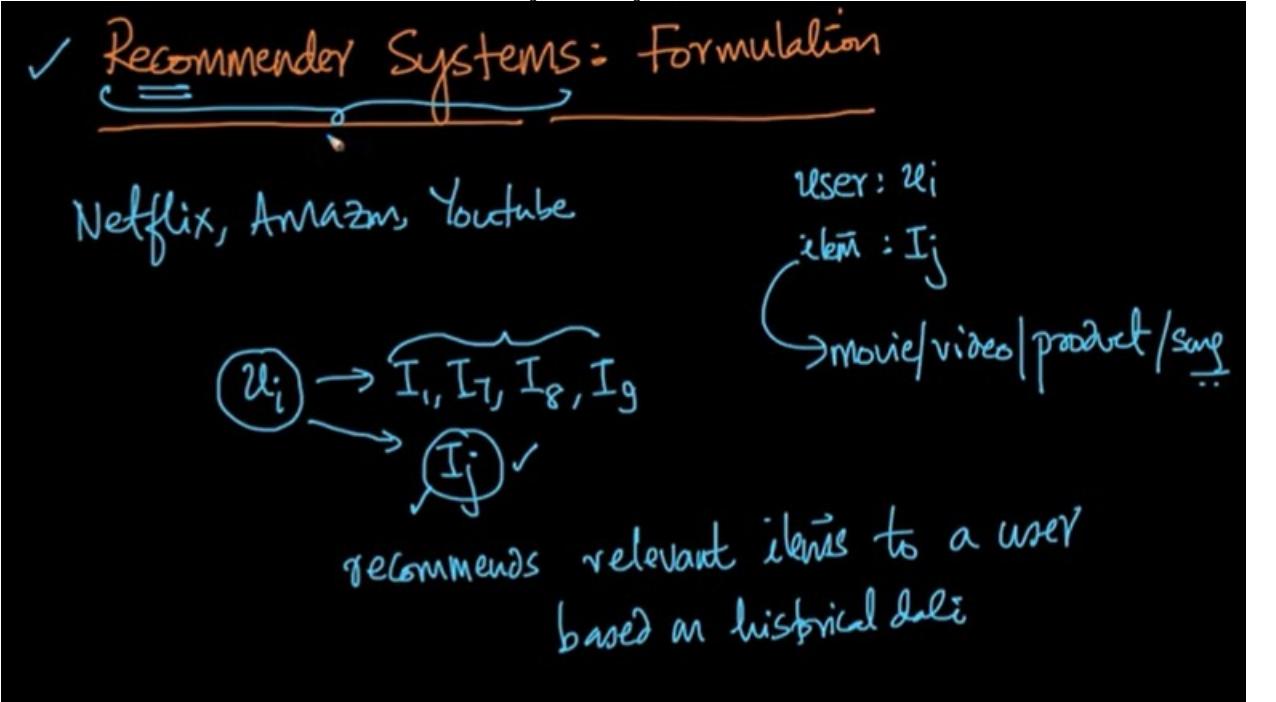


Recommender systems- matrix factorization

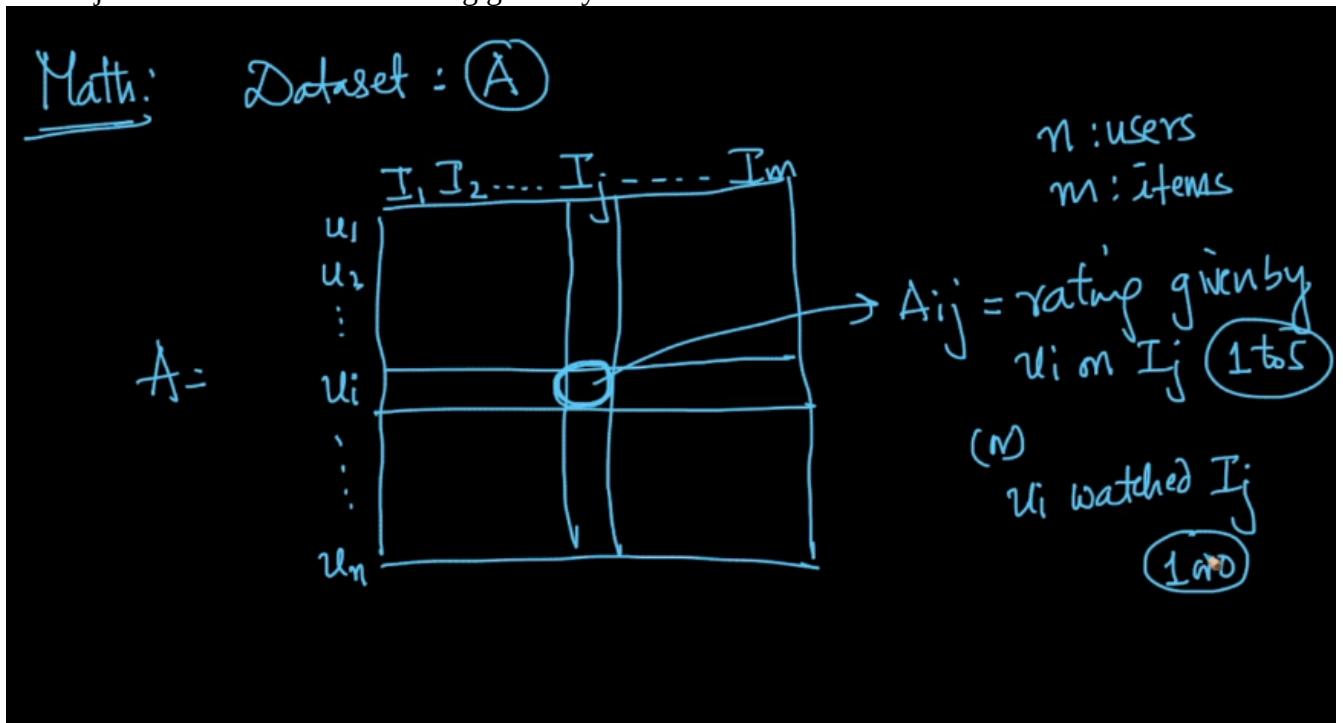
Recommends the items from the users history activity.



Mathematical form: It recommends relevant items to the user based on historical data the person liked.
The data set is in the matrix.

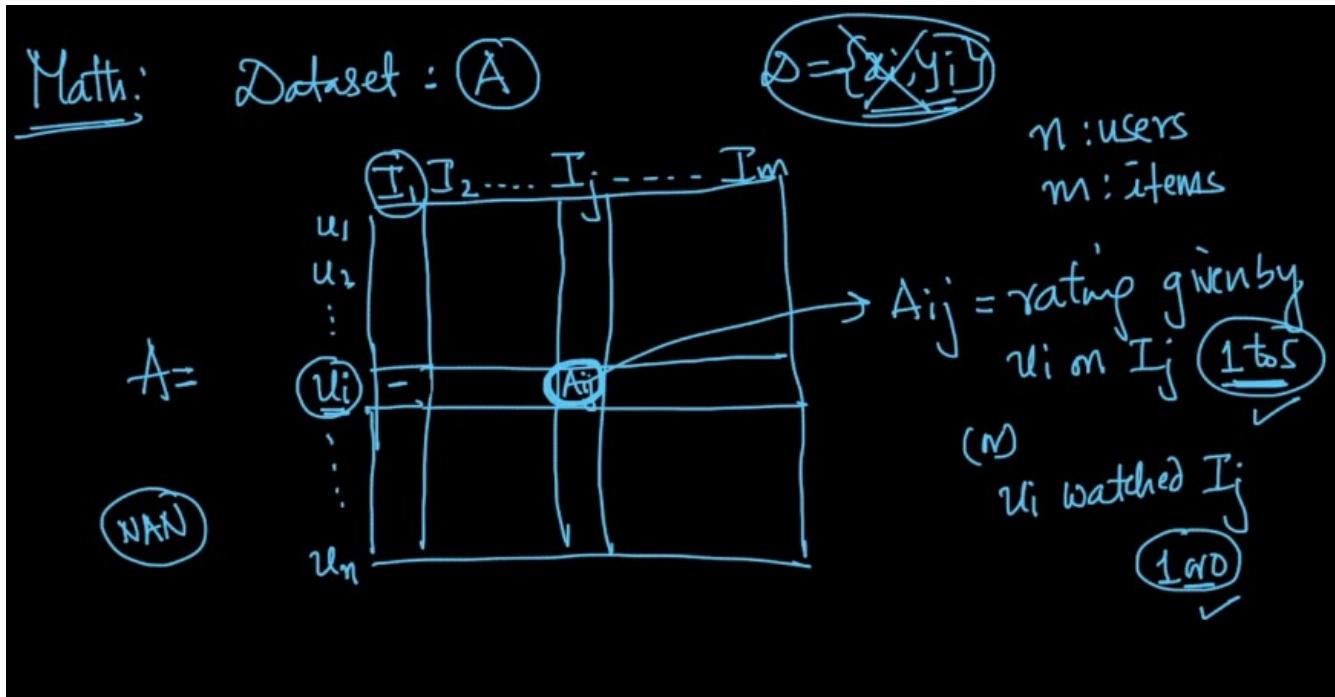
Each row in the matrix is a user, and each column in the matrix is an item(Movie, song).

The A_{ij} is the measure such as rating given by an item J.



Consider the person not watched the movie. We can give one unique number like -1 for that space.

Where ever there is no data leave the cell as a blank. NaN is the not a number in Python.



Property of the matrix:

The matrix can be very sparse.

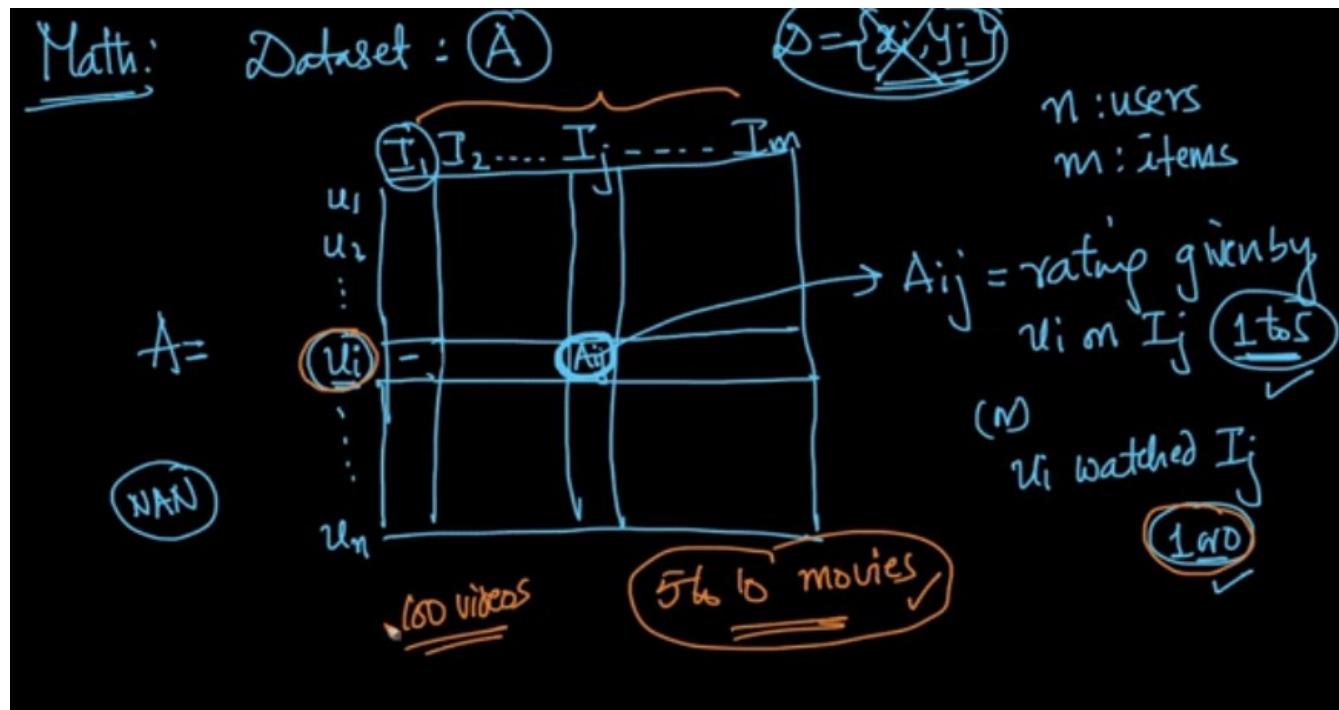
Example:

\mathbb{A} Matrix A of ratings is very sparse

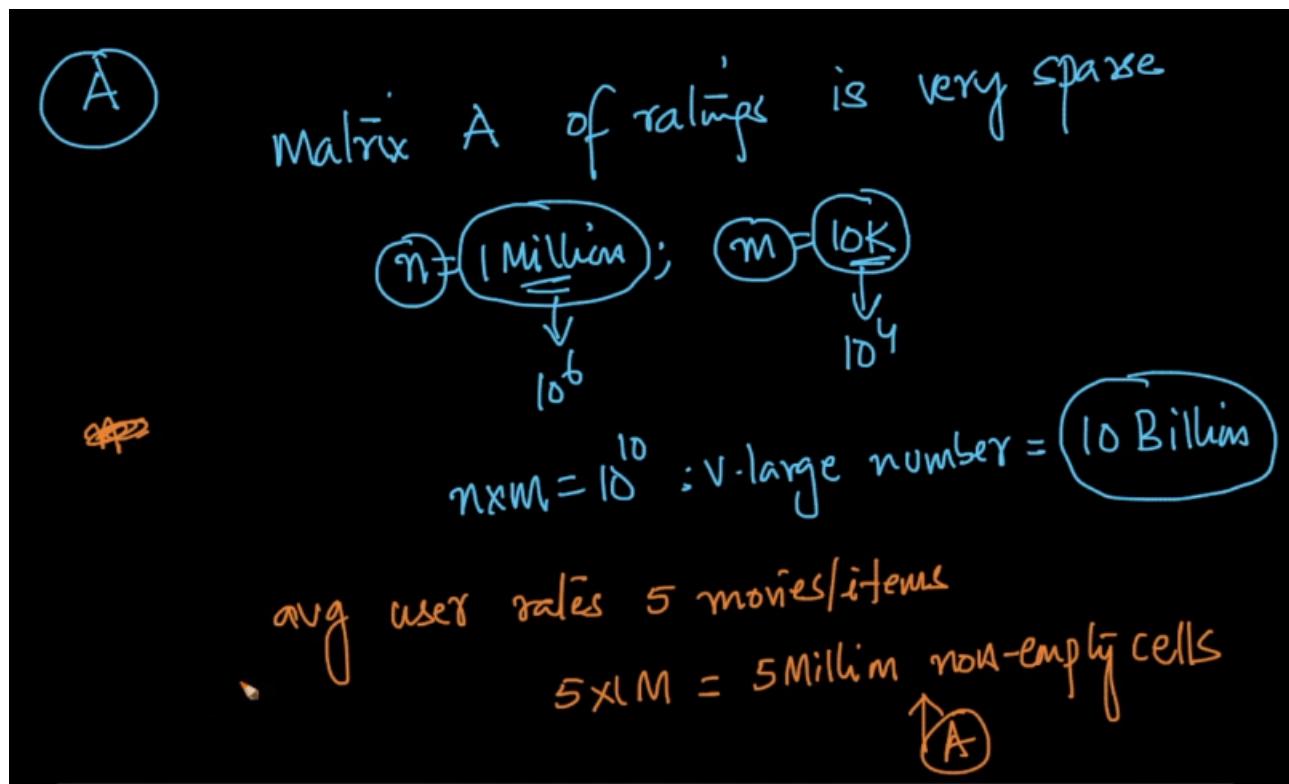
$n = 1 \text{ Million}$; $m = 10K$

$n \times m = 10^10$; v. large number = 10 Billion

Even for 1 million users the matrix is very large, most of the matrix is sparse because a user can rate only 5 – 10 movies.



There will be very few movies of non empty entries.



Calculation of sparsity of matrix:

The big task of building a recommend-er system is to recommend a new item the user .

