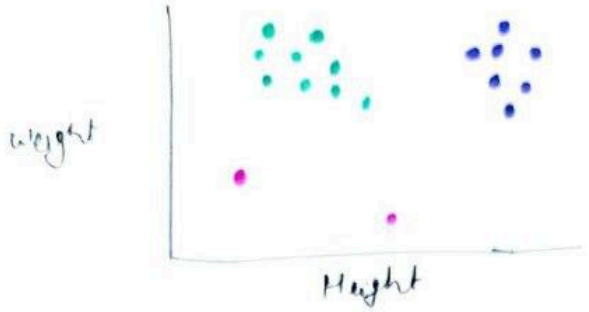


When we are done with all core points and left with only non-core points that can't be added to any clusters, we are done.

These non-core points is called outliers

- Its 2D, so considering circle, hypersphere else.
- Bottom two aren't core point as doesn't contain point greater than equal to m.

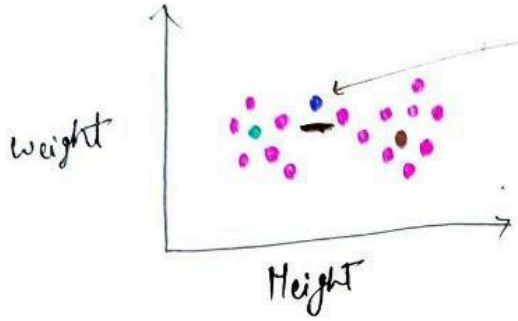
(i) Find core point
↓
based on putting circle disc m points, check if falling as core point, randomly choose any core, find its neighbour
↓
repeat.

(ii) Repeat with different radius values find which 'r' considers more point.

(iii) Choose intutive value of r

There are two parameters is m, r

DBSCAN disadvantage :-

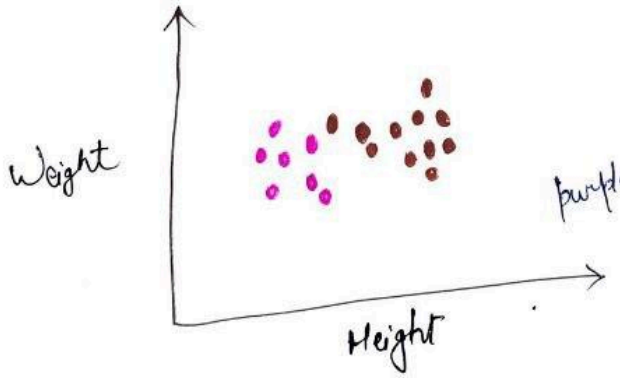


purple is non core point.
So it can't expand a cluster.

Case 1 :- If "Green" point is choosed for expansion :-

purple point will be included in first cluster.

Case 2 :- If "Brown" point is choosed for expansion :-



purple point will get included in second cluster.

$$\frac{\text{intra distance}}{\text{inter distance}} > 0$$

intra cluster distance should be greater than inter class cluster.

Notion of Cluster Quality :—

→ Intra distance$_1$ > Intra distance$_2$ (not good for 1)

. Intra distance should be lesser, more compact cluster.

Inter distance$_1$ < Inter distance$_2$ (not good for 1)

→ Inter distance b/w cluster should be greater. ~~tha~~

DBSCAN : Let $\phi = \{x_1, x_2 \ldots x_n\}$ be a set of points to be clustered.

(i) Choose $\epsilon > 0$ and $\mathfrak{n}$

ϵ = radius, $\mathfrak{n}$ = no of points atleast threshold

(ii) Let A_i be set of points that lies within a disc of radius ϵ from x_i.

Do it for every x_i.

(iii) If $A_i < \mathfrak{n}$ → not considering for calculation.

. Points other than this is called core points.

(iv) Take union of A_i and A_j if $A_i \cap A_j \neq \phi$

- go on doing it till no more union possible.

Disadvantage :-

For a non-core point which falls in mid of 2 clusters,

↓

Clusters depends on from where I start clustering.

Advantage :-

Biggest advantage of DBSCAN is we don't need to know no. of cluster priorly.

But we need to know m, ε.

What to do with points lying on border ?

↳ There might be many points lying on border.

↓

Recheck after converting those point to part of cluster.

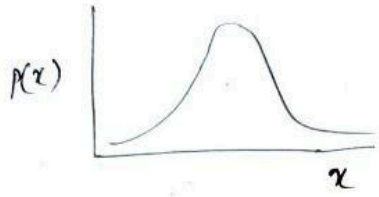
↓

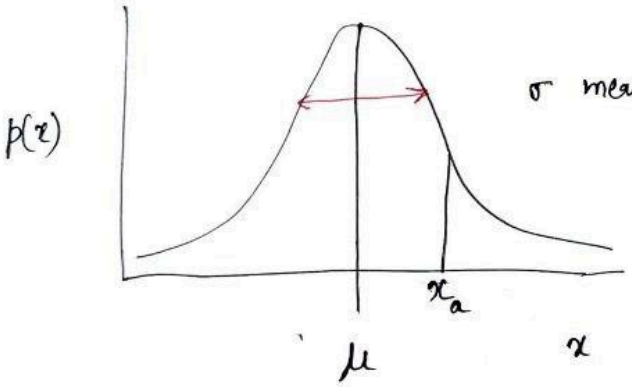
compare.

ε = radius r = no of points.

Gaussian Mixture Model :

- A gaussian (normal) distribution with mean μ and std σ





The belonging of point x_a to normal distribution is :-

$$P\left(\frac{x_a}{\mu,\sigma}\right) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x_a-\mu)^2}{2\sigma^2}}$$

→ If x_a is closer to μ,

x_a is more likely to belong to the distribution.

σ = measure of spread. $\uparrow\uparrow\sigma$ = distribution is wider

$\downarrow\downarrow\sigma$ = distribution is sharp.

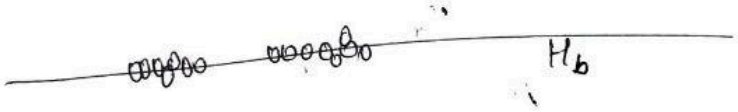
→ If x_a is closer to mean more likely to belong to distribution.

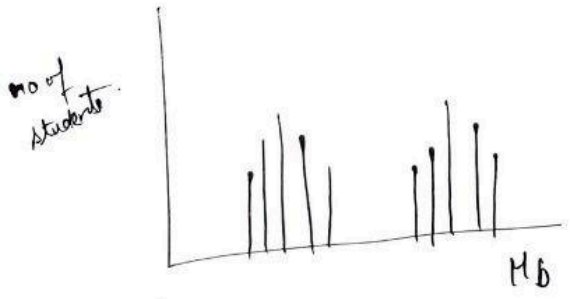
- For multivariate normal distribution :-

For gaussian on multiple random variables, the belongingness of a high-dimensional point x_a to normal distribution is :-

$$p(x_a | \mu, \Sigma) = \frac{1}{\sqrt{2\pi\Sigma}} e^{\frac{-(x_a - \mu)^T \Sigma^{-1} (x_a - \mu)}{2}}$$

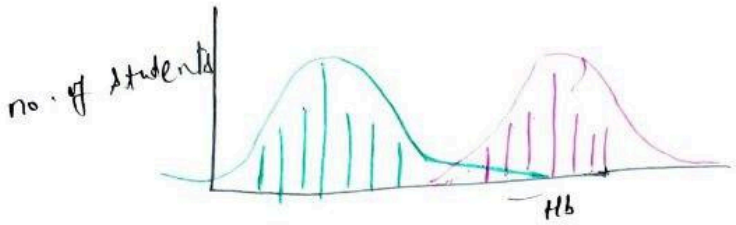
GMM :- I. measure Hb of IITJ students :-



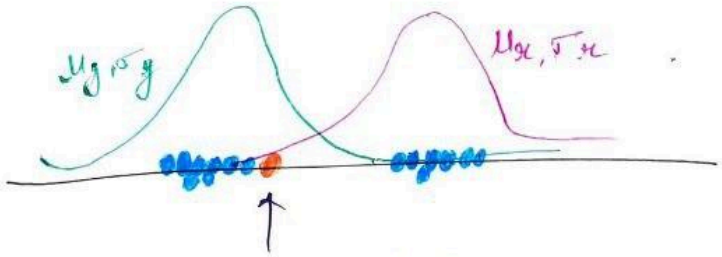


These data points can't be explained by single gaussian.

→ Total distribution of students can be modelled as mixture of two gaussians.



Inverse Fourier transform decompose sum of sin, cosine waveform.



- Orange point is close to green curve.
- Higher probable to green.

$$p_g\left(x^n \mid \mu_g, \sigma_g\right) = \frac{1}{\sqrt{2\pi}\sigma_g} e^{\frac{-\left(x^n - \mu_g\right)^2}{2\sigma_g^2}}$$

$$p_R\left(x^n \mid \mu_R, \sigma_R\right) = \frac{1}{\sqrt{2\pi}\sigma_R} e^{\frac{-\left(x^n - \mu_R\right)^2}{2\sigma_R^2}}$$

→ Clustering using GMM :-

(i) Randomly initialize K gaussians.

(ii) For all data points, check which Gaussian it belongs to.

(iii) Based on belongingness assign each data point to cluster.

(iv) Find μ_i and Σ_i from each cluster and update Gaussians.

(v) repeat step (ii) – (iv) unless termination criteria is satisfied.

Termination Criteria :-

Many termination criterias are possible :-

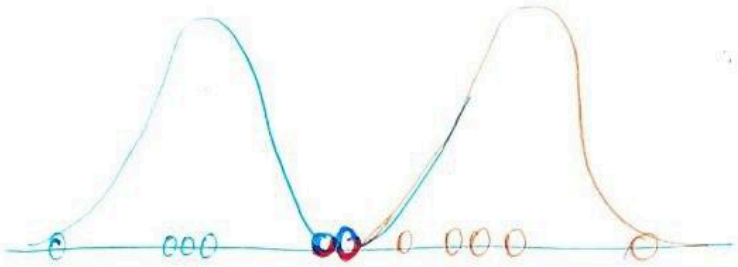
(i) we may terminate if μ_i, Σ_i for each cluster don't change change significantly over a few epochs.

(ii) we may terminate after a certain predefined no of epochs.

(iii) we may terminate when log likelihood of all data points doesnot change over a few epochs

$$l(\theta) = \sum_{j=1}^{N} \ln P\left(\frac{x^j}{\theta}\right)$$

Soft Clustering using GMM



- Suppose we take K clusters
- Randomly Initialize K Gaussians
- The i^{th} Gaussian is given by :- $\mathcal{N}(\mu_i, \Sigma_i)$

→ Suppose we take K clusters $C_1, C_2 \ldots C_K$.

→ Consider a data point x^n.

- $p(x^n)$ = prob. that x^n is in C_1 or prob. that x^n is in C_2 or ..... prob. that x^n is in C_K.

So $p(x^n) = p(x^n, C_1) + p(x^n, C_2) + \cdots + p(x^n, C_K) = \sum_{i=1}^{K} p(x^n, C_i)$

- $p(x^n) = \sum_{i=1}^{K} p(x^n, C_i) = \sum_{i=1}^{K} p\left(\frac{x^n}{C_i}\right) p(C_i) = \sum_{i=1}^{K} p\left(\frac{x^n}{C_i}\right) \pi_i$

- $\pi_i = p(C_i)$ → mixing coefficient.

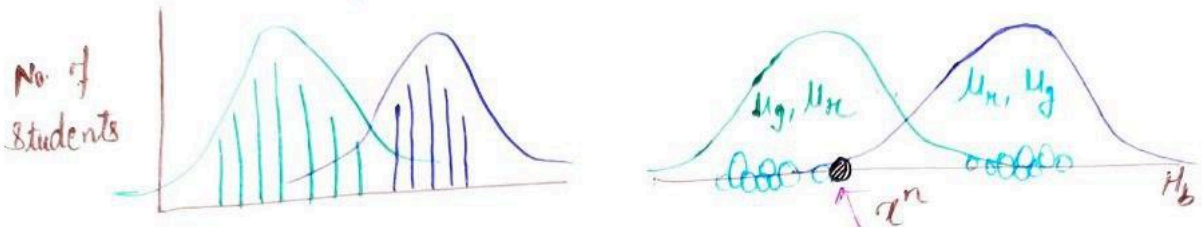
Using Baye's theorem we calculate probability that x^n belongs to cluster i :-

$$P\left(\frac{C_i}{x^n}\right) = \frac{P\left(\frac{x^n}{C_i}\right) \cdot P(C_i)}{P(x^n)} = \frac{\pi_i \, P\left(\frac{x^n}{C_i}\right)}{\sum_{i=1}^{K} \left(\pi_i \, P\left(\frac{x^n}{C_i}\right)\right)}$$

π_i = In what proportion this point is mixed with other cluster.

Lecture-20 Gaussian Mixture Models

Date: 26-09-2024 ①



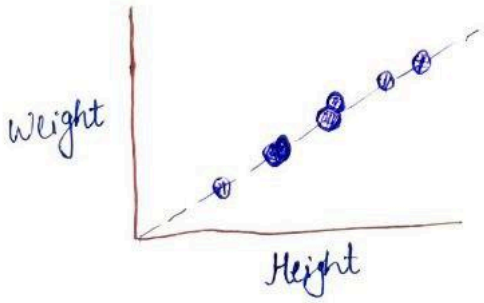
$$p_g\left(x^n \mid \mu_g, \sigma_g\right) = \frac{1}{\sqrt{2\pi}\,\sigma_g} e^{-\frac{(x^n-\mu_g^n)^2}{2\sigma_g^2}}$$

Consider this student

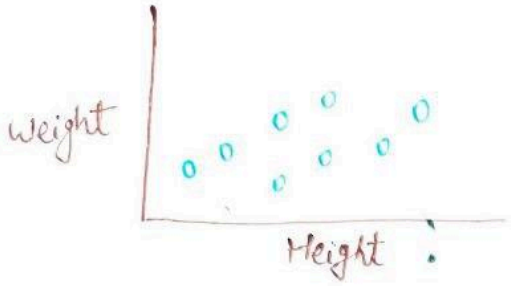
$$p_r\left(x^n \mid \mu_r, \sigma_r\right) = \frac{1}{\sqrt{2\pi}\,\sigma_r} e^{-\frac{(x^n-\mu_r)^2}{2\sigma_r^2}}$$

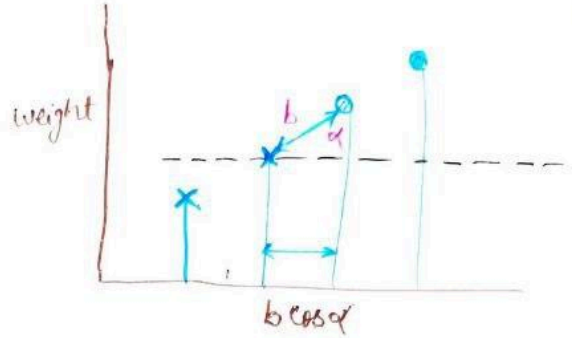
What is belongingness of this student to the Green and Purple Gaussian?

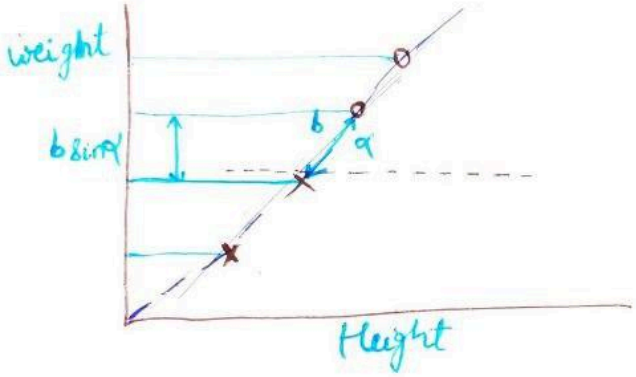
Classification Problem :-



- I want to project dataset on blue line.
- Variance of data indicate spread of data along blue line.







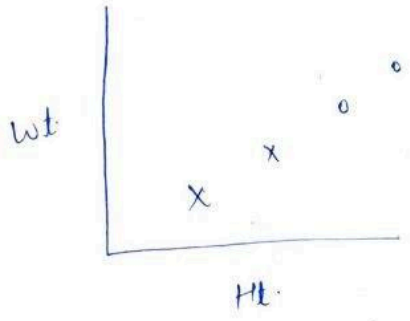
If I try to classify using a combination of ht. and wt

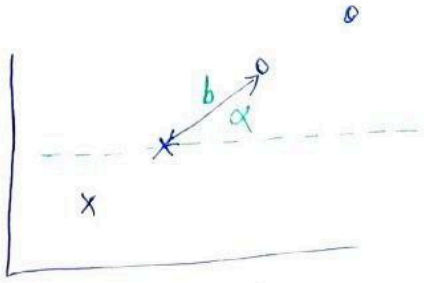
→ This gives best decision Boundary

↓

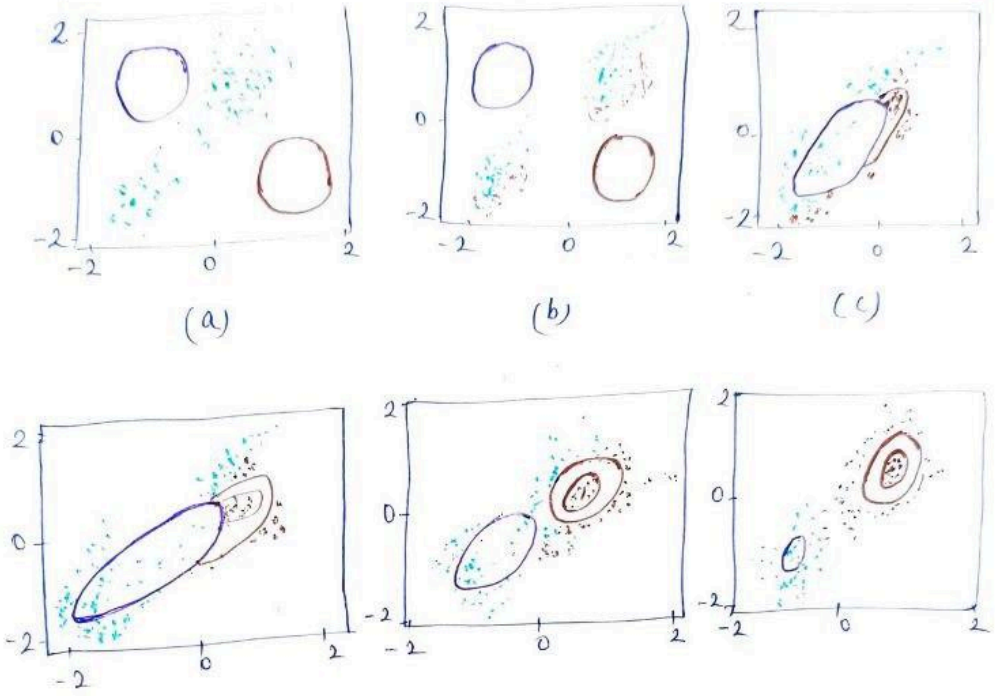
D.B with heighest gap among the samples of different classes.

- The diversity of samples are highest along blue (derived) line.





GMM in action:



Expectation Minimization :-

(i) Randomly initialize K gaussians (randomly initialize μ_i, Σ_i, π_i)
. The i^{th} Gaussian is given by $\mathcal{N}(\mu_i, \Sigma_i)$

(ii) E-step : For each data point, check its belongingness to each of K Gaussians.

(iii) M-step : Using belongingness, recalculate μ_i, Σ_i, π_i for each cluster i to maximize log likelihood.

(iv) Repeat steps 2-4 unless the termination criteria is classified.

- Soft - Clustering using GMM ← Expectation Minimization

Calculating μ_i, Σ_i : It can be shown by taking partial derivative of log-likelihood, that log likelihood will be maximum if :-

$$\mu_i = \frac{\lambda_i^{(1)} x^{(1)} + \lambda_i^{(2)} x^{(2)} + \cdots + \lambda_i^{(N)} x^{(N)}}{\lambda_i^1 + \lambda_i^2 + \cdots \lambda_i^N}$$

$$\sum_i = \frac{\lambda_i^{(1)} (x^{(1)} - \mu_i)(x^1 - \mu_i)^T + \lambda_i^{(2)} (x^{(2)} - \mu_i) + \cdots + \lambda_i^{(N)} (x^{(N)} - \mu_i)(x^{(N)} - \mu_i)^T}{\lambda_i^{(1)} + \lambda_i^{(2)} + \cdots + \lambda_i^{(N)}}$$

- $\mu_i = \frac{\sum_{n=1}^{N} (\lambda_i^n x^n)}{\sum_{n=1}^{N} \lambda_i^n}$ $\quad \Sigma_i = \frac{\sum_{n=1}^{N} \left(\lambda_i^n (x^n - \mu_i)(x^n - \mu_i)^T \right)}{\sum_{n=1}^{N} \lambda_i^n}$

$$\Pi_i = p(C_i) = \frac{\text{Data points belonging to } C_i}{\text{Total Data Points}} = \frac{\lambda_i^1 + \lambda_i^2 + \cdots + \lambda_i^N}{N}$$

$$= \frac{\sum_{n=1}^{N} \lambda_i^n}{N}$$

Soft Clustering using GMM:

- Suppose we take K clusters.
- Randomly initialize K Gaussians. The i^{th} Gaussian is given by :-

$$\mathcal{N}(\mu_i, \Sigma_i)$$

- For each data points, check its belongingness to each of K Gaussians. The belongingness of data point x^n to i^{th} Gaussian is :

$$\lambda_i^{(n)} = \frac{\pi_i \, \mathcal{N}(x^n | \mu_i, \Sigma_i)}{\sum_{m=1}^{K} \left(\pi_m \, \mathcal{N}(x^n | \mu_m, \Sigma_m) \right)}$$

- Using belongingness, recalculate μ_i, Σ_i, π_i for each cluster i to maximize log likelihood :-

$$\sum_{j=1}^{N} \ln \left(\sum_{i=1}^{K} \left(\pi_i \, \mathcal{N}(x^j | \mu_i, \Sigma_i) \right) \right)$$

$$p\left(x^{(n)} | C_i\right) = p\left(x^n | \mu_i, \sigma_i\right) = \frac{1}{\sqrt{2\pi}\,\sigma_i} \exp\left(- \frac{(x^n - \mu_i)}{2\sigma_i^2} \right)$$

$$= \mathcal{N}(x^n | \mu_i, \Sigma_i)$$

- Log-likelihood:

$$l = \sum_{j=1}^{N} \ln P(x^j) = \sum_{j=1}^{N} \ln \left(\sum_{i=1}^{K} P(x^j \mid C_i) \pi_i \right)$$

$$= \sum_{j=1}^{N} \ln \left(\sum_{i=1}^{K} P(x^j \mid C_i) \pi_i \right)$$

$$\downarrow$$

$$\sum_{j=1}^{N} \ln \left(\sum_{i=1}^{K} \left(\pi_i \mathcal{N}(x^j \mid \mu_i, \Sigma_i) \right) \right)$$

Soft Clustering using GMM

- Suppose we take K clusters.
- Randomly initialize K Gaussians. The i^{th} Gaussian is given by: $\mathcal{N}(\mu_i, \Sigma_i)$
- For each data points, check its belongingness to each of K Gaussians.
- The belongingness (responsibility) of data point x^n to the i^{th} Gaussian is:

$$\lambda_i^n = \frac{\pi_i \mathcal{N}(x^n \mid \mu_i, \Sigma_i)}{\sum_{m=1}^{K} \pi_m \mathcal{N}\left(\frac{x^n}{\mu_m, \Sigma_m} \right)}$$

Belongingness of Data points to Different Clusters:

Belongingness of x^n to cluster i :-

$$\lambda_i^{(n)} = p(C_i|x^n) = \frac{p(x^n|C_i)\,p(C_i)}{p(x^n)} = \frac{\pi_i\, p(x^n|C_i)}{\sum_{m=1}^{K} \pi_m\, p\left(\frac{x^n}{C_m}\right)}$$

$$\lambda_i^{(n)} = p\left(\frac{C_i}{x^n}\right) = \frac{p(x^n|C_i)\,p(C_i)}{p(x^n)}$$

$$= \frac{\pi_i\, N(x^n|\mu_i, \Sigma_i)}{\sum_{m=1}^{K} \left(\pi_m N(x^n|\mu_m, \Sigma_m)\right)}$$

- $p(x^n) = p(x^n, C_1) + p(x^n, C_2) + \cdots + p(x^n, C_K) = \sum_{i=1}^{K} p\left(\frac{x^n, C_i}{}\right)$
- $p(x^n) = \sum_{i=1}^{K} p(x^n, C_i) = \sum_{i=1}^{K} p\left(\frac{x^n}{C_i}\right) p(C_i) = \sum_{i=1}^{K} \left(p\left(\frac{x^n}{C_i}\right)\pi_i\right)$
- $\pi_i = p(C_i) \longrightarrow$ mixing coefficient

Using Bayes's theorem, we calculate probability that x^n belongs to cluster i :-

$$p\left(\frac{C_i}{x^n}\right) = \frac{p\left(\frac{x^n}{C_i}\right) p(C_i)}{p(x^n)} = \frac{\pi_i\, p\left(\frac{x^n}{C_i}\right)}{\sum_{i=1}^{K} \left(\pi_i\, p\left(\frac{x^n}{C_i}\right)\right)}$$

$$p\left(\frac{x^n}{C_i}\right) = p\left(x^n \mid \mu_i, \sigma_i\right) = \frac{1}{\sqrt{2\pi}\,\sigma_i} \exp\left(-\frac{(x^n - \mu_i)^2}{2\sigma_i^2}\right)$$

$$= \mathcal{N}\left(x^n \mid \mu_i, \Sigma_i\right)$$

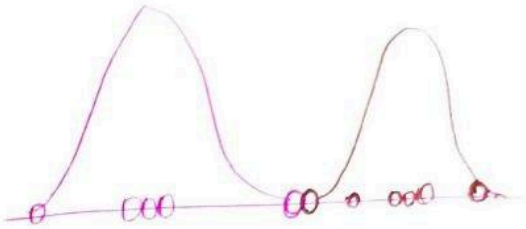
Belongingness of Data Points to different Clusters :-

- Suppose we take K clusters $C_1, C_2 \ldots C_K$
- Consider a data point x^n.
- $p(x^n)$ = probab. that x^n is in C_1 or prob that x^n is in C_2 or ..... prob. that x^n is in C_K.
- So $p(x^n) = p(x^n, C_1) + p(x^n, C_2) + \cdots p(x^n, C_K)$

$$= \sum_{i=1}^{K} p(x^n, C_i)$$

Soft clustering using GMM

- We take K clusters
- Randomly initialize K gaussians



- The i^{th} Gaussian is given by :- $\mathcal{N}(\mu_i, \Sigma_i)$

- For each data points, check its ~~performance~~ belongingness to each of the K gaussians.
  The belongingness of data point x^n to the i^{th} gaussian is

$$\lambda_i^n = \frac{\pi_i \, \mathcal{N}(x^n \mid \mu_i, \Sigma_i)}{\sum_{m=1}^{K} \left(\pi_m \, \mathcal{N}(x^n \mid \mu_m, \Sigma_m) \right)}$$

- Using belongingness, recalculate μ_i, Σ_i, π_i for each cluster i to maximize log likelihood.

$$\sum_{j=1}^{N} \ln \left(\sum_{i=1}^{K} \pi_i \, \mathcal{N}(x^j \mid \mu_i, \Sigma_i) \right)$$

Clustering using GMM

Suppose we take K clusters —

(i) Randomly initialize K gaussians

(ii) For all points, check which gaussian it belongs to

(iii) Based on the belongingness assign each data point to a cluster

(iv) Find μ_i, Σ_i from each cluster and update Gaussians

(v) Repeat 2-4 steps unless termination criteria is satisfied.

