Angel Armendariz
Darren Cruz
Belal Skaik
Juan Uriarte
CPSC 481
Professor Mira Kim
11.20.2023

Movie Recommender

Problem Statement

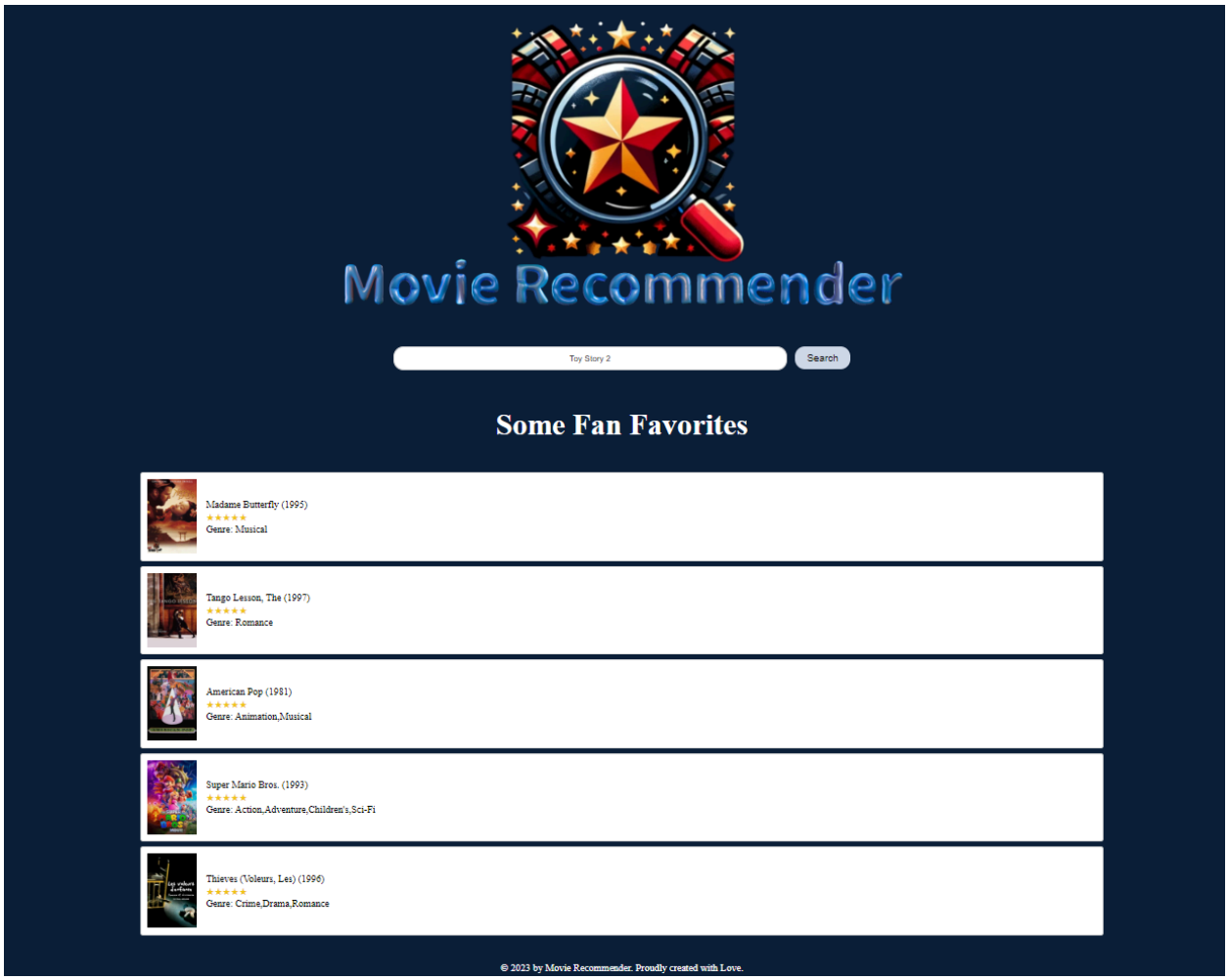
Users often have a problem with finding a new movie to watch to entertain themselves while surfing the web. They often know movies they previously enjoyed and want similar movies recommended. We propose a solution to this problem of not knowing what to watch, a web application that recommends the next movie a user should watch. The user will be able to submit movies. Then based on the movie's attributes, we will be using K Nearest Neighbors to determine movies that should be recommended to the user.

