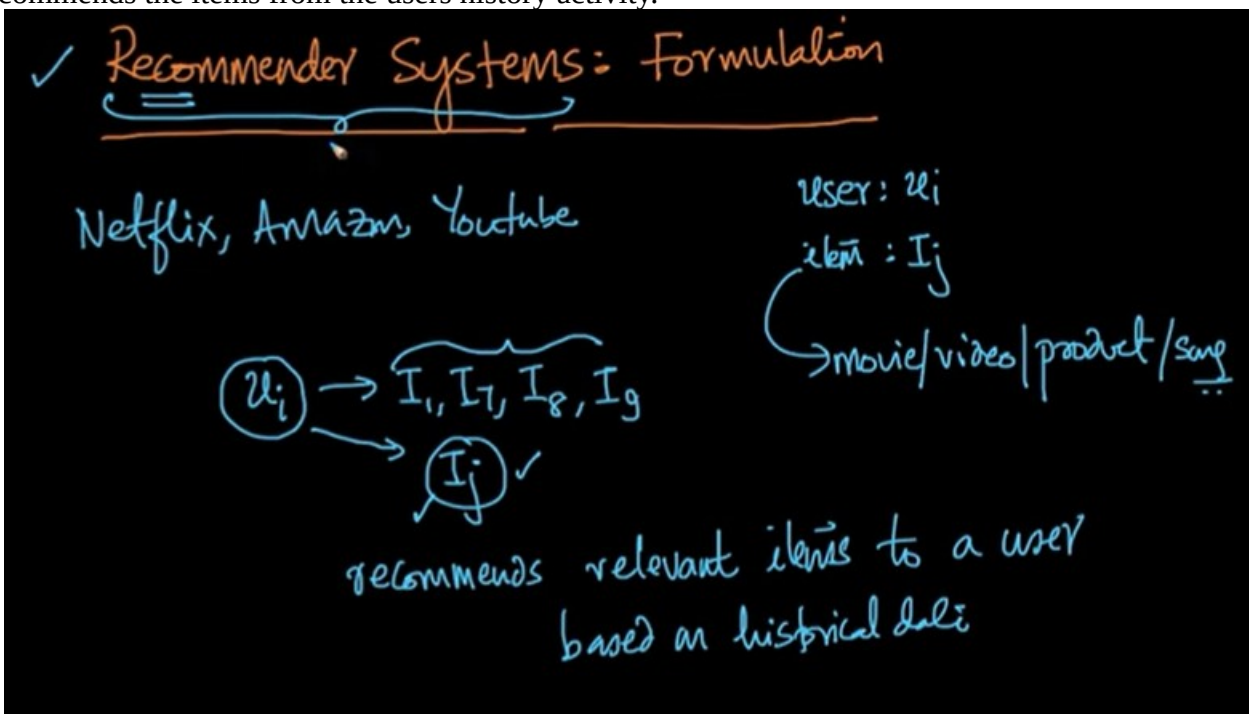
Recommender systems- matrix factorization

Recommends the items from the users history activity.

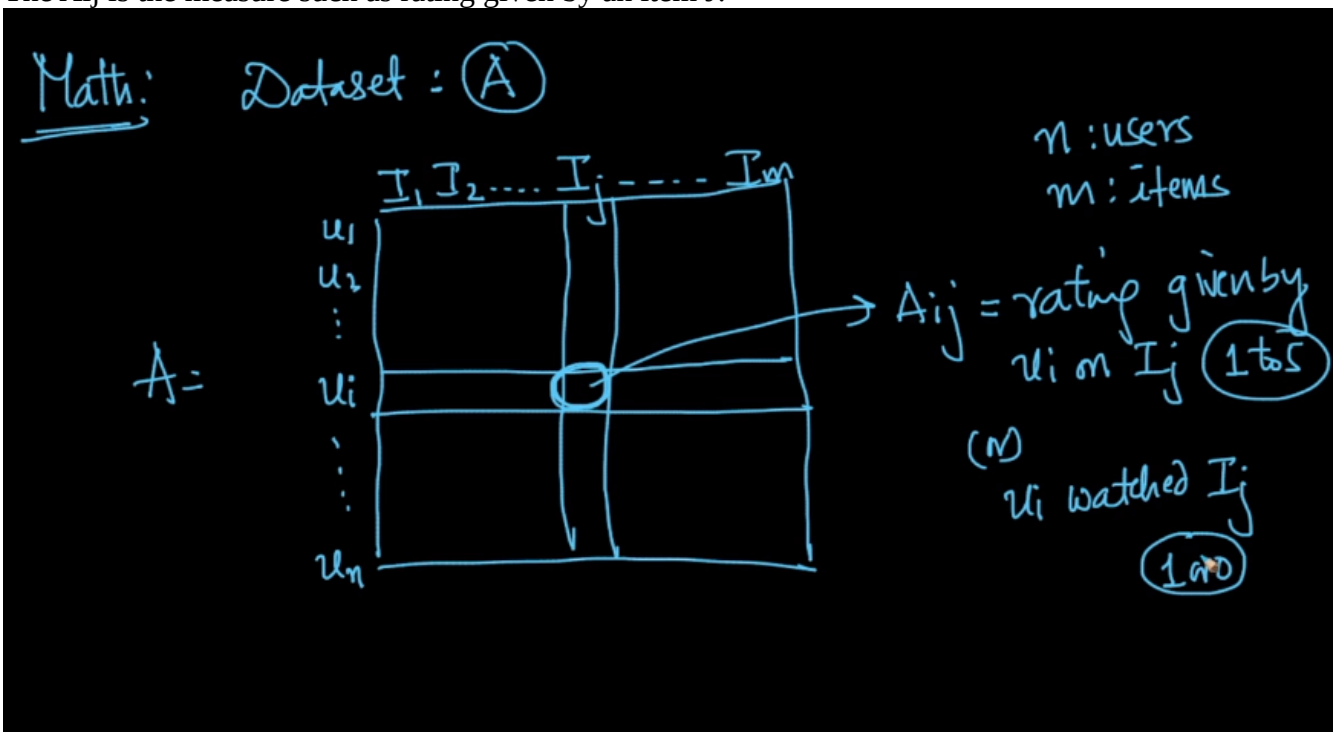


Mathematical form: It recommends relevent items to the user based on historical data the person liked.

The data set is in the matrix.

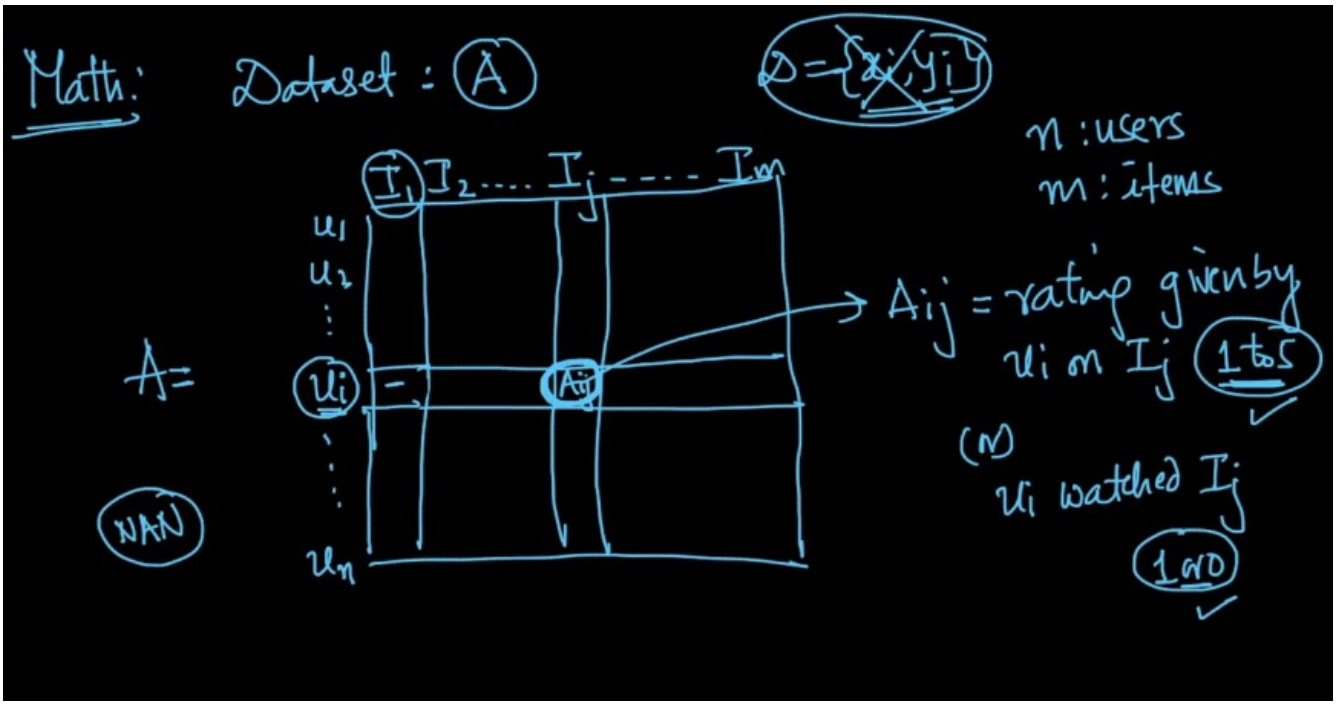
Each row in the matrix is a user, and each column in the matrix is an item(Movie, song).

The Aij is the measure such as rating given by an item J.

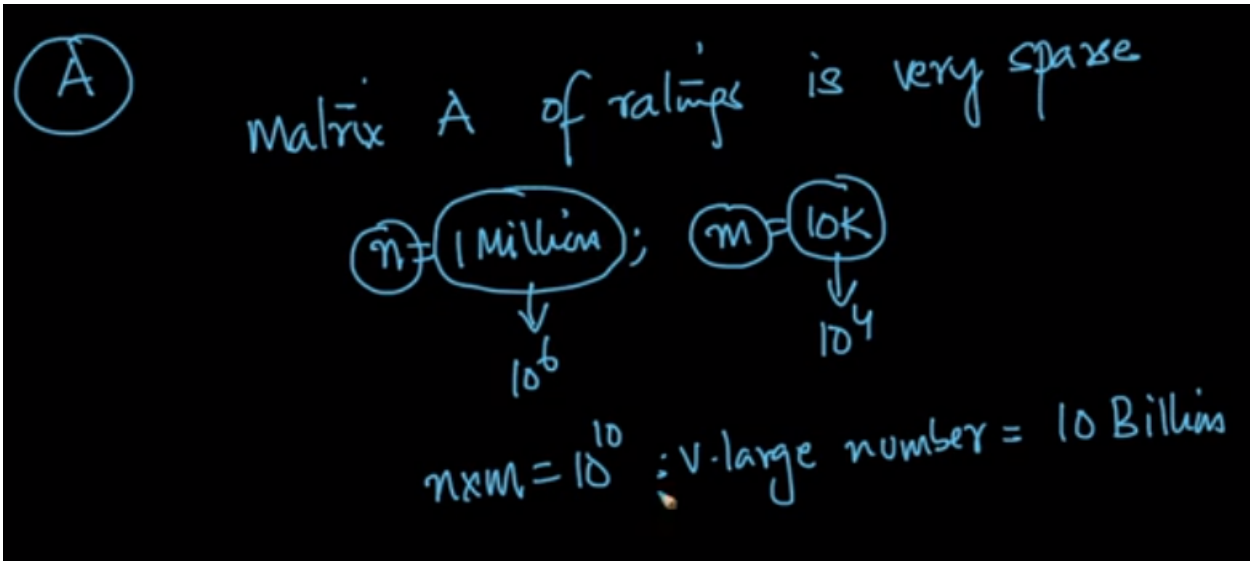


Consider the person not watched the movie. We can give one unique number like -1 for that space.

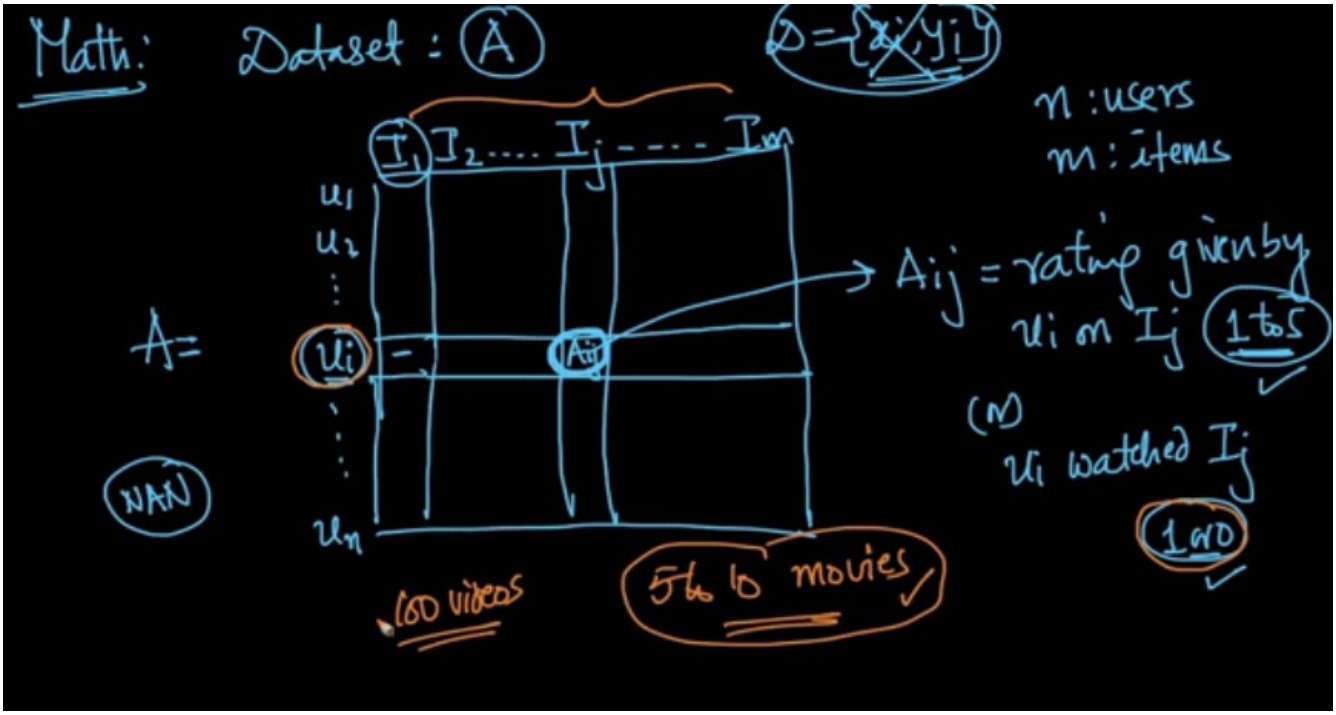
Where ever there is no data leave the cell as a blank. NaN is the not a number in Python.



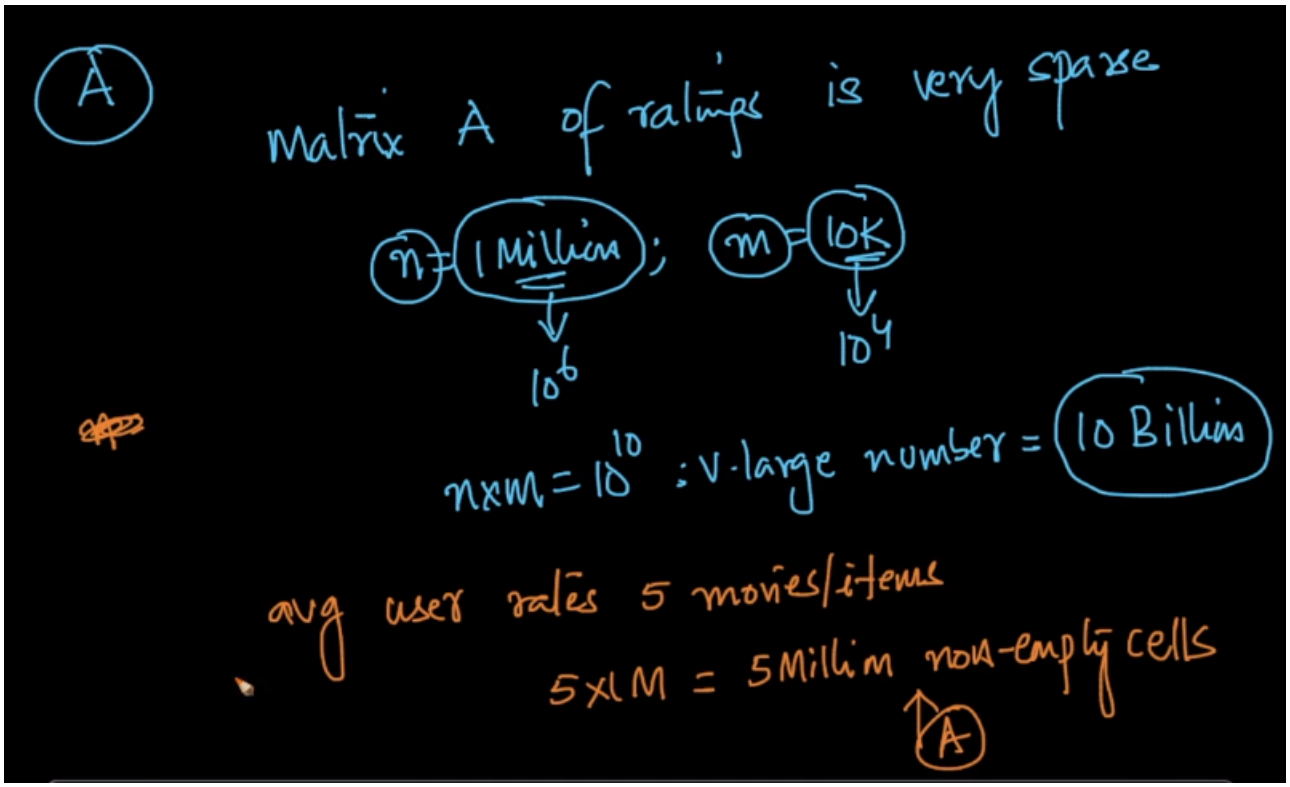
Property of the matrix:
The matrix can be very sparse.
Example:



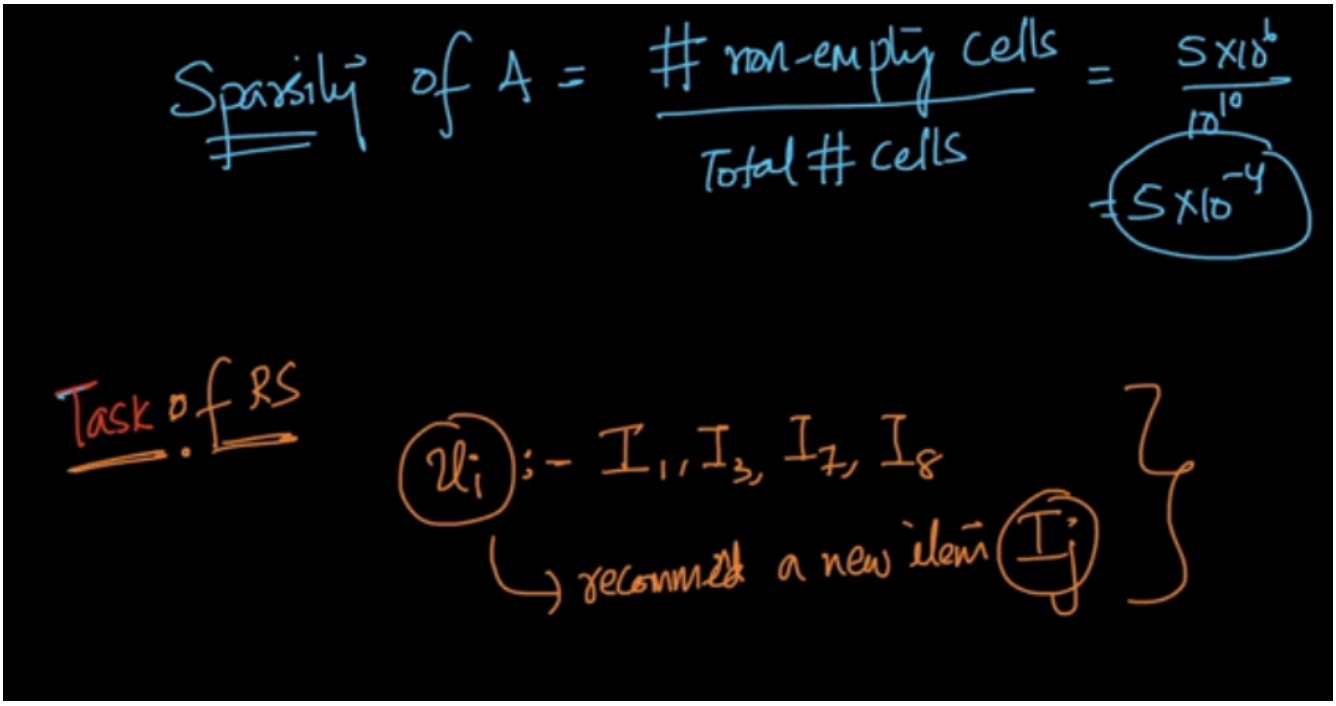
Even for 1 million users the matrix is very large, most of the matrix is sparse because a user can rate only 5 – 10 movies.



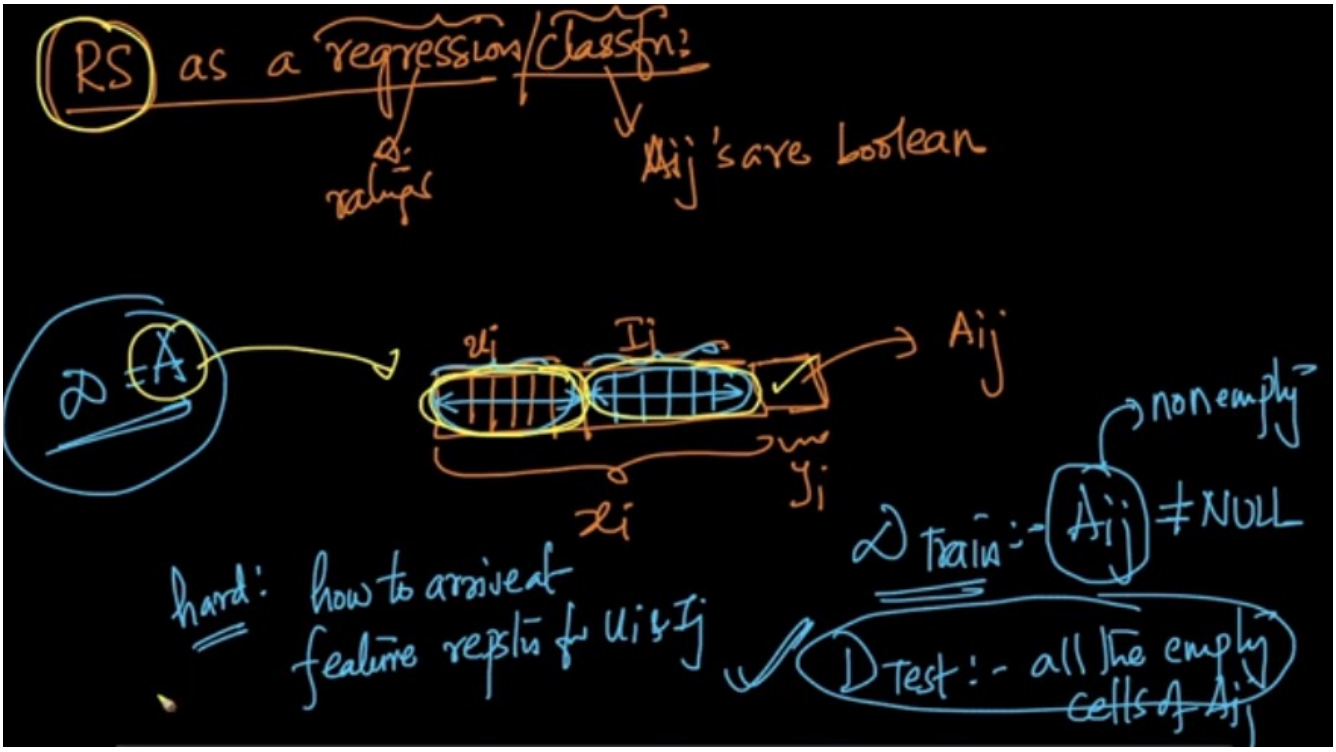
There will be very few movies of non empty entries.



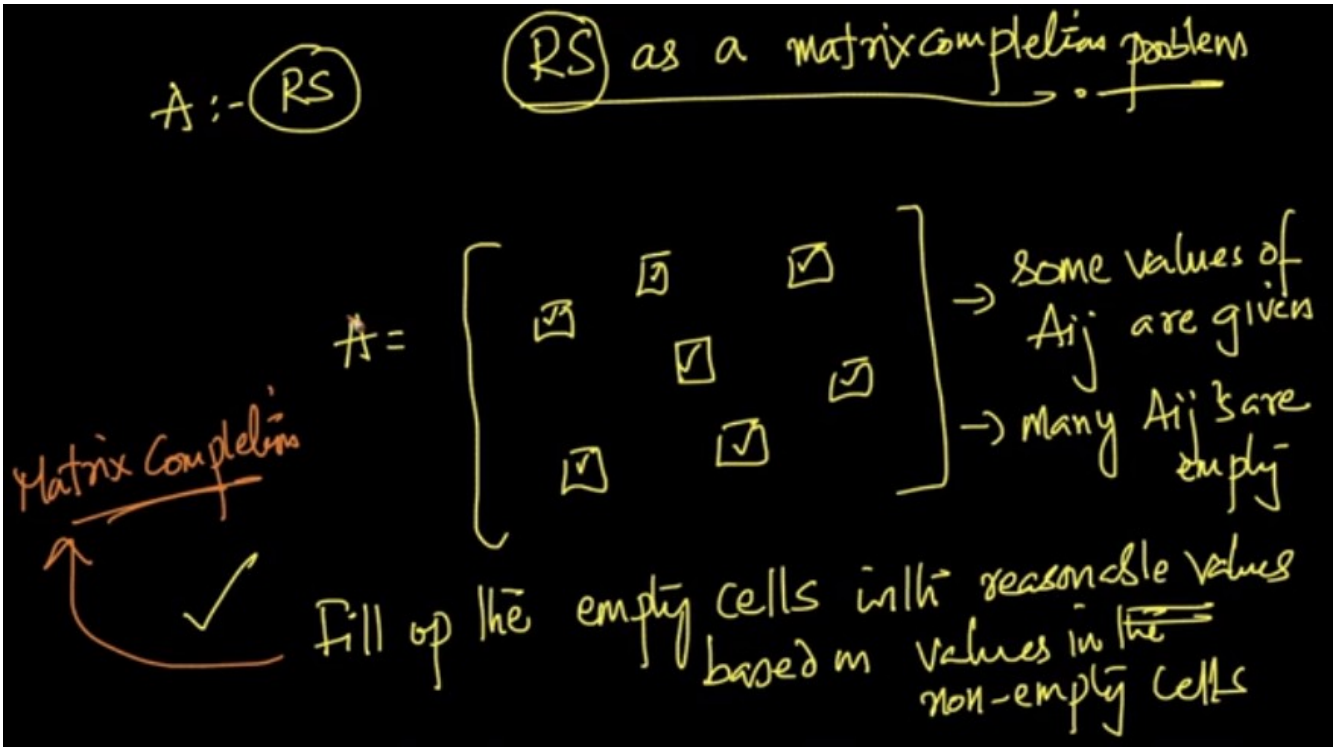
Calculation of sparsity of matrix:
The big task of building a recommend-er system is to recommend a new item the user .