Lecture -21 Date: 27/09/2024

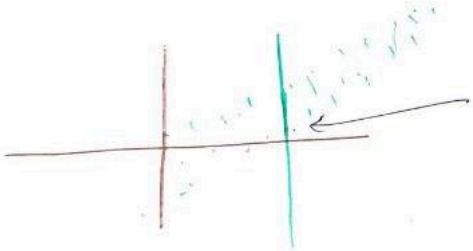
- Let's project data on line (x axis or y-axis).

If projected on line, data has better projection at line.

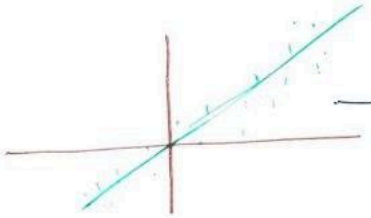
- I can find direction at which data has maximum possible spread.

- spread = max. variability for data.
  - This projection of data = can do tasks like classification.

- Project data on line to find direction



If i project on this line
it don't have maximum diversity after projection.



→ direction of maximum diversity.

We calculate covariance matrix by :-

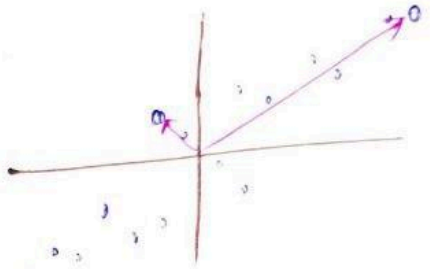
$$\begin{pmatrix} \text{var}x_1 & \text{cov}(x_1, x_2) & \text{cov}(x_1, x_3) \\ \cdot & \text{var}x_2 & \cdot \\ \cdot & \cdot & \text{var}x_3 \end{pmatrix}$$

Covariance → spread
tell ~~direction~~ of data

→ It is possible to find multiple direction.

→ Direction of 2nd max is : 2nd possible component.

- Covariance matrix : indicates spread of data points



eg :- If covariance matrix is :-

$$\begin{pmatrix} 9 & 4 \\ 4 & 3 \end{pmatrix}$$

Step 1 :- First take data

Step 2 :- Center it around origin

Step 3 :- Calculate covariance matrix

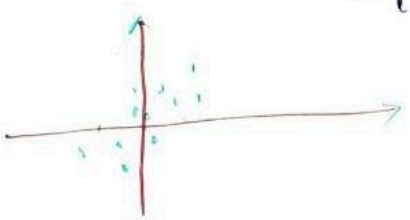
Step 4 :- Calculate eigen vectors of covariance matrix

Step 5 : Find corresponding eigen values

Step 6 :- Project all data points into that direction.

Step 2 :- Center it around origin

$$Z_i^{(N)} = \frac{x_i^N - \mu_i}{\sigma_i}$$ standardization



- To fix range when values vary → centering to origin.

↑ better direction possibility, feature at same scale. (not mandatory)

step 3:- Calculate covariance matrix :-

Calculation of eigen vector = max. distn of spread of data.

$$\boxed{\lambda x = \Sigma x} \qquad \begin{bmatrix} 9-\lambda & 4 \\ 4 & 3-\lambda \end{bmatrix} \qquad \Sigma = \text{covariance matrix}$$

$$(\Sigma - \lambda I)x = 0 \Rightarrow \text{gives eigen values}$$

put

↓

eigen vector

$$\begin{pmatrix} 9 & 4 \\ 4 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 11 \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

↓

Σ = Covariance matrix for 2D.

$\lambda_1 = 11 \quad \lambda_2 = 1$

↓

gives vectors

$\begin{pmatrix} 2 \\ 1 \end{pmatrix} \quad \begin{pmatrix} -1 \\ 2 \end{pmatrix}$ direction.

- For d-dimensional data, Covariance matrix would be $d \times d$.

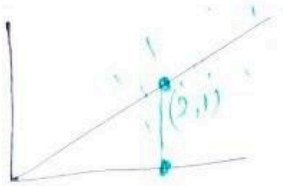
$d \times d$ = d eigen values, vectors

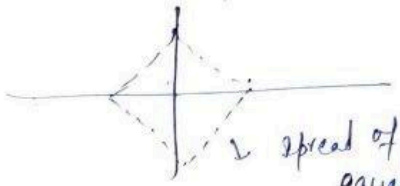
↓

max independent row, column is = d

rank = d

- How to draw line :-





Rank is less than 2 when λ are less than 2
det. = 0

$$\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \quad \det = 0$$

↓

one of the eigen value vector = 0

1 independent column.

Step 4 :- calculate eigenvector of covariance matrix

Co-linear data points = datas in only 1 direction

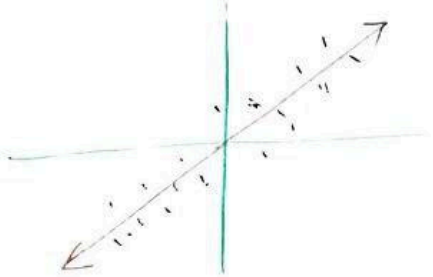
Step 5 :- Find corresponding eigen values

Step 6 :- Take dirⁿ of maximum eigen value

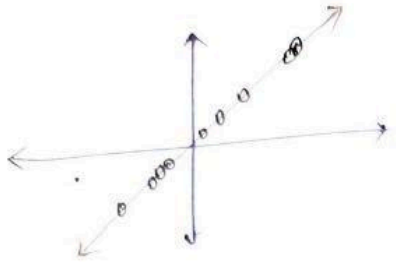
$$\Sigma = \begin{pmatrix} 9 & 4 \\ 4 & 3 \end{pmatrix}$$

$\begin{pmatrix} 2 \\ 1 \end{pmatrix}$ eigen vector

|| eigen value (magnitude)



Step 7:- Project all points into that direction



This projected data is called first principal component

- PCA can be used for data reduction.
- PCA basically gives max direction of spread.
- 5 dimensional data = 5 max. possible P.C.

↓

If we project data in direction of d_1, d_2 :-

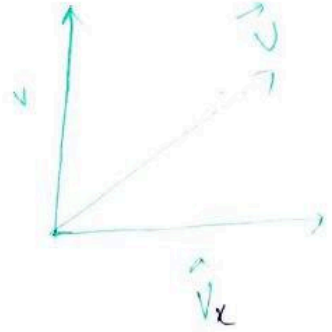
↓ gives 2D plot.

Composite features

$5D \rightarrow 2D$

- we might loose some data.

With PCA Components :- we do our tasks like : compression.



projection still :- $V\cos\theta$

$$V\cos\theta = \frac{|v|\,|v_x|\cos\theta}{|v_x|} = \frac{\vec{v}\cdot\vec{v}_x}{|v_x|}$$

Depending on data : I'll get P.C ⇒ not its features

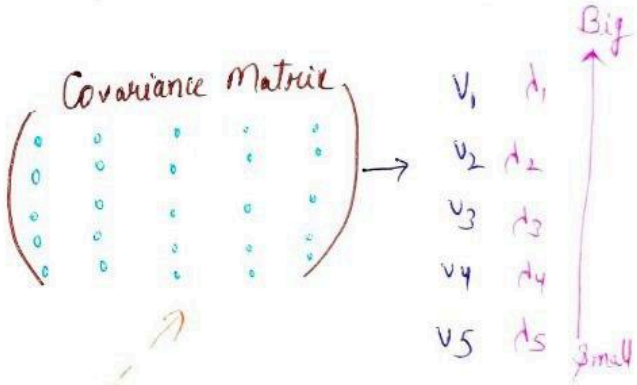
eg:- Diff P.C : If we measure height, weight in diff. Country
eg: Germany, India.

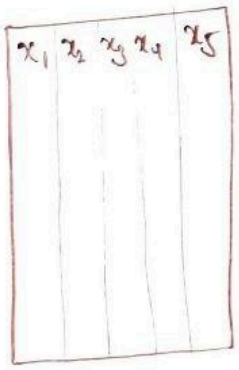
- Test data needs to be projected
  ↑
  P.C on the direction got from Training data

- Composite features = found from 5 features

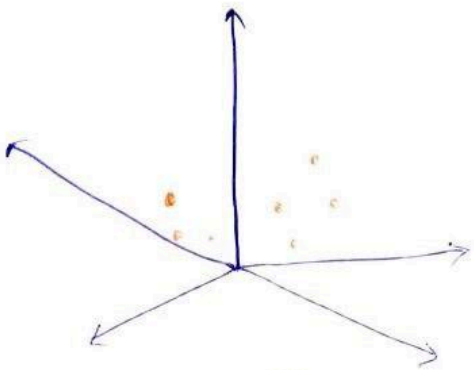
Eigen Stuff

Principal Components





Large Table



5 D plot

If I want to take
2 pc :—

①

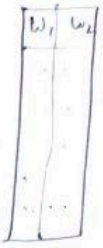
Eigen stuff

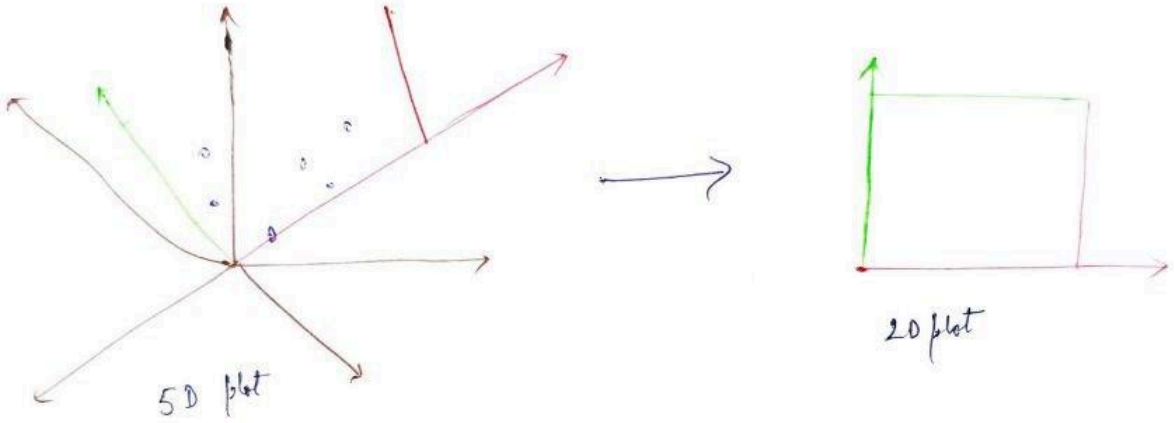
V_1 λ_1 Big

V_2 λ_2 small

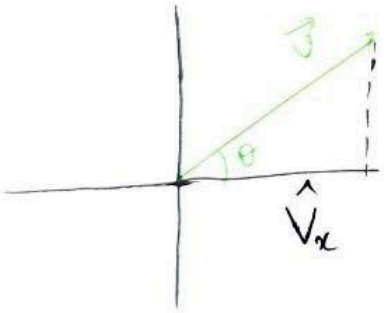
Small table

Reduced Dimension





$\vec{v} \cdot \hat{i} = |v||\hat{i}| \cos\theta$

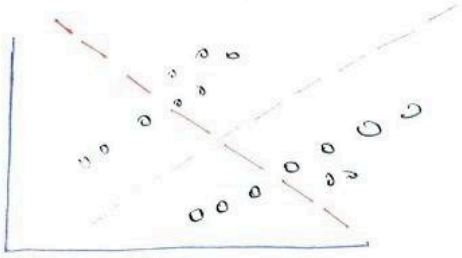


- We'll do standⁿ of train data, also of test data.
- Substract μ, σ of train data to test data.

  ↓

  to get stdⁿ of test data.
- We'll think test data comes from same data distⁿ as of train data.

Date : 27 sept 2024 Lecture : 22



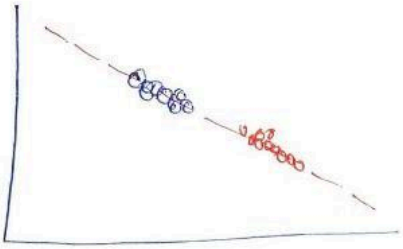
Which line would you like to project these data points to?

- As per PCA it should be Grey

If these 2 sets belong to different classes.

⇒ Red line will be preffered.

- Projection on red line gives Better class seperability and dense clusters corresponding to every class.



Goal: Project all points to a direction such that we get best projection for classification.

For Labelled data → Classification task

↓

don't only seek for best project, seek:

↳ → minimum inner cluster variance

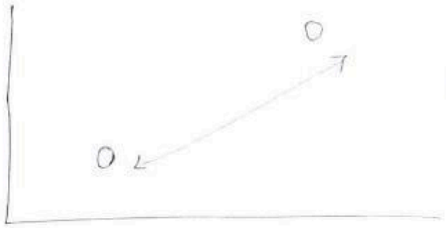
→ max. seperability b/w clusters.

- (i) Collapse all blue points of class, to one point & collapse all yellow class, points to one point.
- (ii) Take these two points as far as possible.

LDA [It is Linear Discriminant Analysis] It can be used for classification task.

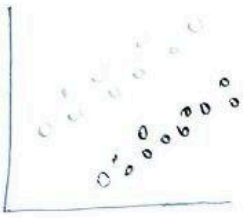
→ we'll try to learn direction.

→ w is unit vector projection.



single points
Try to seperate these

- x is data point (vector)
- w = unit vector
- If x is projected on direction of w.
- Projection is given by : $Z = w^T x$
- Goal :- Find w such that projection of data points towards w are well seperated in terms of their class labels.
- Mean is obtained from vector points, mean is also vector.



If it is 2-class problem with classes C_1 and C_2, mean of classes are:

$$\mu_1 = \frac{1}{N_1} \sum_{\forall i \in C_1} x_i \quad , \quad \mu_2 = \frac{1}{N_2} \sum_{\forall i \in C_2} x_i$$

N_1, N_2 : No. of data points in classes C_1, C_2.

. Mean of clusses from the projected data points are :-

$$\mu_1' = \frac{1}{N_1} \sum_{\forall i \in C_1} z_i \qquad \mu_2' = \frac{1}{N_2} \sum_{\forall i \in C_2} z_i$$

$$\Downarrow$$

$$\mu_1' = \frac{1}{N_1} \sum_{\forall i \in C_1} w^T x_i \qquad \mu_2' = \frac{1}{N_2} \sum_{\forall i \in C_2} w^T x_i$$

Goal :-
- Maximize distance b/w projected class means μ_1' & μ_2'
- Minimize within class variance. (make denser cluster for each class)

$$\mu_1' = w^T \mu_1 \quad , \quad \mu_2' = w^T \mu_2$$

Means after projection

- As points are both +ve and -ves so Square of mean needed.

Covariance of classes before and After projection :

- Covariance of classes from original data points are Σ_1, Σ_2.
  " " " projected data points Σ_1', Σ_2'.

$$\Sigma_1' = w^T \Sigma_1 w \qquad \Sigma_2' = w^T \Sigma_2 w$$

Distance in 2d = $(\mu_1' - \mu_2')^2$

As we go for higher dimensions, x is d dimension so

- Euclidian distance b/w μ_1' and μ_2' is :—

$$\boxed{\beta = w^T (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T w}$$

$$\beta = (\mu_1' - \mu_2')^T (\mu_1' - \mu_2')$$

$$\beta = (w^T \mu_1 - w^T \mu_2)^T (w^T \mu_1 - w^T \mu_2)$$

$$= w^T \underbrace{(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T}_{S_B} w$$

$$\beta = w^T(\mu_1 - \mu_2)(\mu_1 - \mu_2)w$$

$$= w^T S_B w$$

β : distance b/w projected mean.

S_B = Between class covariance

So we have to maximize $w^T S_B w$

Task 2 :- Minimize within class variance (make denser cluster for each class)

→ Covariance of classes from original data points : Σ_1, Σ_2

" " " " " " " " : Σ_1', Σ_2'

$$\Sigma_1' = w^T \Sigma_1 w \qquad \Sigma_2' = w^T \Sigma_2 w$$

Minimize :

$$\gamma = \Sigma_1' + \Sigma_2'$$

$$= w^T \Sigma_1 w + w^T \Sigma_2 w$$

$$= w^T (\Sigma_1 + \Sigma_2) w = w^T S_c w$$

S_c = within class variance

Maximize : $\beta = w^T S_B w$

Minimize : $\gamma = w^T S_c w$

Achieving Goal 1, 2 :-

Equivalent to maximizing : $\frac{w^T S_B w}{w^T S_c w}$

- equivalent to maximizing $w^T S_B w$ such that $w^T S_c w = 1$ (Constant)

Two ways to maximize :- division, substract them.
choosed one : division.
Langrange multiplier is one used for optimization techq.
So numerator is maximized by keeping denomenator constant.

- Using lagrange multiplier for optimization problem.

↳ Solution for w direction is obtained from eqn :-

$$S_c^{-1} S_B w = \lambda w$$

- S_c, S_B = matrics
- λ = scalar
- w = vector

Above eqn can be in form :-

$$Aw = \lambda w$$

w that solve eqn are eigen vector of $A = S_c^{-1} S_B$

w = direction where projected data gives max. b/w class variance, min within class variance.

How many eigen vector possible of $A = S_c^{-1} S_B$?

It can be shown that only one eigen vector are possible for $S_c^{-1} S_B$.

- only 1 eigen vector given direction of maximum spread, minimum covariance.
- one w is obtained we project all data on direction of w ⊢

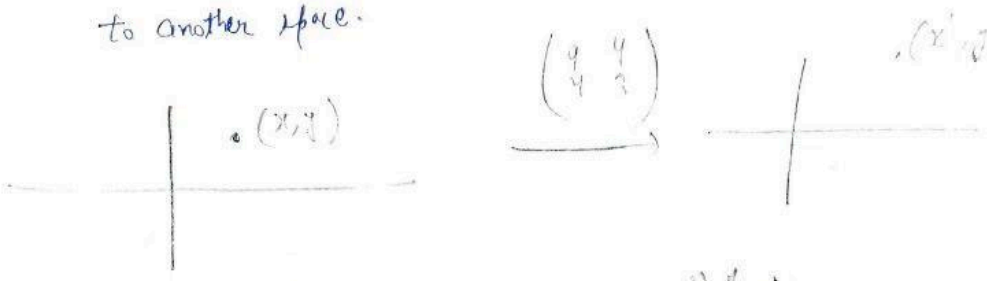
$$Z = w^T x$$

PCA vs LDA

PCA = unsupervised LDA : supervised

Linear Transformation ⊢

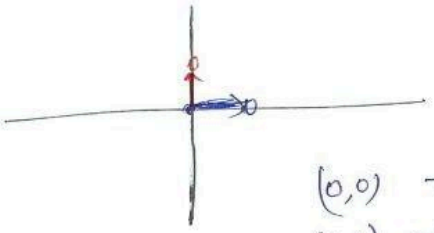
- Transformations (mapping) of points (vectors) from one space to another space.

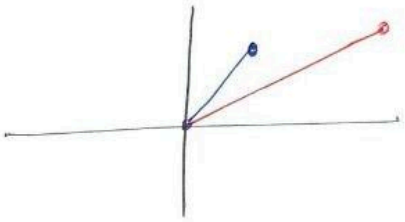


L.T often represented as Matrix

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 9 & 4 \\ 4 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

$$x' = 9x + 4y$$
$$y' = 4x + 3y$$





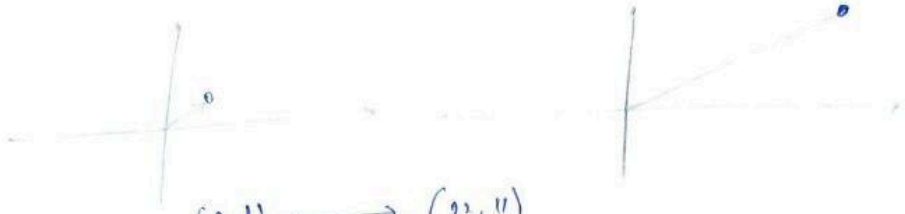
$(0,0) \rightarrow (0,0)$
$(1,0) \rightarrow (9,4)$
$(0,1) \rightarrow (4,3)$

For these points direction of vector changes due to linear transformations.

- Linear transformation doesn't cause any direction change.
  - ↳ Direction of eigen vectors of transformation matrix.
  - → Corresponding eigen value indicates scaling factor.

eg:  = cylinder rotated around its axis gives direction of eigen vector in ↑ vertical direction

- Eigen value : Amount of scaling after Linear Transformation.



$(2,1) \longrightarrow (22, 11)$

$x' = 9x + 4y$

$y' = 4x + 3y$

$\begin{pmatrix} 9 & 4 \\ 4 & 3 \end{pmatrix}$ = Transformation matrix

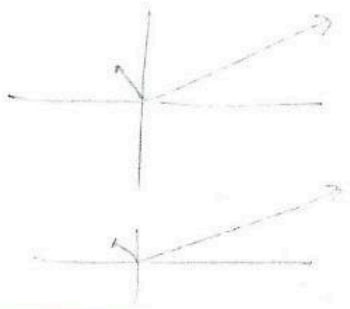
x = point which get linearly transformed.

$\theta = \tan^{-1}\left(\frac{1}{2}\right) \longrightarrow \theta' = \tan^{-1}\frac{11}{22}$

→ Direction doesn't change due to linear transformation.

Another set of points, direction doesn't change of this transformation.

$(-1,2) \longrightarrow (-1,2)$



- For these two directions, linear transformations doesn't change direction.

→ only scale vectors.

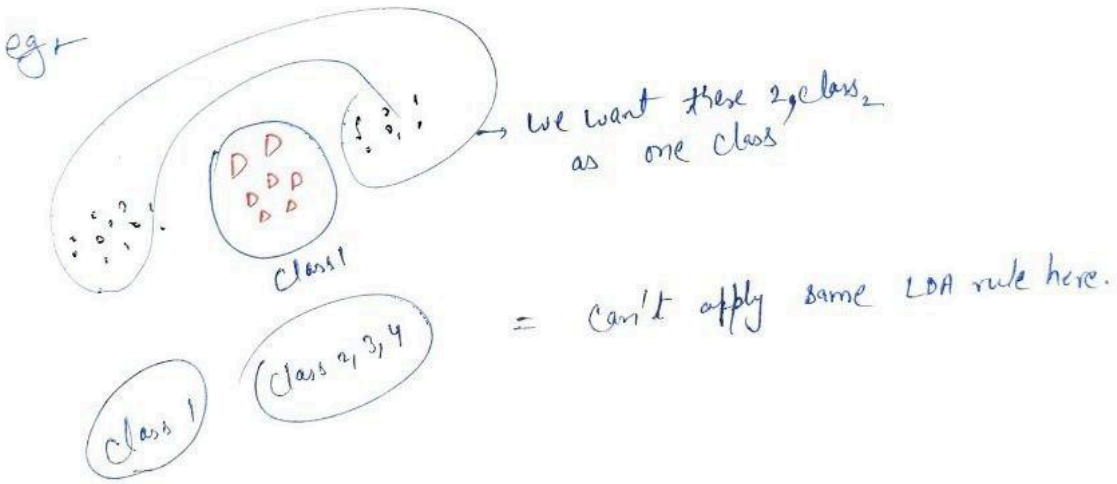
→ Distⁿ of eigen vectors of our transformation matrix.

- Corresponding eigen values indicates scaling factor

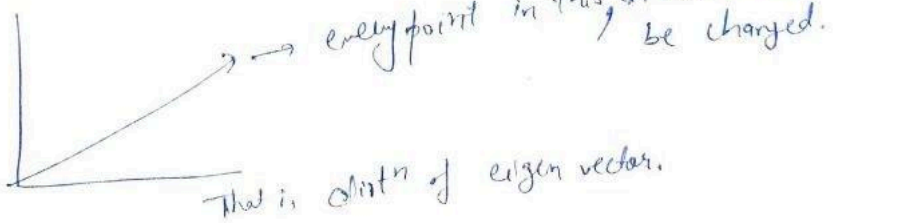
In LDA :-

- Get modified value after projection →
  ↓ Then train model for it , ↓ train for test data

- LDA is applicable for 2 clusters.



- LDA :



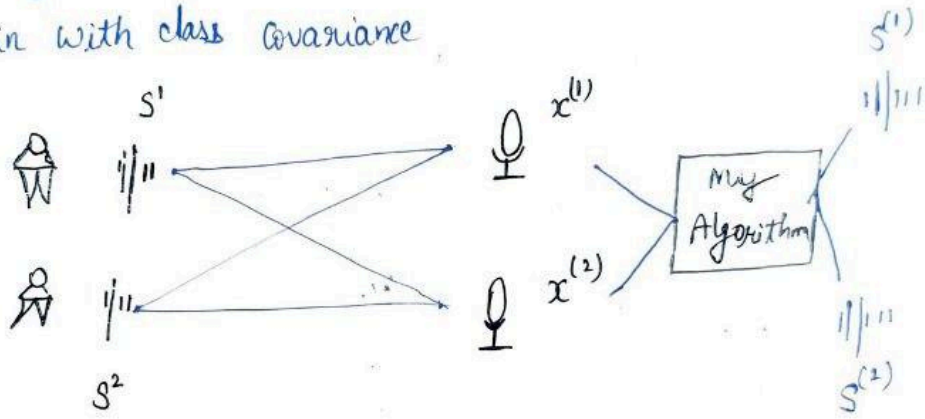
Date: 1 Oct. 2024 Lecture : 23

PCA : Unsupervised

LDA : Supervised

- PCA: Finding direction of max variation of data

LDA: Finding direction of max between class covariance and min with class covariance



Independent Component Analysis

$x = AS$

$x_1 = a_{11} s_1 + a_{12} s_2$

$x_2 = a_{21} s_1 + a_{22} s_2$

$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$

A = mixing matrix

$\therefore s_i$ is Non-Gaussian

$x = As$ $\qquad s = Wx$

- Let $P_s(S)$ be pdf of random variable s.