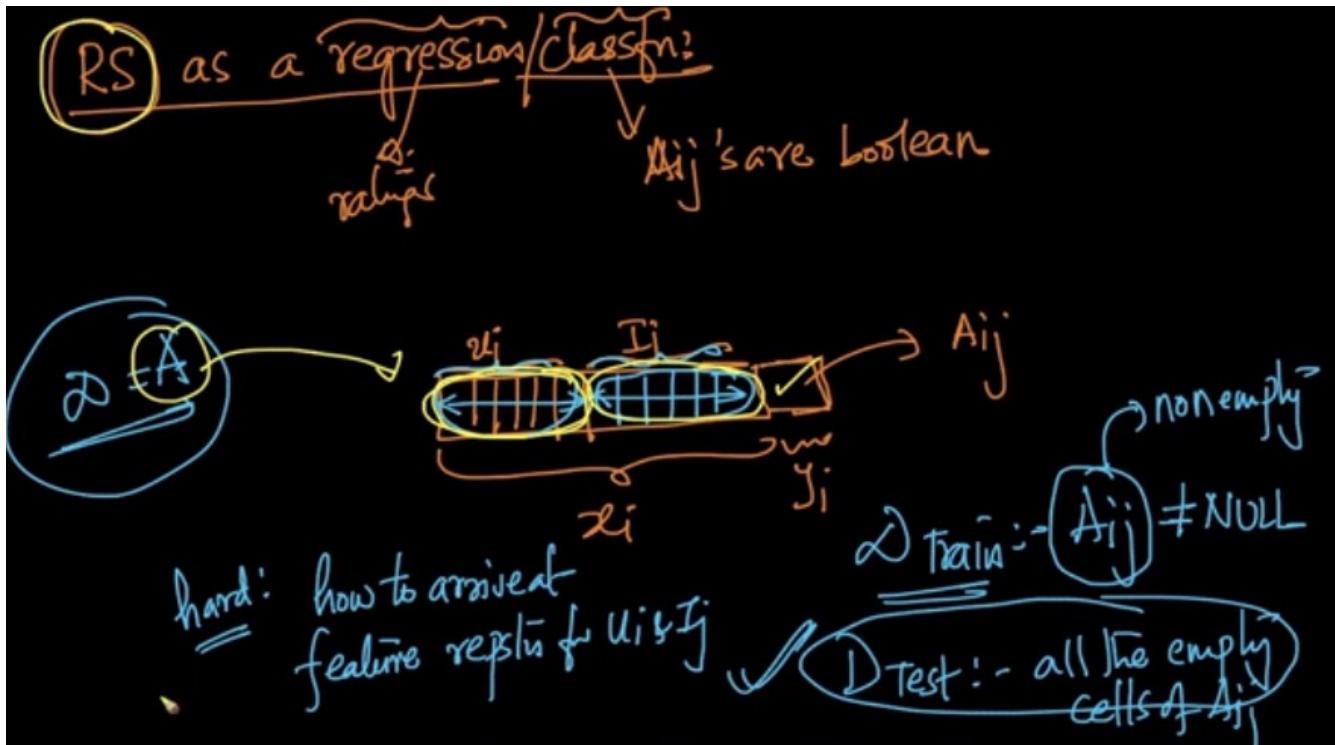
$$\text{Sparsity of } A = \frac{\# \text{ non-empty cells}}{\text{Total } \# \text{ cells}} = \frac{5 \times 10^4}{10^{10}} = 5 \times 10^{-4}$$

Task of RS

\hat{u}_i :- I_1, I_3, I_7, I_8

↳ recommend a new item I_j

We can make the problem to make a regression / classification problem:



Fill the empty cells with the reasonable values, based on the non empty cells.
 This is the problem of matrix fac completion value.

$A := \text{RS}$

(RS) as a matrix completion problem

Matrix Completion

$A = \begin{bmatrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{bmatrix} \rightarrow$ some values of A_{ij} are given
 \rightarrow many A_{ij} 's are empty

Fill up the empty cells with reasonable values based on values in the non-empty cells

$A =$

Predicted or filled value

fill up these empty cells

| | I_1 | I_2 | \dots | I_j | \dots | I_M |
|-------|-------|-------|---------|-------|---------|-------|
| u_i | 3 | 4.5 | 2 | | | 5 |

$\hat{A}_{i2} = 4.5$

\downarrow

$\left\{ \begin{array}{l} u_i \text{ could be } \\ \text{recommend } I_2 \end{array} \right.$

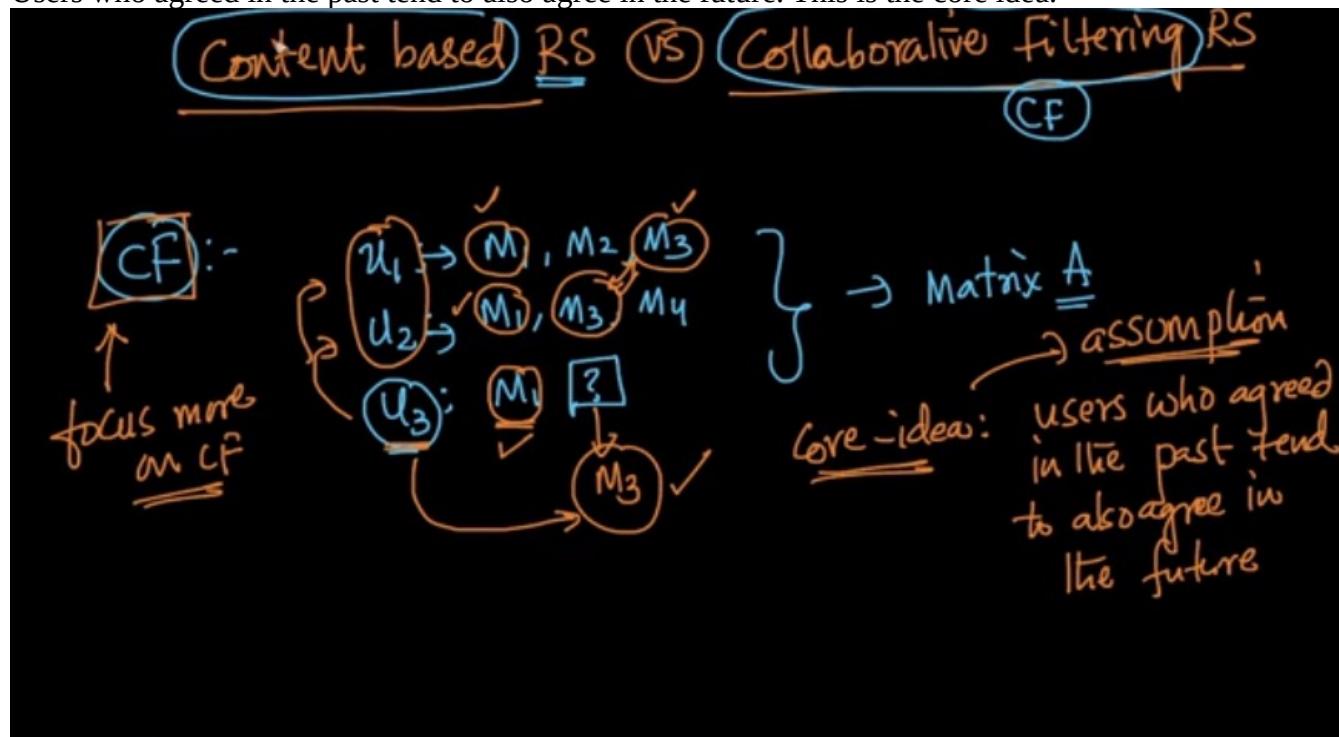
Then we can formulate the matrix completion as matrix factorization.

Content based vs. collaborative filtering:

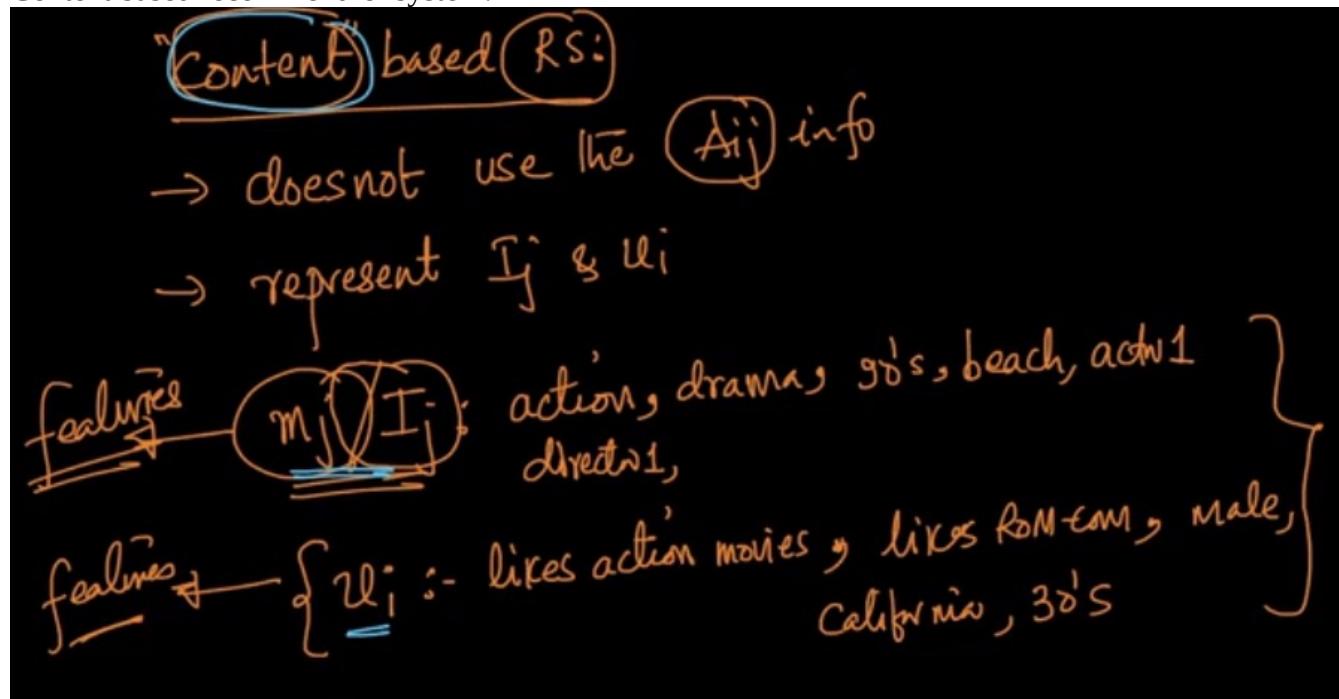
Collaborative filtering:

Example:

Users who agreed in the past tend to also agree in the future. This is the core idea.

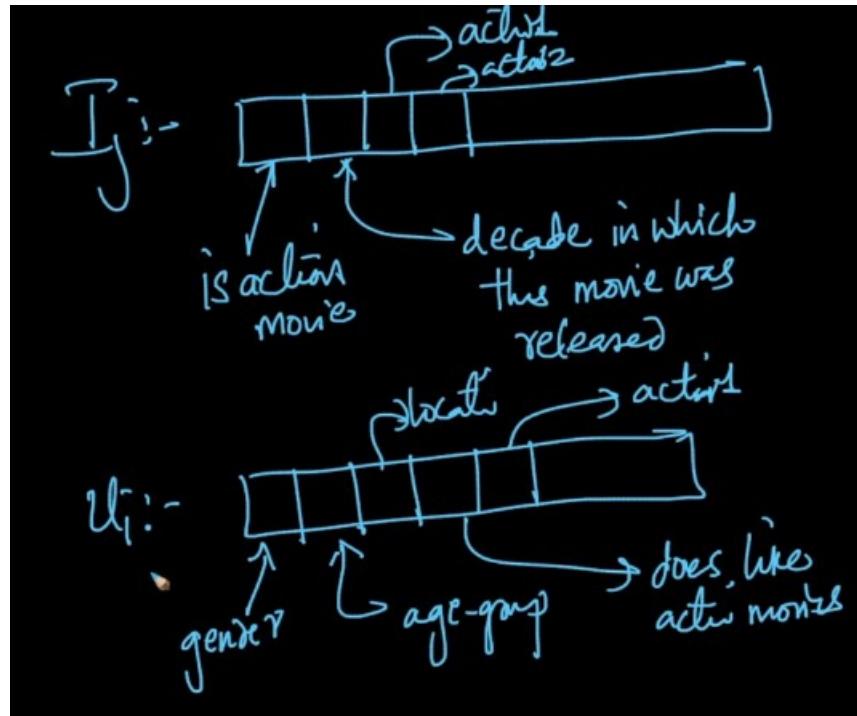


Content based recommend-er system:

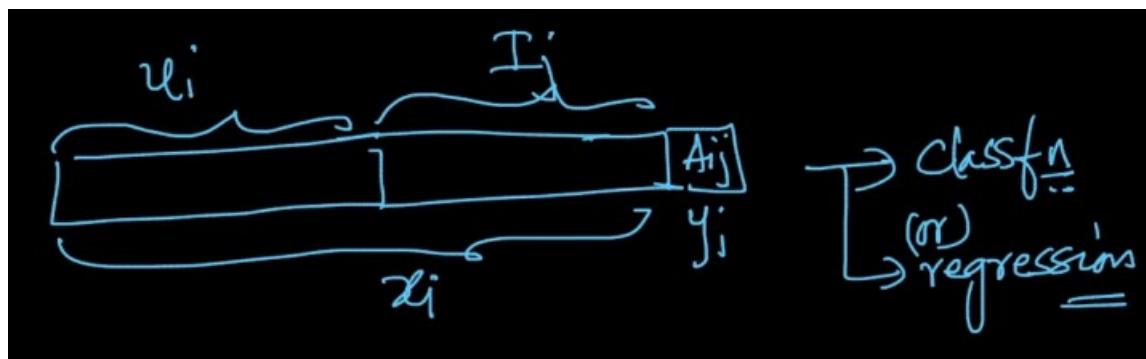


We can represent the collaborative filtering item can be written as:

The items and users can be represented as a vector:



We can pose this as a classification or regression problem.

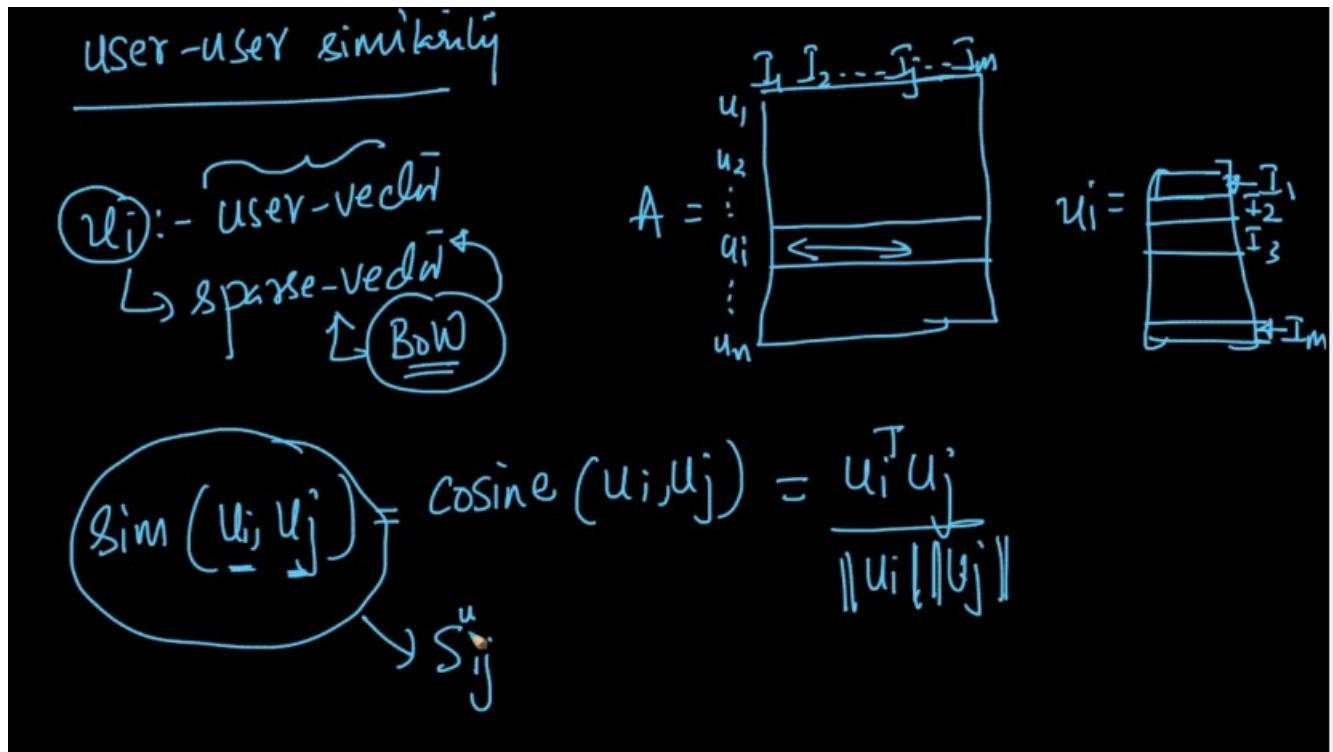


Content based recommend-er system is the classification and regression problems.

Similarity based algorithms:

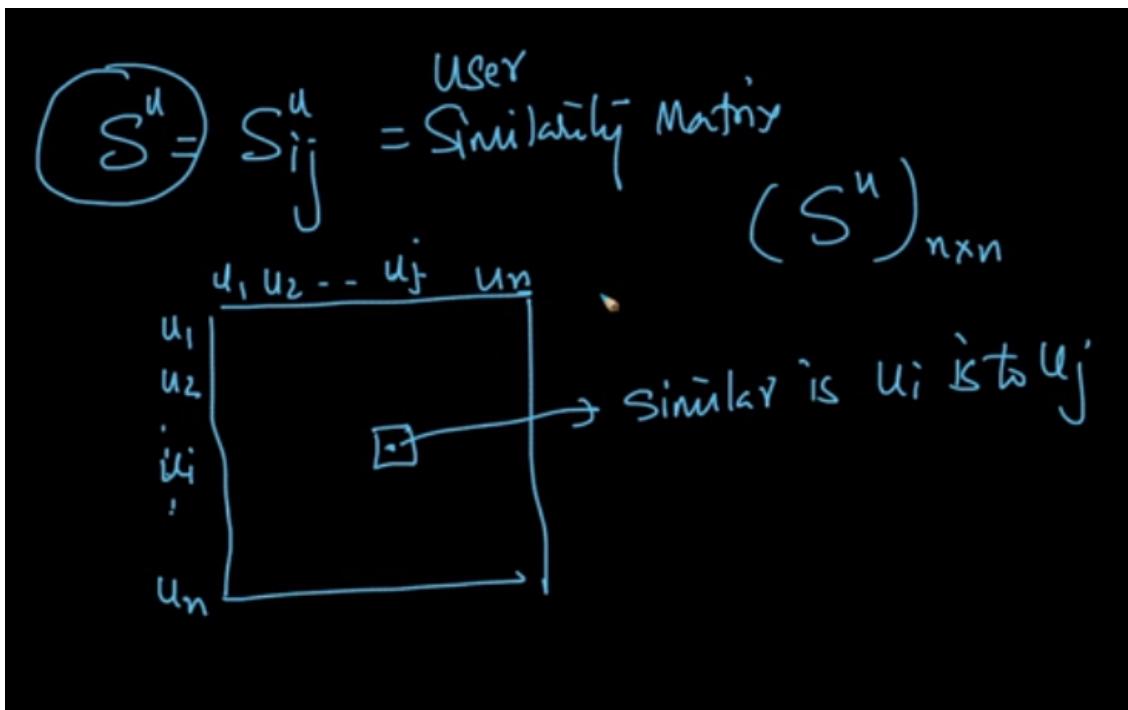
User – User based similarity -

We can compute the similarity measures of the users.



