Principal Component Analysis

[HDV-17]

- → Reachability Distance
- → Comparision of outlier detectors
- → PCA

Reachability Distance :

$$RD(A, B) = \max(KD_B, dist(A, B))$$

$$Density(A) = \frac{1}{\frac{1}{K}\sum RD(A, Neighbours)}$$

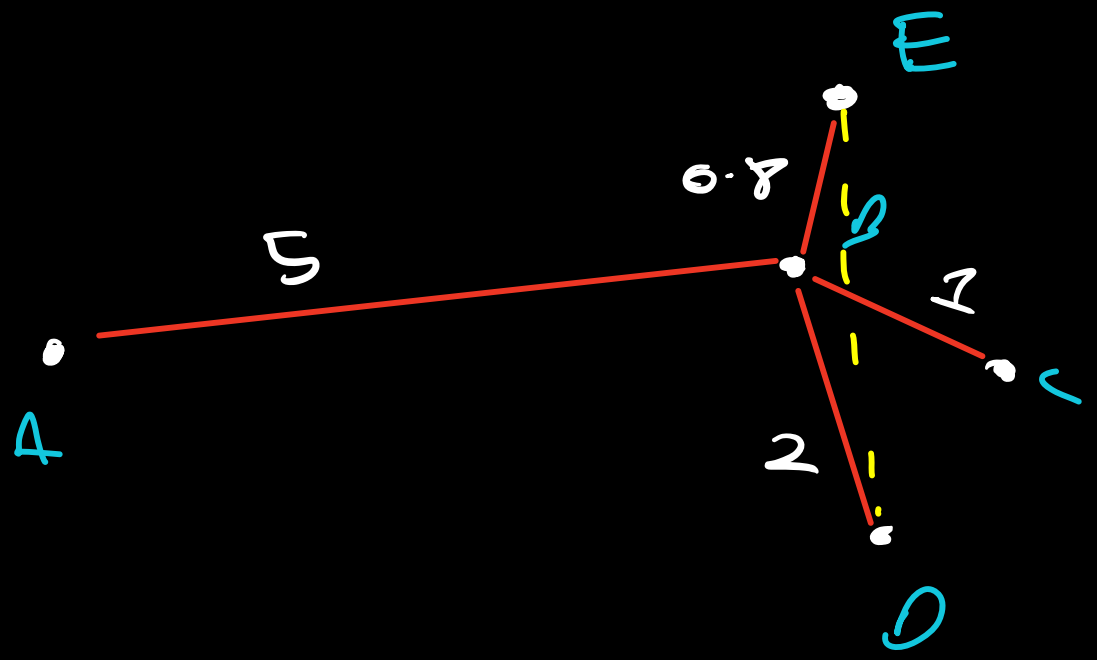
This means that we will use distance between A, B but it has to be min equal to K-dist(B)

in other words!

RD(me, you):

if you are in my top 'K' but I am not in your top 'K' then let's use actual distance but if we both are in each other's top 'K' then let's use your K-dist

Note: RD is not Symmetric.



