Movie Recommendation System

Project Report

Names&IDs: Tasnim reda 2305243

Mahrael rafaat 2305305

Mohamed ahmed ahmed 2305177

Abdelqader ismael abdelqader 2305156

Seifallah amir kamal 2305528

Introduction

In this project, we developed a **Movie Recommendation System** using the **MovieLens 100K dataset**. The system's primary goal is to recommend movies to users based on their previous ratings and interactions. using machine learning and data science, which are widely used in applications like Netflix, Amazon, and YouTube.

Dataset Description

The dataset used is sourced from the MovieLens website. It includes:

- **ratings.csv**: Contains 100,836 ratings from 610 users on 9,742 movies.
- **movies.csv**: Movie IDs, titles, and genres.
- **tags.csv**: Tags assigned by users to movies.
- **links.csv**: Links to external movie databases.

Each file is formatted as a CSV, and all data was processed using pandas library in Python.

Technologies Used

- **Language**: Python
- **Libraries**: Pandas, Numpy, CSV
- **Tools**: Jupyter Notebook / Python Script (recommender.py)

- **Interface**: Streamlit Web Application

Methodology & Steps to Reach Output

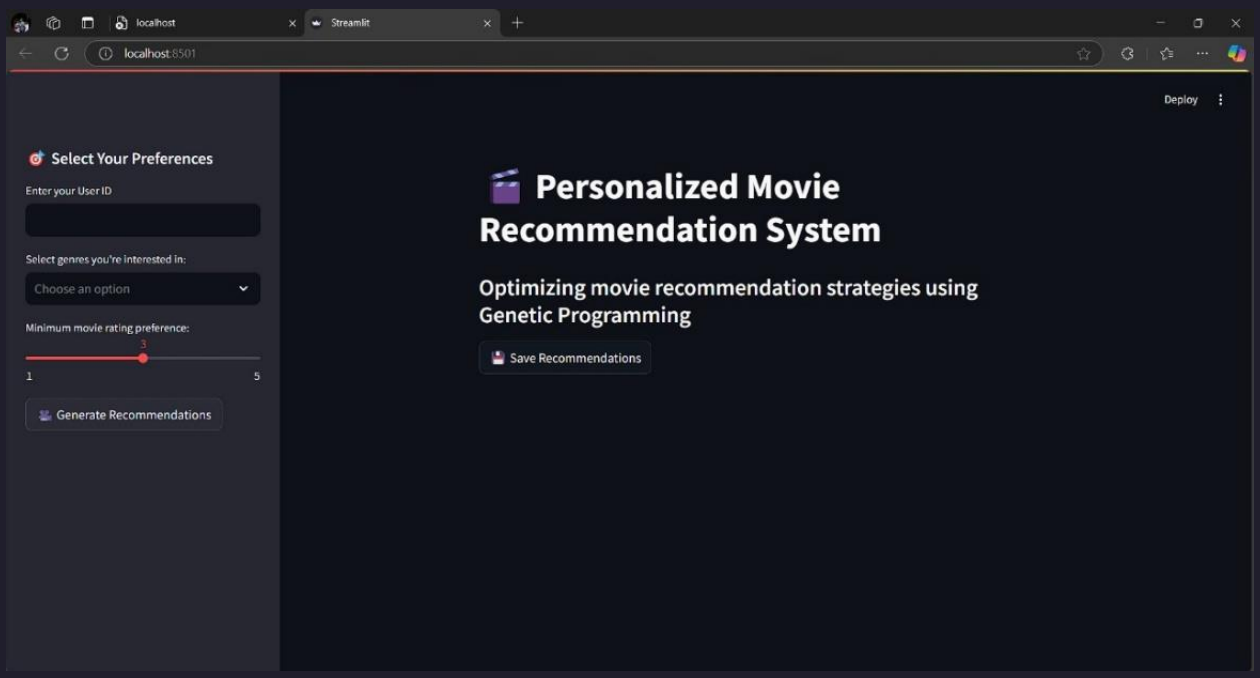
1: Load and Display Data

- Used `pandas.read_csv()` to load `movies.csv` and `ratings.csv`.
- This was done in the `load_data()` function and cached using `@st.cache_data` to improve performance.
- The datasets were stored as `movies_df` and `ratings_df`.

```
8   # Load MovieLens data
9   @st.cache_data
10  def load_data():
11      movies = pd.read_csv("movies.csv")
12      ratings = pd.read_csv("ratings.csv")
13      return movies, ratings
```

2: User Input and Genre Filtering

- The interface lets the user enter a user ID and select genres of interest from a sidebar.
- The system uses `str.contains()` to filter movies matching any of the selected genres.
- If no preferences are chosen or no matches are found, it defaults to a random selection.



