Convert d dim. to b dim. $b < d$

why?

- visualization (usually 2 or 3 features)
- to get less noisy and irrelevant features
- to reduce the size (to be easier in computing)
  (to avoid out of memory)

- It's NOT feature selection

  IT is Transformation (new features)

	X		+
S1	5	1	4
S2	5	3	3
S3	5	5	4
S4	5	7	3

① remove low variance

② projection

Projection is

$$x \cdot u = u \cdot x = u^T x = \sum_{i=1}^{d} u_i x_i.$$

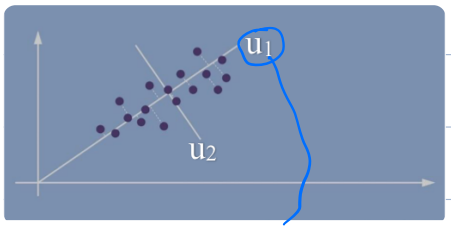
features must be normalized

How to transform:-

① PCA (linear)

come up with new dimentions that have high variance

natural coordinate



Vector with maximum variance

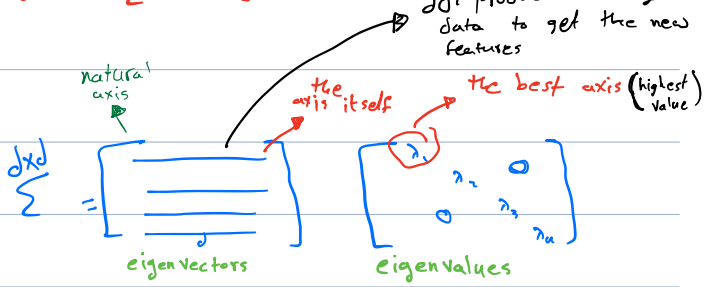
to project any point to any axis ⇒ dot product

variance of X in direction u

total $\sigma = u^T \Sigma u$

$$[1,0] \begin{bmatrix} 1 & 0.85 \\ 0.85 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

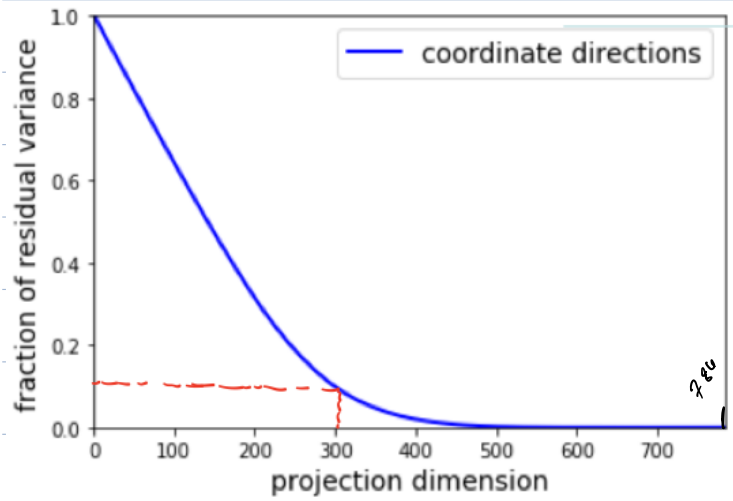
in python SVD



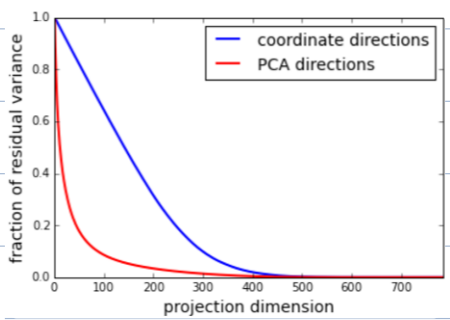
if you don't have exact # of axis you fix % of σ then get $\lambda_1 \dots \lambda_n$ that give you % you want

How σ I will get if I select 3

$$\sigma = \frac{\lambda_1 + \lambda_2 + \lambda_3}{\sum \lambda}$$



if we keep 300 features we will lose just 10% of the variance

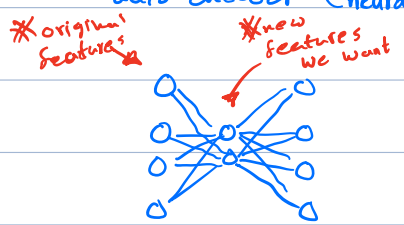


● B: If you use PCA to project d-dimensional points down to j principal coordinates, and then you run PCA again to project those j-dimensional coordinates down to k principal coordinates, with $d > j > k$, you always get the same result as if you had just used PCA to project the d-dimensional points directly down to k principle coordinates.

True

② T-sne (non-linear)

auto-encoder (neural network)



loss = squared loss

$$-y \log \hat{y} - (1-y) \log (1-\hat{y})$$

$$-y \log \hat{y}$$

$$C = 1 - \max p_i$$

$$\text{Gini} = 1 - \sum (p_i)^2$$

$$\text{Entropy} = -\sum p_i \log_2 p_i$$

$$IG(D_p) = I(D_p) - I(D_l) \frac{N_l}{N_p} - I(D_R) \frac{N_R}{N_p}$$

$$J = \text{error rate} + \lambda \ (\#\text{ of leafs})$$

$$e = \frac{\sum \text{of mistakes}}{\sum \text{of All}} \qquad \propto e^{-x}$$

$$x = \frac{1}{2} \ln \left(\frac{1-e}{e} \right)$$