VISVESVARAYA TECHNOLOGICAL UNIVERSITY
JNANASANGAMA, BELAGAVI–590018

A Mini Project Report on

"Movie Recommendation System"

Submitted in partial fulfillment for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

Submitted by

POOJA N M

1AT22MC072

Under the guidance of

Mr. Vasanth C Bhagawat

Assistant Professor

Department of MCA

Atria Institute of Technology

Department of MCA

2023–2024

ATRIA INSTITUTE OF TECHNOLOGY

(Affiliated to Visvesvaraya Technological University, Belgavi)

Anandanagar, Bengaluru-560024

CERTIFICATE

This is to certify that the mini project report entitled **“MOVIE RECOMMENDATION SYSTEM”** is submitted by “**POOJA N M (1AT22MC072)”** in partial fulfillment of Master of Computer Applications course of ***VISVESVARAYA TECHNOLOGICAL UNIVERSITY***, ***BELGAVI*** as a record of Bonafide work done by him /her under our supervision and guidance.

Guide:	HOD:
Vasanth C Bhagawat	Dr.Mamatha T
Assistant Professor	Professor and HOD
Department of MCA	Department of MCA

Internal Examiner	**External Examiner**

Date:

Table of Contents

CHAPTER-1

Introduction

A movie recommendation system in Python leverages collaborative filtering techniques to suggest films tailored to individual preferences. By analyzing user interactions with movies, such as ratings or viewing history, these systems identify patterns of similarity between users or items (movies), allowing for accurate predictions of user preferences. Using Python libraries like Surprise or scikit-learn, developers can build and train recommendation models on datasets like MovieLens, facilitating the creation of personalized movie recommendations. Through continuous evaluation and optimization, these systems enhance user experience by delivering relevant and engaging movie suggestions across various platforms, from web applications to mobile interfaces.

Movie recommendation system, we leverage advanced algorithms to analyze user preferences and movie attributes. Utilizing Pandas for data manipulation, our system processes large datasets containing user reviews and film details. By employing collaborative or content-based filtering techniques, we aim to deliver personalized movie suggestions, enhancing user satisfaction and engagement. The project prioritizes algorithm performance, data scalability, and user-friendly integration, ensuring an effective and enjoyable movie discovery experience

1.1 General description of the area of study

The area of study concerning movie recommendation systems sits within the broader field of recommender systems, a subset of artificial intelligence and information filtering techniques. Recommender systems aim to predict user preferences or interests and offer personalized recommendations accordingly. In the case of movie recommendation systems, algorithms analyze user behavior, such as ratings, reviews, or past viewing history, to suggest films that align with individual tastes. These systems play a pivotal role in various domains, including e-commerce, entertainment, and content streaming platforms, enhancing user satisfaction and engagement by delivering relevant and tailored suggestions. Key techniques employed in movie recommendation systems include collaborative filtering, content-based filtering, and hybrid methods, each offering unique approaches to capturing and leveraging user preferences and item characteristics. As technology evolves, advancements such as deep learning and contextual modeling contribute to further enhancing the accuracy and effectiveness of movie recommendation systems, shaping the future of personalized content discovery experiences.

1.2 Problem analysis

the system should leverage natural language processing (NLP) techniques to extract relevant features and characteristics. Subsequently, it should compare the features with the attributes of other movies in the dataset to identify films with similar content. Additionally, the recommendation system should incorporate user feedback to continuously refine and improve its recommendations over time.

1.3 Purpose of study

The objective of Movie Recommendation System using python is

1. Enhance user satisfaction by providing personalized movie recommendations tailored to individual preferences, improving the overall user experience.
2. Increase user engagement on the platform by offering relevant and intriguing movie suggestions, encouraging users to explore and discover new content.
3. Evaluate and optimize the performance of recommendation algorithms (collaborative filtering, content-based filtering, or hybrid approaches) to ensure accurate and effective movie suggestions.
4. Efficiently handle and preprocess large datasets containing user reviews, movie details, and other relevant information using Python and Pandas, ensuring the system's scalability and reliability.
5. Implement additional features such as user feedback mechanisms, diversity in recommendations, and real-time updates to further refine and enhance the recommendation system.
6. Employ appropriate evaluation metrics (e.g., precision, recall, and F1-score) to assess the effectiveness of the recommendation system and continuously refine algorithms for optimal performance.

