Clustering:

Unsupervised Learning → Unlabeled training set

Good for:

- → Market segmentation
- → Social Network analysis
- → Organizing computer clusters
- → Astronomical data analysis

K-means Algorithm is the most popular algo.

Pseudo Code:

Randomly initialize K cluster centroids $\mu_1, \ldots \mu_K$

```
Repeat {
    for i=1 to m :
        c(i) = index (1 to K) of cluster centroid
               closest to x(i)
    for k=1 to K :
        μ_k = average (mean) of points assigned to
              cluster k
}
```

step1 → first for loop → $c(i) = \text{argmin} \| x^{(i)} - \mu_k \|^2$

step2 → Second for loop → $\mu_k = \frac{1}{n} [x^{(k_1)} + x^{(k_2)} + \ldots + x^{(k_n)}]$

In two steps, cost function is minimized.

Sometimes K-means can get stuck in local optima. To get rid of it we need random initialization

For i = 1 to 100:

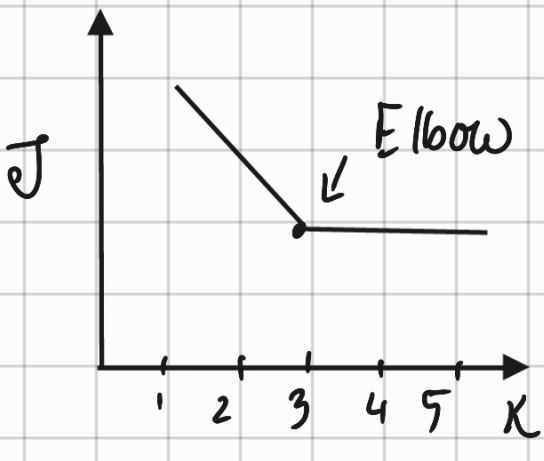
randomly initialize K-means

run K-means to get 'c' and 'μ'

compute the cost function J(c, μ)

pick the clustering that give us the lowest cost

We can choose the number of cluster using "Elbow" method



But sometimes curve has no elbow, than we'll find the later purpose like tshirt sizing → K → 3 or 5

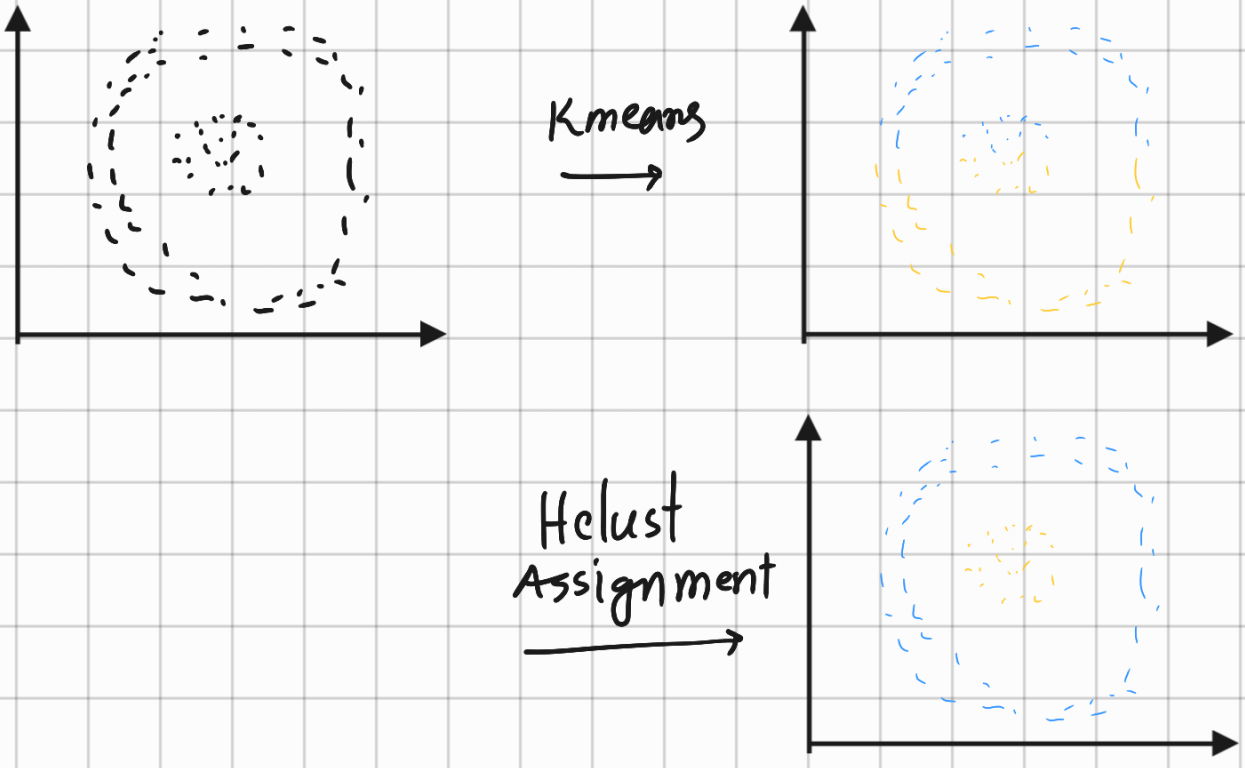
J will always decrease as K is increased unless it got stuck at bad local optima.

