Dimensionality Reduction

why? → to visualize high dimension data.

how? → PCA & t-SNE.

* By default, a vector is column vector.

$x_i \in R^d$ — column vector $d \times 1$

* Representing dataset

$D = \{ x_i, y_i \}_{i=1}^{n}$ — data pts.

$x_i \in R^d$; $y_i \in$ {setosa, virginica, versicolor}.

D is usually represented in 2 matrices — X & Y.

X — rows are data pts & cols are features.

Y — rows are data pts. usually 1 col only.

Data pre-processing : Column normalization

why? — So that the data becomes nicely structured so that data modelling algos will perform well.

normalizat^n :- for every feature, we get

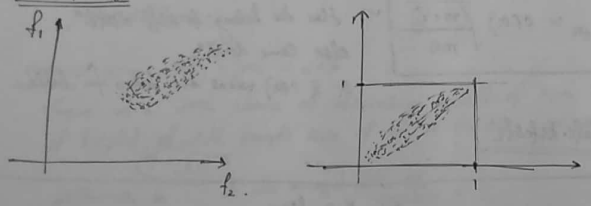
$$a_i' = \frac{a_i - a_{min}}{a_{max} - a_{min}}$$

$a_i' \in [0, 1]$.

we become independent of units → getting rid of scales.

All in the same scale of [0,1] — helps many algos become faster.

Geometrical view.



* without spoiling rel^n b/w the pts we are confining them to a small space.
* In high dimension space, we squish data to hyper-cube.

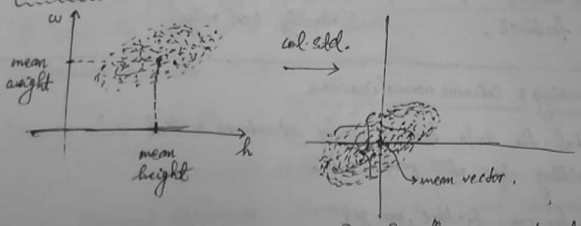
Column Standardizat^n — more often used than col: normx^n.

for every feature, $a_i' = \frac{a_i - \mu}{\sigma}$ μ = mean of feature, σ = std dev^n of feature

So, now for each feature after standardizat^n, mean = 0 & std-dev = 1.

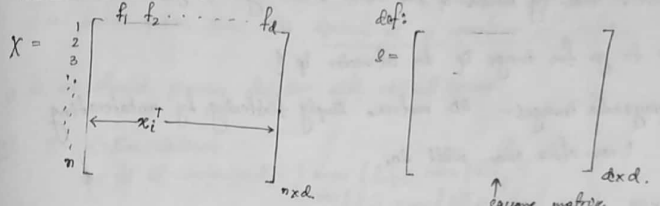
a_i can come fm any distrib^n, not necessarily Gaussian.

Geometric view



① moving the mean vector to origin
② squishing/expanding st. std dev for any feature = 1

Co-variance-matrix



$S_{ij} = cov(f_i, f_j)$.
$i: 1 \to d$
$j: 1 \to d$.

$$cov(X,Y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$

So,
$cov(f_i, f_i) = var(f_i)$.
$cov(f_i, f_j) = cov(f_j, f_i)$

Hence, the diagonal elements will be variance of features & the matrix will be symmetric.

* Let X matrix is col: standardized.

$$cov(f_1, f_2) = \frac{1}{n}\sum_{i=1}^{n}(x_{i1} - \mu_1)(x_{i2} - \mu_2)$$

$\mu_1 = 0$.

$$= \frac{1}{n}\sum_{i=1}^{n} x_{i1} * x_{i2}.$$ → multiplying the feature1 & feature2 values of corresponding data pts. & taking summat^n.

$$= (f_1^T \cdot f_2) * \frac{1}{n}.$$

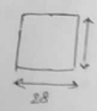
$$S_{d\times d} = \left(X^T_{d\times n}\ X_{n\times d}\right) * \frac{1}{n}$$ — can be proved using simple derivat^n. given X is col: standardized.

