HUST

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Movie Recommendation System
Group 3

Group members

1. **CHU ANH DUC - 20230081**
2. **VU THUONG TIN - 20230091**
3. **TRAN QUANG HUNG - 20235502**
4. **PHAN DINH TRUNG - 20230093**

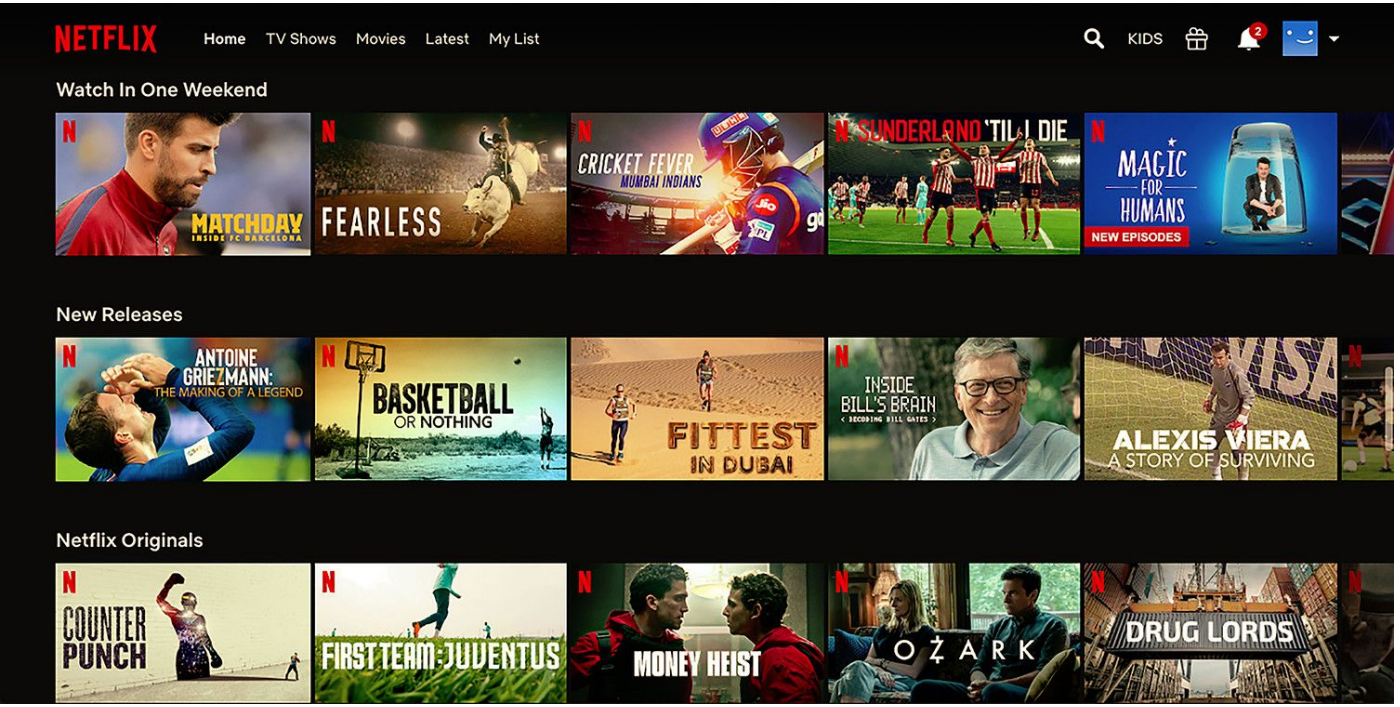
ONE LOVE. ONE FUTURE.

HUST

hust.edu.vn fb.com/dhbkhn

INTRODUCTION

PROBLEM STATEMENT



- **USING SEVERAL TECHNIQUES AND ALGORITHMS TO BUILD SOME TYPES OF BASIC RECOMMENDATION SYSTEM**

- **PRIMARY OBJECTIVE:**
  Ridge the gap between user preferences and available content, fostering engagement, satisfaction, and retention

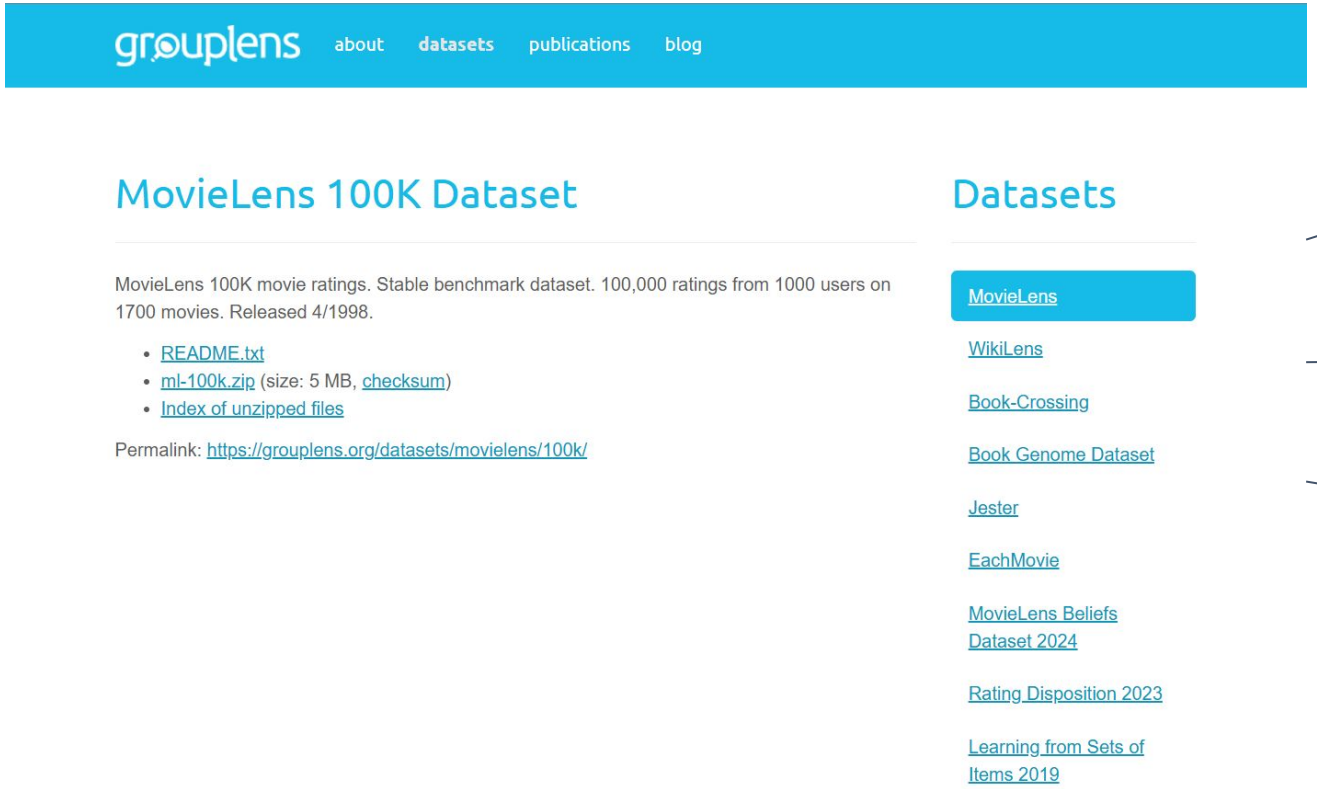
HUST

hust.edu.vn fb.com/dhbkhn

LITERATURE REVIEW

1. SCOPE OF DATASET

- Training dataset using for our project: ***MovieLens 100k datasets***



u.data

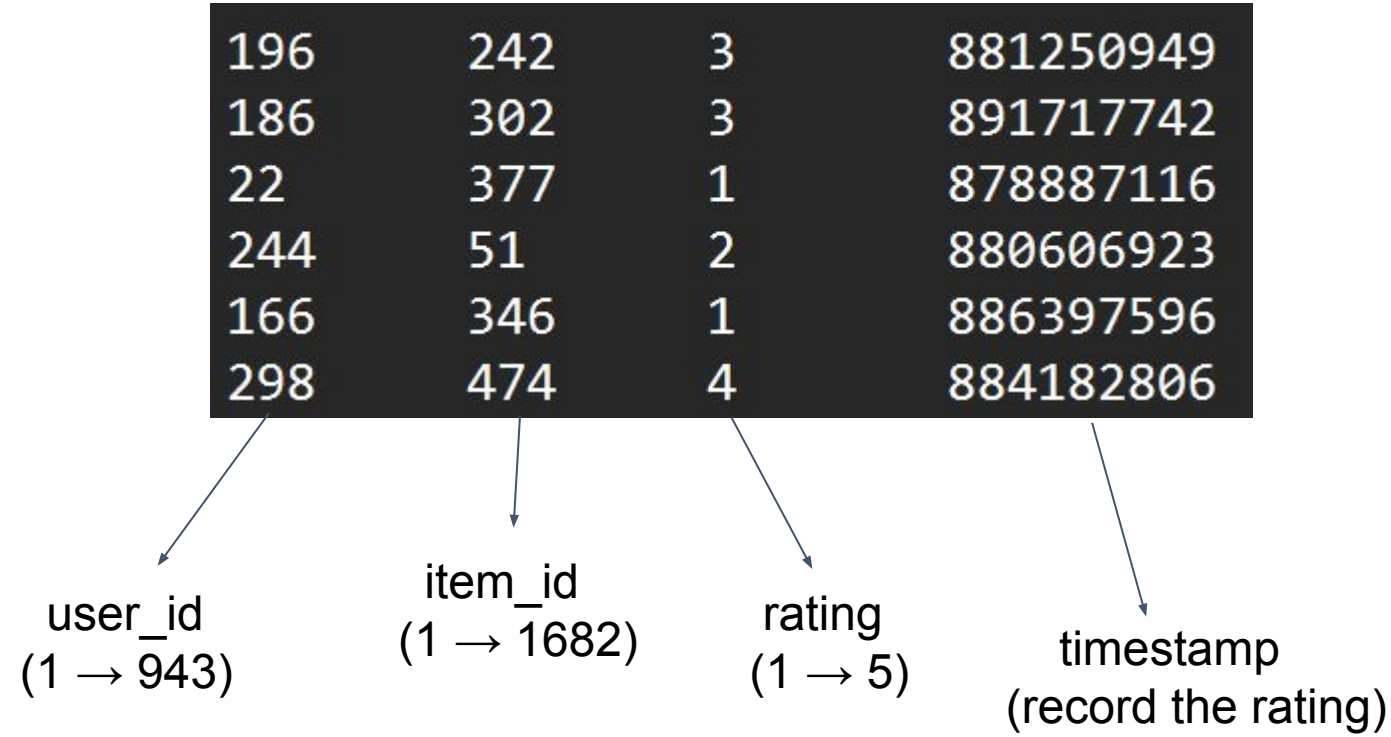
u.item

u.user

SUITABLE FOR TRAINING RECOMMENDATION SYSTEMS !!

1. SCOPE OF DATASET

- **u.data**



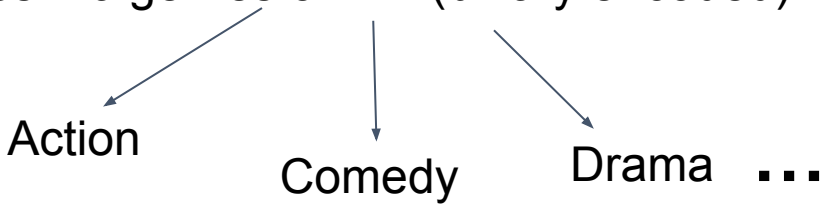
SCOPE OF DATASET

- **u.item**

```
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)|0|0|0|1|1|1|0|0|0
|0|0|0|0|0|0|0|0|0|0
2|GoldenEye (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?GoldenEye%20(1995)|0|1|1|0|0|0|0|0|0|0
|0|0|0|0|0|0|1|0|0
3|Four Rooms (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Four%20Rooms%20(1995)|0|0|0|0|0|0|0|0
|0|0|0|0|0|0|0|0|1|0|0
4|Get Shorty (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Get%20Shorty%20(1995)|0|1|0|0|0|1|0|0
|1|0|0|0|0|0|0|0|0|0|0
5|Copycat (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Copycat%20(1995)|0|0|0|0|0|0|1|0|1|0|0|0
|0|0|0|0|1|0|0
```

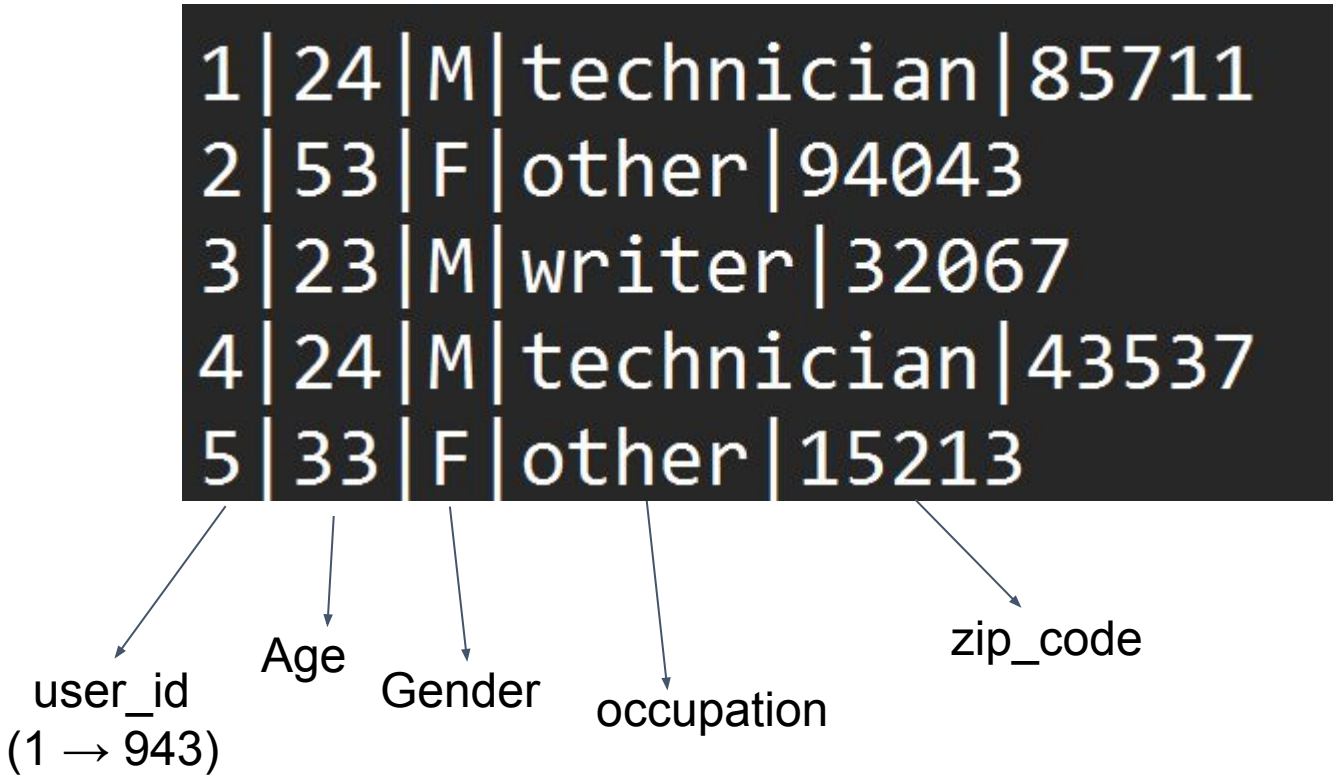
- movie_id: Movie's ID
- movie_title: Title of movie
- release_date: Date of release

