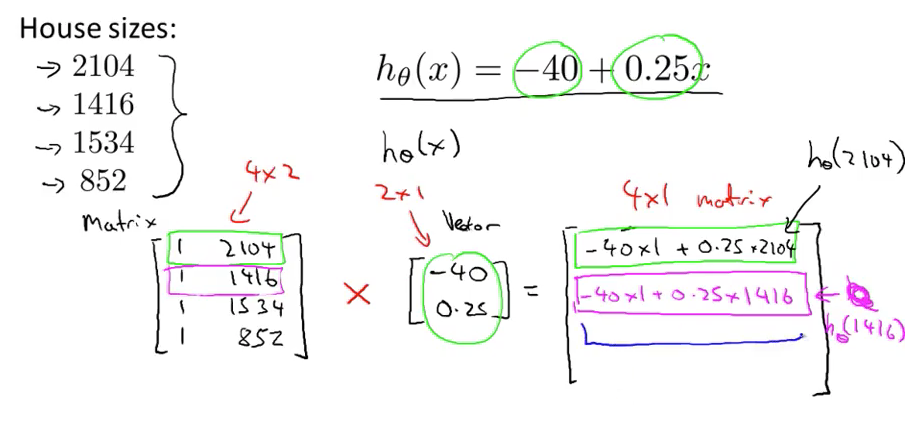
* + 1. Matrix Vector Multiplication



* + - * result is 3 x 1 matrix
      * number of columns in matrix have to match number of rows in vector
      * m x n (matrix) \* n x 1 (vector ) = m x 1 (vector)
      * to get Yi, multiply A’s ith row with elements of vector x and add them up



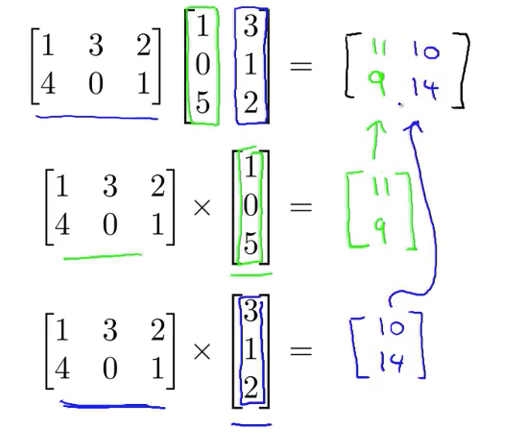




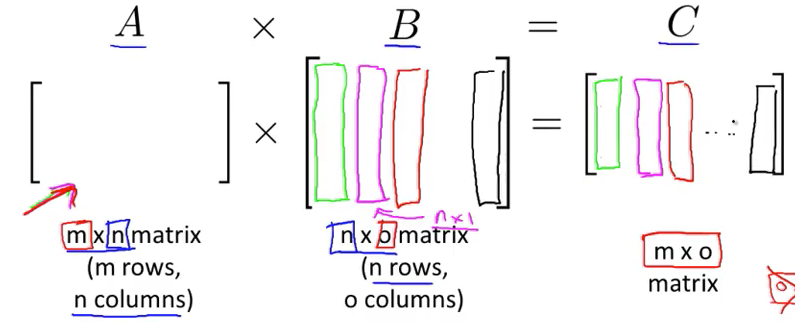
* + - * can use this line of code

prediction = DataMatrix \* parameters

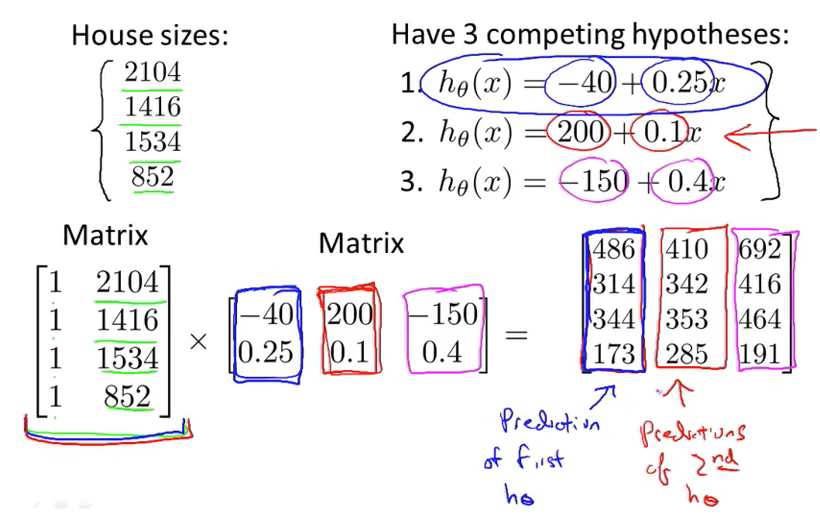
* + - * this code is more efficient than for loops for prediction
    1. Matrix Matrix Multiplication



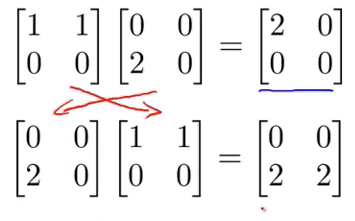
* + - * number of columns in matrix1 must match number of rows in matrix2
      * m x n (matrix) x n x o (matrix) = m x o (matrix)
      * treat each column in B as a vector and multiply matrix A, results will be column of C



