ML - Andrew NG Octave

Supervised learning: ("right ans" given.)

→ Regression. (continuous valued output)
→ classification (discrete valued output) (0 or 1) or more.

→ support vector machine (often with lot no. of features)

Unsupervised learning: (Not labeled)

→ clustering algo.
→ cocktail party algo.

cost func^n

$h(x) = \theta_0 + \theta(x)$ ← parameter.

↓ how to choose Parameter.

(optimize the Pots)

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

$J(\theta_0, \theta_1)$

↓ cost func^n.
or sqd error func^n.

gradient descent algo to minimize cost func^n $J(\theta_0, \theta_1)$

↓

repeat until convergence {

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

}

(for $j=0$ & $j=1$)

↳ learning rate.

$\alpha \uparrow \Rightarrow$ big steps

i.e

$$a_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) ;$$

$$a_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) ;$$

$$\theta_0 = a_0 ;$$

$$\theta_1 = a_1 ;$$

⇒ simultaneous update.

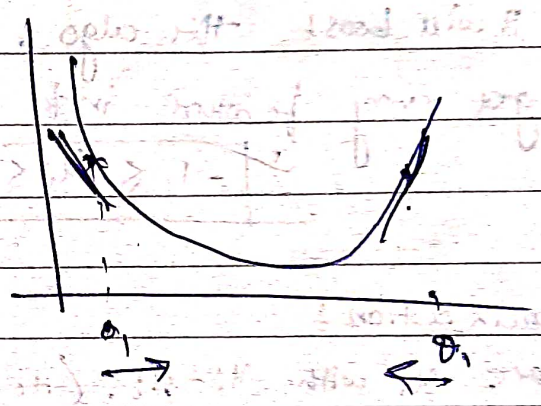
V.V. imp

$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$

Slope ⇒ +ve or -ve.

$\theta_1 = \theta_1 - \alpha (+ve)$

⇒ (α always +ve)



$\theta_1 = \theta_1 - \alpha (-ve)$

$= \theta_1 + \alpha ( )$

"Batch" gradient descent.

LR (with Multi var) (Multivariate)

$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

$x_0^{(i)} = 1$

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

$\theta^T = [\theta_0, \theta_1 \cdots \theta_n]$

$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$ $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$

gradient descent algo for this ($n \geq 1$)

Repeat {

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

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

$(j = 0, \cdots n)$

}

Feature scaling :

(features are in same scale)

It will boost the algo.

get every feature into approximately a $-1 \leq x_i \leq 1$ Range.

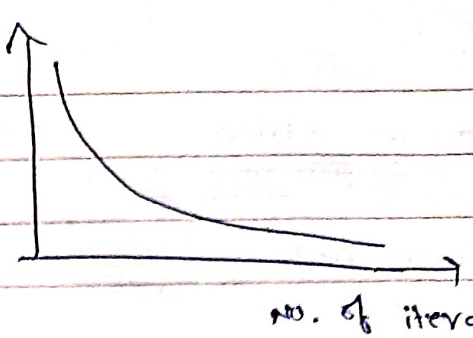
Mean normalization :

Replace x_i with $x_i - \mu_i$ (to make features have approx '0' mean)

μ_i → avg.

$x_i \leftarrow \frac{x_i - \mu_i}{s_i}$ → avg. of x

s_i → range (max - min)



no. of iterations ↑ $J(\theta)$ ↓

⇒ Gradient descent algo working Properly.

Automatic Conv. test :-

Polynomial regression :- $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \cdots$

Normal Eqn :- solve θ analytical.

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

$$\frac{\partial}{\partial \theta_j} J(\theta) = \cdots = 0. \quad \text{(for every } j\text{)}$$

⇒ solve for $\theta_0, \theta_1, \theta_2, \cdots \theta_n$

Normal eqn

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

This θ minimizes the cost fun^n.

(feature scaling is not required in Normal Eqn method)

$(X^T X)^{-1}$ is non-invertible :-

coz :- (i) Redundant feature (i.e. LD)

(ii) Too many features (i.e. $m \leq n$)

(regularization)

Logistic Regression :

Classification prob :

Binary: $y = \{0, 1\}$ 0 = -ve class
1 = +ve class

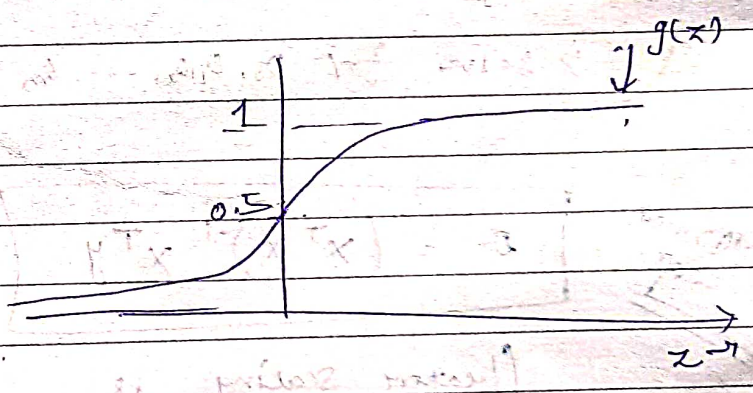
multiclass C.P : $y = \{0, 1, 2, 3\}$

logistic Regression : $0 \leq h_\theta(x) \leq 1$
(classification algo)

hypothesis Representation:

$h_\theta(x) = g(\theta^T x)$ || $g(z) = \frac{1}{1+e^{-z}}$
(sigmoid / Logistic func^n)

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



$h_\theta(x) = P(y=1 \mid x ; \theta)$
Prob that $y=1$ (some), given x, parameterized by θ

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

decision boundary :

→ $y = 1$ if $h_\theta(x) \geq 0.5$ | $\frac{1}{1+e^{-\theta^T x}} \geq \frac{1}{2}$

$\theta^T x \geq 0$ ←

→ $y = 0$ if $h_\theta(x) < 0.5$

$\theta^T x < 0$

Linear DB : $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2) \rightarrow \theta^T x$

Non-linear decision boundary :

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

(decision boundary is a property of the hypothesis set, not a property of the training set.)

Cost func^n :

(Finding θ)

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

$$Cost(h_\theta(x), y) = \frac{1}{2}(h_\theta(x) - y)^2 \rightarrow \text{Linear Regression.}$$

Logistic R : $h_\theta(x)$ is non-convex fer^r.