However, K means fails in some cases.

→ It always makes all cluster spherical

→ All axes have the same distribution and thus variance

One example:



Data Compression:

→ Reduce dimension of our features [3d to 2d]

→ Easy to visualize

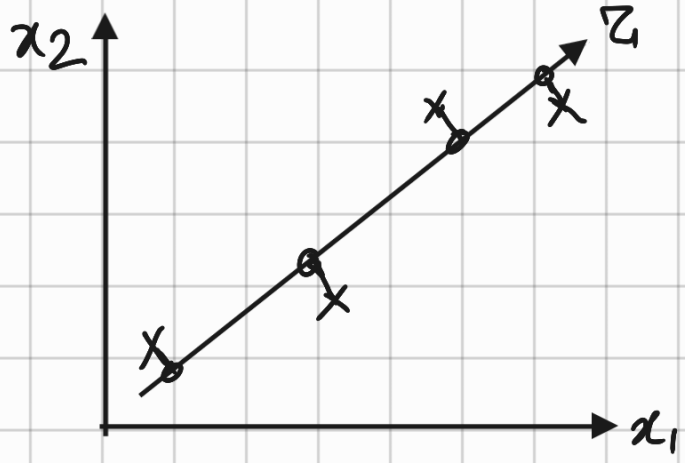
→ Summarize all other features

50D → 2D

Principle Component Analysis (PCA)

The goal of PCA is to reduce the average of all the distances of every feature to the projection line.

In PCA, we are minimizing the shortest distance or shortest orthogonal distances, to our data point



$$X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$$

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

Algorithm:

Data Preprocessing:

→ Given dataset $x_1, x_2 \ldots x_m$

→ Mean normalization $\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$

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

→ Feature scaling, $x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{s_j}$

→ Reduce 2d to 1

Sigma $(\Sigma) = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)}) (x^{(i)})^T$

↓

Covariance Matrix

Sigma = $\frac{1}{m} (X' * X)$;

→ eigenvactor of covar matrix

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

→ Take first k column of U matrix

Ureduce = U (:, 1:k);

→ Finally, $z = X * Ureduce$

Reconstruction from compressed representation

$$X_{approx} = Z * U_{reduce}^{T} \qquad [\text{as} \quad U^{T} = U^{-1}]$$

Choosing # of PCA:

Choose k to the smallest value such that,

$$\frac{\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} - x_{approx}^{(i)} \|^2}{\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} \|^2} \leq 0.01 \longrightarrow \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{m} S_{ii}}$$

99% of the variance is retained.

Algo:

S = diagonal mat

$[U, S, V] = svd(Sigma)$

iterate $k = 1:\infty$,

if $\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{m} S_{ii}} \geq 0.99$, break.

Application of PCA:

→ speed up supervised learning

→ Data compression [mostly used in CV]

→ Visualization of data

100 x 100 pixel imge
$= x \in \mathbb{R}^{10000}$

Bad use of PCA

→ prevent overfitting due of huge # of features.

↳ Using regularization is more effective in that case

"First use Full Machine Learning Algorithm without PCA, if needed then use PCA."