By MONDAY $\longrightarrow$ Dashboard will be updated

10468

MATRIX FACTORIZATION

$$6 = 2 \times 3 \quad (\text{factors of } 6)$$

$$\begin{bmatrix} A \end{bmatrix}_{n \times m} = \begin{bmatrix} B \end{bmatrix}_{n \times d} \times \begin{bmatrix} C \end{bmatrix}_{d \times m}$$

$A_{n \times m}$ where, $n \simeq 10^9$ users

$m \simeq 10^8$ items

$$\text{Total cells in } A = 10^9 \times 10^8 = 10^{17}$$

$\downarrow$

A is a Sparse Matrix.

NOTE: Goal is to complete the matrix (fill empty values)

2009 $\rightarrow$ Netflix Prize Competetion $\rightarrow$ to improve R.S.

Winning Team: 10 % improvement

↳ Matrix Factorization

→ one of the most important algorithm for R.S.

→ frequently used in modern R.S.

Items (Movies) → I_2 → Avengers, I_4 → Titanic

$$
A_{n \times m} = \begin{array}{c|cccc}
\text{Users} & I_1 & I_2 & I_3 & I_4 \\
\hline
U_1 & 5 & ? & 3 & ? \\
U_2 & ? & 4 & ? & 2 \\
U_3 & 1 & ? & 2 & ?
\end{array}
$$

→ Most values are missing (?) because every user rarely gives a rating.

Goal: Predict the missing ratings
↳ for better recommendations

	Action	Romance
U1 → User	0.8	0.2

	Action	Romanc
I_2 → Avengers	0.9	0.1

	Action	Romance
I_4 → Titanic	0.1	0.9

• Intuition behind M.F :

→ Users and Movies can be represented in a small number of hidden (latent) dimensions

Example :

→ Action v/s Romance preference.

→ Serious v/s Light-hearted preference.

→ Old v/s New Movies

→ We don't define these dimensions, algorithm learns them.

• NETFLIX :

$$A_{n\times m} = B_{n\times d} \times C_{d\times m}$$

where,

d → hidden dimensions (latent features)

Suppose, we have 2 latent factors

(i) Action-level

(ii) Romance-level

B →

	Action	Romance
User 1	5	1
User 2	1	4

C →	Action	Romance
Item1 (Avengers)	0.9	0.1
Item 2 (Titanic)	0.1	0.9
Item3 (Inception)	0.8	0.2

Predicted Rating →

$$= [5 \quad 1] \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix}$$

$$= 5(0.9) + 1(0.1)$$

Max. ratings is 5

$$= 4.6 \checkmark$$

↳ very high

↓

User 1 will like Avengers and give 4.6 rating.

M.F re-writes:

Original Big Matrix → A

$$A_{n \times m} = B_{n \times d} \times C_{d \times m}$$

where, d → latent factors (hidden preferences)

$d \sim [10, 100]$

$n \to 1{,}00{,}000$ users

$m \to 10{,}000$ movies

cells in A $\to 10^5 \times 10^4 \simeq 10^9$ cells

$d = 50$

$$\left.\begin{array}{l} B = 10^5 \times 50 \\ C = 10^4 \times 50 \end{array}\right\} \text{smaller matrices}$$

$\downarrow$

captures meaningful structure

• Benefits of M.F :

1. Captures hidden patterns / preferences
2. Handles sparse data
3. Scales to millions of users / items
4. Better personalization / recommendation

• How do we learn / make matrix B and C ?

→ Given very few observed ratings (non-empty cells), our goal is to find 'B' and 'C' such that predicted ratings are close to actual ratings.

Optimization Problem:

Loss Function: actual rating - predicted rating

actual rating $= r$

user latent vector $= b_i$

item latent vector $= c_j$

$$
\begin{array}{cc}
 & \begin{array}{cccc} \text{Item1} & & & \end{array} \\
\begin{array}{c} \\ \text{User2} \\ \\ \end{array} &
\overset{A}{\begin{bmatrix} ? & 1 & ? & 5 \\ 4 & ? & ? & 1 \\ 1 & 2 & ? & ? \end{bmatrix}}_{n \times m}
\end{array}
= \underset{n \times d}{[\,B\,]} \times \underset{d \times m}{[\,C\,]}
$$

$$= B_2 \times C_1$$

$$4 >> 2$$

$$4 \simeq 3.95$$

$$\boxed{L.F = \sum (r - b_i . c_j)^2 + \lambda \left(\|b_i\|^2 + \|c_j\|^2 \right)}$$