- IMDb _url: URL link of the movie
- genres: 19 genres of film (binary encoded)



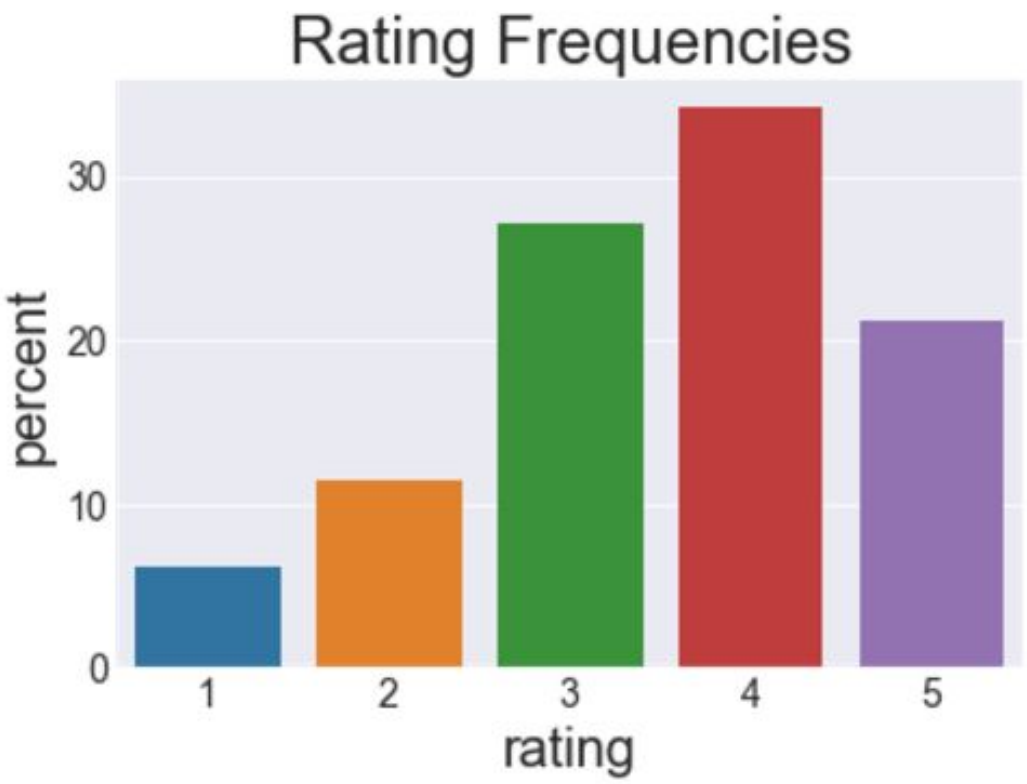
SCOPE OF DATASET

- **u.user**



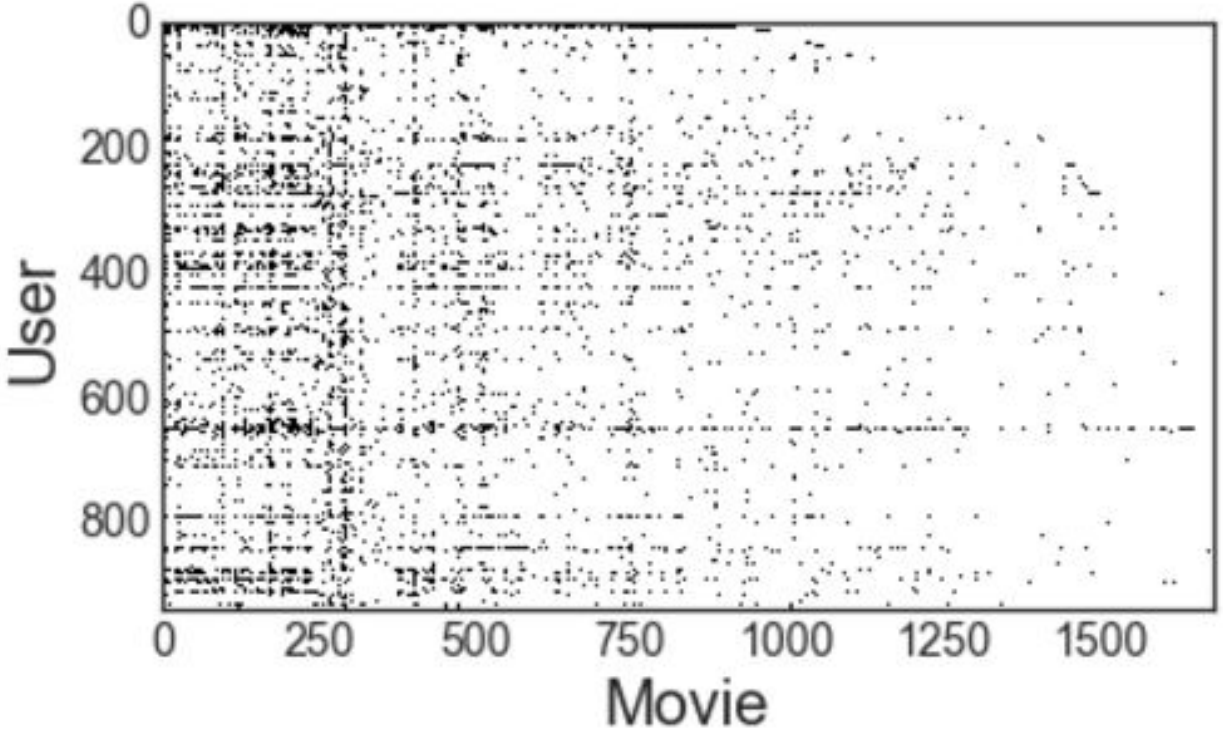
SCOPE OF DATASET

What can we see from the dataset: Rating Frequencies



SCOPE OF DATASET

What can we see from the dataset: Sparsity

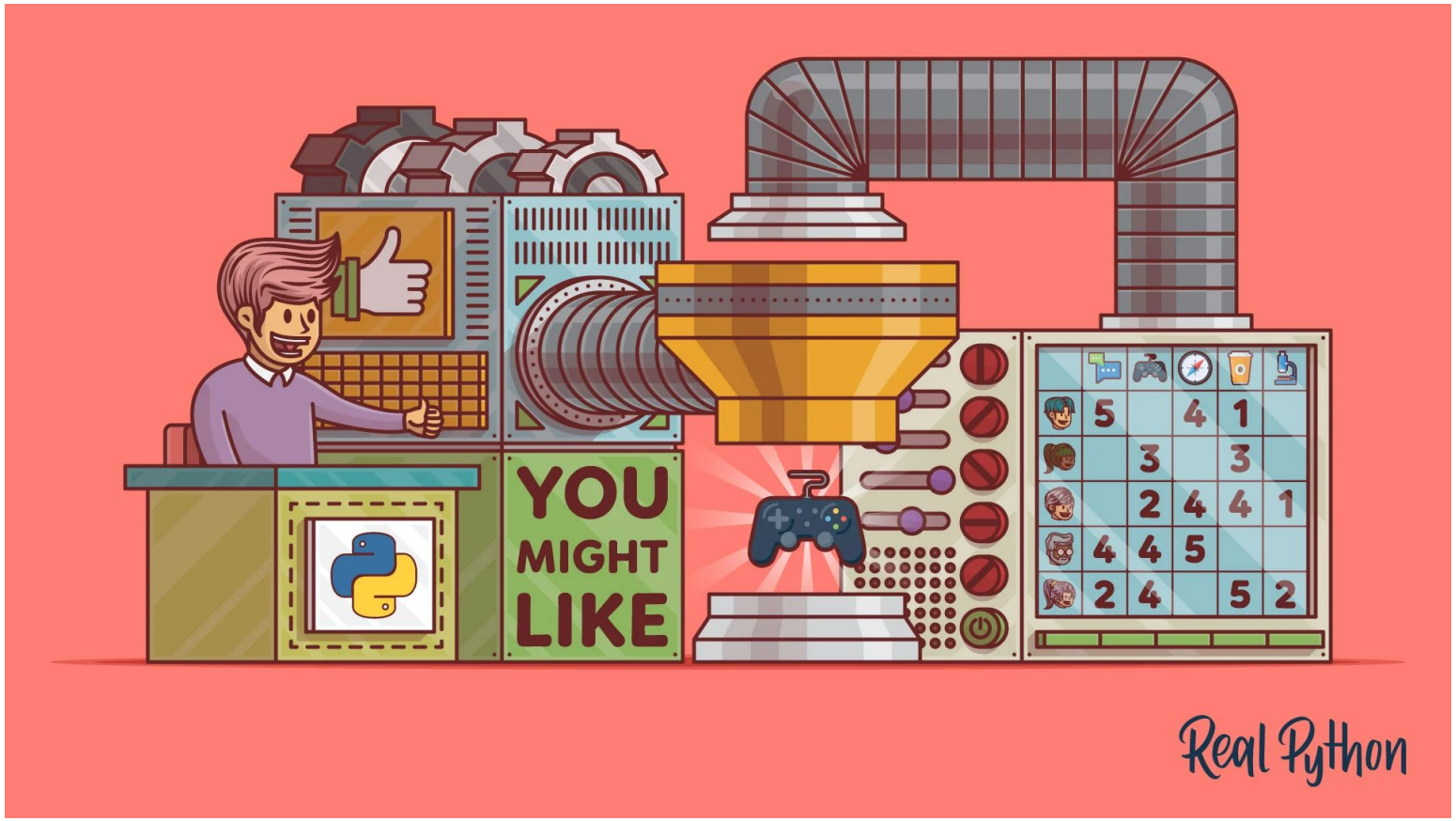


The matrix density is $n_{ratings}/(n_{users} \times n_{movies}) = 0.063$

SCOPE OF DATASET

What can we see from the dataset: Occupation → **Dataset is diverse and objective**

OVERVIEW OF RECOMMENDATION SYSTEM

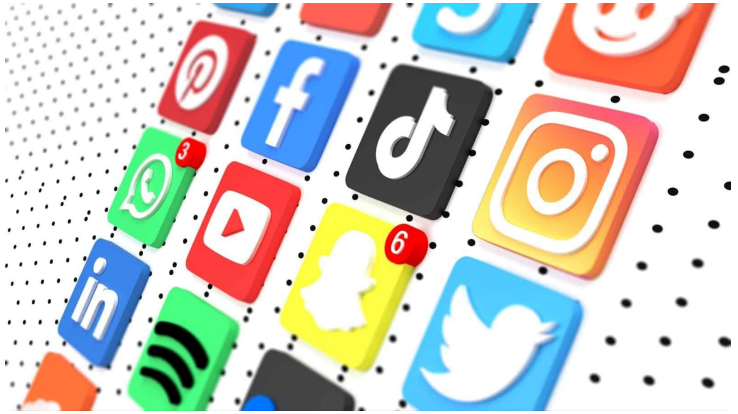


Recommendation systems are tools that provide personalized suggestions based on user preferences and behaviors.

