1. Supervised learning — Intro to ML Andrew N. G
2. Unsupervised learning

Supervised learning:
- regression (house size - price problem)
Continous data of features is given New data is placed in accordance with previous ones

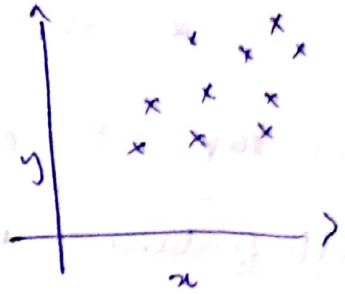
Classification: (cancer problem)
features are classified into 2 or more set outputs

Unsupervised learning (Clustering)
output of features values and specified Algorithm divides values based on clusters

---

Linear Regression:

training data (x) output (y)



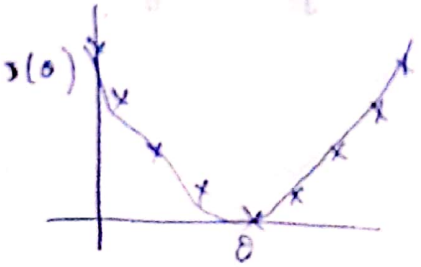
$x^{(i)}$ -> ~~first~~ i^{th} training data
$y^{(i)}$ -> i^{th} output data

$h(x) = \theta_0 + \theta_1 x$
θ_0, θ_1 -> parameters of model.

cost fn $\boxed{J(\theta_0, \theta_1) = \frac{1}{2m} \sum \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2}$ (Square error cost fn)

minimise error

Start with some θ value and plot $J(\theta)$ vs θ



choose θ for which $J(\theta)$ is minimum

Sidhik Uday

for more parameters, contour plots are made.

The values of θs for which contour is at lowest are the values to be chosen.

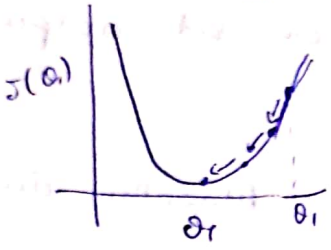
gradient descent: (Batch gradient descent)

to find θ value

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta} J(\theta_0, \theta_1) \qquad j = 0, 1$$

α → learning rate. (determines the number of steps to take)

Choose a θ_1



derivative term is given by tangent. positive slope.

So next value of θ is lower.

This will keep happening till θ is min.

If α is small, gradient descent is slow

α is large, gradient descent can overshoot min

Every step of GD involves all training sets

Multi feature Linear Regression

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

Let $x_0 = 1$.

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

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

θ - parameter vector

x - feature vector.

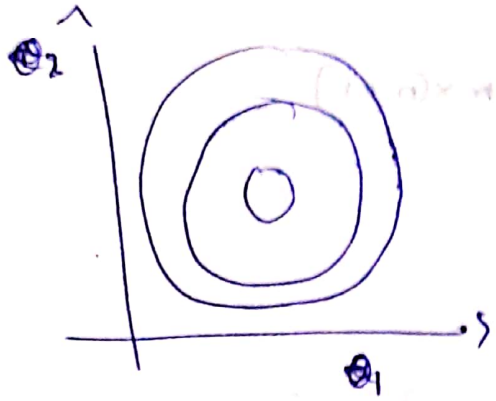
Same gradient descent algorithm for this.

eature Scaling:

if one feature is more has large range than other, then gradient descent will find longer time to

$x_1 = (0 - 2000)$ $x_2 = (1 - 5)$

let $x_1 = \frac{value}{2000}$ $x_2 = \frac{value}{5}$.



much faster.

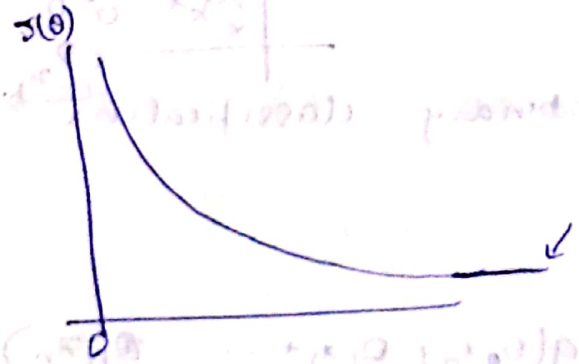
optimum value $-1 < x < 1$.

Mean normalisation.

Replace x_i with $x_i - \mu_i$ to make features approximately zero mean.

$x = \frac{x_i - \text{avg value}}{\text{range}}$ $-0.5 < x < 0.5$.

Making sure G.D works properly



Automatic convergence test: declare convergence if $J(\theta)$ decreases by ~~less~~ than 10^{-3} in one iteration.