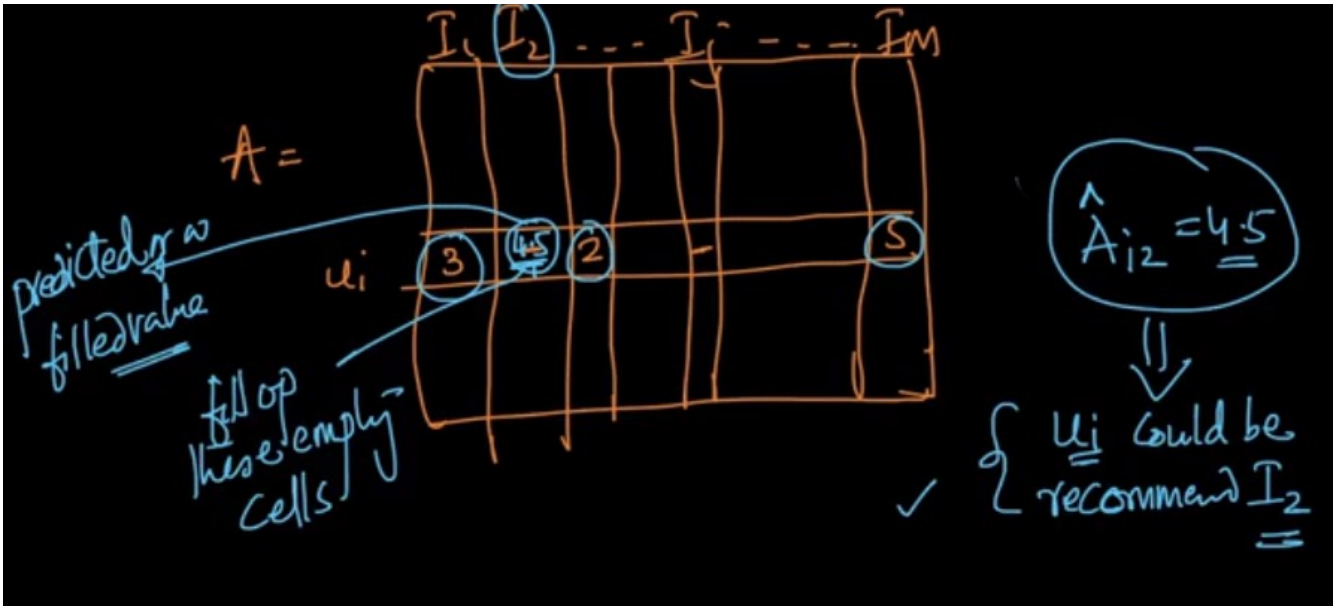


We can make the problem to make a regression / classification problem:



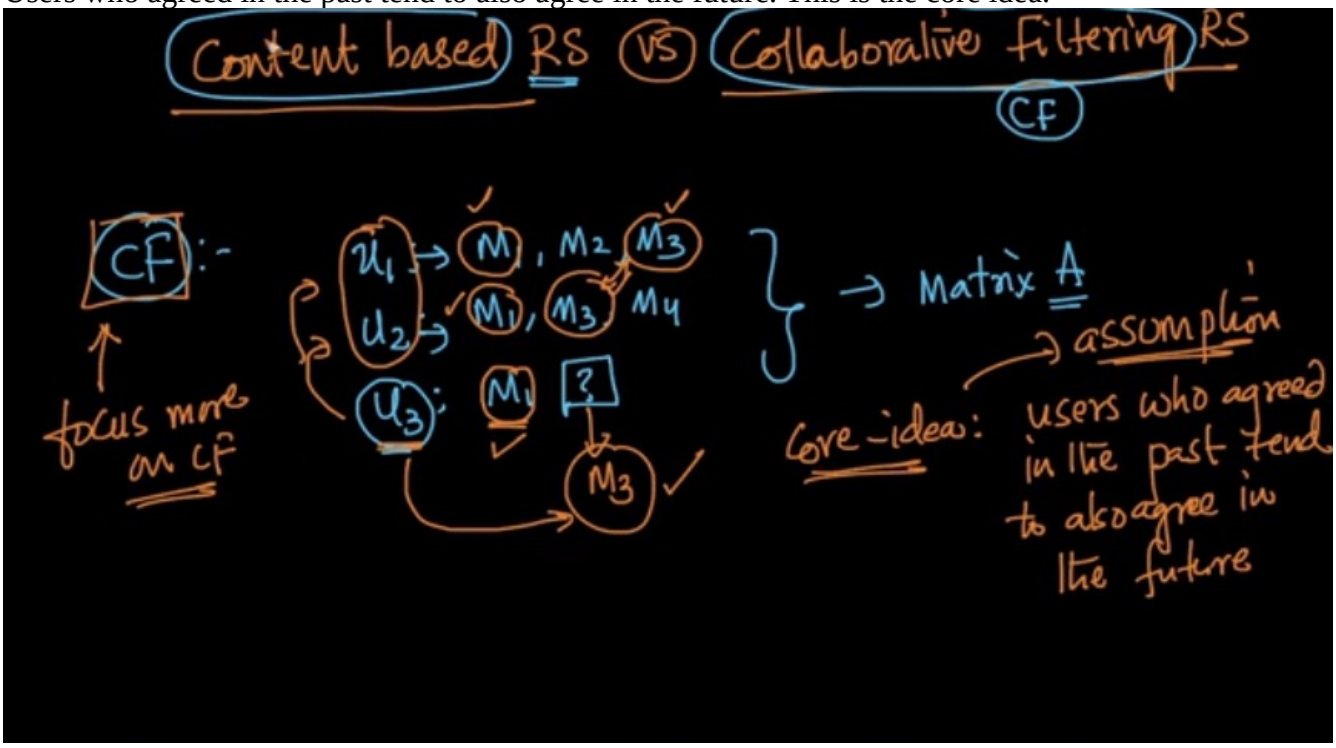
Fill the empty cells with the reasonable values, based on the non empty cells.
This is the problem of matrix fac completion value.



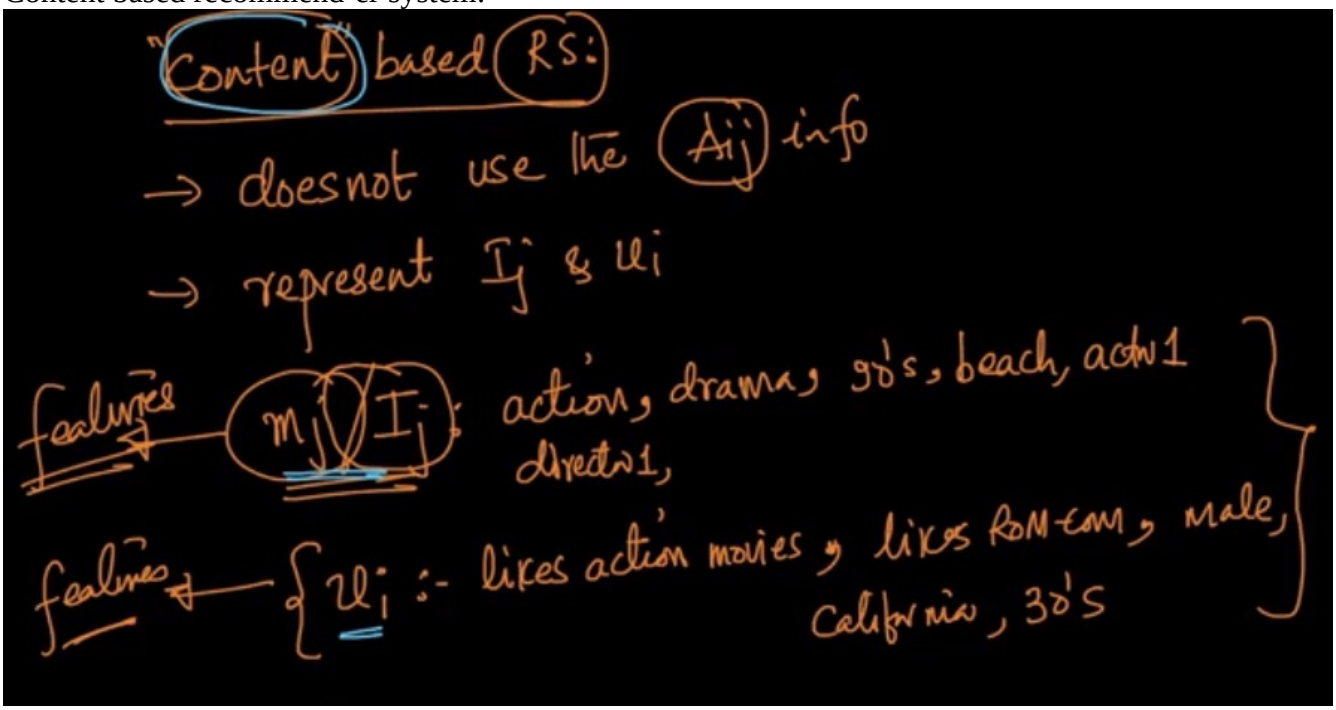


Then we can formulate the matrix completion as matrix factorization.

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.

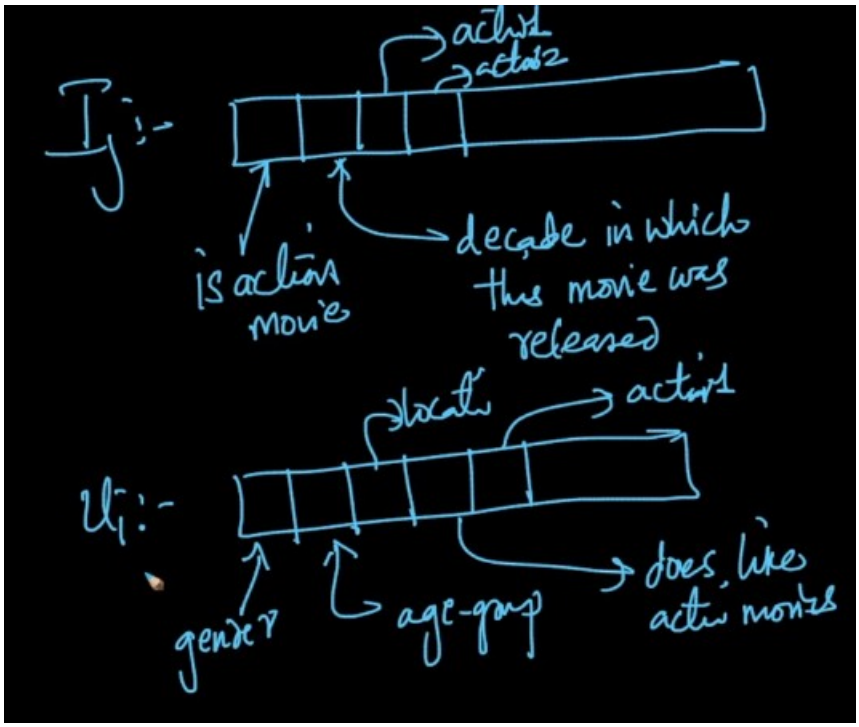


Content based recommend-er system:

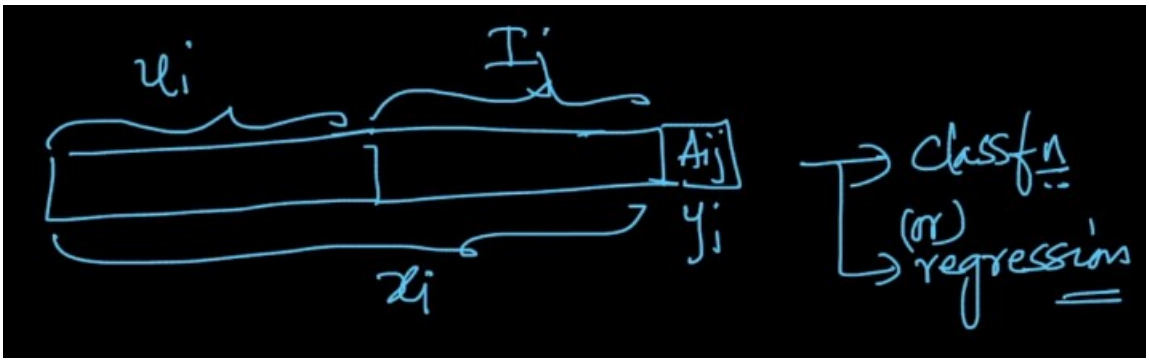


We can represent the collaborative filtering item can be written as:

The items and users can be represented as a vector:



We can pose this as a classification or regression problem.

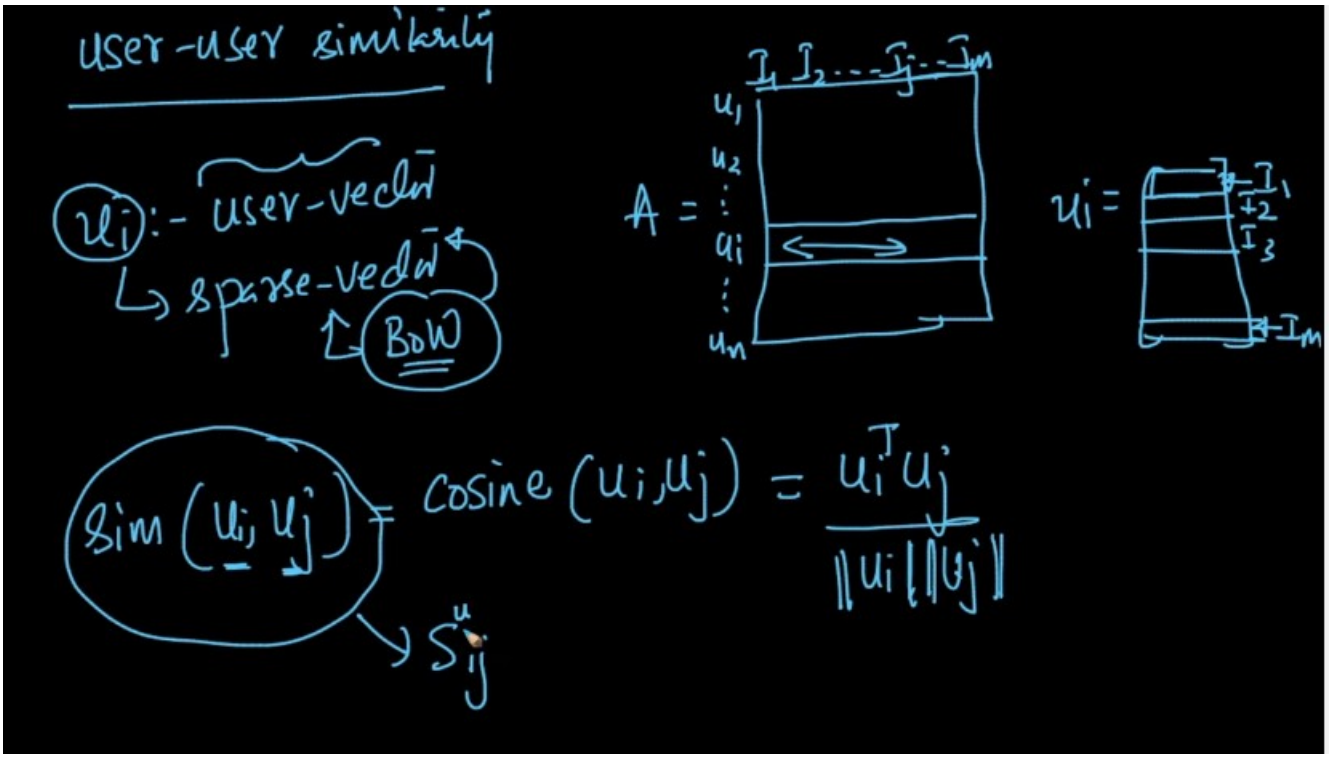


Content based recommend-er system is the classification and regression problems.

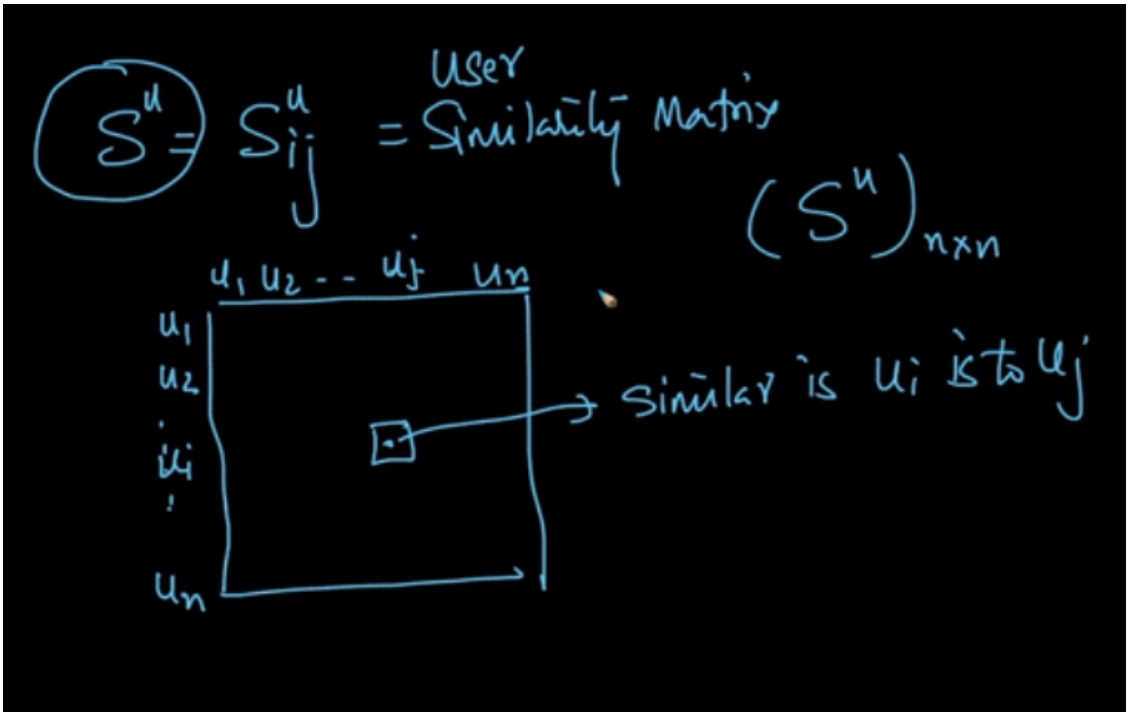
Similarity based algorithms:

User – User based similarity -

We can compute the similarity measures of the users.

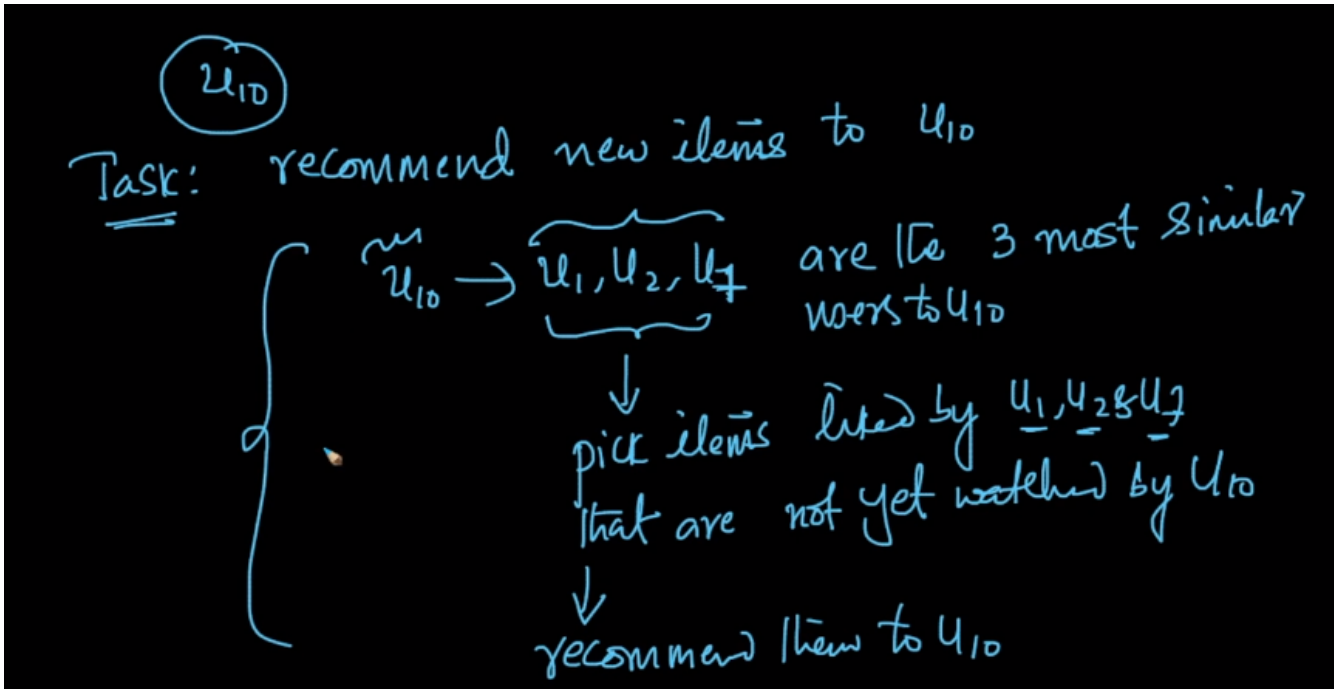


Imagine a matrix S = Sij = User similarity matrix.
This me Un * Un matrix.

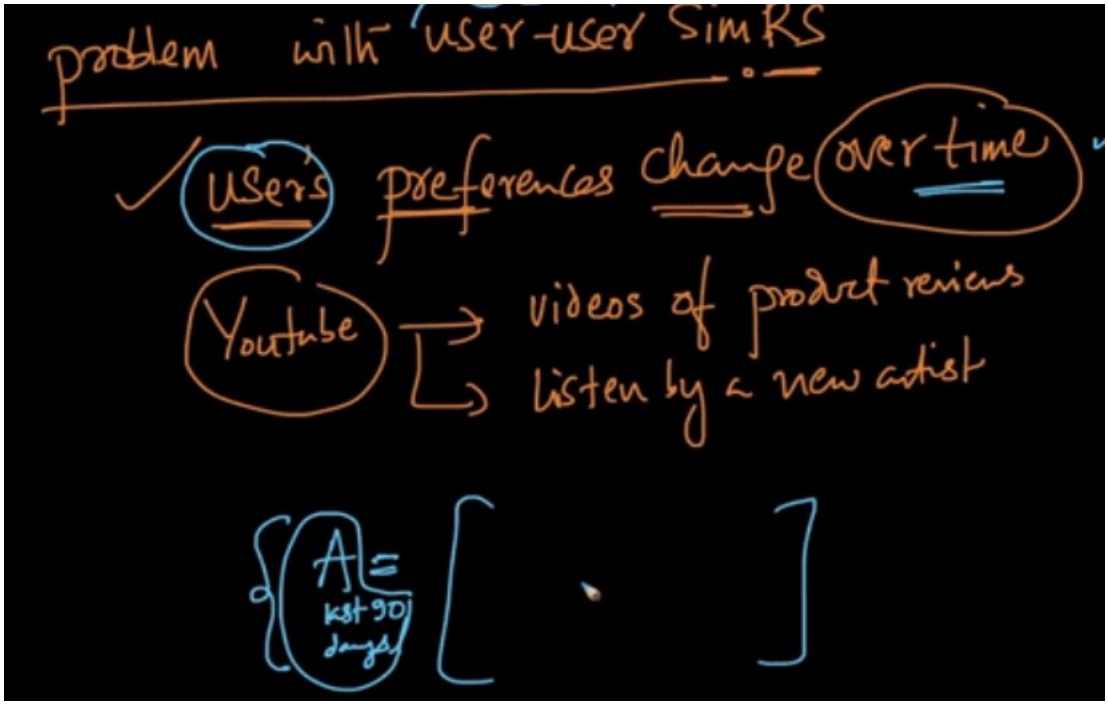


Let’s take the task is to recommend new items to the U10.

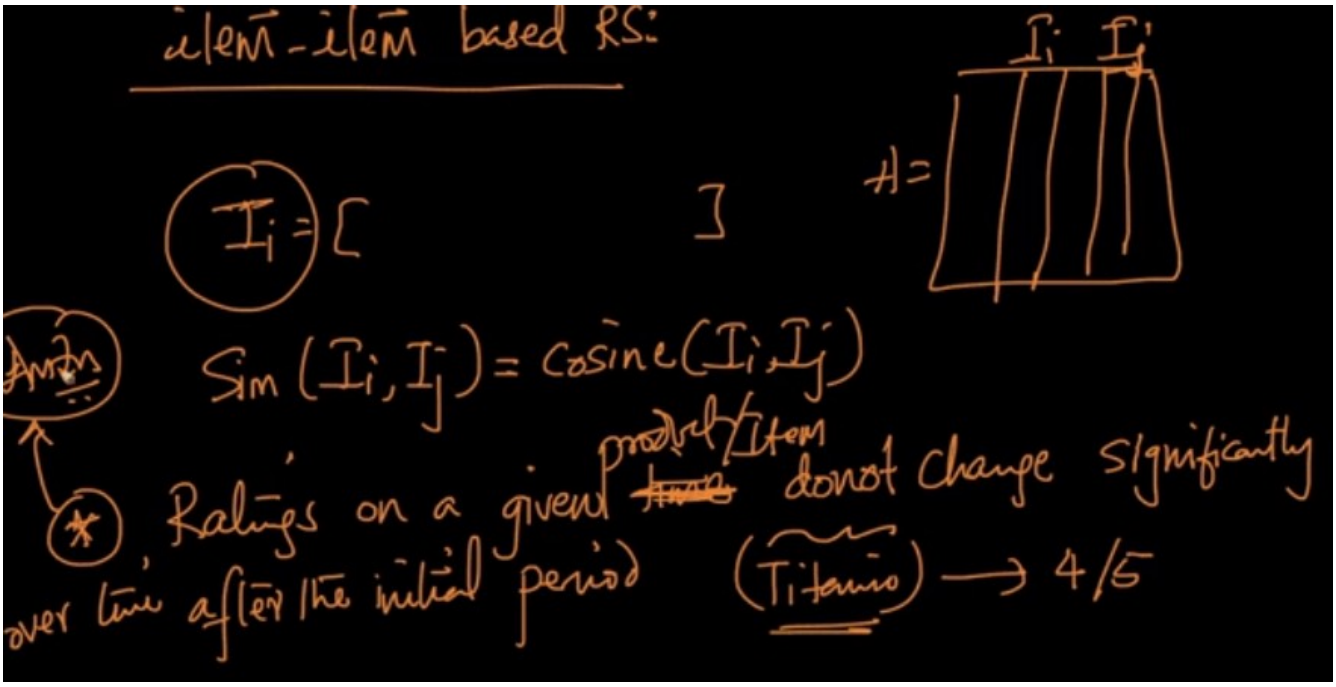
Let’s pick items that are liked by U1, U2 and U7 that are not yet watched by U10 and recommend them to U10.