ROLES OF RECOMMENDATION SYSTEM

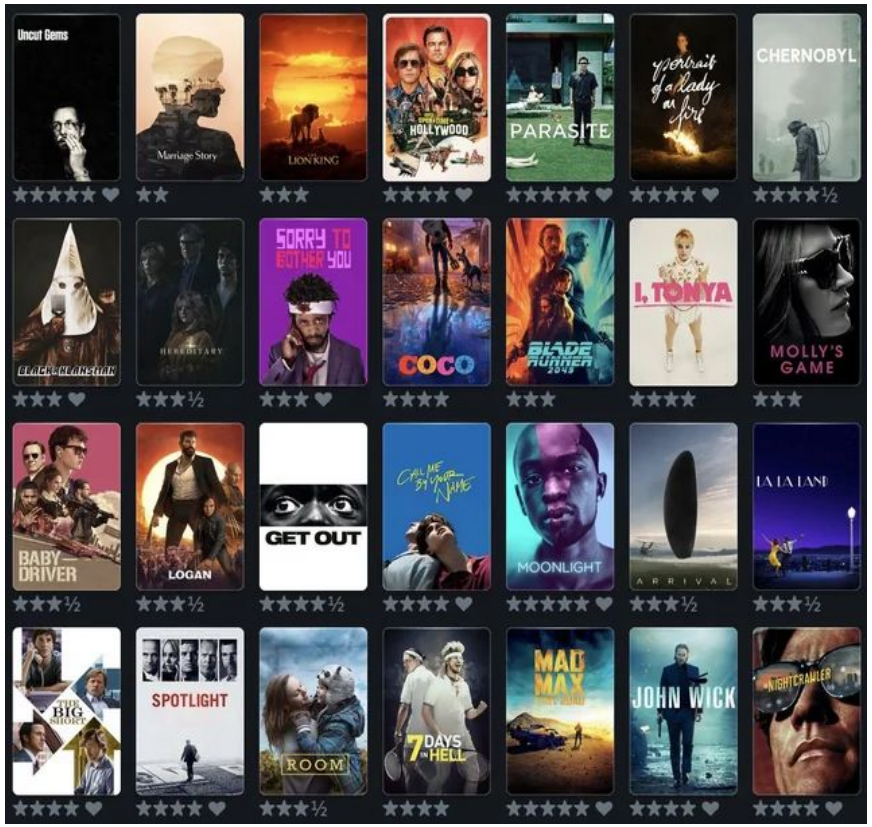
Enhance user experience

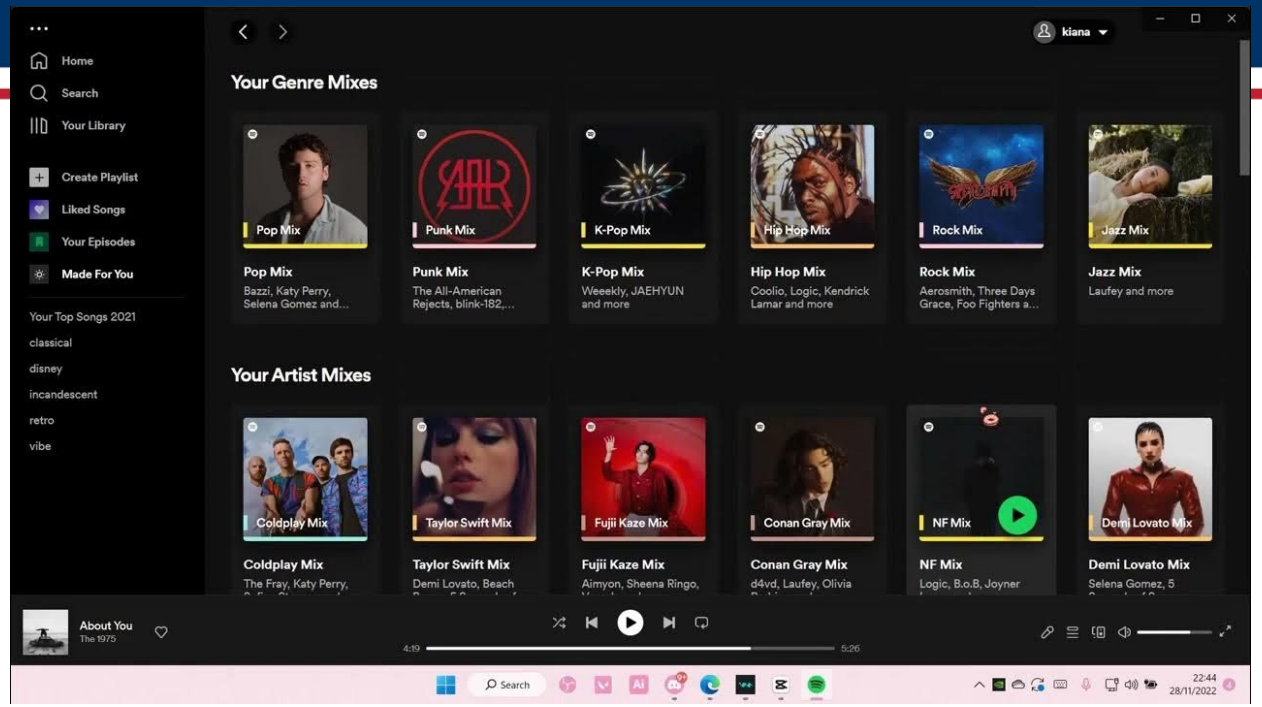
Boost revenue in e-commerce



Optimize content on platforms

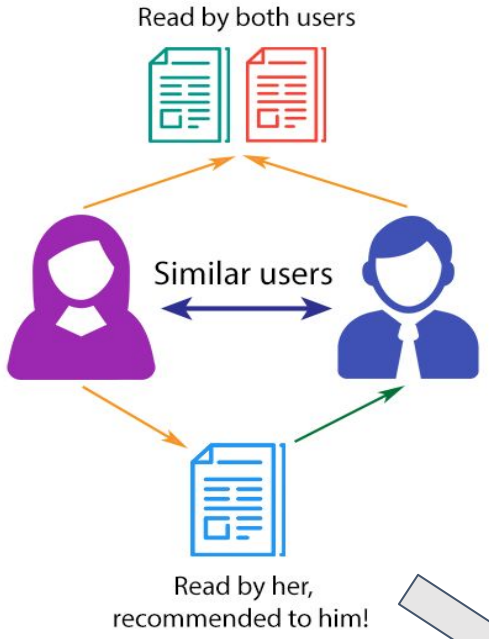
COMMON APPLICATIONS



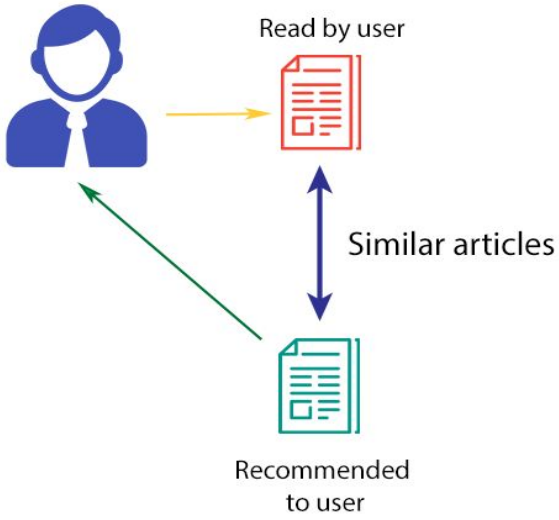


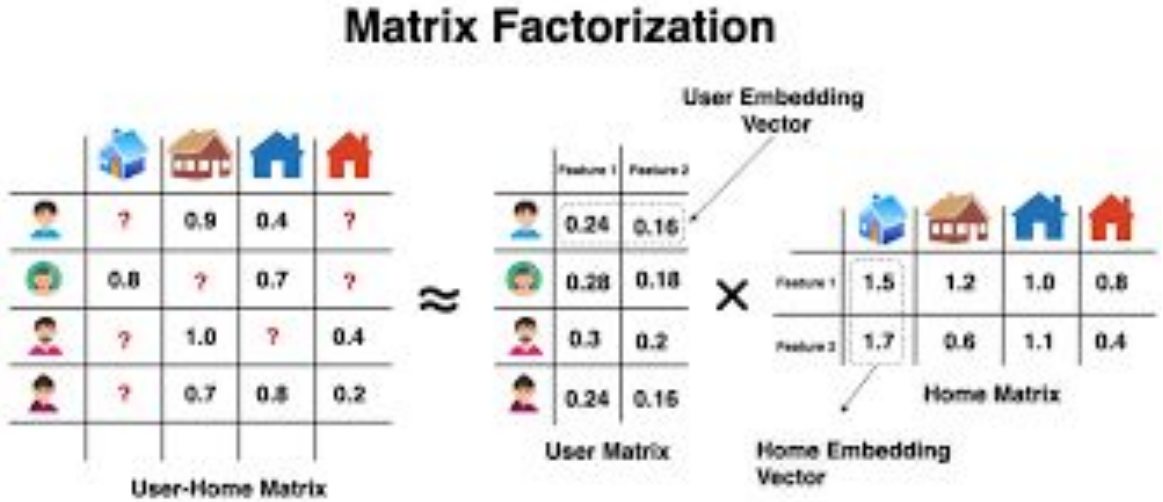
TYPES OF RECOMMENDATION SYSTEMS

COLLABORATIVE FILTERING



CONTENT-BASED FILTERING

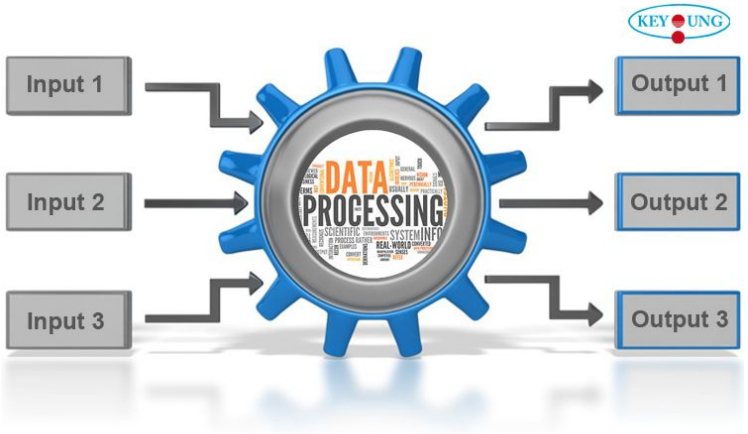




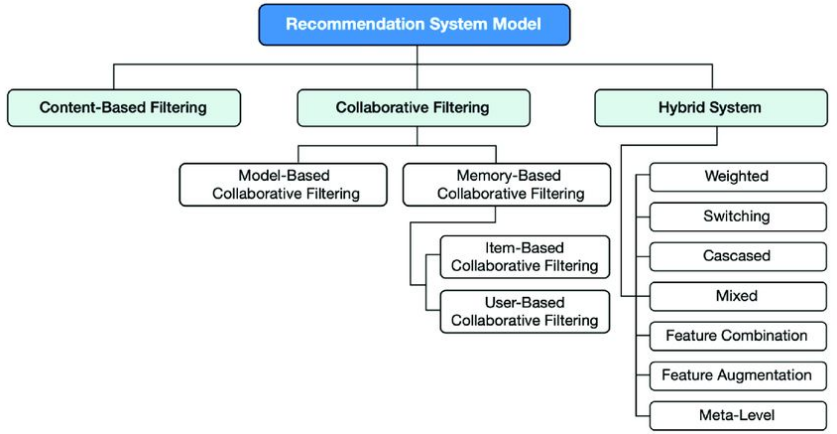
User-user and item-item

WORKFLOW OF RECOMMENDATION SYSTEMS

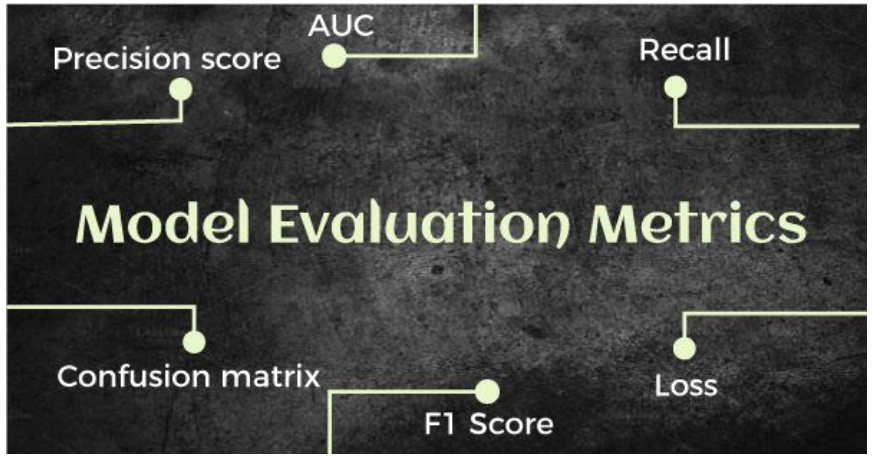
Data collection



Data preprocessing



Build the recommendation model



Evaluate the model

DEPLOY !!

EVALUATING METRICS

ROOT MEAN SQUARE ERROR

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

MEAN ABSOLUTE ERROR

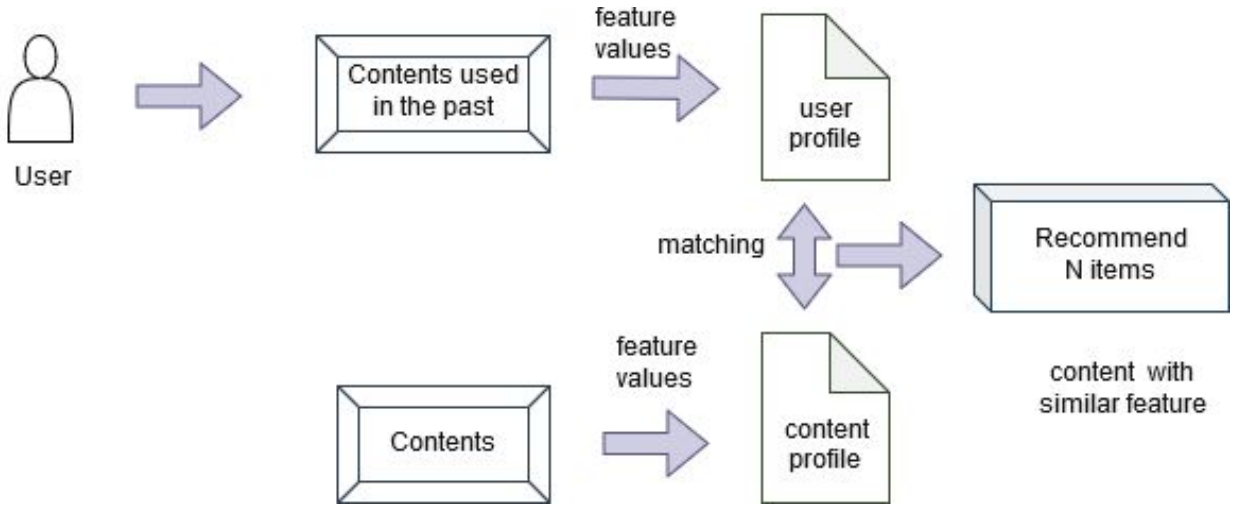
$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

HUST

hust.edu.vn fb.com/dhbkhn

METHODOLOGY

CONTENT-BASED FILTERING



- Content-Based Filtering recommends items based on the attributes (features) of the items and a user's past interactions.
- Assumes that if a user likes an item, they will likely enjoy items with similar characteristics.

CONTENT – BASED FILTERING

1. DATA PREPROCESSING

u.user

```
import pandas as pd
# Reading user file:
u_cols =  ['user_id', 'age', 'sex', 'occupation', 'zip_code']
users = pd.read_csv('ml-100k/u.user', sep='|', names=u_cols, encoding='latin-1')

n_users = users.shape[0]
print('Number of users:', n_users)
users.head()
```

	user_id	age	sex	occupation	zip_code
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213

BASIC INFORMATION OF *u.user*

CONTENT – BASED FILTERING

1. DATA PREPROCESSING

BUILDING ITEM PROFILE:

```
#Reading ratings file:
r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']

ratings_base = pd.read_csv('ml-100k/ua.base', sep='\t', names=r_cols, encoding='latin-1')
ratings_test = pd.read_csv('ml-100k/ua.test', sep='\t', names=r_cols, encoding='latin-1')

rate_train = ratings_base.values
rate_test = ratings_test.values

print('Number of traing rates:', rate_train.shape[0])
print('Number of test rates:', rate_test.shape[0])
rate_train
```

- Read data from the 'ua.base' (training) and 'ua.test' (testing) files.
- Convert the data into NumPy arrays for easier manipulation in subsequent steps.
- Display the number of ratings in both datasets, allowing for quick verification of data integrity and size.