Approach

Our approach to developing the WhatsNext web application was multifaceted, incorporating several programming languages and libraries. We chose Python for its robust data handling and machine learning capabilities, along with HTML, CSS, and JavaScript for the front-end to create an interactive user experience. The core algorithm employed is K-Nearest Neighbors (KNN), typically used in supervised learning but adapted here for unsupervised recommendations based on movie genres. Key Python libraries included Scikit-Learn for implementing the KNN algorithm, Fuzzywuzzy for fuzzy string matching to enhance search accuracy, and Pandas for data manipulation.

The MovieLens 1M Dataset provided a substantial foundation of movie data to work with. The dataset consisted of 1 million movie ratings from 6 thousand users and 4000 movies. The dataset involved three csv files which are movies.csv, ratings.csv, and users.csv. The movies.csv consisted of a MovieId, Title, and Genre. The ratings.csv consisted of a UserId, MovieId, and Rating. Lastly, the user.csv consisted of a UserId, Genre, Age, Occupation, and Zip Code.

Description of Software



Front-End

WhatsNext's interface is crafted with **HTML**, **CSS**, and **JavaScript**, offering a user-friendly experience for movie selection and viewing recommendations while also featuring a dynamic retrieval of movie posters from the **TMDB API**.

Back-End

Powered by **Python** and the **Flask** framework, the back-end efficiently manages web requests and dynamic content delivery. It supports the front-end with an unsupervised genre-based movie recommendation engine using the **K-Nearest Neighbors (KNN)** algorithm. We utilized the **fuzzywuzzy** Python library to enhance our search

functionality. Specifically, we employ the process module from fuzzywuzzy to implement fuzzy string matching, allowing for a robust and fault-tolerant search feature.

Machine Learning

Powered by Python and using the recommendation system based on **K-Nearest Neighbors (KNN)**, the code starts by importing necessary libraries and initializes a Flask framework. Then it loads the movie and ratings from the CSV files into pandas for processing. Once the movie data goes into preprocessing stage, genres are binarized using the MultiLabelBinarizer from sklearn. As a result, this helps us with the recommendation logic.

Using the **K-Nearest Neighbor (KNN)** to train genre data in order to find movies similar to the user's choice of movie involves using two key functions. The first function is **recommend_movies** which helps us recommend movies similar to the given title by the user. As a result, the user does not have to write the exact title of the movie which provides a more user friendly experience. The second function is **recommend_fan_favorities**, this function allows us to have a list of movies that are recommended and have high ratings by users who have utilized our platform.

Code Layout

- Movie-Recommender
    - data
        - movies.csv - holds movie data
        - ratings.csv - holds movie ratings
        - users.csv - holds data for training the model
    - static
        - movieRecommenderLogoBanner.png - logo image for header
        - placeholder.jpg - placeholder for posters
        - script.js - frontend JS to handle clicks and DOM manipulation
        - styles.css - makes it look all cute
    - templates
        - movieRecommender.html - provides structure for the site
    - dependencies.txt - tells the users how to run the program and dependencies
    - movieRecommender.py - holds the functions for movie recommending
    - README.md - has a little description of the project

Data

These are where the data is stored. Movies is where all the movie data is stored, ratings is where the movie ratings are, and users contain some data used to train the model.

