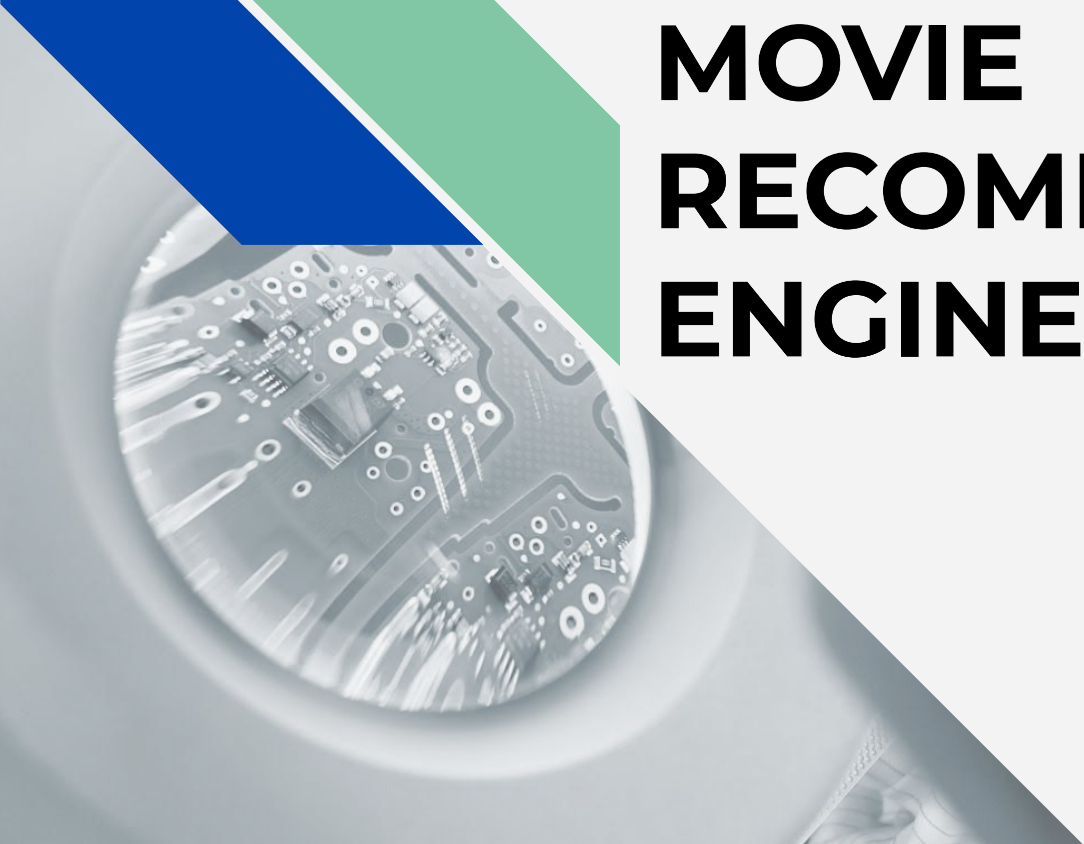


MOVIE RECOMMENDATION ENGINE

Contents

Introduction - Recommender Systems

Recommendation systems produce a ranked list of items on which a user might be interested, in the context of his current choice of an item.

❖ Subclass of Information filtering system that seek to predict the 'rating' or 'preference' that a user would give to them.

❖ Helps deciding in what to wear, what to buy, what stocks to purchase etc.

❖ Applied in variety of applications like movies, books, research articles.