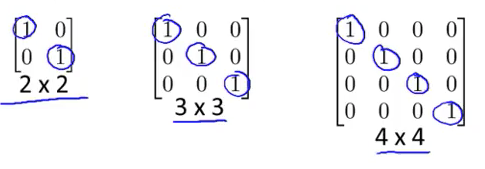
* + - * the I th column of matrix C is obtained by multiplying A with the ith column of B



* + 1. Matrix Matrix Multiplication
       - A x B does not equal B x A in matrix multiplication



* + - * the dimensions also don’t match
      * associative rule works in matrix multiplication
      * A x B x C = (A x B) x C = A x (B x C)
      * Identity Matrix
        + has 1 alone the diagonals and 0 for the rest
        + it is like 1 in numbers for matrix
        + it multiply something else will become something else

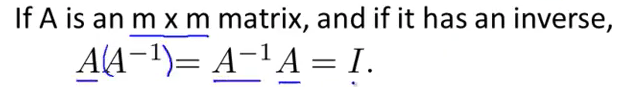


* + - * code

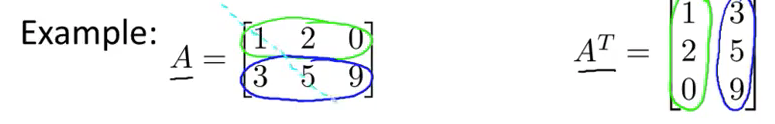
// initialize a 2 x 2 identity matrix

I = eye(2)

* + 1. Inverse and Transpose
       - inverse = if you multiple a number by its inverse, the result is 1



* + - * only works for m x m matrix (square matrix)
      * matrix that don’t have an inverse is called singular or degenerate
      * matrix transpose



* + - * you reverse the position
      * A is a m x n matrix, let B = A^T
      * Then B is a n x m Matrix
        + Bij = Aji
      * code

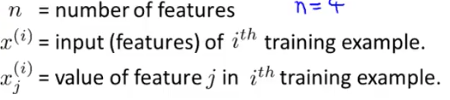
A = [1,2,0; 0,5,6; 7,0,9]

A\_trans = A’

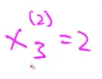
A\_inv = inv(A)

A\_invA = inv(A) \* A

1. Week 2
   1. Multivariate Linear Regression
      1. Multiple Features
         * notations

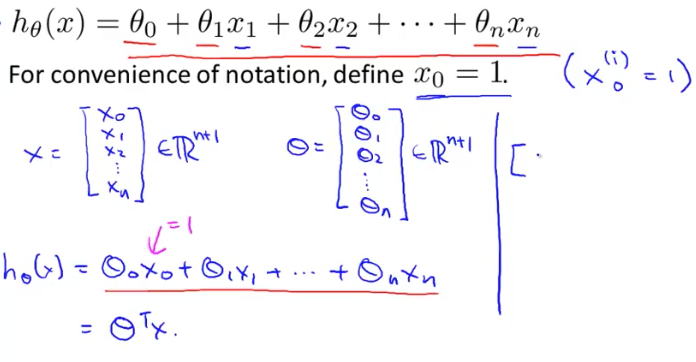


* + - * + x^(2) = [1416; 3; 2; 40]

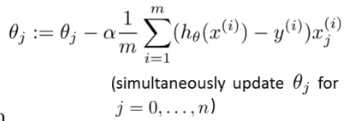


* + - * form of hypothesis

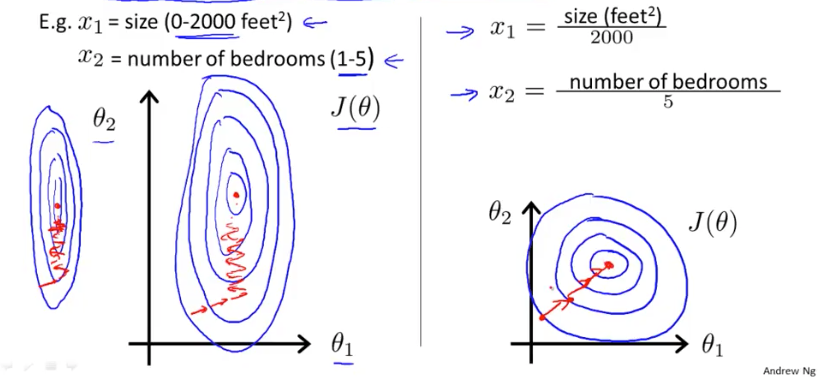
Screen%20Shot%202018-05-16%20at%208.33.15%20AM.png



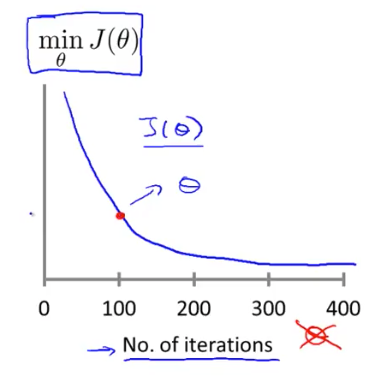
* + 1. Gradient Descent for Multiple Variables
       - very similar to single variables
       - n >= 1
         * repeat {



