Table of Contents

* Introduction
* Problem statement
* Solution overview
* technical implementation - The program
  + Data collection and processing
  + Recommendation algorithms
  + Frontend integration
  + Do we need to explain each code?
* Experimentations and Results
* Challenges and Learning
* Conclusions
* Installations and Running Instructions

**Introduction**

**What is Popcorn?**

Popcorn is a movie recommendation system designed to help users find movies for their movie nights based on collaborative preferences. The system combines modern web technologies with machine learning techniques, including **Nearest Neighbors (KNN)** and **Singular Value Decomposition (SVD),** to generate personalized movie suggestions that appeal to multiple viewers' tastes. By combining insights from historical user preferences with real-time data from external APIs, Popcorn enhances the movie selection process, making it both efficient and enjoyable.

**What is the Movie Night Dilemma?**

The Movie Night Dilemma is a common social challenge that emerges when friends, couples, or families attempt to select a film that satisfies everyone's tastes. Unlike traditional recommendation systems that focus on individual users, the challenge here lies in finding a middle ground between different, often conflicting, viewing preferences. Consider a couple planning their date night - one person might enjoy intense psychological thrillers while their partner prefers lighthearted comedies. Existing streaming platforms typically focus on single-user recommendations, leaving the challenge of reconciling diverse tastes unaddressed.

**The Problem**

1. **Diverse Preferences:**

Different users often have contrasting movie preferences, making it difficult to agree on a single movie. For example, one person might prefer high-action blockbusters, while another enjoys thought-provoking dramas or comedies.

1. **Lack of Group-Focused Solutions:**

Existing streaming platforms and recommendation systems are designed for individual users and do not offer a collaborative way to recommend movies for multiple people.

1. **Information Overload:**

With thousands of movies available across various streaming platforms, users can easily become overwhelmed when trying to choose.

1. **Limited Contextual Information:**

Users often lack quick access to essential information like posters, summaries, or trailers, making it harder to finalize a decision.

The goal of this project is to develop a group-focused movie recommendation system that analyzes preferences from multiple users, suggests movies that are likely to appeal to all participants, provides additional contextual information (e.g., posters, summaries) to enhance the selection process, and solves the Movie Night Dilemma by reducing the friction in group decision-making and transforming it into a seamless, enjoyable experience.

**Solution – Collaborative Filtering**

**Overview**

Popcorn's solution involves using a collaborative filtering approach to analyze historical user preferences and find common ground between two users.

The system uses K-Nearest Neighbors (KNN) and Singular Value Decomposition (SVD)to recommend movies based on similarity scores.

**Key Components**

* User-Movie Matrix: A sparse matrix representing user ratings for various movies.
* Nearest Neighbors Model: A KNN model trained on the user-movie matrix to find similar movies.
* SVD Model: A SVD model trained on the user-movie matrix to find similar movies.
* TMDb API Integration: Real-time data fetching for movie posters and summaries.

**Usage**

The backend processes inputs from both users and generates five movie recommendations that are most likely to be enjoyed by both parties.

Recommendations are generated by both models, allowing for a direct comparison of their performance.

Each recommendation includes a poster and a brief summary fetched from the TMDb API.

**Technical Implementation**

**Data collecting and Processing**

Popcorn uses two data sets:

* Movies Dataset: A CSV file containing movie titles and metadata.
* Ratings Dataset: A CSC file with user ratings for various movies.

The datasets are merged to create a user-movie rating matrix. Sparse representations of the matrix are used to improve computational efficiency.

The datasets, sourced from Kaggle, ensure reliability and quality for our analysis.

**Recommendation Algorithms**

The recommendation engine is built using two powerful algorithms: K-Nearest Neighbours (KNN) and Singular Value Decomposition (SVD).

**Collaborative Filtering**

Collaborative filtering works by analyzing patterns in user behaviour, such as movie ratings, to recommend content. Instead of relying on explicit metadata about movies (e.g., genre or director), it focuses on user interactions to uncover hidden relationships between items and users.

**K-Nearest Neighbours (KNN)**

The KNN algorithm is used for item-based collaborative filtering in this recommendation system. Here’s how it works:

1. **Finding Neighbours:**
   * Given a specific movie as input, the algorithm calculates its similarity to all other movies in the dataset. This similarity is measured using cosine similarity.
   * The algorithm then selects the topmost similar movies, known as the “neighbours.”
2. **Generating Recommendations:**
   * When two input movies are provided, the system finds the top neighbours for each movie independently.
   * A combined list of recommendations is created by merging the two sets of neighbours, ensuring at least 5 unique and highly relevant results.
3. **Key Functions:**
   * **Recommend\_movies\_by\_title\_nn**: Identifies movies most similar to a given title using the KNN model.
   * **Recommend\_for\_two\_users\_with\_five\_results**: Combines recommendations for two users or input movies, guaranteeing at least five unique recommendations for a more diverse result set.