$RD(A, B) = 5$

$RD(C, B) = 2$

$RD(E, B) = 2$

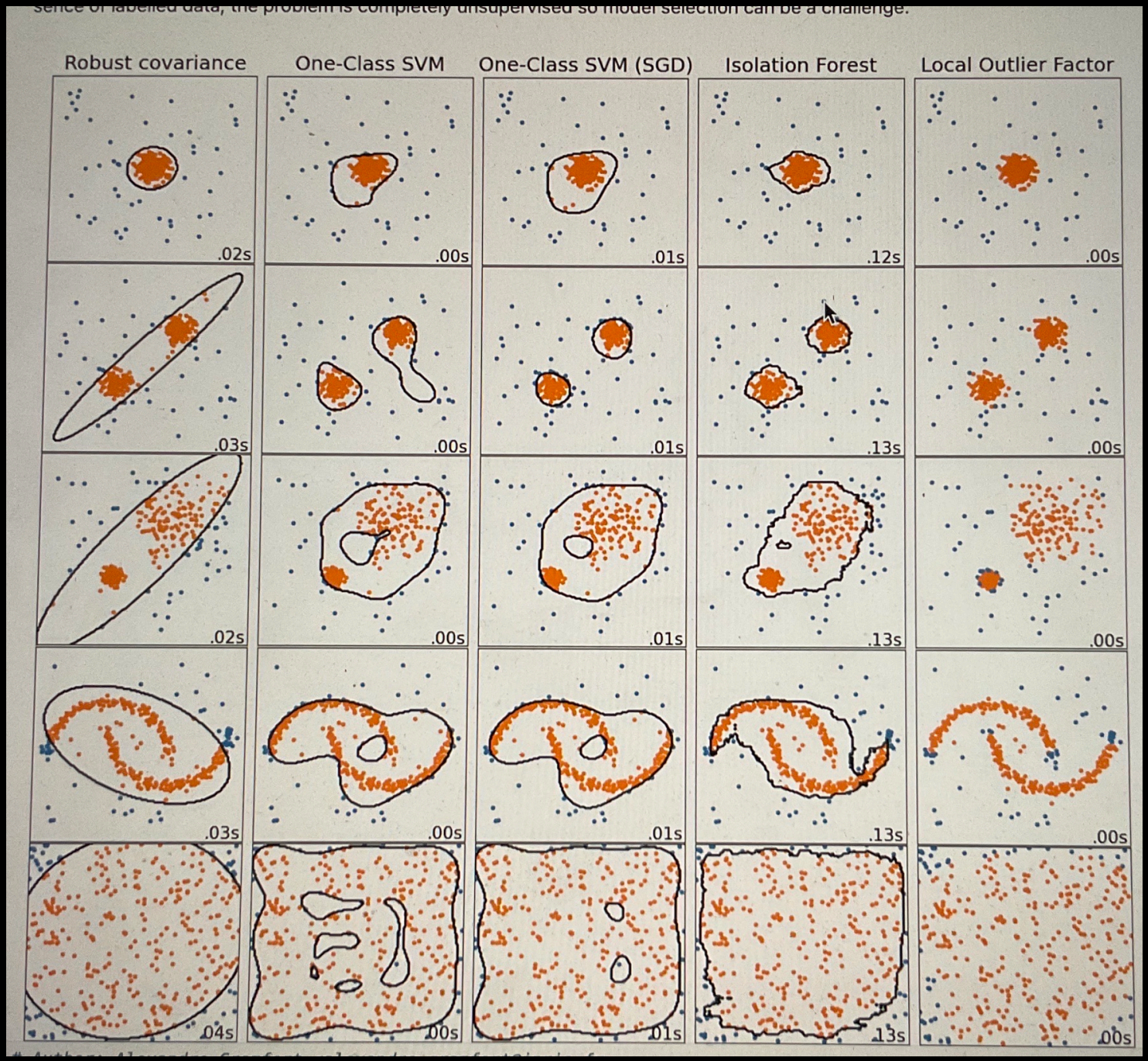
$RD(D, B) = 2$

$\not=$ not 1

$RD(B, E) = \underline{2.5}$

This is less intuitive but has some benefits when 1st neighbour and k^{th} neighbour distance may be very different.

Comparision



Principal Component Analysis

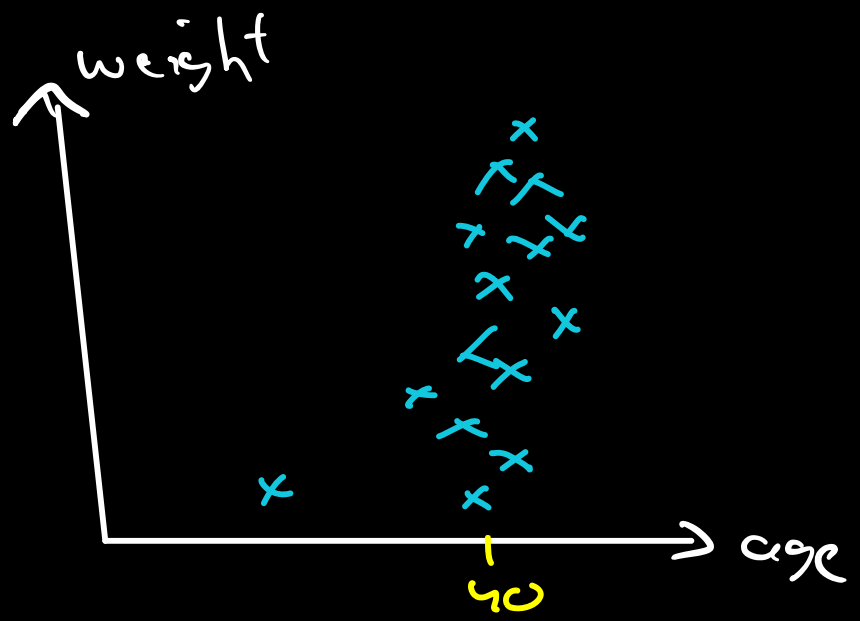
Let's take a few examples:

Diabetes prediction:

W	A	D
:	:	Y
:	:	N
:	:	Y
!	!	!

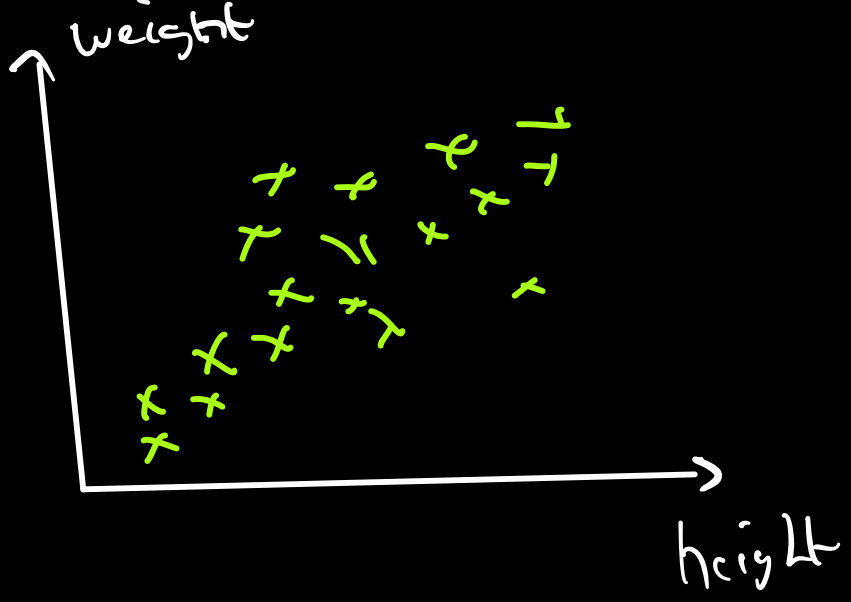
f_1 = weight

f_2 = age



Q: If you could only use 1 feature which would you use?