Recommendation systems has mainly two elements Item and User

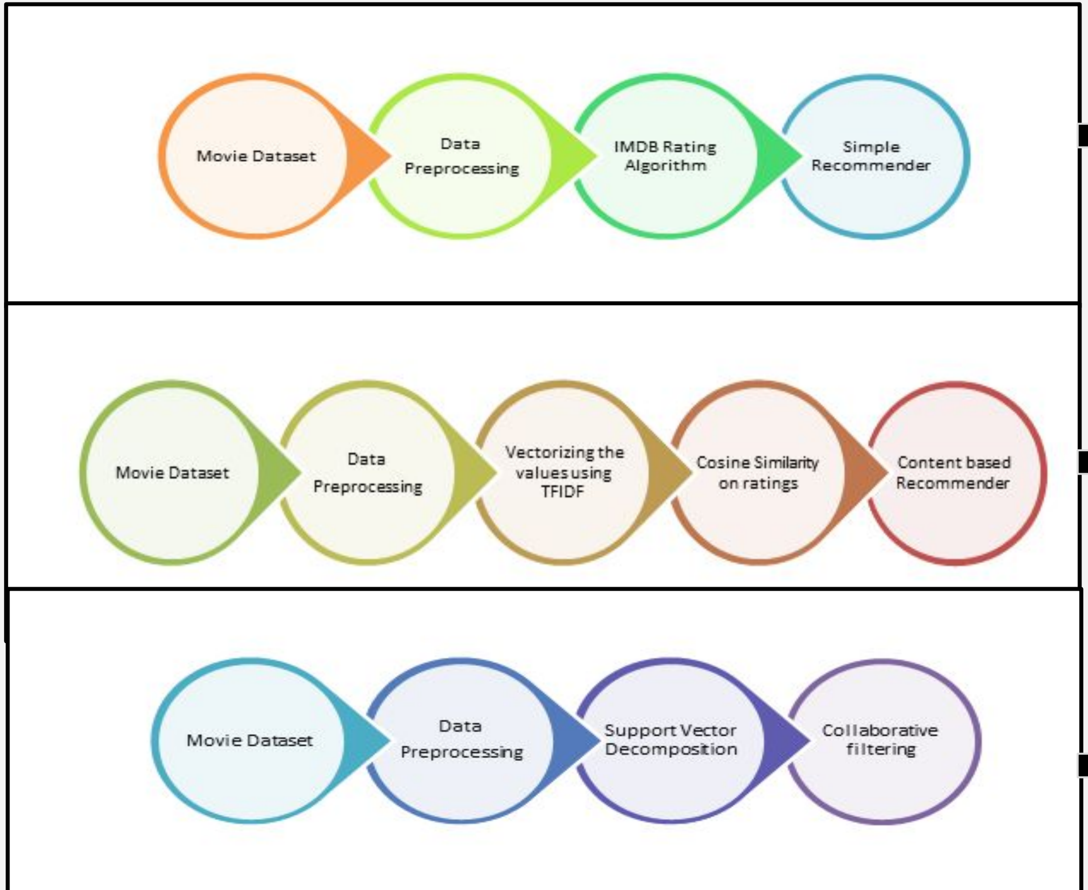
Motivation

Recommendation systems are becoming increasingly important in today’s extremely busy world. People are always short on time with the myriad tasks they need to accomplish in their limited time. Recommendation systems are extremely useful as they help them make the right choices, without having to expend their cognitive resources.

Objective

Nearly everything around us in the internet is a recommendation to us in some way. We have applied different techniques in this project to improve the movies recommended to users rather than old techniques of giving same suggestion to everyone. With this methodology we can make recommendations to user based on his/her watch history, their likes and dislikes and their ratings to a particular genre of movies. We use content-based and collaborative filtering to construct a system that provides precise and accurate recommendations of concerning movies. In the end we also aim to propose a hybrid model which is a mix of content and collaborative filtering and gives more personalized results.

