Machine Learning by Stanford University

Instructor: Andrew Ng || Coursera

# Introduction

## What is machine learning?

* Sorting email into spam/not spam
* Definition by Tom Mitchell: “a computer program is to learn from experience E with respect to some task T and some performance measure P and improves with experience E.”
  + Checkers example: T = play checkers, E = playing the game many times, P = win/lose
* Two main types: supervised (teach), unsupervised (let it learn)

## Supervised learning

* Supervised – given dataset + what the “right” answer is
* Examples of problems: tumor malignant/not malignant, predict house prices give SQFT
* Regression: predict continuous valued output
  + Fitting regression line to data + interpolating
* Classification: predict discrete value output (categorical/binary)
* Can have additional features to be considered in ML algorithm
  + Tumor size, age, thickness, uniformity…etc
  + Can also have infinite number of features !!

## Unsupervised learning

* Given this data, can you find some structure in the data?
  + No feedback, right or wrong answers
* Examples: finding similar news stories + grouping them, labelling different market segments
* Clustering algorithm: Can classify data groups into clusters
* Non-clustering: Cocktail party problem algorithm: separate out two speakers from two audio recordings at different locations
  + One line of code! Singular value decomposition

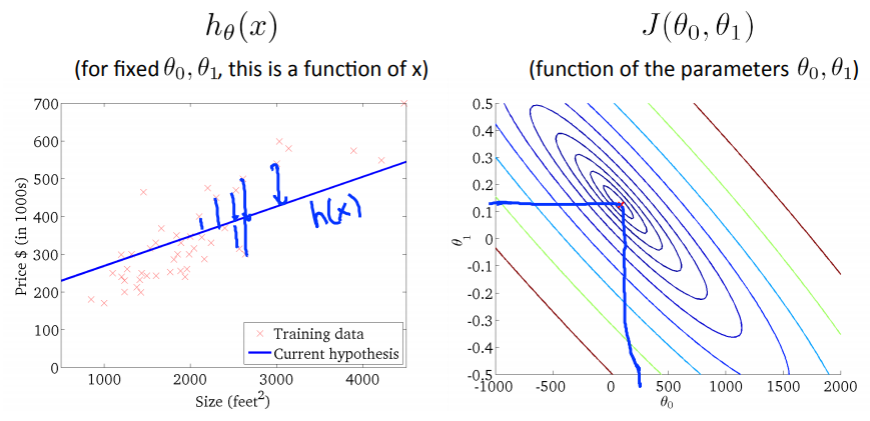
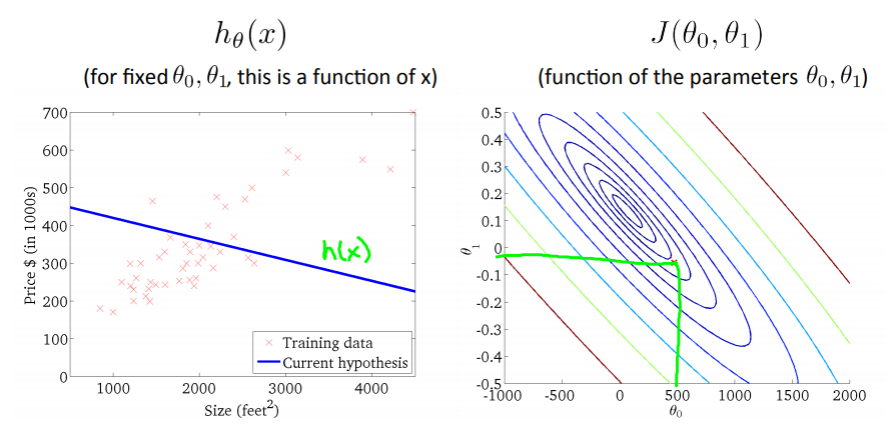
# Linear Regression – One Variable

## Model representation

* Supervised learning, regression problem
* Training set = dataset
  + m – number of training examples
  + x – input variables/features
  + y – output variable/target variable
  + (x,y) – one training example
* Training set -> learning algorithm -> function h(x) (hypothesis)
  + h(x) function maps from x to y
* Can represent h(x) as a line – linear regression
  + – how to choose theta’s?

