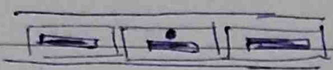
Curse of Dimensionality

It arises when working with "high dimensional data". Leading to increased computational complexity.

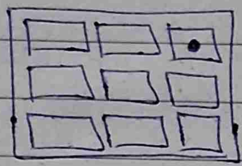
for e.g. → (You lost your wallet)

the searching parameters :-

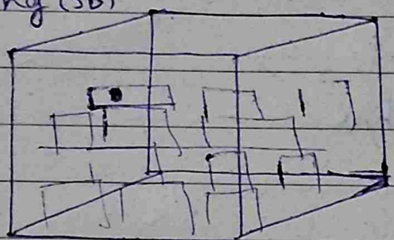
(i) road (linear)



(ii) ground (plane)



(iii) building (3D)



as we increase the dimensionality, more complex it is to solve (find the wallet) problem

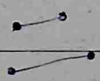
⦿ imagine now 100D space (like ML features) → the no. of possible places you wallet could be is very large.

⦿ ~~Imagine now if 10000D dataset, then not all~~

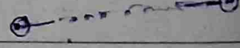
e.g. Avg. distance btw. two point

in 2D = 0.52

3D = 0.66



now in 1,000,000D hypercube = 408.25

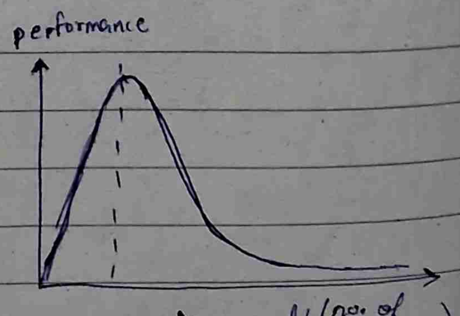


this, "how can two points be so far if they lie in same hyper plane" → implies that data are {"sparse"}

↳ most training instances are far away from eachother

{ leftout space is 0 }

- Performance dec. ↓
- More computation



Dimensionality Reduction

In principle, curse of dimensionality can be mitigated by inc↑ sample size to achieve adequate space converge (less sparsity)

But, no. of training data required per feature also grows.

e.g. consume preference dataset

(age → 100 people

age, income → 100x100 = 10,000 people)

Main Approaches:-

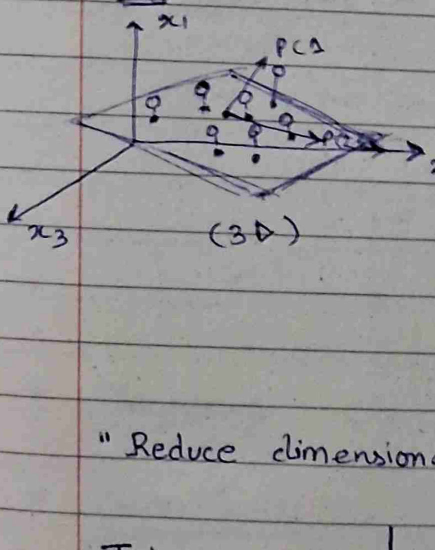
(1) "Projection"

IRL data → training data are not spredout uniformatly across all dimensions

- Many features are constant
- Others are highly correlated

∴ training data lie within (close to) much lower dimensional subspace of hyper-subspace

e.g.

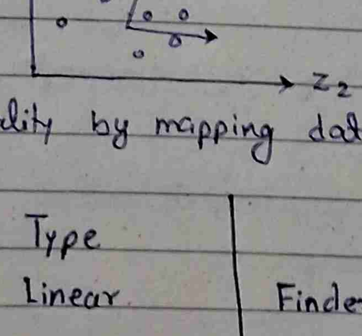


data points are clinging near plane

↓

now (2D) projection

↓



"Reduce dimensionality by mapping data from high-dimensional

Techniques	Type	
→ Principal Component Analysic (PCA)	Linear	Finder hyperplane that maximizes varience of projected data
→ Linear Discriminant Analysis (LDA)	Linear	Find hyperplane that maximizes seperation btw. known class (supervised)

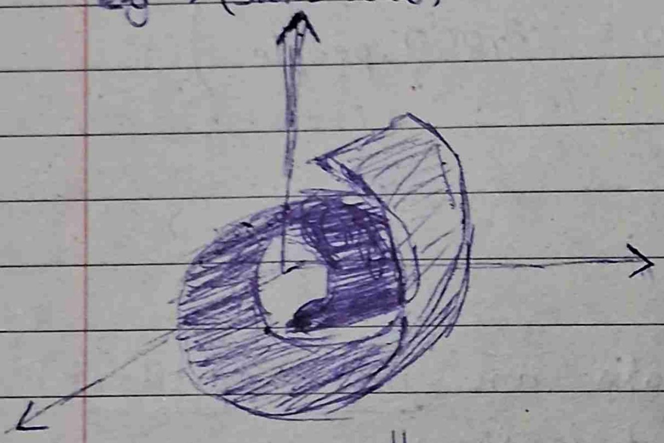
(2) Manifold Learning

(non-linear dimensionality reduction)

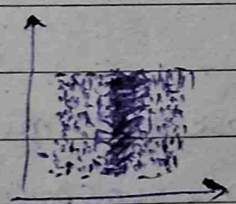
Its based on idea that high-dimensional data often lies on a low dimensional 'manifold' (a low dimensional "surface that is embeded in high-dimensional space).

