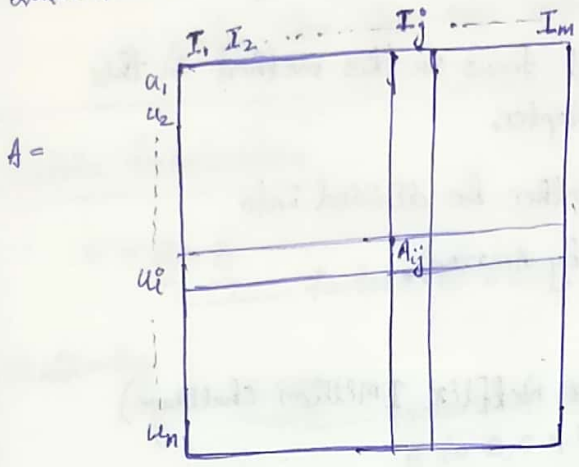
Recommendation System

Formulation

Dataset = A



n: users
m: items.

A_{ij} :- rating given by u_i on I_j (1-5)
or
u_i watched I_j (1 or 0).

* very sparse matrix as avg user would have rated around 5-10 items.

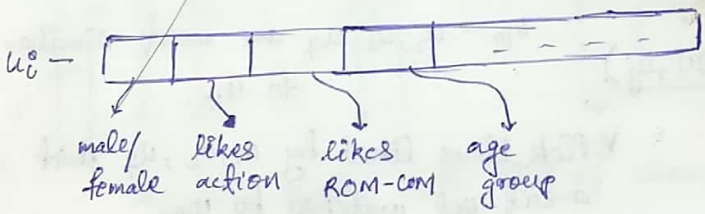
Task of RS:- u_i :- I_1, I_2, I_7, I_8 —— items rated by user u_i.
↳ recommendation of new items based on these ratings ??

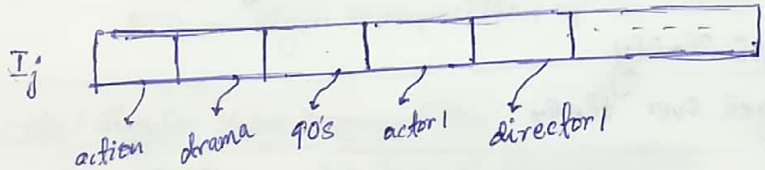
RS algorithms can be broadly classified into 2 types.

1. Content based RS
2. Collaborative filtering based RS.

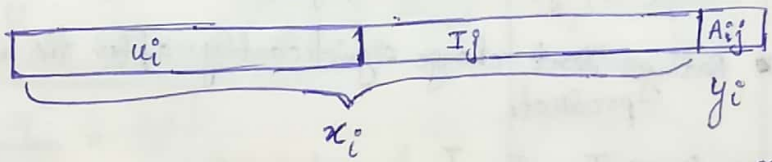
1. Content based RS

This requires feature vectors for user & item.





Now, we pose the training pt as



We won't discuss about this in this chapter.

③ collaborative filtering based RS

u_1 :- M_1, M_2, M_3

u_2 :- M_1, M_3, M_4

u_3 :- M_1, M_3 ? → M_3

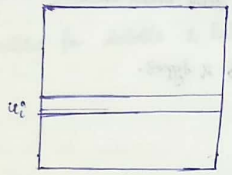
core-idea(assumption) :- users who agreed in the past tend to also agree in future.

We'll focus on this method in this chapter.

collaborative filtering based RS can further be divided into

1. Item-Item based RS. (Popularized by Amazon).
2. User-user based RS.
3. Matrix factorization. (Popular after Netflix 1 Million Challenge) (Matrix completion).

user-user based RS

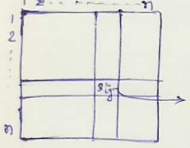


u_i - can be thought of as user-vector.
↳ very sparse like BoW.

So, we can find similarity b/w 2 users as

$$sim(u_i, u_j) = cosine(u_i, u_j) = \frac{u_i^T \cdot u_j}{\|u_i\| \|u_j\|}$$

S^u_{ij} = user similarity matrix.



Task:- recommend new items for u_{10}.

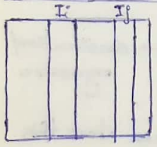
$u_{10} \rightarrow u_1, u_2, u_7$ are most similar to u_{10}.

* Pick items liked by u_1, u_2, u_7 that aren't yet watched by u_{10}.

Problem with user-user similarity

user preferance changes over time.