a) Age b) Weight c) A + W

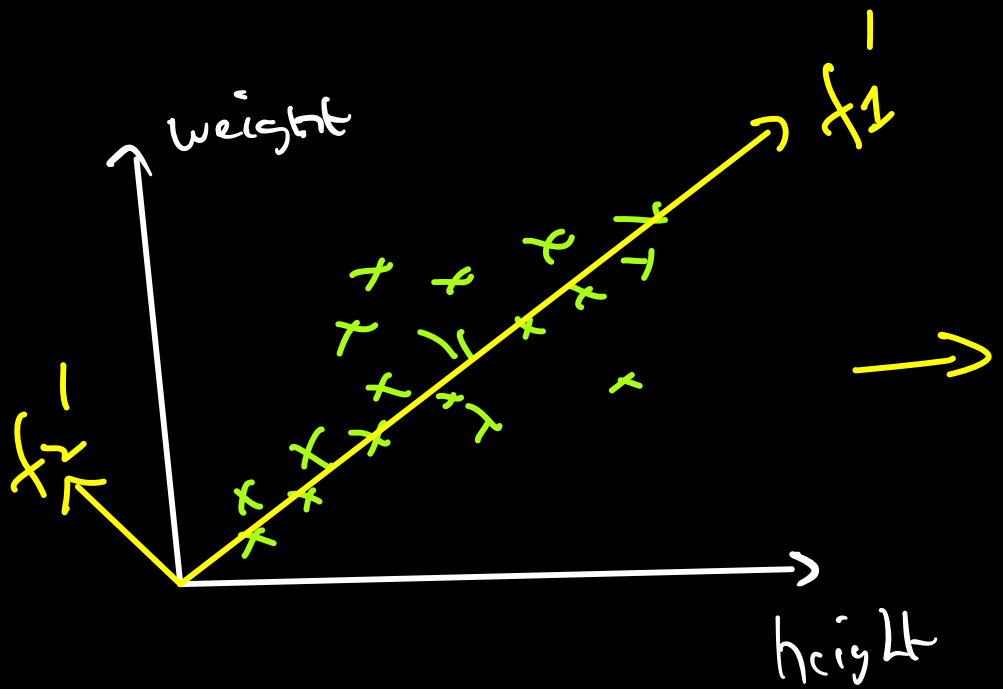


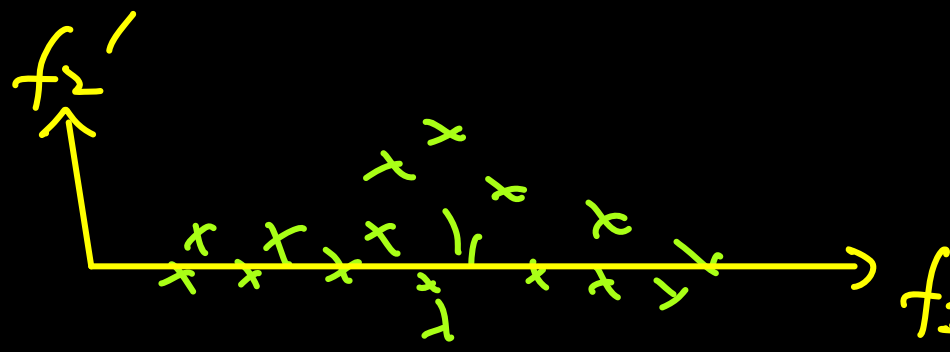
Q: which feature?

a) weight

b) height

c) w + h





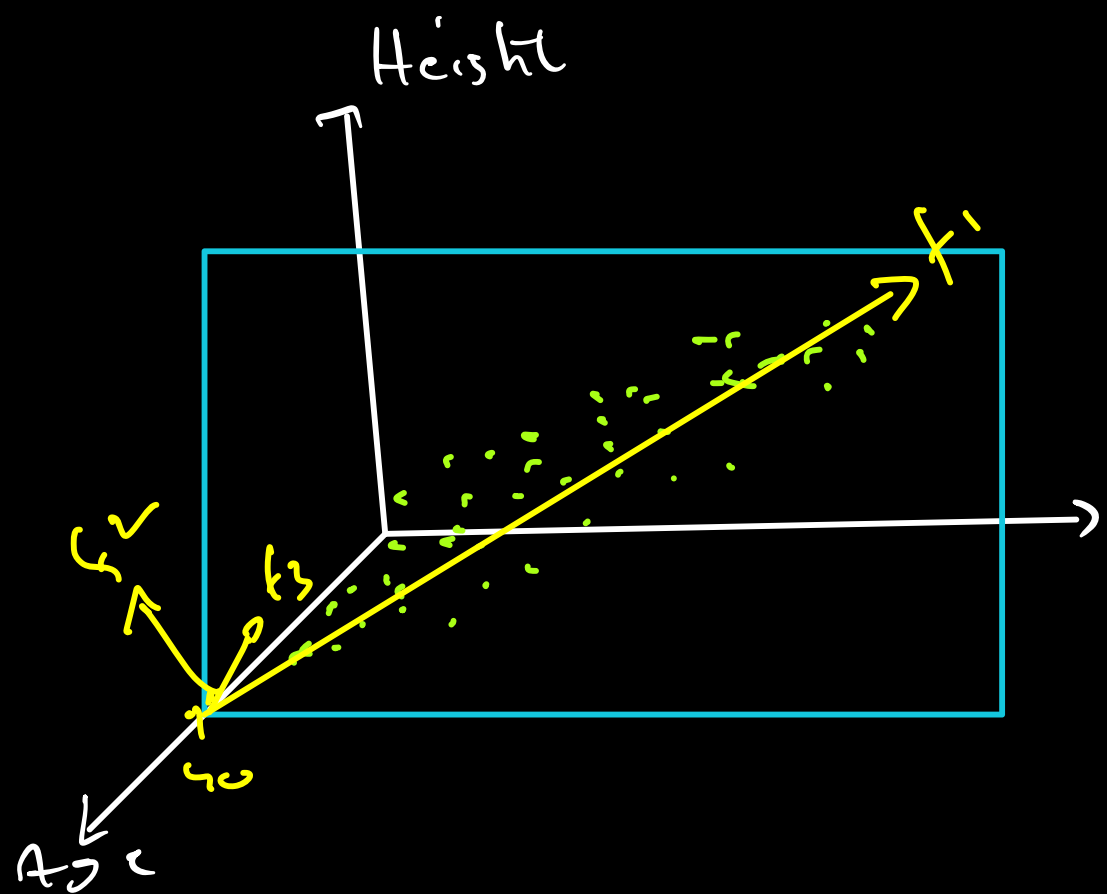
Q: which feature?

a) f1

b) f2

c) $f_1 + f_2$

d) $f_1 - f_2$



Q: What is the order of feature importance?