- Store to items the features - genres of movies

```
#Reading items file:
i_cols = ['movie id', 'movie title' ,'release date','video release date', 'IMDb URL', 'unknown', 'Action', 'Adventure',
 'Animation', 'Children\'s', 'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy',
 'Film-Noir', 'Horror', 'Musical', 'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western']

items = pd.read_csv('ml-100k/u.item', sep='|', names=i_cols, encoding='latin-1')

n_items = items.shape[0]
print('Number of items:', n_items)
items.head()
```

```
X0 = items.values
X_train_counts = X0[:, -19:]
X_train_counts
```

Binary code (19 digits) of movie genres

CONTENT – BASED FILTERING

2. FEATURE EXTRACTION (USING *TF-IDF*)

- Purpose: Extract from the dataset the feature matrix of all movies, as $N \times d$ matrix

In our project: $N = 1682$, $d = 19$

movies

Bag of genres of the movies

INITIAL FEATURE MATRIX AFTER PREPROCESSING

	Genre 1	Genre 2	…	Genre d
Movie 1	1	1	…	0
Movie 2	0	1	…	0
…	…	…	…	
Movie N	1	0		0

each row is a feature vector w.r.t that movie

CONTENT – BASED FILTERING

2. FEATURE EXTRACTION (USING *TF-IDF*)

- For a genre ***t*** and a movie's title ***d*** :

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

TF: Terms Frequency
Frequency of genre ***t***'s appearance in movie ***d***

IDF: Inverse Document Frequency

$$\text{IDF}(t) = \log\left(\frac{N+1}{DF(t)+1}\right) + 1$$

N: *number of movies*

DF(t): *number of movies that belongs to* the genre ***t***

Normalized ***IDF***

CONTENT - BASED FILTERING

3. TRAINING BY USING RIDGE REGRESSION MODEL

- Assume that the TF-IDF feature matrix of the movies has rows: ***x1***, ***x2***, ..., ***xN*** (With ***d*** features)
- Purpose: optimize ***w***(weights) and ***b***(bias) for each movie → weight-bias matrix

Prediction of movie rating

$$\hat{y}_i = \mathbf{x_i}^T \mathbf{w} + b$$

- The MSE of the model:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

MINIMIZE:

$$\text{Ridge Loss Function} = \text{MSE} + \lambda \sum_{j=1}^{p} \theta_j^2$$

- And the L2 regularization $\lambda \sum_{j=1}^{p} \theta_j^2$

λ: *regularization parameter*

$$\theta = (X^T X + \lambda I)^{-1} X^T y$$

X: feature matrix

(Calculating the Gradient of Loss function, then set it to be equal o !!)

CONTENT – BASED FILTERING

4. TRAINING BY USING RIDGE REGRESSION MODEL

```
from sklearn.linear_model import Ridge
from sklearn import linear_model

d = tfidf.shape[1] # data dimension
W = np.zeros((d, n_users))
b = np.zeros((1, n_users))

for n in range(n_users):
    ids, scores = get_items_rated_by_user(rate_train, n)
    clf = Ridge(alpha=0.01, fit_intercept  = True)
    Xhat = tfidf[ids, :]

    clf.fit(Xhat, scores)
    W[:, n] = clf.coef_
    b[0, n] = clf.intercept_
```

After getting the weight-bias matrix during Ridge Regression:

$$\hat{y}_{u,i} = \text{TF-IDF}(i) \cdot W_u + b_u$$

Feature vector of movie ***i***

Weight vector for user ***u***

Bias of user ***u***

CONTENT – BASED FILTERING

5. EVALUATING MODEL

```
print('RMSE for training:', evaluate_RMSE(Yhat, rate_train, n_users, W, b))
print('RMSE for test    :', evaluate_RMSE(Yhat, rate_test, n_users, W, b))
print()
print('MAE for train    :', evaluate_MAE(Yhat, rate_train, n_users, W, b))
print('MAE for test    :', evaluate_MAE(Yhat, rate_test, n_users, W, b))
```

```
RMSE for training: 0.9089804562826721
RMSE for test    : 1.2703282700393033

MAE for train    : 0.7077855282989546
MAE for test    : 0.9592280650131063
```

Error of test dataset is a bit higher that of training dataset !

COLLABORATIVE FILTERING

DRAWBACKS OF CONTENT – BASED FILTERING

- Lack of serendipity: cannot take advantage of information from other users
- Dependency on content descriptions
- Limited Similarity Detection
- Scalability Issues, when the data is large enough

COLLABORATIVE FILTERING

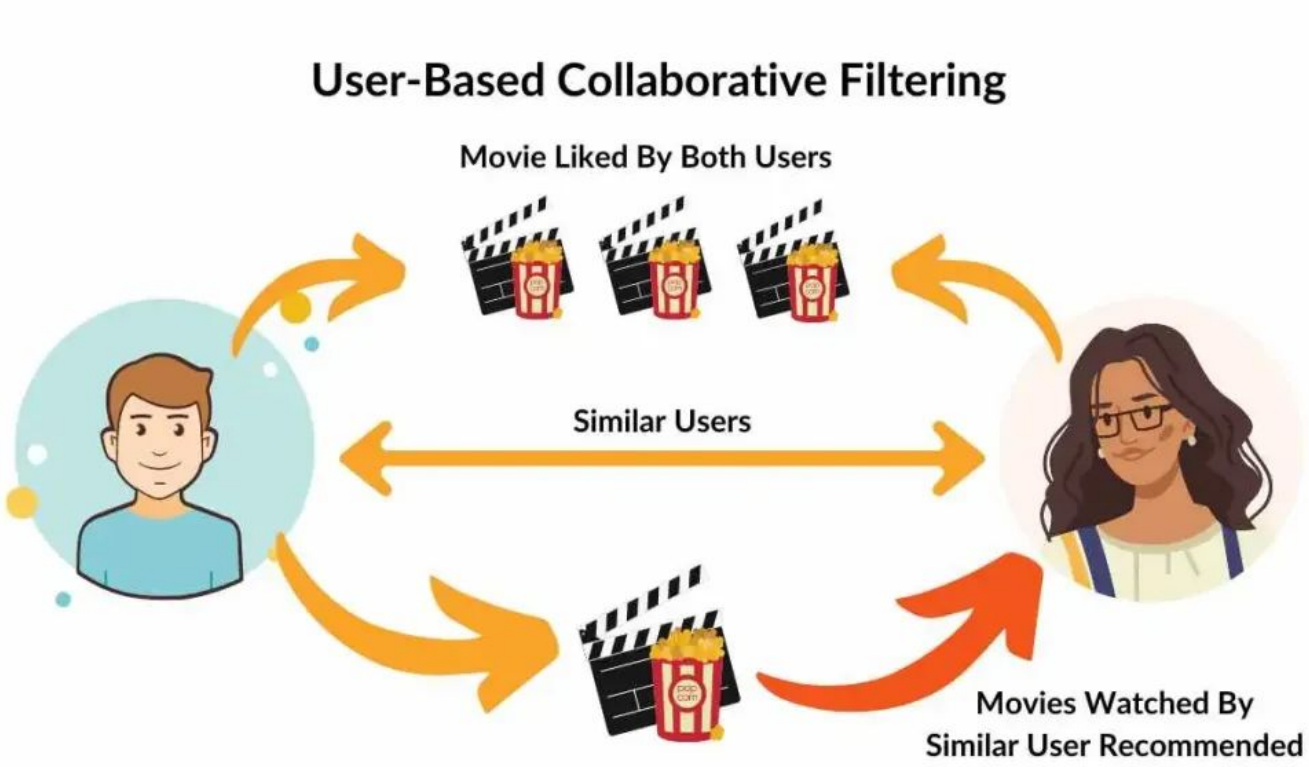
- Generates recommendations by leveraging the preferences or behaviors of similar users (User-User) or the similarity between items (Item-Item).

- ***Key idea:***

  - Users who have agreed in the past will likely agree in the future.

  - Items that were rated similarly in the past will likely be rated similarly in the future.

USER–USER COLLABORATIVE FILTERING



- Compares users based on their interactions (ratings, clicks, or purchases).

- Workflow:

  - Identify users similar to the target user (e.g., User A and User B).

  - Recommend items that these similar users liked but the target user hasn’t interacted with yet.

USER–USER COLLABORATIVE FILTERING

1. DATA PREPROCESSING

```
r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']

ratings_base = pd.read_csv('ml-100k/ub.base', sep='\t', names=r_cols, encoding='latin-1')
ratings_test = pd.read_csv('ml-100k/ub.test', sep='\t', names=r_cols, encoding='latin-1')

rate_train = ratings_base.values
rate_test = ratings_test.values

# indices start from 0
rate_train[:, :2] -= 1
rate_test[:, :2] -= 1
ratings_base.head()
```

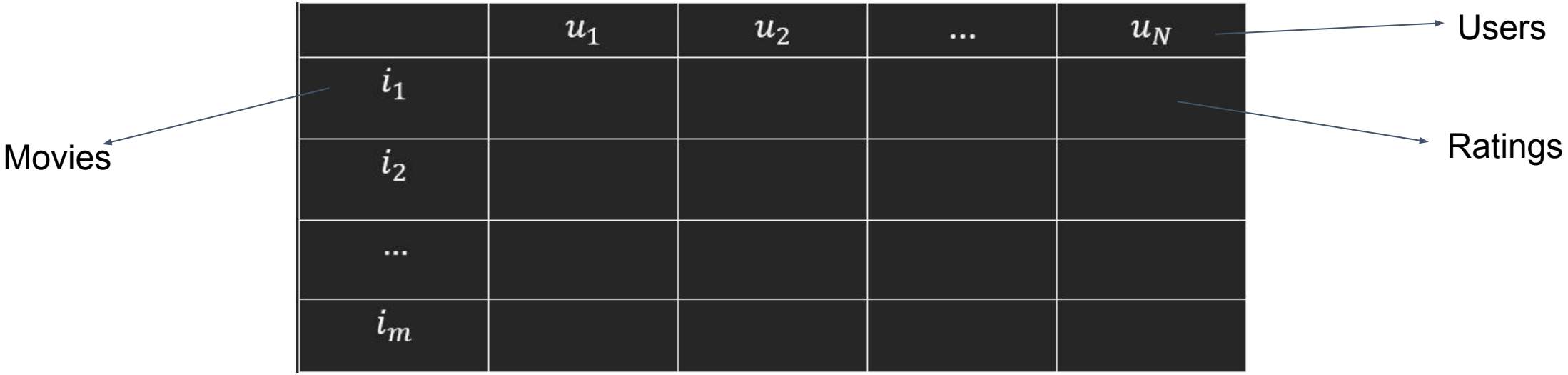
	user_id	movie_id	rating	unix_timestamp
0	0	0	5	874965758
1	0	1	3	876893171
2	0	2	4	878542960
3	0	3	3	876893119
4	0	4	3	889751712

CAN BE USED WITH
THE SAME PURPOSE
AS THE UTILITY MATRIX

USER–USER COLLABORATIVE FILTERING

2. MAIN ALGORITHM (MODEL)

- Utility matrix:



	u_1	u_2	...	u_N
i_1				
i_2				
...				
i_m				

First task: Find the ***SIMILARITY MATRIX*** between ***USERS***
(Measure the ***similarity*** of each pair of users in ***rating the movies***)

USER–USER COLLABORATIVE FILTERING

2. MAIN ALGORITHM (MODEL)

- How to solve the problem of SPARSE UTILITY MATRIX ?

No data recorded !

	u_1	u_2	...	u_n
i_1	$y_{u_1,1}$	$y_{u_2,1}$	...	$y_{u_n,1}$
i_2	?	?		
...				
i_m	?	$y_{u_2,m}$		$y_{u_n,m}$

$\overline{u_1}$	$\overline{u_2}$	...	$\overline{u_n}$

Fill the missing values in the utility matrix by the average rating of known users

USER–USER COLLABORATIVE FILTERING

2. MAIN ALGORITHM (MODEL)

- Normalization:

	u_1	u_2	...	u_n
i_1	$y_{u_1,1} - \overline{u_1}$	$y_{u_2,1} - \overline{u_2}$	...	$y_{u_n,1} - \overline{u_n}$
i_2	0	0		
...				
i_m	0	$y_{u_2,m} - \overline{u_2}$		$y_{u_n,m} - \overline{u_n}$
	$\overline{u_1}$	$\overline{u_2}$	...	$\overline{u_n}$

User-based normalization

Eliminate User Bias

Reduce Impact of Extreme Ratings

Improve Algorithm Performance

Increase Prediction Accuracy

USER–USER COLLABORATIVE FILTERING

2. MAIN ALGORITHM (MODEL)

- Similarity functions between users

COSINE SIMILARITY:

$$\text{cosine\_similarity}(\mathbf{u}_1, \mathbf{u}_2) = \cos(\mathbf{u}_1, \mathbf{u}_2) = \frac{\mathbf{u}_1^T \mathbf{u}_2}{||\mathbf{u}_1||_2 . ||\mathbf{u}_2||_2}$$

1: Perfect similarity.

0: No similarity.

-1: Perfect opposition.

0.5 - 1: Strong to moderate similarity.

0 - 0.5: Weak similarity.

-0.5 - 0: Weak opposition.

-1 - -0.5: Strong opposition.

PEARSON SIMILARITY:

$$\text{pearson similarity}(u, v) = \frac{\sum_{i=1}^{m}(y_{u,i} - \bar{y_u})(y_{v,i} - \bar{y_v})}{\sqrt{\sum_{i=1}^{m}(y_{u,i} - \bar{y_u})^2}\sqrt{\sum_{i=1}^{m}(y_{v,i} - \bar{y_v})^2}}$$

⇒ We will use ***Cosine Similarity function***!

