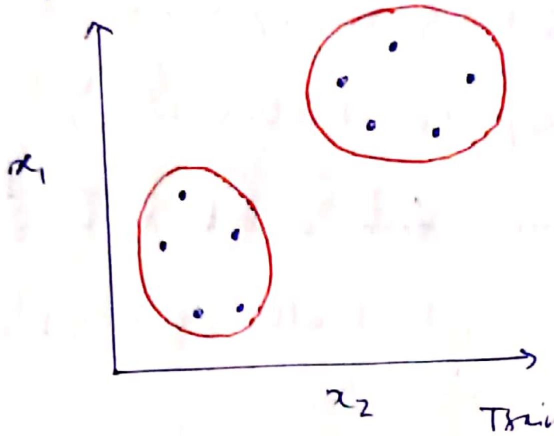
WEEK 8 - Unsupervised learning

clustering algorithms



Training set: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$
no ~~[illegible]~~ label $y^{(i)}$

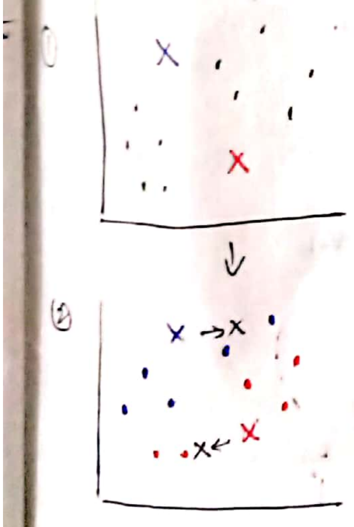
Application:

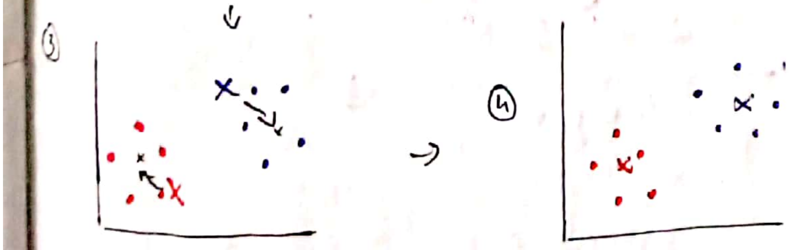
- Market segmentation
- Social network analysis
- Organize computing clusters
- Astronomical data analysis

K-Means algorithm

~~Rate~~ - Randomly initialize 2 points (cluster centroids)

- Two steps in iteration:
  - ○ Cluster assignment step
    Go through every example and see if its close to cluster centroid 1 or 2 and assign it to that group
  - ○ Move centroid step
    Move the centroids to the new centroid of each group 1 & 2





- Here further iteration doesn't change the centroid positions

Input :

- K (number of clusters)
- Training set $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

$x^{(i)} \in \mathbb{R}^n$ (drop $x_0 = 1$ convention)

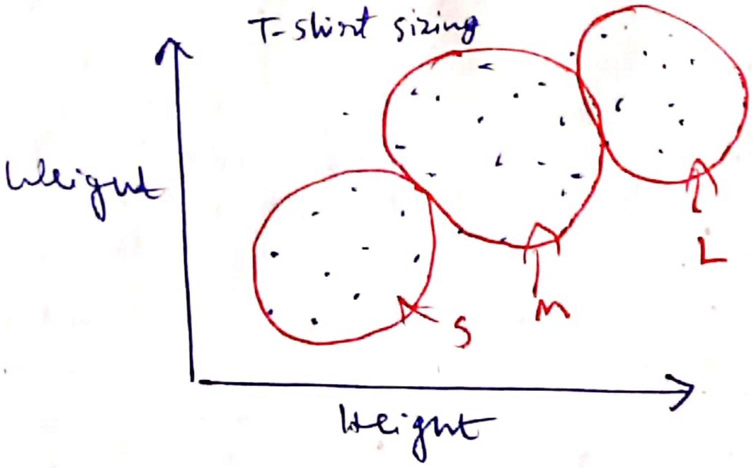
K-means algorithm: ~~(Also called distortion)~~

