Recommender System

Problem formulation :

$r(i,j) = 1$ if user j has rated movie i. {else 0}

$y^{(i,j)}$ = rating by user j on movie i (if defined)

$\theta^{(j)}$ = parameter vector for user j.

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

For user j, movie i, predicted rating :

$$= (\theta^{(j)})^T (x^{(i)}) =$$

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

To learn $\theta^{(j)}$:

$$\min_{\theta^{(j)}} \text{over } \frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T (x^{(i)}) - y^{(i,j)}\right)^2 + \frac{1}{2} m^{(j)} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

↓

least square regression to chose the parameter vector $\theta^{(j)}$ to minimize the squared term. $\frac{1}{2m^{(j)}} \sum (\theta_k^{(j)})^2$ is the regularization term to prevent $\theta^{(j)}$ from becoming too big. So, as $m^{(j)}$ is a constant, we can get rid of it. To learn $\theta^{(j)}$ (parameter for user j):

$$\min_{\theta^{(j)}} \frac{1}{2} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{1}{2} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

where n is the no. of features we have per movie and $\theta^{(j)}$ is a $n+1$ dimensional vector. we don't regularize over θ_0 i.e the biased term

To learn $\theta^{(j)}$ for all users :

$$\min_{\theta^{(1)} \dots \theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{1}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

n_u = no. of users

Gradient descent update :

$$\theta_k^{(j)} = \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} \quad (\text{for } k=0)$$

$$\theta_k^{(j)} = \theta_k^{(j)} - \alpha \Bigg(\sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \Bigg) \quad (\text{for } k \neq 0)$$

$$\frac{\partial}{\partial \theta_k^{(j)}} J(\theta^{(1)}, \dots \theta^{n_u})$$

The above approach to recommendation system is a content based recommendation approach as we assume we have available to us features for different movies, these features capture the content and zonar of the movie and we use that to make predictions.
But for many movies we don't actually have such features that can determine the content of the movie and so we use another recommender system i.e collaborative filtering for our predictions. It does feature learning i.e it learns itself what features to use.
Earlier we had feature vectors and using them we had to predict the rating of a movie by a user which he has not seen till now and recommend it to him.

But now the scene is different and even the feature vectors are unknown.

Problem:

Movie	A	B	C	D	x_1 (drama)	x_2 (action)
a	5	5	0	0	?	?
b	5	?	?	0	?	?
c	?	4	0	?	?	?
d	0	0	5	4	?	?
e	0	0	5	?	?	?

users

features

$\theta^{(1)}$ i.e a 3-dimensional parameter vector (n+1) i.e (2+1) of A is given as $\begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$,

$\theta^{(2)}$ is $\begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$, $\theta^{(3)}$ is $\begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$ and $\theta^{(4)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$

So, this time we have data regarding how much each user likes a particular type of movie i-e drama and action. This would help us guess the feature vectors x_1 and x_2 for different movies like if 'A' liked movie 'a' that means 'a' must not be a action movie but a romantic movie. because $\theta^{(2)}$ tells so. this is also confirmed by the ratings of 'C' and 'D' who hate romantic movies as seen in $\theta^{(3)}$ and $\theta^{(4)}$. So 'a' must be a romantic

movie for 'C' and 'D' to hate it.

So, here X i.e the feature vector is t
calculate. From above, we can say:

$$\left.\begin{aligned}(\theta^{(1)})^T x^{(1)} &\simeq 5\\ (\theta^{(2)})^T x^{(1)} &\simeq 5\\ (\theta^{(3)})^T x^{(1)} &\approx 0\\ (\theta^{(4)})^T x^{(1)} &\approx 0\end{aligned}\right\} \Rightarrow X^{(1)} = \begin{bmatrix} 1 \\ 1-0 \\ 0 \end{bmatrix}$$

Similarly fe
vectors for ot
movies can be c

Optimized algorithm:

Given $\theta^{(1)}, \dots, \theta^{(n_u)}$, to learn $x^{(i)}$:

$$\min_{x^{(i)}} \frac{1}{2} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{1}{2} \sum_{k=1} (x_k^{(i)})$$

Given $\theta^{(1)}, \dots, \theta^{(n_u)}$ to learn $x^{(1)}, \dots, x^{(n_m)}$:

$$\min_{x^{(1)} \dots x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{1}{2} \sum_{i=1}^{n_m} \sum_{k=1} (x_k^{(i)}$$

→ gives a reasonable set of features for all movies.

