$J_R = J + \lambda \|w\|^2 + 1000 w_3^2 + 16 w_4^2$

$\|w\|^2$ → small. w_3^2 → 0, w_4^2 → 0 (to remove a lot of dimensions / features)

— computationally good at the test time.

$W \rightarrow$ sparse (lot of elements = 0, supress features) → Find w as sparse as possible.

can use L_1 norm.

✓ L_2 norm (easier) ⇒ this to make it small ($\|w\|^2$)

⇒ $(x_i, y_i)_{i=1,\dots,N}$ (Data)

$x_i \in R^d$, Nord (large)!? (mostly N).

What if d is large → start with a large X (d measurements) (but not optimal)

* Feature selection ≡ subset selection
  - (given d features, find a subset of those features reducing the cost).
  - → Backtracking
  - — No efficient solution. (polynomial)

not used ≡ because we work with large set of dimensions.

Dimensionality Reduction ⇑

* Feature Extraction ≡ $z = Ux$ (x: initial features ≡ d)

  z: $K \times 1$, U: $K \times d$, x: $d \times 1$.

  U → linear combination of d features

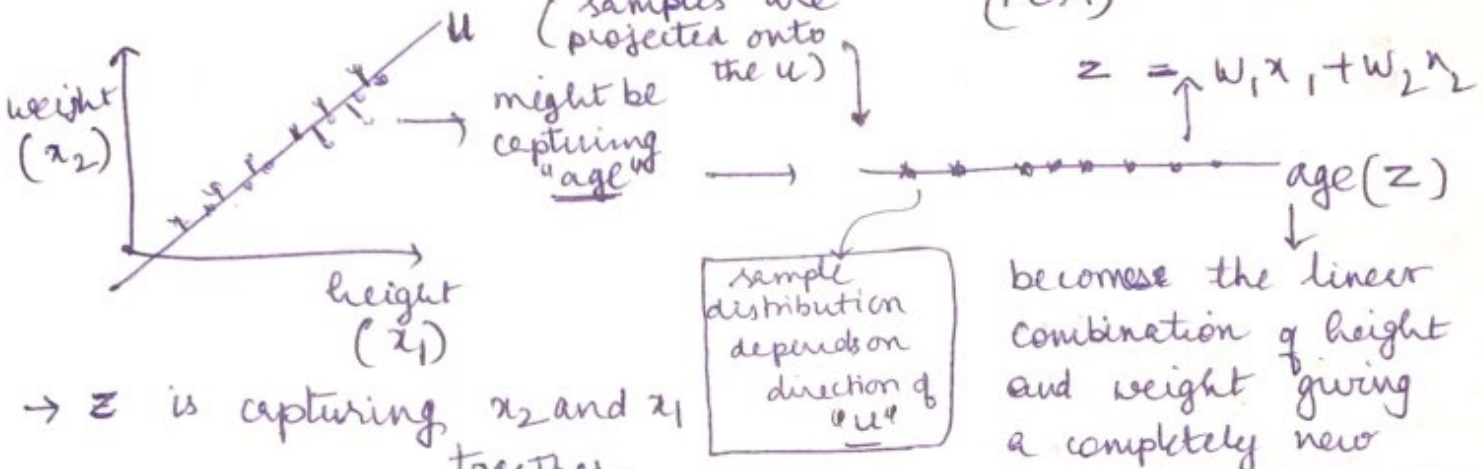
  z → new K features

  $[K < d]$

  - doesn't reduce the cost [because we utilize the combination of all the features]
  - may be useful for low computational devices

Feature extraction — lower computational cost
+
lower dimensional representation.

Dimensionality Reduction = Principal Component Analysis (PCA)



→ z is capturing x_2 and x_1 together.

" u ≡ direction where maximum variance is preserved "

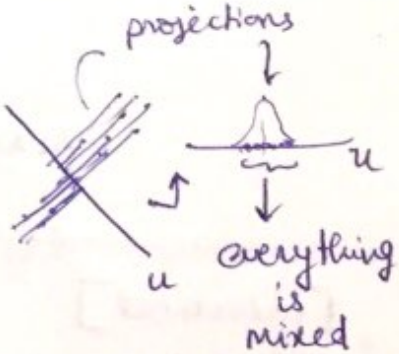
New mean of the data (projections of the samples) ;

$$\bar{Z} = \frac{1}{N}\sum_i z_i = \frac{1}{N}\sum_i^N u^T x_i$$

↓ independent of i.

$$= u^T\left(\frac{1}{N}\sum_i^N x_i\right)$$

$$= u^T\mu$$



If the line is this way, so our objective is to maximize the variance;

$$\text{Max } \frac{1}{N}\sum_{i=1}^{N}(z_i - \bar{z})^2$$

$$\frac{1}{N}\sum_{i=1}^{N}(u^T x_i - u^T\mu)^2$$

$$\frac{1}{N}\sum_{i=1}^{N}(u^T x_i - u^T\mu)(x_i^T u - \mu^T u)$$

$$= \frac{1}{N}\sum_{i=1}^{N} u^T [x_i - \mu][x_i - \mu]^T u$$

$= u^T \Sigma u$, such that $u^T u = 1$ (Solution ≡ an eigen vector)

Σ → covariance matrix = d×d matrix

u is the eigen vector of Σ, corresponding to the largest eigen value, λ.

eigenvectors - orthogonal,

largest eigen value, vector = u_1

'eigen vector matrix' ← $U = [u_1, u_2, \ldots, u_K]$

such that when projected onto this, variance is maximized.

K such directions onto which samples can be projected.

Eigen face = learning a feature

concatenated

N faces of people. N = 1000.

100 × 100 ≡ 10^4 features

$\Sigma_{10^4 \times 10^4}$

$\sum_{1000}^{1000} (x_i - \mu)(x_i - \mu)^T \equiv$ λ_1 λ_2 ... 1000 eigen vectors (Non zero)

$\mu = \frac{1}{N}\sum x_i$

rearrange back as a rectangle.

(also looks like a face but blurred)

[identical]
If there were correct alignment (height & weight)
↳ we would have got single λ_1 and then $\lambda_2 \ldots = 0$.
but not the case here due to redundant, multiple features.

$$z = \begin{bmatrix} x^T u_1 \\ x^T u_2 \\ \vdots \\ x^T u_{100} \end{bmatrix} \begin{matrix} \alpha_1 \\ \alpha_2 \\ \\ \alpha_{100} \end{matrix}$$

relevant

~100

goes to zero.

(as we consider similar features)

these features actually represent you.

These 100 scalars — enough to reconstruct any face.

$$X_{[new]} = \sum_{i=1}^{100} \alpha_i u_i$$

u_i → global basis for faces. (we have 100 u_is) [common for everybody]

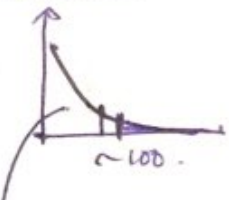
α_i → signature of your face.

$$X = \sum_{i=1}^{10000} \alpha_i u_i = \alpha_1 u_1 + \alpha_2 u_2 - \cdots + \alpha_{100} u_{100} + \cdots \alpha_{10000} u_{10000}$$

changes to

$$= \sum_{i=1}^{100} \alpha_i u_i \longrightarrow$$ all you need is these 100 values.

$\alpha_{100} \cdots \alpha_{10000}$ can be discarded. because ⇒



decay effect ⇒ "only for similar features not for scattered parameters".

~ compression (within minimal loss)

↓ (basis = eigenvector).

KLT (eigenvector basis)

this is data dependent (if it is faces, then one set of u_is else some other set of u_is.)

→ domain is narrow, then we can learn the basis.

For Eg: person specific.

Other Defn → Find u such that we get minimum reconstruction error.

in this case, $N(10^3) < d(10^4)$., $\Sigma_{d \times d}$.

$X_{N \times d}$, $XX^T = N \times N$ $\boxed{XX^T u = \lambda u}$ —①

$X^T X = d \times d$ $\boxed{X^T X v = \lambda v}$ —②

Can we compute eigen vector of one from the other?

$$X^T X (X^T u) = \lambda (X^T u)$$
$$\downarrow v \qquad\qquad\qquad \downarrow v$$
$$X^T X\, v = \lambda v$$

$\Rightarrow$ v can be computed from u
$(d \times d)$ $(N \times N)$

$N \times N \rightarrow u \equiv 1000$

$\downarrow$

$x^T u$

$\downarrow$

$v = x^T u$

$\downarrow$

Computationally attractive. better than $[d \times d]$

while considering eigen values;

$$\rightarrow \frac{\sum_{i=1}^{K} \lambda_i}{\sum_{I=1}^{d} \lambda_i} > 95\%$$

↳ if it is 100% (we are picking all the values)

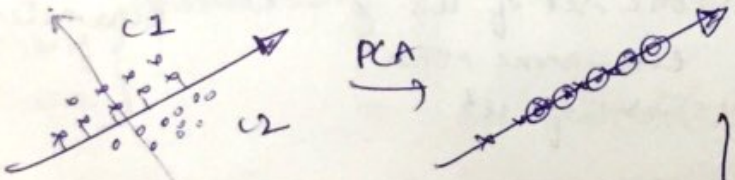
↓ can have nonlinear schemes.

computed and derived from data

PCA — "unsupervised".

↳ class information is not taken into account.

If there are 2 classes;



↓ but the other way,

works for compression

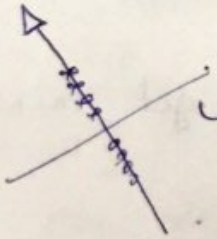
PCA fails miserably

↓

in the case of "classification" problems

↓

use "supervised"



↳ does the classification

"supervised" — uses class information

↓

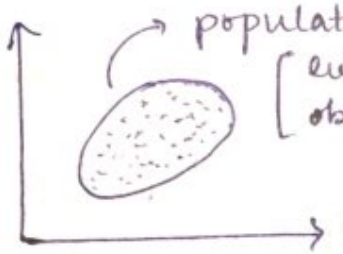
LDA / Fisher.