Randomly initialize K cluster centroids $\mu_1, \mu_2, \dots \mu_K \in$

```
Repeat {
Cluster Assignment Step
    for i = 1 to m
        $c^{(i)}$ := index (from 1 to K) of cluster
                 centroid closest to $x^{(i)}$
                 $= \min_k \|x^{(i)} - \mu_k\|^2$
Move centroid
    for k = 1 to K
        $\mu_k$ := average (mean) of points
                 assigned to cluster k
    Exa: $\mu_2 = \frac{1}{4}[x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}]$ if $c^{(1)}, c^{(5)}, c^{(6)}, c^{(10)} = 2$
}
```

Not a nested loop
for {
}
for {
}

K mean for non-seperated claner



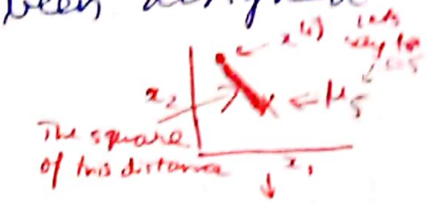
Scanned with CamScanner

K-means optimisation objective

$c^{(i)}$ = index of cluster $(1, 2, \ldots, K)$ to which example $x^{(i)}$ is currently assigned.

μ_K = cluster centroid K $(\mu_K \in \mathbb{R}^n)$

$\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned



Optimisation objective

$$J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K) = \frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

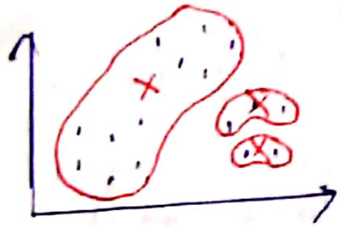
distortion function

$$\min_{\substack{c^{(1)}, \ldots, c^{(m)},\\ \mu_1, \ldots, \mu_K}} J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K)$$

Random initialization

- Should have $K < m$
- Randomly pick K training examples
- & set $\mu_1, \ldots, \mu_K$ equal to these k examples

Problem: Sometimes when we ~~[illegible]~~ randomly initialse, cluster centroids, it can result in local optimas



Solution: ~~[illegible]~~ ~~Initialise~~ Run the algorithm on different values of randomly initialised cluster centroids and pick the best

Running it 100 times

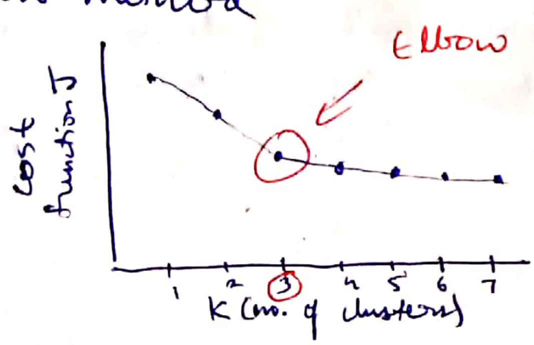
```
For i=1 to 100 {
    → Randomly initalise k-means
    → Run k-means, get $c^{(1)}, \dots c^{(m)}, \mu_1, \dots, \mu_K$
    → Compute cost function (distortion)
        $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$
}
```

Pick clustering that gave lowest cost $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$

Note, this is only good for problems where k = 2-10. For stuff like k=100, the first trial itself will give a good result

Choosing number of clusters

- Common way is to visualise the data and manually decide
- Ellbow method - called elbow cus the ~~tee~~ graph looks like a hand & we choose the elbow.