Block Diagram



Simple Recommender

Content Based Recommender

Collaborative Filtering Recommender

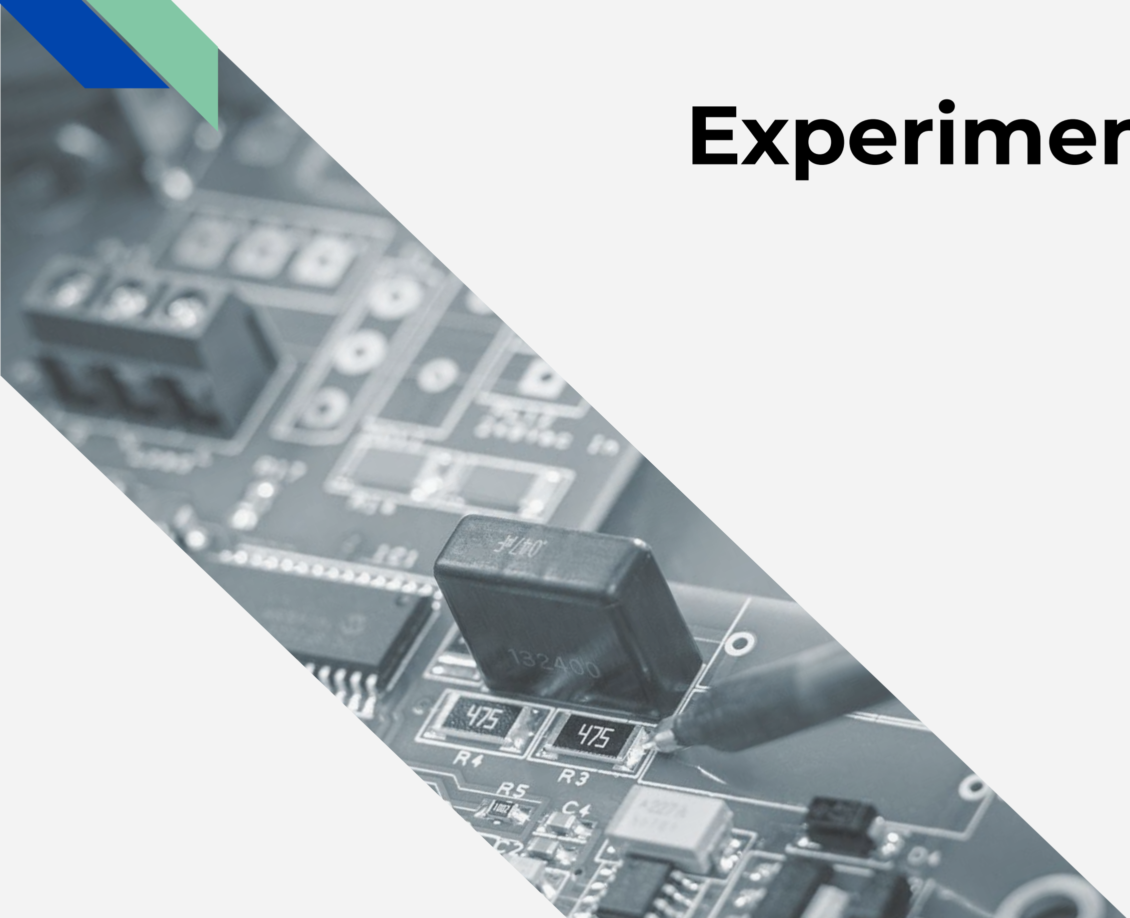
Types of Engines

01 Simple Recommender

02 Content Based Recommendations

03 Collaborative Filtering based Recommendations

Experimental Design



Dataset used

Movielens dataset from Grouplens.org

Our dataset originally contains 45,000 movies and we have built our simple recommender on that dataset only. For our content, collaborative and hybrid model we have used a small dataset of 9,000 rows because of the limited computation power available to us.

	rating	Total Ratings
title		
'71 (2014)	4.000000	1
'Hellboy': The Seeds of Creation (2004)	4.000000	1
'Round Midnight (1986)	3.500000	2
'Salem's Lot (2004)	5.000000	1
'Til There Was You (1997)	4.000000	2
'Tis the Season for Love (2015)	1.500000	1
'burbs, The (1989)	3.176471	17
'night Mother (1986)	3.000000	1
(500) Days of Summer (2009)	3.666667	42
*batteries not included (1987)	3.285714	7

Simple Recommender System

Simple recommender is the most basic type of recommender we have available. It works on it’s specific formula and gives classification based on that regardless of user’s personal taste. The Formula is designed by the designer and we are using IMDB formula in this project. It gives recommendations based on Movie popularity and sometimes genre. Idea is that more popular the movie, higher is the chance that people will like it.