a) $f_1 > f_2 > f_3$

b) $f_3 > f_2 > f_1$

c) $f_1 > f_3 > f_2$

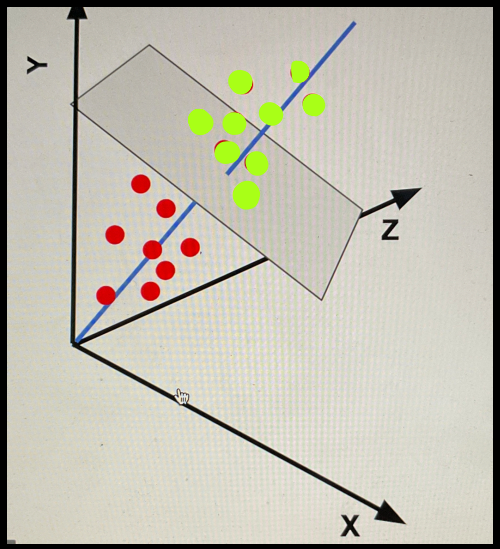
Goal:

→ We see that all the features don't carry equal amounts of info.

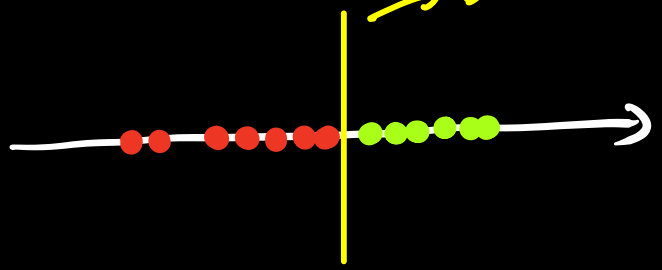
→ Sometimes in ML, we need to reduce the number of features.