USER–USER COLLABORATIVE FILTERING

2. MAIN ALGORITHM (MODEL)

❖ *RATING PREDICTION:*

From Cosine similarity matrix of users → For each user, find out ***k*** nearest user's similarity

Hyperparameter

$$\hat{y}_{u,i} = \frac{\sum_{v \in N(u)} \text{sim}(u,v) \cdot y_{v,i}}{\sum_{v \in N(u)} \text{sim}(u,v)}$$

$N(u)$: $\boldsymbol{k}$ nearest users' similarity of user $\boldsymbol{u}$

$\widehat{y_{u,i}}$: Rating prediction

$y_{v,i}$: Rating on movies after normalization

USER-USER COLLABORATIVE FILTERING

3. EVALUATING:

```python
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from scipy import sparse
class CF(object):
    """docstring for CF"""
    def __init__(self, Y_data, k, dist_func = cosine_similarity, uuCF = 1):
        self.uuCF = uuCF # user-user (1) or item-item (0) CF
        self.Y_data = Y_data if uuCF else Y_data[:, [1, 0, 2]]
        self.k = k
        self.dist_func = dist_func
        self.Ybar_data = None
        # number of users and items. Remember to add 1 since id starts from 0
        self.n_users = int(np.max(self.Y_data[:, 0])) + 1
        self.n_items = int(np.max(self.Y_data[:, 1])) + 1
```

```python
rs = CF(rate_train, k = 30, uuCF = 1)
rs.fit()

n_tests = rate_test.shape[0]
SE = 0 # squared error
abs_error = 0 # absolute error
for n in range(n_tests):
    pred = rs.pred(rate_test[n, 0], rate_test[n, 1], normalized = 0)
    SE += (pred - rate_test[n, 2])**2
    abs_error += abs(pred - rate_test[n, 2])

RMSE = np.sqrt(SE/n_tests)
MAE = abs_error/n_tests
print('User-user CF, RMSE =', RMSE)
print('User-user CF, MAE =', MAE)
```

We get the result:

```
User-user CF, RMSE = 0.9951961876266519
User-user CF, MAE = 0.7866068486443497
```

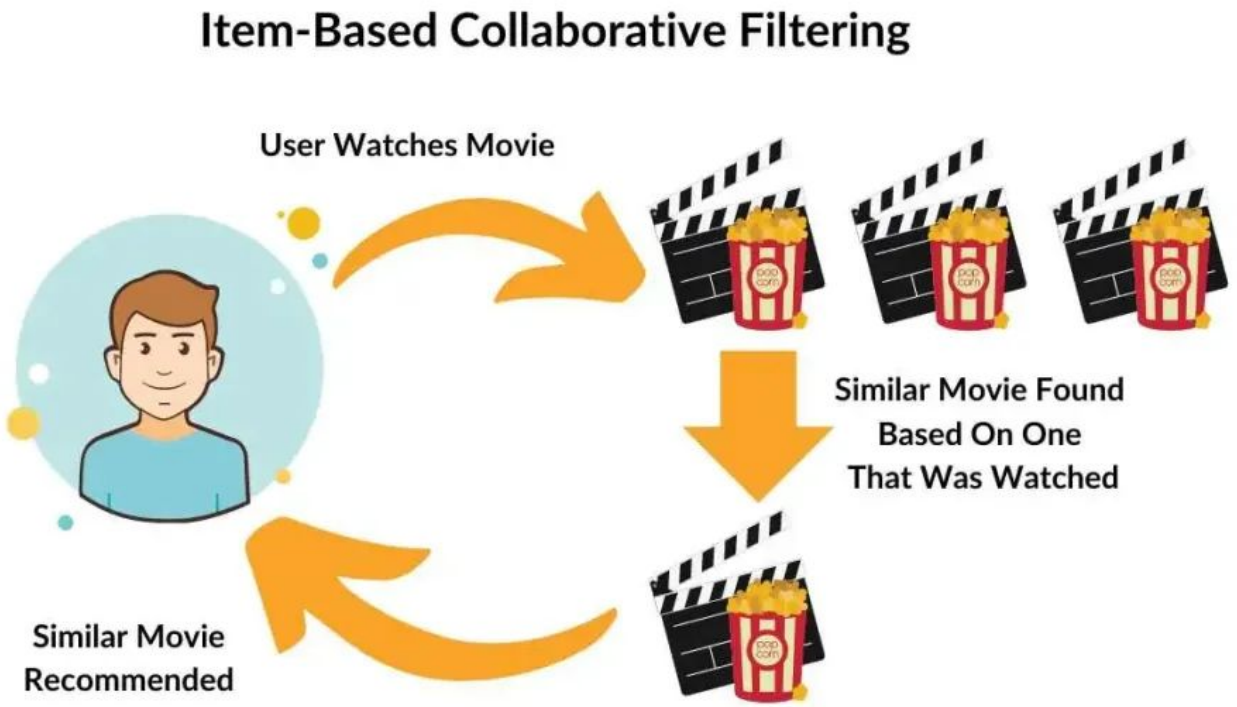
ITEM-ITEM COLLABORATIVE FILTERING

From the evaluating result of *User-based filtering*:

- High Computational Cost
- Performance Decline with Sparse Data (Sparse Utility Matrix)
- Cold Start Problem for Users
- Number of users is usually much larger than number of items
  → Similarity matrix storing is impossible in most cases

⇒ ***ITEM-ITEM COLLABORATIVE FILTERING*** is a better choice !!

ITEM–ITEM COLLABORATIVE FILTERING



- Compares items based on how users interact with them.

- Workflow:
  - Find items similar to what the user has already interacted with.
  - Recommend those items to the user.

ITEM-ITEM COLLABORATIVE FILTERING

NOTE

Item-item collaborative filtering algorithm is **SIMILAR** to user-user collaborative filtering algorithm

EXCEPT FROM

taking the FEATURE MATRIX (UTILITY MATRIX) obtained by taking *transposition*

ITEM-ITEM COLLABORATIVE FILTERING

❖ Utility matrix:

ALL THE PROCESS IS SIMILAR AS USER-BASED FILTERING, BUT NOW WE WORK WITH ITEMS INSTEAD OF USERS

	u_1	u_2	...	u_N
i_1				
i_2				
...				
i_m				

Movies

Users

Ratings

TRANSPOSE:

	i_1	i_2	...	i_m
u_1				
u_2				
...				
u_m				

ITEM-ITEM COLLABORATIVE FILTERING

3. EVALUATING:

```
rs = CF(rate_train, k = 30, uuCF = 0)
rs.fit()

n_tests = rate_test.shape[0]
SE = 0 # squared error
abs_error = 0 # absolute error
for n in range(n_tests):
    pred = rs.pred(rate_test[n, 0], rate_test[n, 1], normalized = 0)
    SE += (pred - rate_test[n, 2])**2
    abs_error += abs(pred - rate_test[n, 2])

RMSE = np.sqrt(SE/n_tests)
MAE = abs_error/n_tests
print('Item-item CF, RMSE =', RMSE)
print('Item-item CF, MAE =', MAE)
```

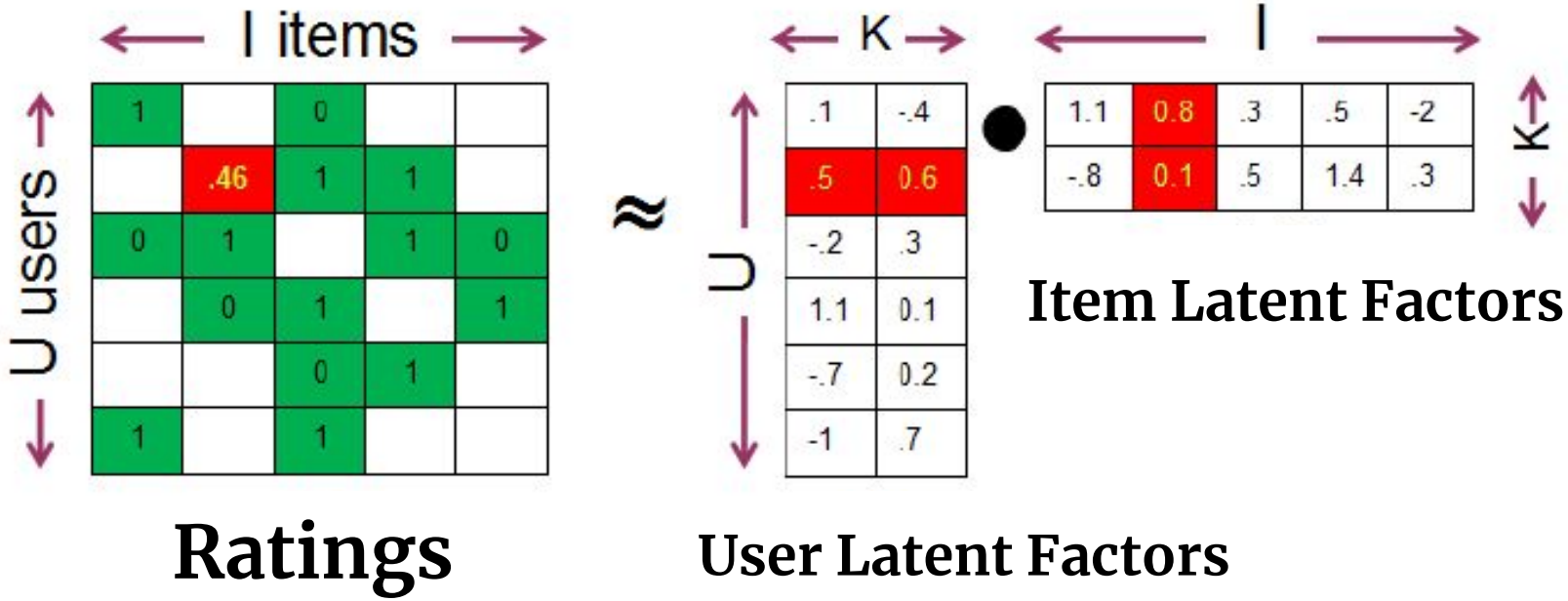
```
Item-item CF, RMSE = 0.9867636589040529
Item-item CF, MAE = 0.7842367973498454
```

MATRIX FACTORIZATION

KEY IDEA: Factorize utility matrix Y:

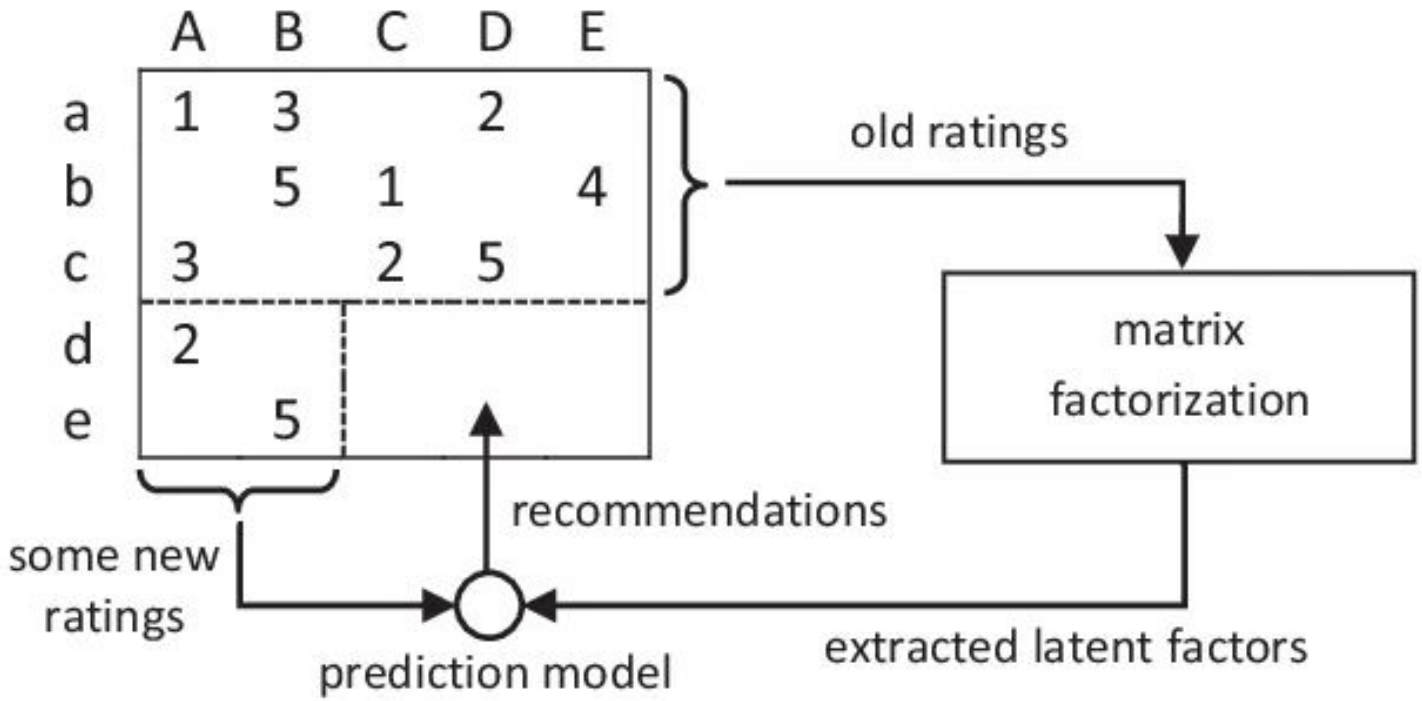
$$Y \approx X \times W$$

Let's call X: user_features and W: item_features (movie)



MATRIX FACTORIZATION

Procedure:



MATRIX FACTORIZATION

Assume we have an utility matrix Y like this:

Item / User	Avenger: Infinity Wars	Fast and Furious 9	Forest Gump	The Nun II	Interstellar
					
Ann	5	5	2	3	4
Bing	3	3	5	2	4
Cane	2	2	2	5	2
Dart	5	5	2	3	4

Factorize matrix Y:

$$Y \approx X \times W$$

Let's call X: user_features and W: item_features (movie)

MATRIX FACTORIZATION

Feature: Action, Comedy, Horror, ... We call them latent factors because we actually can't see them