## Cost function

* How to come up with theta values that best fit data
  + Choose values so that h(x) is close to y for our training examples
  + Minimization problem – find thetas such that average of sum of squared value of error between h(x) and y is minimized
* Mean squared error/Squared error cost function – most commonly used cost function
* For each theta hypothesis, we can plot the cost function (and find its minimum)
  + Question: how to find the best values that minimize cost function?



## Gradient descent

* Algorithm to minimize the cost function
  + Start with some theta 1, 2
  + Keep changing values to reduce cost function until we hopefully end up at a minimum
  + \*each iteration decreases the cost function until reaching a local minima
* For theta 1 and theta 2, assign next value as the current value minus a step factor of the direction of the slope WRT theta 1, 2 at the current point
  + Basically, take a step down
  + Alpha – learning rate, controls size of step
    - Too small: slow learning
    - Too big: may overshoot minimum or diverge
* Applying it to linear regression cost function: batch gradient descent
  + Batch – uses all data points (i=1 to m)

# Linear Algebra Review

## Matrices & Vectors

* Matrix- (rows x cols) rectangular array of vectors, UPPERCASE VARIABLES
* Vector – matrix with one col, lowercase variables
* Addition, scalar multiplication
* Can represent linear equation using matrices
  + Y = mX + b;
* (A\*B)\*C = A\*(B\*C) – associative property

# Multivariate Linear Regression

## Multiple features

* Before:
* Now:
* Notation: – value of ith example in jth  feature/variable set
  + j- variables/parameters/features
  + i- training sets/subjects/examples

## Gradient descent for multiple variables

|  |  |
| --- | --- |
| Previous  Cost function:  Gradient descent (n = 1) | Multivariate  Cost function:  Gradient descent (n >= 1) |

## Feature scaling

* Idea – make sure features are on similar scale
  + Make contours more symmetric/balanced, makes gradient descent process faster
  + Ex: x1 = size (0-2000); x2 = num bedrooms (1-5)
  + Convert: x1’ = x1/2000, x2’ = x2/5
* Have each feature between -1 < x < 1 range
  + Large numbers or too narrow scale are not ideal
* Mean normalization – replace all values with mean subtracted to have approximately zero mean
  + Eqn also helps with feature scaling (s1 – std or range)

## Learning Rate

* Make sure gradient descent is working correctly
  + Plot iterations v. cost function, plot should decrease over iterations
  + Declare convergence if cost function decreases by less than 10^-3 in one iteration
  + If not, then decrease learning rate alpha
* To choose learning rate, try: 0.001, 0.01, 0.1, 1

## Features & polynomial descent

* Can create polynomial regression equation from linear regression
* Use different polynomial equations to best represent the data and what you know about its trajectory
* Feature scaling becomes much important when we introduce exponentials!!!

# Computing parameters analytically

## Normal equation

* A method to solve for cost function minima analytically
  + Derivative of cost function -> 0
  + – where X is coefficient matrix (including x0 = 1)
  + ^they did not prove this
* Do not need to choose learning rate alpha, no iteration
* Need to solve for (XTX)^-1, which is O(n3) – when n > 1e4 consider doing gradient descent
* No feature scaling needed
* Non-invertibility: when matrix X is not linearly independent (redundancy or too many features)

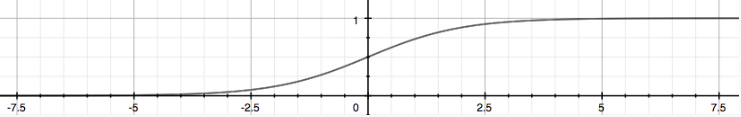
# Classification & Representation

## Classification

* Binary classification
  + Where 0 🡪 negative class; 1 🡪 positive class
* Option 1: use linear regression + set a threshold at 0.5 for classification
  + May have poor performance from extreme data points
  + Linear regression – produces values > 1 // < 0, doesn’t make sense for binary classification
* Want our classifier to output values between 0, 1 …

