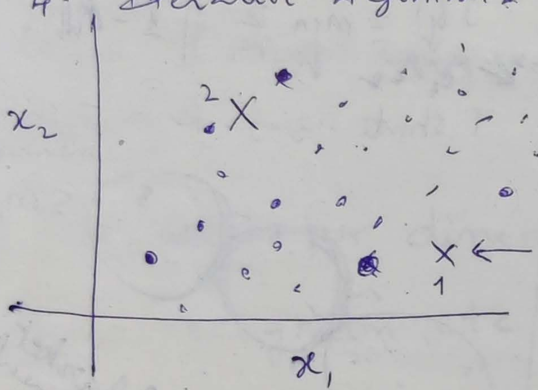
K-means Clustering

1

1. Unsupervised learning. Only $\underline{x}$ (input data/features) given
No labels. Training set consists of $\{\underline{x}^{(1)}, \underline{x}^{(2)}, \cdots \underline{x}^{(m)}\}$
2. Finds coherent subsets from data - Most popular clustering algorithm
or similar. Often the similarity is measured in terms of distance.
3. Application: Market segmentation, Social network analysis,
Astronomical data analysis
4. Iterative Algorithm. Segments the data ~~in~~ ~~to~~ K clusters.

Consider 2D data points $\underline{x}^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \end{bmatrix}$



← cluster centroids (here two cluster centroids chosen i.e. K = 2)

How it works:
The Algorithm involves two steps (iteratively)

1. Cluster Assignment step
2. Move centroid step

1. cluster assignment step: Assign each data point to one of the clusters based on minimum distance criteria.
2. Move the centroid step: Compute the average (mean) of the data pts (vectors) belonging to each centroid and get the new set of clusters

nce Additional iterations
nverged ↳ do not change the cluster centroids. The final two clusters are those which are assigned to these centroids.

Algorithm: Input
K (number of clusters) & Training set, $\underline{x}^{(i)} \in R^n$

Randomly choose K cluster centroids (K data vectors)

$\underline{\mu}_1, \underline{\mu}_2, \ldots \underline{\mu}_K \in R^n$

Repeat {

cluster assignment → for i = 1 to m

$c^{(i)}$ = index of cluster centroid with minimum distance (a number between 1 & K)

$\min_k \|\underline{x}^{(i)} - \underline{\mu}_k\|^2$ → If $\underline{x}^{(i)}$ closer to 1 assign it to μ_1 else μ_2 for $\forall i$ to m, $k = 1, 2 \ldots K$ for $K = 2$

→ objective f? is formed using all examples $\sum_{i=1}^{m}$

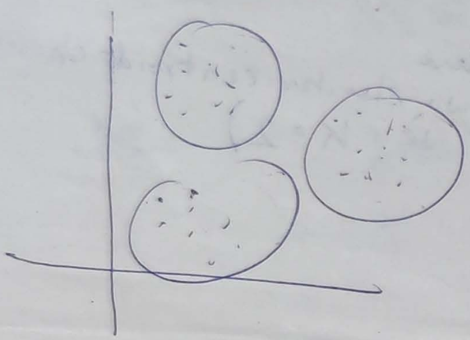
for k = 1 to K

μ_k = mean of data points assigned to k

}

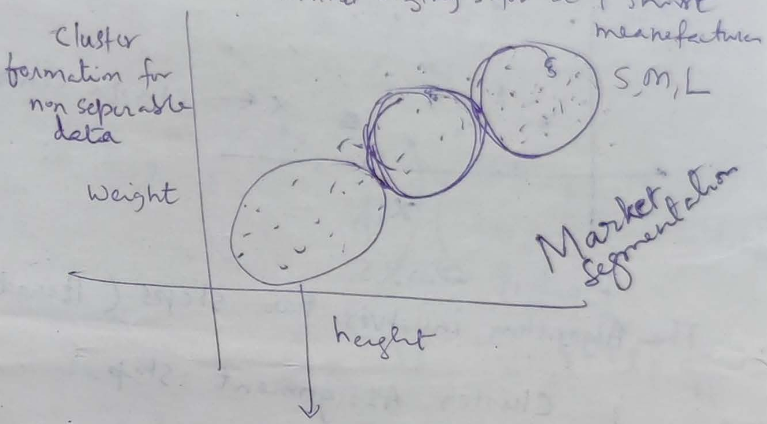
~~$J(c, \mu) = \min_k \sum_{i=1}^{m} \|\underline{x}^{(i)} - \underline{\mu}_k\|^2$~~

