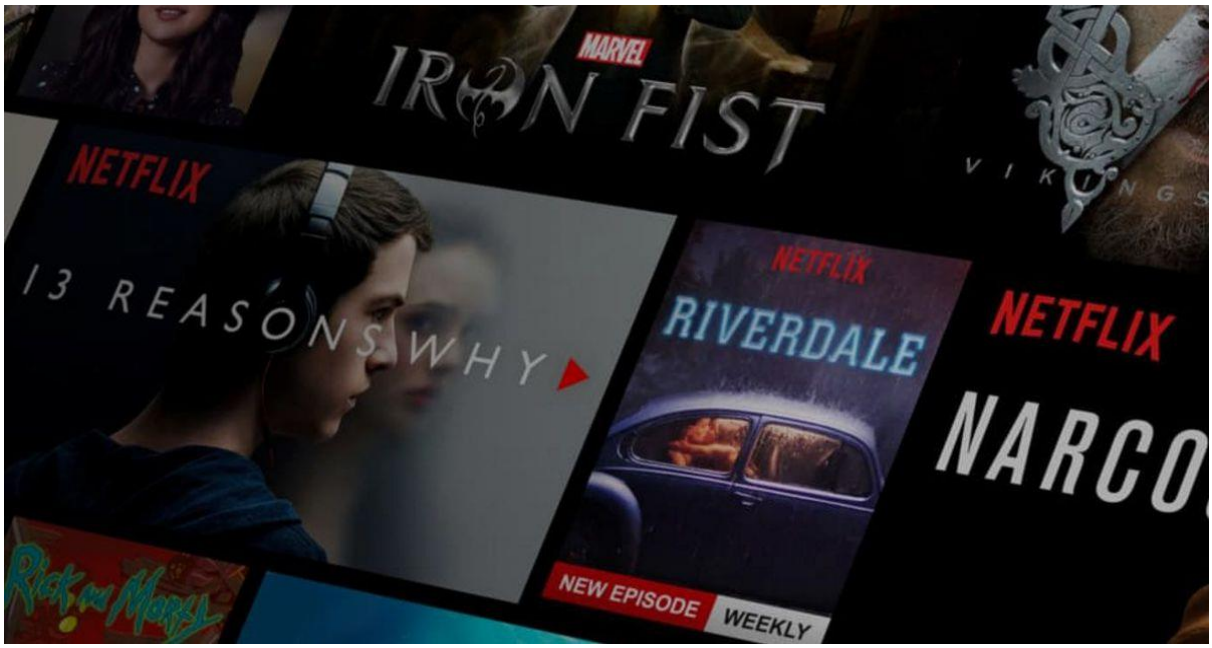


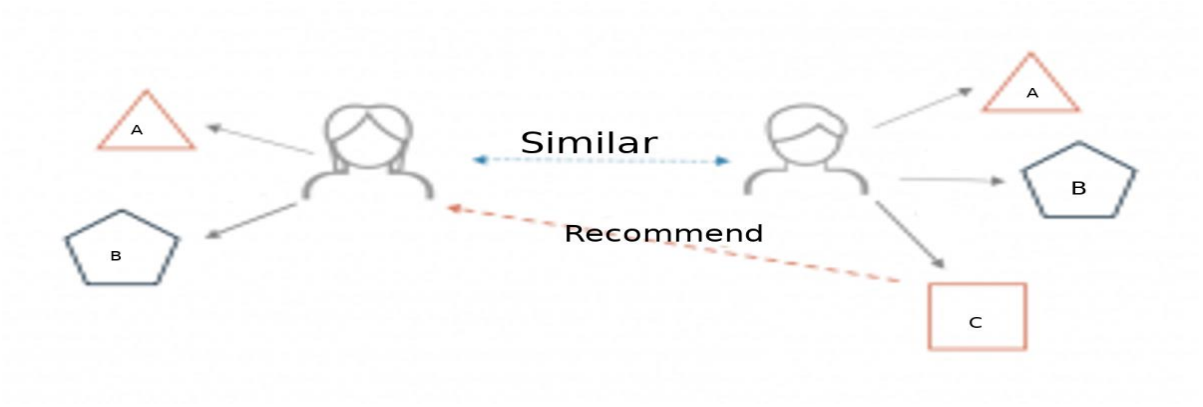
MOVIE RECOMMENDER SYSTEM

CREATOR: KISHAN MISHRA

WHAT IS RECOMMENDER SYSTEM ?