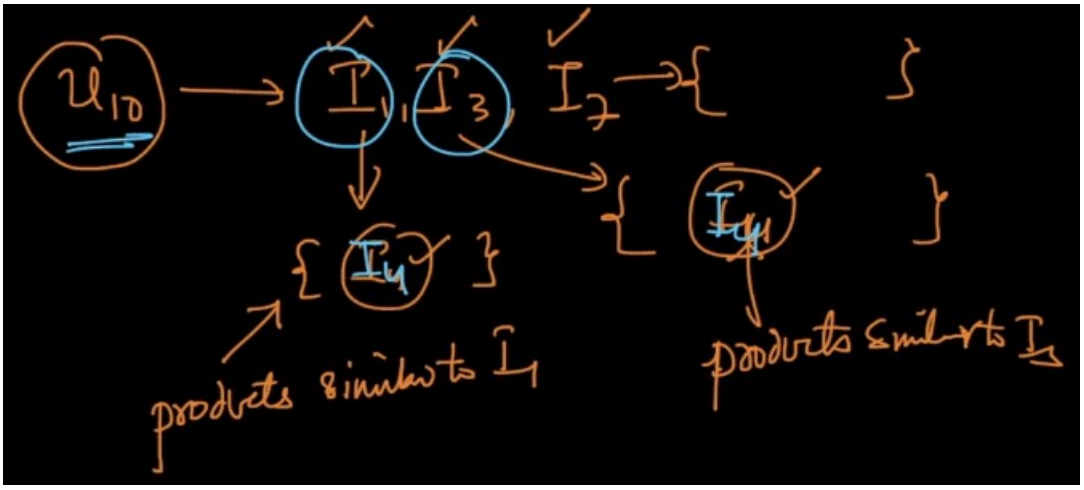
There is one small problem with user – user based sim recommend-er system.
User similarity change overtime.
Users preferences change much more from time to time. We can build there matrix using the last 90 days of data. The matrix becomes sparser.



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.

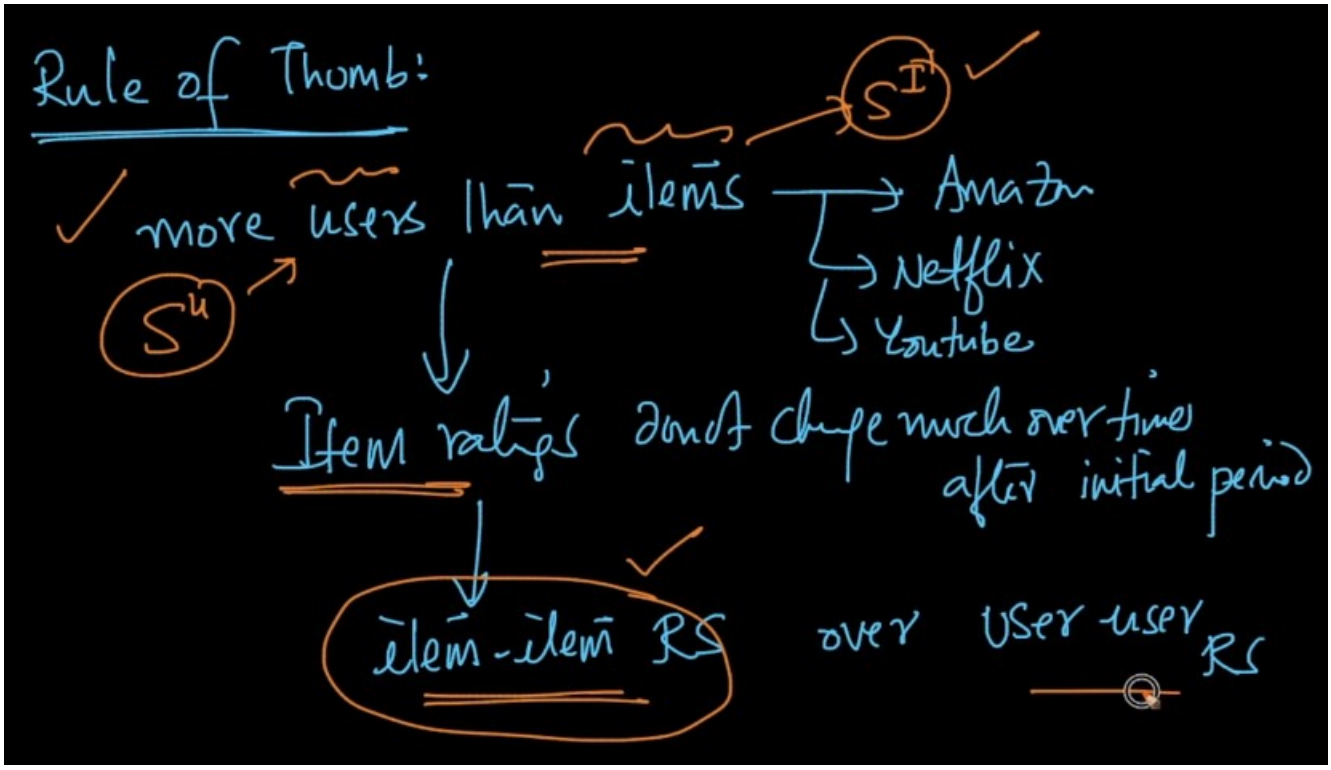


Rating for a product that does not change much, so it is the better chance.

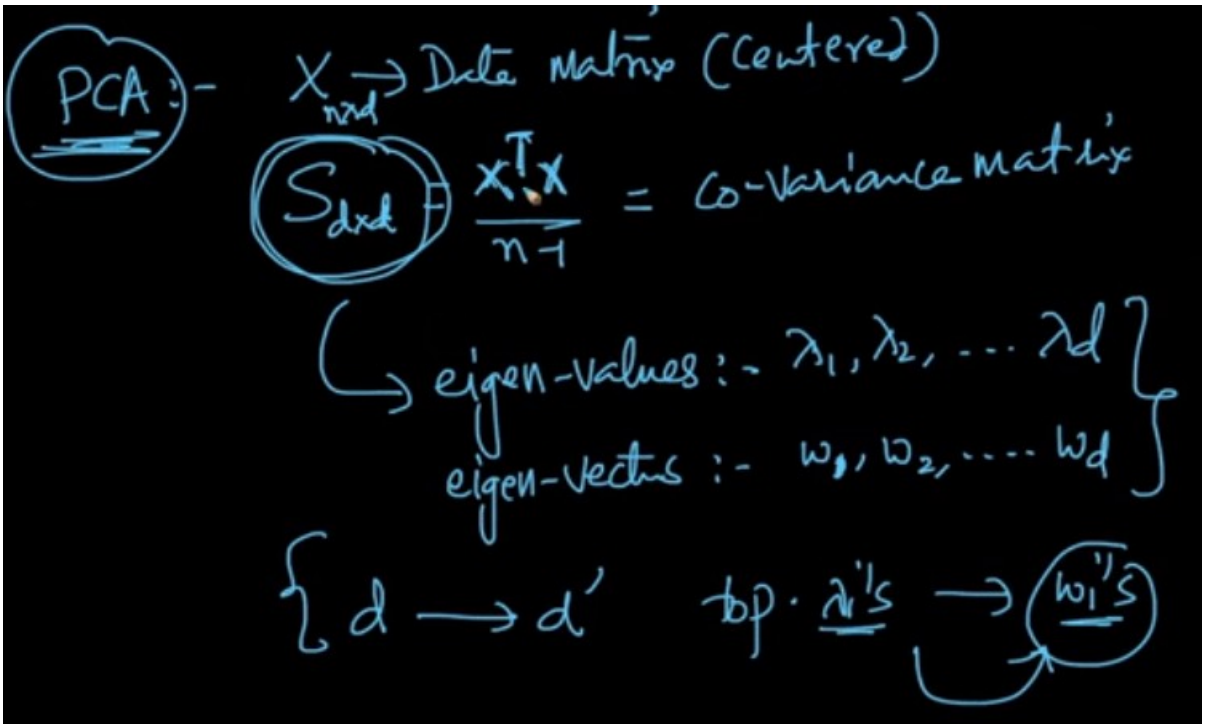


WHEN THE RATING OF THE ITEMS DO NOT CHANGE MUCH OVER TIME AFTER INITIAL PERIOD, THEN WE CAN USE THE ITEM - ITEM BASED SIMILARITY MEASURES.

OVER USER – USER BASED RECOMMENDER SYSTEM.

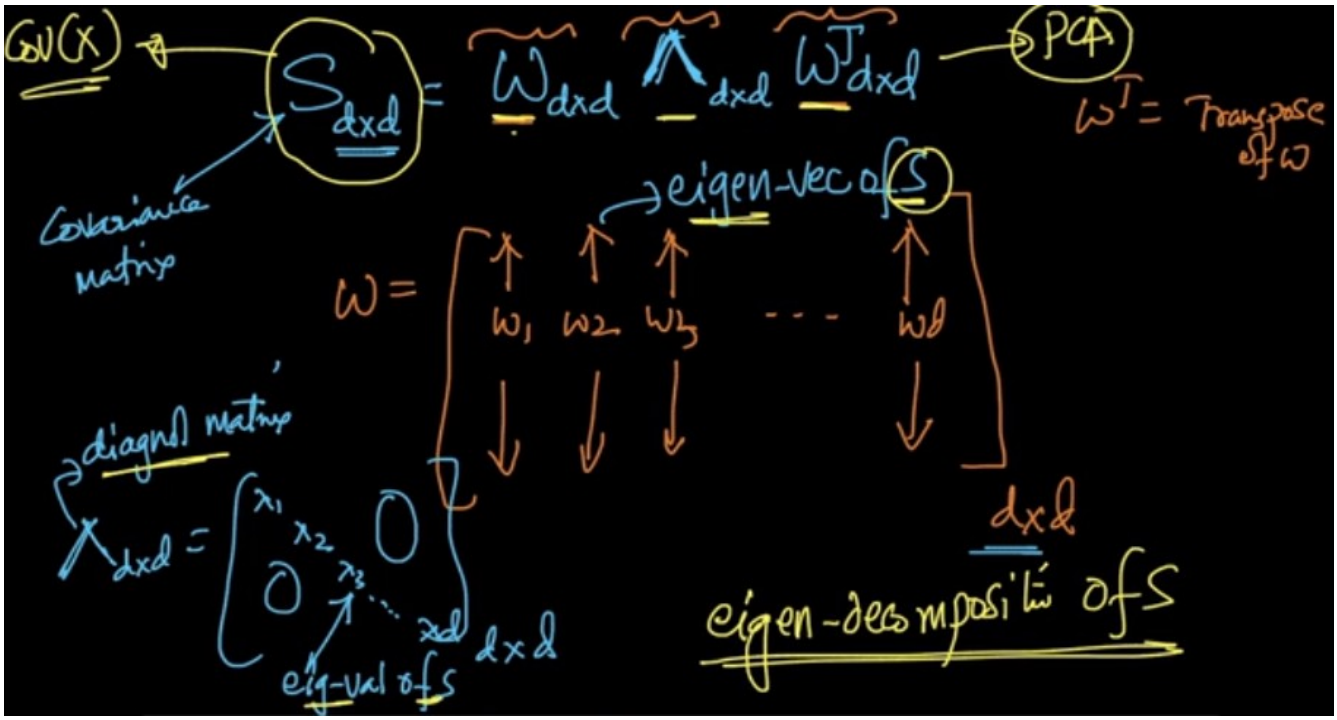


Matrix factorization: PCA, SVD: It is the concept in the dimensional reduction.

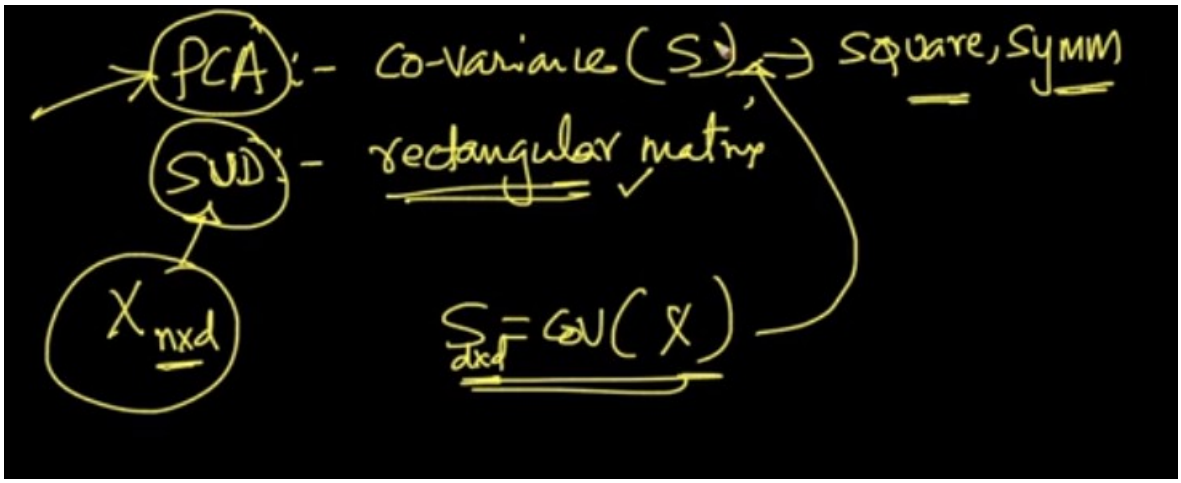


Covariance matrix can be decomposed as the product of three matrices.

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.



$$X_{n \times d} = U_{I \times n} \Sigma_{n \times d} V^T_{d \times d} \rightarrow SVD$$

eig-vectors, eig-values of $S \rightarrow \lambda_1, \lambda_2, \ldots \lambda_d$

$$Cov(X) = S = W \Lambda W^T \rightarrow PCA$$

$\Sigma =$ singular-values of X

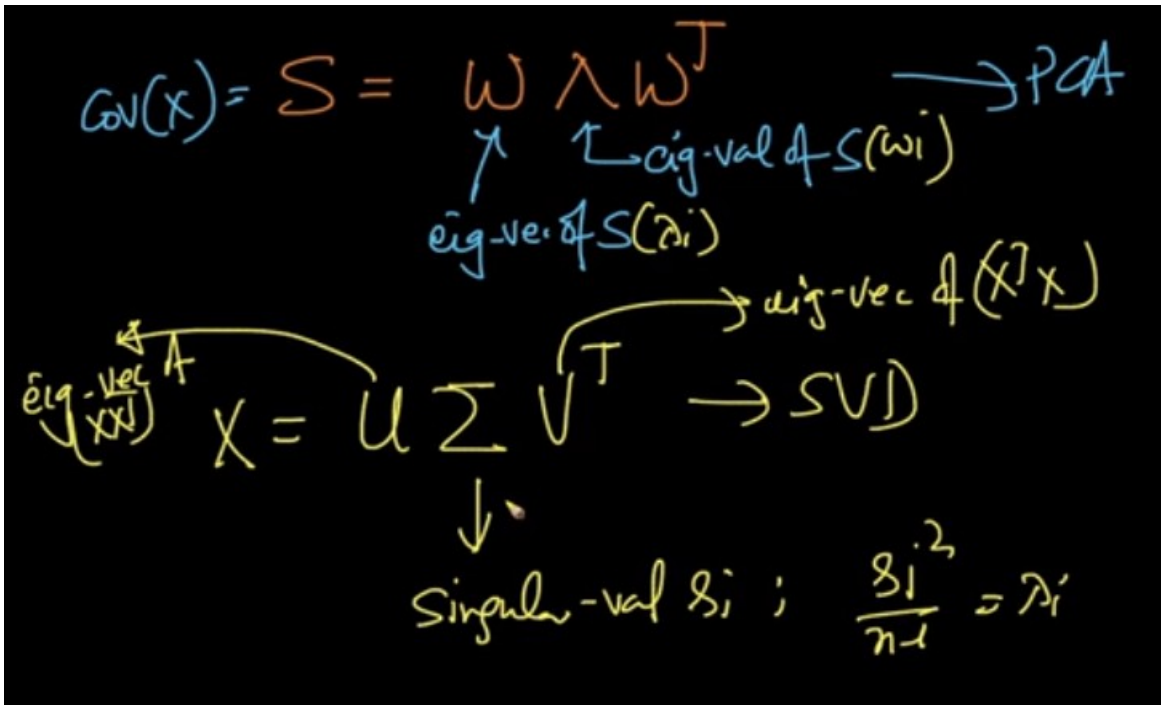
$$\Sigma = \begin{bmatrix} \sigma_1 & & & 0 \\ & \sigma_2 & & \\ & & \sigma_3 \ldots & \\ 0 & & & \sigma_d \\ & & & 0 \\ & & & 0 \end{bmatrix}_{n \times d} \rightarrow \text{diagonal matrix}$$

$\sigma_i^2 = \lambda_i$

eig-val of S (λ_i) & Singular values of X (σ_i)

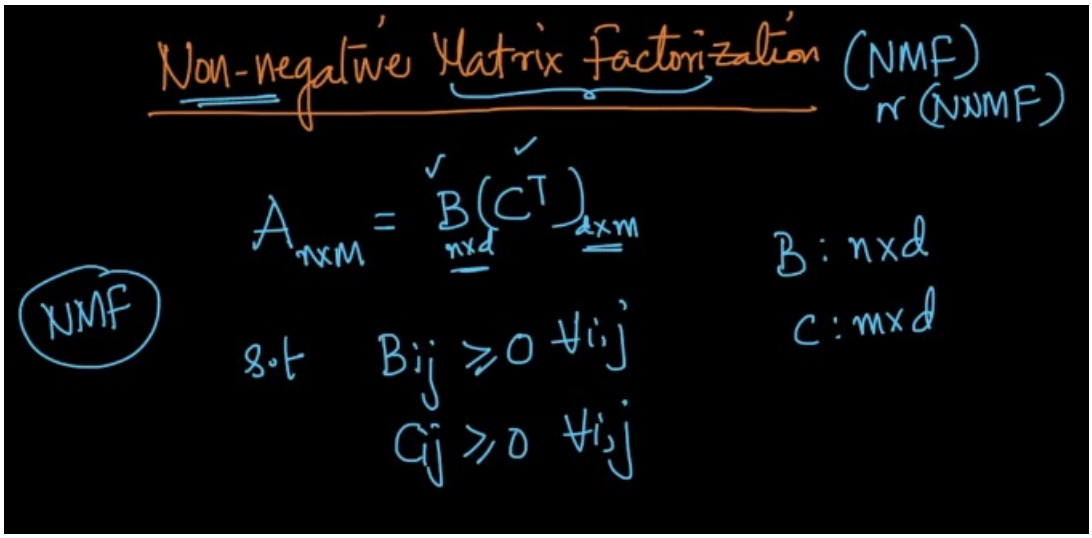
$U_i :-$ i^{th} col. of U = i^{th} eig. vec of $(X_{n \times d} X^T_{d \times n})$ ← not S

$V_i :-$ i^{th} col of V = i^{th} eig-vec of $(X^T_{d \times n} X_{n \times d}) = S$



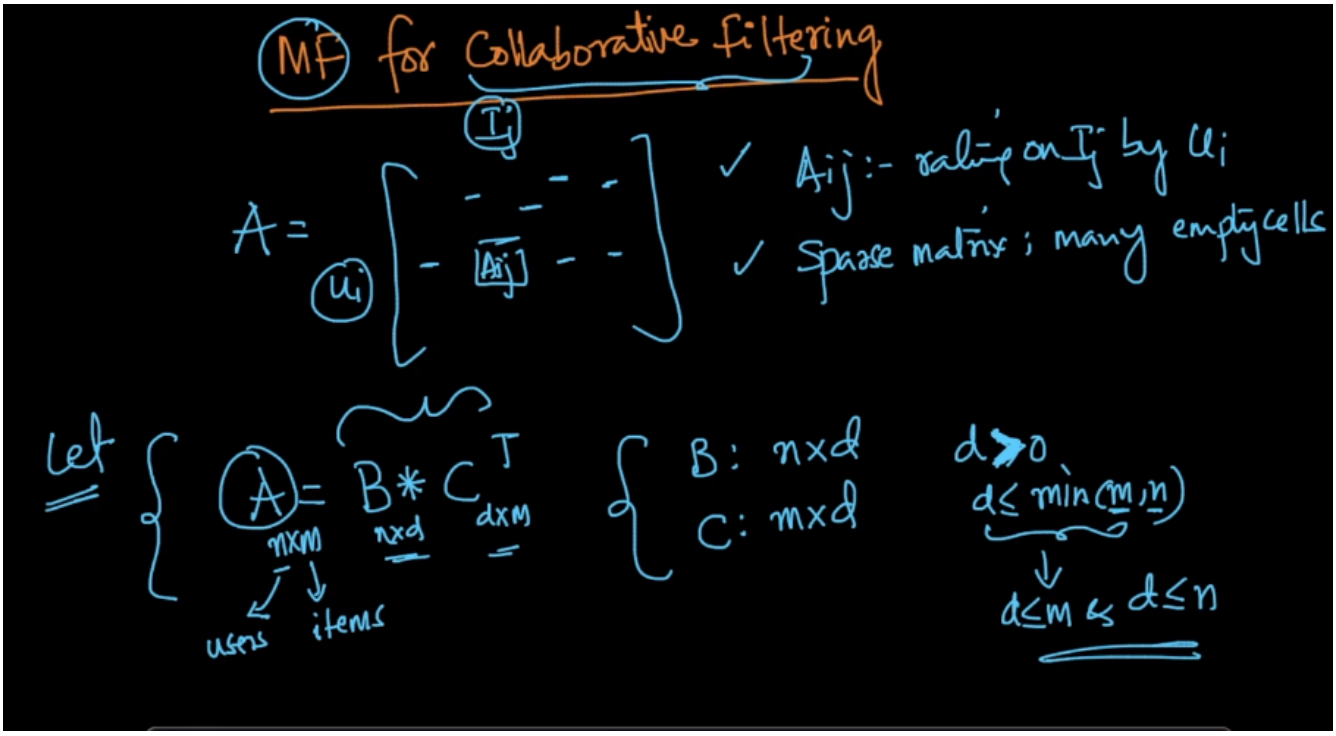
Fsvd

Matrix factorization: NMF

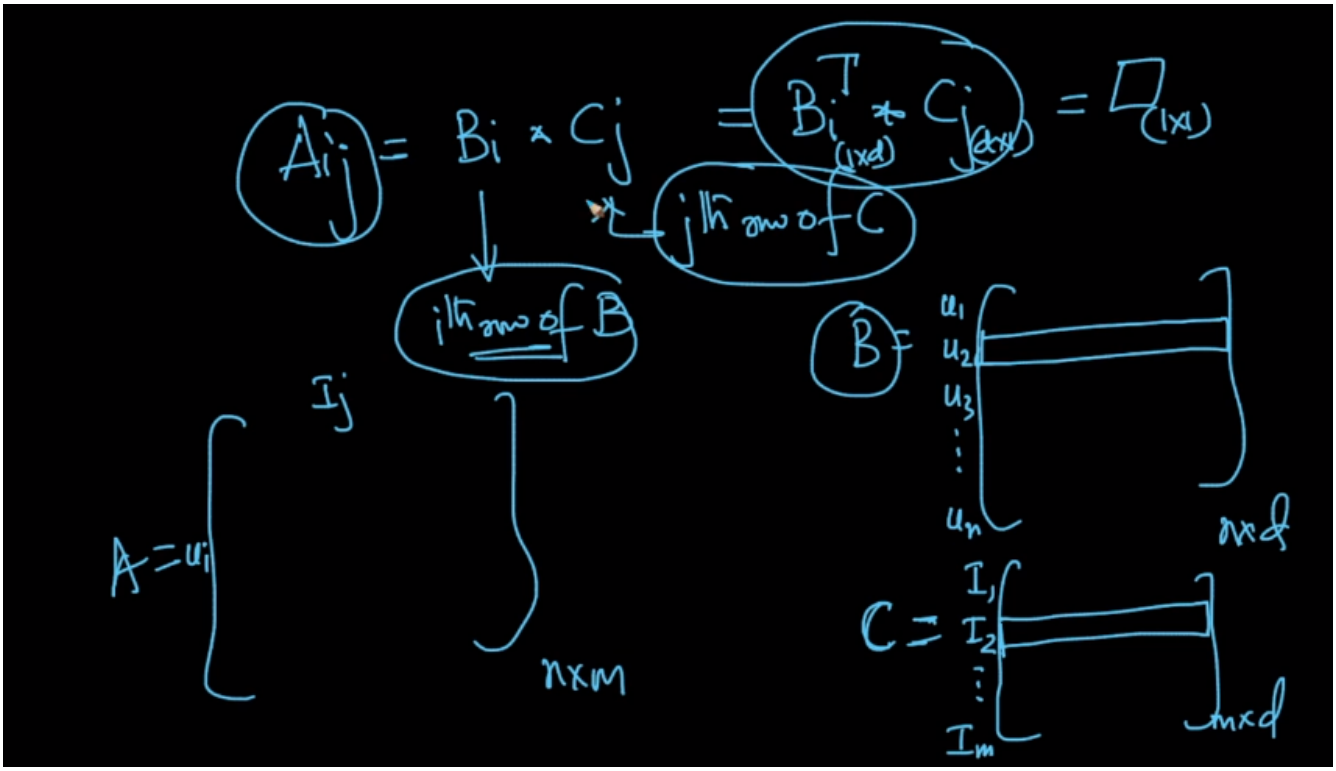


The elements in the NMF are all positive.

