What is this ?

Monday, May 29, 2017
3:22 PM

- Notes of Coursera course on Machine learning by Andrew Ng after discussion

Week 1

Monday, July 11, 2016
8:48 PM

Introduction:

What is Machine Learning ?

a. Science of getting computers to Learn without being explicitly programmed
b. Well Posed Learning Problem: Learn from
   i.

Experience	E
Task	T
Performance Measure	P

   ii. P(T) improves with E
c. Grew out of AI

Examples:

a. Database Mining
   i. Medical Records
   ii. Biology
b. Applications that cant' be programmed by hand
   i. Autonomous Helicopter
   ii. Handwriting recognition
c. Self-Customizing Programs
   i. Customized Recommendations
d. Understanding Human Learning

Supervised Learning (Teach computer how to do something)

a. We have the correct answer to the problems.
b. Regression: Continuous valued output
   i. Ex: House price prediction
   ii. Predict the house price, having list of house and prices
   iii. Predict future sales from past data
c. Classification: Discrete valued output.
   i. Ex; Classification of tumors
   ii. Predict a tumor is malignant or benign
   iii. Predict if the account is hacked or not

Unsupervised Learning (Let the computer learn by itself)

a. No labels in data
b. Given a data set => find the structure in data
    i. Find the clusters in data
c. Example:
    i. Clustering news from different channels on same topic into groups
    ii. Clustering the genes into groups.
d. Used to organize Computers in Cluster
    i. Computers which are interacting frequently can be placed together
e. Grouping Social Network friends into groups
f. Market Segmentation.

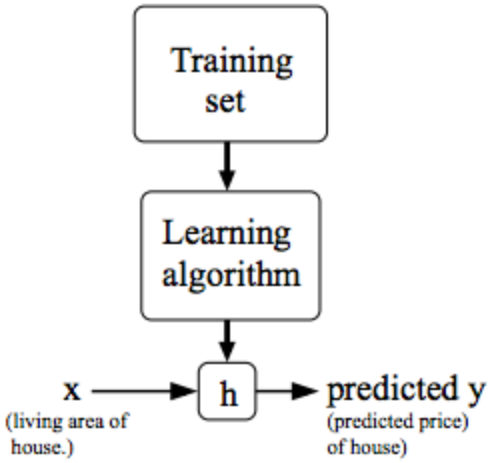
Linear Regression in One Variable:

Hypothesis:	$h_\theta(x) = \theta_0 + \theta_1 x$
Parameters:	θ_0, θ_1
Cost Function:	$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$
Goal:	$\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Model Representation

a. Supervised learning
b. Notations:
    i. x -> Input Variable
    ii. y -> Output Variable
    iii. X -> Input Space
    iv. Y -> Output Space
c. Training Set -> Learning Algorithm -> Hypothesis
d. Size of House (x) -> Hypothesis (h) -> Estimated value of y
e. h maps from x's to y's
f. How to represent h
    i. h(x) = theta0 + theta1 * x
    ii. Linear regression with one variable x.
    iii. Univariate linear regression
    iv. theta0 and theta1 are parameters

g.



Cost Function

a. Used to measure the accuracy of our hypothesis.
b. Choose theta0 and theta1 so that h(x) is closer to y for our training examples
c. Minimize (1/2m) * Sum (h(x) - y)^2 over theta0 and theta1
d. Squared error function or Mean squared error

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

e. $\underset{\theta_0,\theta_1}{\text{Minimize}}\ J(\theta_0,\theta_1)$ (Cost function)

f. Simplified hypothesis function => theta0 = 0

Gradient Descent

a. Take a step in the direction of the deepest descent
b. Different starting points -> Different paths
c. Can go to the local minimum
d. Assignment -> := , Assertion -> =

e.

Gradient descent algorithm

repeat until convergence {

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

}

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

f. If alpha is too small gradient descent can be slow
g. If alpha is too large gradient descent can overshoot the minimum. It may fail to converge or even diverge
h. At local minimum the gradient descent will leave the theta values unchanged as derivative term is zero.
i. No need to decrease the alpha value over time. As the derivative term gets smaller when nearing to local minima => smaller steps when nearing the local minimum.
j. FOR LINEAR REGRESSION:
k.

repeat until convergence: {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$$

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

}

l. Cost function for linear regression will always be a convex function.
   i. Local optimum == Global optimum
m. Batch Gradient Descent is another name for the above algorithm
   i. Each Step of gradient descent uses all the training examples

Linear Algebra Review:

a. Matrix: Rectangular Array of Numbers
b. Dimension: No of Rows X No of Columns
c. Aij => ith row and jth column
d. Vector: An n X 1 matrix -> n dimensional vector.
e. Implicitly 1 indexed arrays unless mentioned
f. Scalar Multiplication.

g. Matrix Vector Multiplication
h. Matrix - Matrix Multiplication
i. Matrix Multiplication properties
    i. Not Commutative
        1. A X B != B X A
        2. A -> m x n, B -> n x m => A X B -> m x m & B X A -> n x n
    ii. Is Associative
        1. A X (B X C) = (A X B) X C
    iii. Identity Matrix
        1. A X I = I X A = A
j. Matrix Inverse
    i. A x Inv(A) = Inv(A) x A = I
    ii. Only square matrices have inverse
k. Matrix Transpose

Questions:

q1: Does regression and classification fall under unsupervised also?
A: The classifications are orthogonal

q2: Learning in unsupervised way ?
A:

q3: what is the inference if the cost function is zero
A:

q4: Diff between linear regression and polynomial regression
A:

q5: Why this cost function for Linear regression ?
A:

q6: Is multivariate cost function convex ?
A:

Notes:

1. Note: YouTube video lectures has derivations

Action Items:

Learn about Maximum likelihood

Week 2

Sunday, September 18, 2016
4:08 PM

Question:

q1: Do we need both mean normalization and Feature scaling?
What is advantage of mean normalization after feature scaling ?
A:

q2: Practical case of alpha varying like WWWW
A:

q3: Need to know more pseudo inverse
Relation between x and x'x invertibility

q4: [TODO][Check] Normal equation only works for Regression but not for classification
A:
Linear Regression with Multiple Variables

Environment Setup Instructions

Multivariate Linear Regression

a. Notation

$$
\begin{aligned}
x_j^{(i)} &= \text{value of feature } j \text{ in the } i^{th} \text{ training example} \\
x^{(i)} &= \text{the column vector of all the feature inputs of the } i^{th} \text{ training example} \\
m &= \text{the number of training examples} \\
n &= \left|x^{(i)}\right|; (\text{the number of features})
\end{aligned}
$$

b. Hypothesis

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n$$

$$h_\theta(x) = \begin{bmatrix} \theta_0 & \theta_1 & \dots & \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

c. Cost Function

Cost function:

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

d. Gradient Descent

repeat until convergence: {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \qquad \text{for j} := 0 \ldots \text{n}$$

}

e. Feature Scaling:

Make sure that the features on the same scale

i. If variables vary on different scale the contour plot can be more elliptical => More skew => More time to global minimum

ii. Get every feature into approximately -1 <= x <= 1
Not strictly 1. Ranges near -1 to 1 should be good
Scale up the features whose range is very less
Scale down the features whose range is very large
Circular contour plot => Less time to global minimum

f. Mean Normalization

i. Replace x with (x - mu) to make features have approximately zero mean (Do not apply for x0 = 1)

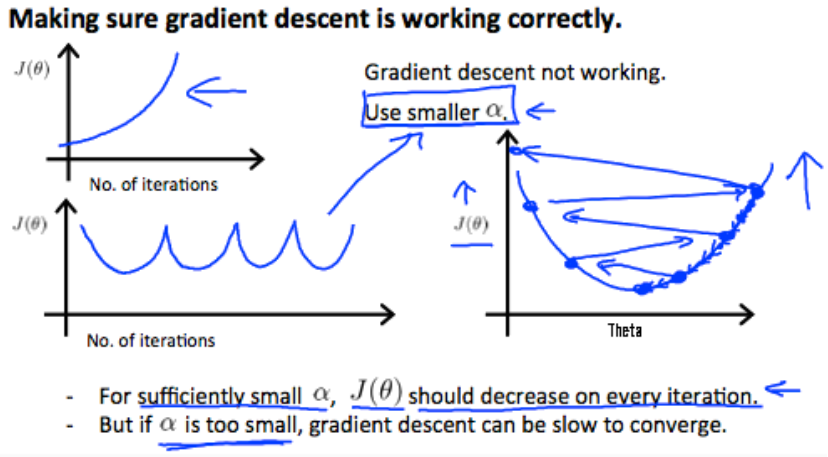
ii. More generally replace
x => (x - mu)/s
mu -> Average, s -> Standard deviation or range i.e. (max - min)

g. Make Sure Gradient descent is working properly

i. Curve of J(theta) vs No of iterations should decrease after every iteration and converge over number of iterations

ii. Automatic convergence test:
Declare convergence if J(theta) is decreasing less than epsilon

iii. Use smaller value of alpha if J(theta) is increasing over number of iterations of if J(theta) is decreasing and increasing continuously.



iv. Try with alpha as below:
0.001, 0.003, 0.01, 0.003, 0.1, 0.3, 1
Plot J(theta) vs #of iterations
Pick alpha which is decreasing the J(theta) rapidly

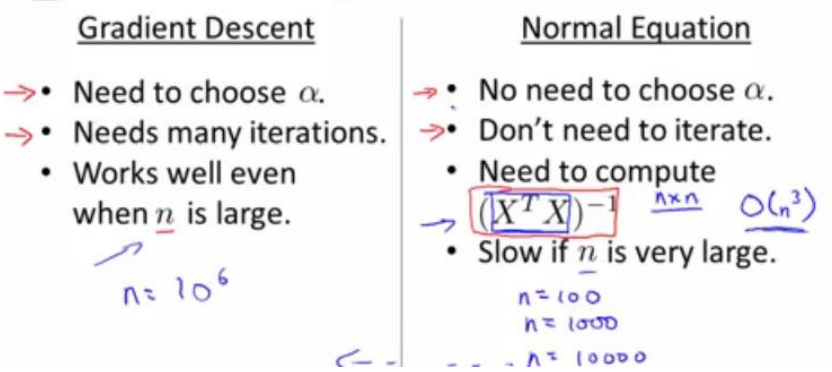
Features and Polynomial regression

a. We can make new features combining the existing features.

b. You can fit a quadratic function or higher order polynomials to our data
   i. Feature scaling is needed when you add higher order terms
   ii. Features can be var, var^2, var^(1/2)

Computing Parameters Analytically

a. Normal Equation: Method to solve for theta analytically
   i. $\theta = (X^T X)^{-1} X^T y$
   ii. Feature scaling is not needed.



iii. What if inv(Xt X) is not invertible
   1. Redundant Features -> Linearly dependents

2. Too many features
3. Use pinv to calculate inv even if the matrix is non invertible

iv. [TODO][Check] Normal equation only works for Regression but not for classification

Octave Tutorial

%	Comment
~=	Not equal
PS1('>> ');	change prompt
;	Suppresses output
disp(a);	Prints a
`disp(sprintf('2 decimals: %0.2f', a))`	`2 decimals: 3.14`
In Matrix => ;	Separates rows
[1 2 3]	Matrix 1 x 3
[1; 2; 3]	Column Vector
1:0.1:2	From 1 to 2 in steps of 0.1
1:6	1 2 3 4 5 6
ones(2,3) 2*ones(2,3)	2 2 2 2 2 2
zeros(1,3)	0 0 0
rand(3, 3)	random numbers from uniform distribution
randn(1,3)	guassian random distribution
hist(w, no_of_bins)	histogram
eye(4)	Identity matrix of 4 x 4
help	documentation

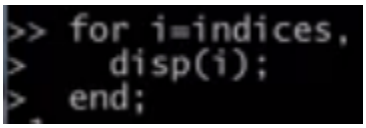
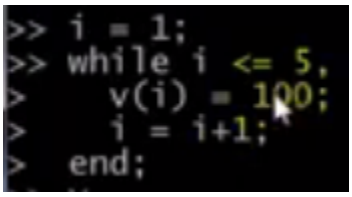
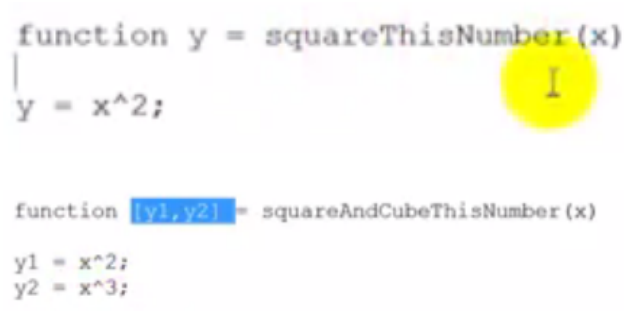
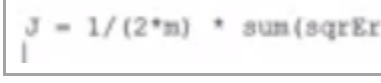
size(A)	#row #columns
length(v)	length of the longest dimension
pwd	current directory
ls	list directories as in Linux

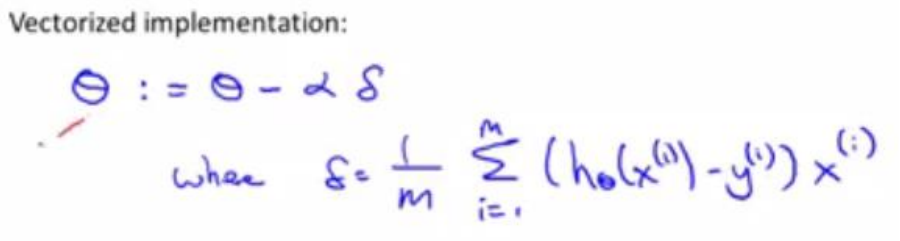
load(features.dat)	loads files
who	shows variable in current scope
whos	gives the detailed view of who
clear varname	removes varname
save hello.mat v	save variable v in hello.mat
load hello.mat	loods
save hello.txt	save as text ASCII
A(3,2)	Element in 3rd row and 2nd column
A(2, :) A(:, 2)	Get everything in second column Get everything in second row
A([1,3], :)	Get everything from first and third rows and all columns
A(:, 2) = [10; 11; 12]	Edit 2nd col
A = [A, [a; b; c]]	append another column
A(:)	Put all elements of A in single vector
C = [A B] or [A, B] C = [A; B]	concatenate A and B

A *C	Matrix multiplication
A .* B	Element wise multiplication
A.^2	Element wise squared
1 ./ V	Element wise reciprocal of V
log(v)	Element wise log
exp(v)	Element wise exp
abs(v)	Element wise abs
-v	-1 * v
A'	Transpose of A
[val, ind] = max(a)	Max value and index
val = max(a)	Column wise max

A < 3	element wise comparison
find(a < 3)	
magic(n)	n x n all rows cols and diags add up to same number
sum(a), prod(a), floor(a)	
max(A, [], 1)	Max of matrix col wise 1 -> first dimension
max(max(A)) or max(A(:))	Max of all the matrix
sum(A, 1) sum(A, 2)	sum of elements along dimensions 1 or 2
sum(sum(A .* eye(9))) sum(sum(A .* flipup(eye(9)))) // flipup, flipud	Sum of diagonal elements
pinv(a)	pseudo inverse

plot(t)	plot function
hold on;	plot new functions on the old ones
xlable, ylabel, legend, title	params of plot
print =dpng 'myplot.png'	save plot
close	get away from the figure displayed during plot
figure(1) figure(2)	Multiple figures
subplot(1,2,1)	first element of 1,2 grid
axis([a b c d])	a b for x range c d for y range
clf;	clears a figure
imagesc(A) imagesc(A), colorbar, colormap gray;	visualize a matrix A
,	chaining multiple commands in same line

	For
	While
break and continue	
	Function
addpath(dir)	find the functions
	Cost functions



Week 3

Monday, May 29, 2017
9:33 PM

Classification and Representation

a. Examples:
   i. Email -> Spam / Not Spam
   ii. Online Transaction -> Fraud /Not Fraud

iii. Tumor -> Malignant /Benign

b. Binary classification

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

0: "Negative Class" (e.g., benign tumor)
1: "Positive Class" (e.g., malignant tumor)

i. Yes/No Type of problems
ii. Positive/Negative assignment for 1 /0 is arbitrary.

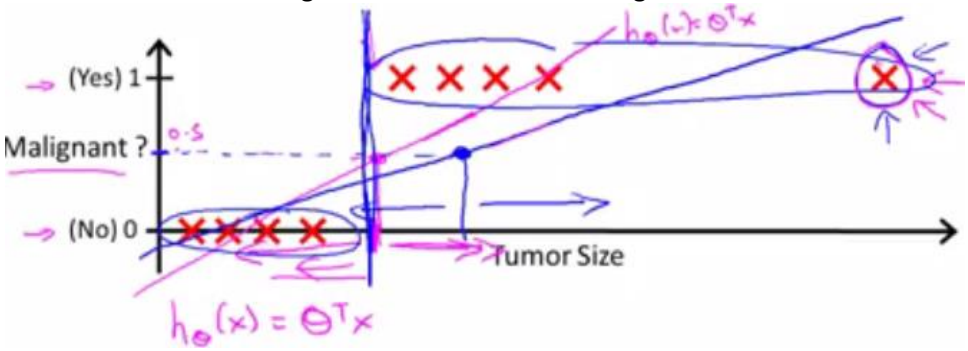
c. Threshold Classifier

Threshold classifier output $h_\theta(x)$ at 0.5:

i. → If $h_\theta(x) \geq 0.5$, predict "y = 1"

If $h_\theta(x) < 0.5$, predict "y = 0"

ii. Outliers in data can change the line that we are fitting.



iii. The output can be >> 1 or << 0 even though our labels are 0 and 1 only

Logistic Regression Model

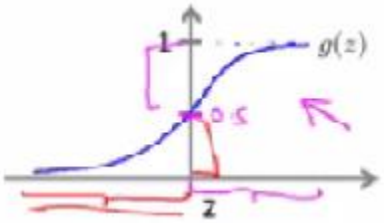
a. Hypothesis

g(z) is sigmoid or logistic function

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

$$z = \theta^T x$$

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



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

b. Decision Boundary

i. We predict

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

ii. But

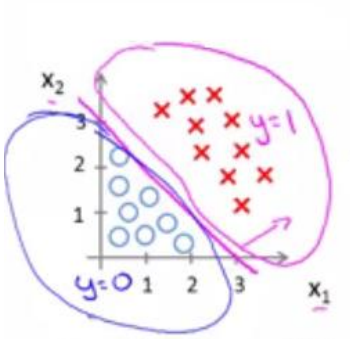
$$g(z) \geq 0.5$$
$$when\ z \geq 0$$

iii. Implies

$$h_\theta(x) = g(\theta^T x) \geq 0.5$$
$$when\ \theta^T x \geq 0$$

iv. Decision boundary is a property of the parameters but not the property of the Data. Once we have Theta that completely defines the decision boundary.

c. Linear Decision Boundary



$$\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

-3 1 1

Predict "$y = 1$" if $-3 + x_1 + x_2 \geq 0$

$\theta^T x$

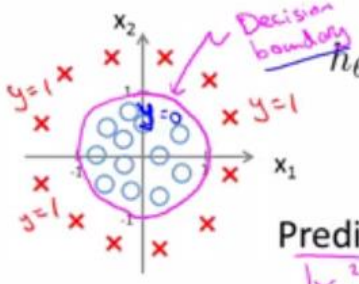
$x_1 + x_2 \geq 3$

$x_1 + x_2 = 3$

$x_1 + x_2 < 3$

x_1, x_2

d. Non Linear decision boundaries



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

-1 0 0 1 1

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

Predict "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$

$x_1^2 + x_2^2 = 1$

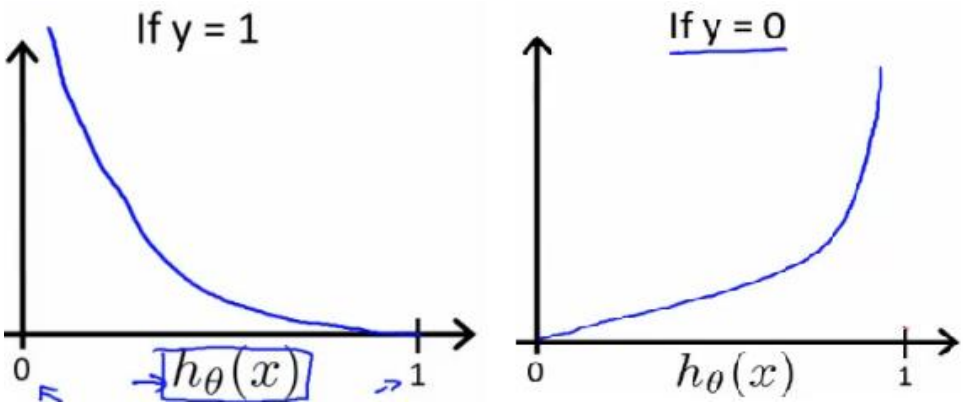
$x_1^2 + x_2^2 \geq 1$

e. Cost Function

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

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

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



f. Simplified Cost Function

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

A vectorized implementation is:

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

This cost function can be derived from principle of maximum likelihood. And this has nice property of being convex.

g. Gradient Descent

Gradient Descent

Remember that the general form of gradient descent is:

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

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

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

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

A vectorized implementation is:

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

h. Check Gradient Descent

Plot $J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$ as a function of the number of iterations and make sure $J(\theta)$ is decreasing on every iteration.

i. Optimization Algorithm for Gradient Descent
   i. Conjugate Gradient
   ii. BFGS
   iii. L-BFGS

   Advantages:
   i. No need to manually pick alpha
   ii. Often faster than gradient descent

   Disadvantages:
   i. More complex

   Octave:
   i. Write costFunction taking input theta and giving output jVal and gradient Vector

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

   ii. options = optimset('Gradobj','on','MaxIter','100');
   iii. initialTheta = zeros(<dim1>,<dim2>);

iv. [optTheta, functionVal, exitVal] = fminunc(@costFunction, initialTheta, options);

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

Multiclass Classification

a. Examples:
   i. Email Tagging
   ii. Medical Diagnosis -> Not ill, Cold, Flu
   iii. Weather -> Sunny, Cloudy, Rainy, Snow

b. One Vs All

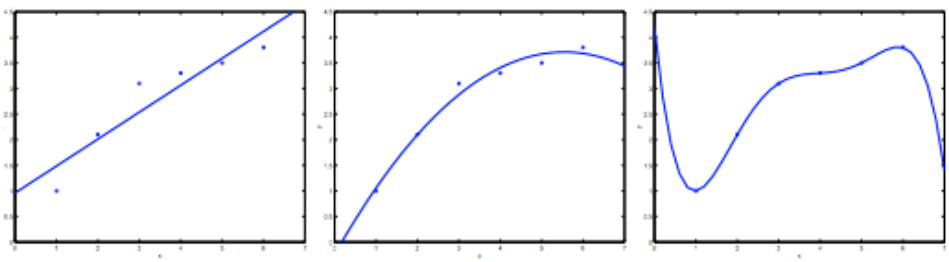
Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$.