NOTE

The above two algorithms if used one after the other for several iterations i.e getting

$$\theta \rightarrow x \rightarrow \theta \longrightarrow x \longrightarrow \dots \text{ so on},$$

we will get a reasonable set of θ and x for the movies and the users. So, this is simultaneous learning of features and parameter vectors i.e θ and X.

this is possible in recommender system as hopefully each user rates multiple movies and each movie is rated by multiple users.

It is a sense of collaboration as every user is helping the system learn better features for the common good.

→ Collaborative filtering algorithm:

1. Initialize $x^{(1)}, \ldots x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)}$ to small random values.
2. Minimize $J(x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, \ldots, \theta^{(n_u)})$ using gradient descent (or an advanced optimization algorithm). e.g. for every $j=1, \ldots, n_u, i=1, \ldots, n_m$: minimize θ and k.
3. For a user with parameters θ and a movie with (learned) features x, predict a star rating of $\theta^T x$.

This algorithm can used to recommend similar products, e.g. if a user is looking a product than the algo can be used to recommend to him similar products.

The problem we were dealing earlier ie to predict the ratings of the users to suggest them movies of their type needs an

alternative solution. the problem can be reduced to a matrix i.e.

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix}$$

(5×4) → no. of movies, no. of users

Predicted Ratings:

$$\begin{bmatrix} (\theta^{(1)})^T(x^{(1)}) & (\theta^{(2)})^T(x^{(1)}) & \cdots & (\theta^{(n_u)})^T(x^{(1)}) \\ \vdots & \vdots & \cdots & \vdots \\ (\theta^{(1)})^T(x^{(n_m)}) & (\theta^{(2)})^T(x^{(n_m)}) & \cdots & (\theta^{(n_u)})^T(x^{(n_m)}) \end{bmatrix}$$

→ In this matrix (i, j) values give the predicted rating given by user j to movie i.

$$X = \begin{bmatrix} -(x^{(1)})^T- \\ -(x^{(2)})^T- \\ \vdots \\ -(x^{(n_m)})^T- \end{bmatrix} \qquad \Theta = \begin{bmatrix} -(\theta^{(1)})^T- \\ -(\theta^{(2)})^T- \\ \vdots \\ -(\theta^{(n_u)})^T- \end{bmatrix}$$

Low Rank Matrix Factorization Algorithm

5 most similar movies to movie i:

find 5 movies j with the smallest $\|x^{(i)} - x^{(j)}\|$.

where for each product i, we learn a feature vector $x^{(i)} \in \mathbb{R}^n$.

x_1 = romance, x_2 = action, x_3 = comedy .....

→ Mean Normalization:

the problem we were discussing had all the users who had rated atleast some movies. Now suppose there is a user 5 who has not rated any movie and we want to recommend him a movie. The resulting matrix is:

$$Y = \begin{array}{c} \begin{matrix} u_1 & u_2 & u_3 & u_4 & u_5 \end{matrix} \\ \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix} \end{array}$$

If we predict all the values of u_5 to be zero, that makes no sense because that would not help us to recommend him any movie. In such a case, mean normalization is used, where we compute the avg. rating that each movie obtained in a vector called μ.

$$\mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix}$$

Now, we will subtract of the mean rating from vector Y:

$$Y - \mu = Y = \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

So, we have normalized each movie to have an

avg. rating of zero. Now this new Y will be used to learn my parameter vector $\theta^{(j)}$ and features $x^{(i)}$, considering the new Y as my original data obtained from users.