CHAPTER-2

Requirement Analysis

2.1 Historical background

Over the years, many recommendation systems have been developed using either collaborative, content based or hybrid filtering methods. These systems have been implemented using various and machine learning algorithms.

Content-based filtering is a recommendation technique that suggests items (movies in this case) based on their features and the user's preferences. In this script, content-based filtering is implemented using the TF-IDF vectorizer and cosine similarity.

TF-IDF is a numerical statistic that reflects the importance of a word in a document relative to a collection of documents (corpus). It is commonly used for text mining and information retrieval. In this script, the TF-IDF vectorizer is applied to the movie overviews ('soup' column) to convert them into a numerical representation. Reference Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. Information Processing & Management, 24(5), 513-523.

Cosine similarity is a measure of similarity between two non-zero vectors in an inner product space that measures the cosine of the angle between them. In this script, cosine similarity is calculated between the TF-IDF vectors of different movies to determine their similarity. Reference Salton, G., Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. Communications of the ACM, 18(11), 613-620.

The script lacks an explicit evaluation of the recommendation system's performance. Metrics such as precision, recall, or user studies could be incorporated for a more comprehensive evaluation. Reference:Ricci, F., Rokach, L., & Shapira, B. (2011). Introduction to Recommender Systems Handbook. In Recommender Systems Handbook (pp. 1-35). Springer.

Future work could involve incorporating user feedback, collaborative filtering, hybrid recommendation systems, and handling larger datasets. Reference: Adomavicius, G., & Tuzhilin, A. (2005). Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Transactions on Knowledge and Data Engineering, 17(6), 734-749.

2.2 Data Collection/ Characteristics of Data Set

To build our recommendation system, we need a comprehensive dataset containing information about movies. The dataset should include details such as movie titles, genres, cast, crew, release dates, and user ratings. Datasets like MovieLens or The Movie Database (TMDb) provide such information and can be used for training and testing our recommendation model.

CHAPTER-3

Methodology

3.1 Tools and Technology

Tools and Technologies included that the python libraries like

Pandas

scikit-learn

TF-IDF

3.2 Hardware and software

Software requirements

Operating System - Windows or Linux or FreeBSD or MacOS

• Platform - x86 – 64

• Language - Python

• Libraries - pandas, scikit-learn

Hardware requirements

• RAM - 512 GB

• Hard disk capacity - 8 GB Minimum

3.3 Features of Data set.

The features of the dataset for a content-based movie recommendation system focusing on similar content and release
date would include:

Title: The title of the movie.

Genres: The genre(s) of the movie (e.g., action, comedy, drama, etc.).

Plot Summary: A brief summary or description of the movie's plot.

Release Date: The release date of the movie.

These features are essential for capturing the content and release date information necessary for recommending similar movies. Each feature provides valuable information that can be used to calculate the similarity between movies and prioritize recommendations based on the release date. It's important to ensure that the dataset is comprehensive and accurately represents the attributes of each movie to build an effective recommendation system.