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

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

$$y \in \{0, 1 \ldots n\}$$
$$h_\theta^{(0)}(x) = P(y = 0|x;\theta)$$
$$h_\theta^{(1)}(x) = P(y = 1|x;\theta)$$
$$\cdots$$
$$h_\theta^{(n)}(x) = P(y = n|x;\theta)$$
$$\text{prediction} = \max_i(h_\theta^{(i)}(x))$$

Solve the Problem of Overfitting:



a. Underfitting -> High Bias
   Hypothesis not even fitting the training set

   Overfitting -> High Variance
   Hypothesis fits training data well but fails to generalize

b. Addressing Overfitting:
   i. Reduce number of features
      Manually select features
      Model selection algorithm
   ii. Regularization
      Keep all features but reduce magnitude values of theta

c. Regularization

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n}\theta_j^2\right]$$

   Smaller values of theta
   i. Shrink the Parameters by adding additional theta terms.
   ii. Simpler hypothesis
   iii. Less prone to overfitting
   iv. We can't select the parameters for shrinking unless we know in prior. So we shrink all the parameters.
   v. Large value of lambda => underfitting
   vi. Note: theta0 is not penalized

d. Regularized Linear Regression:
   Cost Function:

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n}\theta_j^2\right]$$

   Gradient Descent

$$\text{Repeat } \{$$

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

$$\theta_j := \theta_j - \alpha\left[\left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j\right] \qquad j \in \{1,2...n\}$$

$$\}$$

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

Normal Equation

$$\theta = \left(X^T X + \lambda \cdot L\right)^{-1} X^T y$$

$$\text{where } L = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}$$

Note: Below matrix is always invertible

$$\left(X^T X + \lambda \cdot L\right)^{-1}$$

e. Regularized Logistic Regression:
Cost Function:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\ \log(h_\theta(x^{(i)})) + (1-y^{(i)})\ \log(1-h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Gradient Descent:

Repeat {

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

$$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j\right] \quad (j = 1, 2, 3, \ldots, n)$$

}

$\theta_1 \ldots \theta_n$

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

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

Week 4

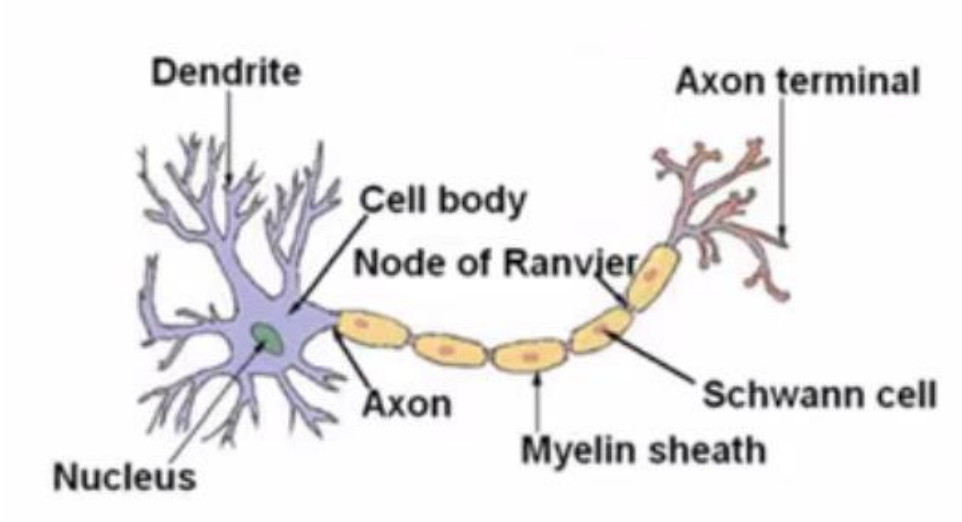
Sunday, June 04, 2017
9:54 AM

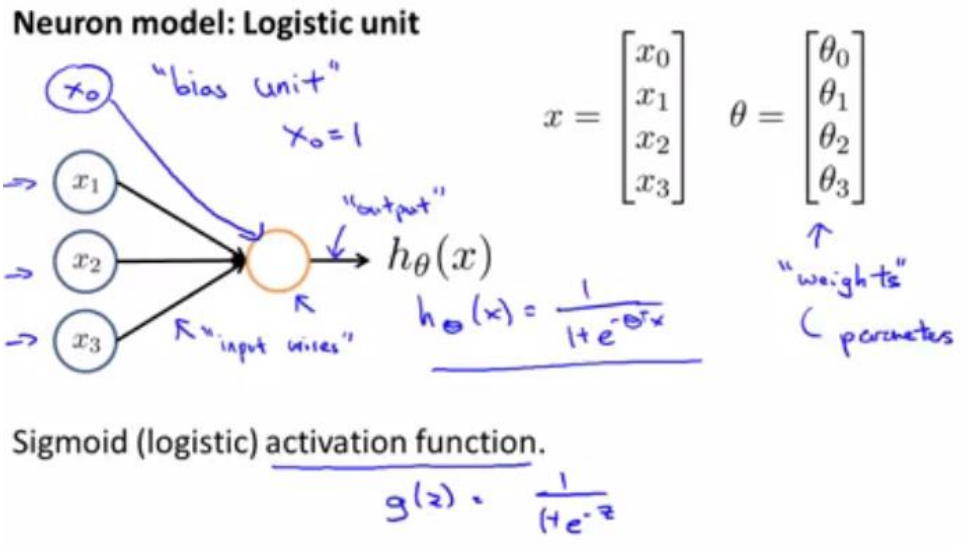
Neural Networks

1. Motivations:
    a. Non Linear Classification
        i. Number of quadratic features increase as O(n^2)
        ii. Also Cubic features increase as O(n^3)
        iii. Soo the number if features to consider is large

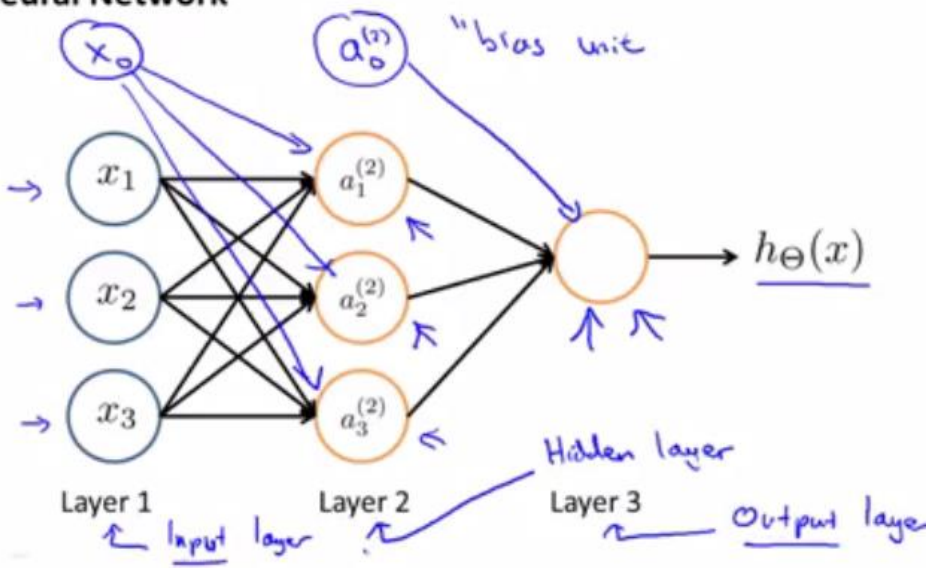
b. Computer Vision
    i. Features are not linearly separable
c. Neurons and Brain: Mimics Brain
    i. Neurons learn to do the respective work.
    ii. Auditory part of brain can learn to see.

2. Model Representation:

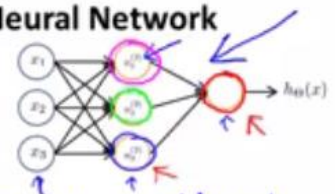




Neural Network



Neural Network



- $a_i^{(j)}$ = "activation" of unit i in layer j
- $\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j+1$

3 units, 3 hidden units. $\Theta^{(1)} \in \mathbb{R}^{3\times 4}$

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

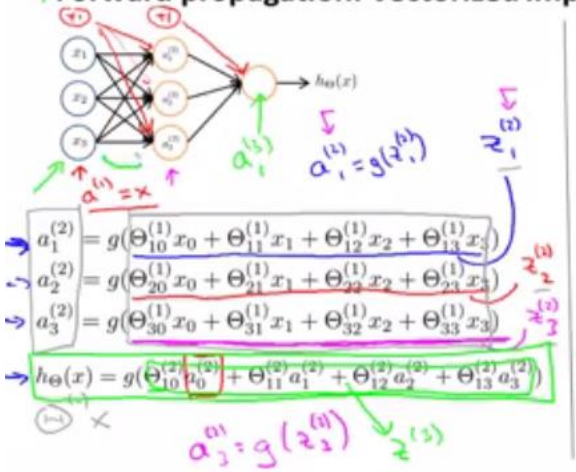
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$. $s_{j+1} \times (s_j + 1)$

Forward propagation: Vectorized implementation



$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

$$h_\Theta(x) = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

Add $a_0^{(2)} = 1$.

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$

$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

Note:

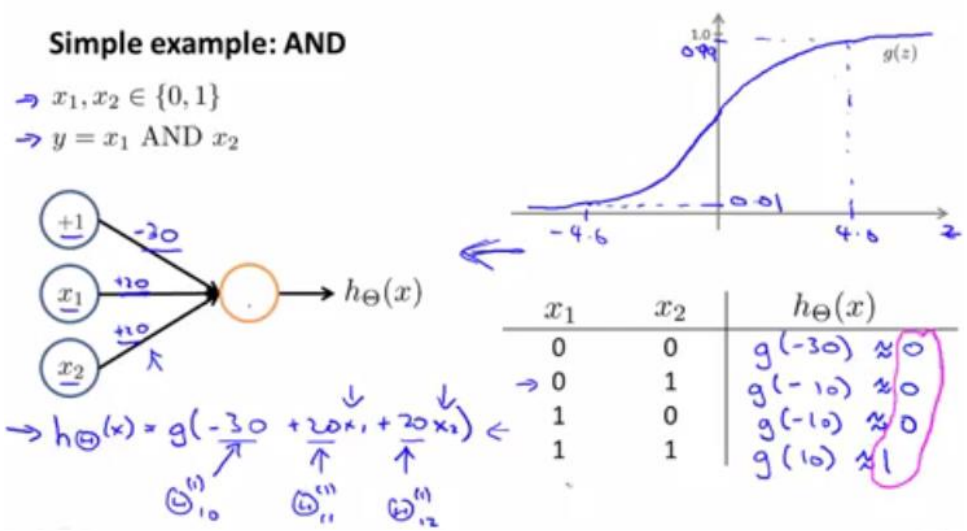
a. $z^{(j)} = \Theta^{(j-1)} a^{(j-1)}$

b. Neural networks will learn its own features. These features are formed from the input features.

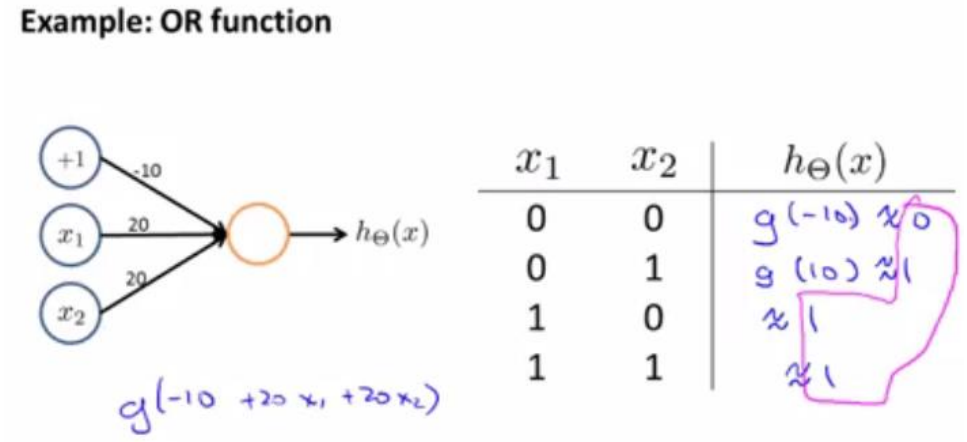
c. We can have neurons connected differently i.e. different architectures.

3. Examples and Intuitions:

a. AND



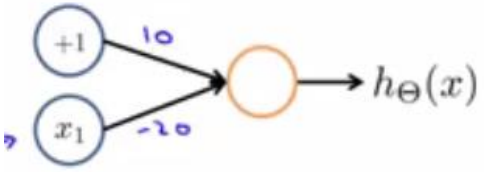
b. OR



c. NOT

Negation:

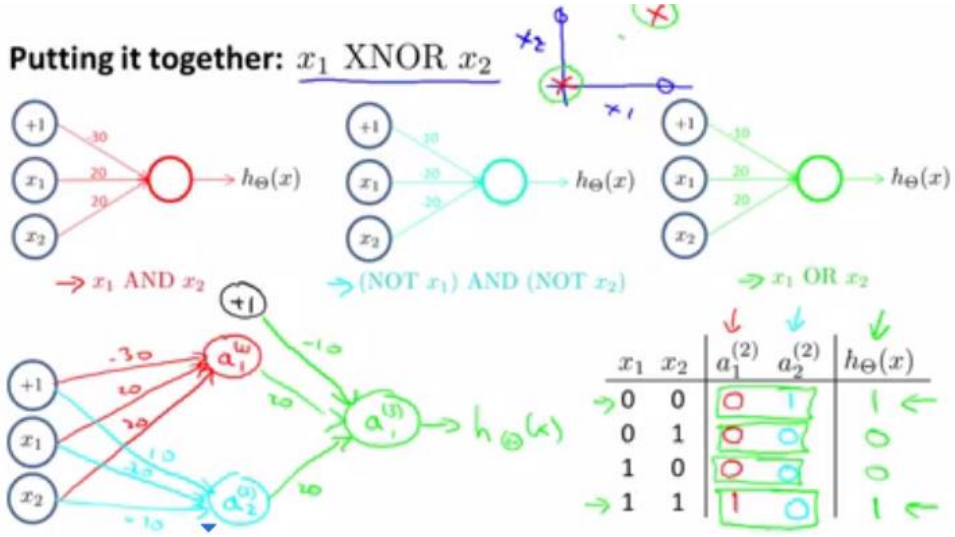
NOT x_1



x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

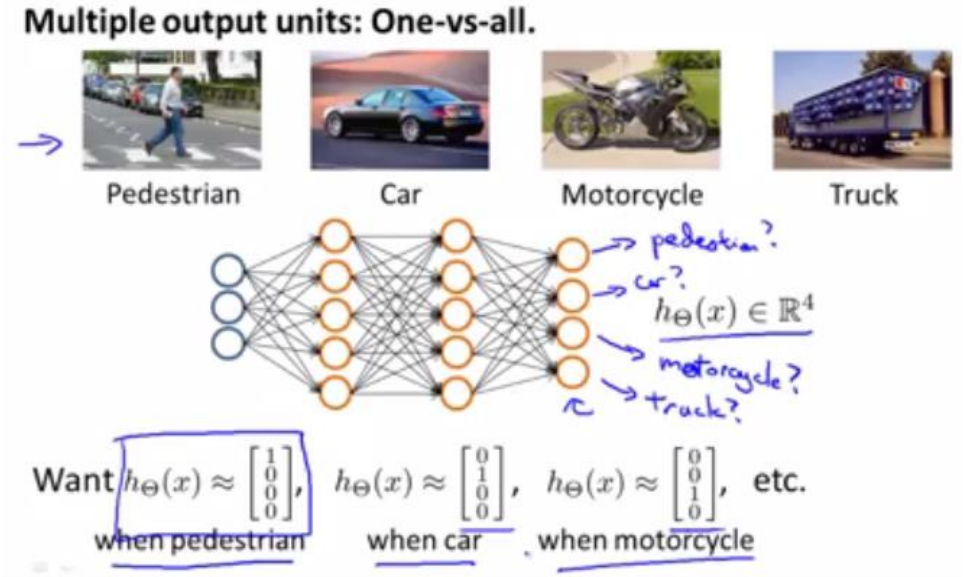
$$h_\Theta(x) = g(10 - 20x_1)$$

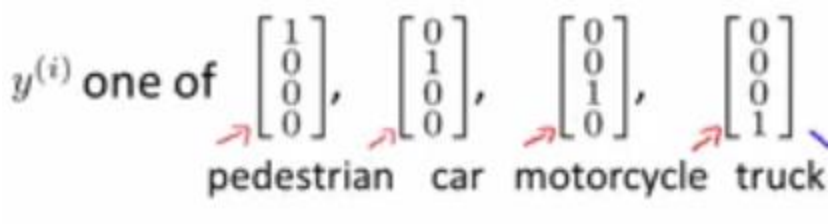
d. XNOR



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

4. Multiclass Classification in Neural networks





Week 5

Sunday, June 04, 2017
6:42 PM

Cost Function and Backpropagation.

Cost Function:

1. Definitions:

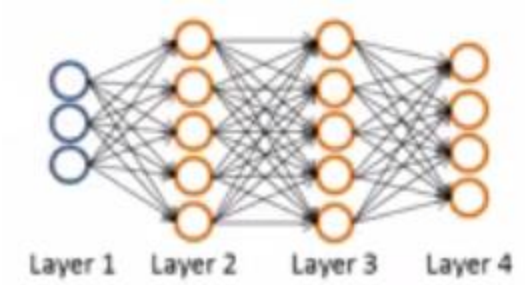
- L = total number of layers in the network
- s_l = number of units (not counting bias unit) in layer l
- K = number of output units/classes

2. Cost Function

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\ \log(h_\theta(x^{(i)})) + (1-y^{(i)})\ \log(1-h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log((h_\Theta(x^{(i)}))_k) + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{j,i}^{(l)})^2$$

3. Backpropagation Algorithm

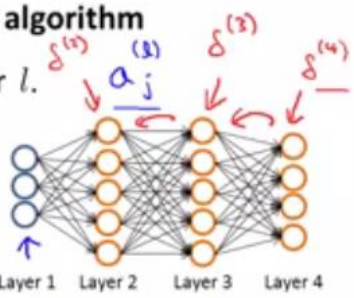


Forward propagation:

$$
\begin{aligned}
a^{(1)} &= x \\
z^{(2)} &= \Theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)}) \quad (\text{add } a_0^{(2)}) \\
z^{(3)} &= \Theta^{(2)} a^{(2)} \\
a^{(3)} &= g(z^{(3)}) \quad (\text{add } a_0^{(3)}) \\
z^{(4)} &= \Theta^{(3)} a^{(3)} \\
a^{(4)} &= h_\Theta(x) = g(z^{(4)})
\end{aligned}
$$

Gradient computation: Backpropagation algorithm

Intuition: $\delta_j^{(l)}$ = "error" of node j in layer l.



For each output unit (layer L = 4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

(No $\delta^{(1)}$)

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

(ignoring λ; if $\lambda = 0$)

Backpropagation algorithm

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j). (used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)

For $i = 1$ to m ← $(x^{(i)}, y^{(i)})$

- Set $a^{(1)} = x^{(i)}$
- Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$
- Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$
- Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$
- $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ ← $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$ if $j \neq 0$

$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$ if $j = 0$

$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$

4. Implementation Note: Unrolling Parameters

In order to use optimizing functions such as "fminunc()", we will want to "unroll" all the elements and put them into one long vector:

```
thetaVector = [ Theta1(:); Theta2(:); Theta3(:); ]
deltaVector = [ D1(:); D2(:); D3(:) ]
```

If the dimensions of Theta1 is 10x11, Theta2 is 10x11 and Theta3 is 1x11, then we can get back our original matrices from the "unrolled" versions as follows:

```
Theta1 = reshape(thetaVector(1:110),10,11)
Theta2 = reshape(thetaVector(111:220),10,11)
Theta3 = reshape(thetaVector(221:231),1,11)
```

Learning Algorithm

- Have initial parameters $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$.
- Unroll to get `initialTheta` to pass to
- `fminunc(@costFunction, initialTheta, options)`

```
function [jval, gradientVec] = costFunction(thetaVec)
```

From `thetaVec`, get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$.

Use forward prop/back prop to compute $D^{(1)}, D^{(2)}, D^{(3)}$ and $J(\Theta)$

Unroll $D^{(1)}, D^{(2)}, D^{(3)}$ to get `gradientVec`.

5. Gradient Checking
   a. Very slow. Use only to verify once.

b. $$\frac{\partial}{\partial \Theta} J(\Theta) \approx \frac{J(\Theta+\epsilon) - J(\Theta-\epsilon)}{2\epsilon}$$

```
epsilon = 1e-4;
for i = 1:n,
  thetaPlus = theta;
  thetaPlus(i) += epsilon;
  thetaMinus = theta;
  thetaMinus(i) -= epsilon;
  gradApprox(i) = (J(thetaPlus) - J(thetaMinus))/(2*epsilon)
end;
```

6. Random Initialization:

Random initialization: Symmetry breaking

- Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\epsilon, \epsilon]$ (i.e. $-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon$)

E.g. (random 10×11 matrix (betw. 0 and 1))

```
Theta1 = rand(10,11)*(2*INIT_EPSILON)
         - INIT_EPSILON;
```

([-ε, ε])

```
Theta2 = rand(1,11)*(2*INIT_EPSILON)
         - INIT_EPSILON;
```

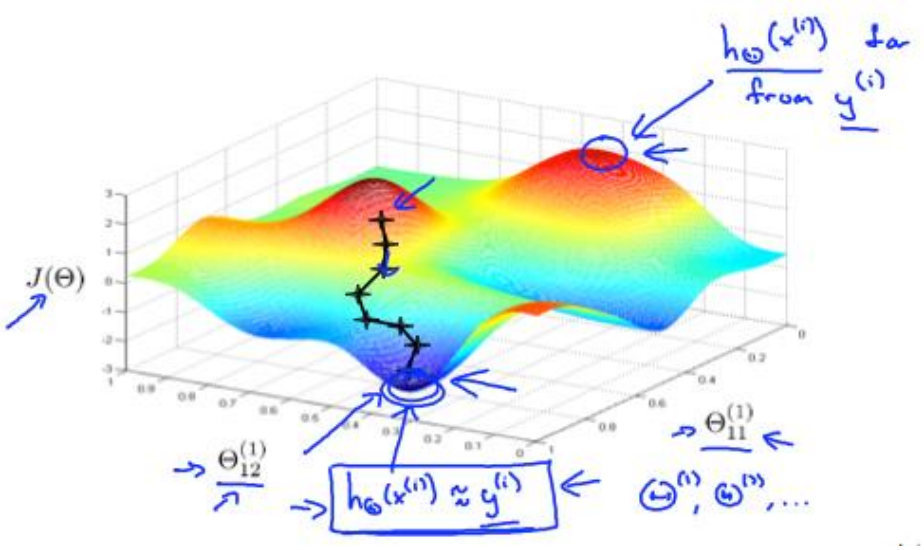
7. Putting it Together

Training a Neural Network

1. Randomly initialize the weights
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement the cost function
4. Implement backpropagation to compute partial derivatives
5. Use gradient checking to confirm that your backpropagation works. Then disable gradient checking.
6. Use gradient descent or a built-in optimization function to minimize the cost function with the weights in theta.