Item-Item based similarity



$sim(I_i, I_j) = cosine(I_i, I_j)$

* Ratings don't change significantly after the initial product.

$u_{10} \rightarrow I_1, I_3, I_{10}$

Find items similar to ones liked by user & recommend. { } ∪ { } ∪ { }

Rule of Thumb →

more users than items ⟶ Amazon, Netflix, Youtube, etc.

⇓

Item ratings don't change much over time after initial period.

⇓

prefer item-item over user-user.

Matrix factorization

$6 = 2 \times 3$ — factors of 6.

similarly, $A = BCD$ — factors of A. and the process of breaking down A to B, C & D is factorization/decomposition.

PCA as matrix factorization (Eigen Decomposition)

$X_{n \times d}$ ⟶ data matrix (standardized).

$S_{d \times d} = \frac{1}{n}(X^T X)$ = co-variance matrix.

$$S_{d \times d} = W_{d \times d} \Lambda_{d \times d} W^T_{d \times d}$$ — (eigen decomposition of S)

{special type of factn/decompn}

$W = [w_1 \; w_2 \; w_3 \; \cdots \; w_d]$ — eigen vectors

$\Lambda = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \lambda_3 & \\ & & & \lambda_d \end{bmatrix}$ — eigen-values.

So, from algebra math pov of view,

PCA — eigen decomposition of co-variance matrix.

SVD (Singular Value Decomposition)

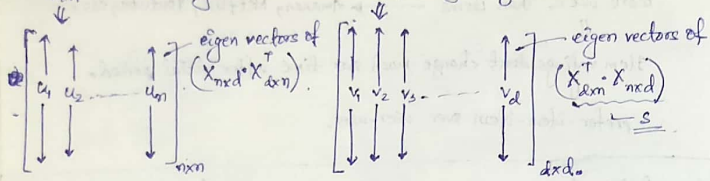
Another special type of decomposition.

$$X_{n \times d} = U_{n \times n} \sum_{n \times d} V^T_{d \times d}$$

$\sum_{n \times d} = \begin{bmatrix} s_1 & & & \\ & s_2 & & \\ & & s_3 & \\ & & & s_d \\ & & & \end{bmatrix}$ — diagonal matrix. — singular values. — n × d.

singular values of $X_{n \times d}$ ⟶ $\frac{s_i^2}{(n-1)} = \lambda_i$ ⟶ eigen-values of $S_{d \times d}$

$U_{n\times n}$ — left singular vectors. $V_{d\times d}$ → right singular vectors.



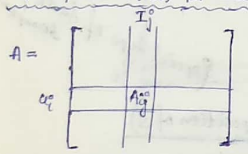
Non-negative Matrix factorization (NMF).

$A_{n\times m} = B_{n\times d} \cdot C^T_{d\times m}$ $\quad B: n\times d$, $C: m\times d$.

s.t. $\begin{cases} B_{ij} \geq 0 \\ C_{ij} \geq 0 \end{cases}$ — constraint

We'll know later why it may be useful.

Matrix factorization for collaborative filtering



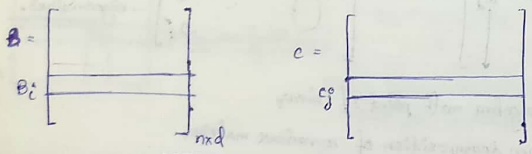
A_{ij} :- rating on I_j by u_i

sparse matrix :- many empty cells.

Say somehow we decompose A as:

$A = B * C^T$ (n×m = n×d, d×m)

$0 < d \leq \min(m, n)$.



$A_{ij} = B_i^T C_j$ — Assuming B_i is i^{th} row of B & a column vector, & C_j is j^{th} row of C & a column vector.

optimization problem —

$\underset{B,C}{\operatorname{argmin}} \sum_{i,j \text{ s.t. } A_{ij} \text{ not empty}} (A_{ij} - B_i^T C_j)^2$

solve using SGD

Solving this will give B & C as mentioned above.

$\hat{A} = B \cdot C^T$ will give the predicted ratings.

