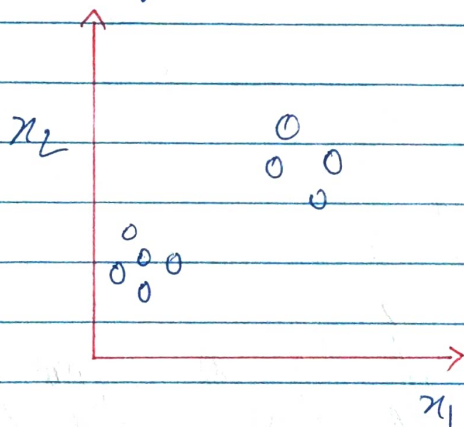


# Un Supervised Learning, recommen- -ders & reinforcement learning.

## ⇒ Clustering

↳ what is clustering?

→ It is an ~~an clustering~~ <sup>Unsupervised</sup> algorithm which finds the similarity between the data & groups them accordingly.



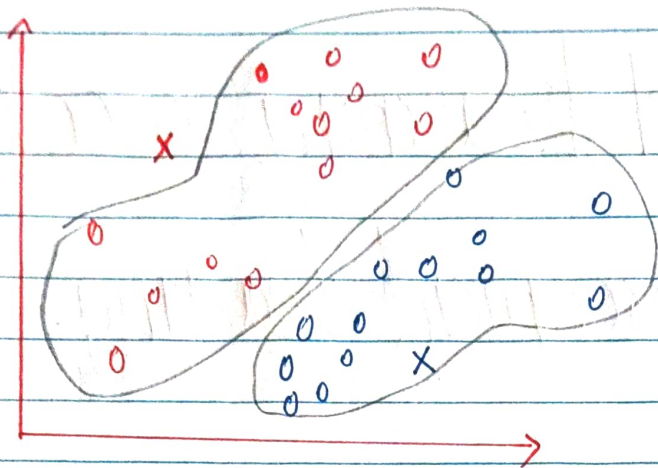
} its examples are like grouping similar news in the google news part & Astronomical data analysis

## { K-means Algorithm }

### Intuition

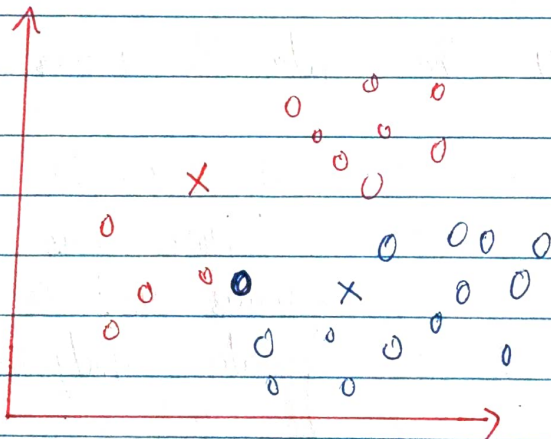
→ The first thing that the k-means algorithm does is it will take a random guess: at where might be the centres of two clusters that you might ask it to find.

& then it will go through each point's guessing whether it is closer to red cross or blue cross.



⇒ The other step that k-means does, it'll look at all the red points & take an average of them. & it

will move the red cross to whatever is the average location of the red dots. & it does that same for blue cross



Now we'll look through all of the 30 training examples again & check for every one of them, whether it's closer to the red or blue cluster centroid for the new locations & then we will associate them to the closer cluster.

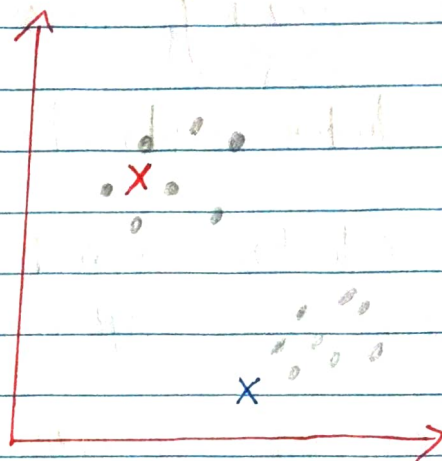
& repeat this process till you find that there are no changes to the colors of the points or location of the centroids, & this means the clustering algorithm has converged.



## Beyond intuition

Randomly initialize  $K$  cluster centroids  $\mu_1, \mu_2, \dots, \mu_K$   
{ here we are taking only two centroids  
red yellow & blue }

Here  $\mu_1$  &  $\mu_2$  are vectors with same dimensions as training examples of an example in 2d having 2 components



# Assign points to cluster centroids

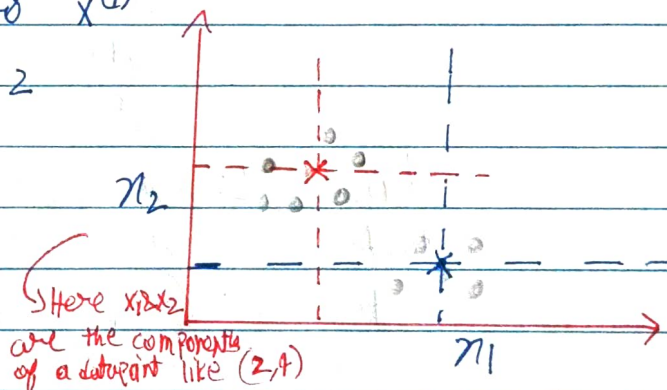
for  $i=1$  to  $m$

$c^{(i)} :=$  index (from 1 to  $K$ ) of cluster centroid closest to  $x^{(i)}$

$$\min_k \|x^{(i)} - \mu_k\|^2$$

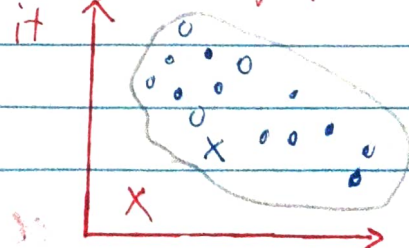
# Move cluster centroids

for  $k=1$  to  $K$



$\mu_k :=$  Average (mean) of points assigned to cluster  $k$

if there is a clustering centroid which hasn't been assigned any points then it's better to delete it  $\rightarrow (K=K-1)$ , also you reinitialize it hoping it gets some points.



## k-means optimization objective

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

$\mu_k$  = cluster centroid  $k$

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

ex  $\rightarrow$  Suppos we have 3 clusters ( $k=3$ ) & a few data points in 2D space

Points:  $(x^i)$

$\left. \begin{array}{l} x^{(1)} = (1, 2), \\ x^{(2)} = (2, 1), \\ x^{(3)} = (8, 8), \\ x^{(4)} = (9, 9), \\ x^{(5)} = (0, 0) \end{array} \right\} \begin{array}{l} \text{After one iteration of k-means,} \\ \text{the algorithm groups them into} \\ \text{clusters:} \end{array}$

Cluster 1:  $x^{(1)}, x^{(2)}, x^{(5)}$

Cluster 2:  $x^{(3)}, x^{(4)}$

Cluster 3: (empty for now, assume later it get points)

Step (1) : Compute centroids ( $\mu_k$ )

$\rightarrow \mu_1 = \text{mean of } (1, 2), (2, 1), (0, 0) = (1, 1)$

$\rightarrow \mu_2 = \text{mean of } (8, 8), (9, 9) = (8.5, 8.5)$

$\rightarrow \mu_3 = (\text{not defined yet since no points are assigned})$

So,

$\mu_1 = (1, 1), \mu_2 = (8.5, 8.5)$



Step 2: Assign each point a cluster index ( $c(i)$ )

$\rightarrow x^{(1)}$  belongs to cluster 1  $\rightarrow c^{(1)} = 1$

$x^{(2)}$  " " " " 2  $\rightarrow c^{(2)} = 2$

$x^{(3)}$  " " " " 2  $\rightarrow c^{(3)} = 2$

$x^{(4)}$  " " " " 2  $\rightarrow c^{(4)} = 2$

$x^{(5)}$  " " " " 1  $\rightarrow c^{(5)} = 1$

Step 3: Interpret  $\mu_{c(i)}$

This happens means the centroid of the cluster to which point  $x(i)$  is assigned

$\rightarrow$  for  $x^{(1)} = (1, 2)$ ,  $c^{(1)} = 1$ ,

so  $\mu_{c^{(1)}} = \mu_1 = (1, 2)$ .