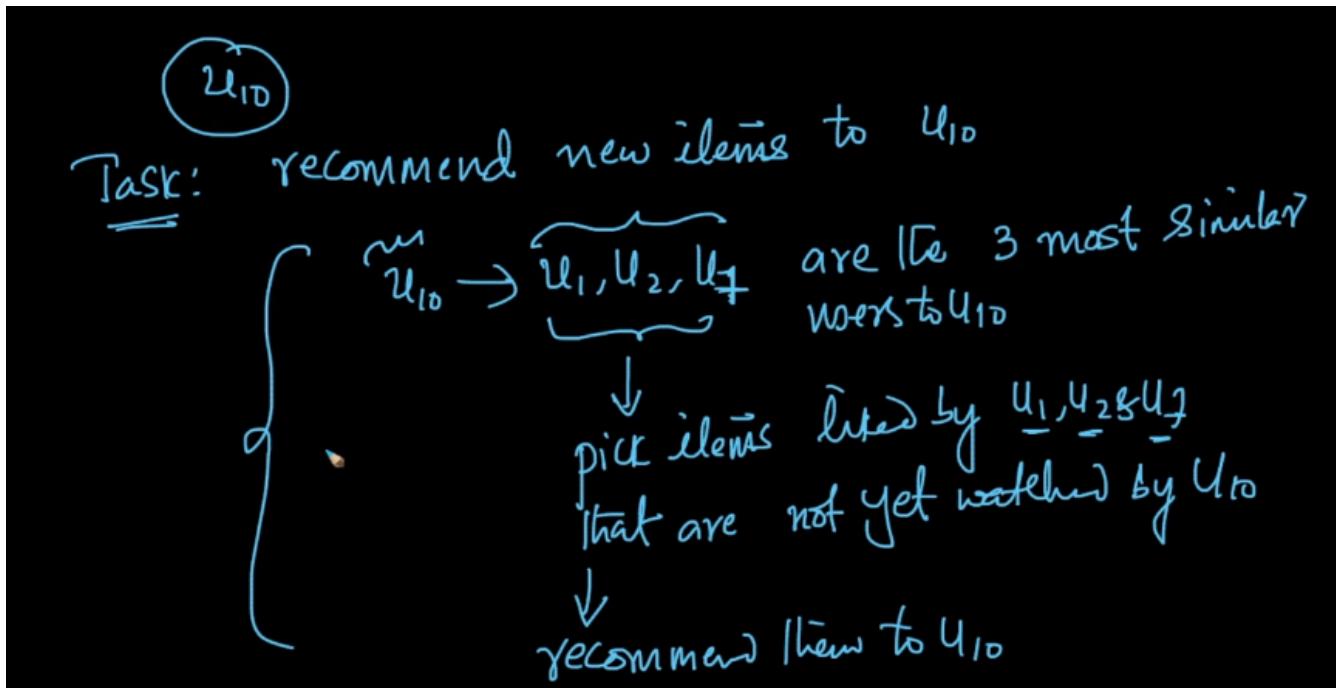
Imagine a matrix $S = S_{ij}$ = User similarity matrix.

This me $U_n * U_n$ matrix.



Let's take the task is to recommend new items to the U10.

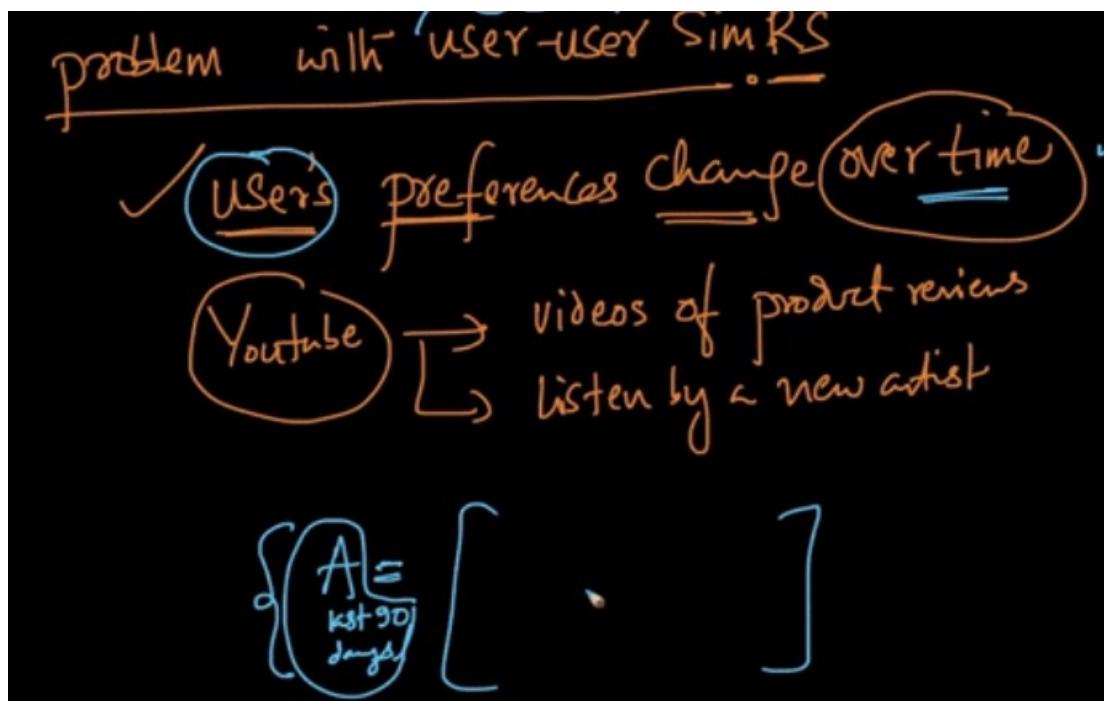
Let's pick items that are liked by U1, U2 and U7 that are not yet watched by U10 and recommend them to U10.



There is one small problem with user – user based sim recommend-er system.

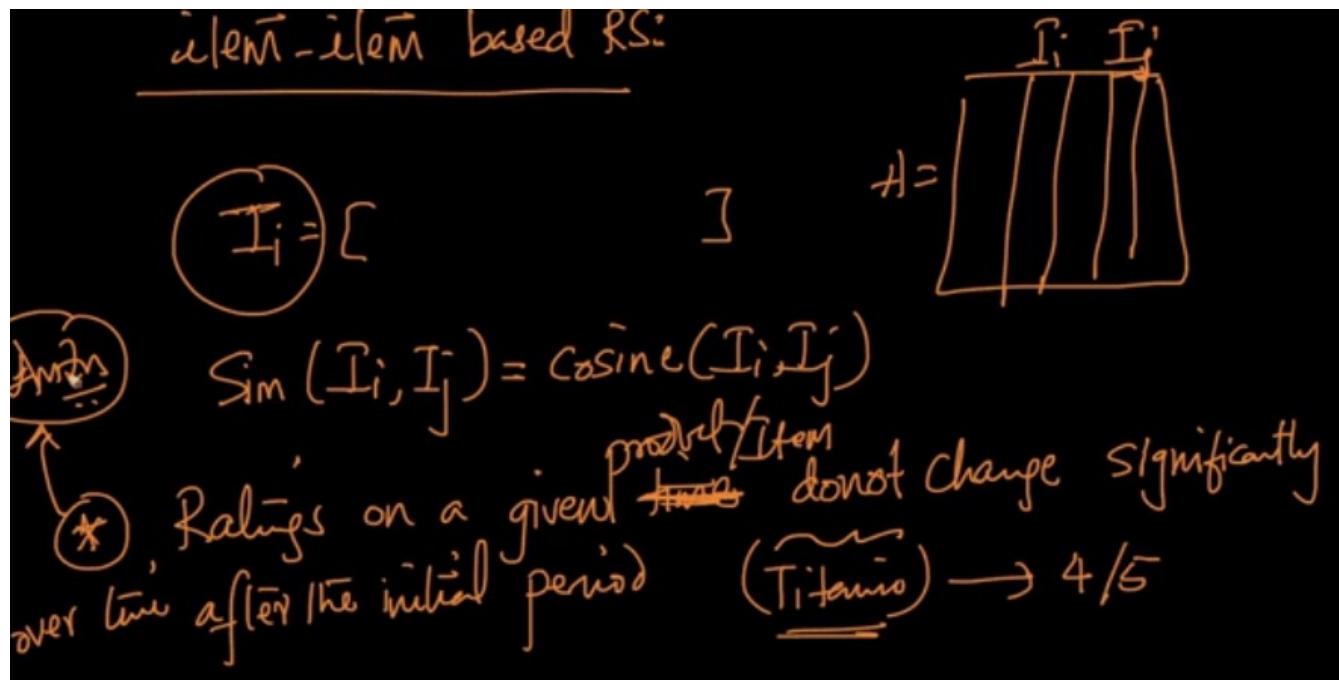
User similarity change overtime.

Users preferences change much more from time to time. We can build there matrix using the last 90 days of data. The matrix becomes sparser.

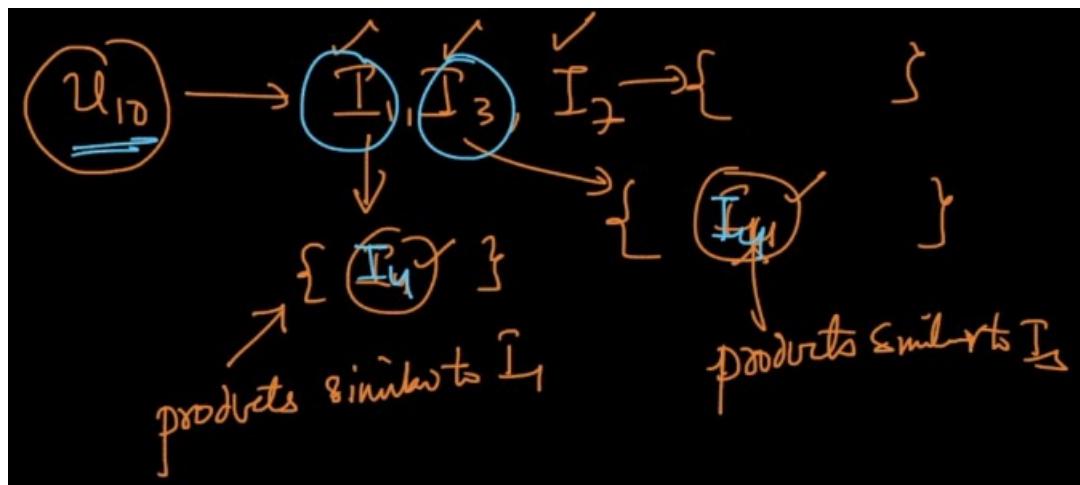


The alternative approach for this is item – item based RS:

I will represent each item as a vector, We can represent as a cosine similarity of two items. The rating of the item could not change significantly.

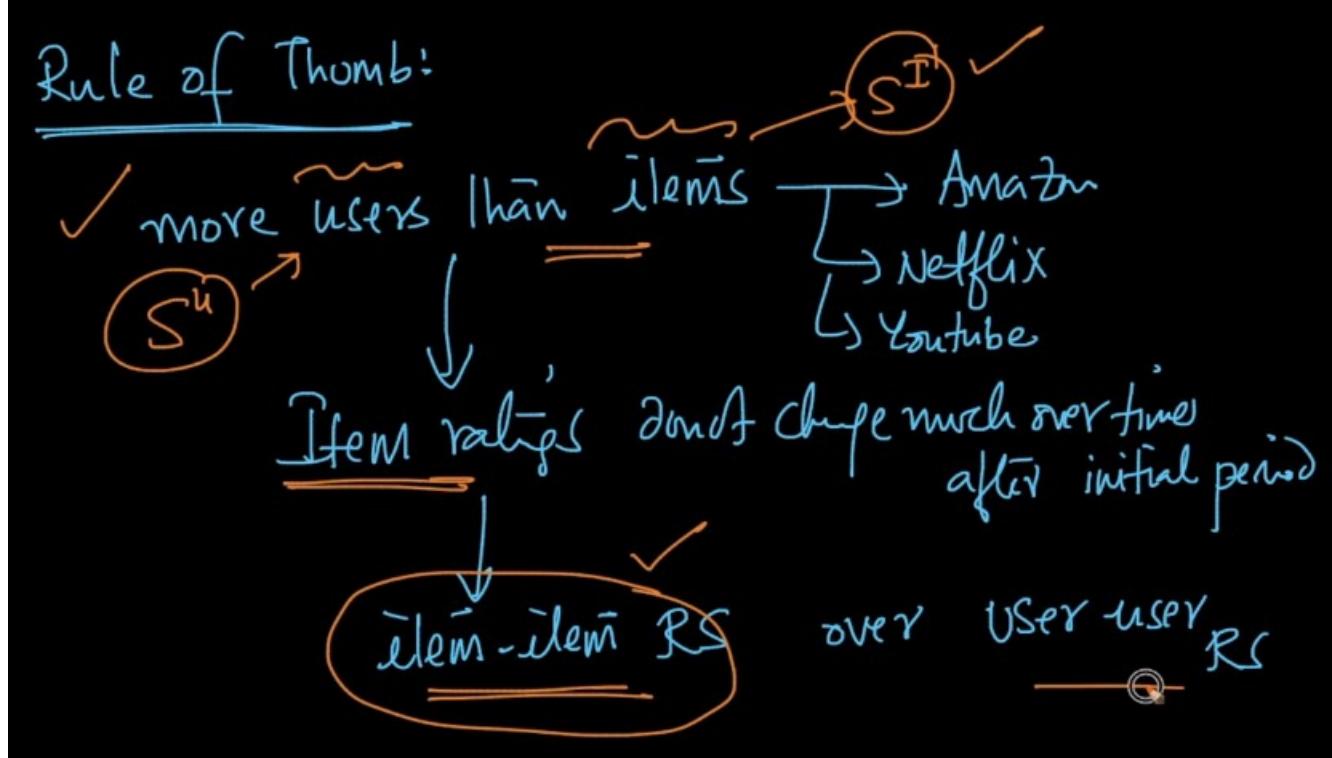


Rating for a product that does not change much, so it is the better chance.



WHEN THE RATING OF THE ITEMS DO NOT CHANGE MUCH OVER TIME AFTER INITIAL PERIOD, THEN WE CAN USE THE ITEM - ITEM BASED SIMILARITY MEASURES.

OVER USER – USER BASED RECOMMENDER SYSTEM.



Matrix factorization: PCA, SVD: It is the concept in the dimensional reduction.

PCA :- $X \xrightarrow{n \times d}$ Data Matrix (centered)

$$S_{d \times d} \quad \frac{X^T X}{n-1} = \text{Co-variance Matrix}$$

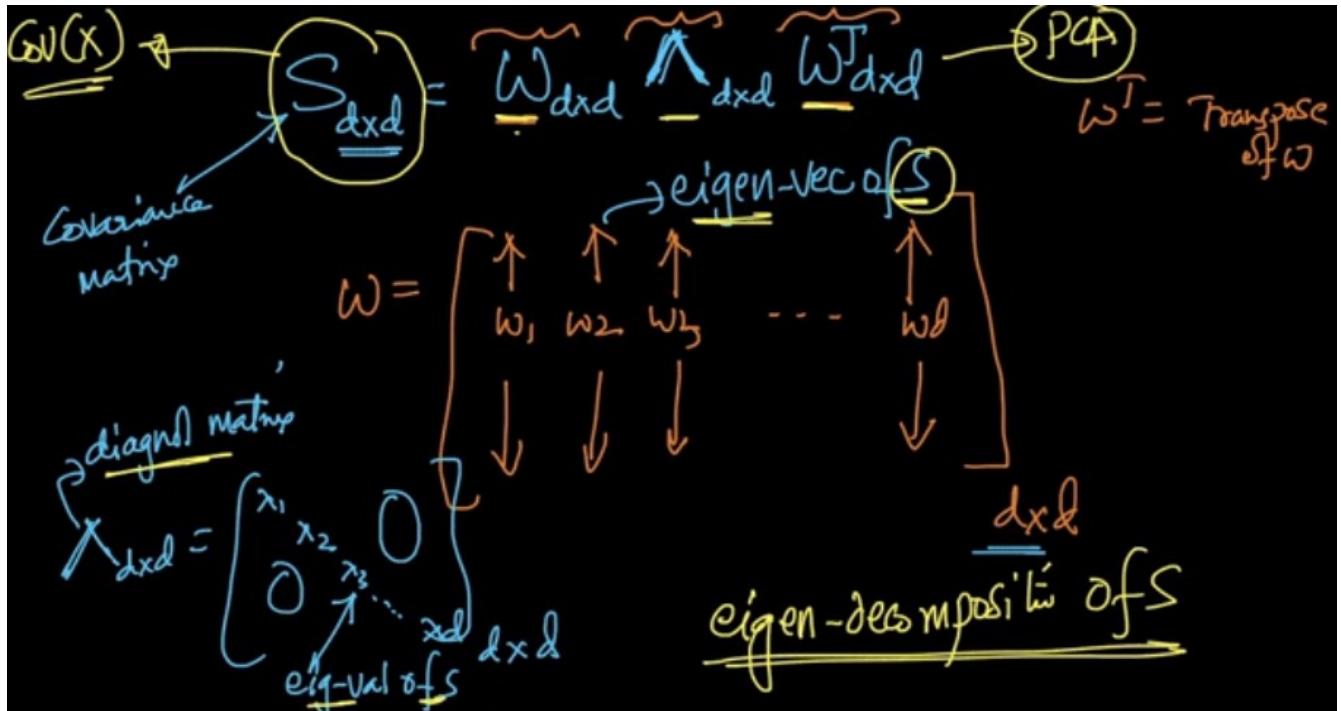
\hookrightarrow eigen-values : - $\lambda_1, \lambda_2, \dots, \lambda_d$

\hookrightarrow eigen-Vectors : - w_1, w_2, \dots, w_d

$\left\{ d \rightarrow d' \quad \text{top. } \underline{\lambda}'s \rightarrow \underline{w'}s \right\}$

Covariance matrix can be decomposed as the product of three matrices.