X: user_features

Feature / User	Action	Comedy	Horror
Ann	1	0	0
Bing	0	1	0
Cane	0	0	1
Dart	1	0	0

W: item_features

Item / Feature	Avenger: Infinity Wars	Fast and Furious 9	Forest Gump	The Nun II	Interstellar
Action	5	5	2	3	4
Comedy	3	3	5	2	4
Horror	2	2	2	5	2

MATRIX FACTORIZATION

So we don't need Y anymore, just need X and W to produce prediction since

$$\mathbf{X} \times \mathbf{W} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 5 & 5 & 2 & 3 & 4 \\ 3 & 3 & 5 & 2 & 4 \\ 2 & 2 & 2 & 5 & 2 \end{bmatrix} = \begin{bmatrix} 5 & 5 & 2 & 3 & 4 \\ 3 & 3 & 5 & 2 & 4 \\ 2 & 2 & 2 & 5 & 2 \\ 5 & 5 & 2 & 3 & 4 \end{bmatrix} \approx \mathbf{Y}$$

MATRIX FACTORIZATION

So we don't need A anymore, just need X and W to produce prediction since

$$\mathbf{X} \times \mathbf{W} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 5 & 5 & 2 & 3 & 4 \\ 3 & 3 & 5 & 2 & 4 \\ 2 & 2 & 2 & 5 & 2 \end{bmatrix} = \begin{bmatrix} 5 & 5 & 2 & 3 & 4 \\ 3 & 3 & 5 & 2 & 4 \\ 2 & 2 & 2 & 5 & 2 \\ 5 & 5 & 2 & 3 & 4 \end{bmatrix} \approx \mathbf{Y}$$

Example: the rating of user_2 for item_4 can be predicted as:

$$\text{rating} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 3 \\ 2 \\ 5 \end{bmatrix} = 2$$

MATRIX FACTORIZATION

1. DATA PREPROCESSING

Normalize utility matrix:

item_id	1	2	3	4	5	6	7	8	9	10	...	1673	1674	1675	1676	1677	1678	1679	1680	1681	1682
user_id																					
1	5.0	3.0	4.0	3.0	3.0	5.0	4.0	1.0	5.0	3.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
939	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
940	0.0	0.0	0.0	2.0	0.0	0.0	4.0	5.0	3.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
941	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
942	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
943	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

943 rows × 1680 columns

MATRIX FACTORIZATION

1. DATA PREPROCESSING

Transform test matrix:

item_id	1	2	3	4	5	6	7	8	9	10	...	1673	1674	1675	1676	1677	1678	1679	1680	1681	1682
user_id																					
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	0	0	0	0	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	0	0	0	0	0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	0	0	0	0	0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	0	0	0	0	0
5	4.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
939	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	0	0	0	0	0
940	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	0	0	0	0	0
941	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0	...	0	0	0	0	0	0	0	0	0	0
942	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	0	0	0	0	0
943	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	0	0	0	0	0

943 rows × 1680 columns

MATRIX FACTORIZATION

2. MAIN ALGORITHM (MODEL)

How to find **X,W**:

1. Gradient Descent
2. Singular Values Decomposition

MATRIX FACTORIZATION

How to find X,W:

1. Gradient Descent

   - First, take a first guess of X and W

$$\mathbf{X} = \begin{bmatrix} 0.6 & 0.2 & 0.3 \\ 0.1 & 0.8 & 0.2 \\ 0.1 & 0.2 & 0.7 \\ 0.9 & 0.1 & 0.2 \end{bmatrix} \qquad \mathbf{W} = \begin{bmatrix} 4.8 & 3.5 & 1.4 & 3.5 & 2.2 \\ 3.6 & 2.4 & 4.3 & 1.9 & 3.4 \\ 2.4 & 2.8 & 2.3 & 3.9 & 1.5 \end{bmatrix}$$

MATRIX FACTORIZATION

- Second, multiply X and W to produce $\hat{Y}$

$$\mathbf{X} = \begin{bmatrix} 0.6 & 0.2 & 0.3 \\ 0.1 & 0.8 & 0.2 \\ 0.1 & 0.2 & 0.7 \\ 0.9 & 0.1 & 0.2 \end{bmatrix} \mathbf{x}\, \mathbf{W} = \begin{bmatrix} 4.8 & 3.5 & 1.4 & 3.5 & 2.2 \\ 3.6 & 2.4 & 4.3 & 1.9 & 3.4 \\ 2.4 & 2.8 & 2.3 & 3.9 & 1.5 \end{bmatrix} = \hat{\mathbf{Y}} = \begin{bmatrix} 4.32 & 3.42 & 2.39 & 3.65 & 2.45 \\ 3.84 & 2.83 & 4.04 & 2.65 & 3.24 \\ 2.88 & 2.79 & 2.61 & 3.46 & 1.95 \\ 5.16 & 3.95 & 2.15 & 4.12 & 2.62 \end{bmatrix}$$

MATRIX FACTORIZATION

- Thirdly, compute loss function

$$\mathscr{L}(X,W) = \frac{1}{2s} \cdot \text{Differences} + \frac{\lambda}{2} \cdot (\|X\|_F^2 + \|W\|_F^2)$$

$$\mathbf{X} \times \mathbf{W} = \hat{\mathbf{Y}} = \begin{bmatrix} 4.32 & 3.42 & 2.39 & 3.65 & 2.45 \\ 3.84 & 2.83 & 4.04 & 2.65 & 3.24 \\ 2.88 & 2.79 & 2.61 & 3.46 & 1.95 \\ 5.16 & 3.95 & 2.15 & 4.12 & 2.62 \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} 5 & 5 & 2 & 3 & 4 \\ 3 & 3 & 5 & 2 & 4 \\ 2 & 2 & 2 & 5 & 2 \\ 5 & 5 & 2 & 3 & 4 \end{bmatrix}$$

$$\Rightarrow \quad \mathscr{L}(X,W) = 1.1045$$

MATRIX FACTORIZATION

- Lastly, optimize loss function using gradient descent algorithm and update new X, W until convergence.

$$\mathbf{X} \times \mathbf{W} = \hat{\mathbf{Y}} = \begin{bmatrix} 4.32 & 3.42 & 2.39 & 3.65 & 2.45 \\ 3.84 & 2.83 & 4.04 & 2.65 & 3.24 \\ 2.88 & 2.79 & 2.61 & 3.46 & 1.95 \\ 5.16 & 3.95 & 2.15 & 4.12 & 2.62 \end{bmatrix} \Rightarrow \mathscr{L}(X, W) = 1.1045$$

$$\mathbf{X} = \begin{bmatrix} 0.6 & 0.2 & 0.3 \\ 0.1 & 0.8 & 0.2 \\ 0.1 & 0.2 & 0.7 \\ 0.9 & 0.1 & 0.2 \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} 4.8 & 3.5 & 1.4 & 3.5 & 2.2 \\ 3.6 & 2.4 & 4.3 & 1.9 & 3.4 \\ 2.4 & 2.8 & 2.3 & 3.9 & 1.5 \end{bmatrix}$$

Optimize Loss Using Gradient Descent

After first iteration, X, W seem to converge

MATRIX FACTORIZATION

2. Singular Values Decomposition (SVD)

- It has been proved that an arbitrary matrix A (size m×n) can be factorize into the form:

$$A = U\Sigma V^T$$

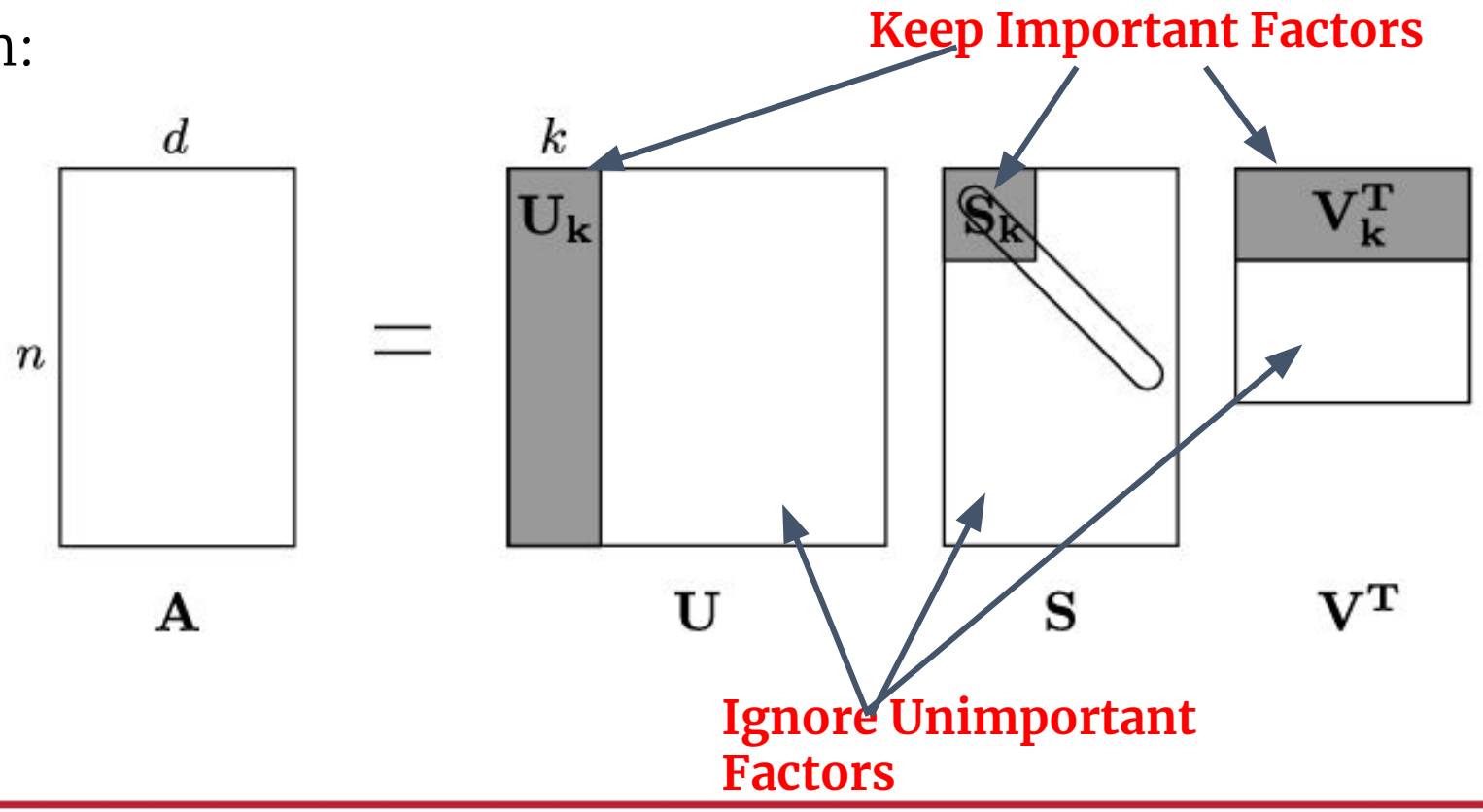
$$\mathbf{Y} = \begin{bmatrix} 5 & 5 & 2 & 3 & 4 \\ 3 & 3 & 5 & 2 & 4 \\ 2 & 2 & 2 & 5 & 2 \\ 5 & 5 & 2 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 0.57 & 0.41 & -0.11 & -0.71 \\ 0.48 & -0.45 & 0.75 & 0 \\ 0.36 & -0.68 & -0.64 & 0 \\ 0.57 & 0.41 & -0.11 & -0.71 \end{bmatrix} \begin{bmatrix} 15{,}51 & 0 & 0 & 0 & 0 \\ 0 & 3.34 & 0 & 0 & 0 \\ 0 & 0 & 3.24 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.50 & 0.50 & 0.35 & 0.40 & 0.46 \\ 0.41 & 0.41 & -0.60 & -0.55 & 0.03 \\ -0.05 & -0.05 & 0.63 & -0.73 & 0.26 \\ -0.75 & 0.61 & -0.09 & 0.01 & 0.22 \end{bmatrix}$$

MATRIX FACTORIZATION

- How to apply SVD in recommender system:

  Only extract k dimensions of latent factors from utility matrix without losing too much information

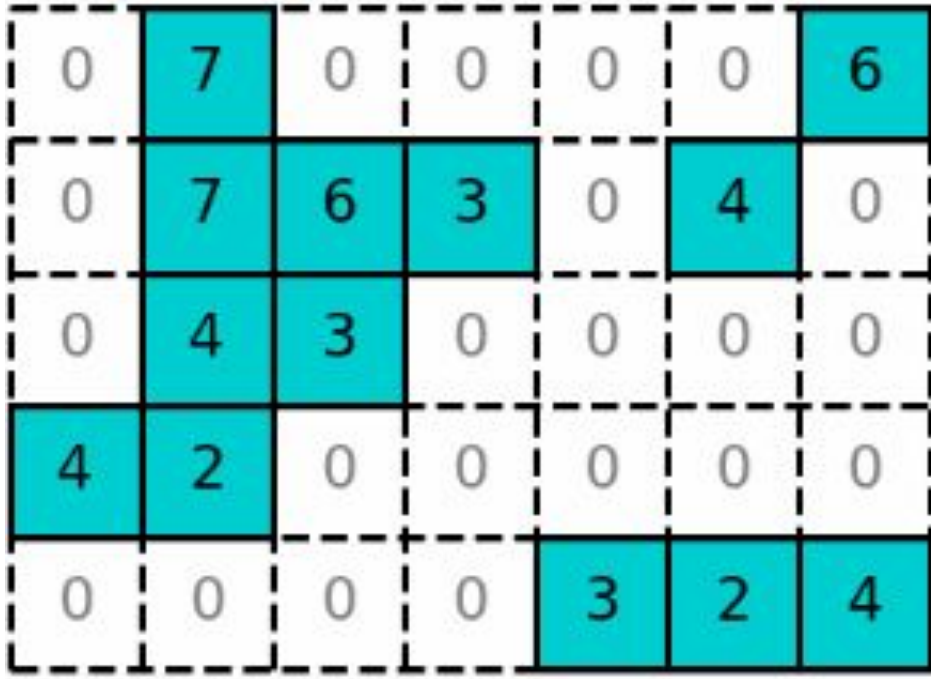
- Intuition:



MATRIX FACTORIZATION

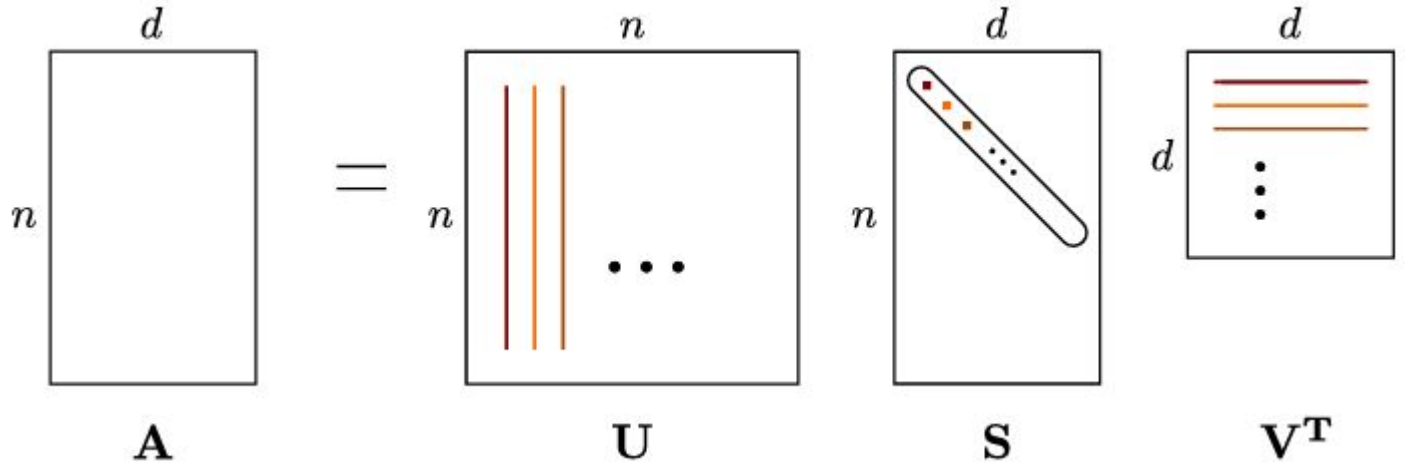
- Why Matrix Factorization

1. Handling Sparse Data



MATRIX FACTORIZATION

- Why Matrix Factorization
2. Latent Feature Extraction



Relation between user and movie

Combination of genre, i.e Action Comedy

Trending

MATRIX FACTORIZATION

- Why Matrix Factorization

3. Scalability

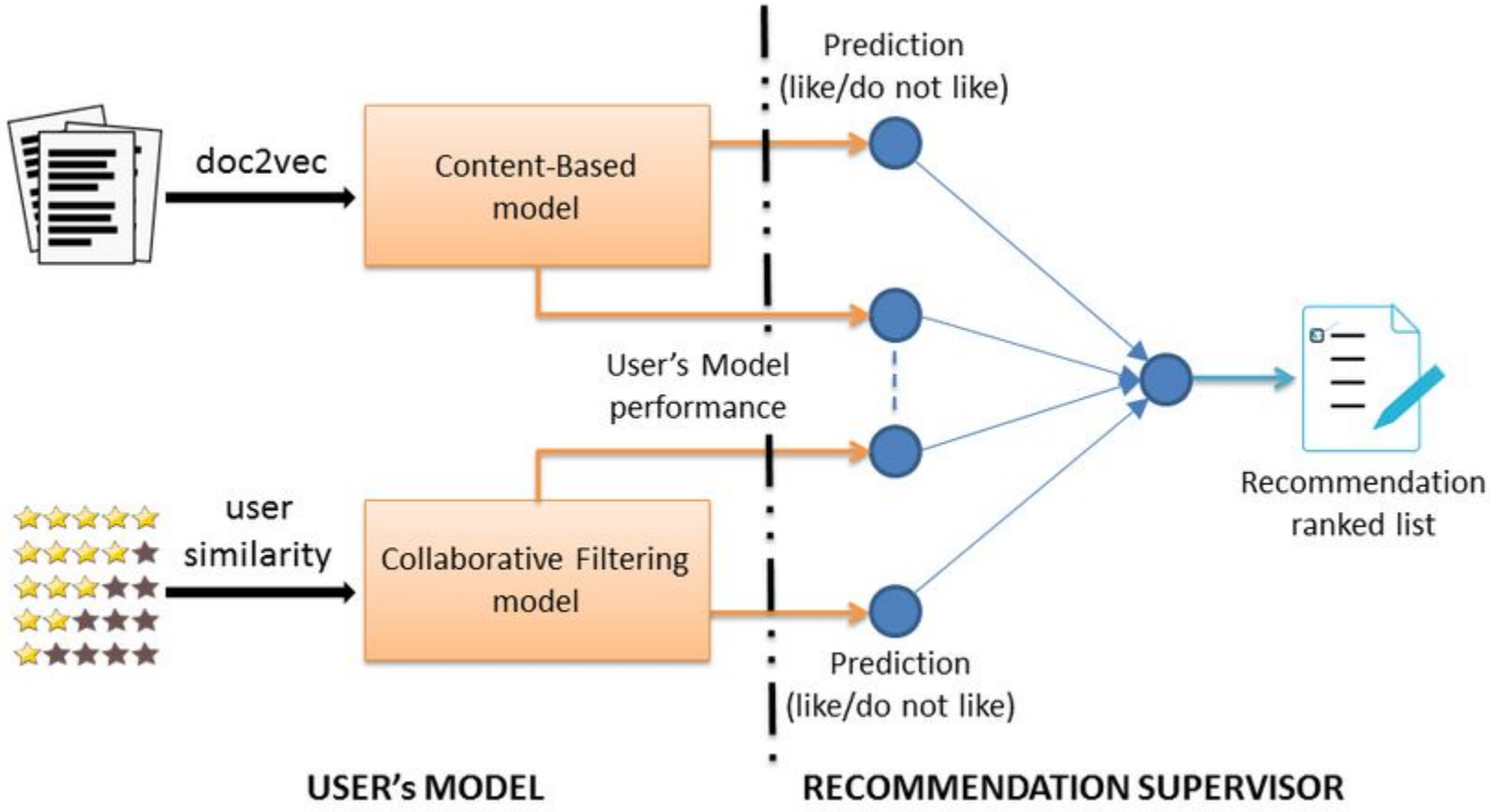
	user_id	movie_id	rating	unix_timestamp
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291
...	...	...	...	...
1000204	6040	1091	1	956716541
1000205	6040	1094	5	956704887
1000206	6040	562	5	956704746
1000207	6040	1096	4	956715648
1000208	6040	1097	4	956715569

1000209 rows × 4 columns

MATRIX FACTORIZATION

- Why Matrix Factorization

4. Easy to Combine with Other Techniques



MATRIX FACTORIZATION

- Drawback of Matrix Factorization

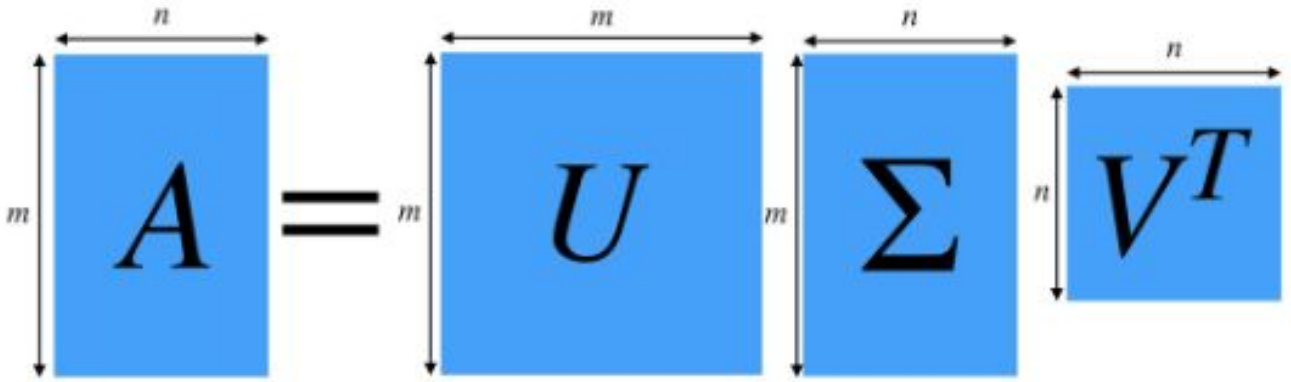
1. Cold-Start Problem

$$R = \begin{pmatrix} 1 & ? & 2 & ? & ? \\ ? & ? & ? & ? & 4 \\ 2 & ? & 4 & 5 & ? \\ ? & ? & 3 & ? & ? \\ ? & 1 & ? & 3 & ? \\ 5 & ? & ? & ? & 2 \end{pmatrix} \begin{matrix} \text{Alice} \\ \text{Bob} \\ \text{Charlie} \\ \text{Daniel} \\ \text{Eric} \\ \text{Frank} \end{matrix}$$

MATRIX FACTORIZATION

- Drawback of Matrix Factorization
2. High Computational Requirements

$$A = U\Sigma V^T$$



MATRIX FACTORIZATION

- Drawback of Matrix Factorization

3. Lack of Interpretability

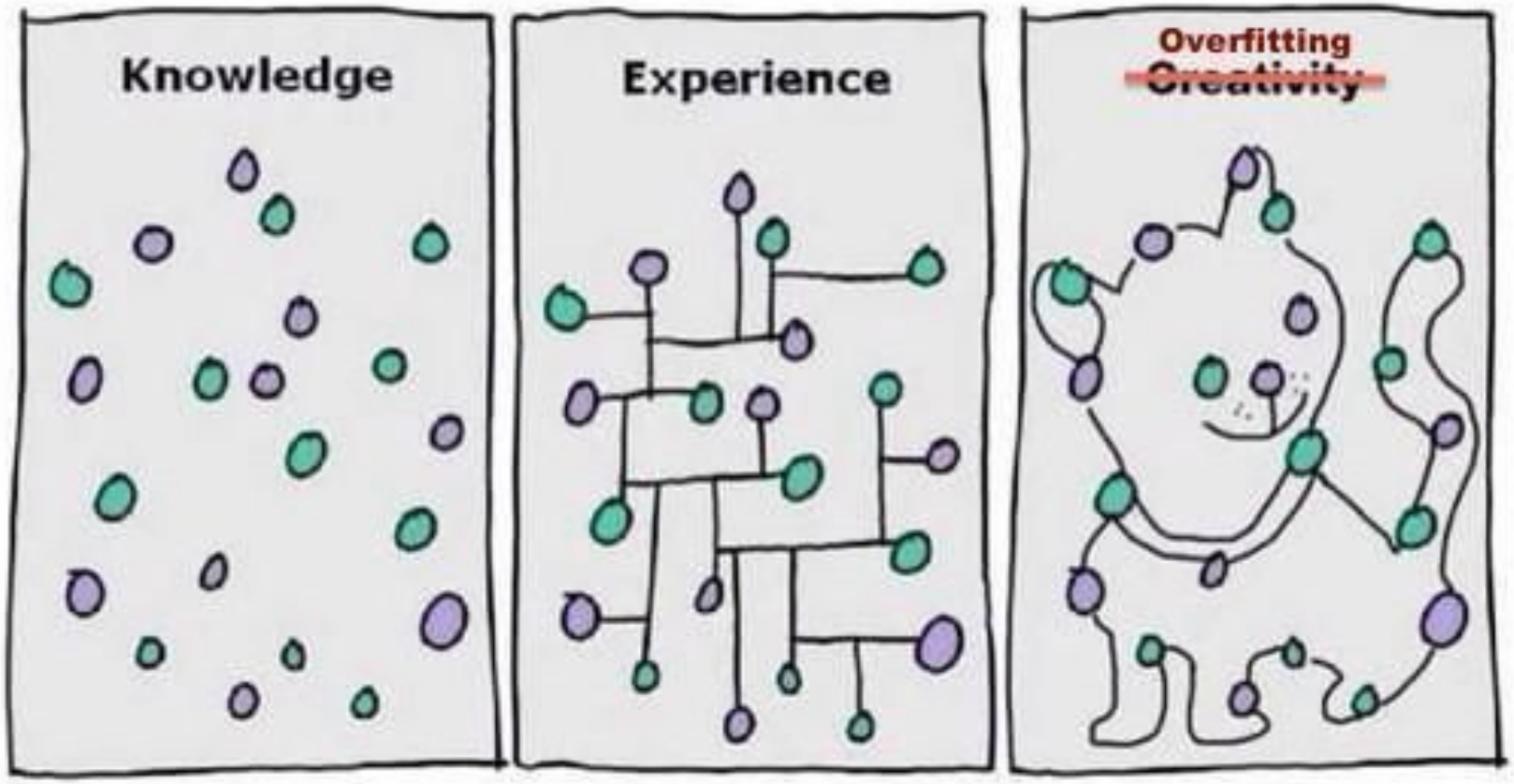
what is this all about??

	movie_feature
0	[-0.008237677625811513, -0.04385397562328572, ...
1	[-0.009534130009529785, 0.010758593294863972, ...
2	[0.026462292374362 51, -0.042173424255894286, -...
3	[0.009378435411580649, -0.002215083366000788, ...
4	[-0.026377899617597633, 0.013580775148324113, ...
...	...
1675	[0.0024835308322352 93, 0.0005008313242390305, ...
1676	[8.649112351068208e-05, 1.7441886913797126e-05...
1677	[0.0012850109778729877, 0.0002591366055764 1386...
1678	[3.648748852488039e-05, -3.5980097420737646e-0...
1679	[-1.4293525254351376e-05, 0.0002628641679064 05...