$\rightarrow$  for  $x^{(3)} = (8, 8)$ ,  $c^{(3)} = 2$ , so

$\mu_{c^{(3)}} = \mu_2 = (8.5, 8.5)$

Cost function for k-means also known as "distortion function".

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

The cost function good for k-means is the average squared distance between every training example  $x^{(i)}$  & the location of the cluster centroid to which the training example  $x^{(i)}$  has been assigned.

in the process of converging the cost function must never go up, if it does that it means there's a bug in your code.

& if the cost function isn't reducing it means there is a bug in your code. & staying the same it means you have converged.

# Initializing k-means

## Random initialization

always choose  $k < m$ ,

$k \rightarrow$  no of cluster centroids

$m \rightarrow$  training examples

~~Randomly pick "k" few training examples & assign them as k~~

Randomly pick "k" training examples.

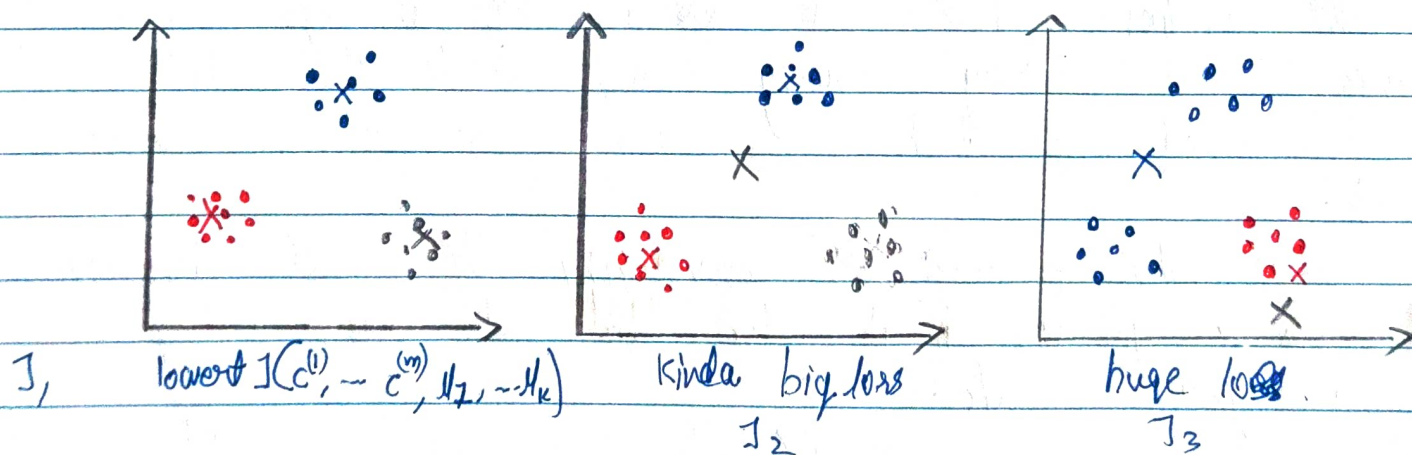
set  $\mu_1, \mu_2, \dots, \mu_k$  equal to these k examples

$\rightarrow$  but if the initialization is different then the algo will form totally different clusters,

So how

are we gonna do it?

we will initialize multiple times & calculate cost functions & whichever initialization has the lowest cost or loss we will pick that.





in the algorithmic way you can write.

```
for i = 1 to 100 {
```

Randomly initialize k-means.

Run k-means. Get  $c^{(1)}, \dots, c^{(m)}, \mu_1, \mu_1, \dots, \mu_k$

Compute cost function (distortion)  
 $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \mu_1, \dots, \mu_k)$

```
}
```

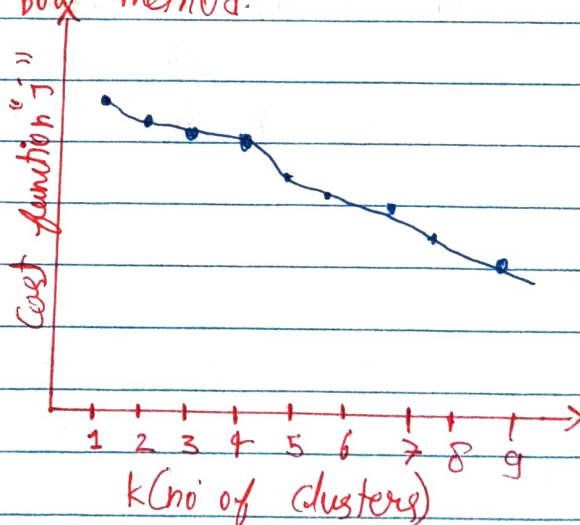
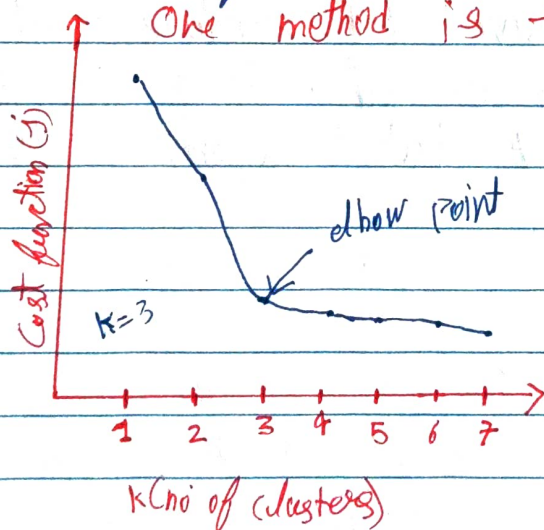
we usually initialize between like 50-1000 times because after 1000 it's get computationally heavy. & we pick one with the lowest loss "J".

## Choosing the number of clusters to have.

The k-means algorithm requires "k" as one of its inputs  
 $k \rightarrow$  The number of clusters you want to find

but how do you decide how many cluster to use.

One method is  $\rightarrow$  Elbow method.



