Coursera

Machine Learning

Stanford University
by Andrew Ng

Fall 2019
[Sep-Nov]

Machine Learning - Andrew Ng Coursera

Goal! → Minimization : $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

Measure the accuracy of our hypothesis

Cost Function

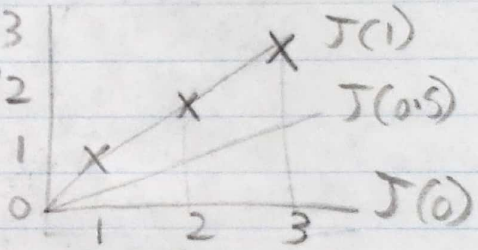
$h_\theta(x^{(i)}) = \hat{y}_i$ Squared error function calculated vs actual

$\frac{1}{2}$ cancels out with the derivative term of square function

- That function is basically $\frac{1}{2}\bar{x}$ where $\bar{x}$ is the mean of the squares $h_\theta(x_i) - y_i$
- $h_\theta(x) = \theta_0 + \theta_1 x$ (Basically $y = mx + c$)

$J(\theta_1)$
$\theta_0 = 0$
Simplified Version

e.g. $J(1) = \frac{1}{2\times3}\left[(1-1)^2 + (2-2)^2 + (3-3)^2\right]$
$= 0$



$J(0.5) = \frac{1}{2\times3}\left[(-0.5)^2 + (-1)^2 + (-1.5)^2\right]$
$= 0.58$

$J(0) = \frac{1}{6}\left[(-1)^2 + (-2)^2 + (-3)^2\right]$
$= 14/6$

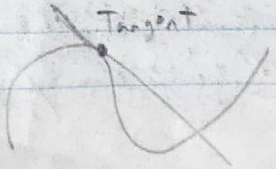
To each value of θ_1 → we derive a different accuracy
→ objective to minimize θ_1) ≈ J(1)

Contour graphs — Most efficient

- Need a software to minimize θ_0 & θ_1 for J automatically

- Derivative of a function of a single variable at a chosen input value
→ Best Linear Approximation
→ Instantaneous rate of change

$h_\theta(x)$ = prediction



("Batch") Gradient descent for Linear Regression

Iterative method

⇒ Convex shape → 1 Local optimum (Global optimum)

Other functions → Ther shapes → can have more than 1 optimum

J is a convex quadratic function $ax^2+bx+c=0$

$$x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

$y = mx + b$

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

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

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

$x \rightarrow$ inputs

$y \rightarrow$ outputs

Need to find θ_0 & θ_1 for optimal linear function

Quiz

2) $J(0,1) \approx h_\theta(x) = 0 + x$ $[y = 1x + 0]$ $[y = x]$

$= x$

$$\frac{1}{2\times4}\left[(3-2)^2 + (1-2)^2 + (0-1)^2 + (4-3)^2\right]$$

$$\frac{1}{8}[1+1+1+1] = \frac{4}{8} = \frac{1}{2}$$

3) $\theta_0 = 0$ $\theta_1 = 1.5$ $h_\theta(x) = \theta_1 x + \theta_0$

$h_\theta(2) = 1.5 \times 2 + 0$

$= 3$

Matrix properties

- Commutative but matrix multiplication is not commutative
- Associative & matrix multiplication is associative as well

Identity matrix $\begin{bmatrix}1&0\\0&1\end{bmatrix}$ $\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$ .....

$A \cdot I = I \cdot A = A$

Inverse $A(A^{-1}) = I$ if no inverse = singular

A^T Transpose $\underset{2\times3}{\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}} = \underset{3\times2}{\begin{bmatrix}1&4\\2&5\\3&6\end{bmatrix}}$

$A_{11} = A_{11}$
$A_{23} = A_{32}$

$\begin{bmatrix}a&b\\c&d\end{bmatrix}^{-1} = \frac{1}{ad-bc}\begin{bmatrix}d&-b\\-c&a\end{bmatrix}$

$ad-bc$ ↑ determinant

Quiz: $16 - 8 - 12 = -4$

More than 1 input / variable → Multivariate linear regression

$h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots\dots$

↓ $1 \times (n+1)$ matrix

$x_0^{(i)} = 1$ in this course

$n \geq 1$

Repeat $\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$ θ_j for $j = 0, \dots n$

To speed up gradient descent

- Feature scaling → get every feature into $-1 \leq x_i \leq 1$ approx. for fast convergence [Not much bigger or smaller]
  $\frac{x_1}{s_1}$
- Mean normalization $\xrightarrow{\text{e.g.}}$ $x_1 = \frac{\text{size} - 1000}{2000}$ $x_2 = \frac{\text{bedrooms} - 2}{5}$
  $x_i \to \frac{x_1 - \mu_1}{s_1 \text{ (range)}}$

  approx $-0.5 \leq x_1 \leq 0.5$, $-0.5 \leq x_2 \leq 0.5$

If α is too small : slow convergence
large : may not converge

Debugging gradient descent : Plot
Automatic convergence test : Declare convergence if $J(\theta)$ decreases by less than E in one iteration (E ↑ small value)

Improve features?
→ Combine into 1 e.g. $x_3 = x_1 x_2$
→ Hypothesis function can be linear, quadratic, cubic, square root...
e.g. if we only have 1 feature x_1, we can create $x_2 = x_1^2$ for better $x_3 = x_1^3$ fit curves

If that is done, feature scaling is important

- Besides gradient descent, there is the normal equation

$$\theta = (X^T X)^{-1} X^T y$$

→ m training examples & n features

Gradient Descent	Normal Equation
→ Need to chose α	→ No need to chose α
→ Needs many iterations	→ Don't need to iterate
→ Works well when n is large	→ Need to compute $(X^T X)^{-1}$
	$O(n^3)$
	Slow if n is large > 10,000
	• Doesn't work in so many cases

randn — normal distribution
rand — uniform distribution

[are] What if $X^T X$ is non-invertible e.g. Singular
• Redundant features. 2 features closely related
• Too many features

1 2 1 1 2 3
2 2 2 3 4
1 3 4 5

Quiz

1) $\frac{89-81}{25} = 0.32$

3) $((8\times23)(23\times6))^{-1}(6\times23)(23\times1)$
$(6\times6)^{-1}(6\times1)$
$(6\times6)(6\times1) = (6\times1)$

X = data(:,1) % All of column 1

Y = data(:,2) % All of column 2

$J = \frac{1}{2m}\sum_{i=1}^{m}((\theta_0 x_0 + \theta_1 x_1) - y)^2$

e.g $0\times1 + 0\times6 - 17.5^2 = 320.41$

Compute cost

Assignment

City Population	Truck profit

K = 1 : m
theta(i) .* X(K,2)

$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(\theta_0 x_0^i + \theta_1 x_1^i - y^i)$

$\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}(\theta_0 x_0^i + \theta_1 x_1^i - y^i)\, x_1^{(i)}$

sum in Matlab to sum array elements

Week 3

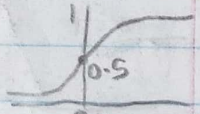
Logistic Regression → For classification

$0 \le h_\theta(x) \le 1$

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

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

$\Rightarrow h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ Logistic fn or Sigmoid fn

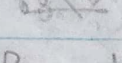


$h_\theta(x) = P(y=1 \mid x;\theta)$

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

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

Eg • Two y values 1 & 0 if $z \ge 0 = 1$ $z = \theta^T x$
$z < 0 = 0$

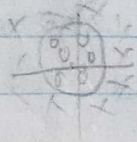


• Decision Boundary → Based on hypothesis and not data set

- if non linear e.g $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$
(-1, 0, 0, 1, 1)

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

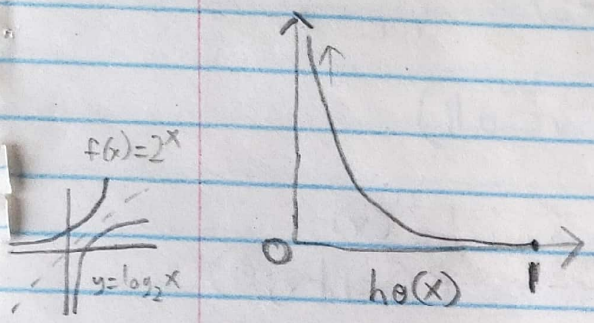
$x_1^2 + x_2^2 = 1$



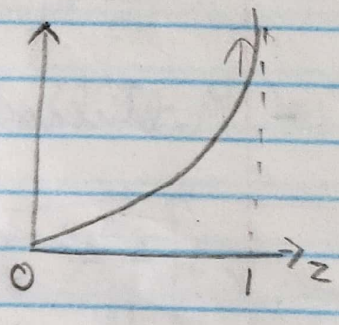
Logistic regression cost function

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

if $y=1$



if $y=0$



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

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

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

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

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

Vectorized $J(\theta) = \frac{1}{m}\cdot(-y^T\log(h) - (1-y)^T\log(1-h))$

To minimize θ, Gradient Descent

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

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

Gradient of cost function

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

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

Same equation as linear regression, only difference is that h_θ has a new definition

Linear $h_\theta = \theta^T x$ $\qquad$ Logistic $h_\theta = \frac{1}{1+e^{-\theta^T x}}$