Simple Recommender Design

We have used IMDb rating chart formula for this model.

Mathematically it is shown as :

Weighted rating(WR) = $V.R/(V+M) + M.C/(V+M)$

- V is the number of votes for the movie
- M is the minimum votes required to be listed in the chart
- R is the average rating of the movie
- C is the mean vote across the whole report

Research variables used for the system

- Number of votes for the movie
- Average rating of the movie
- Mean vote of all the movies across the dataset

Results Generated From Simple Recommender

We created a function to create simple recommendations based on algo shown in previous slides. Results are shown here.

```python
def build_chart(genre,percentile=0.85):
  df = md[md['genres']==genre]
  print(df.shape)
  vote_counts = df[df['vote_count'].notnull()]['vote_count'].astype('int')
  vote_averages = df[df['vote_average'].notnull()]['vote_average'].astype('int')
  C = vote_averages.mean()
  m = vote_counts.quantile(percentile)
  print("For {} movies, mean vote average is {} and minimum vote counts consiidered are {}".format(genre,C,m))

  columns_to_include = ['title', 'release_date', 'vote_count', 'vote_average', 'popularity','genres']
  qualified = df[(df['vote_count'].notnull()) & (df['vote_average'].notnull()) & (df['vote_count']>=m)][columns_to_include]
  qualified['vote_count'] = qualified['vote_count'].astype('int')
  qualified['vote_average']=qualified['vote_average'].astype('int')
  qualified['wr'] = qualified.apply(lambda x: (x['vote_count']/(x['vote_count']+m) * x['vote_average']) + (m/(m+x['vote_count']) * C), axis=1)
  qualified = qualified.sort_values('wr',ascending=False)
  return qualified
```

```python
recommendations1= build_chart('Action')
recommendations1.head(15)
```

```
(4489, 24)
For Action movies, mean vote average is 5.167335115864527 and minimum vote counts consiidered are 209.94999999999982
```

	title	release_date	vote_count	vote_average	popularity	genres	wr
15480	Inception	2010-07-14	14075	8	29.1081	Action	7.958368
4135	Scarface	1983-12-08	3017	8	11.2997	Action	7.815703
1910	Seven Samurai	1954-04-26	892	8	15.0178	Action	7.460304
43190	Band of Brothers	2001-09-09	725	8	7.903731	Action	7.363904
14551	Avatar	2009-12-10	12114	7	185.071	Action	6.968779
26564	Deadpool	2016-02-09	11444	7	187.86	Action	6.966984
23753	Guardians of the Galaxy	2014-07-30	10014	7	53.2916	Action	6.962366
26553	Mad Max: Fury Road	2015-05-13	9629	7	29.3618	Action	6.960893
18252	The Dark Knight Rises	2012-07-16	9263	7	20.5826	Action	6.959382
2458	The Matrix	1999-03-30	9079	7	33.3663	Action	6.958578
12588	Iron Man	2008-04-30	8951	7	22.0731	Action	6.957999
26555	Star Wars: The Force Awakens	2015-12-15	7993	7	31.626	Action	6.953094
10122	Batman Begins	2005-06-10	7511	7	28.5053	Action	6.950166
26558	Avengers: Age of Ultron	2015-04-22	6908	7	37.3794	Action	6.945944
42170	Logan	2017-02-28	6310	7	54.581997	Action	6.940986