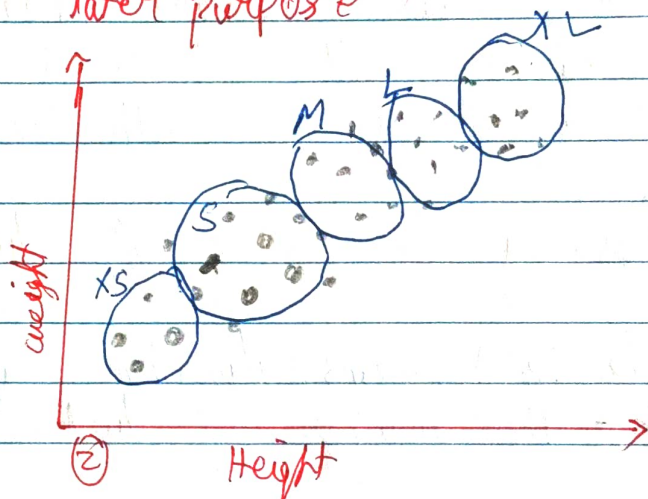
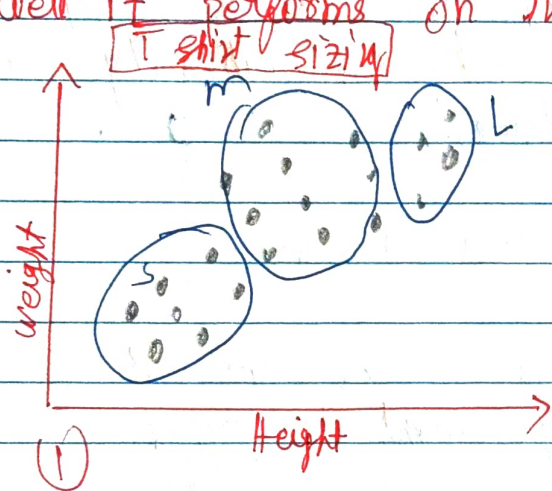
In the elbow method, Computes the cost function while increasing the no of clusters & when you find the elbow point, the point from where suddenly the cost function starts

decreasing slowly. then you stop & call the number of clusters,

But it doesn't work if the loss decreases gradually & there is no elbow point.

{ So it's not recommended }

→ often, you want to get clusters for some later (downstream) purpose. Evaluate k-means based on how well it performs on that later purpose



→ Now you have to do trade off's

in (1) it's ~~more~~ <sup>less</sup> expensive to produce but the fit gets compromised a little bit

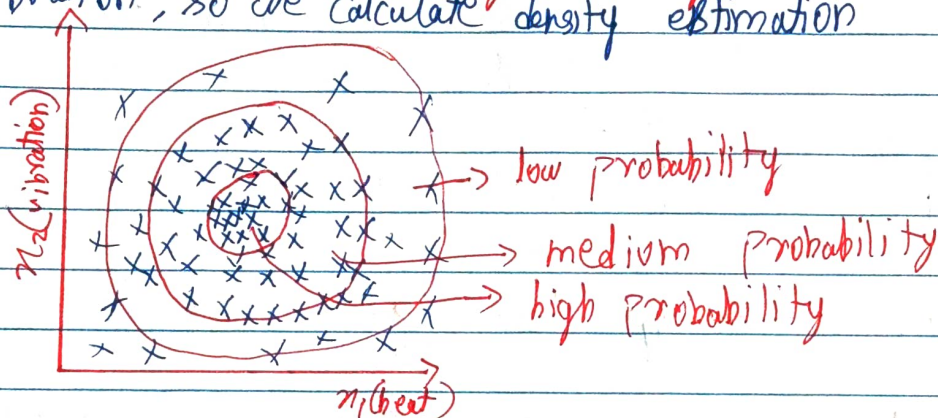
(2) it's better fit but expensive to produce



# Anomaly detection

⇒ It is a way to detect if an example lets say " $x_{test}$ " is anomalous.

ex → Let's say we have a dataset:  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$  there are ~~all~~ examples of an engine, & they have two parts heat & vibration, so we calculate density estimation



model  $P(x)$  → Probability of  $x$

$$\text{if } P(x_{test}) < \epsilon$$

↳  $\epsilon$ : epsilon: a small number

then it's an anomaly

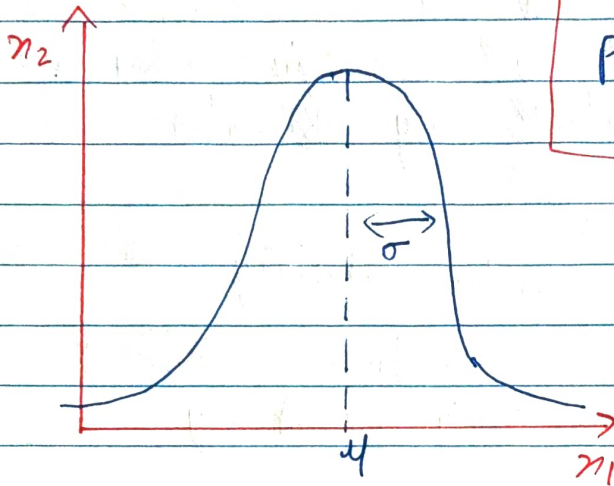
$$\text{if } P(x_{test}) \geq \epsilon$$

then it's not an anomaly

# Gaussian (Normal) distribution

Let  $x$  is a number.

probability of  $x$  is determined by a gaussian with mean  $\mu$ ,  
variance  $\sigma^2$ .



$$P(x) = \frac{1}{\sqrt{2\pi} \sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

take a dataset :  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ , now calculate mean  
 $\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$ , & also variance,  $\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)^2$

& now you can plot gaussian distribution,

for anomaly detection  
take a test example & use the formula for  $P(x)$  above  
to find the value,

if  $P(x)$  is high, the point is close to the mean & "normal".  
if  $P(x)$  is very small, the point is far from mean.  
or just don't do it just compare it to the mean  
& if it's too far from mean call it an anomaly



## Anomaly detection algorithm

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

Each example  $\vec{x}^{(i)}$  has  $n$  features,  $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$

$$P(\vec{x}) = P(x_1; \mu_1, \sigma_1^2) * P(x_2; \mu_2, \sigma_2^2) * \dots * P(x_n; \mu_n, \sigma_n^2)$$

or

$$P(\vec{x}) = \prod_{j=1}^n P(x_j; \mu_j, \sigma_j^2)$$

### ⇒ STEPS

① Choose  $n$  features  $x_i$  that you think might be indicative of anomalous examples.

② Fit Parameter  $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}, \quad \sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$$

or in vectorized formula,

$$\vec{\mu} = \frac{1}{m} \sum_{i=1}^m \vec{x}^{(i)} \quad \vec{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{bmatrix}$$

③ Given a new example  $x$ , compute  $P(x)$ :

$$P(x) = \prod_{j=1}^n (P(x_j; \mu_j, \sigma_j^2)) = \prod_{j=1}^n \frac{1}{\sqrt{2\pi} \sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2 \sigma_j^2}\right)$$

anomaly if,  $P(x) < \epsilon$

# Developing & evaluating an anomaly detection system

→ while making an anomaly detection model it is easier to add or remove different features or increase or decrease the value of ' $\epsilon$ ' epsilon if we have a way to evaluate the model.

To do this we do "rel-number evaluation"

Assume we have some labelled data, of anomalous & non-anomalous examples.  
 $y=0$   $y=1$

Training set:  $x^{(1)}, x^{(2)}, \dots, x^{(m)}$  (assume normal examples/not anomalous)  
 $y=0$  for all training examples.