* + - * + }
        + apply each training set to the hypothesis and subtract it by actual y
        + minimize the difference
        + very similar to single variable
    1. gradient descent in practice 1 – feature scaling
       - if you have multiple features
       - make sure features are on a similar scale (normalize)
       - if not features with large values (size) will have more significance than smaller values (number of rooms) for predicting prices
       - this will also make it faster for gradient descent
       - ex:

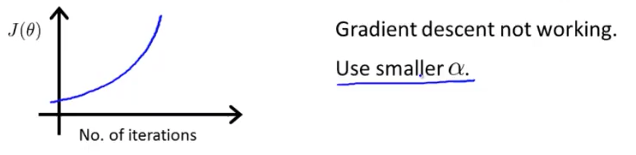


* + - * feature scaling
        + get every feature into approximately -1 <= xi <= 1 range
      * mean normalization
        + replace xi with xi – mean
        + to make features have approximately zero mean
        + do not apply to xo = 1
        + get z-score
    1. Gradient Descent in Practice II – Learning Rate
       - make sure gradient descent is working correctly



* + - * J(theta) should decrease after each iteration
      * hard to know how much iteration you need to do gradient descent
      * convergence test
        + declare if J(theta) decreases by less than 10^-3 in one iteration

hard to determine how much it should decrease by



* + 1. Features and Polynomial Regression
       - you can combine multiple features into one
       - we change use diff polynomial equations to predict y
       - have to keep account of how to scale it
  1. Computing Parameters Analytically
     1. Normal Equation
        + method to solve for theta analytically

Screen%20Shot%202018-05-16%20at%2011.14.31%20AM.png

* + - * octave command
        + pinv(X’ \* X) \* X’ \* y
        + pinv = computing inverse of matrix
      * this minimize J(theta)
      * m training exampls, n features
        + gradient descent

need to choose alpha

needs many iterations

works well even when n is large

* + - * + normal equation

no need to choose alpha

don’t need to iterate

need to compute (X^T \* X )^ -1

slow if n is very large

* + - * let us find optimum theta without iteration
    1. Normal Equation Non-invertibility
       - what if X^T \* X is non-invertible? (singular/degenerate)
       - octave
         * pinv (pseudo inverse command in octave)
         * inv (inverse command in octave)
       - cause for it to be non-invertible

redundant features

using the same measure with the different unit or representation

ex: size in feet and size in meter

too many features

delete some features, use regularization

m <= n

more features than training examples

* 1. Octave/Matlab Tutorial
     1. basic operations
        + variables
          - not equals is ~=
          - &&
          - ||
          - true = 1
          - false = 0
          - ; suppresses the print
          - disp(sprint(‘2 decimals: %0.2f’, a))
          - use format to change the data type
        + data structure matrix and vectors
          - A = [1 2; 3 4; 5 6]
          - no commas
          - V = 1:0.1:2

makes a vector that starts with 1 and increments by 0.1 until it reaches 2

* + - * + V = 1: 6

makes a vector that starts with 1 and increments by 1 until it reaches 6

* + - * + V = ones(2,3)

makes a matrix with 2 \* 3 with ones filled in

* + - * + V = zeros(1,3)
        + rand(3,3)

makes a matrix 3 x 3 with random number between 0 and 1

* + - * + randn(1,3)

makes matrix with random normal number

Gaussian distribution

* + - * + hist(w)

will plot a histogram

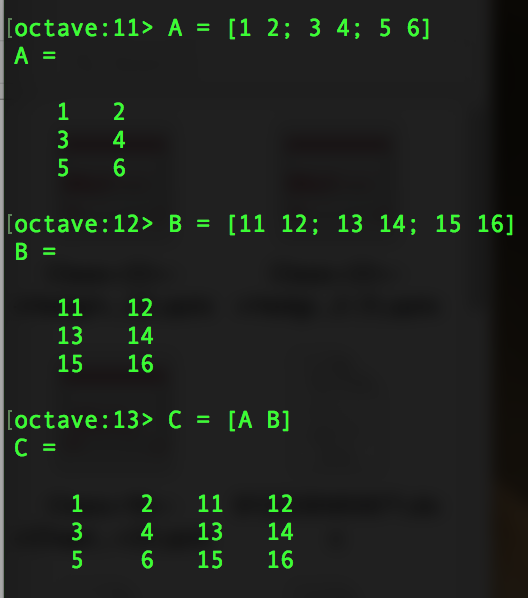
* + - * + hist(w,50)

plot histogram with 50 bins

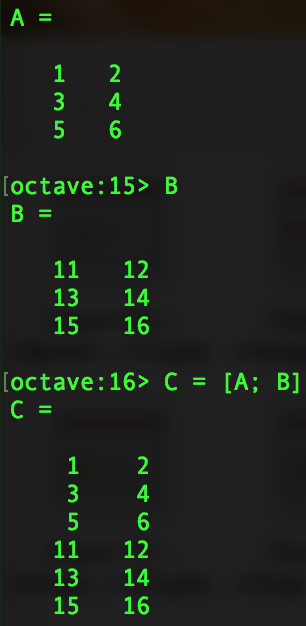
* + - * + eye(4)