~~$k=2$ ... μ_1, μ_2, c_1, c_2~~



K = 3

T shirt sizing - for a T shirt manufacturer S, M, L



look at these height & weight to design a T shirt for S, M & for M, L

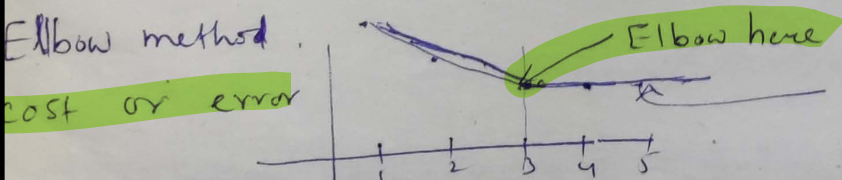
How to avoid local optimum

Different initialization may lead to different solutions (Clusters) giving local optimum.

Try multiple random initializations to get global optimum (say 100 times) and choose those clusters that give minimum error.

How to choose number of clusters (K): There is no perfect ~~right~~ answer for this. Difficult to find.

Elbow method



error is almost constant so choose K=3. This method may not always work because you may not get clear elbow.

Principal Component Analysis (PCA)
(An nsupervised ML method)

- Machine Learning:

A technique to discover meaningful patterns in large amount of data.

Supervised – Classification problem (Logistic regression, SVM, Random Forest etc)

Unsupervised – Clustering problem (K-means clustering, PCA, ICA, GMM etc)

Basics we need to know:

- Consider a square matrix A

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

Orthogonal matrix

- A square matrix with real entries. Both rows and columns are orthonormal
- Orthonormal: Orthogonal + unit norm
- Row/column vectors have unit norm (length) and dot product is zero between vectors
- Complex version : Unitary matrix

- Consider a square matrix of size NXN

$$A^T A = I \quad if \quad A \; is \; orthogonal,$$

$$So \qquad A^{-1} = A^T$$

$$A^{-1} = A^{*T}$$ For unitary matrix

Orthogonal transform

The picture can't be displayed.

- It is the transformation using a orthogonal matrix
- Let x be input vector and y is the transformed vector, then $y = T[x]$
- That is y=Ax where A is the transformation matrix, A being orthogonal matrix. Here T is linear

- Transform domain (Analysis) $y = Ax$

- Inverse transform (synthesis)

$$x = A^{-1}y, \quad or \quad x = A^{T}y$$

- No need to find the inverse of a matrix to get back x!

$$Complex \quad A, \ orthogonal \ matrix \ is \ called \ unitary \ matrix$$

$$A^{-1} = A^{*T}$$

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

$$\mathbf{x} = \mathbf{B}\mathbf{y} \quad (\mathbf{B} = \mathbf{A}^{-1} = \mathbf{A}^{T})$$

– N=2

$$y_n = \sum_{i=0}^{N-1} x_i a_{n,i}$$

$$\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$

Transform coefficients

Basis vectors

$$\begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{10} \\ a_{01} & a_{11} \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = y_0 \begin{bmatrix} a_{00} \\ a_{01} \end{bmatrix} + y_1 \begin{bmatrix} a_{10} \\ a_{11} \end{bmatrix}$$

$$a_{00}a_{10} + a_{01}a_{11} = 0 \qquad (\textit{orthogonal basis vectors})$$

$$\sqrt{a_{00}^2 + a_{01}^2} = \sqrt{a_{10}^2 + a_{11}^2} = 1 \qquad (\textit{norm is unity})$$

Why orthogonal transforms?

- One can see that orthogonality preserves energy/power (Parsevel's theorem) $y^T y = x^T A^T A x = x^T x$
- Tend to redistribute energy. Most of it in few transformed coefficients.
- Very useful in compression, where few coefficients can represent the original signal.

Orthogonal transforms very useful in signal/image compression

- Correlation: Speech, images (x in earlier slide) etc. have highly correlated samples or pixels i.e., they have gradual variations with occasional discontinuities.

- Decorrelation: Transformation using orthogonal matrix A results in (un)decorrelated values y.

- Correlated data has redundancy i.e., more samples than what is required to represent.
- Decorrelation: No correlation between sample values.

Discrete cosine transform (DCT) (An Orthogonal Transformation)

$$y(k) = C(k).\sum_{n=0}^{N-1} x(n)\cos\left[\frac{\pi(2n+1)k}{2N}\right]$$

$$x(n) = \sum_{k=0}^{N-1} C(k)\,y(k)\cos\left[\frac{\pi(2n+1)k}{2N}\right]$$

$$C(k) = \sqrt{\frac{1}{N}} \quad for\ k = 0, \qquad and\ \sqrt{\frac{2}{N}} \quad for\ k \neq 0$$

Let, N=2

$$\begin{bmatrix} y(0) \\ y(1) \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \end{bmatrix}, \qquad y = Ax$$

$$\begin{bmatrix} x(0) \\ x(1) \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} y(0) \\ y(1) \end{bmatrix}, \qquad x = A^{-1} y = A^T y$$

DFT (Unitary Transform)

The picture can't be displayed.

$$y(k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}nk}, \; k = 0,1,\ldots,N-1, \qquad (1)$$

$$x(n) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} y(k) e^{+j\frac{2\pi}{N}nk}, \; n = 0,1,\ldots,N-1, \qquad (2)$$

- Consider N=4 (4 point DFT)

$$\begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \qquad y = Bx$$

$$\begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix} = \tfrac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} \begin{bmatrix} X[0] \\ X[1] \\ X[2] \\ X[3] \end{bmatrix}$$

$$x = B^{-1} X = B^{*^T} X$$

Advantages of DCT over DFT

- Basis vectors are real i.e. A is real. So, less computational complexity.
- Better energy compaction.
- Meaning of energy compaction: The sum of squared error between the source signal x[n] and reconstructed signal with k<N transformed coefficients i.e., truncated number of transform coefficients.

Application DCT

- Most important is in image compression JPEG uses DCT.

An 8×8 block from the Y image of 'Lena'

200	202	189	188	189	175	175	175
200	203	198	188	189	182	178	175
203	200	200	195	200	187	185	175
200	200	200	200	197	187	187	187
200	205	200	200	195	188	187	175
200	200	200	200	200	190	187	175
205	200	199	200	191	187	187	175
210	200	200	200	188	185	187	186

$f(i,j)$

515	65	-12	4	1	2	-8	5
-16	3	2	0	0	-11	-2	3
-12	6	11	-1	3	0	1	-2
-8	3	-4	2	-2	-3	-5	-2
0	-2	7	-5	4	0	-1	-4
0	-3	-1	0	4	1	-1	0
3	-2	-3	3	3	-1	-1	3
-2	5	-2	4	-2	2	-3	0

$F(u,v)$

After quantization, coefficients are coded.

Fig. 9.2: JPEG compression for a smooth image block.