Disadvantage of Simple Recommender

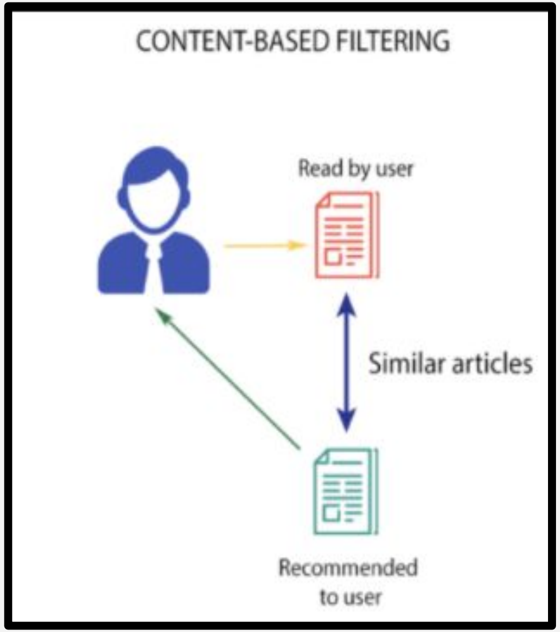
The Simple recommender system suffers some severe limitations,as it gives the same recommendation to everyone, regardless of the user's personal taste.

For eg : If a person who loves romantic movies (and hates action) were to look at our Top 15 Chart, s/he wouldn't probably like most of the movies.

Content based Recommender System

Content based recommender as the name states gives recommendations on the basis of similarity to the content you are already watching. There are many similarity measures available to us out there but we are using cosine similarity because of it's efficiency and correctness. It is also magnitude independent and is very easy to compute.

A good example could be YouTube, where based on your history, it suggests you new videos that you could potentially watch.



Content based Recommender Design

In this system, we computed Term Frequency-Inverse Document Frequency (TF-IDF) vectors for each document. This gives a matrix where each column represents a word in the overview vocabulary (all the words that appear in at least one document), and each row represents a movie, as before.

The TF-IDF score is the frequency of a word occurring in a document, down-weighted by the number of documents in which it occurs. This is done to reduce the importance of words that frequently occur in plot overviews and, therefore, their significance in computing the final similarity score.

We used the cosine similarity to calculate a numeric quantity that denotes the similarity between two movies.We can use the cosine similarity score since it is independent of magnitude and is relatively easy and fast to calculate (especially when used in conjunction with TF-IDF scores).

Mathematically, it is defined as follows:

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}^\intercal}{||\mathbf{x}|| \cdot ||\mathbf{y}||} = \frac{\sum_{i=1}^{n} \mathbf{x}_i \cdot \mathbf{y}^\intercal{}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{x}_i)^2} \sqrt{\sum_{i=1}^{n} (\mathbf{y}_i)^2}}$$

Research variable used for the system

- Movie genre
- Movie overviews and taglines,
- Movie cast
- Crew
- Keywords

Result Generated from Content Based System

Code snippet for tf-idf and cosine similarity functions

```
tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
tfidf_matrix = tf.fit_transform(smd['description'])
```

```
tfidf_matrix.shape
```

```
(9099, 268124)
```

```
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

```
cosine_sim[0]
```

```
array([1.        , 0.00680476, 0.        , ..., 0.        , 0.00344913,
       0.        ])
```

```
smd = smd.reset_index()
titles = smd['title']
indices = pd.Series(smd.index, index=smd['title'])
```

```
def get_recommendations(title):
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:31]
    movie_indices = [i[0] for i in sim_scores]
    return titles.iloc[movie_indices]
```

```
get_recommendations('The Dark Knight').head(10)
```

```
8031         The Dark Knight Rises
7648                     Inception
6218                 Batman Begins
2085                     Following
6623                  The Prestige
3381                       Memento
4145                      Insomnia
8613                  Interstellar
7659    Batman: Under the Red Hood
1134                Batman Returns
Name: title, dtype: object
```