∴ k = 3

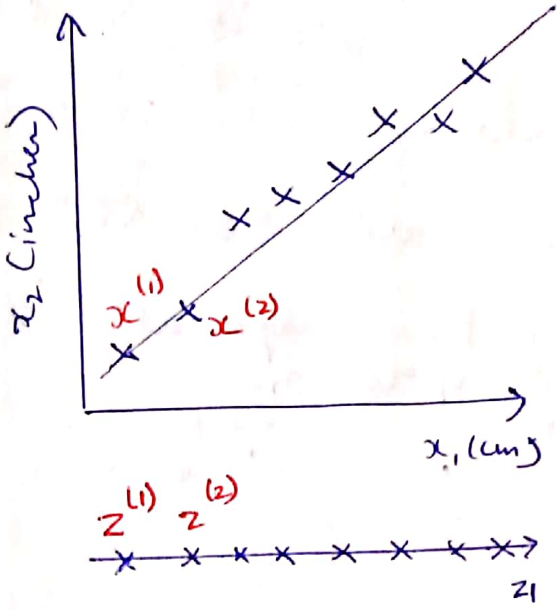
Note: Most times we can't make out the elbow so we don't use this much

- Depending on problem.
  Idea: If we ~~[illegible]~~ need to catagorise t-shirt sizes into S, M, L. Then we have k=3

Data compression

Sometimes we have redundant features that we can compress into 1 feature.

Ex:



Reduce data from 2D to 1D

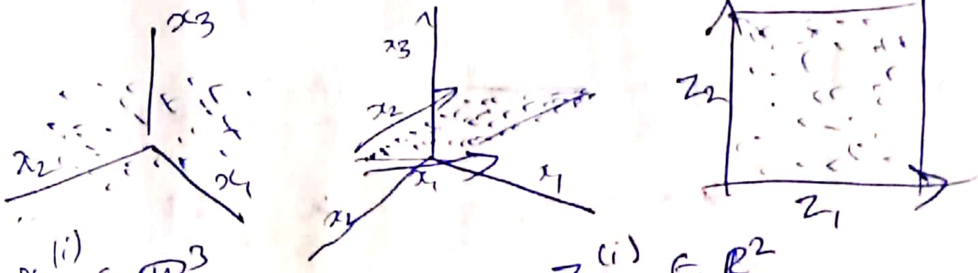
$x^{(1)} \in \mathbb{R}^2 \rightarrow z^{(1)} \in \mathbb{R}$