3.4 Algorithms used

Text Feature Extraction using TF-IDF: Utilize the Tfidf Vectorizer from scikit-learn to convert movie overviews into TF-IDF (Term Frequency-Inverse Document Frequency) vectors. This step transforms textual data into numerical vectors, capturing the importance of words in the overviews

Cosine Similarity Matrix Construction: Calculate cosine similarity between movie TF-IDF vectors to create a similarity matrix. This matrix reflects the pairwise similarity scores between movies based on their overviews

CHAPTER-4

Analysis and Design

4.1 Data Presentation and Analysis.

Our goal is to create a movie recommendation system that provides personalized recommendations to users. This system aims to increase user engagement, satisfaction, and the platform's overall retention rate. The recommendation engine will utilize content-based filtering, focusing on movie features such as genres, actors, and user preferences. Steps in the Development Process

1. Data Collection

To build our recommendation system, we need a comprehensive dataset containing information about movies. The dataset should include details such as movie titles, genres, cast, crew, release dates, and user ratings. Datasets like MovieLens or The Movie Database (TMDb) provide such information and can be used for training and testing our recommendation model.

2. Data Preprocessing

Once the dataset is obtained, we need to clean and preprocess the data. This involves handling missing values, converting categorical variables into a suitable format, and creating features that can be used by the recommendation algorithm. For example, we might create a "soup" feature by combining relevant textual information about the movie.

3. Feature Engineering

Feature engineering is a crucial step in building an effective recommendation system. We may extract features such as movie genres, director names, and actor names from the dataset. Additionally, we can use techniques like TF-IDF to transform textual data into numerical vectors.

4. Model Selection

For this case study, we will use a content-based filtering approach. We'll employ algorithms like TF-IDF and cosine similarity to find movies similar to the ones users have liked in the past. These algorithms focus on the content of the items rather than collaborative filtering, which relies on user interactions.

5. Training the Model

The model will be trained on the preprocessed dataset, learning the relationships between movie features and user preferences. Training involves creating a TF-IDF matrix, computing cosine similarity, and developing a mechanism to generate recommendations based on user input.

6. Building the Recommendation System

The recommendation system will be integrated into a user interface, and a web application will be developed using Flask, a micro web framework for Python. The application will allow users to input their preferences, and the system will provide personalized movie recommendations.

7. Evaluation

To assess the performance of the recommendation system, various metrics can be considered, such as precision, recall, and mean squared error. User feedback and A/B testing can also be valuable in understanding how well the system meets user expectations

4.2 Data Interpretation.

English Movies Distribution:

Analyzed the distribution of movies based on language, specifically focusing on English-language movies. Identified the proportion of English movies within the dataset compared to movies in other languages.

Genre Analysis for English Movies:

Examined the distribution of genres within English-language movies. Identified popular genres among English movies and any unique trends or patterns compared to movies in other languages.

Release Year Trends for English Movies:

Investigated the release year trends specifically for English-language movies. Observed any notable patterns or shifts in the production of English movies over time.

User Engagement with English Movies:

Analyzed user engagement metrics, such as the number of ratings provided, specifically for English-language movies. Explored whether users exhibit different engagement behaviors when interacting with English movies compared to movies in other languages.

CHAPTER-5

Implementation

5.1 Data Cleaning.

Date Parsing and Standardization:

Parsed release date information from the dataset and standardized date formats to a consistent format (e.g., YYYY-MM-DD). Extracted relevant information such as year, month, and day from the release date for analysis and modeling purposes.

Handling Missing Values:

Identified and addressed any missing values in the datasets, particularly in columns such as movie titles, user IDs, ratings, genres, and release years. Used appropriate techniques such as imputation (e.g., filling missing values with mean, median, or mode) or deletion of incomplete records.

Removing Duplicates:

Checked for and removed any duplicate entries in the datasets, ensuring each movie or user appears only once to avoid skewing the analysis or recommendations.

Dealing with Inconsistencies:

Addressed inconsistencies in data formats, such as inconsistent naming conventions for genres or movie titles. Standardized naming conventions or corrected inconsistencies to ensure uniformity across the dataset.

5.2 Handling Missing Data

1. Identify Missing Values: Use descriptive statistics or visualization techniques to identify missing values in the dataset. Check for missing values in important columns such as user IDs, movie IDs, ratings, genres, and release years.

2. Imputation Techniques: Mean/Median Imputation: Replace missing numerical values with the mean or median of the respective column. This is suitable for numerical data like ratings or release years, Mode Imputation: Replace missing categorical values with the mode (most frequent value) of the respective column, such as genres.