Let $P_x(x)$ be pdf of random variable x.

It can be proved that :-

$$P_x(x) = P_s(s)\,|W|$$

- Signal from each person is independent of others (random variable)

→ s_i and s_j are independent random variables.

- $x = As$
- $s = Wx$

$$W = \begin{pmatrix} w_1^T \\ w_2^T \\ \vdots \\ w_N^T \end{pmatrix}$$

So if we have N persons ←

$$P_s(S) = P_s(S_1, S_2 \cdots S_N) = P_s(S_1)\,P_s(S_2) \cdots P_s(S_N)$$

$$= \prod_{i=1}^{N} P_s(S_i)$$

It can be proved that

$$P_x(x) = P_s(S)\,|W|$$

If there are N points :—

$$P_x(x) = \left(\prod_{i=1}^{N} P_s(s_i) \right) |W|$$

$$P_x(x) = \left(\prod_{i=1}^{N} P_s\left(w_i^T x\right) \right) |W|$$

$$\begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_N \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & & \vdots \\ w_{N1} & w_{N2} & & w_{Nm} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}$$

$$\begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_N \end{pmatrix} = \begin{pmatrix} w_1^T \\ w_2^T \\ \vdots \\ w_N^T \end{pmatrix} x$$

$$s_i = w_i^T x$$

Goal :— To find W such that $P_x(x)$ is maximized.

Maximum likelihood estimation.

- If we have D data points $x^1, x^2 \cdots x^D$:—

likelihood :—

$$l(w) = \ln \left(\prod_{j=1}^{D} P_x\left(x^j\right) \right)$$

putting $P_x(x^j) = \left(\prod_{i=1}^{N} P_s(w_i^T x^j)\right)|W|$

$\downarrow$ in

$$l(w) = \sum_{j=1}^{D} \ln\left(\left(\prod_{i=1}^{N} P_s(w_i^T x^j)\right)|W|\right)$$

MLE using gradient descent to maximize :—

$$l(w) = \sum_{j=1}^{D} \ln\left(\left(\prod_{i=1}^{N} P_s(w_i^T x^j)\right)|W|\right)$$

where $q'(s) = \frac{d\,q(s)}{d\,s} = P_s(s)$

α = learning rate

update rule :—

$$W = W + \alpha \nabla_W (l(w))$$

Assuming $P_s(\cdot)$ to be an uniform distribution $\in [0,1]$ for data point j, update rule is :—

$$\nabla_w(l(w)) = \begin{pmatrix} 1-2q(w_1^T x^j) \\ \vdots \\ 1-2q(w_N^T x^j) \end{pmatrix} (x^j)^T + (W^T)^{-1}$$

- Through several gradient update steps, we can find W
- Once W is found.
  ↓
  we can find $s^{(j)} = W x^{(j)}$ for j^{th} training data.

→ So we can find source values s^j like this for each training measurement x^j

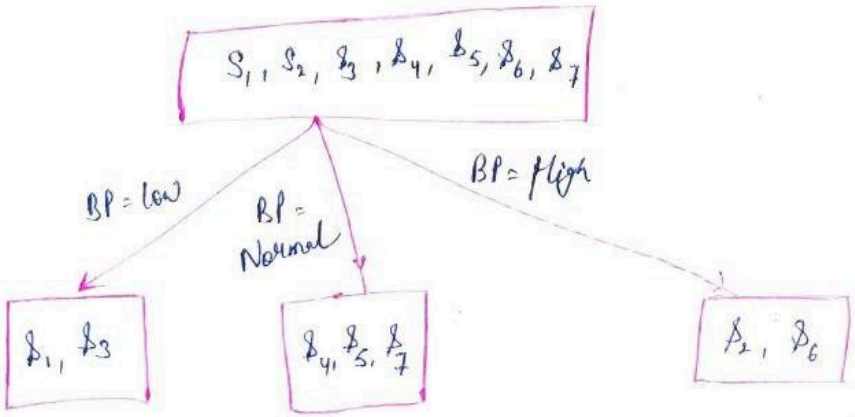
Sample number	Appetite	Weight	BP	Class
S_1	Low	Normal	Low	No Anemia
S_2	Low	Low	High	Anemia
S_3	Normal	Low	Low	Anemia
S_4	Low	Low	Normal	No Anemia
S_5	Normal	Low	Normal	Anemia
S_6	Normal	Normal	High	Anemia
S_7	Normal	Normal	Normal	No Anemia

Training data

↓

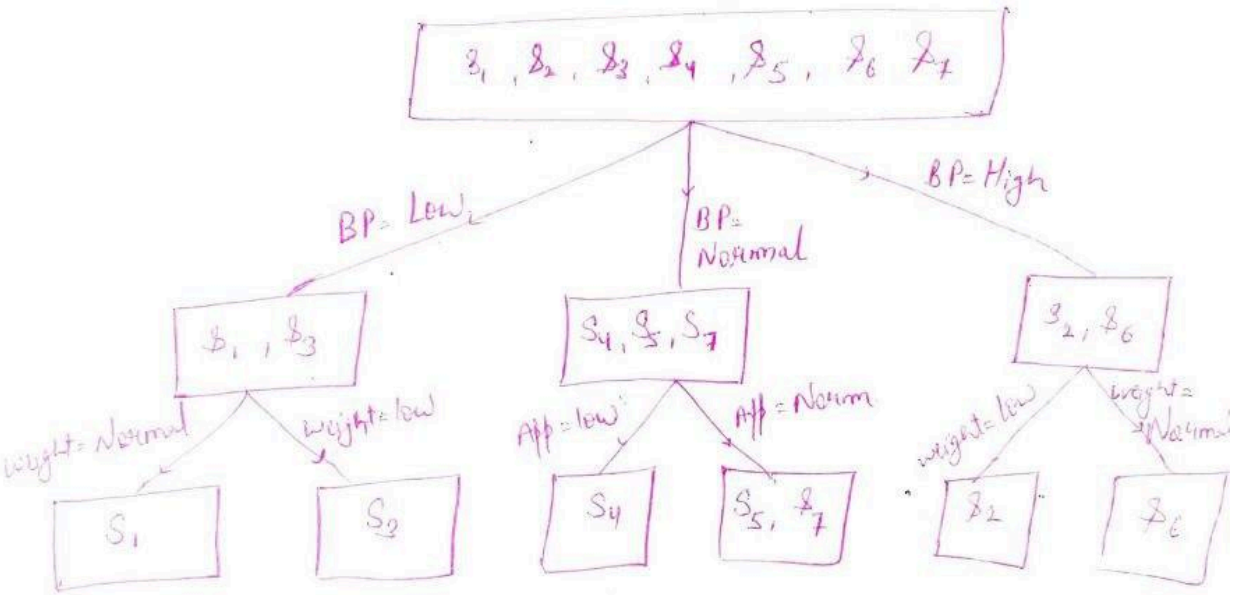
If new sample comes with normal appetite, normal weight, low BP. Find if person has Anemia ?

- **Constructing Decision Tree** :- I split the training samples into smaller bags using some criteria based on feature value.



which feature and which value should be choosen ?

- Splitting : based on some criteria based on feature value.
- Repeat the process in each of smaller bags, may be using some other feature.



Q2: How long should we continue splitting ?

Goal :- For new patient I want to classify patient having anemia or not having Anemia.

Tree satisfied : Training is done.

→ collection of nodes connected by branch = D.T

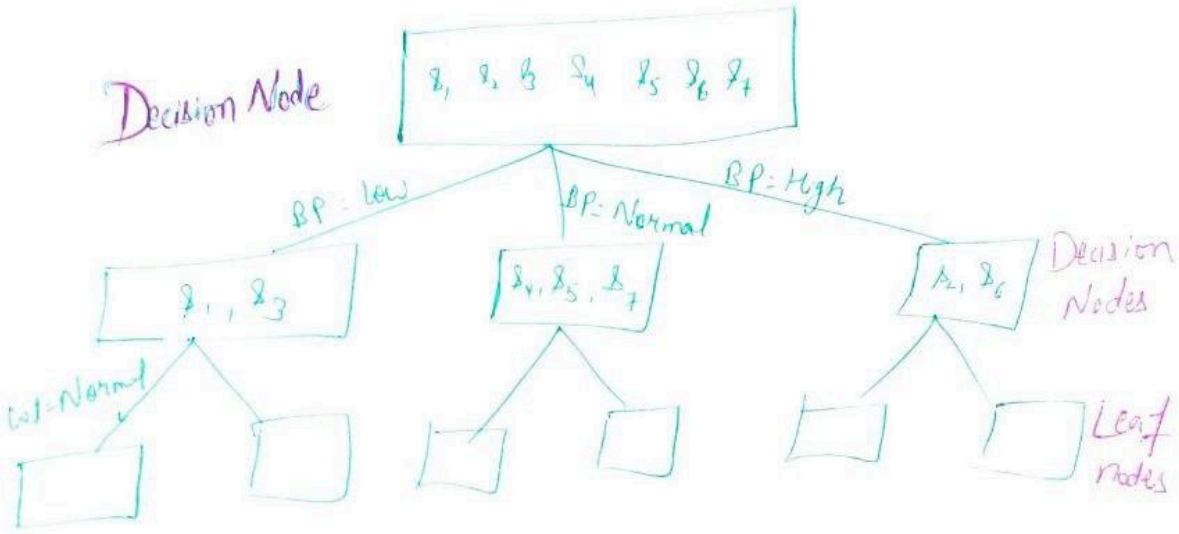
- Leaf node may not lie on same height

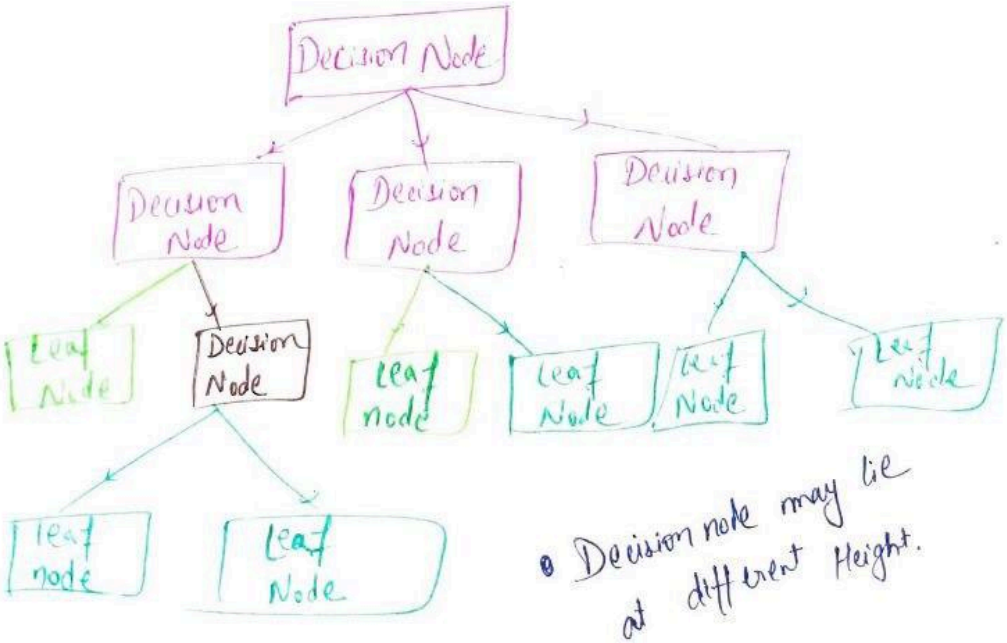
Data Reduction is done for :—

(i) To compress data, dimensionality reduction

(ii) D.R also helps to do downstream tasks.

. Leaf node doesn't get splitted further.





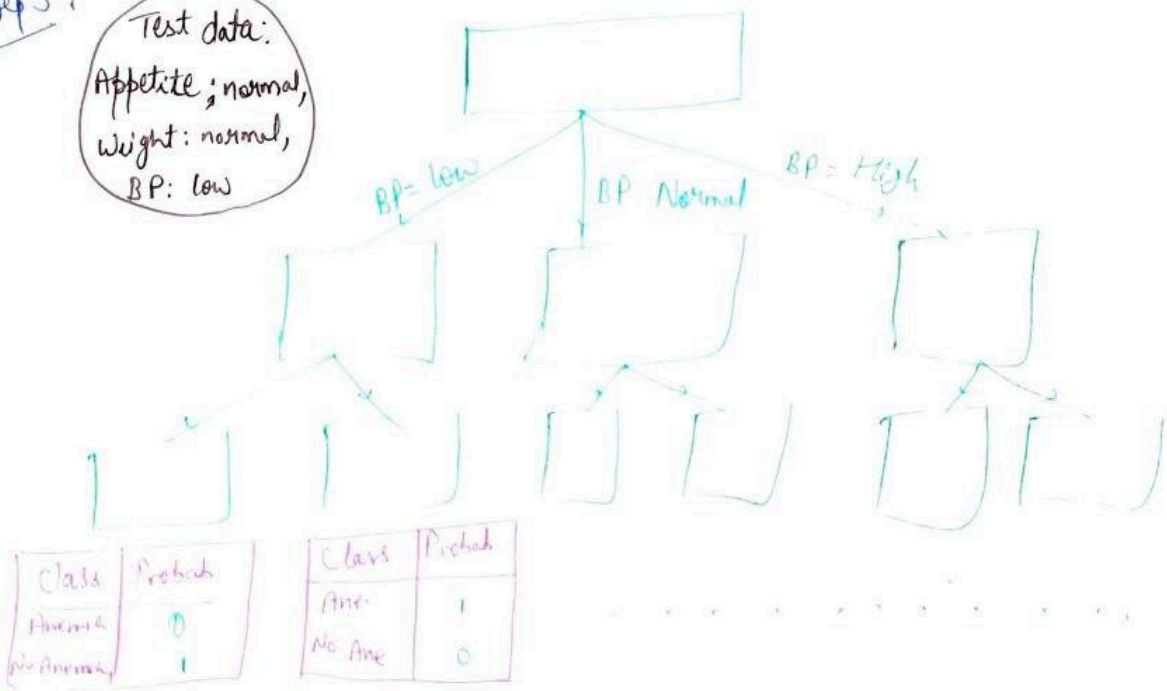
• Decision node may lie at different Height.

Step 1:- Find out probabilities at each leaf node.

Step 2:- Push down test data through trained tree using the decision rule at each decision node.

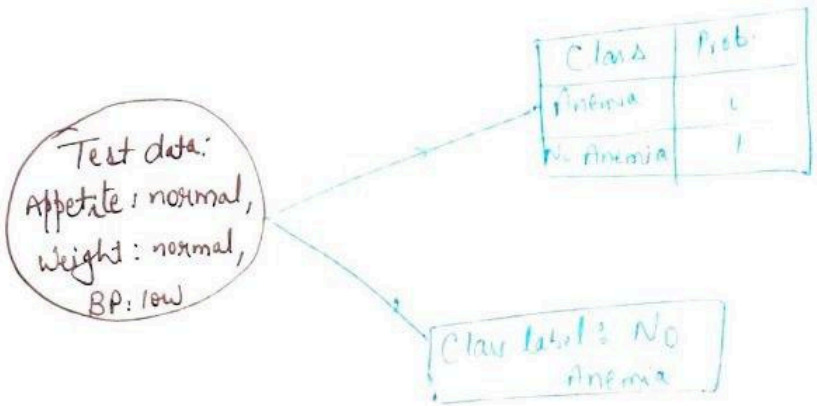
Step 3:-

Test data:
Appetite: normal,
Weight: normal,
BP: low



Step 4: Check the class probability in leaf node where test data arrives.

Inference Using a Trained Decision Tree



- for S_1 probability of Anemia = 0

  for S_5, S_7 = 0.5

- 4th node on left can't be assigned Hard Class

↑ Soft Class = Probability

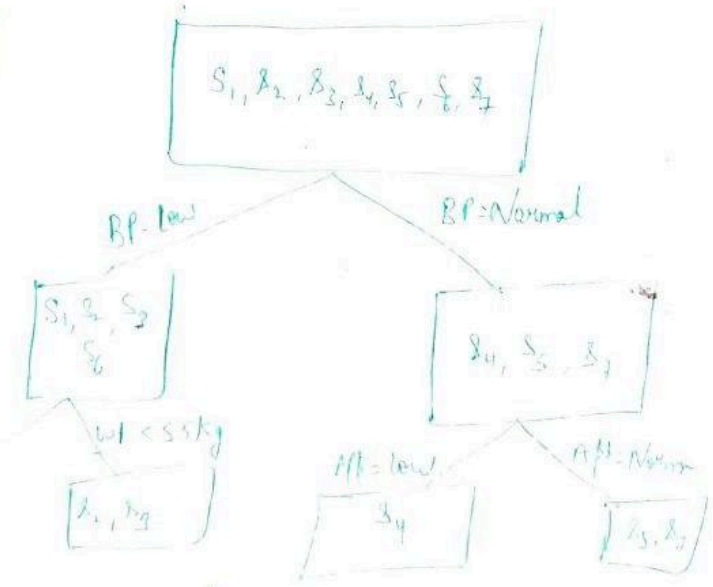
Some Standard Practices :-

- We usually prefer trees with :-
  - ↳ low depth
  - ↳ smaller no of nodes
  - ↳ usually construct binary DT

  ] Inductive Bias of DT

- Each parent node has atmost 2 children.

Sample no	Appetite	Wt	BP	Class
S_1				
⋮				
S_7				

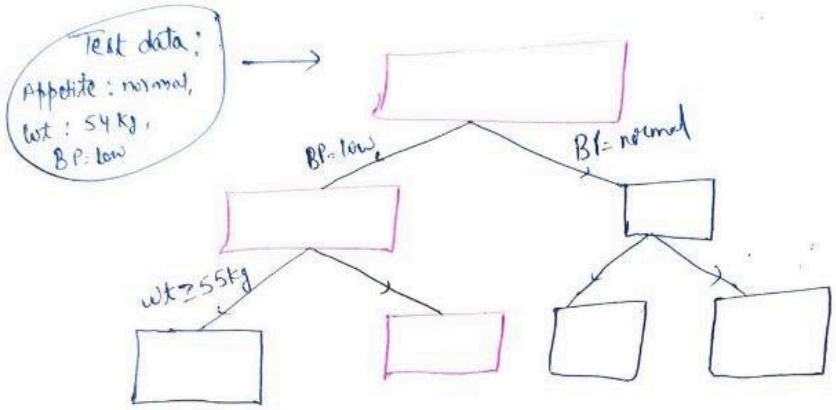


I can either stop growth of tree here or can continue further.

↓ Calculate class probabilities at leaf nodes.

Based on tree we can now test data.

So if Test data : Appetite : normal,
Weight : 54 kg, BP = low



- More is depth of Tree : More comparison needed
  We want low depth Tree.
- Inductive Bias of DT : lower depth
  lower no of node
  Binary DT

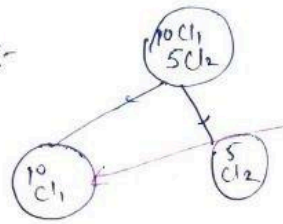
Binary has Criteria of 1 feature

Goal is for Pure Node

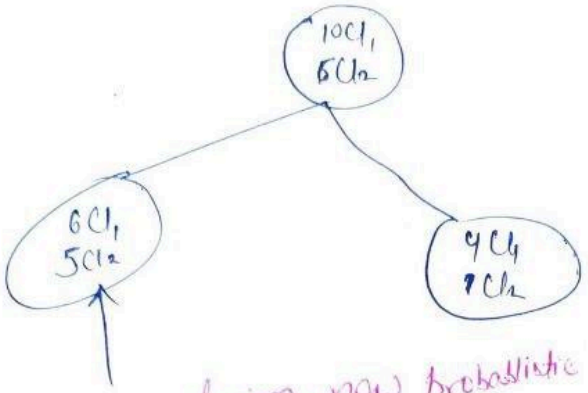
↓

Ideally we want child to be of 1 class, while splitting nodes.

eg :-



We can take decision surely for it

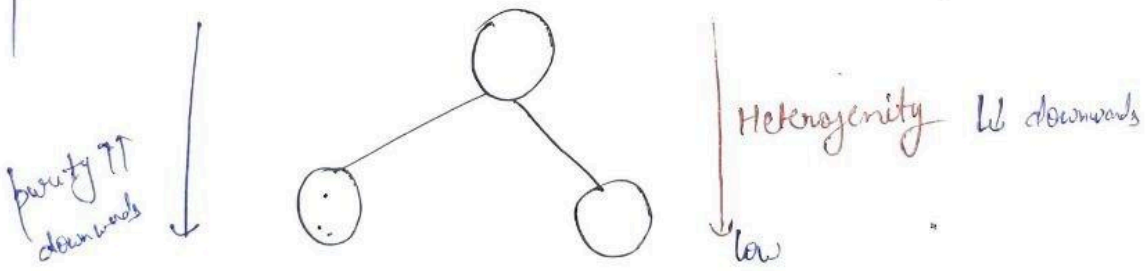


There is confusion, now probabilistic answer

→ Ambiguity, slight change in data lead to ambiguous decision for testing.

• In D.T after split we want node to be as Pure as possible.

• Class level at child node should be as Low as possible.



Heterogenity is fun of class level.

• Heterogenity increases as we move root to child level.

• We should choose feature which is most homogenous.

Information $\propto \frac{1}{p}$

Entropy : measure of randomness.

→ Something obvious is of less interest $\propto \frac{1}{\text{Probability}}$

→ inversely proportional to probability of it.

If certainty :- $P = 1$ info $= \frac{1}{1} = 1$

$P = 0.1$ Info $= \frac{1}{0.1} = 10$ Info

P ↓ Info ↑↑

→ If P_i is very low, info increases very high So taking log to min

event = K

$$I_k = \log \frac{1}{P_K}$$

- If random variable = X

Avg = $x P(x)$, for discrete = $\sum x P(x)$

Average Info = Entropy

$$= \sum_{K=1}^{N} P_K I_K$$

$$\text{Avg Info} = \sum_{k=1}^{N} P_k \log \frac{1}{P_k}$$

$$\boxed{\text{Avg Info} = -\sum_{k=1}^{N} P_k \log P_k}$$

$P_k = 1 \quad = -\sum P_k \log P_k = 0 =$ no randomness

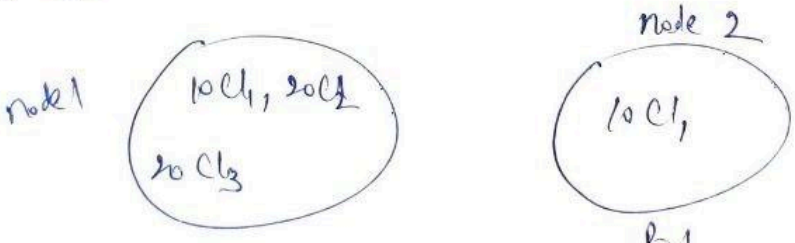
Case 2 : $p_1 = 0.5, \quad p_2 = 0.5$

$- 0.5 \log 0.5 \quad - 0.5 \log 0.5$

$= -\log 0.5 = \log 2$

↑↑ events = more entropy

If nodes are like ←



Probability of $C_1 = \frac{10}{50}$

Probability of $C_2 = \frac{20}{50}$

Pro. of $C_3 = \frac{20}{50}$

more impurity → more entropy

$P = 1$

entropy = 0

entropy of pure node = 0

From Top to Bottom in DT = Entropy ↓↓

⇒ We want to reduce Entropy.

We can measure Heterogenity by :
Ginni Index
Variance measure.

Entropy ↑↑ = Heterogenity ↑↑ = Info loss

Lecture - 25 , Nidhi Soni, Date : 17/10/2024

Now we have two questions for Decision-tree :-

(i) Which feature to use for splitting ?

(ii) Until when to split

- After training we want each leaf node to contain data of only one class.
  - → This is called a Completely homogenous (pure) node.
  - → In such a case, confidence in classification is high.

- From root to leaf nodes, we want nodes to become more and more homogeneous.

→ For both categorical, continuous type of feature : Same strategy for making tree.

Top most bag = Root node

- For good classification : Leaf node should be as homogenous as possible.

→ Ideally node should be as pure

↓

else some amount of error possible.

↓

child node should become more homogenous to parent.

Pure node = entropy = 0

- For classification : child is more homogenous than parent when we move downwards.

→ Entropy is one of the measure of Homogeneity.

$$E = p \ln \frac{1}{p} \quad \text{when } p=1 \quad E = 1 \cdot \ln \frac{1}{1} = 0$$

→ Gini Index is also measure of homogeneity.

$$G_I = 1 - \sum_{k=1}^{K} p_i^2$$

If p_j is true class pure node, 1

$$G_I = 1 - \sum 1 = 1 - 1 = 0$$

- Gini is not 0 ∴ Impure node
- ID3 = Binary DT
- CART = Any DT

- Goal : In child node entropy is as low as possible
- Calculate entropy based on splitting of all possible feature Binary

D.T :-

$$E = E_1 + E_2$$

Measuring Homogeneity of a Node : Entropy

- Consider a node that contains :
  - n_1 no. of data from class 1
  - n_2 " " " class 2
  - ⋮
  - n_K no of data from class K.

Let $N = n_1 + n_2 + \cdots n_K$

- In the node, probability
  - → of class 1 is $P_1 = \frac{n_1}{N}$
  - → of class 2 is $P_2 = \frac{n_2}{N}$
  - ⋮
  - → of class K is $P_K = \frac{n_K}{N}$

- Entropy of Node :

$$E = \sum_{i=1}^{K} P_i \ln \frac{1}{P_i}$$

If node is pure, data from only one class.

- In pure node, we have data of only one class j.

$P_j = 1$

$P_i = 0$ if $i \neq j$.

So $E = 0$

for pure node, entropy is 0.

- As impurity increases, entropy ↑↑.

So entropy is measure of impurity.