$x^{(2)} \in \mathbb{R}^2 \rightarrow z^{(2)} \in \mathbb{R}$

$\vdots$

$x^{(m)} \in \mathbb{R}^2 \rightarrow z^{(m)} \in \mathbb{R}$

In 3D to 2D also similar



$x^{(i)} \in \mathbb{R}^3$

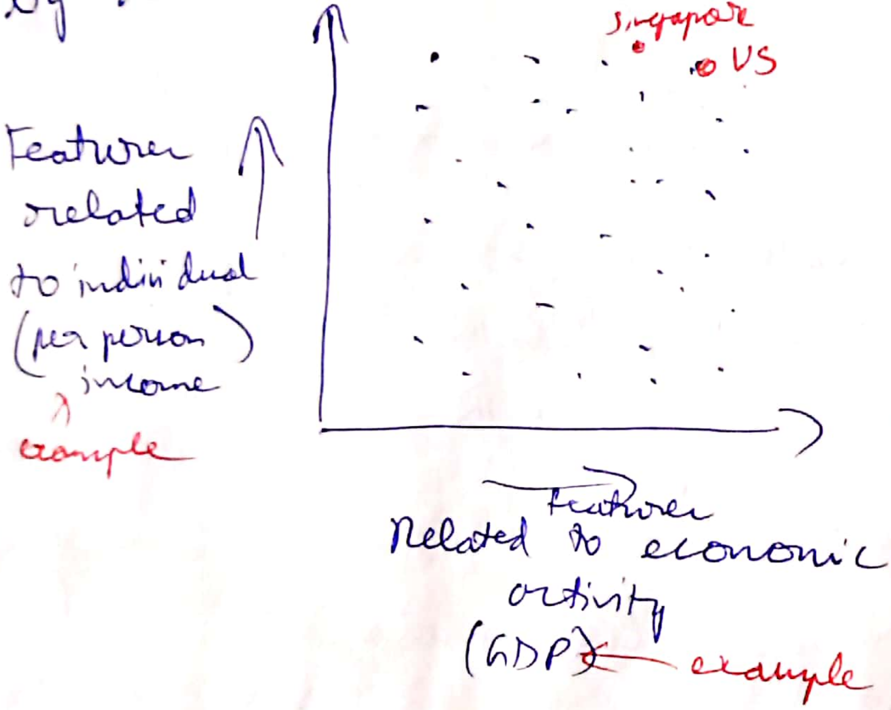
$z^{(i)} \in \mathbb{R}^2$

where $z^{(i)} = \begin{bmatrix} z_1^{(i)} \\ z_2^{(i)} \end{bmatrix}$

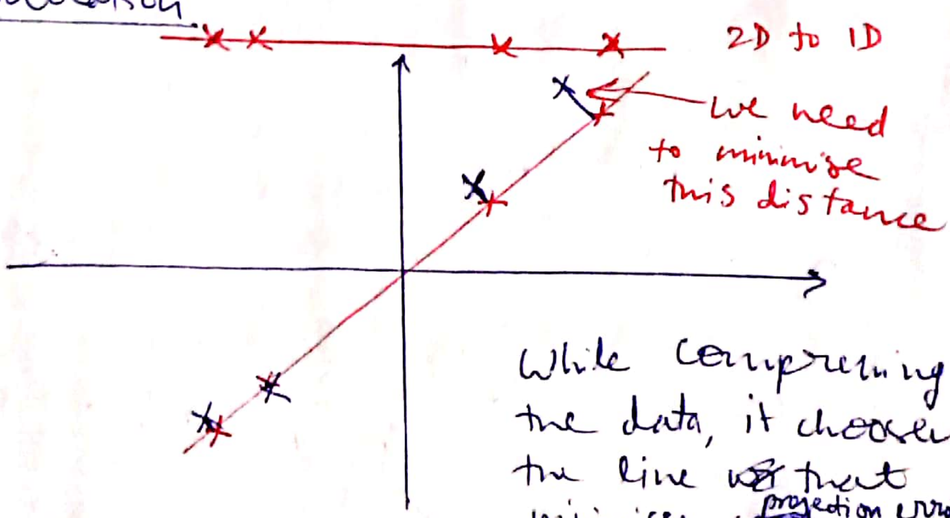
Data visualisation

Consider a dataset of 50 features ~~information~~ (GDP, per person income, poverty, etc) of countries of the world. So how do you visualise for $x \in \mathbb{R}^{50}$.

You reduce data from 50D to 2D by selecting two imporant features by data compression



Principal Component Analysis (PCA) problem formulation

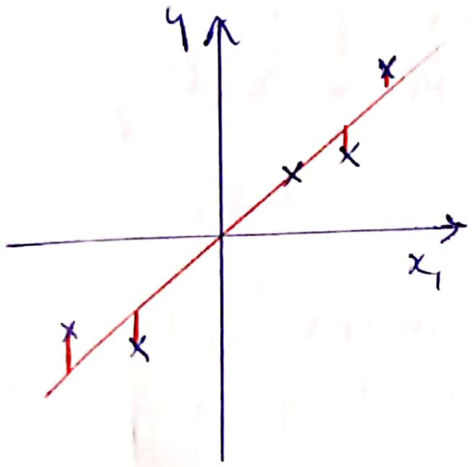


While compressing the data, it chooses the line ~~wrt~~ that minimises projection error (distance between actual & projected points)

Reduce from 2D to 1D: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimise the projection error..