When we perform forward and back propagation, we loop on every training example:

```
for i = 1:m,
   Perform forward propagation and backpropagation using example (x(i),y(i))
   (Get activations a(l) and delta terms d(l) for l = 2,...,L
```



J(theta) is not convex we can get stuck in local minima.

-> Activation of node j in layer l

-> Error due to node j in layer l

Week 6

Wednesday, June 28, 2017
1:56 AM

Evaluating the Learning Algorithm:

1. Deciding what to try next:

   Suppose you have implemented regularized linear regression to predict housing prices.

   $$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{m}\theta_j^2\right]$$

   However, when you test your hypothesis on a new set of houses, you find that it makes unacceptably large errors in its predictions. What should you try next?

   a. Get more training examples
   b. Try smaller set of features
   c. Try getting additional features
   d. Try adding polynomial features
   e. Try decreasing lambda -> Regularization
   f. Try increasing lambda

2. Machine Learning Diagnostic

Diagnostic: A test that you can run to gain insight what is/isn't working with a learning algorithm, and gain guidance as to how best to improve its performance.

3. Evaluating a Hypothesis:
   a. Low training error does not imply good hypothesis.
   b. We can plot the curve and check for overfitting
      i. Not possible when there are multiple variables
   c. How to Evaluate ?
      i. Split the data into Training Set and Test Set in 7:3 Ratio.
      ii. mtest = number of test examples
      iii. If the data is overfitting train error will be low and test error will be high.
   d. Procedure
      i. Learn parameter theta from training data minimizing training error J(theta)
      ii. Compute test set error:

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left(h_\theta(x_{test}^{(i)}) - y_{test}^{(i)} \right)^2$$

$$J_{test}(\theta) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} y_{test}^{(i)} \log h_\theta(x_{test}^{(i)}) + (1 - y_{test}^{(i)}) \log h_\theta(x_{test}^{(i)})$$

In case of logistic regression you can use:
Misclassification Error (0/1 misclassification error)

$$err(h_\theta(x), y) = \begin{cases} 1 & \text{if } h_\theta(x) \geq 0.5, \ y = 0 \text{ or if } h_\theta(x) < 0.5, \ y = 1 \\ 0 & \text{otherwise} \end{cases} \quad \text{error}$$

$$\text{Test error} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_\theta(x_{test}^{(i)}), y_{test}^{(i)})$$

4. Model Selection and Train/Validation/Test Sets
   a. Once parameters theta0, theta1,.. Theta4 were fir to some set of data (training set), the error of the parameters as measured on that data (the training error J(theta)) is likely to be lower than the actual generalization error.
   b. Model Selection

$d=1$ 1. $\rightarrow h_\theta(x) = \theta_0 + \theta_1 x$

$d=2$ 2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$

$d=3$ 3. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_3 x^3$

$\vdots$ $\vdots$

$d=10$ 10. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10}$

Take theta values for all the models.
Measure their performance in the test set.
How well does this model generalize ?
Report test set error.

c. Problem:
We chose the parameter d to test set. The hypothesis will overfit for the test set while selecting d.
Solution:
Split it into three pieces Test Set/ Validation Set/ Test in ration say 6 : 2 : 2
Training/ validation / test error:

Training error:

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Cross Validation error:

$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

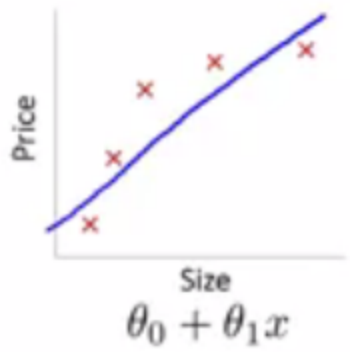
Test error:

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

d. Now use Validation test to get more hyper parameters.
Test the models on validation test.
Pick up the model with lowest Cross validation error.
And use Test set to evaluate.
Use different data for all the Sets above.

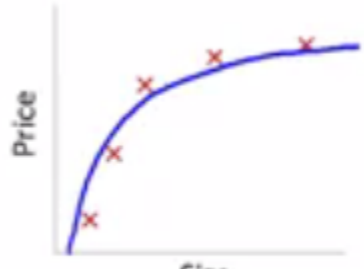
e. Procedure
   i. Optimize the parameters in Θ using the training set for each polynomial degree.
   ii. Find the polynomial degree d with the least error using the cross validation set.
   iii. Estimate the generalization error using the test set with Jtest(Θ(d)), (d = theta from polynomial with lower error);

5. Diagnosing Bias vs Variance
   a. High Bias:



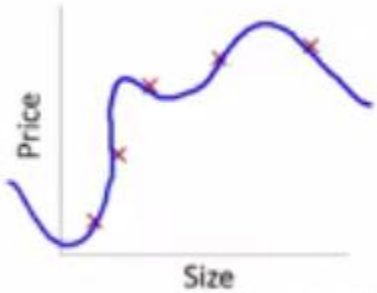
$$\theta_0 + \theta_1 x$$

b. Just Right



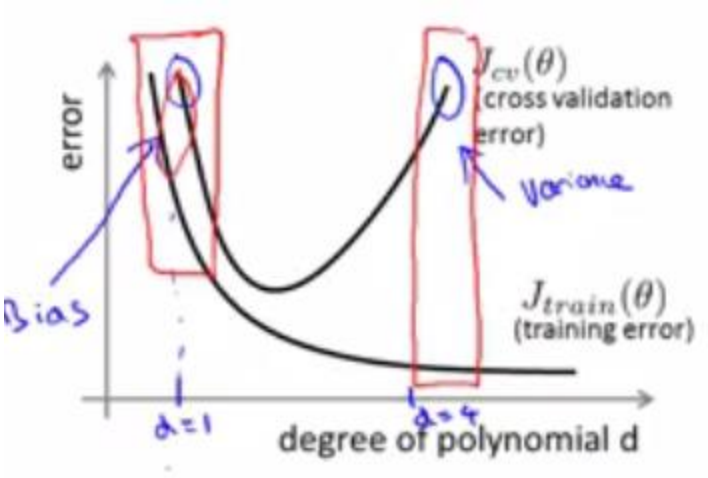
$$\theta_0 + \theta_1 x + \theta_2 x^2$$

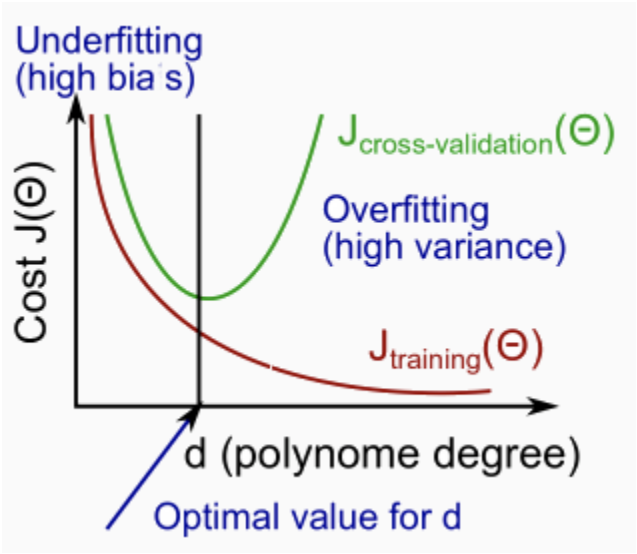
c. High Variance:

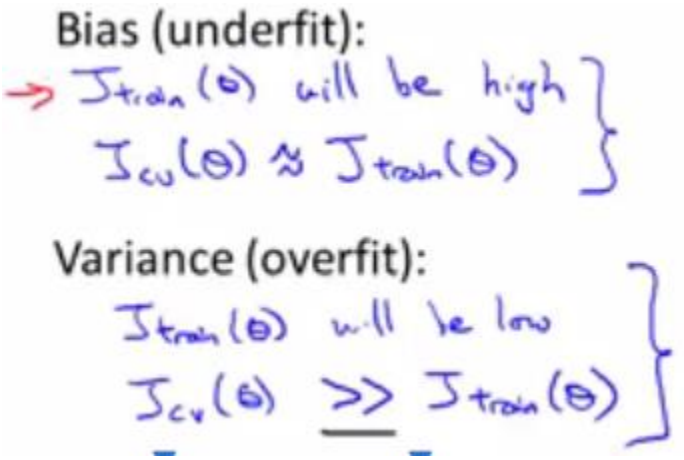


$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

d. Plot Error vs Degree of polynomial d.

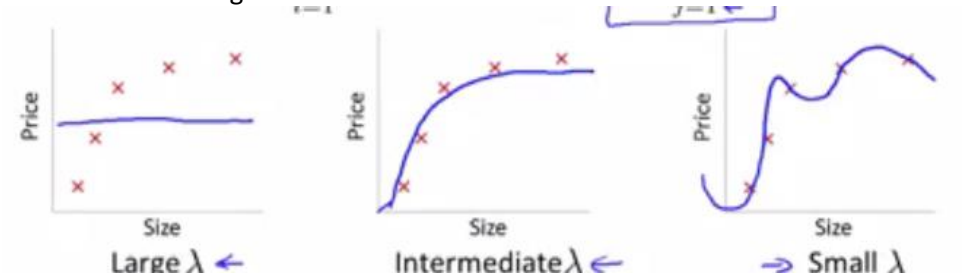






6. Regularization and Bias/Variance
   Large lambda => High Bias => Underfit
   Intermediate lambda => Just Right

Small lambda => High Variance => overfit



Model selection for Lambda

Model: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

1. Try $\lambda = 0$ → $\min_\theta J(\theta)$ → $\theta^{(1)}$ → $J_{cv}(\theta^{(1)})$
2. Try $\lambda = 0.01$ → $\min_\theta J(\theta)$ → $\theta^{(2)}$ → $J_{cv}(\theta^{(2)})$
3. Try $\lambda = 0.02$ → $\theta^{(3)}$ → $J_{cv}(\theta^{(3)})$
4. Try $\lambda = 0.04$
5. Try $\lambda = 0.08$ → $\theta^{(5)}$

⋮

12. Try $\lambda = 10$ → $\theta^{(12)}$ → $J_{cv}(\theta^{(12)})$

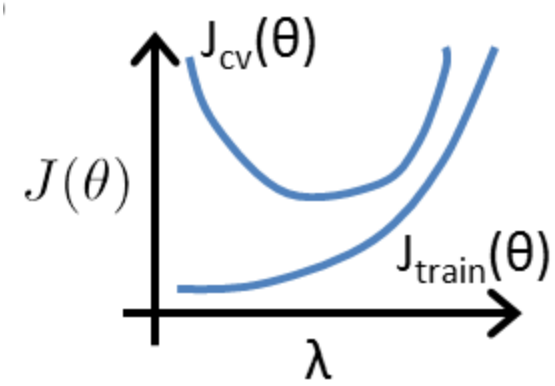
↑ 10.24

Pick (say) $\theta^{(5)}$. Test error: $J_{test}(\theta^{(5)})$

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

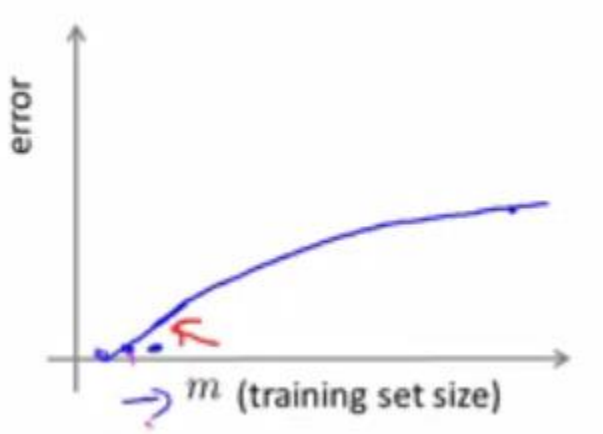
$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$



Procedure:

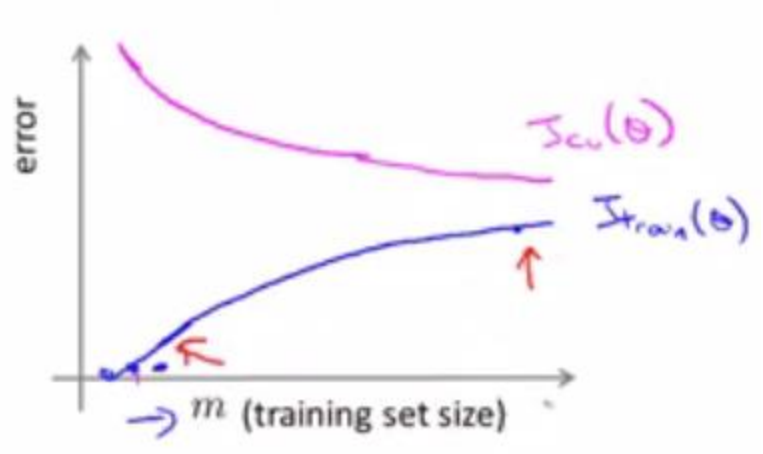
a. Create a list of lambdas (i.e. λ∈{0,0.01,0.02,0.04,0.08,0.16,0.32,0.64,1.28,2.56,5.12,10.24});
b. Create a set of models with different degrees or any other variants.
c. Iterate through the λs and for each λ go through all the models to learn some Θ.
d. Compute the cross validation error using the learned Θ (computed with λ) on the JCV(Θ) **without** regularization or λ = 0.
e. Select the best combo that produces the lowest error on the cross validation set.
f. Using the best combo Θ and λ, apply it on Jtest(Θ) to see if it has a good generalization of the problem.

7. Learning Curves:
   a. Training set error:

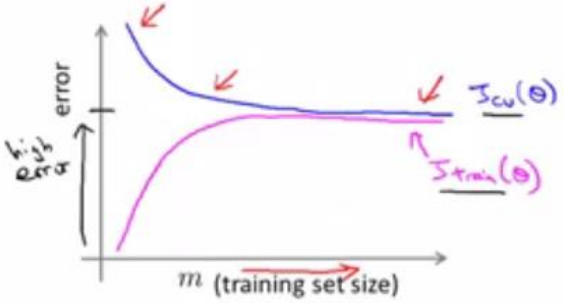


Easy to fit smaller data.

   b. Validation or Test error;

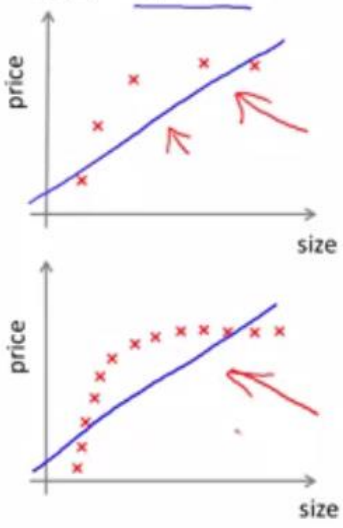


High bias

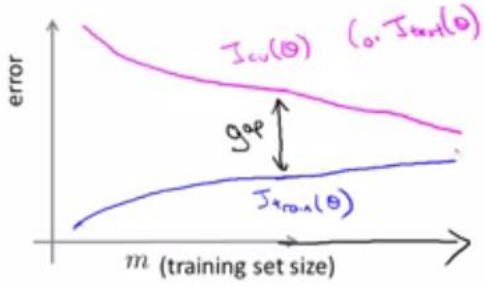


If a learning algorithm is suffering from high bias, getting more training data will not (by itself) help much.

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

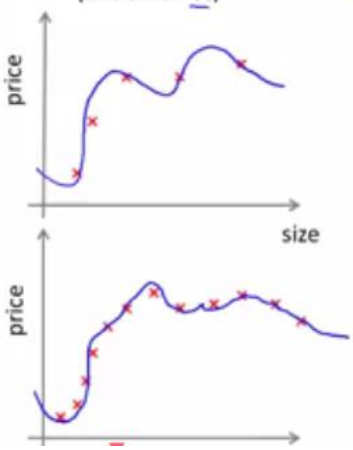


High variance



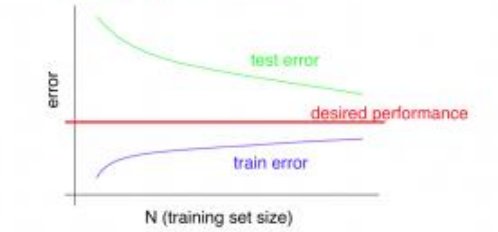
If a learning algorithm is suffering from high variance, getting more training data is likely to help.

$h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{100} x^{100}$
(and small λ)



Typical learning curvefor high bias(at fixed model complexity):

Typical learning curve for high variance(at fixed model complexity):

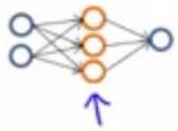


8. Deciding What to Do Next:

Get more training examples → fixes high variance
Try smaller sets of features → fixes high variance
Try getting additional features → fixes high bias
Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, \text{etc})$ → fixes high bias.
Try decreasing λ → fixes high bias
Try increasing λ → fixes high variance

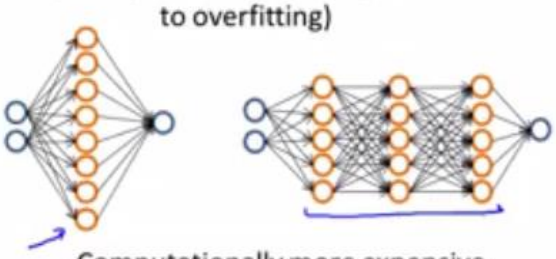
Neural Networks and Overfitting:

"Small" neural network (fewer parameters; more prone to underfitting)



Computationally cheaper

→ "Large" neural network (more parameters; more prone to overfitting)



Computationally more expensive.

Use regularization (λ) to address overfitting.

Diagnosing Neural Networks

- A neural network with fewer parameters is **prone to underfitting**. It is also **computationally cheaper**.
- A large neural network with more parameters is **prone to overfitting**. It is also **computationally expensive**. In this case you can use regularization (increase λ) to address the overfitting.

Using a single hidden layer is a good starting default. You can train your neural network on a number of hidden layers using your cross validation set. You can then select the one that performs best.

Model Complexity Effects:

- Lower-order polynomials (low model complexity) have high bias and low variance. In this case, the model fits poorly consistently.
- Higher-order polynomials (high model complexity) fit the training data extremely well and the test data extremely poorly. These have low bias on the training data, but very high variance.
- In reality, we would want to choose a model somewhere in between, that can generalize well but also fits the data reasonably well.

Machine Learning System Design

1. Building a spam classifier

    a. Supervised Learning:
    X => features of email
    Y => spam (1) or not spam (0)

    b. How to spend time:
        i. Collect lots of data
        ii. Develop sophisticated features based on email routing
        iii. Develop sophisticated features based on message body
        iv. Develop sophisticated algorithm to detect misspelling
        It is difficult to tell which is more useful

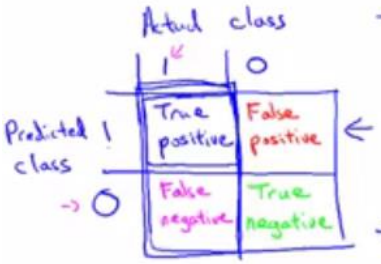
2. Recommended approach:
    a. Start with a simple algorithm that you can implement quickly. Implement it and test it on your cross validation data.
    b. Plot learning curves to decide if more data, more features etc. are likely to help
    c. Error Analysis: Manually examine the examples in cross validation set that your algorithm mode errors on. See if you spot any systematic trend in what type of examples it is making errors on.

3. Error Metrics for Skewed Classes:

a. Consider cancer classification example
Train logistic regression model y = 1 if cancer or y =0 otherwise
Find that got 1% Error on test set
Only 0.50% of patients have cancer
Y = 0 always => 0.5% Error.

b. Precision/Recall

$y = 1$ in presence of rare class that we want to detect



→ **Precision**
(Of all patients where we predicted $y = 1$, what fraction actually has cancer?)

$$\frac{\text{True positives}}{\#\text{predicted positive}} = \frac{\text{True positive}}{\text{True pos + False pos}}$$

→ **Recall**
(Of all patients that actually have cancer, what fraction did we correctly detect as having cancer?)

$$\frac{\text{True positives}}{\#\text{actual positives}} = \frac{\text{True positives}}{\text{True pos + False neg}}$$

c. Trading Off Precision and Recall.

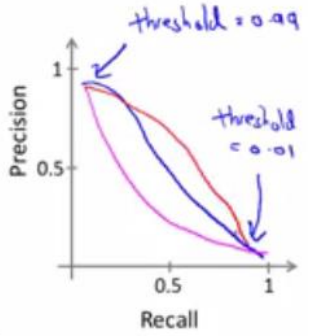
Trading off precision and recall

- Logistic regression: $0 \le h_\theta(x) \le 1$
  Predict 1 if $h_\theta(x) \ge 0.5$ 0.7 0.9 0.3
  Predict 0 if $h_\theta(x) < 0.5$ 0.7 0.9 0.3
- Suppose we want to predict $y = 1$ (cancer) only if very confident.
  → Higher precision, lower recall.
- Suppose we want to avoid missing too many cases of cancer (avoid false negatives).
  → Higher recall, lower precision.

More generally: Predict 1 if $h_\theta(x) \ge$ threshold.

→ precision = $\frac{\text{true positives}}{\text{no. of predicted positive}}$

→ recall = $\frac{\text{true positives}}{\text{no. of actual positive}}$



d. How to compare precision/recall numbers ?

	Precision(P)	Recall (R)	Average	F_1 Score
Algorithm 1	0.5	0.4	0.45	0.444
Algorithm 2	0.7	0.1	0.4	0.175
Algorithm 3	0.02	1.0	0.51	0.0392

Predict y=1 all the time

Average: $\frac{P+R}{2}$

F_1 Score: $2\frac{PR}{P+R}$

P=0 or R=0 ⇒ F-score = 0.