Cross validation set:  $(x_{cv}^{(1)}, y_{cv}^{(1)}), \dots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$  include a few

Test set:  $((x_{test}^{(1)}, y_{test}^{(1)}), \dots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})}))$  anomalous examples, mostly normal examples

Use Cross validation set to tune " $\epsilon$ " or to test what features to add or remove, & we use test set to actually test the model using precision & recall because the data is skewed, There are not equal amount of bad & good engines, it's primarily an unsupervised algorithm because in the training process there is no labelled data

if you have very few anomalies you can skip test set & have only cross validation set.



## Anomaly detection

vs

## Supervised learning

Very small no of +ve examples ( $y=1$ ). (0-20 is common) Large number of -ve ( $y=0$ ) examples

Large no of +ve & -ve examples

Many different "types" of anomalies. hard for any algorithm to learn from +ve examples what the anomalies look like; future anomalies may look nothing like any of the anomalous example we've seen so far.

Enough +ve examples for algorithm to get a sense of what +ve examples are like, future +ve examples like to be similar to ones in training set.

→ ex → Manufacturing - find new previously unseen defects in manufacturing (e.g. aircraft engine)

ex → finding known previously seen defects

## What features to use

So, in your dataset you will have not always have gaussian distribution - you will have to modify non gaussian features into gaussian

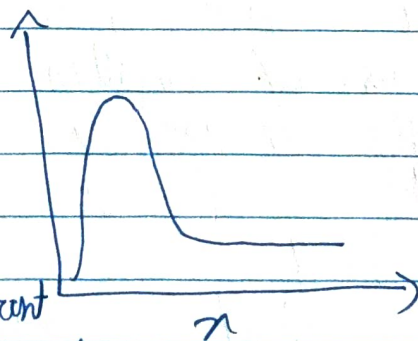
for example you can modify  $x$  by,

$$x \rightarrow \log(x)$$

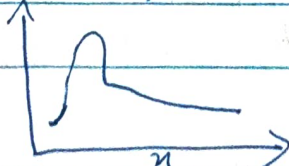
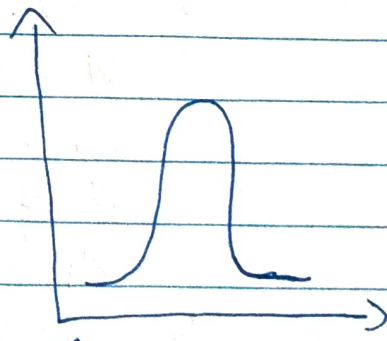
$$x \rightarrow \log(x+c) \quad \text{any constant}$$

$$x \rightarrow \sqrt{x}$$

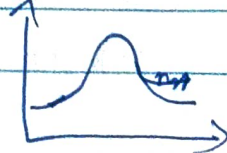
$$x \rightarrow x^{1/3}$$



$$\ln \log(x)$$

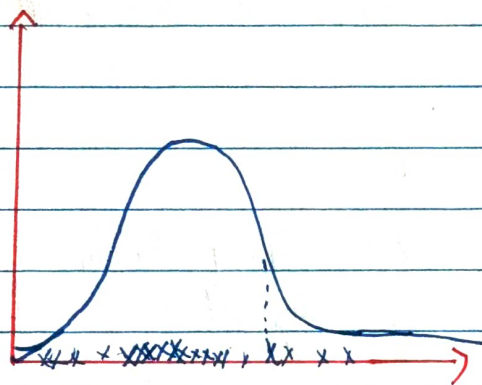


$$\sqrt{x}$$



Other than making sure that your data is approximately gaussian, after you have trained your anomaly detection algo, you can do "error analysis for anomaly detection" if it doesn't do well on anomaly cross-validation set

you might end up having a very common problem:-  
 $P(x)$  is comparable for normal & anomalous examples  
( $P(x)$  is large for both)



$x_1$  num transactions }  $\rightarrow$  (Just taking a random example)

in this case you choose a feature that help you differentiate between them, choose features that might take on unusually large or small values in the event of an anomaly,

ex  $\rightarrow$  monitoring computers in a data center.

$x_1$  = memory use of computer

$x_2$  = no of disk accesses/sec

$x_3$  = CPU Load

$x_4$  = network traffic

in data centers high CPU load & high network traffic is common but what's not common is high CPU Load



but low network traffic, in this case you need to modify the current features so that they can be ~~helpful~~ helpful, ex  $\rightarrow$  divide  $x_3$  by  $x_4$  this will give you an idea of CPU load per traffic, & this new feature will help you detect this anomaly.

Say 
$$x_5 = \frac{x_3}{x_4} = \frac{\text{CPU Load}}{\text{network traffic}}$$

# Recommender Systems

↳ An <sup>unsupervised</sup> machine learning algorithm that helps you recommending products & movies etc..

Ex ⇒ Predicting movie ratings

↳ User rates movies using zero to five stars.

| Movie                | Alice (1) | Bob (2) | Carol (3) | Dave (4) |
|----------------------|-----------|---------|-----------|----------|
| Love at last         | 5         | 5       | 0         | 0        |
| Romance forever      | 5         | ?       | ?         | 0        |
| Cute puppies of love | ?         | 4       | 0         | ?        |
| Nonstop car chases   | 0         | 0       | 5         | 4        |
| Swords vs karate     | 0         | 0       | 5         | ?        |

? → the question mark denotes that the user hasn't rated the movie

$n_u$  → no. of users

$n_m$  → no. of movies

$r(i, j)$  → 1 if user "j" has rated movie "i"

$$\begin{cases} r(1, 1) = 1 \\ r(1, 3) = 0 \end{cases}$$

$y(i, j)$  → rating given by user "j" to movie "i" (defined only if  $r(i, j) = 1$ )

$$\nexists \{y(3, 2) = 4\}$$



## Using per-item features

→ Building a recommender system if we had features of each item, or features of each movie.

| Movie                | Alice (1) | Bob (2) | Carol (3) | Dave (4) | $x_1$<br>(Romance) | $x_2$<br>(action) |
|----------------------|-----------|---------|-----------|----------|--------------------|-------------------|
| Love at first        | 5         | 5       | 0         | 0        | 0.9                | 0                 |
| Romance forever      | 5         | ?       | ?         | 0        | 1.0                | 0.01              |
| 1000 Puppies of Love | ?         | 4       | 0         | ?        | 0.99               | 0                 |
| Non stop car chases  | 0         | 0       | 5         | 4        | 0.1                | 1.0               |
| Swords vs katana     | 0         | 0       | 5         | ?        | 0                  | 0.9               |

$x_1$  → tells how much it's an ~~action~~ <sup>Romance</sup> movie

$x_2$  → tells how much it's an action movie

$n$  → no of features = 2

$$x^{(1)} = \begin{bmatrix} 0.9 \\ 0 \end{bmatrix} = \text{features of movie (1)}, \quad x^{(2)} = \begin{bmatrix} 1.0 \\ 0.01 \end{bmatrix} = \text{features of movie (2)}$$

for user 1: Predict rating for movie "i" as:  $w \cdot x^{(i)} + b$  } Just linear regression

$$\text{Say, } \left. \begin{aligned} w^{(1)} &= \begin{bmatrix} 5 \\ 0 \end{bmatrix}, \quad b^{(1)} = [0] \\ x^{(3)} &= \begin{bmatrix} 0.99 \\ 0 \end{bmatrix} \end{aligned} \right\} w^{(1)} \cdot x^{(3)} + b^{(1)} = 4.95$$

for user j: Predict rating's given by user "j" for movie "i" as

$$w^{(j)} \cdot x^{(i)} + b^{(j)}$$

## Cost function

Notation :

