Movie Recommendation System

Project Overview

Implementing a movie recommendation system that makes recommendations for films based on content-based filtering was the aim of this project. The algorithm determines how similar two films are by using essential elements from each film, including cast, crew, genre, and keywords. The textual content of these qualities serves as the foundation for the recommendations. Furthermore, experiments were conducted with several machine learning algorithms to evaluate their efficacy in enhancing suggestion accuracy.

Dataset Collection

The dataset for this project was collected from Kaggle, specifically the **TMDb Movie Metadata**. The dataset includes two CSV files:

- **tmdb_5000_movies.csv**: This file contains metadata like movie ID, title, overview, genres, and keywords.
- **tmdb_5000_credits.csv**: This file contains additional information like cast and crew details for each movie.

To establish a single dataset with all the attributes needed to construct the recommendation system, the two files were combined based on the title column. It's crucial to remember that the dataset is quite modest for an extensive recommendation system—there are only **4809** entries in all. The dataset is also unbalanced, meaning that some actors, directors, or genres are over-represented while others are under-represented. This mismatch may limit the diversity of recommendations and have an impact on the recommendation system's performance.

Data Preprocessing

Several preprocessing steps were applied to the dataset:

- **Merging Data**: The movies and credits datasets were merged using the title as the common column.
- **Handling Missing and Duplicated Data**: Missing values were identified and removed to ensure clean data for processing. Duplicated entries were also checked and handled accordingly.
- **Conversion of Columns**: The genres, keywords, cast, and crew columns, which contained lists of dictionaries in string format, were converted into readable lists using Python's ast module and custom functions. This conversion allowed for easier manipulation and usage of these columns.

- **Text Normalization**: The overview column was tokenized into individual words, and spaces in names within the genres, keywords, cast, and crew columns were removed to facilitate more efficient searching and tagging.
- **Creation of Tags**: A new column, tags, was created by concatenating the overview, genres, keywords, cast, and crew columns into a single text block. This column served as the basis for calculating movie similarity.
- **Lowercasing**: All the text in the tags column was converted to lowercase to ensure uniformity during the comparison.

Algorithm Selection

The recommendation system was built using **content-based filtering**. The following techniques were employed:

- **Text Vectorization (CountVectorizer)**: The tags column, which contained combined text information, was vectorized using **CountVectorizer** from the sklearn library. This technique converts text data into numerical form by creating a matrix where each column represents a word, and the rows represent movies. The value of each matrix cell corresponds to the frequency of the word in the respective movie's tags.
- **Cosine Similarity**: After vectorizing the text, **cosine similarity** was used to calculate the similarity between different movies. Cosine similarity measures the angle between two vectors (in this case, the vector representations of two movies). The smaller the angle, the more similar the movies are.
- **Porter Stemming**: To further improve text matching, **Porter Stemming** was applied to reduce words to their base or root form. For example, "running" would be reduced to "run", ensuring that similar words are counted as the same.

Model Training and Accuracy

Several machine learning algorithms were used to measure the system's performance.

- **Naive Bayes Algorithm**: Initially, the system used the **Naive Bayes** algorithm to classify movie recommendations. However, the **Naive Bayes Accuracy** was **26.30%**, indicating that it was not well-suited for this particular dataset due to its assumptions of feature independence and the imbalance in data distribution.
- **Random Forest and Decision Tree Algorithms**: To improve performance, **Random Forest** and **Decision Tree** algorithms were implemented. These algorithms proved to be more effective in capturing the relationships between the features in the dataset.
- **Voting Classifier**: Finally, a **Voting Classifier** was used, which combines the predictions of multiple algorithms (Random Forest and Decision Tree in this case). This ensemble method

significantly improved the model's performance, achieving a **Voting Classifier Accuracy** of **84.82%**. This was a substantial improvement over the Naive Bayes model and demonstrated that an ensemble approach can yield better recommendations.

Recommendation Logic

For a given movie:

- The system retrieves the index of the movie from the dataset.
- The cosine similarity of this movie with all other movies is computed.
- The movies are then sorted based on similarity scores, and the top 5 most similar movies are returned as recommendations.

Performance Metrics

The system's performance was evaluated based on accuracy metrics from machine learning algorithms and user satisfaction:

- **Naive Bayes Accuracy**: 26.30% (poor accuracy due to imbalanced data).
- **Voting Classifier Accuracy**: 84.82% (significantly better accuracy).

Challenges and Improvements

- **Small and Imbalanced Dataset**: With only 4809 entries, the dataset is relatively small for a robust recommendation system. Moreover, the imbalance in the distribution of genres, cast, and crew makes it harder to ensure a diverse set of recommendations. This limitation could be addressed by using a larger and more diverse dataset in future iterations.
- **Handling Large Datasets**: As the dataset grows, the system's performance may slow down due to the time taken to compute cosine similarities across many movies. Future improvements could include the use of dimensionality reduction techniques like **TruncatedSVD** to speed up similarity calculations.
- **Cold Start Problem**: Like many content-based systems, this recommendation system struggles with the cold start problem. Movies with little metadata or new movies may not have sufficient information to generate accurate recommendations. Incorporating user ratings and collaborative filtering could mitigate this issue.
- **Improved Text Processing**: Future improvements could involve using more advanced text processing techniques like **Word2Vec** or **TF-IDF** to capture deeper semantic relationships between words in the tags column.

Future Work and Deployment

- **Larger Dataset**: In future versions, a larger and more diverse dataset should be used, potentially incorporating trending and recent movie data to ensure that recommendations are more current and varied.
- **Local Content Recommendation**: The system can be expanded to recommend not only English movies but also content in Hindi and Bangla, including **local movies, TV shows (Natok), and songs**. This will help create a more localized and culturally relevant recommendation system for different regions.
- **Web Deployment**: Future include connecting the recommendation system with a website for real-world deployment, allowing users to search for movies and receive recommendations through a user-friendly interface. The website can use popular web frameworks like **Flask** or **Django** for backend integration with the recommendation system.

Sample of our project output for Recommendations:

Using the system, here are some example recommendations:

- For the movie **Avatar**:
    - The similarity-based recommendations are:
        - Pirates of the Caribbean: At World's End
        - Pirates of the Caribbean: Dead Man's Chest
        - The Curious Case of Benjamin Button
        - Spider-Man 3
        - Spider-Man
- For the movie **Batman Begins**:
    - The similarity-based recommendations are:
        - The Dark Knight
        - Batman
        - Batman v Superman: Dawn of Justice
        - The Dark Knight Rises
        - Man of Steel