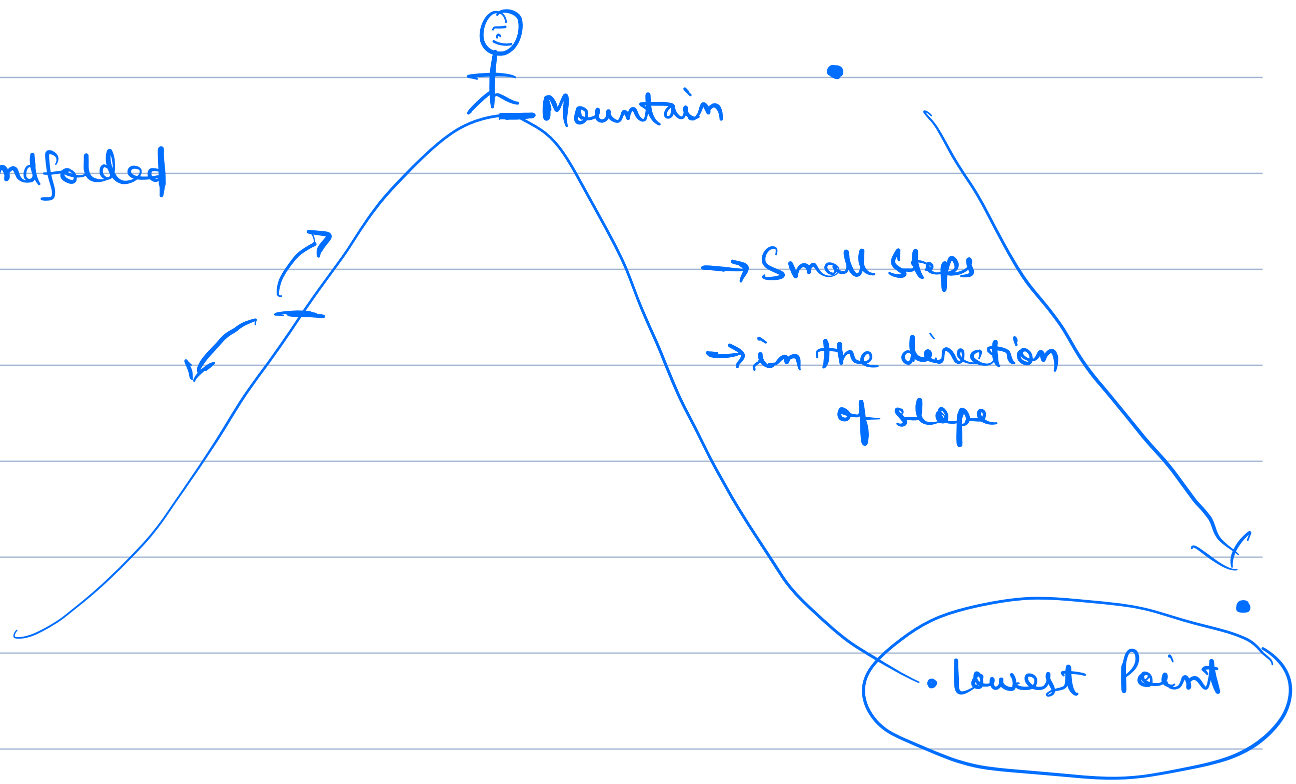
$\lambda \Rightarrow$ regularization term (prevents overfitting)

- 2 common Optimization techniques:

  (1) SGD → Stochastic Gradient Descent

  (2) ALS → Alternating Least Squares

→ blindfolded



[B] [C]

L.F → error

↓

Modify B and C

Repeat till

→ error convergence

→ small improvement

10 ⟷ 2 → 4 → 6 → 9 ↓

Steps of SGD:

Stochastic = Random

SGD updates user vector (B) and item vector (C) directly by the following the gradient of loss function

1. Initialise → randomly assigning values of B and C.
2. Loop over each known rating:
   - (i) Store actual rating
   - (ii) Predict rating using "B × C"
   - (iii) Compute the error
3. Update user vector (B_i) and item vector (C_j)
4. Repeat above steps until: (epochs)
   - ↳ error convergence
   - ↳ improvement becomes small

Mathematically,

1. Initialize: user vector → B_i (Assign small values)
   item vector → C_j

2. For each non-empty rating A_{ij} →
   - (i) Predict Rating → $B_i \times C_j$

$$\boxed{\text{(ii) Compute Error} \quad \rightarrow e_{ij} = (A_{ij} - B_i \times C_j)}$$

③ Update user vector $= B_i \leftarrow B_i + \eta\,(e_{ij} \cdot C_j - \lambda B_i)$

Update item vector $= C_j \leftarrow C_j + \eta\,[e_{ij}\, B_i - \lambda C_j]$

η = learning rate

λ = regularization term (prevents overfitting)

- Repeat above until convergence

NOTE: Happens incrementally, rating by rating

② ALS → Alternating Least Squares

Fix items (C) → Adjust users (B)

Fix users (B) → Adjust items (C)

ALS says:

"Let not adjust both together - it's messy"

$$\boxed{\text{Loss Function} = \min. \sum (A_{ij} - B_i \cdot C_j)^2}$$

(Least Squares Problem)

SGD → B × C → getting updated together

10

2 × 3 →

3 × 4 →

ALS →

we fix B → adjust C →

we fix C → adjust B

	SGD	ALS
Update Style	one rating at a time	all users first then items
Math Type	gradient descent	least squares
Speed	slow for huge datasets	faster
Stability	Sensitive to learning rate	very stable
Used in	Small / medium datasets	Big data

H.W :

→ PCA → Eigen value Decomposition

↳ SVD → Singular Vector Decomposition

Code Implementation of M.F :