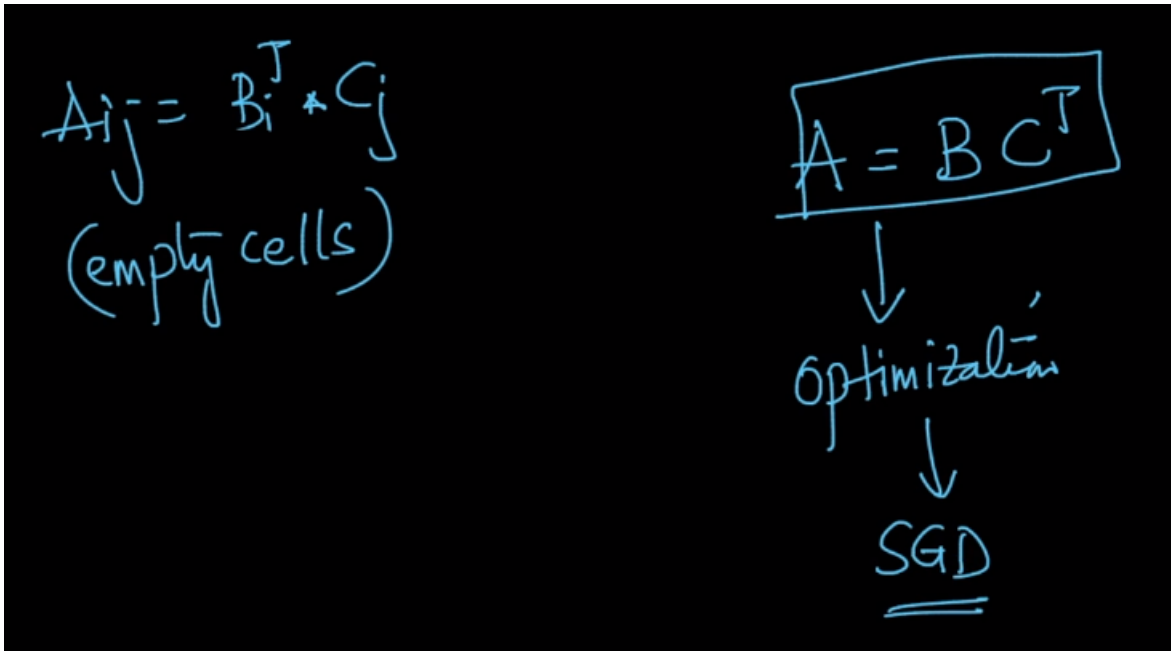
Matrix factorization for collaborative filtering:



Aij is the product of bi and cj.

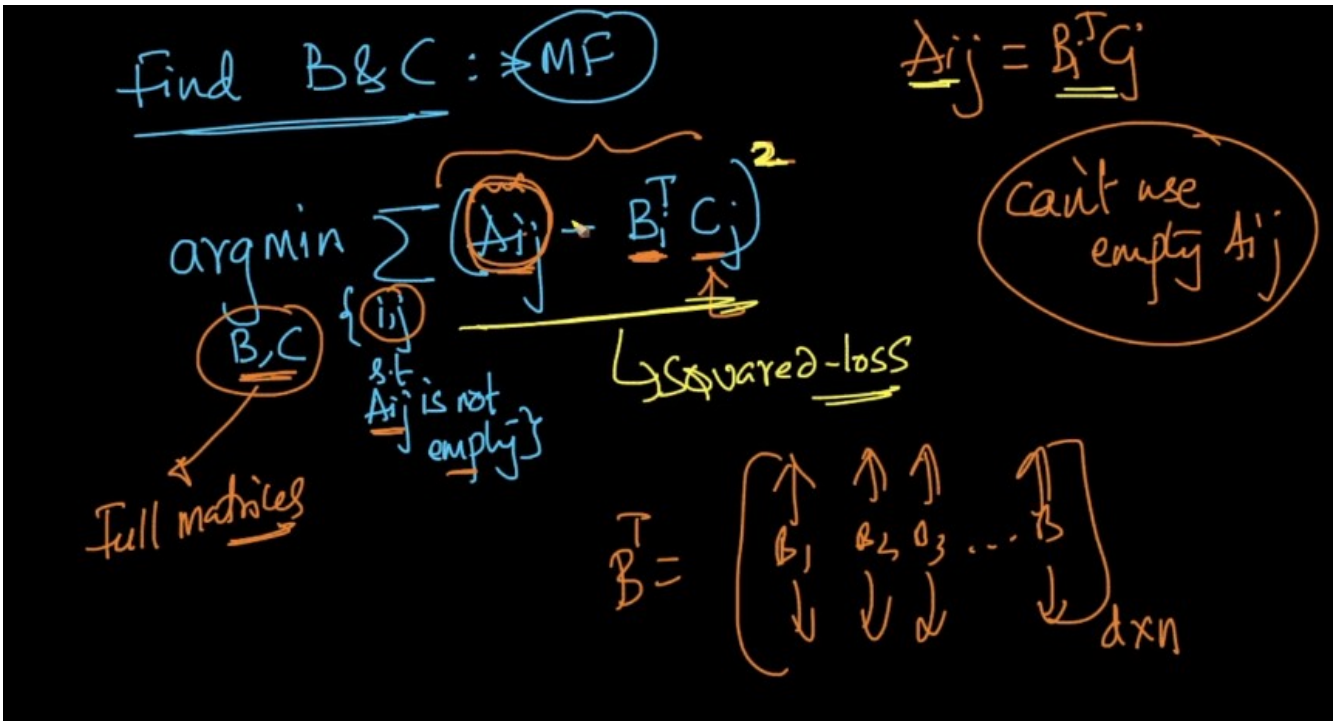


We can write this as follows:

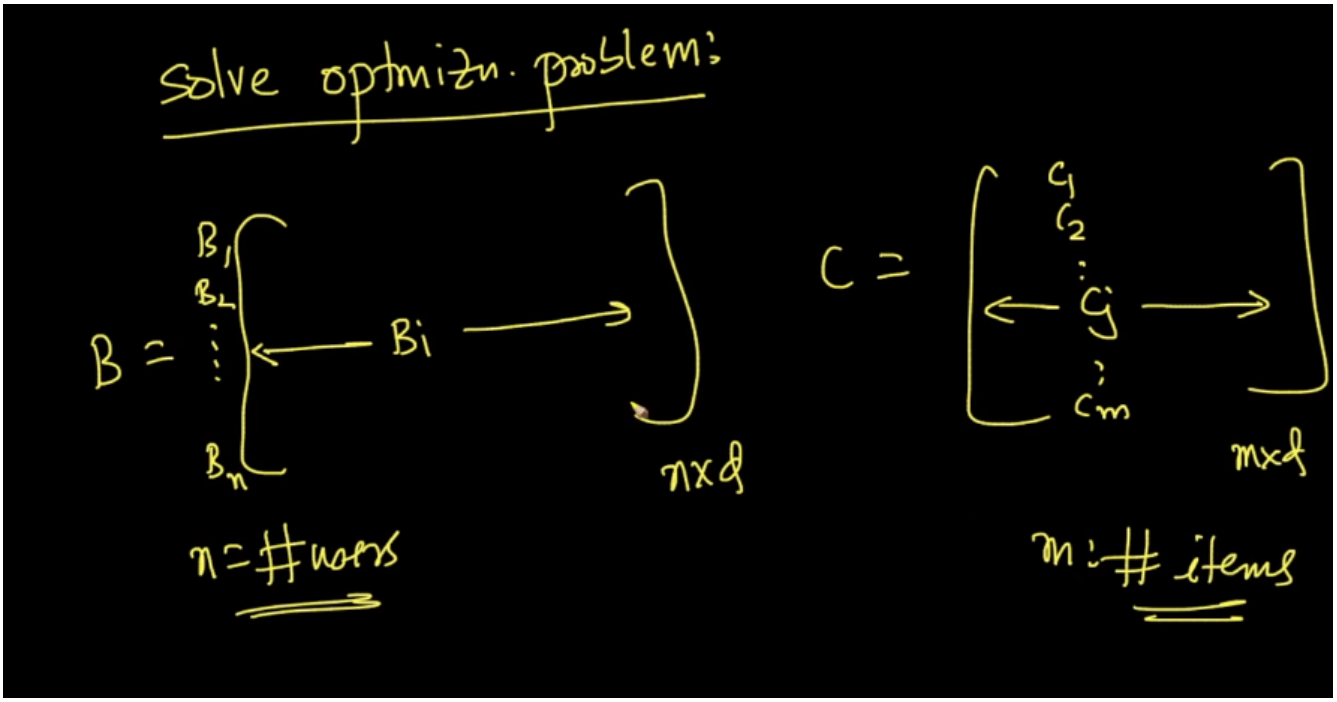


To find B and C: Matrix factorization

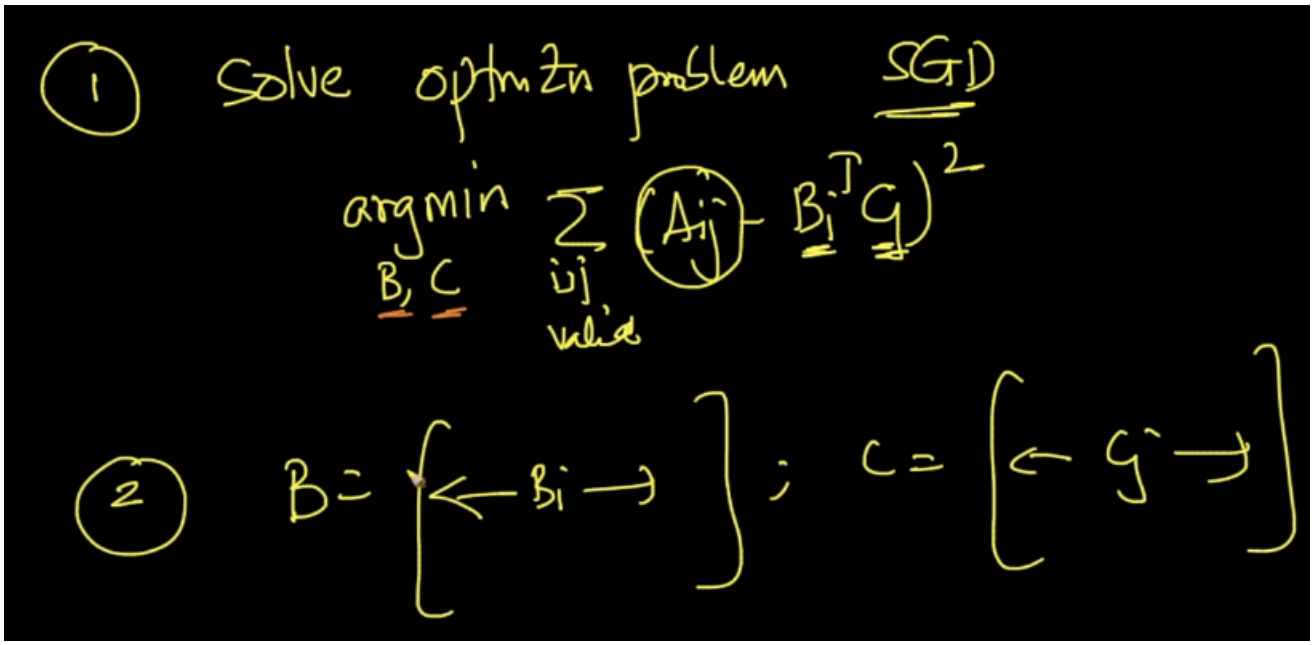
Lets reduce the sparse matrix by solving the optimization problem.



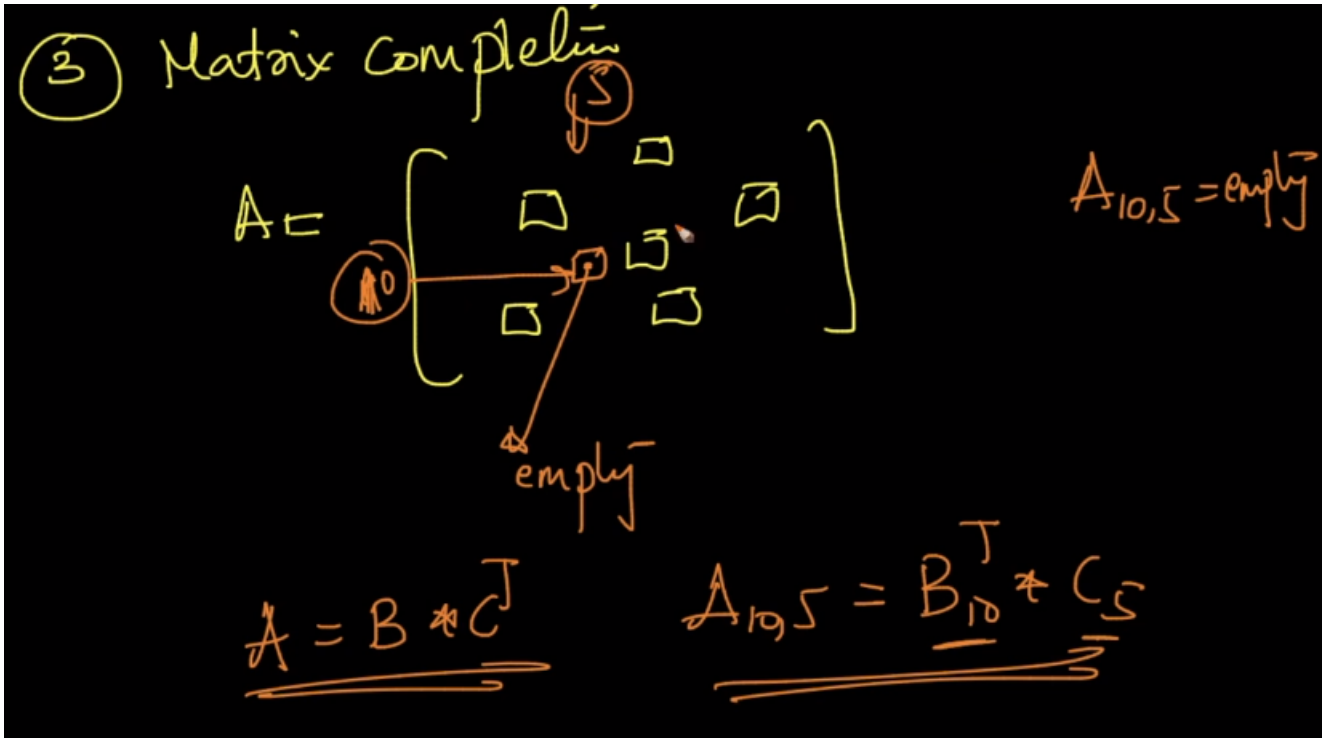
If we find the matrices B.T and C then the matrices are good.
We are minimizing the squared loss like regression problems.
We can apply SGD to solve this optimization problem. At the end we get the matrices B and C.

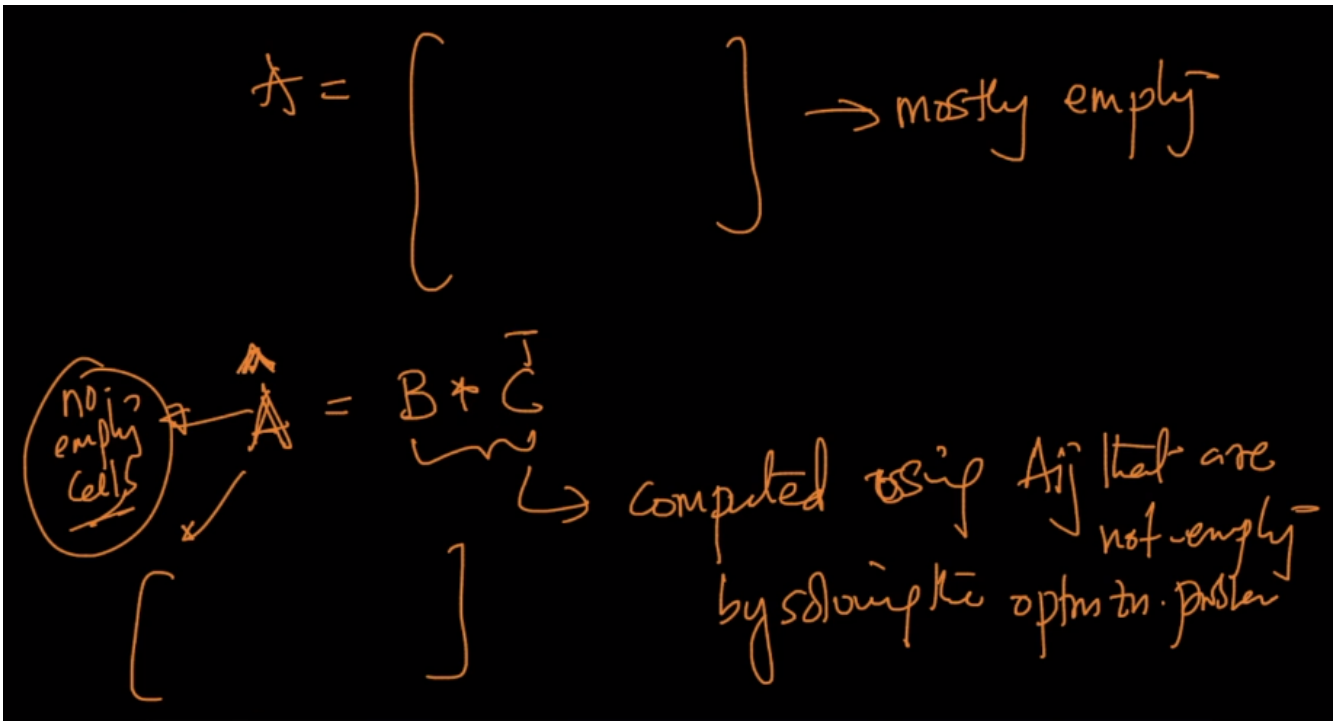


At the end of the problem we get the Bi's and cj's
Step – 1
Solve the optimization problem.

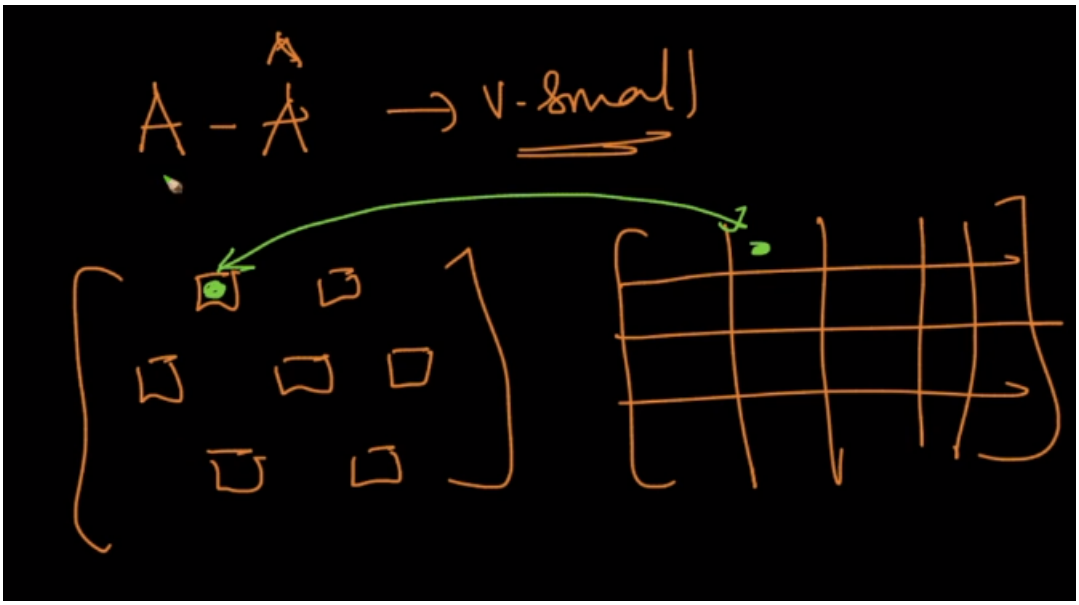


Third step is matrix completion:

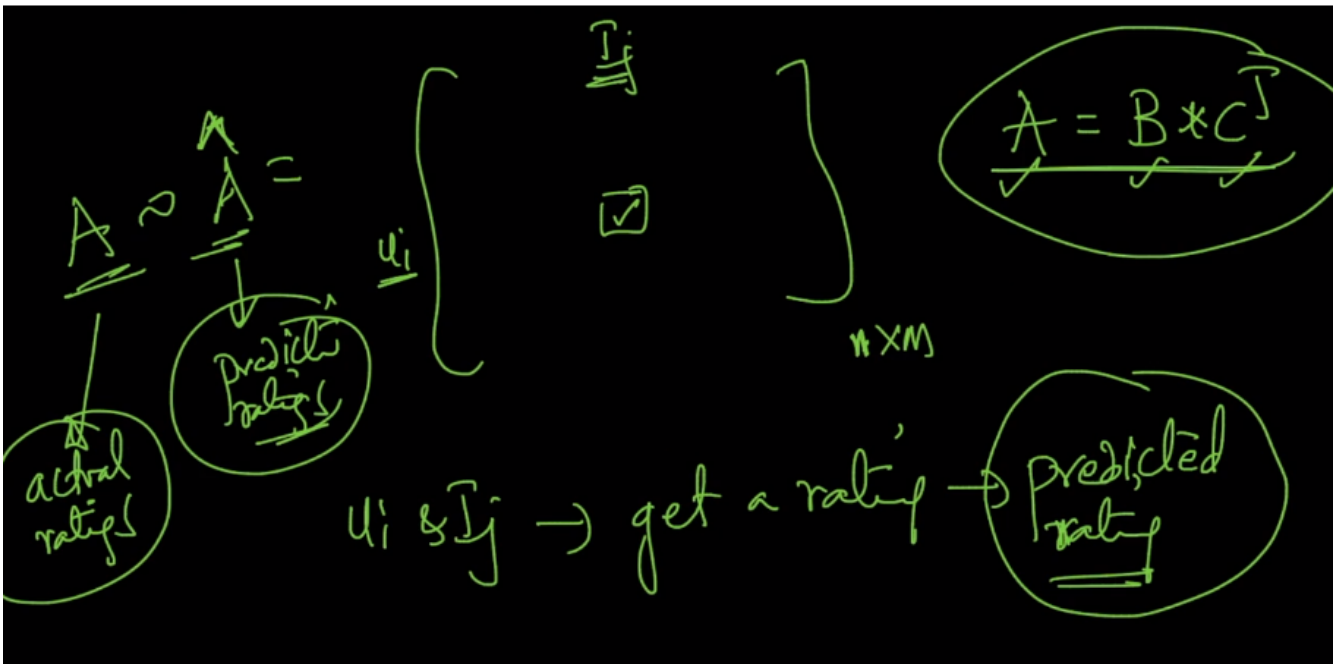




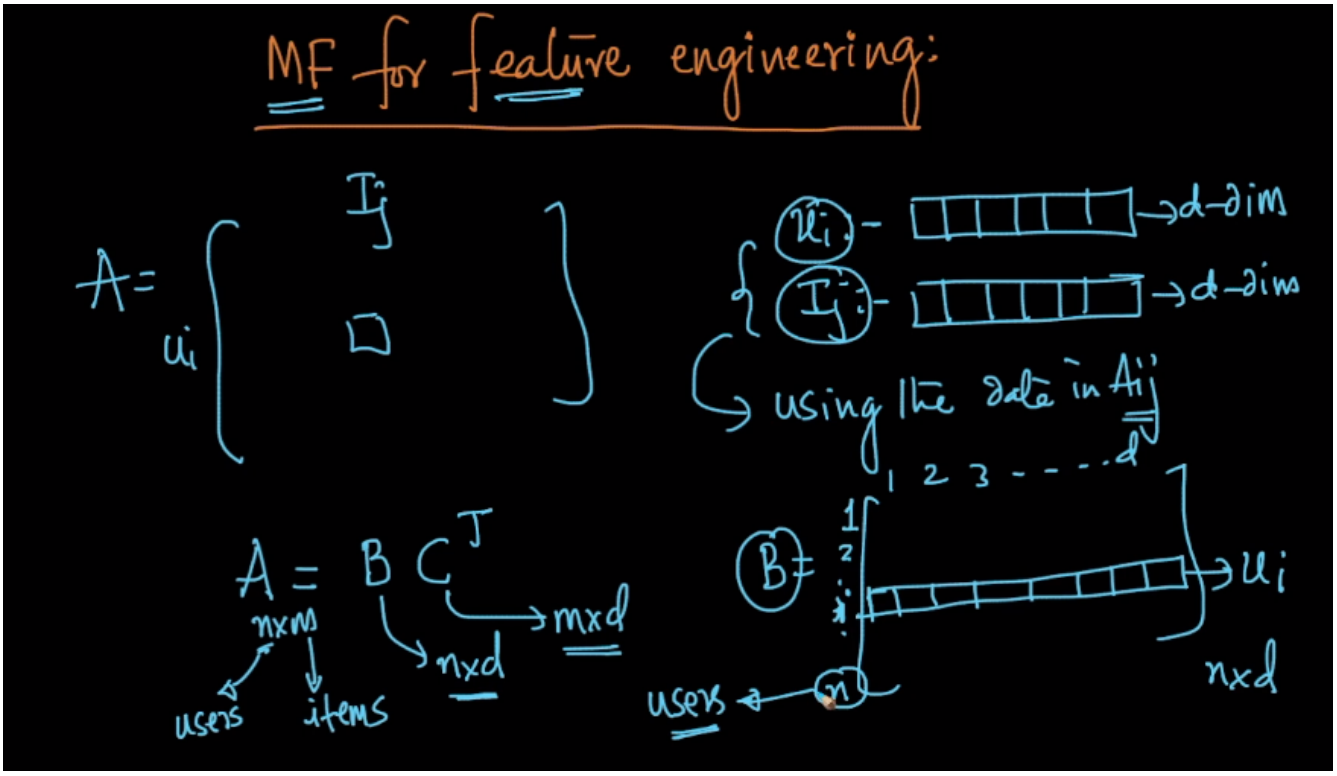
We have to ensure that the real and absolute values are very similar.

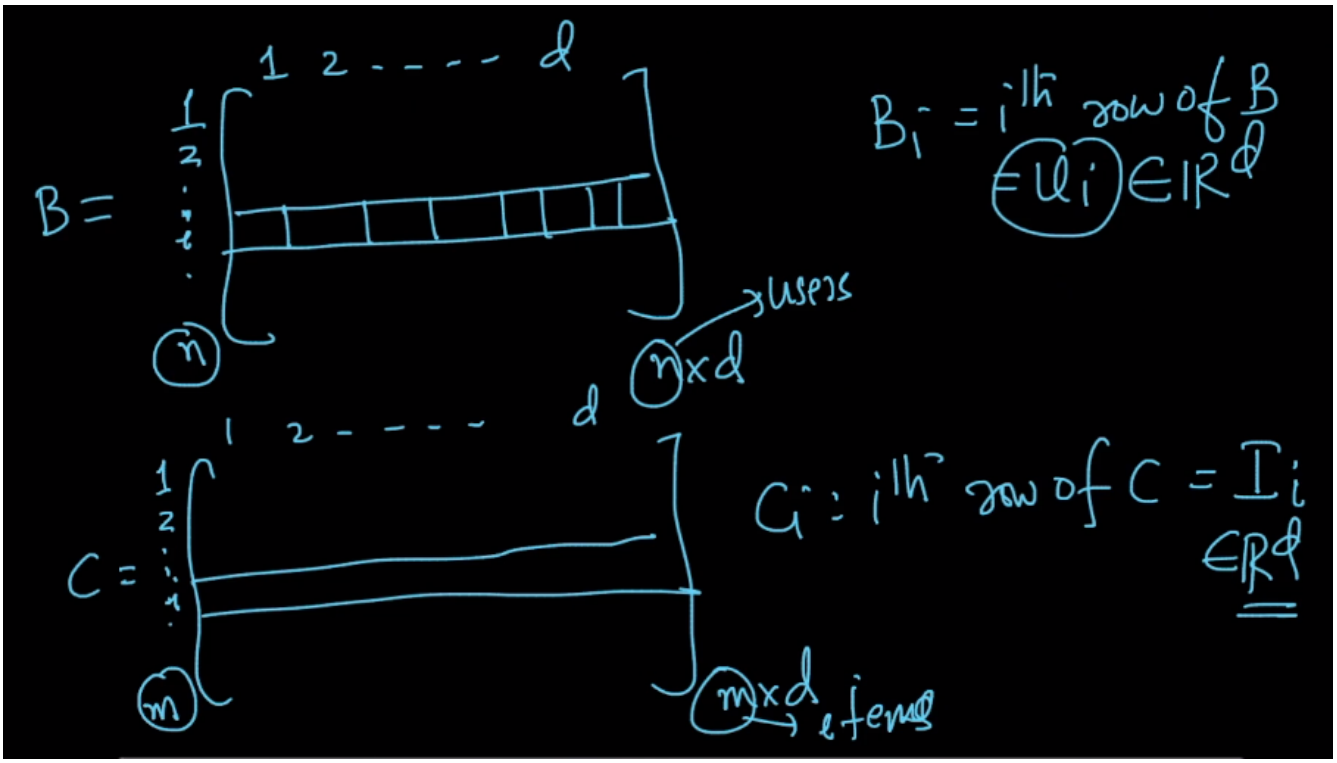


A^ consists of predicted rating.

