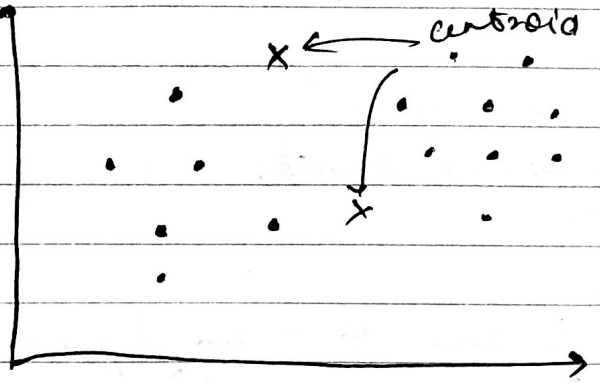
K-means Algorithm

- Randomly assign 2 points (clustering centroids)



Step 1 :- cluster assignment
Step 2 :- Move centroid.
(Move centroid to mean of data)

i/p - No. of clusters (K)
Training set. $\{x^{(1)}, x^{(2)}, \ldots x^{(m)}\}$

Algo.
- Randomly initialize K cluster centroids $(\mu_1, \mu_2 \ldots \mu_K)$
- Repeat {

minimize w.r.t $c^{(1)}, c^{(2)} \ldots$ (Assignment) { for $i = 1$ to m
$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$ $\left(\min \|x^{(i)} - \mu_K\| = c^{(i)}\right)$

minimize w.r.t $\mu_1, \mu_2 \ldots$ (Move centroid) { for $k = 1$ to K
μ_K := average (mean) of points assigned to k

if a cluster has no points, eliminate it.

What if there are not well separated clusters.

Optimization objective.

$c^{(i)}$ → index of cluster
μ_k → cluster centroid k.
$\mu_{c(i)}$ = cluster centroid of cluster which example $x^{(i)}$ has been assigned

$x^{(i)} \rightarrow 5$
$c^{(i)} \rightarrow 5$
$\mu_{c^{(i)}} = \mu_5$; that means $x^{(i)}$ is assigned of cluster

Optimization objective:

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

Find c & μ for min J.

RANDOM INITIALIZATION

How to randomly initialize cluster centroids μ_1, μ_2, μ_K

→ should have $K < m$.
→ randomly pick K training examples
→ set $\mu_1, \dots \mu_K$ equal to these K examples

K means can end at local minima,

If you don't want to, try multiple initialization.

eg:-

Running k-means 100 times.

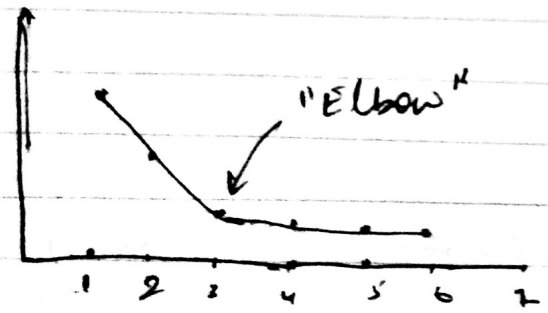
```
for i = 1 to 100 {
    Randomly initialize k means
    Run k means
    Get c^(1) & μ^(1)
    J(c^(1), ... c^(m), μ_1 ... μ_n)
}
```

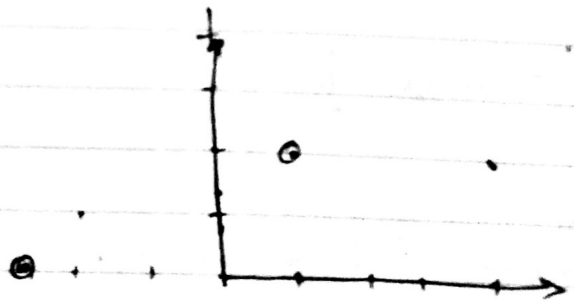
Pick min cost $J(c^{(1)}, \ldots c^{(m)}, \mu_1, \ldots, \mu_n)$

If k is small (2-10), random initialization will work definitely.

Choosing the No. of clusters.

Elbow method.





Not good method.

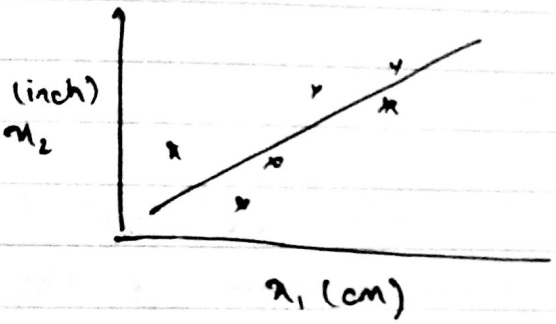
To chose no. of clusters - Human insights.

Dimensionality Reduction

★ Data compression

→ Maybe if you have a lot of features. And if you have a feature in length (cm) and a feature in inches. Then your dimensionality is reduced.

or



reduce dimension from 2D to 1D.

This makes algorithm runs quickly.

FOR 3D, if we project it on to a plane. And from x_1, x_2, x_3, you get z_1 & z_2.