This decomposition is called eigen decomposition of matrix.



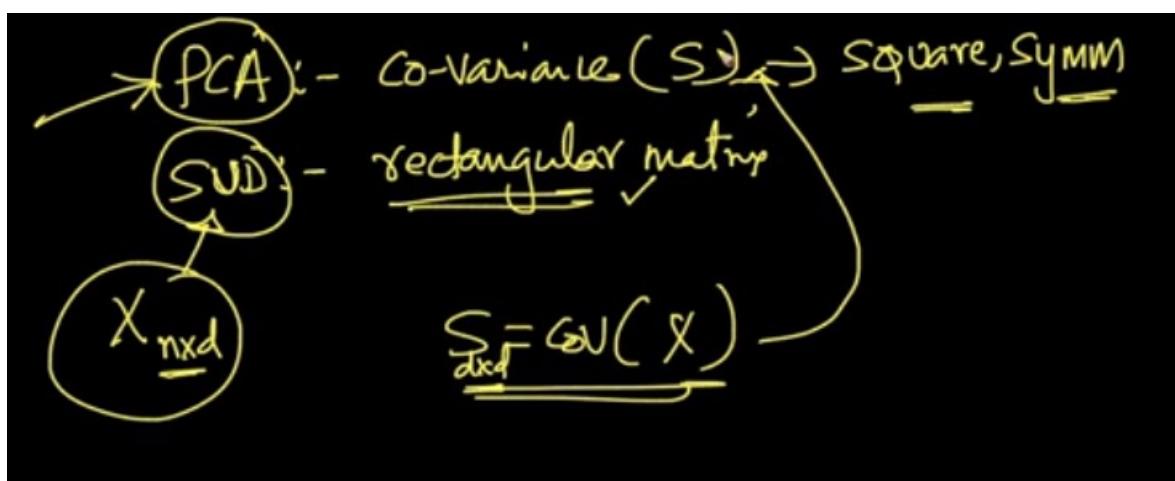
One application of PCA is dimensional reduction and also called matrix factorization.

This is also called eigen decomposition.

Singular value decomposition (SVD): It is related to PCA.

PCA CAN ONLY WORK ON SQUARE (OR) SYMMETRIC MATRIX.(LIKE CO-VARIANCE MATRIX)

SVD CAN BE APPLIED ON ANY RECTANGULAR MATRIX.



$$X_{n \times d} = U \sum_{n \times n} V^T_{d \times d} \rightarrow SVD$$

eig-vecs eig-val of $S \rightarrow \lambda_1, \lambda_2, \dots, \lambda_d$

$$\text{cov}(X) = S = U \Lambda U^T \rightarrow PCA$$

$$\sum = \begin{cases} \text{(S)} & \text{singular values of } X \\ \begin{matrix} s_1 & & & \\ \vdots & s_2 & s_3 & \dots & s_d \\ \hline m-1 & & & & 0 \end{matrix} & n \times d \end{cases} \rightarrow \text{diagonal matrix}$$

$s_i^2 = \lambda_i$

eig-val of S & singular values of X
 (λ_i) (s_i)

$$\left\{ \begin{array}{l} U_i : - i^{th} \text{ col. of } U = \text{eig. vec of } (X_{n \times d} X_{d \times n}^T) \\ V_i : - i^{th} \text{ col. of } V = \text{eig. vec of } (\underline{X}_{d \times n}^T \underline{X}_{n \times d}) = S \end{array} \right.$$

↑
not \subseteq

$$G_{\mathbf{V}}(\mathbf{x}) = \mathbf{S} = \mathbf{W} \Lambda \mathbf{W}^T \rightarrow \text{PCA}$$

↗ eig-val of $S(w_i)$
 ↘ eig-vec of $S(x_i)$

$$\mathbf{X} = \mathbf{U} \sum \mathbf{V}^T \rightarrow \text{SVD}$$

↓
 Singular-val δ_i ; $\frac{\delta_i^2}{n-1} = \lambda_i$

Fsvd

Matrix factorization: NMF

Non-negative Matrix factorization (NMF)
 or (NNMF)

$$\mathbf{A}_{n \times m} = \underbrace{\mathbf{B}_{n \times d} (\mathbf{C}^T)^T}_{d \times m} \quad \mathbf{B}: n \times d \quad \mathbf{C}: m \times d$$

(NMF)
 s.t. $B_{ij} \geq 0 \quad \forall i, j$
 $C_{ij} \geq 0 \quad \forall i, j$

The elements in the NMF are all positive.

Matrix factorization for collaborative filtering:

MF for Collaborative filtering

$A = \begin{bmatrix} & & & \\ & - & - & - \\ u_i & - & A_{ij} & - \\ & - & - & - \end{bmatrix}$

✓ A_{ij} :- value on I_j by u_i
 ✓ Sparse matrix; many empty cells

Let $\left\{ \begin{array}{l} A = \underbrace{B * C^T}_{n \times m} \\ \downarrow \quad \downarrow \\ \text{users} \quad \text{items} \end{array} \right.$ $\left\{ \begin{array}{l} B: n \times d \\ C: m \times d \\ d > 0 \\ d \leq \min(m, n) \\ d \leq m \leq d \leq n \end{array} \right.$

A_{ij} is the product of b_i and c_j .

$$A_{ij} = B_i \times C_j$$

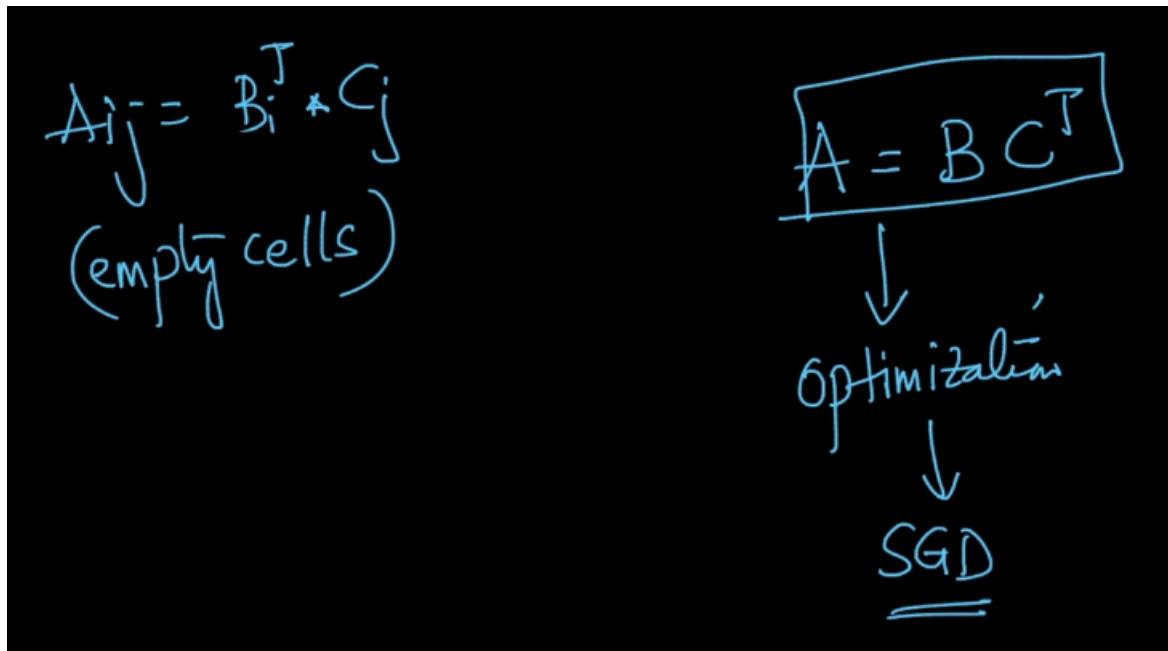
B_i is the i^{th} row of B
 C_j is the j^{th} row of C

$A = u_i \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ \vdots \\ I_m \end{bmatrix}_{n \times m}$

$B = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{bmatrix}_{n \times d}$

$C = \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ \vdots \\ I_m \end{bmatrix}_{m \times d}$

We can write this as follows:



To find B and C: Matrix factorization

Lets reduce the sparse matrix by solving the optimization problem.

find $B \& C$: MF

$\arg \min_{B, C} \sum_{\{i, j\}} (A_{ij} - B_i^T C_j)^2$

s.t. A_{ij} is not empty

Full matrices

$B^T = \begin{pmatrix} & \uparrow & \uparrow & & \uparrow \\ b_1 & b_2 & b_3 & \dots & b_n \\ & \downarrow & \downarrow & & \downarrow \end{pmatrix}_{d \times n}$

$A_{ij} = B_i^T C_j$

can't use empty A_{ij}

Squared-loss

If we find the matrices B.T and C then the matrices are good.

We are minimizing the squared loss like regression problems.

We can apply SGD to solve this optimization problem. At the end we get the matrices B and C.

solve optimizn. problem:

$$B = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} \quad n \times d$$
$$C = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} \quad m \times d$$

n = # users

m = # items

At the end of the problem we get the Bi's and cj's

Step - 1

Solve the optimization problem.

① Solve optimizn problem SGD

$$\underset{B_i \in \mathcal{C}}{\operatorname{argmin}} \sum_{i,j} \underset{\text{value}}{(A_{ij} - B_i^T g_j)^2}$$

② $B = \left[\begin{array}{c} \leftarrow B_i \rightarrow \\ \vdots \end{array} \right]; \quad C = \left[\begin{array}{c} \leftarrow G_j \rightarrow \\ \vdots \end{array} \right]$

Third step is matrix completion:

③ Matrix Completion

$$A \in \left[\begin{array}{cccc} \text{empty} & \square & \square & \square \\ \square & \text{empty} & \square & \square \\ \square & \square & \text{empty} & \square \end{array} \right]$$

$A_{10,5} = \text{empty}$

$$\hat{A} = \underbrace{B * C^T}_{\text{mostly empty}}$$

$$\hat{A}_{10,5} = \underbrace{\overline{B_{10}} * \overline{C_5}}_{\text{mostly empty}}$$

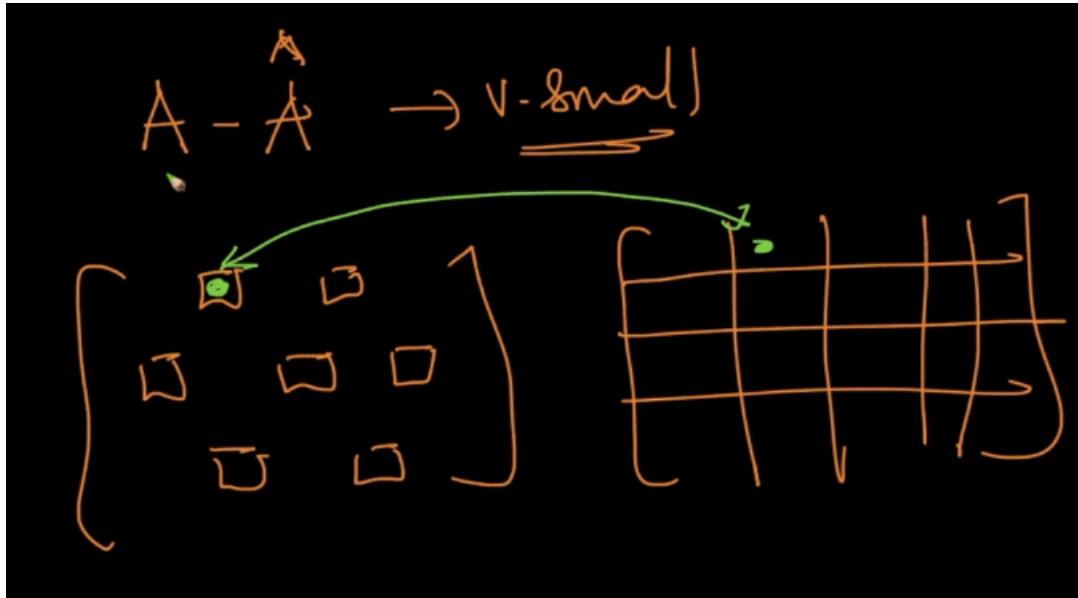
$$\hat{A} = \left[\quad \right] \rightarrow \text{mostly empty}$$

\hat{A} = $B * C^T$

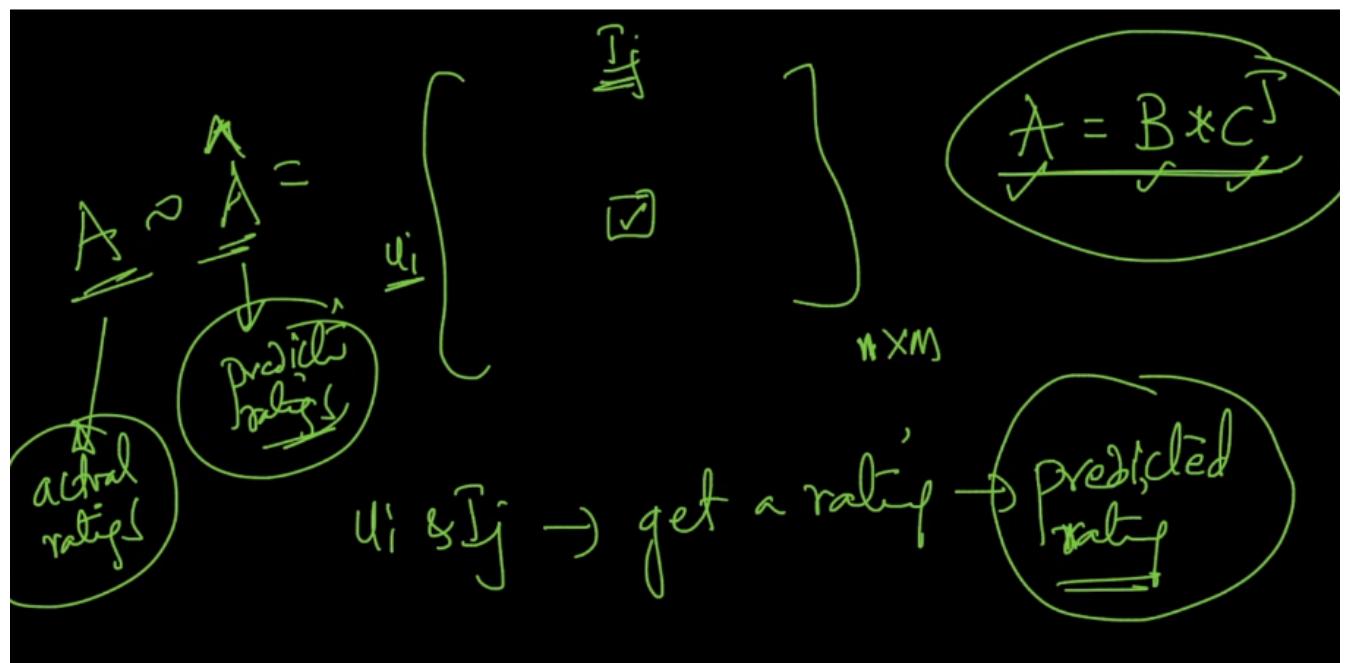
no empty cells

computed using A_{ij} that are not empty by solving the optim. problem

We have to ensure that the real and absolute values are very similar.



