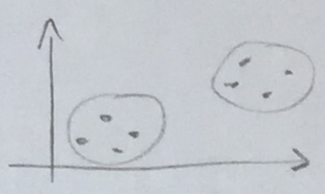
ML COURSERA: WEEK 8 | LEO ROBINOVITCH

Unsupervised Learning: Clustering

⇒ Unlabeled data (no y vector) → give data, find structure in data!

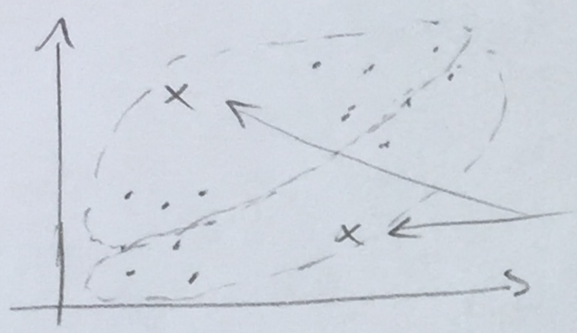
eg. find clusters:



⇒ Clustering → social networks / graphs, market segmentation, resource allocation, galaxies, etc

⇒ K-Means Algorithm: groups data into coherent subsets (clusters)

eg.



⇒ assign all pts to one centroid or another (cluster assignment)

⇒ then move cluster centroids to mean location of all pts in their group

↳ eventually, cluster centroids converge to single location.

⇒ parameters in k-means:
- K (# clusters)
- training set $(x^{(1)} \dots x^{(m)})$
  - ↳ note no $x_0 \rightarrow x^{(i)} \in \mathbb{R}^n$

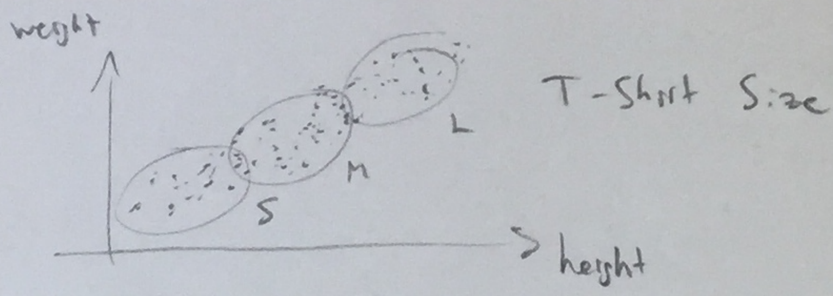
Algorithm: Randomly initialize K cluster centroids $\mu_1, \mu_2 \dots \mu_K \in \mathbb{R}^n$

repeat:

CLUSTER ASSIGNMENT:
for i = 1 to m (all pts.)
  $c^{(i)}$ = index (from 1 to K) of centroid closest to $x^{(i)}$ $\left(\min \|x^{(i)} - \mu_k\|^2\right)$

MOVE CLUSTER CENTROIDS:
for k = 1 to K
  μ_k = average of pts. assigned to cluster k ($\mu_k \in \mathbb{R}^n$)

=> Non-separated clusters:



↳ useful for market segmentation!

Optimization Objective of K-Means:

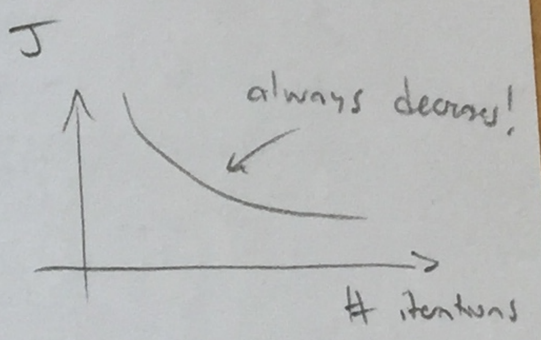
=> keeping track of $c^{(i)}$'s (cluster index to which given $x^{(i)}$ is assigned) and μ_K (loc. of cluster centroid) -> combined, $\mu_{c^{(i)}}$ is centroid to which $x^{(i)}$ assigned

$J(c^{(1)} \dots c^{(m)}, \mu_1 \dots \mu_K) = \frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - \mu_{c^{(i)}} \|^2$ -> average squared distance b/n $x^{(i)}$ & centroid it is assigned to

$\min_{\substack{c^{(1)} \dots c^{(m)} \\ \mu_1 \dots \mu_K}} J(c^{(1)} \dots c^{(m)}, \mu_1 \dots \mu_K)$ (Distortion cost function)

↳ cluster assignment step minimizes J over $c^{(1)} \dots c^{(m)}$ holding μ's fixed

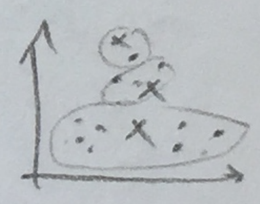
move centroid step minimizes J w.r.t. $\mu_1 \dots \mu_K$



=> Random Initialization: best method!