P=1 and R=1 ⇒ F-score = 1

4. Data for High Accuracy learning system

   Classify between confusable words {to, two, too}, {then, than}
   Algorithms Used:
   Perceptron, Winnow, Memory-based, Naïve Bayes
   [Banko and Brill, 2001]: Increasing data => Increase in Perf

   Features must represent data and many features => Low bias
   Large training set => Low variance

Week 7

Thursday, June 29, 2017
6:28 PM

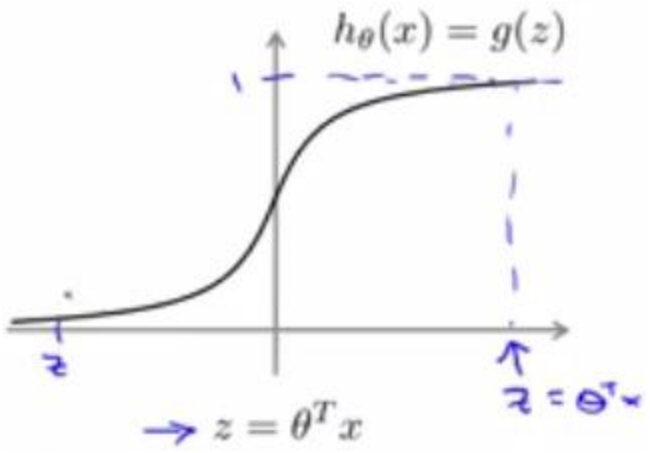
Support Vector Machines:

1. Optimization Objective (Mathematical Definition of SVMs)
   a. Alternative view of logistic regression
      Sigmoid function:

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

If $y = 1$, we want $h_\theta(x) \approx 1$, $\theta^T x \gg 0$

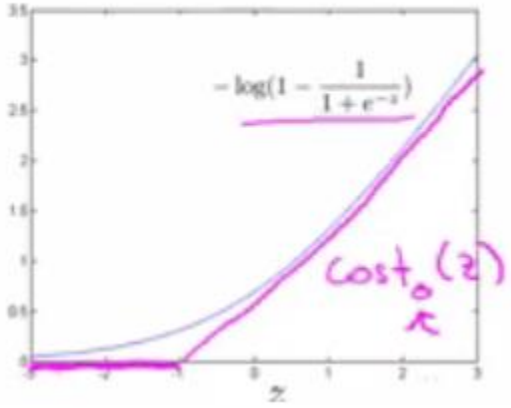
If $y = 0$, we want $h_\theta(x) \approx 0$, $\theta^T x \ll 0$



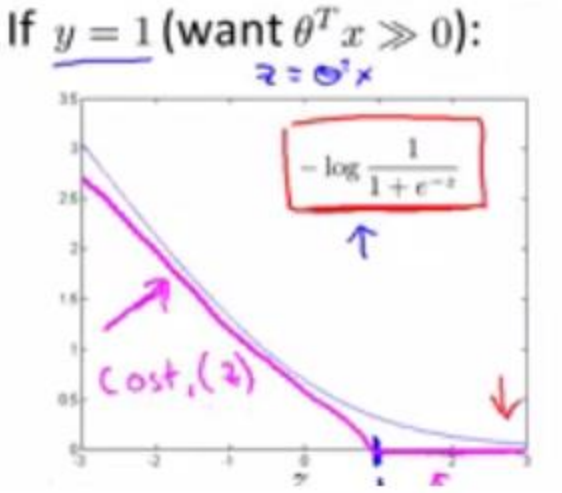
Cost of example: $-(y \log h_\theta(x) + (1-y)\log(1-h_\theta(x)))$

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

If $y = 0$ (want $\theta^T x \ll 0$):



If $y = 1$ (want $\theta^T x \gg 0$):



Logistic regression:

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\underbrace{\left(-\log h_\theta(x^{(i)})\right)}_{\text{cost}_1(\theta^T x^{(i)})} + (1-y^{(i)})\underbrace{\left(-\log(1-h_\theta(x^{(i)}))\right)}_{\text{cost}_0(\theta^T x^{(i)})}\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Support vector machine:

$$\min_\theta \frac{1}{m}\sum_{i=1}^{m} y^{(i)}\,\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\,\text{cost}_0(\theta^T x^{(i)}) + \frac{\lambda}{2m}\sum_{i=0}^{n}\theta_j^2$$

We can assume Logistic regression as A + (lambda) * B
SVMs use a different way of regularization which is like C * A + B

$$A + \lambda B \qquad C = \frac{1}{\lambda}$$
$$\rightarrow C\,A + B$$

b. Hypothesis:

SVM Does not predict the Probability but gives directly 0 or 1

SVM hypothesis

$$\rightarrow \min_\theta C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

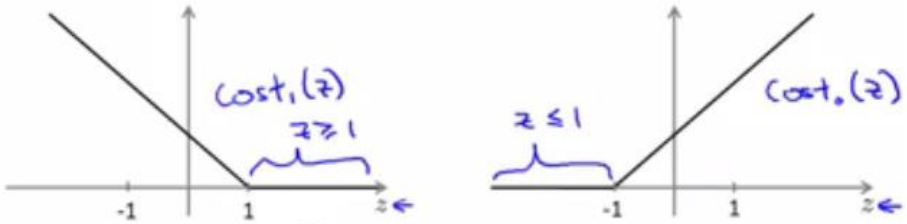
Hypothesis:

$h_\theta(x)$ { 1 if $\theta^T x \geq 0$
0 otherwise

2. Large Margin Intuition:
We need the prediction to be confident i.e. > or < 0 is not enough we need > or < 1

Support Vector Machine

$$\rightarrow \min_\theta C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$



If $y = 1$, we want $\theta^T x \geq 1$ (not just ≥ 0)
If $y = 0$, we want $\theta^T x \leq -1$ (not just < 0)

SVM Decision Boundary

$$\min_\theta C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

$= 0$

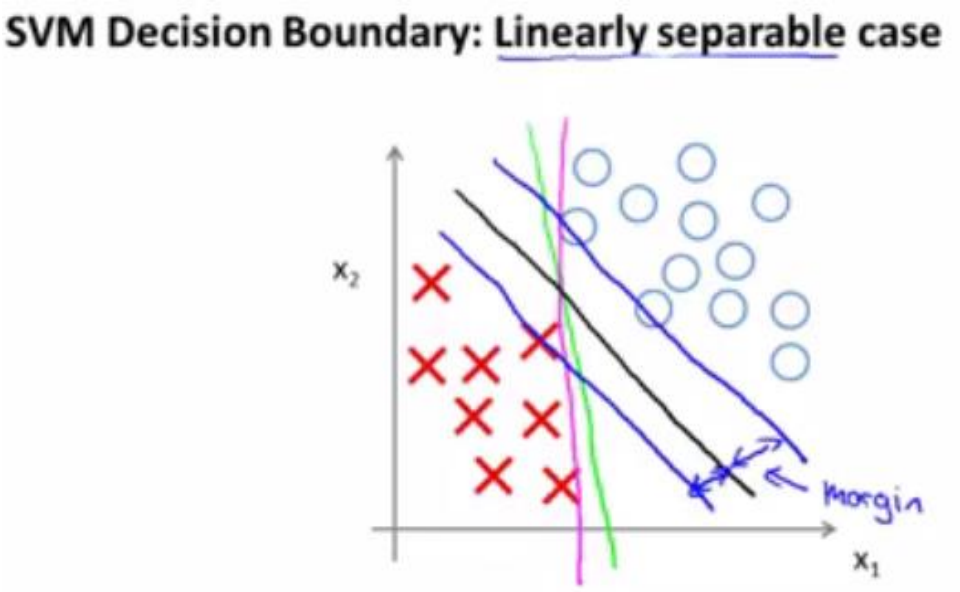
Whenever $y^{(i)} = 1$:
$\theta^T x^{(i)} \geq 1$

$\min_\theta$ ~~$C \cdot 0$~~ $+ \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$
s.t. $\theta^T x^{(i)} \geq 1$ if $y^{(i)} = 1$
$\theta^T x^{(i)} \leq -1$ if $y^{(i)} = 0$

Whenever $y^{(i)} = 0$:
$\theta^T x^{(i)} \leq -1$

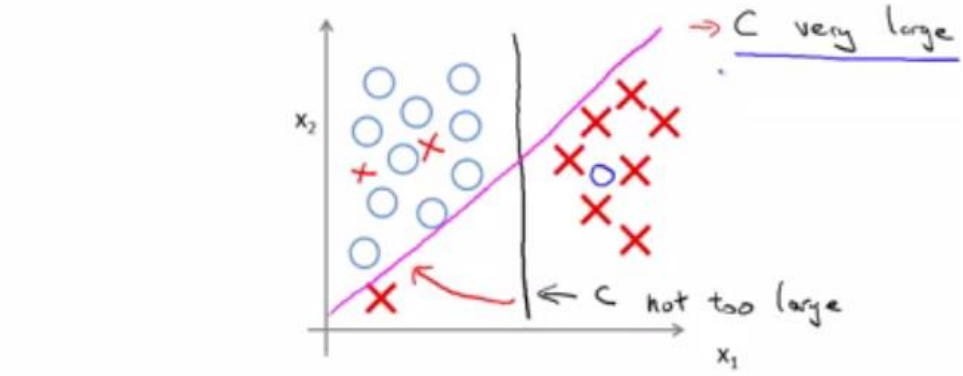
This Optimization of the cost function will lead to a large margin classifier as below
SVMs select the Black Decision Boundary but not the Magenta or the Green ones.

SVM Decision Boundary: Linearly separable case

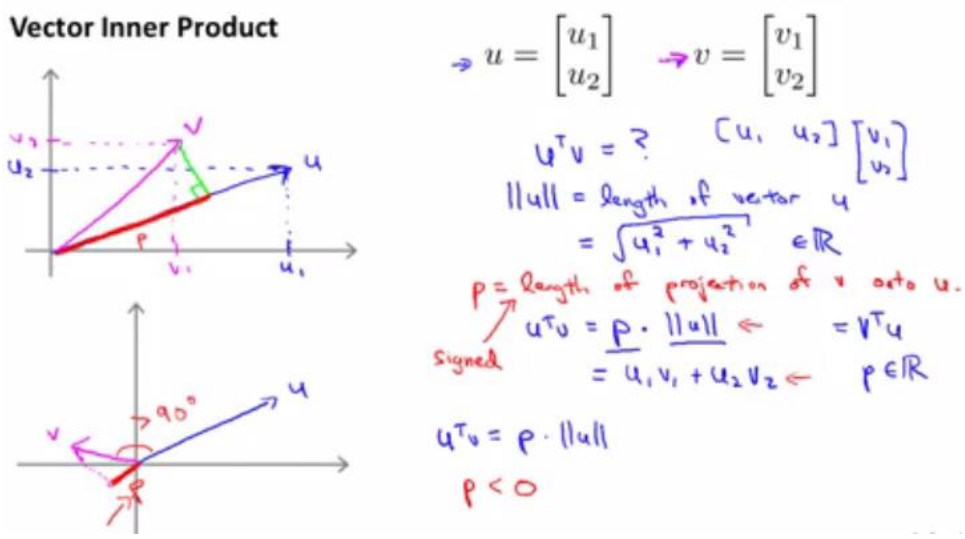


Large margin classifier

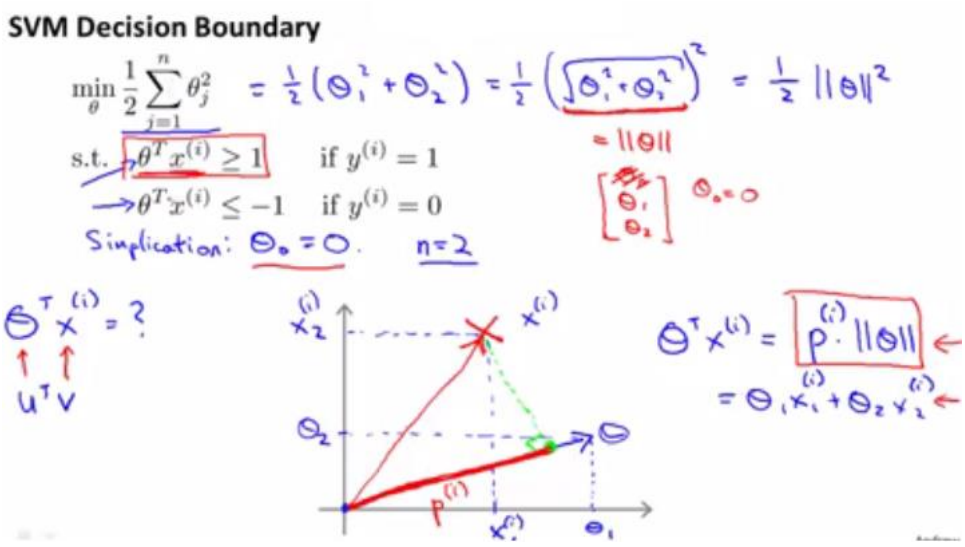
Large margin classifier in presence of outliers

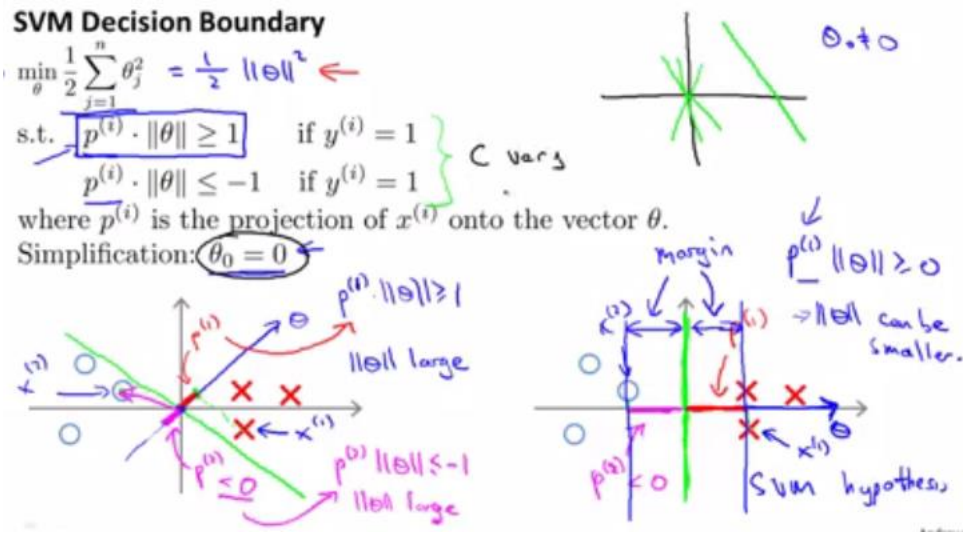


3. Mathematics Behind Large Margin Classifier



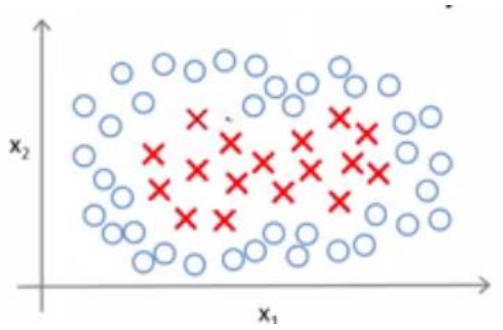
In the below diagram while computing the decision boundary we are trying to get the length of the theta vector.





Kernels:

1. Non Linear decision boundary
   a. We can add high order polynomial

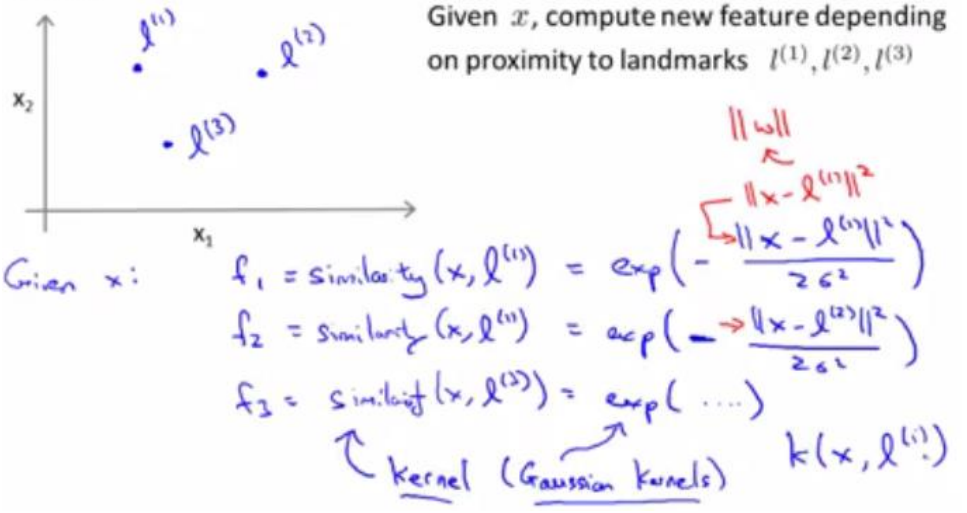


Using high order polynomial makes things computationally expensive.

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \cdots \geq 0$$

   b. Instead of selecting new polynomial features
   We can choose few points calling them landmarks now compute the new features based on the proximity to the landmarks.

Kernel



In the below the new feature i.e. Similarity is the gaussian distribution peaking at landmarks

Kernels and Similarity

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(1)})^2}{2\sigma^2}\right)$$

If $x \approx l^{(1)}$:

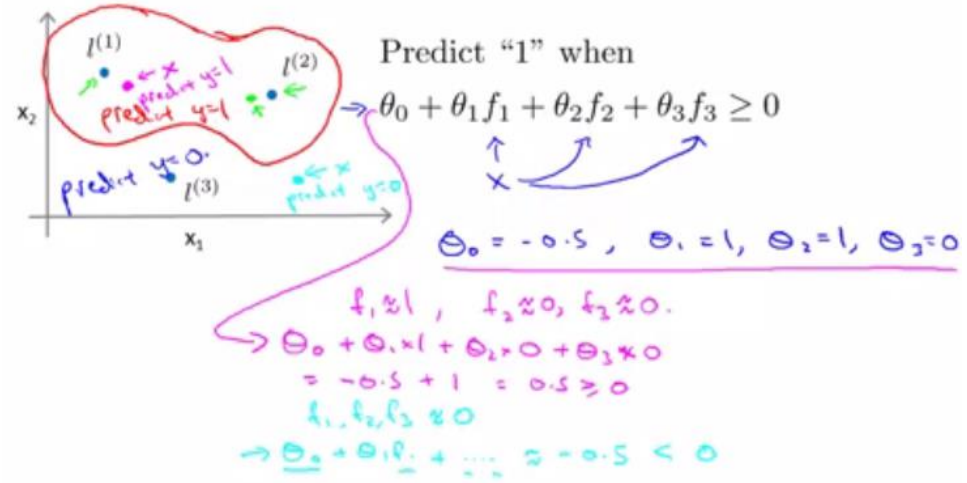
$$f_1 \approx \exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$$

$l^{(1)} \rightarrow f_1$
$l^{(2)} \rightarrow f_2$
$l^{(3)} \rightarrow f_3$.

If x if far from $l^{(1)}$:

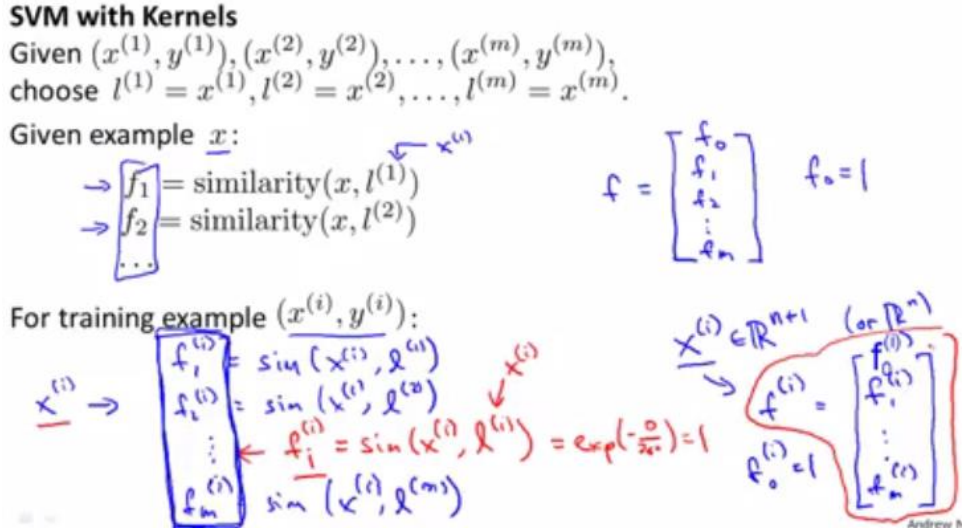
$$f_1 = \exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \approx 0.$$

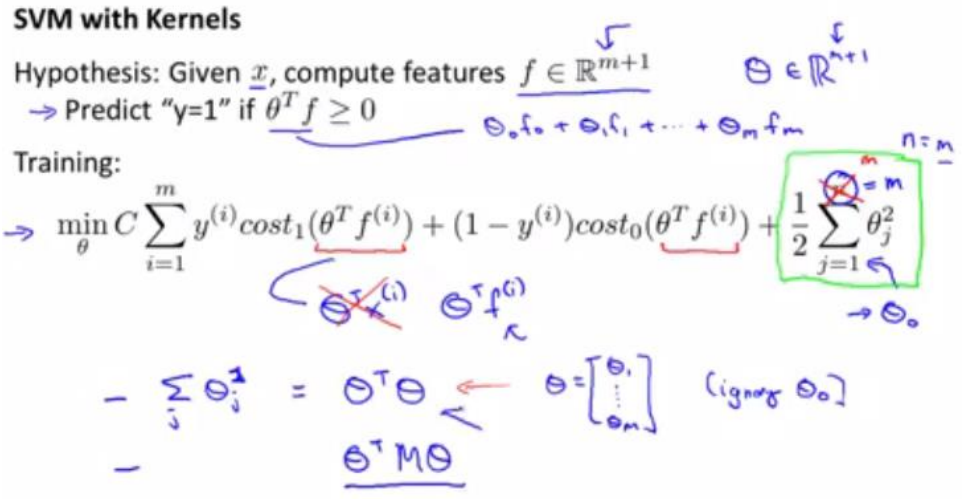
Example:



c. Where do we get the landmarks from:

i. Chose the landmarks from the training examples.





Kernels can be applied to Logistic regression also.
But the SVMs are Numerically simple and this simplicity cannot be achieved in Logistic Regression.

The M above is used for Numerical optimization.

d. SVM Parameters:
Larger sigma => Smoother curves => Less Variance
Smaller sigma => Sharp curves => High Variance

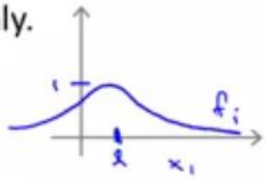
C ($= \frac{1}{\lambda}$). → Large C: Lower bias, high variance. (small λ)

→ Small C: Higher bias, low variance. (large λ)

σ^2 Large σ^2: Features f_i vary more smoothly.

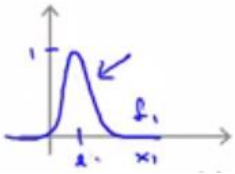
→ Higher bias, lower variance.

$\exp\left(-\frac{\|x-l^{(i)}\|^2}{2\sigma^2}\right)$



Small σ^2: Features f_i vary less smoothly.

Lower bias, higher variance.