$z = Ux$ (linear)

⇓

can be replaced by nonlinear (Non-Linear Dimensionality Reduction)

Parameter Estimation → after learning how apples and oranges look like, how probable that my understanding is correct?



→ population [every apple you could ever observe in the world (problem)]

→ have some model in mind

Ex: distribution of apples is an ellipse/ fitting line.

Model: Line.

→ Goal – estimate w

* Params: $\langle w_0, w_1, w_2 \rangle^T$

Population → not observable

(you hardly have the full population in any problem, if you have, then the classification becomes easy).

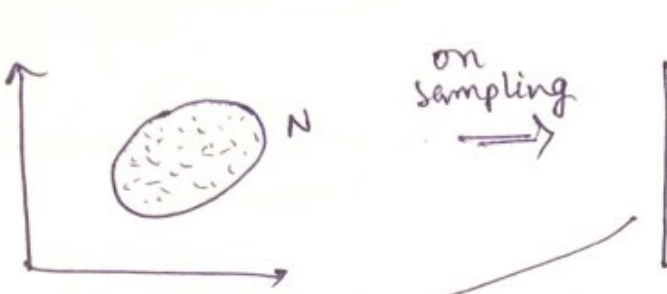
Randomly pick up some samples [if I want apples ∈ Ejachibouli, go to all shops in Ejachibouli]

* Any sample has same probability of getting picked (pick uniformly) ⇓ "Uniform sampling".

→ * Not distribution of samples is uniform but "sampling process" is uniform

→ out of a bag of "n" apples

uniformsampling = probability of picking an object will always be same – $1/n$



Uniform sampling – Put all apples in a bag, pick one.

samples

Distribution of this tends to follow the original distribution

Parameter Estimation

model

Model = Line / Ellipse

Params $\langle w_0, w_1, w_2 \rangle$

[Case of rare apples]

$$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 1.2 \\ -3.1 \end{bmatrix}$$

↑ once this is known, I can do classification / line fitting whatever.

goal: finding out the values of w_0, w_1, w_2 of the population, but we can observe only samples.

(population ↔ parameters corresponds to samples)

Two approaches:

1. Compute θ (vector) parameters to be estimated that best explains D (samples – whatever we observe)

   [frequentist approach] **MLE**

   ↓ set of training samples.

   Called as maximum likelihood estimate (MLE)

$$\hat{\theta} = \arg\max_{\theta} P(D/\theta)$$

   find that θ that maximizes

[bayesian approach]

2. Use bayes theorem to compute $\hat{\theta}$, ("case of prior belief")

MAP

$$P(\theta/D) = \frac{P(D/\theta) \cdot P(\theta)}{P(D)}$$

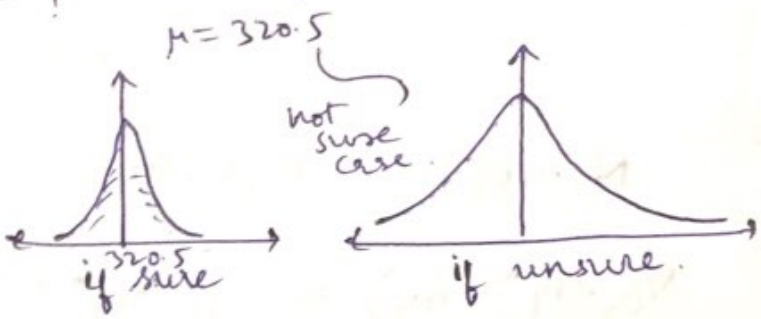
$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\ P(D/\theta) \cdot P(\theta)$$

- $\hat{\theta}$ → would also be a density function
- $P(D/\theta)$ (density fn)
- $P(\theta)$ (density fn) → prior belief is being combined with what you observe.
- MAP = Maximum Apriori Estimate

$\theta \Rightarrow \mu \equiv$ mean of apples (if normal distribution)

$P(\theta) = ?$

In bayesian world, everything is a density function, (nothing is a scalar)



MLE =

$$\underset{\theta}{\operatorname{argmax}}\ P(D/\theta) = \prod_{k=1}^{n} P(x_k/\theta)$$

(x_k → each sample.)

Taking log ⇒

$$= \sum_{k=1}^{N} \log\left(P(x_k/\theta)\right)$$

differentiate wrt θ,

$$\nabla_\theta l = \sum_{k=1}^{N} \nabla_\theta \ln P(x_k/\theta) = 0 \rightarrow \text{zero vector.}$$

θ is not a scalar, but a vector

Model ≡ Normal Distribution.

Case 1: $N(\mu, \Sigma)$ (Σ → known) → can we estimate μ?

(N → model)

Likelihood : $\ln P(x_k/\mu) = ?$

$$\Rightarrow \ln(\ldots) - \frac{1}{2}(x_k - \mu)^T \Sigma^{-1} (x_k - \mu)$$

↓ independent of θ

$$\nabla_\mu \ln(P(x_k/\mu)) = 0 - \frac{2}{2}(-1)\Sigma^{-1}(x_k - \mu)$$

$$= \Sigma^{-1}(x_k - \mu)$$

Normal distrib:

$$e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} * \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} =$$

$$\Rightarrow (2\pi)^{-d/2} |\Sigma|^{-1/2} e^{\frac{-1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

$$\nabla_\mu \ln\left(P(x_k/\mu)\right) = \Sigma^{-1}(x_k - \mu) \Rightarrow * \quad \nabla_\mu \ln(P(x_k/\mu)) = \Sigma^{-1}(x_k - \mu)$$

$$\hat{\theta}_{MLE} \Rightarrow \sum_{k=1}^{n} (\Sigma^{-1})(x_k - \mu) = 0.$$

Σ → covariance ≡ "Σ" → gives an estimate of the world mean

$$\hat{\mu}_{MLE} = \sum_{k=1}^{n} (\Sigma^{-1})(x_k - \hat{\mu}) = 0.$$

$$\sum_{k=1}^{n} x_k - \sum_{k=1}^{n} \hat{\mu} = 0.$$

$$\hat{\mu}_{MLE} = \frac{1}{n} \sum_{k=1}^{n} x_k$$

step 2:

$N(\mu, \sigma^2)$ $\quad$ $N(\theta_1, \theta_2)$

$$\nabla_{\theta_2} \ln(x_k/\theta_2) = \begin{bmatrix} \nabla_{\theta_1} \\ \nabla_{\theta_2} \end{bmatrix}$$ → same as before.

$$-\frac{1}{2\theta_2} + \frac{(x_k - \theta_1)^2}{2\theta_2^2}$$

$$\sum_{k=1}^{n} -\frac{1}{2\theta_2} + \frac{(x_k - \theta_1)^2}{2\theta_2^2} = 0.$$

$$\frac{1}{2\theta_2^2} \sum_{k=1}^{n} (x_k - \theta_1)^2 = \frac{n}{2\theta_2}$$

multiply both sides by $\left(\frac{2\theta_2^2}{n}\right)$

$$\hat{\theta}_{2\,MLE} = \frac{1}{n} \sum_{k=1}^{n} (x_k - \hat{\theta}_1)$$

$$= \frac{1}{n} \sum_{k=1}^{n} (x_k - \hat{\mu})^2$$

for normal* densities

∴ mean of population = mean of samples*

As the sampling changes, $\hat{\mu}$ will change (will move around)
(take another 100 samples)

What is the distribution of $\hat{\mu}$?

↓ what will be μ of $\hat{\mu}$? will it converge to the actual μ?

↓ Expected value of $\hat{\mu}$.

$$\underset{\text{(average)}}{\mathcal{E}(\hat{\mu})} = \mu \left(\begin{matrix}\text{actual } \mu \\ \text{of popl}^n\end{matrix}\right)$$

* [when n (no. of samples that u take) increases, $\hat{\mu}$ changes very little.]

$$\underset{\text{(average)}}{\mathcal{E}(\hat{\sigma}^2)} \neq \sigma^2 \quad \text{(variance is diff)}$$

actually, $\mathcal{E}\hat{\sigma}^2 = \frac{n-1}{n}\cdot\sigma^2$ (Slightly smaller than σ^2)

* MLE, $\hat{\sigma}^2$ is a biased estimate. (we have estimated a smaller one)

MLE, $\hat{\mu}$ is unbiased estimate.

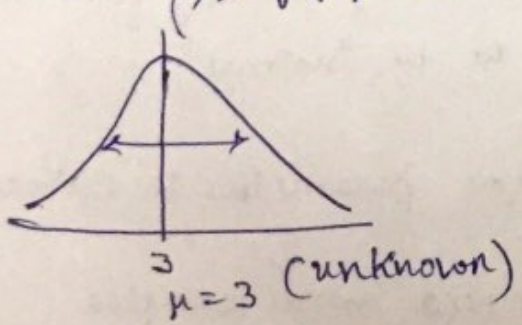
u-MLE

$$\Rightarrow \hat{\sigma}^2 = \frac{1}{n-1}\sum_{k=1}^{m}(x_k - \hat{\mu})^2$$

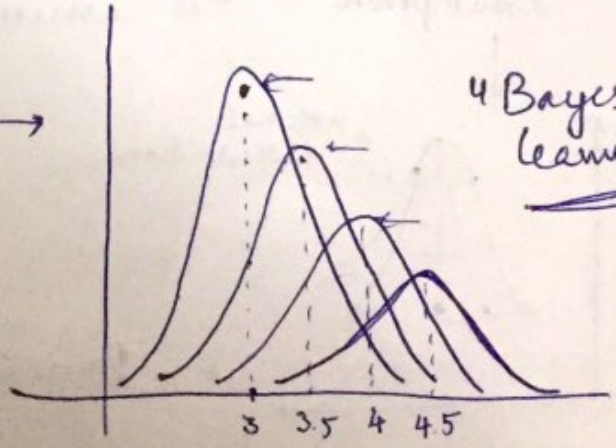
↳ increase slightly becomes.