$r(i, j) = 1$  if user  $j$  has rated movie  $i$  (0 otherwise)

$y(i, j) =$  rating by user  $j$  to movie  $i$  (if defined)

$w^{(j)}, b^{(j)} =$  Parameters for user  $j$

$x^{(i)} =$  feature vector for movie ' $i$ '

For user  $j$  & movie  $i$ , predict rating:  $w^{(j)} \cdot x^{(i)} + b^{(j)}$

$m^{(j)} =$  no. of movies rated by user  $j$

To learn  $w^{(j)}, b^{(j)}$ ,

$$J(w^{(j)}, b^{(j)}) = \frac{1}{2m^{(j)}} \sum_{i: r(i, j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i, j)})^2 + \lambda \sum_{k=1}^n (w_k^{(j)})^2$$

↖ no. of features

& to find the optimal value for  $w$  &  $b$  just minimize the cost fn above.

for recommender's system it would be convenient to actually eliminate the division by  $m^{(j)}$  term from both cost function & regularization fn. it's a constant value so it doesn't matter.



To learn parameters for all users:

$w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \dots, w^{(n_u)}, b^{(n_u)}$

$$\frac{1}{2} \sum_{j=1}^{n_u} \sum_{i: r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{1}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2$$

$$\mathcal{J} \left( \begin{matrix} w^{(1)}, \dots, w^{(n_u)} \\ b^{(1)}, \dots, b^{(n_u)} \end{matrix} \right)$$

Now, you have the cost function & you can use gradient descent to minimize it & get a good set of weights & biases.

## Collaborative filtering algorithm

So you saw how if you have features for each movie, such as feature  $x_1$  &  $x_2$ , then you can use linear regression to learn to predict movie rating. But what if you don't have those,

features,  $x_1$  &  $x_2$ ? let's see how you can learn or come up with those features  $x_1$  &  $x_2$  from data.

| movie                | Alice(1) | Bob(2) | Carol(3) | Dave(4) | $x_1$<br>(romance) | $x_2$<br>(action) |
|----------------------|----------|--------|----------|---------|--------------------|-------------------|
| Love at last         | 5        | 5      | 0        | 0       | ?                  | ?                 |
| Romance forever      | 5        | ?      | ?        | 0       | ?                  | ?                 |
| Cute puppies of love | ?        | 4      | 0        | ?       | ?                  | ?                 |
| Non-stop car chases  | 0        | 0      | 5        | 4       | ?                  | ?                 |
| Swords vs. karate    | 0        | 0      | 5        | ?       | ?                  | ?                 |

let's say we somehow got the parameters,

$$\left. \begin{matrix} w^{(1)} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}, w^{(2)} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}, w^{(3)} = \begin{bmatrix} 0 \\ 5 \end{bmatrix}, w^{(4)} = \begin{bmatrix} 0 \\ 5 \end{bmatrix} \\ b^{(1)} = 0, b^{(2)} = 0, b^{(3)} = 0, b^{(4)} = 0 \end{matrix} \right\} \begin{matrix} \text{use this } \rightarrow w^{(j)} \cdot x^{(i)} + b^{(j)} \\ \text{to predict rating "i"} \\ \text{by user "j"} \end{matrix}$$

$x_1 \rightarrow$  feature (rating)

$x_2 \rightarrow$  feature vector of a movie

lets see how we can try to guess what might be reasonable features for movie one,

based on the parameters  $w^{(j)}$  &  $b^{(j)}$   
Using  $w^{(j)} \cdot x^{(j)} + b^{(j)} \rightarrow 0$  because all the values of  $b$  are zero

So  
for movie (1)

$$\left. \begin{aligned} w^{(1)} \cdot x^{(1)} &\approx 5 \\ w^{(2)} \cdot x^{(1)} &\approx 5 \\ w^{(3)} \cdot x^{(1)} &\approx 0 \\ w^{(4)} \cdot x^{(1)} &\approx 0 \end{aligned} \right\} \Rightarrow x^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$\Rightarrow$  given the value of "w" what choice of  $x^{(1)}$  would cause the values above,

if you have the parameters of all 4 users & if you have 4 ratings you are trying to match, you can take a reasonable guess at what is the feature vector,  $x^{(1)}$  for movie (1) that would make good prediction for these four ratings.

if you have the parameter vectors, you can also try to come up with a feature vector  $x^{(2)}$  for the second movie, feature vector  $x^{(3)}$  for the third movie & so on.

lets come up with a cost function for learning the values of  $x_1$  &  $x_2$ ,

this works <sup>only</sup> because we have parameters for four users, that's what allows us to try to guess features  $x^{(j)}$



## Cost function

given  $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \dots, w^{(n_u)}, b^{(n_u)}$

to learn  $x^{(i)} \rightarrow$  the features  $x^{(i)}$  for a specific movie  $(i)$

$$J(x^{(i)}) = \frac{1}{2} \sum_{j: r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^{n_i} (x_k^{(i)})^2$$

$\rightarrow$  no. of features

$\uparrow$  so if we define this as a cost fn of  $x^{(i)}$ , then if we minimize this as a fn of  $x^{(i)}$  you be choosing for the movie  $(i)$  features of movie "i".

therefore all the users  $j$  that

have rated movie  $i$ ,  $j: r(i,j)=1$ , we will try to minimize the squared difference between "what your choice of features  $x^{(i)}$  results in" in terms of movie rating,

$$(w^{(j)} \cdot x^{(i)} + b^{(j)}) - y^{(i,j)}$$

movie rating that the user had given it,  $y^{(i,j)}$  "minus the actual

& usually just add an Regularization term so that the model don't overfit

To learn  $x^{(1)}, x^{(2)}, \dots, x^{(n_m)}$ :

$$\frac{1}{2} \sum_{i=1}^{n_m} \sum_{j: r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

$j(x^{(1)}, x^{(2)}, \dots, x^{(n_m)})$ , if you have parameters

$w$  &  $b$  of all the users then minimizing this cost function as an gradient descent step will give the values of input features.

but we have assumed that we have parameters  $w$  &  $b$  for the different users,

where do you get these parameters from?

let's put together the algorithm from the last topic for learning  $w$  &  $b$  and what we just talked about in this <sup>topic</sup> for learning  $x$  & that will give us our collaborative filtering algorithm

## Collaborative filtering

(i) Cost fn to learn  $w^{(1)}, b^{(1)}, \dots, w^{(n_u)}, b^{(n_u)}$ :

$$\min_{w^{(1)}, b^{(1)}, \dots, w^{(n_u)}, b^{(n_u)}} \Rightarrow \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i: r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2$$

(ii) Cost fn to learn  $x^{(1)}, \dots, x^{(n_m)}$ :

$$\min_{x^{(1)}, \dots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j: r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

Putting (i) & (ii) together:

$$\frac{1}{2} \sum_{(i,j): r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2 +$$

$$\frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

→  $J(w, b, x)$



## Gradient Descent

### collaborative filtering

$$w_i^{(j)} = w_i^{(j)} - \alpha \frac{\partial}{\partial w_i^{(j)}} J(w, b, r)$$

$$b^{(j)} = b^{(j)} - \alpha \frac{\partial}{\partial b^{(j)}} J(w, b, r)$$

we need to optimize wrt "r" as well

$$r_k^{(i)} = r_k^{(i)} - \alpha \frac{\partial}{\partial r_k^{(i)}} J(w, b, r)$$

doing this will give you good values of  $(w, b, r)$

## Binary labels:

favorites, likes & clicks

many applications of recommender system or collaborative filtering algorithms involved binary labels where insted of a user giving you a one to five star or zero to five star rating, they just somehow give you a sense of they like this item or they didn't like this item.

