

Minor Project Report On

How to Build a Content-Based Recommendation System

Submitted in partial fulfillment of the requirements for the
minor project [ARP 455]

Name- Vedant Roy

Enrollment Number- 12019011621

**Under the Supervision of Dr. Amar Arora, Assistant
Professor, USAR**



**UNIVERSITY SCHOOL OF AUTOMATION AND ROBOTICS
GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY
EAST DELHI CAMPUS, SURAJMAL VIHAR, DELHI- 110032**

Table of Contents

S.no.	Chapters	Pg no.
1	Abstract	2
2	Introduction	3
3	Hardware and Software Requirements	4
4	Problem Statement	6
5	Related work (Literature Survey)	8
6	Major Modules of Project	10
7	Snapshot of portal designed/ stimulation performed (till now).	12
8	References	14

List of Figures- In Order

1. Fig. 6.1 Screenshot of the movie recommender web interface
2. Fig. 6.2 Screenshot of the Front-End Code of Movie Recommender System

Abstract

The Movie Recommender System is a cutting-edge project leveraging machine learning techniques to deliver tailored movie recommendations to users based on their preferences. This project is built upon the TMDb 5000 Movies dataset, a comprehensive dataset containing information such as movie genres, keywords, cast, crew, budget, revenue, and more. These features are meticulously analyzed to understand the relationships and similarities between movies, enabling the system to recommend movies that align with a user's interests.

The project is implemented in Python, utilizing a structured workflow that includes data preprocessing, feature engineering, and machine learning model development. Data preprocessing involves cleaning and merging datasets, extracting relevant attributes, and converting complex information (e.g., JSON fields for genres, cast, and crew) into simplified, usable tags. Feature extraction transforms these tags into a machine-readable format using natural language processing tools like CountVectorizer, which converts textual data into numerical feature vectors.

The recommendation engine is built using cosine similarity, a mathematical approach that measures the similarity between two vectors. By computing the similarity of feature vectors for all movies in the dataset, the system identifies and ranks movies most closely related to a given input movie. This technique ensures accurate and meaningful recommendations, as it takes into account multiple factors simultaneously.

To make the system accessible and user-friendly, an interactive web application has been developed using Streamlit, a modern Python framework for building data-driven web apps. The app features an intuitive user interface with a movie selection dropdown and displays recommendations alongside posters fetched dynamically using the TMDb API. This integration enhances the user experience by providing visual context to the recommendations.

The project is designed to be deployed on Render, a cloud-based hosting platform, ensuring scalability and availability for end-users. The deployment-ready architecture allows users to explore and enjoy personalized movie suggestions seamlessly, making this system a practical and impactful application of machine learning in the domain of entertainment.

Chapter- 1

Introduction

In today's digital-first world, where users are often overwhelmed by the abundance of choices, personalized recommendation systems have become indispensable. Platforms like Netflix, Amazon, and Spotify owe much of their success to their ability to curate content tailored to individual users. Inspired by this, the **Movie Recommender System** aims to create a machine-learning-driven solution that recommends movies to users based on their preferences.

The **TMDb 5000 Movies dataset**, sourced from Kaggle, provides a rich repository of movie-related data, including genres, keywords, cast, crew, and popularity. By leveraging this dataset, the project extracts meaningful insights to identify movies that are similar in theme, style, or content. The recommender system is designed to answer a fundamental question: *What movies might a user enjoy next, based on their interest in a specific movie?*

The project encompasses three core components:

1. **Data Preprocessing and Feature Engineering:** The dataset is preprocessed to clean, merge, and refine the raw information. JSON fields like *genres*, *keywords*, *cast*, and *crew* are converted into simplified tags that represent the essence of each movie. This structured transformation makes the data suitable for computational analysis.
2. **Machine Learning for Recommendations:** Using techniques like **CountVectorizer** to create numerical representations of movie tags, and **cosine similarity** to compute the closeness between movies, the project implements a robust recommendation engine. This ensures that users receive accurate and contextually relevant suggestions.
3. **Interactive Web Application:** To make the system accessible, a dynamic web application has been developed using **Streamlit**, a Python-based framework. This application allows users to select a movie from a dropdown menu and view personalized recommendations alongside movie posters fetched dynamically using the **TMDb API**.