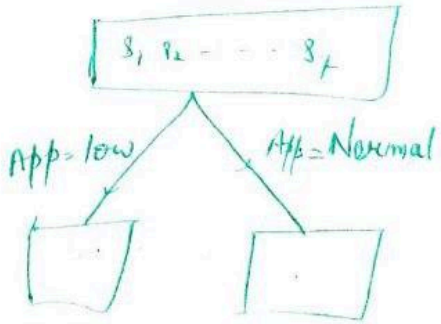
$$G_1 = 1 - \sum_{i=1}^{k} p_i^2$$

- $G_1 = 0$ for pure node, Ginni impurity = 0
- Impurity ↑↑ ⇒ Ginni ↑↑

[After training we want each leaf node to contain data of only 1 class.
called completely homogeneous (pure) node.
∴
In such case confidence in classification is high

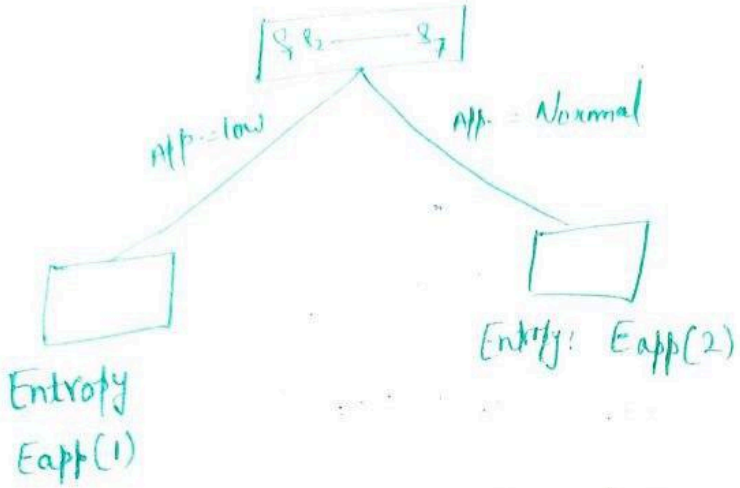
Q1 :- Which feature and which value should be choosen for Splitting a Node ?

Sample no.	App.	Col.	BP	Class
S_1				
S_2				
S_3				
S_4				
S_5				
S_6				
S_7				



First considering feature = Appetite

Possible values : low, Normal



Hypothetically split the node using these values.

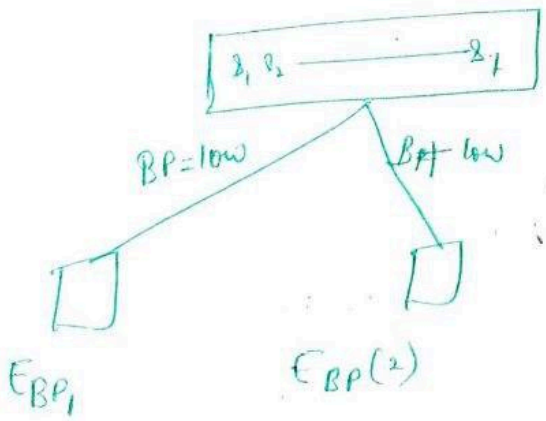
- Calculate total entropy of child nodes

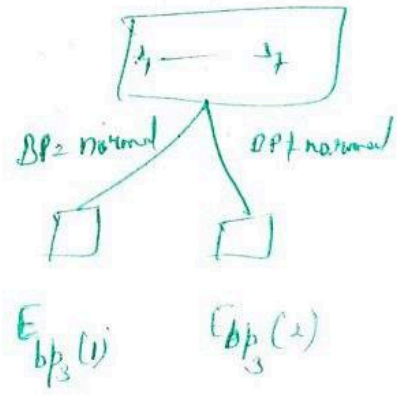
$$E_{app} = E_{app}(1) + E_{app}(2)$$

Case 2: Now Let's consider next feature BP

Possible values : low, normal, high.

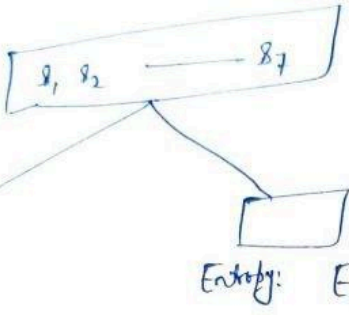
Multiple splits are possible.





$$E_{bp_1} = E_{bp_1}(1) + E_{BP_1}(2)$$

$$E_{bp3} = E_{bp_3}(1) + E_{bp_3}(2)$$



Entropy: $E_{bp_2}(2)$

Entropy: $E_{bp_2}(1)$

$$E_{bp_2} = E_{bp_2}(1) + E_{bp_2}(2)$$

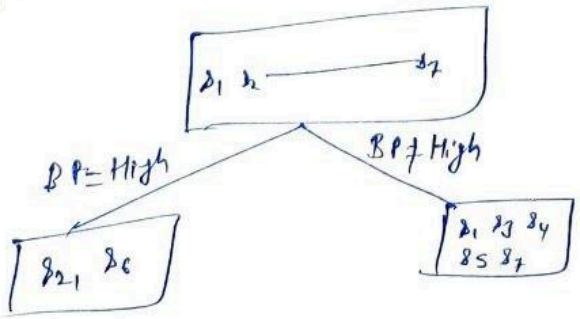
- We have total entropy of child nodes for various hypothetical splits:
  - E_{app} using feature appetite
  - E_{wt} " " weight.
  - E_{bp_1} using first split point (low vs ≠ low) on feature BP

    Similarly E_{bp_2}, E_{bp_3}

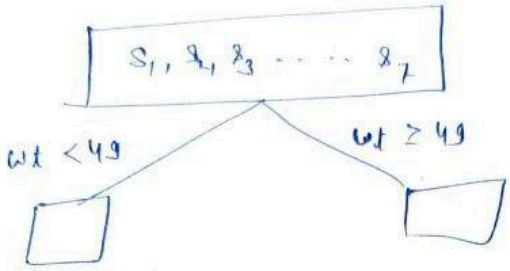
→ Find out hypothetical split that leads to <u>smallest total</u> entropy of child nodes.

→ Suppose E_{bp_2} is smallest among the above.

- We'll split the node using first split point (High vs ≠ High) on feature BP.



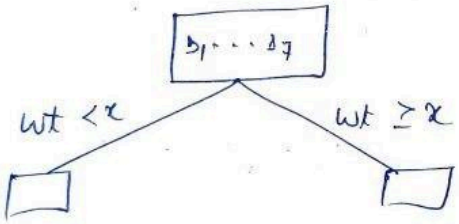
What happens for Continuous feature?



Total entropy of child nodes = E_{49}

For hypothetical split using weight, try hypothetical split with every possible value of weight from dataset.

- Possible values of wt :- 47, 49, 50, 51, 58, 61, 63



And so on...

- Compare these total entropies with total entropies from other features to find out best split point.

Advantage of DT : • Calculations are highly parallelized.
• Less time consumption.

Disadvantage : Computationally Intensive.

- For Continuous feature : Split for all possible weights.

more split

In some cases DT can be divided into interval based bins :

eg :- [49 51) [51 — 63] [64 70]

Binning

- For floating point too approach is same.

- How long should we continue Splitting :—

There may be diff. Criteria :—

(a) Split a node until entropy of node $< \epsilon$ (a pre-defined value)

threshold to stop.

(b) split a node until node contains $< n$ no of data points

eg : if 10 data left after splitting

(c) split a node until height of tree reaches h

(a pre-defined value)

Drawback of D.T :- A single tree may not be a good classifier or regressor.

- If 1000 features, all features might not be used for splitting, not looked into DT splitting for all features.

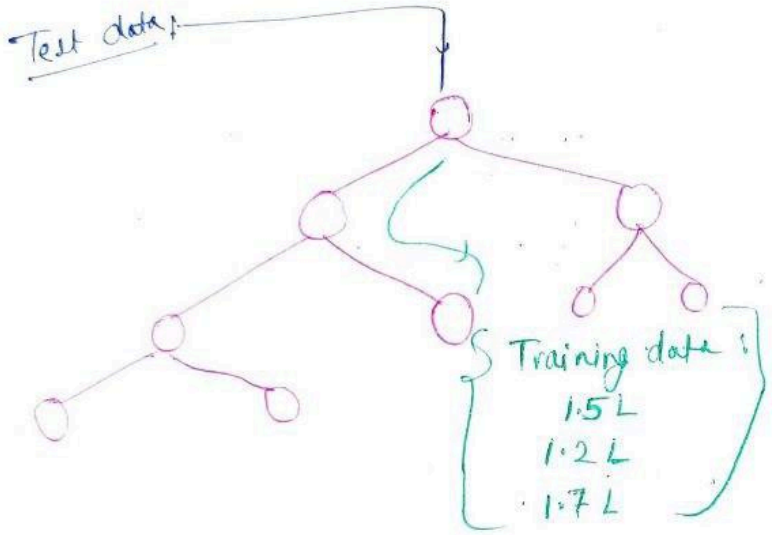
While doing Regression = we are doing Pattern Identification.

while doing Classification: judging data

If node is pure = variance should be low.

If node is impure = variance ↑↑

I want to find salary of Employ,



Take average, median of these ↑ to calculate.

Info gain = Entropy of parent − $\sum$ Total entropy of children

Should be maximized = coz entropy at child should be low.

- A tree may not exploit all inp features of data.
- Splitting of nodes may be done using unimportant features.
- Poor quality of tree

Tree to forest :

- Take several D.T construct Forest.
- voting = taking all possible classification

→ Perform voting from ensemble (forest) to make decision about specific classification (or regression)

In order to get good splitting :- Need to have Diversity.

→ We want to grow set of DT diff from each other.

else biasness may be there.

(a) Diversity from perspective of data :-

- All the trees must not be grown with same data.
- Each tree should look at diff. training data.

less bias of each tree towards entire training data.

(b) Diversity from perspective of features :-

→ Diff. nodes should look at diff features space of data and then split.

→ Allowing all features to compete to split nodes.

Lecture - 26 , Date : 18/10/2024

- We want heterogenity as less as possible.
- All tree may grow parallely.
- If we sample from 100 data items as 10

  ↓

  data loozes.

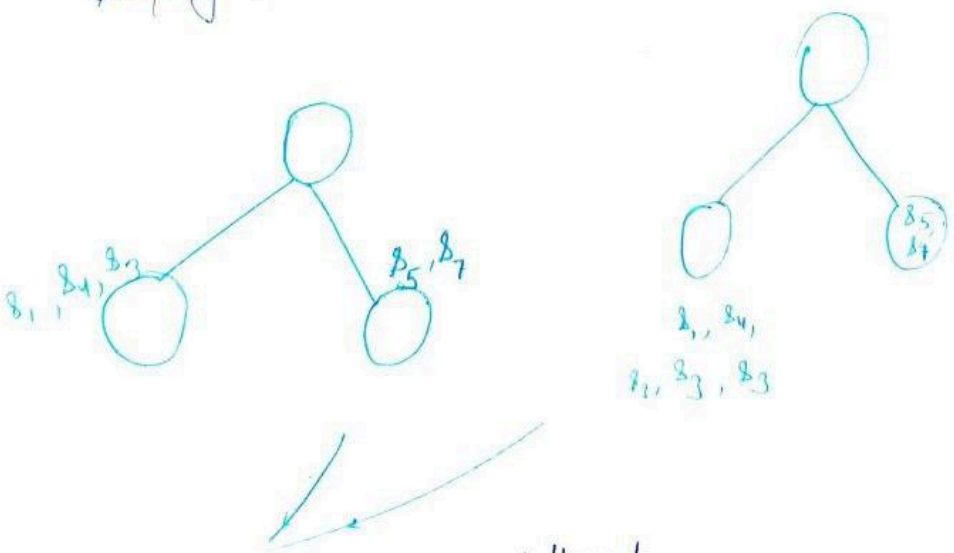
- Randomly choosed 70 data = Tree 1

  Randomly choosed 70 data = Tree 2.

Overfitting = less data

> Randomly choosing 100 sample with replacement : Bootstraping

- what if samp data is replaced x no. of time during sampling :—



entropy for both trees are different.

But info doesn't change

- Feature selection will be affected depending on repeating data or replacement.
- Suppose, there are N training data points and we want to grow T no of trees:—
  - Take a random subset of N_s training data to grow Tree 1
  - Take another random subset N_s training data to grow Tree 2
  - ⋮
  - Take another random subset N_s training data, to grow Tree T.

Diversity from the Prospective of Data

- N training data points and we want to grow T no of trees.
- Training data for growing diff trees would be different:—

↓

if $N_s << N$: each tree will be grown with small training data,

↓

more chance of overfitting.

eg: If 10 participants, 1 singer is excellent in soul music, else are pop, hip hop etc singer.

- But all should get chance
- every time 1 singer wins : soul music ⟶ So

- Dividing data features into small - small _ pairs then concluding by comparision.
- Reducing feature into smaller parts : More Quality come out.

Bootstrap Sampling : suppose we want to grow T trees with N training data ($N = 8$).

Sample no.	f_1	f_2	f_3	Class
S_1				
S_2				
:				
S_8				

- For growing tree 1, randomly choose N samples with replacement.
- Let samples :- $S_5, S_2, S_3, S_6, S_3, S_1, S_2, S_3$

$$D_1 = \{ S_5, S_3, S_2, S_3, S_6, S_3, S_1, S_2 \}$$

D_1 = called bootstrap sample for growing tree 1

Similarly, for Tree 2, bootstrap sample =

$$D_2 = \{ S_2, S_1, S_2, S_4, S_4, S_1, S_5, S_8 \}$$

Similarly for T trees, we have bootstrap samples

- $D_1, D_2, \dots D_T$

(2) Diversity from Perspective of Features :-

- During growth of forest, trees should exploit as many features as possible.
- In a D.T, every feature has to compete with every other feature to be choosen for splitting.

if m = no of feature to choosed

→ n = no. of trees

$m = \sqrt{n}$ $\quad m = \sqrt{10} = 3$

- Quasi-Random : Bootstrapping
  Splitting is random

→ Choosing feature :-

- This becomes tough competition
- If there are a few dominant features, several feature may never get selected for splitting a node.
- Such feature would not be utilized in decision making.

⇒ We want a node splitting process where every feature gets a reasonable chance to be selected for splitting.

- Random forest mostly used in → medical, Bioscience areas.

→ All trees can be grown in parallel.

- Split point different in all trees.

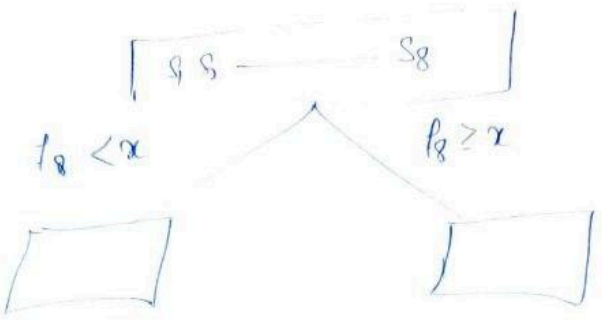
→ Particular feature may be best than another.

[$f_8 < x_1$] — same feature might have diff. split
[$f_8 < x_4$]

Splitting a Node →

S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8

- Suppose there are features $f_1, f_2 \dots f_{10}$ in my data.
- For splitting node, 1st randomly choose m no of features (without replacement) out of f_1 f_2 - - f_{10}
- Suppose m = 3 and I choose f_3, f_7, f_8.
- Find out best feature out of only f_3, f_7, f_8 using your criteria. (entropy, ginni impurity etc).
- Let split point be x on feature f_8.



Each node effectively look into a subset of features (a feature subspace)

→ Based on subspace, a split point is chosen.
Competition among feature reduces.

→ This randomness makes tree diverse.

[This tree is called Random Decision Tree.
Growth is continued until a node meets termination criteria.]

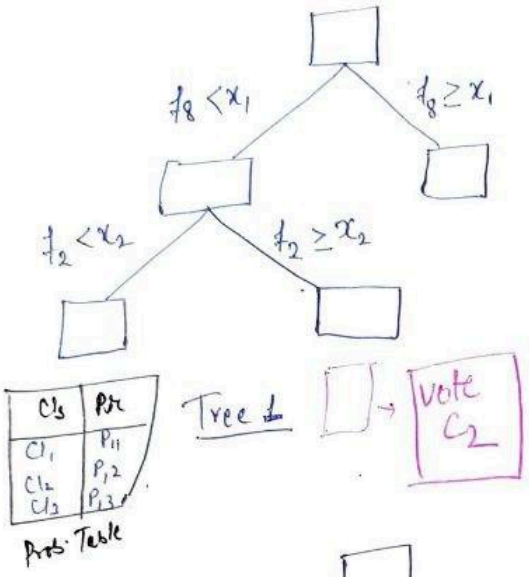
Random forest :-

- Decide :-
  - → How many trees (T) are to be grown
  - → How many features are to be selected randomly for splitting a node (m)
  - → Node splitting criteria (Entropy, Gini Impurity, variance etc)
  - → Termination criteria for splitting a node.

- For i = 1:T
  - → Create bootstrap sample for Tree i.
  - → Grow Tree i using technique mentioned before
  - → Calculate class probabilities at each leaf node.

Random Forest

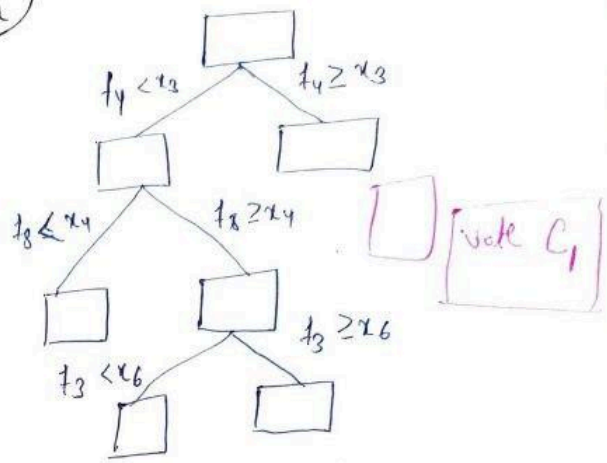
Test data



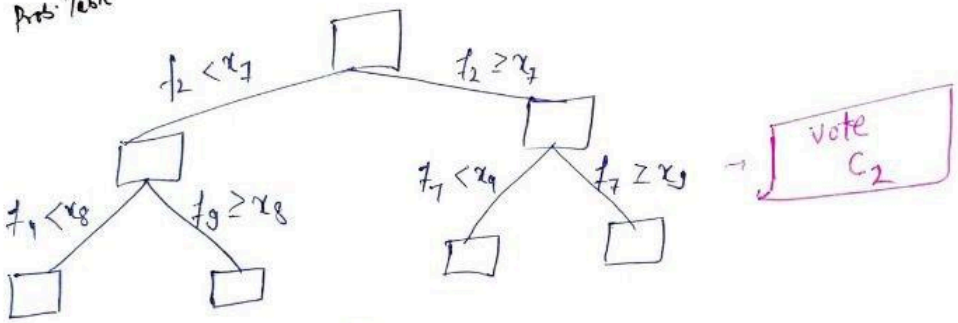
Cl_s	P_{jc}
Cl_1	P_{11}
Cl_2	P_{12}
Cl_3	P_{13}

Prob. Table

Tree 1 → vote C_2



Tree 2 → vote C_1



Tree 3 → vote C_2

→ Calculate class probabilities at leaf nodes

→ Push test data through all trees

→ At each tree, test data will reach a particular leaf nodes.

! Find the class with highest probability at these nodes.

Class with highest vote is the class for test data.

Class	vote
C_1	1
C_2	2
C_3	0

→ Class of test data is Class 2.

At each tree, test data will reach a particular leaf node. Find class probabilities at those nodes.

Considering all trees, find avg class probabilities :-

Class	
C_1	$\frac{P_{21} + P_{71} + P_{91}}{3}$
C_2	$\frac{P_{22} + P_{72} + P_{92}}{3}$
C_3	$\frac{P_{23} + P_{73} + P_{93}}{3}$

→ Class with largest avg. class probability is assigned to test data.

Class of test data is C_3.

Random Forest :- Bagging

→ Each tree in in Forest is called a 'weak learner'.

→ Each tree is grown independently.

- It can be shown statistically about 66% of training samples will be included in growing trees.

→ other samples are called out of bag sample.

Performance of random forest depends on 2 quantities :-

- Strength : indicates the abilities of trees for classification / regression
- Coorelation : Similarity among trees, lower better.

R.f is one of the bagging technique.

- not learning all data some part of data = weak learner.

. 34% out of bag sample.

. 66% sample used.

→ Error rate in R.F converges.

. Variable importance : Importance of features

Lecture : 27 , 22/10/2024

Boosting

RF was not for all nodes.

→ Each tree is creating a hypothesis.

→ Each hypothesis gives decision.

Bagging → Ensemble learning

Boosting →

→ RF :- We put equal weightage to each sample

• not all training data gonna correctly classify.

↓

There is impure data : incorrect result.

In Boosting :- We put more weightage to incorrect classified data

DT grow serially in Boosting then we check which is correctly classified, which is not!

→ RF all trees growing parallely.

→ In Boosting growing serially → Based on that weightage o/p.

↓

here Stump are independent to each other

↓

Stumps can grow parallely.

→ Boosting can't grow parallel as whole.

Boosting :-

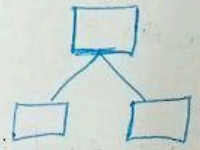
Sample no.	f_1	f_2	...	f_{10}	Class
S_1					C_2
S_2					C_1
⋮					⋮
S_8					C_1

we have 8 training samples.

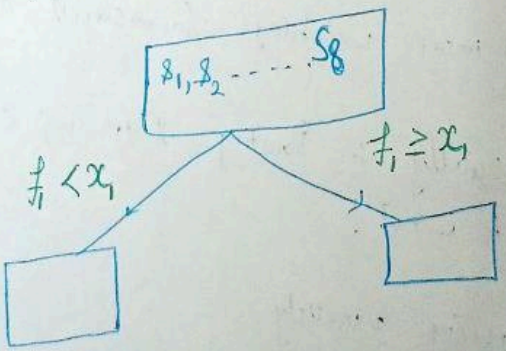
→ Assigning equal weights to each sample initially.

→ with these data we grow stump.

→ Stump is kind of decision tree.
(with a root, 2 leaves)



Step 1:- Find out best split point of feature f_i.
Let split point be x_1



Cl_3	P_{3L}
Cl_1	P_{11}
Cl_2	P_{12}
Cl_3	P_{13}

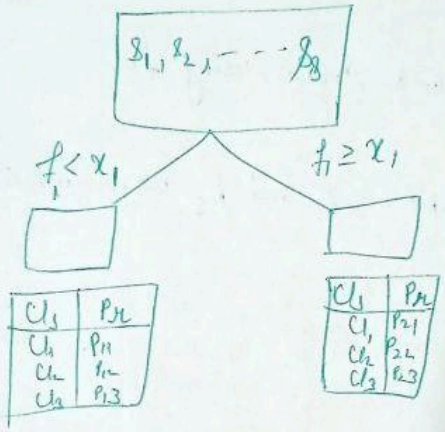
Cl_3	P_{3R}
Cl_1	P_{21}
Cl_2	P_{22}
Cl_3	P_{23}

Step 2:- Do hypothetical split.
Calculate class probabilities at leaf nodes

Step 3 : Push all training data through the stump and see how many training data is misclassified.

Step 4 : Calculate Gini Index / Entropy of Stump.

Let it be g_1.



Sample no	f_1	f_2	....	f_{10}	Class	Weight
S_1					C_2	$\frac{1}{8}$
					C_1	$1/8$
S_2					C_1	$1/8$
S_3					C_2	$1/8$
					C_3	$\frac{1}{8}$
S_4						
S_5					C_1	$\frac{1}{8}$
S_6					C_2	$1/8$
S_7					C_1	$1/8$
S_8						

Step 5 : Calculate Gini Index of stump. Let g_2.

↑

Similarly find best split point of feature f_2.

Let split point be x_2

→ do hypothesis split, calculate class Probabilities at leaf nodes, push training data downwards, find how many misclassified?

Step 6: So on until f_{10}.

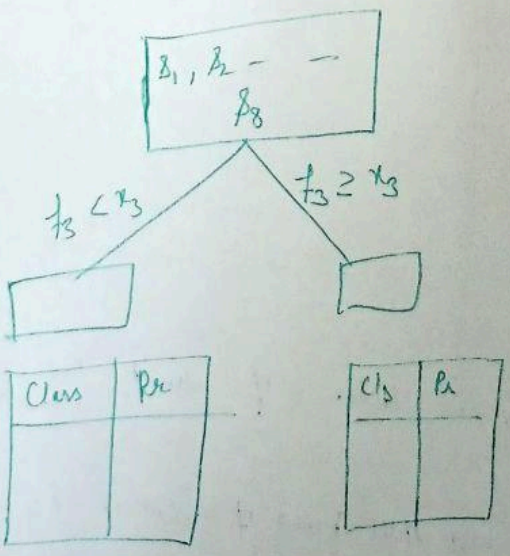
Gini Indices of ten stumps (grown using $f_1, f_2 \cdots f_{10}$) are $g_1, g_2 \cdots g_{10}$ respectively.

- Find out stump with lowest Gini Index.

Let it be stump grown using f_3.

- Assume training samples s_3 and s_5 were misclassified by first stump.

Sample no	f_1	f_2	.. f_{10}	Class	weight
s_1					
s_2				c_1	$1/8$
s_3					$1/8$
s_4				c_2	
s_5					
s_{10}					



Total error for stump = Sum of weights of misclassified samples $= \frac{1}{4}$

$\frac{1}{8} + \frac{1}{8} = \frac{1}{4}$

- Low total error ⟶ more imp. stump.
- Importance of a stump = $\frac{1}{2} \log \left(\frac{1 - \text{total error}}{\text{total error}} \right)$

> Importance ↑↑ = error ↓↓
> Imp ↓↓ = error ↑↑

- Incorrectly classified samples by this stump should be treated with more emphasis so that they can be correctly classified by future stumps.
  ↓
  Boosting.
- Before growing next stump :—
  → increase the weights of incorrectly classified samples
  → Decrease the weights of correctly classified samples.
- Updated wt. of incorrectly classified samples :—

$$W_{new} = W_{old} * \exp(\text{importance of stump})$$

updated wt of Correctly classified samples :—

$$W_{new} = W_{old} * \exp(-\text{importance of stump})$$

If new value is more than 1, we need to normalise
↓
so divide by sum of all new wts.

→ grow next stump using new wt.

we'll not calculate normal ginni index — now weighted ginni index

Earlier = $g = 1 - \sum_{i=1}^{N} p_i^2$

weighted ginni = $1 - \sum_{i=1}^{N} w_i p_i^2$

put more emphasis on incorrectly classified data.

entropy (old) $e = p_i \log \frac{1}{p_i}$

entropy (new) = $e' = w_i p_i \log \frac{1}{w_i p_i}$

p_i = probability of particular class.