will make an identity matrix of 4 x 4

* + 1. Moving Data Around
       - size(A) will return 1 x 2 matrix that tells you the dimension of the matrix passed in
       - length(A) will return the size of the longest dimension
       - you can load data
       - by using
       - load filename.txt or load(‘filename.txt’)
       - who
         * tells you want variables you have in your octave environment
       - whos
         * tells you more detailed info about variables in octave environment
       - clear variableName
         * lets you delete the variable from octave env
       - A= B(1:10)
         * will set A to be the first 10 elements of B
       - save filename.txt variable;
         * will save the variable to the filename.txt
       - clear
         * will clear all variables in the octave env
       - hello.mat
         * saved into binary format with more compressed way
       - save filename.txt v –ascii
         * save it as a human readable way
       - A(3,2)
         * index into matrix A row 3 col 2
       - A(2, :)
         * gets everything in 2nd row
       - A(:, 2)
         * gets everything in 2nd col
       - A([1 3], :)
         * get everything in 1st row and 3rd row
       - A(:, 2) = [1; 2; 3]
         * assign the 2nd col to be 1 2 3
       - A = [A, [100; 101; 102]]
         * this appends the vector [100; 101; 102] to matrix A
       - A(:)
         * puts all elements of A matrix into a vector
       - C = [A B] or C = [A, B]



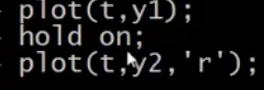
* + - * + this will create a new matrix C that is matrix A concatenated by matrix B
        + appends B to new column
      * C = [A ; B ]
        + appends B to new row



* + 1. Computing Data
       - A .\* B
         * this will multiple the corresponding elements of A with B
         * not traditional matrix multiplication
       - A .^ 2
         * this will square the corresponding elements of A
       - 1 ./ B
         * this will make a new matrix that is reciprocal for each element in B
       - log(v)
       - exp(v)
       - abs(v)
       - -v
         * will make a new matrix that each element is multiplied by negative
       - A’
         * A transpose
       - val = max(A)
         * sets val to be the largest element in matrix A
       - [val ind] = max(A) for vectors
         * this will save max into val
         * save index of max into ind
         * for matrix

will do the column wise max and index

* + - * a < 3
        + will evaluate the condition on all elements in the matrix
        + 1 = true
        + 0 = false
      * find( a < 3)
        + will find the elements in matrix a less than 3
        + return matrix with index of those positions
      * magic(num)
        + will return a matrix of num x num
        + its rows, cols, diag sum up to the same number
      * [r c ] = find(A >= 10)
        + will find element greater than 10 and store it in r and c
        + for r, it will do a row wise comparison and save the index of the row
        + for c, it will do a column wise comparision and save the index of the col
        + both r and c is vector
      * sum(A)
        + will sum all the elements of matrix A
      * prod(A)
        + will multiple all the elements of matrix A
      * floor(A)
        + will round down all elements of A
      * ceil(A)
        + will round up all elements of A
      * max(A, [], 1) or max(A)
        + this will make a new vector
        + each element is the column max
      * max(A, [], 2)
        + row max of matrix A
      * to get single max of matrix A
        + max(max(A))
        + max(A(:))
    1. Plotting Data
       - plot(x,y)
         * command for plotting x and y
         * can only plot 1 command at a time
         * use command ‘hold on’ to plot multiple things



* + - * xlabel(‘label name’) will label the x axis with your string
      * ylabel(‘label name’) will label the y axis with your string
      * legend(‘name’, ‘name’) to label the legends
      * title(‘name’) to label the title with the name string
      * saving the plot
        + cd ‘~/place/to/save’; print –dpng ‘fileName.png’
        + will save the plot as filename.png at the directory listed
      * you can plot several plots
        + figure(1); plot(x,y)
        + figure(2); plot(x1,y1)
      * subplot(1,2,1)
        + makes a subplot 1x2 grid and access the 1st element
    1. Control Statements: for, while, if
       - for loop

v = zeros(10,1);

for i = 1:10,

v(i) = 2^i;

end

* + - * + another way

indicies = 1:10;

for (i = indicies),

disp(i);

end

* + - * while loops

// example 1

i = 1;

while i <= 5;

v(i) = 100;

i = i + 1;

end

// example 2

i =1

while(true),

v(i) = 99;

i = I + 1;

if i == 6

break;

end;

end;

* + - * if and else statements

v(1) = 2;

if v(1) == 1,

disp(“Value is one”);

elseif v(1) == 2,

disp(“Value is two’);

else,

disp(“value is not one or two”);

end;

* + - * function
        + create a file with the function name
        + extension must be m

// example

// defines the function is called squareThisNumber()

// takes in one input x

function y = squareThisNumber(x)

// body of the function

y = x^ 2

// example 2

// can use to return multiple values

function [y1, y2] = squareAndCubeThisNumber(x)

y1 = x^2;

y2 = x^3;

// example 3

function J = costFunctionJ(X, y, theta)

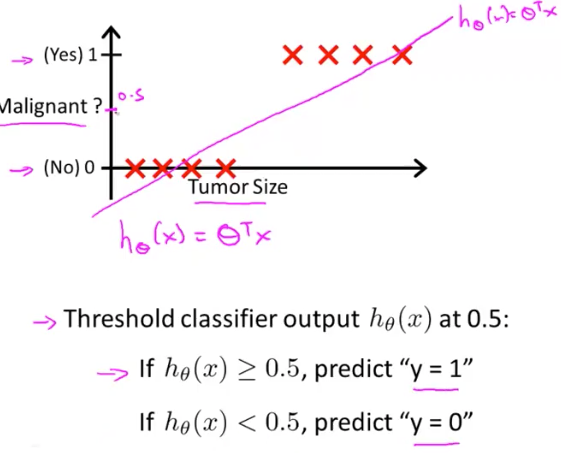
m = size(X,1);

prediections = X\*theta;