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

Simplified C.f :

$$Cost(h_\theta(x), y) = -y \log(h_\theta(x)) - (1-y) \log(1 - h_\theta(x))$$

(deriving Mam Likelihood)

G.D. $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{cost}$

Repeat {

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

}

$$\frac{\partial}{\partial \theta_j} J(\theta) = \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)}$$ || same as Linear,

Advanced optimization :

Optimization algos :
- → Gradient descent
- → conjugate gradient
- → BFGS
- → L-BFGS

Multiclass classification : one vs all : (or one vs rest algo)

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

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

Regularization

Prob of overfitting : (or high variance)

(underfit or high bias)

Addressing overfitting : e no. of features.

izatior.

educe mag/val of parameters θ_j)

cost funcⁿ :-

Small values of parameters $\theta_0, \theta_1, \cdots \theta_n$

⇒ simpler hypothesis.

⇒ Less prone to overfitting.

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

Regularization term.

↳ Regularization parameter.

If λ is too large,

⇒ $h_\theta(x) = \theta_0$ (as $\theta_1 \approx \theta_2 \approx \cdots \theta_n \approx 0$)

(cause of underfitting)

Regularized Linear Regression :-

Repeat {

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

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

$(j = 1, 2, 3, \cdots)$

}

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

$\left(1 - \alpha\frac{\lambda}{m}\right) < 1$

grad descent.

Normal Eqn →

$$X = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}_{m \times (n+1)} \qquad y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m$$

$$\min_{\theta} J(\theta)$$

$$\Rightarrow \theta = \left(X^T X + \lambda \underbrace{\begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \end{bmatrix}}_{(n+1) \times (n+1)} \right)^{-1} X^T y$$

Regularized Logistic Regression :

Same as linear. but $h_\theta(x) = \frac{1}{e^{-\theta^T x} + 1}$

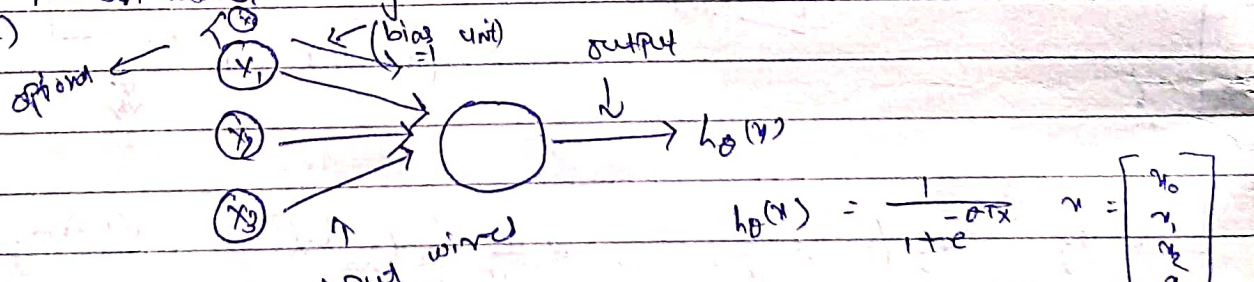
Neural Networks Representation

Non - Linear Hypothesis :

model Representation :

In a neuron
- dendrite - input wires.
- Axon - output wires.

Simple Neuron model : Logistic Unit : (single)



$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} \qquad x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

(Sigmoid (logistic) activation func^n)

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

↑ weights or parameters.

In Neural Network : (multi layers)

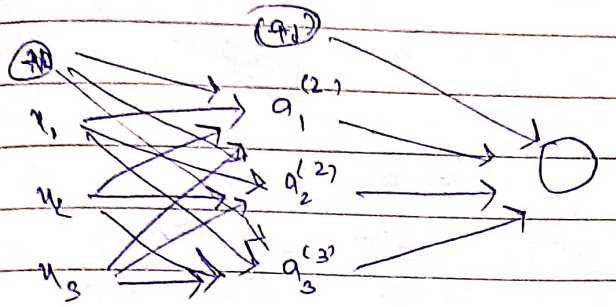
layer 1 : Input layer : last layer : output layer.

(Intermediate layers : hidden layer.)

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

$\Theta^{(j)}$ = Matrix of weights controlling func mapping from layer j to j+1.

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



$$a_1^{(2)} = g\left(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3\right) \rightarrow z_1^{(2)}$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3\right) \rightarrow z_2^{(2)}$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3\right) \rightarrow z_3^{2}$$

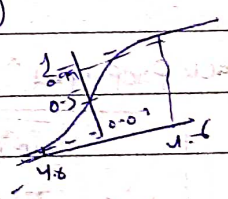
$$h_\Theta(x) = g\left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right)$$

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

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

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

(forward propagation)



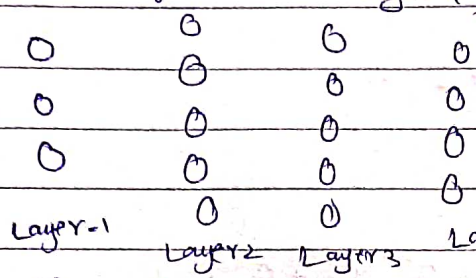
(multiclass classification)

Neural Network Learning

Cost func^n: (application in classification Problems)

(L = No. of layers) $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \dots (x^{(m)}, y^{(m)})\}$

(S_L = No. of units in layer L)



$S_1 = 3$

$S_2 = 5 = S_3$

$S_4 = 4$

Binary classifi^n:

$y = 0$ or 1

(1 output unit)

$\Rightarrow S_L = 1 \Rightarrow K = 1$

$h_\theta(x) \in \mathbb{R}$

Multiclass

$y = \mathbb{R}^K$ eg:- $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}\begin{bmatrix}0\\1\\0\\0\end{bmatrix}\begin{bmatrix}0\\0\\1\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

K output units

$h_\theta(x) \in \mathbb{R}^K$ // $(K \geq 3)$

$S_L = K$

$h_\theta(x) \in \mathbb{R}^K$ $(h_\theta(x))_i = i^{th}$ output.

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

$$- \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}} \left(\theta_{ji}^{(l)}\right)^2$$

Back Propagat^n algo :-

(minimize the cost func^n)

$\min J(\theta)$

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

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

(L=4 in the given eqn)

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

$\downarrow$

$(h_\theta(x))_j$

$$\delta^{(3)} = (\theta^{(2)})^T \delta^{(4)} .* g'(z^{(3)}) \rightarrow a^{(3)} .* (1 - a^{(3)})$$

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

NO $\delta^{(1)}$
(as input layer)

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

(ignoring λ; if the regularization term $\lambda = 0$)

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

Backprop algorithm
(to compute derivative of min cost func^n)

Set $\Delta_{ij}^{(l)} = 0$ (for all i, j, l)

For $i = 1$ to m

- 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 $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$
- $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

or $\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

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

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

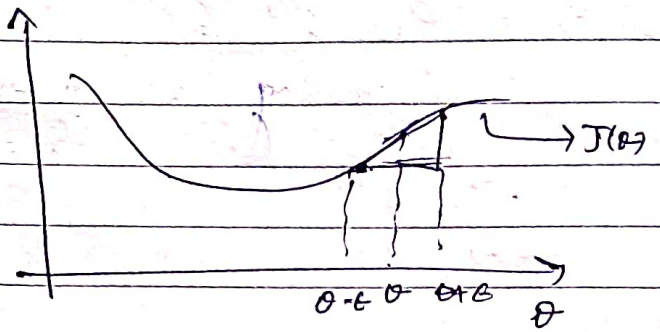
$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = D_{ij}^{(l)}$$

Backpropagation Intuition :

$$\delta_1^{(4)} = y^{(i)} - a_1^{(4)}$$

Gradient Checking :

Numerical estimation of gradients :



$$\frac{d}{d\theta}[J(\theta)] \approx \frac{J(\theta+\epsilon) - J(\theta-\epsilon)}{2\epsilon}$$

Random initialization :

(solves symmetric breaking issue)

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

Advice for Applying M.L.:

Machine Learning diagnostic:

↓

a test

(takes time)

Evaluating hypothesis:

→ Dataset: divide into training set & test set

(70%) (around) (30%)

Learn parameter θ from training set.