```
get_recommendations('Mean Girls').head(10)
```

```
3319               Head Over Heels
7332    Ghosts of Girlfriends Past
6277             Just Like Heaven
1329              The House of Yes
6959     The Spiderwick Chronicles
7905          Mr. Popper's Penguins
4763                 Freaky Friday
8883                      The DUFF
6698          It's a Boy Girl Thing
3712          The Princess Diaries
Name: title, dtype: object
```

Result Generated from Content Based+Rating system

Code snippet for content and Simple recommender mixed

```
def improved_recommendations(title):
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:26]
    movie_indices = [i[0] for i in sim_scores]

    movies = smd.iloc[movie_indices][['title', 'vote_count', 'vote_average']]
    vote_counts = movies[movies['vote_count'].notnull()]['vote_count'].astype('int')
    vote_averages = movies[movies['vote_average'].notnull()]['vote_average'].astype('i
    C = vote_averages.mean()
    m = vote_counts.quantile(0.60)
    qualified = movies[(movies['vote_count'] >= m) & (movies['vote_count'].notnull()) &
    qualified['vote_count'] = qualified['vote_count'].astype('int')
    qualified['vote_average'] = qualified['vote_average'].astype('int')
    qualified['wr'] = qualified.apply(weighted_rating, axis=1)
    qualified = qualified.sort_values('wr', ascending=False).head(10)
    return qualified
```

```
improved_recommendations('The Dark Knight')
```

	title	vote_count	vote_average	wr
7648	Inception	14075	8	7.917588
8613	Interstellar	11187	8	7.897107
6623	The Prestige	4510	8	7.758148
3381	Memento	4168	8	7.740175
8031	The Dark Knight Rises	9263	7	6.921448
6218	Batman Begins	7511	7	6.904127
8872	Captain America: Civil War	7462	7	6.903532
7583	Kick-Ass	4747	7	6.852979
8419	Man of Steel	6462	6	5.952478
9024	Batman v Superman: Dawn of Justice	7189	5	5.013943

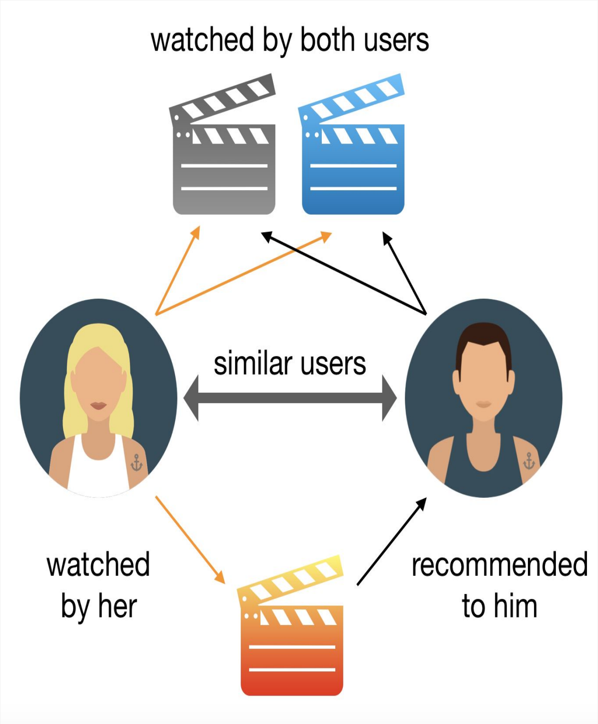
Disadvantage of Content based Recommender

Content based recommender suffers from some severe limitations.

It is only capable of suggesting movies which are close to a certain movie. That is, it is not capable of capturing tastes and providing recommendations across genres.

Also, it is not really personal as it doesn't capture the personal tastes and biases of a user. Anyone querying this system for recommendations based on a movie will receive the same recommendations for that movie, regardless of who s/he is.

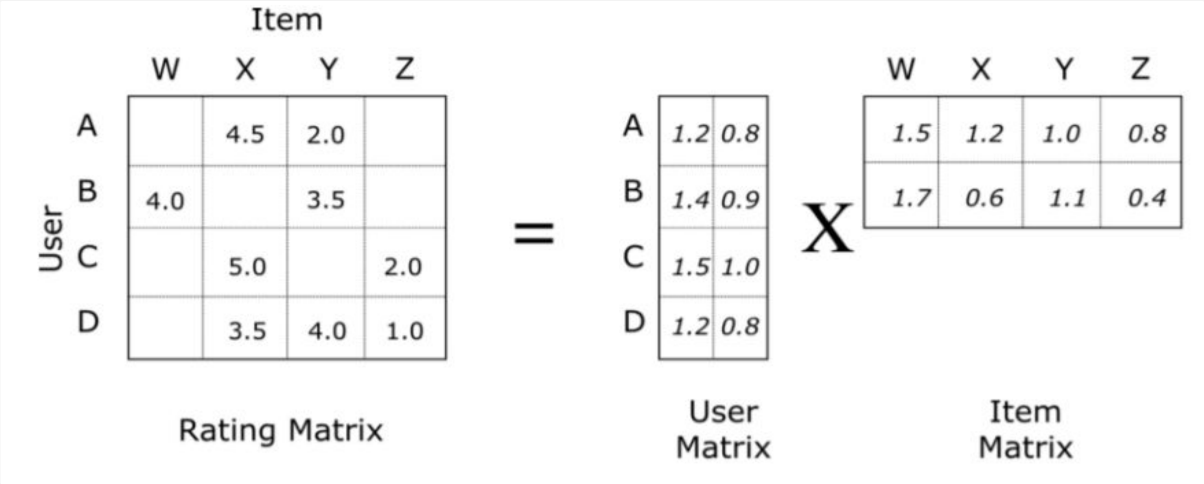
Collaborative Filtering based Recommender



Collaborative filtering is basically a user similarity recommendation technique where preferences of users can be used to correlate each other and give further recommendations to similar users which have not watched that particular movie or item.

Collaborative Filtering based Recommender

In the case of collaborative filtering, matrix factorization algorithms work by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices. One matrix can be seen as the user matrix where rows represent users and columns are latent factors. The other matrix is the item matrix where rows are latent factors and columns represent items.



Collaborative Filtering based Recommender

We have used RMSE as our metric in our matrix factorization technique.

From the image attached, it is clear that we get the RMSE as 0.8944, which is good enough to go thorugh with SVD(Support Vector Decomposition)