3. Forward/Backward Fill: Use the value from the previous or next row to fill missing values in time-series data.

4. Interpolation: Use interpolation methods like linear or polynomial interpolation to estimate missing values based on surrounding data points.

5. Domain-specific Imputation: For movie release years, consider imputing missing values based on the average release year for movies of similar genres or directors, For movie ratings, consider imputing missing values based on the average ratings given by users with similar viewing patterns or preferences.

6. Delete Rows or Columns: If missing values are too numerous or cannot be reliably imputed, consider deleting rows or columns containing missing values. Be cautious with this approach, as it may result in loss of valuable information.

7. Advanced Techniques: Utilize advanced imputation techniques such as multiple imputation, which generates multiple imputed datasets to account for uncertainty in missing data.

5.3 Data Modeling

Data modeling is a crucial step in building a movie recommendation system as it involves designing the algorithm or model that will generate recommendations based on the available data. Here's a high-level overview of the data modeling process for a movie recommendation system:

1. Choose a Modeling Approach:

Decide on the modeling approach based on the characteristics of the dataset and the requirements of the recommendation system. Common approaches include collaborative filtering, content-based filtering, and hybrid methods.

2. Data Representation:

Represent the dataset in a suitable format for modeling. For collaborative filtering, this typically involves constructing a user-item interaction matrix where rows represent users, columns represent items (movies), and the cells contain ratings or interaction scores.

3. Splitting Data:

Split the dataset into training and testing sets to evaluate the performance of the recommendation model. Common splitting techniques include random splitting or time-based splitting for temporal datasets.

4. Selecting a Model:

Choose an appropriate recommendation model based on the selected approach. For collaborative filtering, popular models include:

5. User-Based Collaborative Filtering:

Recommends items based on similarities between users.
Item-Based Collaborative Filtering: Recommends items based on similarities between items.
Matrix Factorization Techniques: Decompose the user-item interaction matrix into latent factors to capture user and item preferences.

6. Model Training:

Train the selected recommendation model using the training data. This involves learning the parameters or latent factors that best capture the underlying patterns in the user-item interactions.

CHAPTER-6

Result Obtained

6.1 Findings of the Analysis

User Interaction:

- Users access the web application and enter the name of a movie they enjoyed.
- The recommendation system processes the input, calculates similarity scores, and returns user's input.

Release Year Trends:

- There is a noticeable increase in the number of movies produced in recent years, indicating growth in the film industry.
- Older movies tend to have fewer ratings compared to more recent releases, suggesting that user engagement may be biased towards newer content.

Collaborative Filtering Patterns:

- Collaborative filtering algorithms show promise in predicting user preferences based on historical ratings and user-item interactions.
- Similar users tend to have similar preferences, as evidenced by the accuracy.

6.2 Data Visualization

Rating Distribution Histogram:

Visualize the distribution of movie ratings using a histogram to understand the distribution of ratings given by users.

Genre Distribution Bar Chart:

Plot a bar chart showing the distribution of movie genres to identify the most popular genres among users.

Heatmap of Rating Matrix:

Generate a heatmap representing the user-item rating matrix to visualize the sparsity ratings and identify patterns in user-item interactions.

Scatter Plot of Movie Ratings vs Release Year:

Create a scatter plot with movie release year on the x-axis and average ratings on the y-axis to explore how movie ratings vary over time.

Word Cloud of Movie Titles:

Generate a word cloud visualization of movie titles to identify common words or phrases and gain insights into popular themes or topics.

Network Visualization of Similar Movies:

Create a network visualization to illustrate the relationships between similar movies based on collaborative filtering recommendations.

HTML Structure:
<!DOCTYPE html>: Declares the document type and version of HTML being used.
<html lang="en">: Defines the root element of the HTML document with the language attribute set to English.
<head>: Contains meta-information about the HTML document, such as character set, title, and viewport settings.