- Visualisation
- Compression (Space saving)
- Faster ML training

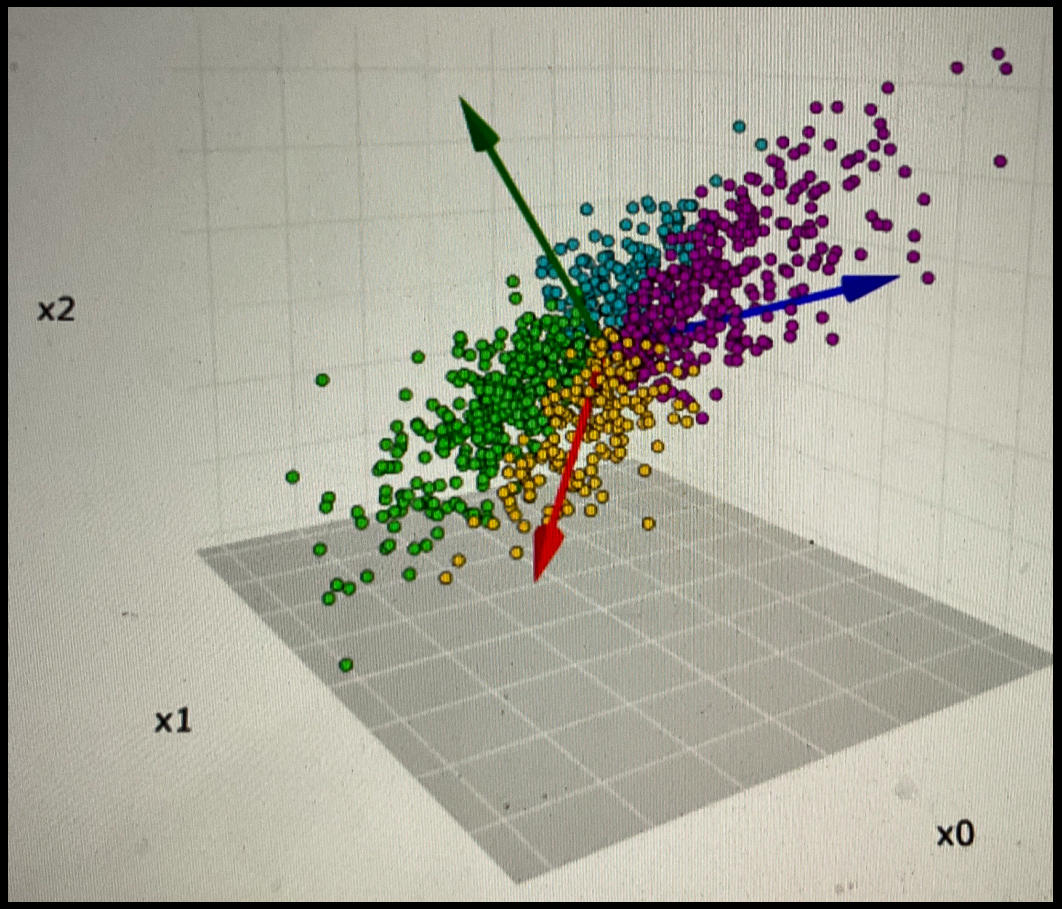
3D → 1D

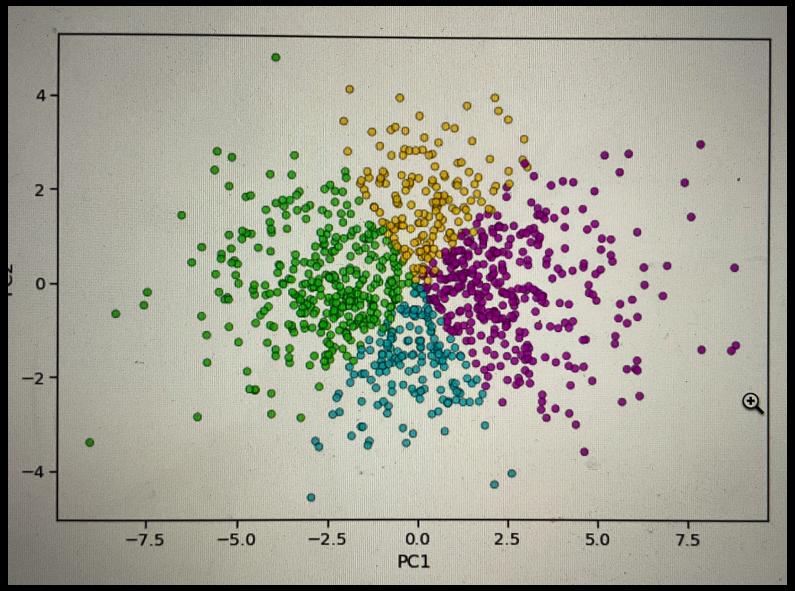


→ Decision boundary



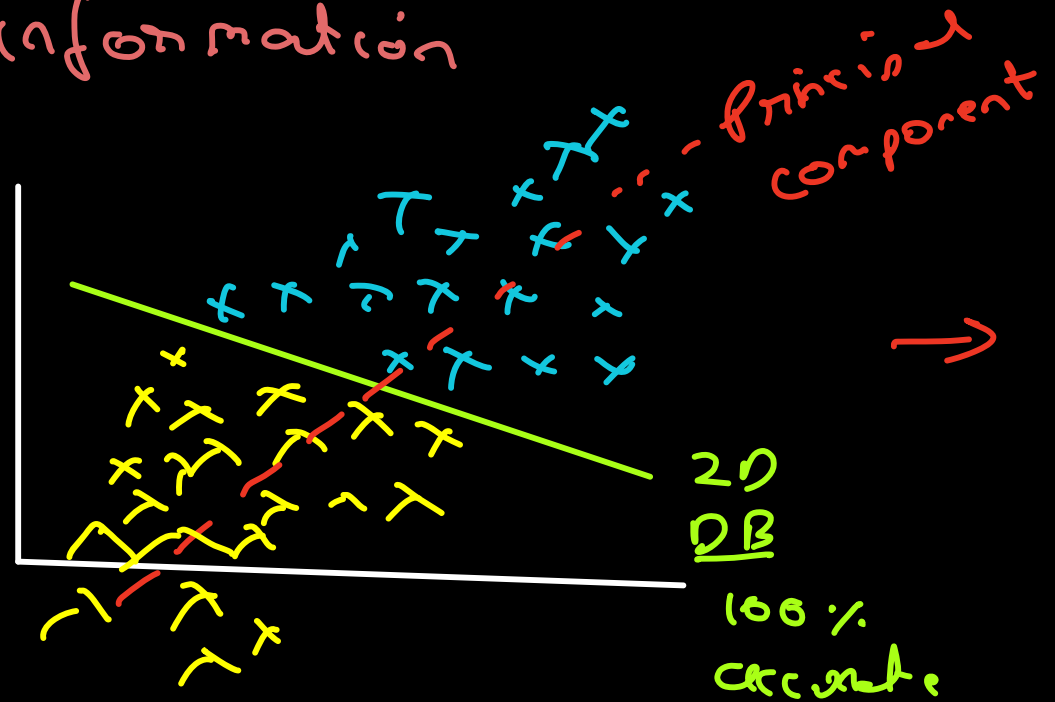
3D - 2D

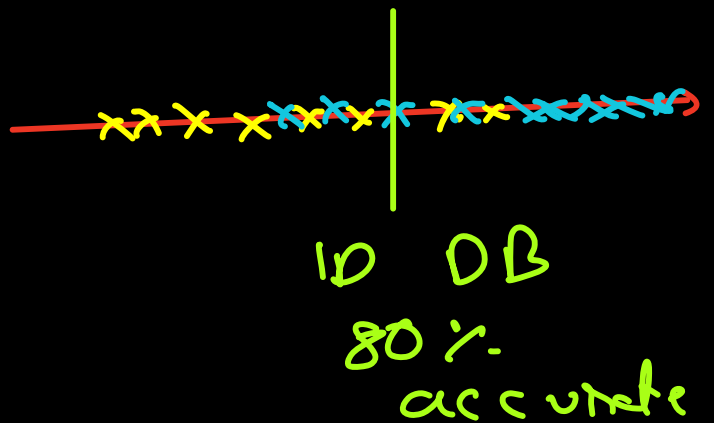




Principal component analysis refers to the act of finding new axis to represent the data so that a few principal components may contain most information.

However, this may lead to some loss of information



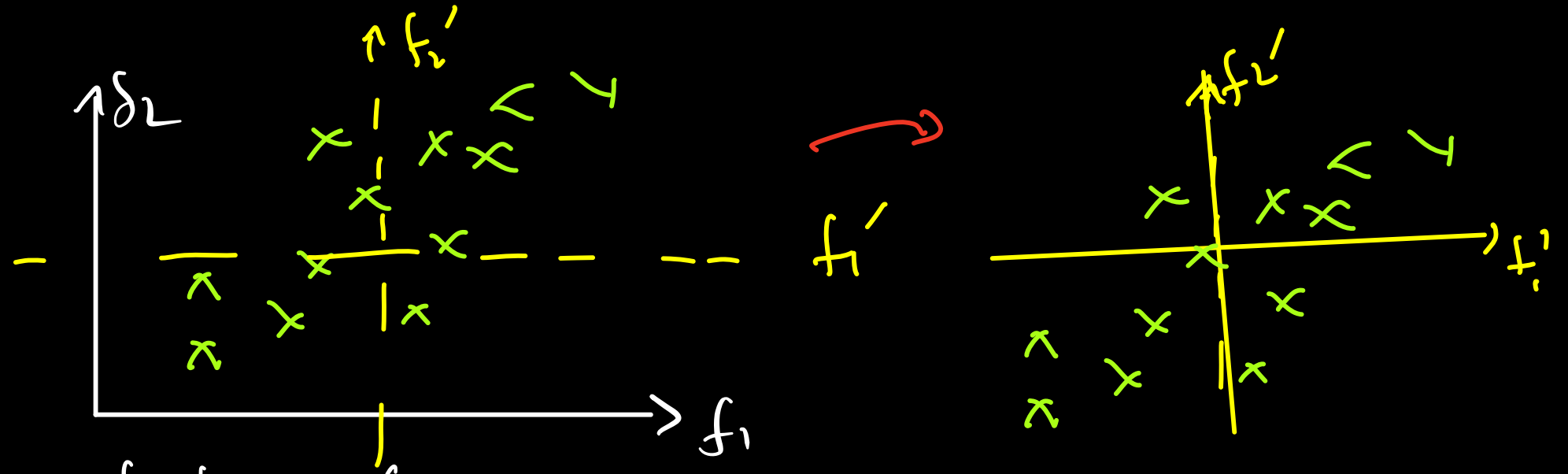


PCA Mathematics

→ Change of axis

→ Retain most information in few axis.