```
svd = SVD()
evaluate(svd, data, measures=['RMSE', 'MAE'])
```

```
Evaluating RMSE, MAE of algorithm SVD.


------------
Fold 1
RMSE: 0.8952
MAE:  0.6908
------------
Fold 2
RMSE: 0.8971
MAE:  0.6899
------------
Fold 3
RMSE: 0.8946
MAE:  0.6892
------------
Fold 4
RMSE: 0.8951
MAE:  0.6911
------------
Fold 5
RMSE: 0.8944
MAE:  0.6879
------------
```

Disadvantage of Collaborative filtering based Recommender

Collaborative filtering based recommender also suffers from some limitations :

Cold-Start: It doesn’t work with cold-start user or items, since the dot product will be all 0s. It can’t recommend anything.

Sparsity: Similarly, it doesnt work with sparse data, since the intersection between 2 users is 0, the dot product is also 0.

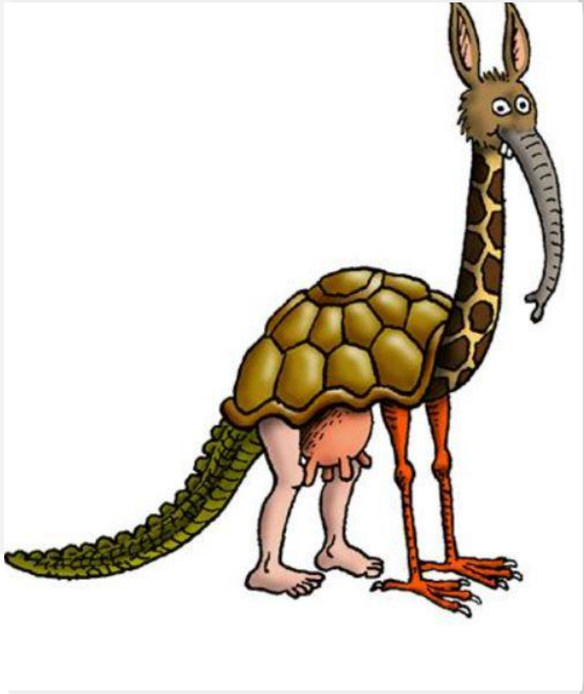
Scalability: We need to calculate the user similarity or item similarity matrix. This is a large matrix that doesn’t scale with large number of users.

Hybrid Model

In this section, we have tried to build a simple hybrid recommender that brings together techniques we have implemented in the content based and collaborative filter based engines.

This is how it will work:

1. **Input:** User ID and the Title of a Movie
2. **Output:** Similar movies sorted on the basis of expected ratings by that particular user.engines. This is how it will work:



Results Generated From Hybrid Recommender

Code snippet for content + collaborative based models