For user j, on movie i predict:

$\rightarrow (\theta^{(j)})^T (x^{(i)}) + \mu_i$

for user 5:

$\theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ considering $n = 2$ i.e 2 features.

$(\theta^{(5)})^T (x^{(i)}) + \mu_i$ for user 5 $= \mu_i$

$= 0$ for user 5

so, this is sensible to predict μ_i as the rating given by user 5.

Any movie with no ratings is analogous to a user who has not rated any movie. In that case we will normalize the columns to have mean 0 instead of the rows. Also, if a movie has no rating, that shouldn't be recommended to any user anyway.

Mean normalization is a sort of pre-processing step for collaborative filtering.

The RMSE that calculates the error of the model suffers from following problem:

① Predic° diversity: If a user has watched Harry Potter 1 then he would be recommended rest parts of Harry Potter only thus lacking on diverse recommend°

② Prediction content: If a user is manip to us, he buys a lot of books on US but once he is back he is not interested in those books anymore. But the recommend° would include US books.

③ Order of predictions: If a user has seen Sherlock season 1 then the order of recommendations should be sequential i.e 2 followed by 3 followed by 4 and so on.

④ In practice we care only about high ratings but RMSE does not distinguish b/w high and low ratings. Being root mean square error it checks for the difference b/w actual and predicted values. So, to solve this we build a system which checks if my model correctly predicts the higher rated movies by

a user, so that the model can correctly predict a larger fraction of the high rated movies.

A better approach to Recommender Systems [Singular Value Decomposition or [Spectral Decomposition of a Matrix]]

Note Rank of a matrix A is the no. of linearly independent columns of A

Eg. Matrix $A = \begin{bmatrix} 1 & 2 & 1 \\ -2 & -3 & 1 \\ 3 & 5 & 0 \end{bmatrix}$ has rank = 2

bcoz row1 and row2 are linearly independent but all three are linearly dependent.

$R_1 = R_2 + R_3$

So, the data is given to us in 3 Dimensional but the matrix is actually 2-dimensional.

We can write A as two 'basis' vectors : [1 2 1] [-2 -3 1]

and all the rows as a combination of these two vectors : [1 0] [0 1] [1 1]

So, we have reduced the no. of coordinates in our new coordinate system.

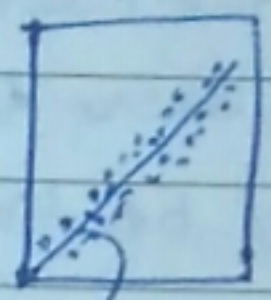
earlier: [1 0 0] [0 1 0] [0 1 1]

now : [1 0] [0 1] [1 1] with new coordinate

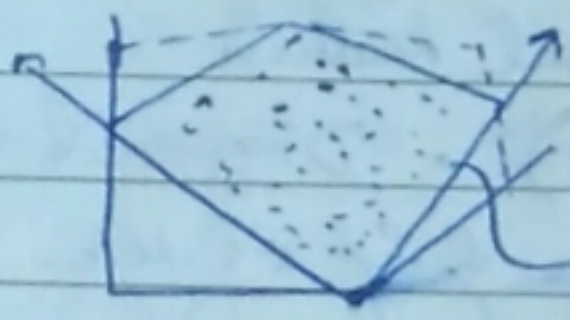
system : [1 2 1] [-2 -3 1]

two vectors whose linear combination can represent all the data points.

Note Goal of dimensionality reduction is to discover the axis of data.



Rather than representing every point ∈ two coordinates, we represent each point with 1 coordinate w.r.t. line (axis).



This is bcoz data is in high dimensions but occupy only a part of the dimensional space, i.e a subspace. This dimensionality reduc^n aims at reducing dimension while incurring as little error as possible and maintaining the richness of data.