MATRIX FACTORIZATION

- Drawback of Matrix Factorization

4. Overfitting



3. Numerical Results

A. With Normal Gradient Descent Approaches

1. Root Mean Squared Error

```
iter = 10 , loss = 5.625837968750378 , RMSE train = 1.1819136932345946
iter = 20 , loss = 2.6165951032805994 , RMSE train = 1.0065835053733445
iter = 30 , loss = 1.3234555241335497 , RMSE train = 0.9968083367117461
iter = 40 , loss = 0.7344752009461328 , RMSE train = 0.9962419353128245
iter = 50 , loss = 0.4645951835200977 7 , RMSE train = 0.996190269109156 4
iter = 60 , loss = 0.3408501183723601 , RMSE train = 0.996182649855542
iter = 70 , loss = 0.2841053324274915 , RMSE train = 0.9961812464193471
iter = 80 , loss = 0.2580837643115683 , RMSE train = 0.9961809717086768
iter = 90 , loss = 0.2461509110216751 6 , RMSE train = 0.996180917287143 2
iter = 100 , loss = 0.2406787784252933 4 , RMSE train = 0.996180906504838 8

Item-based MF, RMSE = 1.0498047443678356
```

3. Numerical Results

A. With Normal Gradient Descent Approaches

2. Predicted rating for user_id 1

Ratings in test set

item_id	
20	4.0
33	4.0
61	4.0
117	3.0
155	2.0
160	4.0
171	5.0
189	3.0
202	5.0
265	4.0

Predicted ratings

movie_id	
20	3.344263
33	3.460674
61	3.857143
117	3.690625
155	3.093750
160	3.426229
171	3.929825
189	4.135595
202	3.755814
265	3.868293

3. Numerical Results

A. With Normal Gradient Descent Approaches

3. Top 10 recommendation for user_id 1

```
Top 10 recommended items for user 1:
1. Umbrellas of Cherbourg, The (Parapluies de Cherbourg, Les) (1964), rating: 5.0
2. Young Guns II (1990), rating: 5.0
3. Grosse Fatigue (1994), rating: 5.0
4. City of Industry (1997), rating: 5.0
5. Enfer, L' (1994), rating: 4.999999993896226
6. Margaret's Museum (1995), rating: 4.999999969208974
7. Kim (1950), rating: 4.9999999839784594
8. Celluloid Closet, The (1995), rating: 4.999999809871991
9. Favor, The (1994), rating: 4.9999999688765547
10. Simple Wish, A (1997), rating: 4.999998824985527
```

3. Numerical Results

A. With Normal Gradient Descent Approaches

4. Top 10 similar movies with “Toy Story (1995)”

```
0. Toy Story (1995), score = 0.9999999999999999
1. Amazing Panda Adventure, The (1995), score = 0.7916097432122087
2. True Lies (1994), score = 0.7638840494613782
3. Of Human Bondage (1934), score = 0.7635907558820832
4. Ulee's Gold (1997), score = 0.7597144805484715
5. Jumanji (1995), score = 0.7594638126812567
6. Assignment, The (1997), score = 0.7594215636262232
7. Coneheads (1993), score = 0.738662314098027
8. Mute Witness (1994), score = 0.7264105567070672
9. Incognito (1997), score = 0.7229390598221068
10. Belle de jour (1967), score = 0.719591882369232
```

3. Numerical Results

B. With Singular Values Decomposition Approaches

1. Root Mean Squared Error

```
With 5 latent factors, RMSE: 1.01132858421744
With 20 latent factors, RMSE: 1.0021807187839333
With 50 latent factors, RMSE: 1.0077798294022753
With 100 latent factors, RMSE: 1.0185106902349355
With 200 latent factors, RMSE: 1.0314867027709214
With 500 latent factors, RMSE: 1.0422729763497751
```

3. Numerical Results

B. With Singular Values Decomposition Approaches

2. Predicted rating for user_id 1

item_id	Ratings in test set	With 20 latent factors	With 50 latent factors	With 200 latent factors
20	4.0	3.479526	3.455037	3.655076
33	4.0	3.437117	3.927906	3.730119
61	4.0	3.924983	4.012259	3.773611
117	3.0	3.893334	2.872215	3.525825
155	2.0	3.128656	3.138306	3.450575
160	4.0	3.587362	3.502351	3.735698
171	5.0	4.010156	4.020317	3.516926
189	3.0	4.255272	3.648682	3.652427
202	5.0	4.067275	3.913290	3.515839
265	4.0	4.227520	3.654497	3.632145

MATRIX FACTORIZATION

3. Top 10 recommendation for user_id 1

Top 10 recommended items for user 1 (20 latent factors):
1. (Leaving Las Vegas (1995)), rating: 4.84
2. (Jackie Chan's First Strike (1996)), rating: 4.66
3. (Close Shave, A (1995)), rating: 4.50
4. (Scream (1996)), rating: 4.43
5. (Schindler's List (1993)), rating: 4.34
6. (Grand Day Out, A (1992)), rating: 4.26
7. (Hunt for Red October, The (1990)), rating: 4.23
8. (Secrets & Lies (1996)), rating: 4.21
9. (City of Lost Children, The (1995)), rating: 4.16
10. (People vs. Larry Flynt, The (1996)), rating: 4.16

Top 10 recommended items for user 1 (50 latent factors):
1. (Like Water For Chocolate (Como agua para chocolate) (1992)), rating: 4.43
2. (Jackie Chan's First Strike (1996)), rating: 4.41
3. (William Shakespeare's Romeo and Juliet (1996)), rating: 4.34
4. (Conspiracy Theory (1997)), rating: 4.28
5. (People vs. Larry Flynt, The (1996)), rating: 4.27
6. (In the Name of the Father (1993)), rating: 4.27
7. (Lawrence of Arabia (1962)), rating: 4.25
8. (Secrets & Lies (1996)), rating: 4.24
9. (Schindler's List (1993)), rating: 4.17
10. (Beautiful Girls (1996)), rating: 4.15

MATRIX FACTORIZATION

4. Top 10 similar movies with “Toy Story (1995)”

20 latent factors:

0. Toy Story (1995), score = 1.00
1. King of the Hill (1993), score = 0.63
2. Wizard of Oz, The (1939), score = 0.58
3. Beauty and the Beast (1991), score = 0.57
4. It's a Wonderful Life (1946), score = 0.57
5. Blue Sky (1994), score = 0.56
6. Gattaca (1997), score = 0.56
7. Haunted World of Edward D. Wood Jr., The (1995), score = 0.55
8. Three Lives and Only One Death (1996), score = 0.55
9. E.T. the Extra-Terrestrial (1982), score = 0.53
10. Hunchback of Notre Dame, The (1996), score = 0.51

50 latent factors:

0. Toy Story (1995), score = 1.00
1. Touch (1997), score = 0.48
2. Beauty and the Beast (1991), score = 0.45
3. Underworld (1997), score = 0.41
4. Aladdin (1992), score = 0.40
5. Craft, The (1996), score = 0.38
6. Perez Family, The (1995), score = 0.38
7. Set It Off (1996), score = 0.36
8. Lion King, The (1994), score = 0.34
9. Wrong Trousers, The (1993), score = 0.33
10. Ransom (1996), score = 0.32

HUST

hust.edu.vn fb.com/dhbkhn

IMPLEMENTATION

IMPLEMENTATION

- Let’s try to get recommendation for a new user
- Now, let’s install movie_recommendation project

Installation

1. Clone the repository:

```
git clone https://github.com/Duckduck-05/Movie_Recommendation.git
```

2. Navigate to the project directory:

```
cd Movie_Recommendation
```

3. Install the required dependencies:

```
pip install -r requirements.txt
```

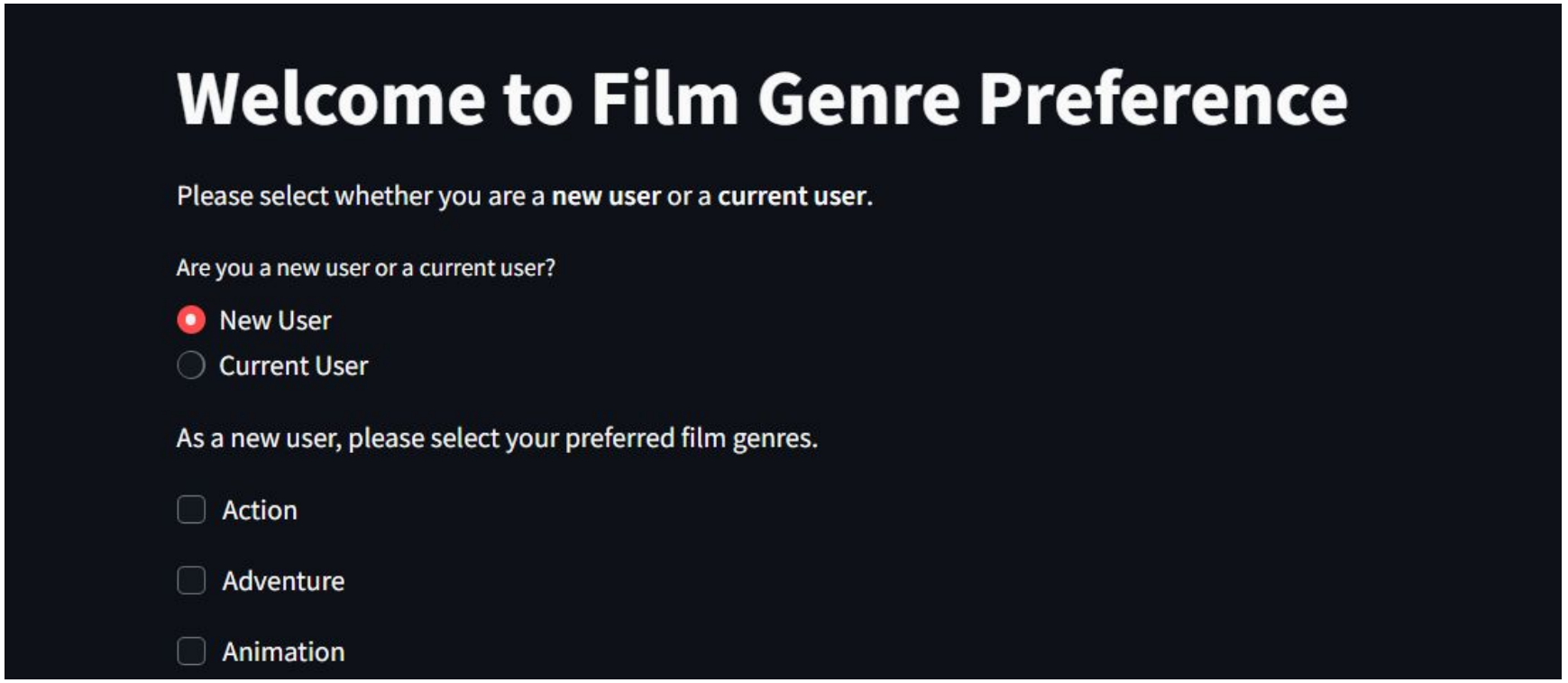
IMPLEMENTATION

- Please running the web by using this instruction

1. Run the application:

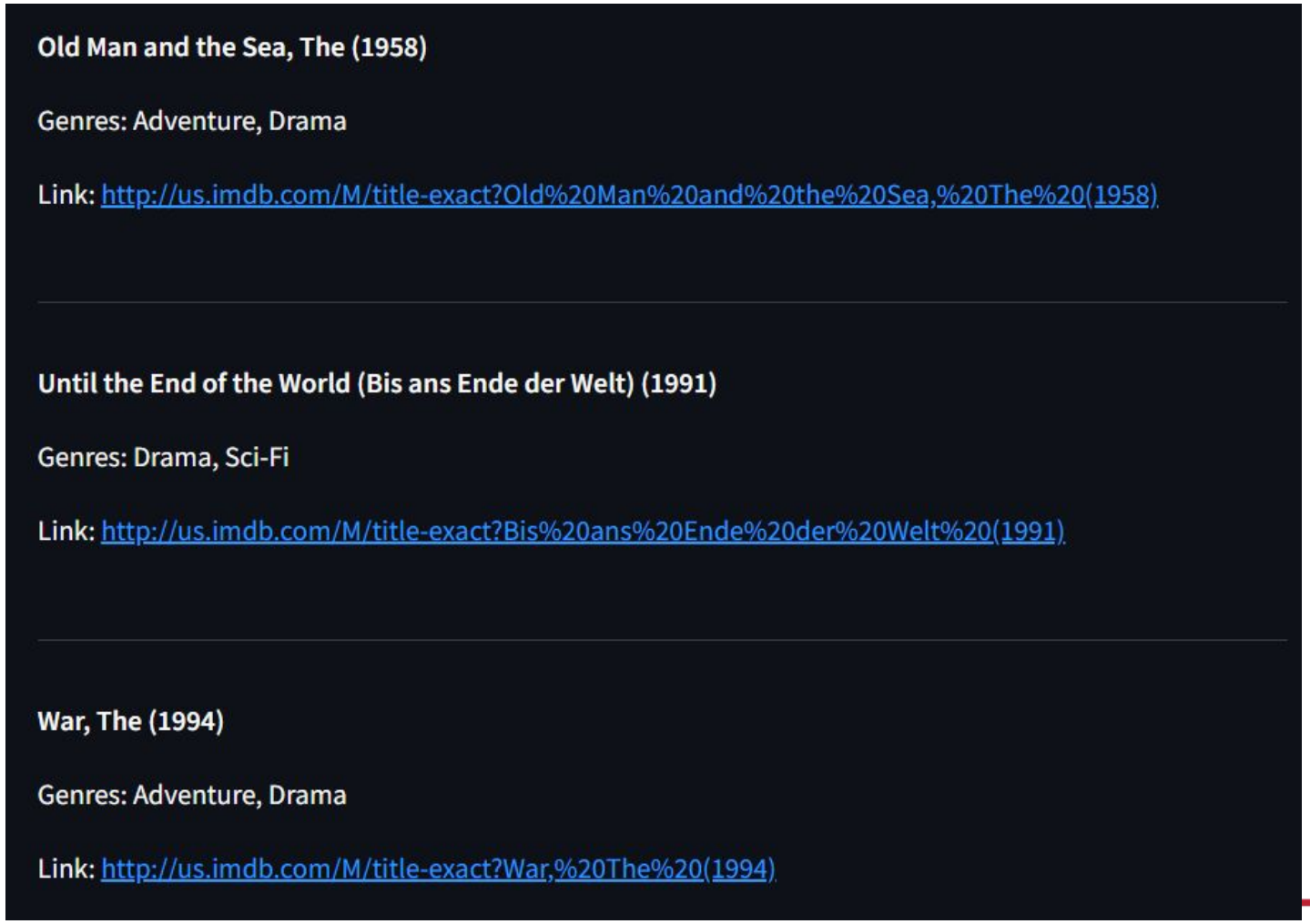
```
streamlit run src/new_web.py
```

- This is your appearance interface. Let’s try with your preference



IMPLEMENTATION

- For example: I am a new user, I prefer Adventure, Drama and Sci-Fi then here are recommend films



- Click the link and enjoy your film 😎

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !