Movie Recommendation System: Step-by-Step Explanation

1. Introduction

A **Movie Recommendation System** suggests movies to users based on their preferences. This system combines data preprocessing, vectorization, and similarity measurement to recommend movies. This document explains the theoretical concepts, practical implementation, and code details involved in creating the system using Python, Streamlit, and machine learning concepts.

---

2. Data Preprocessing (Jupyter Notebook)

The preprocessing is handled in the Movie-Recommender.ipynb file. Key steps:

2.1 Importing Libraries

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pickle
```

- **pandas**: For handling tabular data.
- **numpy**: For numerical computations.
- **CountVectorizer**: For text vectorization.
- **cosine_similarity**: To compute the similarity between movies.

2.2 Loading and Inspecting Data

```
movies = pd.read_csv('movies.csv')
```

- **Purpose**: Loads the dataset containing movie information.
- **Structure**: Contains columns like title, genres, keywords, cast, and crew.

2.3 Text Preprocessing

Steps:

1. Combine relevant text data (e.g., genres, keywords, cast).
2. Lowercase and remove spaces to standardize the text.

3. Use stemming to reduce words to their root forms (e.g., "running" → "run").

Code:

```
from nltk.stem.porter import PorterStemmer

ps = PorterStemmer()


def stem(text):
    words = text.split()
    return ' '.join([ps.stem(word) for word in words])


movies['tags'] = movies['tags'].apply(stem)
```

2.4 Vectorization

Convert text data into numerical vectors using **Bag-of-Words (BoW)**.

Theory:

- **Bag-of-Words**: Represents text as a collection of word frequencies.
- **CountVectorizer**: Converts text into a sparse matrix of token counts.

Code:

```
cv = CountVectorizer(max_features=5000, stop_words='english')

vectors = cv.fit_transform(movies['tags']).toarray()
```

2.5 Similarity Computation

Use **Cosine Similarity** to compute the similarity between movies based on their vector representations.

Theory:

- Measures the cosine of the angle between two vectors in a multi-dimensional space.
- **Formula**: cosine similarity=A·B‖A‖‖B‖\text{cosine similarity} = \frac{A \cdot B}{\|A\| \|B\|}

Code:

```
similarity = cosine_similarity(vectors)
```

2.6 Saving Processed Data

Save processed data to disk for use in the web application.

```
pickle.dump(movies_dict, open('movie_dict.pkl', 'wb'))
```

```
pickle.dump(similarity, open('similarity.pkl', 'wb'))
```

---

3. Web Application (Streamlit)

The app.py file implements the web interface for the recommendation system.

3.1 Setting Up the Environment

1. **Streamlit Framework**:
    - Lightweight framework for building interactive web apps in Python.
    - Run the app using streamlit run app.py.

3.2 Loading Data

Load the preprocessed movie data and similarity matrix:

```
similarity = pickle.load(open('similarity.pkl', 'rb'))
```

```
movies_dict = pickle.load(open('movie_dict.pkl', 'rb'))
```

```
movies = pd.DataFrame(movies_dict)
```

3.3 Movie Recommendation Logic

Define a function to recommend movies based on user selection:

Code:

```
def recommend(movie):
    movie_index = movies[movies['title'] == movie].index[0]
    distances = similarity[movie_index]
    movies_list = sorted(list(enumerate(distances)), reverse=True, key=lambda x: x[1])[1:6]
```

```
    recommended_movies = []
    for i in movies_list:
        recommended_movies.append(movies.iloc[i[0]].title)

    return recommended_movies
```

Steps:

1. Find the index of the selected movie.
2. Retrieve similarity scores with other movies.
3. Sort movies by similarity and pick the top 5.

3.4 Fetching Movie Posters

Use **The Movie Database (TMDb) API** to fetch movie posters.

```
import requests

def fetch_poster(movie_id):
    response = requests.get(f'https://api.themoviedb.org/3/movie/{movie_id}?api_key=YOUR_API_KEY')
    data = response.json()
    return "https://image.tmdb.org/t/p/w500/" + data['poster_path']
```

3.5 Building the Web Interface

1. Display the app title:
2. st.title('Movie Recommender System')
3. Create a dropdown to select a movie:
4. selected_movie_name = st.selectbox('Enter the movie name:', movies['title'].values)

5. Display recommendations:
6. if st.button('Recommend'):
7. names, posters = recommend(selected_movie_name)
8. cols = st.columns(5)
9. for i, col in enumerate(cols):
10. if i < len(names):
11. col.image(posters[i], caption=names[i])

---

4. Theoretical Concepts to Understand

4.1 Stemming

- Reduces words to their base form.
- Example: "playing", "played" → "play".
- Library: nltk.stem.porter.

4.2 Vectorization (Bag-of-Words)

- Converts text into numerical form.
- Ignores grammar and word order but considers word frequency.

4.3 Cosine Similarity

- Measures similarity between two vectors.
- Values range from 0 (completely dissimilar) to 1 (identical).

4.4 APIs

- TMDb API provides movie metadata and posters.

---

5. Conclusion

This project combines data preprocessing, machine learning, and web development to create an interactive Movie Recommendation System. Key techniques like stemming, vectorization, and cosine similarity ensure accurate recommendations. The integration of Streamlit makes the system user-friendly, while the TMDb API enhances the user experience with visuals.