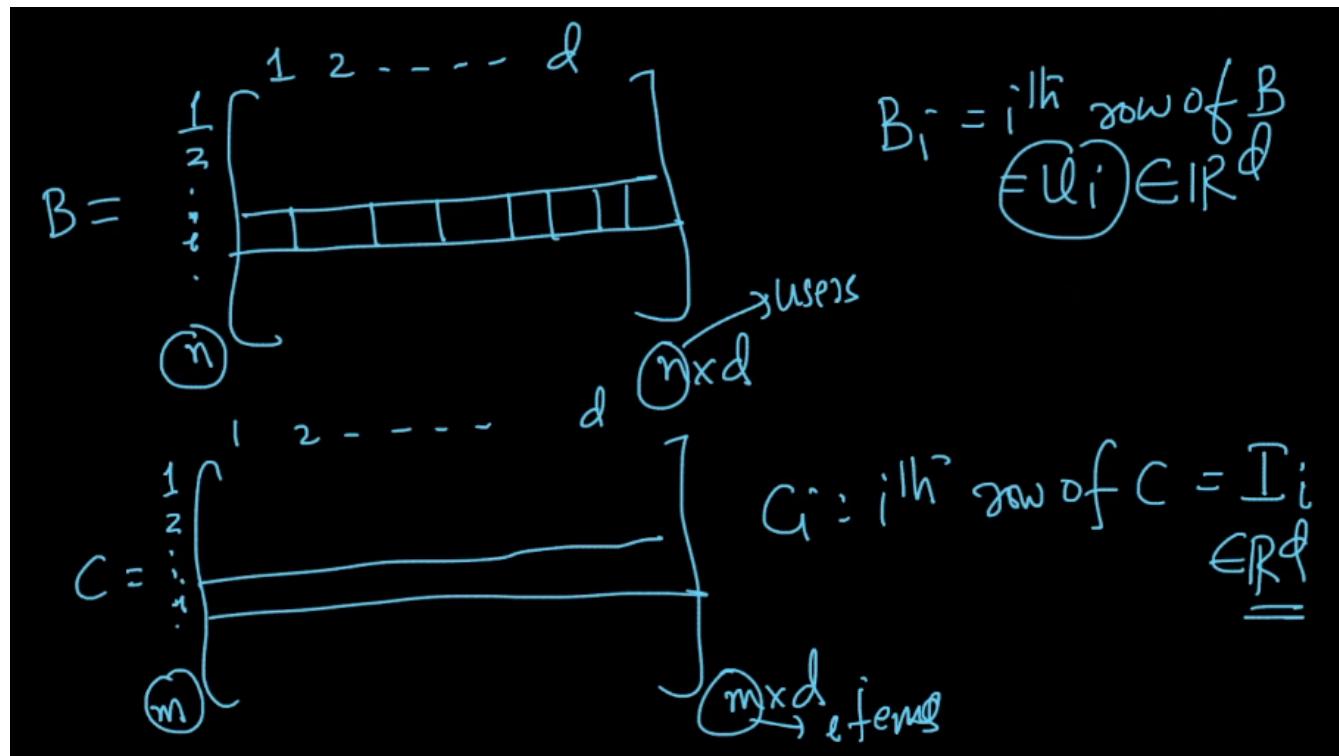
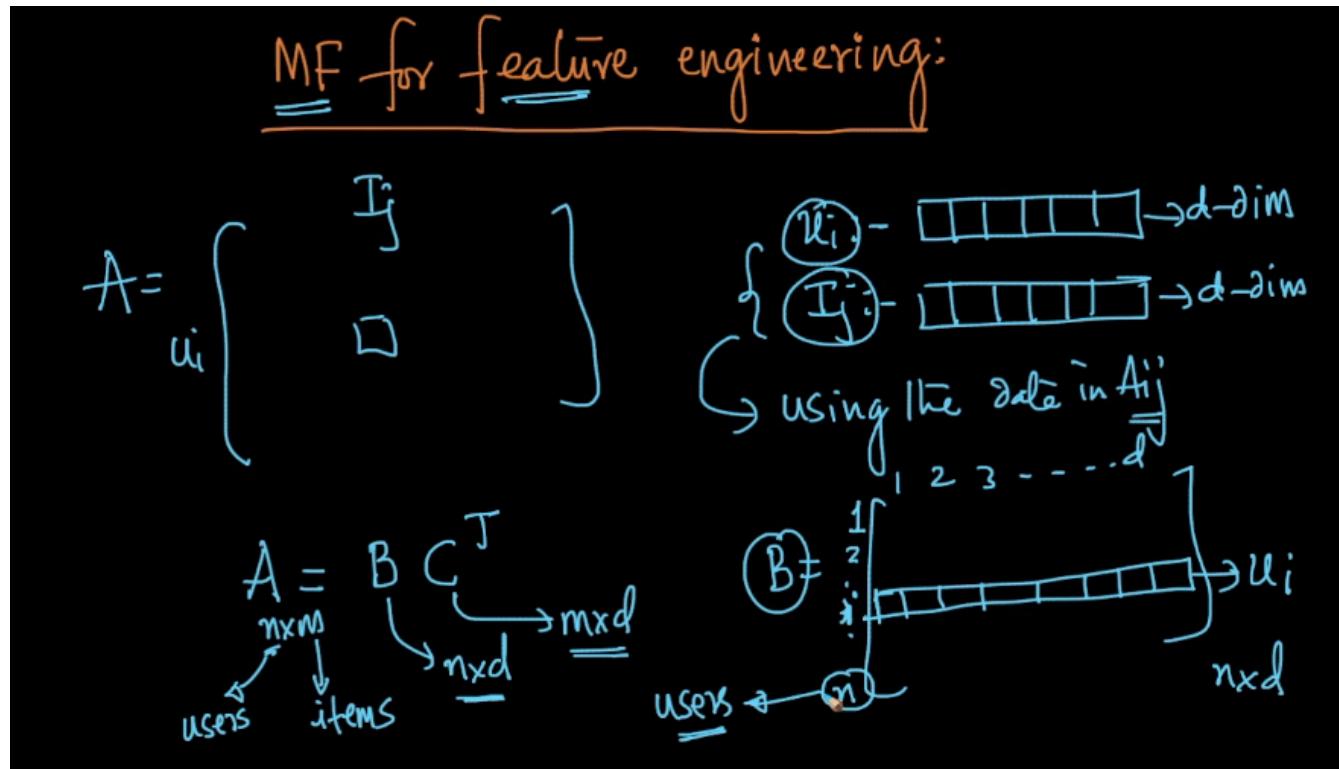
\hat{A}^{\wedge} consists of predicted rating.



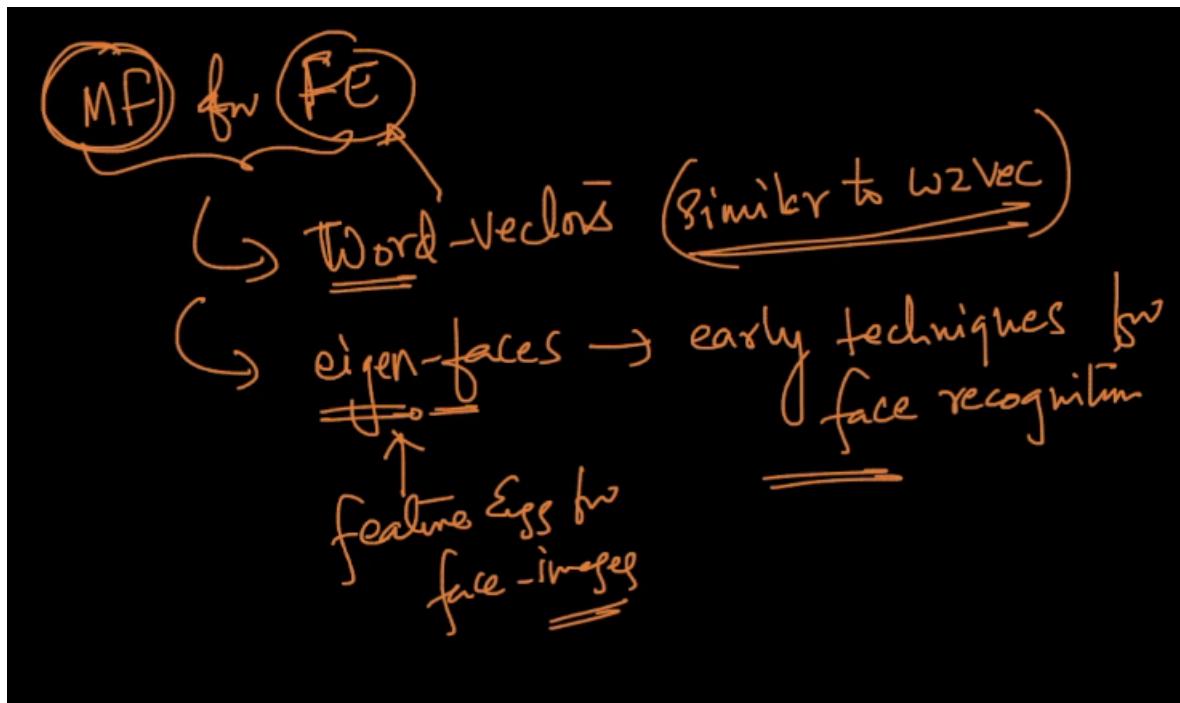
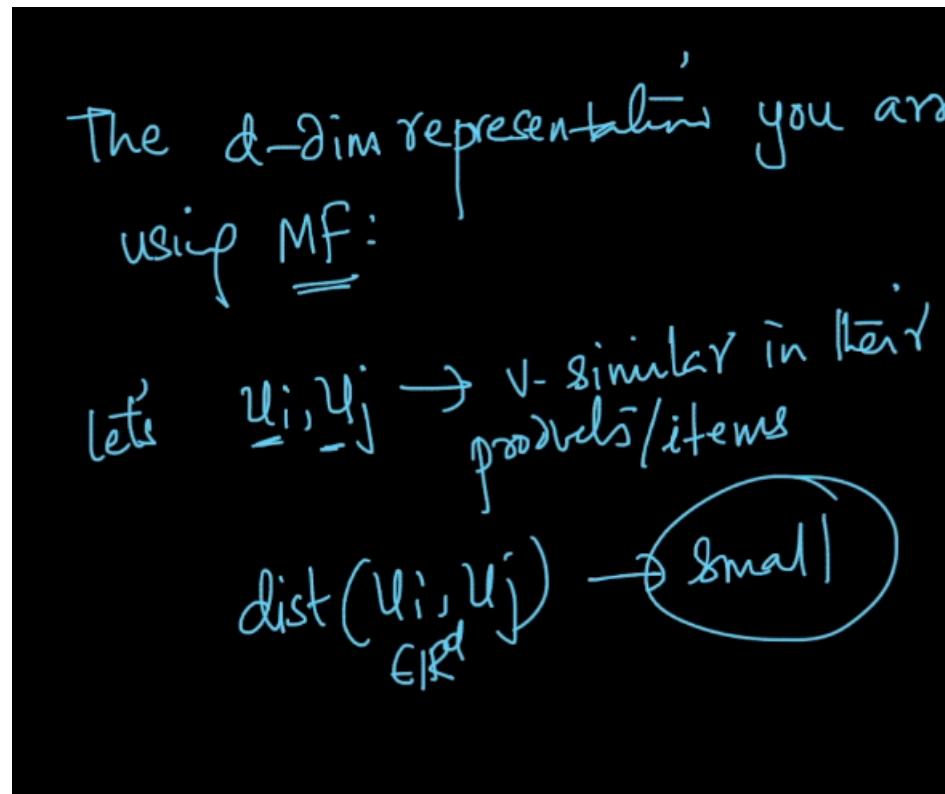
This is how we use matrix factorization.

Matrix factorization for feature engineering:

We want to create the vectors of users and items. The d-dim vectors for user and items.



The most important thing we got at MF:



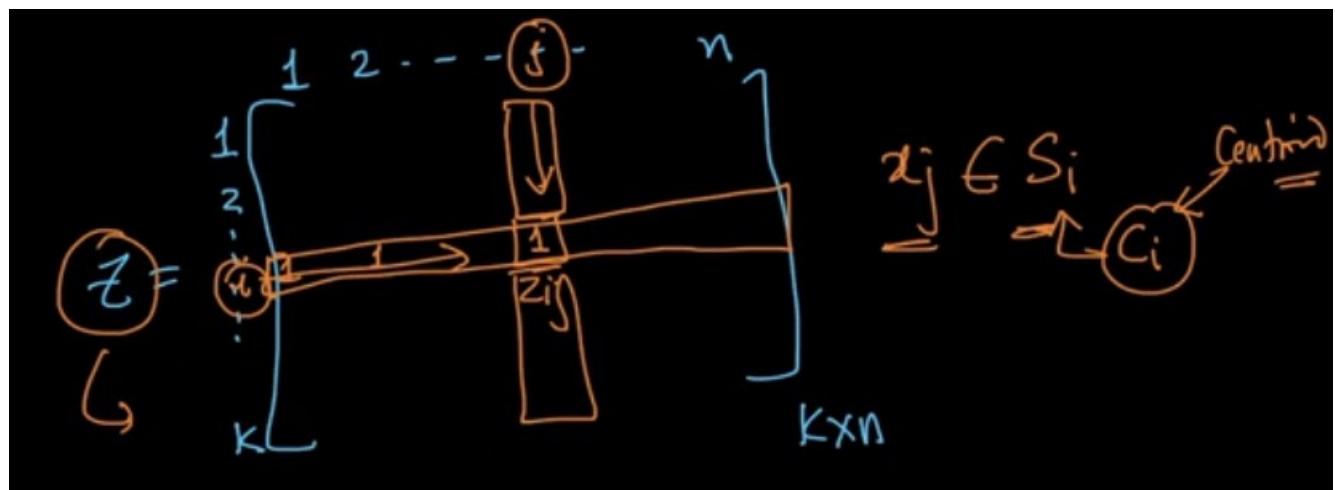
Clustering as MF:

MF and Clustering: $\Rightarrow k\text{-means}$

$k\text{-means}$: $D = \{x_i\}$; $\{S_1, S_2, S_3, \dots, S_k\} \xrightarrow{\text{task}}$ $\{C_1, C_2, C_3, \dots, C_k\}$

$$\sqrt{\min_{c_i} \sum_{i=1}^k \sum_{x \in S_i} \|x - c_i\|^2} \quad x \in \mathbb{R}^d$$

Let's define z s.t $z_{ij} = \begin{cases} 1 & \text{if } x_j \in S_i \\ 0 & \text{o/w} \end{cases}$



$$\min_{c_i} \sum_{i=1}^K \left(\sum_{x_j \in S_i} \|x - c_i\|^2 \right)$$

\downarrow

$$\min_{c_i, z_{ij}} \sum_{i=1}^K \left(\sum_{x_j \in S_i} z_{ij} \|x_j - c_i\|^2 \right)$$

$x_j \in S_i$

$(z_{ij}) = \begin{cases} 1 & \text{if } x_j \in S_i \\ 0 & \text{otherwise} \end{cases}$

$$X = \begin{pmatrix} \uparrow & \uparrow & \uparrow & \uparrow \\ x_1 & x_2 & \dots & x_n \\ \downarrow & \downarrow & \dots & \downarrow \end{pmatrix}_{d \times n}$$

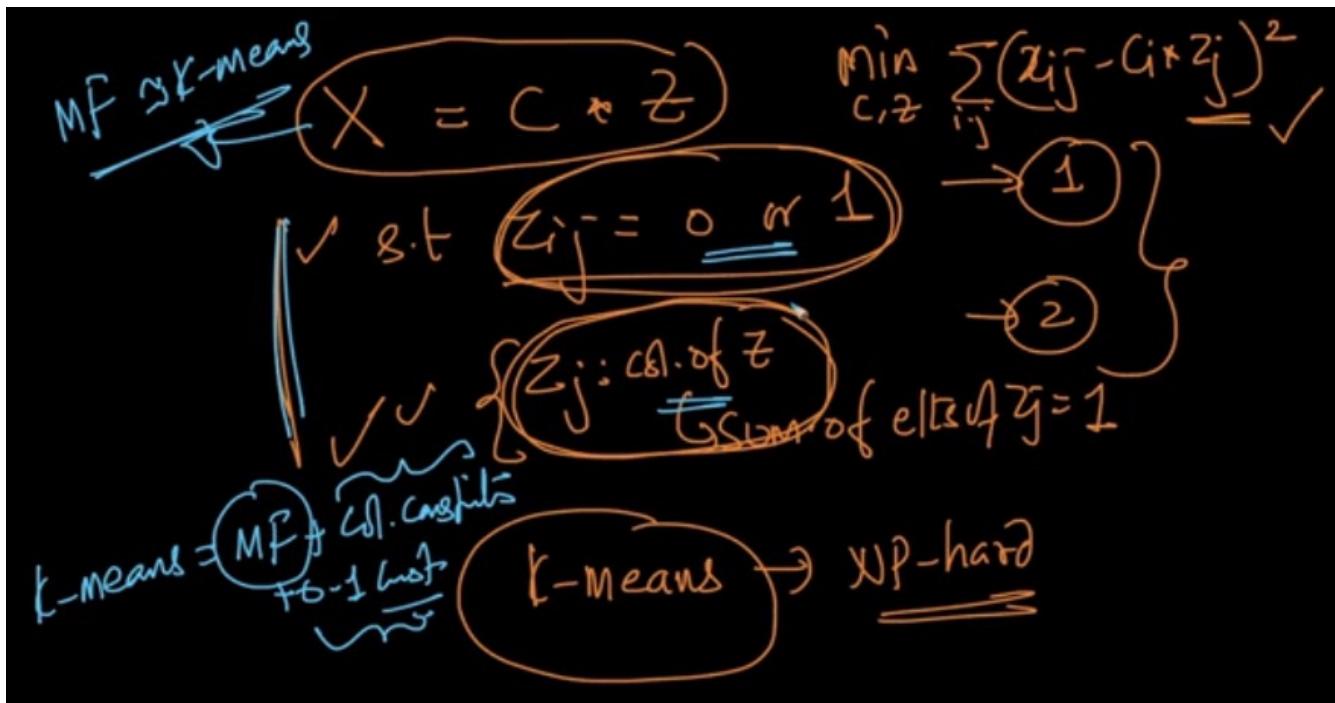
$$C = \begin{pmatrix} \overset{c_1}{\uparrow} & \overset{c_2}{\uparrow} & \dots & \overset{c_k}{\uparrow} \\ C_1 & C_2 & \dots & C_k \\ \downarrow & \downarrow & \dots & \downarrow \end{pmatrix}_{d \times k}$$

$\Rightarrow z_{ij} = \begin{cases} 1 & \text{if } x_j \in S_i \\ 0 & \text{otherwise} \end{cases}$

The initial problem we have is that minimize the distance of the points to the centroid.