The goal is to "unfold" this manifold

eg → (swiss roll)

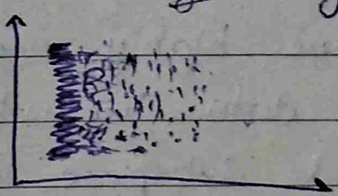


- is non-linear data
- when reduced from 2D → 3D, then data points at diff. elevation gets embedded at same point, not ~~ga~~ being useful.



squashing by projection

or



unrolling the swiss roll

PCA : Principal Component Analysis

* Used for dimensional reduction of linear data
* Preserving the variance - maximizes the variance

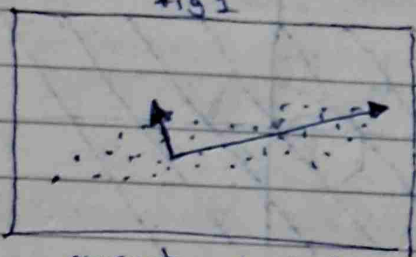
Maths :-

PCA finds a new set of dimensions such that all dimensions are orthogonal(& hence linearly independent) and ranked according to variance of data.

Find transformations such that

- Transformed features are linearly independent
- Dimensionality can be reduced by taking only the dimensions with highest importance
- Those newly found dimensions should min↓ projection error
- Projected points should have max spread (variance).

fig 1



correct principal axis

* max spread along it

↓

(projection)

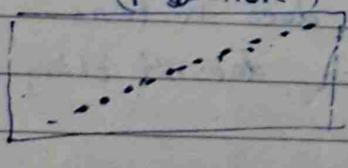
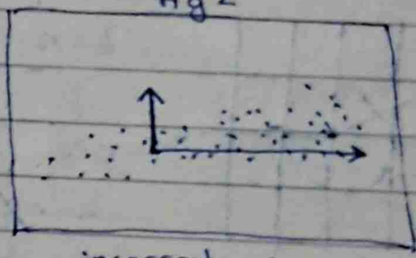


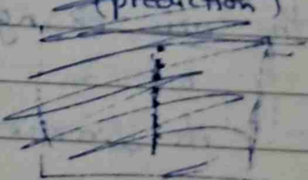
fig 2



incorrect principal axis

↓

~~(prediction)~~



* **Variance** : $Var(x) = \frac{1}{n} \Sigma (x_i - \bar{x})^2$

* **Co-Variance Matrix** : $CoVar(X,Y) = \frac{1}{n} \Sigma (X_i - \bar{X})(Y_i - \bar{Y})^T$

* $$\begin{bmatrix} Cov(X,X) & Cov(X,Y) \\ Cov(Y,X) & Cov(Y,Y) \end{bmatrix}$$

* **Eigen Vectors, Eigen Values**
  - The arrows (PC1, PC2) are the eigen vectors of covariance Matrix of data
  - The direction represent the principal component i.e. new axes after PCA transformation

The eigenvectors point in direction of max. variance & the corresponding eigen values indicates the importance of its corresponding eigen vector

eigen value

Matrix $A\vec{v} = \lambda \vec{v}$

$A\vec{v} = \lambda I \vec{v}$

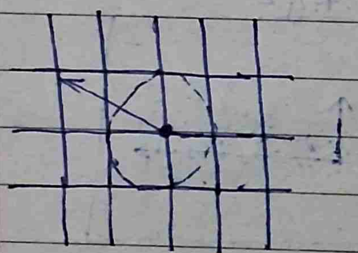
$(A - \lambda I)\vec{v} = 0$

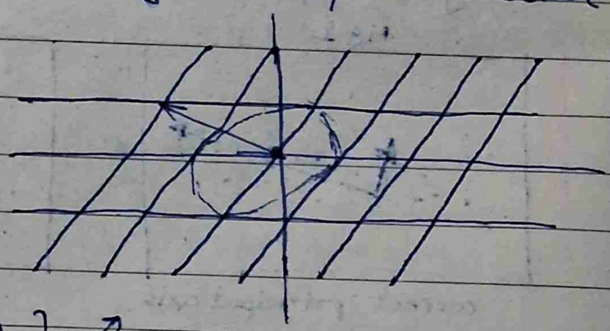
$|A - \lambda I| = 0$

find out values of (λ)

<u>Note:-</u> (Eigen Vectors)

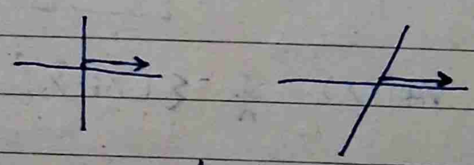
Even after linear transformation, its direction doesn't changes, making it a special vector





$$\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}$$

the axis changes and so does vectors but an eigen vector doesn't changes its direction



direction didn't change, just magnitude changes

another <u>e.g.</u>

$$\begin{bmatrix} 0.5 & -1 \\ -1 & 0.5 \end{bmatrix}$$

(Eigen Values):- How much eigen vectors are streching & shrinking after linear transformation (whats the difference in the scale)

PCA and Eigen Vectors :-

(EigenDecomposition of Covariance Matrix)

In PCA we use co-varience matrix. ∴ give co-variance matrix in the PCA use eigen vectors to find such a vector that maximizes varience (spread)

In other words, the largest eigen vector of co-variance matrix points into the direction of largest variance of data

Coding :-

sklearn-decomposition → PCA

PCA ~~attributes~~ Parameters :-

- n-components → no. of PCA lines (dimensions) you want
  - → Integer (K)
  - → Float ($0 < f \leq 1$)
  - → None

Attributes :-

- [pca. explained-varience_] → gives array of eigen values
- [pca. explained-varience-ratio_] → used to create scree plot
- [pca-components_] → Is a matrix. Each row is one principal component (eigen vector)