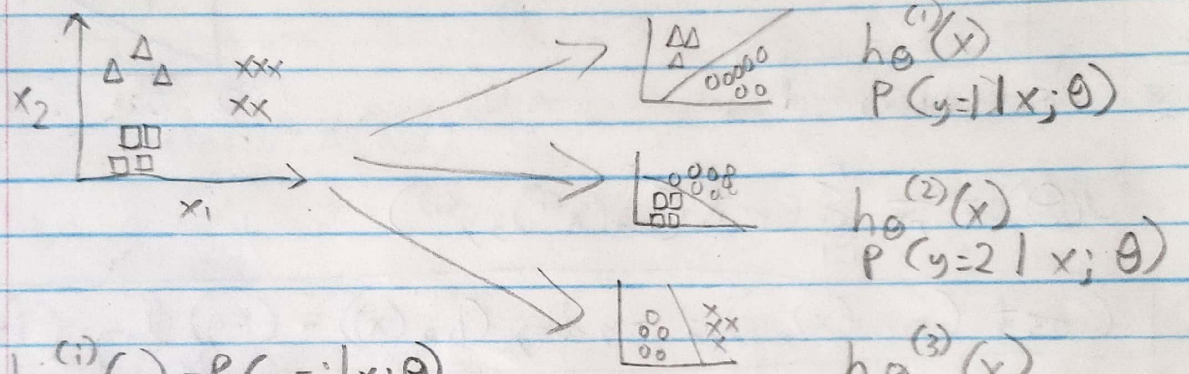
Besides Gradient Descent, there is:
- Conjugate gradient, BFGS, L-BFGS → Better, but complex!
- fminunc Octave/Matlab function

e.g. $\theta = \begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}$ $J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$ → ;Val

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

$\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$] → Gradient

- Multiclass classification (One vs all)



$h_\theta^{(i)}(x) = P(y=i|x;\theta)$
$(i = 1,2,3)$

Pick max $h_\theta^{(i)}(x)$ //

Solving Overfitting

- Regularization → small values for parameters $\theta_0, \theta_1, \theta_2, \ldots \theta_n$
  - To fit a fn appropriately
  - features are $x_1, x_2, \ldots$

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

What λ does? Balance → Fitting data set + keeping params small (Regularization term)

- If λ too high → Algorithm results in underfitting
  ~ It'll just be a straight line horizontally

• Regularized Linear Regression

- Gradient Descent

Repeat {

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

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

$$\simeq \theta_j \underbrace{\left(1 - \alpha \frac{\lambda}{m}\right)}_{\text{Always } < 1} - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

}

- Normal Equation

Regularization makes this always invertible even if $m < n$ (examples < features)

$$\theta = (X^T X + \lambda \cdot L)^{-1} X^T y \qquad L = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

Dimension $(n+1) \times (n+1)$

• Regularized Logistic Regression

Gradient Descent → Same exactly as above for linear
→ Only difference is $h_\theta = \frac{1}{1+e^{-\theta^T x}}$ instead of $\theta^T x$

Cost function

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

$\theta_j^2 = \theta_j^T \theta_j \quad j \geq 1$

Gradient of cost function

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \qquad j = 0$$

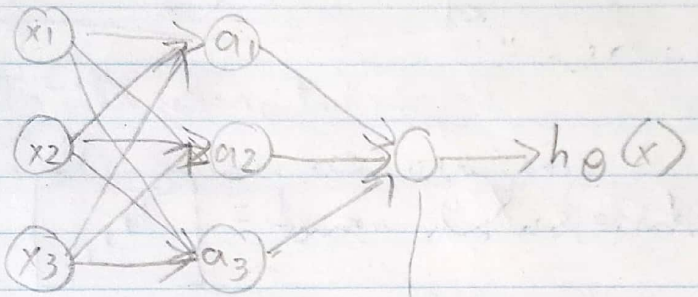
$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j \qquad j \geq 1$$

Week 4 - Neural Networks

- Can't use logistic regression when features (n) is very large → computationally expensive
- Neural networks → Similar to how the neurons in the brain work (Simulating neurons)

Dendrite → (computation → Axon) —electrical impulse→ Neuron

Neuron



Can add bias x_0, a_0 etc

Layer 1 — input (x) | Layer 2 — hidden | 3 — Output (y)

$a_i^{(j)}$ = "activation of unit i in layer j"

$\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j+1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"} \quad \text{"}$$

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

$\Theta^{(1)} \in \mathbb{R}^{3\times 4}$

- s_j units layer j − s_{j+1} units layer $j+1$ = $\Theta^{(j)}$ will be $s_{j+1} \times (s_j + 1)$
- eg layer 1: 2 input nodes, layer 2: 4 activation nodes — $\Theta^{(1)} = 4\times 3$, $(s_{j+1} \times [s_j + 1])$

Vectorized Implementation

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} \rightarrow \begin{array}{l} a_1^{(2)} = g(z_1^{(2)}) \\ a_2^{(2)} = g(z_2^{(2)}) \\ \cdots \end{array}$$

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

x can be replaced with $a^{(1)}$ for consistency

add $a_0^{(2)} = 1$

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

- Can be used to simulate logical gates
- Can have multiple output units

$$\begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} \rightarrow \begin{bmatrix} a_0^{(2)} \\ a_1^{(2)} \\ \cdots \\ a_n^{(2)} \end{bmatrix} \rightarrow \begin{bmatrix} a_0^{(3)} \\ a_1^{(3)} \\ \cdots \\ a_n^{(3)} \end{bmatrix} \rightarrow \cdots \rightarrow \begin{bmatrix} h_\Theta(x)_1 \\ h_\Theta(x)_2 \\ h_\Theta(x)_3 \\ h_\Theta(x)_4 \end{bmatrix}$$

if 4 classes

- Multiple output units: One-vs-all

Week 5 - Cost Function & Backpropagation

Neural Network Cost Function:

#of outputs

$h_\Theta(x) \in \mathbb{R}^K$

$(h_\Theta(x))_i = i^{th}$ output

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

L = total no of layers in a network $\leftarrow L = 4$

s_l = no of units in layer l $\leftarrow s_1 = 3$, $s_3 = 5$, $s_4 = 4$

$y = 0$ or 1 — 1 output unit

$y \in \mathbb{R}^K$ — K output units

e.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$ $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$ $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$ $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

Notes for cost function:

- Double sum adds up the regression costs calculated for each cell in the output layer
- Triple sum adds up the squares of all individual Θs in the entire network
- i in triple sum does not refer to training example i

← Backpropagation

To minimize cost function (same as gradient descent in logistic & linear regression)

To minimize $J(\Theta)$, we need code to compute

- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$ ← By backpropagation

Forward propagation recap

$a^{(1)} = x$

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

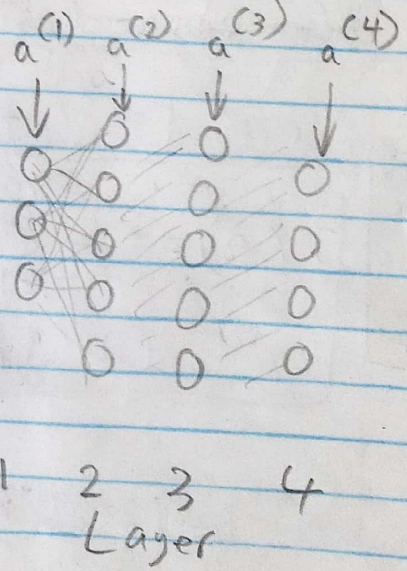
$a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)

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

$a^{(3)} = g(z^{(3)})$ (add $a_0^{(3)}$)

$z^{(4)} = \Theta^{(3)} a^{(3)}$

$a^{(4)} = h_\Theta(x) = g(z^{(4)})$



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

For each output unit (layer $L=4$)

$$\delta_j^{(4)} = a_j^{(4)} - y_j \qquad (h_\theta(x))_j$$

$$\delta^{(3)} = (\theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)}) \qquad a^{(3)} .* (1 - a^{(3)})$$

$$\delta^{(2)} = (\theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)}) \qquad a^{(2)} .* (1 - a^{(2)})$$

$$\delta^{(1)} = X$$ Input layer → Features observed so no errors

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = a_j^{(l)} \delta_i^{(l+1)}$$ (ignoring λ)

Backpropagation algorithm basically:

- $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$
- set $\Delta_{ij}^{(l)} = 0$

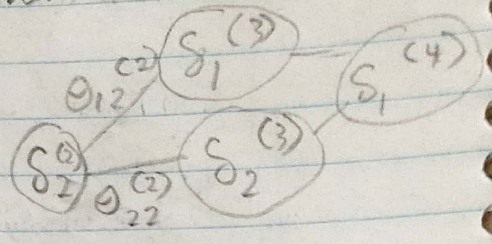
- for $i = 1$ to m
  - set $a^{(1)} = x^{(i)}$
  - perform FP $a^{(l)}$ for $l = 2, 3, \ldots, L$
  - using $y^{(i)}$ compute $\delta^{(L)} = a^{(L)} - y^{(i)}$
  - compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots \delta^{(2)}$
  - $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ $\quad [\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T]$

- $D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \theta_{ij}^{(l)}$ $\quad j \neq 0$
- $D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)}$ $\quad j = 0$

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = D_{ij}^{(l)}$$

• BP is basically the opposite of FP in calculations

e.g. $\delta_2^{(2)} = \theta_{12}^{(2)} \delta_1^{(3)} + \theta_{22}^{(2)} \delta_2^{(3)}$