$$\begin{aligned}
 & \min_{\underline{z}_{ij}} \sum_{i=1}^k \sum_{j=1}^n \|x_j - c_i\|_2^2 \\
 & \quad \text{subject to } z_{ij} = \begin{cases} 1 & \text{if } x_j \in C_i \\ 0 & \text{otherwise} \end{cases} \\
 & \quad \text{and } \sum_{j=1}^n z_{ij} = 1 \\
 & \quad \text{and } \sum_{i=1}^k c_i = \underline{x} \\
 & \quad \text{and } c_i \geq 0 \quad \forall i
 \end{aligned}$$

$$\begin{aligned}
 & \min_{\underline{c}, \underline{z}} \sum_{i=1}^k \|x - c_i z\|_2^2 \\
 & \quad \text{subject to } z_{ij} = \begin{cases} 1 & \text{if } x_j \in C_i \\ 0 & \text{otherwise} \end{cases} \\
 & \quad \text{and } \sum_{j=1}^n z_{ij} = 1 \quad \forall i
 \end{aligned}$$

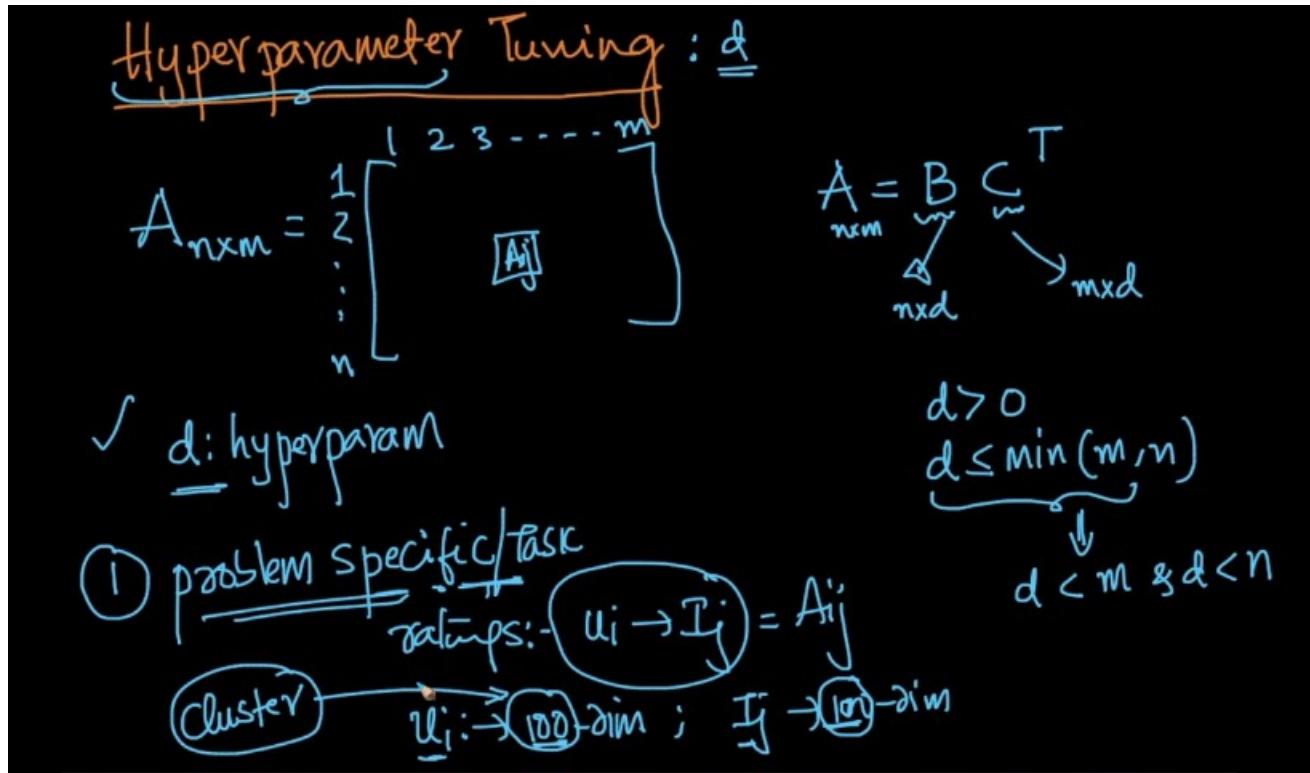


$$\left\{
 \begin{array}{l}
 \text{MF: } C \otimes Z \\
 \text{NMF: } z_{ij} \geq 0; c_{ij} \geq 0 \\
 \text{k-means: } z_{ij} = 0 \text{ or } 1 \\
 \text{col-sum of } z_j = 1
 \end{array}
 \right.$$

Hyper parameter tuning:

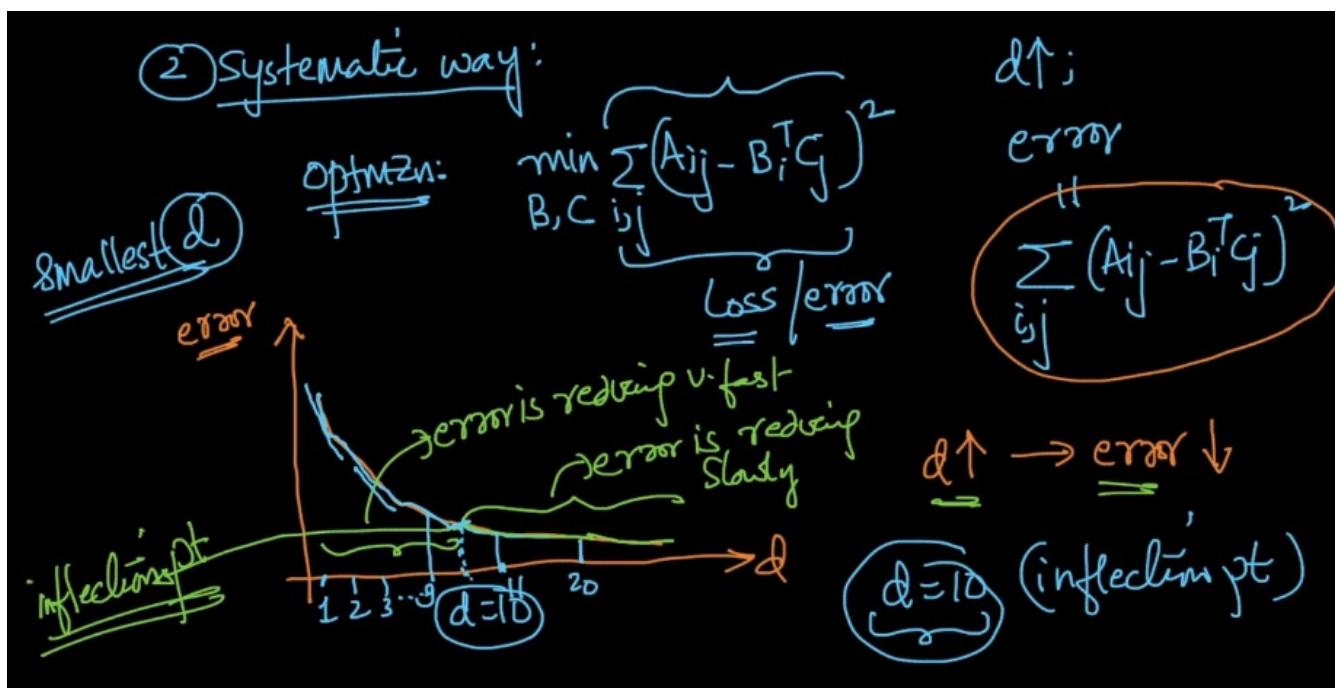
The d should be greater than zero.

Determining right D:

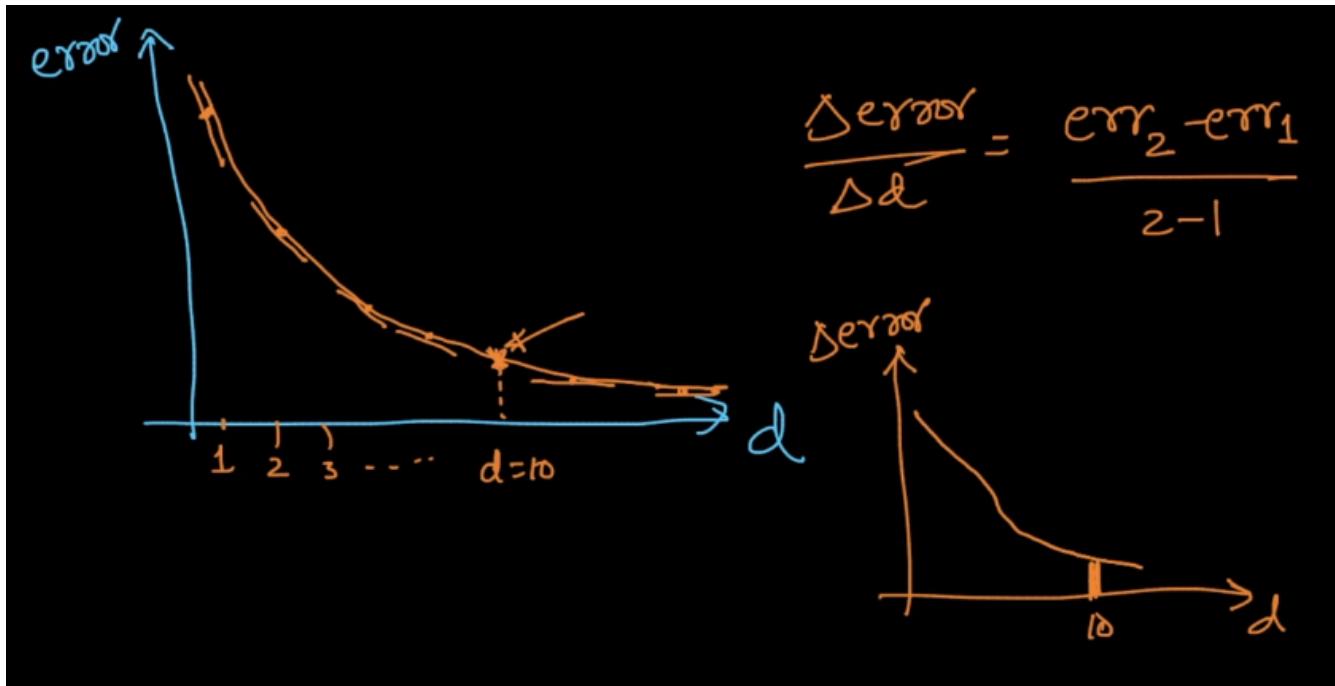


But as usual we have systematic way for finding best 'd':

The Error is reducing very fast at the beginning and slow at the ending. This point is called inflection point where the rate of change of error is less decreasing.



We will take the gradient plot in this case for choosing the right K.
 We will find the inflection point by looking at visually the plot.



Netflix prize:

That streams video on demand. They gave the loss metric – root mean square error.

A_{ij} → \hat{y}_{ui} → rating u on i

q_i → item vector

p_u → user vector

sqr. loss

$$① \min_{q, p} \sum_{(u, i)} \left(\hat{y}_{ui} - q_i^T p_u \right)^2 + \lambda \left(\|q\|_2^2 + \|p\|_2^2 \right)$$

\downarrow

$(A_{ij} - B_i^T g_j)^2$

L_2 -reg

\downarrow

to avoid overfitting

In netfilx MF we use the regularization.

$$\min_{P, Q} \left\{ \sum_{u, i} (r_{ui} - q_i^T p_u)^2 + \lambda (||q_i||^2 + ||p_u||^2) \right\}$$

solve:

(1) SGD: $\frac{\partial L}{\partial p_u}, \frac{\partial L}{\partial q_i} \rightarrow$ takes more time

(2) Alternating least squares (ALS):

(a) fix p_u ; gradient descent to find q_i 's

(b) fix q_i ; gradient descent to find p_u 's

faster algo

We use the alternating least squares as the approach for optimizing the loss function.

bias $\rightarrow b_u, b_i \rightarrow$ scalars \Rightarrow mean-term

$$\min_{Q, P, b} \sum_{u, i} (r_{ui} - \mu - b_u - b_i - p_u^T q_i) + \lambda \left\{ ||q_i||^2 + ||p_u||^2 + b_u^2 + b_i^2 \right\}$$

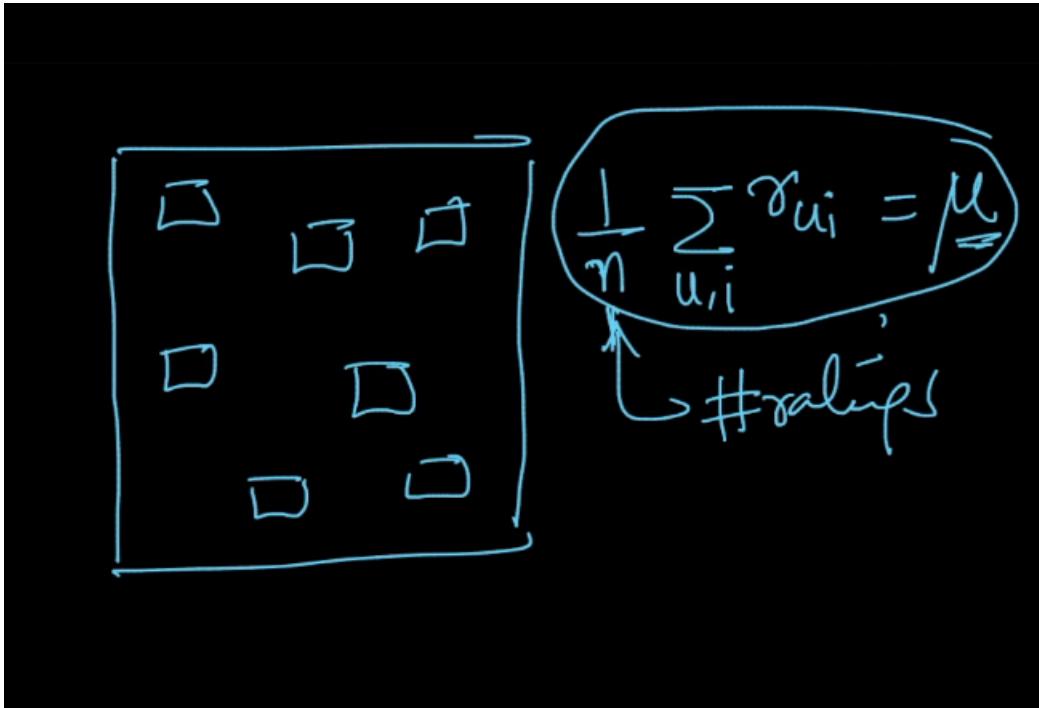
avg. rating across all users & items ecosystem

user bias (+ve, -ve)

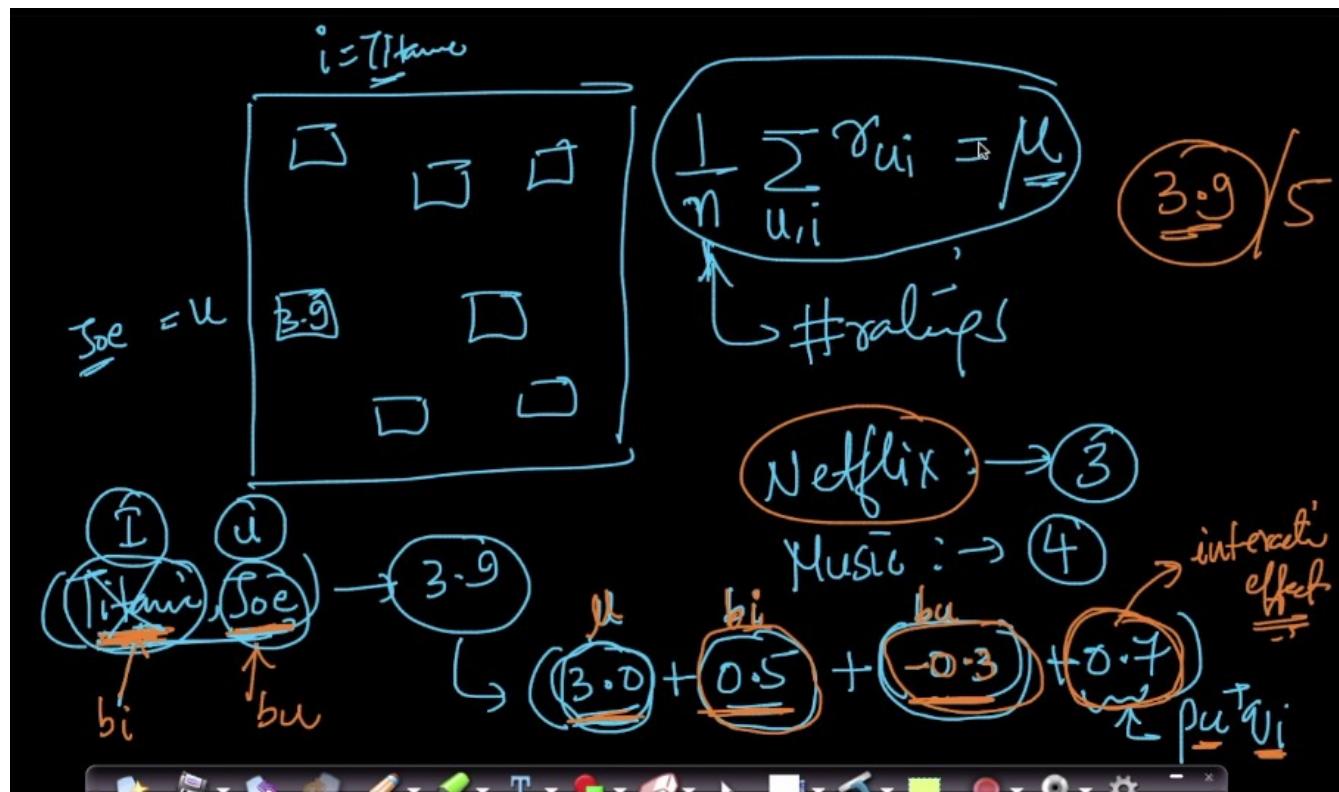
item-bias

interaction

The U:



There is no point in determining the average rating, titanic movie is rated as follows.



$$\begin{aligned}
 2) \quad & \min_{\mathbf{u}, \mathbf{p}, b} \sum_{u_i} (\hat{r}_{ui} - \underbrace{\mu}_{\text{bias}} - \underbrace{b_u}_{\text{item bias}} - \underbrace{b_i}_{\text{user bias}} - \mathbf{p}^T \mathbf{v}_i)^2 + \lambda \left\{ \|\mathbf{v}_i\|^2 + \|\mathbf{p}\|^2 + \frac{b_u^2}{2} + \frac{b_i^2}{2} \right\} \\
 1) \quad & \min_{\mathbf{u}, \mathbf{p}} \sum_{u_i} (\hat{r}_{ui} - \mathbf{p}^T \mathbf{v}_i)^2 + \lambda \left\{ \|\mathbf{v}_i\|^2 + \|\mathbf{p}\|^2 \right\} \\
 \text{items-item} \quad & \text{Big-advantages} \rightarrow \text{MF} \\
 \text{Optimization} \quad & \text{Optimization} \\
 \text{Optimization} \quad & \text{MF optimization}
 \end{aligned}$$

We can add the custom terms into the loss function.