**Singular Value Decomposition (SVD)**

SVD is a matrix factorization technique widely used in recommendation systems, particularly for collaborative filtering. It works by decomposing the user-item interaction matrix into three smaller matrices:

1. **User Matrix (U):** Represents the relationship between users and latent factors.
2. **Singular Values (Σ):** Captures the strength of each latent factor.
3. **Item Matrix (V^T):** Represents the relationship between items (movies) and latent factors.

The decomposition allows us to project both users and items into a shared latent feature space, enabling the system to predict ratings for unseen movies by calculating the dot product of the corresponding user and item vectors.

**What Are Latent Features?**

Latent features are hidden factors that capture underlying patterns in the data. For example, in the context of movies, latent features might represent abstract concepts like "genre preference," "movie popularity," or "actor influence." These features are not explicitly labelled but are inferred from the data during matrix factorization.

**How SVD Generates Recommendations:**

1. **Decomposing the Matrix:**
   * The original user-item matrix (containing user ratings) is factorized into the U, Σ, and V^T matrices.
   * This step identifies latent factors that capture hidden relationships between users and movies.
2. **Predicting Ratings:**
   * For a given user and a specific movie, the system calculates a predicted rating using the reconstructed user-item matrix derived from the factorized components.
3. **Ranking Recommendations:**
   * Movies with the highest predicted ratings for a user are recommended. These movies are ranked based on their scores.
4. Key Functions for SVD:
   * find\_similar\_movies\_svd: Identifies movies similar to a given title based on latent features.
   * recommend\_for\_two\_users\_with\_five\_results\_svd: Combines recommendations for two input movies, ensuring a diverse list of at least five unique results.

**Frontend Integration**

The web interface is built using HTML, CSS, and JavaScript. Flask is used as the background framework to handle API requests and serve the frontend.

* Movie Input: Users enter movie names, and suggestions are fetched dynamically.
* Recommendation Display: Recommended movies are displayed as a carousel with posters and summaries.

We provide separate recommendations for both models.

* Random Movie Generator: a dice button allows the users to get random movies suggestions.

**Add pictures of the web app**

**Experiments and Results**

**Experiment 1 – KNN model with varied number of inputs**

The goal of this experiment was to evaluate how KNN model performs when given different numbers of input movies. Specifically, we tested the system with:

* 2 movies input: ["The Princess Diaries (2001)", "Hannah Montana: The Movie (2009)"]
* 5 movies input: ["The Princess Diaries (2001)", "Hannah Montana: The Movie (2009)", "The Other Woman (2014)", "Just Friends (2005)", "Fight Club (1999)"]
* 10 movies input: ["The Princess Diaries (2001)", "Hannah Montana: The Movie (2009)", "The Other Woman (2014)", "Just Friends (2005)", "Fight Club (1999)", "Titanic (1997)", "Friends with Benefits (2011)", "The Pink Panther (2006)", "The Addams Family (1991)", "Father of the Bride (1950)"]

**Results and Observations**

**For 2 Input Movies**:  
Recommendations:

* The Princess Diaries 2: Royal Engagement (2004)
* Legally Blonde (2001)
* Freaky Friday (2003)
* Camp Rock (2008)
* How to Lose a Guy in 10 Days (2003)

Observation:  
The recommendations were primarily focused on lighthearted family comedies and dramas, closely matching the genres of the input movies. This shows that the system successfully identified relevant movies based on genre and tone.

**For 5 Input Movies**:  
Recommendations:

* The Matrix (1999)
* Memento (2000)
* American Beauty (1999)
* The Lord of the Rings: The Fellowship of the Ring (2001)
* Pulp Fiction (1994)

Observation:  
The recommendations shifted to a mix of action, thrillers, and classic dramas, reflecting the increased diversity of genres in the input set. This demonstrates that the system adapts well to varied input by providing a broader range of recommendations.

**For 10 Input Movies**:  
Recommendations:

* The Matrix (1999)
* Just Go with It (2011)
* The Proposal (2009)
* We're the Millers (2013)
* The Ugly Truth (2009)

Observation:  
Despite the larger and more diverse input set, the system provided a relatively focused set of romantic comedies and lighthearted movies. This suggests that movies with overlapping genres or themes received higher cumulative similarity scores, dominating the results.