### Example applications

- 1> Did user i purchase an item after being shown?
- 2> Did user i fav/like an item?
- 3> Did user i spend at least 30 sec with an item?
- 4> Did user i click on an item?   
 meaning  $\rightarrow 1$  engaged after being shown   
 meaning  $\rightarrow 0$  didn't engage after being shown   
 ?  $\rightarrow$  item not yet shown.

| Movie                | Alice (1) | Bob (2) | Carol (3) | Dave (4) |
|----------------------|-----------|---------|-----------|----------|
| Love at last         | 1         | 1       | 0         | 0        |
| Romance forever      | 1         | ?       | ?         | 0        |
| Cute puppies of love | ?         | 1       | 0         | ?        |
| Nonstop car chases   | 0         | 0       | 1         | 1        |
| Swords vs katana     | 0         | 0       | 1         | ?        |

## From regression to binary classification

Previously :-

Predict  $y^{(i,j)}$  as  $w^{(j)} \cdot x^{(i)} + b^{(j)}$

for binary labels :-

Predict that the probability of  $y^{(i,j)} = 1$

is given by  $y^{(i,j)} = g(w^{(j)} \cdot x^{(i)} + b^{(j)})$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Previous cost fn :-

$$\frac{1}{2} \sum_{(i,j): x^{(i,j)}=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{1}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2 + \frac{1}{2} \sum_{j=1}^{n_l} \sum_{k=1}^n (w_k^{(j)})^2$$



loss for binary labels :-  $y^{(i,j)} : f(w, b, x) (x) = g(w^{(j)} \cdot x^{(i)} + b^{(j)})$

$$L(f(w, b, x) (x), y^{(i,j)})$$

$$L(f(w, b, x) (x), y^{(i,j)}) = -y^{(i,j)} \log(f(w, b, x) (x)) - (1 - y^{(i,j)}) \log$$

↳ loss for single example

↳ binary cross entropy fn

$$(1 - f(w, b, x) (x))$$

$$J(w, b, x) = \sum_{(i,j): x(i,j)=1} L(f(w, b, x) (x), y^{(i,j)})$$

↳ cost for all examples

## Tensorflow implementation.

you can use gradient descent tensorflow implementation to find the ~~cost~~ derivative of the cost fn, it differentiates the cost fn automatically & helps you do gradient descent or you can use adam optimization algo

# Instantiate an optimizer

```
Optimizer = keras.optimizers.Adam(learning_rate = 1e-1)
```

```
iterations = 200
```

```
for iter in range(iterations):
```

# use Tensorflow's gradientTape

# to record the operations used to compute the cost with tf.GradientTape() as tape:

# Compute the cost (forward pass is included in cost)

```
Cost-value = CostFuncV(x, m, b, y_norm, R,  
num-users, num-movies, lambda)
```

# use the gradient tape to automatically retrieve

# the gradients of the trainable variable with the respect to the loss

```
grads = tape.gradient(Cost-value, [x, w, b])
```

# Run one step of gradient descent by updating

# the value of the variables to minimize the loss

```
optimizer.apply_gradients(zip(grads, [x, w, b]))
```

we use "auto diff" automatic differentiation here.



## Mean Normalization

⇒ So, Imagine you have a new viewer & they haven't given any rating. What algorithm will do is, it will predict that the new user will give 0 rating to all of the movies.

$$\frac{1}{2} \sum_{(i,j): r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{1}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2 +$$

↳ This part of the eqn will not be zero because the conditions don't full fill,

$$\frac{1}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

So  $w^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ ,  $b^{[5]} = 0$

So,  $w^{[5]} \cdot x^{(i)} + b^{(5)} = 0$

So, what we will do is "mean normalization" to prevent this from happening.   
 ↳ Subtracting mean from every prediction.

|   |   |   |   |   |
|---|---|---|---|---|
| 5 | 5 | 0 | 0 | ? |
| 5 | ? | ? | 0 | ? |
| ? | 4 | 0 | ? | ? |
| 0 | 0 | 5 | 4 | ? |
| 0 | 0 | 5 | 0 | ? |

$$u = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.5 \\ 2.5 \\ 1.5 \end{bmatrix}, \quad \begin{bmatrix} 2.5, 2.5, -2.5, -2.5, ? \\ 2.5, ?, ?, -2.5, ? \\ ?, 2, -2, ?, ? \\ -2.25, -2.25, 2.75, 1.75, ? \\ -1.25, -1.25, 3.75, -1.25, ? \end{bmatrix}$$

For user  $j$ , on movie  $i$  predict:

$(w^j \cdot x^{(i)} + b) + u_j$ , So now if  $(w^j \cdot x^{(i)} + b)$  becomes zero the new prediction will be the mean of the prediction of all users.

$$\text{ex} \rightarrow w^{(5)} x^{(1)} + b^{(5)} + \eta_1 = 0 + b + 2.5 \\ \Rightarrow \boxed{2.5}$$

## Finding related items -:

The features  $x^{(i)}$  of item  $i$  are quite hard to interpret.

To find other items related to it, find item

$k$  with  $n^{(k)}$  similar to  $x^{(i)}$

↳ to find how ~~similar~~ similar they are we do,

$$\sum_{l=1}^n \left( n_l^{(k)} - n_l^{(i)} \right)^2, \text{ this finds the square distance}$$

$$\| n^{(k)} - n^{(i)} \|^2$$

↳  $l_2$  norm

"the smaller the distance the closer they are.  
or ~~similar~~ similar they are!"

## Limitations of collaborative filtering

Cold start problem, How to

→ rank new items that few users have rated?

→ show something reasonable to new users who have rated few items?



Use side information about items or users:

- Item: Genre, movie stars, studio, - - -
- User: Demographics (Age, gender, location) expressed preferences.

Collaborative filtering vs content based filtering.

→ Collaborative filtering:-

Recommended items to you based on ratings of users who gave similar ratings as you.