=> $K < m$. Randomly pick K examples -> set $\mu_1 \dots \mu_K$ = these examples

=> depending on initialization, can hit local optimum ->



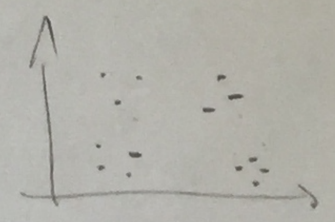
↳ try multiple initial guesses to get global optimum

for $i = 1 \to 100$: run K-means, get $J^{(i)}$ -> pick lowest $J^{(i)}$
(w/ unique initialization)

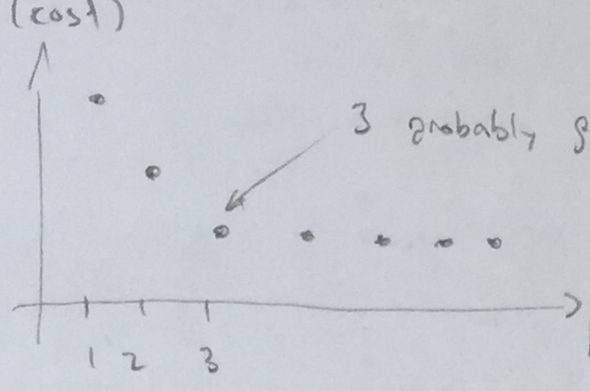
=> Choosing K, # clusters:

↳ most commonly → visually / by hand

↳ sometimes hard / ambiguous (part of unsupervised learning)

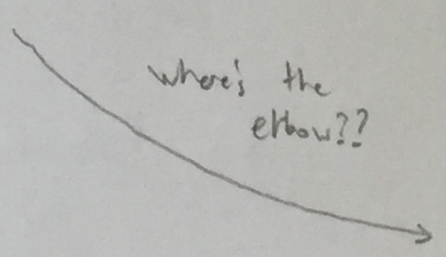


=> elbow method:



=> NOT USED OFTEN:

usually

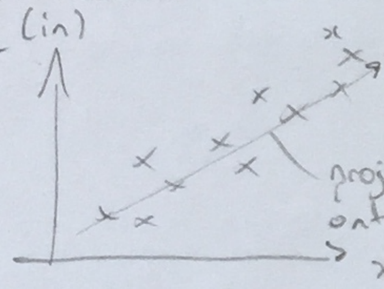


=> can choose # clusters based off results in some downstream purpose

eg. # t-shirt sizes

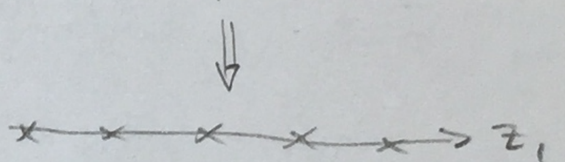
Dimensionality Reduction: compression

eg)



rounded, ∴ not

=> redundant! Only need 1D, not 2D



$x \in \mathbb{R}^2$

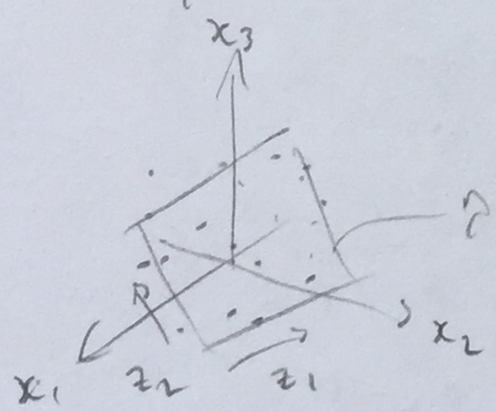
$z \in \mathbb{R}^1$

↳ approximation of original training set!

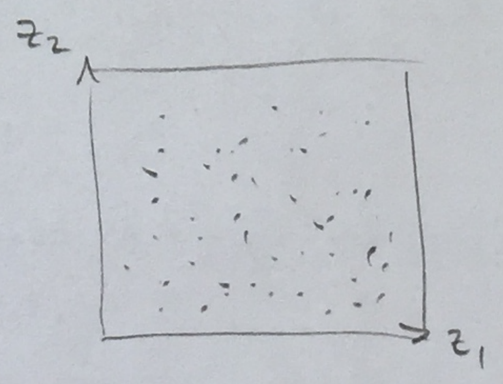
=> examine correlation b/n features

=> reduces disc space reqd & speeds up learning algorithms

=> Compression → 3D to 2D:



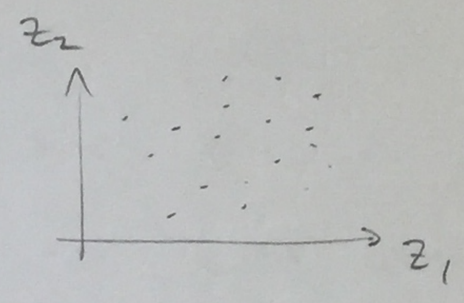
project data onto plane:



↳ data should already more or less lie on a plane (strong correlation)

Hyper Dimensional Data Viz:

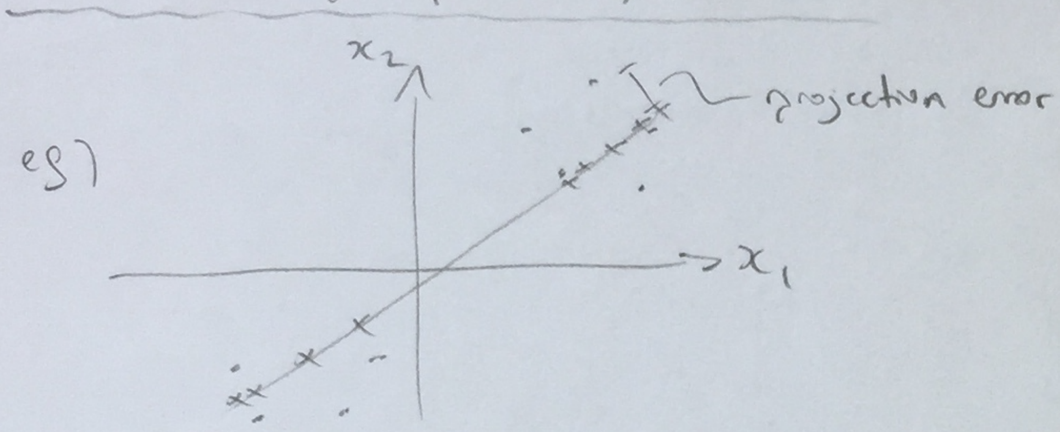
Say $x \in \mathbb{R}^{50}$. Reduce $x \to z \in \mathbb{R}^2$



eg. $z_1 \propto$ country size, x_1 = pop.
$z_2 \propto$ per person GDP x_2 = GDP
x_3 = land area etc.

⇒ reduce to 2 or 3D for plotting → PCA helps do this

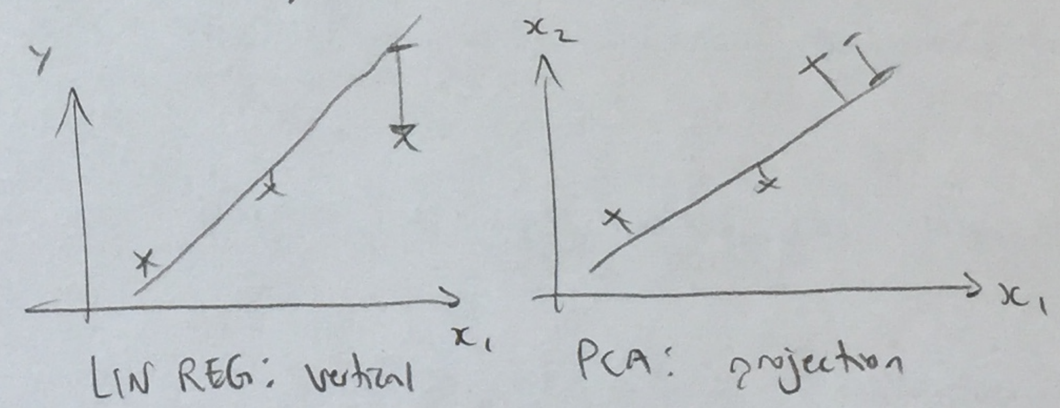
Principal Component Analysis: PCA



1) Mean normalization & feature scaling x_1, x_2
2) Minimize sum-of-squares of projection error
   ↳ find $\pm \underline{u}_{best}$ to project data

⇒ reducing from n-dim to k-dim: Find k-vectors $\underline{u}^{(1)}, \underline{u}^{(2)} \dots \underline{u}^{(k)}$
onto which to project data (eg. 3D→2D, find $\underline{u}^{(1)}$ & $\underline{u}^{(2)}$ to define plane)

⇒ PCA is NOT linear regression!
   ↳ no special feature "y" being predicted



LIN REG: vertical PCA: projection

⇒ Algorithm:

1) Pre-processing: mean norm / feature scaling: $\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$
   ↳ replace $x_j^{(i)}$ w/ $\frac{x_j - \mu_j}{s_j}$ ← standard dev of feature j

2) Reducing n-dim to k-dim:
   ↳ compute covariance matrix: $\Sigma = \frac{1}{m} \sum_{i=1}^{n} (x^{(i)})(x^{(i)})^T$ (Sigma)
   ↳ compute eigenvectors of Σ → [U, S, V] = svd(Sigma);
      ↳ singular value decomposition

=> $\underline{\underline{U}}$ matrix is $n \times n$, columns will be $\underline{u}^{(1)}, \underline{u}^{(2)}, \underline{u}^{(3)}$ etc.

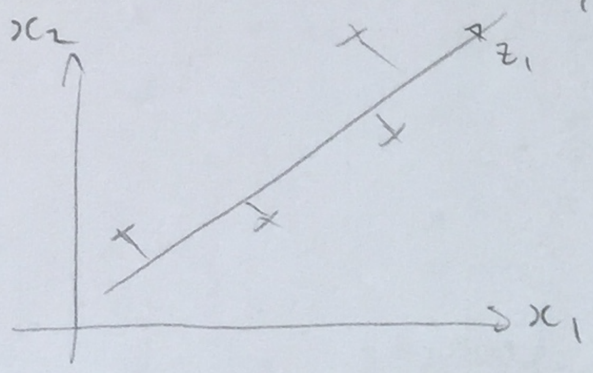
↳ take first k $u^{(}$ vectors!

3) $x \in \mathbb{R}^n \rightarrow z \in \mathbb{R}^k$:

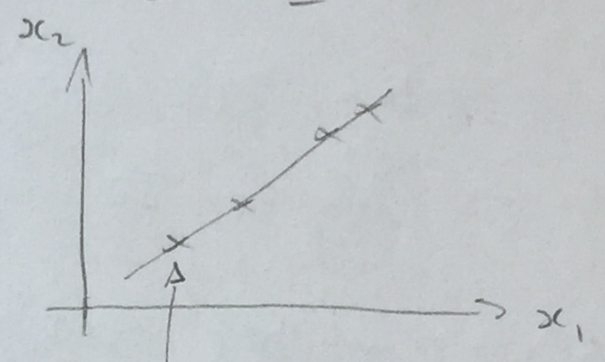
↳ $\underline{\underline{U}}_{reduced} = [n \times k] \rightarrow z = \underline{\underline{U}}_{red}^{T} x \in [k \times 1]$

$[k \times n]$ $[n \times 1]$

=> Reconstruction from Compressed Representation $\rightarrow \underline{z} \Rightarrow \underline{x}$!



=> can recover



$x_{approx}^{(i)} = \underline{\underline{U}}_{red} \underline{z}^{(i)} \in [n \times 1]$ approximate reconstruction of data

$[n \times k]$ $[k \times 1]$

=> Choosing # Principal Components: Choosing k

=> PCA minimizes $\frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - x_{approx}^{(i)} \|^2$ ← projection. Define variation $\frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} \|^2$

↳ choose smallest k such that $\frac{\text{avg. square proj error}}{\text{variation}} \leq 0.01$ (1%)

↳ "99% of variance is retained"

algorithm: PCA w/ $k=1$ -> $\underline{\underline{U}}_{red}$, $\underline{x}$ & $\underline{z}$ -> check variance retained

↳ increase k until variance constraint satisfied

=> helpful: for given k, $\frac{\text{avg sq. proj. error}}{\text{variation}} = 1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}}$ ← $\underline{\underline{S}}$ diagonal $\begin{bmatrix} S_{11} & 0 & 0 \\ 0 & S_{22} & 0 \\ 0 & 0 & \ddots \end{bmatrix}$

↳ don't need to find eigenvalues every time!

=> pick smallest k for $\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$

Advice for PCA:

⇒ Speeding up algorithms: say 10,000 features, $\underline{x} \in \mathbb{R}^{10,000}$

↳ extract inputs $x^{(1)} \rightarrow x^{(m)}$, apply PCA such that $z^{(1)} \rightarrow z^{(m)}$

↳ then pair $(z^{(1)}, y^{(1)}) \ldots (z^{(m)}, y^{(m)})$ → get $h_\theta(z) = \frac{1}{1 + e^{-\theta^T z}}$

⇒ take $x \rightarrow$ map to $z \rightarrow h_\theta(z)$

*Note: map $x^{(i)} \rightarrow z^{(i)}$ only on training set! Apply mapping to $\underline{x}_{cv}$, $\underline{x}_{test}$

⇒ For compression → retain 99% variance

For visualization → 2D or 3D

⇒ Misuse of PCA → avoiding overfitting (reducing # features)

↳ use regularization instead! PCA loses data/info w/o looking at y vals.

⇒ Always implement model w/o PCA first (most info retained), then use PCA if memory or speed unsatisfactory