(Min training error $J(\theta)$)

& (Cross validation set. or validation set) (5%)

Compute test set error

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

test set error (for linear regression)

Similar for logistic regression: (Classification Prob)

test set error using misclassification error (0/1 misclassification error)

Model selection:

training / validation / test set

Train / validation / Test error:

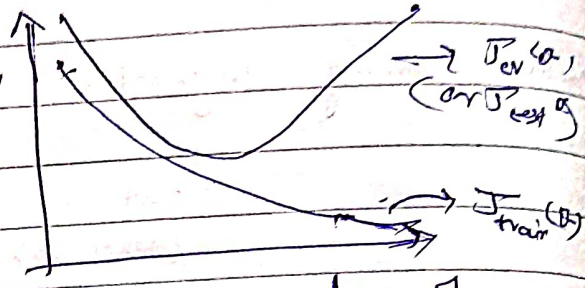
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

similarly $J_{cv}(\theta)$, m_{cv}, x_{cv}, y_{cv}

& $J_{test}(\theta)$, m_{test}, x_{test}, y_{test}.

J_{train} - training error.
J_{cv} = cross validation error.

Diagnosing Bias vs Variance :



Left of $J_{cv}(\theta)$
$\Rightarrow$ high bias (underfit) Prob
right $\Rightarrow$ high var (overfit) Prob.

Bias (underfit) $\Rightarrow$ $J_{train}(\theta)$ $\approx$ $J_{cv}(\theta)$
(both high)

Variance (overfit) $\Rightarrow$ $J_{train}(\theta)$ is low but $J_{cv}(\theta)$ is high.
($J_{cv}(\theta) \gg J_{train}(\theta)$)

Regularization & Bias/Variance :
(help prevent overfitting)

Reg parameter λ : very high $\Rightarrow$ underfit.
intermediate $\Rightarrow$ good fit
very low $\Rightarrow$ overfit.

model

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

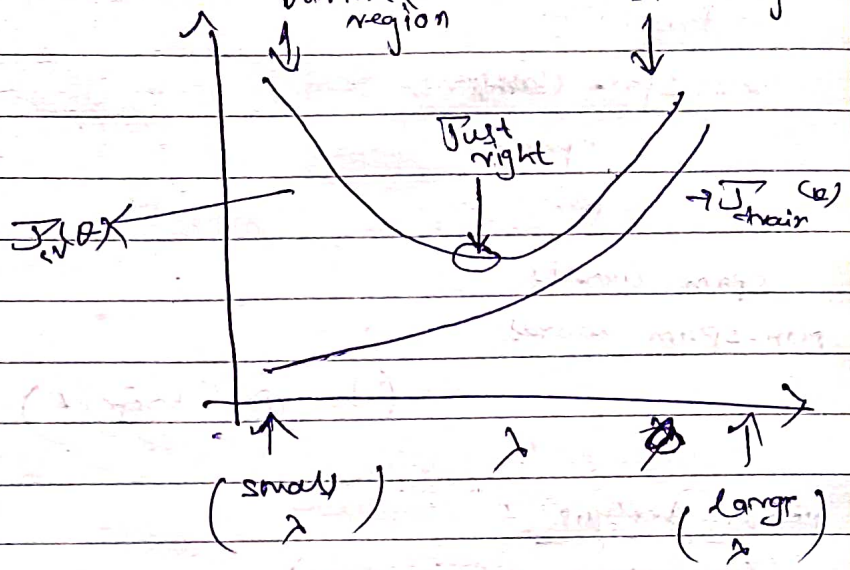
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

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

$J_{cv}(\theta)$
$J_{test}(\theta)$
→ without regularization term.

Try diff λ = 0, 0.01, 0.02, ..., 10 → Put these λ in $J_{train}(\theta)$ ⇒ get value of θ for each ⇒ Put that θ in $J_{CV}(\theta)$ ⇒ get for which $J_{CV}(\theta)$ is min ⇒ $J_{test}(\theta)$

Bias/Var as a func^n of reg. parameter λ:



Learning Curves

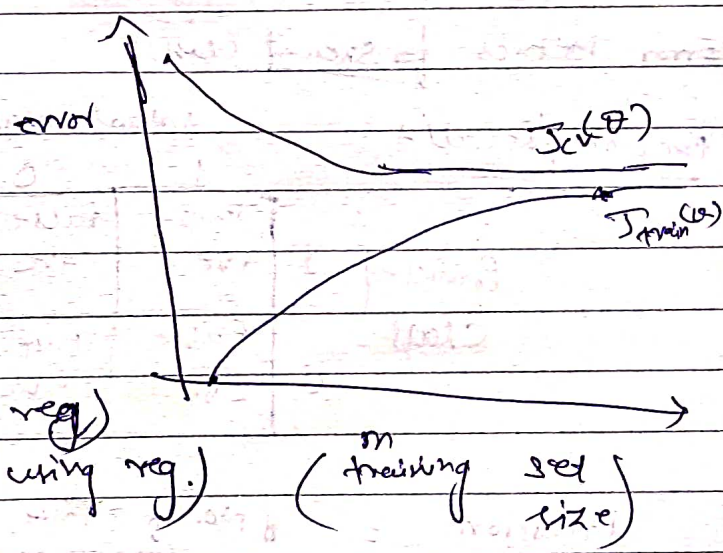
$J_{train}(\theta)$ =

$J_{CV}(\theta)$ =

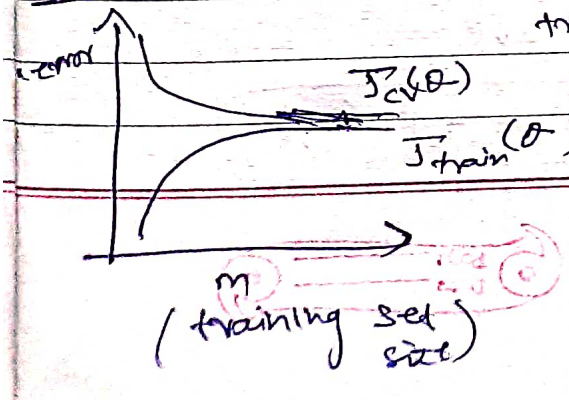
(for quadratic model)

$m = 1, 2, 3 \Rightarrow J_{train}(\theta) = 0$ (if not using reg)

(slightly > 0 if using reg.)