- To use functions in matlab for θ & D
  - thetaVector = [Theta1(:); Theta2(:); Theta3(:);]
    deltaVector = [D1(:); D2(:); D3(:);]
  - Theta1 = reshape(thetaVetor(1:110), 10, 11)
    Theta2 = reshape(thetaVetor(111:220), 10, 11)
    Theta3 = reshape(thetaVetor(221:231), 1, 11)

$\Theta 1$ 10x11
$\Theta 2$ 10x11
$\Theta 3$ 1x11

• Gradient Checking: To see if backpropagation works as intended

$$\frac{\partial}{\partial \theta} J(\theta) \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon} \qquad \varepsilon = 10^{-4} \text{ (small value)}$$

- Compare gradApprox (from backpropagation) $\approx$ deltaVector
- Makes code very slow – Only do it once

• Initializing θ → must be random weights

eg Theta1 = rand(10,11) * (2 * INIT_EPSILON) - INIT_EPSILON
Theta2 = " (10,11) "
Theta3 = " (1,11) "

rand = Between 0&1

RECAP • Training a neural network:
1) Randomly initialize weights
2) Implement FP to get $h_\theta(x^{(i)})$ for any $x^{(i)}$
3) Compute $J(\theta)$
4) Implement BP to compute $\frac{\partial}{\partial \theta_{jk}^{(l)}} J(\theta)$
5) Gradient Checking [Then disable]
6) Use GD or advanced optimization methods with backpropagation to try to minimize $J(\theta)$ as a function of parameters θ

Evaluating a Learning Algorithm - Week 6

- Can trouble shoot errors in our predictions by
  - Getting more training examples
  - Decreasing / Increasing features
  - Trying polynomial features
  - Increasing or decreasing λ

- To evaluate a hypothesis we can split up data into training and test set. It's usually a 70/30 split
  1) Learn Θ & minimize $J_{Train}(\Theta)$ using the training set
  2) Compute $J_{Test}(\Theta)$

- Test set error
  - Linear regression: $J_{test}(\Theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h\Theta(x_{test}^{(i)} - y_{test}^{(i)})^2$
  - Classification: $err(h_\theta(x), y) = \begin{matrix} 1 & \text{if } h_\theta(x) \geq 0.5 \; y=0 \text{ or } h<0.5 \; y=1 \\ 0 & \text{otherwise} \end{matrix}$

Proportion of misclassified test data

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

- Given many models with different polynomial degrees, we can use a systematic approach to identify best function. Test each degree of polynomial and look at error result.
  Training (60-70) / Validation (15-20) / Test (15-20)

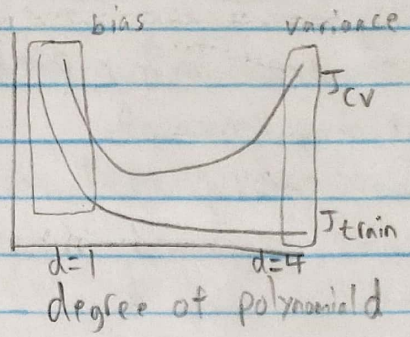
- Can now calculate 3 separate error values for 3 different sets:
  1) Optimize Θ for training set using each polynomial degree
  2) Find polynomial degree with least error using cross validation set
  3) Estimate generalization error using test set with $J_{test}(\Theta_d)$ ← From step (2)

This way, polynomial degree not trained using test set

- High bias → Underfit High variance → Overfit

Example:

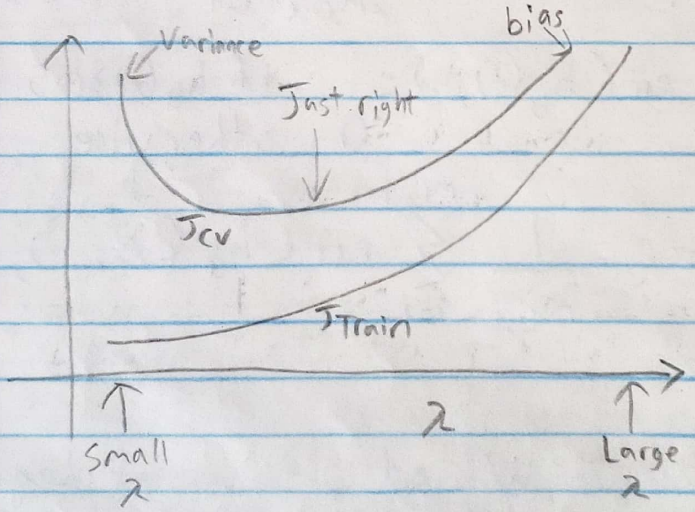
$J_{CV} \approx J_{Test}$



- Bias (underfit): High J_{Train} & J_{CV}
- Variance (overfit): High J_{CV} but low J_{Train}

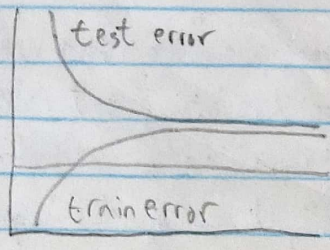
- Choosing λ for regularization
  1) Try $\lambda = 0$ → $\theta^{(1)}$ → $J_{CV}(\theta^{(1)})$
  2) Try $\lambda = 0.01$ → $\theta^{(2)}$ → $J_{CV}(\theta^{(2)})$
  3) $\lambda = 0.02$ 〃 〃
  4) $\lambda = 0.04$ 〃 〃
  〃 〃
  12) $\lambda = 10.24$ → $\theta^{(12)}$ → $J_{CV}(\theta^{(12)})$

Lets say we pick $\theta^{(5)}$
∴ $J_{test}(\theta^{(5)})$



- As training set gets larger, error for quadratic function increases
- The error value will plateau after certain m, or training set size

- High Bias

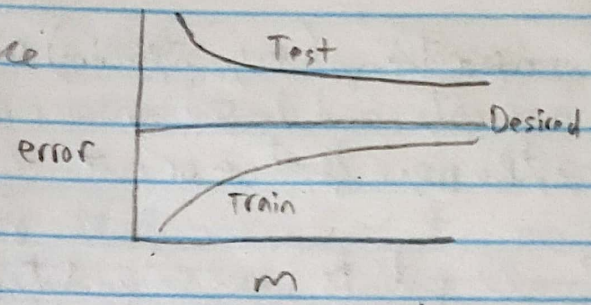


Low m: High J_{CV}, Low J_{Train}

High m: High J_{CV}, High J_{Train}

Getting more data won't help!

- High Variance



Low m: High J_{cv}, Low J_{train}

High m: $J_{train} < J_{cv}$, Difference stays big

More data helps!

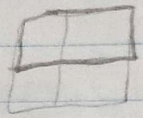
- Some fixes
  - More training examples → Fixes high variance
  - Smaller set of features → Fixes high variance
  - Get additional features → Fixes high bias
  - Add polynomial features → Fixes high bias
  - Decrease λ → Fixes high bias
  - Increase λ → Fixes high variance
- So for Neural Networks
  - Small NN → Fewer parameters → More prone for underfitting (computationally cheap)
  - Large NN → More parameters → More prone for overfitting (Expensive)
    → Use λ to address overfitting
- Building spam classifier via supervised learning
  x = features of email (words in email) y = spam (1) or not spam (0)

  How to improve?
  - Collect lots of data (doesn't always work)
  - Develop sophisticated features (e.g. email header data)
  - Develop algorithms to process input in different ways (e.g. misspellings in spam)
- How to solve machine learning problems? (cross validation)
  1) Start with a simple algorithm & test it early on CV data
  2) Plot learning curves to decide if more data, features etc are needed
  3) Manually examine errors on examples in CV set & spot trend where most errors have made

Quiz P=0.088179
R=0.85

- We should try new things (e.g. Stemming) and get a numerical value for our error rate and based on our result decide whether we want to keep the new feature or not

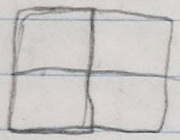
- Precision: Of all patients where we predicted $y=1$, what fraction has cancer?



$$\frac{\text{True positives}}{\text{True positives} + \text{False positives}}$$

Methods of evaluating Learning Algorithms for Skewed Data

Recall: Of all patients that actually have cancer, what fraction did we correctly detect as having cancer?



$$\frac{\text{True positives}}{\text{True positives} + \text{False negatives}}$$

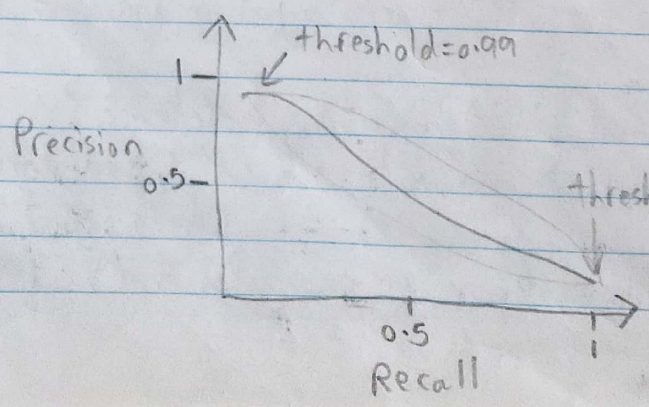
		Actual Class: 1	Actual Class: 0
Predicted Class	1	True positives	False positives
	0	False negatives	True negatives

Accuracy

$$\frac{\text{True pos} + \text{True neg}}{\text{Total}}$$

- Trading off precision & recall

$h_\theta(x) \geq$ threshold



- F_1 Score: $\frac{2PR}{P+R}$ - Good single numerical analysis metric

Use threshold that maximizes F_1 on CV set.