2. Using and SVM

   a. Few things:
      i. Use SVM Software packages e.g. liblinear, libsvm
      ii. Need to specify
         1. Parameter C
         2. Kernel function
         3. Sigma square value

   b. How to choose a kernel
      i. No Kernel:

No kernel ("linear kernel")

Predict "y = 1" if $\theta^T x \geq 0$

$\theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n \geq 0$

→ n large, m small

$x \in \mathbb{R}^{n+1}$

      ii. Gaussian Kernel:

Gaussian kernel:

$f_i = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$, where $l^{(i)} = x^{(i)}$.

Need to choose σ^2.

$x \in \mathbb{R}^n$, n small

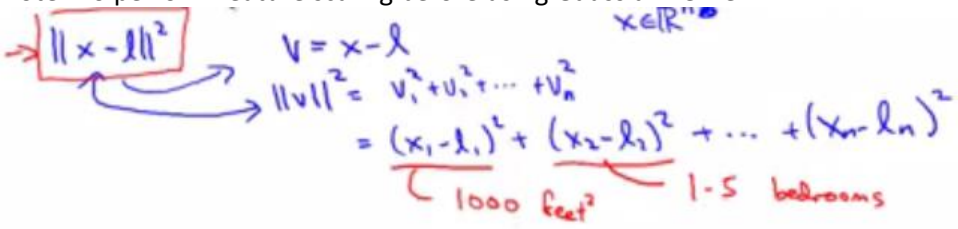
and/or m large

   c. Kernel Similarity functions

Kernel (similarity) functions:

```
function f = kernel(x1,x2)
    f = exp( -|| x1 - x2 ||^2 / (2σ^2) )
return
```

Note: Do perform feature scaling before using Gaussian kernel.



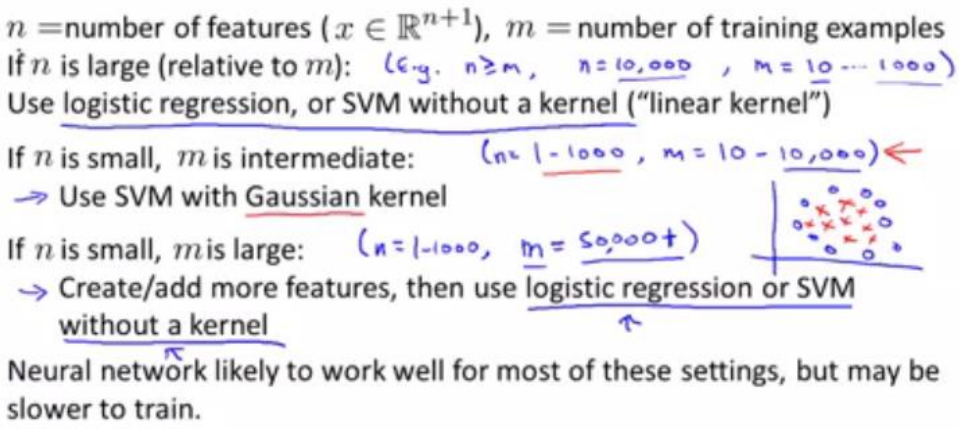
One features magnitude will dominate other ones with smaller magnitude.

Note: Not all similarity functions $\text{similarity}(x, l)$ make valid kernels. (Need to satisfy technical condition called "Mercer's Theorem" to make sure SVM packages' optimizations run correctly, and do not diverge).

d. Few Off Shelf kernels:
   i. Polynomial kernel
   ii. String kernel
   iii. Chi-square kernel
   iv. Histogram intersection kernel

e. Multi-class classification
   i. Use one vs all method
   ii. Train K SVMs for K Class classification

f. Logistic Regression vs SVMs:



Optimization problem that SVM has is a Convex optimization problem and so the good SVM from software packages will find a global minimum.

We need not worry about local optimum in SVM.

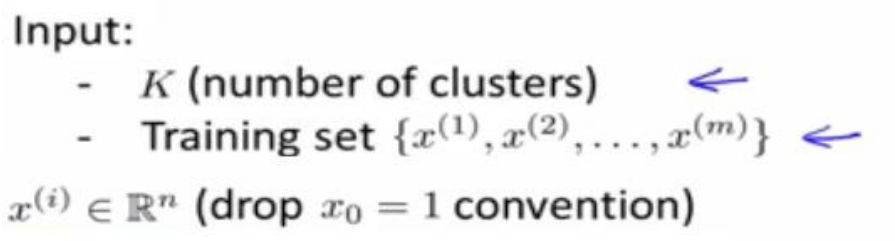
Week 8

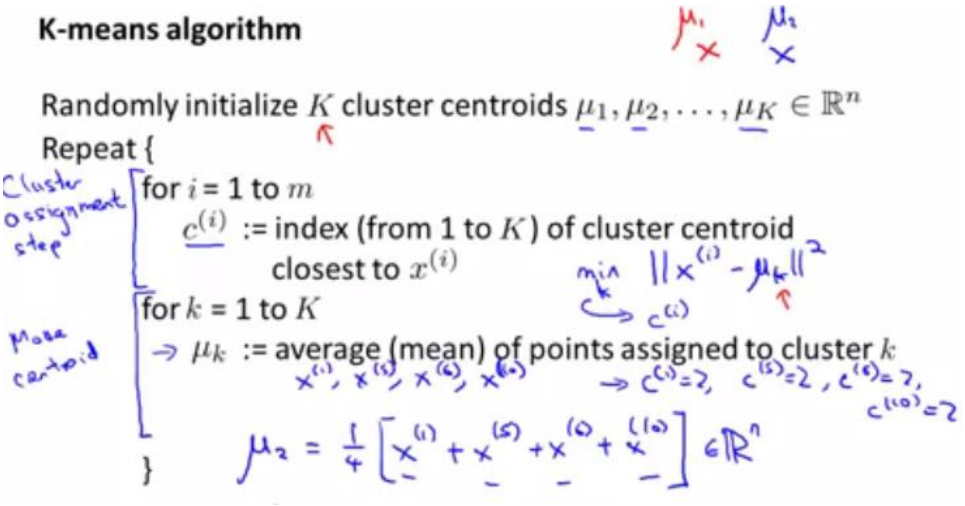
Wednesday, July 12, 2017
1:24 AM

Unsupervised Learning:

1. Few points
    a. No Labels in the training set.
    b. Find the structure in the training set.
    c. Clustering Algorithm.
    d. Applications of clustering
        i. Market segmentation
        ii. Social Network Analysis
        iii. Organize computing clusters
        iv. Astronomical data analysis

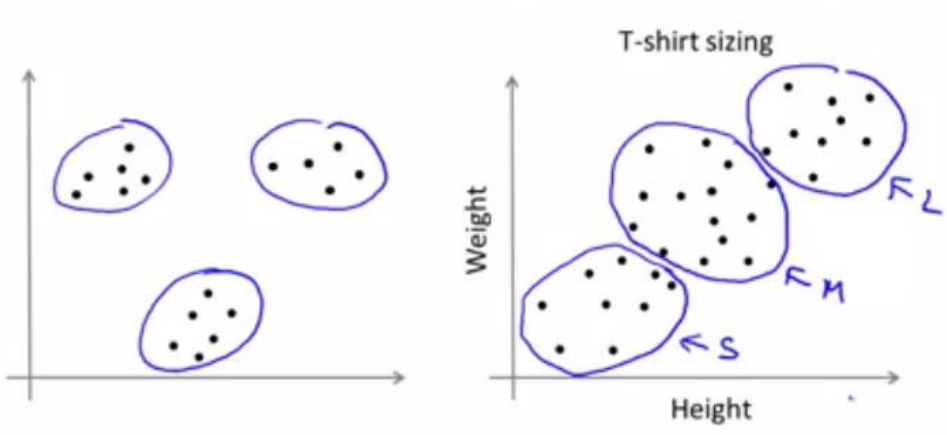
2. K-Means Clustering Algorithm:





a. If you get a cluster centroid with no values assigned to it
   i. You can eliminate that class and use K -1 classification
   ii. Else you can randomly initialize the

b. K Means for not well separated Clusters



3. Optimization Objective:

K-means optimization objective

→ $c^{(i)}$ = index of cluster (1,2,…,K) to which example $x^{(i)}$ is currently assigned

→ μ_k = cluster centroid k ($\mu_k \in \mathbb{R}^n$)

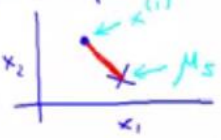
$\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned

Optimization objective:

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2$$

$$\min_{c^{(1)},\dots,c^{(m)}, \mu_1,\dots,\mu_K} J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$$

Distortion



K-means algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$

Repeat {

Cluster assignment step: minimize $J(\dots)$ wrt $c^{(1)}, c^{(2)}, \dots, c^{(m)}$ (holding $\mu_1, \dots, \mu_K$ fixed)

for i = 1 to m

$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$

move centroid

for k = 1 to K

μ_k := average (mean) of points assigned to cluster k

}

minimize $J(\dots)$ wrt $\mu_1, \dots, \mu_K$

4. Random Initialization:

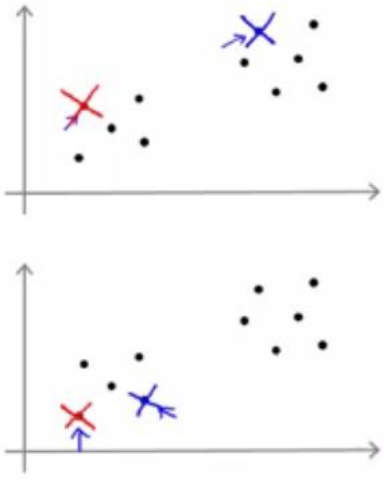
Random initialization

Should have $K < m$

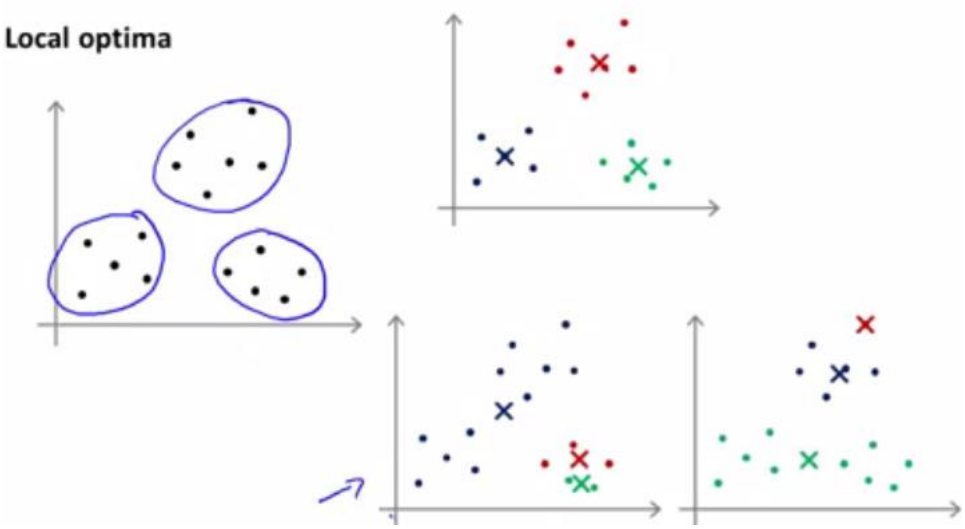
$K = 2$

Randomly pick K training examples.

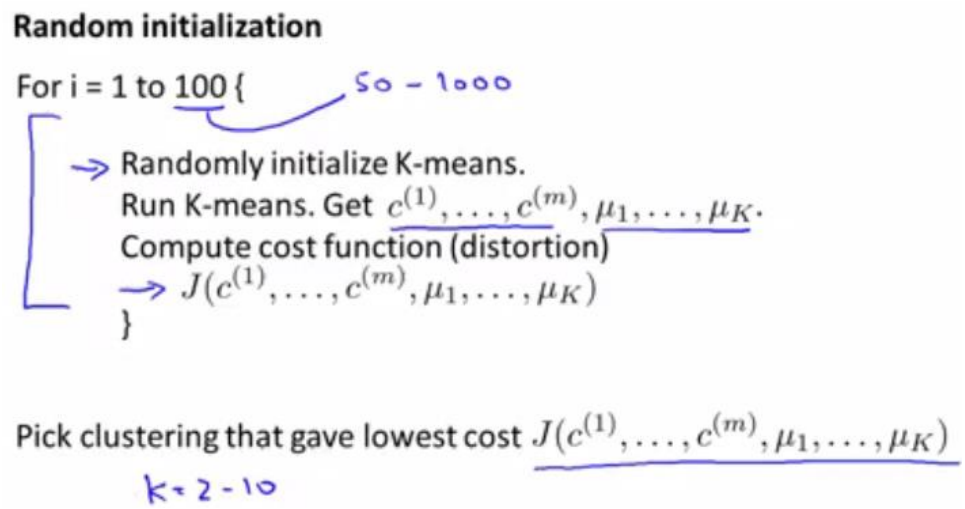
Set $\mu_1, \dots, \mu_K$ equal to these K examples. $\mu_1 = x^{(i)}$, $\mu_2 = x^{(j)}$



Local Optima problem: This can happen when we are get unlucky random initialization.
This is the local optima of the J Function i.e. distortion function.



When we K to be small then there are more chances that the random initialization will make a difference else when our K is very large it is highly probable that the first randomly selected values will give us a good optimization.

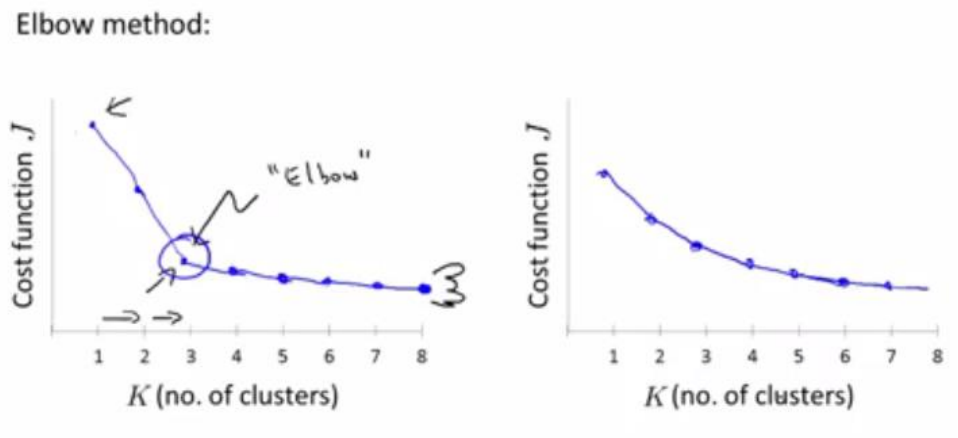


5. Choosing the number of Clusters:

    a. What is the value of K ?
        i. We can group in many ways

    b. Elbow method:
        i. Run K - means with multiple values and then calculate the cost function
        ii. Plot the Cost function vs K graph

iii. This finds an elbow in the Graph i.e. the point where the distortion goes down rapidly and goes down slowly thereafter select that value of K
iv. It's harder to location elbow in the graph on right.

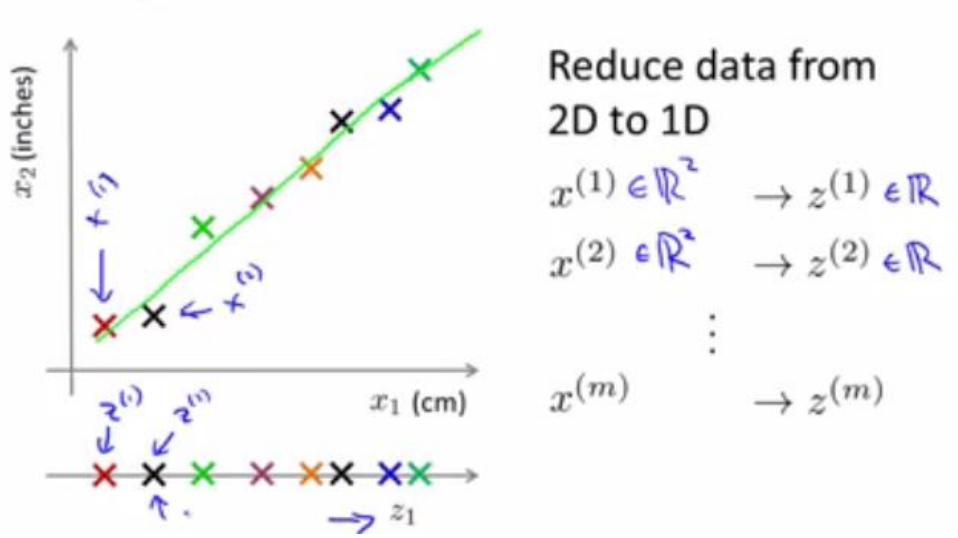


c. Choosing value of K:
Evaluate K-means based on a metric for how well it performs for later purpose.

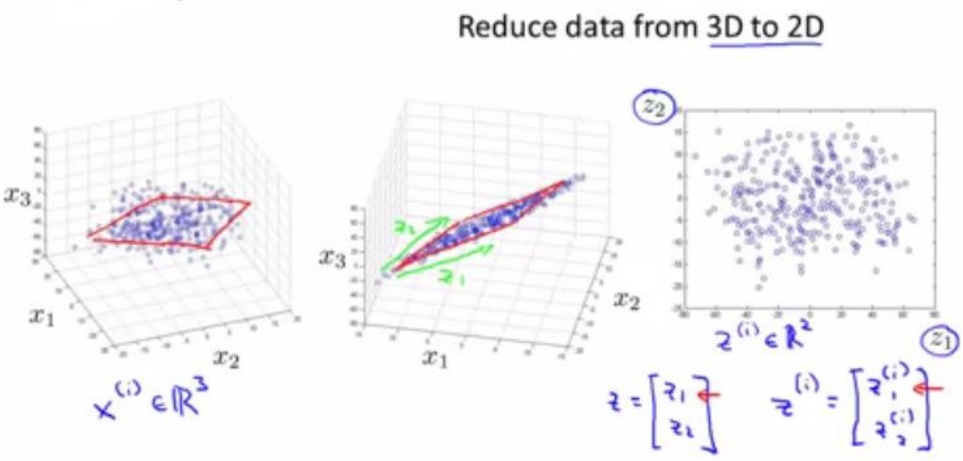
d. On every iteration of K-means, the cost function $J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_k)$ (the distortion function) should either stay the same or decrease; in particular, it should not increase.

Principal Component Analysis:

1. Dimensionality Reduction:
   a. Data Compression
      i. We have redundant features.
      ii. Features need not be exactly linearly dependent.
      iii. Memory Requirement can be reduced.
      iv. Project data from 2D Plane to 1D Line



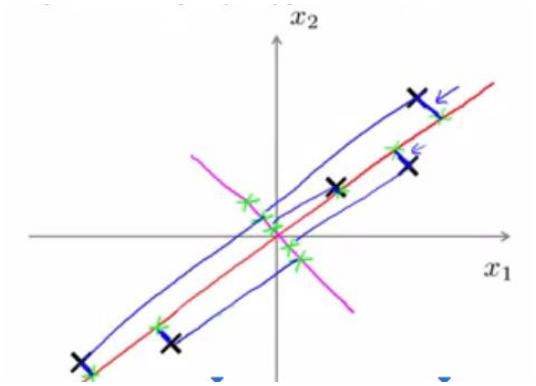
i. Project data from 3D Space to 2D Plane.



b. Data Visualization:
   i. Often we convert data in high dimension space to 2D or 3D so that we can plot the data to visualize them

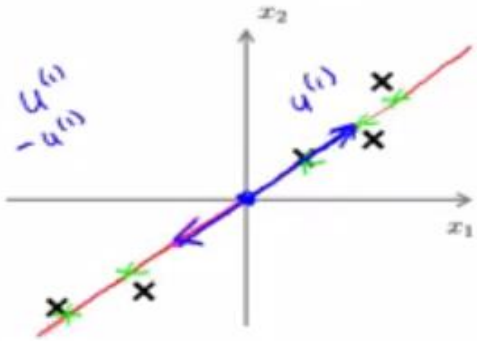
2. Principal Component Analysis Problem Formulation:

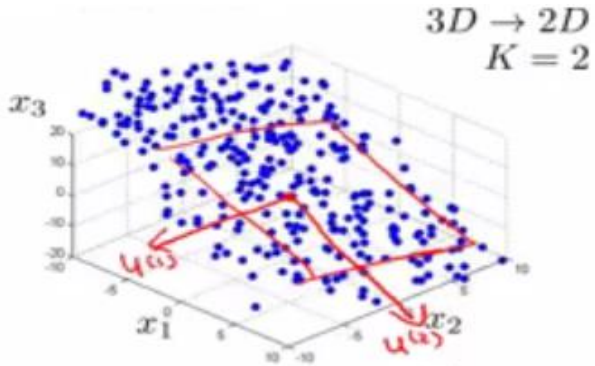
   a. Intro
      i. Minimize Projection Error



ii. Mean normalization and Feature Scaling is needed

b. PCA Problem Formulation:

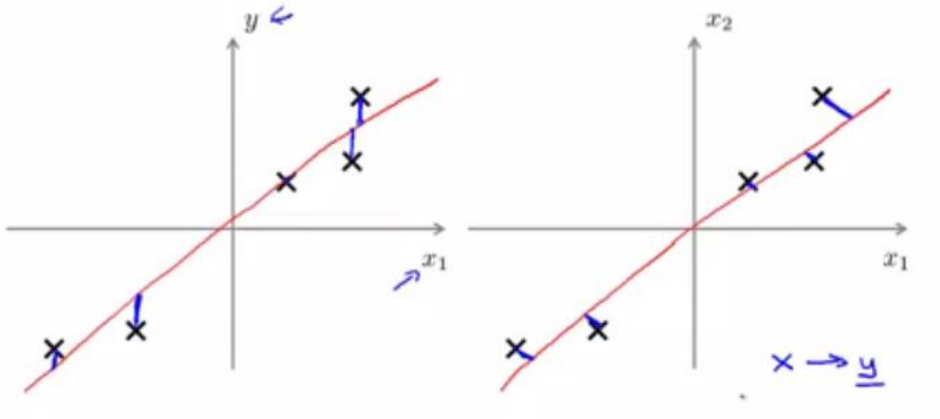




Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error.

Reduce from n-dimension to k-dimension: Find k vectors $u^{(1)}, u^{(2)}, \ldots, u^{(k)}$ onto which to project the data, so as to minimize the projection error.

c. PCA is not linear Regression:



d. PCA Algorithm:

i. Data Preprocessing:

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

Preprocessing (feature scaling/mean normalization):

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$$

Replace each $x_j^{(i)}$ with $x_j - \mu_j$.

If different features on different scales (e.g., x_1 = size of house, x_2 = number of bedrooms), scale features to have comparable range of values.

