Project Report:

Content-Based MovieRecommendation System

Aayush Tiwari (524110033)
Rahul Gupta(524110004)

---

1. Introduction

1.1 Project Goal

The primary objective of this project was to develop a **Content-Based Movie Recommendation System**. This system is designed to suggest movies to a user based on the content features of a movie they have already enjoyed, thereby enhancing user experience and discovery on a movie platform.

1.2 Data Source

The system utilizes the **TMDB 5000 Movie Dataset**, which comprises detailed metadata for approximately 5,000 films. This dataset includes essential features such as overview, genres, keywords, cast, and crew information, all crucial for determining movie similarity.

2. Methodology: Content-Based Filtering

2.1 Feature Engineering and Preprocessing

To prepare the raw data for modeling, a robust data processing pipeline was implemented using the **Pandas** library in Python:

1. **Data Merging:** The tmdb_5000_movies.csv and tmdb_5000_credits.csv files were merged on the unique movie ID.
2. **Feature Extraction:** Complex JSON-string columns (genres, keywords, cast, crew) were parsed using the ast module.
    - For **cast**, the top three actors were extracted.
    - For **crew**, the director's name was extracted (as indicated by the 'Director' job).
3. **Feature Transformation (Tag Generation):** All relevant textual features (overview, genres, keywords, cast, director) were concatenated into a single, clean string column called **tags**. Spaces were removed from multi-word names (e.g., "Science Fiction" became "ScienceFiction") to ensure the vectorizer treats them as single, distinct entities.

2.2 Model Selection and Development

The core of the system is a **Content-Based Filtering** approach, which relies on measuring the similarity between a movie and all other movies in the dataset based on their content features.

1. **Text Vectorization (CountVectorizer):** The final tags column was converted into a matrix of token counts using **Scikit-learn's CountVectorizer**. This vectorizer was configured to extract the 5,000 most common words, transforming the movie tags into a high-dimensional vector space.
2. **Similarity Calculation (Cosine Similarity):** The similarity between every pair of movies in the vector space was calculated using the **Cosine Similarity** metric.
    - **Cosine Similarity** measures the cosine of the angle between two vectors. A cosine value closer to $1$ indicates higher similarity, while a value closer to $0$ indicates lower similarity, regardless of the magnitude of the vectors. The resulting **similarity matrix** is the recommendation model.
3. **Model Persistence:** The final processed movie data (movies.pkl) and the similarity matrix (similarity.pkl) were saved using **Python's pickle** module for fast loading by the web application.

3. Implementation: Web Application Stack

The recommendation model was deployed as a functional web application with a client-server architecture.

3.1 Backend (Python / Flask)

- **app.py:** A **Flask** application was built to serve the model predictions via a RESTful API.
- **Endpoint:** A primary POST endpoint (/recommend) accepts a JSON payload containing the user's selected **movie title**.
- **Prediction Logic:**
    1. The Flask app loads movies.pkl and similarity.pkl on startup.
    2. Upon receiving a request, it uses the movie title to find the movie's index in the dataset.
    3. It fetches the corresponding row from the similarity matrix.
    4. The scores are sorted, and the top **5 most similar movies** (excluding the input movie itself) are returned as a JSON array.

3.2 Frontend (HTML / JavaScript / Tailwind CSS)

- **recommendations.html:** A single-page web interface was developed using **HTML**, modern **JavaScript**, and **Tailwind CSS** for responsive, clean styling.
- **Libraries:** The **Select2** library was integrated to provide an efficient and searchable dropdown for the user to easily select from the list of all available movies.
- **Interaction:** The JavaScript code fetches the list of all movie titles from the backend at startup and populates the dropdown. When a user clicks the "Get Recommendations" button, a fetch request is sent to the Flask API, and the returned list of 5 movies is dynamically displayed on the page.

4. Results and Conclusion

The system successfully generates relevant recommendations by effectively combining content features. The use of the **CountVectorizer** and **Cosine Similarity** proves highly efficient for this size of dataset, providing near-instantaneous results.

Key Performance Indicators:

- **Latency:** Recommendations are generated in under 100 milliseconds, providing a seamless user experience.
- **Relevance:** The content-based approach ensures that suggested movies share core features (actors, genres, director) with the input movie, meeting the project's goal.

In conclusion, the project delivers a fully functional and scalable Content-Based Movie Recommendation System, demonstrating proficiency in data science, machine learning, and full-stack web development integration.

5. Future Scope

To further enhance the project, the following areas could be explored:

1. **Hybrid Approach:** Implement a **Collaborative Filtering** component (e.g., using matrix factorization) and combine it with the current content-based results to improve the diversity and accuracy of recommendations.
2. **Deployment:** Deploy the Flask application to a cloud platform (e.g., AWS, GCP) to make the system publicly accessible and scalable.
3. **Sentiment/Review Analysis:** Incorporate natural language processing (NLP) on movie reviews or synopses to capture deeper, nuanced content similarity beyond basic keywords.