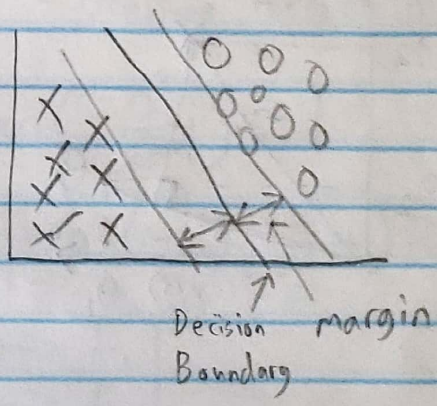
Week 7 - Support Vector Machines

$$J(\theta) = \min_\theta C \sum_{i=1}^{m} \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

The difference between that & logistic regression $J(\theta)$:
- $\text{cost}_1(\theta^T x^{(i)}) = -\log h_\theta(x^{(i)})$ & same for $\text{cost}_0(\theta^T x^{(i)})$
- Constants $\frac{1}{m}$ taken out
- λ replaced by C, where C controls non-regularized parameter $C = 1/\lambda$

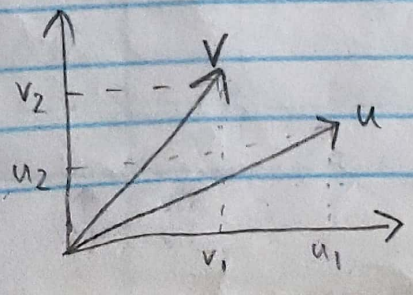
Hypothesis: $h_\theta(x) \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$ s.t. $\theta^T x^{(i)} \geq 1$, $\theta^T x^{(i)} \leq -1$

- Large margin intuition



Want to maximize margin!

- Vector Inner Product



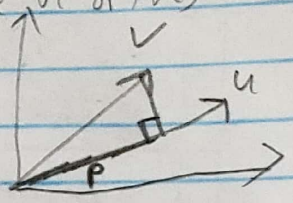
$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$ $v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$

$u^T v = ?$

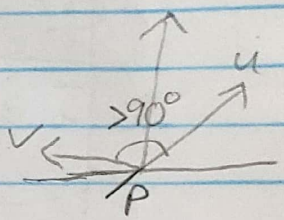
$\|u\|$ = length of vector u $= \sqrt{u_1^2 + u_2^2}$

p = length of projection v onto u (could be -ve or +ve)

$u^T v = p \cdot \|u\| = \begin{bmatrix} u_1 & u_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = u_1 v_1 + u_2 v_2$ $[= v^T u]$



- If angle between u & v $> 90°$, p will be -ve



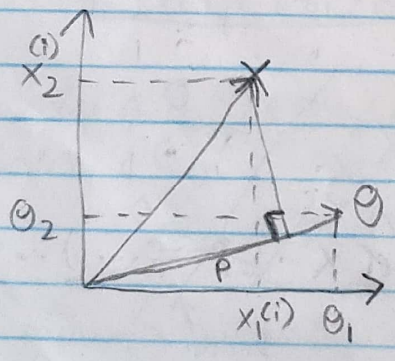
- SVM Decision Boundary mathematics

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^{m} \theta_j^2 = \frac{1}{2}(\theta_1^2 + \theta_2^2) = \frac{1}{2}\left(\sqrt{\theta_1^2 + \theta_2^2}\right)^2 = \frac{1}{2}\|\theta\|^2$$

s.t $\theta^T x^{(i)} \geq 1$ if $y^{(i)} = 1$

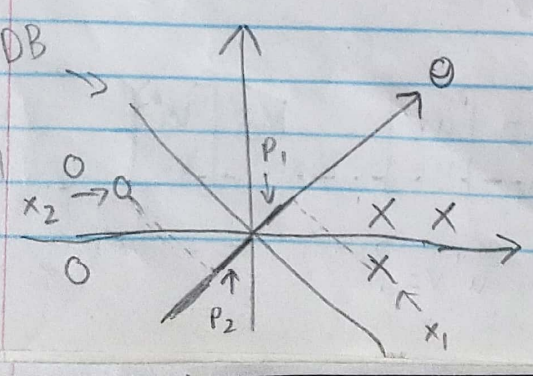
$\theta^T x^{(i)} \leq -1$ if $y^{(i)} = 0$

Simplification $\theta_0 = 0$
(means decision boundary needs to pass (0,0))

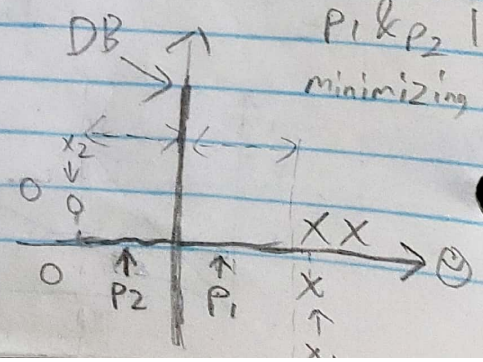


$$\theta^T x^{(i)} = p^{(i)} \cdot \|\theta\| = \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)}$$

e.g



$p^{(1)}$ & $p^{(2)}$ small
$\therefore \|\theta\|$ need to be large
$\therefore$ not good



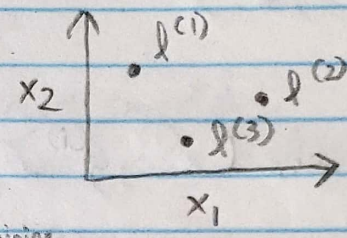
P_1 & P_2 large
minimizing $\|\theta\|$

Kernels for non-linear decision boundaries

e.g. $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \ldots \geq 0$

$x_1 \to f_1$, $x_2 \to f_2$, $x_1 x_2 \to f_3$, $x_1^2 \to f_4$, $x_2^2 \to f_5$

- Given x, compute new feature depending on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$



Training example

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

$f_2 = ///$

$f_3 = ///$

Kernel - Gaussian Kernels $K(x, l^{(1)})$

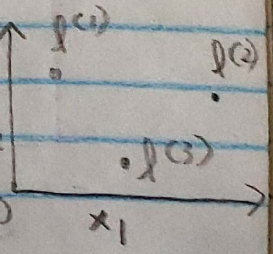
- If $x \approx l^{(1)}$ $\quad f_1 \approx \exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$

If x is far from l $\quad f_1 \approx \exp\left(-\frac{\text{large}\#^2}{2\sigma^2}\right) \approx 0$

each landmark define new feature: $l^{(1)} \to f_1$, $l^{(2)} \to f_2$, $l^{(3)} \to f_3$

- Predict "1" when $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

$\to \theta_0 = -0.5, \theta_1 = 1, \theta_2 = 1, \theta_3 = 0$



If point close to $l^{(1)}$ $f_1 \approx 1$ $f_2 \approx 0$ $f_3 \approx 0$ formula $= 0.5 \geq 0 = 1$ //

If point away from everything $f_1 = f_2 = f_3 \approx 0$ formula $-0.5 < 0 = 0$ //

SVM with Kernels

Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$
Choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)}$

Given example x: ← $x^{(i)}$

$f_1 = \text{sim}(x, l^{(1)})$
$f_2 = \text{sim}(x, l^{(2)})$
$\ldots$

For training example $(x^{(i)}, y^{(i)})$:

$x^{(i)} \rightarrow$
$f_1^{(i)} = \text{sim}(x^{(i)}, l^{(1)})$
$f_2^{(i)} = \text{sim}(x^{(i)}, l^{(2)})$
$\ldots$
$f_m^{(i)} = \text{sim}(x^{(i)}, l^{(m)})$

$f_i^{(i)} = \text{sim}(x^{(i)}, l^{(i)}) = 1$ ← $x^{(i)}$

$$f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ \ldots \\ f_m^{(i)} \end{bmatrix}$$

$f_0^{(i)} = 1$

- Hypothesis: Given x, compute features $f \in \mathbb{R}^{m+1}$
  → Predict "$y = 1$" if $\underline{\theta^T f} \geq 0$
  $\theta_0 f_0 + \theta_1 f_1 + \ldots + \theta_m f_m$

Training:

$$\min_\theta C \sum_{i=1}^{m} y^{(i)} \text{cost}_1(\theta^T f^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{m} \theta_j^2$$

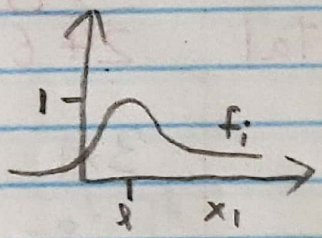
$\sum \theta_j^2 = \theta^T \theta$
$= \theta^T M \theta$ ← M = training examples

- Kernels can be used for other logistic techniques, but high computational costs

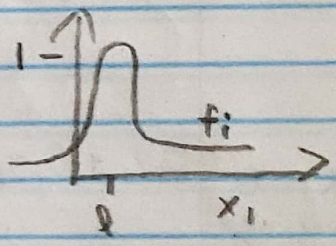
- SVM parameters:

$C \left[= \frac{1}{\lambda}\right]$ → Large C: Lower Bias, Higher Variance
→ Small C: Higher Bias, Low Variance

σ^2 Large σ^2
→ Vary more smooth
$\exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$ → Higher Bias, Lower Variance



Small σ^2
→ Vary less smooth
→ Lower Bias, Higher Variance



Machine Learning - Andrew Ng Coursera

Week 7 - SVMs in Practice (Continue)

- Use SVM software package to solve for parameters θ
  Need to specify:
  → parameter C
  → Kernel

- Linear Kernel → when n is large & m is small (n: features, m: examples)
  Gaussian Kernel → need to choose σ^2

There are other Kernels available

- Perform feature scaling before using Gaussian Kernel to avoid having certain features dominate

- Choose Kernel that performs best on CV data

- Multi-class classification
  → Many SVM packages already have built-in multi-class classification
  → Otherwise, use one vs all method (Train K SVMs, one to distinguish $y=i$ from the rest, for $i=1,2,\dots K$) get $\theta^1, \theta^2, \dots \theta^K$ (θ^1: $y=1$, θ^2: $y=2$)
  Pick class i with largest $(\theta^{(i)})^T x$

- If n is large relative to m
  → Logistic regression or SVM without a kernel
  If n is small, m is intermediate
  → Use SVM with Gaussian Kernel
  If n is small, m is large
  → Add features then use logistic regression or SVM without a kernel
  Neural network likely to work well for most of these settings but may be slower to train

Week 8 - Unsupervised Learning

Clustering

- Unsupervised learning : When we are given unlabeled data
  → We give it to an algorithm and we ask it to give us some structure

- Applications of clustering
  - Market segmentation
  - Organize computing clusters
  - Social network analysis
  - Astronomical data analysis

- Supervised : $\{(x^1, y^1), (x^2, y^2), (x^m, y^m)\}$
  Unsupervised : $\{x^1, x^2, x^m\}$

Clustering: K-Means Algorithm

- Input:
  - K (number of clusters)
  - Training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

  $x^{(i)} \in \mathbb{R}^n$ (drop $x_0 = 1$ convention)

Algorithm
- Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots \mu_K \in \mathbb{R}^n$
  Repeat {
  - Cluster Assignment: for i = 1 to m
    $c^{(i)}$:= index (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
  }

CA: $\min_K \Rightarrow_{c^{(i)}} \|x^{(i)} - \mu_K\|^2$

MC: $\mu_K = \frac{1}{n} \{x^n \in \mathbb{R}^n$

Optimization

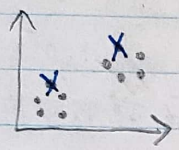
$$J(c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_K) = \frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

Cluster Assignment: Minimizes c
Move Centroid: Minimizes μ

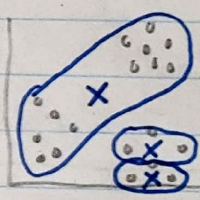
- How to randomly initialize the cluster centroids?

Most popular way Random Initialization

1) Should have $K < m$
2) Randomly pick K training examples
3) Set $\mu_1,\dots,\mu_K$ equal to these K examples



- Local optima might get stuck e.g



when its obvious

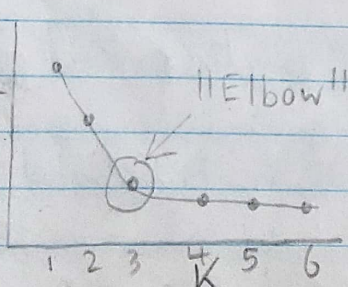
Random Initialization Algorithm

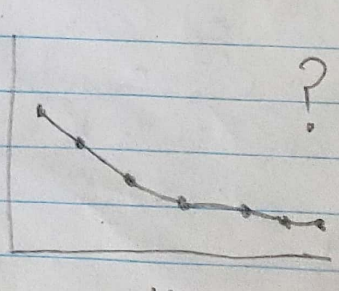
- For i=1 to 100 {
  Randomly initialize K-means
  Run K-means. Get $c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_K$
  Compute cost function $J(c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_K)$
  }

Pick clustering that gave lowest cost $J(\dots)$

Choosing value K [# of clusters]

- Method: Elbow Method





Does not always work

Method 2 : Evaluation

You're running K-means to get clusters to use for some later / downstream purpose. Evaluate K-means based on a metric for how it performs for that later purpose.

e.g. K=3 S, M, L or K=5 XS, S, M, L, XL
Can base on how much each sells / appeals to customers

Dimensionality Reduction

Motivation ① Data compression e.g. Reduce data from 2D to 1D

via graphs

$x^{(1)} \in \mathbb{R}^2 \rightarrow z^{(1)} \in \mathbb{R}$
$x^{(2)} \in \mathbb{R}^2 \rightarrow z^{(2)} \in \mathbb{R}$
$\vdots$
$x^{(m)} \rightarrow z^{(m)}$

$x^{(i)}$ is the result of x_1 & x_2
$z^{(i)}$ is new feature based on best fit line

Can reduce data from 3D to 2D [Projection]

$x^{(i)} \in \mathbb{R}^3 \rightarrow z^{(i)} \in \mathbb{R}^2$ $\quad z^{(i)} = \begin{bmatrix} z_1^{(i)} \\ z_2^{(i)} \end{bmatrix}$

Motivation ② Data Visualization - Also reduce dimensions
- To get a general overview / idea of our data

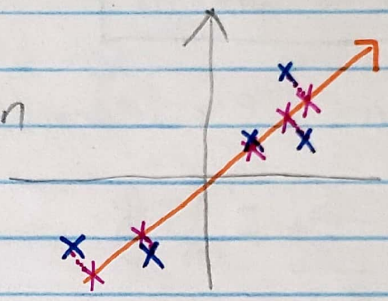
via data tables

$z^{(i)} \in \mathbb{R}^K$ is K-dimensional, we expect K=2 or 3 as we can plot 2D or 3D data but don't have ways to visualise higher dimensions.

Dimensionality Reduction - Principal Component Analysis

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

Blue actual points
Pink is the reduction



2D → 1D

- 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 = house size, x_2 = # of bedrooms), scale features to have comparable range of values

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

(s_j: s.d of j)

- Algorithm:
  - Reduce data from n-dimensions to K-dimensions
  - Compute "covariance matrix"

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

$x^{(i)} = n \times 1 \quad (x^{(i)})^T = 1 \times n$

$= n \times n$ matrix

- Compute "eigenvectors" of matrix Σ:

Octave Syntax → `[U, S, V] = svd(Σ)` → Singular Value Decomposition

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

$u^{(1)}, \ldots, u^{(K)}$ (first K columns)

- Reduce from n to K

- $z^{(i)} = U^T x = \underbrace{\begin{bmatrix} — u^{(1)} — \\ — u^{(2)} — \\ \vdots \\ — u^{(K)} — \end{bmatrix}}_{K \times n} \underbrace{x^{(i)}}_{n \times 1}$ $\quad z^{(i)} = K \times 1$

- Reconstruct from compressed representation:

$x_{approx} = U \cdot z$ ($n \times K$ · $K \times 1$) $\quad x_{approx} \in \mathbb{R}^n$ $n \times 1$

- Choosing the number of principal components

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

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

Typically choose K to be smallest value so that

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

[could be 0.05, 0.10]

→ "99% variance retained"

[95% - 90%]

Maximum Variance = Minimizes squared

Algorithm of choosing K:

Not the best ① - Try PCA with K=1 then 2 then 3 .....
- Compute $U, z^{(1)}, z^{(2)}, \ldots, z^{(m)}, x_{approx}^{(1)}, \ldots, x_{approx}^{(m)}$
- Check if equation in previous point ≤ 0.01

Octave Function ② $[U, S, V] = svd(Sigma)$

$$\begin{bmatrix} S_{11} & & 0 \\ & S_{22} & \\ & & \ddots \\ 0 & & S_{nn} \end{bmatrix}$$

For given k $$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$$

- For both pick the smallest value of K that retains 99% of the variance

Supervised learning speedup

- $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$
- Extract inputs:

Unlabeled dataset: $x^{(1)}, x^{(2)}, \ldots, x^{(m)} \in \mathbb{R}^{10000}$

$\downarrow$ PCA

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

$x \downarrow z$

- New training set:

$(z^{(1)}, y^{(1)}), (z^{(2)}, y^{(2)}), \ldots, (z^{(m)}, y^{(m)})$ $\quad h_\theta(z) = \frac{1}{1+e^{-\theta^T z}}$

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 $x_{cv}^{(i)}$ and $x_{test}^{(i)}$ in the CV & test sets.

Summary

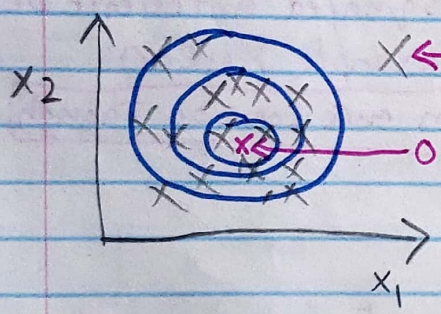
PCA Applications:
- Compression
  - Reduce memory/disk needed to store data
  - Speed up learning algorithm
- Visualization

Advices

- Bad use of PCA: To prevent overfitting
  - Use regularization instead
    - → Less likely to throw away valuable info
- Before implementing PCA, first try running whatever you want to do with the original / raw data $x^{(i)}$. Only if that doesn't do what you want, then implement PCA and consider using $z^{(i)}$.

Week 9 - Anomaly Detection

Density Estimation



$p(x_{test}) < \varepsilon \rightarrow$ Anomaly

$p(x_{test}) \geq \varepsilon \rightarrow$ OK

↑ Threshold

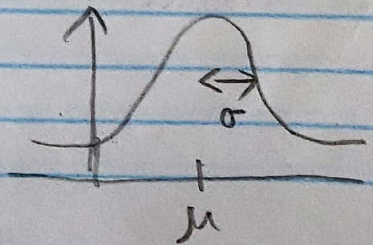
Examples

- Fraud Detection:
  - $x^{(i)}$ = features of user i's activities
  - Model $p(x)$ from data
  - Identify unusual users by checking which have $p(x) < \varepsilon$
- Manufacturing
- Monitoring computers in a data center

- Gaussian (Normal) distribution

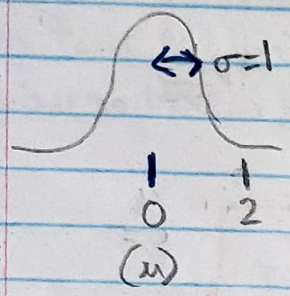
$x \sim N(\mu, \sigma^2)$

↑ Distributed; ↑ mean; ↑ Variance

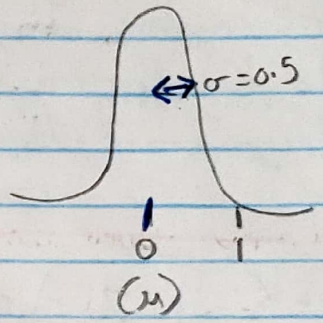


Density: $p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$

- $\mu = 0$ $\sigma = 1$



- $\mu = 0$ $\sigma = 0.5$ $\sigma^2 = 0.25$



Integral under curve equals 1 at all times

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

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

squared difference between example & mean

or m-1

Both work, m more common though

- Algorithm:

- Training set: $\{x^{(1)}, \ldots, x^{(m)}\}$ - Each example is $x \in \mathbb{R}^n$

- $p(x) = p(x_1; \mu_1, \sigma_1^2)\, p(x_2; \mu_2, \sigma_2^2) \ldots p(x_n; \mu_n, \sigma_n^2)$

where $x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$
$x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$
$x_m \sim \mathcal{N}(\mu_m, \sigma_m^2)$

$= \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$

Product

↖ Background Info

1) Choose features x_i that you think might be indicative of anomalous examples

2) Fit parameters $\mu_1, \dots, \mu_n$ & $\sigma_1^2, \dots, \sigma_n^2$

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

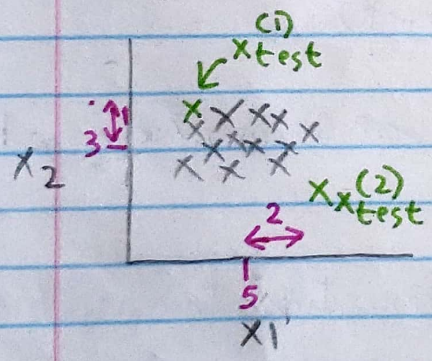
Both can be vectorized

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

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

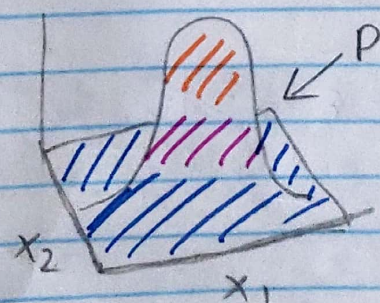
→ Anomaly if $p(x) < \varepsilon$

- Algorithm example



x_1 : $\mu_1 = 5$ $\sigma_1 = 2$ $\sigma_1^2 = 4$

x_2 : $\mu_2 = 3$ $\sigma_2 = 1$ $\sigma_2^2 = 1$



$$p(x) = p(x_1; \mu_1, \sigma_1^2) \times p(x_2; \mu_2, \sigma_2^2)$$

$\varepsilon = 0.02$

$p(x_{test}^{(1)}) = 0.0426 \geq \varepsilon$

$p(x_{test}^{(2)}) = 0.0021 < \varepsilon$ → Anomaly!

Building an Anomaly Detection System

- Aircraft engines motivating example
  - → 10000 good engines
  - → 20 flawed engines

→ Training set: 6000 good engines ($y=0$)
→ CV: 2000 good engines ($y=0$), 10 anomalous ($y=1$)
→ Test: 2000 good engines ($y=0$), 10 anomalous ($y=1$)

- Algorithm evaluation

1) Fit model $p(x)$ on training set $\{x^{(1)}, \ldots, x^{(m)}\}$
2) On a cross validation/test example x predict

$$y = \begin{cases} 1 & \text{if } p(x) < \varepsilon \text{ (anomaly)} \\ 0 & \text{if } p(x) \geq \varepsilon \text{ (normal)} \end{cases} \qquad (x_{test}^{(i)}, y_{test}^{(i)})$$

Note: Data is skewed as $y=0$ is more common than $y=1$

→ Possible evaluation metrics
- True positives, false positives, false negatives, true negatives
- Precision / Recall
- F_1 score

- Can also use CV set to choose ε & test on test set
- Can select ε that maximizes one of the evaluation metrics
  e.g. F_1 score

Anomaly Detection vs Supervised Learning

- Very small number of positive examples ($y=1$) (0-20 is common)
- 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

- Large number of positive & negative examples
- 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

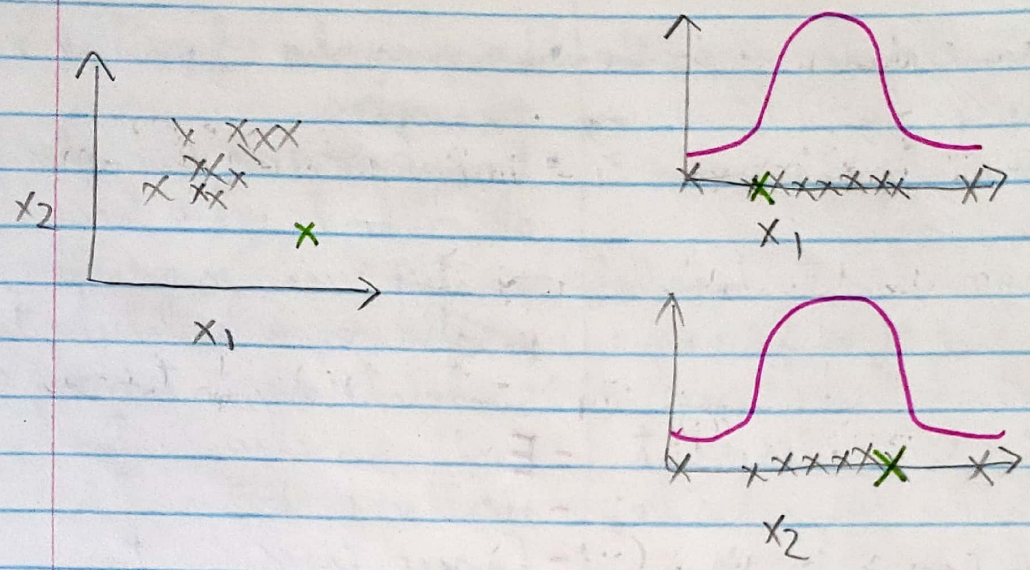
- Email spam classification
- Weather prediction
- Cancer classification

- Non-gaussian features: $x_1 \leftarrow \log(x_1)$ to transform it
  $x_2 \leftarrow \log(x_2 + c)$ "
  $x_3 \leftarrow x_3^{1/2}$ "
  Basically log & exponential functions

- Error analysis for anomaly detection
  → Most common problem:
  $p(x)$ is comparable (say, both large) for normal & anomalous examples
  → Solution? Add more features to distinguish anomalies

- Example: 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 = $\frac{\text{CPU load}}{\text{network traffic}}$ ← created feature

Multivariate Gaussian Distribution



- Not bad for both individually, but it's an anomaly
- $x \in \mathbb{R}^n$. Don't model $p(x_1), p(x_2)$ ... separately
- Model $p(x)$ all in one go
- $\mu \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$ (covariance matrix)

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

↖ Determinant

- e.g. From $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ to $\Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 1 \end{bmatrix}$ ← Variance of x_1 decreases

$$\Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$ ← Diagonal shift/turn

- Changing μ changes where the curve is centred at

Algorithm

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

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

2) Given new example x, compute

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

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

Relationship to original model

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

Corresponds to multivariate Gaussian

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

Original model is a special case of the multivariate model

Original Model	Multivariate Gaussian
• Manually create features to capture anomalies e.g. $x_3 = \frac{x_1}{x_2}$	• Automatically captures correlation between features
• Computationally cheaper (scales better to large n)	• Computationally more expensive
• OK even if m (training set) is small	• Must have $m > n$ or else Σ is non-invertible e.g. $m \geq 10n$ — make sure we have enough data to fit Σ parameters

Non-invertibility could be due to redundant features

Recommender Systems

Content-Based

Example:

	Movie	Alice (1) $\theta^{(1)}$	Bob (2) $\theta^{(2)}$	Carol (3) $\theta^{(3)}$	Dave (4) $\theta^{(4)}$	x_1 (romance)	x_2 (action)
$x^{(1)}$	Love last	5	5	0	0	0.9	0
$x^{(2)}$	Romance VI	5	?	?	0	1.0	0.01
$x^{(3)}$	Cute cats	?	4	0	?	0.99	0
$x^{(4)}$	Car chases	0	0	5	4	0.1	1.0
$x^{(5)}$	Swords	0	0	5	?	0	0.9

Users (columns) · attributes (x_1, x_2) · items (rows)

$n_u = 4$ #users

$n_m = 5$ #movies

$n = 2$ #categories

if $x_0 = 1$ $\quad x^{(1)} = \begin{bmatrix} 1 \\ 0.9 \\ 0 \end{bmatrix}$ $\quad x^{(2)} = \begin{bmatrix} 1 \\ 1 \\ 0.01 \end{bmatrix}$ ...