- Why reduce dimensions?

1) Discover hidden correlations
2) Remove redundant & noisy features
3) Interpretation and visualization
4) Easier storage and processing of data.

SVD is the first data dimensionality reduction matrix.

$$A_{[m \times n]} = U_{[m \times r]} \Sigma_{[r \times r]} (V_{[n \times r]})^T$$

A: input data matrix.

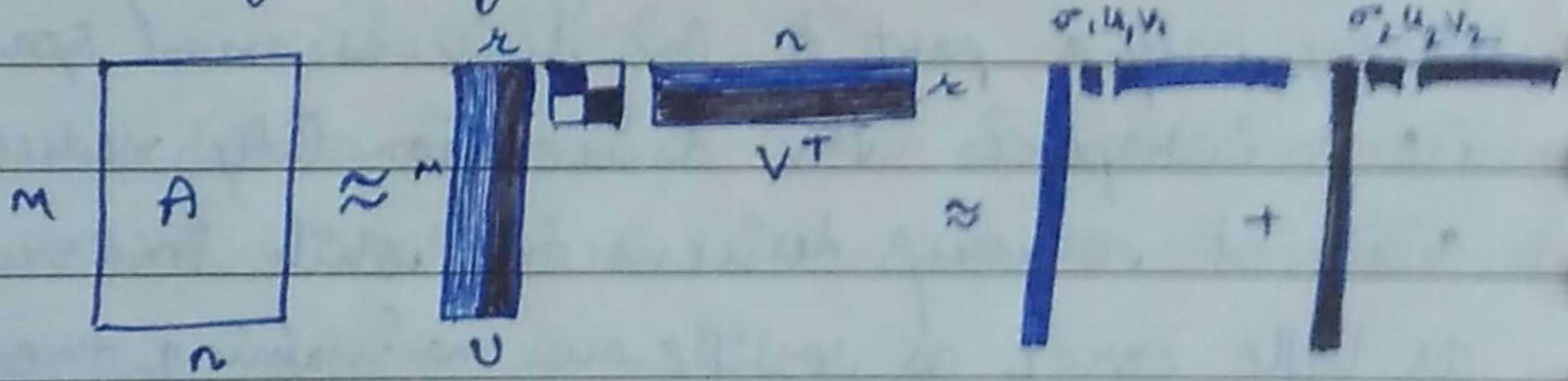
We can consider m rows as the user and n columns as movie and 0 if user did not watch the movie while 1 if he did.

U: left singular vectors

Σ: Singular values ; digonal matrix

r (rank of matrix A) . Diagonal elements are sorted in dsip order.

V: Right singular vectors.



$$A \approx U \Sigma V^T = \sum_i \sigma_i u_i \circ v_i^T$$

σ_i: scalar, u_i: vector, v_i^T: vector

- SVD theorem says it is always possible to decompose a real matrix A into this product of matrices U, Σ and V^T, containing real values and not complex numbers
- SVD theorem says U, V and Σ are unique.

- U and V are <u>column orthonormal</u> i.e columns of U and V have euclidian length 1. So the sum of the squared values in each column of these two matrices = 1.
- U and V are orthogonal i.e if I take two columns of U or two columns of V and multiply them with each other, the dot product is zero.
- Σ (diagonal values only are there, rest all are 0) Entries are +ve and stored in sorted order.

$$\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \sigma_4 \geq 0$$

eg. Users to Movies.

	A	B	C	D	E
a	1	1	1	0	0
b	3	3	3	0	0
c	4	4	4	0	0
d	5	5	5	0	0
e	0	2	0	4	4
f	0	0	0	5	5
g	0	1	0	2	2

(Original)

This matrix has 5 columns but a pattern can be seen. A, B, C seem to be alike and D, E seem to be alike. a, b, c, d are alike and e, f, g are alike.