A recommender system is a simple algorithm whose aim is to provide the most relevant information to a user by discovering patterns in a dataset. The algorithm rates the items and shows the user the items that they would rate highly. An example of recommendation in action is when you visit Amazon and you notice that some items are being recommended to you or when Netflix recommends certain movies to you. They are also used by Music streaming applications such as Spotify and Deezer to recommend music that you might like.

Below is a very simple illustration of how recommender systems work in the context of an e-commerce site.



Two users buy the same items A and B from an e-commerce store. When this happens the similarity index of these two users is computed. Depending on the score the system can recommend item C to the other user because it detects that those two users are similar in terms of the items they purchase.

DIFFERENT TYPES OF RECOMMENDATION ENGINES

The most common types of recommendation systems are **content-based** and **collaborative filtering** recommender systems. In collaborative filtering, the behaviour of a group of users is used to make recommendations to other users. The recommendation is based on the preference of other users. A simple example would be recommending a movie to a user based on the fact that their friend liked the movie.

There are two types of collaborative models **Memory-based methods** and **Model-based methods**.

The advantage of memory-based techniques is that they are simple to implement and the resulting recommendations are often easy to explain. They are divided into two:

- **User-based collaborative filtering**: In this model, products are recommended to a user based on the fact that the products have been liked by users similar to the user. For example, if Derrick and Dennis like the same movies and a new movie come out that Derick like, then we can recommend that movie to Dennis because Derrick and Dennis seem to like the same movies.
- **Item-based collaborative filtering**: These systems identify similar items based on users' previous ratings. For example, if users A, B, and C gave a 5-star rating to books X and Y then when a user D buys book Y they also get a recommendation to purchase book X because the system identifies book X and Y as similar based on the ratings of users A, B, and C.

Model-based methods are based on Matrix Factorization and are better at dealing with sparsity. They are developed using data mining, machine learning algorithms to predict users' rating of unrated items. In this approach techniques such as dimensionality reduction are used to improve accuracy. Examples of such model-based methods include Decision trees, Rule-based Model, Bayesian Model, and latent factor models.

Content-based systems use metadata such as genre, producer, actor, musician to recommend items say movies or music. Such a recommendation would be for instance recommending Infinity War that featured Vin Diesel because someone watched and liked The Fate of the Furious. Similarly, you can get music recommendations from certain artists because you liked their music. Content-based systems are based on the idea that if you liked a certain item you are most likely to like something that is similar to it.

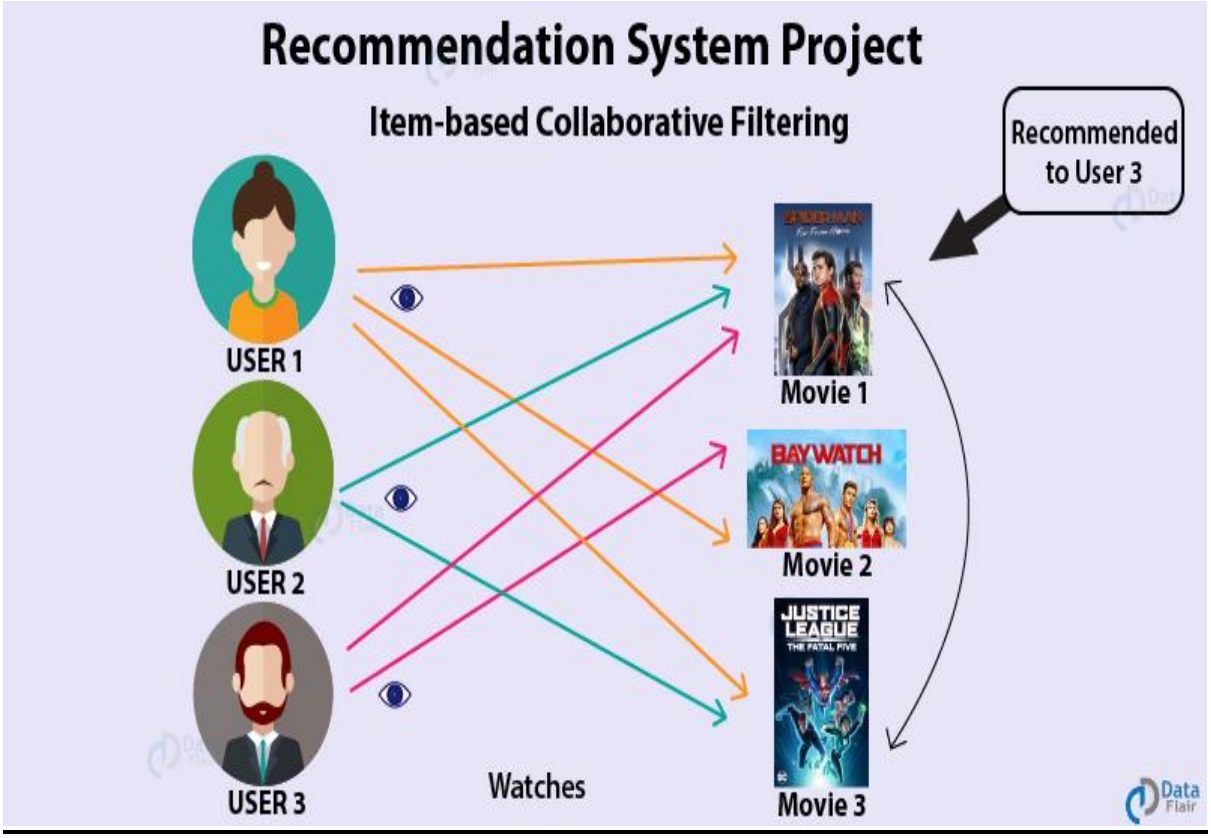
Machine Learning Concepts used to prepare this Movie Recommender System as follows:

1.**Collaborative Filtering**

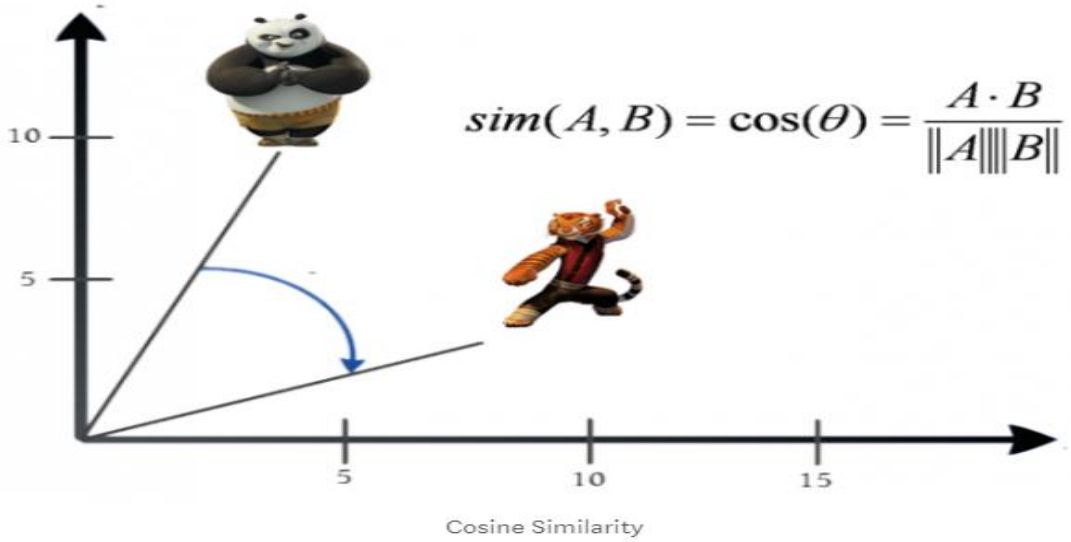
- Item-based collaborative filtering

2. **Cosine Similarity**

Collaborative filtering (explained in above pages):



Cosine Similarity : Cosine similarity is a metric used to measure how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance (due to the size of the document), chances are they may still be oriented closer together. The smaller the angle, higher the cosine similarity.



Cosine Similarity

WALKTHROUGH OF BUILDING A RECOMMENDER SYSTEM

We are going to use the movie lens datsets to build a simple item similarity-based recommender system. The first thing we need to do is to import the libraries we needed.

- import pandas as pd *#It is used to perform data manipulation and analysis.*
- import numpy as np *#It is used for working with arrays, linear algebra and matrices.*
- import matplotlib.pyplot as plt *#It provides an object-oriented API for embedding plots into applications.*
- from sklearn.metrics import pairwise_distances *# It provides a safe way to take a distance matrix as input, while preserving compatibility with many other algorithms that take a vector array.*
- from scipy.spatial.distance import cosine, correlation *#Compute the Cosine distance between 1-D arrays*

Next, we select the column's name and load in the data set using pandas **read_csv()** utility. The dataset is tab separated so we pass in **'|'** to the **sep** parameter. We then pass in the column names using the names parameter.If a column or index cannot be represented as an array of datetimes, say because of an unparsable value or a mixture of timezones, So we use **parse_dates=True** such that the column or index will be returned unaltered as an object data type.