PCA (Orthogonal Transform)

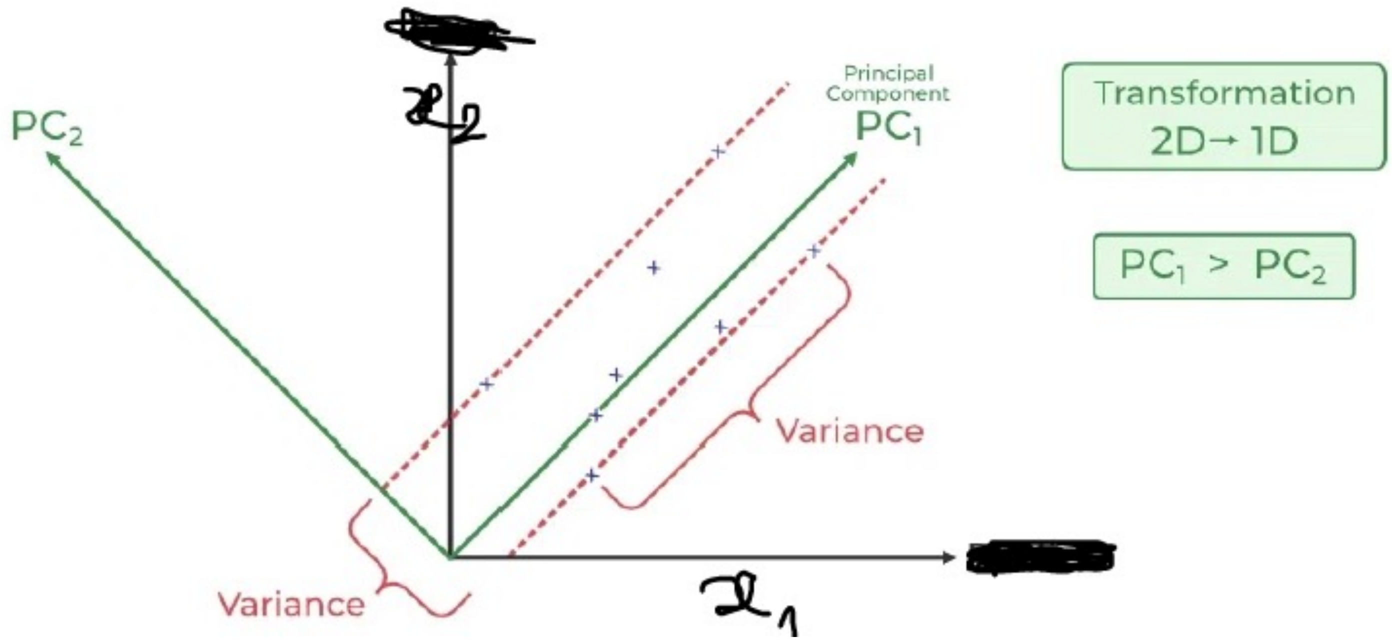
- Optimum transform (in MSE sense), better than DFT and DCT
- Why Optimum? Because the transformation matrix is derived from input data (X)
- Rows of transformation matrix correspond to eigen vectors (orthogonal directions) obtained from covariance matrix of input data

How to derive A in PCA

- Given X (input data), PCA performs Y=AX
- To understand the concept consider each data point is 2D (in practice if we consider an image NXN, it would be N^2) and there are m data points.
- Let us approximate 2D with 1D only by PCA (dimensionality reduction or 2D original data representation as 1D feature)

$$Y \qquad = \qquad A \qquad X$$

$$\begin{bmatrix} y_1^{(m)} \ldots\ldots\ldots\ldots y_1^{(1)} \\ y_2^{(m)} \ldots\ldots\ldots\ldots y_2^{(1)} \end{bmatrix}_{2xm} = \begin{bmatrix} & \\ & \end{bmatrix}_{2x2} \begin{bmatrix} x_1^{(1)} \ldots\ldots\ldots\ldots x_1^{(m)} \\ x_2^{(1)} \ldots\ldots\ldots\ldots x_2^{(m)} \end{bmatrix}_{2xm}$$


PC2
Principal Component
PC1
Transformation
2D→1D
PC1 > PC2
Variance
Variance
x2
x1