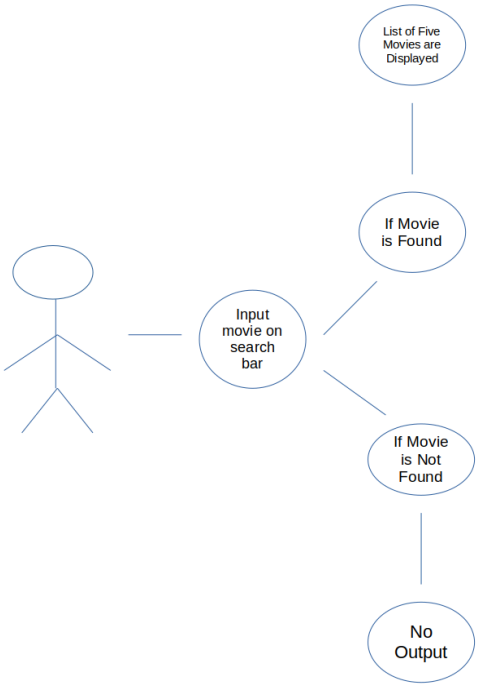
Static

This folder holds all the files for the frontend except for the HTML so the images, css, and JS. This format is needed to use Flask.

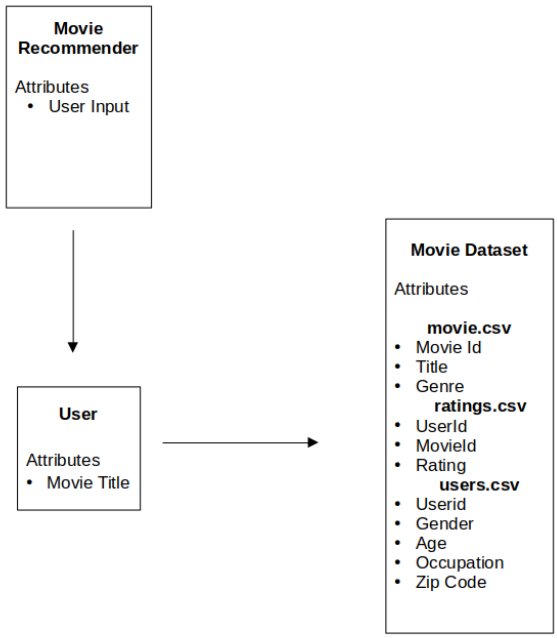
Templates

The templates folder holds the html files for the program, this format is needed to use Flask.