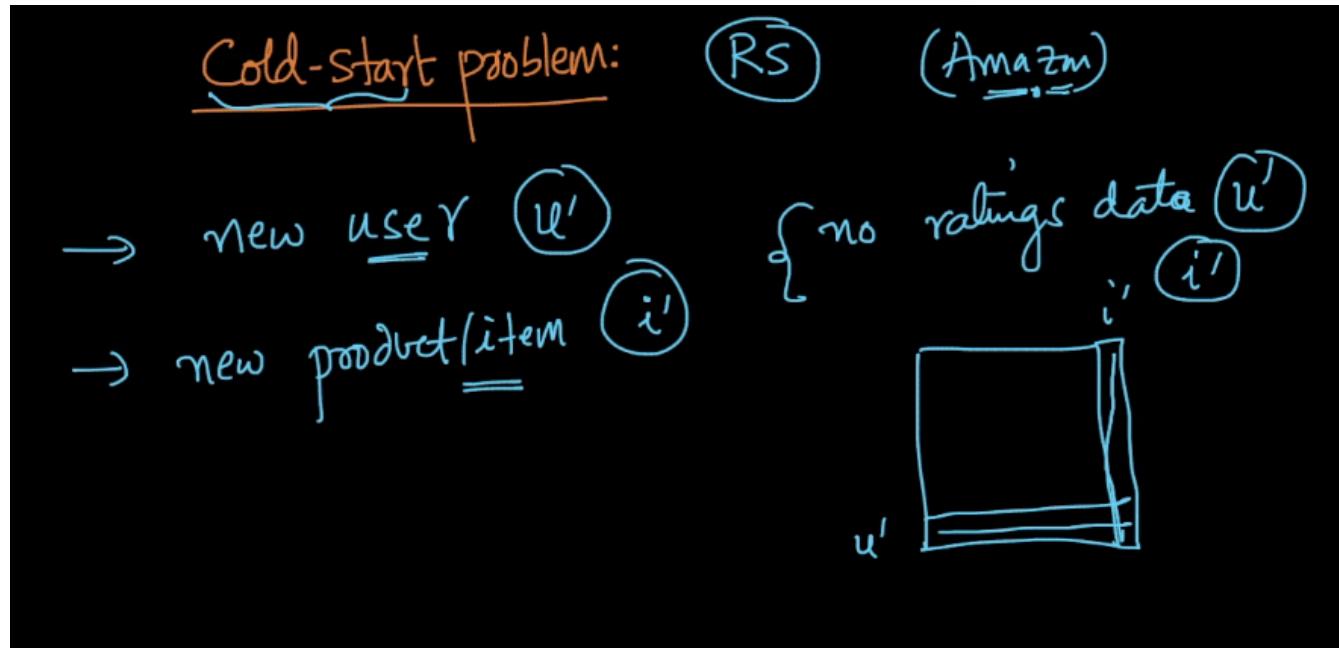
Research-paper:

ratings by users & for items are time dependent

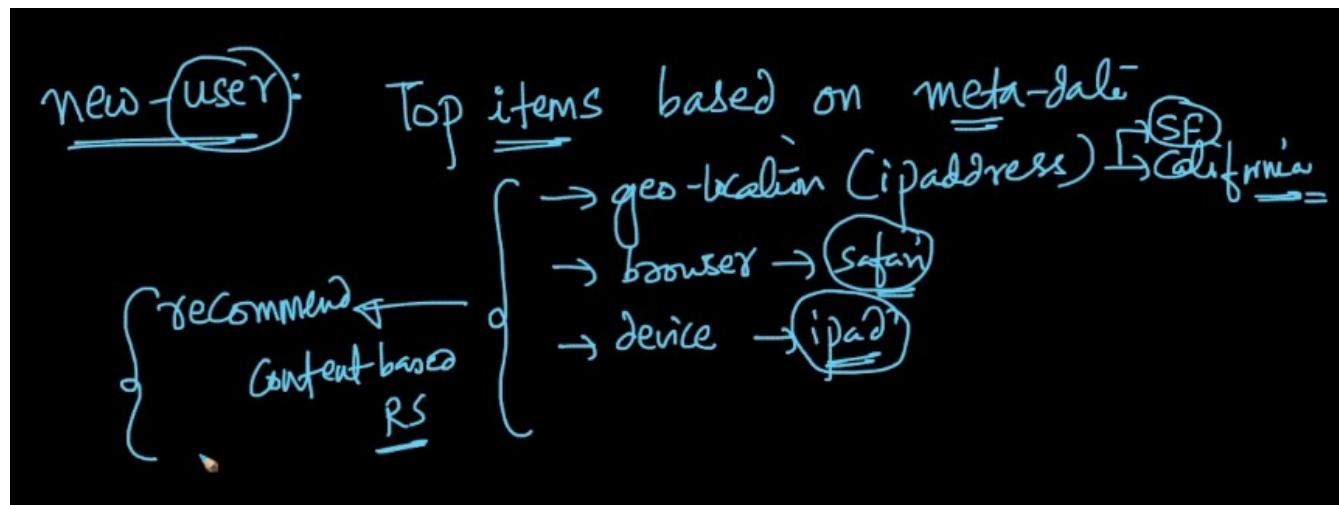
$\checkmark b_u(t)$ $\checkmark p_u$
 $\checkmark b_i(t)$ $\checkmark r_{ui}(t)$

Cold problem:

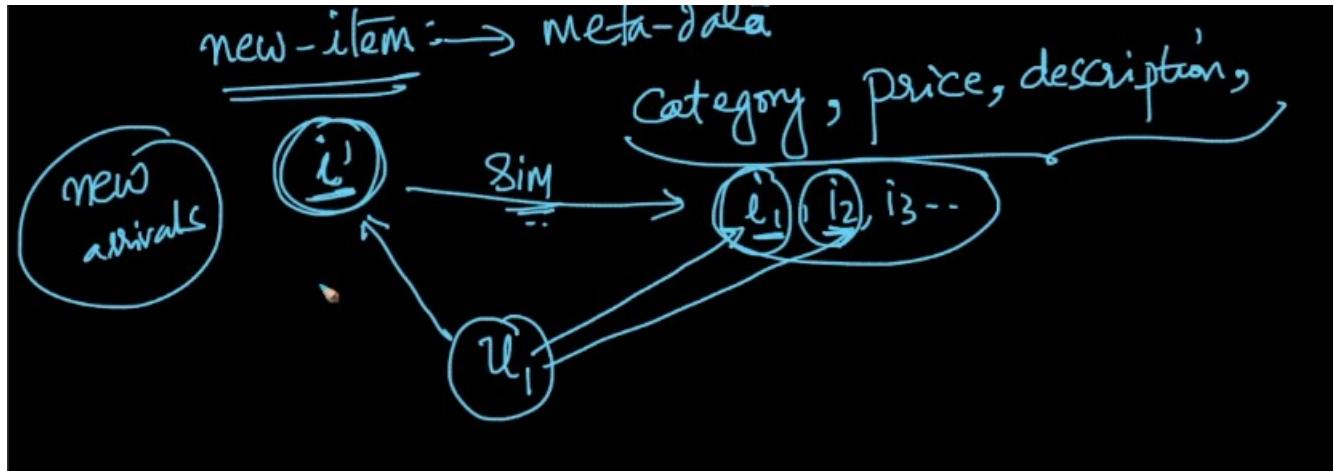
When a new user (or) item comes into ecosystem.



The practical way of doing is as follows;

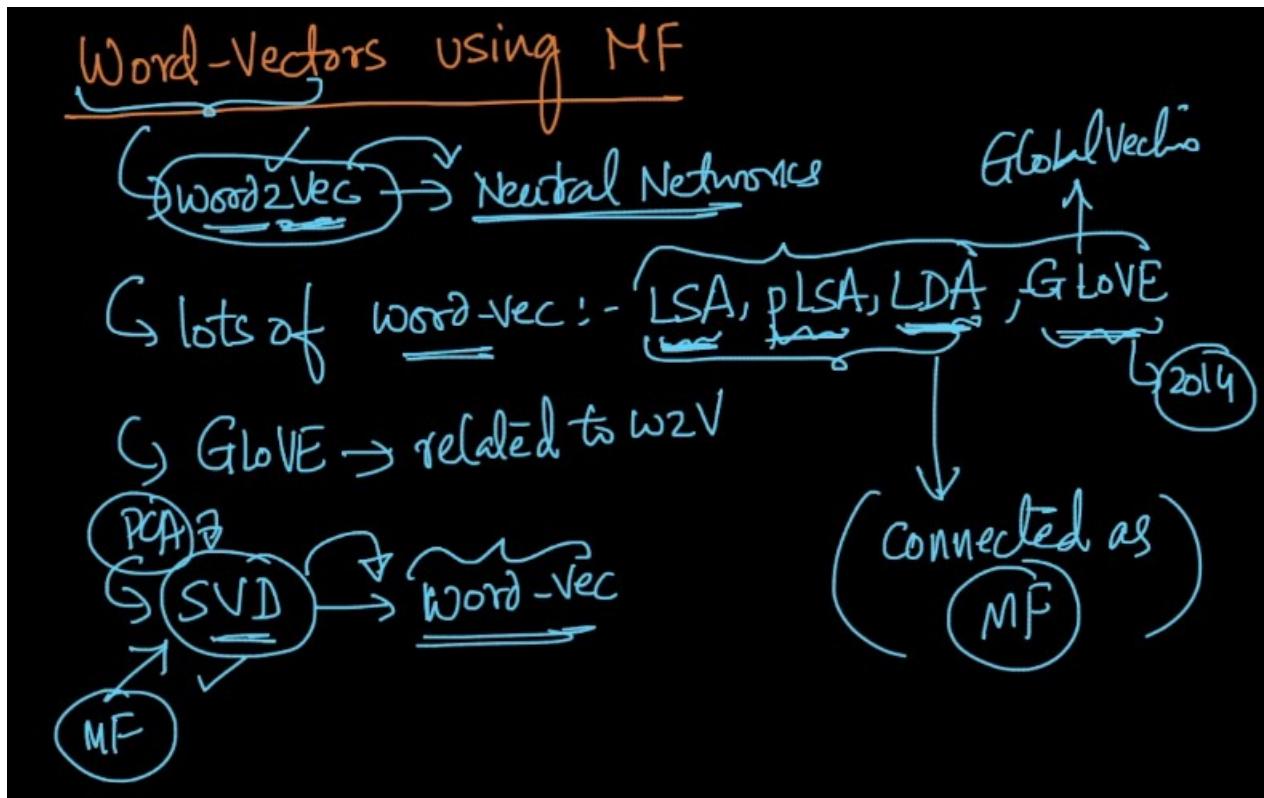


Similarity if we have the new item then meta-data -
 Category, Price, description.
 I1, i2, i3, i4.

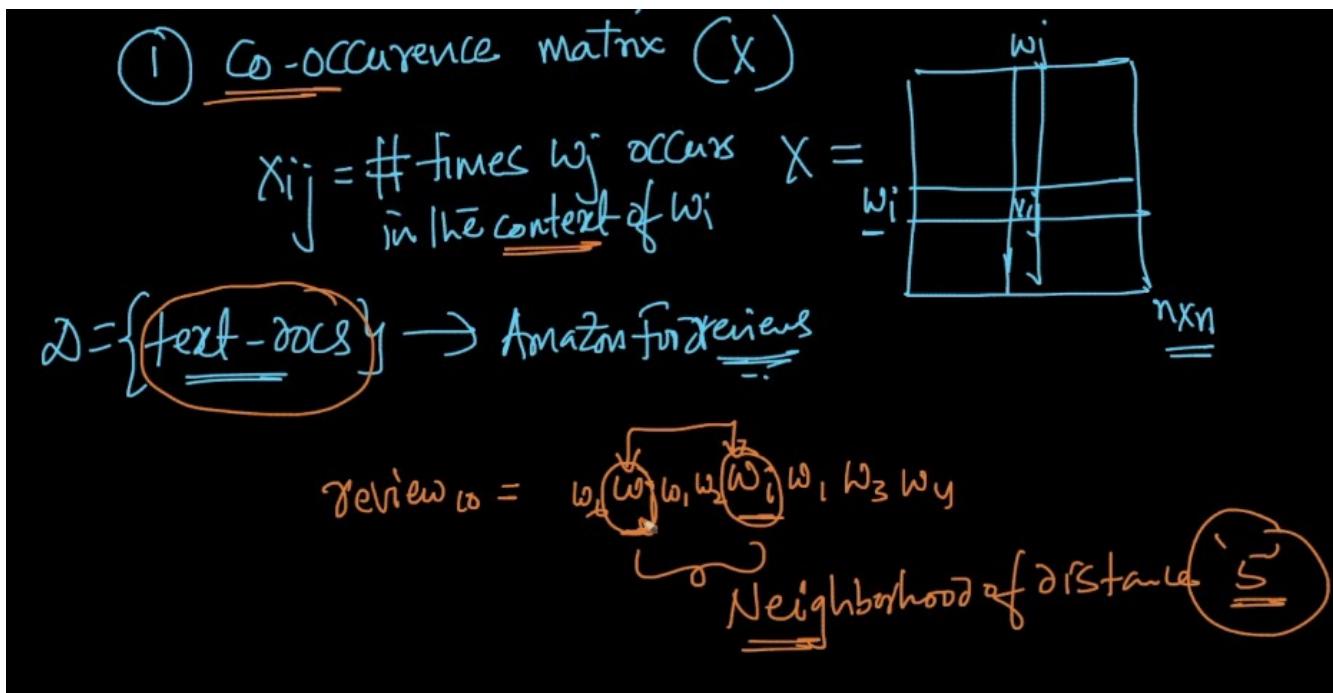


This problem is called cold start item.(content based approach).

Word vectors using MF:



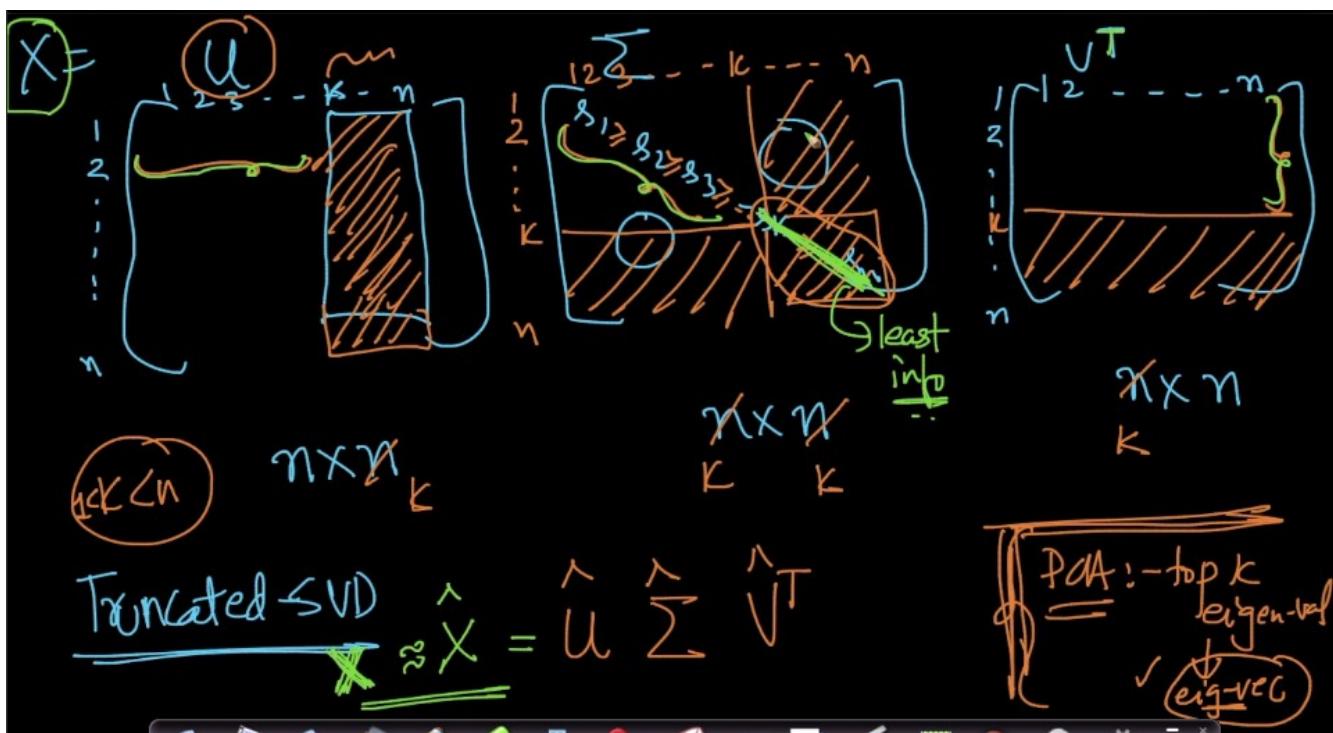
Co-occurrence matrix:



These two words are co-occurring across all the docs.