→ Content-based filtering:

Recommended items to you based on features of users & item to find good match

→ User features:

- Age
- gender
- Country (1 not, 200)
- movies watched
- Average rating per genre

}  $x_u^{(j)}$  for user  $j$

→ Movie features:

- Year
- Genre/Genres
- Reviews
- Average rating

}  $x_m^{(i)}$  for movie  $i$

## Content-based filtering: Learning to match

→ Predict rating of user  $j$  on movie  $i$  as

$$w^{(j)} \cdot x^{(i)} + b^{(j)}$$

↓                  ↓

$V_u^{(j)}$       ,       $V_m^{(i)}$

↪ This doesn't affect the algorithm

$V_u^{(j)}$  → is a vector computed from the features of user  $j$  or  $(x_u^{(j)})$

$V_m^{(i)}$  → is a vector computed from the features of like the ones you saw on the previous page.

if we can come up with an appropriate value of both vectors then their dot product between these 2 vectors will be a good prediction of the rating

User preferences  $V_u = \begin{bmatrix} 4.9 \\ 0.1 \\ \vdots \\ 3.0 \end{bmatrix}$  ,  $V_m = \begin{bmatrix} 4.5 \\ 0.2 \\ \vdots \\ 3.5 \end{bmatrix}$

So, how are you going to predict make  $V_u$  &  $V_m$  from  $x_u^{(j)}$  &  $x_m^{(i)}$  has to be of the same size because dot product can only be done in same size array

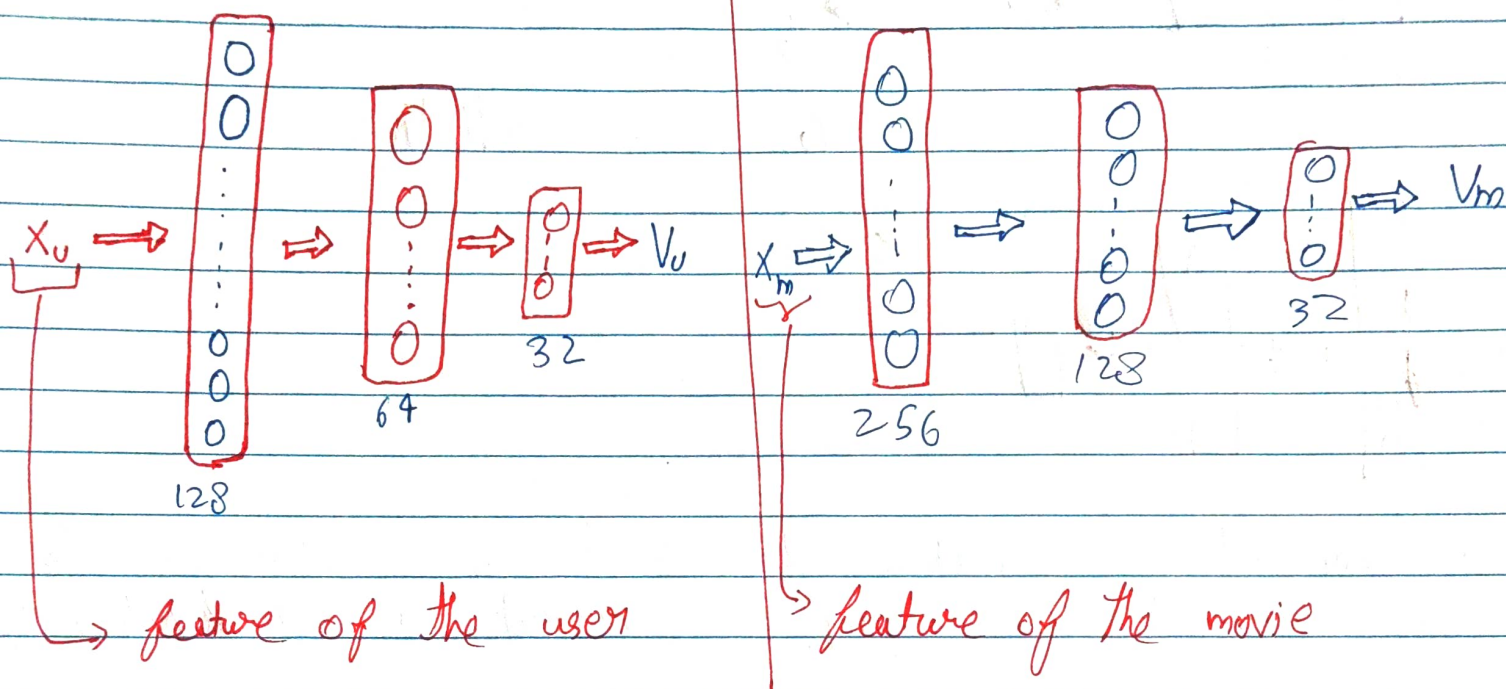


# Deep learning for content-based filtering -

## Neural Network architecture

$X_u \rightarrow V_u$  User network

$X_m \rightarrow V_m$  movie network



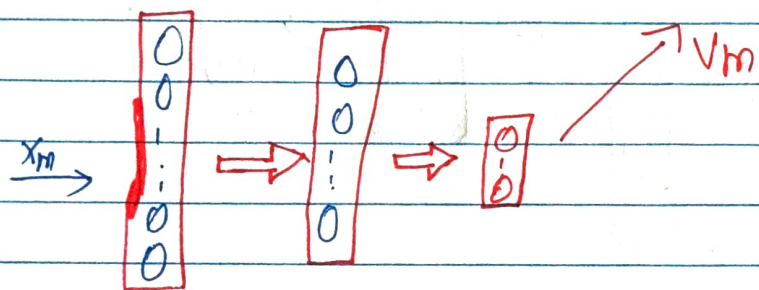
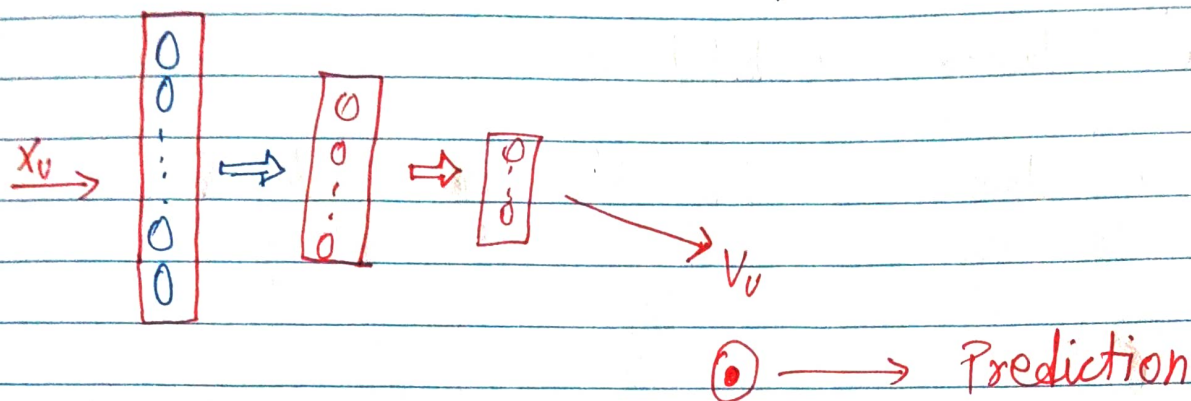
Prediction:  $V_u \cdot V_m$

→ We need two neural networks with output layers being ~~same~~, because the dot product requires the vector size being same.

you can ~~skip~~ a sigmoid fn of top of the prediction to predict the probability that  $g(i, j)$  is 1. so

$$g(V_u \cdot V_m)$$

## Neural network architecture :-



to train all the parameters of both the user network & the movie network we're going to construct a cost function  $J$ ,

$$\text{Cost fn } J = \sum_{(i,j): r(i,j)=1} (V_u^{(j)} \cdot V_m^{(i)} - y^{(i,j)})^2 + \text{NN regularization term}$$

This cost fn describes how far are we from the prediction ~~for~~ for a single example, the lower the better

## Learned user & item vectors:

- $V_u^{(j)}$  is a vector length 32 that describes user  $j$  with features  $x_u^{(j)}$ .
- $V_m^{(i)}$  is a vector of length 32 that describes movie  $i$  with features  $x_m^{(i)}$ .