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

ii. PCA

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

$$\Sigma = \frac{1}{m}\sum_{i=1}^{n} (x^{(i)})(x^{(i)})^T$$

$(n \times 1)(1 \times n)$ → $n \times n$ matrix. Sigma

Compute "eigenvectors" of matrix Σ:

```
[U,S,V] = svd(Sigma);
```

→ Singular value decomposition

eig (Sigma)

$n \times n$ matrix.

$$U = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & & | \end{bmatrix} \quad U \in \mathbb{R}^{n \times n}$$

$u^{(1)}, \ldots, u^{(k)}$

k

From `[U,S,V] = svd(Sigma)`, we get:

$$U = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(n)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times n}$$

k

$x \in \mathbb{R}^n \rightarrow z \in \mathbb{R}^k$

$$z = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & & | \end{bmatrix}^T x = \begin{bmatrix} — (u^{(1)})^T — \\ \vdots \\ — (u^{(k)})^T — \end{bmatrix} x$$

$z \in \mathbb{R}^k$

$n \times k$

U_{reduce}

$k \times n$ $\quad n \times 1$

$k \times 1$

iii. Summary

Principal Component Analysis (PCA) algorithm summary

→ After mean normalization (ensure every feature has zero mean) and optionally feature scaling:

Sigma = $\frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T$

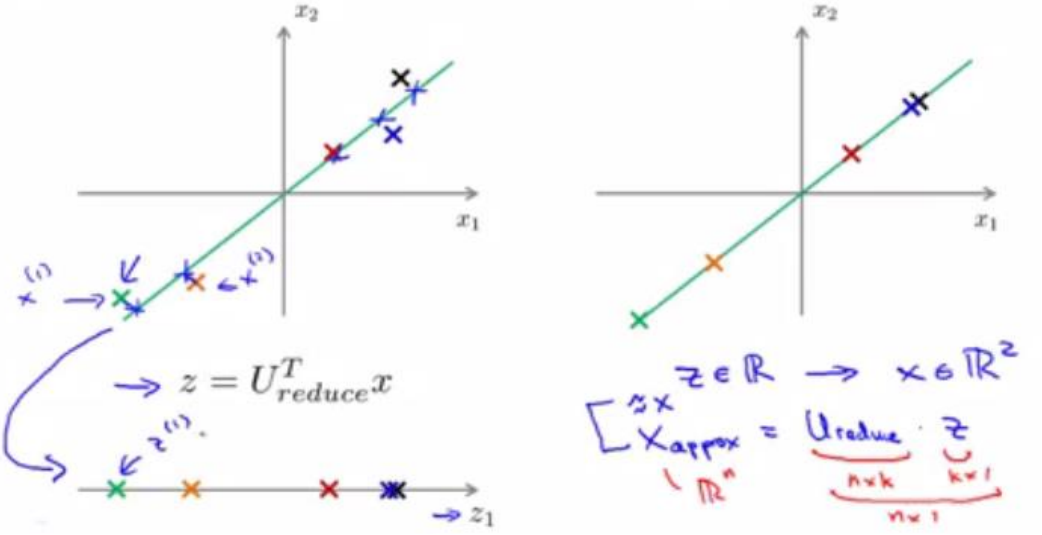
$X = \begin{bmatrix} — (x^{(1)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$

Sigma = (1/m) * X' * X;

```
[U,S,V] = svd(Sigma);
Ureduce = U(:,1:k);
z = Ureduce'*x;
```

$x \in \mathbb{R}^n$ $x_0 = 1$

e. Reconstruction from compressed Representation:



f. Choosing the number of Principal components:

Choosing k (number of principal components)

Average squared projection error: $\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x_{approx}^{(i)}\|^2$

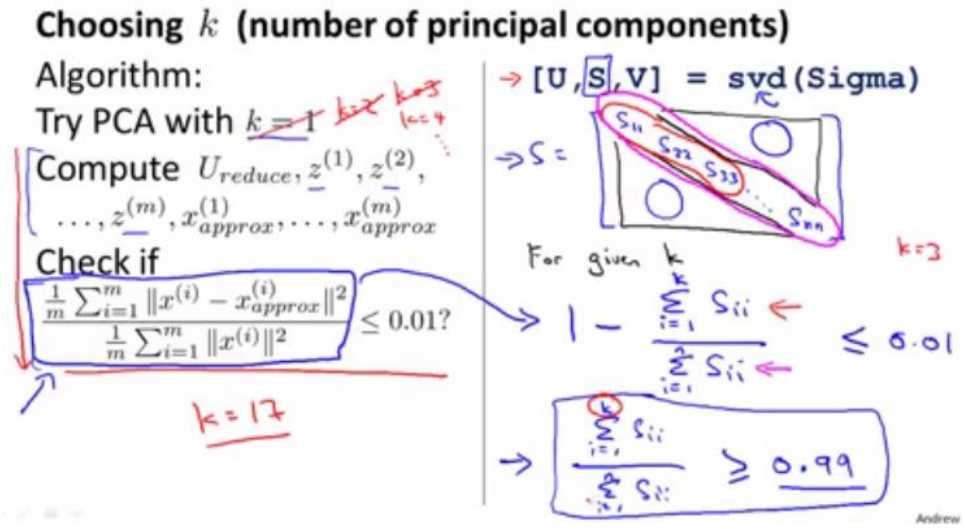
Total variation in the data: $\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2$

Typically, choose k to be smallest value so that

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \leq 0.01 \quad (1\%)$$

› "99% of variance is retained"

How to choose the value of K which gives the above variance.
Use the S Matrix that was computed with svd function



Summary:

[U,S,V] = svd(Sigma)

Pick smallest value of k for which

$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{m} S_{ii}} \geq 0.99$$

(99% of variance retained)

g. Advice for applying PCA:

i. Supervised Learning Speedup
Lower memory and computation power.

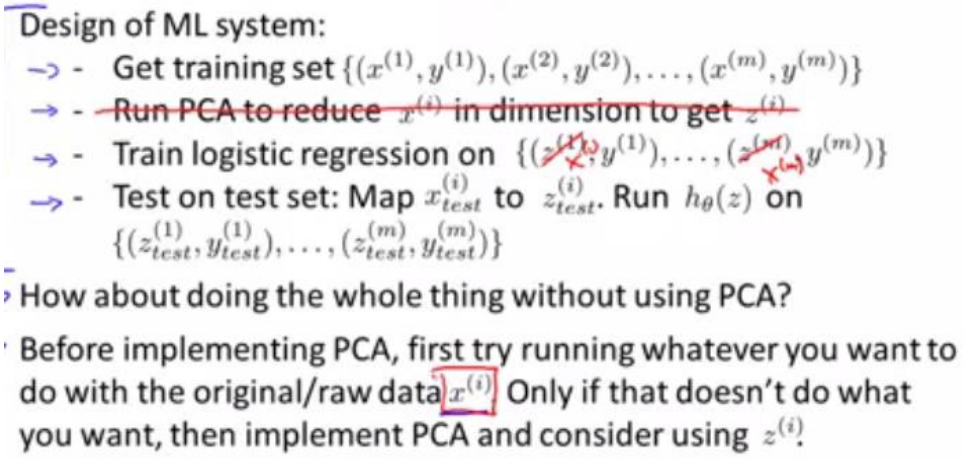
$$x^{(1)}, x^{(2)}, \ldots, x^{(m)} \in \mathbb{R}^{10000}$$
$$\downarrow PCA$$
$$z^{(1)}, z^{(2)}, \ldots, z^{(m)} \in \mathbb{R}^{1000}$$

Note: Mapping $x^{(i)} \rightarrow z^{(i)}$ should be defined by running PCA only on the training set. This mapping can be applied as well to the examples $x^{(i)}_{cv}$ and $x^{(i)}_{test}$ in the cross validation and test sets.

ii. BAD Use of PCA: To Prevent Overfitting
1. Use regularization instead of PCA when overfitting.

2. PCA does not use labels hence throws away some data when you are compressing
   Without seeing the labels

iii. PCA Should not be used to Design ML System



Week 9

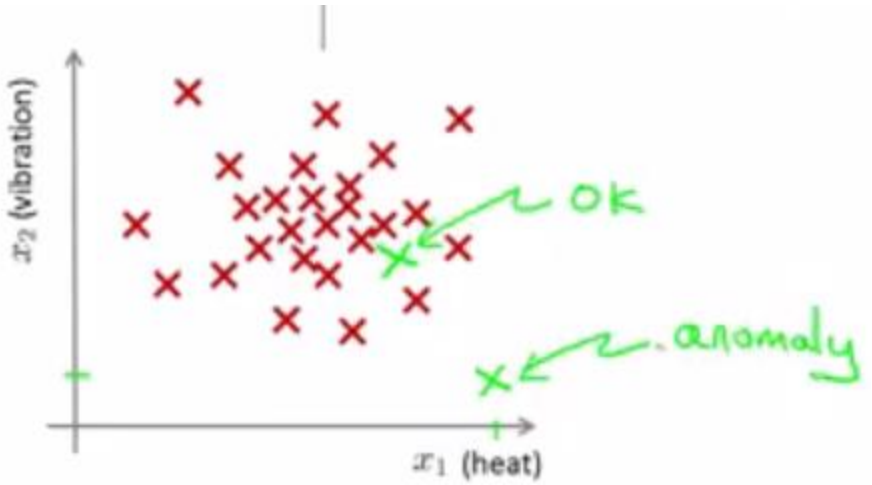
Monday, July 24, 2017
12:19 AM

Anomaly Detection:

Problem Motivation:

As a part of QA Testing of Aircraft engine we use features like
x1 => heat detected
x2 => vibration intensity

The problem here is that if the new engine with parameters should be sent for further testing does it look different than normal engines.

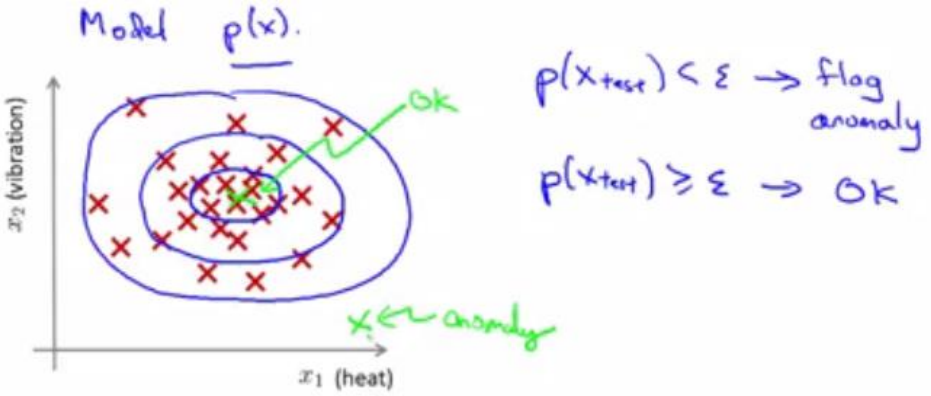


Given:

Dataset: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

We need determine

Is x_{test} anomalous?



Example:

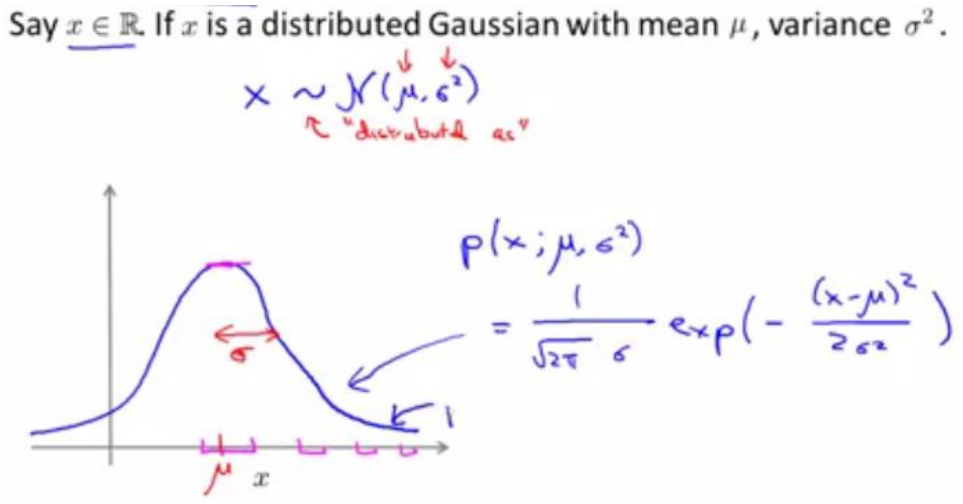
i. Fraud Detection
ii. Manufacturing QA
iii. Monitoring Computers in Data Center

Gaussian Distribution: (Or Normal Distribution)

sigma	Standard deviation
u	Mean

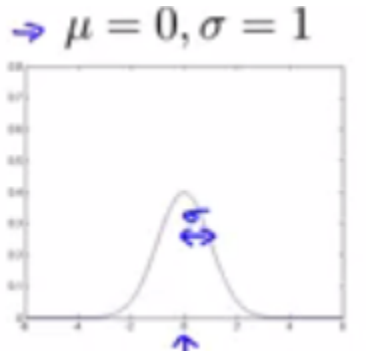
sig square	Variance

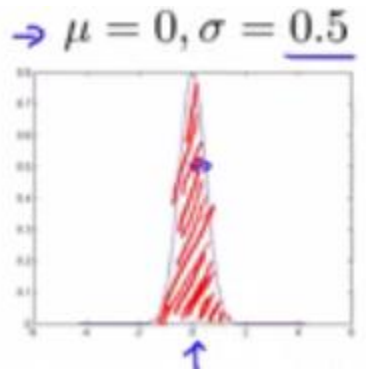
Probability of x taking the values away from mean reduces.

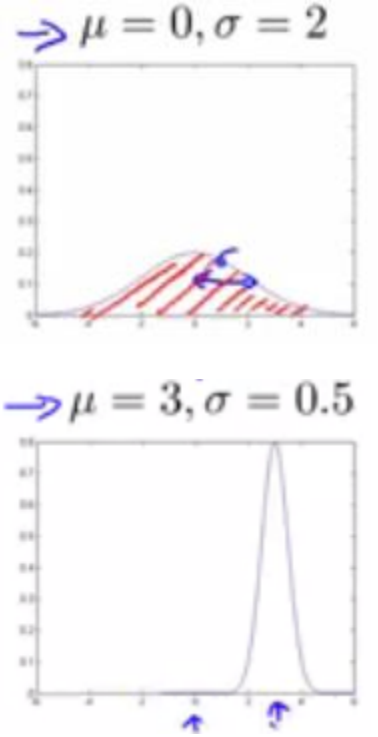


Example:

Area of curve must integrate to 1.
This is the property of the PDF.







Parameter Estimation:

If we have to predict the u and sigma square from the dataset given. We have to estimate the gaussian distribution that the data came from.

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)^2$$

These estimated are the maximum likelihood estimates of these parameters.

In Variance sometimes people use 1/(M-1). in ML people use M and M being large this does not matter much practically

Algorithm for Anomaly detection:

Density estimation

Training set: $\{x^{(1)}, \ldots, x^{(m)}\}$

Each example is $x \in \mathbb{R}^n$

We assume that features are distributed with gaussian distributions with some mean and variance.

$x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$

$x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$

$x_3 \sim \mathcal{N}(\mu_3, \sigma_3^2)$

$$p(x) = p(x_1; \mu_1, \sigma_1^2)\, p(x_2; \mu_2, \sigma_2^2)\, p(x_3; \mu_3, \sigma_3^2) \cdots p(x_n; \mu_n, \sigma_n^2) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$$

$$\sum_{i=1}^{n} i = 1+2+3+\cdots+n \qquad \prod_{i=1}^{n} i = 1 \times 2 \times 3 \times \cdots \times n$$

Putting it all together:

Anomaly detection algorithm

1. Choose features x_i that you think might be indicative of anomalous examples. $\{x^{(1)}, \ldots, x^{(m)}\}$
2. Fit parameters $\mu_1, \ldots, \mu_n, \sigma_1^2, \ldots, \sigma_n^2$

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

$$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^{m} (x_j^{(i)} - \mu_j)^2$$

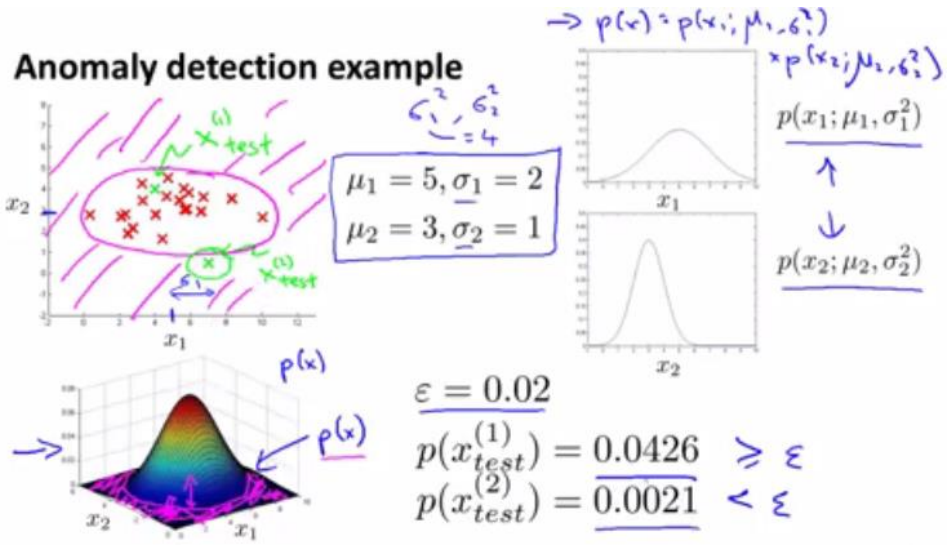
$p(x_j; \mu_j, \sigma_j^2)$ $\quad \mu_1, \mu_2, \ldots, \mu_n$ $\quad \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix} = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$

3. Given new example x, compute $p(x)$:

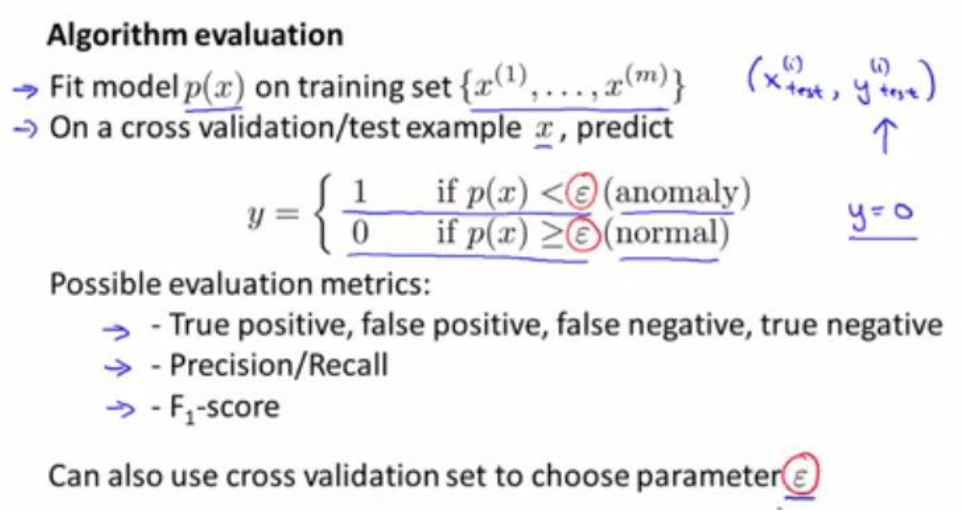
$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

Anomaly if $p(x) < \varepsilon$

Example:



Developing and Evaluating the algorithm:



Anomaly Detection vs Supervised Learning:

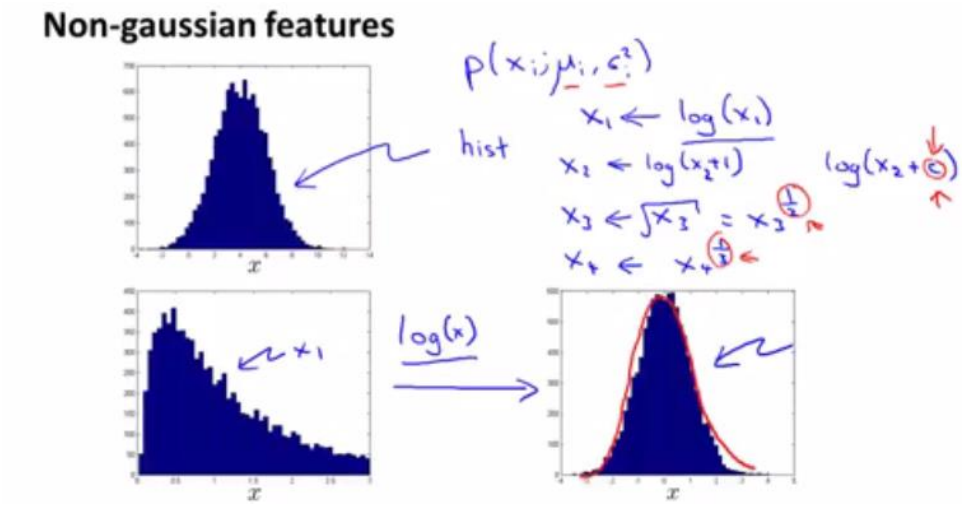
Anomaly detection	vs.	Supervised learning
Very small number of positive examples ($y = 1$). (0-20 is common).		Large number of positive and negative examples.
Large number of negative ($y = 0$) examples.		
Many different "types" of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like; future anomalies may look nothing like any of the anomalous examples we've seen so far.		Enough positive examples for algorithm to get a sense of what positive examples are like, future positive examples likely to be similar to ones in training set.

Anomaly detection	vs.	Supervised learning
• Fraud detection		• Email spam classification
• Manufacturing (e.g. aircraft engines)		• Weather prediction (sunny/rainy/etc).
• Monitoring machines in a data center		• Cancer classification
⋮		⋮

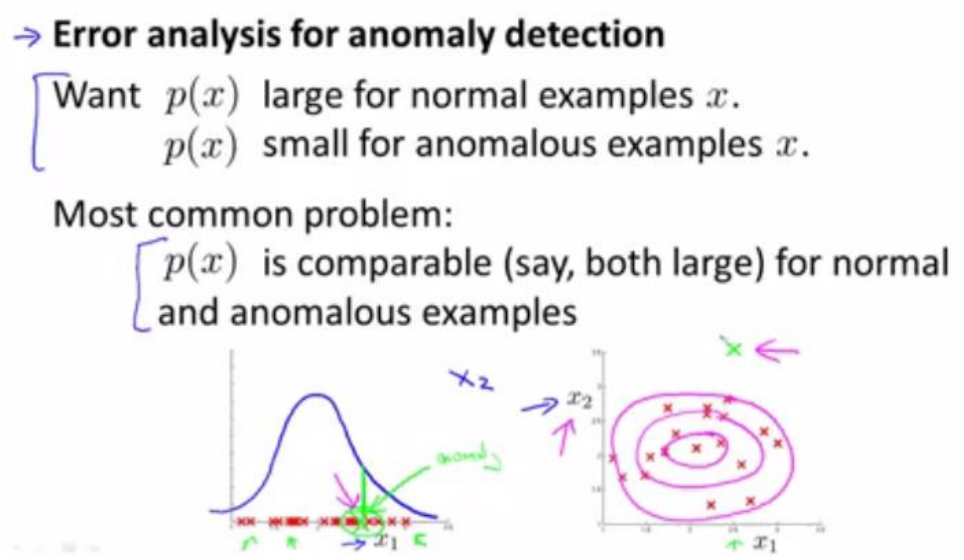
Choosing what features to Use:

Even in case the data looks not gaussian the algorithms work fine.
Its better though to convert the data to look like gaussian distribution.

