Suggesteria Report

Datasets

The datasets that we have used is movies.csv and ratings.csv files which have the following properties:
`movies.csv` provides metadata about movies, including **title and genres**.
`ratings.csv` links **users to movies** via `movieId`, storing **ratings and timestamps**.
The datasets can be merged using `movieId` to analyze **movie popularity, trends, or personalized recommendations**.

Algorithms Implemented in `movie_recommender.py` and Their Justification

This movie recommender system is based on **collaborative filtering** using the **k-Nearest Neighbors (k-NN) algorithm with cosine similarity**. Below is a breakdown of the approach:

---

1. Data Preprocessing

- The `ratings.csv` is converted into a **user-item interaction matrix** (`final_dataset`) where:
  - **Rows:** Movies (`movieId`)
  - **Columns:** Users (`userId`)
  - **Values:** Ratings given by users
- Missing values are replaced with `0` (i.e., unrated movies).

Why this preprocessing?

- The interaction matrix is needed to apply collaborative filtering.
- Movies rated by fewer than **10 users** and users who rated fewer than **50 movies** are removed to filter out **sparse data** and improve recommendation quality.

---

2. Converting to a Sparse Matrix

csr_data = csr_matrix(final_dataset.values)

- Converts the dense user-movie matrix into a **compressed sparse row (CSR) matrix** to save memory and improve computation efficiency.
- This is crucial because **most users rate only a few movies**, making the dataset highly sparse.

Why use a sparse matrix?

- Reduces memory usage.
- Makes operations like similarity calculations much faster.

---

3. k-Nearest Neighbors (k-NN) Algorithm for Recommendation:

- Uses **k-NN with cosine similarity** to find movies that are **most similar** to a given movie.
- **Brute-force search** is used because the dataset size is manageable.
- The number of neighbors (`n_neighbors=20`) helps retrieve the most relevant movies.

Why k-NN with Cosine Similarity?

- **k-NN** works well for collaborative filtering when the dataset is not too large.
- **Cosine similarity** is a good metric for recommendation problems since it measures how similar two movies are **based on user ratings**, regardless of the absolute rating values.

---

4. Movie Recommendation Function

How it Works?

1. Finds the `movieId` for the input movie title.
2. Uses k-NN to find the **10 most similar movies** based on cosine similarity.
3. Returns a **DataFrame** with recommended movies sorted by similarity.

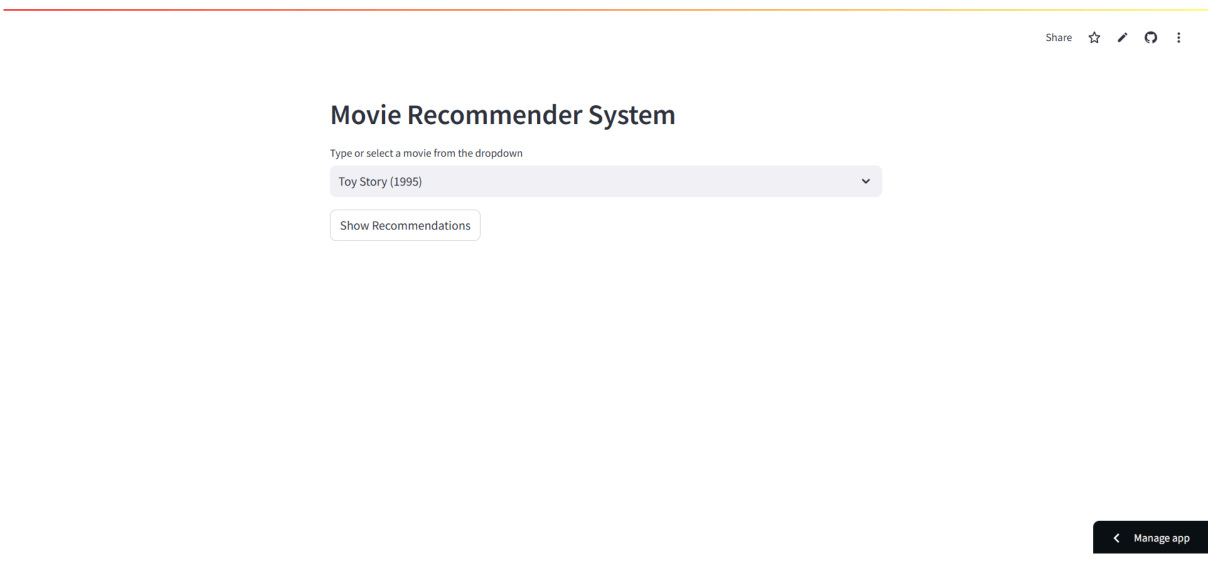
Why this approach?