Reduce from n-dimension to k-dimension: Find k vectors $u^{(1)}, u^{(2)}, \dots, u^{(k)}$ onto which to project the data, so as to minimise the projection error

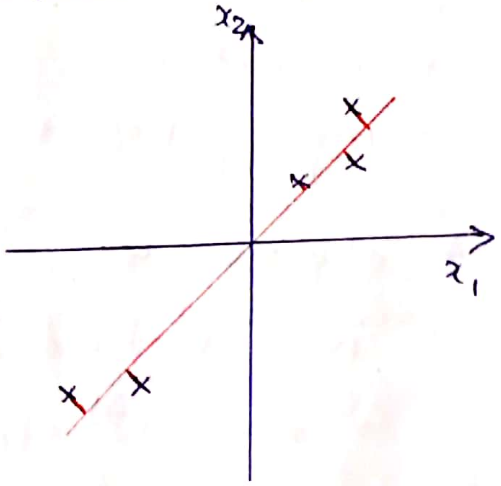
PCA is not linear regression



Linear regression

The lines are vertical drop from the point to the line

We are predicting y



PCA

The lines are shortest distance between the point and the line

We are reducing dimensionality

Data preprocessing

Training set: $x^{(1)}, x^{(2)}, \dots, x^{(m)}$

Preprocessing (feature scaling & mean normalisation)

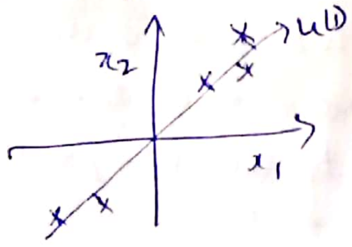
- $\underset{\text{(Mean) Average of all } x_j}{\mu_j} = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$ – Mean normalisation

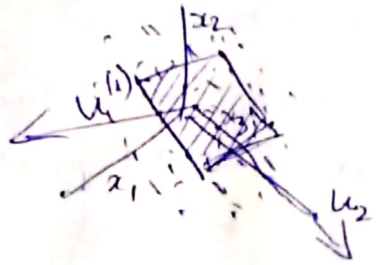
  Replace each $x_j^{(i)}$ with $x_j - \mu_j$

- If different features on different scales (e.g. x_1 = size of house, x_2 = no. of bedrooms) scale features to have comparable range of values. – feature scaling

PCA algorithm

We need to find a vector $u^{(1)}$ (2D → 1D) or $u^{(1)}$ and $u^{(2)}$ (3D → 2D) that fits the graph well.





Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix"

$$\underset{\text{sigma}}{\Sigma} = \frac{1}{m} \sum_{i=1}^{n} \underbrace{\underset{n\times 1}{(x^{(i)})} \underset{1\times n}{(x^{(i)})^T}}_{n\times n \text{ matrix}}$$

compute eigenvectors of matrix Σ:

$[U, S, V] = \text{svd}(\text{sigma});$ ← singular value decomposition ← can even use eig(sigma)

$$\underset{n\times n \text{ matrix}}{U} = \begin{bmatrix} | & | & | & & | \\ u^{(1)} & u^{(2)} & u^{(3)} & \dots & u^{(n)} \\ | & | & | & & | \end{bmatrix}$$

$$U = \underbrace{\begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(n)} \\ | & | & & | \end{bmatrix}}_{k} \in \mathbb{R}^{n \times n}$$

$U_{reduce} \downarrow$

$$\begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times k}$$

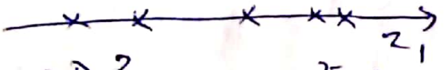
We need to $x \in \mathbb{R}^n \rightarrow z \in \mathbb{R}^k$

$$\therefore Z = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & & | \end{bmatrix}^T x = \underbrace{\underbrace{\begin{bmatrix} -(u^{(1)})^T- \\ \vdots \\ -(u^{(k)})^T- \end{bmatrix}}_{k \times n} \underset{n \times 1}{\overset{\uparrow}{x}}}_{k \times 1}$$

