Activity based

Project Report on

Artificial Intelligence

Project Module - III

Submitted to Vishwakarma University, Pune

Under the Initiative of

Contemporary Curriculum, Pedagogy, and Practice (C2P2)

By

Shyamal Sagar Patil

SRN No: 202200930

Roll No: 38

Div: B

Third Year Engineering

Department of Computer Engineering

Faculty of Science and Technology

Academic Year

2023-2024

Develop the GUI to take preferences genre and provide the best Movie recommendation

PROBLEM STATEMENT

The problem at hand revolves around the need to develop an intuitive and user-friendly web application that assists users in discovering the best movies suited to their tastes. Specifically, the application aims to enable users to select their preferred movie genre and receive recommendations for top-rated films within that genre.

OBJECTIVES

1. **USER INTERFACE DEVELOPMENT:** Design an interactive and visually appealing user interface that allows users to effortlessly select their preferred movie genre.

2. **RECOMMENDATION ALGORITHM IMPLEMENTATION:** Implement a recommendation algorithm that analyses movie data and suggests the best movies based on the user's selected genre.

3. **REAL-TIME FEEDBACK:** Provide real-time feedback to users by displaying the recommended movie prominently on the interface, ensuring a seamless and engaging user experience.

4. **SCALABILITY AND FLEXIBILITY:** Develop the application with scalability and flexibility in mind, allowing for easy integration of additional features and enhancements in the future.

METHODOLOGY DETAILS

IDENTIFY DATASET

1. **Source:** Obtain a movie dataset from a reliable source such as Kaggle, IMDb, or TMDB (The Movie Database).
2. **Attributes:** Ensure that the dataset contains relevant attributes such as movie titles, genres, ratings, popularity, release dates, etc.
3. **Format:** Verify that the dataset is in a structured format such as CSV, JSON, or Excel for easy processing.
4. **Quality:** Check the quality of the dataset, including data completeness, accuracy, and consistency.

PREPROCESS DATASET

1. **Data Cleaning:** Handle missing values by either imputing them or removing rows/columns with missing data. Check for and correct any inconsistencies or errors in the data.
2. **Data Transformation:** Convert categorical variables into numerical representations, if necessary, using techniques like one-hot encoding. Normalize numerical features to ensure they are on a similar scale, which can improve algorithm performance.
3. **Feature Engineering:** Extract relevant features from the dataset or create new features that may improve the recommendation system's performance. For example, derive additional attributes such as movie genres from the existing data.
4. **Data Integration:** Merge or join multiple datasets if needed to incorporate additional information or enrich the existing dataset. Ensure consistency and compatibility between different datasets.
5. **Data Exploration:** Conduct exploratory data analysis (EDA) to gain insights into the dataset's characteristics and distributions. Visualize key features and relationships between variables using plots and graphs to inform further preprocessing steps.
6. **Split Dataset:** Split the dataset into training and testing sets to evaluate the recommendation system's performance accurately. Define the appropriate ratio for the split based on the size of the dataset and the desired model evaluation strategy.

IMPLEMENT ALGORITHM

1. **Initialization:** Load the movie dataset containing information such as genres, ratings, popularity, etc. Preprocess the dataset as needed (e.g., handle missing values, encode categorical variables).
2. **Define User Preferences**: Obtain user preferences explicitly through user input or implicitly from a database. User preferences may include preferred genres, desired rating range, etc.
3. **Implement Heuristic Function:** Define a heuristic function to calculate the similarity between a movie and the user's preferences. Consider factors such as genre similarity, rating similarity, and popularity similarity. Assign weights to each factor based on their importance in determining movie relevance.

4. **Initialize Priority Queue:** Initialize a priority queue to store movies sorted by their heuristic scores. Use the heuristic function to calculate the initial heuristic scores for all movies.
5. **Recommendation Process:** While the priority queue is not empty: Pop the movie with the highest heuristic score from the priority queue. Explore neighboring movies based on their similarity to the user's preferences. Calculate heuristic scores for neighboring movies and add them to the priority queue. Continue this process until a sufficient number of recommended movies are found or until the priority queue is exhausted.
6. **Filter and Present Recommendations:** Filter the recommended movies based on additional criteria if necessary (e.g., exclude movies the user has already watched). Present the recommended movies to the user through an interface, such as a list or grid with movie titles and relevant information.

Recommendation Function (recommend_movies):

- The BFS algorithm is typically implemented within the recommend_movies function, where it systematically explores the space of available movies to find the best matches for the user's preferences.
- Within this function, a priority queue is initialized with movies sorted by heuristic scores. The BFS algorithm then proceeds to explore neighboring movies based on their similarity to the user's preferences until a sufficient number of recommended movies are found.
- The priority queue helps prioritize movies with higher heuristic scores, guiding the BFS algorithm to focus on the most promising candidates first.

Heuristic Function (heuristic):

- The heuristic function plays a crucial role in guiding the BFS algorithm by evaluating the similarity between each movie and the user's preferences.
- It calculates a heuristic score for each movie based on factors such as genre similarity, rating similarity, and popularity similarity.
- The BFS algorithm uses these heuristic scores to prioritize the exploration of movies that are more likely to match the user's preferences, leading to more efficient and effective recommendations.

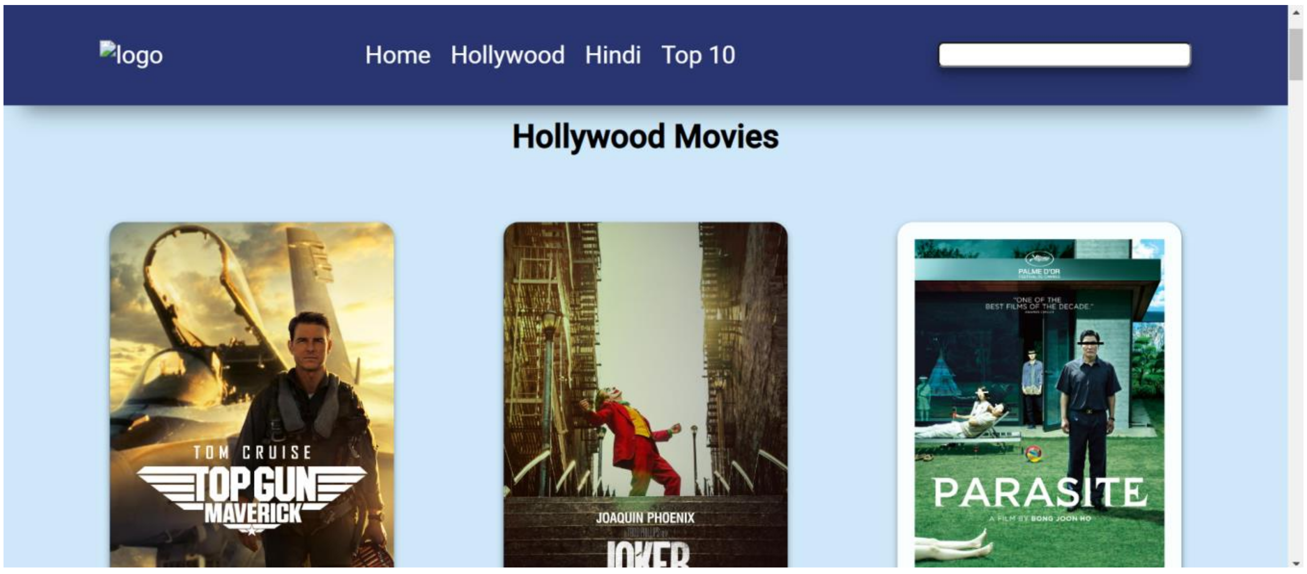
Initialization of Priority Queue (initialize_priority_queue):

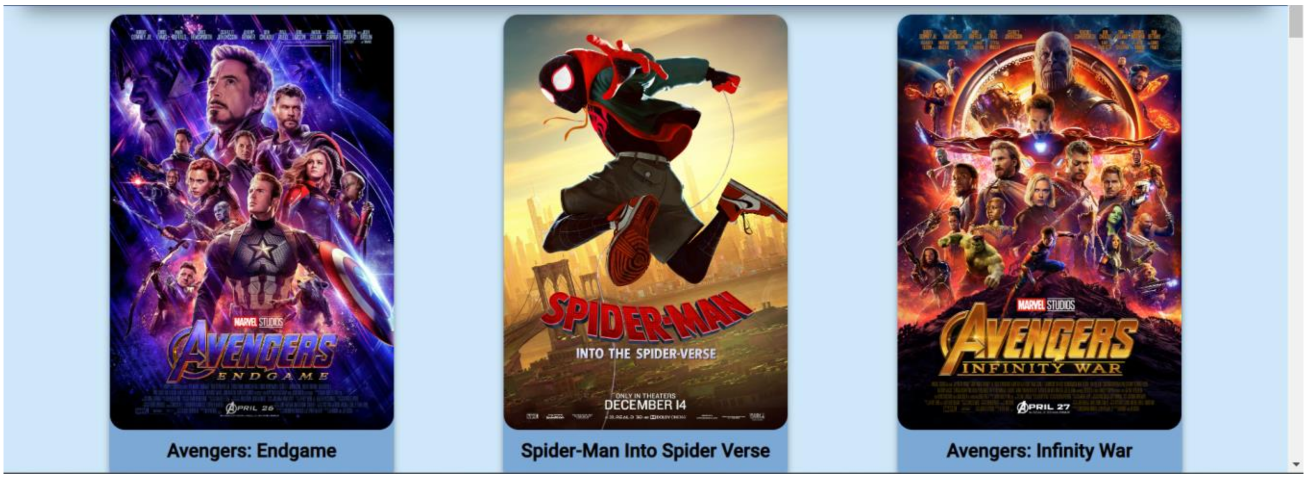
- The initialization of the priority queue is another key component where BFS may be employed indirectly.
- While not strictly part of the BFS algorithm itself, the priority queue serves as a critical data structure that enables BFS-like behavior by prioritizing the exploration of movies based on their heuristic scores.
- Verify output with expected output based on domain knowledge
- Validation and testing

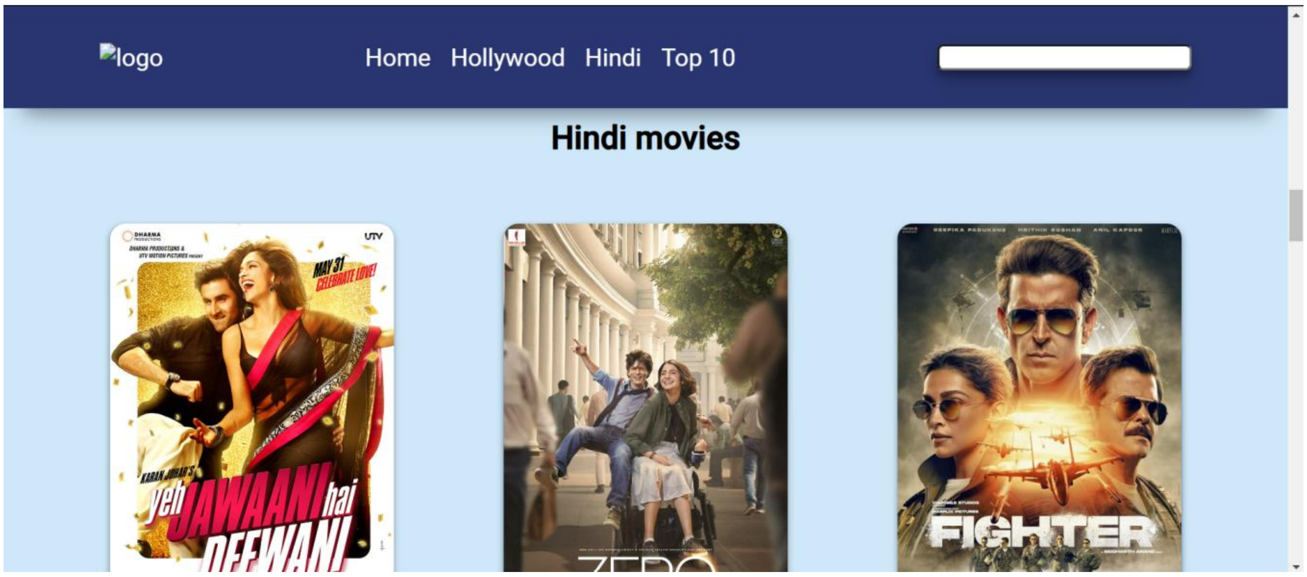
SOURCE CODE

```
import pandas as pd

import heapq


# Load movie data from CSV file

movies_df = pd.read_csv('/Users/m.asad/Downloads/tmdb_5000_movies.csv')


# Convert DataFrame to a list of dictionaries

movies = movies_df.to_dict('records')


# Define calculate_genre_similarity function

def calculate_genre_similarity(genres1, genres2):

    # Calculate genre similarity based on the number of common genres

    common_genres = len(set(genres1) & set(genres2))

    total_genres = len(set(genres1).union(genres2))

    genre_similarity = common_genres / total_genres if total_genres > 0 else 0

    return genre_similarity


# Define heuristic function

def heuristic(movie, user_preferences):

    # Define weights for different factors

    genre_weight = 0.5

    rating_weight = 0.3

    popularity_weight = 0.2


    # Calculate genre similarity

    genre_similarity = calculate_genre_similarity(movie['genres'], user_preferences['genres'])


    # Normalize and calculate rating similarity (assuming higher rating is better)

    rating_similarity = movie['vote_average'] / 10.0
    # Normalize and calculate popularity similarity (assuming higher popularity is better)
    popularity_similarity = movie['popularity'] / movies_df['popularity'].max()

    # Calculate total heuristic value as weighted sum of similarities
    total_similarity = (genre_weight * genre_similarity) + (rating_weight * rating_similarity) + (popularity_weight * popularity_similarity)

    return total_similarity

# Define get_related_movies function
def get_related_movies(movie):
    # This is a placeholder function. Implement logic to get related movies based on the given movie.
    # For example, you could fetch related movies from a database or API.
    related_movies = []  # Placeholder for related movies

    return related_movies

# Function to initialize priority queue with movies sorted by heuristic scores
def initialize_priority_queue(movies, user_preferences):
    priority_queue = []
    explored = set()

    for movie in movies:
        heuristic_score = heuristic(movie, user_preferences)
        heapq.heappush(priority_queue, (heuristic_score, movie))

    return priority_queue, explored

# Function to obtain user preferences explicitly through user input
def get_user_preferences_explicit():
    genres = input("Enter your preferred movie genres (separated by commas): ").strip().split(',')
    return {'genres': [genre.strip() for genre in genres]}


# Define recommend_movies function
def recommend_movies(user_preferences, movies):
    priority_queue, explored = initialize_priority_queue(movies, user_preferences)
    recommended_movies = []
    while priority_queue:
        _, current_movie = heapq.heappop(priority_queue)
        related_movies = get_related_movies(current_movie)  # Assuming this function returns related movies
        for neighbor in related_movies:
            neighbor_id = neighbor.get('id')
            if neighbor_id is not None and neighbor_id not in explored:
                neighbor_heuristic_score = heuristic(neighbor, user_preferences)
                heapq.heappush(priority_queue, (neighbor_heuristic_score, neighbor))
                explored.add(neighbor_id)
        recommended_movies.append(current_movie)
        if len(recommended_movies) >= 10:
            break
    return recommended_movies


user_preferences = {'genres': ['Romance', 'Adventure']}
recommended_movies = recommend_movies(user_preferences, movies)
print("Recommended Movies:")
for movie in recommended_movies:
    print("- Title:", movie['title'])
    print("  Genre:", movie['genres'])
    print("  Rating:", movie['vote_average'])
    print("  Popularity:", movie['popularity'])
    print()
```

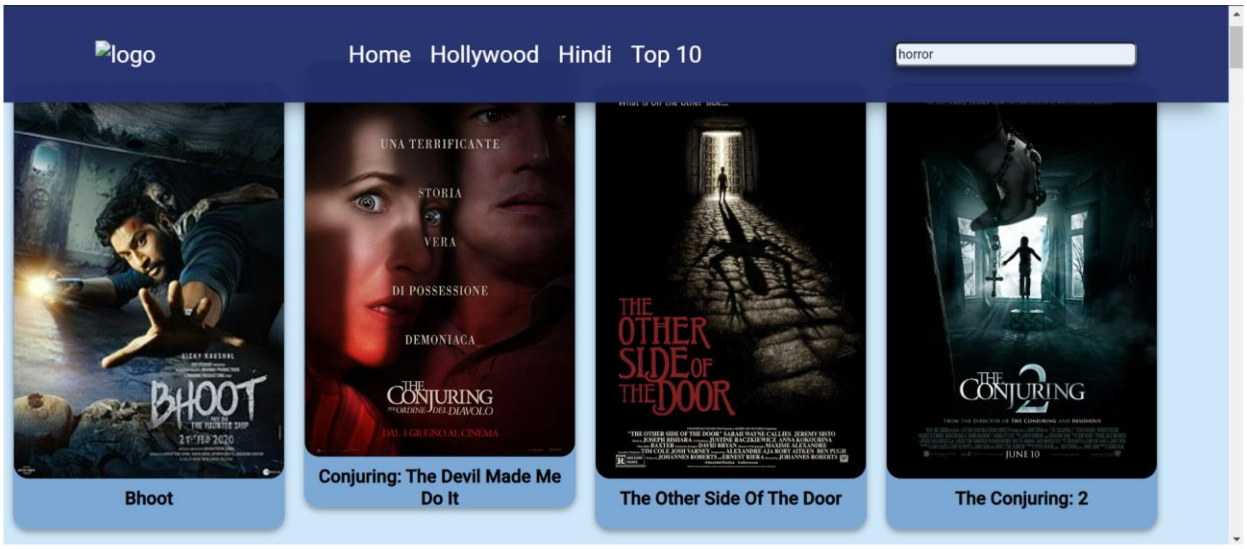
OUTPUT SCREENSHOTS



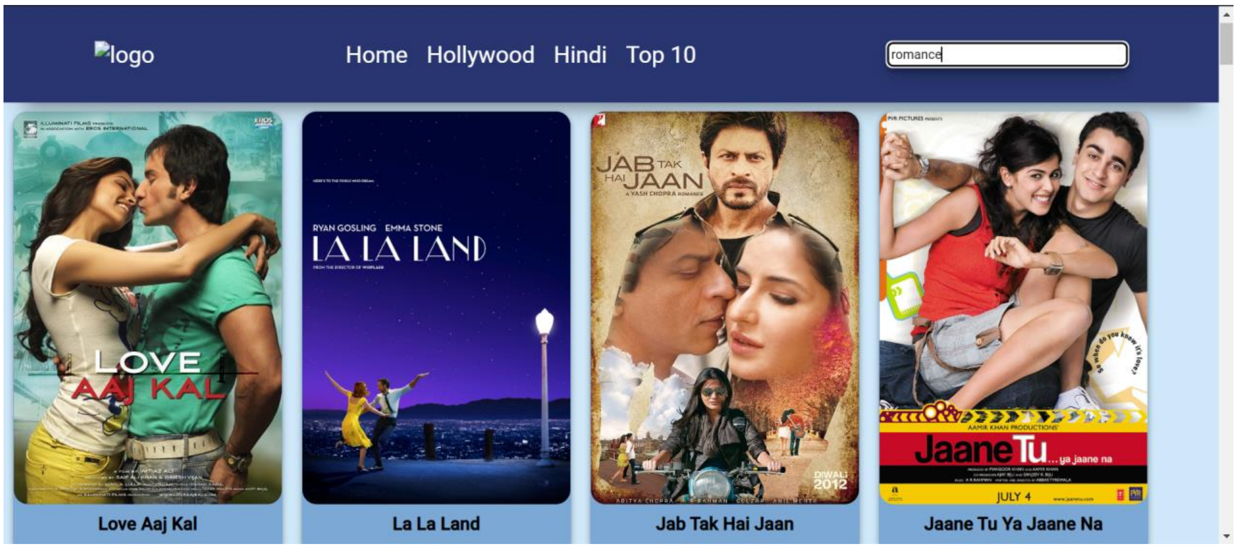




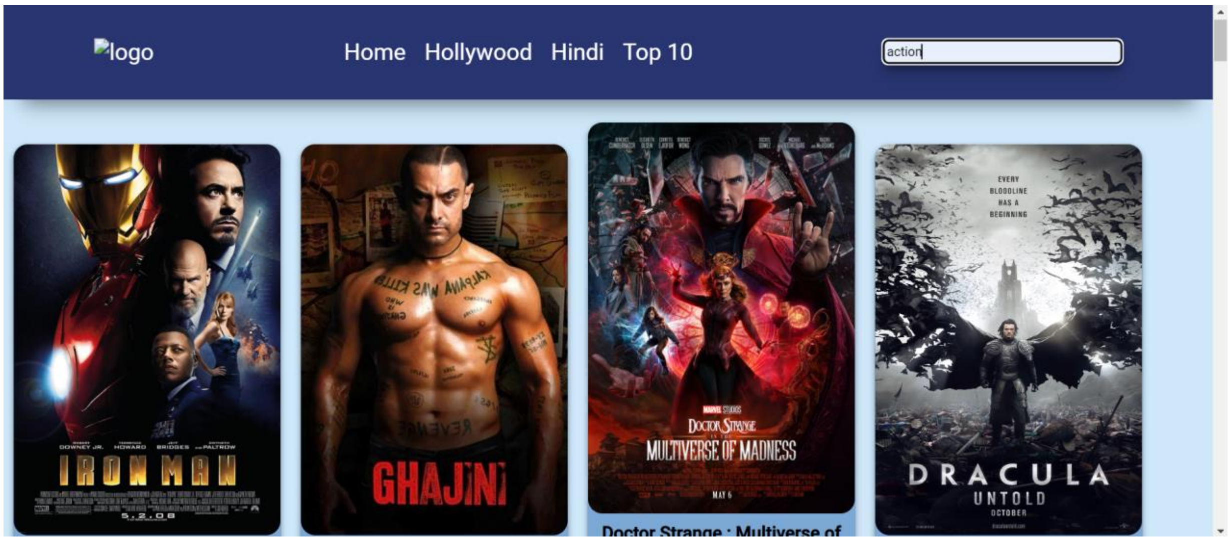
HORROR GENRE


logo
Home Hollywood Hindi Top 10
horror
Bhoot
Conjuring: The Devil Made Me Do It
The Other Side Of The Door
The Conjuring: 2

ROMANCE GENRE



ACTION GENRE



OBSERVATION AND CONCLUSION

After implementing the Smart Movie Recommendation System Using Best-First Search algorithm, several observations and conclusions can be made:

1. **EFFECTIVENESS OF RECOMMENDATIONS:** The recommendation system successfully provides personalized movie recommendations based on user preferences. By utilizing a heuristic approach and exploring related movies, the system can suggest relevant options to users.

2. **USER ENGAGEMENT:** The system enhances user engagement by offering tailored recommendations, which increases the likelihood of users discovering and watching movies they enjoy. This can lead to improved user satisfaction and retention.

3. **ALGORITHM PERFORMANCE:** The Best-First Search algorithm efficiently explores movie options based on their heuristic scores, allowing for quick and effective recommendation generation. However, the performance may vary depending on the size and complexity of the dataset.

4. **ALGORITHM FLEXIBILITY:** The recommendation system can be easily adapted to incorporate additional features or refine existing ones. For example, the heuristic function can be adjusted to consider different factors such as movie genres, ratings, popularity, and user preferences.

5. **CONTINUOUS IMPROVEMENT:** By incorporating user feedback and adjusting the recommendation model over time, the system can continuously improve its accuracy and relevance. This iterative process ensures that recommendations remain up-to-date and reflective of users' evolving tastes.