**MOVIE RECOMMENDATION SYSTEM**

**SUBMITTED BY :**

**NAME : SANSKAR GARG**

**ROLL NO. : 21052449**

**BRANCH : CSE**

**INTRODUCTION**

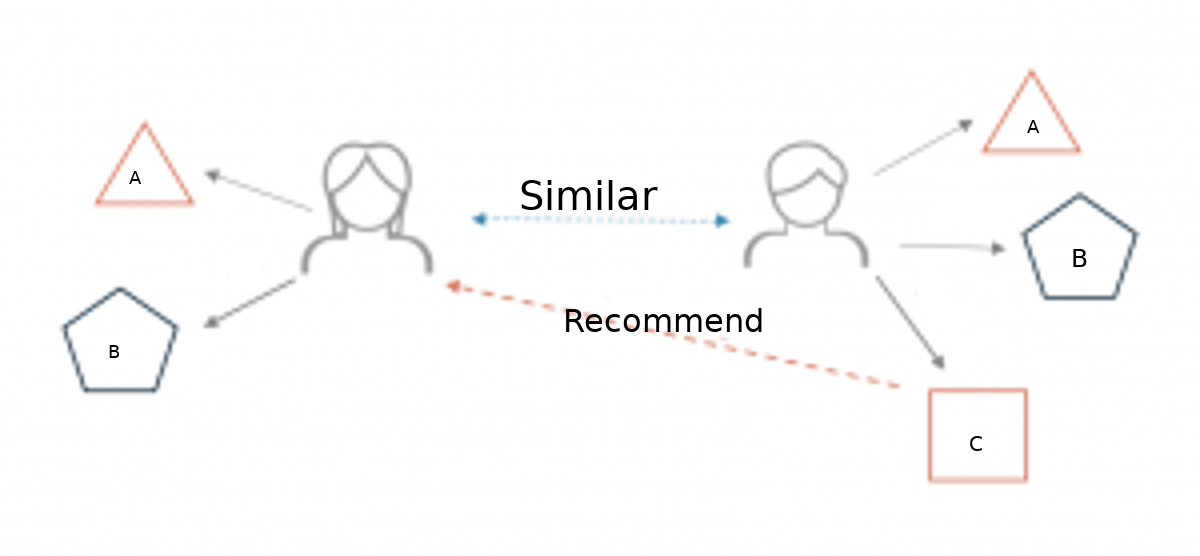
In this project recommendation algorithm is used to recommend the movies based on past data of user like it is in Netflix, Prime Video, etc.

Technology used : Core Python, Numpy, Pandas, Matplotlib, Machine Learning supervised (KNN) or usupervised (K means) algorithm.

**ABOUT THE PROJECT**

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 behavior 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.

In this project **Content-based system** type ofrecommendation technology has been used where the movies are recommended based on past data of user.

**ALGORITHM OF THE PROJECT**

The approach to build the movie recommendation engine consists of the following steps :

1. Perform Exploratory Data Analysis (EDA) on the data
2. Build the recommendation system
3. Get recommendations

#### Step 1: Perform Exploratory Data Analysis (EDA) on the data :

The dataset contains two CSV files, credits, and movies. The credits file contains all the metadata information about the movie and the movie file contains the information like name and id of the movie, budget, languages in the movie that has been released, etc.

Now the movie dataset is loaded using Pandas.

For viewing the dataframes we now write movies\_df.head() and credits\_df.head()

We only need the id, title, cast, and crew columns of the credits dataframe. Now merge the dataframes into one on the column ‘id’.

credits\_df.columns = ['id','title','cast','crew']

movies\_df = movies\_df.merge(credits\_df, on="id"

And now if write movies\_df.head() new dataframe is created where dataframes of boths credits and movies are merged in one dataframe.

#### Step 2: Build the Movie Recommender System :

The accuracy of predictions made by the recommendation system can be personalized using the “plot/description” of the movie.

But the quality of suggestions can be further improved using the metadata of movie. Let’s say the query to our movie recommendation engine is “The Dark Knight Rises”. Then the predictions should also include movies directed by the director of the film. It should also include movies with the cast of the given query movie.

For that, we utilize the following features to personalize the recommendation: cast, crew, keywords, genres.

The movie data is present in the form of lists containing strings, we need to convert the data into a safe and usable structure. Now apply the literal\_eval() function to the features.

Now we write some functions to extract information like director from the above features.