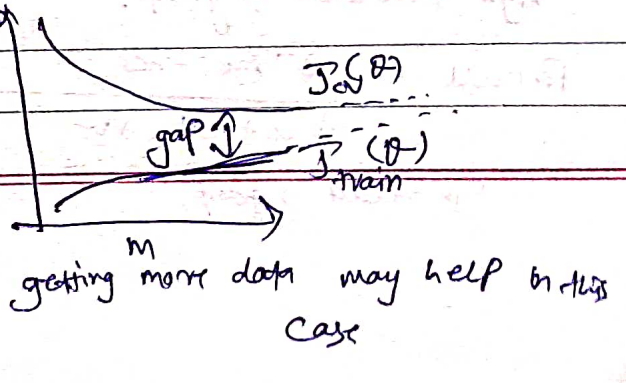


high bias :



getting more training data won't help if algo is suffering from high bias

high variance :



getting more data may help in this case

Small N.N → fewer parameters ⇒ Underfitting.
Large N.N → More " ⇒ Overfitting.

Machine Learning System Design:

Prioritizing:

ex: Spam classifier:

Spam - 1

Nonspam - 0

Spam words:

Non-spam words:

("honeypot" Project)

Error Analysis:

(Pre-mature optimization)

Error Metrics for Skewed class:

Precision / Recall:

Actual class

Predicted class	1	0
1	True +ve	False +ve
0	False -ve	True -ve

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

$$\text{Precision} = \frac{\#\text{No. of True +ve}}{\#\text{Predicted +ve}} = \frac{\#\text{True +ve}}{\#\text{True +ve} + \#\text{False +ve}}$$

high precision ⇒ good.

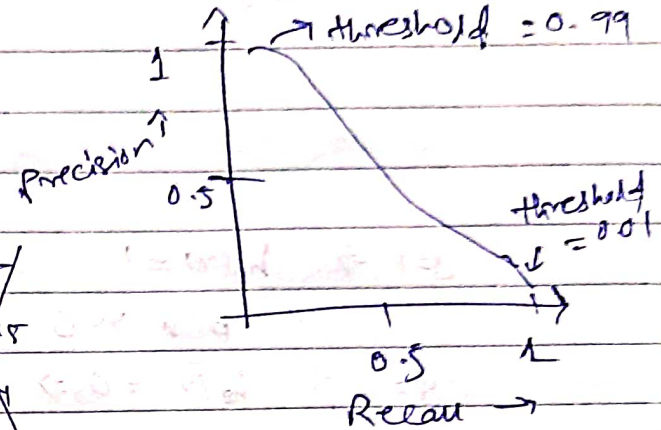
$$\text{Recall} = \frac{\#\text{True +ves}}{\#\text{actual +ve}} = \frac{\#\text{True +ve}}{\#\text{True +ve} + \#\text{False -ve}}$$

high recall ⇒ good.

Trading off Precision & Recall :

~~Higher~~ Precision, ~~lower~~ recall

generally, Predict 1 if $h_\theta(x) \geq$ threshold,



F1 score (f score)

	Precision (P)	Recall (R)	Avg
Algo 1	0.5	0.4	0.45
2	0.7	0.1	0.4
✗ 3	0.02	1.0	0.51

(x)

Not a good way.

F. score

1 → 0.444

2 → 0.175

3 → 0.0392

$$\text{F1 score} = 2\frac{PR}{P+R}$$

$(\in [0,1])$

Data for ML :

→ $J_{train}(\theta)$ will be small

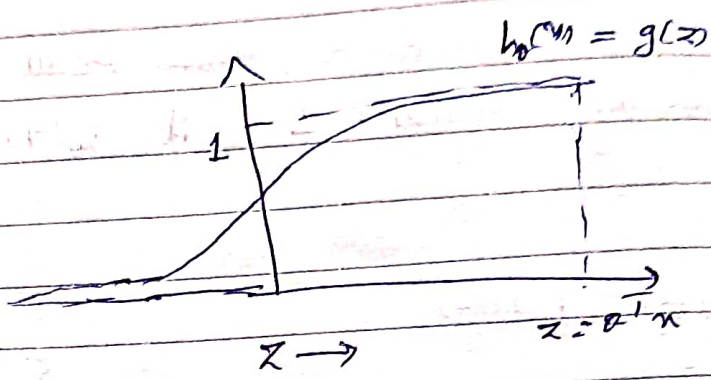
using a v. large dataset → $J_{train}(\theta) \approx J_{test}(\theta)$ (some cases)

⇒ $J_{test}(\theta)$ will also be small.

Support Vector Machine : (SVM)

optimization objective :

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



$y=1 \Rightarrow h_\theta(x) = 1$

$\Rightarrow \theta^T x \gg 0$

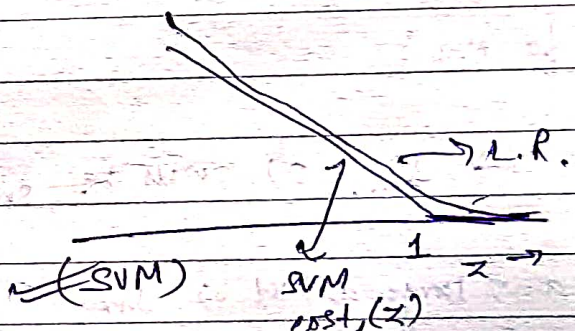
$y=0 \Rightarrow h_\theta(x) = 0 \Rightarrow \theta^T x \ll 0$

cost func = $-\left(y \log h_\theta(x) + (1-y) \log h_\theta(x)\right)$

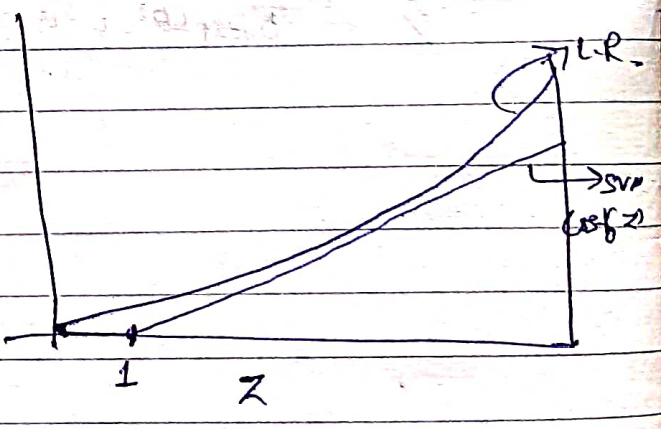
$= -y \log \frac{1}{1+e^{-\theta^T x}} * -(1-y) \log \frac{1}{1+e^{-\theta^T x}}$

if $y = 1$ (using $\theta^T x \gg 0$)