Suppose, A, B, C are sci-fi while D, E are romance.

The above matrix is decomposed as:

$$\begin{bmatrix} 0.13 & 0.02 & -0.01 \\ 0.41 & 0.07 & -0.03 \\ 0.55 & 0.09 & -0.04 \\ 0.68 & 0.11 & -0.05 \\ 0.15 & -0.59 & 0.65 \\ 0.07 & -0.73 & -0.67 \\ 0.07 & -0.29 & 0.32 \end{bmatrix} \times \begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 1.3 \end{bmatrix} \times \begin{bmatrix} 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\ 0.40 & -0.80 & 0.40 & 0.09 & 0.09 \end{bmatrix}$$

(first 4 users belong heavily to the sci-fi concept; strength of sci-fi concept; strength of romance concept)

sci-fi | Romance

U is a user to concept matrix i.e how much a user belong to a concept.

Σ gives the strength of the concept.

V^T gives the movie to concept matrix i.e how much a movie belongs to a concept.

V matrix shows 1st 3 movie belong heavily to the sci-fi concept and last 2 movies to the romance concept.

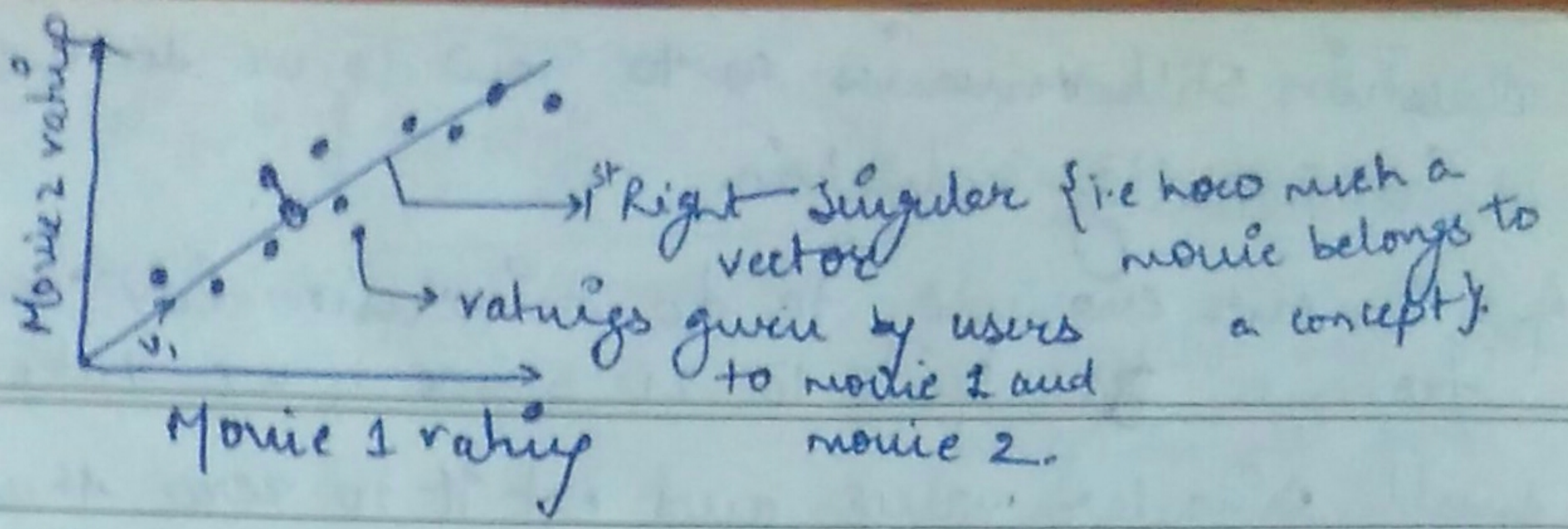
The strength of the 3rd concept is 1.3 and so we ignore it considering it to be a noise in our data.