MNIST dataset (refer colab: github/colab)

60k - training pts
10k - test pts.

Data of handwritten digit grayscale photos. of dim 28 x 28 pixels.

$D = \{x_i, y_i\}_{i=1}^{60k}$

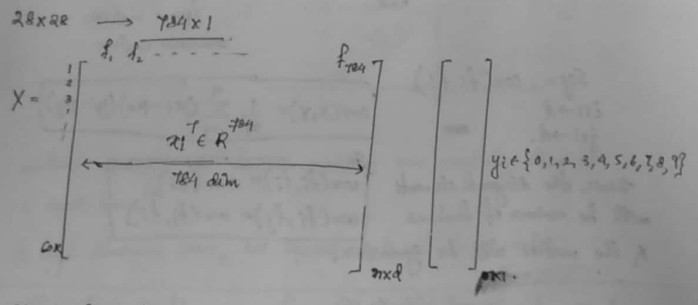
x_i: 

$y_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

objective: - classify which of the class the image belongs to.

* how to go fm image ip to col vector ip?

↳ grayscale images - 2D matrix. simply flattening by concatenating 1 row after other will do.



can't visualize this in 2D.

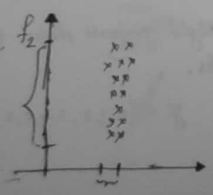
Principal Component Analysis (PCA).

↳ dimensionality red^n → d-dim to d'-dim. $d' < d$.

↳ why → visualize data.

Geometric Interpretation

we'll get intuitions fm examples reducing
2 dim → 1 dim & generalize to
d-dim → d' dim $d' < d$.

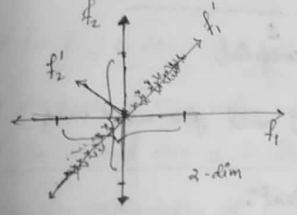


f_1: blackness of hair
f_2: height
spread of f_2 high
spread on f_1 very minimal.

If we have to convert this data to 1-D, skipping the blackness feature is better option since spread is a measure of data.

* So, we should preserve the dim with maximal spread.

ex: X = 2-dim dataset.
say its col standardized - $\begin{cases} \text{mean}\{f_1\} = \text{mean}\{f_2\} = 0 \\ \text{var}\{f_1\} = \text{var}\{f_2\} = 1. \end{cases}$



1 dim. → spread on both f_1 & f_2 is significant.

but if we consider f_1' & f_2'
spread(f_2') << spread(f_1').
{f_1' & f_2' are ⊥r to each other}.

So, basically we are rotating the given axes pair f_1-f_2 by an angle θ such that the variance of of x_i's projected ⊥o f_1' dir^n is maximum & then drop f_2'

Variance maximizing optimization

Let u_1 be unit vector in dir^n of f_1'

So, $\|u_1\| = 1$.

$x_i' = \text{proj}_{u_1} x_i = \dfrac{u_1 \cdot x_i}{\|u_1\|^2} = u_1^T \cdot x_i$.

So, $\underline{x_i' = u_1^T \cdot x_i}$

$\overline{x_i}' = u_1^T \bar{x}$ ← mean of orginal pts.

mean of projected pts

we have to find u_1 s.t. $\text{var}\{\text{proj}_{u_1} x_i\}_{i=1}^n$ is maximum.

$$\text{var}\{u_1^T x_i\}_{i=1}^n = \frac{1}{n}\sum_{i=1}^n (u_1^T x_i - u_1^T \bar{x})^2$$

$\bar{x}$ → 0 since col std. $[0, 0, 0, \ldots 0]$.

$$\text{var}\{u_1^T x_i\}_{i=1}^n = \frac{1}{n}\sum_{i=1}^n (u_1^T x_i)^2$$

$$\max_{u_1} \frac{1}{n}\sum_{i=1}^n (u_1^T x_i)^2 \quad \text{s.t.} \quad u_1^T \cdot u_1 = 1 = \|u_1\|$$

↓ objective of optimizat^n problem

↓ constraint.

If no constraint, then $u_1 = [\infty, \infty]$ would give max^m value.

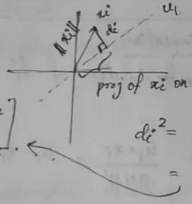
Alternative formulation:— distance minimizat^n.

Earlier formulat^n:- find u_1 which maximizes projected variance.

Reformat:

for each x_i, we have d_i: distance of x_i fm u_1

$$\min_{u_1} \sum_{i=1}^n d_i^2$$



proj of x_i on $u_1 = u_1^T x_i$.

$$d_i^2 = \|x_i\|^2 - (u_1^T x_i)^2$$

$$= (x_i^T x_i) - (u_1^T x_i)^2$$

$$\min_{u_1} \sum_{i=1}^n \underbrace{\left[(x_i^T x_i) - (u_1^T x_i)^2\right]}_{d_i^2}$$

s.t. $\boxed{u_1^T u_1 = 1}$

both the optimizat^n problems are the same.

Solution to our optimization problems:

$S_{d\times d}$ = co-variance matrix. will give d-pairs of (eigen value, eigen vector)

$\lambda_1, \lambda_2, \lambda_3 \ldots \lambda_d$ $(\lambda_1 \geq \lambda_2 \geq \ldots \lambda_d)$.

$\downarrow \quad \downarrow \quad \downarrow \qquad \downarrow$

$v_1 \quad v_2 \quad v_3 \qquad v_d$.

defn of E.val E.vec:— $\lambda_i v_i = S v_i$ — ale pair of λ_i, v_i will satisfy this relationship.

* for $\forall i, j$ s.t. $i \neq j$, $v_i \perp v_j \Rightarrow v_i^T v_j = v_i \cdot v_j = 0$
* Every pair of eigen vectors are perpendicular.
* Our required dirn e_1 is basically v_1. (vector corresponding to largest eigen val).

Geometric interpretation of $\lambda$is & $v$is

Say $x_i \in R^{10}$ thus dimensions = 10.

v_i's

$\lambda_1 \geq \lambda_2 \geq \lambda_3 \ldots \lambda_{10}$

$\downarrow \quad \downarrow \quad \downarrow \qquad \downarrow$

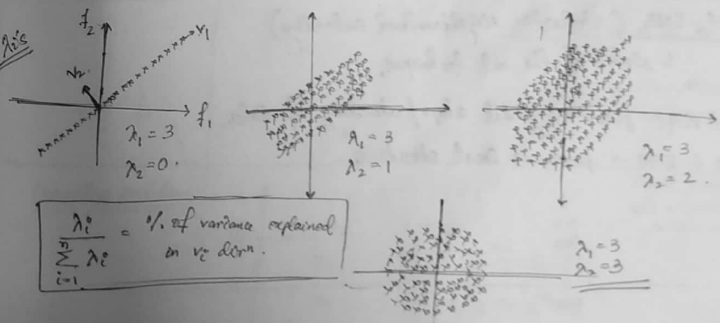
$v_1 \quad v_2 \quad v_3 \qquad v_{10}$.

↑ direct^n with max^m variance

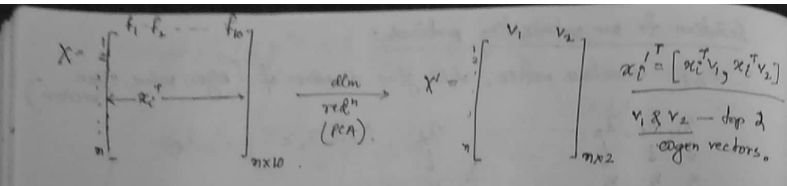
↑ direct^n with 2nd most variance

↑ direct^n with least variance.