$\Rightarrow$ c.f $= -y \log \frac{1}{1+e^{-\theta^T x}}$



if $y=0$ (using $\theta^T x \ll 0$)



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

$= A + \lambda B$

L.R : $A + \lambda B$

SVM : $CA + B$; $C = 1/\lambda$

then L.R & SVM give same value.

hypothesis : $h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$

Large Margin Intuition :-

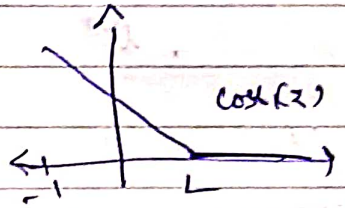
(Large Margin classifier)

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

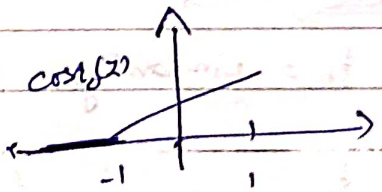
very large C $\Rightarrow$ to minimize, we have to put this 0.

then

when $y^{(i)} = 1$, $\theta^T x^{(i)} \geq 1$



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



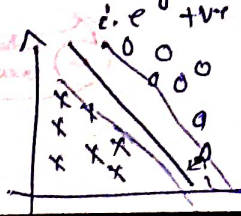
$\therefore \min_\theta C \times 0 + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2 = \min_\theta \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$

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

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

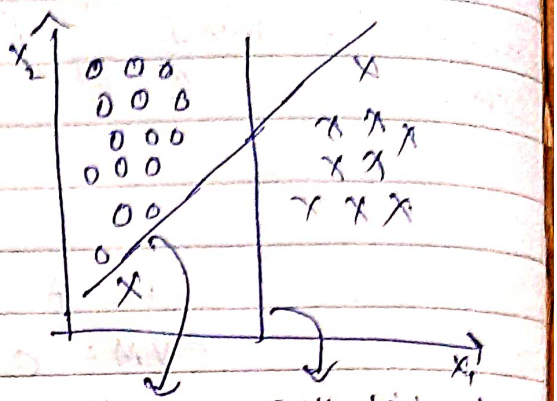
SVM Decision Boundary :- (Linearly separable case)

Margin of SVM



i.e +ve & -ve examples are separable by a st. line,

Large Margin classifier in presence of outliers:



if C is too large

SVM decision boundary (if C not too large)

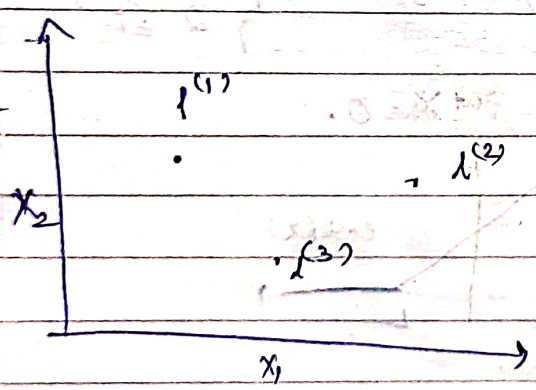
$(C \sim 1/\lambda)$

Math behind Large Margin classification:

Kernel:

non linear decision boundary:

$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \cdots$ $\quad h_\theta(x) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 f_1 + \cdots \geq 0 \\ 0 & \text{otherwise} \end{cases}$



$l^{(1)}, l^{(2)}, l^{(3)}$ → landmarks.

Given x: $f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$ ← (euclidean dist)

$f_2 = \text{''} \quad (x, l^{(2)}) = \exp\left(-\frac{\|x - l^{(2)}\|}{2\sigma^2}\right)$

$f_3 = \text{''} \quad (x, l^{(3)}) = \exp\left(-\frac{\|x - l^{(3)}\|}{2\sigma^2}\right)$

similarity is kernel fun^n. || $k(x, l^{(i)})$

(Gaussian kernels)

is used here.

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

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

If x is far from $l^{(1)} \Rightarrow f_1 = \exp\left(-\frac{(\text{large no.})^2}{2\sigma^2}\right)$

≈ 0

for practice

Landmarks will be at the same location of the training examples.

Parameter $C \simeq \frac{1}{\lambda}$: large $C \Rightarrow$ lower bias, (small λ) high variance.

small $C \Rightarrow$ lower var (large λ) high bias

σ^2 : Large $\Rightarrow$ features f_i vary more smoothly. (higher bias, low var)

small $\rightarrow$ less smoothly. (lower bias, higher var)

SVM with kernel's algo $\rightarrow$ watch again (refer to any internet stuff)

Most used kernels in SVM :
Linear Kernel (no kernel),
Gaussian Kernel.

Using A SVM

→ SVM software Packages :- liblinear, libsvm etc.
to solve parameter θ.

→ Choice of Parameter C
→ Choice of Kernel (sim fun^n)

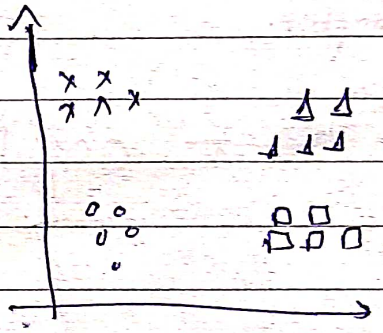
Gaussian Kernel : need to choose σ^2.

Mercer's Theorem

This is a tech cond^n that should be satisfied by kernels.

Multiclass classification :-

Use built in functionality
or
Use one-vs-all method.



Logistic Regression vs SVMs :-

n = no. of features , m = no. of training ex.

→ n is large : LR or SVM w/o kernel.
(n >> m) (linear kernel)

→ n is small, m is intermediate : SVM with gaussian kernel

→ " , m is large : create / add more features, then LR or SVM w/o kernel.

unsupervised learning

Clustering (in Labeled data)

K-Means Algo :

Input : K - (no. of clusters)
training set $\{x^{(1)}, x^{(2)} \dots x^{(m)}\}$
$x^{(i)} \in \mathbb{R}^n$ (drop $x_0 = 1$ convention)

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

Repeat {

cluster assignment step [for i = 1 to m
$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$
$\min \|x^{(i)} - \mu_k\|^2$.

move centroid step [for k = 1 to K
μ_k = avg (mean) of points assigned to cluster k.