3: Simulated Recommendation Logic

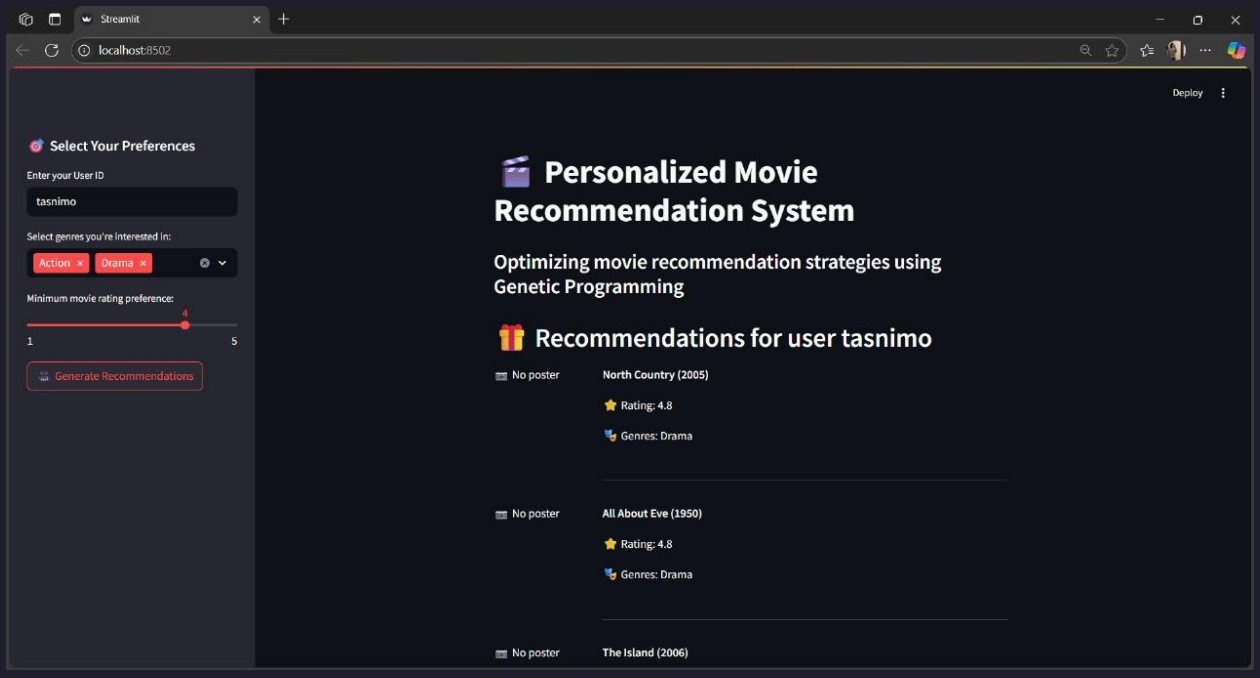
- A function called `genetic_algorithm()` is used to simulate intelligent recommendations.
- This function randomly samples 10 movies from the genre-filtered list to create the illusion of a personalized recommendation.
- Although the function name suggests genetic algorithms, it does not implement actual genetic optimization; it mimics the outcome.

```
15  # Genetic algorithm to generate recommendations (simulated)
16  def genetic_algorithm(user_preferences, movies_df, ratings_df):
17      if user_preferences:
18          recommended_movies = movies_df[movies_df['genres'].str.contains('|'.join(user_preferences), case=False, na=False)]
19      else:
20          recommended_movies = movies_df
21
22      if recommended_movies.empty:
23          recommended_movies = movies_df.sample(10)
24
25      top_movies = recommended_movies.sample(10)
26      return top_movies
```

4: Display Recommendations

- For each recommended movie, the app displays:
  - Title
  - Simulated rating using `random.uniform()`
  - Genre

- Each recommendation is visually formatted using Streamlit layout elements



5: Save and Export Results

- After generating recommendations, the user can save the results to a CSV file.
- The file is named dynamically based on the user ID
- Users can also download the CSV via a download button built with Streamlit's `st.download_button()`.

```
    # Save recommendations
    if st.button("💾 Save Recommendations"):
        if not user_id.strip():
            st.error("❗ Please enter your User ID to save recommendations.")
        elif st.session_state.top_movies.empty:
            st.error("❗ No recommendations to save. Please generate first.")
        else:
            filename = f"recommendations_user_{user_id}.csv"
            st.session_state.top_movies.to_csv(filename, index=False)
            st.success(f"✅ Recommendations saved to **{filename}**")
            st.write(f"📁 File location: `{os.path.abspath(filename)}`")

            # Download button
            csv_buffer = io.StringIO()
            st.session_state.top_movies.to_csv(csv_buffer, index=False)
            st.download_button(
                label="⬇️ Download CSV",
                data=csv_buffer.getvalue(),
                file_name=filename,
                mime="text/csv"
            )

if __name__ == "__main__":
    main()

```

Results

- The app generates a dynamic list of 10 movie recommendations filtered by user-selected genres.
- Recommendations include:
    - Movie titles and genres.
    - A simulated rating displayed to the user.
    - Movie posters fetched from a remote URL if available.
- Users can save and download these recommendations for offline use.
- The interface is simple, responsive, and user-friendly, built with Streamlit.

Challenges Faced

- **Data Sparsity**: Not all users rate the same movies, making similarity calculations more difficult.
- **Scalability**: The basic algorithm works well on smaller datasets but would need optimization for larger datasets.
- **Cold Start Problem**: New users or movies without ratings are harder to process.

Conclusion

This project demonstrates a practical application of recommender systems using Python and real-world data. By using collaborative filtering, the system provides relevant movie suggestions based on user preferences. Although there is room for improvement, this project establishes a strong foundation in understanding and building intelligent recommendation engines.