The get\_director() function extracts the name of the director of the movie.

The get\_list() returns the top 3 elements or the entire list whichever is more.

Now we apply both the functions get\_director() and get\_list() to our dataset.

movies\_df["director"] = movies\_df["crew"].apply(get\_director)

features = ["cast", "keywords", "genres"]

for feature in features:

movies\_df[feature] = movies\_df[feature].apply(get\_list)

In the above code, we passed the “crew” information to the get\_director() function, extracted the name, and created a new column “director”.

For the features cast, keyword and genres we extracted the top information by applying the get\_list() function

Now we see how the data looks like after the above transformations using

movies\_df[['title', 'cast', 'director', 'keywords', 'genres']].head()

The next step would be to convert the above feature instances into lowercase and remove all the spaces between them.

def clean\_data(row):

if isinstance(row, list):

return [str.lower(i.replace(" ", "")) for i in row]

else:

if isinstance(row, str):

return str.lower(row.replace(" ", ""))

else:

return ""

features = ['cast', 'keywords', 'director', 'genres']

for feature in features:

movies\_df[feature] = movies\_df[feature].apply(clean\_data)

Now, let’s create a “soup” containing all of the metadata information extracted to input into the vectorizer.

def create\_soup(features):

return ' '.join(features['keywords']) + ' ' + ' '.join(features['cast']) + ' ' + features['director'] + ' ' + ' '.join(features['genres'])

movies\_df["soup"] = movies\_df.apply(create\_soup, axis=1)

print(movies\_df["soup"].head())

Our movie recommendation engine works by suggesting movies to the user based on the metadata information. The similarity between the movies is calculated and then used to make recommendations. For that, our text data should be preprocessed and converted into a vectorizer using the CountVectorizer. As the name suggests, CountVectorizer counts the frequency of each word and outputs a 2D vector containing frequencies.

We don’t take into account the words like a, an, the (these are called “stopwords”) because these words are usually present in higher amounts in the text and don’t make any sense.

There exist several similarity score functions such as cosine similarity, Pearson correlation coefficient, etc. Here, we use the cosine similarity score as this is just the dot product of the vector output by the CountVectorizer.

We also reset the indices of our dataframe.

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

count\_vectorizer = CountVectorizer(stop\_words="english")

count\_matrix = count\_vectorizer.fit\_transform(movies\_df["soup"])

print(count\_matrix.shape)

cosine\_sim2 = cosine\_similarity(count\_matrix, count\_matrix)

print(cosine\_sim2.shape)

movies\_df = movies\_df.reset\_index()

indices = pd.Series(movies\_df.index, index=movies\_df['title'])

Create a reverse mapping of movie titles to indices. By this, we can easily find the title of the movie based on the index.

indices = pd.Series(movies\_df.index, index=movies\_df["title"]).drop\_duplicates()

print(indices.head())

#### Step 3: Get recommendations for the movies :

The get\_recommendations() function takes the title of the movie and the similarity function as input. It follows the below steps to make recommendations.

* Get the index of the movie using the title.
* Get the list of similarity scores of the movies concerning all the movies.
* Enumerate them (create tuples) with the first element being the index and the second element is the cosine similarity score.
* Sort the list of tuples in descending order based on the similarity score.
* Get the list of the indices of the top 10 movies from the above sorted list. Exclude the first element because it is the title itself.
* Map those indices to their respective titles and return the movies list.

Create a function that takes in the movie title and the cosine similarity score as input and outputs the top 10 movies similar to it.

def get\_recommendations(title, cosine\_sim=cosine\_sim):

idx = indices[title]

similarity\_scores = list(enumerate(cosine\_sim[idx]))

similarity\_scores= sorted(similarity\_scores, key=lambda x: x[1], reverse=True)

similarity\_scores= sim\_scores[1:11]

# (a, b) where a is id of movie, b is similarity\_scores

movies\_indices = [ind[0] for ind in similarity\_scores]

movies = movies\_df["title"].iloc[movies\_indices]

return movies

print("################ Content Based System #############")

print("Recommendations for The Dark Knight Rises")

print(get\_recommendations("The Dark Knight Rises", cosine\_sim2))

print()

print("Recommendations for Avengers")

print(get\_recommendations("The Avengers", cosine\_sim2))

**RESULT**

In this machine learning project, we build movie recommendation systems. We built a content-based recommendation engine that makes recommendations given the title of the movie as input.