$\therefore$ Algorithm summary

- → Preprocessing (mean normalization & feature scaling)
- → $\text{Sigma} = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)})(x^{(i)})^T$ ← In vector form $\text{Sigma} = (1/m) * X' * X$; given $X = \begin{bmatrix} — x^{(1)T} — \\ — x^{(2)T} — \\ \vdots \\ — x^{(m)T} — \end{bmatrix}$
- → $[U, S, V] = \text{svd}(\text{Sigma})$;
- → $U_{reduce} = U(:, 1:k)$;
- → $z = U_{reduce}' * x$;

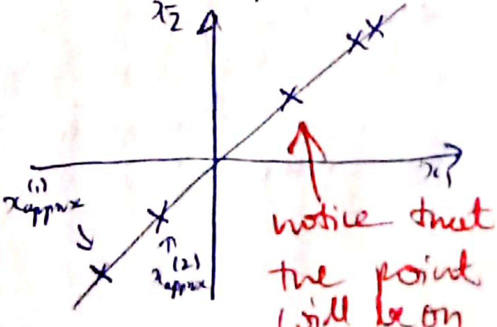
Reconstruction from compressed representation (1D to 2D)

If we have  (1D)

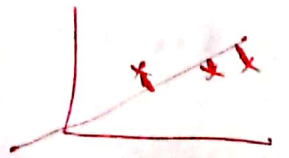
how to get back 2D?

$$\boxed{X_{approx} = U_{reduce} \cdot Z}$$

Since $Z = U_{reduce}^T x$



notice that the point will be on the line in the approximation unlike ←



Choosing k (no. of principal components)

Avg. squared projection error: $\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x^{(i)}_{approx}\|^2$

Total variation in the data: $\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2$

Typically choose k to be smallest value so that

$$\frac{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x^{(i)}_{approx}\|^2}{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2} \leq 0.01$$

→ For this example we say "99% of variance is retained"

For 0.05 → 95%

For 0.1 → 90% and so on

Algorithm

k=2, k,=3, k=4..

→ try PCA with k=1

→ Compute Ureduce, $z^{(1)}, z^{(2)}, \ldots, z^{(m)}, x^{(1)}_{approx}, \ldots, x^{(m)}_{approx}$

→ Check if $$\frac{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x^{(i)}_{approx}\|^2}{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2} \leq 0.01$$

→ Repeat for another k value

till it is ≤0.01

However the simpler method is to use

[U, S, V] = svd (sigma)

Here $S = \begin{bmatrix} S_{11} & & \cdots & 0 \\ & S_{22} & & \\ 0 & & S_{33} & \\ & & & S_{nn} \end{bmatrix}_{n \times n}$

$\frac{\Sigma \text{selected } S_{ii} \text{ values}}{\Sigma \text{All } S_{ii} \text{ Values}}$

For given k,

$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.01 \quad \text{or} \quad \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$$

Exa for k=3

Applying PCA

We can speed up supervised learning by reducing the no. of features without affecting the accuracy of the classification.

Note: We should only define mapping (learning Ureduce) $x^{(i)} \to z^{(i)}$ by running PCA only on the training set. Later this mapping can be applied to the examples $x_{cv}^{(i)}$ and $x_{test}^{(i)}$ in the cross validation & test sets.

Application:

- Compression:
  - Reduce memory/disk needed to store data
  - Speed up learning algorithm

Choose k by % of variance retained

- Visualisation:
  - Reduce dimensionality to k=2 or k=3 so it can be visualized

Bad use of PCA: To prevent overfitting

Since reducing features from n to k (k<n) results in overfitting happening less likely (cuz of less features), pple use it. But don't use it, use regularization instead. (cuz it throws away some information because it doesn't consider y labels)

Note: While designing an ML system, pple first apply PCA on the data and then ~~test~~ train logistic regression and test it. Don't do this. Run it on the raw data, later if it doesn't do what you want, then try implementing PCA and consider using $z^{(i)}$ instead of $x^{(i)}$