Collaborative filtering using implicit feedback

Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
from scipy.sparse import csr_matrix

import implicit

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Loading the Dataset

We are going to load the dataset "MovieLens" which has two of the following files

- ratings.csv
- movies.csv

In [2]:
```
ratings = pd.read_csv('/Users/bharath/Documents/ASm/ratings.csv')
movies = pd.read_csv('/Users/bharath/Documents/ASm/movies.csv')
```

In [3]:
```
ratings.head()
```

Out[3]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

Transforming the data

We are going to create a Function that generates a sparse matrix from ratings dataframe.

In [4]:
```
def create_X(df):
    N = df['userId'].nunique()
    M = df['movieId'].nunique()

    user_mapper = dict(zip(np.unique(df["userId"]), list(range(N))))
    movie_mapper = dict(zip(np.unique(df["movieId"]), list(range(M))))

    user_inv_mapper = dict(zip(list(range(N)), np.unique(df["userId"])))
    movie_inv_mapper = dict(zip(list(range(M)), np.unique(df["movieId"])))

    user_index = [user_mapper[i] for i in df['userId']]
    movie_index = [movie_mapper[i] for i in df['movieId']]

    X = csr_matrix((df["rating"], (movie_index, user_index)), shape=(M, N))

    return X, user_mapper, movie_mapper, user_inv_mapper, movie_inv_mapper
```

In [5]:
```
X, user_mapper, movie_mapper, user_inv_mapper, movie_inv_mapper = create_X(r
```

Creating Movie Title Mappers

In [6]:
```
!pip install thefuzz
from thefuzz import fuzz
from thefuzz import process

def movie_finder(title):
    all_titles = movies['title'].tolist()
    closest_match = process.extractOne(title,all_titles)
    return closest_match[0]

movie_title_mapper = dict(zip(movies['title'], movies['movieId']))
movie_title_inv_mapper = dict(zip(movies['movieId'], movies['title']))

def get_movie_index(title):
    fuzzy_title = movie_finder(title)
    movie_id = movie_title_mapper[fuzzy_title]
    movie_idx = movie_mapper[movie_id]
    return movie_idx

def get_movie_title(movie_idx):
    movie_id = movie_inv_mapper[movie_idx]
    title = movie_title_inv_mapper[movie_id]
    return title
```

```
Requirement already satisfied: thefuzz in /Users/bharath/opt/anaconda3/lib/python3.9/site-packages (0.19.0)
```

```
/Users/bharath/opt/anaconda3/lib/python3.9/site-packages/thefuzz/fuzz.py:11: UserWarning: Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove this warning
  warnings.warn('Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove this warning')
```

It's time to test it out! Let's get the movie index of Legally Blonde.

In [7]:
```
get_movie_index('Legally Blonde')
```

Out[7]:
```
3282
```

Let's pass this index value into get_movie_title(). We're expecting Legally Blonde to get returned.

In [8]:
```
get_movie_title(3282)
```

Out[8]:
```
'Legally Blonde (2001)'
```

Building Our Implicit Feedback Recommender Model

In [9]:
```
model = implicit.als.AlternatingLeastSquares(factors=50)
```

```
WARNING:root:Intel MKL BLAS detected. Its highly recommend to set the environment variable 'export MKL_NUM_THREADS=1' to disable its internal multithreading
```

fitting our model with our user-item matrix.

In [10]:
```
model.fit(X)
```

```
  0%|          | 0/15 [00:00<?, ?it/s]
```

Now, let's test out the model's recommendations. We can use the model's similar_items() method which returns the most relevant movies of a given movie. We can use our helpful get_movie_index() function to get the movie index of the movie that we're interested in.