Diagrams

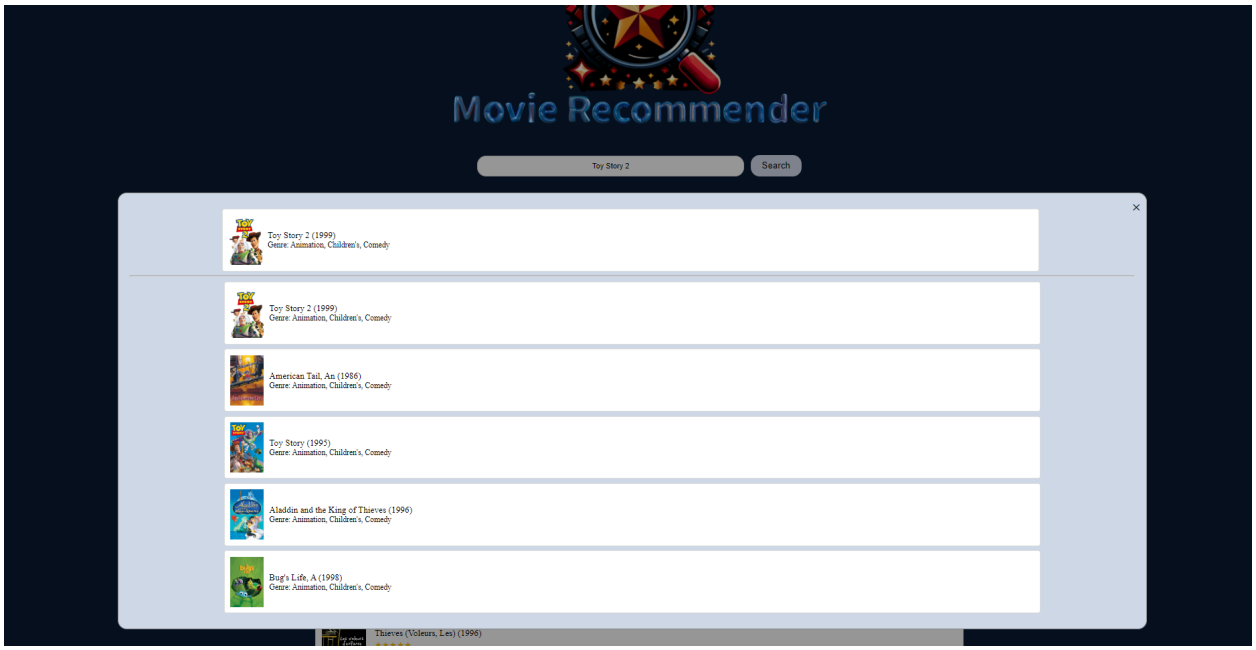


Activity Diagram



Class Diagram

Evaluation Metrics

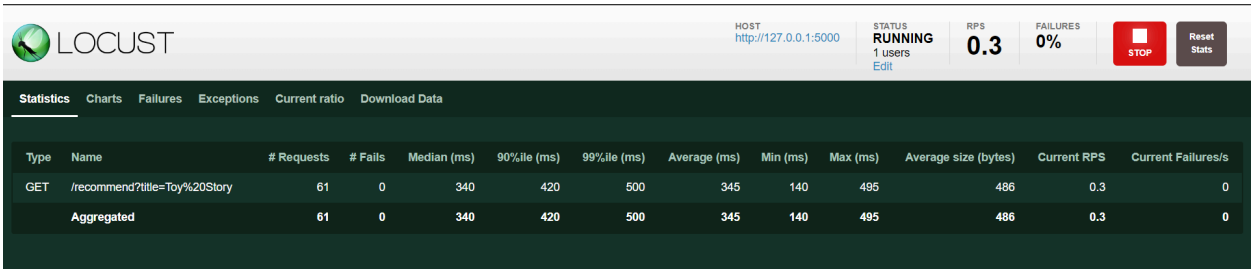


The screenshot above showcases the user interface of our Movie Recommender application. This particular example demonstrates the outcome when 'Toy Story' is used as the search input. The application's response includes 'Toy Story 2', 'An American Tail', 'Aladdin and the King of Thieves', and 'A Bug's Life'. The presence of 'Toy Story 2' in the recommendations list confirms the application's feature of potentially including the search query in the results, validating its ability to recognize and suggest closely related titles. This reflects the application's proficiency in generating relevant and comparable movie suggestions based on user input.