You can use some transformations to make the data into gaussian.



When you are getting an anomaly in the p(x) > ep. There you might try to more features that will get the anomaly.



We will have to select the features that might help us distinguish the anomalies from the rest.

Monitoring computers in a data center

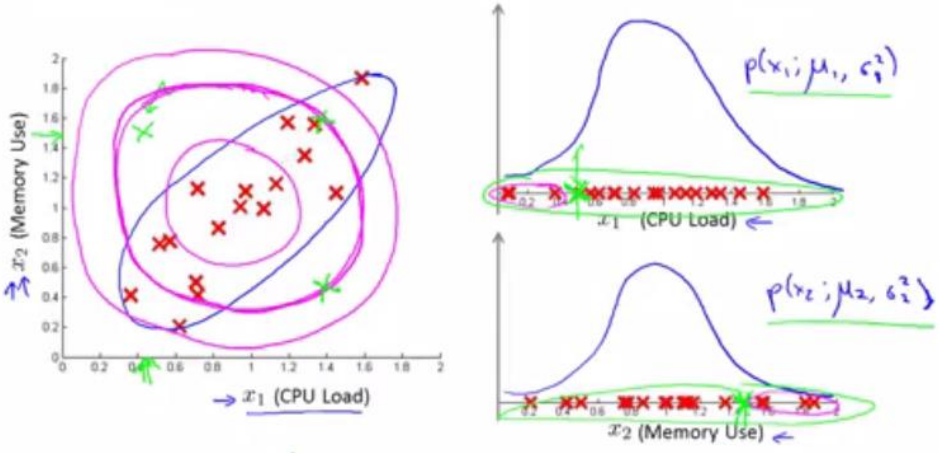
Choose features that might take on unusually large or small values in the event of an anomaly.

- x_1 = memory use of computer
- x_2 = number of disk accesses/sec
- x_3 = CPU load
- x_4 = network traffic

$x_5 = \frac{\text{CPU load}}{\text{network traffic}}$ $\quad x_6 = \frac{(\text{CPU load})^2}{\text{network traffic}}$

Multivariate Gaussian Distribution:

Motivating example: Monitoring machines in a data center



Multivariate Gaussian (Normal) distribution

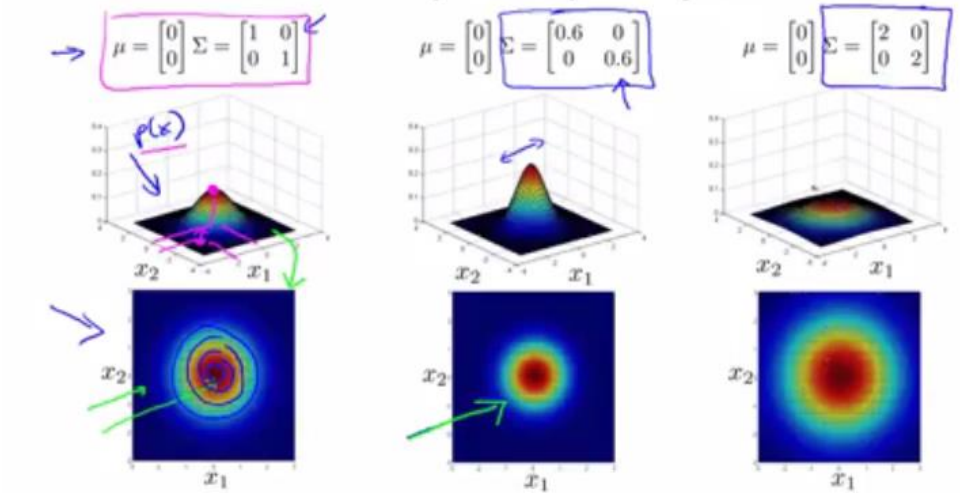
$x \in \mathbb{R}^n$. Don't model $p(x_1), p(x_2), \ldots$, etc. separately. Model $p(x)$ all in one go.

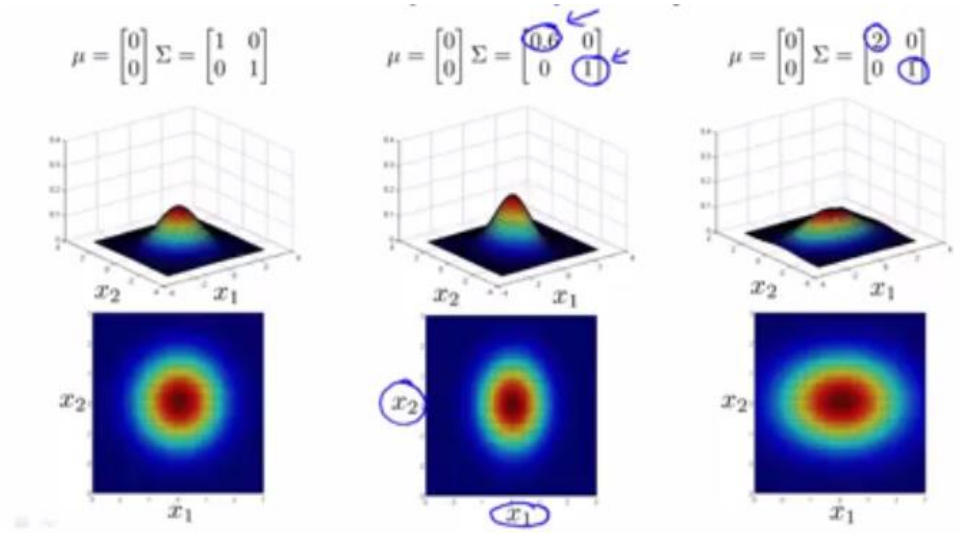
Parameters: $\mu \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$ (covariance matrix)

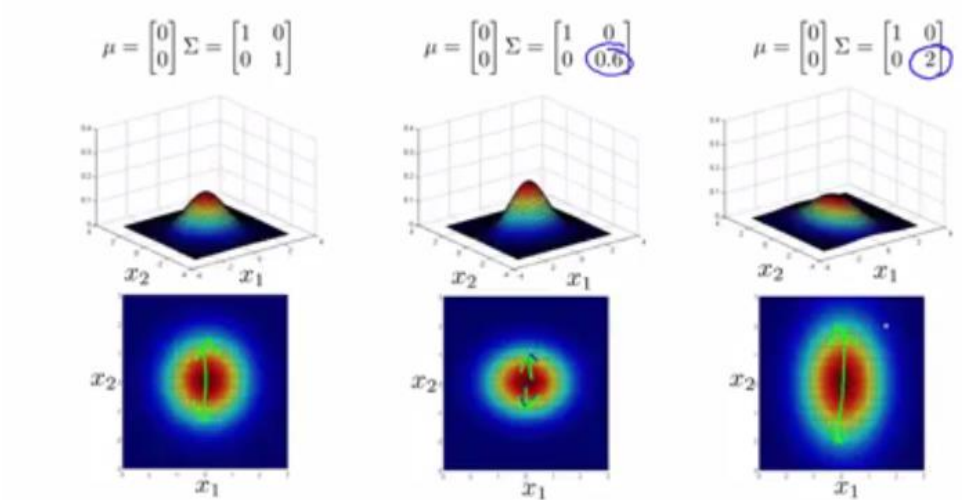
$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

$|\Sigma|$ = determinant of Σ | det (Sigma)

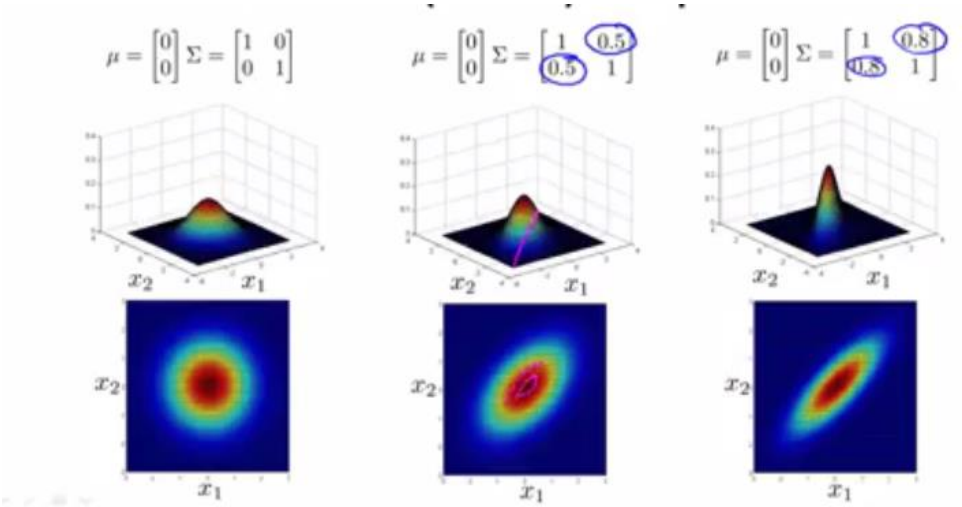
Multivariate Gaussian (Normal) examples

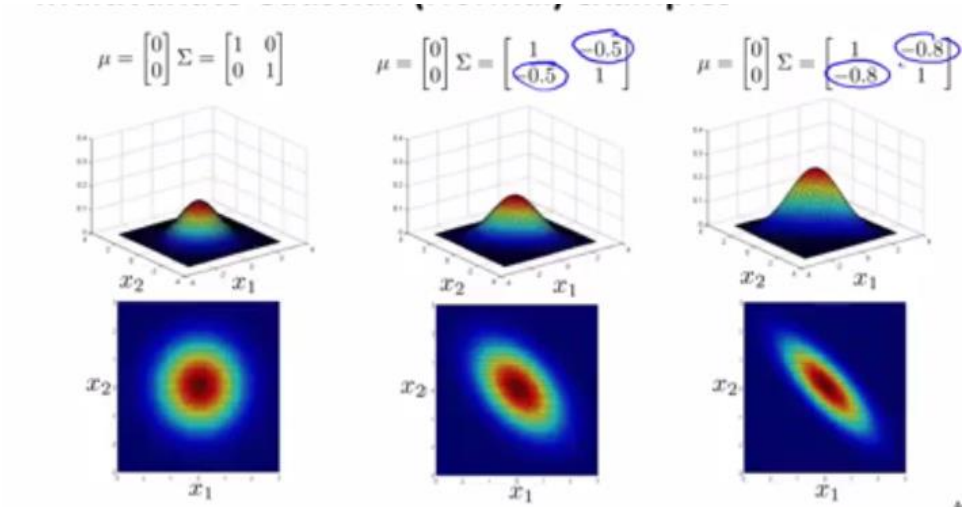




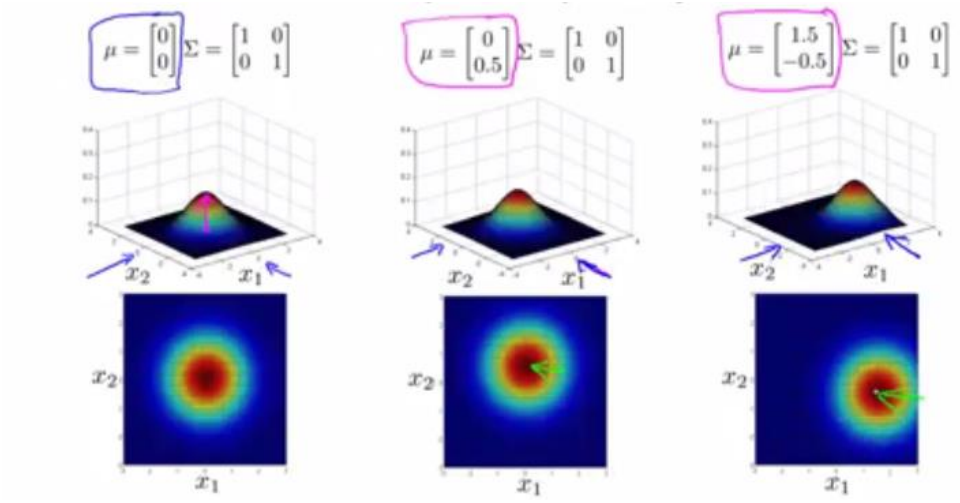


Off diagonal elements allow us to model the correlation





Varying the u we can shift the peak to other point in the graph

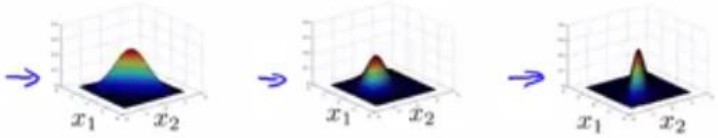


Anomaly Detection with Multi variate gaussian distribution:

Multivariate Gaussian (Normal) distribution

Parameters μ, Σ $\quad \mu \in \mathbb{R}^n \quad \Sigma \in \mathbb{R}^{n \times n}$

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$



Parameter fitting:

Given training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ $\quad x \in \mathbb{R}^n$

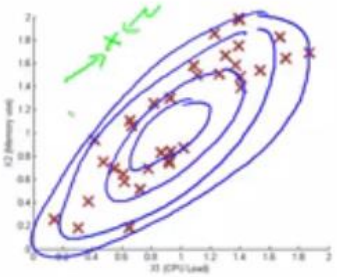
$$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} \qquad \Sigma = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

Anomaly detection with the multivariate Gaussian

1. Fit model $p(x)$ by setting

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)}-\mu)(x^{(i)}-\mu)^T$$



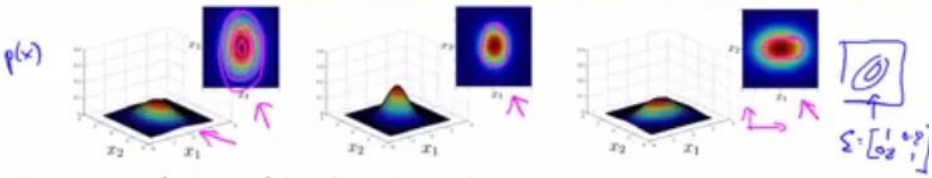
2. Given a new example x, compute

$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Flag an anomaly if $p(x) < \varepsilon$

Relationship to original model

Original model: $p(x) = p(x_1;\mu_1,\sigma_1^2) \times p(x_2;\mu_2,\sigma_2^2) \times \cdots \times p(x_n;\mu_n,\sigma_n^2)$



Corresponds to multivariate Gaussian

$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

where $\Sigma = \begin{bmatrix}\sigma_1^2 & & & 0\\ & \sigma_2^2 & & \\ & & \ddots & \\ 0 & & & \sigma_n^2\end{bmatrix}$

Original model	vs.	Multivariate Gaussian
$p(x_1;\mu_1,\sigma_1^2)\times\cdots\times p(x_n;\mu_n,\sigma_n^2)$		$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}\lvert\Sigma\rvert^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$
Manually create features to capture anomalies where x_1, x_2 take unusual combinations of values. $x_3 = \frac{x_1}{x_2} = \frac{\text{CPU load}}{\text{memory}}$		Automatically captures correlations between features
Computationally cheaper (alternatively, scales better to large n) $n=10{,}000$, $n=100{,}000$		$\Sigma \in \mathbb{R}^{n\times n}$ Σ^{-1} Computationally more expensive
OK even if m (training set size) is small		Must have $m > n$ or else Σ is non-invertible. $m \geq 10n$

People generally capture extra features and use the original model instead of Multivariate Gaussian Model.

If you find the sigma matrix to be singular or non-invertible

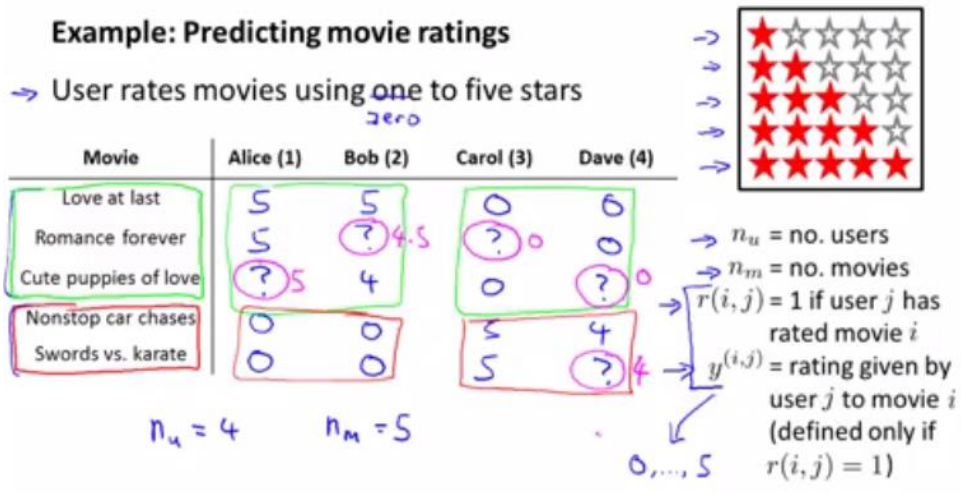
a. Check m >= n or say m >= 10 n
b. You might have redundant features
   i. x1 = x2
   ii. x3 = x1 + x2

   Features are Linearly independent.

Recommender Systems:

Problem Formulation;

Example: Predicting movie rating:
Given the Matrices Below r and y we have to find the missing data i.e. here the data of the movies not watched.



Here for this analysis we use user rating from 0 to all starts. Generally ppl user from 1 - all

Content Based Recommendations:

Content-based recommender systems

$n_u = 4$, $n_m = 5$, $x_0 = 1$, $x^{(1)} = \begin{bmatrix} 1 \\ 0.9 \\ 0 \end{bmatrix}$

Movie	Alice (1) $\theta^{(1)}$	Bob (2) $\theta^{(2)}$	Carol (3) $\theta^{(3)}$	Dave (4) $\theta^{(4)}$	x_1 (romance)	x_2 (action)
Love at last 1	5	5	0	0	0.9	0
Romance forever 2	5	?	?	0	1.0	0.01
Cute puppies of love 3	? 4.95	4	0	?	0.99	0
Nonstop car chases 4	0	0	5	4	0.1	1.0
Swords vs. karate 5	0	0	5	?	0	0.9

$n = 2$

- For each user j, learn a parameter $\theta^{(j)} \in \mathbb{R}^3$. Predict user j as rating movie i with $(\theta^{(j)})^T x^{(i)}$ stars. ($\theta^{(j)} \in \mathbb{R}^{n+1}$)

$x^{(3)} = \begin{bmatrix} 1 \\ 0.99 \\ 0 \end{bmatrix}$ $\leftrightarrow$ $\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$ $(\theta^{(1)})^T x^{(3)} = 5 \times 0.99 = 4.95$

Problem formulation

- $r(i,j) = 1$ if user j has rated movie i (0 otherwise)
- $y^{(i,j)}$ = rating by user j on movie i (if defined)
- $\theta^{(j)}$ = parameter vector for user j
- $x^{(i)}$ = feature vector for movie i
- For user j, movie i, predicted rating: $(\theta^{(j)})^T (x^{(i)})$ $\quad \theta^{(j)} \in \mathbb{R}^{n+1}$
- $m^{(j)}$ = no. of movies rated by user j

To learn $\theta^{(j)}$:

$$\min_{\theta^{(j)}} \frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T (x^{(i)}) - y^{(i,j)} \right)^2 + \frac{\lambda}{2m^{(j)}} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

We can ignore mj as optimization does not change.

Optimization objective:

To learn $\theta^{(j)}$ (parameter for user j):

$$\min_{\theta^{(j)}} \frac{1}{2} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

To learn $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(n_u)}$:

$$\min_{\theta^{(1)}, \ldots, \theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

$\theta^{(1)}, \ldots, \theta^{(n_u)}$

Optimization algorithm:

$$\min_{\theta^{(1)},\ldots,\theta^{(n_u)}} \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2$$

$J(\theta^{(1)},\ldots,\theta^{(n_u)})$

Gradient descent update:

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})x_k^{(i)} \quad (\text{for } k = 0)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})x_k^{(i)} + \lambda\theta_k^{(j)}\right) \quad (\text{for } k \neq 0)$$

$\frac{\partial}{\partial \theta_k^{(j)}} J(\theta^{(1)},\ldots,\theta^{(n_u)})$

Collaborative Filtering:

Problem Motivation:

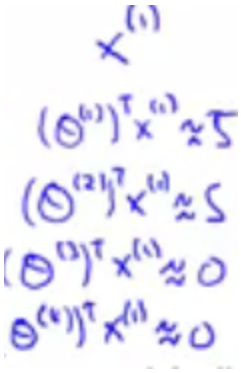
It is difficult to get the feature vector for movies.
Instead we get the taste of the user from him.
We now infer the values of the feature vectors of movies.

Movie	Alice (1) $\theta^{(1)}$	Bob (2) $\theta^{(2)}$	Carol (3) $\theta^{(3)}$	Dave (4) $\theta^{(4)}$	x_1 (romance)	x_2 (action)
Love at last	5	5	0	0	?	?
Romance forever	5	?	?	0	?	?
Cute puppies of love	?	4	0	?	?	?
Nonstop car chases	0	0	5	4	?	?
Swords vs. karate	0	0	5	?	?	?

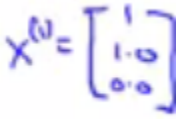
$$\theta^{(1)} = \begin{bmatrix}0\\5\\0\end{bmatrix}, \theta^{(2)} = \begin{bmatrix}0\\5\\0\end{bmatrix}, \theta^{(3)} = \begin{bmatrix}0\\0\\5\end{bmatrix}, \theta^{(4)} = \begin{bmatrix}0\\0\\5\end{bmatrix}$$

$\theta^{(j)}$

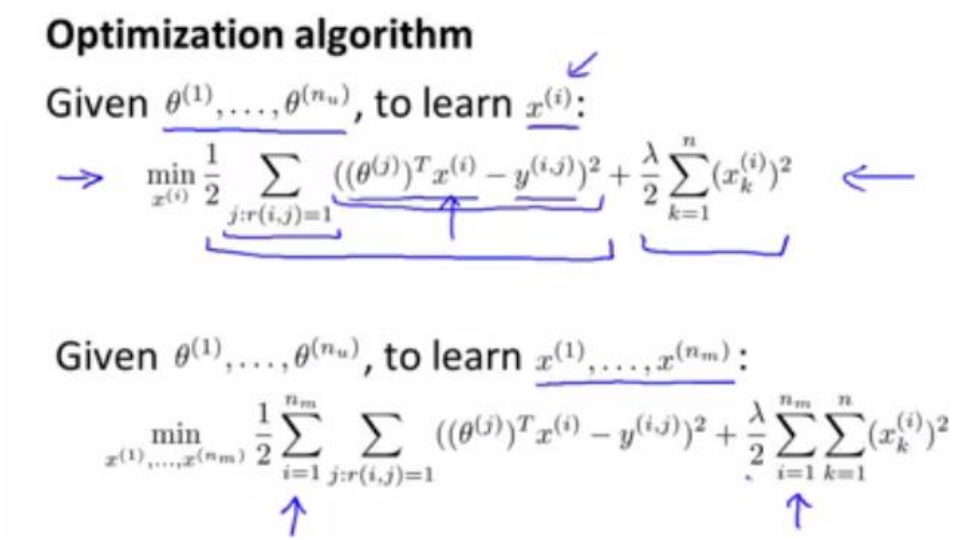
Now that we know the values of theta we predict the values of x using the rating that the user has given to the movie.