$\lambda$is



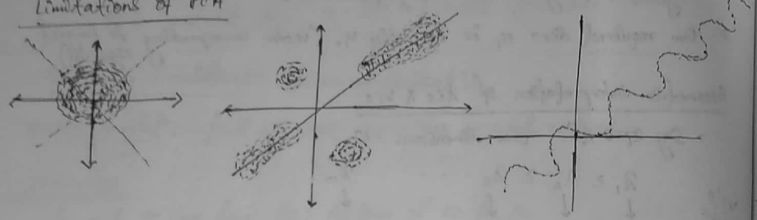
$\frac{\lambda_i}{\sum_{i=1}^n \lambda_i}$ = % of variance explained in v_i dirn.



* Preserve 99% of variance.

Then let $\frac{\lambda_1 + \lambda_2 + \cdots + \lambda_k}{\sum_{i=1}^{d} \lambda_i} = 0.99$ so, k-dimensions used.

Limitations of PCA



Suppose we have to reduce fm 784 dim to 200 dim

$X_{15K \times 784} \times V_{784 \times 200} = X'_{15K \times 200}$

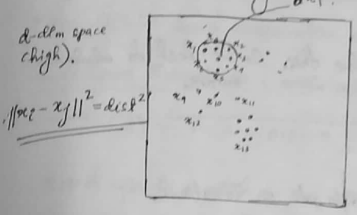
V_1 V_2 ... V_{200}

t-SNE (Stochastic Neighbourhood Embedding)

└ state of the art technique.

* PCA :- preserves global shape / structure of data.
* t-SNE → preserves local structure.

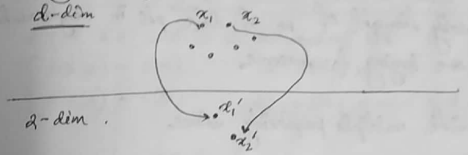
Neighbourhood Embedding



$N(x_i) = \{x_j \mid x_i \& x_j$ are geometrically close$\}$

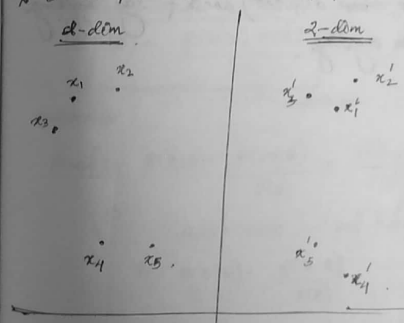
$N(x_1) = \{x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$ doesn't contain $x_9, x_{10}, x_{11} \ldots$

Embedding (mapping)



for every point in the high dimensional space if we are finding corresponding pt in low dimensional space, such a thing as called embedding.

* t-SNE preserves distance of pts in neighbourhood.



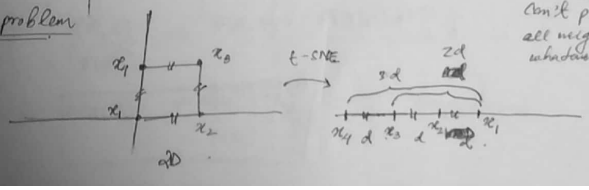
$d(x_1, x_2) \approx d(x_1', x_2')$.

$d(x_1, x_3) \approx d(x_1', x_3')$.

$d(x_4, x_5) \approx d(x_4', x_5')$

$d(x_1, x_4) \not\approx d(x_1', x_4')$.

Crowding problem



Can't preserve all neighbours whatever we do

So, sometimes it becomes impossible to preserve distance of all the neighbours.

So, use t-distribution to resolve this. Do its best to minimize errors.

Analysing t-SNE (distill.pub).

* Iterative algo.- goes through all data pts on iterations.
* Probabilistic algo- can give diff o/p on multiple runs.
* perplexity can be loosely thought of no. of data pts in neighbourhood of each pt we are trying to preserve.
* Always run t-SNE with multiple perplexity values.
* always perplexity < #data pts.
* Always check whether shape has stabilized by running it for more no of times.
* cluster size (more tightly packed, more diffused) doesn't say anything.
* distance b/w clusters don't mean anything.