$\Rightarrow$ $\frac{\text{sum of wt belonging to class}}{\text{sum of wt of all class.}}$

Step 6: Normalize updated weights

eg:- Normalizing wt → weighted more for wrongly classified.

	old	new
S_1	1/8	0.17
S_2	1/8	0.17
S_3	1/8	0.37
⋮	⋮	0.37
S_8	1/8	0.17

→ Normalize = 0.17 / (sum of all new wts)
0.37 / (sum of all weights)
⋮

Criteria of Growing boosting :—

(a) till 50 stump
(b) Total error < threshold

[while growing new stump using same procedure as that of previous stump

[while growing 10^{th} stump
  → we will cover 9^{th} stump
  → we can also consider all earlier 9 stump.

Finding best split point
↓
use weighted Gini impurity

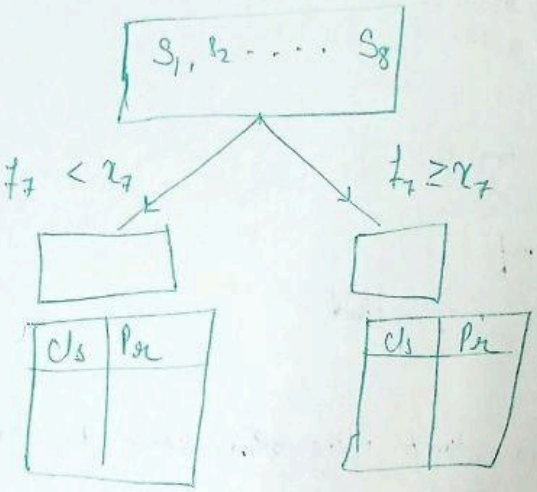
$$g = 1 - \sum_{i=1}^{N} w_i p_i^2$$

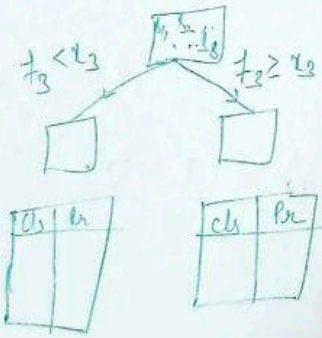
- using best feature and split point, grow the stump :—
  Let feature be f_7 and split point to be x_7

- Recalculate weight of samples and normalize.

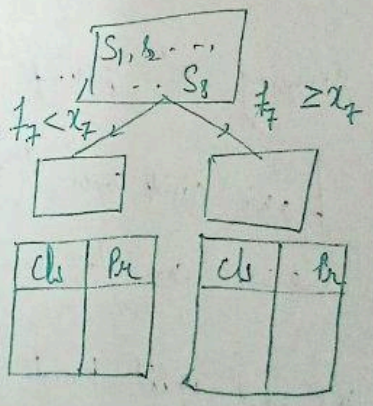
→ Grow next stump with normalized updated weights.

→ Continue growing more stumps until termination criteria is reached.

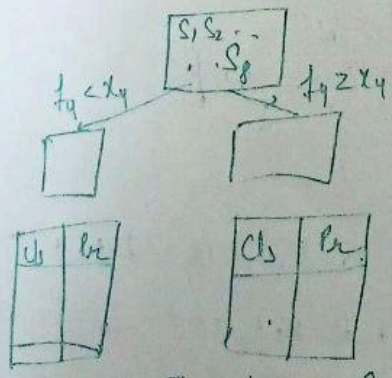




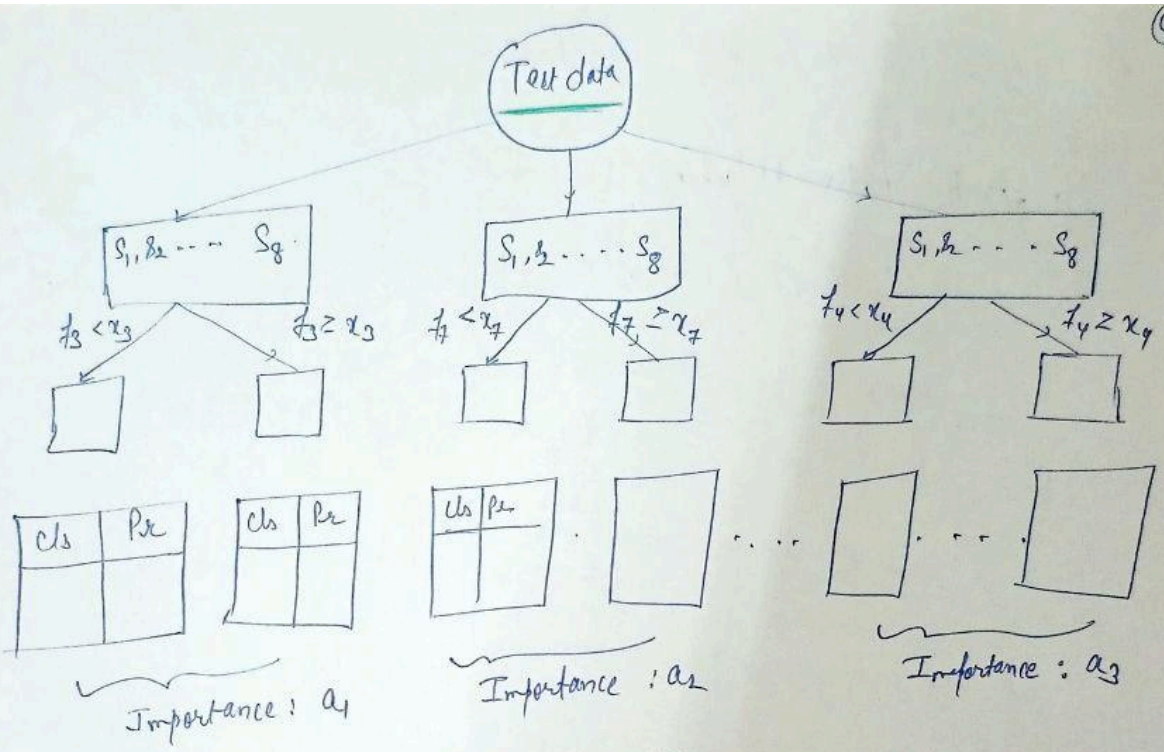
Importance: a_1



Importance: a_2



Importance: a_3



Class vote: C_3

Class vote: C_1

Class vote: C_1

Calculate total Importance score of each class :—

Class	Total Importance score
Cl_1	$a_2 + a_3$
Cl_2	0
Cl_3	a_1

- variants of AdaBoost : XGBoost

Support Vector Machine :-

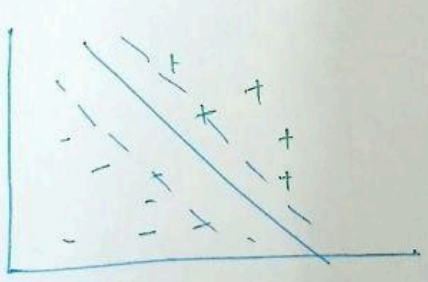
Let's say 2 class classification problem :-

Linear classification.

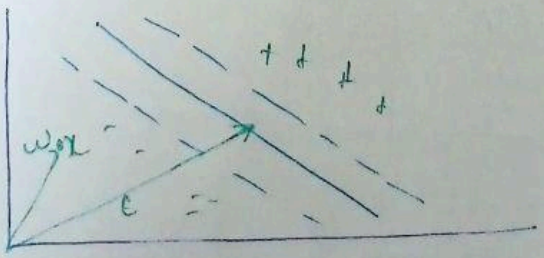
→ Find a line decision boundary among these class points.

. Infinite decision boundary are possible.

St. line : Linear Decision Boundary.



Find DB that minimizes error.



Find D.B whose margin max (ϕ_1, ϕ_2)

→ we want DB to be in mid of class boundary / margin.

- x is Point in any class : vector.

→ Take projection of x in direction of w.

$\vec{x} \cdot \vec{w} = d$ = Scalar quantity = distance from origin

Date - 24/10/2024 Lecture - 28

Goal :- How to draw Linear Boundary

Initially in SVM, trying to solve linear decision boundary.

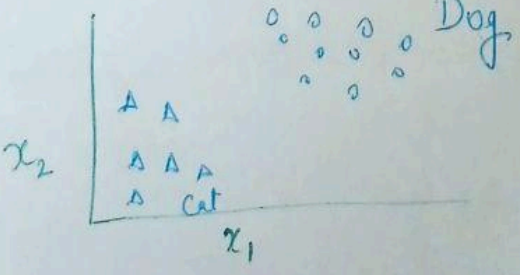
→ Linear classification.

- We want good seperation of points so that they are far apart from D.B.
- Slight change in feature level (Ht, wt here) should not change D.B classification.

∴ There may be many hypothesis possible, from this Hypothesis spce

↓

Choose hypothesis with good seperation for class points.

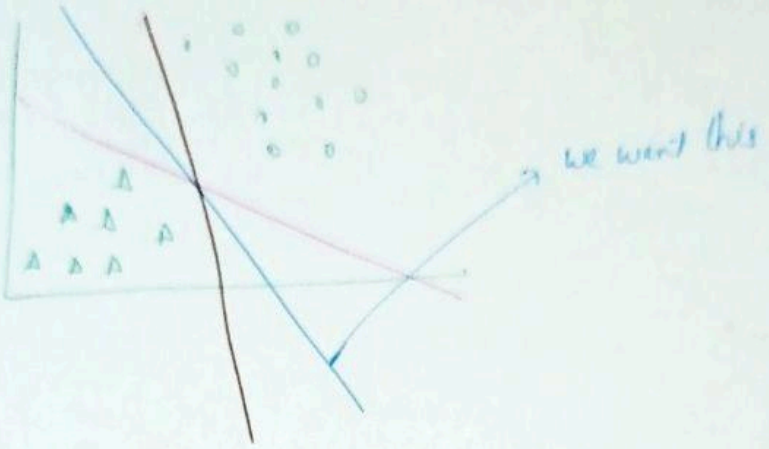


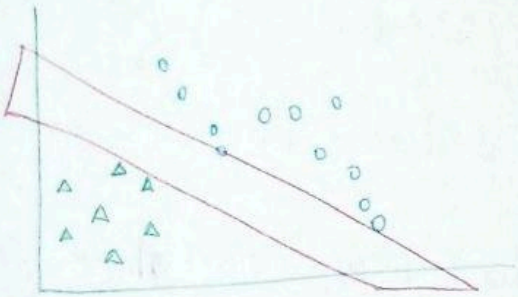
= We have 2 - class classification problem.

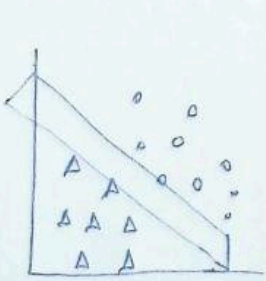
We want to draw a Decision boundary.

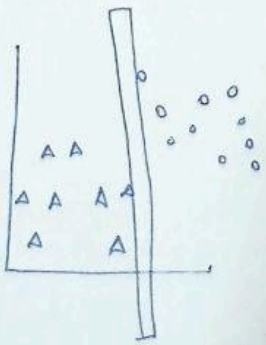
x_1 : Height

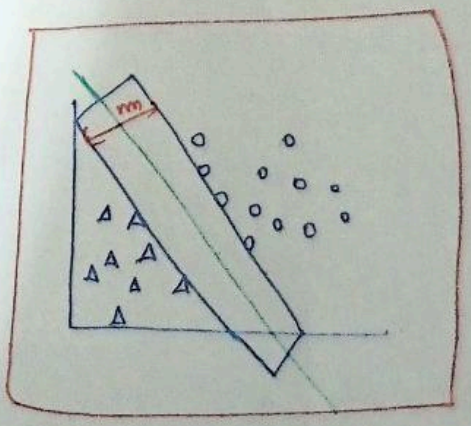
x_2 : weight.







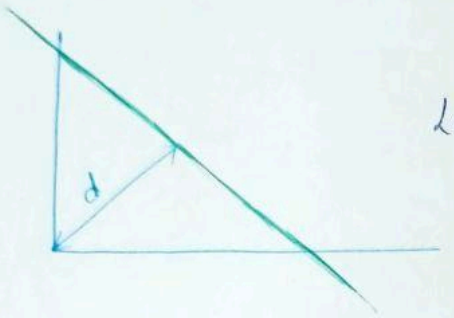




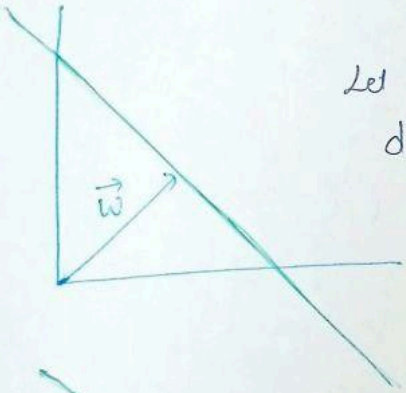
→ The best decision boundary is line through the middle of margin of 2 classes.

→ 'm' is maximum margin b/w 2 classes.

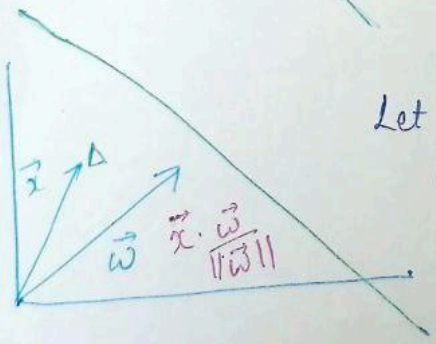
'm' : represents maximum seperation b/w two-class boundaries.



Let 'd' be distance of DB from origin.



Let 'w' be a vector perpendicular to decision boundary.

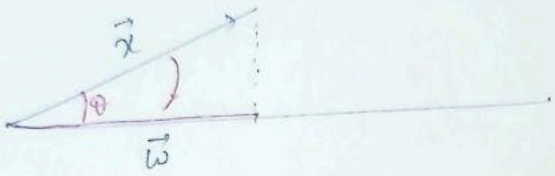


Let $\vec{x}$ be another vector representing a data point.

- Projection of $\vec{x}$ on direction of $\vec{w}$ is :-

$$\vec{x} \cdot \frac{\vec{w}}{\|\vec{w}\|}$$

- Data point x is vector = 2d point, at some angle.

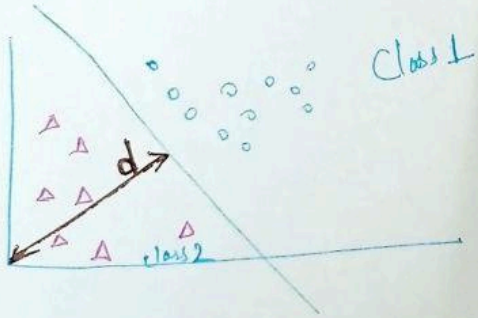


$\vec{x} \cdot \vec{w} \cos\theta$

- If projection $\vec{x} \cdot \frac{\vec{w}}{\|\vec{w}\|} \geq d$

$\Downarrow$

data point $\vec{x}$ belong to class 1 else it belongs to class 2.



So if $\vec{x}$ belongs to class 1 if

$$\vec{x} \cdot \frac{\vec{w}}{\|\vec{w}\|} \geq d$$

$$\vec{x} \cdot \vec{w} \geq d \|\vec{w}\|$$

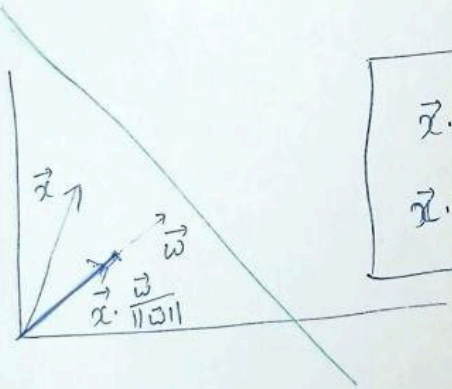
$$\vec{x} \cdot \vec{w} + b \geq 0 \quad \text{where} \quad b = -d \|\vec{w}\|$$

So if we can say $\vec{x}$ belong to class 2 if

$$\frac{\vec{x} \cdot \vec{w}}{\|\vec{w}\|} < d$$

$$\vec{x} \cdot \vec{w} < d\|\vec{w}\|$$

$$\vec{x} \cdot \vec{w} + b < 0 \quad \text{where } b = -d\|\vec{w}\|$$



$\vec{x} \cdot \vec{w} + b \geq 0$, class 1

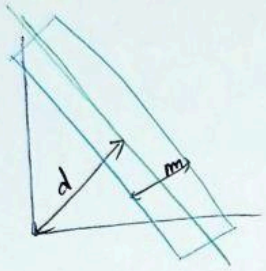
$\vec{x} \cdot \vec{w} + b < 0$, class 2

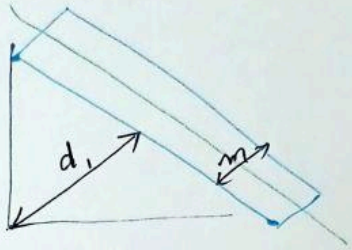
- if length of projection of x towards d is :—

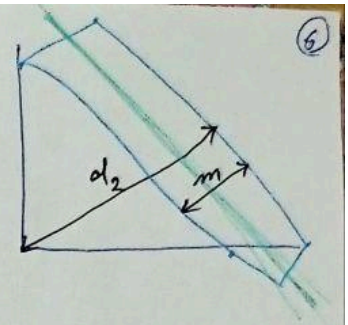
  - $\downarrow\downarrow d$ point x belongs to class i.
  - $\uparrow\uparrow d$ belongs to class j.

- Slight change in position lead to different class.

So we want Margin.







$$m = d_2 - d_1$$

Case 1 :- Suppose, we want $\vec{x}$ belongs to class 1 if :-

$$\vec{x} \cdot \frac{\vec{w}}{\|\vec{w}\|} \geq d_2$$

$$\vec{x} \cdot \vec{w} \geq d_2 \|\vec{w}\|$$

$$\Rightarrow \vec{x} \cdot \vec{w} + b \geq 1 \qquad \text{where } b = 1 - d_2\|\vec{w}\|$$

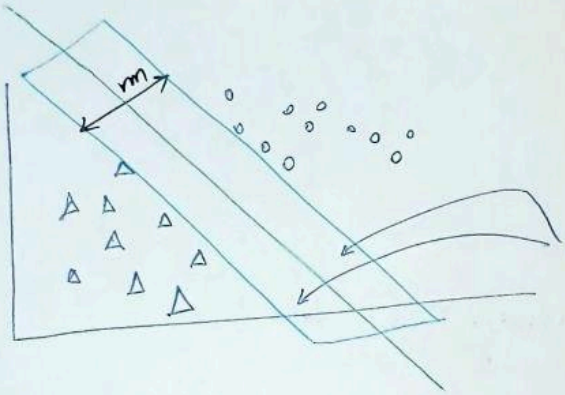
Case 2 :- if $\vec{x}$ belongs to class 2 :-

$$\vec{x} \cdot \frac{\vec{w}}{\|\vec{w}\|} \leq d_1$$

$$\vec{x} \cdot \vec{w} \leq d_1 \|\vec{w}\|$$

$$\vec{x} \cdot \vec{w} + b \leq -1 \quad , \quad \text{where } b = -1 - d_1\|\vec{w}\|$$

From training data we want to find w, b



we want sample to fall in these regions.

$\vec{x} \cdot \vec{w} + b \geq 1$ then class 1.

$\vec{x} \cdot \vec{w} + b \leq -1$ then class 2.

- For classification, support vector is important.

Support vector help in making classification decision.

For i^{th} sample x_i with class label y_i :-

$\vec{x_i} \cdot \vec{w} + b \geq 1$, then class 1 (so, $y_i = 1$)

$\vec{x_i} \cdot \vec{w} + b \leq -1$ then class 2 (so, $y_i = -1$)

$\Downarrow$

our decision boundary rule is :-

$$y_i (\vec{x_i} \cdot \vec{w} + b) \geq 1$$

$\vec{x}_i \cdot \vec{w} + b \leq -1$ & $y_i = -1$

$y_i(\vec{x}_i \cdot \vec{w} + b) \geq 1$ $\quad (-1 * -1 = +1)$

So we need to classify Test data :—

Data $\vec{x}$,

if $\vec{x} \cdot \vec{w} + b \geq 1$, then $\vec{x}$ belong to class 1.

if $\vec{x} \cdot \vec{w} + b \leq -1$, then $\vec{x}$ belongs to class 2.

So test data be $\vec{x}_t$, so ←

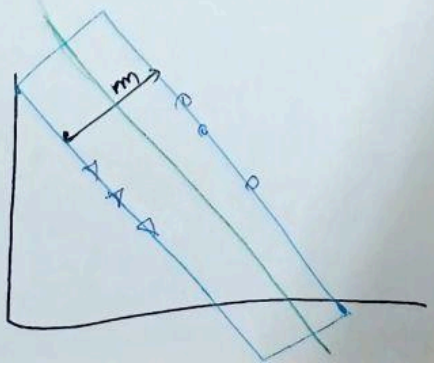
if $\vec{x}_t \cdot \vec{w} + b \geq 1$, then $\vec{x}_t$ belong to class 1.

if $\vec{x}_t \cdot \vec{w} + b \leq -1$, then $\vec{x}_t$ belong to class 2.

↓

To make this decision, we need to find out $\vec{w}$, b.

The goal of SVM is to find out $\vec{w}, b$ from training data.



$\vec{x}_i \cdot \vec{w} + b = 1$, class 1

$y_i = 1$

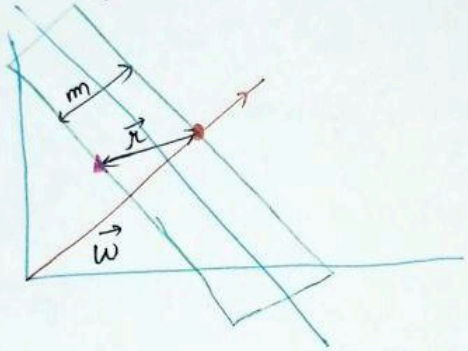
$\vec{x} \cdot \vec{w} + b = -1$, class2

$y_i = -1$

$$y_i(\vec{x}_i \cdot \vec{w} + b) = 1$$

↓

These training data points are called Support vectors.



Let $\vec{r}$ be vector b/w samples on two class-boundaries.

Width m = projection of $\vec{r}$ on perpendicular direction of boundary

m = $\vec{r}$. (unit vector on perpendicular direction of boundary)

$$m = \vec{r} \cdot \frac{\vec{w}}{\|\vec{w}\|}$$

Width $m = \vec{r} \cdot \frac{\vec{w}}{\|\vec{w}\|}$ $\qquad \vec{r} = (\vec{x}_2 - \vec{x}_1)$

Width $m = (\vec{x}_2 - \vec{x}_1) \cdot \frac{\vec{w}}{\|\vec{w}\|} = \frac{(\vec{x}_2 \vec{w} - \vec{x}_1 \vec{w})}{\|\vec{w}\|} = \frac{2}{\|\vec{w}\|}$

- We want to find such a $\vec{w}$, b so that width is maximized, subject to constraint of our decision rule

$$y_i\left(\vec{x_i}\cdot\vec{w}+b\right) \geq 1$$

- To solve constrained optimization problem :—

Langrangian Function.

- This is equivalent to minimizing $\frac{1}{2}\|\vec{w}\|^2$ subject to :—

$$y_i\left(\vec{x}_i\cdot\vec{w}+b\right)\geq 1$$

$m \uparrow\uparrow$	$w \downarrow\downarrow$	$m = \frac{2}{\|w\|}$
$m \downarrow\downarrow$	$w \uparrow\uparrow$	

L = take term want to minimize $- \sum_{i=1}^{N}$ add or substract other constraint.

α_i = value will diff. for each training data.

$$L = \frac{1}{2}\|\vec{w}\|^2 - \sum^{N}\left[\alpha_i\left\{y_i(\vec{x_i}\cdot w+b)-1\right\}\right]$$

α_i = Lagrange multiplier

So, considering N training data points, the lagrangian fun^n is :-

$$L = \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{N}\left[\alpha_i\{y_i(\vec{x}_i\cdot\vec{w}+b)-1\}\right]$$

→ I want to minimize L w.r.t $\vec{w}$ and b, so :-

$$\frac{\partial L}{\partial w} = 0 \quad \text{and} \quad \frac{\partial L}{\partial b} = 0$$

Note : The minus (-) sign in the Lagrangian fun^n indicates that I want to maximize L w.r.t α.

setting $\frac{\partial L}{\partial w} = 0$, we get :-

$$\vec{w} = \sum_{i=1}^{N}\left(\alpha_i y_i \vec{x}_i\right)$$

setting $\frac{\partial L}{\partial b} = 0$, we get :-

$$\sum_{i=1}^{N}\alpha_i y_i = 0$$

After putting value and simplifying, we get :-

$$L = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\left(\vec{x}_i\cdot\vec{x}_j\right) + \sum_{i=1}^{N}\alpha_i$$

Subject to :- $\alpha_i \geq 0$, $\forall i \in [1, N]$

$$\sum_{i=1}^{N} (\alpha_i y_i) = 0$$

$$L = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (\vec{x}_i \cdot \vec{x}_j) + \sum_{i=1}^{N} \alpha_i$$

- Optimization depends on dot product of pair of training samples $(x_i \cdot x_j)$

→ For all support vector :-

$$\alpha_i \left(y_i \left(\vec{w}_i \cdot \vec{x} + b \right) - 1 \right) = 0$$

Then either

$$\alpha_i = 0$$

or

$$y_i (w_i \vec{x} + b) - 1 = 0$$

(i) for support vector :-

$$y_i \left(w_i \vec{x} + b \right) = 1$$

(ii) for all not support vector $\alpha_i = 0$

It can be shown that solution must satisfy :—

$$\alpha_i \geq 0 \quad , \forall i \in [1, N]$$

$$y_i (\vec{x}_i \cdot \vec{w} + b) - 1 \geq 0$$

and

$$\alpha_i \left(y_i (\vec{x}_i \cdot \vec{w} + b) - 1 \right) = 0$$

Last condition requires —

- either $y_i(\vec{x}_i \cdot \vec{w} + b) = 1$ (which is true for support vectors)

or

$\alpha_i = 0$ (i.e :- for training data points, other than support vectors, $\alpha_i = 0$)

$\alpha_i = 0$ = non-support vectors

Can also $\alpha_i = 0$ for support vector.