Step-1 : Centering



f_1 f_2 .. f_n

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad X$$

centers $\longrightarrow$ $[\bar{f_1}\ \ \bar{f_2}\ ..\ \bar{f_n}]$

$X - X.mean() = X_c$

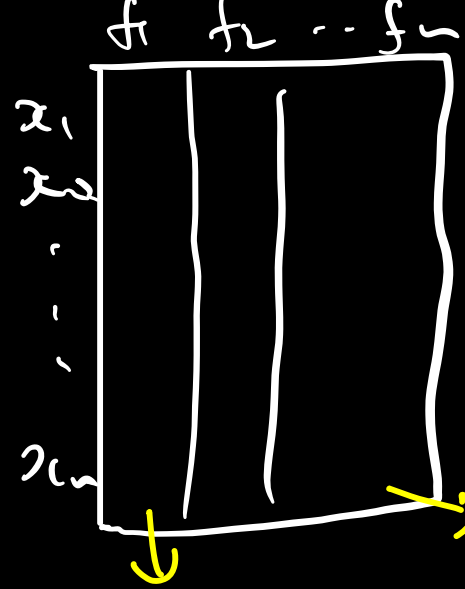
Step -2: Standardize / Scaling

We need to make different features such as height (4 - 6) ft and weight (40 - 150) kg comparable.

Best is to use Standard Scaling because it is less affected by outliers

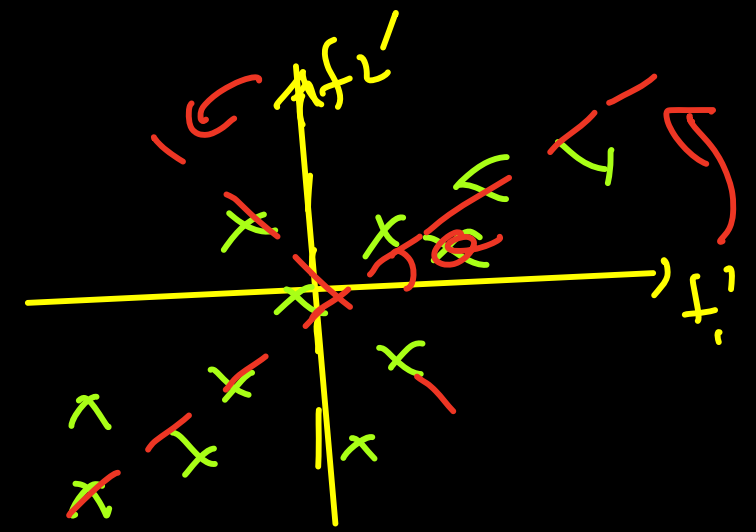
Combining step 1 and 2

$$x' = \frac{x - x.mean()}{x.std()}$$



→ use sklearn

Step - 3: Rotate

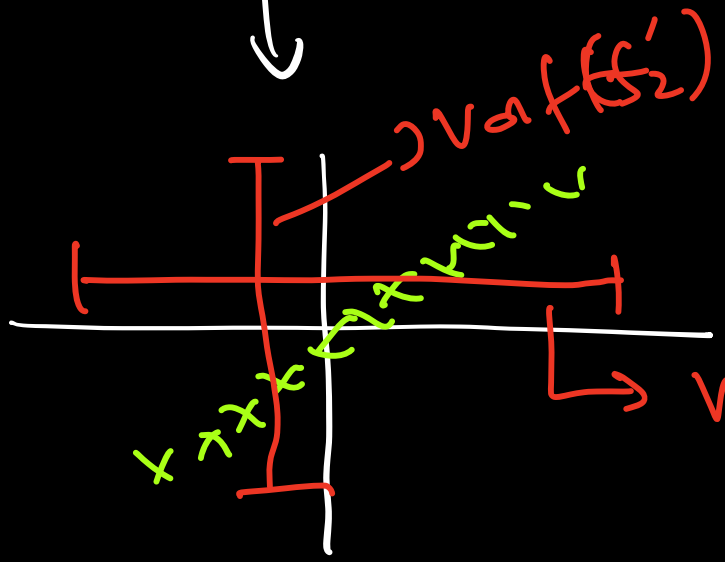


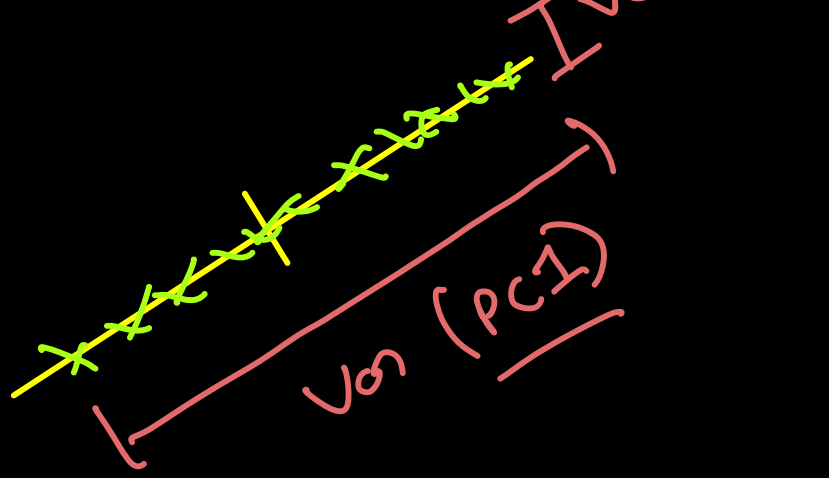
How to rotate?