To find movies similar to movie  $i$  :- Squared

$$\|V_m^{(k)} - V_m^{(i)}\|^2 \quad \left. \vphantom{\|V_m^{(k)} - V_m^{(i)}\|^2} \right\} \begin{array}{l} \text{L2 norm of the} \\ \text{difference btw 2} \\ \text{vectors} \end{array}$$

Let's say you want to find a similar movie to movie vector  $V_m^{(i)}$ , then given another movie vector  $V_m^{(k)}$  just take the L2 norm of the difference & that will tell you how similar are these vectors. "the lower the better"

Note: This can be pre computed ahead of time

"In these algorithms engineering good features is important"

→ it is computationally expensive to run if you have a large catalog of a lot of movies

## Recommending from a large Catalogue

↳ 2 steps: Retrieval & Ranking.

Retrieval:

→ Generate large list of plausible item candidates  
e.g.

1. → for each of the last 10 movies watched by the user, find 10 most similar movies

2. → for most viewed 3 genres, find the top 10 movies

3. → Top 20 movies in the country.

→ Combine retrieved items into list, removing duplicates & items already watched/purchased

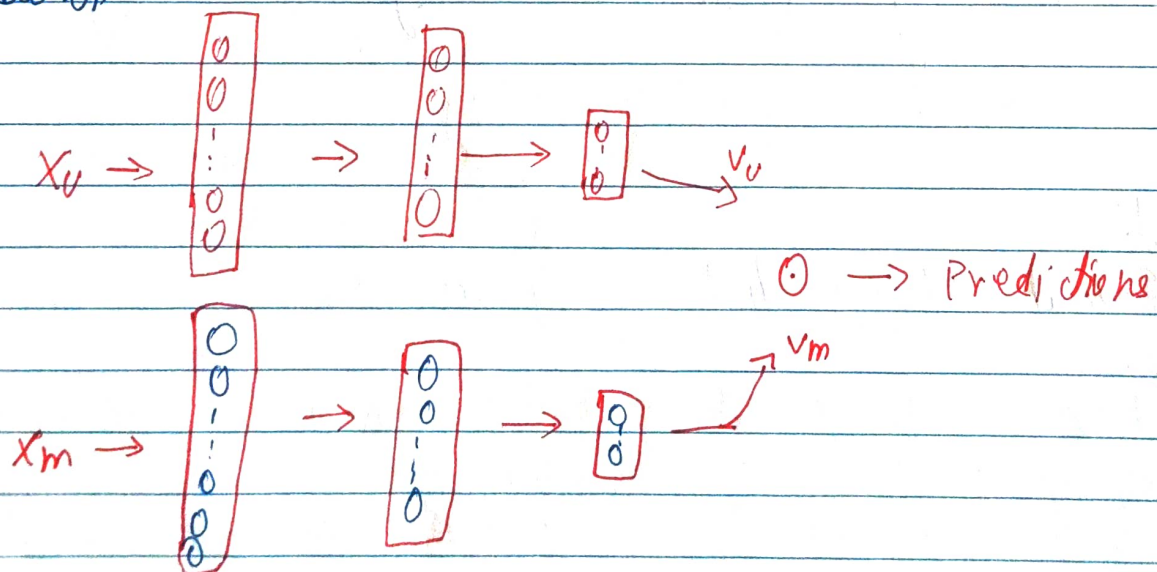
{ During the "retrieval" step, we will generate a large list of plausible item candidates, that tries to cover a lot of possible things that you might recommend to the user & it's ok during the retrieval step if you include a lot of item that the user isn't likely to like. & then during the "Ranking" we will fine tune & pick the best item to recommend to the user }

### Ranking:

→ Take list retrieved & rank using learned model

→ Display ranked items to user

→ you will feed the user feature vector  $x_u$  & movie feature vector  $x_m$  to the model & then it will decide what movies to choose for recommendation by prediction





One of the decisions that you will have to make during the retrieval step is to how many items you want to retrieve.

- ↳ Retrieving more items results in better performance but slower recommendations
- ↳ To analyse/optimize the trade-off, carry out offline experiments to see if retrieving additional items results in more relevant recommendations (i.e.,  $P(y_{ij}) = 1$  of items displayed to user are higher).

## Content based filtering

Tensorflow implementation :-

```
User-NN = tf.keras.models.Sequential([  
    tf.keras.layers.Dense(256, activation='relu'),  
    tf.keras.layers.Dense(128, activation='relu'),  
    tf.keras.layers.Dense(32)])
```

```
Item-NN = tf.keras.models.Sequential([  
    tf.keras.layers.Dense(256, activation='relu'),  
    tf.keras.layers.Dense(128, activation='relu'),  
    tf.keras.layers.Dense(32)])
```

# create the user input & point to the base network

```
input_user = tf.keras.layers.Input(shape=(num-user-features))  
Vu = User-NN(input_user)  
Vu = tf.linalg.l2_normalize(Vu, axis=1)
```

# create the item input & point to the base network

```
input_item = tf.keras.layers.Input(shape=(num-item-features))  
Vm = item-NN(input_item)  
Vm = tf.linalg.l2_normalize(Vm, axis=2)
```

# measure the similarity of the two vector outputs

```
Output = tf.keras.layers.Dot(axes=1)([Vu, Vm])
```



# Specify the inputs & output of the model

```
model = Model([input-user, input-item], Output)
```

# Specify the cost function

```
Cost-fn = tf.keras.losses.MeanSquaredError()
```

Code explanation

→ User-NN, line 5 creates a NN with 3 Dense layers with 32 logits in the output.

→ item-NN, creates a NN with 3 Dense layers with 32 logits in the output.

in, input-user, we are creating a tensor with the shape which is equal to the no of features.

in, V<sub>u</sub>, we pass the input's to the user-NN & store it into "V<sub>u</sub>". again in "l2-normalize" we perform l2 normalization in which we divide each row by their respective row's ~~by their~~ l2 norms so that we can get a value between (0-1).

Same for input-item,

in the Output, we do a vector dot product to find the similarity, & since we did the ~~l2~~ l2 normalization, the closer the value to "1" the similar the vectors are.

## PCA $\rightarrow$ Principal component analysis

its dimensionality reduction, it reduces the number of features in a given dataset so that we can visualize it properly.

Do it from the mathematics from machine learning notes, it's more beautiful

## PCA in Scikit learn code

- ①  $\text{PCA\_1} = \text{PCA}(n\_components = 1)$
- ②  $\text{PCA\_1}.fit(x)$
- ③  $\text{PCA\_1}.explained\_variance\_ratio$
- ④  $X\_trans\_1 = \text{PCA\_1}.transform(x)$
- ⑤  $X\_reduced\_1 = \text{PCA\_1}.inverse\_transform(X\_trans\_1)$

Code explanation  $\therefore$

in ①, we are creating an object  $\text{PCA\_1}$ , which will do "PCA" to the data passed & reduce it to one component or feature.

②  $\rightarrow$  we pass "x" data to the fit function of this object

③  $\rightarrow$  this explains the contribution of each principal component towards capturing that spread

④  $\rightarrow$  it is done " $.transform(x)$ " to project your original data into new coordinate system.

⑤  $\rightarrow$  It tries to map out the transformed coordinates to the original feature space.