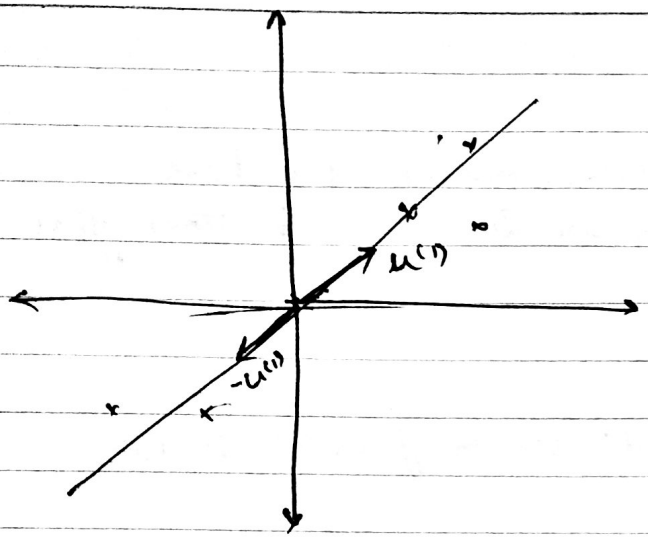
★ Data Visualization

How to plot 50 features (50-D) You could take out 2 most important features

PRINCIPLE COMPONENT ENVIRONMENT (for Dimensionality Reduction).

before pca, perform mean normalization & feature scaling. so that features have comparable range.

$u^{(1)} = \sqrt{(\frac{1}{\sqrt{2}})^2 + (\frac{1}{\sqrt{2}})^2}$



(shortest distance, maybe at any angle).
Minimizes squared error.

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

→ Reduce n-dim to k-dim : find k vectors $u^{(1)}, u^{(2)} \dots u^{(k)}$

Data processing.
★ Training set
★ Preprocessing (feature scaling/ mean normalization)

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

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

Reduce data from n-dimensions to k-dimensions.
↳ Compute "covariance matrix":

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

↳ compute "eigenvectors" of matrix Σ

→ $[U, S, V] = \text{svd}(\text{sigma})$ ... (Singular value decomposition)

you can use eig(sigma)

Sigma will be n x n Matrix.

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

$U = \begin{bmatrix} | & | & | & | \\ u^{(1)} & u^{(2)} & u^{(3)} & \cdots & u^{(n)} \\ | & | & | & | \end{bmatrix} \in \mathbb{R}^{n \times n}$

Take first k dimensions.

You will get Ureduce [] with k-dim

$Ureduce = U(:, 1:k);$

$z = Ureduce' * x;$

$z = Ureduce^T x$

$= \begin{bmatrix} — u^{(1)} — \\ — u^{(2)} — \\ — u^{(k)} — \end{bmatrix} x$

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

$= \left(\frac{1}{m}\right) * X' * X \quad \ldots \quad X = \begin{bmatrix} — x^{(1)T} — \\ \vdots \\ — x^{(m)T} — \end{bmatrix}$

Reconstruction from compressed representation

$x_{approx} = Ureduce * z$

$(n \times k) \quad (k \times 1)$

$(n \times 1)$

K = No. of principle components.

→ PCA minimizes avg. squared projection error.

$$\text{Squared projection error} = \frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} - x_{approx}^{(i)} \|^2$$

$$\text{Total variation} = \frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} \|^2$$

↑ distance from origin.

k we choose to be smallest value so that

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

(choose it.)

"99% of variance is retained"

Algo

Try PCA : k=1

Compute Ureduce, $z^{(1)}, z^{(2)}, \ldots z^{(m)}$

$x^{(1)}_{approx} \ldots x^{(m)}_{approx}$

check if

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

This is less efficient

$[U, S, V] = \text{svd}(\text{Sigma})$

$S = \begin{bmatrix} s_{11} & & & \\ & s_{22} & & \\ & & s_{33} & \\ & & & \ddots \\ & & & & s_{nn} \end{bmatrix}$

for given k

$$1 - \frac{\sum_{i=1}^{k} s_{ii}}{\sum_{i=1}^{n} s_{ii}} \leq 0.01$$

or $\dfrac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$

Supervised learning speedup

→ $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots (x^{(m)}, y^{(m)})$.

one important use is to use it in computer vision.
for 100 × 100 pixels

with PCA, our algo will run smoothly.

1). Extract inputs

Unlabeled datasets: $x^{(1)}, x^{(2)}, \ldots, x^{(m)} \in \mathbb{R}^{10000}$

↓ PCA

$z^{(1)}, z^{(2)}, \ldots z^{(m)} \in \mathbb{R}^{1000}$

2) New training set

$(z^{(1)}, y^{(1)}), (z^{(2)}, y^{(2)}), \ldots (z^{(m)}, y^{(m)})$

~~for cross validation~~

Mapping should be done only on training set. Run PCA only on training set. Apply mapping to cross validation & test set.

Application
- compression
- visualization.

Problem: It should not be use to reduce overfitting. Use regularization to reduce overfitting.