Covariance matrix

- 2D data so 2X2 covariance matrix

$$C_X = data\ \ \text{cov}\,ariance\ \ matrix = \begin{bmatrix} \text{cov}(X_1, X_1) & \text{cov}(X_1, X_2) \\ \text{cov}(X_2, X_1) & \text{cov}(X_2, X_2) \end{bmatrix}$$

$$\text{cov}(X_{1,}X_2) = E(X_1 X_2) - E(X_1)E(X_2)$$

$where\, X_1\, (first\, row\, values)\, and\ \ X_2\ \ (\text{sec}\, ond\ \ row\ \ values)\, are\, the\ random\ \text{var}\, iables\, with\ m\ values\ each.$

- Compute 2 eigen values and 2 eigen vectors (each of size 2X1) from covariance matrix
- Looking at the eigen values, place the eigen vector with larger value as the first row and the other eigen vector as the 2^{nd} row. This gives the transformation matrix A of size 2x2.

- Now we can compute the Y matrix (2xm)
- Note: With this Y and X, if we take the inverse, we get back exact X matrix (2xm).
- However, beauty of PCA is, if only first row of Y is used for inverse (with second row filled with zeros), you can still get X matrix (2xm) with almost similar values of original (i.e., exact X matrix)
- Important point: covariance matrix of Y is diagonal i.e., Y1 and Y2 are decorrelated.

In Lab on PCA

- Input X is of dimension 112x92x400=10304x400 face images
- There are 10304 random variables. So covariance matrix is of size 10304X10304
- Using this matrix we get 10304 eigen vectors each of size 10304x1
- These are used as rows of transformation matrix A (first row with highest eigen value and so on)

- First few rows represent principal components (depends on how many we retain while taking the inverse to get reconstructed x with as low MSE as possible)
- Note: eigen vectors are orthogonal and have unit norm

- We have Y=AX

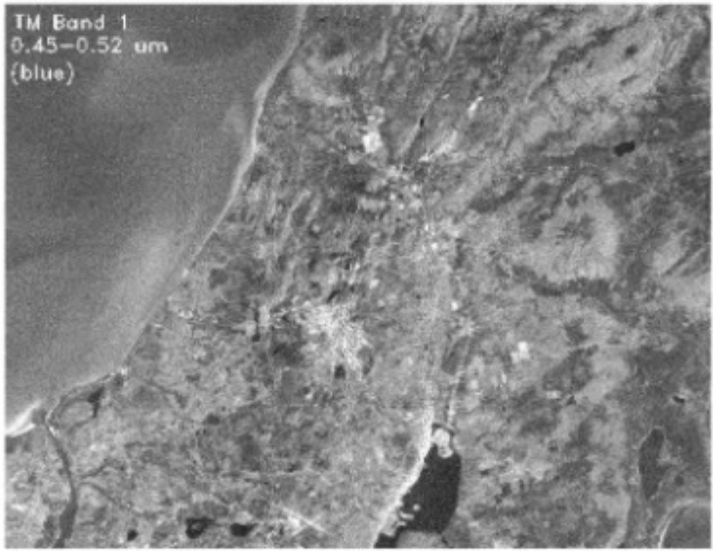
- Since A is 10304x10304 and X is 10304x400, Y will be 10304x400 (transformed matrix i.e., output matrix)
- If we perform X=A transpose Y, we get back X with no error (MSE between original X and reconstructed X = 0)

- However, let us say we have retained 100 eigen vectors (all other rows filled with 0s). The A becomes 100x10304 and A transpose will be 10304x100.
- Then Reconstructed x = (A transpose) Y is still 10304x400. This can be done by making Y as 100x400 by retaining only 100 rows of Y

- Each reconstructed image has error (but by retaining about 400 rows we saw in lab that the error is minimum)
- This means each image which has originally 10304 values (pixels) can be represented in PCA domain by using only 400 values achieving compression or dimensionality reduction.

PCA compression after 93% power (variance) retained in Y (only 3 out of 7 transformed images of Landsat 7 images retained)

band 1



PC 1