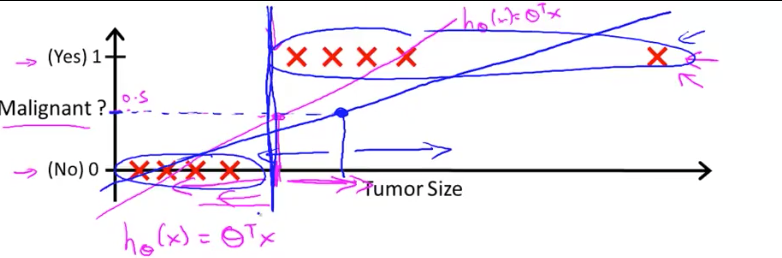
sqrErrors = (prediections – y).^2;

J = 1/(2\*m)

1. Week 3
   1. Classification and Representation
      1. Classification
         * email: spam or not ?
         * online transaction: fraudulent or not ?
         * tumor: malignant or benign?
         * y can be 0 or 1
         * 0 = negative class
         * 1 = positive class
         * multiclass classification problem
           + can be 0 1 2 3
         * threshold classifier output



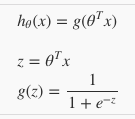
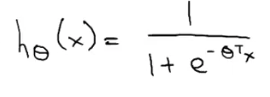
* + - * + threshold classifier output at 0.5
        + if y >= 0.5, predict y = 1
        + if y < 0.5, predict y = 1
        + this is shitty cuz outliers can completely change the slope and line of best fit



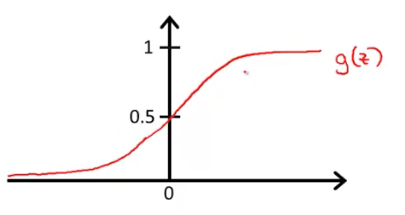
* + - * + linear regression can give you an answer that is greater than 1 or less than 0

output should just be 1 or 0

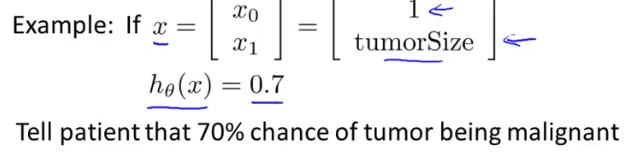
* + - * we are going to learn logistic regression
        + the output will always be between 0 and 1
        + just a bad name
    1. hypothesis representation

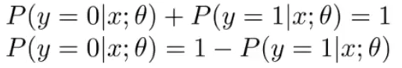
* + - * g(z) = 1 / (1 + e^-z)
        + this is called logistic/ sigmoid function
        + this is shape of logistic/ sigmoid function



* + - * this asymptotes at 0 and 1
      * this will ensure that predicted y is between 0 and 1
      * prediction = estimated probability that y = 1 on input x



* + - * probability of y is equal to 1 given that x is parameterized by theta

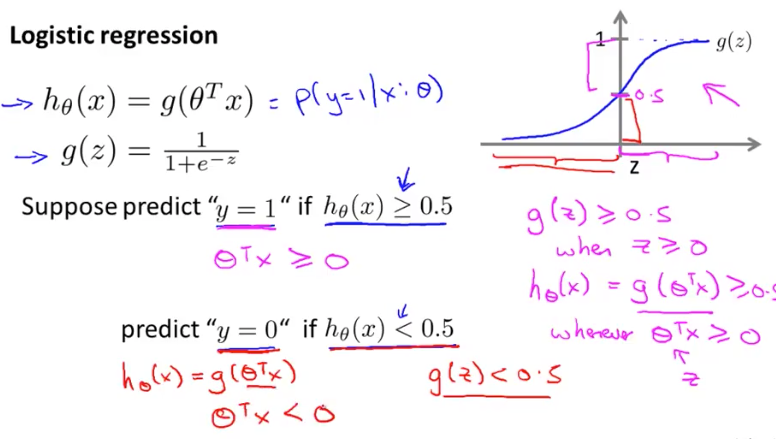


* + - * + the first one

probability of y = 1 given x plus probability of y = 0 given x is equal to 1

probability of y = 0 given x is equal to 1 – probability of y = 1 given

* + 1. Decision Boundary



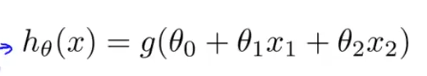
* + - * + basically for g(z) to be greater than 0.5 this occurs when z greater than or equal to 0

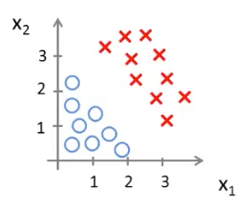
this can be put that theta transpose times x has the be larger than or equal to 0 for it to work (vice versa)

* + - * + for g(z) to be less than 0.5 this occurs when z is less than 0.5

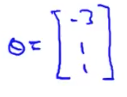
equivalent to be theta transpose times x is less than 0

* + - * + basically it is saying if z is positive => there is 50% chance for y to be 1
      * decision boundary