So, SVD helps interpret data in terms of movies, concepts and users.

↳ different genres or topics

Note How do we discover that our movie data really had only two real kind of strong concept and the last 3rd concept was more like noise and it was okay to remove it from our analysis. So, how SVD actually works and how do we think about it in terms of dimensionality reduction and what SVD really does is:

1) It gives us the best axis to project on. {i.e. minimum sum of squares of projection error} It helps in minimum reconstruction error.



SVD minimizes the sum of the squares of projec^n i.e the (difference b/w actual posi^n and its projec^n on the line)^2.

v_1 in the above figure is nothing but the first row of our V^T matrix calculated before for the user-movie problem. This v_1 is the axis of highest variation and the corresponding singular value in Σ tells us the <u>variance along that dimension or spread</u> of the values along that given dimension. The 3rd concept, other than the sci-fi and romance does not really have much data spread about it and so has lesser variance or strength in Σ matrix.

$$(U\Sigma)^T = \begin{bmatrix} 1.61 & 0.19 & -0.01 \\ 5.08 & 0.66 & -0.03 \\ 6.82 & 0.85 & -0.05 \\ 8.43 & 1.04 & -0.06 \\ 1.86 & -5.60 & -0.84 \\ 0.86 & -6.93 & -0.87 \\ 0.86 & -2.75 & 0.41 \end{bmatrix}$$

variation along the 3rd axis is much smaller.

projec of users on sci-fi axis i.e loc of every user along first right singular vector

projec on romance axis.

Question still remains as to how do we do the dimensionality reduction.

Ans In our example, to do dimensionality reduction from a 3D space to 2D space, we take the small singular value and set it to zero thus setting to zero the last column of U, the last row of Σ and V^T as the variance along them was very small.

Thus new matrices are:

$$U \approx \begin{bmatrix} 0.13 & 0.02 \\ 0.41 & 0.07 \\ 0.55 & 0.09 \\ 0.68 & 0.11 \\ 0.15 & -0.59 \\ 0.07 & -0.73 \\ 0.07 & -0.29 \end{bmatrix} \qquad \Sigma = \begin{bmatrix} 12.4 & 0 \\ 0 & 9.5 \end{bmatrix}$$

$$V^T = \begin{bmatrix} 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\ -0.12 & -0.02 & 0.12 & -0.69 & -0.69 \end{bmatrix}$$

$$U\Sigma V^T = \begin{bmatrix} \\ \\ \\ \\ \\ \\ \\ \end{bmatrix}_{(7\times 5)} = B \approx \text{Original matrix which was given to us initially.}$$

So, we reduced our original matrix to lower dimensions without much reconstruction error.

Frobenius norm: frobenius norm of two matrices is the sum of the differences of their entries

