🎥Movie Recommendation System – Project Documentation

Objective

To build a **content-based movie recommendation system** using textual data (metadata) from movies and suggest similar movies based on cosine similarity of their features.

🧩Dataset Used

- `tmdb_5000_movies.csv`
- `tmdb_5000_credits.csv`

These datasets were merged on the `title` column to bring all relevant metadata into a single DataFrame.

Step-by-Step Process

1. Data Loading

```
movies = pd.read_csv('tmdb_5000_movies.csv')
credits = pd.read_csv('tmdb_5000_credits.csv')
movies = movies.merge(credits, on='title')
```

2. Data Cleaning

- Selected important columns: `['id', 'title', 'overview', 'genres', 'keywords', 'cast', 'crew']`
- Removed rows with missing (`NaN`) values.
- Dropped duplicate records if any.

🪢 Data Preprocessing

3. Parsing JSON columns

Columns like `genres`, `keywords`, `cast`, and `crew` contain JSON strings. These were converted to lists of relevant values (e.g., genre names).

```
def convert(obj):
    l = []
```

```
    for i in ast.literal_eval(obj):
        l.append(i['name'])
    return l

movies['genres'] = movies['genres'].apply(convert)
movies['keywords'] = movies['keywords'].apply(convert)
```

4. Limiting Cast Members

Only the **top 3 cast members** were retained to keep the tags concise:

```
def convert3(obj):
    l = []
    counter = 0
    for i in ast.literal_eval(obj):
        if counter >= 3:
            break
        l.append(i['name'])
        counter += 1
    return l
```

5. Extracting Director

From the `crew` column, only the name of the **Director** was extracted:

```
def fetch_director(obj):
    for i in ast.literal_eval(obj):
        if i['job'] == 'Director':
            return [i['name']]
```

🥈 Creating Tags

- Converted the `overview` to a list of words using `.split()`.
- Removed spaces from names/keywords.
- Combined all features (`overview + genres + keywords + cast + crew`) into a new column `tags`.

```
movies['tags'] = movies['overview'] + movies['genres'] + movies['keywords'] +
movies['cast'] + movies['crew']
```

🎲Text Preprocessing

6. Converting Tags to Text

```
new_df['tags'] = new_df['tags'].apply(lambda x: " ".join(x))
```

7. Stemming

Used `PorterStemmer` from NLTK to reduce words to their root form.

```
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()

def stem(text):
    return " ".join([ps.stem(i) for i in text.split()])
```

8. Lowercasing

```
new_df['tags'] = new_df['tags'].apply(lambda x: x.lower())
```

Feature Extraction

9. Text Vectorization

Used `CountVectorizer` from scikit-learn (bag-of-words model):

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=5000, stop_words='english')
vectors = cv.fit_transform(new_df['tags']).toarray()
```

Similarity Measurement

10. Cosine Similarity

Used `cosine_similarity` instead of Euclidean distance due to high dimensional data:

```
from sklearn.metrics.pairwise import cosine_similarity
similarity = cosine_similarity(vectors)
```

🎯Recommendation Function

```
def recommend(movie):
    movie_index = new_df[new_df['title'] == movie].index[0]
    distances = similarity[movie_index]
    movies_list = sorted(list(enumerate(distances)), reverse=True, key=lambda
x: x[1])
    for i in movies_list[1:6]:  # Top 5 excluding itself
        print(new_df.iloc[i[0]].title)
```

Example:

```
recommend('Batman Begins')
```

Would return:

- The Dark Knight
- Batman
- The Dark Knight Rises
- 10th & Wolf
- etc.

🎻Saving Models and Data

Used `pickle` to save the required data for later use:

```
import pickle
pickle.dump(new_df, open('movies.pkl', 'wb'))
pickle.dump(new_df.to_dict(), open('movies_dict.pkl', 'wb'))
pickle.dump(similarity, open('similarity.pkl', 'wb'))
```

Summary

Step	Technique/Library Used
Data Merging	`pandas.merge()`
Parsing JSON	`ast.literal_eval()`
Feature Engineering	Tags creation from metadata
Text Preprocessing	Lowercase + Stemming
Vectorization	`CountVectorizer`

Step	Technique/Library Used
Similarity Calculation	`cosine_similarity()`
Model Storage	`pickle`