Polynomial Regression:

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

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

Normal Equation method.

$X = [x_0 \; x_1 \; x_2 \; \dots \; x_n] \quad m \times (n+1)$

$y = [y] \quad m \times 1$

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

no need to choose α

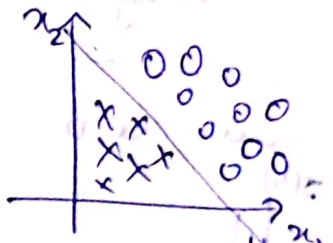
no need to iterate

Slow if n (no of feature) is large

$X^T X$ can be non-invertible when

- redundant feature
- lot of feature ($m < n$)

Classification



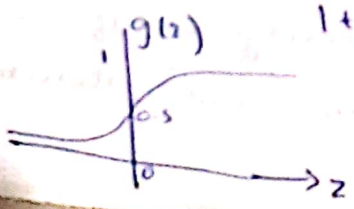
Logistic Regression:

2 output $y = \{0, 1\}$ binary classification

or $h_\theta(x) < 1$

$h_\theta(x) = g(\theta^T x) = g(\theta_0 x_0 + \theta_1 x_1 + \dots \theta_n x_n)$

$g(z) = \frac{1}{1 + e^{-z}}$ → logistic function
Sigmoid function.

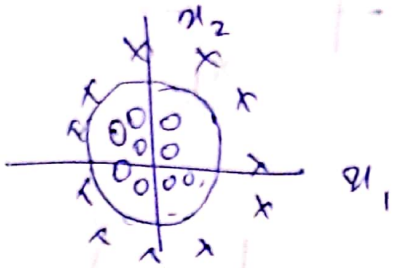


$h_\theta(x) = P(y=1 \mid x;\theta)$

probability that $y=1$, given x, parameterised by θ.

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

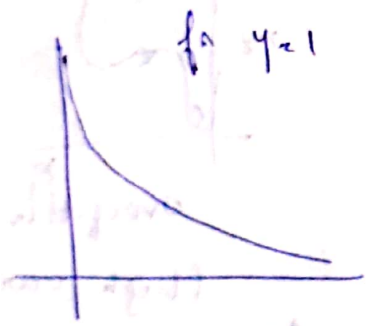
Non linear decision boundary

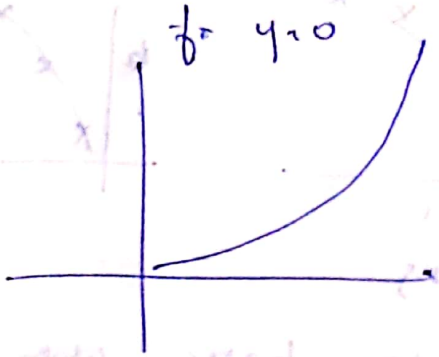


$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

Cost function:

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





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

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

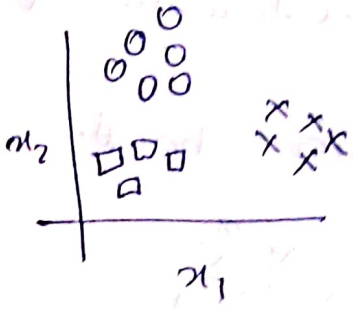
$$\theta_j = \theta_j - \alpha \frac{\delta}{\delta \theta_j} J(\theta) \quad \rightarrow \text{GD}$$

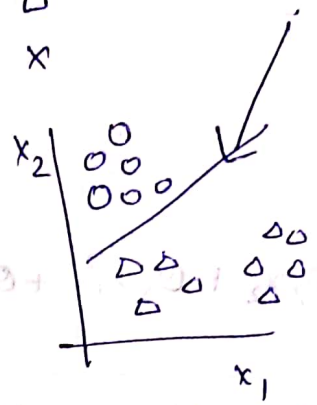
optimisation algorithms such as conjugate gradient, BFGS, L-BFGS.

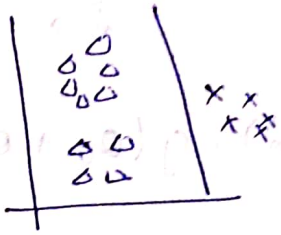
Multi Class Classification

$y = \{1, 2, 3, 4\}$

Class 1: O
Class 2: □
Class 3: ×

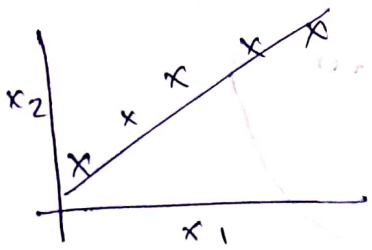




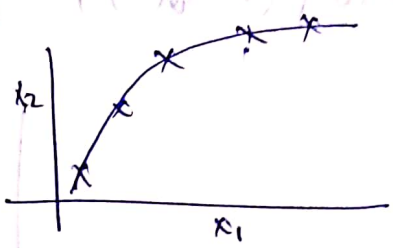


similarly do for class 2, 3.