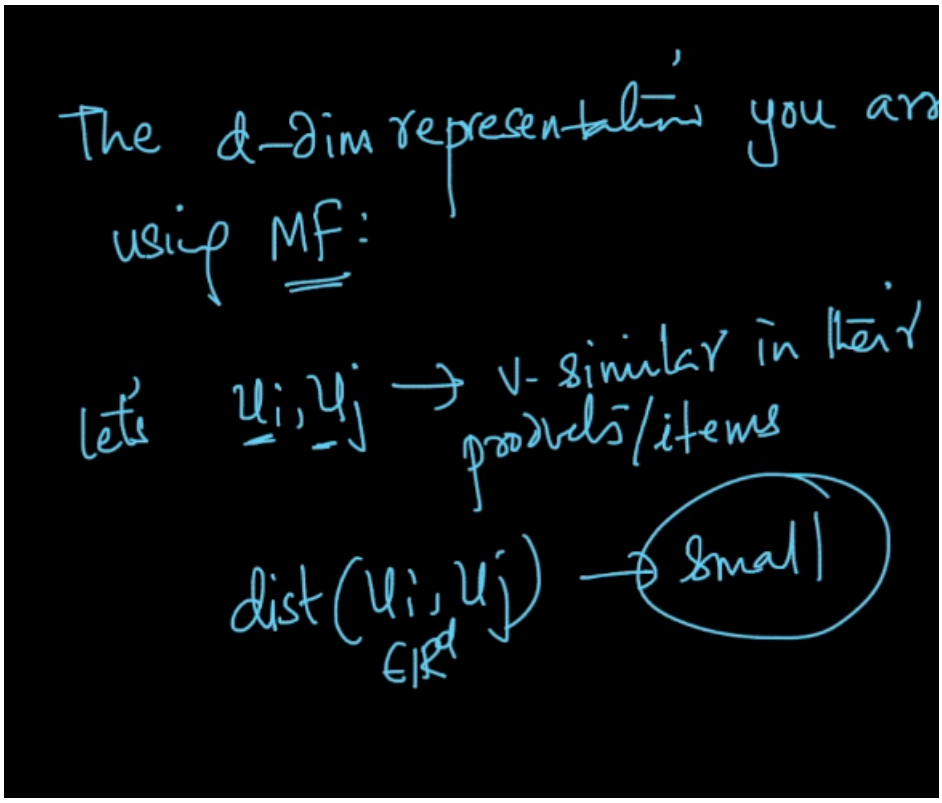


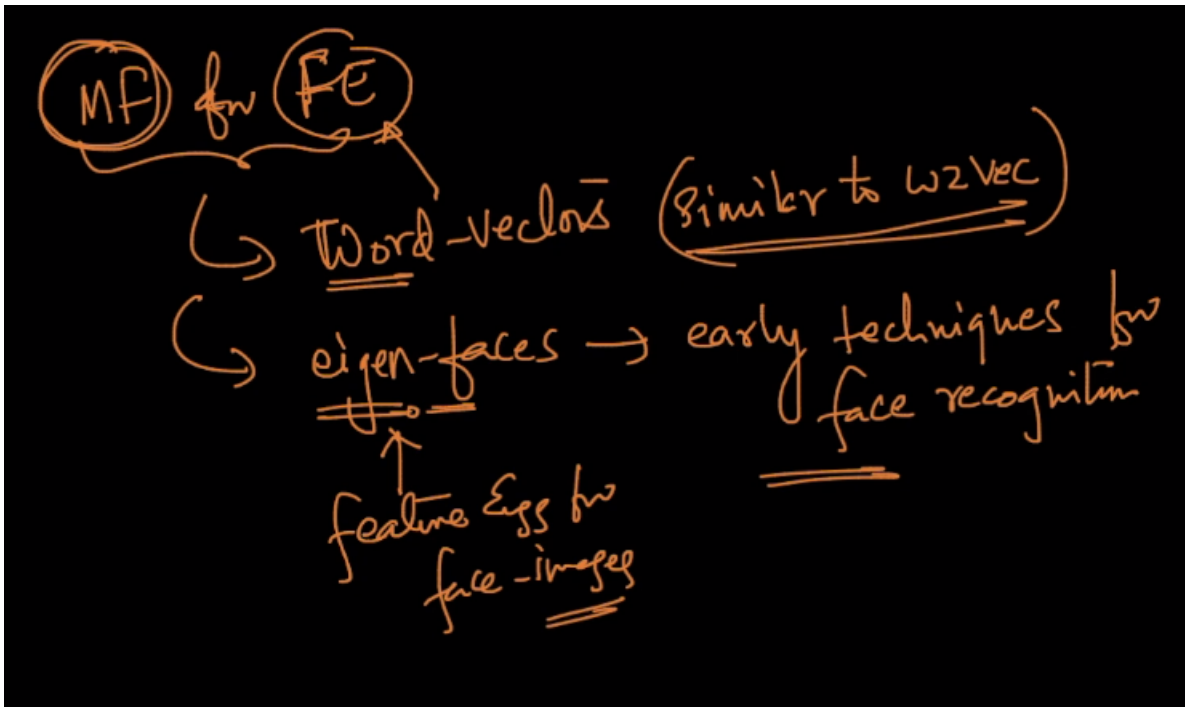
This is how we use matrix factorization.
Matrix factorization for feature engineering:
We want to create the vectors of users and items in the matrix. The d-dim vectors for user and items.



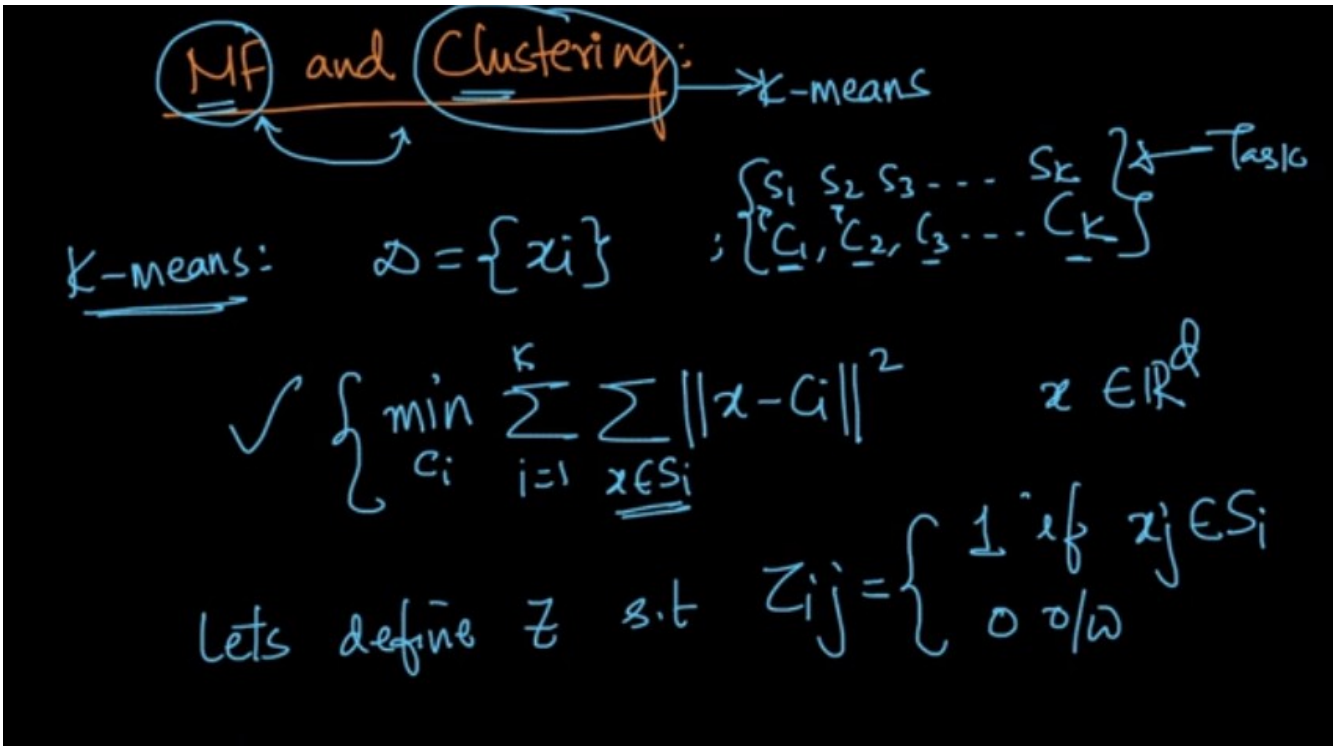


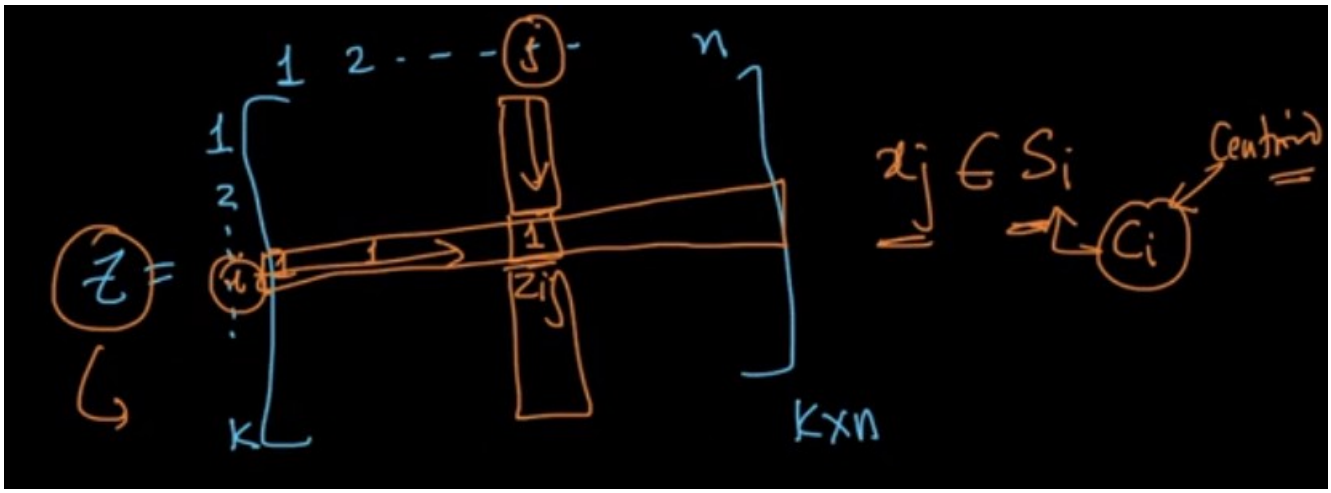
The most important thing we got at MF:

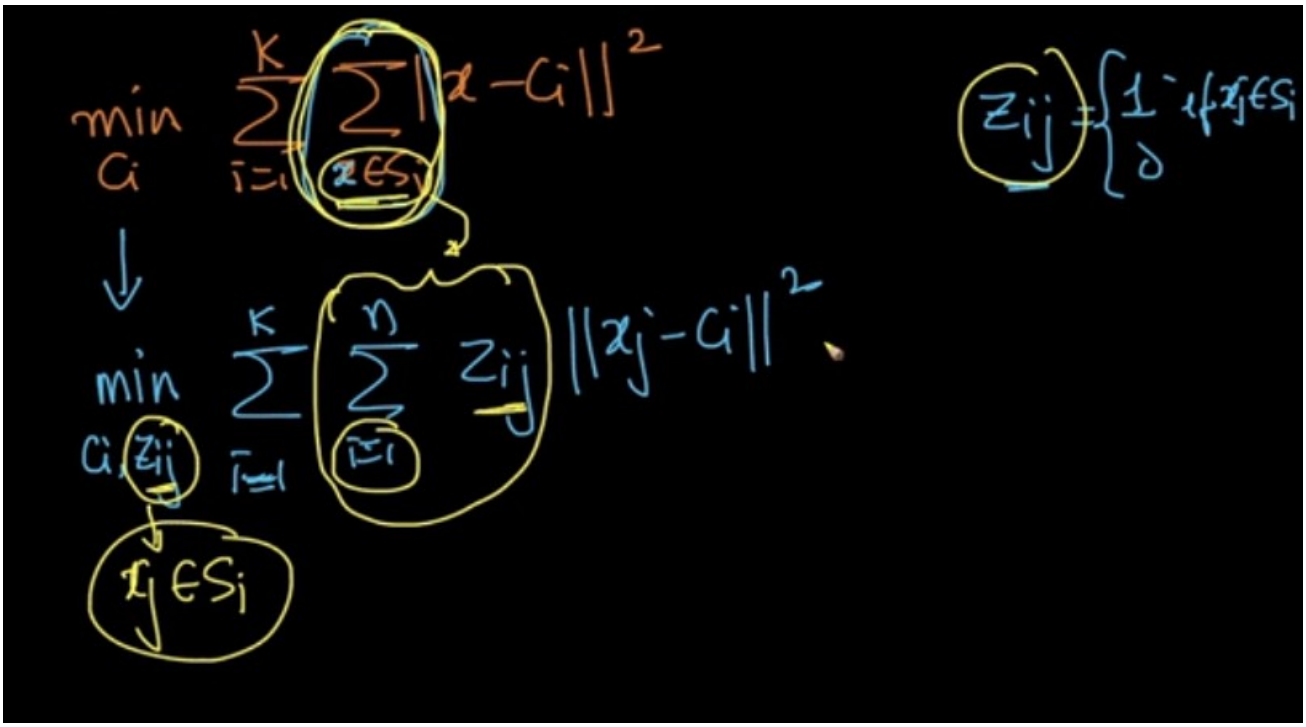


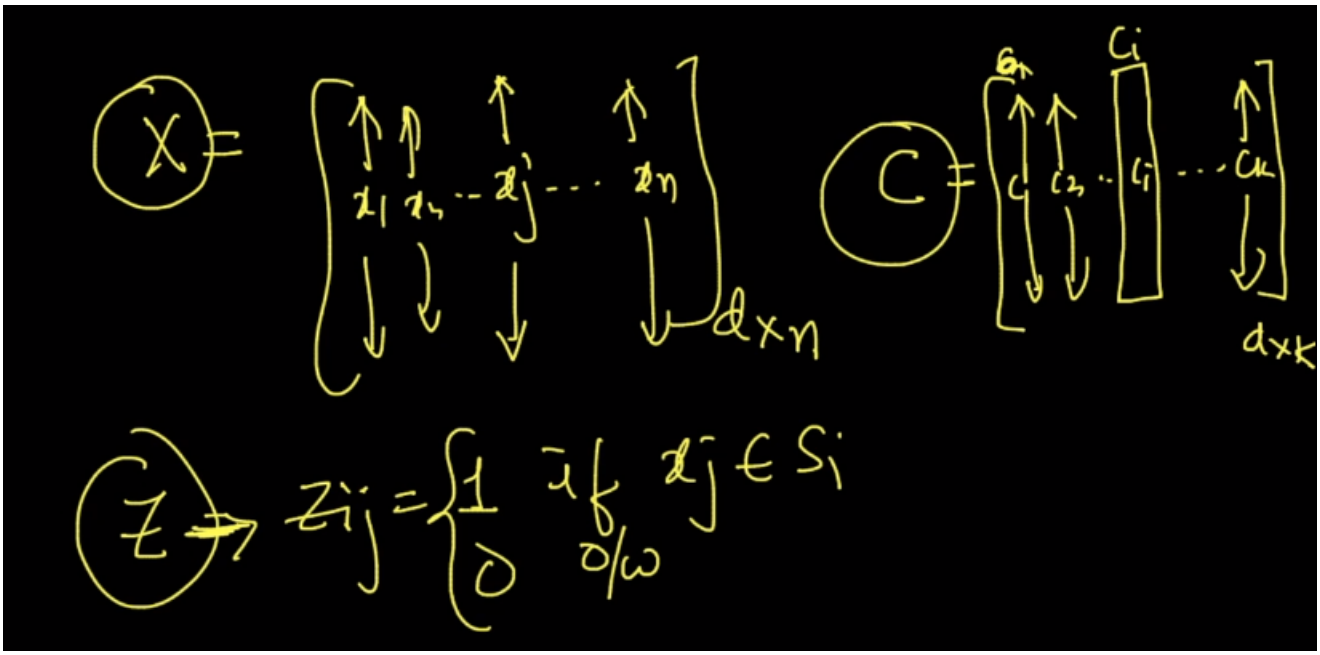


Clustering as MF:

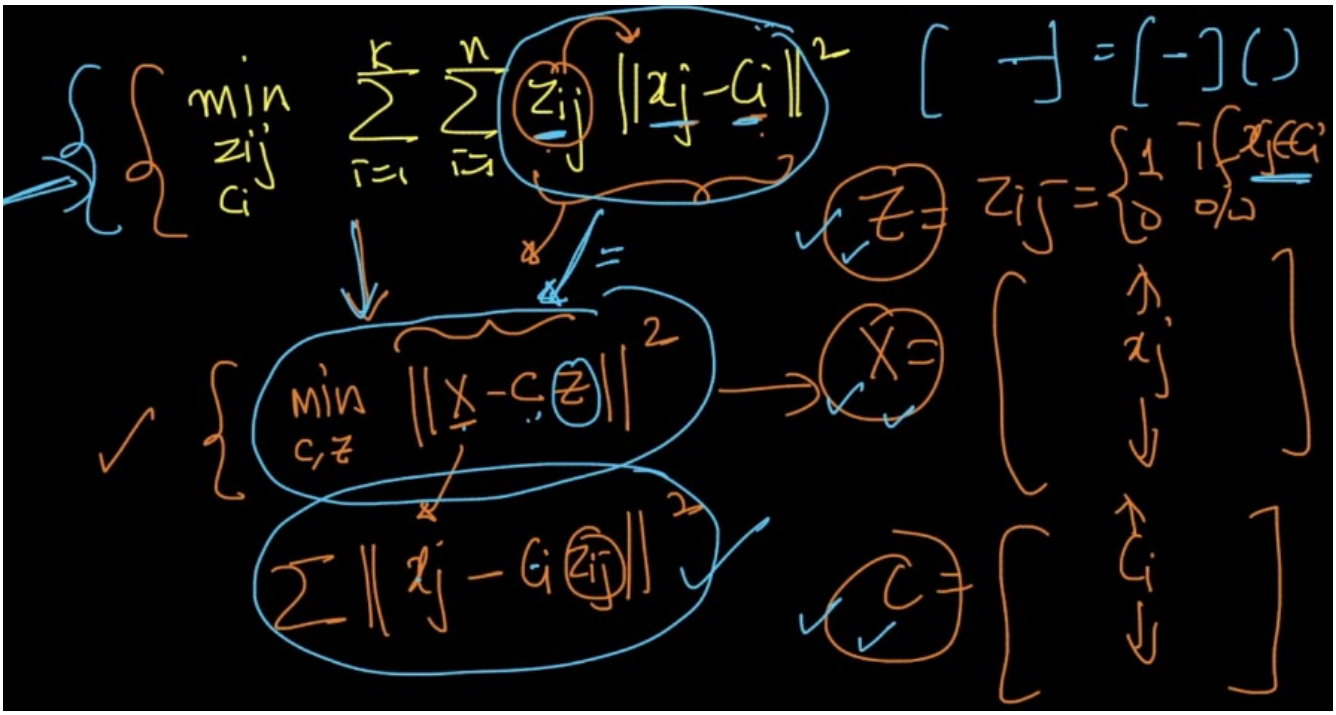


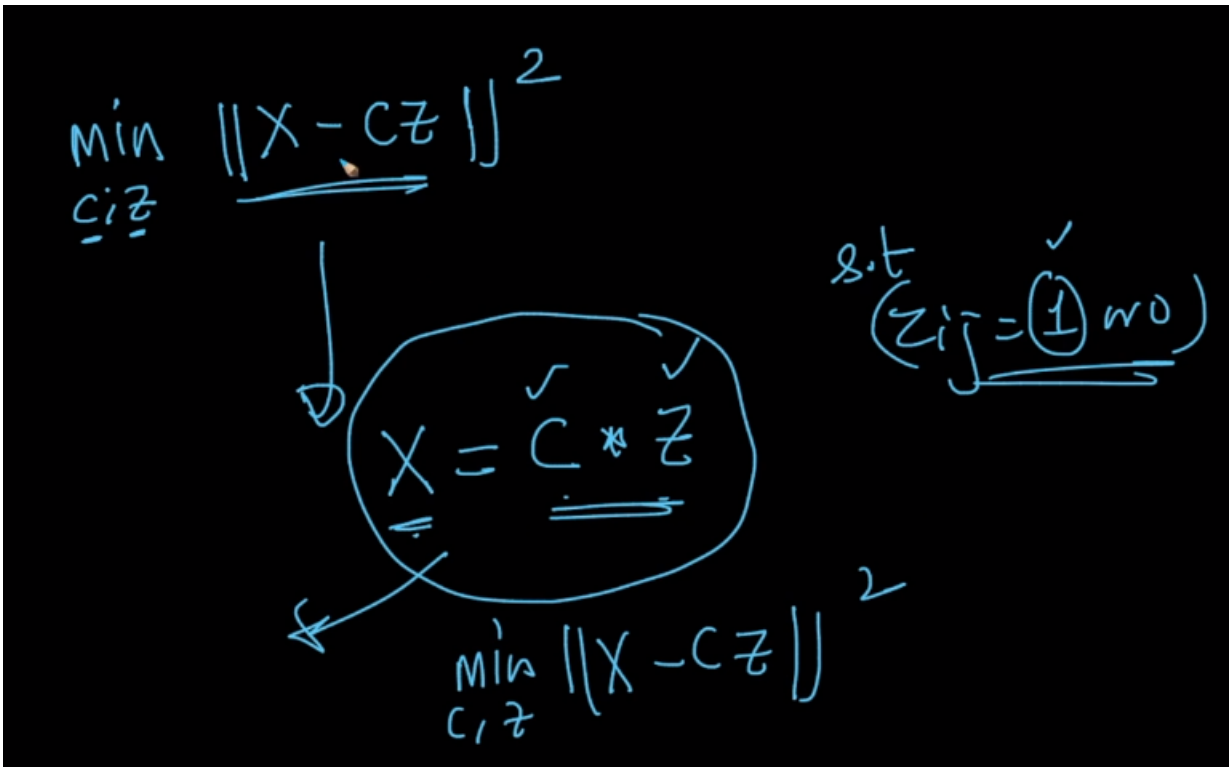


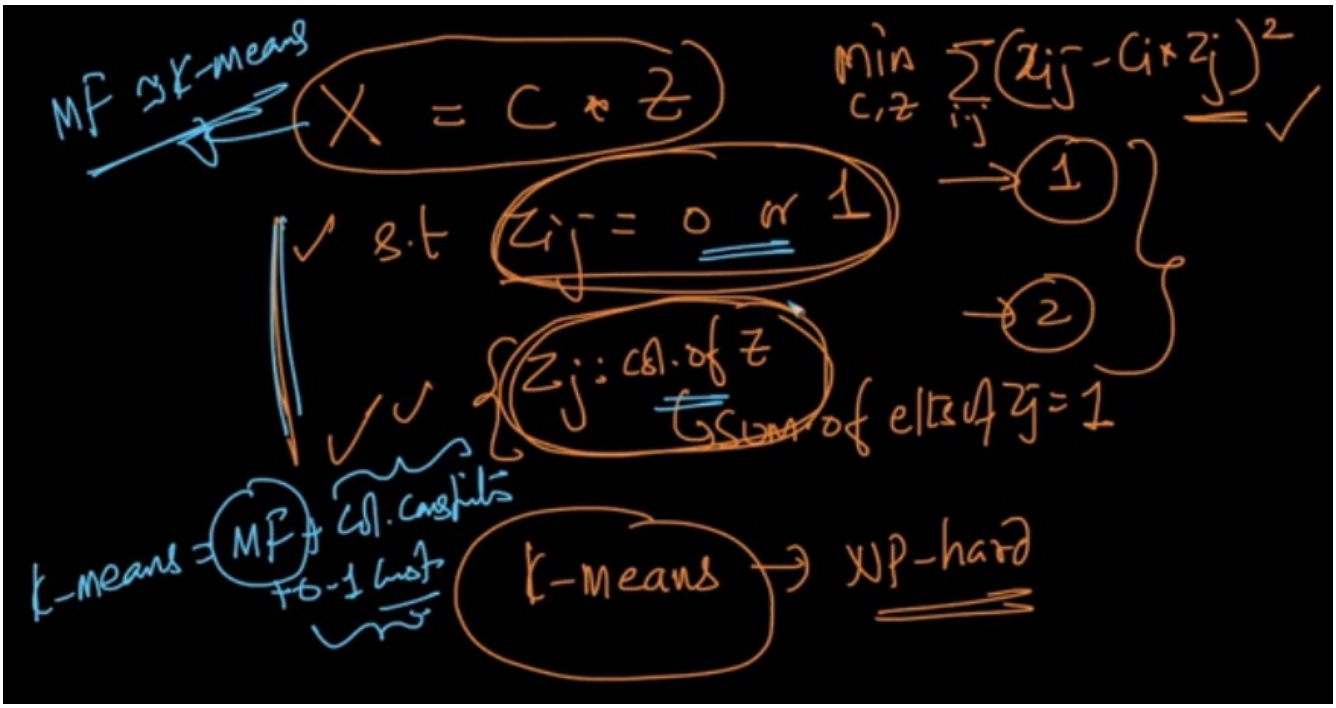

min
ci
$\sum_{i=1}^{k} \sum_{x \in S_i} \|x - c_i\|^2$
$z_{ij} = \begin{cases} 1 & \text{if } x_j \in S_i \\ 0 & \end{cases}$
$\sum_{i=1}^{k} \sum_{j=1}^{n} z_{ij} \|x_j - c_i\|^2$
$x_j \in S_i$