* + - * lets choose the parameter to be

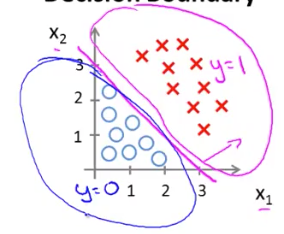
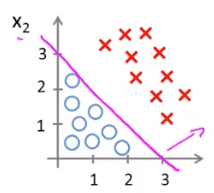


* + - * + theta\_0 = -3
        + theta\_1 = 1
        + theta\_2 = 1
        + predict y = 1 if -3 + x1 + x2 >= 0

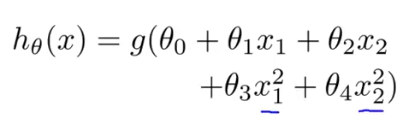
-3 + x1 + x2 = theta\_transpose times x

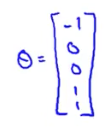
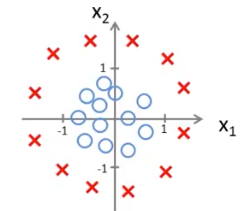
or the condition x1 + x2 >= 3

graph of the equation x\_1 + x\_2 = 3



* + - * + the line is called the decision boundary
      * nonlinear decision boundaries





* + - * + let’s say

theta\_0 = -1

theta\_1 = 0

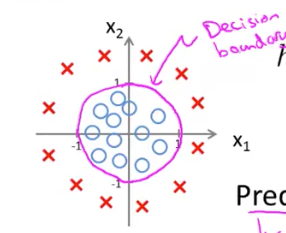
theta\_2 = 0

theta\_3 = 1

theta\_4 = 1

* + - * + predict y = 1 if -1 +( x\_1)^2 \_ (x\_2) ^ 2 >= 0

x1^2 \_ x2^2 >= 1

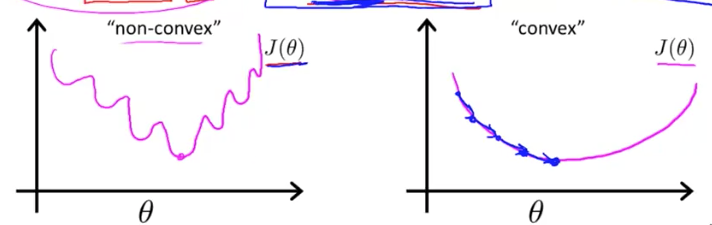


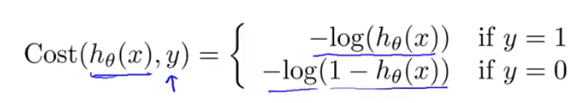
decision boundary is

training set is used to fit the parameters theta

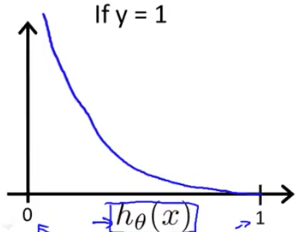
parameter theta defines decision boundary

* 1. Logistic Regression Model
     1. cost function
        + cost function is what the ai will pay for getting it wrong
        + how to choose parameters theta?
        + for logistic regression have to use a diff cost function because it is a convex
        + this guarantees a convex function that has global min that gradient descent will find





* + - * if y = 1
        + cost function is –log(h(x))



* + - * + this means if you predict 1 or close to 1 and y is 1

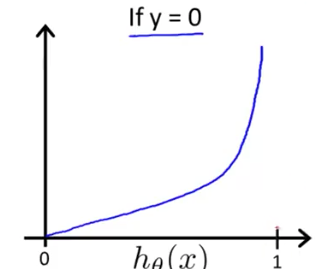
the cost is very small, almost 0

* + - * + if you predict 0 or close to 0 and y is 1

the cost if HUGE

we penalize the learning algorithm by a very large cost

* + - * if y = 0, cost function is –log(1 – h(x))



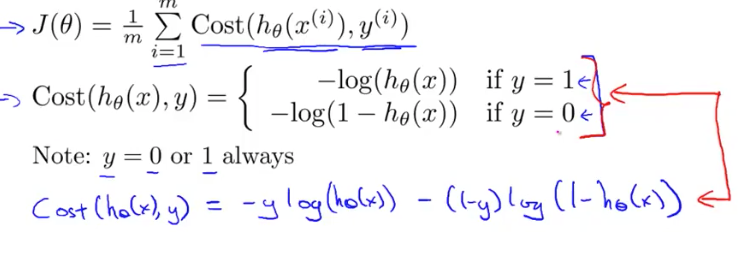
* + - * + same idea as befor
        + if y = 0 and we predict close to 1

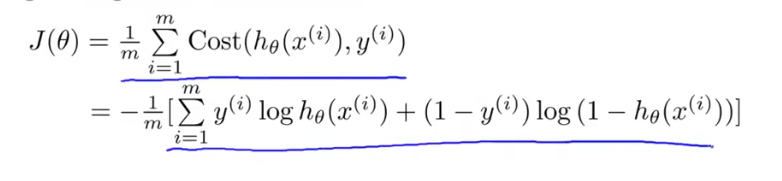
we penalize the AI by a very large amount

