Name: Deepthi Chenthil Kumar Nadar

Movie Recommendation System

1. Introduction

A Movie Recommendation System is an intelligent application designed to suggest movies to users based on their preferences. With the rapid growth of digital content, recommendation systems play a crucial role in helping users discover relevant movies efficiently.

This project implements a **Content-Based Movie Recommendation System** using **Python** and **Machine Learning techniques**. The system recommends movies by analyzing movie metadata such as genres, keywords, cast, director, and tagline. It uses **TF-IDF Vectorization** and **Cosine Similarity** to find movies that are most similar to a user's favorite movie.

2. Objective of the Project

The main objectives of this project are:

- To build a content-based movie recommendation system
- To recommend movies similar to the user's favorite movie
- To apply text vectorization techniques on movie metadata
- To measure similarity between movies using cosine similarity
- To provide accurate and meaningful movie suggestions

3. Technologies and Libraries Used

Programming Language

- Python

Libraries

- **Pandas** – Data loading and manipulation
- **NumPy** – Numerical operations

- **Difflib** – Finding close matches of movie names
- **Scikit-learn** – Machine learning utilities
    - TfidfVectorizer
    - Cosine Similarity

4. Dataset Description

The dataset used in this project is a movie dataset containing **4803 movies** and **24 columns**. Each row represents a movie with various attributes.

Important Columns in the Dataset:

- `title` – Movie title
- `genres` – Movie genres
- `keywords` – Keywords related to the movie
- `tagline` – Movie tagline
- `cast` – Main actors
- `director` – Movie director

5. Data Collection and Preprocessing

5.1 Loading the Dataset

The dataset is loaded using Pandas from a CSV file.

```
#loading the dataset
df=pd.read_csv('/content/movies.csv')
```

5.2 Data Inspection

- First 5 rows are displayed using `df.head()`

```
1]
Os
#printing 5 rows of the data
df.head()
```

	index	budget	genres	homepage	id	keywords	original_la
0	0	237000000	Action Adventure Fantasy Science Fiction	http://www.avatarmovie.com/	19995	culture clash future space war space colony so...	

original_language	original_title	overview	popularity	...	runtime	spoken_languages	status
en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437577	...	162.0	[{"iso_639_1": "en", "name": "English"}, {"iso...	Released

status	tagline	title	vote_average	vote_count	cast	crew	director
Released	Enter the World of Pandora.	Avatar	7.2	11800	Sam Worthington Zoe Saldana Sigourney Weaver S...	[{'name': 'Stephen E. Rivkin', 'gender': 0, 'd...	James Cameron

- **Dataset shape is checked using df.shape**
- **Total records: 4803 rows and 24 columns**

```
#number of rows and columns
df.shape
```

```
(4803, 24)
```

5.3 Feature Selection

Only relevant features are selected for building recommendations:

- Genres
- Keywords
- Tagline
- Cast
- Director

```
#Feature selection (selecting the relevant features for recommendations)
selected_features = ['genres','keywords', 'tagline', 'cast', 'director']
print(selected_features)
```

```
['genres', 'keywords', 'tagline', 'cast', 'director']
```

These features contribute significantly to defining movie content.

5.4 Handling Missing Values

Missing values in selected features are replaced with empty strings to avoid errors during text processing.

```
#replacing the null values with the null strong

for feature in selected_features:
    df[feature] = df[feature].fillna('')
```

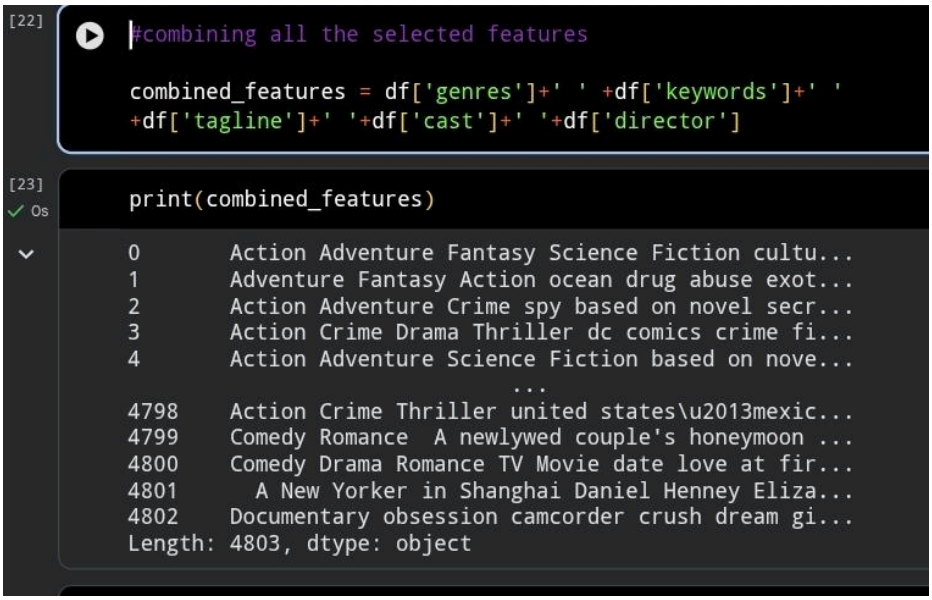
---

6. Feature Engineering

6.1 Combining Selected Features

All selected features are combined into a single text string for each movie:

- Genres + Keywords + Tagline + Cast + Director



```
[22]
#combining all the selected features

combined_features = df['genres']+' ' +df['keywords']+' '
+df['tagline']+' '+df['cast']+' '+df['director']

[23]
✓ 0s
print(combined_features)

0       Action Adventure Fantasy Science Fiction cultu...
1       Adventure Fantasy Action ocean drug abuse exot...
2       Action Adventure Crime spy based on novel secr...
3       Action Crime Drama Thriller dc comics crime fi...
4       Action Adventure Science Fiction based on nove...
                              ...
4798    Action Crime Thriller united states\u2013mexic...
4799    Comedy Romance  A newlywed couple's honeymoon ...
4800    Comedy Drama Romance TV Movie date love at fir...
4801      A New Yorker in Shanghai Daniel Henney Eliza...
4802    Documentary obsession camcorder crush dream gi...
Length: 4803, dtype: object
```

This combined text represents the movie's content profile.

6.2 Text Vectorization (TF-IDF)

TF-IDF (Term Frequency – Inverse Document Frequency) is used to convert text data into numerical vectors.

- Each movie becomes a vector
- Important words get higher weight
- Common words get lower weight

```
[ ]  #converting the text data to feature vectors

     vectorizer=TfidfVectorizer()

[ ]  feature_vectors = vectorizer.fit_transform(combined_features)

[ ]  print(feature_vectors)

     <Compressed Sparse Row sparse matrix of dtype 'float64'
             with 124266 stored elements and shape (4803, 17318)>
       Coords        Values
       (0, 201)      0.07860022416510505
       (0, 274)      0.09021200873707368
       (0, 5274)     0.11108562744414445
       (0, 13599)    0.1036413987316636
       (0, 5437)     0.1036413987316636
       (0, 3678)     0.21392179219912877
```

This results in a sparse matrix of size:

- **4803 × 17318**

7. Similarity Measurement

7.1 Cosine Similarity

Cosine Similarity measures the similarity between two vectors based on the angle between them.

- Value ranges from 0 to 1
- 1 means identical movies
- 0 means no similarity

```
Cosine Similarity

[98] ✓  #getting the similarity scores using cosine Similarity

        similarity = cosine_similarity(feature_vectors)

[32] ✓ 0s  print(similarity)
           similarity.shape

     [[1.         0.07219487 0.037733   ... 0.         0.         0.        ]
      [0.07219487 1.         0.03281499 ... 0.03575545 0.         0.        ]
      [0.037733   0.03281499 1.         ... 0.         0.05389661 0.        ]
      ...
      [0.         0.03575545 0.         ... 1.         0.         0.02651502]
      [0.         0.         0.05389661 ... 0.         1.         0.        ]
      [0.         0.         0.         ... 0.02651502 0.         1.        ]]
     (4803, 4803)
```

A similarity matrix of size **4803 × 4803** is generated.

8. User Interaction and Movie Matching

8.1 User Input

The user enters their favorite movie name.

8.2 Handling Spelling Mistakes

The `difflib.get_close_matches()` function is used to find the closest matching movie title from the dataset.

This ensures:

- Minor spelling errors are handled
- User experience is improved

9. Recommendation Process

Step-by-Step Flow:

1. Take user input (movie name)
2. Find the closest matching movie title
3. Get the index of that movie
4. Retrieve similarity scores for that movie
5. Sort movies based on similarity score (descending)
6. Display top 10 recommended movies

10. Sample Outputs

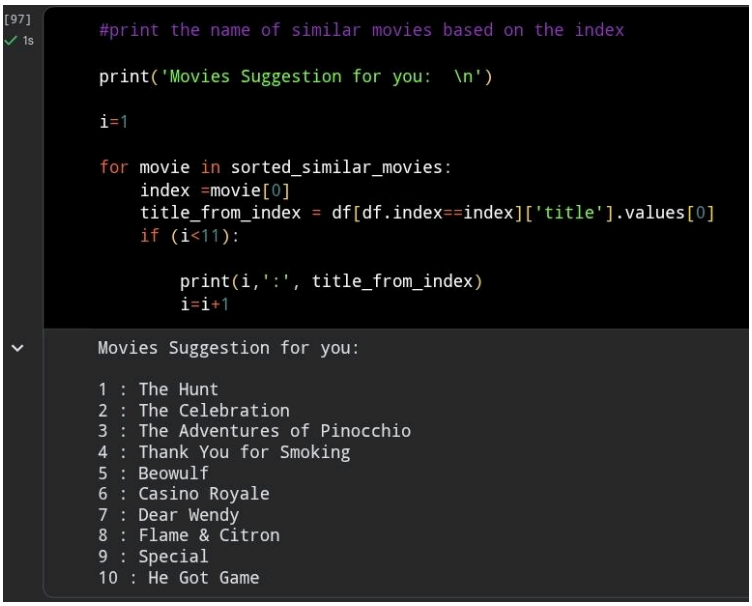
Example 1:

Input: The Nun

Matched Movie: The Hunt

Top Recommendations:

1. The Hunt
2. The Celebration
3. The Adventures of Pinocchio
4. Thank You for Smoking
5. Beowulf
6. Casino Royale
7. Dear Wendy
8. Flame & Citron
9. Special
10. He Got Game



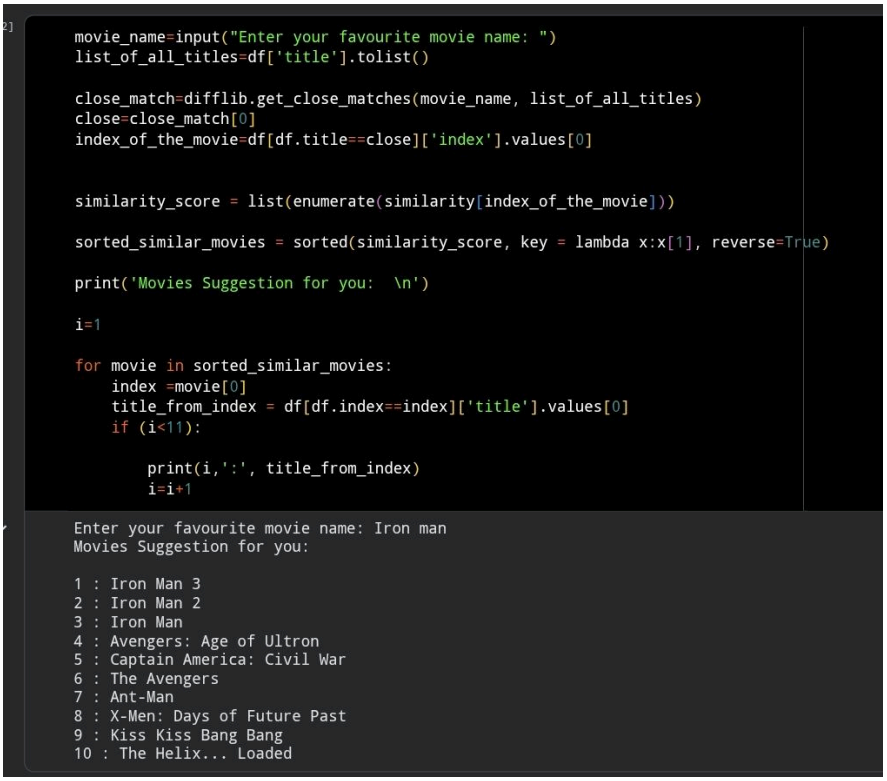
Example 2:

Input: Iron Man

Top Recommendations:

1. Iron Man 3
2. Iron Man 2
3. Iron Man
4. Avengers: Age of Ultron
5. Captain America: Civil War
6. The Avengers
7. Ant-Man
8. X-Men: Days of Future Past
9. Kiss Kiss Bang Bang

10. The Helix... Loaded



```
movie_name=input("Enter your favourite movie name: ")
list_of_all_titles=df['title'].tolist()

close_match=difflib.get_close_matches(movie_name, list_of_all_titles)
close=close_match[0]
index_of_the_movie=df[df.title==close]['index'].values[0]


similarity_score = list(enumerate(similarity[index_of_the_movie]))

sorted_similar_movies = sorted(similarity_score, key = lambda x:x[1], reverse=True)

print('Movies Suggestion for you:  \n')

i=1

for movie in sorted_similar_movies:
    index =movie[0]
    title_from_index = df[df.index==index]['title'].values[0]
    if (i<11):

        print(i,':', title_from_index)
        i=i+1
```

```
Enter your favourite movie name: Iron man
Movies Suggestion for you:

1 : Iron Man 3
2 : Iron Man 2
3 : Iron Man
4 : Avengers: Age of Ultron
5 : Captain America: Civil War
6 : The Avengers
7 : Ant-Man
8 : X-Men: Days of Future Past
9 : Kiss Kiss Bang Bang
10 : The Helix... Loaded
```

11. Advantages of the System

- No need for user ratings
- Personalized recommendations
- Works well with limited user data
- Simple and efficient implementation

12. Limitations

- Does not consider user behavior or preferences over time
- Recommendations are limited to metadata similarity
- Cannot handle collaborative filtering scenarios

13. Future Enhancements

- Add collaborative filtering
- Use user ratings and watch history
- Improve text preprocessing using NLP techniques
- Deploy as a web application
- Integrate deep learning-based recommendation models

14. Conclusion

This project successfully demonstrates the implementation of a **Content-Based Movie Recommendation System** using Python and Machine Learning. By leveraging TF-IDF vectorization and cosine similarity, the system effectively recommends movies similar to a user's preference. This approach forms a strong foundation for more advanced recommendation systems.

15. References

- Scikit-learn Documentation
- Pandas Documentation
- Python Official Documentation