↳ unbiased estimate.

MAP-



prior belief = 4.5, ↓ it moves towards 3

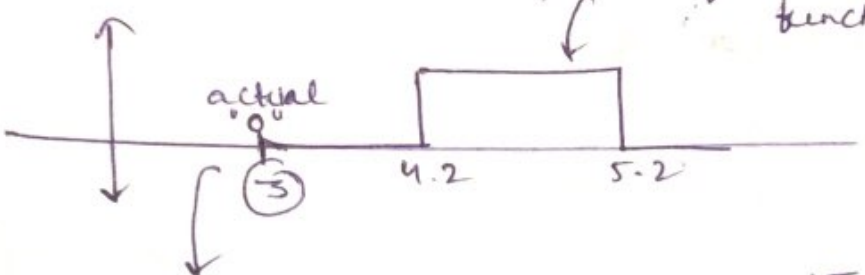


"Bayesian learning"

→ as you start observing more and more samples, μ goes toward [actual μ].

If prior belief is this way →

no matter how much data is observed, this would at max converge to 4.2



If your prior belief is 0, at a particular point

Then posterior at that point is also 0

↓

and you'll never reach 3 in this case.

* If your prior belief is bad, you'll never learn.
  - Be slightly skeptical about your beliefs :-

"Wise Man is never confident"

Bayesian Parameter Estimation ≡

recap:

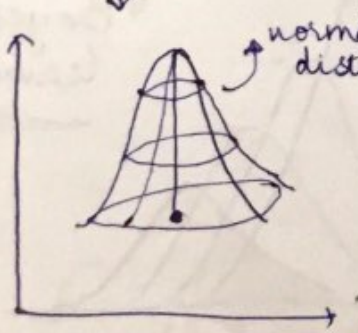
$$P(\omega_i/x) = \frac{P(x/\omega_i)\,P(\omega_i)}{P(x)}$$

→ observing x, such that we're talking about apples, ω_i

$P(x/\omega_i)$ ← generative model.

to compute $P(x/\omega_i)$ → we use 2 approaches where the nature of probability density function is assumed to be "normal",

strong assumption



more parameters to estimate ↕ need more samples.

Bayesian decisions ≡ optimal ↓ based on the assumed nature of the function.

Bayes doesnot work well because our assumptions are not always correct. of estimates.

In the case of 4000 parameters,
$\mu[4000]$, covariance matrix $[4000 \times 4000]$ → we will be need a minimum of 80 million samples to estimate.

MLE:

$D = \{x_1, x_2 \ldots x_n\}$

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

$p(\theta) = \underset{\theta}{\text{argmax}}\ P(D/\theta)$

$= \underset{\theta}{\text{argmax}} \prod_{k=1}^{n} P(x_k/\theta)$

$\hat{\mu}_{MLE} = \frac{1}{n}\sum_{k=1}^{n} x_k$

$\hat{\sigma}^2_{MLE} = \frac{1}{n}\sum_{k=1}^{n}(x_k - \hat{\mu})^2$

} MLE estimates of μ, σ^2

MAP:

$\hat{\theta}_{map} = \underset{\theta}{\text{argmax}}\ p(D/\theta) \cdot p(\theta)$

→ θ – random variable [density function].

$P(\theta/D)$ → could be a density function.

density functions of parameters → posterior [density function] → maximum of the probs.

$P_\theta(x/\omega_i) \sim N(\mu, \sigma^2)$

$P(\mu) \sim N(\mu_0, \sigma_0^2)$ [Prior belief]

$P(\mu/D) = P(D/\mu) \cdot P(\mu)$.

x = weights of apples. → it should be estimated from samples "prior info".

Let $N(\mu, \sigma^2)$ — μ: 250 g, $\sigma = 50$ (absolute) – not known

$P(\mu) \sim N(\mu_0, \sigma_0^2)$ → variance of the average weights.

- expert knowledge
- belief about the average weight of apples
- → uncertainity in the understanding about the average (μ_0) of apples

"Bayesian learning"

On observing more and more apples (samples), the Σ (actual μ, estimated μ) becomes lesser and lesser.

Case 1: $P(x/\mu) = N(\mu, \sigma^2)$, σ^2 is known

$P(\mu) \sim N(\mu_0, \sigma_0^2)$ [prior understanding of μ]

$$P(\mu/D) = \frac{P(D/\mu) \cdot P(\mu)}{\int P(D/\mu) \cdot P(\mu) \cdot d\mu}$$

→ likelihood of observing data set. ~ product of samples.

→ constant.

$$= \propto \prod_{k=1}^{n} P(x_k/\mu) \cdot P(\mu)$$

both are normal density of μs.

Posterior (Normal density)

~ Reproducing densities.

{Posterior ~ Prior ~ Normal}

→ shape remains the same

$$= \propto \cdot \prod_{k=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x_k - \mu}{\sigma}\right)^2} \cdot \frac{1}{\sqrt{2\pi}\sigma_0} e^{-\frac{1}{2}\left(\frac{\mu - \mu_0}{\sigma_0}\right)^2}$$

observations ↓ prior.

$$P(\mu/D) = \frac{1}{\sqrt{2\pi}\sigma_n} e^{-\frac{1}{2}\left(\frac{\mu - \mu_n}{\sigma_n}\right)^2}$$

understanding of μ after observing samples.

$P(\mu/D) \Rightarrow (\mu_n, \sigma_n)$ = updated values

$$\mu_n = \frac{n\sigma_0^2}{n\sigma_0^2+\sigma^2}\hat{\mu}_n + \frac{\sigma^2}{n\sigma_0^2+\sigma^2}\mu_0$$

$\hat{\mu}_n$ → MLE estimate "observation"

μ_0 → prior

$\frac{n\sigma_0^2}{n\sigma_0^2+\sigma^2}$

What happens when n increases? → σ (actual variance)

(more and more samples)

if $\sigma \uparrow$, you need more n.

$\frac{n\sigma_0^2}{n\sigma_0^2+\sigma^2}$ → goes to 1

$\frac{\sigma^2}{n\sigma_0^2+\sigma^2}$ → decreases

[goes from $\mu_0 \to \mu_n$]

$$\sigma_n^2 = \frac{\sigma_0^2\sigma^2}{n\sigma_0^2+\sigma^2}$$

[uncertainity after observing n samples]

more and more you observe, lesser will be the samples.

uncertainity.

become sure about the distribution.

[uncertainity reduces]

- Parameter update can be done per sample also. ~ batch update. → gives same convergence

Recursive ≡

$$P(D^n/\theta) = P(x_n/\theta) \cdot P(D^{n-1}/\theta)$$

$$P(x/D) = \int P(x/\mu) \cdot P(\mu/D)\, d\mu$$

real estimate.

plugging in the uncertainity

$$\sim N(\mu_n, \sigma^2 + \sigma_n^2)$$

Known, uncertainity in μ

[true bayesian]

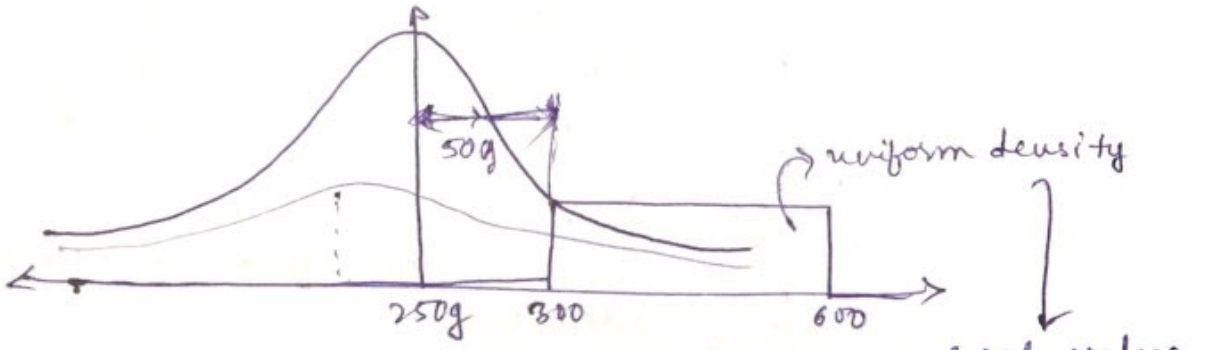
Limiting case, more samples, $\sigma_n \to 0$. $\Rightarrow$ MAP estimate will be fine.

$\sim N(\mu_n, \sigma^2)$.

(prior belief)

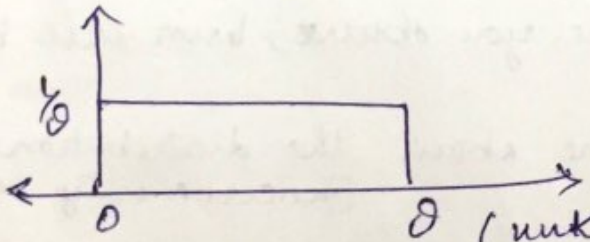
In bayesian, whatever may be the density function, it will converge upon observing more and more samples.

except for uniform density function. (prior belief)



Ex: Special Case –

$\Rightarrow P(x/\theta) \sim U(0,\theta) \begin{cases} 1/\theta, & 0 \le x \le \theta \\ 0, & \text{otherwise} \end{cases}$



Observed 4 numbers

$D = \{4, 7, 2, 8\}$

you cannot observe these samples if $\theta = 5$

$P(7/\theta) = 0$

what should be θ.