→ 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

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

- $r(i,j) = 1$ 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: $(\theta^{(j)})^T (x^{(i)})$
- $m^{(j)}$ = no. of movies rated by user j
- To learn $\theta^{(j)}$:

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

$\theta^{(j)} \in \mathbb{R}^{n+1}$

$i:r(i,j)=1$: movies rated

$m^{(j)}$ could be eliminated

- To learn $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$ given $x^{(1)}, x^{(2)}, \dots, x^{(n_m)}$

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

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

- Gradient Descent Update:

$\frac{\partial}{\partial \theta_K^{(j)}} J(\dots)$

$$\theta_K^{(j)} := \theta_K^{(j)} - \alpha \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) x_K^{(i)} \quad (\text{for } K = 0)$$

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

Collaborative Filtering

- Difference here & previous example:
  → We know $\theta^{(j)}$ from the user, and we try to predict $x_1, x_2, \dots$

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

Looking at Love last $(\theta^{(1)})^T x^{(1)} \approx 5$ $(\theta^{(3)})^T x^{(1)} \approx 0$

$(\theta^{(2)})^T x^{(1)} \approx 5$ $(\theta^{(4)})^T x^{(1)} \approx 0$

$$\rightarrow x^{(1)} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

- Given $\theta^{(1)}, \dots, \theta^{(n_u)}$, to learn $x^{(1)}, \dots, x^{(n_m)}$; [Optimization]

$J(x^{(1)}, \dots, x^{(n_m)})$

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

→ Guess θ → Construct X → Better θ → Better X → so on....

→ Each user rates multiple movies, and each movie is rated by multiple users

→ Collaborative filtering: With large amounts of users, this algorithm basically makes all these users collaborate for better movie ratings for everyone. Every user is helping the algorithm a little bit by learning features → Better movie predictions for everyone.

- Combining both for the optimization objective:
Minimizing $x^{(1)}, \ldots, x^{(n_m)}$ and $\theta^{(1)}, \ldots, \theta^{(n_u)}$ simultaneously:

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

Instead of going from $\theta \to X \to \theta \to X \to$ so on....

→ $x \in \mathbb{R}^n$ no $x_0 = 1$ $\quad$ $\theta \in \mathbb{R}^n$ no θ_0

- Algorithm

1) Initialize $x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)}$ to small random values to ensure the algorithm learns features $x^{(1)}, \ldots, x^{(n_m)}$ that are different
2) Minimize $J(x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)})$ using gradient descent (or an advanced optimization algorithm

$$\frac{\partial}{\partial x_k^{(i)}} J(x^{(1)}, \ldots, x^{(n_m)}) \quad \& \quad \frac{\partial}{\partial \theta_k^{(i)}} J(\theta^{(1)}, \ldots, \theta^{(n_u)})$$

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

$$(\theta^{(j)})^T (x^{(i)})$$

Low rank matrix factorization

From previous eg $Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix}$ Predicted Ratings:

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

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

Predicted $\approx X\Theta^T$

- Eg 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, ...

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 $\|x^{(i)} - x^{(j)}\|$

- Mean Normalization:

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \leftarrow 2.5 \\ 5 & ? & ? & 0 & ? \leftarrow 2.5 \\ ? & 4 & 0 & ? & ? \leftarrow 2 \\ 0 & 0 & 5 & 4 & ? \leftarrow 2.25 \\ 0 & 0 & 5 & 0 & ? \leftarrow 1.25 \end{bmatrix} \quad \mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix} \rightarrow Y = \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.5 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

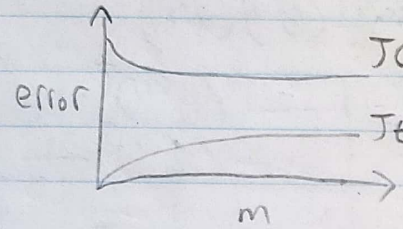
users

predict: $(\theta^{(j)})^T(x^{(i)}) + \mu_i$

learn $\theta^{(j)}, x^{(i)}$

Week 10 - Gradient Descent with Large Datasets

- How can you tell if using all of the data is likely to perform much better than using a small subset of the data (e.g. m=1000)?
  → Plot a learning curve for a range of values of m and verify that the algorithm has high variance when m is small



If high bias, adding more data won't help.

Stochastic Gradient Descent

- The original gradient descent method I know is called Batch gradient descent e.g. Repeat algo for every single m (# of examples)

Batch Gradient Descent

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

→ Repeat {

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

$\frac{\partial}{\partial \theta_j} J_{train}(\theta)$

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

}

→ Adjusts θ after passing through all examples

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 $i = 1, \ldots, m$ {
   $\theta_j := \theta_j - \alpha \underbrace{(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}}$
   (for $j = 0, \ldots, n$)
   }
   } 1-10x

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

→ Basically adjust θ for every example

Mini-Batch Gradient Descent

- Batch gradient descent: Uses 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: 2-100 typically

e.g. b=10

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

$$i := i+10$$

More formally: if 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 $j = 0, \ldots, n$)
  }
}

- Could be even faster than SGD if vectorized due to parallel processing
- Disadvantage: Need a good b value, so we have to fiddle around with it

Stochastic Gradient Descent Convergence

- Batch gradient descent:
  → Plot $J_{train}(\theta)$ as a function of the number of iterations of gradient descent
  → $J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

Stochastic gradient descent:

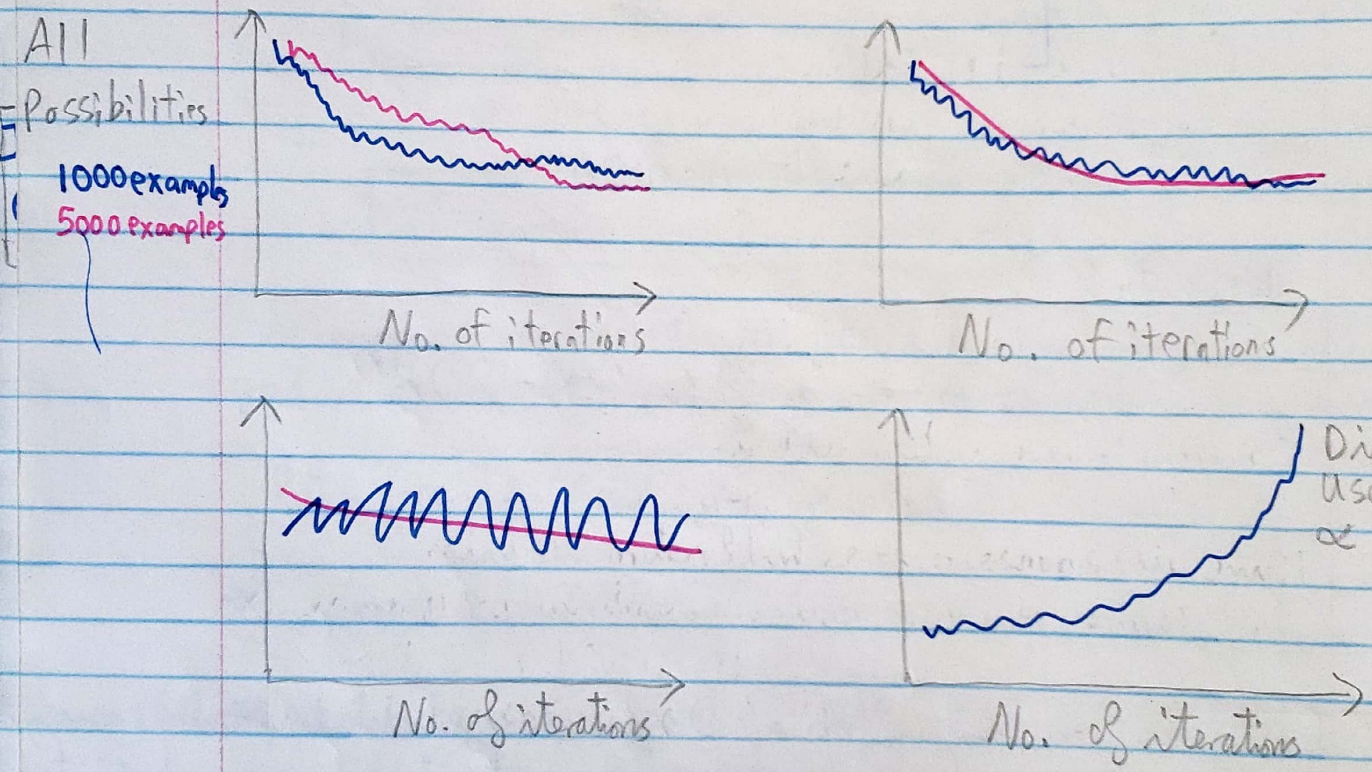
→ $\text{cost}(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$