Based on the results, the following improvements could enhance the recommendation system:

1. **Introduce diversity constraints**: Prevent domination by a single genre in larger input sets by enforcing a minimum level of diversity in the recommendations.
2. **Incorporate genre balancing**: Implement a mechanism that balances recommendations across genres when input sets are large and diverse.

**Experiment 2 – SVD Model with Varied Number of Inputs**

The goal of this experiment was to evaluate how the SVD-based recommendation system performs when given different numbers of input movies. Specifically, we tested the system with the following input sets (same input sets we used in the KNN experiment):

* **2 movies input:** ["The Princess Diaries (2001)", "Hannah Montana: The Movie (2009)"]
* **5 movies input:** ["The Princess Diaries (2001)", "Hannah Montana: The Movie (2009)", "The Other Woman (2014)", "Just Friends (2005)", "Fight Club (1999)"]
* **10 movies input:** ["The Princess Diaries (2001)", "Hannah Montana: The Movie (2009)", "The Other Woman (2014)", "Just Friends (2005)", "Fight Club (1999)", "Titanic (1997)", "Friends with Benefits (2011)", "The Pink Panther (2006)", "The Addams Family (1991)", "Father of the Bride (1950)"]

**Results and Observations**

**For 2 Input Movies:**

Recommendations:

* 10 Things I Hate About You (1999)
* The Devil Wears Prada (2006)
* How to Lose a Guy in 10 Days (2003)
* The Proposal (2009)
* Miss Congeniality (2000)

Observation:  
The recommendations largely consisted of romantic comedies and light-hearted dramas, closely aligning with the tone and genre of the input movies. This indicates that the SVD model successfully captured the latent features of the input movies and provided relevant recommendations based on genre similarity.

**For 5 Input Movies:**

Recommendations:

* 10 Things I Hate About You (1999)
* The Proposal (2009)
* How to Lose a Guy in 10 Days (2003)
* The Devil Wears Prada (2006)
* Mean Girls (2004)

Observation:  
With more diverse input movies, the recommendations still leaned heavily toward romantic comedies and teen dramas. While the system provided consistent results with high relevance, it showed limited adaptability to the broader range of genres present in the input.

**For 10 Input Movies:**

Recommendations:

* 10 Things I Hate About You (1999)
* The Proposal (2009)
* Easy A (2010)
* How to Lose a Guy in 10 Days (2003)
* The Devil Wears Prada (2006)

Observation:  
Despite the significantly larger and more varied input set, the recommendations remained focused on romantic comedies and teen-focused movies. This suggests that movies with overlapping genres and themes received higher cumulative similarity scores, dominating the results and limiting the diversity of recommendations.

**Suggested Improvements for the SVD Model**

1. **Implement Diversity Constraints:** Prevent the system from overly favoring specific genres or themes, particularly when dealing with larger input sets. This could involve enforcing a minimum representation of underrepresented genres.
2. **Adjust Similarity Aggregation:** Experiment with weighting strategies to give equal importance to all input movies, ensuring that less dominant genres contribute meaningfully to the final recommendations.

**Experiment 3 – KNN and SVD models comparison on 2 movies**

Do when the ui is ready – with photos

1. **Conclusions**

**Popcorn successfully addresses the "Movie Night Dilemma" by providing a recommendation system tailored for groups. By combining collaborative filtering techniques with an intuitive interface, it offers a unique solution that enhances the movie selection experience.**

**Future enhancements could include:**

* **Adding support for more than two users.**
* **Incorporating additional filtering options (e.g., genre, release year).**
* **Expanding the database to include more movie metadata.**

After experiment 3

**Code Overview**

**Files Description**

1. **app.py**: Handles the Flask server, routes, and API endpoints.
2. **logic.py**: Implements the core recommendation logic and TMDb API integration.
3. **index.html**: Provides the structure for the web interface.
4. **script.js**: Manages frontend interactivity, including dynamic suggestions and carousel updates.
5. **styles.css**: Defines the styles and themes for the web interface.

**Important Functions**

1. **recommend\_for\_two\_users\_with\_five\_results**: Generates five movie recommendations based on input from two users.
2. **get\_movie\_poster\_tmdb**: Fetches movie posters and summaries using the TMDb API.
3. **setupMovieInput** (JavaScript): Manages dynamic movie suggestions and poster updates.
4. **fetchResults** (JavaScript): Fetches recommendations and updates the carousel.

When adding comments do this as well with chat

**Installations and Running Instructions**

**Prerequisites**

* Python 3.8+
* Flask
* Required Python libraries

**Installation Steps**

1. Clone the Repository:

git clone https://github.com/your-repo/popcorn

1. Run the Application:

python app.py

1. Access the Web Interface:

Open a web browser and go to http://127.0.0.1:5000.