}

K-means for non-separated clusters :

Optimization objective :

$c^{(i)}$ = Index of cluster to which $x^{(i)}$ is assigned.
$(1, 2, \dots K)$
μ_k = cluster centroid k ($\mu_k \in \mathbb{R}^n$)
$\mu_{c^{(i)}}$ = cluster centroid of the cluster to which $x^{(i)}$ has been assigned.

Optimization Obj :

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

$\min_{c^{(1)} \dots c^{(m)}, \mu_1 \dots \mu_K} J$ (Distortion)
(cost func^n).

→ Cluster assignment step minimizes $J(\dots)$ w.r.t. $c^{(1)}, c^{(2)} \dots c^{(m)}$ (holding $\mu_1, \mu_2 \dots \mu_K$ fixed)
→ move centroid step minimizes $J(\dots)$ w.r.t. $\mu_1 \dots \mu_K$.

Random Initialization :

Randomly initialize cluster centroids :

(initialize k-means multiple times)

for i = 1 to 100 {
    Randomly initialize k-means.
    Run k-means. get $c^{(1)}, \ldots c^{(m)}, \mu_1, \ldots \mu_k$
    compute cost func^n (distortion) $J(\ldots)$
}

Pick lowest $J(\ldots)$.

(It works good for small no. of clusters:
(eg : k = 2 to 10)

Choosing No. of Clusters :

Elbow Method:

Dimensionality Reduction :

Application :

Data Compression :

Data visualization;

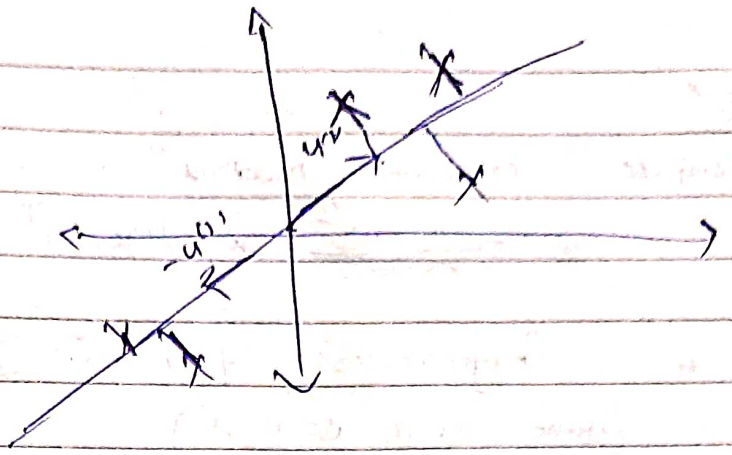
Principal Component Analysis :

(PCA)

(algo for dimensionality reduction)

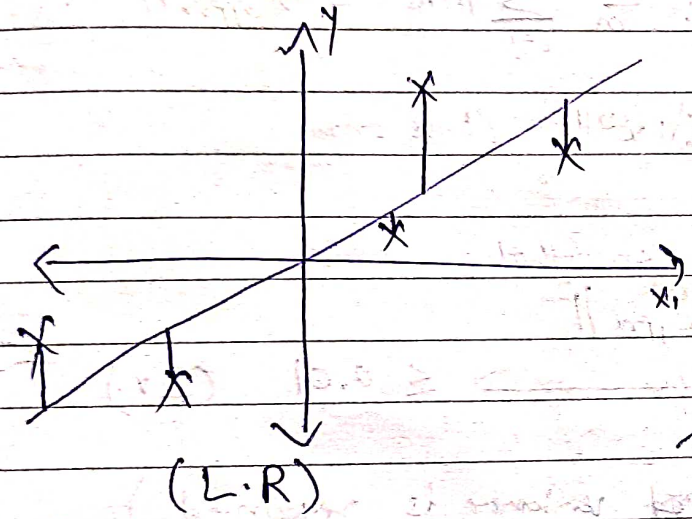
Projection error → Minimize.

Reduce from 2-D to 1-D : By finding an dir^n (a vector) onto which to project the data so as to minimize the projection error.



Reduce n-D to k-D $\Rightarrow$ find k-vectors : $u^{(1)}, u^{(2)}, \ldots u^{(k)}$

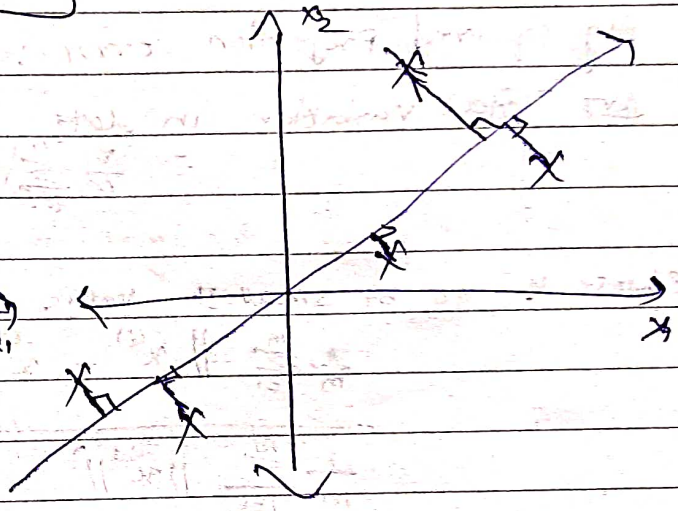
PCA is not Linear Regression.



(L.R)

(Min vertical dist)

We want to predict the special var / distinguish var y using x_1 in L.R.



(PCA)

(Min projection)

there is no sp var distinguish var in PCA

$x_1, x_2, \ldots x_n$ → are features (treated equally)

Data Preprocessing (Before PCA) :

(feature scaling / mean normalization)

Mean = 0

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

Replace $x_j^{(i)}$ with $x_j - \mu_j$ / in data)

If x_j have diff scale :

$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$ → (max-min range)

Algo (PCA)

→ Compute "covariance matrix" :

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

covariance matrix satisfies "Symmetric Positive Definite."

→ " "eigenvectors" of matrix Σ

(take 1st k vectors for $u^{(1)} \ldots u^{(k)}$)

Choosing No. of Principal Components (k)

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

~~Step~~ Total variation in data

$$= \frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} \| \quad \text{(dist origin)}$$

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)

Reconstruction from compressed Representation:

$$z = U_{reduce}^{T} x$$

$$\therefore x_{approx}^{(i)} = \underset{n \times k}{U_{reduce}} \; \underset{k \times 1}{z^{(i)}}$$

$\mathbb{R}^n$

$\approx x$

Supervised Learning Speedup :

$(x^{(1)}, y^{(1)}), \ldots (x^{(m)}, y^{(m)})$ $\quad x^{(i)} \in R^{10,000}$

extract inputs,

Unlabeled dataset : $x^{(1)}, x^{(2)} \ldots x^{(m)} \in R^{10,000}$

↓

PCA

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

New training set : $(z^{(1)}, y^{(1)}), (z^{(2)}, y^{(2)}), \ldots (z^{(m)}, y^{(m)})$

$h_\theta(z) = \frac{1}{1+e^{-\theta^T z}}$ (for L.R).