```python
def hybrid(userId, title):
    idx = indices[title]
    tmdbId = id_map.loc[title]['id']
    #print(idx)
    movie_id = id_map.loc[title]['movieId']

    sim_scores = list(enumerate(cosine_sim[int(idx)]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:26]
    movie_indices = [i[0] for i in sim_scores]

    movies = smd.iloc[movie_indices][['title', 'vote_count', 'vote_average', 'id']]
    movies['est'] = movies['id'].apply(lambda x: svd.predict(userId, indices_map.loc[x]['movieId']).est)
    movies = movies.sort_values('est', ascending=False)
    return movies.head(10)
```

```python
hybrid(1, 'Avatar')
```

	title	vote_count	vote_average	id	est
8401	Star Trek Into Darkness	4479.0	7.4	54138	3.159255
974	Aliens	3282.0	7.7	679	3.147986
899	Platoon	1236.0	7.5	792	3.126822
522	Terminator 2: Judgment Day	4274.0	7.7	280	3.104779
987	Alien	4564.0	7.9	348	3.100046
1011	The Terminator	4208.0	7.4	218	3.066871
922	The Abyss	822.0	7.1	2756	3.027235
5301	Cypher	196.0	6.7	10133	2.965623
4987	Battle Royale	992.0	7.3	3176	2.951908
2014	Fantastic Planet	140.0	7.6	16306	2.798646

```python
hybrid(500, 'Avatar')
```

	title	vote_count	vote_average	id	est
4987	Battle Royale	992.0	7.3	3176	3.444521
899	Platoon	1236.0	7.5	792	3.389479
974	Aliens	3282.0	7.7	679	3.320112
5301	Cypher	196.0	6.7	10133	3.312177
1011	The Terminator	4208.0	7.4	218	3.223638
7065	Meet Dave	381.0	5.1	11260	3.194728
8401	Star Trek Into Darkness	4479.0	7.4	54138	3.194711
6316	Star Wreck: In the Pirkinning	27.0	6.6	15493	3.171771
987	Alien	4564.0	7.9	348	3.151968
922	The Abyss	822.0	7.1	2756	3.098245

References

1. Geetha, G., Safa, M., Fankknncy, C., & Saranya, D. (2018, April). A hybrid approach using collaborative filtering and content based filtering for recommender system. In Journal of Physics: Conference Series (Vol. 1000, No. 1, p. 012101).

2. Christakou, C., Vrettos, S.Stafylopatis, A. (2007). A hybrid movie recommender system based on neural networks. International Journal on Artificial Intelligence Tools, 16(05), 771-792.

3. Machine Learning by Tom M. Mitchell

Thank you!


NETFLIX
GODLESS
NETFLIX
JIM&ANDY
NETFLIX
NETFLIX
THE BABYSITTER
NETFLIX
BIG MOUTH