Overfitting



underfitting



overfitting
(high variance)

overfitting can happen when there are too many features. Cost function but it may fail to categorise new data.

Regularisation:
we can make some parameters ≈ 0.
(penalising parameters)

Small values of θ - less prone to overfitting.
- Simpler hypothesis.

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

helps in achieving this.

linear regression.

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

$\downarrow$

< 1

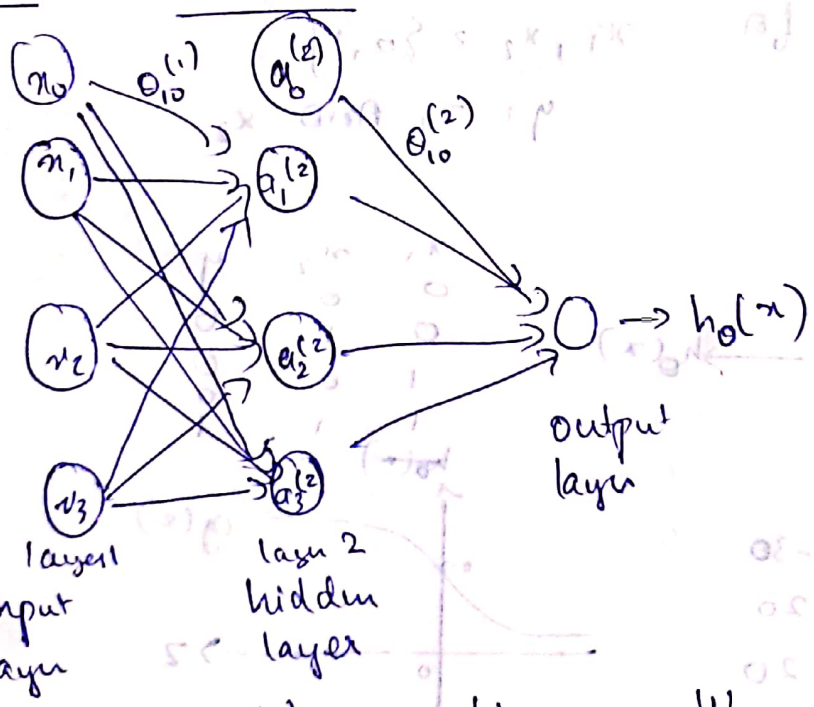
Shrinking parameter.

for logistic regression.

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

NEURAL NETWORKS.

(θ = parameters / weights)



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

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

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

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

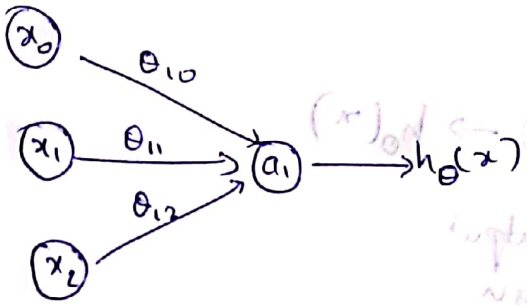
$$a_1^{(2)} = g(z_1^{(2)})$$
$$a_2^{(2)} = g(z_2^{(2)})$$
$$a_3^{(2)} = g(z_3^{(2)})$$
$$z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$z^{(2)} = \theta^{(1)} x$$
$$a^{(2)} = g(z^{(2)})$$
$$z^{(3)} = \theta^{(2)} a^{(2)}$$
$$h_\theta(x) = a^{(3)} = g(z^{(3)})$$

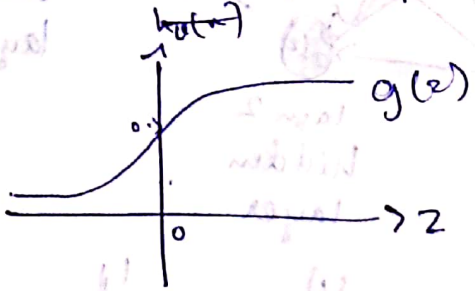
forward propogation.

Neural network for $x_1, x_2 = \{0, 1\}$

$y = x_1$ AND x_2



x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



let $\theta_{10} = -30$
$\theta_{11} = 20$
$\theta_{12} = 20$

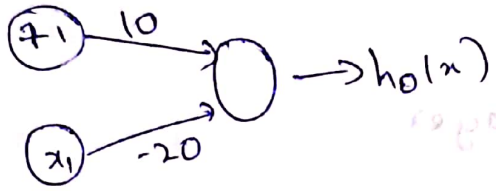
x_1	x_2	$h_\theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

al network for OR.

$\theta_{10} = -10$ $\theta_{12} = 20$ $\theta_{13} = 20$

x_1	x_2	$h_\theta(x)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	$g(10) \approx 1$
1	1	$g(30) \approx 1$

Neural network for NOT.

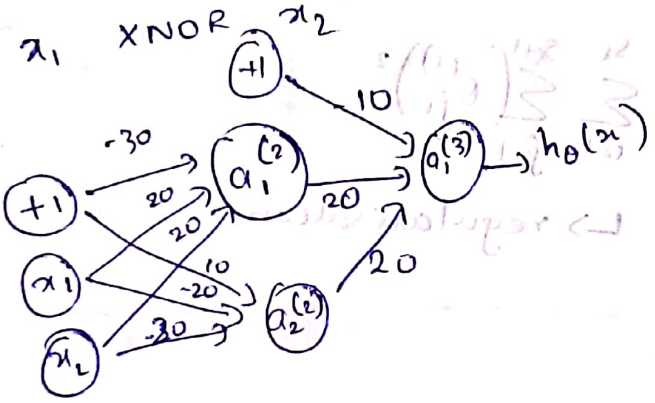


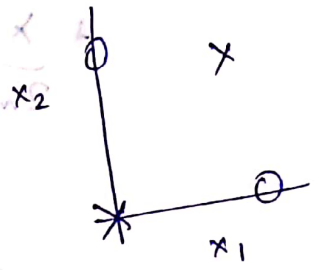
x_1	$h_\theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

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

---

x_1 XNOR x_2





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

Multiclass classification.

No of outputs of Neural n/w should be no of classes.

$y(i)$.

$$\begin{bmatrix}1\\0\\0\\0\end{bmatrix} \quad \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \quad \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \quad \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

different outputs

Cost function for Neural Network.

$L \rightarrow$ no of layers

$S_l \rightarrow$ no of neurons in a layer

for binary classification,

$S_L = 1$.

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

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

$\rightarrow$ regularisation

To minimise cost function.

Backpropogation algorithm

for a training example,

do forward propogation

$$a^{(1)} = x$$

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

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

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

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

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

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

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

$$\delta_j^{(4)} = a_j^{(4)} - y_i = h_\theta(x) - y_i$$

y → value of y in training set

$a_j^{(4)}$ → hypothesis output

$$\delta^{(4)} = a^{(4)} - y$$

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

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

$$g'(z^{(3)}) = a^{(3)} .* (1 - a^{(3)})$$

There is no $\delta^{(1)}$.

Set $\Delta_{ij}^{(l)} = 0$.

$$\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)} \rightarrow \text{accumulate } \delta \text{ terms.}$$

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

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

$$j = 0$$

$$D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)}$$

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = D_{ij}^{(l)} \rightarrow \text{can substitute this in a GD algorithm.}$$

$$\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} cost(i)$$

$$cost(i) = y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log h_\theta(x^{(i)})$$

When δ terms change, it changes the cost function also, hence increases its efficiency. Thus change weights.

... to be zero initially.

Training a NN

i) Pick a network architecture,
ii) Randomly initialing weights
iii) Implement forward propogation
iv) find cost function $J(\Theta)$
v) Implement back prop to get $\frac{\delta J(\Theta)}{\delta \Theta^{(l)}_{ijk}}$
vi) use gd & any optimisation method with backprop to minimin $J(\Theta)$

Evaluating hypothesis:

training set : 60%.
cross validation : 20%.
test set : 20%.

select different hypothesis & train & find cross validation error.
Select model with least CV error.

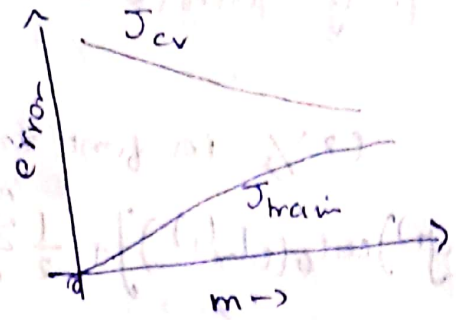
---

bias Vs Variance

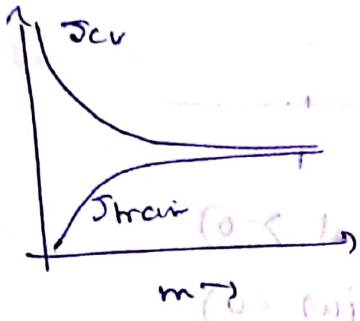
if degree of polynomial is low, then its bias problem.
if degree of polynomial is larg, then its variance problem.

Bias (underfit)	Variance (overfit)
J_{train} is high.	J_{train} is low
$J_{cv} \approx J_{train}$	J_{cv} is high

ning curves

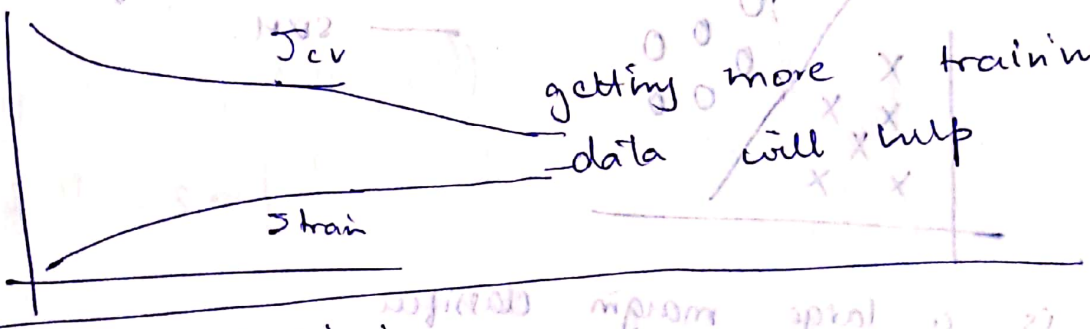


high bias



for high bias, getting more training data won't help.

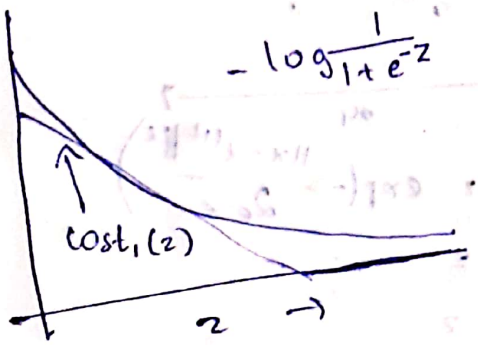
high variance



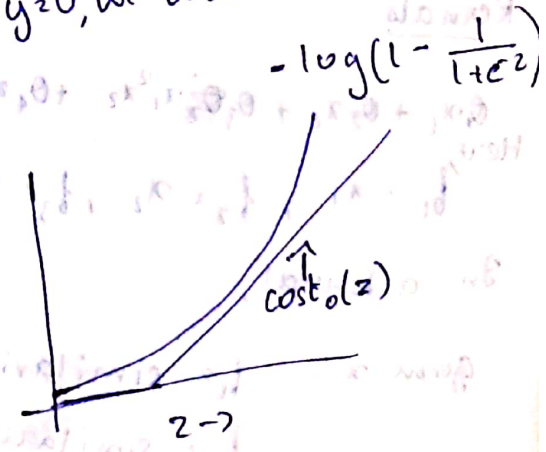
getting more training data will help

Support Vector Machine:

when $y=1$, we want $\theta^T x \gg 0$



$y=0$, we want $\theta^T x \ll 0$

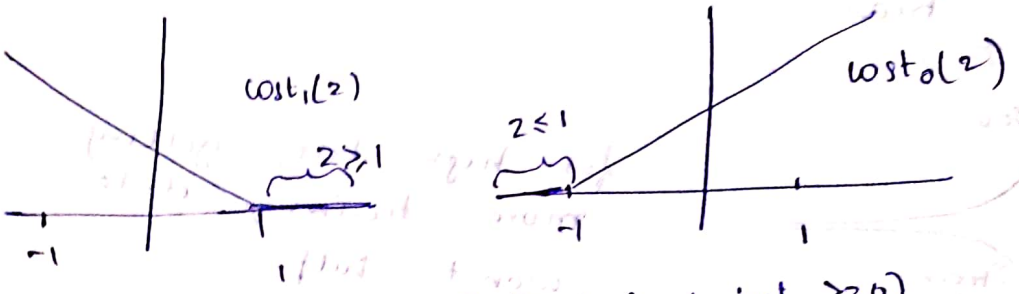


∴ for support vector machine,

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

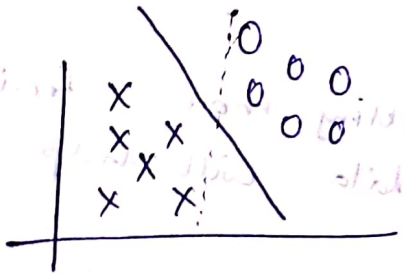
ignore m, introduce (2) in first term.

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



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

SVM decision boundary:



.... → linear regression
—— SVM.

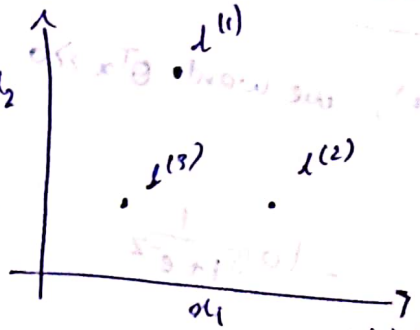
it is a large margin classifier.

Kernals

$\theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1 x_2^2 + \ldots$

Here, $f_1 = x_1$, $f_2 = x_2$, $f_3 = x_1 x_2 \ldots$

In a kernal,



k-landmarks

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

$f_2 = \text{similarity}(x, l^{(2)}) = \ldots$

$f_3 = \text{similarity}(x, l^{(3)}) = \ldots$

kernal fn (gaussian kernal)

$\|w\|$ = length of vector w

if $x \approx l^{(1)}$

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

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

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

Compute values of θ for different values of $f_1, f_2, f_3 \ldots$

∴ In the plot we will have regions where $y=1$ and regions where $y=0$. Hence we will get the decision boundry

---

SVM with Kernels

given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$,

choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)}$

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

$$f = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_m \end{bmatrix}$$

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

using SVM algorithm, replace $x^{(i)}$ by $f^{(i)}$

SVM parameter: $C = 1/\lambda$.

large C: lower bias, high variance

low C: higher bias, low variance

σ^2

large σ^2: feature f vary more smoothly, High bias low variance

Small σ^2: vary less smoothly, lower bias, higher variance.

To use svm software package,

- specify C
- choose kernal

For svm packages' optimisations to work properly, it should satisfy Mercer's theorem.

Other kernals:

polynomial kernal: $k(x, l) = (x^T l)^2$

$(x^T l + 2)^4$

:

For Multiclass classification,

many kernals support this, otherwise follow one versus all method.

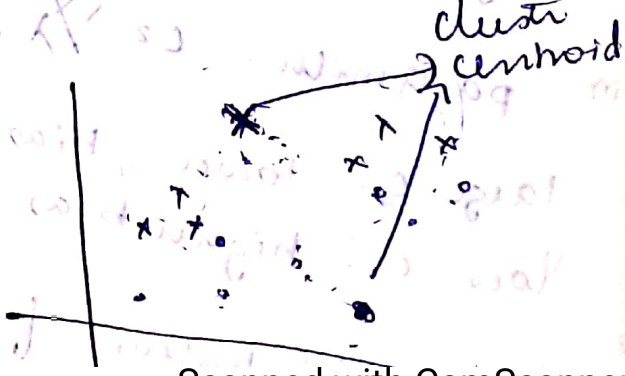
UNSUPERVISED LEARNING

Clustering: data is given without it being classified. Algorithm groups it on its own.

K-Means algorithm

1) cluster assignment:

In case you need to group it into 2 groups, initialize two cluster centroids (random). It will go through each data and it will assign to one of the cluster centroid depending on which is near to it.



2) Now take mean of tw two groups & move the centroid to mean value.

Repeat step ① & ② for the new centroids, till the centroids won't change.

k → no of clusters.

Randomly initialise K cluster centroids $\mu_1, \mu_2, \mu_3 \cdots \mu_k$

Repeat { for i = 1 to m

$c^{(i)}$ = index (1 to K) of cluster centroid closest to $x^{(i)}$
$(\|x^{(i)} - \mu_k\|)$

for k = 1 to K

μ_k = avg(mean) of points assigned to k.

}

Optimisation objective:

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

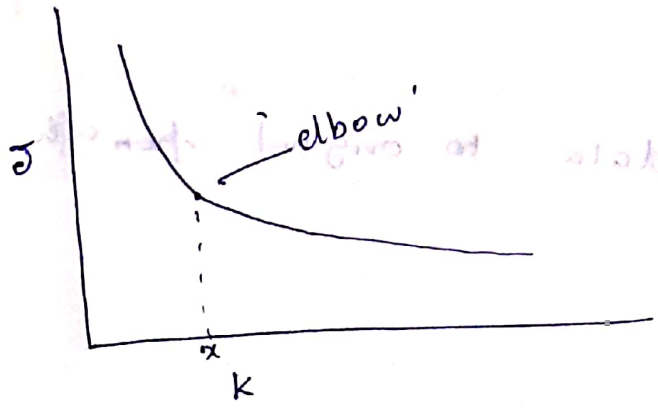
minimise $J(\ )$

In step 1, need to minimise J for c values keeping μ constant & in step two need to minimise J for μ value,

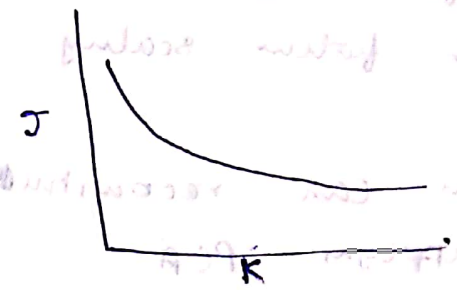
---

Choosing value of k

elbow method

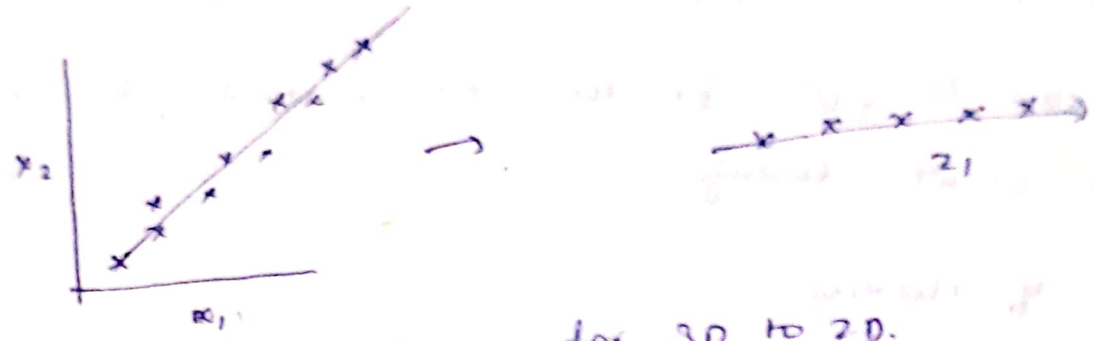


take K = x



difficult to choose k

Data Compression 2D → 1D

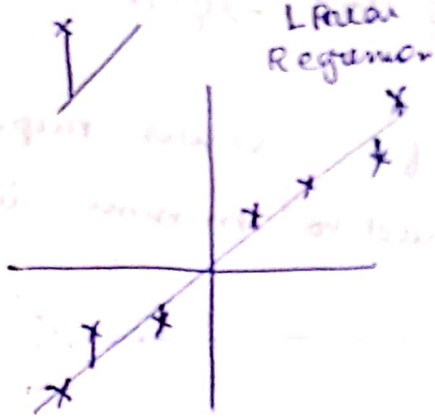


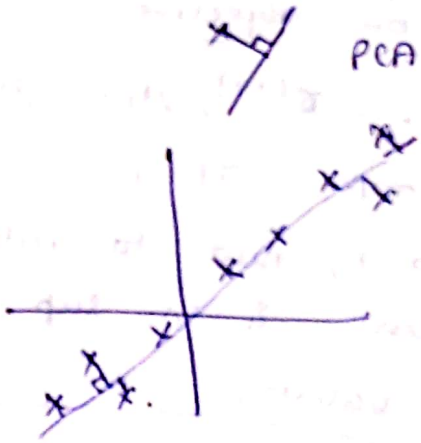
Same can be done for 3D to 2D.

when large no of feature are given, you can compute combination of them & reduce no of feature for better visualisation.

PCA - Principal Component Analysis

It tries to find a lower dimensional surface onto which data is projected so that sum of square from data to projected plane / line is minimum.



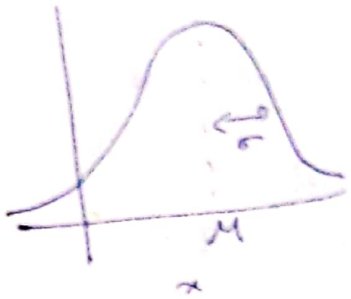


Perform mean normalisation before PCA along with feature scaling

You can reconstruct data to original from after applying PCA

Gaussian Distribution

x is distributed gaussian with mean μ, variance σ^2



$$p(x, \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Density estimation

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

$$= \prod p(x_j, \mu_j, \sigma_j^2)$$

Anomaly detection:

Choose x_i that might be indicative of anamalous examples.

Fit $\mu_1, \ldots, \mu_n, \sigma_1^2, \ldots, \sigma_n^2$

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

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

for a new x,

$$p(x) = \prod p(x_j, \mu_j, \sigma_j^2)$$

Anomaly if $p(x) < \epsilon$

Algorithm evaluation

Fit $p(x)$ on training set

On cv/test x, predict

$$y = \begin{cases} 1 & \text{if } p(x) < \epsilon \text{ (anomaly)} \\ 0 & \text{if } p(x) \geq \epsilon \text{ (normal)} \end{cases}$$

Possible evaluation metrics

- true positive, false positive, true negative, false negative
- Precision / Recall — F1-score

Anamoly detection vs Supervised learning

- Small no of positive (anamoly) examples
- Large no of both

Fraud detection
Manufacturing dfct

Email spam classification
weather prediction

Choosing features for anomaly detection.

- better when selected features have proper curve
- if a feature won't give out proper curve, transform it. Eg $x_1 \rightarrow \log(x_1 + 1)$

Predicting movie problem.
train feature with user input
collaborative filtering algorithm

Dealing with large datasets. Stochastic G.D

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

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

Randomly shuffle dataset

Here we are looking at training example & modifying parameters
while in Batch G.D (all at once) whose feature sum is made & parameters changed.

Mini-Batch G.D

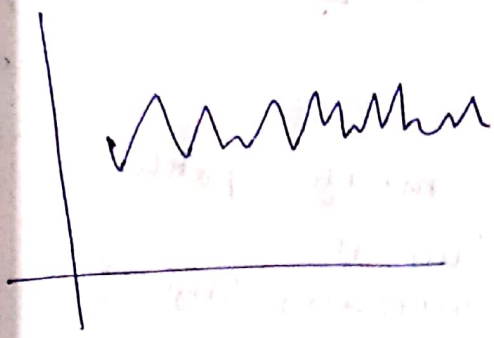
In stochastic, we use only 1 example in each iteration
In mini-Batch, use b examples in each iteration

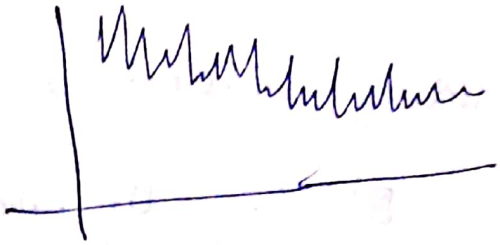
$$\theta_j = \theta_j - \alpha \frac{1}{10}\sum_{k=i}^{i+9}(h_\theta(x^{(k)}) - y^{(k)})x_j^{(k)}$$

Checking for convergence.

Cost fn vs No of iterations curve isn't smooth.

Eg





can slowly decrease α over time

$= \frac{\text{const}}{\text{iteration no} + \text{const2}}$

Map Reduce

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

Split training set into several sub sets.

Machine 1: first 100

$temp_j^{(1)} = \sum_{i}^{100} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$

Machine 2: next 100

$temp_j^{(2)} \ldots$

Combine

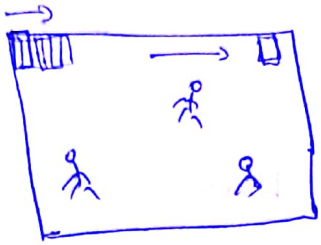
$\theta_j = \theta_j - \frac{\alpha}{400} (temp_j^{(1)} + temp_j^{(2)} + temp_j^{(3)} + temp_j^{(4)})$

In single computers with multiple core machines, it can split & computated.

Photo OCR pipeline.

Image → Text detection → Character segmentation → Character Recognition

Sliding window detection.



take small image patch & keep sliding it. And keep increasing size of patch.

Decision - Tree.

Data is continously split according to a certain parameter.

two entities;
decision nodes – data is split
leaves – final outcomes.

two types:
classification trees
decision variable is categorical

Regression trees
outcome variable is continous

algorithm for decision tree

ID3 algorithm, Iterative Dichotomiser 3

Basic definitions:

Entropy: for a finite S, it is the measure of uncertainty or randomness in data

Information gain: (Kullback - Leibler divergence) $IG(S,A)$ for a set S is the effective change in entropy after deciding on a particular attribute A. It measures a relative change in entropy with respect to the independent variables.

$$IG(S,A) = H(S) - H(S,A)$$
$$= H(S) - \sum P(x) * H(x)$$

$IG(S,A)$ is the information gained by applying feature A.

steps:

- create root node for the tree
- If all examples are positive, return leaf node positive.
- If all examples are negative, return leaf node negative
- Calculate the entropy of the current state $H(S)$
- For each attribute, calculate the entropy with respect to the attribute 'x' denoted by $H(S,x)$
- Select the attribute which has maximum value of $IG(S,x)$
- Remove the attribute that offers highest IG from set of attributes.
- Repeat until we run out of all attributes, or the decision tree has all leaf nodes.

Select the feature as root node which has max information gain.
and so on for following nodes.