The system is designed for deployment on **Render**, a cloud-based platform, ensuring scalability and accessibility. This end-to-end implementation combines technical innovation with user-centric design, demonstrating the potential of machine learning to transform user experiences in the entertainment industry.

By addressing challenges like data preprocessing, feature extraction, and similarity computation, this project not only builds a functional system but also provides valuable insights into the workings of machine learning in real-world applications.

Chapter- 2

Hardware and Software Requirements

This project requires both hardware and software resources to handle data preprocessing, machine learning computations, and web application development efficiently. The requirements have been chosen to ensure seamless execution, effective development, and smooth deployment of the system.

2.1 Hardware Requirements

To handle the computational and memory demands of this project, the following hardware specifications are recommended:

1. **Processor:**
 - Minimum: Intel Core i5 (or equivalent AMD Ryzen 5)
 - Recommended: Intel Core i7 or higher (or equivalent AMD Ryzen 7) for faster computation, especially during data preprocessing and similarity matrix computation.
2. **RAM:**
 - Minimum: 8 GB
 - Recommended: 16 GB or more to accommodate large datasets, reduce processing time, and ensure a lag-free development experience.
3. **Storage:**
 - Minimum: 10 GB free space for storing datasets, Python environments, libraries, and intermediate files during development.
 - Recommended: SSD (Solid State Drive) storage for faster read/write operations, especially when loading large datasets.
4. **Display:**
 - A high-resolution monitor is recommended for better visualization during Streamlit app testing and debugging.
5. **Network:**
 - A stable internet connection is essential for downloading libraries, accessing the TMDb API, and deploying the application on Render

2.2 Software Requirements

To develop and deploy the system effectively, the following software components are utilized:

1. **Programming Language:**
 - **Python:** The core language used for data preprocessing, feature extraction, machine learning, and web app development. Python is chosen for its extensive libraries and community support for machine learning and data science.
2. **Python Libraries:**
 - **numpy:** For numerical computations.
 - **pandas:** For data manipulation and analysis.
 - **sklearn:** To implement machine learning techniques like CountVectorizer and cosine similarity.
 - **streamlit:** To build an interactive and user-friendly web application.
 - **requests:** To fetch movie posters from the TMDb API.
 - **pickle:** For saving and loading preprocessed data and the similarity matrix.
3. **Development Tools:**
 - **PyCharm:** The primary IDE for writing, testing, and debugging Python code, chosen for its robust debugging tools and support for project management.
 - **Anaconda Jupyter Notebook:** Used during the initial stages for exploratory data analysis, code prototyping, and visualization.
4. **Deployment Platform:**
 - **Render:** A cloud hosting platform for deploying the Streamlit application. Render provides an efficient and scalable environment to host the application, making it accessible to users online.
5. **Dataset Source:**
 - **TMDb 5000 Movies Dataset:** Sourced from Kaggle, this dataset serves as the foundation for building the recommendation system. It includes attributes like genres, keywords, cast, and crew that are critical for computing movie similarities.
6. **Operating System:**
 - The project is cross-platform and can run on Windows, macOS, or Linux. However, Linux-based systems are recommended for deployment due to better support for server-side operations.

This combination of hardware and software resources ensures an efficient development pipeline and provides a robust infrastructure for deploying the Movie Recommender System.

Chapter- 3

Problem Statement

The entertainment industry faces a critical challenge: understanding audience preferences and delivering personalized content in a landscape saturated with thousands of movies. While platforms today have access to extensive databases of films, many struggle to provide recommendations that align intuitively with individual user tastes. This gap often leaves users overwhelmed by choices and dissatisfied with generic suggestions.