→ During learning, compute $\text{cost}(\theta, (x^{(i)}, y^{(i)}))$ before updating θ using $(x^{(i)}, y^{(i)})$

→ Every 1000 iterations (say), plot $\text{cost}(\theta, (x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples processed by algorithm

N.B. Doesn't always converge to global minimum, but instead oscillates around it

All Possibilities



→ Learning rate α is typically held constant. Can slowly decrease α over time if we want θ to converge. $\left(\text{E.g } \alpha = \frac{\text{const 1}}{\text{iteration \# + const 2}}\right)$

Online Learning

Example

- Shipping service website where user comes, specifies origin and destination, you offer to ship their package for some asking price, and users sometimes choose to use your shipping service ($y=1$) sometimes not ($y=0$).
- Features x capture properties of user, of origin/destination and asking price. We want to learn $p(y=1 \mid x;\theta)$ to optimize price.

logistic regression

Repeat forever {
  Get (x,y) corresponding to user.
  Update θ using (x,y):
    $\theta_j := \theta_j - \alpha (h_\theta(x) - y) \cdot x_j \quad (j=0,\dots,n)$
}

→ Can adapt to changing user preference

- → Product search (learning to search)
- → User searches for "Android phone 1080p camera"
- → Have 100 phones in store. Will return 10 results.
- → x = features of phone, how many words in user query match name of phone, how many words in query match description
- → $y=1$ if user clicks on link. $y=0$ otherwise
- → Learn $p(y=1 \mid x;\theta)$ → Predicted CTR (Click through rate)
- → Use to show user the 10 phones they're most likely to click on.

- Other examples: Choosing special offers to show users; customized selection of news articles; product recommendations; ....

Map-reduce and data parallelism

- Batch gradient descent: $\theta_j := \theta_j - \alpha \frac{1}{400} \sum_{i=1}^{400} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$

  → Machine 1: Use $(x^{(1)}, y^{(1)}), \ldots, (x^{(100)}, y^{(100)})$.
  $Temp_j^{(1)} = \sum_{i=1}^{100} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$

  → Machine 2: Use $(x^{(101)}, y^{(101)}), \ldots, (x^{(200)}, y^{(200)})$
  $Temp_j^{(2)} = \sum_{i=101}^{200} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$

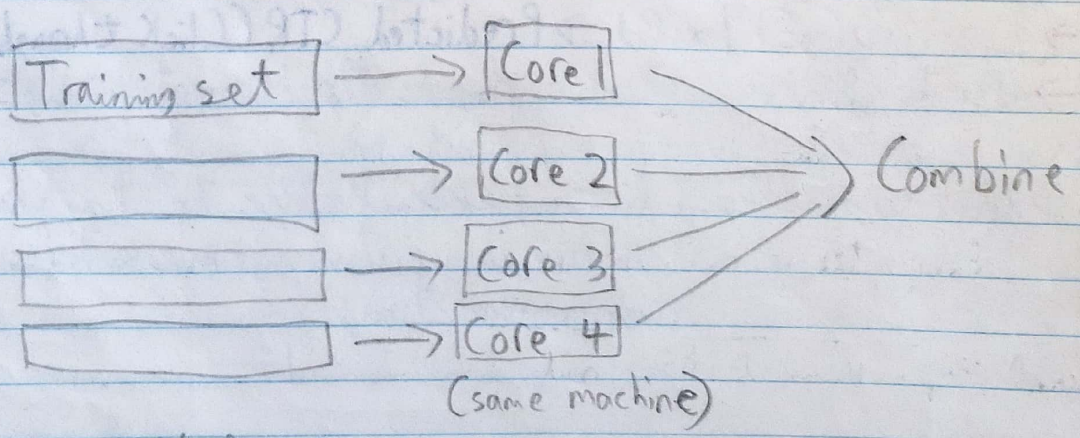
  → Machine 3: 201 → 300

  → Machine 4: 301 → 400

  Combine: $\theta_j := \theta_j - \alpha \frac{1}{400} (temp_j^{(1)} + temp_j^{(2)} + temp_j^{(3)} + temp_j^{(4)})$
  $j = 0, \ldots, n$

- Many learning algorithms can be expressed as computing sums of functions over the training set.

- Multi-core machines



Adv: Don't have to worry about network latency

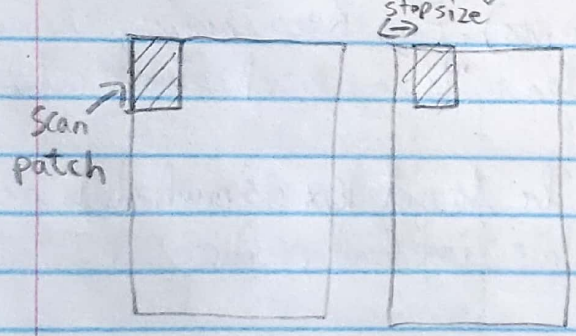
Week 11 - Photo OCR

- OCR : Optical Character Recognition

- Pipeline (Generic)
  1. Text Detection from image
  2. Character Segmentation
  3. Character Classification

- Pipeline advantage : Allows work to be divided between Engineers

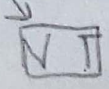
Example
- Pedestrian Detection in an image
  → Train supervised model with m pictures, where y=1 indicates pedestrian and y=0 indicates not pedestrian
  → Test on new image using Sliding Window Technique

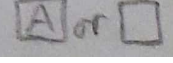


and so on scanning the entire image. Increase block size, scan again & so on

- Similar technique for text detection
  → But as text is continuous, once we detect text in a window, we merge the close by (e.g. within 2 blocks) detections together in a technique called "expansions"
  → From the various blocks of detections we have, we discard the ones with a height-length ratio that don't represent how text should flow from left to right e.g. ▯ rather than ▭

- Similarly for character segmentation, train supervised model where image patches that represent gaps between letters are y=1, otherwise y=0





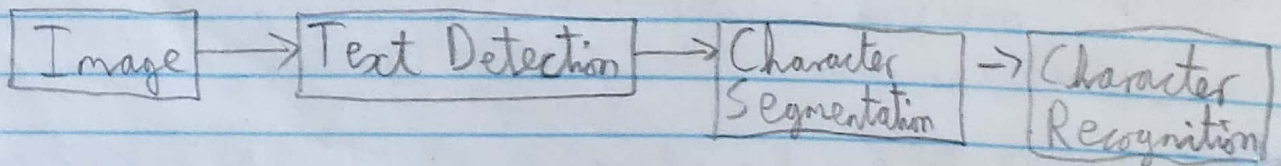
  → Then sliding window technique on test image

Artificial Data Synthesis

- By using synthetic data that work like real data e.g. Character segmentation/classification, we have unlimited supply of real data
- Can synthesize data by introducing distortions e.g. for images, speech recog and having much more data to use for training
- Distortion introduced should be representation of the type of noise in the test set e.g. Audio: Background noise, bad cellphone connection
  → Usually does not help to add purely random/meaningless noise to your data.
- 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?" Calculate time and resources required
     - Artificial data synthesis
     - Collect/label it yourself
     - "Crowd Source"

Ceiling Analysis

- Estimating the errors due to each component (ceiling analysis)

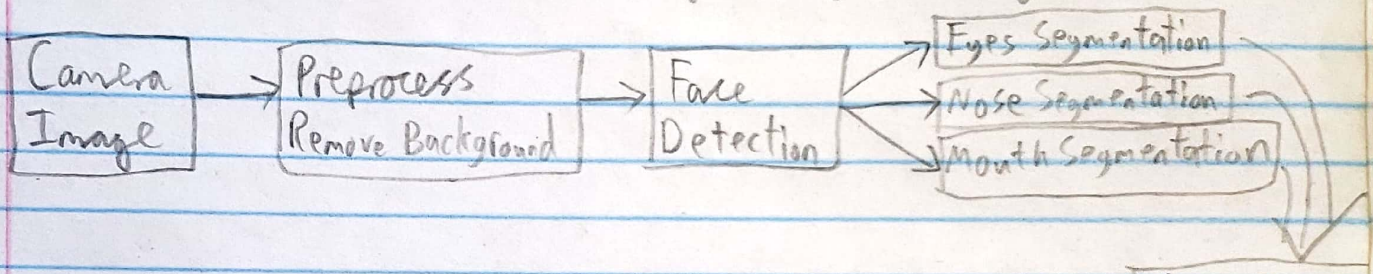


Which part of the pipeline should you spend the most time trying to improve?

Example

Component	Accuracy (overall system)	
Overall System	72%	
With 100% Text Detection Acc	89%	↓ 17% ← Most Worthwhile to improve
" " " Charac Segmen	90%	↓ 1%
" " " " Charac Recog	100%	↓ 10%

- Another example: Face Recognition from images



Component	Accuracy (overall system)		
Overall System	85%	0.1%	
Preprocess	85.1%	5.9%	← Face Detection most worthwhile to work on and improve
Face detection	91%	4%	
Eyes Segmentation	95%	1%	
Nose Segmentation	96%	1%	
Mouth Segmentation	97%	3%	
Logistic Regression	100%		