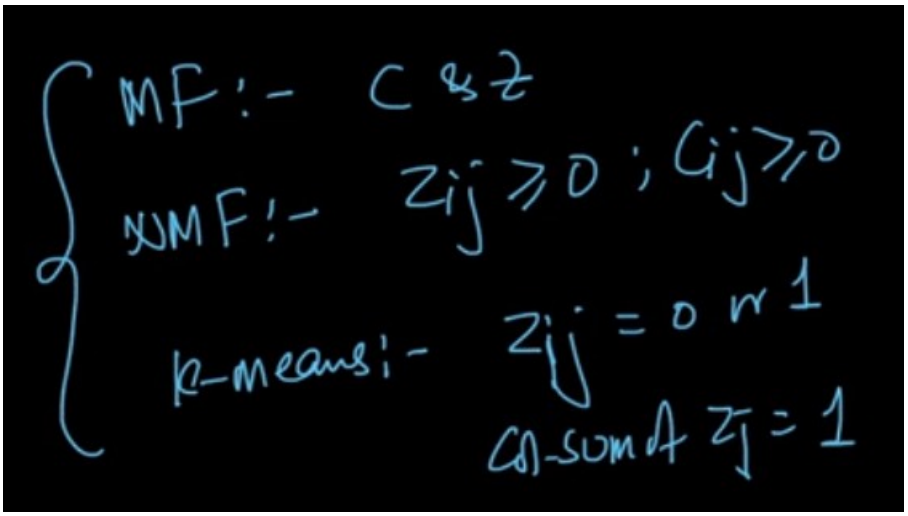

X =
x_1 x_2 x_j x_n
d x n
C =
c_1 c_2 c_i c_k
d x k
Z → $z_{ij} = \begin{cases} 1 & \text{if } x_j \in S_i \\ 0 & \text{o/w} \end{cases}$

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

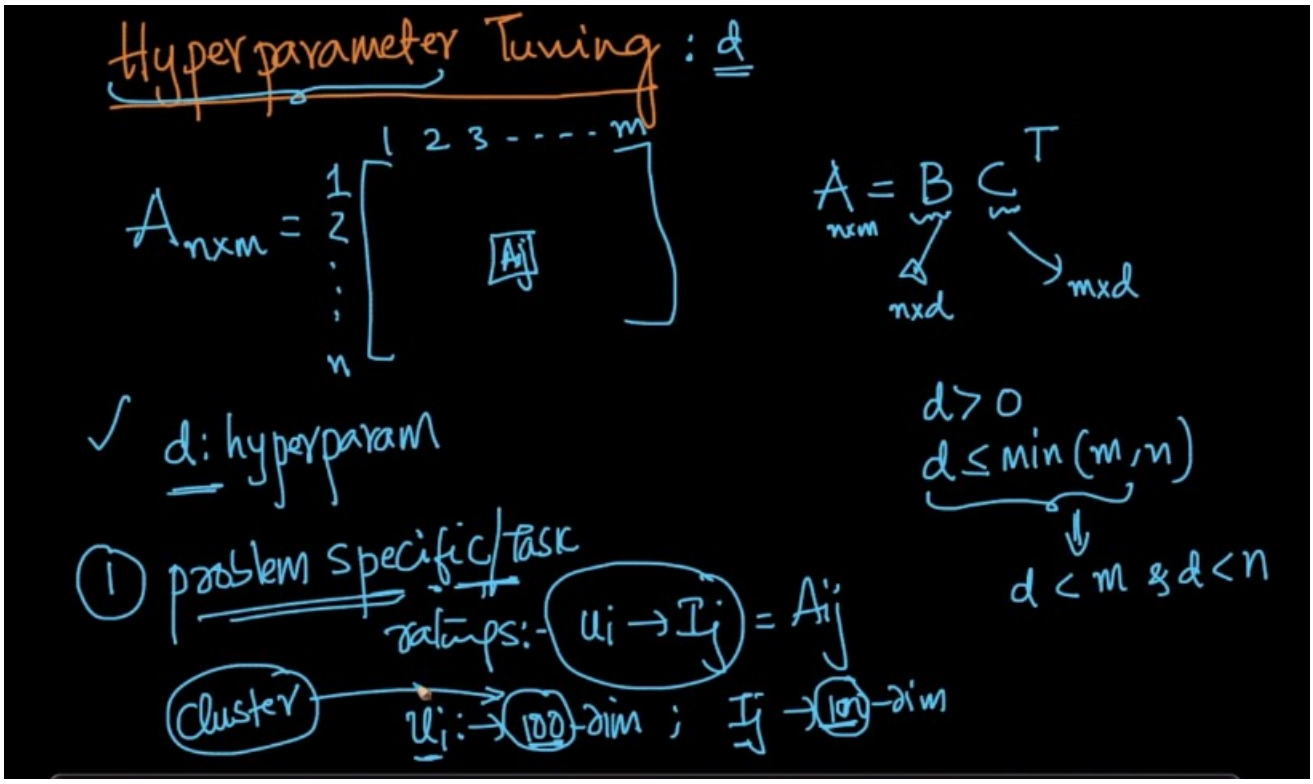




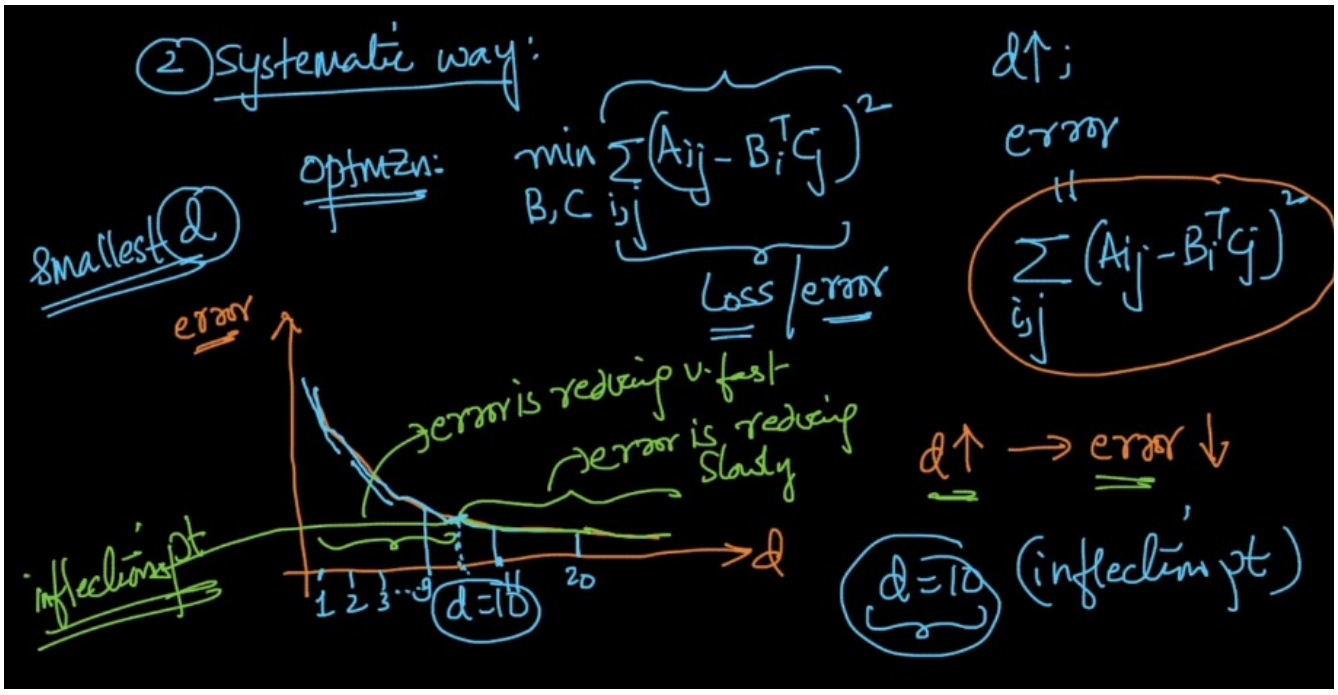

MF ≥ K-means
X = C * Z
min C,Z Σij (Xij - Ci * Zj)^2
s.t Zij = 0 or 1
1
2
Zj : col. of Z
sum of elts of Zj = 1
col. constraints
K-means = MF + 1-0 constraint
K-means → NP-hard


MF :- C & Z
NMF :- Zij ≥ 0 ; Cij ≥ 0
K-means :- Zij = 0 or 1
col-sum of Zj = 1

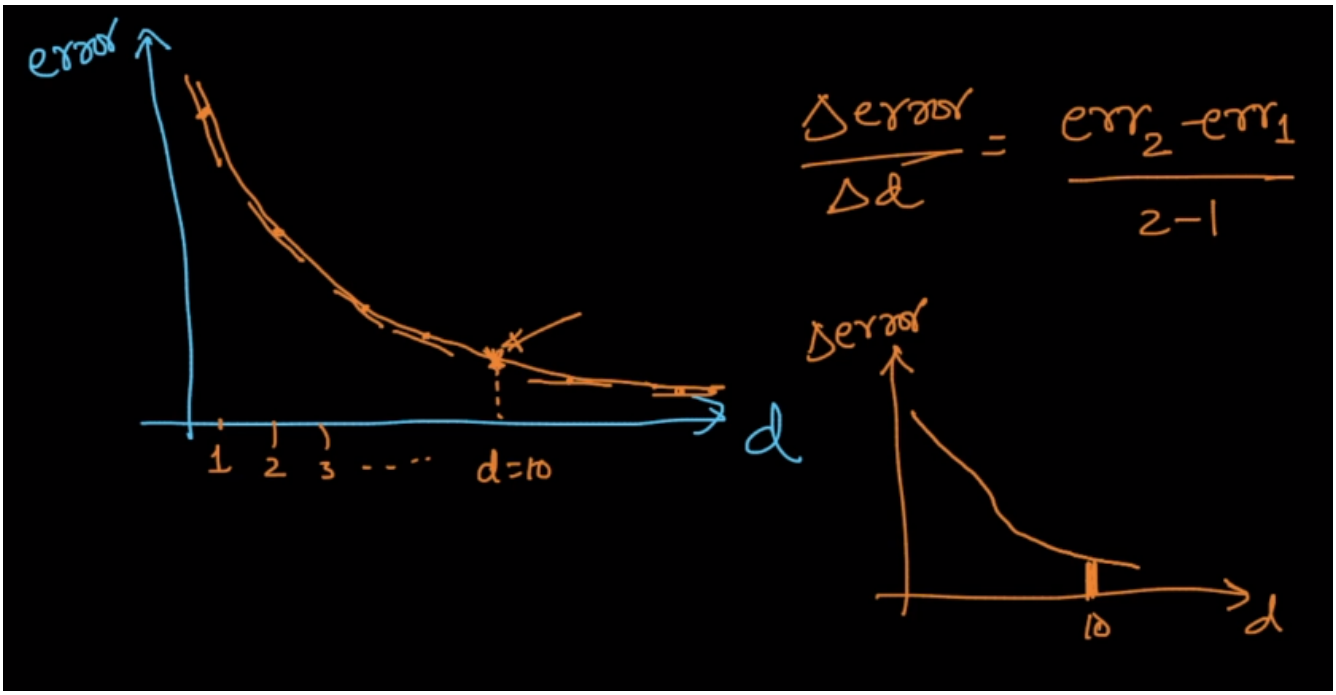
Hyper parameter tuning:
The d should be greater than zero.
Determining right D:



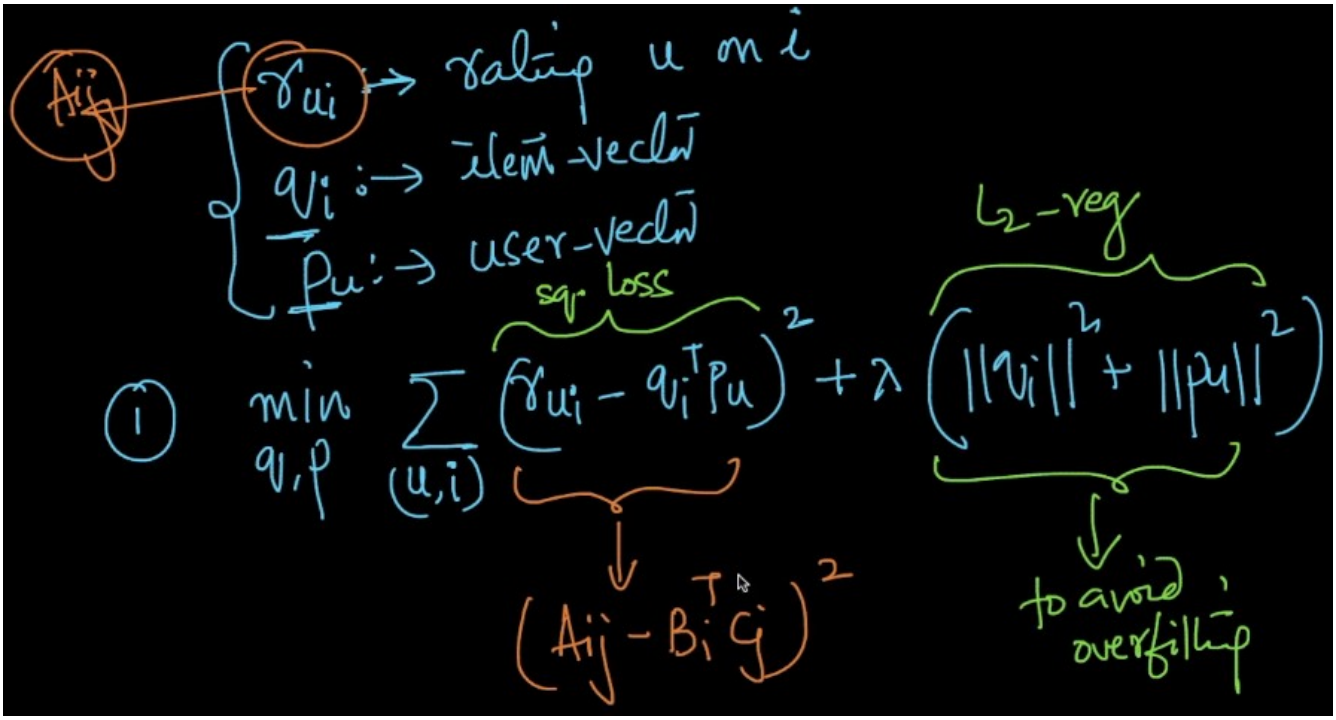
But as usual we have systematic way for finding best ‘d’:
The Error is reducing very fast at the beginning and slow at the ending. This point is called inflection point where the rate of change of error is less decreasing.



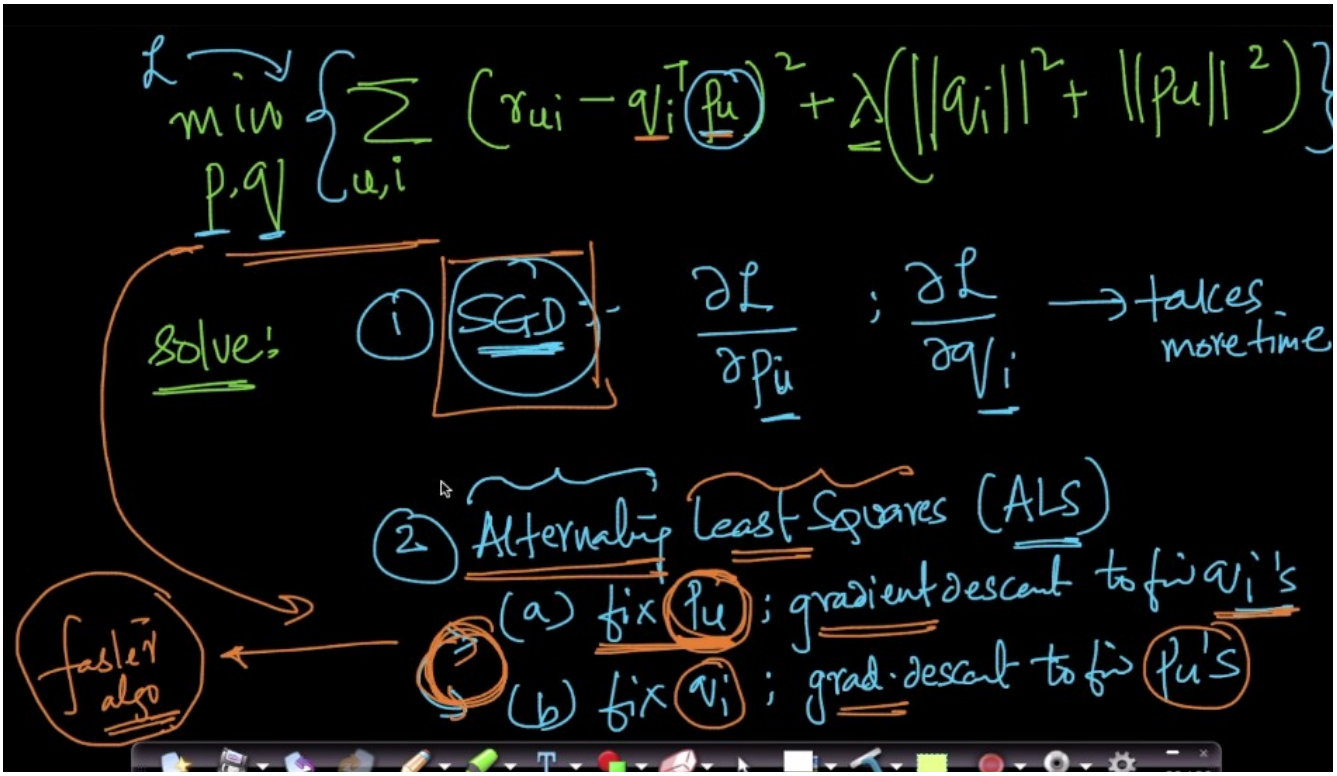
We will take the gradient plot in this case for choosing the right K.
We will find the inflation point by looking at visually the plot.



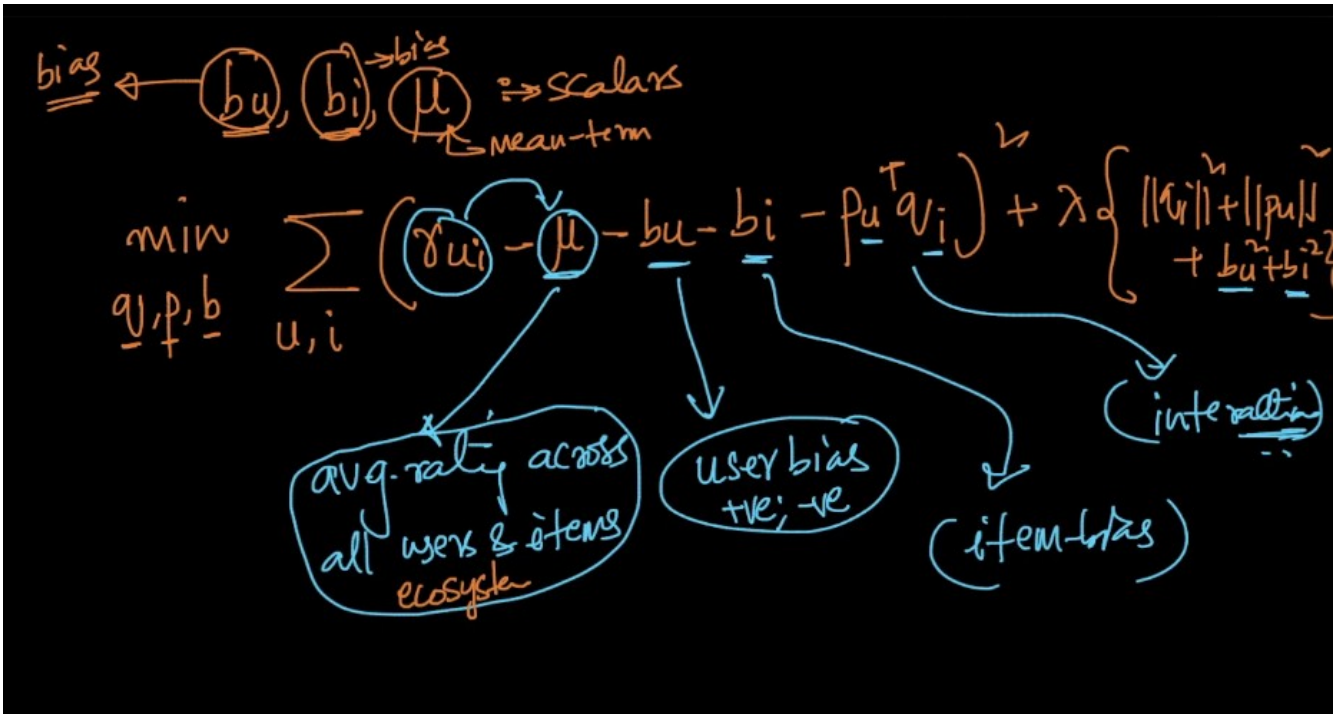
Netflix prize:
That streams video on demand. They gave the loss metric – root mean square error.



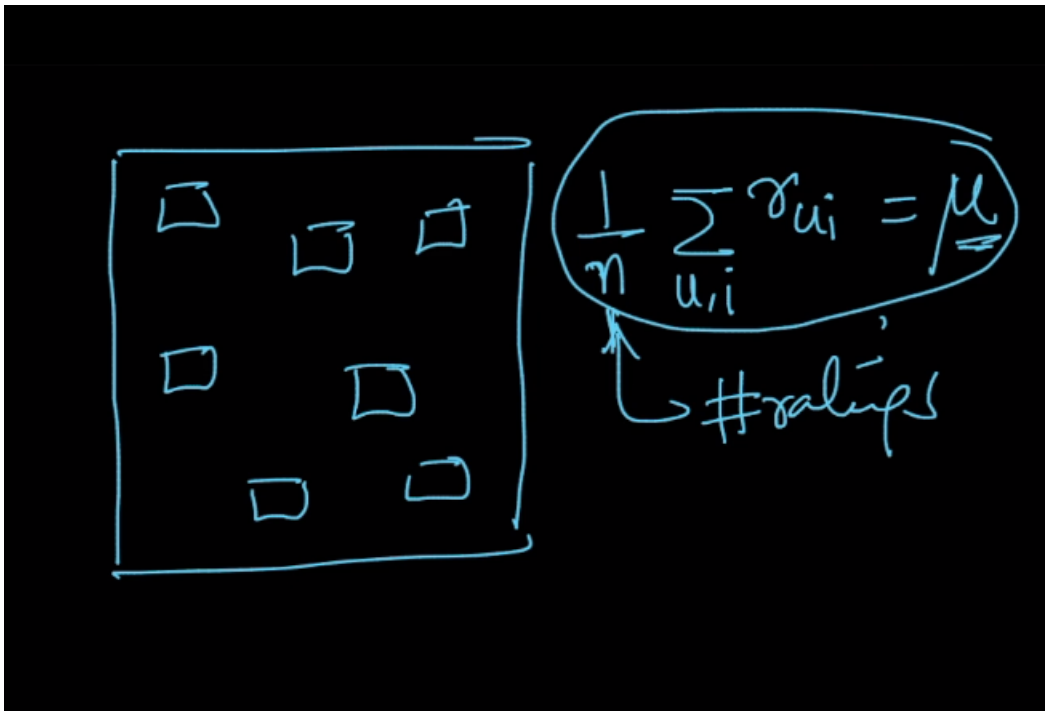
In netfilx MF we use the regularization.



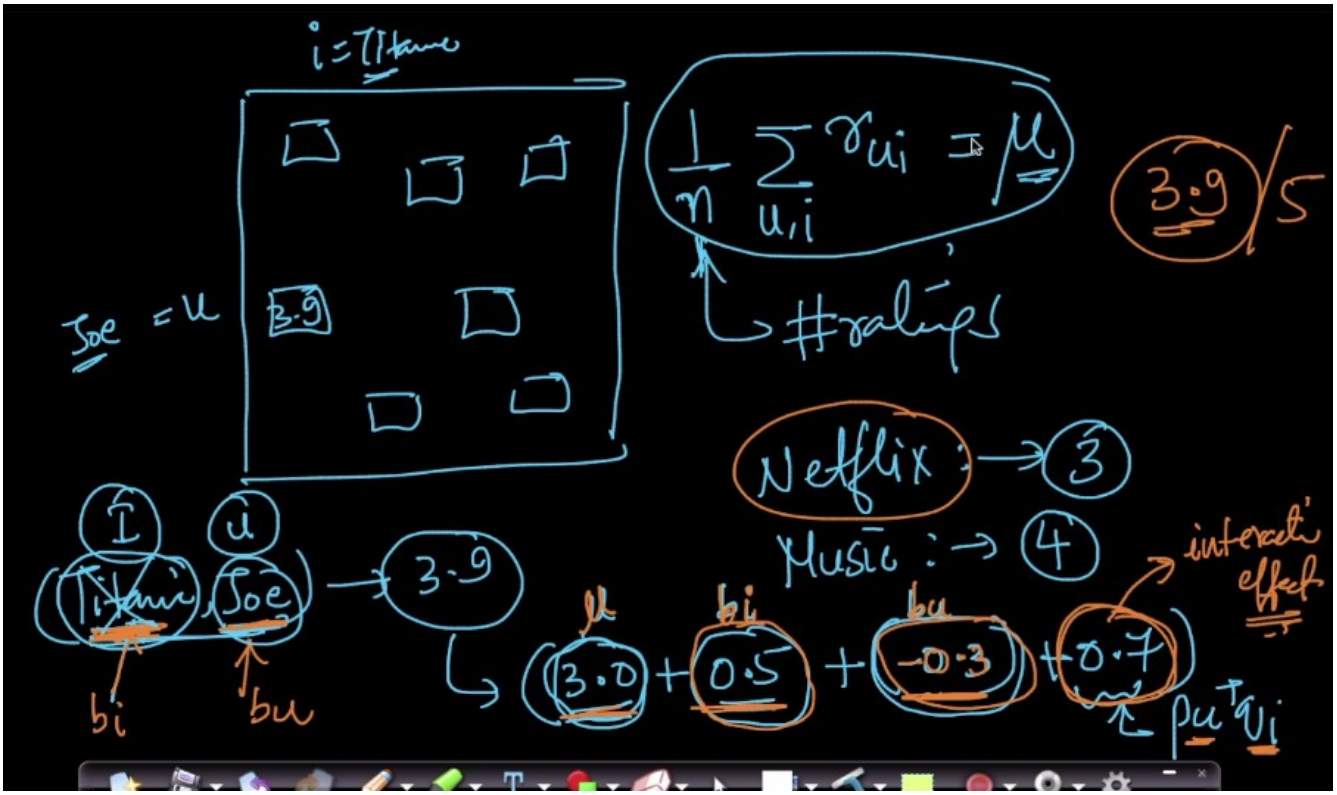
We use the alternating least squares as the approach for optimizing the loss function.