Matrix factorization as feature engineering

↳ Just in the above section,

$A = B \cdot C^T$ $\quad B = n\times d$, $C = m\times d$ } these matrices can be thought of as feature vectors of users & items

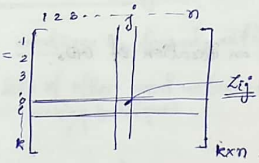
Similarly, later we'll see word-vectors & eigen-faces - using MF.

clustering as MF

In K-Means, the original optimization function was

$$\min_{c_i} \sum_{i=1}^{k} \sum_{x \in s_i} \|x - c_i\|^2 \qquad x \in R^d$$

We define z s.t. $z_{ij} = \begin{cases} 1, & \text{if } x_j \in s_i \\ 0, & \text{otherwise.} \end{cases}$

So, $z =$ 

The above optimization function can be re-written as

$$\min_{c_i, z_{ij}} \sum_{i=1}^{k} \sum_{j=1}^{n} z_{ij} \|x_j - c_i\|^2$$

If we have $X = [x_1\ x_2 \ldots x_n]_{d\times n}$ & $C = [c_1\ c_2 \ldots c_k]_{d\times k}$

we can re-write the optimization function with vectorization as

$$\min_{c, z} \|X - Cz\|^2 \quad \text{s.t. } z_{ij} = 0 \text{ or } 1 \quad \text{— NP-hard}$$

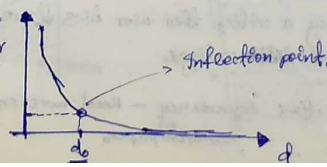
& column-sum$(z) = 1$.

So, clustering = MF + column-constraint + 0-1 constraint.

Hyper-parameter tuning

① problem specific. — depends on context & requirement.

② systematic way.



$d = d_0$ ←

Netflix Prize (2009).

Loss-metric – RMSE $\sqrt{\frac{1}{|n|}\sum_{i,j}(r_{ij}-\hat{r}_{ij})^2}$

Research paper → "datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf."

r_{ui} – rating by u to i.

q_i – item vector.

p_u – user vector.

Initial formulation

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

sq. loss. L2-regularization.

Solves:- ① SGD – find $\frac{\partial L}{\partial p_u}$ & $\frac{\partial L}{\partial q_i}$ & perform an iteration of GD.

It solves but takes long time.

② Alternate Least Squares (ALS).

faster Algo. ⓐ fix p_u, gradient descent to fix q_i's

ⓑ fix q_i, gradient descent to fix p_u's.

Modified formulation –

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

μ – avg rating across all users & items. (say 3.0).

It's better to separate the base avg fm interaction of user & item.

b_i – item bias – say a Nolan movie. So, it's supposed have a better rating. So, item bias ↑.

b_u – user bias – say a critic, then user bias ↓. as he gives less ratings.

$b_i(t)$; $b_u(t)$ & $r_{ui}(t)$ – time dependency – Read more on research paper.

cold-start problem

↳ New user (u) So, we have no rating data of them.

or new product (i).

New-user → Recommend top items based on meta-data.

- geo-location (ip-address) – california
- browser → safari
- device → ipad

} content based recommendation.

new item → meta-data

i' → category, price, description. ....

recommend as new arrival (i_1, i_2, i_3 ... – similar items.

word-vectors using MF (Truncated SVD).

word2vec → formed with help of neural networks.

Lots of other word vectors → LSA, PLSA, LDA, GLOVE (2014). ↳ related to word2vec.
making techniques.

① co-occurence matrix (X).

X_{ij} = #times w_i occurs in context of w_j.

$w_1\ w_2\ w_i\ w_3\ w_4\ w_j\ w_5\ w_6$

Say two words are said to be in context of each other iff they occur within neighbourhood of distance 5.

n = #words

$X_{n\times n} = U_{n\times n} \Sigma_{n\times n} V^T_{n\times n}$.

$\Sigma = \mathrm{diag}(s_1, s_2, \ldots, s_n)$, $s_i^2 = \lambda_i$, $s_1 \geq s_2 \geq s_3 \geq \ldots \geq s_n$

$U_{n\times k}$ = $[u_1\ u_2 \ldots u_k]$ $n\times k$

$V^T_{k\times n}$ = $[v_1; v_2; \ldots; v_k]$ $k\times n$

PCA – top k-eigen-values & corresponding vectors. — analogous to what we did

$$X \approx \hat{X}_{n \times n} = \hat{U}_{n \times k} \hat{\Sigma}_{k \times k} \hat{V}^T_{k \times n}$$

$\hat{u}_i$ = i^{th} row of $\hat{u}$ of k-dimensions can be thought of as the word vector of k-dimensions.

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

problem with concurrence matrix (X)

$\Downarrow$

n = # words very large.

soln:- use tf-idf to get top words & build X from them.

eigen-faces is nothing but PCA on image data.

The eigen-vectors are basically called the eigen-faces.