$P(4/\theta) \cdot P(7/\theta) \cdot P(2/\theta) \cdot P(8/\theta)$

Most Likelihood value of $\theta \equiv 8$

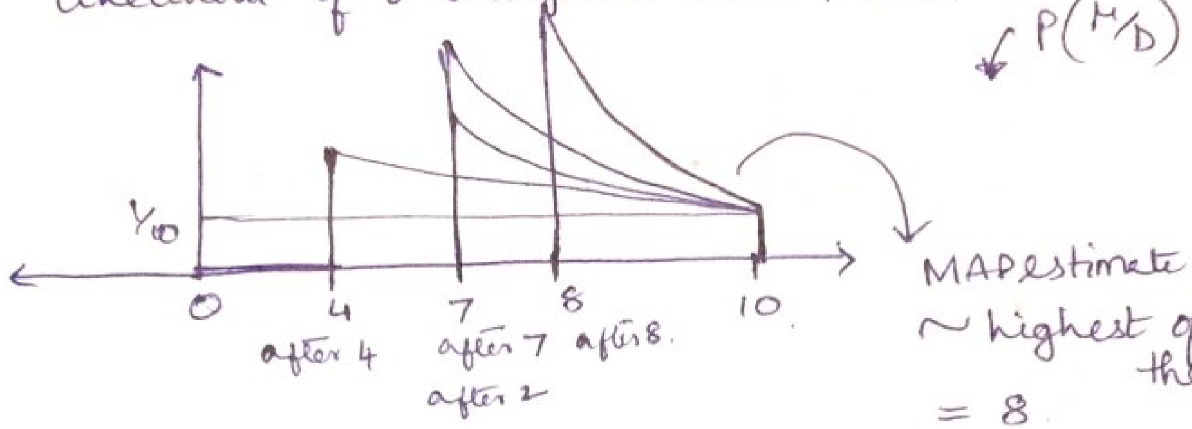
if $\theta = 7.9$ $\sim$ product $= 0$

max $\theta = 8$ $\sim \left(\frac{1}{8}\right)^4$ ✓

$\theta = 10$ $\sim \left(\frac{1}{10}\right)^4$

After observing 4, θ value should be greater than 4

likelihood of θ being less than 4 = 0.



MAP estimate ~ highest of all these = 8.

After observing 4, MLE = 4

" 7, MLE = 7

2, MLE = 7

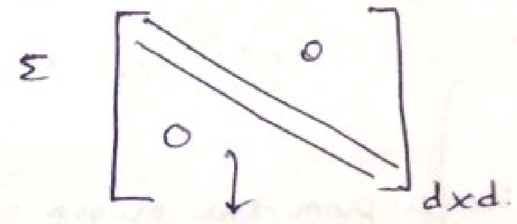
8, MLE = 8.

Naive Bayes Classifier ≡ Features are independent of each other.

Assume that Σ (covariance) ≡ diagonal matrix

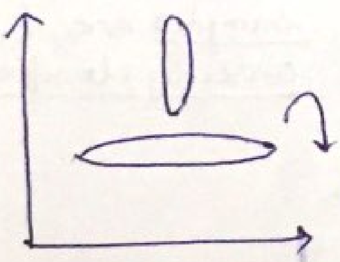
$\mu \rightarrow d \times 1$ (reduced to)

(when, Parameters↑ Samples↓)



↓ off diagonal = 0.

— correlations of the covariances.



$P(x) = P(x_1).P(x_2) \ldots P(x_d)$ → can independently model, "deal with one feature at a time"

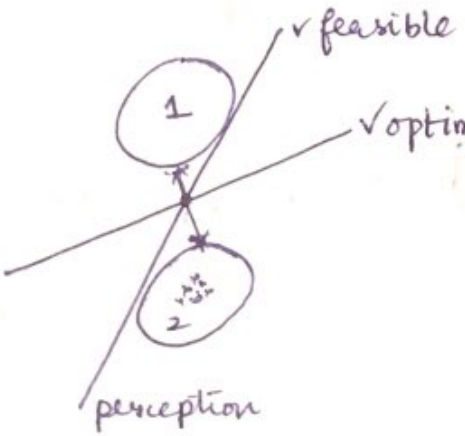
↓ this would cause the no. of parameters to estimate to come down.

in the case of apples; {colour, weight} → can be modelled separately.

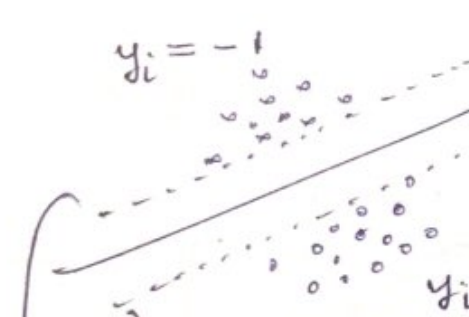
$P(x) = P(x_1).P(x_2) \ldots P(x_d)$

can be density functions (need not be of the same form).

Hyperplane → such that the margin between classes is large ↔ desirable.



Linear SVMs — Support Vector Machines. ≡ max margin



+1, -1 can be any quantity
↓
but equal margin

Objective ≡ to maximize the margin such that samples are correctly classified

distance from the origin ≡ $\frac{b+1}{\|w\|}$

distance from the origin ≡ $\frac{b-1}{\|w\|}$

$$\frac{b+1}{\|w\|} - \frac{(b-1)}{\|w\|} = \frac{2}{\|w\|} \equiv \text{margin (has to be maximized)}$$

$$w^T x_i + b \geq +1 \quad , y_i = +1$$
$$w^T x_i + b \leq -1 \quad , y_i = -1$$

combining both, we can write

$$\Rightarrow \quad y_i (w^T x_i + b) \geq 1 \quad \forall i$$

$\frac{2}{\|w\|}$ has to be maximized, → can be converted to minimization problem ⇒

$$\text{Min } \frac{1}{2} w^T w.$$

$\min \frac{1}{2} w^T w$ $\equiv$ Primal objective.

$y_i (w^T x_i + b) \geq 1$ $\Rightarrow$ **Hard Margin SVM**

$y_i \in \{+1, -1\}$

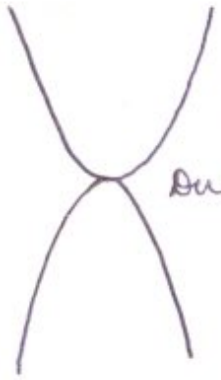
↳ can be solved using gradient descent*

Advantage of SVMs $\equiv$ Convex optimisation problems can be solved using SVMs.

↓

[not worrying about local minima and all]

Optimisation problem –



Dual $\equiv$ $\max \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j$ N, no. of samples.

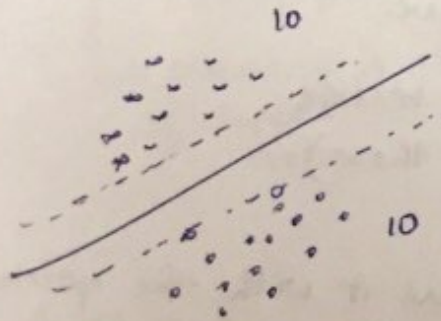
$\sum_{i=1}^{N} \alpha_i y_i = 0 \; ; \alpha_i \geq 0$

(quadratic programming solvers) – Not trivial

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

max margin separating hyperplanes. → always get the same solution using SVMs

↳ but using perceptron, we get different.



y_i – label
x_i – sample

if we take 10, 9 samples on each side and still get the same soln

↓

then utility of that sample = 0.

$[\alpha_i = 0]$

So, do all the points have $\alpha_i = 0$?

No, not for the case of boundary points, they decide the boundary margin.

$\alpha = 0$ points $\Rightarrow$ dont really matter in the solution

α vector - [sparse = for most of the samples values of α are zero]

SVMs $\rightarrow$ return non-zero "α"s $\rightarrow$ can be used to calculate optimal "w"

$$\max \left[\sum_i^N \alpha_i - \sum_i^N \sum_j^N \alpha_i \alpha_j y_i y_j x_i^T x_j \right]$$

$$w = \sum \alpha_i y_i x_i$$

sum it over the non-zero α's

if no. of non zero "α"s increase $\leftrightarrow$ difficult problem

SV $\equiv$ Support vectors (with non zero α)

decide the boundary.

$$w = \sum_{i \in SV} \alpha_i y_i x_i$$

(set of support vectors)

if 1 million samples $\rightarrow$ have 100 support vectors so,

do we only care about 100 support vectors?

No, we process all the samples, when we start $\rightarrow$ we dont know which are useful and which are not, Use the million to solve

$\downarrow$

we get 100 support vectors

(we care only about these)

For testing,

$$\Rightarrow \quad w^T x + b \lessgtr^{?} 0$$ (compare it with the optimal first line)

$$\Rightarrow \quad \underbrace{\sum \alpha_i y_i x_i^T}_{w} x + b \lessgtr 0$$

Computational Complexity → depends on support vectors.
[if $\underline{w}$ is not precomputed]

$$(w)^T x = \left(\sum \alpha_i y_i x_i\right)^T x = \sum \alpha_i y_i x_i^T x$$

these are scalars.

Samples come into the problem — as "dot product"

$(x_i^T . x_j)$.

scalar independent of the dimension of vector.

Number of Support Vectors ⇒

Error on the future value – (performance on the test data)

training set, test set

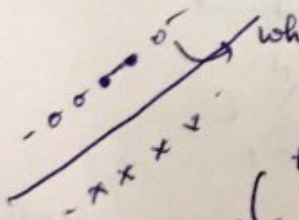
→ leave one out (TAKE 19 and train on them and test on the one out of 20)

Upper bound on the test Error ≡ $\left\{ \frac{\#SV}{N} \right\}$.

"estimate"

if support vector is left out, then we might get error.