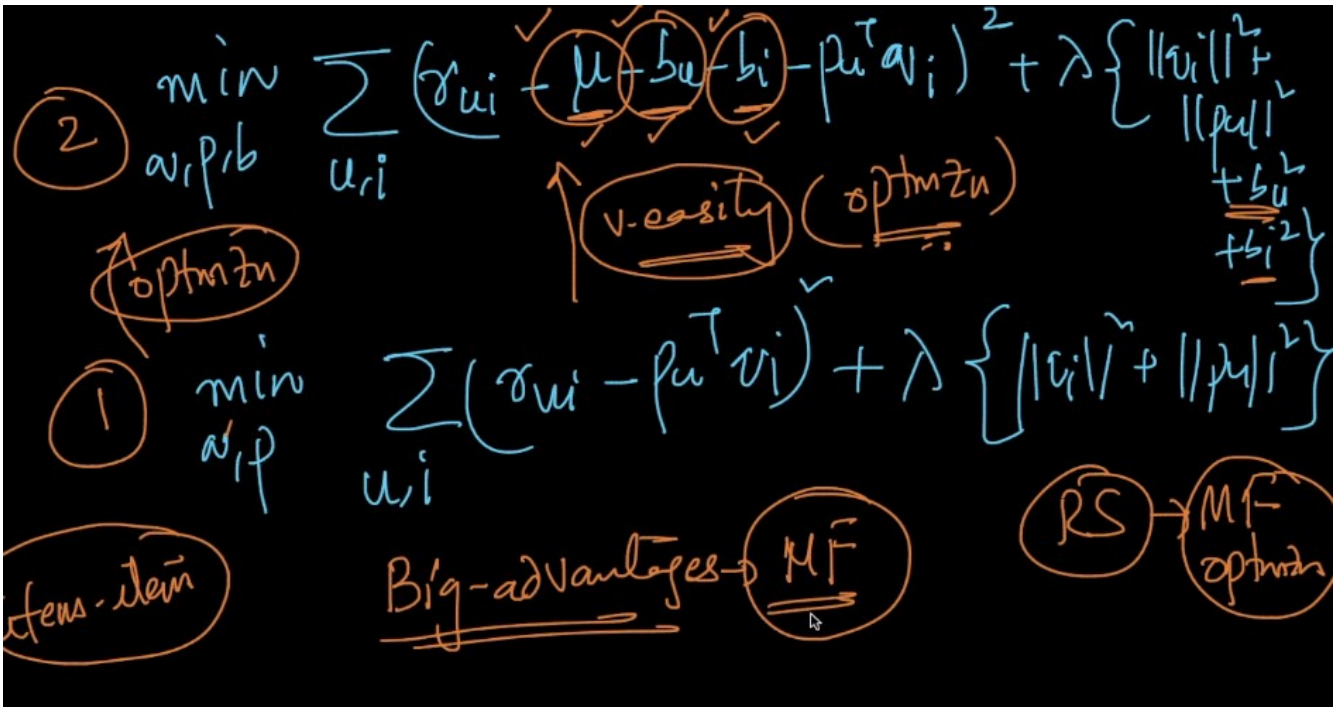


The U:

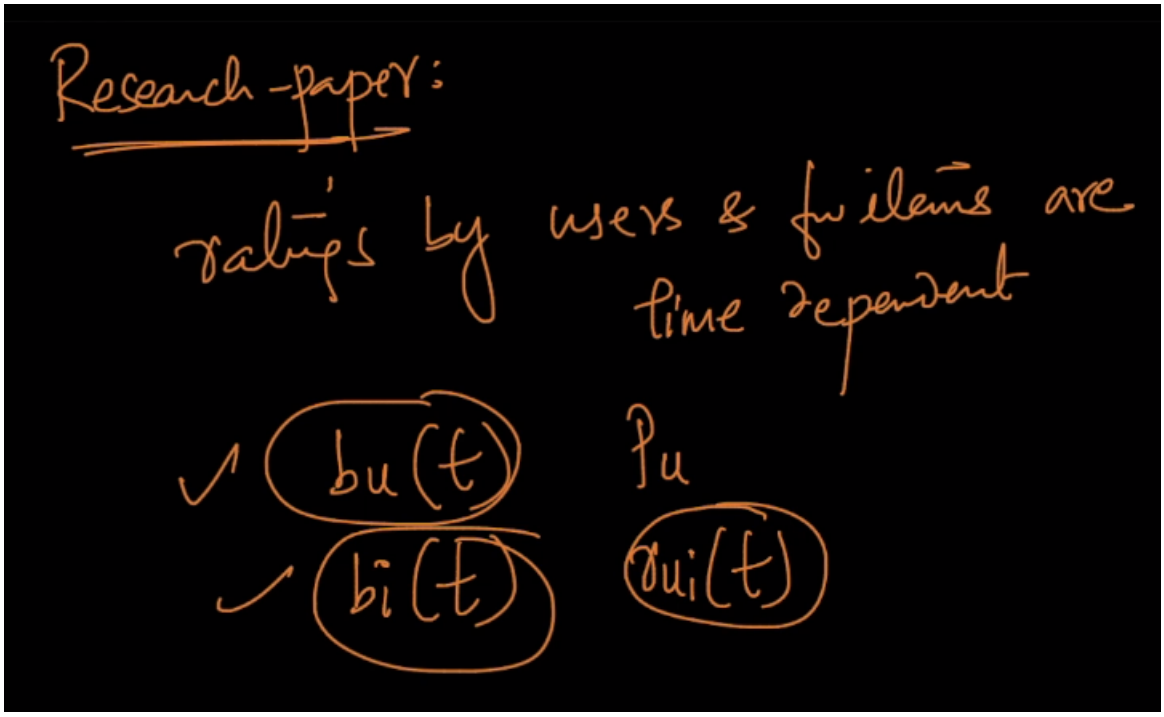


There is no point in determining the average rating, titanic movie is rated as follows.



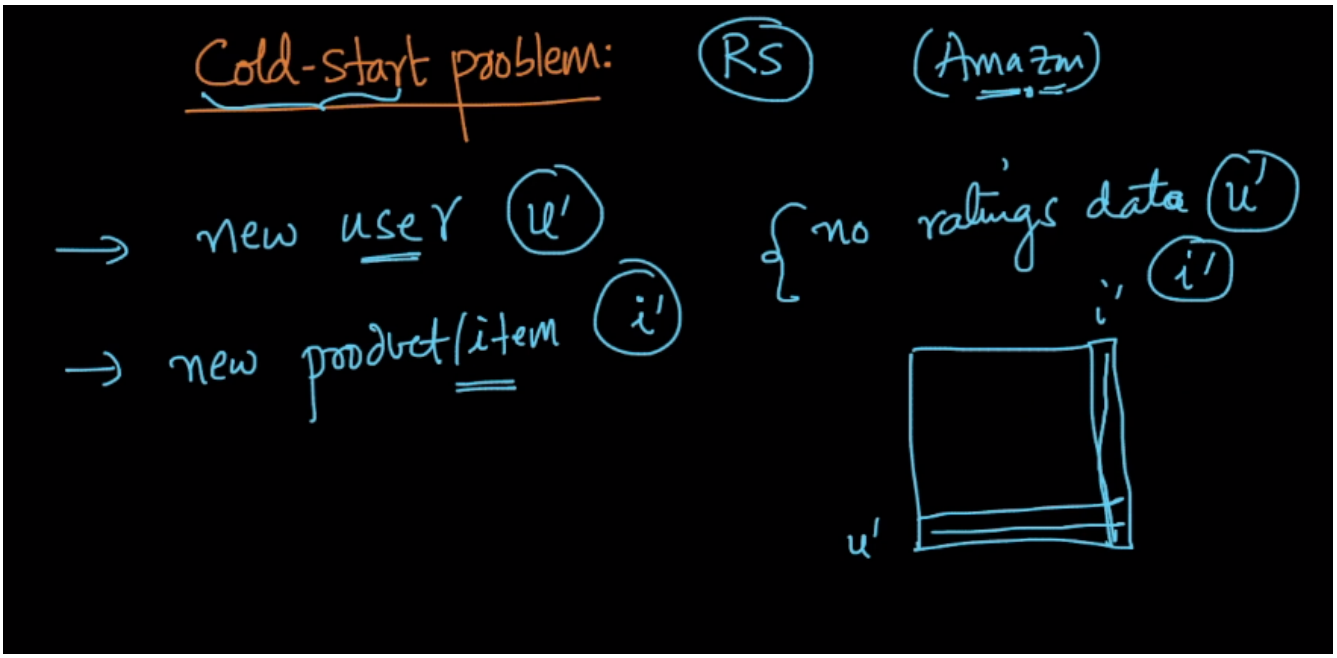


We can add the custom terms into the loss function.

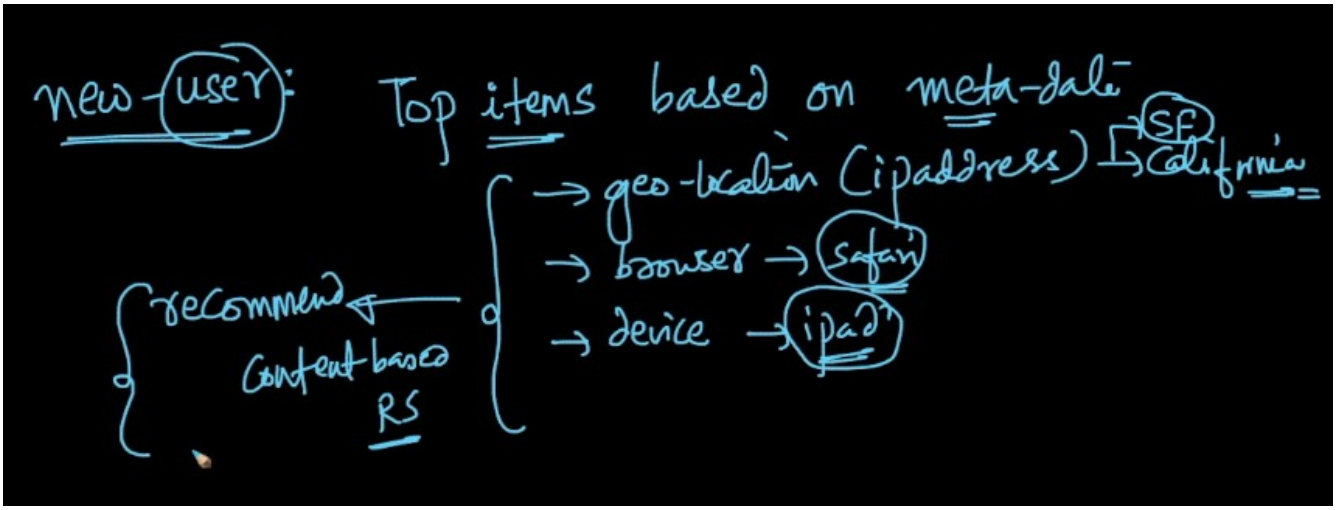


Cold problem:

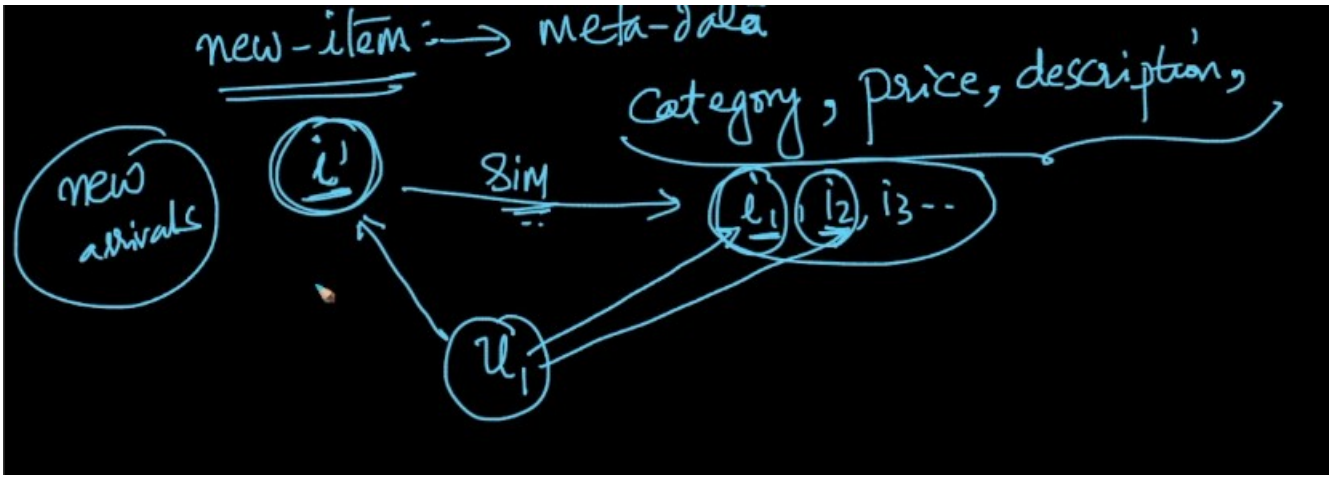
When a new user (or) item comes into ecosystem.



The practical way of doing is as follows;

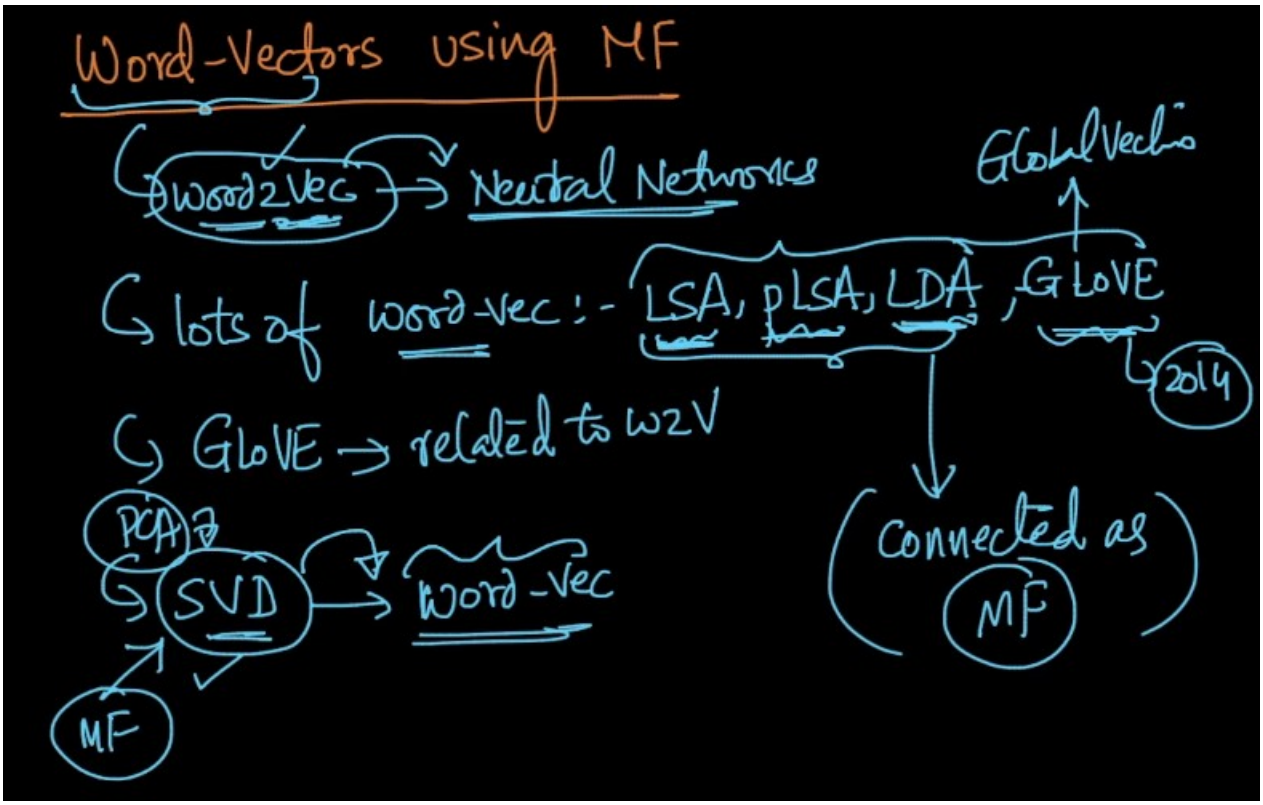


Similarity if we have the new item then meta-data -
Category, Price, description.
I1, i2, i3, i4.

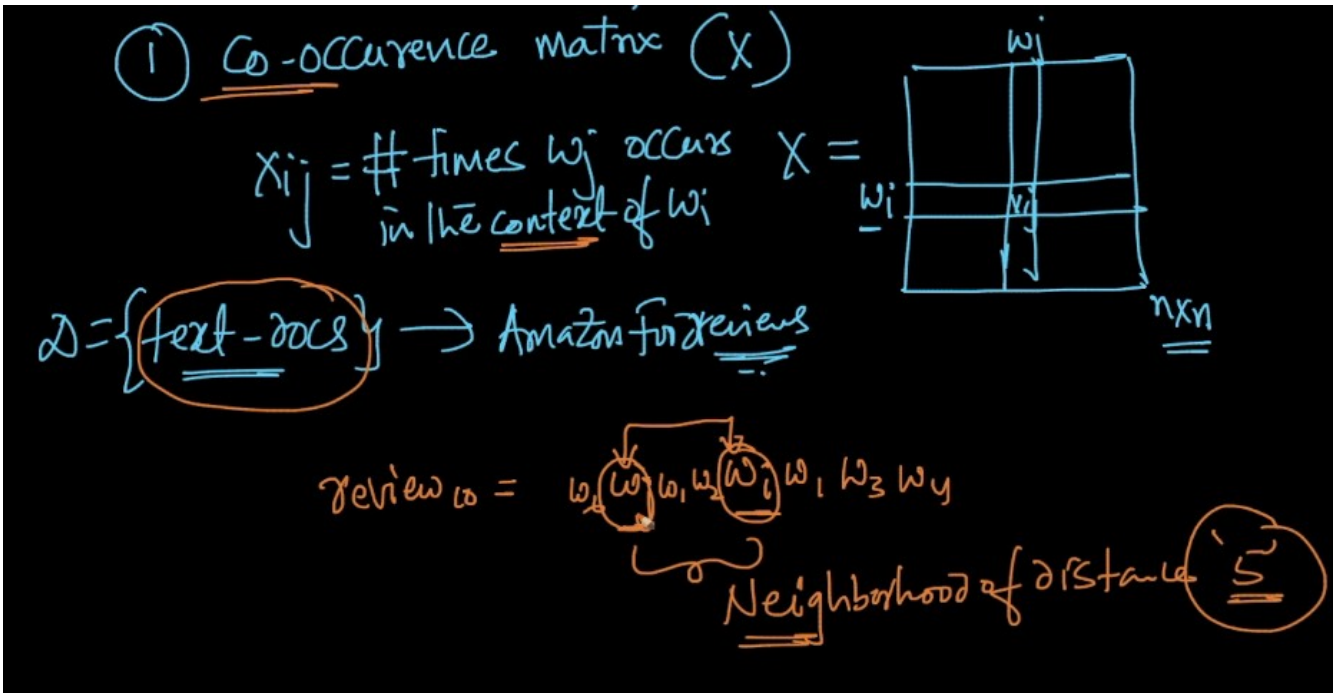


This problem is called cold start item.(content based approach).

Word vectors using MF:



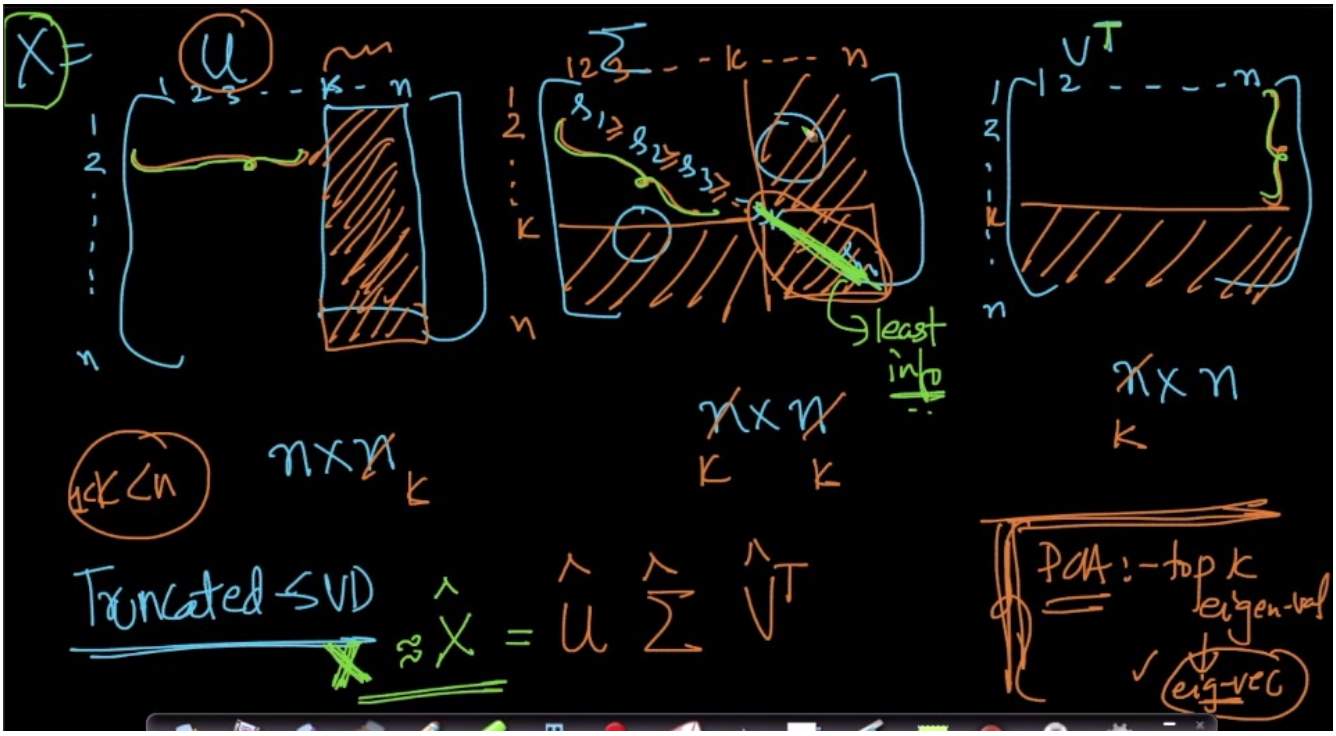
Co-occurrence matrix:

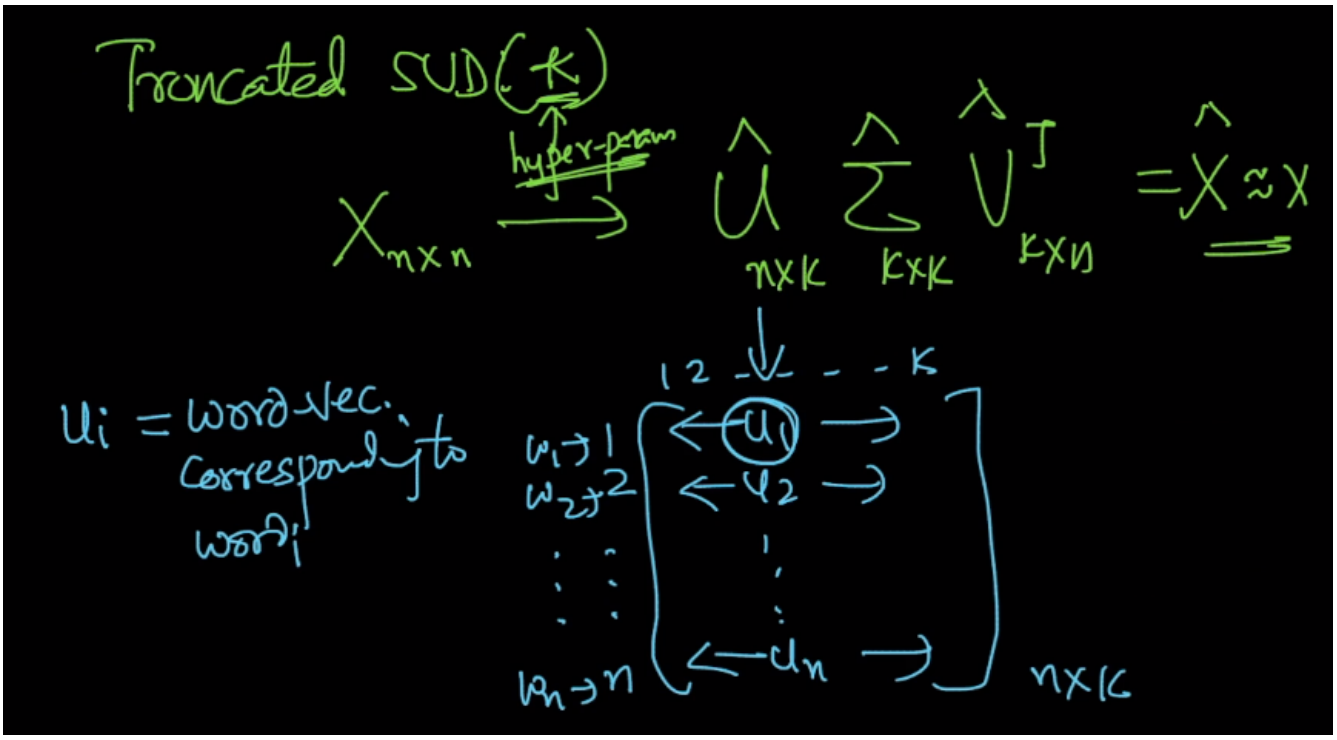


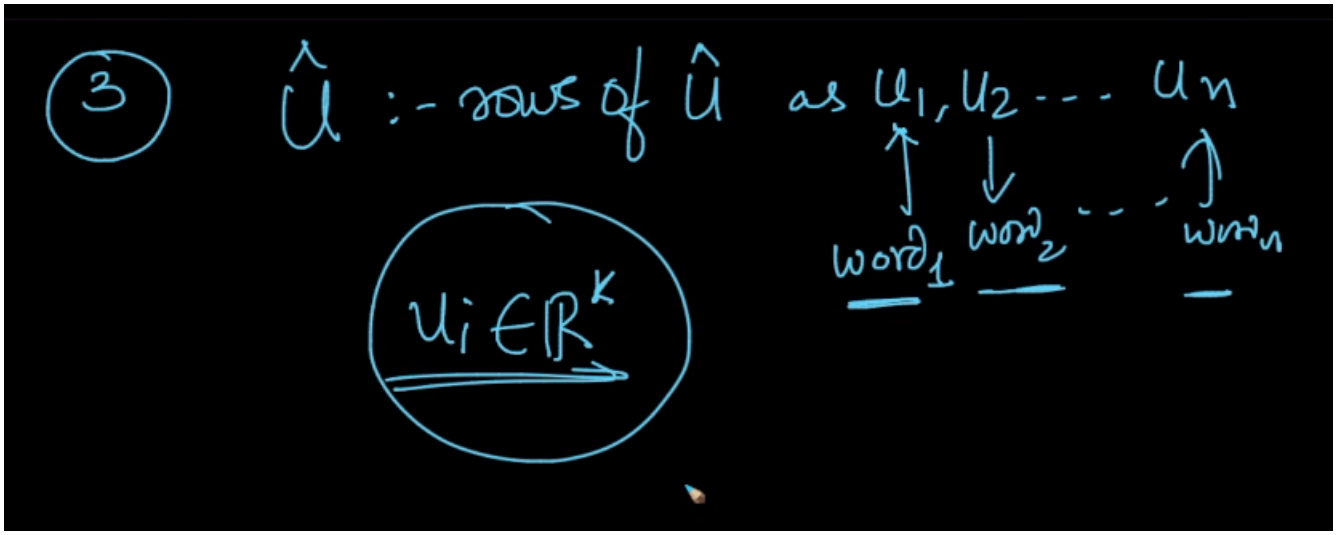
These two words are co-occurring across all the docs.