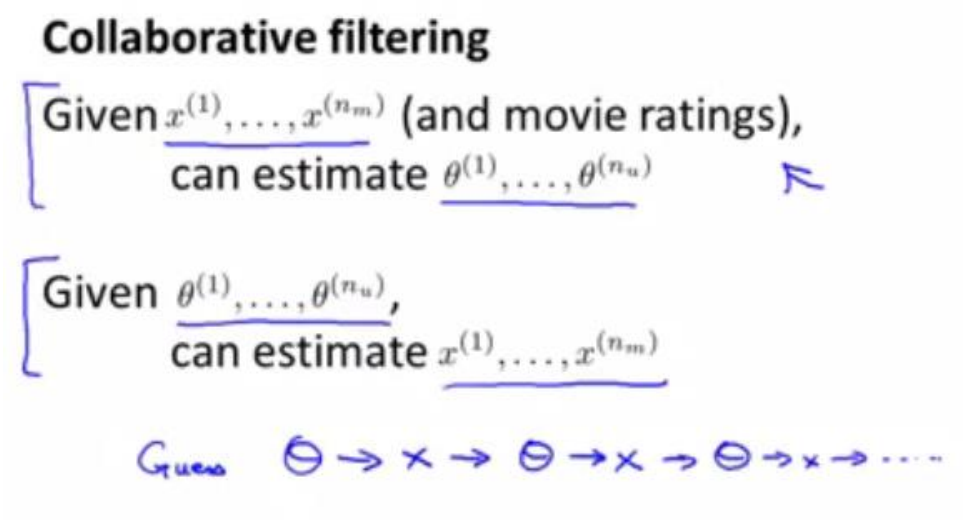


Implies

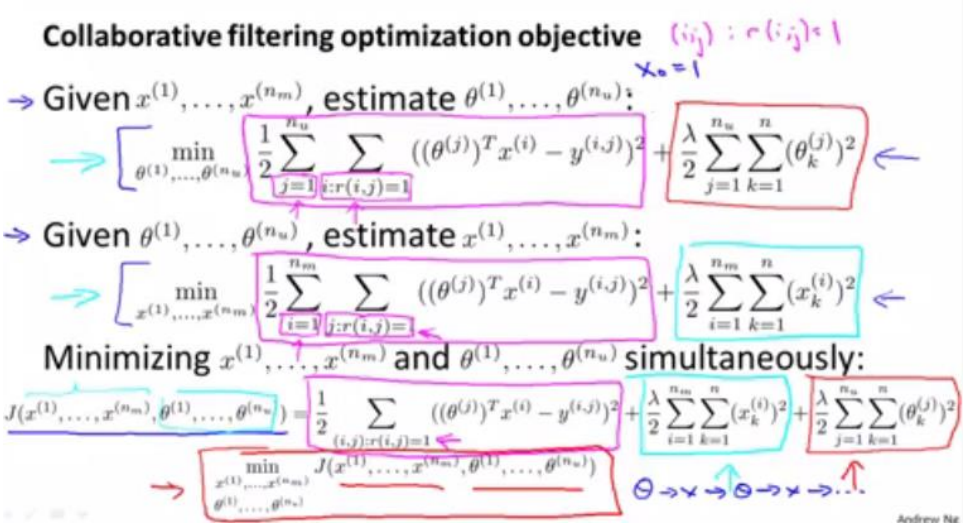


Optimization Algorithm:





Collaborative Filtering Optimization Objective:



Collaborative filtering algorithm

1. Initialize $x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)}$ to small random values.
2. Minimize $J(x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)})$ using gradient descent (or an advanced optimization algorithm). E.g. for every $j = 1, \ldots, n_u, i = 1, \ldots, n_m$:

$$x_k^{(i)} := x_k^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$$

3. For a user with parameters θ and a movie with (learned) features x, predict a star rating of $\theta^T x$.

Vectorization: Low Rank Matrix Factorization

Collaborative filtering

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at last	5	5	0	0
Romance forever	5	?	?	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	5	4
Swords vs. karate	0	0	5	?

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix}$$

Collaborative filtering

Predicted ratings:

$$\begin{bmatrix} (\theta^{(1)})^T (x^{(1)}) & (\theta^{(2)})^T (x^{(1)}) & \cdots & (\theta^{(n_u)})^T (x^{(1)}) \\ (\theta^{(1)})^T (x^{(2)}) & (\theta^{(2)})^T (x^{(2)}) & \cdots & (\theta^{(n_u)})^T (x^{(2)}) \\ \vdots & \vdots & \vdots & \vdots \\ (\theta^{(1)})^T (x^{(n_m)}) & (\theta^{(2)})^T (x^{(n_m)}) & \cdots & (\theta^{(n_u)})^T (x^{(n_m)}) \end{bmatrix}$$

$$X = \begin{bmatrix} -(x^{(1)})^T- \\ -(x^{(2)})^T- \\ \vdots \\ -(x^{(n_m)})^T- \end{bmatrix} \qquad \Theta = \begin{bmatrix} -(\theta^{(1)})^T- \\ -(\theta^{(2)})^T- \\ \vdots \\ -(\theta^{(n_u)})^T- \end{bmatrix}$$

Low rank matrix factorization

Finding related movies

For each product i, we learn a feature vector $x^{(i)} \in \mathbb{R}^n$.

→ x_1 = romance, x_2 = action, x_3 = comedy, x_4 = ...

How to find movies j related to movie i?

small $\|x^{(i)} - x^{(j)}\|$ → movie j and i are "similar"

5 most similar movies to movie i:

→ Find the 5 movies j with the smallest $\|x^{(i)} - x^{(j)}\|$.

Implementation Detail: Mean Normalization

Users who have not rated any movies

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	Eve (5)
→ Love at last	5	5	0	0	?
Romance forever	5	?	?	0	?
Cute puppies of love	?	4	0	?	?
Nonstop car chases	0	0	5	4	?
→ Swords vs. karate	0	0	5	?	?

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix}$$

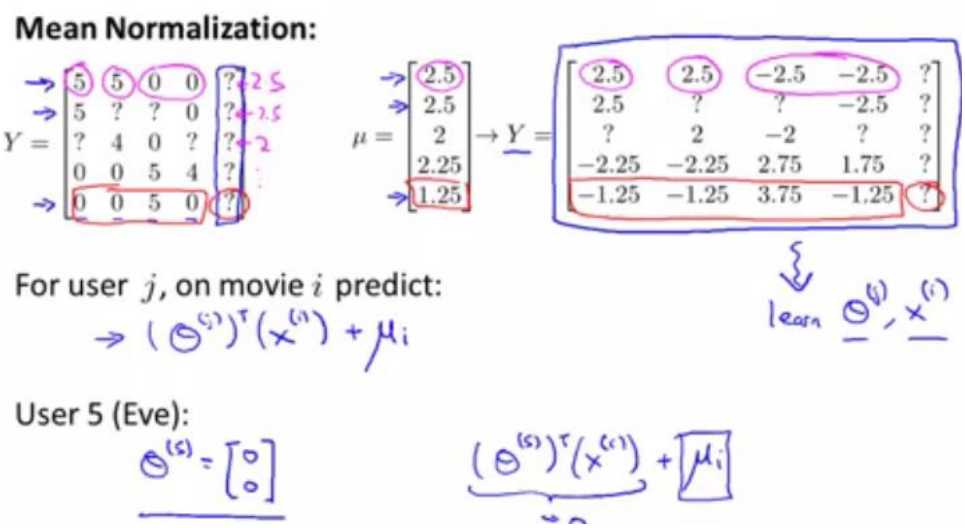
$$\min_{x^{(1)},\dots,x^{(n_m)}, \theta^{(1)},\dots,\theta^{(n_u)}} \frac{1}{2} \sum_{(i,j):r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

$n = 2$ $\quad \theta^{(5)} \in \mathbb{R}^2$ $\quad \theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$\frac{\lambda}{2} \left[(\theta_1^{(5)})^2 + (\theta_2^{(5)})^2 \right]$ ←

$(\theta^{(5)})^T x^{(i)} = 0$

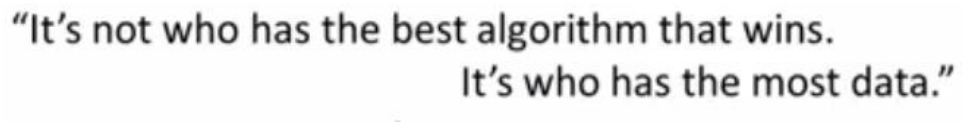
Mean Normalization:



Week 10

Monday, July 24, 2017
12:27 AM

Gradient Descent with Large Datasets

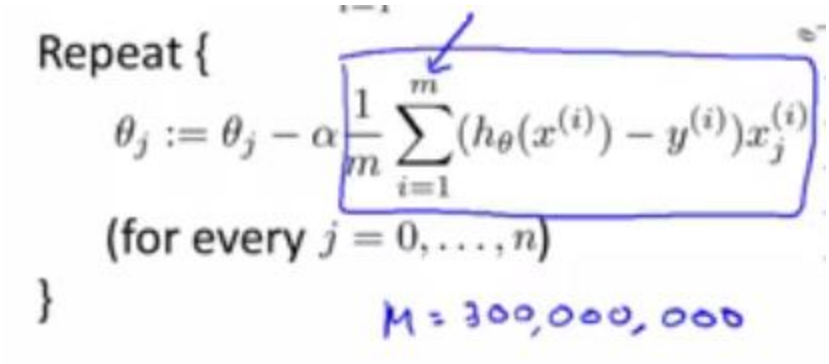


Large data => More computation.

How can you say that using more data will improve performance

> Plot a learning curve for a range of values of m and verify that the algorithm has high variance when m is small.

Stochastic Gradient Descent:

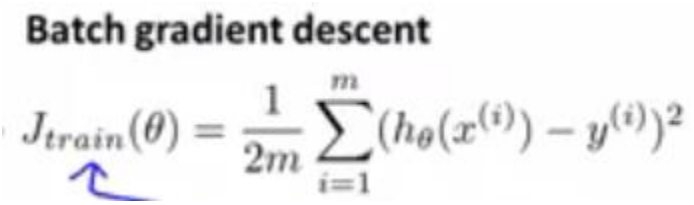


The highlighted term above will be very computer intensive.
This is called Batch Gradient Descent.
One step of the gradient descent needs us to go through all the data.

We need not go through all the data for making a gradient descent update. We do this with every example.

Every iteration will be faster and each iteration can move towards or away from the global minimum.

We end up going close to global min and keep moving there.



Stochastic gradient descent

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$$

1. Randomly shuffle dataset.

2. Repeat {

for $i = 1, \dots, m$ {

$\theta_j := \theta_j - \alpha\,(h_\theta(x^{(i)}) - y^{(i)})\cdot x_j^{(i)}$

(for $j = 0, \dots, n$)

}

}

$\rightarrow \frac{\partial}{\partial \theta_j} cost(\theta, (x^{(i)}, y^{(i)}))$

Mini Batch Gradient Descent:

Batch gradient descent: Use all m examples in each iteration

Stochastic gradient descent: Use 1 example in each iteration

Mini-batch gradient descent: Use b examples in each iteration

b = mini-batch size. $b = 10$. 2 - 100

Mini-batch gradient descent

Say $b = 10, m = 1000.$

Repeat {

for $i = 1, 11, 21, 31, \ldots, 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

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

}

}

Makes more progress over the batch gradient decent.
Why b why not 1 ? Ans: Vectorization will be faster. => Parallelization.
Takes advantage for the vectorization.
Needs to tune one more parameter b.

Stochastic Gradient Descent Convergence:

Week 11

Sunday, August 06, 2017
2:22 PM

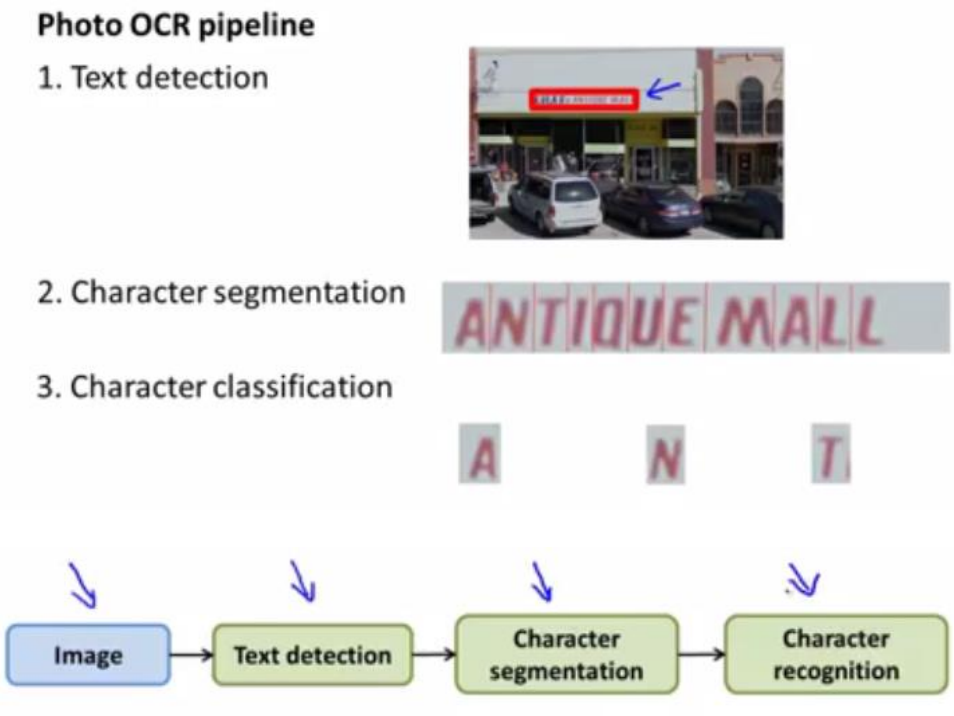
Application Example:

Problem Description and Pipeline:

Photo OCR - Photo Optical Character Recognition.

i. Given an Image photo. The algorithm should detect and read the text in the image.
ii. This could be helpful for the build people
iii. Also for the Car Navigation.

Phot OCR Pipeline:



Sliding Windows:

Text detection:

i. Text aspect ratio varies.
ii. We can co relate this to pedestrian detection in the image.
    1. This is simpler we can use fixed aspect ratio for different pedestrians.
    2. They are be at different distances but the aspect ratio i.e. height to width ratio will be same

Supervised learning can be used for pedestrian detection. For checking if the image has the pedestrian or not.

Sliding Window is traversed over the image to check if the window has the pedestrian or not

We generally take a larger patch and resize it to detect the pedestrians.

Similar to pedestrian detection we can get positive examples for images with and without test.

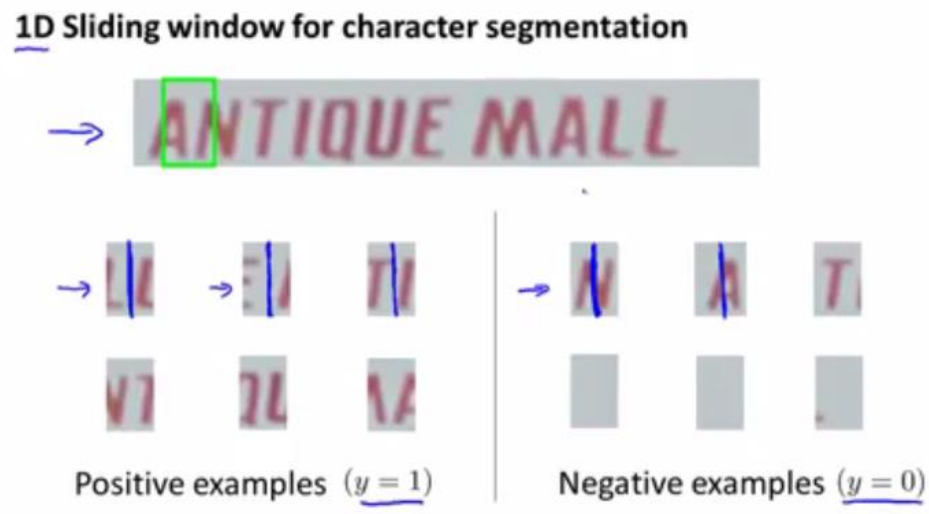
i. We now train model a supervised learning.
ii. Do sliding window detection.

We need to draw rectangle around the text regions. We use the Expanders by checking the proximities of the pixels and make the in one rectangles.

We can discard few rectangles based on the aspect ratios that does not look normal for the test boxes.

Character Segmentation:

Get positive and negative examples for detecting the split between the characters. Now we can train another supervised learning algorithm



Character Classification:

We can apply Neural Nets or etc. to classify the images to alphabets.

Getting lots of Data and Artificial Data:

We have two main variations:

i. Generate new data from scratch
ii. Get data from already existing data.

For images: (Photo OCR)

i. Color does not matter for text detection here.
ii. We can get different font styles here.
iii. We can change the background of the character.
iv. We can use Scaling or Cropping or Rotations.
v. We can introduce distortions into the already existing images.

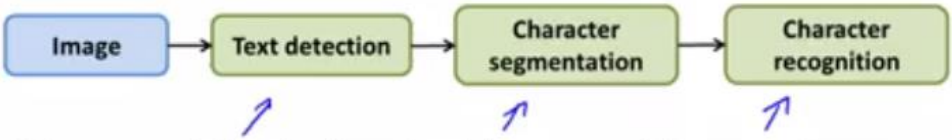
Discussion on getting more data

1. Make sure you have a low bias classifier before expending the effort. (Plot learning curves). E.g. keep increasing the number of features/number of hidden units in neural network until you have a low bias classifier.
2. "How much work would it be to get 10x as much data as we currently have?"
   - Artificial data synthesis
   - Collect/label it yourself
   - "Crowd source" (E.g. Amazon Mechanical Turk)

Ceiling Analysis: What part of Pipeline to Work on Next

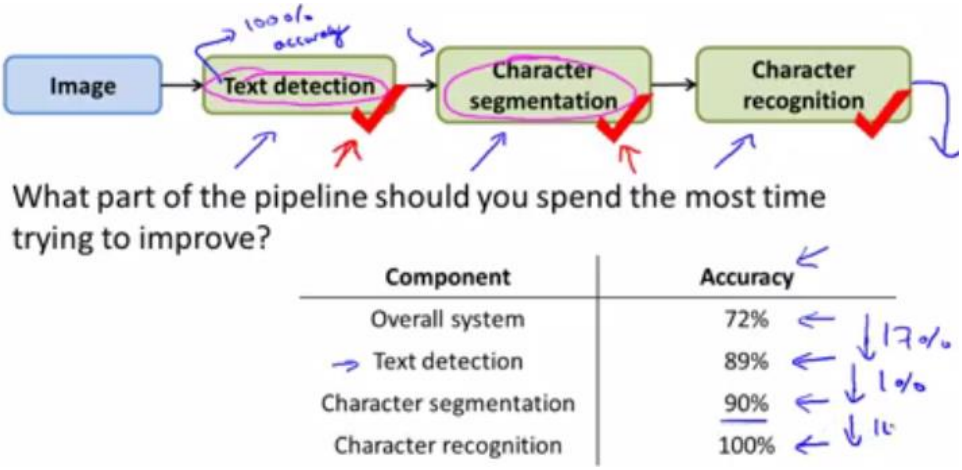
We must find the part of the work that will give maximum performance improvement to focus the efforts on.

Estimating the errors due to each component (ceiling analysis)

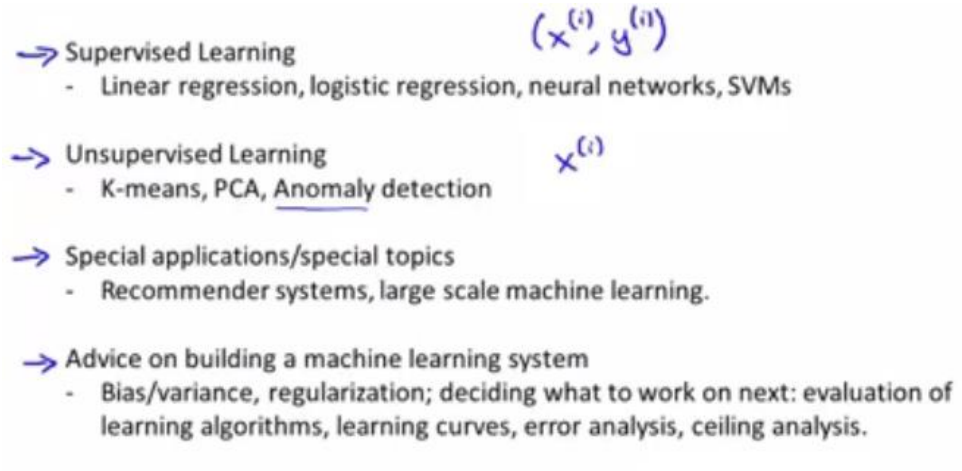


What part of the pipeline should you spend the most time trying to improve?

We will find the accuracy of the system by going through each block at a time.
We will assume that all the blocks earlier in the pipeline are working with 100% accuracy while I am checking the current block.
We can simply use the labels as predictions to act 100% accuracy of the previous blocks.
This way we will find the accuracy for the entire system every time by focusing on the individual block.



Summary:



Further reading

Monday, May 29, 2017
3:43 PM

- Andrew Ng on L1 vs L2 regularization
  - **Summary:** L2 is the "popular" technique for regularization. However, in this paper, Andrew Ng argues that L1 is often more efficient, taking logistic regression as an example; given (among other factors) that it offers "automatic" feature selection.
  - **Note**: The main difference is that in L1, we use a parameter lambda to minimize $|\Theta|$ rather than Θ^2
    ; L1 has a property called sparsity due to which many parameters are not just reduced but made 0, thus offering built-in feature selection

- Back Propagation.
  http://neuralnetworksanddeeplearning.com/chap2.html