Q: Any suggestions on how to determine the best axis?

→ maximise variance

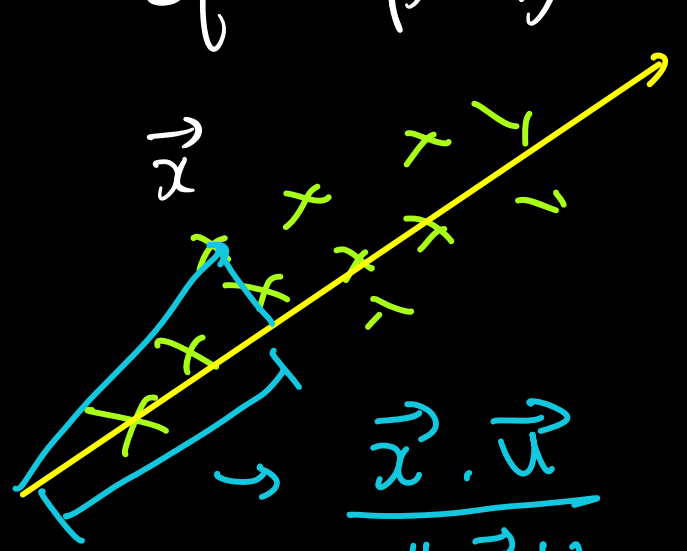
→ max length of projection





$$\text{var}(PC1) > \text{var}(f_1') > \text{var}(f_2') > \text{var}(PC2)$$

We can also think in terms of length of projection:



Best $\vec{v}$ will be where

$$\max_{\vec{v}} \sum_{i=1}^{n} \frac{\vec{x_i} \cdot \vec{u}}{\|\vec{v}\|} \cdot \frac{1}{n}$$

However, projections can also be -ve

$$\max_{\vec{v}} \sum_{i=1}^{n} \frac{|\vec{x_i} \cdot \vec{u}|}{\|\vec{v}\|} \cdot \frac{1}{n}$$ → But this is not differentiable

$$\max_{\vec{u}} \frac{1}{n} \sum_{i=1}^{n} \frac{(\vec{x_i} \cdot \vec{u})^2}{\|\vec{u}\|}$$ → still hard to diff

↓

Constraint: $\|u\| = 1$

↓

$$\max_{\vec{u}, \lambda} \frac{1}{n} \sum_{i=1}^{n} (\vec{x_i} \cdot \vec{u})^2 + \lambda(\|\vec{u}\| - 1)$$

Lagrangian loss function for P.C.A

This can be solved using gradient descent.

However there is also a direct

Linear algebra way.

Math Ahead!! → Not required to remember/reproduce

→ We will use some identities w/o proof

→ We will learn a new concept

(digression cold snow (harmless))

Variance ↓

$$\Rightarrow \max_{\vec{u},\lambda} \frac{1}{n}\sum_{i=1}^{n}(\vec{x_i}\cdot\vec{u})^2 + \lambda(\|u\|^2 - 1)$$

$$\Rightarrow \max_{\vec{u},\lambda} \frac{(X\vec{u})^2}{n} + \lambda(\|u\|^2 - 1)$$

identity : $A^2 = A^T A = \|A\|^2$

if A is invertible square matrix

$$\rightarrow \max_{\vec{u},\lambda} \frac{(X\vec{u})^T(X\vec{u})}{n} + \lambda(u^T u - 1)$$

identity $(AB)^T = B^T A^T$

$$\rightarrow \max_{\vec{u},\lambda} \vec{u}^T \frac{X^T X}{n} u + \lambda(u^T u - 1)$$

$$\Rightarrow \max_{\vec{u},\lambda} \quad u^T V u + \lambda(u^T u - 1)$$

V matrix:

$$\frac{X^T X}{n} \quad \frac{1}{n}\begin{bmatrix} \vdots \\ f_1 \\ \vdots \\ f_d \end{bmatrix}_{d \times n} \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T \quad X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_{n \times d} \begin{bmatrix} f_1 & f_2 & \cdots & f_d \end{bmatrix}$$

V = Covariance matrix

$$= \begin{bmatrix} \mathrm{var}(f_1) & \mathrm{cov}(f_1, f_2) & \cdots & \mathrm{cov}(f_1, f_d) \\ & \mathrm{var}(f_2) & \cdots & \vdots \\ & & & \mathrm{var}(f_d) \end{bmatrix}$$

Is This matrix has pairwise covering of all features [similar to pd.corr()]

So now we have

$$L = u^T V u + \lambda(u^T u) - \lambda$$

$$\frac{\partial L}{\partial u} = 2Vu + 2\lambda u$$

$$\frac{\partial L}{\partial \lambda} = u^T u - 1$$

Now we set both to $= 0$

$\therefore u^T u = 1$

identity

$\frac{\partial A^T B A}{\partial A} = 2BA$

$\frac{\partial A^T A}{\partial A} = 2A$

this is like

$\frac{\partial x \cdot x}{\partial x} = 2x$

but in matrix

$$Vu = -\lambda u$$

$$Vu = \lambda' u$$

matrix · vector · scalar → vector

Interesting

→ we have V, which is just pp.cov(X)

→ $\vec{v}$ must be such that when I multiply V to it, it is equivalent to multiplying some scalar to it.