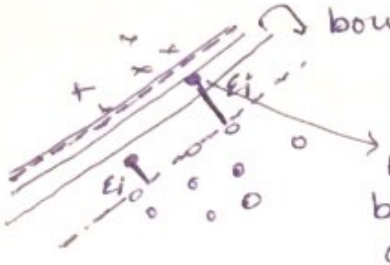
Doubt —



why take all the support vectors if all we need is 2 SVs to define the boundary?

→ this is 2 D data, (simplified) Data in reality is multidimensional.

→ pruning is also usually done of support vectors.



boundary has to be this way. (hard margin)

$\Leftarrow$ Linear separability is not the case. $\rightarrow$ have to include ξ_i else there wont be any soln possible

might be an outlier.

— Modify $\equiv$ $\{\xi_i$ has to be small.$\}$.

$$w^T x_i + b \geq +1 - \xi_i \qquad y_i = +1$$

allows some of the samples that are not linearly separable.

$\Rightarrow$ $y_i(w^T x_i + b) \geq 1$ $\rightarrow$ might not give solutions to all the samples. (linear inseparable)

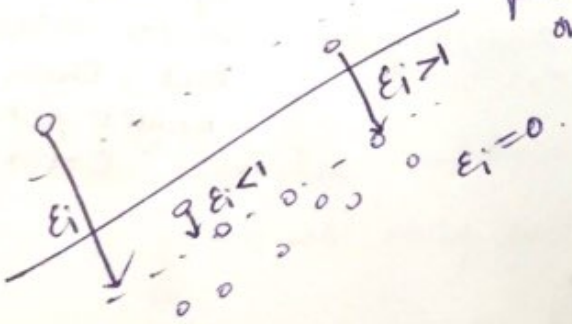
modify $\equiv$ $y_i(w^T x_i + b) \geq 1 - \xi_i$ $\rightarrow$ many samples will have ξ_i, non zero

$$\text{Min}\left\{\frac{1}{2}w^T w + \sum_{i=1}^{N}\xi_i\right\}$$

* allowing violations with penalty.

ξ_i has to be minimized also else any line is possible.

ξ_i (allowing a penalty)



penalty constraints on the samples.

* Soft Margin "C-constant"

$$\Rightarrow \text{Min } \frac{1}{2}w^T w + C\sum_{i=1}^{N}\xi_i$$

neither large nor small

— C has to be fixed $\rightarrow$ searched over a range.

*Tuning (power) type of penalty

$$\text{Min } \frac{1}{2}w^T w + C\sum_{i}^{N}\xi_i^K \rightarrow K = 1, 2, \ldots$$

L1 SVM

L2 SVM

:

Tuning no. of samples incurring penalty → "consider it to be an outlier"

Primal ⇔ Dual.

not considering penalty.

New objective for Primal — constraint is added.

$$\frac{1}{2}w^Tw - \sum_{i=1}^{N}\alpha_i\left[y_i(w^Tx_i+b)-1\right] = J(w,b,\alpha)$$

$\alpha_i = 0$ [beyond the plane]

$= zero$ [on the wall]

w^Tw $\{i\ j\}$ or they become indistinguishable.

$$\Rightarrow \frac{\partial J}{\partial w} = 0$$

$$\Rightarrow w - \sum_{i=1}^{N}\alpha_i y_i x_i = 0$$

$$\boxed{*w = \sum_{i=1}^{N}\alpha_i y_i x_i}$$

$$\Rightarrow \frac{\partial J}{\partial b} = 0.$$

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

$$J(w,b,\alpha) = \frac{1}{2}\left[\sum_{i}^{N}\alpha_i y_i x_i\right]^T\left[\sum_{j}^{N}\alpha_j y_j x_j\right] - \sum_{i=1}^{N}\alpha_i y_i w^T x_i - \sum_{i=1}^{N}\alpha_i y_i b + \sum_{i=1}^{N}\alpha_i$$

Can be written as $0 \left[\because \sum_{i=1}^{N}\alpha_i y_i = 0\right]$

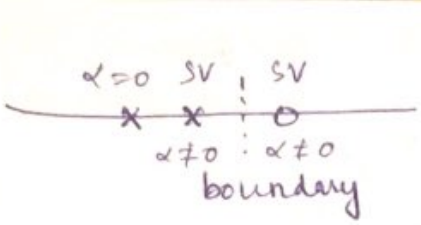
$$= \sum_{i=1}^{N}\alpha_i y_i x_i^T w.$$

$$= \sum_{i=1}^{N}\alpha_i y_i x_i^T \sum_{j=1}^{N}\alpha_j y_j x_j$$

$$= \frac{1}{2}\left[\sum_{i}^{N}\alpha_i y_i x_i^T\right]\left[\sum_{j}^{N}\alpha_j y_j x_j\right] - \sum_{i=1}^{N}\alpha_i y_i x_i \sum_{j=1}^{N}\alpha_j y_j x_j + \sum_{i=1}^{N}\alpha_i$$

Maximisation problem $= \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\alpha_i y_i x_i^T \sum_{j=1}^{N}\alpha_j y_j x_j$

$$= \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j x_i^T x_j \quad \text{(Dual)}$$



→ objective functions?

. Kernels and Non Linear SVMs ≡

$[W] = d$ (dimensions)

$[\alpha] = (\leq N)$

(sparse set)

Linear SVMs ≡

$$\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

$$\alpha_i \geq 0, \quad \sum_{i=1}^{N} \alpha_i y_i = 0.$$

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

$$w^T x = \sum \alpha_i y_i x_i^T x$$

$w^T x$: $O(d)$.

$\sum \alpha_i y_i x_i^T x$: $d \times \#SV$, $O(|SV| d)$

Linear dimensionality reduction:-

$x \longrightarrow x' = Mx$ ⇒ Computation easier

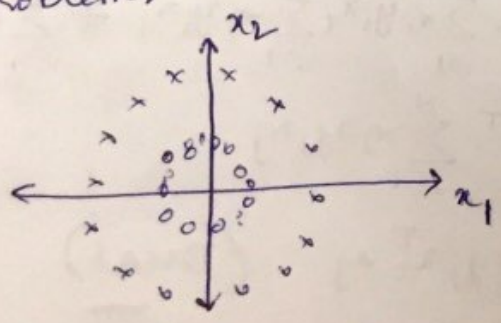
⇒ useless information is thrown away

can be nonlinear.

Dimensionality reduction from $x \longrightarrow \Phi(x)$ mapping (linear / nonlinear)

linear non separable problem.

Linear separable problem: → but not linear.



$\phi \equiv$ transformation (No inc/ decr)

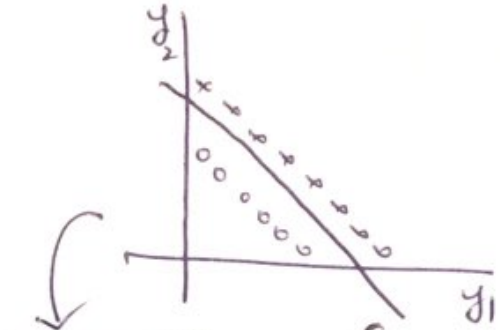
$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \xrightarrow{\phi} \begin{bmatrix} x_1^2 \\ x_2^2 \end{bmatrix} \begin{matrix} y_1 \\ y_2 \end{matrix}$$

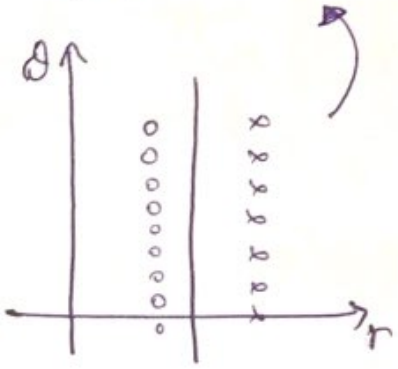
Upon transformation $\Rightarrow$ $y_1^2 + y_2^2 = r^2$

$y_1 + y_2 = r^2$

2 lines

$\begin{bmatrix} x \\ y \end{bmatrix} \xrightarrow{\phi} \begin{bmatrix} r \\ \theta \end{bmatrix}$





Linear SVM can be applied now.

becomes linearly separable upon applying transformation.

ϕ - feature map

It is not dimensionality reduction, instead dimensionality explosion

(2,3 dimensions $\rightarrow$ infinite)

Can we do the algorithm in Nonlinear space? Yes, Kernels.

Let 2 points,

$$p = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \quad q = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \Rightarrow p^T q = p_1 q_1 + p_2 q_2$$

applying ϕ $\quad \phi(p) \equiv$

$$\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \xrightarrow{\phi} \begin{bmatrix} p_1^2 \\ p_2^2 \\ \sqrt{2} p_1 p_2 \end{bmatrix}$$

same on $q \rightarrow \quad \phi(q) \equiv$

$$\begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \xrightarrow{\phi} \begin{bmatrix} q_1^2 \\ q_2^2 \\ \sqrt{2} q_1 q_2 \end{bmatrix}$$

$$\phi(p)^T \cdot \phi(q) = p_1^2 q_1^2 + p_2^2 q_2^2 + \sqrt{2} p_1 q_1 \sqrt{2} p_2 q_2$$

$$= p_1^2 q_1^2 + p_2^2 q_2^2 + 2 p_1 p_2 q_1 q_2$$

$$= (p_1 q_1 + p_2 q_2)^2$$

$$= (p^T q)^2 \equiv K(p, q)$$

(Kernel)

Computing dot product in feature space $\Rightarrow$ Computing dot product in input space with square

gives dot product in 3D

$\sim$ depends on ϕ

Samples in SVM come as dot products. (dual algorithm $\equiv \sum y_i y_j x_i^T x_j$)

for this $\phi =$ } suppose if we put square over there (in SVMs dual obj func^n) then it means we have solved it in the & new complex feature space

→ Homogeneous kernels

$$\phi(p)^T \phi(q) = (p^T q)^2 = K(p,q)$$

"Kernel function"

$(p^T q)^2 \rightarrow \phi \equiv \begin{bmatrix} p_1^2 \\ p_2^2 \\ \sqrt{2} p_1 p_2 \end{bmatrix}$

$$K(p,q) = \phi(p)^T \phi(q)$$

If we know the Kernel function, then we dont care about ϕ (feature map)

→ Polynomial kernels

ϕ corresponding to $K(p,q) \equiv (p^T q + 1)^2$ ~ equivalent to mapping to a 6

$$\equiv p_1^2 q_1^2 + p_2^2 q_2^2 + 2 p_1 q_1 + 2 p_2 q_2 + 2 p_1 p_2 q_1 q_2 + 1$$

$$\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \xrightarrow{\phi} \begin{bmatrix} p_1^2 \\ p_2^2 \\ \sqrt{2} p_1 \\ \sqrt{2} p_2 \\ \sqrt{2} p_1 p_2 \\ 1 \end{bmatrix}$$ ~ 6 dimensional feature space

Kernels can be anything ~

Ex: $K(x,y) = e^{-\gamma x^T y}$

$K(x,y) = \tanh(x^T y)$

Equivalent to infinite dimensions.