~~Same $x \rightarrow z$ (for test set)~~

PCA

Mapping $x^{(i)} \Rightarrow z^{(i)}$ should be defined by running PCA only on the training set

find the mapping $x \rightarrow z$

the apply this mapping to $x_{test}^{(i)}$ & $x_{cv}^{(i)}$

→ Do Not Use PCA to Prevent Overfitting

→ " to Design of ML system

Anomaly Detection

Density estimation :

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

→ Is x_{test} anomalous ?

model - $p(x)$ || $p(x_{test}) < \epsilon \rightarrow$ flag anomaly

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

appln : fraud detection

ex:- Fraud Detection :-

→ $x^{(i)}$ = features of user i's activities.

→ Model $p(x)$ from data.

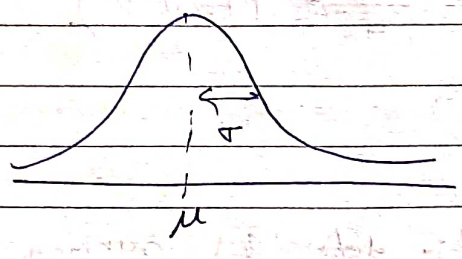
Identify unusual users by checking which have $p(x) < \epsilon$

ex:- Manufacturing

ex:- Monitoring computers in a data center.

Gaussian Distribution :-

(Normal)



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

Parameter estimation :-

Dataset : $\{x^{(1)}, \ldots, x^{(m)}\}$, $x^{(i)} \in R$

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

unknown.

finding μ, σ^2 is aim of Parameter estimation Problem.

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

algo

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

$x \in R^n$ ⇒ $x_i \sim N(\mu_i, \sigma_i^2)$

$$p(x) = p(x_1; \mu_1, \sigma_1^2) \cdots p(x_n; \mu_n, \sigma_n^2)$$

(events are closed to independent)

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

Anomaly Detection vs Supervised Learning :

A.D	S.L
→ Small no. of +ve examples. ($y=1$) Large " -ve " . ($y=0$) (Normal)	→ Large +ve & -ve examples.
eg : → Fraud detection. → Manufacturing → Monitoring machines in data center etc.	eg : → email spam classificⁿ. → weather Predictⁿ. → cancer classificatⁿ

#

Try to make the data Gaussian.

If not gaussian : $\log(x_1)$ → gaussian.
or $\log(x_2 + c)$
or $\log(\sqrt[k]{x_3})$

Error Analysis : (Min some features to create a new feature & examine)

Multivariate Gaussian Dist. :

$x \in R^n$

Don't model $p(x_1), p(x_2)$ --- etc separately.

Model $p(x)$ all in one go.

Parameters : $\mu \in R^n$

$\Sigma \in R^{n \times n}$ (Co-variance matrix)

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

$$\Sigma = \begin{matrix} \\ x_1 \\ x_2 \end{matrix} \begin{matrix} \begin{matrix} x_1 & x_2 \end{matrix} \\ \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \end{matrix}$$

means x_1 has var = 2

x_2 has var = 1

co-relation $(x_1, x_2) = 0$

Recommender System : (appl^n of ML)

Adv. formulation :

Product recommendation by Amazon, Netflix etc.

Content based recommendations :

eg : (predicting movie rating)

n_u = No. of users || n_m = no. of movies.

$r(i,j)$ = 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.

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

(for all j)

Optimization algo (min θ)

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

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

$$= J(\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)})$$

grad descent update :

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

(we don't regularize $\theta_0^{(j)}$)

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

regularization term only for $k \neq 0$

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

for linear R.

we have $\frac{1}{m}$ here

Collaborative filtering : (Andrew ng)

(uses feature learning)

opt. algo :

given $\theta^{(1)}, \dots, \theta^{(n_u)}$, to learn $x^{(i)}$:

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

to learn $x^{(1)}, \dots, x^{(n_m)}$:

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

so $\theta \to x \to \theta \to x \dots$

Collaborative filtering algo : (Improved)

Putting this & this together :

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

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

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

→ Initialize $x^{(1)}, \dots, x^{(n_m)}$, $\theta^{(1)}, \dots, \theta^{(n_u)}$ to small random values

→ Min $J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$

→ for a user with parameter θ & a movie with (learned) feature x, then predict a star rating of $\theta^T x$

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

Vectorization: Low Rank Matrix Factorization:

(LRMF)

Predicted ratings:

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

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

This has a Mathematical Property called: (Low Rank Matrix)

(Prop.)

(Rank deficient - If Matrix doesn't have the full rank)

#How to find movies j related to movie i?

$\Rightarrow$ $\| x^{(i)} - x^{(j)} \|$ = small (feature vectors of movies)

then movies j & i are "similar"

Implementation Detail: Mean Normalization:

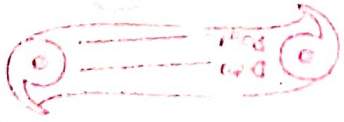
Take Mean.

Normalize i.e $(x_i - \bar{x})$.

Matrix

↓

Use it to learn $\theta^{(j)}, x^{(i)}$.