Objective:

This project aims to bridge this gap by building a **Movie Recommender System** capable of analyzing movie features and delivering personalized movie suggestions. By leveraging machine learning and data from the TMDb 5000 Movies dataset, the system models the underlying attributes of movies to offer tailored recommendations.

3.1 Key Challenges Addressed

1. Identifying Movie Similarities

One of the core issues is determining how similar two movies are based on their attributes. Unlike structured user preference data, the similarity needs to be derived by analyzing a combination of features such as genres, keywords, cast, crew, and plot summaries. Developing a reliable similarity computation method is pivotal to ensuring meaningful recommendations.

2. Understanding Attributes Influencing Preferences

Movie preferences are subjective and influenced by numerous factors. For instance:

- A user may like a movie for its genre (e.g., action, romance).
- Another might prioritize the cast or director.
- Some could be drawn to thematic keywords or the overall storyline.

The system needs to balance these elements and determine which features most impact user preferences for a better recommendation algorithm.

3. Delivering an Intuitive User Experience

Beyond accurate recommendations, presenting them through an intuitive and engaging interface is essential. A user-friendly platform that is visually appealing and easy to navigate enhances the overall experience. The system must display recommendations clearly while offering engaging visuals, such as movie posters, to attract users.

3.2 Key Questions Addressed

To solve the problem effectively, the project tackles the following questions:

1. **How can we determine the similarity between movies?**
The system uses feature extraction techniques like `CountVectorizer` to convert textual features (tags) into vectors. By employing **cosine similarity**, the project computes the degree of relatedness between movies, enabling accurate recommendations.
2. **What attributes contribute most to user preferences?**
Features such as *genres*, *keywords*, *cast*, *crew*, and *overview* are extracted from the dataset. By combining these into a single tag for each movie, the system identifies the critical factors influencing user preferences.
3. **How can we deliver recommendations through a user-friendly interface?**
Using **Streamlit**, a modern Python framework, the system creates an interactive web application. The interface includes:
 - A dropdown for selecting a movie.
 - Recommendations displayed with corresponding posters fetched via the **TMDb API**.
 - A visually engaging design with minimalistic navigation.

3.3 Impact of the Project

By addressing these challenges, the Movie Recommender System offers:

- **Personalized Recommendations:** Delivering suggestions aligned with user tastes.
- **Enhanced User Experience:** A visually engaging and easy-to-use interface.
- **Scalability:** A system designed for deployment, capable of serving large user bases on platforms like Render.

This project demonstrates the application of machine learning to solve real-world problems in the entertainment industry, paving the way for improved decision-making and user satisfaction in content discovery.

Chapter- 4

Related Work (Literature Survey)

Recommendation systems have become an integral part of modern digital platforms, especially in streaming services, e-commerce, and social networks. These systems analyze user behavior and content attributes to deliver personalized suggestions, aiming to enhance user satisfaction and engagement. Below is an overview of key techniques and studies that laid the foundation for this project:

4.1 Recommendation Techniques

1. Collaborative Filtering:

- Relies on user behavior and interactions.
- Example: If two users rate similar movies highly, their preferences can be used to recommend new movies to each other.
- Widely used by platforms like Netflix, which employs matrix factorization and deep learning to uncover latent user-item interactions.

2. Content-based Filtering:

- Analyzes the features of items (e.g., movies) to find similarities.
- For example, a user who likes *Inception* might be recommended *Interstellar* based on shared attributes like genre (*sci-fi*), director (*Christopher Nolan*), or themes (*mind-bending concepts*).
- This project employs content-based filtering by extracting movie features such as *genres*, *keywords*, *cast*, *crew*, and *overview*.

3. Hybrid Methods:

- Combine collaborative and content-based filtering for enhanced performance.
- Example: Netflix blends collaborative filtering with metadata analysis to achieve better recommendations.

4.2 Relevant Studies and Applications

1. Netflix Prize Competition (2006–2009):

- Highlighted the importance of predictive modeling in recommendation systems.
- Teams competed to improve the accuracy of Netflix's movie recommendations, showcasing innovations like matrix factorization.
- Inspired the use of sophisticated algorithms in modern recommendation engines.