Eigen Values and Vectors

For any matrix A, there exists* one or more vectors, such that

$$A\vec{x} = \lambda\vec{x}$$

$\vec{x}$ → eigen vector

λ → eigen value

Geometrically

Lets say: $\vec{x} = \begin{bmatrix} 2 \\ 5 \end{bmatrix}$, $A = \begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix}$

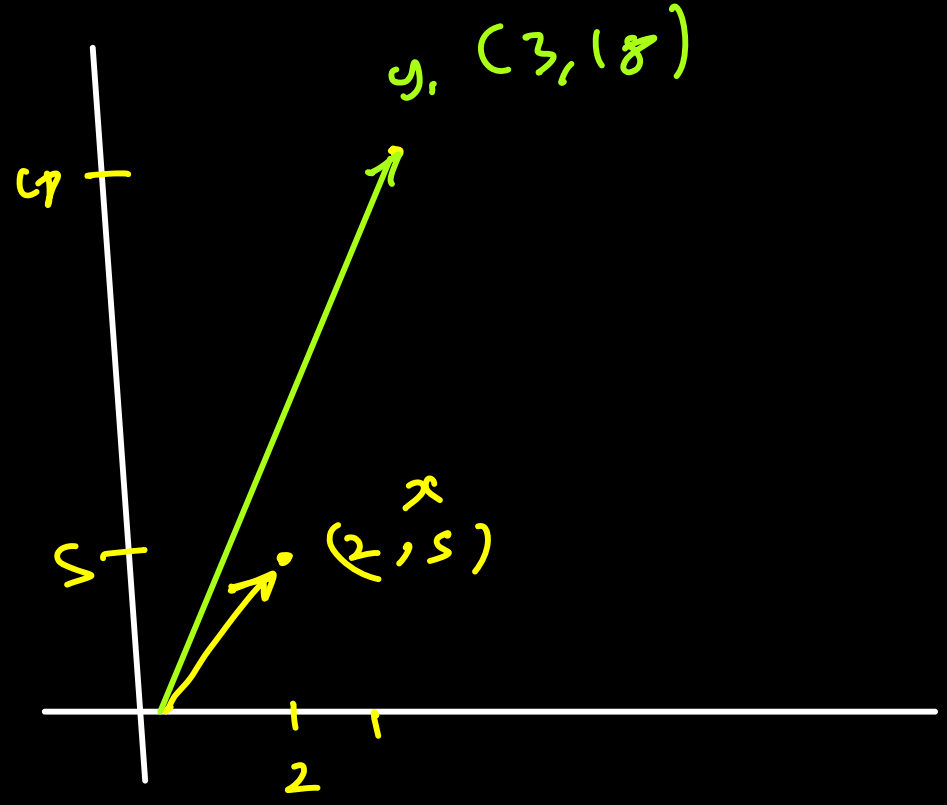
when we multiply $A\vec{x} = \vec{y}$, then

$\vec{y}$ has a magnitude and direction, but if the direction of $\vec{x}$ and $\vec{y}$ are same, then $\vec{x}$ is said to be an eigen vector of A with eigen value

$$\lambda = \frac{\|y\|}{\|x\|}$$

There can be multiple eigen vectors, which are always orthogonal (⊥) to each other.

Eg: $\begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} 2 \\ 5 \end{bmatrix} = \begin{bmatrix} -12 + 15 \\ 8 + 10 \end{bmatrix} = \begin{bmatrix} 3 \\ 18 \end{bmatrix}$

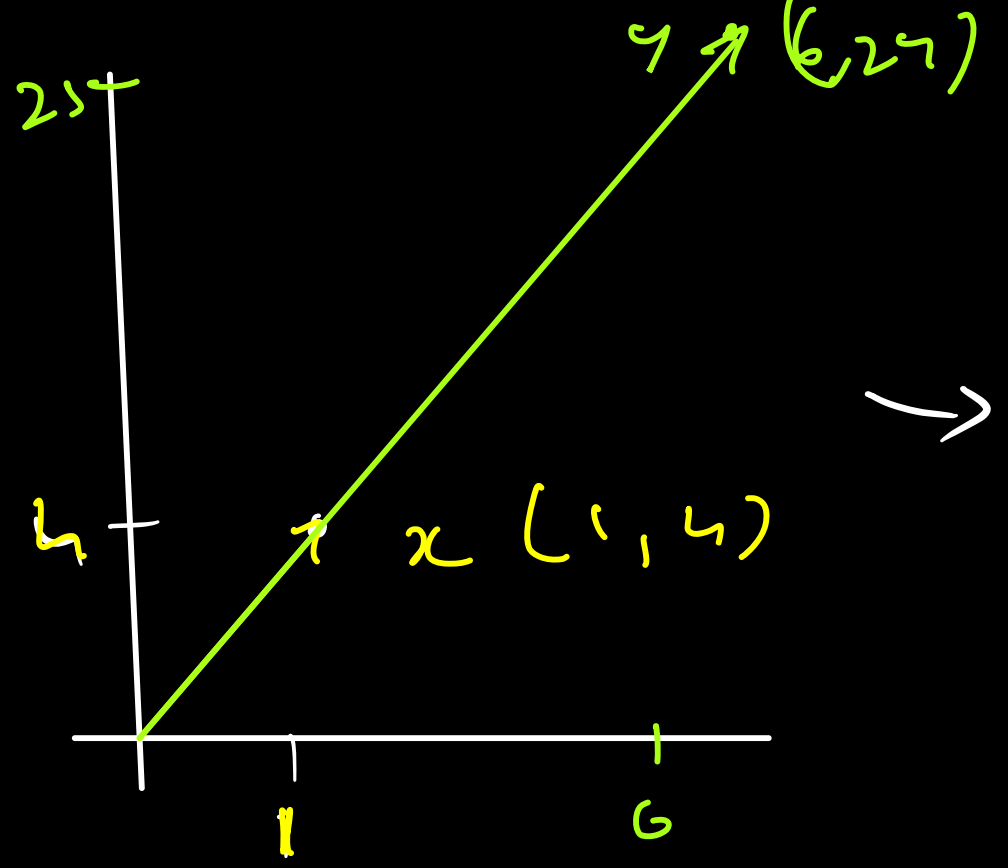


But, if $x = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$

$$\begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 4 \end{bmatrix} = \begin{bmatrix} 6 \\ 24 \end{bmatrix} = 6 \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

eigen value

eigen vector



$y = \underline{6 \cdot x}$

→ x is such a vector that will not change direction when multiplied by 6

We can find eigen values and vectors very easily in numpy.

→ code