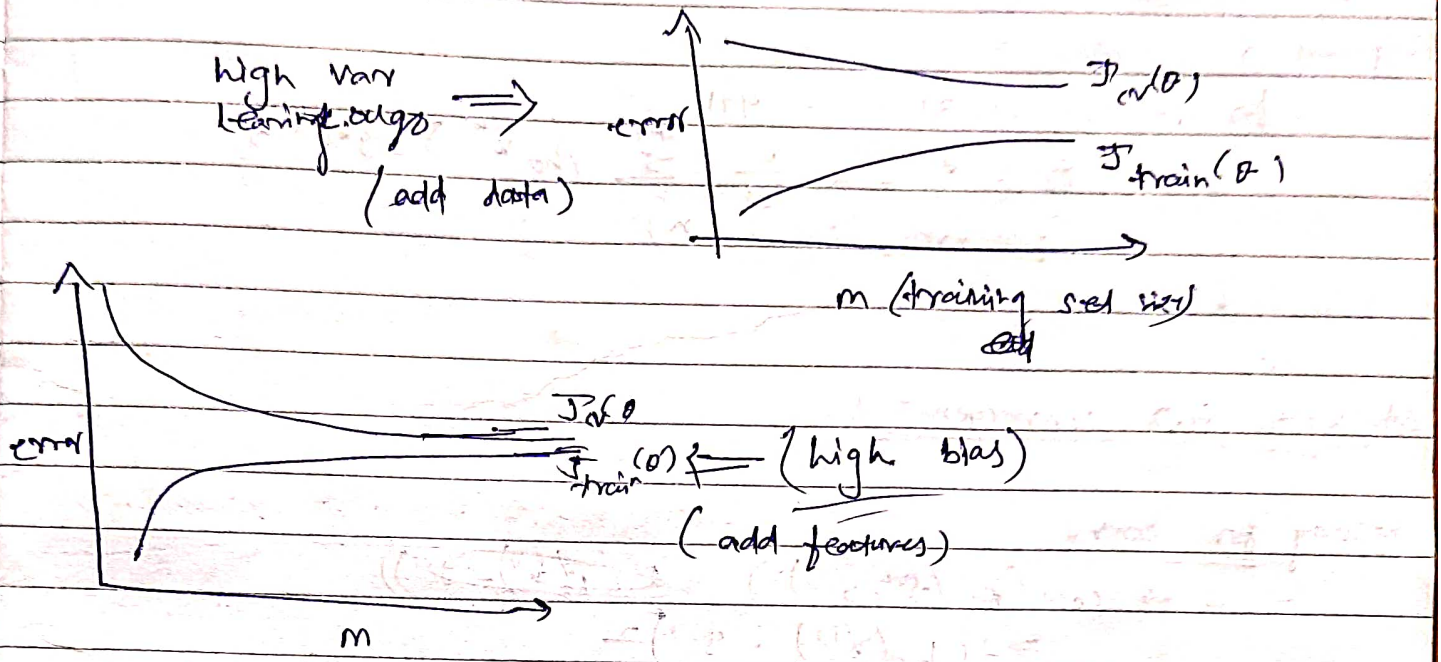


For user j on movie i predict :

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

Large Scale M.L.

Learning with large datasets :



- Stochastic Gradient descent :

~~(Batch G.D)~~ (~~stop~~ gradient descent = Batch G.D)

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

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

Just 1 random training eg. instead of all.

→ Randomly shuffle dataset.

→ Repeat { for $i = 1, \ldots, m$

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

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

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

} }

→ $(x^{(1)}, y^{(1)})$ → progress → $(x^{(2)}, y^{(2)})$ → progress → - - - -

(progress towards global minima)

Batch G.D : Use all m rm. in each iteration.
Stochastic G.D : Use 1 " "

Mini Batch grad. descent :

vs

here — Use b rms. in each iter.
b = Mini-batch size.

say : b = 10, m = 1000 :

Repeat {
for i = 1, 11, 21, 31, - - - -991
{ $\theta_j = \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} \left(h_\theta(x^{(k)}) - y^{(k)}\right) x_j^{(k)}$
(for every $j = 0, \ldots n$)
} }

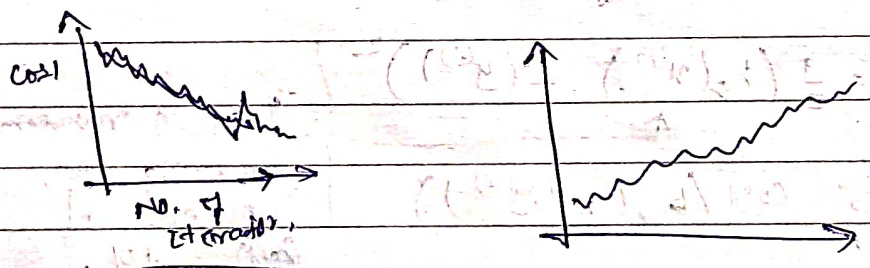
Stochastic G.D convergence :

Checking for conv :

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

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

→ Every 1000 iter (say), plot cost averaged over the last 1000 rms processed by algo.



← algo is diverging.
Soln : Use small learning rate (α).

L.R ⇒ cost, can slowly decrease α over time if we θ to converge.

$\alpha = \frac{const1}{iter\ no. + const2}$

coz iter no ↑ ⇒ α ↓

Online Learning :

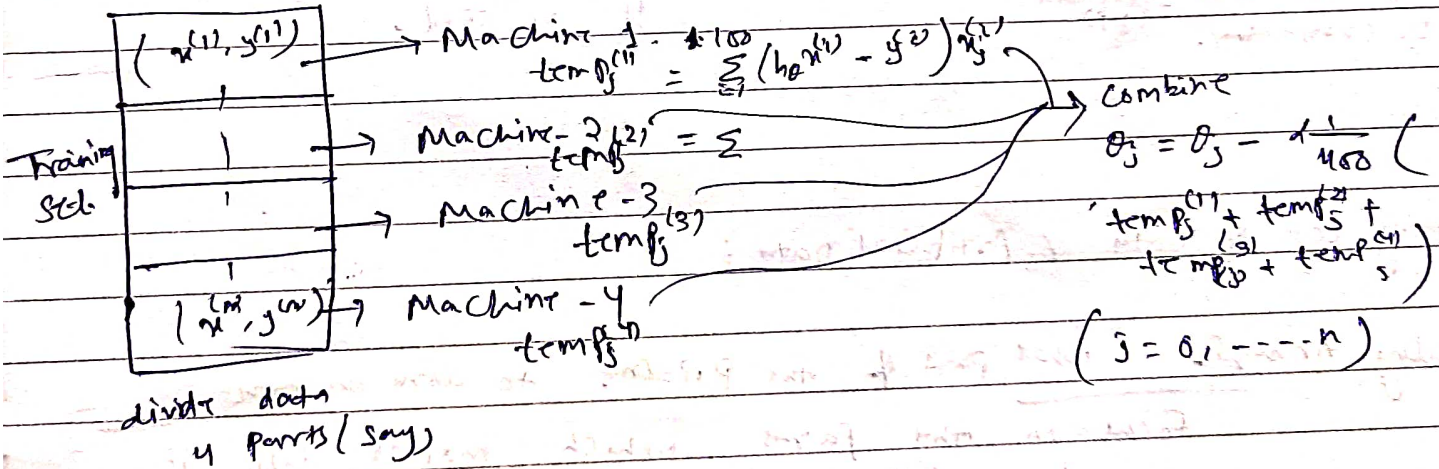
(Logistic regression or N.N.

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

(It can adapt to changing user preference)

Problem (Predicted CTR Prob) $P(y = 1 \mid x; \theta)$

Map Reduce & Data Parallelism



Application Example: Photo OCR :
(Photo Optical Character Recognition)

Problem description & Pipeline:

Pipeline of photo OCR

→ Text Detection.
→ Character segmentation.
→ Character Classification.

Sliding windows
(supervised learning classifier)

Step_size / Stride
"Expansion" operator.

for char. segmentation :
(run supervised learning model :)

+ve ex + (y=1)
If there is a split bet^n (gap)
2 characters.

-ve ex :/ (y=0)
If no split.

Getting Lots of Data & Artificial Data :

Ceiling Analysis : what part of the pipeline to work on next

focus on the parts which makes diff in overall model.