- The Dual form :-

It can be shown that solution must satisfy :-

$$\alpha_i \geq 0 \quad , \forall i \in [1, N]$$

$$y_i(\vec{x}_i \cdot \vec{w} + b) - 1 \geq 0$$

and

$$\alpha_i \left(y_i(\vec{x}_i \cdot \vec{w} + b) - 1 \right) = 0$$

Last condition requires :-

either $y_i(\vec{x}_i \cdot \vec{w} + b) = 1$ { which is true for support vectors }

or

$\alpha_i = 0$ { i.e for training data points, other than support vectors , $\alpha_i = 0$ }

[Lagrange multiplier of S.V may/may not be 0]

By maximization of :-

$$L = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left(\vec{x}_i \cdot \vec{x}_j \right) + \sum_{i=1}^{N} \alpha_i$$

→ Maximization of L w.r.t α, we can find values of α_i

$\forall i \in [1, N]$ (i.e., all training data)

We can find :-

$$\vec{w} = \sum_{i=1}^{N} (\alpha_i y_i \vec{x}_i)$$

and, it can be shown that :-

$$b = \frac{1}{N_s} \sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{N} \alpha_j y_j (\vec{x}_i \cdot \vec{x}_j) \right)$$

N_s : no. of support vectors

If not support vector $\alpha_j = 0$
⇓
$w, b = 0$

(non-support vectors don't contribute to Calculation)

St. line only depends on support vectors.
↓
Ignore other points than S.V not useful in decision making.

Started with ..
$x_t \cdot w + b$
↑ ↑
replaced

Testing :-

Suppose I want to find class label of test data x_t.

I'll calculate :-

$$q_t = (\vec{x}_t \cdot \vec{w} + b) = \vec{x}_t \left(\sum_{i=1}^{N} (\alpha_i y_i \vec{x}_i) \right) + b = \sum_{i=1}^{N} (\alpha_i y_i \vec{x}_i \cdot \vec{x}_t) + b$$

If $q_t \geq 1$, test data x_t belongs to class $y_t = 1$.

If $q_t \leq -1$, test data x_t belongs to class $y_t = -1$.

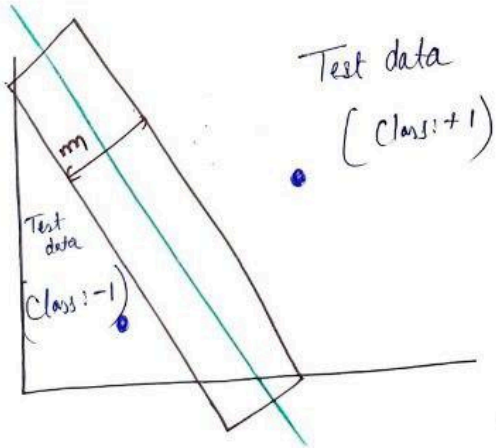
here $\sum_i \alpha_i y_i (x_i \cdot x_t) + b$

dot product of training with data of test shows :—

→ All training data contribute to decision making for testing data.

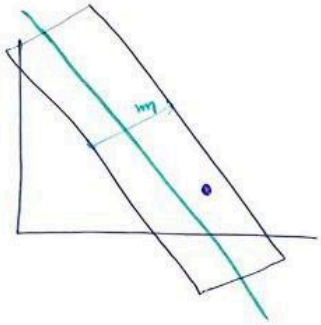
here also $\alpha_i = 0$ for non-support vector.

only those data lying on boundary of class.
They decide for decision of test data.



$$q_t = \sum_{i=1}^{N} (\alpha_i y_i \vec{x}_i \cdot \vec{x}_t) + b$$

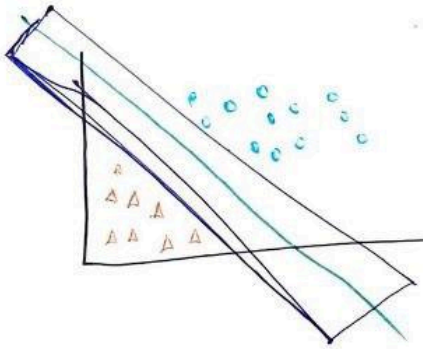
Therefore, data points which are not support vector don't play role in determining class label of test data.



For finding out class label of test data x_t :-

what happens if $1 > q_t > -1$?

Maximum Margin Classifier :-



→ Since we are trying to maximize margin during training, It is called as :- Maximum Margin Classifier.

ξ_i = Shi = Slack variable.

SVM with Training data within Margin :- Soft Margin SVM

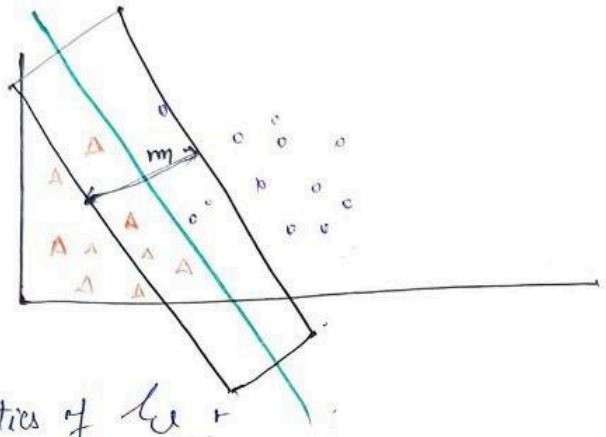
Decision rule for i^{th} sample x_i with class label y_i

$\vec{x_i} \cdot \vec{w} + b \geq 1 - \xi_i$, then class 1 (so, $y_i = 1$)

$\vec{x_i} \cdot \vec{w} + b \leq -(1 - \xi_i)$ then class 2 (so, $y_i = -1$)

Considering both conditions, our decision rule :-

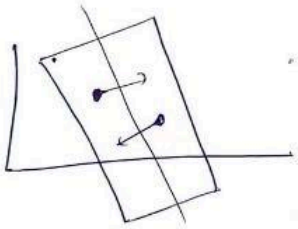
$$y_i(\vec{x_i} \cdot \vec{w} + b) \geq 1 - \xi_i$$



Characteristics of ξ_i :-

$\xi_i \geq 0$ is called slack variable.

if $\xi_i > 1$, data is misclassified.



If $\xi_i = 1$, data lies on decision boundary / LoC

- $\xi > 1$ = data crossed decision boundary.
- $\xi = 1$ exactly lying on D.B
- $0 < \xi < 1$: Soft Margin SVM

For Hard Margin Classifier :-

$$y_i(\vec{x}_i \cdot \vec{w} + b) \geq 1 - \underbrace{\xi_i}_{0}$$

$$y_i(\vec{x}_i \vec{w} + b) \geq 1$$

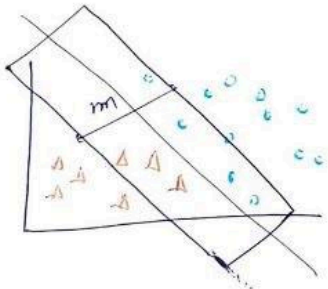
Ideally we want ξ_i to be as less as possible.

or $\xi_i = 0$ (want)

Soft Margin SVM:-

$$\frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N} \xi_i$$

Subject to :- $y_i(\vec{x}_i \cdot \vec{w} + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$



C : determines weightage on penality on slack variable.

{ for training data, we need to know w, b, ξ_i }

C = constant, which is how much penality we apply if $\xi_i > 0$

→ Second term in the fun to be minimized is penality on points crossing class boundary

→ Minimization of 2nd term ensures that many training data points don't cross boundary by a large amount.

Soft Margin SVM : Lagrangian function :-

Minimize :-

$$\frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N} \xi_i$$

Subject to :

$$y_i\left(\vec{x}_i \vec{w} + b\right) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0$$

The Lagrangian fun is :-

$$L = \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N}\left[\alpha_i\left\{y_i(\vec{x}_i \cdot \vec{w}+b)-1 + \xi_i\right\}\right] - \sum_{i=1}^{N} \mu_i \xi_i$$

$$L = \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N}\left[\alpha_i\left\{y_i(\vec{x}_i \cdot \vec{w}+b)-1+\xi_i\right\}\right] - \sum_{i=1}^{N} \mu_i \xi_i$$

$\alpha_i \geq 0$, $\mu_i \geq 0$: lagrange Multiplier

From this we can solve for $\vec{w}$, b, and ξ_i

Minimize :- $\frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N} \xi_i$

Subject to $y_i\left(\vec{x}_i \cdot \vec{w} + b\right) \geq 1 - \xi_i$ and $\xi_i \geq 0$

C is a hyperparameter of method.

$(\vec{x_i} \cdot \vec{w} + b) \geq 1 - \zeta_i \rightarrow$ class 1 (+1)

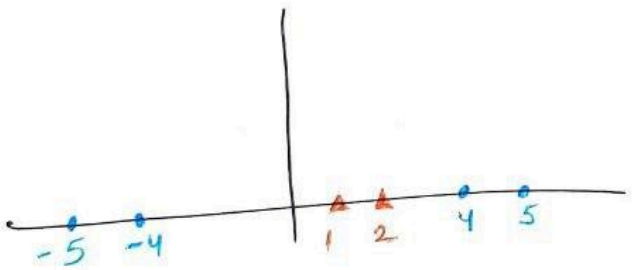
$\leq -(1 - \zeta_i) \rightarrow$ class 2 (-1)

w, b = we'll minimize

for ζ_i = we'll maximize.

as $-\zeta_i$ = already -ve sign there

Let's consider :-



- Considering I have one-dimensional dataset with feature x_1.

There are 2 classes.

Can we classify data using our SVM designed so far?

No, because data points are not linearly seperable.

Now if we still want to classify linearly :—

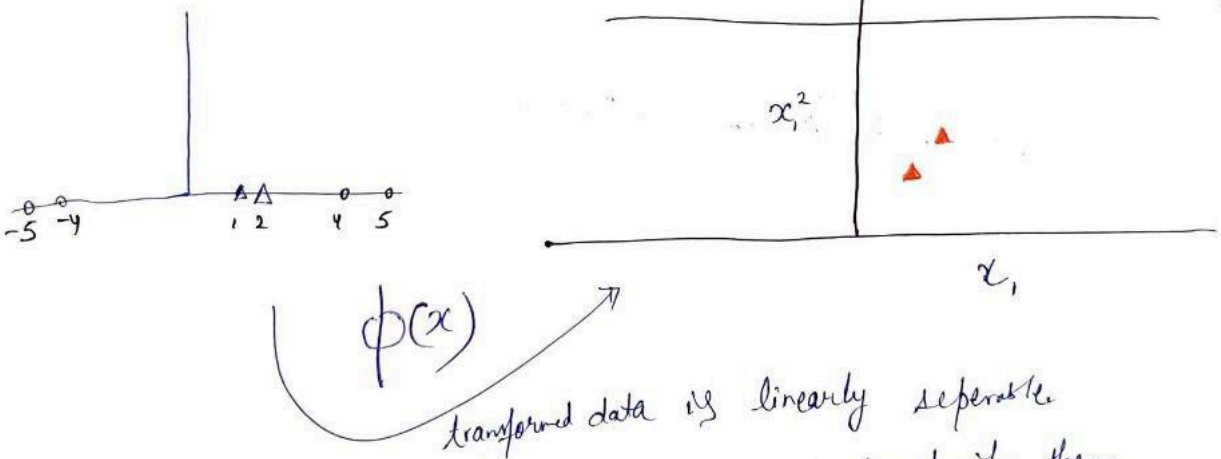
Making new Dimension :- x_1^2

If we still want to classify:

new dimension x_1^2.

$x_1 \xrightarrow{\phi(x)} (x_1, x^2)$

1 →	(1,1)	-4 →	(-4, 16)
2 →	(2,4)	-5 →	(-5, 25)
3 →	(3, 9)		
4 →	(4, 16)		
5 →	(5, 25)		



transformed data is linearly seperable.
So SVM can be used to classify them.

$\phi(x)$: takes a vector x and transforms it into a higher-dimensional vector $\phi(x)$.

SVM with Transformed Data : Testing :-

SVM with original SVM :-

Suppose I want to find out the class label of test data x_t.

I'll calculate :-

$$q_t = (\vec{x}_t \cdot \vec{w} + b)$$

$$= \vec{x}_t \left(\sum_{i=1}^{N} (\alpha_i y_i \vec{x}_i) \right) + b$$

$$= \sum_{i=1}^{N} (\alpha_i y_i \cdot \vec{x}_i \cdot \vec{x}_t) + b$$

SVM with Transformed Data :

Suppose I want to find out class label of test data x_t.

I'll calculate :-

$$q_t = \left(\phi(\overrightarrow{x_t}) \cdot \vec{w}_t + b \right)$$

$$= \vec{x}_t \cdot \left(\sum_{i=1}^{N} \left(\alpha_i \cdot y_i \overrightarrow{\phi(x_t)} \right) \right) + b$$

$$= \sum_{i=1}^{N} \left(\alpha_i y_i \overrightarrow{\phi(x_i)} \cdot \overrightarrow{\phi(x_t)} \right) + b$$

- For every data point, finding transformation $\phi(x)$.
  Taking dot products is computationally challenging.

The Dual Form :-

After putting values and simplifying, we get :-

$$L = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left(\overrightarrow{\phi(x_i)} \cdot \overrightarrow{\phi(x_j)} \right) + \sum_{i=1}^{N} \alpha_i$$

We want to maximize L w.r.t α

Subject to :-

$$\alpha_i \geq 0, \quad \forall i \in [1, N]$$

$$\sum_{i=1}^{N} (\alpha_i y_i) = 0$$

Testing :- To find out class label of test data x_t :-

$$q_t = \left(\overrightarrow{\phi(x_t)} \cdot \vec{w} + b \right)$$

$$q_t = \sum_{i=1}^{N} \left(\alpha_i y_i \overrightarrow{\phi(x_i)} \cdot \overrightarrow{\phi(x_t)} \right) + b$$

If we take data to higher dimension space we can often not always classify linearly data.

now what we are doing :- Transform data → Take Dot Product

What if we can do!—

The Dual Form :-

After putting values and simplifying :-

$$L = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \left(\overrightarrow{\phi(x_i)}\cdot\overrightarrow{\phi(x_j)}\right) + \sum_{i=1}^{N}\alpha_i$$

$$L = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j K\left(\vec{x}_i, \vec{x}_j\right) + \sum_{i=1}^{N}\alpha_i$$

Why Kernel :-

if 100 million data $\xrightarrow{\text{Higher Dimen}}$ $(100 \text{ million})^2$

What if 100 million data with 200 dimension ?

↓

It is difficult to compute sq. for all dimension wise square.

What if infinite dimension : Hard

Practically not feasible for transforming data into that much Higher D

Testing :-

Suppose I want to find out class label of test data x_t

$$q_t = \left(\overrightarrow{\phi(x_t)}\cdot\vec{w} + b\right)$$

$$= \sum_{i=1}^{N}\left(\alpha_i y_i K\left(\vec{x}_i, \vec{x}_j\right)\right) + b$$

$$K(\vec{x_i}, \vec{x_j}) = \phi(\vec{x_i}) \cdot \phi(\vec{x_j})$$

Suppose x in 1D, ϕ transforms x to 2D then dot product

Kernel fun^n :- It helps us getting dot product of transformed data without requiring calculation of transformation of every data point.

Kernel fun^n :- Converges dot product after transformation to original data product and (square if 1D → 2D)

example:- We have 2D points :-

one data point is $x : (x_1, x_2)$

Another data point is $z : (z_1, z_2)$

using transformation :-

$$\phi(x) = (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$

$$\phi(z) = (z_1^2, \sqrt{2}\, z_1 z_2, z_2^2)$$

So $\phi(\vec{x}) \cdot \phi(\vec{z}) = (x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 x_2 z_1 z_2)$

$$= (x_1 z_1 + x_2 z_2)^2$$

$$= (\vec{x} \cdot \vec{z})^2$$

$$= K(\vec{x}, \vec{z})$$

no need to go for Higher dimension go- (dot product)^2 ie Kernel

$$K(\vec{x}, \vec{z}) = (\vec{x} \cdot \vec{z})^2$$

By using kernel, got rid of need to calculate $\phi(x)$ and $\phi(z)$

eg:- $A = (a_1, a_2)$

$B = (b_1, b_2)$

$A \cdot B = (a_1 b_1 + a_2 b_2)$

(Dot product is = Multiplication & addition)

↓

we got rid of higher computational overhead to changing data point to higher dimension to dot product and square in same dimension.

Kernel function : If I have 2D data points

one data point is $x = (x_1, x_2)$

Another data point $z = (z_1, z_2)$

↓

Using Transformation :

$$\phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

$$\phi(z) = (z_1^2, \sqrt{2}z_1z_2, z_2^2)$$

So $\phi(\vec{x}) \cdot \phi(\vec{z}) = (x_1^2z_1^2 + 2x_1x_2z_1z_2 + x_2^2z_2^2)$

$$(x_1z_1 + x_2z_2)^2$$

$$(\vec{x} \cdot \vec{z})^2$$

$$= K(\vec{x} \cdot \vec{z})$$

$$K(\vec{x}, \vec{z}) = (\vec{x} \cdot \vec{z})^2$$

Using Kernel, I get rid of need to calculate $\phi(x)$, $\phi(z)$

2D ⟶ 3D

$(x_1, x_2)(z_1 z_2)$ $\quad x_1^2, \sqrt{2}x_1x_2, x_2^2$

Dot product $\phi(x) \cdot \phi(z) = (x \cdot z)^2$

no need to go in Higher dimension

Kernel fun" helps to get rid of moving/transforming to higher D.

eg:- PCA need to find dot product to find PCA components.

↓

Kernel fun" can be applied to PCA

↓

Kernel PCA

RBF Kernel (Radial Bias fun" / Gaussian Kernel)

$$K(\vec{x}, \vec{z}) = \exp\left(-\frac{\|\vec{x} - \vec{z}\|^2}{2\sigma^2}\right)$$

↓ can also be represented as:

$$K(\vec{x}, \vec{z}) = \exp\left(-\gamma \|\vec{x} - \vec{z}\|^2\right)$$

$$= \exp\left(-\gamma\left(\|\vec{x}\|^2 + \|\vec{z}\|^2 - 2\|\vec{x}\|\|\vec{z}\|\right)\right)$$

$$\exp\left(-\gamma\left(\|\vec{x}\|^2 + \|\vec{z}\|^2\right)\right) \exp\left(2\gamma\|\vec{x}\|\|\vec{z}\|\right)$$

$$= \exp\left(-\gamma\left(\|\vec{x}\|^2 + \|\vec{z}\|^2\right)\right)\left[1 + \frac{2\gamma\|\vec{x}\|\|\vec{z}\|}{1!} + \frac{\left(2\gamma\|\vec{x}\|\|\vec{z}\|\right)^2}{2!} + \cdots\right]$$

$$= \phi^2\left[1 + \frac{a^2\|\vec{x}\|\|\vec{z}\|}{1!} + \frac{\left(a^2\|\vec{x}\|\|\vec{z}\|\right)^2}{2!} + \cdots\right]$$

where $\phi^2 = \exp\left(-\gamma\left(\|x\|^2 + \|\vec{z}\|^2\right)\right)$, $a^2 = 2\gamma$

RBF : Circular type of Bias fun^n

$$\gamma = \frac{1}{2\sigma^2}$$

for support vector user need to input γ.

---

when using kernel if one need to go in infinite space

- infinite term in $\phi(x)$
- " " " $\phi(z)$

$\phi(x)\,\phi(z)$ can take data to ∞ term / space

Normally can't take data from finite to infinite dimension space. Kernel fun^n do.

→ RBF Kernel is used widely.
→ RBF kernel practically utilizes an ∞ dimensional transformation.

Polynomial Kernel

$$K(\vec{x}, \vec{z}) = (\vec{x}\cdot\vec{z} + r)^d$$

eg:- if $r = \frac{1}{2}$, $d = 2$

$$K(\vec{x}, \vec{z}) = \left(\vec{x}\cdot\vec{z} + \frac{1}{2}\right)^2$$

$$= \|\vec{x}\|\|\vec{z}\| + \|\vec{x}\|\cdot\|\vec{z}\|^2 + \frac{1}{4}$$

$$= \left(\|\vec{x}\|, \|\vec{x}\|, \frac{1}{2}\right)\cdot\left(\|\vec{z}\|, \|\vec{z}\|^2, \frac{1}{2}\right)$$

$$= \phi(\vec{x})\cdot\phi(\vec{z})$$

$$\phi(\vec{x}) = \left(\|\vec{x}\|, \|\vec{x}\|^2, \frac{1}{2}\right)$$

$$\phi(\vec{z}) = \left(\|\vec{z}\|, \|\vec{z}\|^2, \frac{1}{2}\right)$$

- every fun^n is not Kernel function.
- Need to follow some criteria (Gram Matrix)

for a fun^n, I have to have to prove Gram Matrix to be Semi definite.

$\phi(\vec{x}) \cdot \phi(\vec{z})$ should be able to $= K(\vec{x} \cdot \vec{z})$

Neural Networks & SV need train data label, supervised.

→ Random forest need to find probability can't be applied. (supervised)

- NN can be applied to unsupervised data too.
- If data is image, video convert data into feature vector in supervised ML.

  NN can directly used image, video in input.

W = How much importance to $x_1, x_2, \cdots$ Layer linear.

y = linear, every layer produces linear o/p

Neural N/w :

why NN :— It can be used for different type of tasks :—

- Classification
- Regression
- Detection
- Segmentation
- Translation
- Feature extraction

many more....

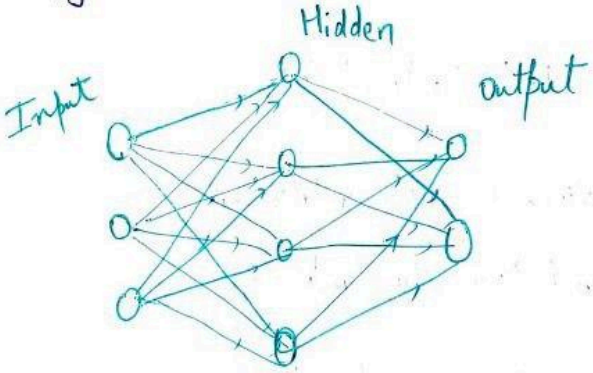
Why now :— Availability of hardware :—

- GPU
- Storage
- Cloud services

- Availability of large datasets.

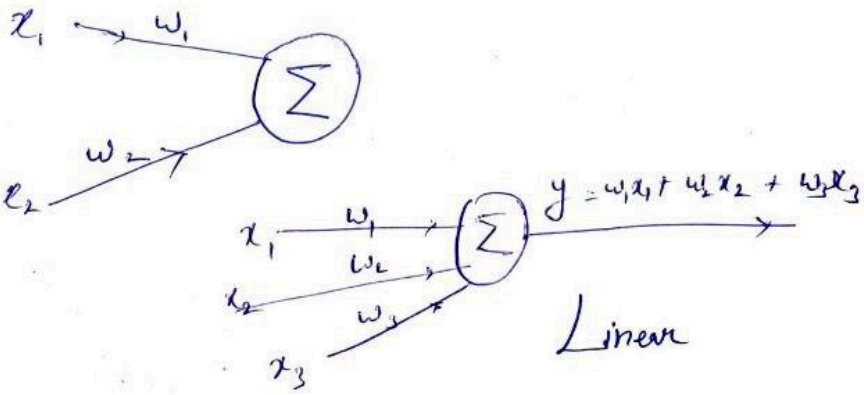
→ Availability of implementation platforms :—

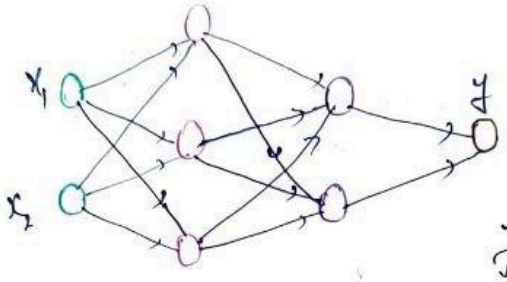
- PyTorch
- Tensorflow



Neural n/w came from biological context.

Neuron :—





$y = f(x_1, x_2)$

f = linear function

x_1 : No. of cars per sq. km

x_2 : No. of industries per sq. km

y : Predicted AQI (air quality index)

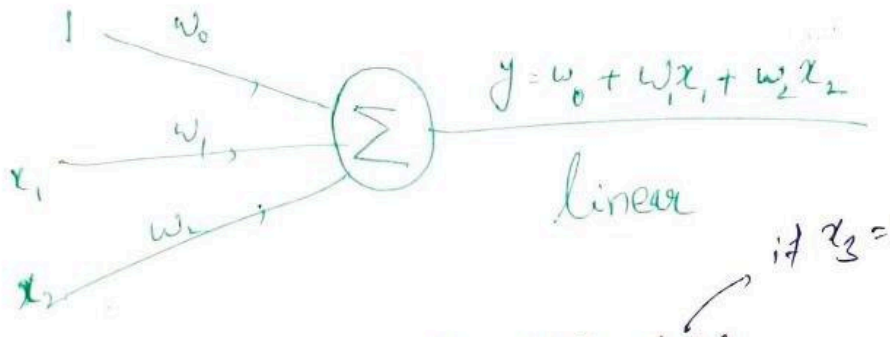
$$y = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3$$

goal is to find $\alpha_1, \alpha_2, \alpha_3$

- Assumptions : linear relationship

→ we don't know constants α_i

N.N can learn each α_i



if $x_3 = 1$

$$y = w_1 x_1 + w_2 x_2 + w_0$$

(w_1, w_2 : weights) (w_0 : Bias)

$$w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$y = w_1 x_1 + w_2 x_2 + b$$

$$\downarrow$$

$$w^T x + b = w'^T x'$$

$$w' = \begin{bmatrix} w_1 \\ w_2 \\ w_0 \end{bmatrix} \qquad x' = \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}$$

Also if $y(1) = w_1 x_1(1) + w_2 x_2(1) + b$

$$\downarrow$$

$$w^T x(1) + b$$

$$w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \qquad x(1) = \begin{bmatrix} x_1(1) \\ x_2(1) \end{bmatrix}$$

↑ for one data points (its features)

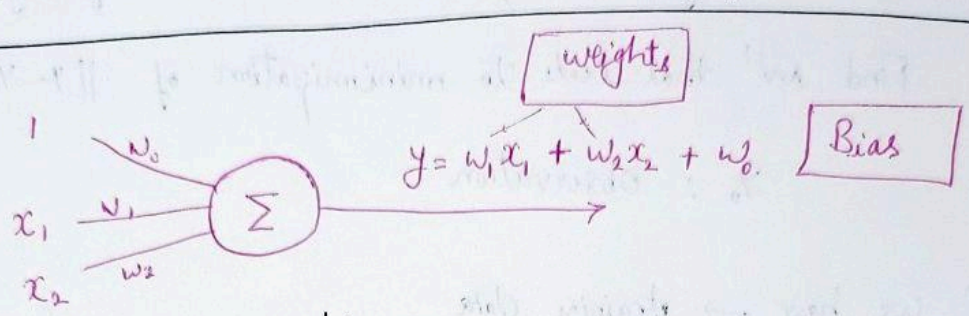
Suppose n such data points :-

$x(1), x(2) \ldots x(n)$ then :-

$$X = \begin{bmatrix} x(1) & x(2) \ldots x(n) \end{bmatrix} = \begin{bmatrix} x_1(1) & \cdots & x_1(n) \\ x_2(1) & \cdots & x_2(n) \\ x_3(1) & \vdots & \vdots \\ \vdots & & \end{bmatrix}$$

$x(1)$ represent 1st data.

n = no. of data points, each with some specific dimension.