## Hypothesis Representation

* New hypothesis function:
* Sigmoid/Logistic function – used due to unique property of asymptotes @ 0 and 1



* Interpretation: output gives us probability that output is 1

\*probability of y=1, given x, parameterized by theta

## Decision Boundary

* Example: predict y=1 if h(x) >= 0.5, otherwise predict y=0
* Decision boundary – line that separates regions that represent y=1, y=0
  + Non-linear decision boundaries – higher order polynomials

# Logistic Regression Model

## Cost Function

Cannot use linear regression cost function – will produce ‘wavy’ curves with many local minima if you plot a parameter vs. the cost function.

Cost function for logistic regression:

* Captures the logic: Cost = 0 if h(x) = y
* Guarantees that error function is convex for logistic regression (one minima)
* Related to the principle of maximum likelihood estimation



* If y = 1, and the hypothesis h(x) = 1, then the value of the cost function is 0 😊
* If y = 1, and h(x) approaches 0, then the cost exponentially increases -> infinity
  + Good because want to heavily penalize + adjust next hypothesis
* If y = 0, and h(x) = 0, then the value of the cost function is 0
* If y = 0 and h(x) approaches 1, then value of cost function exponentially increases

## Simplified Cost Function + Gradient Descent

* Can condense the cost function into one line:
* Then can find parameters by minimizing the J cost function using gradient descent

repeat

\*vectorized

* Same equation as for linear gradient descent, but note that the h(x) hypothesis function is different

## Advanced optimization

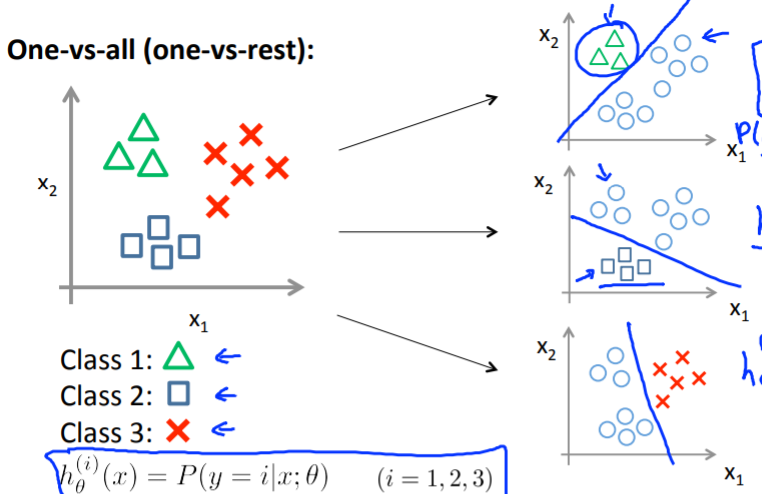
* Writing code to compute J(theta) and partial derivative of J(theta)
* Optimization algorithms (other than gradient descent)
  + Conjugate gradient, BFGS, L-BFGS
  + Advantages: they pick step size, often faster than gradient descent
  + Disadvantages: more complex
  + Kinda like stiff solvers

Example - Octave:

* >> options = optimset(“GradObj’,’on’,’MaxIter’,’100’);
* >> initialTheta = zeros(2,1);
* >> [optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta, options)
* Function [jVal, gradient] = costFunction(theta) {jVal = zeros(2,1)…, gradient = zeros(2,1)…}
* fminunc = Function minimum unconstrained

## Multi-class Classification: One vs. all

* Classifying in categories (need to quantify using discrete numbers)
  + Ex: work (y = 1), friends (y = 2), family (y = 3), hobbies (y = 4)
* Need to train a logistic regression classifier (h(x)) for each class to predict the probability whether the output is that class or not that class
  + Will end up with a classifier for each class