- Finds recommendations **based on user behavior** rather than just genre similarity.
- The **distance metric (cosine similarity)** ensures that movies with similar audience preferences are recommended.

Overall Justification of the Approach

- **Collaborative Filtering (k-NN based)** was chosen because:
    - It **does not require explicit movie features** (e.g., genre, cast) but instead learns from user interactions.
    - It **scales well** for mid-sized datasets.
    - Cosine similarity captures similarity **based on rating behavior**, not just numeric values.

- **Sparse Matrix Optimization** ensures **efficient computation** despite many missing ratings.
- **Preprocessing Steps (Filtering low-rated movies/users)** ensure **better-quality recommendations**.

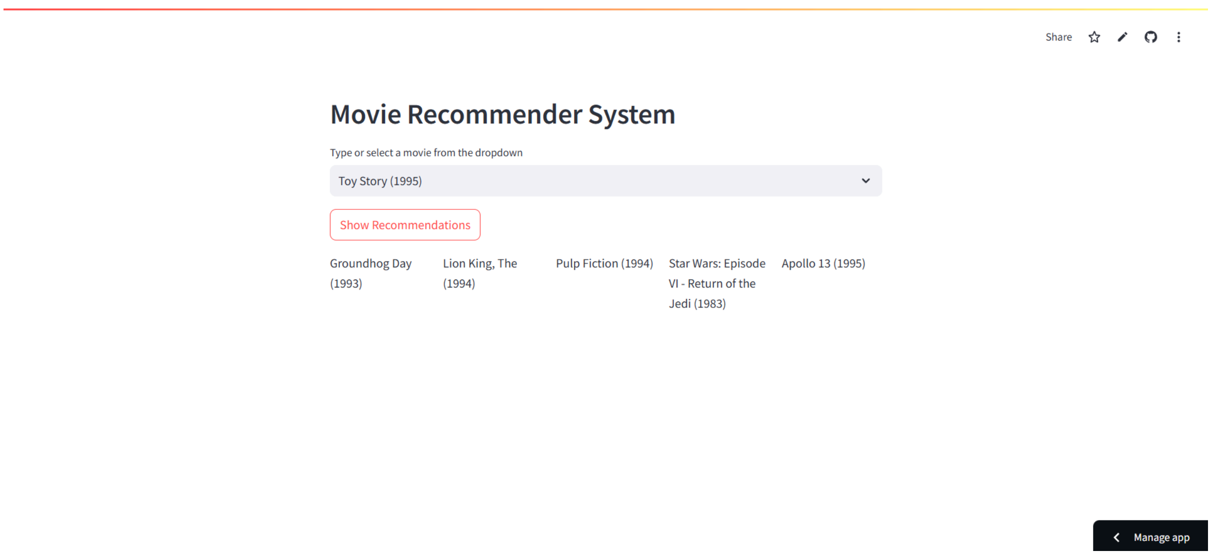


Movie Recommender System

Type or select a movie from the dropdown

Toy Story (1995)

Show Recommendations



Movie Recommender System

Type or select a movie from the dropdown

Toy Story (1995)

Show Recommendations

Groundhog Day (1993) | Lion King, The (1994) | Pulp Fiction (1994) | Star Wars: Episode VI - Return of the Jedi (1983) | Apollo 13 (1995)

<u>Challenges Faced:</u>

- Deploying the app was a big challenge as it was being done for the first time. There were many errors which seem unsolvable even after trying multiple approaches but we figured a way out after brainstorming on it for a long period of time. The challenges include setting up the repository on github and get its reflection on the streamlit interface, getting the code run on the deployed app and many others.

- We also faced challenges while writing the code for the model, like which algorithm to use, many logical errors, but we always found a way out to those problems by searching the internet and using ChatGPT.
- We struggled hard to find a dataset for ourselves. We wanted to make a recommendation system on movies and when we started looking for the data we were not able to find any which suits our needs.