Independent of dimensions, dot product is a "scalar"

Kernel :-

* Power of Non linearity
* Computation efficiency of linear

$$\sum \alpha_i - \frac{1}{2} \sum\sum \alpha_i \alpha_j y_i y_j x_i^T x_j$$

↓ can be replaced according to kernel

2 feature maps ⟶ corresponding to same kernel.

For ex ≡ $\phi_1 \rightarrow \begin{bmatrix} p_1^2 \\ p_2^2 \\ p_1 p_2 \\ p_2 p_1 \end{bmatrix}$ $\phi_2 \rightarrow \begin{bmatrix} p_1^2 \\ p_2^2 \\ \sqrt{2} p_1 p_2 \end{bmatrix}$ $\phi_1, \phi_2 \rightarrow K(p,q) \equiv (p^T q)^2$

ϕ = feature map

$K(\ ,\ ) =$

$(x_i^T x_j)^d$, $(x_i, x_{j+1})^d$,

$e^{-\gamma x_i^T x_j}$ (exponential kernel)

Kernel Matrix

$K = \left[K(x_i, x_j) \right]$

it is symmetric, square matrix of dimensions $N \times N$ ($N \equiv$ samples)

⇒ if K is PSD, ∃ ϕ

+ve semi definite.

⇒ (random ϕ doesn't give a PSD. K)

Kernel computation ≈ small computation

* Not all the Kernel functions are valid ⇒

rbf Kernel — most popular Kernel

parameter γ ≡ (right one has to be selected)

Kernel can be defined on programs also, not only on the feature representation of an entity

$K(P_1, P_2)$ { ... }

entity of interest = program, tree, graph, string, which have to be classified.

Not needed.

equivalent to "similarity function"

→ can come up with own Kernel functions.

x → entity that can be compared.

$\phi \equiv K$

↑ only need Kernel computation is needed, not ϕ computation.

$\Rightarrow K_3(\,) = K_1(\,) + K_2(\,)$ (PSD, PSD, PSD)

if K_1, K_2 are valid, then K_3 is valid.

$\Rightarrow$ Linear combination is also a valid Kernel of valid Kernels

$$K = \sum_{i=1}^{p} \alpha_i K_i(\,)$$

(K: valid; K_i: valid.)

$$\Rightarrow \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \underline{K(x_i, x_j)}$$

$\underline{K(x_i, x_j)} \rightarrow \phi^T(x_i)\phi(x_j)$

$$= \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K_{ij}$$

K_{ij} ↓ Kernel matrix = pre compute Kernelmatrix

While testing → $\text{sign}(w^T x + b)$ ✓ (linear)

$$\text{sign}\left(\sum_{i=1}^{N} \alpha_i y_i \underline{K(x_i, x)} + b \right) \equiv \text{(Non linear SVM)}$$

that is why never pre compute w, computing Kernels for support vectors.

⇒ [you need to carry $\{(\alpha_i, x_i)\}$ after training]

↓ adds storage complexity.

unlike perceptron (where only w is used for testing)

* SVMs became non linear with the help of Kernels

$w^T x + b$ → for linear, is fine but
for non-linear, you have to store $\{(x_i, x_i)\}$

⇒ training data → $\phi(x)$ } if we have feature map, then no need of support vectors.

test data
$x \rightarrow \phi(x)^T w$.

⇒ Kernel Matrix (N,N) ≡ Killer of this problem (SVM)
↓
struggle with large date.
(N >>>)
memory space complexity.

⇒ PCA: Limitations

→ It is linear (KPCA) ≡ Kernel PCA.
↑ using Kernels.
→ LDA.

Linear PCA ≡
⇒ given, $x_1 \ldots x_m$ examples, $x_i \in \mathbb{R}^N$ (N dimensional)
centered data.

Compute the covariance matrix, $C = \frac{1}{M}\sum_{i=1}^{M} x_i x_i^T$ → NXN, C has $\min(N,M) \in V$

$$Cv = \lambda v$$

Compute eigen vectors and project a new sample x on $x^T v_i$
↓
require explicit computation of C

This is linear, we should Kernelise this.

if $\sum_{i=1}^{M} \phi(x_i x_i^T)$ → we get ∞ × ∞ covariance matrix → gives ∞ eigen vectors
put ϕ
(avoid such comput^n) ✗

$$C = \frac{1}{M}\sum_{i=1}^{M} \phi(x_i)\phi(x_i)^T \longrightarrow Cv = \lambda v$$

($\sum$ → all the samples)

$$\lambda v = \frac{1}{M}\sum_{i=1}^{M} \phi(x_i)\boxed{\phi(x_i)^T v}$$

scalar

$$v = \frac{1}{\lambda M}\sum_{i=1}^{M} \phi(x_i)\phi(x_i)^T v$$

$$= \frac{1}{\lambda} \sum_{i=1}^{M} \frac{\phi(x_i)}{M} \underbrace{\phi(x_i)^T \cdot V}_{\alpha_i}$$ (scalar)

KPCA :-

$$V = \sum_{i=1}^{M} \alpha_i \phi(x_i)$$

PCA in the feature space?

calculating after eigenvector, $\equiv$ project. $V^T X$ → projection can be computed without V

$$= \sum_{i=1}^{M} \alpha_i \phi(x_i)^T \phi(x)$$

(linear PCA) ∨

Kernel.

$$= \sum_{i=1}^{M} \alpha_i K(x_i, x)$$ → can be rbf, ... any kernel

linear combination of Kernels of x_i, x.

KPCA (primary problem) ?

non linear surface - [sample(M) αs → to find → independent of the feature space, we have to find only M

1 set of αs → 1 eigen vector.

$$CV = \lambda V$$

$$\sum_{j=1}^{M} \phi(x_j) \phi(x_j)^T \sum_{i=1}^{M} \alpha_i \phi(x_i) = \lambda \sum_{i=1}^{M} \alpha_i \phi(x_i)$$

↓ get rid of ϕ, because we know only K — we have odd no. of ϕs on both the sides, so multiply.

— on multiplying $\phi(x_k)^T$ on both the sides.

$$\phi(x_k)^T \frac{1}{M} \sum_{j=1}^{M} \phi(x_j) \phi(x_j)^T \sum_{i=1}^{M} \alpha_i \phi(x_i) = \lambda \phi(x_k)^T \sum_{i=1}^{M} \alpha_i \phi(x_i)$$

$\forall x_k$

[Trick: Replace $\phi^T\phi$ with K]

— moving ϕ's across summations will not have an impact

$$\frac{1}{M}\sum_{j=1}^{M}\phi(x_k)^T\phi(x_j)\sum_{i=1}^{M}\alpha_i\,\phi(x_j)^T\phi(x_i) =$$

$$\lambda\sum_{i=1}^{M}\alpha_i\,\phi(x_k)^T\phi(x_i) \quad \forall x_k$$

$$\Rightarrow \frac{1}{M}\sum_{j=1}^{M}K_{kj}\sum_{i=1}^{M}\alpha_i K_{ji} = \lambda M\sum_{i=1}^{M}\alpha_i K_{ki} \quad \forall x_k$$

($\forall x_k$: 1 of M samples)

$x \rightarrow \phi(x)$

$\phi(x_i)^T\phi(x_j) \rightarrow K(x_i, x_j)$ (kernel matrix) K_{ij}

α_k (one of M samples)

$\sum_j K_{kj}$ → row of a kernel matrix → k^{th} sample row

$$\begin{bmatrix} K \end{bmatrix}\begin{bmatrix} \bar{\alpha} \end{bmatrix}$$

$K\bar{\alpha}$ (α vector)

$$K \cdot K\bar{\alpha} = \underline{\lambda' M}\, K\bar{\alpha}$$

$$K\bar{\alpha} = \lambda'\bar{\alpha}$$

K: $M \times N$; $\bar{\alpha}$: $N \times 1$

→ Max, M eigen vectors possible

→ α is eigen vector of the Kernel matrix, $K_{M\times N}$

→ Non linear projection is possible with this new, KPCA if there exists a right Kernel.

Algo:

- Compute Kernel matrix
- Compute Eigen vectors of Kernel matrix
- Each eigen vector has one projection. ($\sum_i^M \alpha_i k(x_i, x)$)

But there's a limitation —

Assumption: data is centered, but in ϕ space?