In [11]:
```
movie_of_interest = 'forrest gump'

movie_index = get_movie_index(movie_of_interest)
related = model.similar_items(movie_index)
related
```

Out[11]:
```
[(314, 1.0),
 (277, 0.92106205),
 (510, 0.85805476),
 (257, 0.85661566),
 (461, 0.76611966),
 (97, 0.7537822),
 (418, 0.7102524),
 (43, 0.6950457),
 (123, 0.6937208),
 (46, 0.6629923)]
```

The output of similar_items() is not user-friendly. We'll need to use our get_movie_title() function to interpret what our results are.

In [12]:
```
print(f"Because you watched {movie_finder(movie_of_interest)}...")
for r in related:
    recommended_title = get_movie_title(r[0])
    if recommended_title != movie_finder(movie_of_interest):
        print(recommended_title)
```

```
Because you watched Forrest Gump (1994)...
Shawshank Redemption, The (1994)
Silence of the Lambs, The (1991)
Pulp Fiction (1994)
Schindler's List (1993)
Braveheart (1995)
Jurassic Park (1993)
Seven (a.k.a. Se7en) (1995)
Apollo 13 (1995)
Usual Suspects, The (1995)
```

Generating User-Item Recommendations

In [13]:
```
user_id = 95
```

In [14]:
```
user_ratings = ratings[ratings['userId']==user_id].merge(movies[['movieId',
user_ratings = user_ratings.sort_values('rating', ascending=False)
print(f"Number of movies rated by user {user_id}: {user_ratings['movieId'].n
```

```
Number of movies rated by user 95: 168
```

User 95 watched 168 movies. Their highest rated movies are below:

In [15]:
```
user_ratings = ratings[ratings['userId']==user_id].merge(movies[['movieId',
user_ratings = user_ratings.sort_values('rating', ascending=False)
top_5 = user_ratings.head()
top_5
```

Out[15]:

	userId	movieId	rating	timestamp	title
24	95	1089	5.0	1048382826	Reservoir Dogs (1992)
34	95	1221	5.0	1043340018	Godfather: Part II, The (1974)
83	95	3019	5.0	1043340112	Drugstore Cowboy (1989)
26	95	1175	5.0	1105400882	Delicatessen (1991)
27	95	1196	5.0	1043340018	Star Wars: Episode V - The Empire Strikes Back...

Their lowest rated movies:

In [16]:
```
bottom_5 = user_ratings[user_ratings['rating']<3].tail()
bottom_5
```

Out[16]:

	userId	movieId	rating	timestamp	title
93	95	3690	2.0	1043339908	Porky's Revenge (1985)
122	95	5283	2.0	1043339957	National Lampoon's Van Wilder (2002)
100	95	4015	2.0	1043339957	Dude, Where's My Car? (2000)
164	95	7373	1.0	1105401093	Hellboy (2004)
109	95	4732	1.0	1043339283	Bubble Boy (2001)

We'll use the recommend() method, which takes in the user index of interest and transposed user-item matrix.

In [17]:
```
X_t = X.T.tocsr()

user_idx = user_mapper[user_id]
recommendations = model.recommend(user_idx, X_t)
recommendations
```

Out[17]:
```
[(855, 1.2092698),
 (898, 0.9775134),
 (1644, 0.9395924),
 (15, 0.89407647),
 (5, 0.80936944),
 (2258, 0.8077935),
 (3633, 0.80654395),
 (1043, 0.80218315),
 (1210, 0.784987),
 (46, 0.76805365)]
```

We can't interpret the results as is since movies are represented by their index. We'll have to loop over the list of recommendations and get the movie title for each movie index.

In [18]:
```
for r in recommendations:
    recommended_title = get_movie_title(r[0])
    print(recommended_title)
```

```
Abyss, The (1989)
Princess Bride, The (1987)
Untouchables, The (1987)
Casino (1995)
Heat (1995)
Princess Mononoke (Mononoke-hime) (1997)
Lord of the Rings: The Fellowship of the Ring, The (2001)
Star Trek: First Contact (1996)
Hunt for Red October, The (1990)
Usual Suspects, The (1995)
```

User 95's recommendations consist of action, crime, and thrillers. None of their recommendations are comedies.