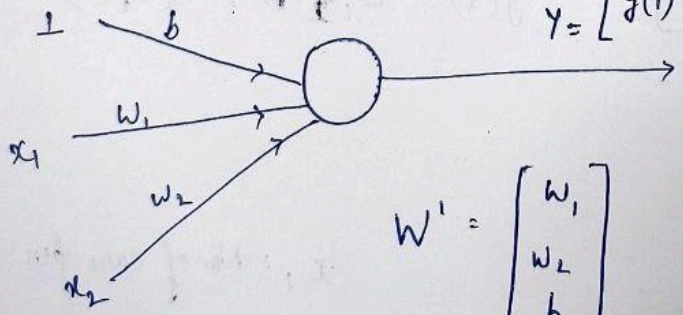
Linear

$$w' = \begin{bmatrix} w_1 \\ w_2 \\ w_0 \end{bmatrix} \qquad x' = \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}$$

height weight

$$x(1) = [\, x_1(1) \quad x_1(2) \ldots]$$

$$x(2) = [\, x_2(1) \quad x_2(2) \ldots]$$



$$Y = [y(1)\ y(2) \ldots y(n)]^T = W^T X + b$$

$$\downarrow$$

$$W'^T X'$$

$$W' = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix}$$

$$X' = [x'(1) \quad x'(2) \quad \ldots \quad x'(n)] = \begin{bmatrix} x_1(1) & \ldots & x_1(n) \\ x_2(1) & \ldots & x_2(n) \\ 1 & \ldots & 1 \end{bmatrix}$$

$X = [x(1) \quad x(2) \quad \ldots \quad x(n)]$ = AQI

$Y_0 = [y_0(1) \quad y_0(2) \quad \ldots \quad y_0(n)]$ and NN outputs

$Y = [y(1) \quad y(2) \ldots y(n)]$

How to find W', i.e w_1, w_2, b analytically ?

Find W' that leads to minimization of $\|y - y_0\|^2$

y_0 : observation

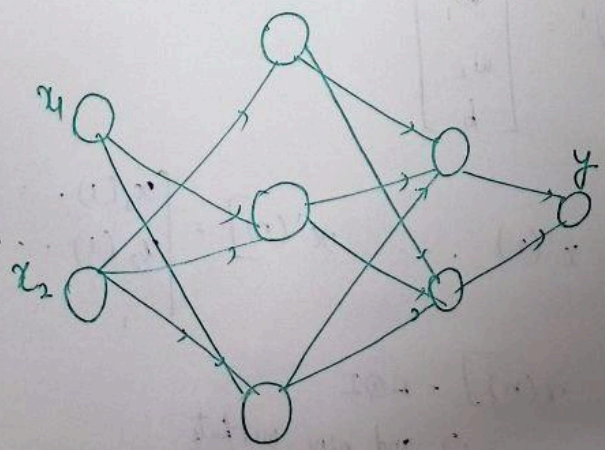
- We have → training data
  → Actual Hb value
- Closed form solution : Analytical approach
  ↓
  iteratively updating eqn w, b.
  equating $w, b = 0$ then updating eqn.

$x(1) = \{x_1(1) \quad x_2(1)\}$

$x_2(2) = \{x_2(1) \cdots x_2(2)\}$

$y(1) = w_1 x_1(1) + w_2 x_2(1)$

$y(2) = w_1 x_2(1) + w_2 x_2(2)$.



x_1 : No. of cars per sq. km

x_2 : No. of industeries per sq. Km

- Assumption : Linear Relationship

↓

But we don't know constants α_i

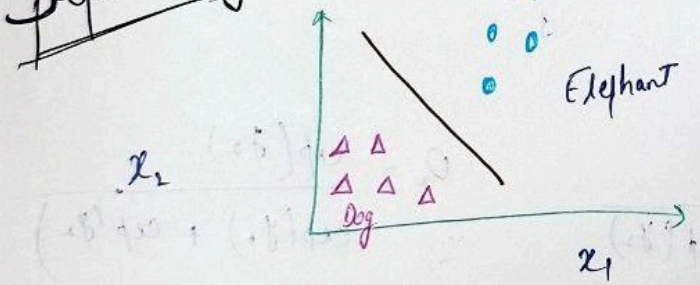
- NN can learn each α_i.

$$y = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3$$

- Solution for w' will give values of α_i

→ One particular dimension will be 1, else all 0.

Seperability :-



elephant vs dog classification

x_1 : Height

x_2 : Weight

Linearly seperable

Class Name	Class Number	Encoding
Elephant	1	10
Dog	2	01

if n no. of classes

Class 1 :- 1 0 0 0 . . . 0

Class 2 :- 0 1 0

⋮

Class n :- 0 0 0 1

$$y_1 = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3$$

$$y_2 = \beta_1 x_1 + \beta_2 x_2 + \beta_3$$

There is not gaurantee that y_i will be 0 or 1

↓

So Softmax

$$O_i = \frac{\exp(y_i)}{\sum_j \exp(y_j)}$$

output class label :- $\underset{j}{\text{argmax}}\ O_j$

$$O_1 = \frac{\exp(y_1)}{\exp(y_1) + \exp(y_2)} \qquad O_2 = \frac{\exp(y_2)}{\exp(y_1) + \exp(y_2)}$$

So on ...

$O_1 \ldots O_n$ = will be range of $(0,1)$

exp of anything = $\frac{\text{+ve value}}{\text{Larger term}}$

↓

range $(0,1)$

* Each neuron corresponding o/p → Probability o/p.

eg :- $y_1 = o_1$ 0.3 → 0

$y_2 \to o_2$ 0.5 → 1

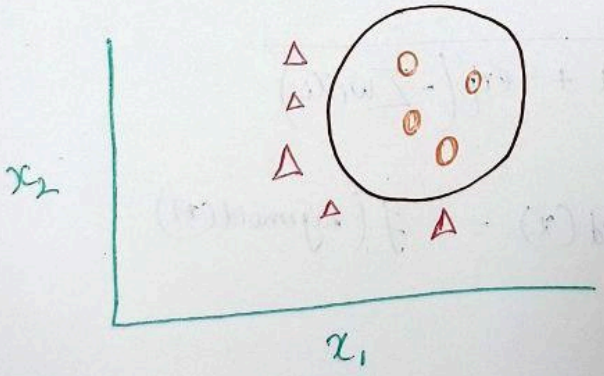
$y_3 \to o_3$ 0.2 → 0

So o/p class = class 2.

→ for getting o/p :⇒ either Softmax

→ or 1 hot encoding

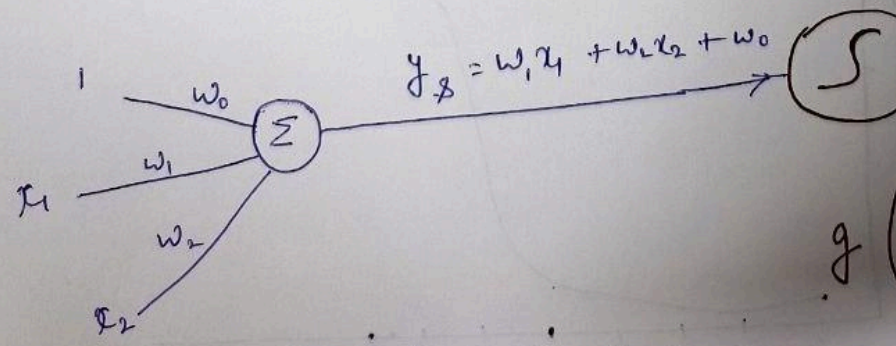
But if data is not linearly seperable ?



Activation function

g

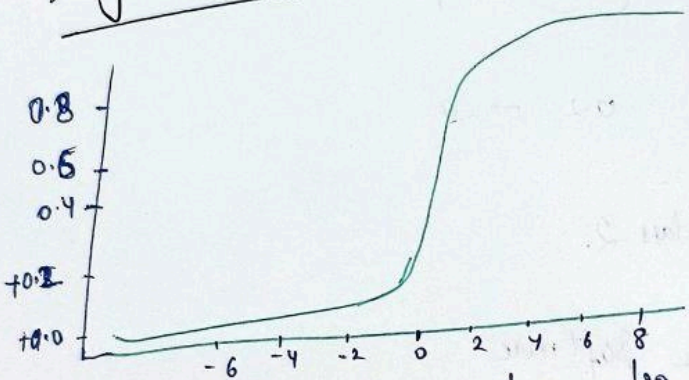
$\hat{y} = g(y_s)$



$g\left(\sum w_i x_i\right)$

Another Non-linear fun^n:-

Sigmoid :-



$$\text{Sigmoid}(x) = \frac{1}{1+\exp(-x)}$$

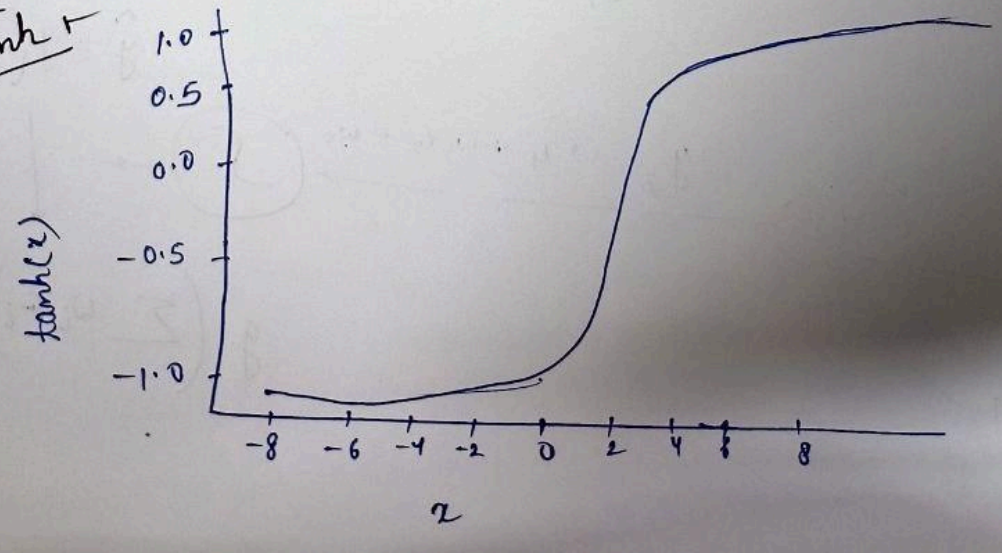
derivative of Sigmoid = also in terms of Sigmoid

$$y_0 = w_1 x_1 + w_2 x_2 + w_0 = \sum_{i=0}^{2} w_i x_i$$

$$= \frac{1}{1 + \exp\left(-\sum w_i x_i\right)}$$

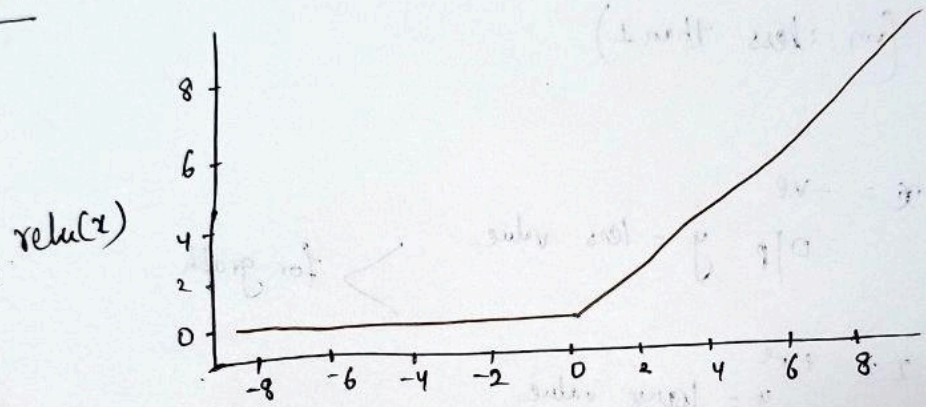
$$\frac{d}{dx}\,\text{sigmoid}(x) = f(\text{sigmoid}(x))$$

Tanh :-



$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$

ReLU :-

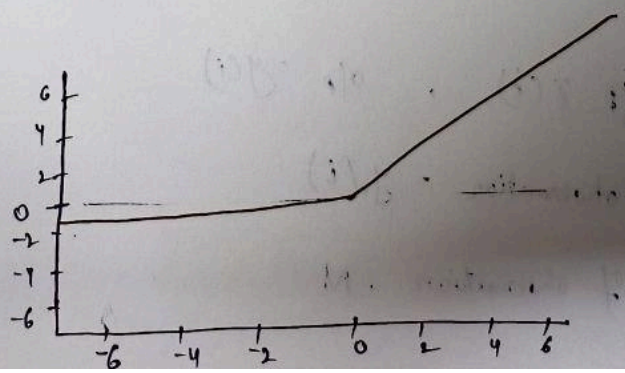


$$\text{ReLU}(x) = \max(0, x)$$

$$\text{ReLu}(x) = x \text{ if } x > 0$$
$$= 0 \text{ otherwise}$$

$$\text{ReLu}\left(\sum w_i x_i\right) = \sum w_i x_i \text{ if } \sum w_i x_i > 0$$
$$0 \text{ otherwise}$$

Leaky ReLu :-



LeakyRelu(x) = max (mx, x)

m = gradient of line

m = very small value eg:- m = 0.001

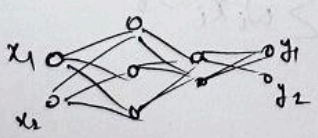
(m : less than 1)

if x = -ve
o/p y = less value > for graph

if x = +ve
y = large value

- By observation view :- LeakyRelu works better than ReLu.

How to measure Performance of NN ?



= Loss ↓ (Diff. b/w o/p and reality (observation))

For ip x(i) , o/p = y(i)

i^{th} observation = $y_o(i)$

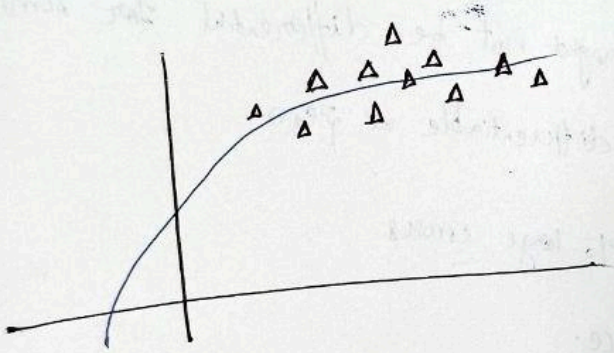
No. of observation : N

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_o(i) - y(i))^2$$

Why square ? ⇒ +ve, -ve loss may cancel out.

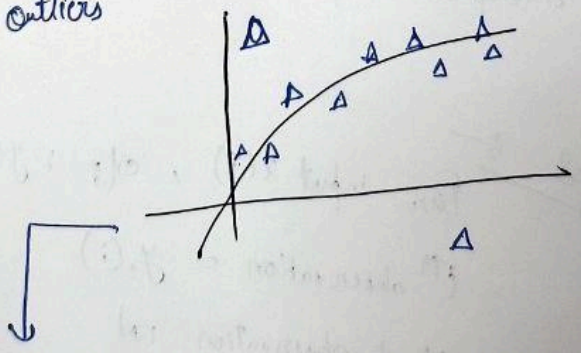
- MSE loss is mostly used in Regression.

Case 1: if graph is



= low MSE

if 2 outliers



Overall MSE ↑↑ even by 2 outliers

∴ high MSE.

Mean Absolute Error :-

For i/p $x(i)$, output = $y(i)$

i^{th} observation = $y_o(i)$

No. of observation = N

$$MAE = \frac{1}{N} \sum_{i=1}^{N} |y_o(i) - y(i)|$$

Drawback of MAE :- might not be differential for some cases.

eg:- not differentiable at zero.

→ less sensitive to large errors

→ slow convergence.

→ less interpretable for variance

Huber loss :-

For input $x(i)$, o/p : $y(i)$

i^{th} observation = $y_o(i)$

No. of observation : N

$$L_\delta(a) = \begin{cases} \frac{1}{2} a^2 & \text{if } |a| \le \delta \\ \delta\left(|a| - \frac{1}{2}\delta\right) & \text{if } |a| > \delta \end{cases}$$

$a = y - f(x)$ is residual, δ = threshold parameter.

Why huber loss :-

(i) MSE penalizes large errors more heavily due to
(a) squared term, so sensitive to outliers.

(b) MAE :- less sensitive to outliers as it only considers absolute error.
Sometime it reduces emphasis on large errors that may need correction.

(ii) Huber loss :- it introduces a threshold parameter (delta).

(a) error less than delta = MSE
↓
penalizes error sharply

(b) error > threshold
↓
MAE
→ limits influence of outliers.

(iii) (Huber loss) is differentiable everywhere.

(iv) ↓ allows model to converge faster than MAE by penalizing error more similar to MSE.

(v) Prevents overfitting of noisy data.

- Smooth gradients, sensitivity to smaller error (like MSE)
- Robustness against outliers (like MAE)

Cross Entropy loss :

Observed label = $y_o = [y_{o1} \quad y_{o2}]$

Predicted label = $y = [y_1 \quad y_2]$

$$L = -\sum_{i=1}^{C} y_{oi} \log y_i$$

CEL :- How much similarity b/w predicted and actual value.

eg :- $y_o = [y_{o1} \quad y_{o2}]$ $y = [y_1 \quad y_2]$

$y_{o1} = 1, \; y_{o2} = 0$; $y_1 = 0.9, \; y_2 = 0.1$

$$L = -(1 \log 0.9 + 0 \log 0.1)$$

$= -\log 0.9$

$= -(-0.04) = +ve$ value

confidence in prediction ↑↑ ⇒ loss ↓↓

if $y_1 \quad y_2$

$[0.6 \quad 0.4]$

$$L = -(1 \log 0.6 + 0 \cdot \log 0.4)$$

$= -\log 0.6 = 0.2$

Confidence in prediction ↓ = loss ↑↑

if $y_0 = \begin{bmatrix} y_{01} & y_{02} \\ 1 & 0 \end{bmatrix}$ $\quad y = \begin{bmatrix} y_1 & y_2 \\ 0.999 & 0.0001 \end{bmatrix}$

$$L = -(1 \log 0.999 + 0 \log 0.0001)$$
$$= 0.0004$$

if $y_0 = [1 \quad 0]$ $\quad y = [1 \quad 0]$

$$L = -(1 \log 1 + 0 \log 0)$$
$$= 0$$

absolute value $y = 1$, loss $= 0$

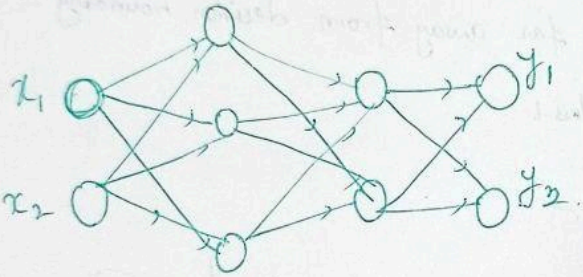
Binary cross entropy = if 2 classes

Multi cross entropy = if more than 2 classes.

Cross - Entropy loss :-

observed label : $y_o = [y_{o1}\ y_{o2}]$

Predicted label : $y = [y_1\ y_2]$



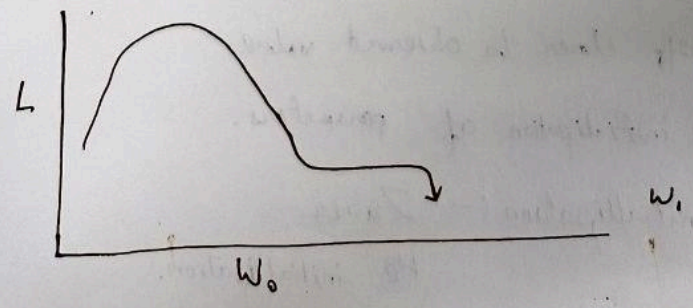
$$L = -\sum_{i=1}^{c} y_{oi} \log y_i = f(w)$$

Loss is a function of weights and biases (parameters).

Training of NNs :-

We can't have an analytical solution of NNs with Non-linearity.

→ Goal of training NN :- get w, b that minimizes loss.



○ Svm :-

Can Svm be used with multiclass classification ?

Two Strategies :-

(a) One versus rest / all classification

(b) One versus group of classes.

eg:- 1 vs (2,3) group of class

. If test data is far away from decision boundary = Class 1.

. less distant not class 1.

• another such cases may be :-

2 / (1 3)

3 / (1 2)

. whichever probability is higher ⟶ assign.

(b) One versus one

Probability 1 : avg. cross all Decision boundary.

Probability 2 : avg. cross all D.B.

:

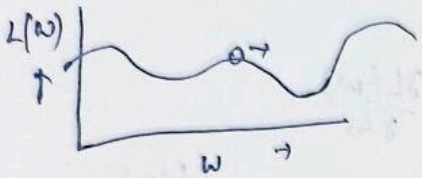
We want o/p closer to observed value.

Randomly initialization of parameters.

Techq to initiallization :- Zavier

He initiallization.

How to reach Global Minima :-



if moving to Right :-

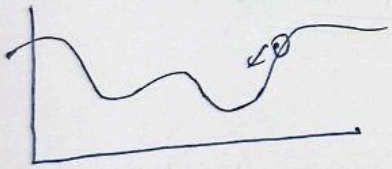
- increasing value of w reduces value of $L(w)$.

$$\frac{\partial L(w)}{\partial w} < 0$$

To increase value of w :-

$$w \leftarrow w - \eta \frac{\partial L(w)}{\partial w}$$

η : positive constant



if moving to left :-

$$\frac{\partial L(w)}{\partial (w)} > 0$$

To decrease value of w :-

$$w \leftarrow w - \eta \frac{\partial L(w)}{\partial w} \qquad \eta = \text{+ve constant}$$

When to stop updating parameter ?

a) If decreasing of loss becomes constants

↓

Loss Saturates ← Stop at that point

b) Pre-decide no of epoch to run loop for updating loss.

c) Try and error (on validation data first).

Gradient Descent :-

(i) initialize parameters

(ii) loop until convergence

(a) Compute gradient $\frac{\partial L(w)}{\partial w}$

(b) update parameters $w \leftarrow w - \eta \frac{\partial L(w)}{\partial w}$

(iii) Return parameters

This is called (Hyper Parameter tuning)

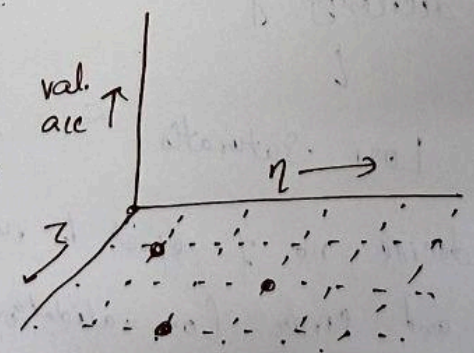
Best possible technique is Grid Search :-

η : learning rate , τ epoch.

$\eta : \{ 0.001 \rightarrow 0.0001 \}$ range of values
$\quad\quad 50 \rightarrow 100$

$\eta_1 = 0.0001$ $\quad$ $\eta_2 = 0.0002$ $\quad$ $\eta_3 = 0.00019$
$\tau_1 = 50$ $\quad$ $\tau_2 = 55$ $\quad$ $\tau_3 = 55$

try diff. diff config of parameter to find best one from grid.

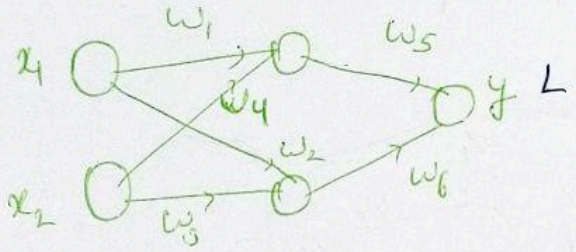


· Can do grid search on even 6 dimensional data.

Parameter : which N.N learns by gradients

Hyperparameter : user set by diff mechanism.

Computing Gradients : Backpropagation :



partial derivate here :

$$\frac{\text{Loss}}{\text{diff } w} = \text{partial}$$

$$w_6 \leftarrow \underset{\text{old value}}{w_6} - \alpha \frac{\partial L}{\partial w_6}$$

To update w_5 :

$$w_5 \leftarrow w_5 - \alpha \cdot \frac{\partial L}{\partial w_5}$$

$L = f(y, y_0)$: observed value constant

$$L = f(y)$$

$$y = g(z_1 w_5 + z_2 w_6)$$

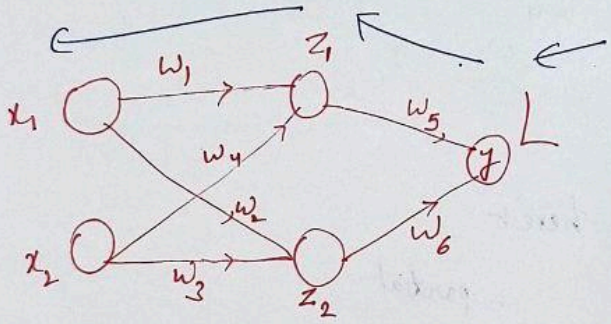
From last layer I can calculate w_5, w_6 last layer's connecting parameters.

- Can't directly

$$z_1 = (x_1, w_1, w_4, x_2)$$

$$z_2 = (w_2, w_3, x_1, x_2)$$

z_1, z_2 are fun^n of above parameters



Backpropagation :- To update w_1, we'll do :-

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial y} \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial w_1}$$ chain rule

loss function and output of nodes needs to be differentiable.

It is not conflict

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial w_1} + \frac{\partial L}{\partial y} \frac{\partial y}{\partial z_2} \frac{\partial z_2}{\partial w_1}$$

$\frac{\partial z_2}{\partial w_1} \to 0$; $\frac{\partial y}{\partial z_2}$: no connection.

$\frac{\partial L}{\partial y} \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial w_1}$

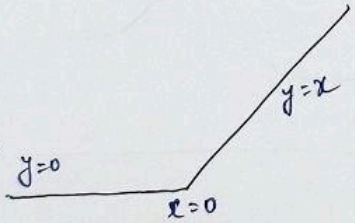
Backpropagation :- It automatic calculation looks like propogating.

only forward propagation there.