- u_cols = ['user_id', 'age', 'sex', 'occupation', 'zip_code']

```
users = pd.read_csv('u.user', sep='|', names=u_cols,encoding='latin-1', parse_dates=True)
```

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

```
ratings = pd.read_csv('u.data', sep='\t', names=r_cols,encoding='latin-1')
```

- m_cols = ['movie_id', 'title', 'release_date', 'video_release_date', 'imdb_url']

```
movies = pd.read_csv('u.item', sep='|', names=m_cols, usecols=range(5),encoding='latin-1')
```

Since the **movie_id** columns are the same in movies and ratings datasets ,we can merge these datasets .

- movie_ratings = pd.merge(movies, ratings)

Similarly merging the new movie_ratings and user datasets with **user_id** in common .

- df = pd.merge(movie_ratings, users)

Now let's check the head of the data to see the data we are dealing with.

- df.head(2)

	movie_id	title	release_date	video_release_date	imdb_url	user_id	rating	unix_timestamp	age	sex	occupation	zip_code
0	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	308	4	887736532	60	M	retired	95076
1	4	Get Shorty (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Get%20Shorty%...	308	5	887737890	60	M	retired	95076

Similarly we use head to check each datasets content.

- ratings.head(1)
- movies.head(1)
- users.head(1)

Now we can remove the columns that really not needed to obtain the final output .So we use the drop method to remove specified rows or columns**.Axis** is used to determine whether to drop labels from the index (0 or 'rows') or columns (1 or 'columns') and **inplace = True** means the data is modified in place.

- df.drop(df.columns[[3,4,7]], axis=1, inplace=True)
- ratings.drop("unix_timestamp", inplace = True, axis = 1)
- movies.drop(movies.columns[[3,4]], inplace = True, axis = 1)
- df.info()

Using the **describe** or **info** commands we can get a brief description of our dataset.

Let's now create a data frame with the average rating for each movie and the number of ratings. We are going to use these ratings to calculate the similarity between the movies later. We will use the **Cosine Similarity** from Sklearn, as the metric to compute the similarity between two items. In order to create this data frame we use pandas **groupby** functionality. We group the dataset by the title column and compute its **size** to count the number of elements along a given axis and **mean** to obtain the average rating for each movie.

- movie_stats = df.groupby('title').agg({'rating': [np.size, np.mean]})
- movie_stats.head()

Now ,Setting a threshold of atleast 50 ratings for better analysis.

- min_50 = movie_stats['rating']['size'] >= 50

- movie_stats[min_50].sort_values([('rating', 'mean')], ascending=False).head()

Let's now plot a Histogram using pandas plotting functionality to visualize the distribution of user's ratings.

- ratings.rating.plot.hist(bins=50)
- plt.title("Distribution of Users' Ratings")
- plt.ylabel('Number of Ratings')
- plt.xlabel('Rating (Out of 5)');
-

Similarly use the same method to visualize the distribution of user's age.(Optional)

From the diagram we can see that there is a positive relationship between the average rating of a movie and the number of ratings. The graph indicates that the more the ratings a movie gets the higher the average rating it gets. This is important to note especially when choosing the threshold for the number of ratings per movie.

Let's now move on swiftly and create a simple user-based recommender system. In order to do this, we need to convert our dataset into a matrix with the user_id as the columns, the `movie_id` as the index and the ratings as the values. By doing this we shall get a data frame with the columns as the userid and the rows as the movie_ids. Each column represents all the ratings of a user to all movies. We shall use this matrix to compute the similarities between the ratings of a single movie and the rest of the movies in the matrix. We use pandas `pivot_table` utility to create the matrix.