↓ we assumed mean is always subtracted.

$$\phi(x_i) - \underbrace{\frac{1}{M}\sum_{i}^{M}\phi(x_i)}_{M_{\phi(x)}}$$

$$K_{mn} = \phi(x_m)^T \phi(x_n)$$

m,n^{th} element of new K matrix

$$\left[\phi(x_m) - \frac{\sum_{i=1}^{M}\phi(x_i)}{M}\right]^T \left[\phi(x_n) - \frac{\sum_{i=1}^{M}\phi(x_i)}{M}\right] \approx \phi^T\phi$$

(Centered) $K_{ij} \leftarrow$ original(K_{ij})

Can be derived from

→ Given $x_1, \ldots x_M$, $x_i \in \mathbb{R}^N$

→ Compute Kernel matrix $K'_{M \times M}$

→ Compute Kernel matrix of centered K. [Centered data]

→ Find α by;

$$K\bar{\alpha} = \lambda\bar{\alpha}$$

(after computing α, we cannot throw away x)

→ For each eigen vector α^k,

$$\sum_{i=1}^{M} \alpha_i^k K(x_i, x) \rightarrow \text{projection}$$

Kernel LDA > Kernel PCA > LDA > PCA

→ Kernelised based variations are much better.

Linear PCA: $x^T v_i$ (linear projection) ⟶ KPCA: $\sum_{i}^{M} \alpha_i K(x_i, x)$ (projection in higher dimensional space) Non linear.

LDA ⇒

→ After projection, samples within classes should come

① closer to mean of the respective classes.

↳ within (class) scatter — minimized

↓

should become compact

↓

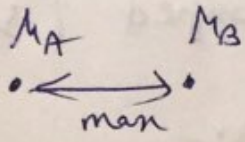
and well separated.

② between

→ classes scatter — maximized ↑

A, B — 2 classes ⇒

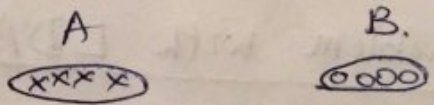
$$(\text{between}) S_B = [\mu_A - \mu_B][\mu_A - \mu_B]^T$$



$$(\text{within}) S_W = S_A + S_B$$

$$= \sum_{i=1}^{N_A} [x_i - \mu_A][x_i - \mu_A]^T + \sum_{i=1}^{N_B} [x_i - \mu_B][x_i - \mu_B]^T$$

$$\Rightarrow \max_u \left[\frac{u^T S_B u}{u^T S_W u} \right] \quad \text{(eigenvector.)} \Rightarrow$$



↓

$$\max_u [u^T S_B u] \quad \text{such that} \quad u^T S_W u = 1$$

↓

converted to →

$$\max_u \left[\frac{1}{2} u^T S_B u - \frac{\lambda}{2}(u^T S_W u - 1) \right]$$

$$\Rightarrow \quad S_B u = \lambda S_W u$$

$$\boxed{S_W^{-1} S_B u = \lambda u}$$

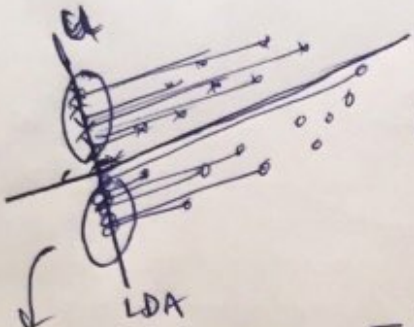
PCA doesn't know class labels ≡

LDA is not always orthogonal to PCA

mixed up. [not separable]

PCA ∼ Compression friendly

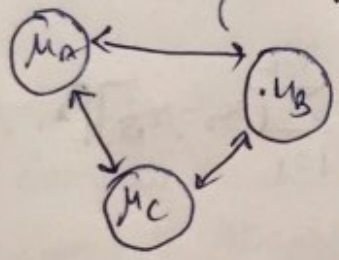
↳ after representation, loss of info is small.



LDA ⟶ after projection, separability is high.

nicely classified. [linearly seperable]

for 3 classes:-



Problem with LDA ≡⟩ $S_W^{-1} S_B u = \lambda u$

small sample problem.

1.) can be resolved: ← $S_W \leftarrow S_W + \rho I$ (ε (small))

(regularise matrices)

upon doing it, rank will increase

changes the singularity.

computationally hectic.

may not be a full rank matrix because, samples < dimensions [M < N]

↳ Not invertible

2) PCA → LDA

dim: 65536
samples: 520

⇒ apply PCA and reduce it below 520, now it wont have any inverse problem

3) $S_W = \sum_{i=1}^{M} \sum_{j=1}^{M} [x_i - x_j][x_i - x_j]^T$

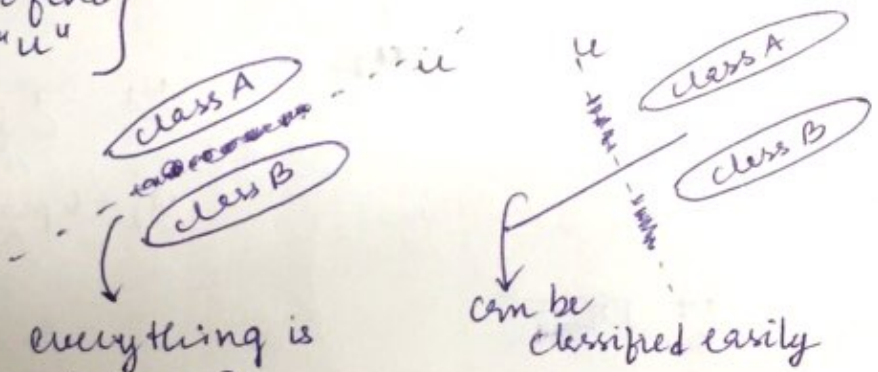
new definition of S_W

Generalised Eigen Vector problem —

projected onto a plane.

⇒ Linear dimensionality reduction techniques:

Unsupervised technique	Supervised technique
PCA	LDA
$y = u^T x_i$	class labels are needed
Goal: to find "u"	



everything is mixed up ⇒ Classes are not well separated.

can be classified easily

LDA ≡

* classes are compact and well separated.
  - compact: within class → S_W → has to be minimized.
  - well separated: between class. → S_B → has to be maximized

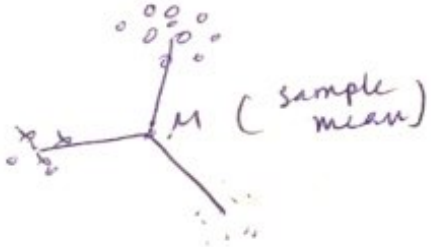
For n-classes :-

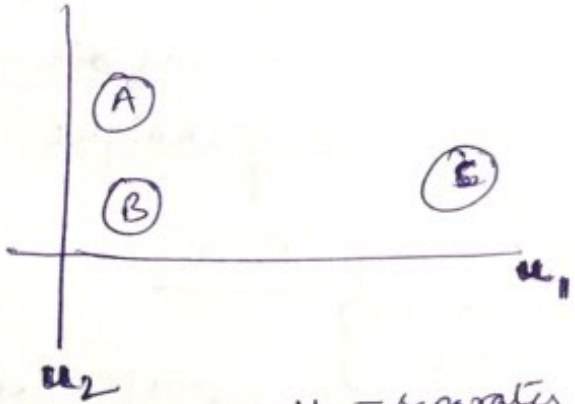
$$S_W = S_A + S_B + S_C \cdots S_C$$

$$= \sum_{i=1}^{C} \sum_{j=1}^{N_i} [x_j - \mu_i][x_j - \mu_i]^T$$

$$S_B = \sum_{i=1}^{C} [\mu_i - \mu][\mu_i - \mu]^T$$

better way $\equiv \sum_{i=1}^{C} \sum_{j=1}^{C} [\mu_i - \mu_j][\mu_i - \mu_j]^T$





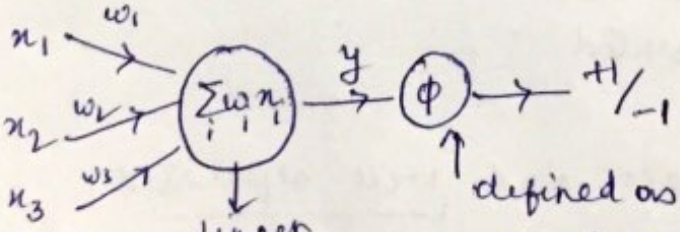
u_1 – separates C from A & B

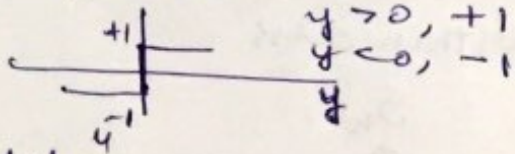
u_2 – separates A & B

* Neural Networks ≡

• Previous algorithms ≡ dot product

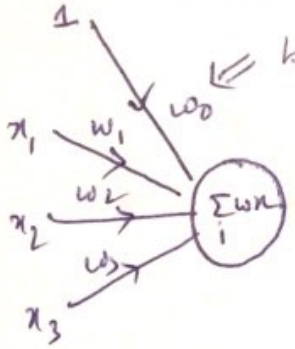
→ $\text{sign}(w^T x)$

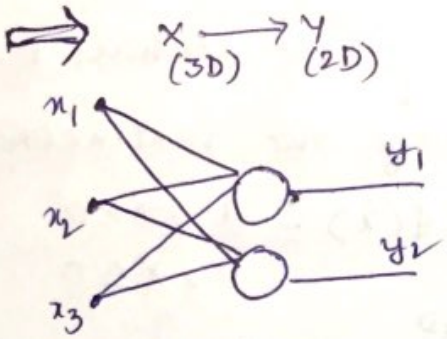




Neuron model

bias added → for generalisation purpose.





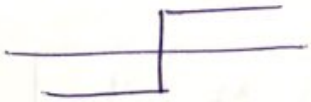
ϕ?