<meta charset="UTF-8">: Specifies the character encoding for the document (UTF-8 in this case).
<title>: Sets the title of the webpage that appears in the browser tab.
<meta name="viewport" content="width=device-width, initial-scale=1.0">: Configures the viewport for better responsiveness on different devices.
<link>: Includes external stylesheets, such as the Normalize.css and Google Fonts stylesheets.
CSS Styles:
:root: Defines global CSS variables that can be used throughout the stylesheet.
@media: Defines media queries for different screen widths, adjusting the styling based on the device's screen size.
body: Styles the overall appearance of the webpage, including background color, font, and spacing.
.movie: Styles the mov ie-related elements, such as the form, input fields, and buttons.
#movie_name, #submission_button: Styles specific HTML elements with the corresponding IDs.
h2: Styles level 2 headings (movie recommendation system title).
.creds: Styles the paragraph providing instructions or information.
Form and Input Elements:
<form>: Creates a form to take user input.
<input>: Defines input fields, such as text input and buttons, with specific attributes for styling and functionality.
action and method attributes in the <form> tag specify where the form data is sent and the HTTP method used for the request (in this case, a POST request to the '/main' route).
JavaScript Libraries:
<script>: Includes external JavaScript libraries (jQuery and Anime.js) using content delivery networks (CDNs).

BACK END :

1. TF-IDF Vectorization:
The script uses the TfidfVectorizer from scikit-learn to transform the movie descriptions into a TF-IDF matrix. This matrix represents the importance of each word in each movie's description.
tfidf_matrix = tfidf.fit_transform(df2['soup']): This line fits the TF-IDF vectorizer on the 'soup' column of the DataFrame, which likely contains preprocessed textual information about the movies.
2. Cosine Similarity:
The script calculates the cosine similarity between movies based on the TF-IDF matrix.
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix): Computes the cosine similarity between each pair of movies, resulting in a similarity matrix.
3. Recommendation Function:
The get_recommendations function takes a movie title as input, retrieves the index of that movie,
and then finds the most similar movies based on cosine similarity scores.
It prints the movieId and score pairs for the top similar movies.
It constructs a DataFrame (return_df) with information about the recommended movies (titles, homepages, and release dates).
4. Flask Web Application:
The Flask web application has two routes:
'/': Handles both GET and POST requests. For a GET request, it renders the main template ('index.html'). For a POST request, it processes the form input, checks if the entered movie title exists, and either renders a 'notFound.html' template or displays recommended movies using the 'found.html' template.
'main' (implicitly handled by the @app.route('/', methods=['GET', 'POST'])): This is the main route where the movie recommendation logic is implemented

CHAPTER-7

Conclusion and Future Scope

Movie recommendation system leverages a content-based filtering approach to provide personalized movie suggestions to users. By employing the Term Frequency-Inverse Document Frequency (TF-IDF) technique and cosine similarity, the system analyzes the textual content of movies, capturing their inherent features and relationships. The Flask web framework facilitates a user-friendly interface for input and output, enhancing the system's accessibility. The recommendation system successfully retrieves and ranks movies based on their similarity to the user's input. The inclusion of additional features such as movie homepages and release dates enriches the user experience, providing comprehensive information about the recommended movies. Despite its effectiveness, the system may benefit from future enhancements, such as incorporating user preferences, collaborative filtering, or hybrid approaches to further refine recommendations. Additionally, continuous updates to the movie database and improvements in the recommendation algorithm can contribute to the system's accuracy and relevance over time. Overall, this movie recommendation system serves as an interactive and informative tool for users seeking tailored movie suggestions based on content similarity, contributing to an enhanced entertainment experience.

Bibliography

1. https://ieeexplore.ieee.org/document/6382910/authors#authors.
2. https://www.kaggle.com/code/rounakbanik/movie-recommender-systems
3. https://github.com/asif536/Movie-Recommender-System.