* + - * + if y = 0 and we predict close to 0

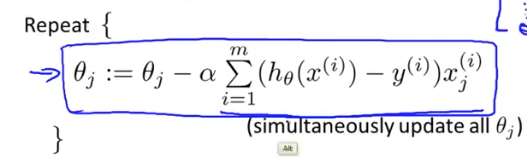
the cost is very little almost 0

* + 1. Simplified Cost Function and Gradient Descent

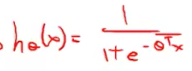




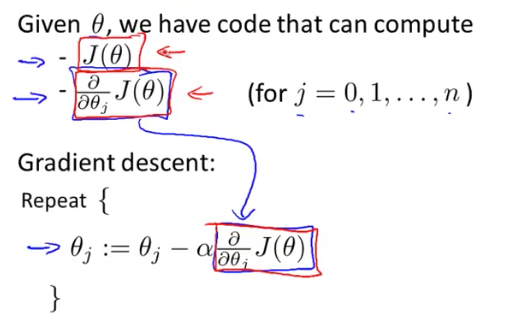
* + - * now that we have cost function we have to figure out how to find the best theta
      * gradient descent



* + - * + this algorithm is identical to linear regression
        + this is cuz h(x) has changed



* + - * feature scaling will also make gradient descent run faster
    1. Advanced Optimization
       - cost function
       - need to supply code to compute J(theta) and partion derivative



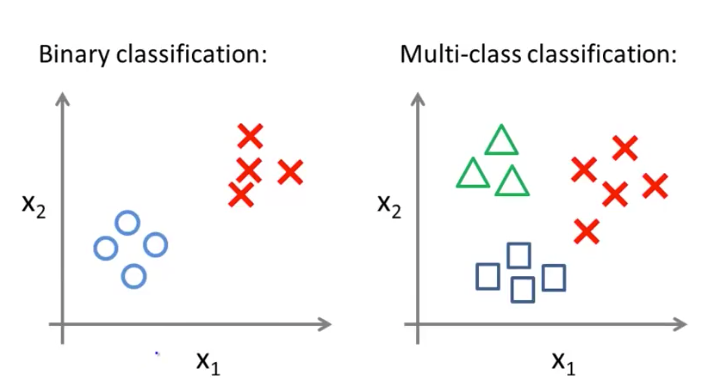
* + - * optimization algorithms to use
        + gradient descent
        + conjugate gradient
        + bfgs
        + l-bfgs
      * advantages
        + no need to manually pick alpha
        + often faster than gradient descent
      * disadvantages
        + more complex
      * have clever inner loop
        + picks a great learning rate alpha for each iteration



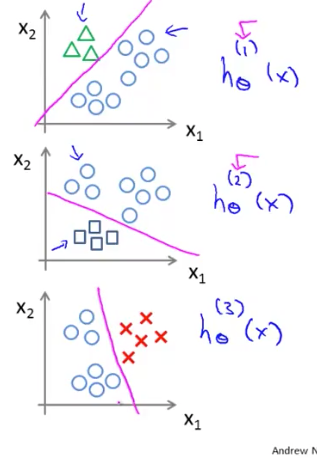
* + - * can use advanced optimization
        + function minization unconstrained s



* + - * + max iteration = 100
        + gradObj = on, you provide the gradient
        + jVal = cost
        + gradient = 2 by 1 vector correspond to partial derivate terms
  1. Multiclass Classification
     1. Multiclass Classification: One vs All
        + ex:
          - email foldering/tagging: work, friends, family, hobby
          - medical diagrams: not ill, cold, flu
          - weather: sunny, cloudy, rain, snow

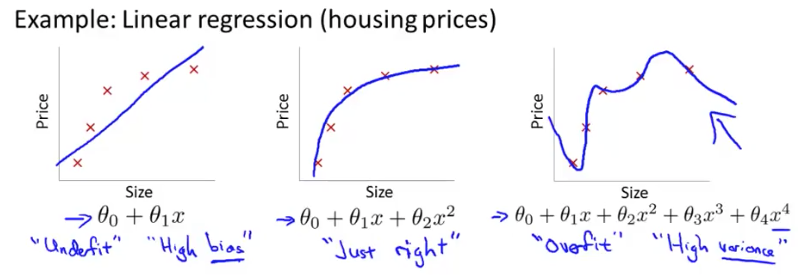


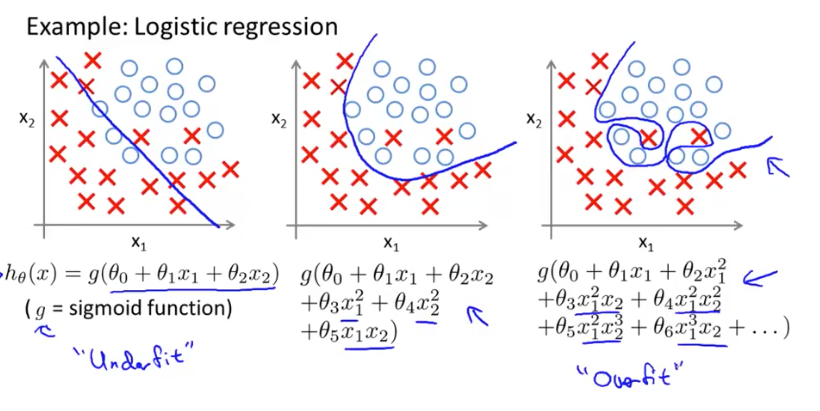
* + - * one versus all
        + basically just train a classifier to separate one group from the rest
        + do this for each group
        + each classifier is h subscript 2 (x)