At each epoch :- every parameter w, b update.

- each of these fut" has to be differential.
- Loss fut" and o/p of nodes need to be differentiable.

Relu is not differentiable at 0



$\frac{\partial y}{\partial x}$ at $x=0$

$\frac{\partial y}{\partial x} = 1$, $\frac{\partial y}{\partial x} = 0$ at -ve , $\frac{1+0}{2} = \frac{1}{2}$ ~~possibilities~~ approximated

at +ve

$(x+\delta)(x-\delta)$ ← differential now

Optimizing loss :- may be challenging because of

- → local minima
- → Saddle points
- → Vanishing gradients

- local minima how to find Optimum local minima.

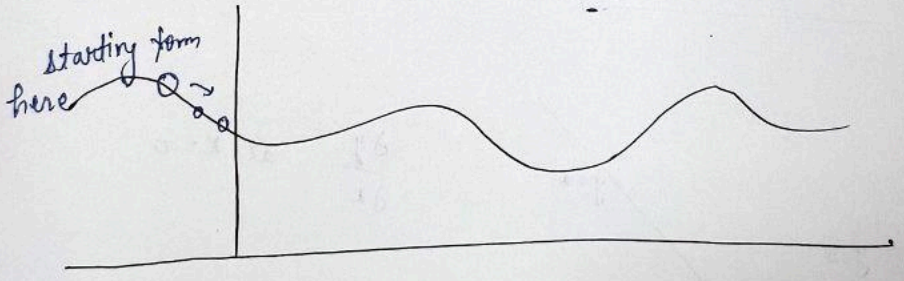
  Gradient may 0 at local minima.

→ value of gradient = 0 at saddle point. → Training Stops

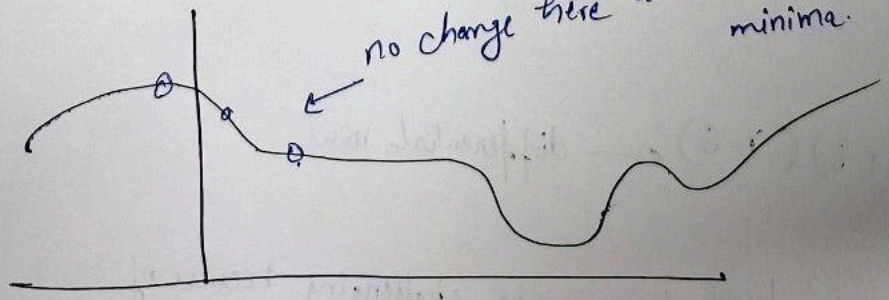
[Most NN saddle point predominant than Local minima.

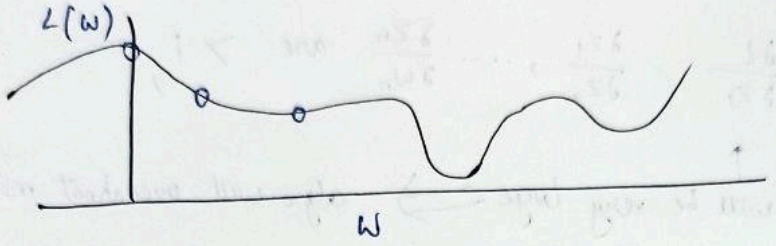
→ Mostly we try to get rid of saddle point.

Local Minima :→



Saddle Points





$\frac{\partial L}{\partial w_1} = 0$ it's minima

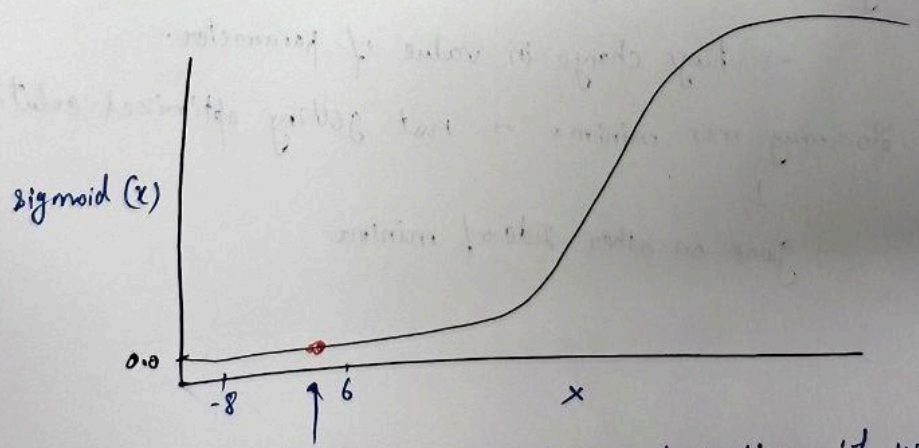
$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

w_1 doesn't change further, same situation happens for other parameters.

Vanishing G.D :- $w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial w_1}$

for w_1 to change $\frac{\partial L}{\partial z_1}$ and $\frac{\partial L}{\partial w_1}$ must be non-zero

$$w_n \leftarrow w_n - \eta \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial z_2} \cdots \frac{\partial z_n}{\partial w_n}$$



for any of nodes related to derivatives, if we are there, derivative term will be zero.

for w_1 to change, each of $\frac{\partial L}{\partial Z_1}, \frac{\partial Z_1}{\partial Z_2} \cdots \frac{\partial Z_n}{\partial w_n}$

must be non-zero.

→ a) If each $\frac{\partial L}{\partial Z_1}, \frac{\partial Z_1}{\partial Z_2}, \cdots \frac{\partial Z_n}{\partial w_n}$ are > 1,

↑

product will be very large ⟹ algo will overshoot minima.

b) for vanishing GD = product term$_i$ < 1 product smaller.

↓

c) If any of DNN one term ≈ 0

↓

overall update ≈ 0 = vanishing Grd.

To prevent vanishing Gr.D →

(a) residual connection

(b) skip connection.

Exploiding Gr.D → each differential product has very large value.

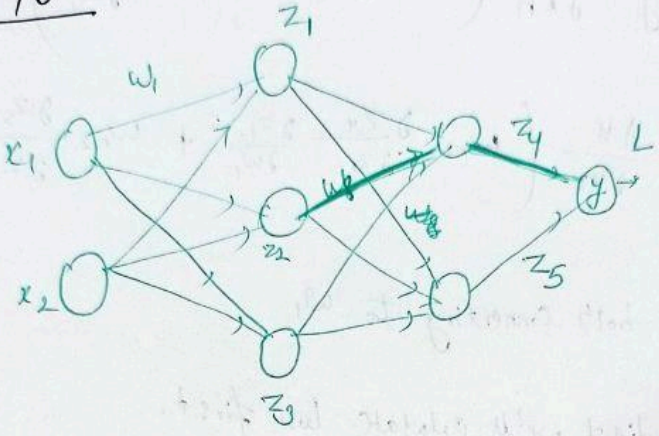
→ huge change in value of parameter.

So may miss minima → not getting optimized solution.

↓

gone on other side of minima.

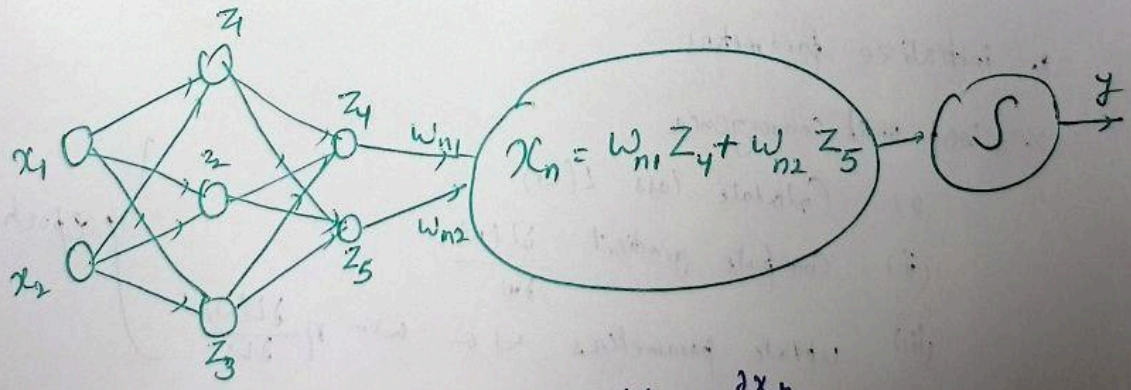
Backpropagation :-



$$\frac{\partial L}{\partial w_8} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial z_4} \frac{\partial z_4}{\partial w_8}$$

z_1 impacting z_4, z_5 both considering path.

→ every node has same probability no vanishing gradient here.



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial x_n} \frac{\partial x_n}{\partial w_1}$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} \frac{\partial y}{x_n} \frac{\partial (w_{n1} z_4 + w_{n2} z_5)}{\partial w_1}$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial x_n} \left(w_{n1} \frac{\partial z_4}{\partial w_1} + w_{n2} \frac{\partial z_5}{\partial w_1} \right) \quad (1)$$

$$\Rightarrow \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial x_n} \left(w_{n1} \frac{\partial z_4}{\partial z_1} \frac{\partial z_1}{\partial w_1} + w_{n2} \frac{\partial z_5}{\partial z_1} \frac{\partial z_1}{\partial w_1} \right)$$

here z_4, z_5 both connecting to w_1

- To calculate gradient we'll calculate loss first.

Gradient Descent : challenges :-

[suppose 10 training data
if 10 million training data :- to update every parameter to pass through nn is very hard]

1. initialize parameters
2. loop until convergence
   (i) Calculate loss $L(w)$
   (ii) Compute gradient $\frac{\partial L(w)}{\partial w}$
   (iii) update parameters $w \leftarrow w - \eta \frac{\partial L(w)}{\partial w}$
   } epoch
3. Return parameters.

In Batch GD with N samples (data points)

$$L(w) = \frac{1}{N} \sum_{i=1}^{N} L_i(w)$$

$L_i(w)$: loss for i^{th} data point

$$L(w) = E\left(L_i(w)\right) = \frac{1}{N}\sum_{i=1}^{N} L_i(w)$$

So if we calculate loss for each sample ⟹ Loss for batch
↓
and take expected value

[I calculate loss for one sample = $L_j(w)$
do parameter update

[do it again for another sample = Continue doing for each individual samples
same impact on parameters.

→ Each update would require only one gradient computation
(less time, less storage for gradient values)
↓
better than this

Stochastic G.D :-

1. initialize parameters
2. loop until convergence
   (i) Randomly shuffle samples in training dataset
   (ii) For training sample $i = 1, 2, 3 \cdots N$
      (a) compute gradient $\frac{\partial L_i(w)}{\partial w}$
      (b) update parameters $w \leftarrow w - \eta \frac{\partial L_i(w)}{\partial w}$
   (iii) Return parameters

epoch.

BGD → not data efficient when data is similar
→ high computational time for each step

SGD → [not computationally efficient
noisy result since at a time, one data point is considered.
↓ Solution

MiniBatch GD →

1. Initialize parameters
2. Loop until convergence
   (i) Randomly shuffle samples in training dataset
   (ii) From training dataset, Create b minibatches of size m
   (iii) for each mini batch $i = 1, 2 \ldots b$
      (a) Compute gradient $\frac{\partial L_i(w)}{\partial w}$
      (b) update parameters
      $$w \leftarrow w - \eta \frac{\partial L_i(w)}{\partial w}$$
3. Return parameters

Adaptive learning Rate + L.R :- How much gradient update at every step.

Do we want complete or shorter step update.

→ larger step might miss Minima.

→ Smaller takes longer period of time.

There should be some trade-off for learning rate.

Time dependent learning rate :—

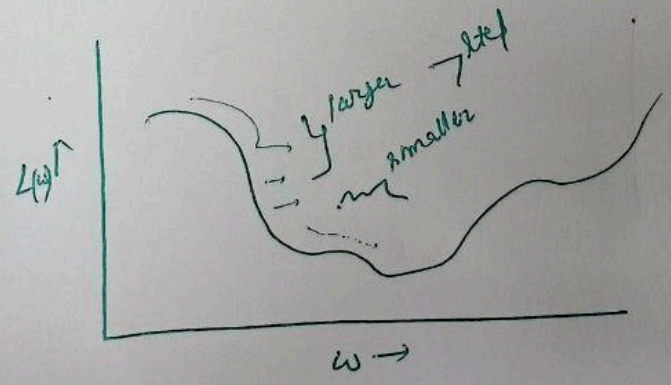
$$\eta(t) = \eta_i \quad \text{if } t_i \le t \le t_{i+1}$$ piecewise constant

$$\eta(t) = \eta_0 \cdot e^{-\lambda t}$$ exponential decay

$$\eta(t) = \eta_0 \cdot (\beta t + 1)^{-\alpha}$$ polynomial decay

initially l.r → larger step

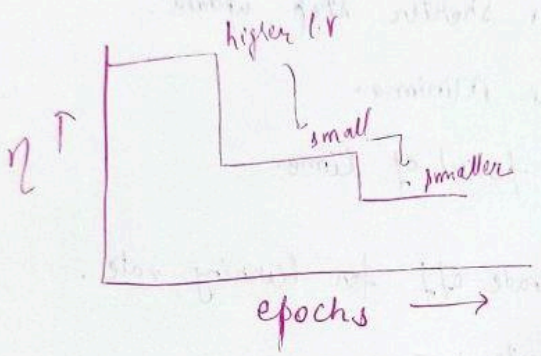
near minima → smaller step.



(a) Interval wise :-

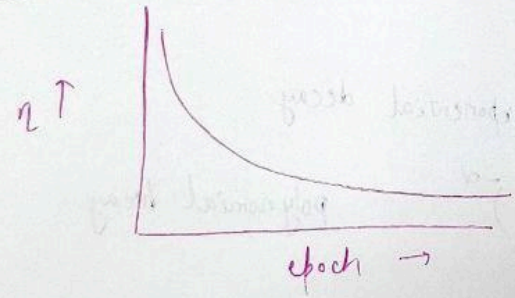
eg:- 10^{3} , 10, 11 . . . . , 10^{-2}

such kind of l.r in intervals.



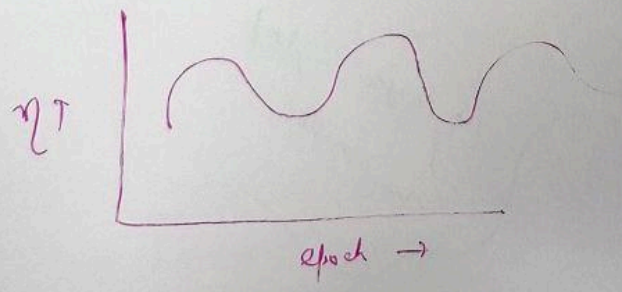
= linear piecewise

(b) exponential :-



∴ with ↑↑ in epoch
↓
l.r ↓↓

(c) some times Cosine L.R :-



Why NN :- It can be applied to all fields / problems

→ Something applicable and gives good result across different problem.

- In ML we mostly try to find curve.

Universal Approximation Theorem :-

→ Even wid single layer of NN, NN can approximate any fun^n, wid arbitrary precision.

Single layer → a lot of neuron

Multilayer Perceptron :- In (MLP) node of previous layer connected to all future.

(I/p layer hidden layer o/p layer)

- Single layer may need huge no. of node.
  having that much wider n/w is very hard.
- Better to take deeper n/w wid less wider n/w.

UAT :- A feedforward n/w with linear o/p layer and atleast 1 hidden layer with any "squashing" activation fun^n i.e logistic sigmoid activation fun^n.

↓

Can approximate any fun^n

↓

from one finite dimensional space to another

↓

with any desired non-zero amount of error

↓

provided n/w is given enough hidden units.

[Single perceptron ~~any~~ can't capture Temporal sequence]

Representation learning :-

FNN data is represented in vector form.

→ Pixel wise operation might not give good result of image.

Case :- Based on prior knowledge

↓

extracting handcrafted feature is hard.

. Hand crafted feature are useful in ML if correct features are extracted.

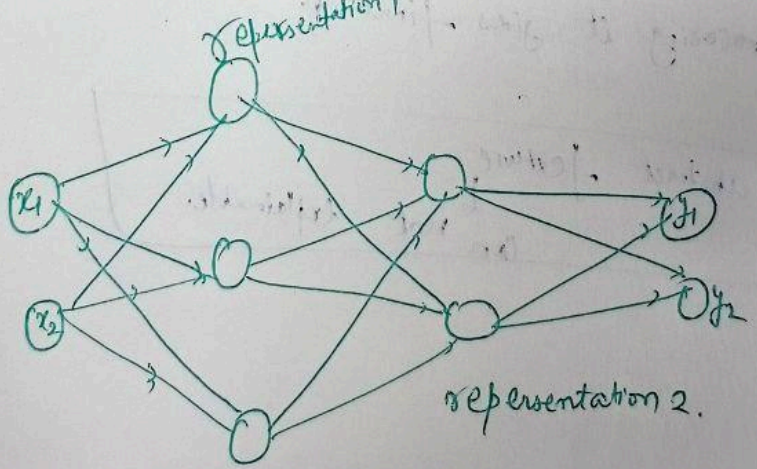
It's often difficult to know correct features (specially images, audios, videos etc)

→ Various NN (RNN, CNN etc) not only map i/p data to o/p
  ↳ but also learns a representation of data in various layers.

Representation learning ↓ don't have to rely on prior knowledge.
↓
apply raw data, model itself will extract feature

→ Instructing meaningful data

Raw data space → learnable feature space



x_1, x_2 = 2 features.

Linearly Combining fun^n → Activation fun^n

- Initial layers learn low level explainable feature
  (eg: edge, corner, texture etc)
- Deeper layer learns high-level abstract features
  (by combining suitable low-level features)

→ Top node of layer : getting some form of modified / derived feature

layer 2 : getting more derived feature
layer 3 : getting more " "

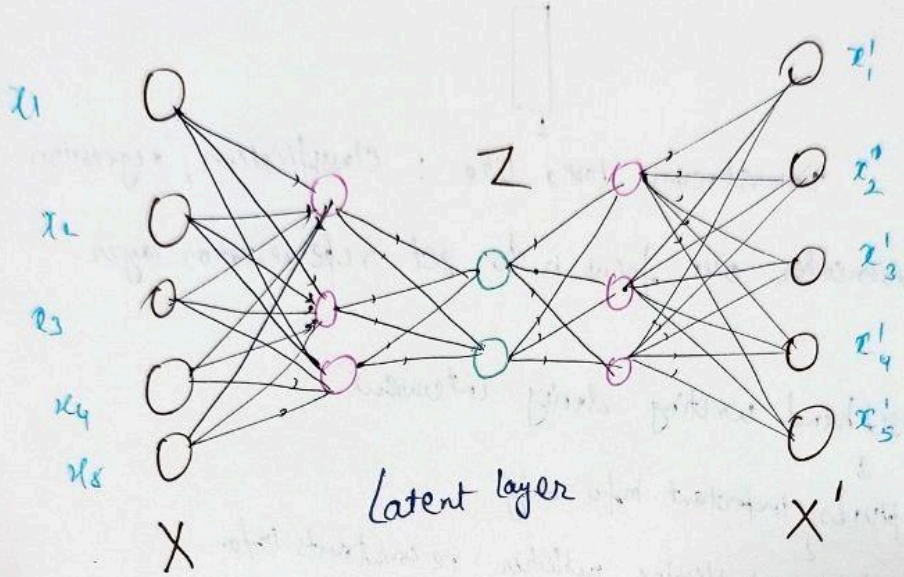
[Derived feature vector : Modified representation of raw i/p data.]

→ Each layer given representation of i/p data.
Processing it gives final o/p.

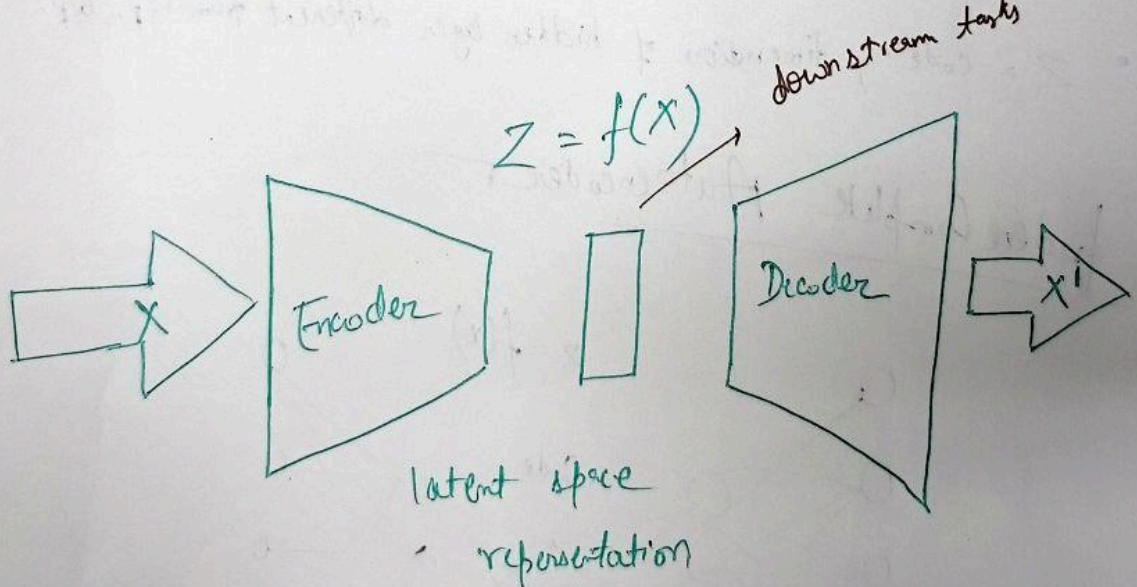
abstract feature
↓
can not explainable.

Auto encoders ←

$X' = g(z)$



$Z = f(x)$: Latent space representation / code



- Encoder creates code Z from X
- Decoder reconstructs X from Z. so $X' = X$ ideally.

Representation space → $Z = f(x)$

Downstream tasks like : classification, regression

. In Autoencoder our focus is to get representation layer.

⋮

eg: Shorthend writing during interview

↓

captures important info only

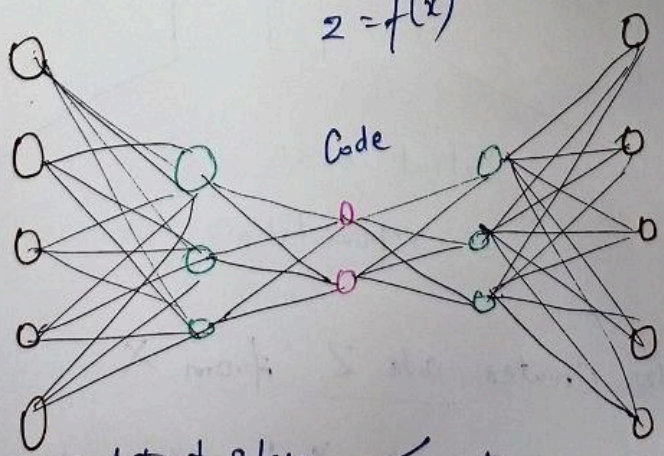
↓

during publishing publisher reconstructs info.

Having good Z only → can get X'

- Z = code, dimension of hidden layer different from i/p, o/p.

Under Complete Autoencoder



dimension of latent space < dimension of i/p / o/p space.

Usages :- (i) may be used for dimensionality reduction / Compression of data.

(ii) Extraction of salient features

$$\text{Loss} = \text{MSE}(X, X')$$

- Minimization of this loss requires X' to be as similar as to X.
- In order to do, Z must contain salient features.

Extraction of Salient features :-

eg:- To diff among zebra and dog we need combination of features (may be linear) alone 1 feature can't differentiate them

x_1 : weight
x_2 : Height
x_3 : major color 1
x_4 : major color 2.
x_5 : Repetition of color pattern (yes/no)

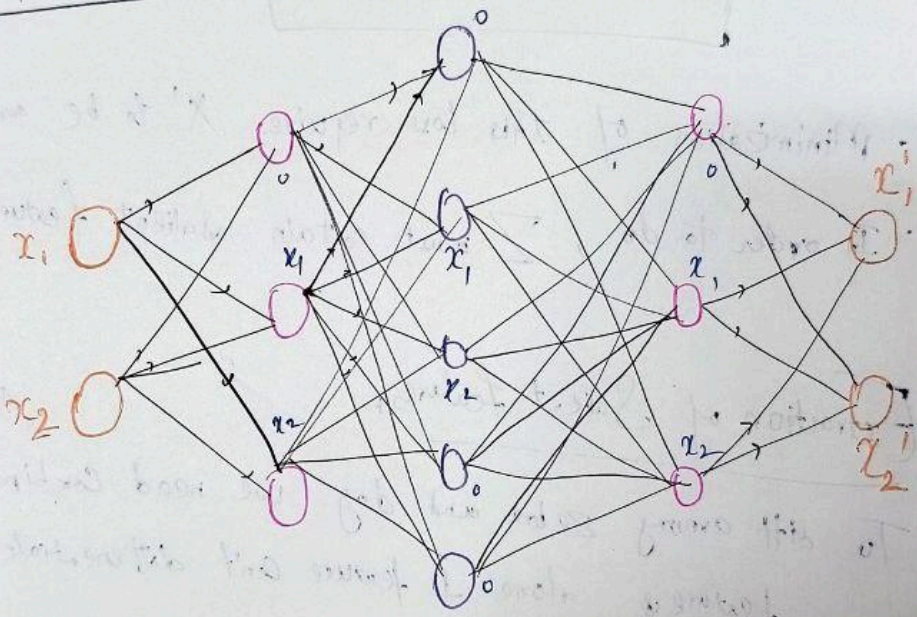
→ we are not using label here.

Self Supervised learning :- data itself teaching / data teaching

- This blending of features is created by AE in latent space.
- Latent space creates a salient representation of data.

Over Complete Autoencoders :-

Code $Z = f(x)$



Dimension of latent space > dimension of i/p, o/p space

latent space ↑↑

→ It might copy data

→ might not learn anything new then why we need it?

(i) Better Feature learning

→ more neuron in hidden layer than i/p forces to learn more hierarchical, abstract or sparse representation of data.

→ Rather than directly copying data
↓
It learns to represent data in more informative, compressed form that captures underlying structure.

(ii) Regularization, Sparsity :-

By L_1 regularization, sparsity autoencoders where only a few units in hidden layer are active at any given time.

→ It helps in reducing overfitting, forces to learn more compact, efficient features.

(iii) Non-linearities and complex representations :-

It can learn complex, non linear mapping from i/p to hidden layer

→ It allows to create a higher-dimensional representation of data, making it more effective, generalized.

(iv) Improved generative modeling

(v) Generalization to unseen data