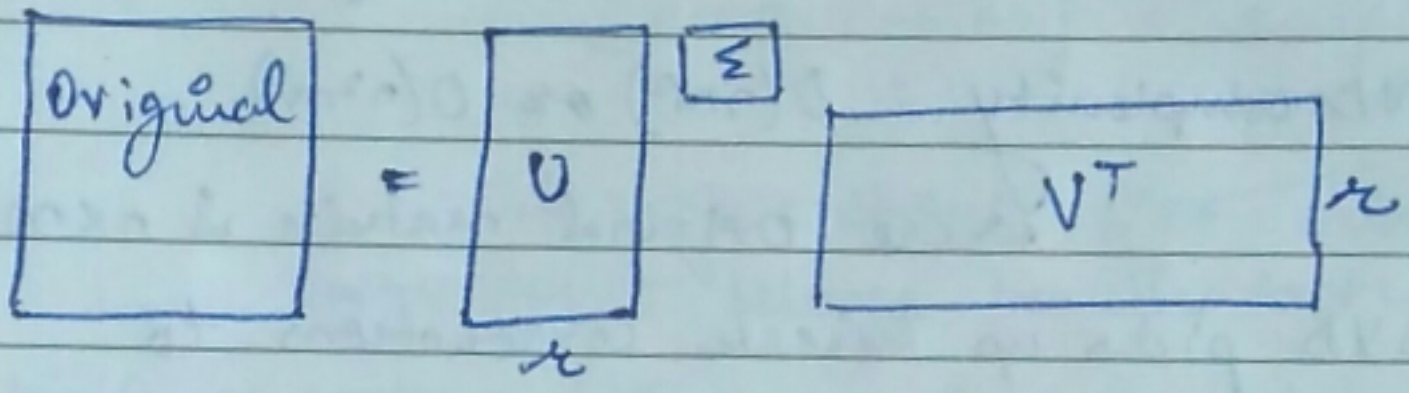
$$|M|_F = \sqrt{\Sigma_{ij} M_{ij}^2}$$

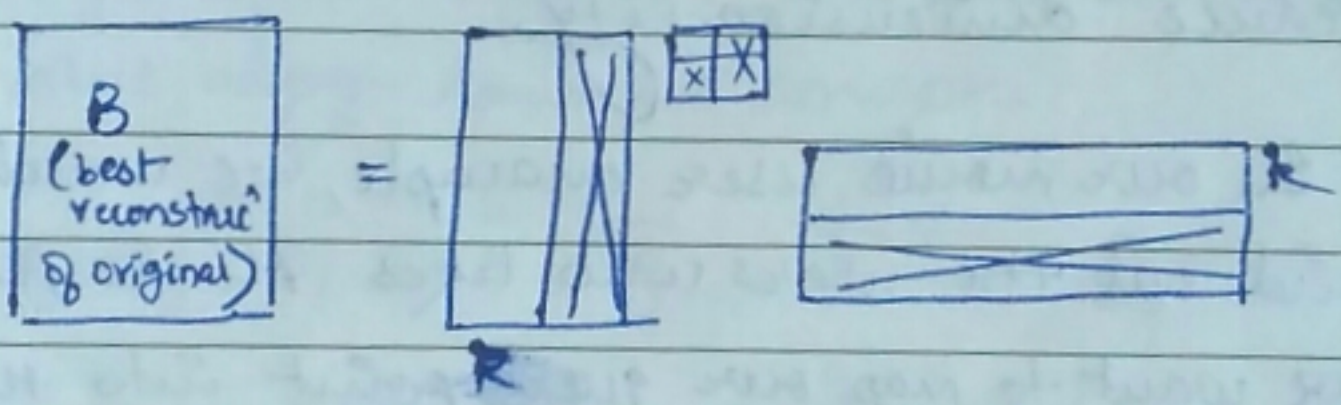
$$|A-B|_F = \sqrt{\Sigma_{ij} (A_{ij} - B_{ij})^2}$$

Frobenius norm.

This comes out to be very small for the matrix B and our original matrix. This is the speciality of SVD.

The Best 'B' is when the frobenius norm value is the least possible.





So, original had r but reduced matrices have just k concepts. Original ≈ B

B is a solution to $\min |Original - B|_F$ where $Rank(B) = k$.

no. of dimensions we allowed ourselves to work with.

$$B = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \ldots\ldots \sigma_k u_k v_k^T$$

← k terms →

$\sigma_1 \geq \sigma_2 \geq \ldots \geq 0$

Q. Why is setting small σ_i to 0 the right thing to do?

Ans Bcoz U and V are column orthonormal, so σ is the one affecting their value in multiplication. So, if σ is small, it can be dropped to zero.

Q. How many k (i.e singular values σ) should S keep?

Ans $$\frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{r} \sigma_i^2} \approx 0.9$$ i.e we should keep 80-90% of 'energy' atleast while dimensionality reduction.