Single layer networks

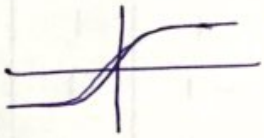
Regression ≡

ϕ ≡

Classification ≡



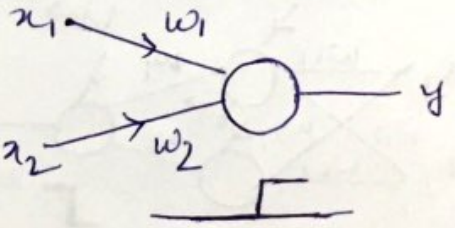
sigmoid, ϕ ≡



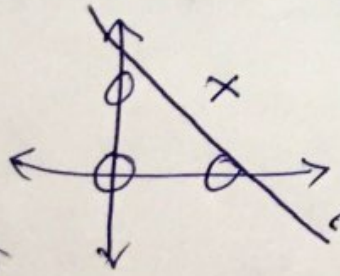
Network design ≡

Logic – AND

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

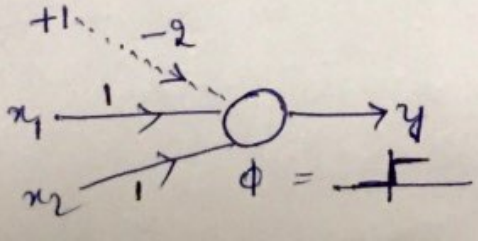


w_1, w_2 = ?



→ for this line, we need bias also change the non linearity → to classify

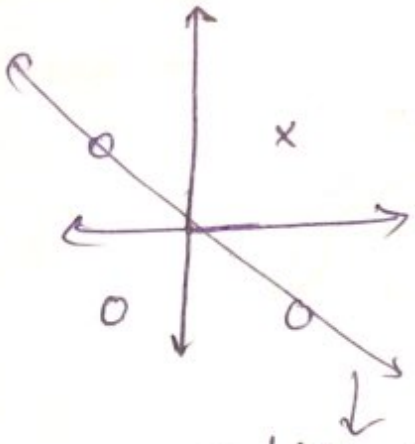
↓ introducing bias into the network.



$\equiv \begin{cases} 1, & x \geq 0 \\ 0, & x < 0. \end{cases}$

if AND in terms of -1, 1 ⇒

x_1	x_2	y
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

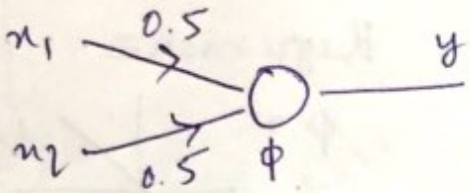


↓

[Line without bias

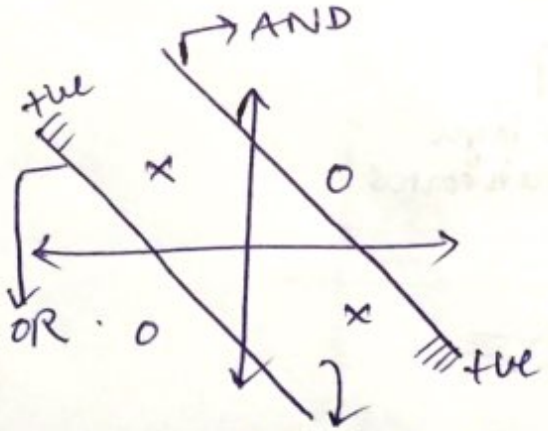
Change the nonlinearity.

→ $\phi(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$

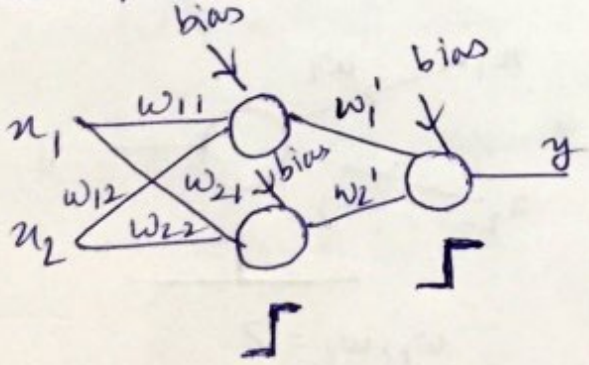


XOR ⇒

x_1	x_2	y
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

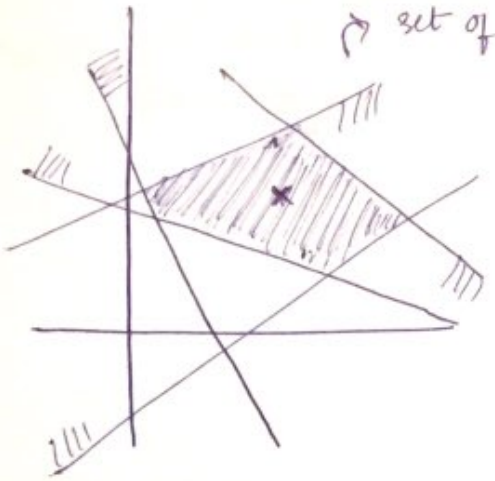


→ Cannot be solved with a linear classifier.



visually represented as

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$



↳ set of 2D lines on a plane

Non linearities ↓
getting converted to +1/0s

↓

Multilayer Perceptron

↓

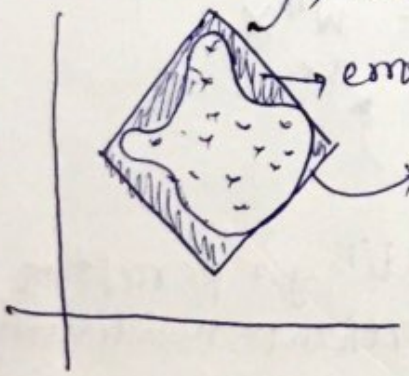
Singlelayer perceptron – only applicable for linearly separable data.

Single layer → Multilayer perceptron → will be able to classify anything existing in the space.

[half planes] [can classify any convex regions]

⇒ with more layers → we can get arbitrary shapes.
(multiple convex polygons).

Non convex shape



→ multilayer perceptron approximation

→ errors.

→ with more neurons ≡ we can increase the polygon sides (less error and more efficient)

→ increasing neurons in the ~~middle~~ hidden layer → ↑ sides of polygon (but still a convex shape)

↓

but width increase in depth ↓ arbitrary shapes

↓

can approximate any set (positive, negative).

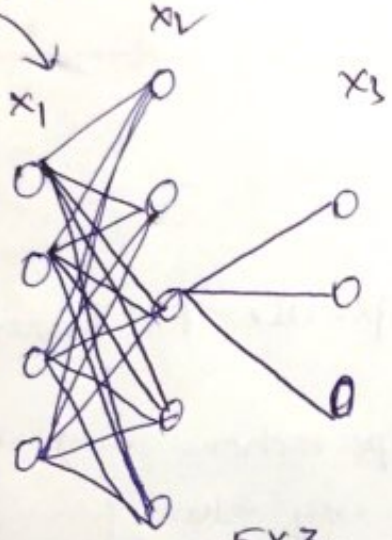
↓

you will not get a non convex shape.

you are given only the examples [what is inside/outside] → decide the boundary/shape (approximate)

multi layer Perceptron:

fully connected networkes



5×3.

4×5

↓

weights have to be learned

$x_2 = W x_1$

5×4

$x_3 = W' x_2$

3×5

$x_3 = W' x_2$

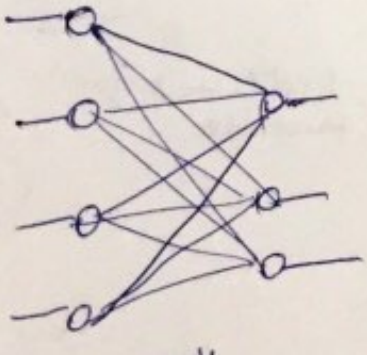
$= W' W x_1$

$= W'' x_1$

3×4

⇒ any no of layers ≈ can be equated to a single layer of network

→ multilayer perceptron (without non linearity) ≡ single layer perceptron



W''

= This is working because of the absence of non-linearity

but when we introduce non linearity;

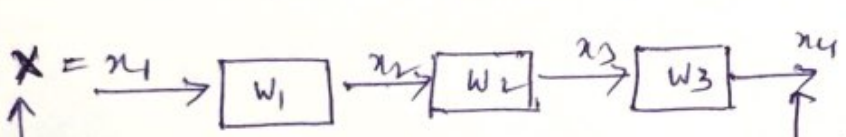
$$x_2 = \phi(w x_1)$$
$$x_3 = \phi(w' x_2)$$

$$x_3 = \underbrace{w' w}\, x_1 = w'' x_1$$ ✗ Not possible

$\phi \equiv$ 

Back Propagation ≡ "For learning weights"

Neural network layers → as computational blocks.



input ≡ vector

$x_2 = \phi(w_1 x_1)$

$x_3 = \phi(w_2 x_2)$

$x_4 = \phi(w_3 x_3)$

output ≡ vector

All the non-linearities ≡ need across layers } not be the same

$w_1 \leftarrow w_1 - \eta \frac{\partial E}{\partial w_1}$ → matrix; → diff of loss wrt w_1

$w_2 \leftarrow w_2 - \eta \frac{\partial E}{\partial w_2}$

$w_3 \leftarrow w_3 - \eta \frac{\partial E}{\partial w_3}$

if sample is in class 2

$Y_{\text{output layer vector}} \equiv \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

we have to compute, $\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots$ ≡ to update and improve the solution using gradient descent.

↓

we use "chain rule"

we can compute $\frac{\partial E}{\partial w_3}, \frac{\partial E}{\partial x_3}$ directly, but

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial x_3} \cdot \frac{\partial x_3}{\partial w_2}$$ [chain rule].

partial derivatives of output with respect to parameters (input) and should be computable.

$$\boxed{\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial x_3} \cdot \frac{\partial x_3}{\partial x_2} \cdot \frac{\partial x_2}{\partial x_1}}$$

"Chain rule"

then we can apply chain rule to compute E wrt any of the parameters.