Truncated SVD:

What we discarded here is the least information



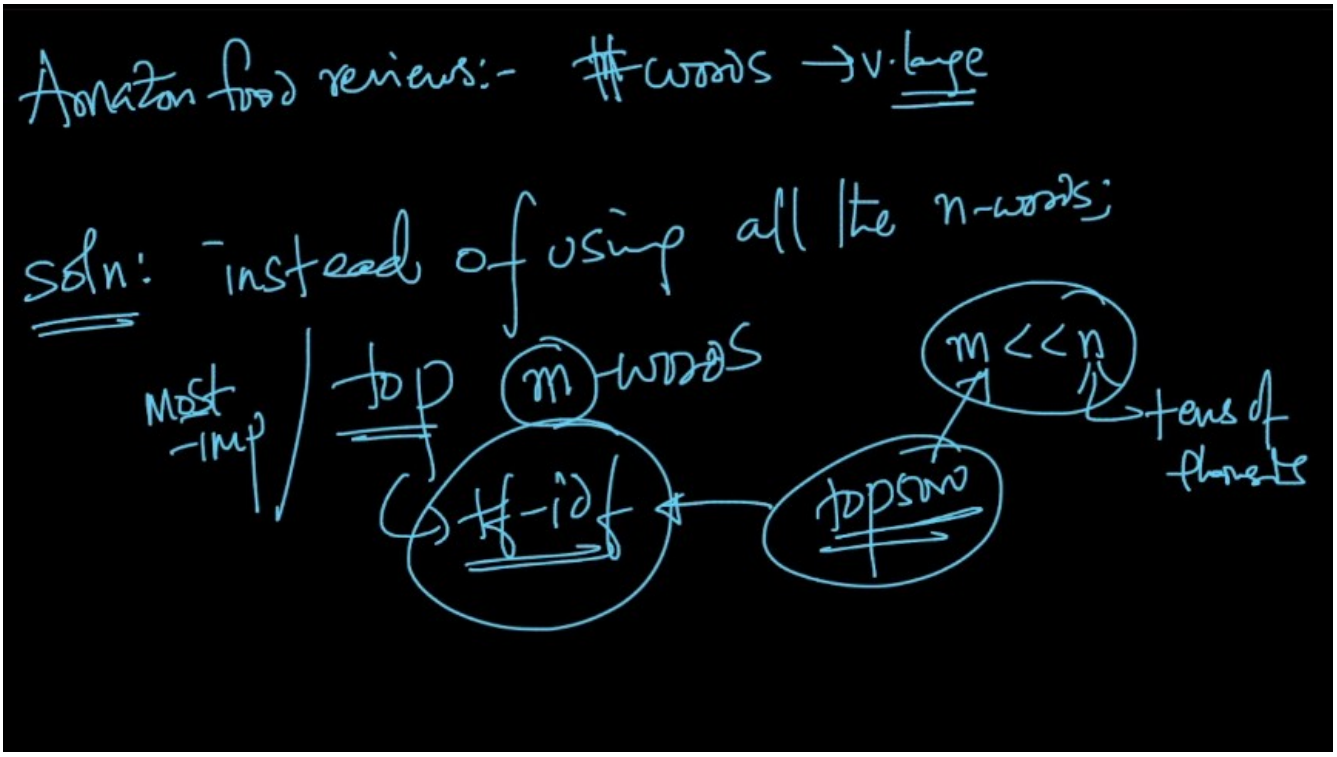




There is one problem here, coocurraence matrix + Truncated matrix.

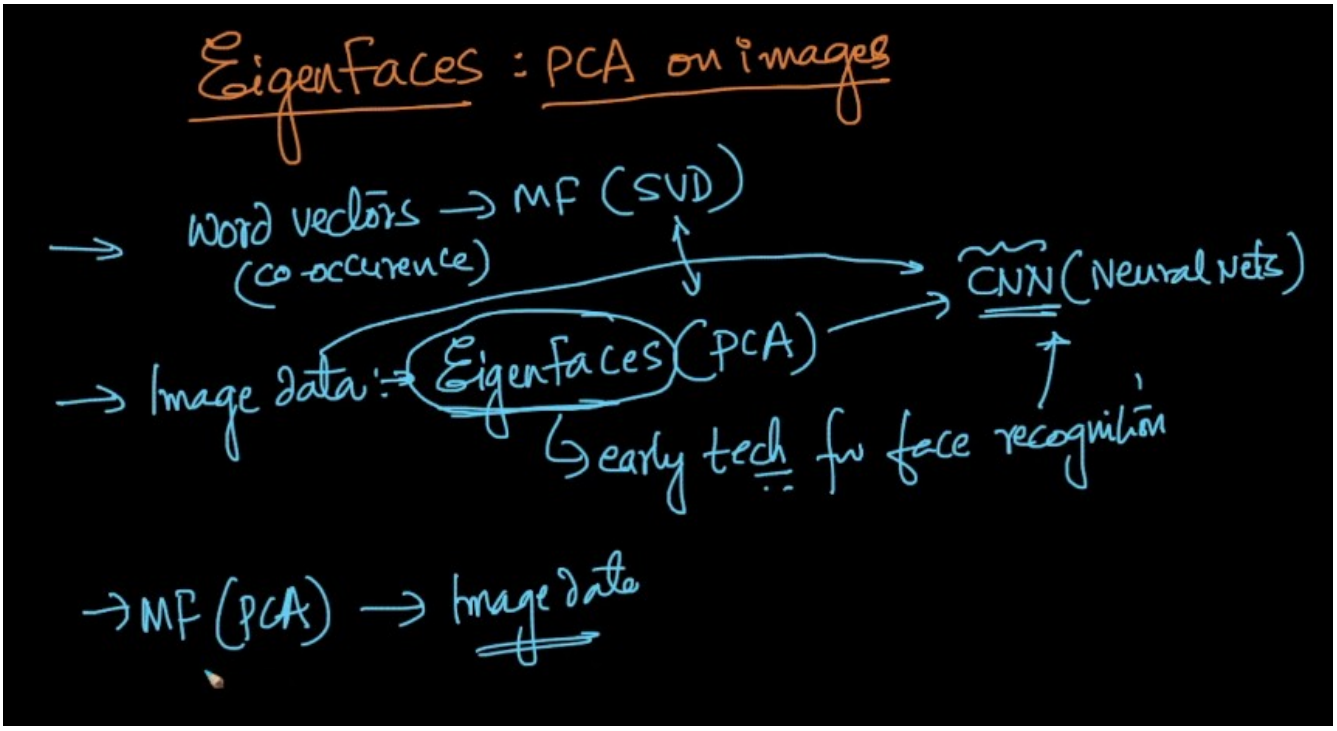
The number of words is very large, computing is very difficult.

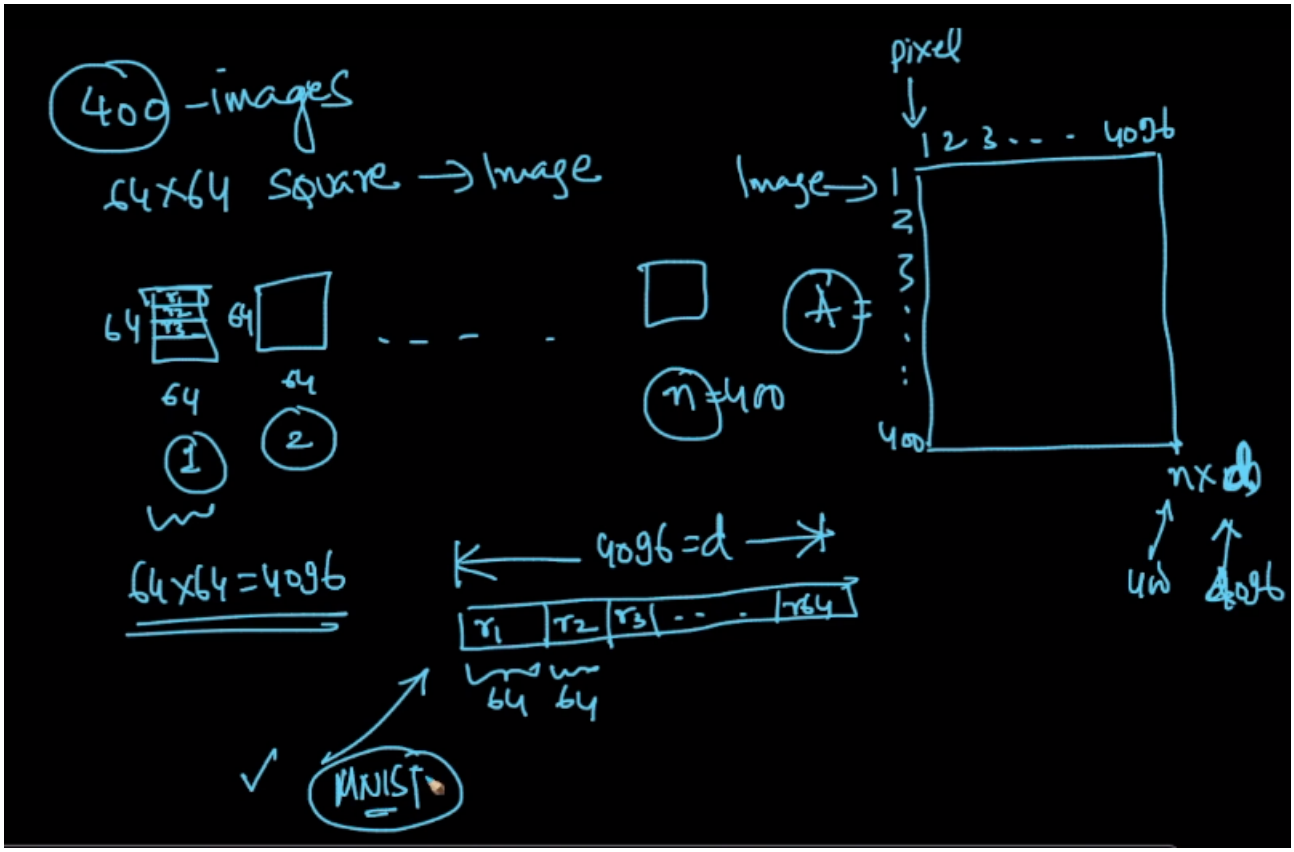
Instead of using all the words, lets use the top m-words by using the tf-idf.



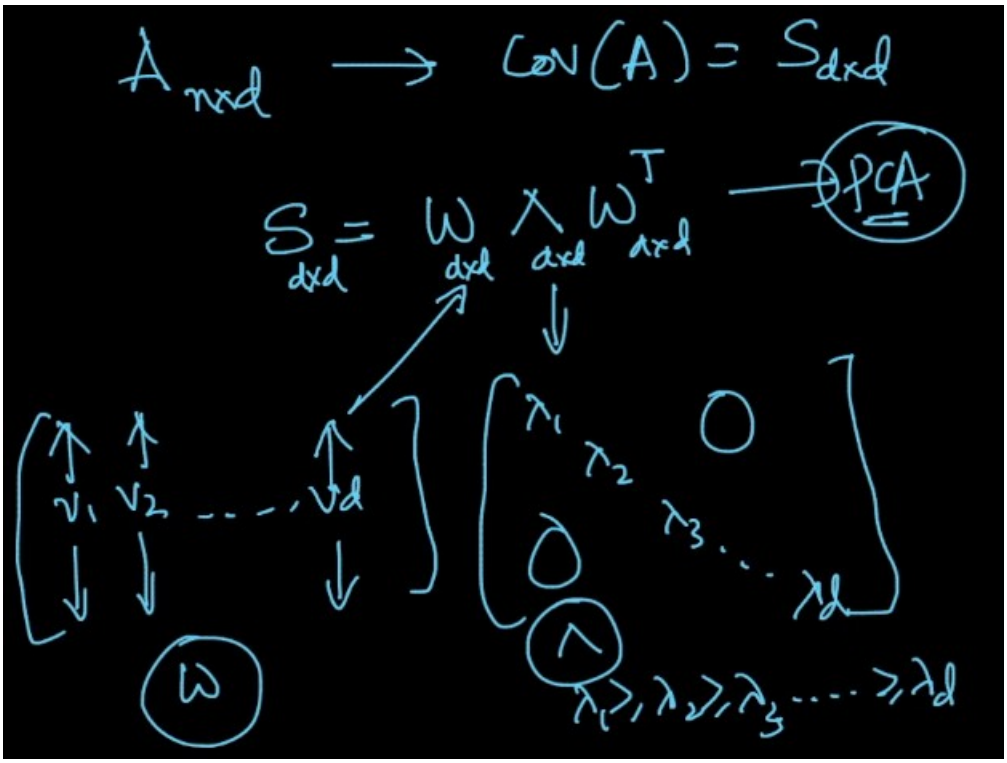
We could build the smaller matrix, on the subset of most important words.

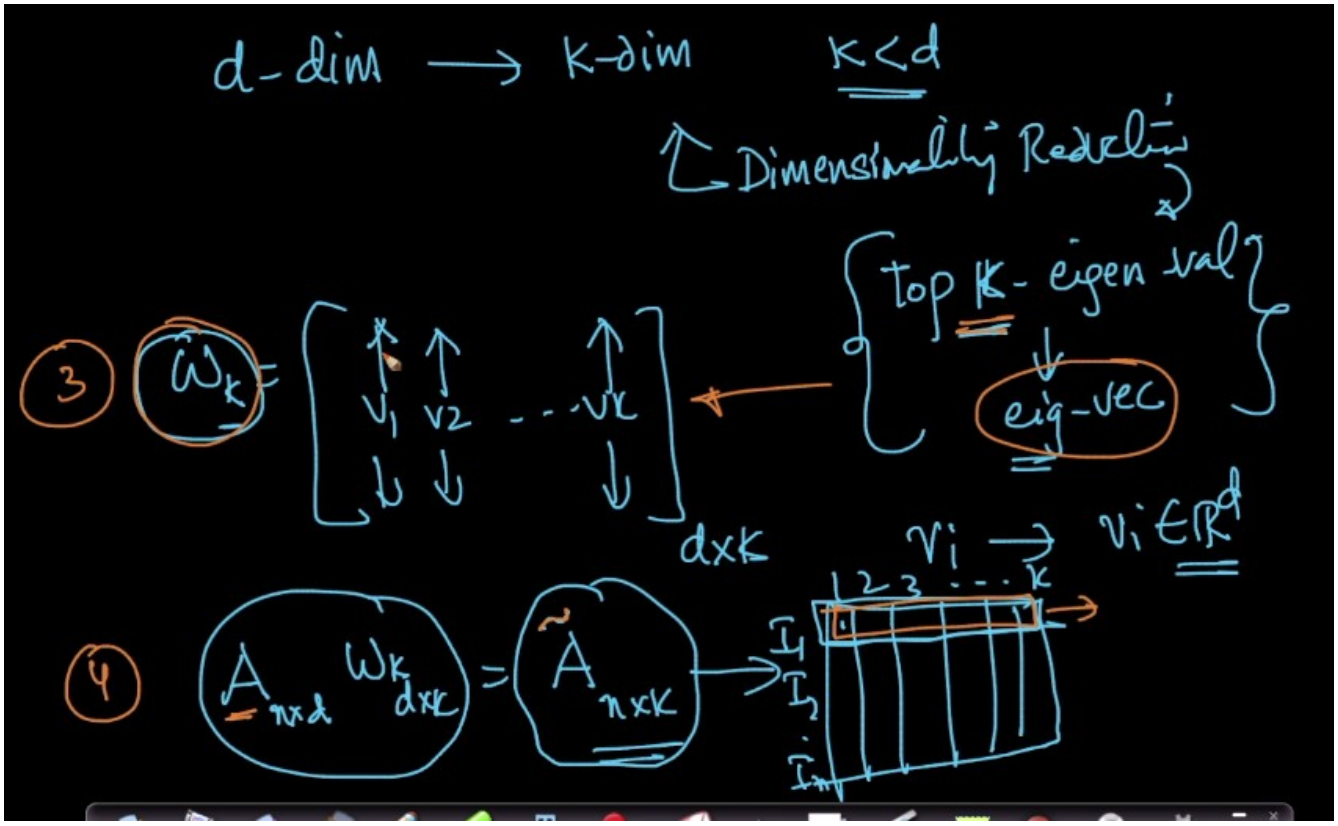
Eigen-Faces: For images we use the eigen faces is one of the early tech for face recognition.



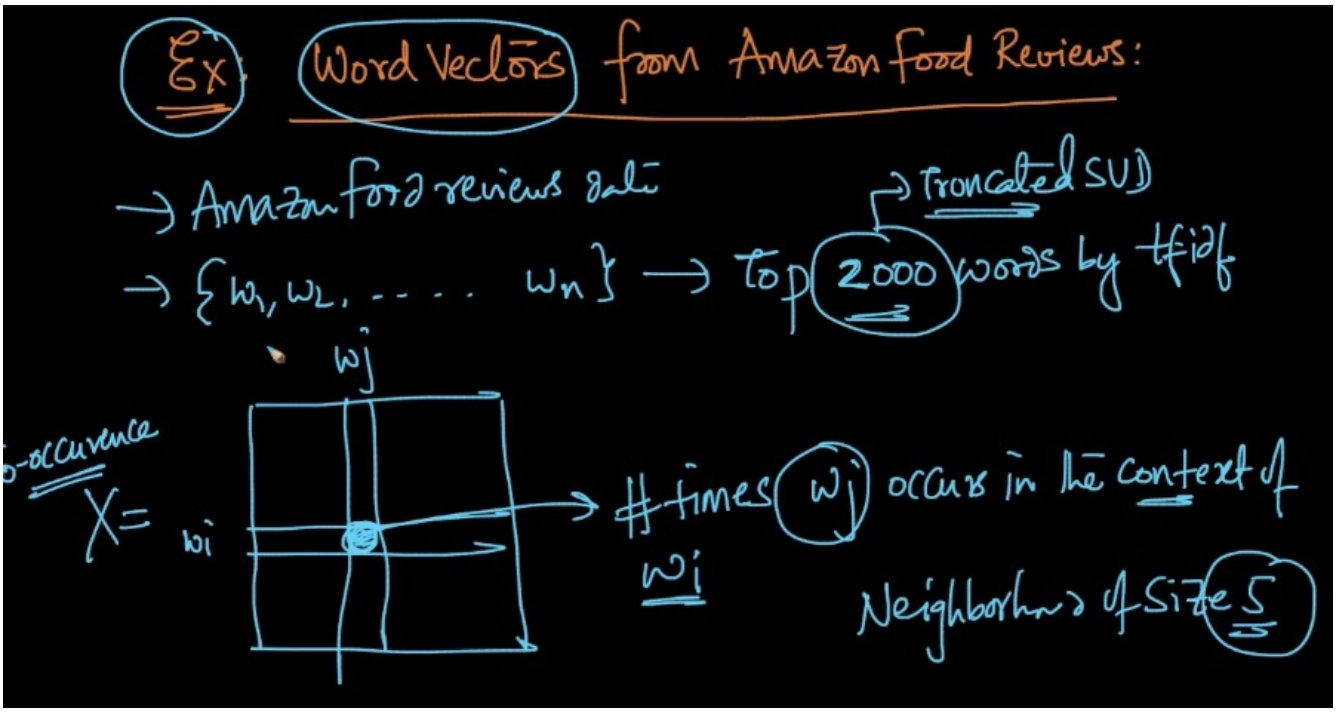


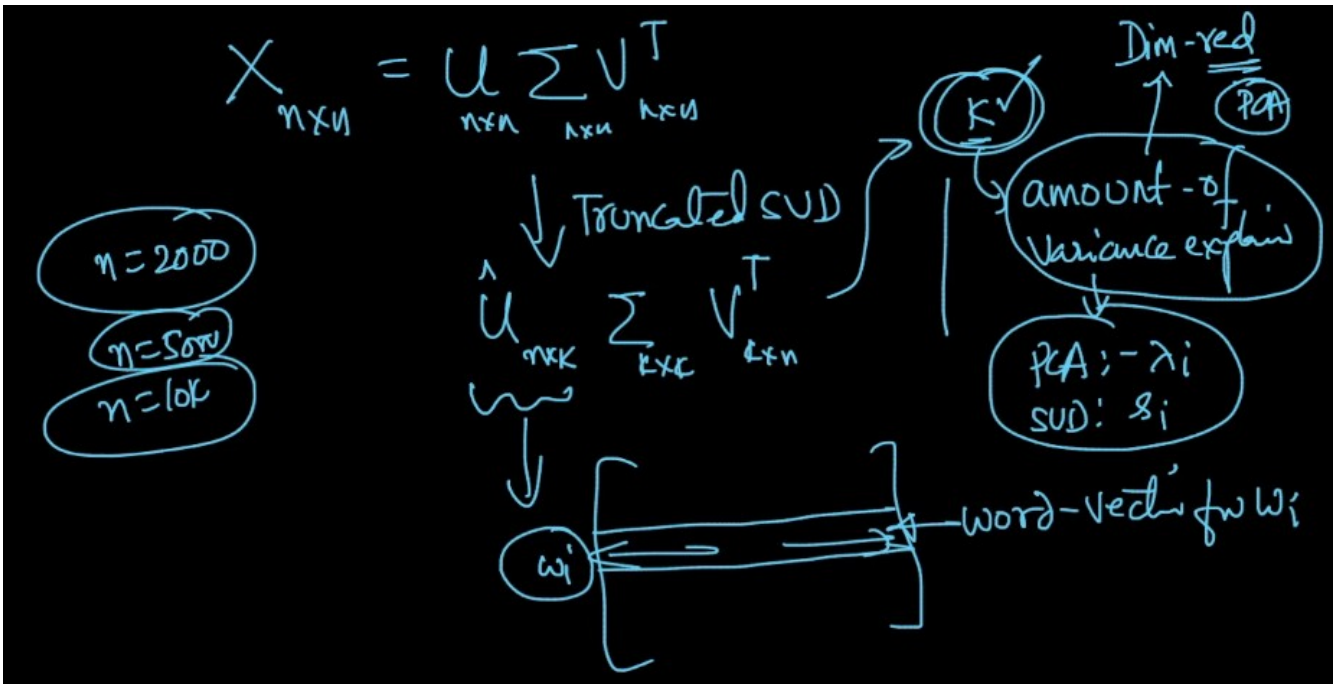
Given my matrix A, we can calculate A.

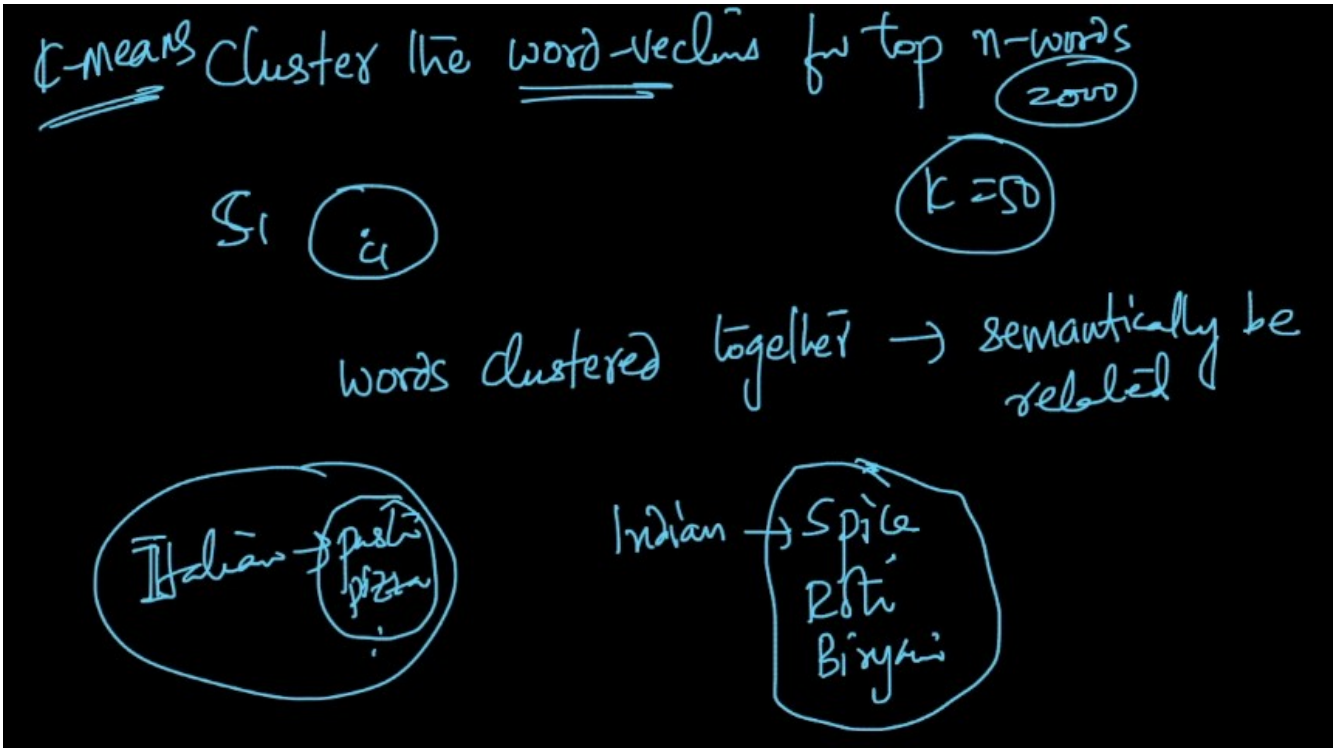




Each of the vectors vi are the ‘d’ dimensional vector, d=4096.
2000 words using tf-idf, coocurance matrix with neigh = 5 to wi.




$X_{n \times d} = U_{n \times n} \Sigma_{n \times d} V^T_{d \times d}$
Truncated SVD
$\hat{U}_{n \times k} \Sigma_{k \times k} V^T_{k \times n}$
n = 2000
n = 500
n = 100
k
Dim-red
PCA
amount-of variance explain
PCA: $-\lambda_i$
SVD: s_i
w_i
word-vector for w_i


K-means Cluster the word-vectors for top n-words
2000
s_i
i
k = 50
words clustered together → semantically be related
Indian
pasta
pizza
Indian
Spice
Roti
Biryani