Truncated SVD:

What we discarded here is the least information



Truncated SVD(k)

$$X_{n \times n} \xrightarrow{\text{hyper-param}} \hat{U}_{n \times k} \hat{\Sigma}_{k \times k} \hat{V}^T_{k \times n} = \hat{X} \approx X$$

u_i = word-vec.
corresponds to
word $_i$

$$\begin{matrix} u_1 \mapsto 1 \\ u_2 \mapsto 2 \\ \vdots \\ u_n \mapsto n \end{matrix} \left[\begin{matrix} u_1 & u_2 & \dots & u_k \end{matrix} \right]_{n \times k}$$

③

\hat{U} :- rows of \hat{U} as u_1, u_2, \dots, u_n

$$u_i \in \mathbb{R}^k$$

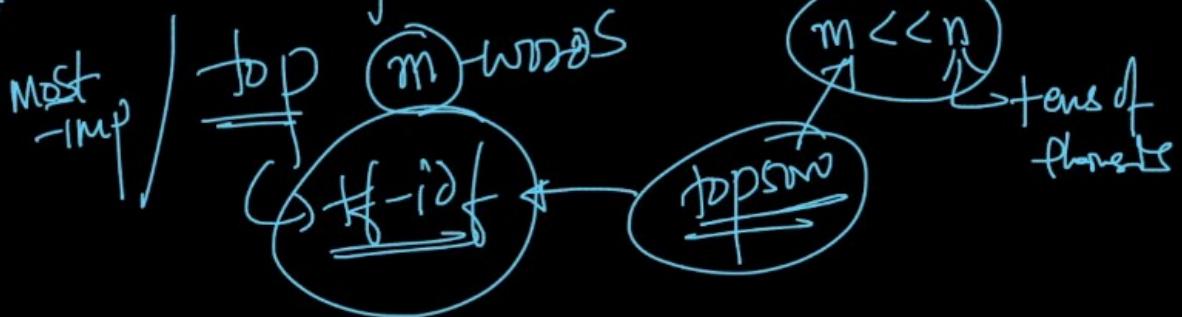
There is one problem here, coocurrence matrix + Truncated matrix.

The number of words is very large, computing is very difficult.

Instead of using all the words, lets use the top m-words by using the tf-idf.

Amazon food reviews:- # words \rightarrow v. large

Soln: instead of using all the n-words;



We could build the smaller matrix, on the subset of most important words.

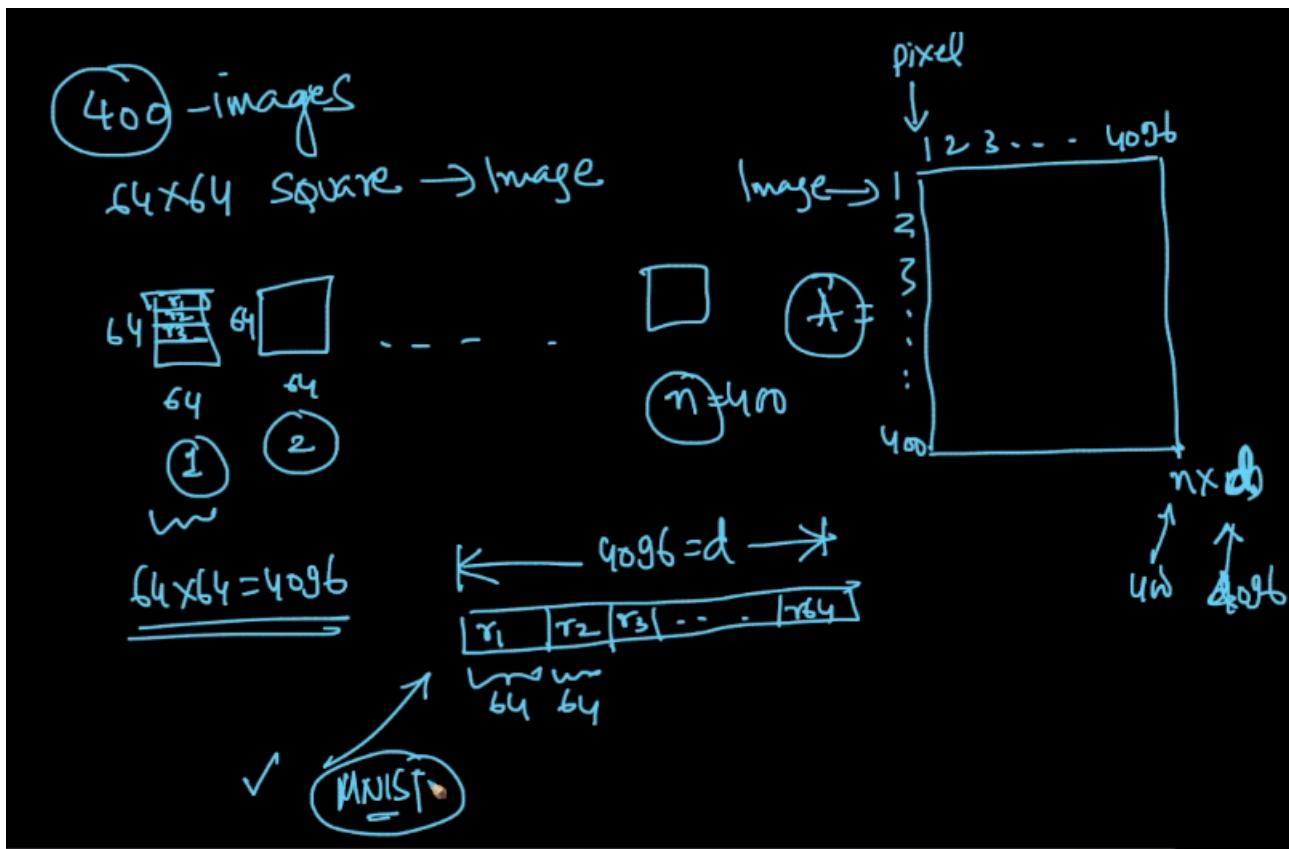
Eigen-Faces: For images we use the eigen faces is one of the early tech for face recognition.

Eigenfaces : PCA on images

\rightarrow Word vectors \rightarrow MF (SVD)
(co-occurrence)

\rightarrow Image data: \rightarrow Eigenfaces (PCA) \rightarrow CNN (Neural nets)
↳ early tech. for face recognition

\rightarrow MF (PCA) \rightarrow Image data



Given my matrix A, we can calculate A.

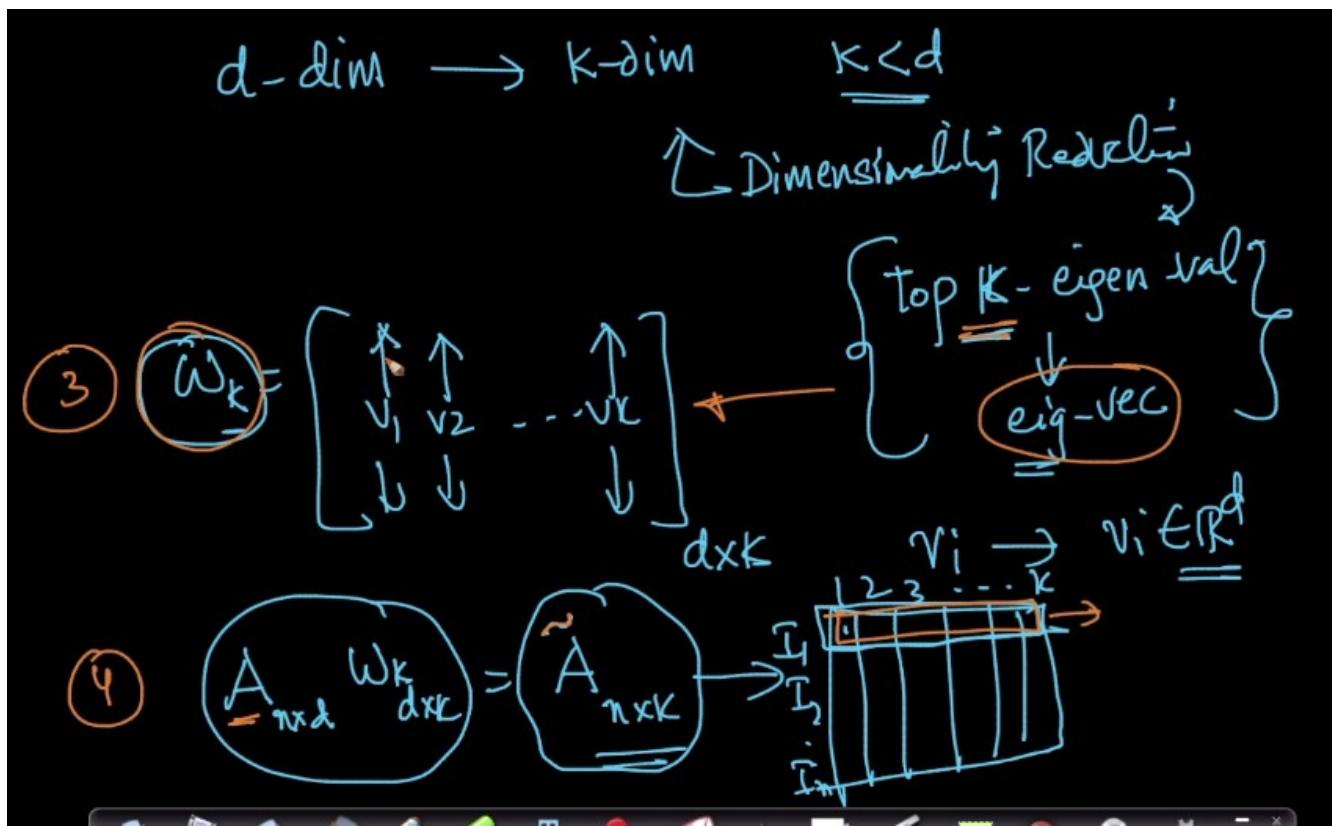
$$A_{n \times d} \rightarrow \text{cov}(A) = S_{d \times d}$$

$$S_{d \times d} = W_{d \times d} \Lambda_{d \times d} W_{d \times d}^T \quad \text{PCA}$$

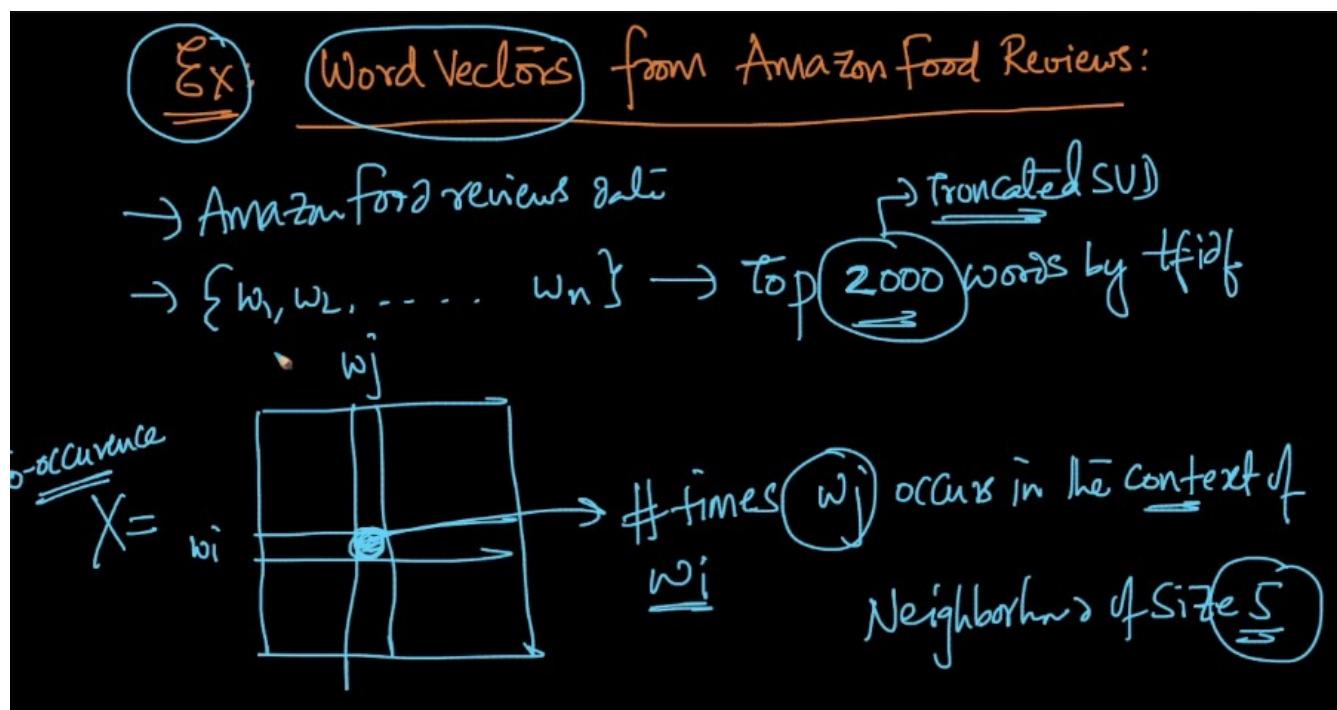
$$W = \left[\begin{matrix} v_1 & v_2 & \dots & v_d \end{matrix} \right]$$

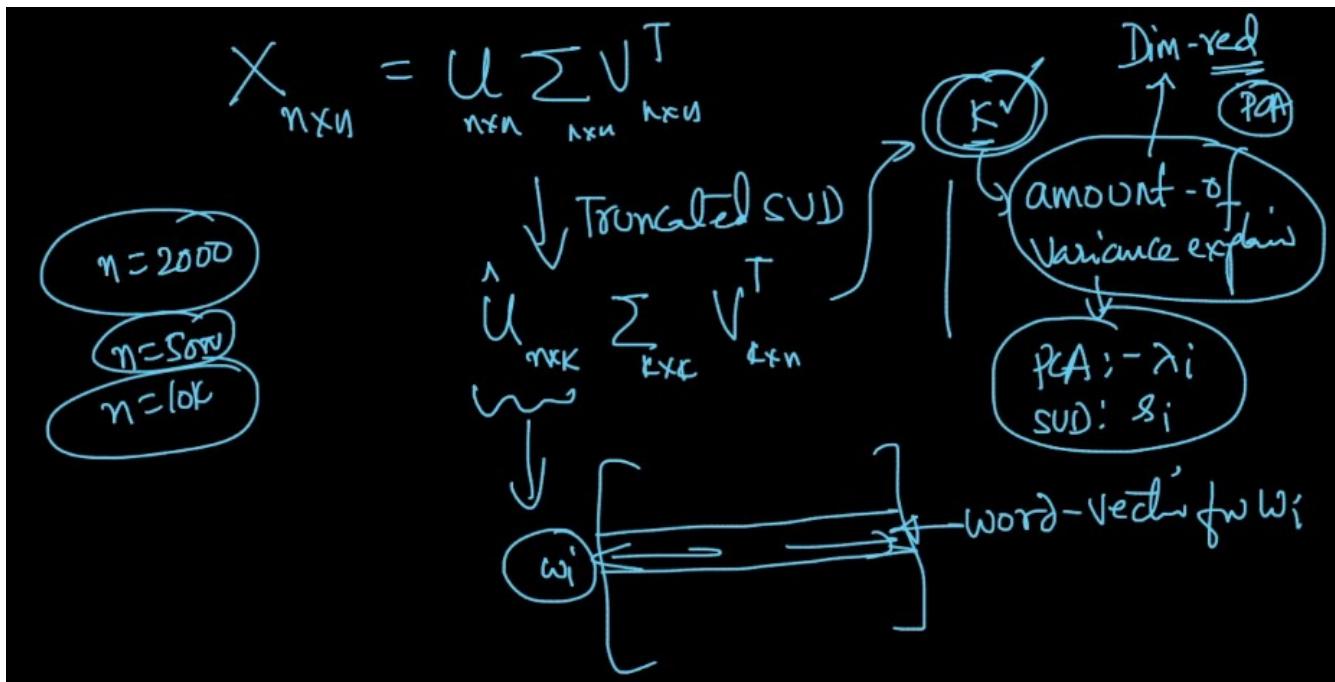
$$\Lambda = \begin{pmatrix} \lambda_1 & & & 0 \\ & \lambda_2 & & \\ & & \ddots & \\ 0 & & & \lambda_d \end{pmatrix}$$

$$\lambda_1 > \lambda_2 > \lambda_3 > \dots > \lambda_d$$



Each of the vectors v_i are the 'd' dimensional vector, $d=4096$.
 2000 words using tf-idf, coocurance matrix with neigh = 5 to w_i .





C-means Cluster the word-vectors for top n -words

S_i a

words clustered together \rightarrow semantically be related

Italian \rightarrow pasta pizza

Indian \rightarrow Spice Rotti Biryani