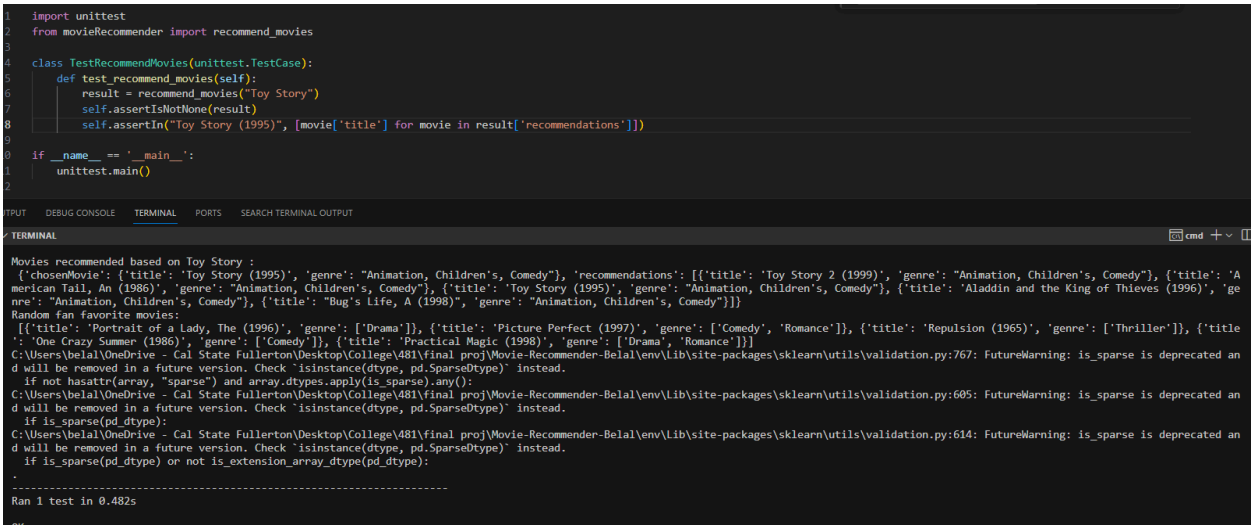
Evaluating the Movie Recommender application involved inputting a movie in the search bar and evaluating the results. When inputting a movie into the search bar, it gave us a list of five movies that are similar to the movie we inputted. For example, when inputting Toy Story we managed to have a list of movies that involved Toy Story 2, American Tail, Aladdin and the King of Thieves, and Bug's Life. All these movies resemble a similarity to Toy Story and as a result we considered this a great news. After evaluating the application further, we noticed that the movie being inputted is sometimes on the list of movies being recommended as shown in the picture below.

Additionally, this is an issue because we don't want the same movie being seen twice, we would like the user to watch movies that are similar to the movie being inputted. We also managed to have a link to the movies that are currently fan favorites so that the

user can look more in detail through other web services. Furthermore, we concluded that the application is working but there are some minor issues that need to be fixed.



The image above displays the Locust load testing tool's statistics dashboard while evaluating the Movie Recommender application. It records the system's response to 61 GET requests made to the endpoint '/recommend?title=Toy%20Story'. The test shows no failures, indicating the application's capability to handle consistent traffic efficiently. The detailed metrics, including median, 90th percentile, and 99th percentile response times, alongside the average, minimum, and maximum response times, provide insights into the application's performance under load. The dashboard's active status, with a low request rate of 0.3 requests per second, indicates that the test is running smoothly on the local server at the address '127.0.0.1:5000'



Conclusion & The Future

One of the key lessons learned in this project is implementing K-Nearest Neighbors in a real world scenario. Being able to use datasets for our needs, in this case a movie recommender system, and to use the data efficiently and effectively has brought us the importance of processing the data correctly. Using our datasets, we also noticed the

importance of title recognition as it would improve our rate of accuracy for our recommendations. The incorporation of the backend and machine learning logic was also a key takeaway to future projects.

Future incorporations of this project is going to include the accessibility for the user to rate a movie based on their opinion. The incorporation of this data can help us further determine what movies to recommend for future users. Another incorporation we would like to add is to have a bigger dataset that contains newer movies. This is important to add because it adds diversity to who is able to use our application. The more users we have, the more data we can collect to further enhance the user's experience. As a result of utilizing a bigger dataset, we would also have to figure out what changes we need to make on the system to enhance performance optimization. Additionally, another incorporation we would like to add is to fully utilize our dataset by incorporating the zip codes and ages as a way to recommend movies to the user when the user is making an account. The user would have to input their zip code and age in order to give them recommendations on movies based on their locality and their age.

References

GroupLens. (n.d.). MovieLens 1M Dataset. Minneapolis, Minnesota, United States of America. Retrieved November 20, 2023, from https://grouplens.org/datasets/movielens/1m/