- ratings_mat=ratings.pivot_table(index=['movie_id'],columns=['user_id'],values='rating').reset_index(drop=True)
- ratings_mat.fillna(0, inplace = True)
- ratings_mat.head()

user_id	1	2	3	4	5	6	7	8	9	10	...	934	935	936	937	938	939	940	941	942	943
0	5.0	4.0	0.0	0.0	4.0	4.0	0.0	0.0	0.0	4.0	...	2.0	3.0	4.0	0.0	4.0	0.0	0.0	5.0	0.0	0.0
1	3.0	0.0	0.0	0.0	3.0	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	5.0
2	4.0	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
3	3.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	4.0	...	5.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0
4	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
5	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	...	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

movie_id

Now ,let's assume that a user has watched X and Y movies. We would like to recommend movies to this user based on this watching history. The goal is to look for movies that are similar to X and Y which we shall recommend to this user. We can achieve this by computing the cosine similarity between

these two movies' ratings and the ratings of the rest of the movies in the dataset.We will use the **sklearn.metrics.pairwise_distances** from Sklearn,to compute the similarity between two movies.Cosine similarity is a metric used to measure how similar two items are. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The output value ranges from **0–1**.0 means no similarity, where as 1 means that both the items are 100% similar.

- movie_similarity = 1 - pairwise_distances(ratings_mat.values, metric="cosine")
- np.fill_diagonal(movie_similarity, 0) *#Filling diagonals with 0s for future use when sorting is done*
- ratings_mat = pd.DataFrame(movie_similarity)
- ratings_mat.head()

Now, here comes the last part of the project, which is to print the names of the movies similar to the one we have given as input to the system through the **user_inp** variable, by using some simple python code. It compares the cosine similarities between the **user_inp** ratings and the rest of the movie's ratings. Clearly, by changing the threshold for the number of reviews, we get different results and at last display all the movies to be recommended sorted w.r.t to its similarities to **user_inp** in descending order. (most similar to least)

```
try:
    user_inp=input('Enter the reference movie title based on which recommendations are to be made: ')
    inp=movies[movies['title']==user_inp].index.tolist()
    inp=inp[0]
    movies['similarity'] = ratings_mat.iloc[inp]
    movies.columns = ['movie_id', 'title', 'release_date','similarity']
    movies.head(2)
except:
    print("Sorry, the movie is not in the database!")
```

- print("Recommended movies based on your choice of ",user_inp ,": \n", movies.sort_values(["similarity"], ascending = False)[1:10])

Output:

```
Enter the reference movie title based on which recommendations are to b
e made: True Lies (1994)
Recommended movies based on your choice of  True Lies (1994) :
     movie_id                                  title release_date   similarity
567       568                           Speed (1994)  01-Jan-1994     0.719504
160       161                        Top Gun (1986)  01-Jan-1986     0.710609
81         82                    Jurassic Park (1993)  01-Jan-1993     0.702316
225       226                      Die Hard 2 (1990)  01-Jan-1990     0.697793
95         96          Terminator 2: Judgment Day (1991)  01-Jan-1991     0.693396
565       566            Clear and Present Danger (1994)  01-Jan-1994     0.685770
209       210  Indiana Jones and the Last Crusade (1989)  01-Jan-1989     0.680514
194       195                   Terminator, The (1984)  01-Jan-1984     0.674579
402       403                           Batman (1989)  01-Jan-1989     0.664393
```

DATASETS REQUIRED

For this project we used Movielens 100k dataset . This dataset was put toget her by the Group lens research group at the University of Minnesota. It cont ains 1, 10, and 20 million ratings. Movie lens also has a website where you c an sign up, contribute reviews and get movie recommendations.
Link : https://grouplens.org/datasets/movielens/100k/

RESULTS AND DISCUSSION

Movie recommender system plays a significant role in identifying a set of movies for users based on user interest. Although many more recommendation systems are available for users, these systems have the limitation of not recommending the movie efficiently to the existing users.

From the results we found that the system can be improved by building a Memory-Based Collaborative Filtering based system. In this case, we'd divide the data into a training set and a test set. We'd then use techniques such as cosine similarity to compute the similarity between the movies. An alternative is to build a Model-based Collaborative Filtering system. This is based on matrix factorization. Matrix factorization is good at dealing with scalability and sparsity than the former. You can then evaluate your model using techniques such as Root Mean Squared Error (RMSE).

REFERENCES

1. https://www.kaggle.com/ashishpatel26/movie-recommendation-of-movie-lens-data-set – Movie recommendation using Movielens 100k dataset.
2. https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26 -Introduction to recommender system.
3. https://www.machinelearningplus.com/nlp/cosine-similarity/#:~:text=Cosine%20similarity%20is%20a%20metric,in%20a%20multi%2Ddimensional%20space.&text=The%20smaller%20the%20angle%2C%20higher%20the%20cosine%20similarity. – Concepts on Cosine Similarity
4. https://scikit-learn.org/ -Modules and Libraries
5. https://analyticsindiamag.com/how-to-build-your-first-recommender-system-using-python-movielens-dataset/