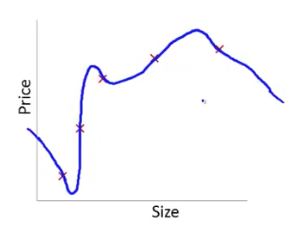
../Screen%20Shot%202018-05-23%20at%208.18.24%20AM.png

* + - * + predict probability that y = i given x
        + pick the class i that maximizes the hypothesis
  1. Solving the Problem of Overfitting
     1. The Problem of Overfitting
        + under fit
        + high bias
          - algorithm has strong conception to believe that there is a linear relationship when clearly there isn’t one
        + over fit
          - all data points go through
        + high variance
        + overfitting = if we have too many features, the learn hypothesis may fit the training set very well, but fail to generalize to new examples (predict prices on new examples)





* + - * addressing overfitting
        + looking at graphs
        + to determine by eye



* + - * + however it may not work if there are many features, like 100

even if the features are reasonable

* + - * + how to address overfitting

reduce number of features

manually select which features to keep

model selection algorithms

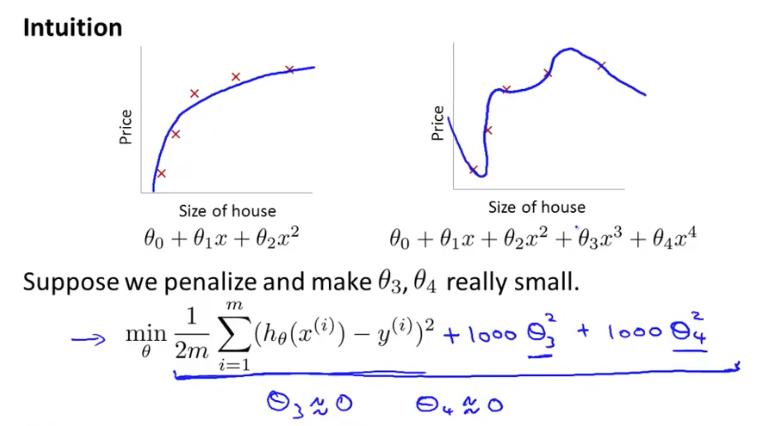
alogirhtm help you determine which features to throw away

regularization

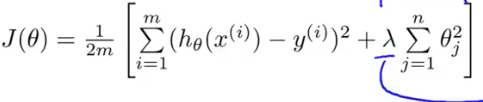
keep all the features, but reduce magnitude/values of parameters

works well when we have a lot of features, each of which contributes

* + 1. cost function of regularization
       - make theta\_3 and theta\_4 small by making the penalty for it very large in the cost function



* + - * small values for the parameters will make the hypothesis more simple
        + this will make it less prone to overfitting
      * since you don’t know which features are more important or not its hard to pick in advance
      * you add a regularization function to the cost function



* + - * + lambda = regularization parameter

will control the tradeoff between 2 different goals

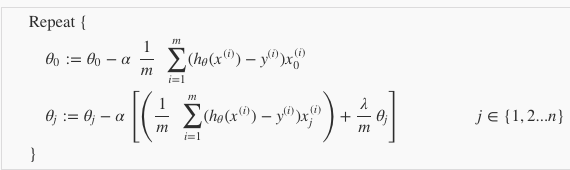
goals: fit training data well vs keep parameters small

if lambda is large, it will penalize all parameters very heavily

this will make all parameters very small

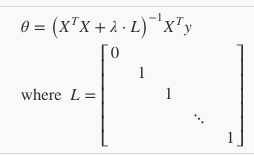
this will under fit it

the ai has a strong bias



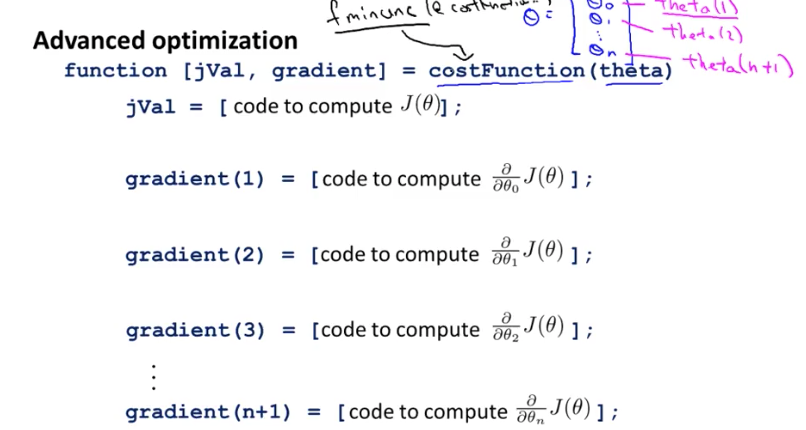
../Screen%20Shot%202018-05-23%20at%209.14.53%20AM.png

* + - * it is reducing the value of theta\_j by some amount on every update
      * this makes every theta smaller
      * normal equation



* + - * + if m < n, then Xt \* X is non-invertible, adding term lambda alpha makes it invertible
    1. regularized logistic regression
       - similar idea to regularized for linear regression
       - regularized cost function

../Screen%20Shot%202018-05-23%20at%209.26.48%20AM.png



* + - * treat theta\_0 differently
  1. Assignment
     + - fminunc is optimization solver that finds the min of uncontrained function
       - optimize the cost function with parameters theta