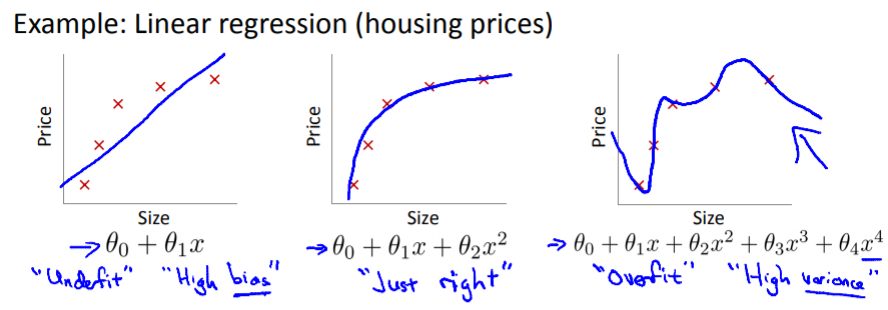
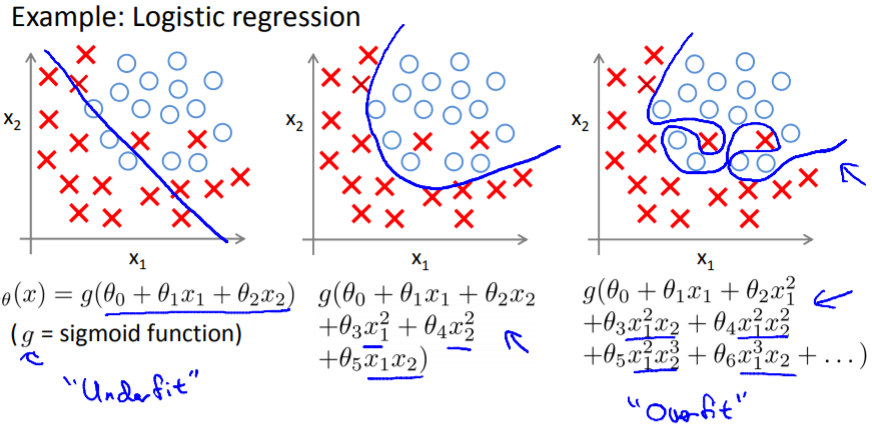


* To predict which class an input is in, run all three classifiers and pick the class that has the highest probability

# Model Overfitting

## Problem of Overfitting

* Too many features – learned hypothesis may not be good at generalization/not robust to new examples = overfitting
  + Fits the training set VERY well, cost function may be very close to zero

To address overfitting:

* Reduce number of features
  + Manually select which features to fit
  + Model selection algorithm
* Regularization
  + Keep all the features, but limit the magnitude of parameters (theta)
  + Works well when there are many features, each feature contributes a little

## Cost Function pt. 2

* Regularization – small values for the parameters results in a simpler hypothesis

Ex: can modify cost function to penalize certain parameters:

Modified cost function for regularization:

* First sigma iterates through all values (1 to m) – calculates initial cost function
* Second sigma iterates through all parameters except theta0 (1 to n) and is the regularization term – keeps all the parameters relevant but small
  + Lambda is the regularization parameter – balances fitting the data well (first sigma) and avoiding overfitting (second sigma)
  + If we set lambda too large – all parameters will be close to zero, result in underfitting

## Regularized Linear Regression

Cost function:

Gradient descent:

\*theta0 does not get penalized

\*1-alpha\*lambda/m is usually going to be a number slightly less than one

* What this does is on every iteration, the theta\_j term shrinks a little, while the step modification is exactly the same, encourages theta to be smaller

Normal equation:

* Ones on all diagonal except for first one (theta0)
* If XTX is non-invertible, adding the lambda matrix term will make it invertible

## Regularized Logistic Regression

Cost function:

Gradient descent:

Advanced optimization:

* >> function[jVal, gradient] = costFunction(theta)
* >> jVal = [code to computer J(theta)]
* >> gradient(1) = [code to compute ]
* >> gradient(2) = [code to compute ]

REMINDER THAT THETA 0 IS NOT REGULARIZED SO IT NEEDS TO BE CALCULATED SEPARATELY FROM ALL THE REST OF THE THETA VALUES.

