Movie Recommendation System

Abstract

This project implements a hybrid movie recommendation system using the MovieLens 100K dataset, combining collaborative filtering on user–item ratings with content-based filtering from genre features. A Streamlit UI collects user preferences and displays the top five personalized movie recommendations. The system balances accuracy and coverage by leveraging both behavioral patterns and content attributes.

Introduction

Recommender systems help users discover relevant content from large catalogs by learning preference signals from ratings and metadata. Collaborative filtering exploits patterns in user ratings to infer taste similarity among users, while content-based filtering leverages item features such as genres to propose similar titles. This project builds a hybrid approach on the MovieLens 100K dataset to combine the strengths of both methods, improving robustness under sparse data conditions. The solution wraps the recommendation logic in a simple Streamlit application for interactive demonstration and practical use.

Tools Used

- **Programming Language**: Python 3.x
- **Core Libraries**: pandas (data manipulation), numpy (numerical operations), scikit-learn (TF-IDF vectorization, cosine similarity)
- **UI Framework**: Streamlit (interactive web interface)
- **Optional NLP**: TextBlob (sentiment analysis on reviews, if available)
- **Dataset**: MovieLens 100K, containing u.data (user ratings) and u.item (movie metadata with genre flags)
- **Development Environment**: Python virtual environment (venv) for dependency isolation

Steps Involved in Building the Project

1. Data Acquisition and Loading

Download the MovieLens 100K dataset and place the u.data file (user_id, movie_id, rating, timestamp) and u.item file (movie metadata with one-hot encoded genre flags) in the data folder. Read the files using pandas with appropriate delimiters (tab for u.data, pipe for u.item) and encoding (latin-1 for u.item). Use raw strings or forward slashes in Windows file paths to avoid unicode escape errors.

2. Data Preprocessing and Feature Engineering

Create a unified movie table with a derived genres field by concatenating active genre flags into a space-separated string (e.g., "Action Comedy Drama"). Handle missing values by filling empty genre strings and ensuring consistent column naming. Verify data integrity by checking for duplicate entries and standardizing movie IDs across both datasets.

3. Collaborative Filtering Implementation

Construct a user–movie rating matrix with users as rows and movies as columns, filling missing ratings with zeros for similarity computation. Calculate cosine similarity between all user pairs to identify nearest neighbors. For a target user, retrieve top similar users and aggregate their highly rated movies that the target user has not yet seen. Rank these candidate movies by the mean rating from similar users and return the top 5 recommendations.

4. Content-Based Filtering Using Genres

Vectorize the concatenated genre strings using TF-IDF with a custom token pattern that treats each genre as a distinct token. Compute the cosine similarity matrix between all movie pairs based on their genre vectors. For a selected favorite movie or set of preferred genres, retrieve the most similar titles by sorting similarity scores in descending order and selecting the top 5 matches (excluding the input movie itself).

5. Hybrid Recommendation Strategy

Combine candidate sets from collaborative and content-based methods using either intersection (for high-confidence recommendations when both methods agree) or weighted union. Optionally implement a weighted scoring function: $s = \alpha \cdot CF\_score + (1 - \alpha) \cdot CB\_score$, where α can be tuned through validation experiments. This hybrid approach balances the personalization strength of collaborative filtering with the interpretability and cold-start resilience of content-based filtering.

6. Streamlit User Interface Development

Build an interactive web application with input controls for user ID selection, favorite movie entry (via text input or dropdown), and number of recommendations. Implement separate buttons or tabs for collaborative, content-based, and hybrid recommendation modes. Display the top 5 recommendations in a clean table format showing movie titles and genres, with optional columns for predicted ratings or similarity scores.

7. Optional Sentiment Analysis Integration

If review text data is available, compute sentiment polarity for each review using TextBlob and aggregate to movie-level average sentiment scores. Use these scores as a post-filtering mechanism to exclude movies with consistently negative sentiment or as a tie-breaker when ranking candidates with similar collaborative or content scores. This enhancement adds an emotional quality dimension to the recommendations.

Conclusion

The hybrid movie recommendation system successfully combines collaborative filtering with content-based genre similarity to deliver accurate and diverse recommendations on the MovieLens 100K dataset. By leveraging both user behavior patterns and item attributes, the system maintains good performance even when rating data is sparse. The lightweight Streamlit interface makes the solution interactive and demo-ready for academic presentations or classroom use. The modular code architecture supports straightforward extensions, including improved hybrid weighting schemes, incorporation of implicit feedback signals, integration of sentiment-aware re-ranking, and deployment to cloud platforms for broader accessibility.