SVD complexity : $O(nm^2)$ or $O(n^2m)$

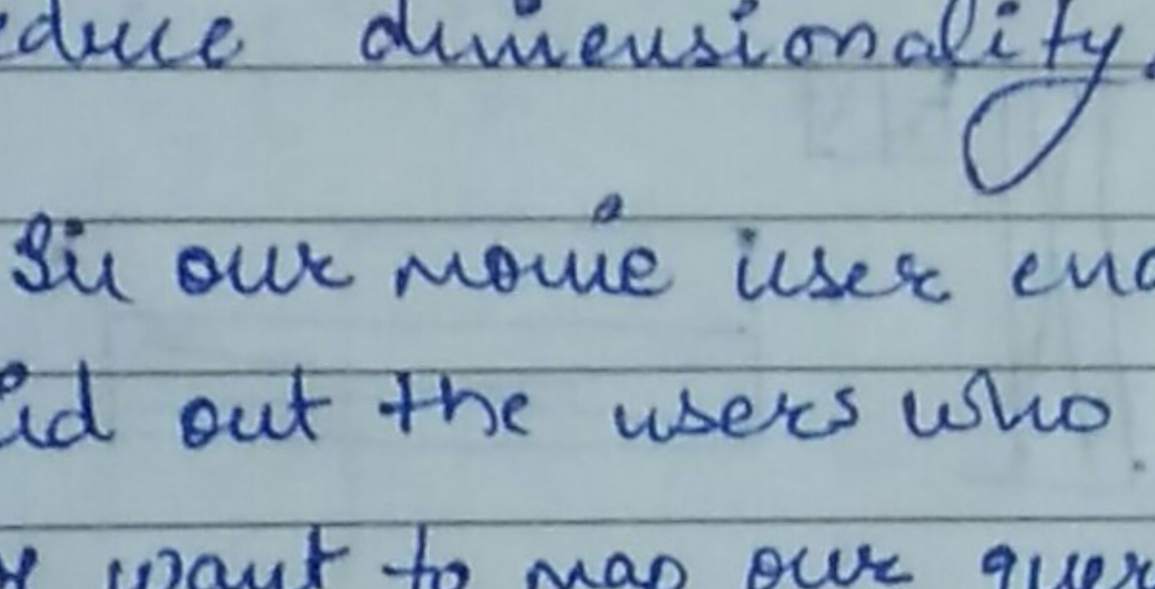
where original matrix is $n \times m$.

SVD picks up linear correlations to reduce dimensionality.

Eg. In our movie user example, we want to find out the users who likes Movie A.

Ans we want to map our query point into the concept space

query point (q) = [5 0 0 0 0]



we want to find users having similar q and lying near the query point in the concept space.

the multiplication of q with $(V^T)^T$ (movie to concept matrix) gives the loca? of q. q times the first singular vector will tell the loca? of q along v_1 and the second singular vector will tell loca? of q along v_2 by giving its projec? on the axis.

$$q_{concept} = qV$$

$$q = [5\ 0\ 0\ 0\ 0] \times \begin{bmatrix} 0.56 & 0.12 \\ 0.59 & -0.02 \\ 0.56 & 0.12 \\ 0.09 & -0.69 \\ 0.09 & -0.69 \end{bmatrix} = [2.8\ \ 0.6]$$

sci-fi concept

this vector shows that our query point belongs heavily to the sci-fi concept and has very low coordinate value along Romance concept.

$$q = \begin{matrix} A & B & C & D & E \\ [0 & 4 & 5 & 0 & 0] \end{matrix} \xrightarrow{q \times V} [2.8 \quad 0.6]$$

$$d = [5\ 0\ 0\ 0\ 0] \xrightarrow{q \times V} [5.2 \quad 0.4]$$

(zero ratings in common)

(the q and d points are near in the concept space as they both heavily belong to the sci-fi concept)

Observa?: User d is similar to q though they have zero ratings in common. This was possible booz of SVD as it found out the common grounds in q and d.
In collaborative filtering d and q are not neighbours.