# Neural Networks: Representation

## Motivations

**Non-linear Hypotheses**

* Scaling up will result in many features, may be more beneficial to use a subset of features
  + Example: classifying car img vs. non-car img, each pixel is one feature, 50x50 pixel image = 2500 pixels, considering quadratic features ~ 3 million features
  + Quadratic features (), think of it as a correlation matrix with diagonals just being the squared features and the rest of the space are the quadratic features
  + Num quadratic features ~ N^2/2

**Neurons and the Brain**

* Neural networks - algorithms that try to mimic the brain
* “One learning algorithm” hypothesis – with neural rewiring experiments, they were able to rewire the visual stream to the auditory or SM cortex so that those regions learned how to “see”, suggests that the mechanisms there are the same mechanisms that can process vision and there is one singular learning algorithm for the brain

## Neural Networks

**Model Representation**

* Neurons are modeled as computational units that take in inputs (dendrites) and produce outputs (axons)
  + Input = input features (x1,x2…xn)
  + Output = result of our hypothesis function
  + Input node x0 – called “bias unit”, always equal to 1
* Use same logistic function (sigmoid) – called activation function
* Parameters (theta) – called weights
* Simple representation: inputs go into next node/layer, which outputs the hypothesis function
* Can have intermediate (hidden) layers
  + Label hidden layer nodes a0, a1, … , an – activation units (a0 is bias unit)
* Can represent matrix of weights that maps layer j to later j+1
* Therefore, output layer is:
* And each layer gets its own matrix of weights
  + If layer 1 has N units and layer 2 has M units, then weight matrix will have dimension
  + M \* (N + 1)
  + The +1 is because you add the bias

**Model Representation II: Forward Propagation**

* Previous equation can be generalized in vector form by defining the linear combination within the activation function as z:
* Add a bias unit for the hidden layer (a2 = 3 dimensional)
* Neural network can be used to essentially learn its own features – each pair of layers essentially is logistic regression, so in a 2-layer network with input vector x, the second (hidden) layer essentially is the network picking the features it cares about the most and then performing logistic regression on it
* Last theta weight matrix will only have one row (assuming output is one value)

## Applications: Examples + Intuitions

**Example 1: Predicting logical ‘and’ operator**

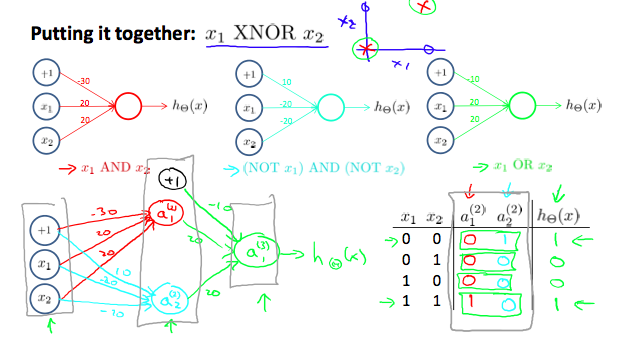
-- checking --

* For ‘or’ operator, just set theta = [-10, 20, 20]

Example 2: Combining layers

* With same setup, here are the matrices for difference logic operators:
  + AND: Q = [-30 20 20]
  + OR: Q = [-10 20 20]
  + NOR: Q = [10 -20 -20]
* To create XNOR, we can create a 2 layer NN to essentially combine these logic operators

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X1 | X2 | A1  (x1 AND x2) | A2  (x1 NOR x2) | Out  (a1 OR a2) |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |



**Multiclass Classification – one vs. all method**

* Output function has multiple elements, let’s say 4
* Each distinct output can be assigned to a discrete classification