2. Use of Cosine Similarity in Recommender Systems:

- Cosine similarity, a key technique in this project, measures the angle between feature vectors to determine similarity.

- Studies have shown its effectiveness in content-based filtering, particularly when working with high-dimensional data like text features.
 - Frequently used in movie recommendation systems to compare attributes like plot summaries or tags.
3. **Research on Feature Engineering for Recommendations:**
- Papers emphasize the significance of extracting and processing relevant features to improve recommendations.
 - For example, the inclusion of multi-modal features (text, metadata, user reviews) has shown to improve recommendation accuracy in academic studies.
4. **Personalization in Streaming Platforms:**
- Services like Amazon Prime Video, Disney+, and YouTube employ recommendation systems as a critical engagement tool.
 - These systems analyze metadata (e.g., genres, release years), user interaction (e.g., clicks, ratings), and contextual data (e.g., time of day) to deliver personalized suggestions.

4.3 Content-Based Filtering in This Project

This project builds on the principles of content-based filtering to develop a recommendation engine that:

1. Extracts features such as genres, keywords, cast, and crew from the TMDb 5000 dataset.
2. Combines these attributes into a unified "tag" for each movie to create a meaningful representation.
3. Uses **cosine similarity** to compute the relevance between movies based on their tags.

4.4 Innovative Application

By implementing these techniques, this project demonstrates:

- **Practical Utility:** An accessible, web-based tool for recommending movies based on user-selected preferences.
- **Scalability:** The ability to extend the system with additional features or integrate hybrid models for improved performance.
- **Relevance to Industry Trends:** A solution aligned with real-world needs in the entertainment industry, inspired by successful systems like Netflix and IMDb.

Chapter- 5

Major Modules of Project

The **Movie Recommender System** is designed as a web-based application that leverages machine learning techniques to recommend movies to users based on their preferences. The project can be divided into the following major modules:

5.1 Data Collection and Preprocessing

- **Purpose:** Collect and clean data to ensure it is in a usable format for building the recommendation model.
- **Details:**
 - We used the TMDb 5000 Movies dataset, which includes information about movie titles, genres, cast, crew, budget, revenues, and more.
 - The dataset was preprocessed by merging the movie data and credits (cast and crew), cleaning missing or irrelevant data, and extracting key features like genres, keywords, cast, and crew.
 - Textual information, including overviews and descriptions, was transformed into tags that can be used as input to the model. These tags were created by concatenating genres, keywords, cast, crew, and the movie's overview.

5.2 Feature Engineering and Similarity Computation

- **Purpose:** Create feature vectors representing each movie and compute similarity between movies for recommendations.
- **Details:**
 - **Feature Extraction:** The features (genres, keywords, cast, crew, and overview) were transformed into tags using a text-based approach. These tags were then processed using a **CountVectorizer** to create a feature matrix.
 - **Cosine Similarity:** We applied cosine similarity to compute how similar two movies are based on their feature vectors. This forms the basis for recommending movies similar to the one selected by the user.
 - **Similarity Matrix:** A similarity matrix was generated using the feature vectors, and it was stored for efficient retrieval during the recommendation process.

5.3 Recommendation System

- **Purpose:** Recommend movies similar to the one selected by the user.
- **Details:**
 - Using the similarity matrix, the system identifies the top 5 most similar movies to the user's selected movie.
 - The movies are ranked based on their similarity score, and the system outputs the names of the recommended movies, along with their posters (fetched from the TMDb API).

5.4 Web Interface (Streamlit)

- **Purpose:** Create an interactive user interface that allows users to select movies and view recommendations.
- **Details:**
 - **User Interface:** Built using Streamlit, the interface features a dropdown menu that allows users to select a movie.
 - **Movie Display:** When a movie is selected, the system displays its details along with the top 5 recommended movies in the form of images (posters) and movie names.
 - **API Integration:** The TMDb API is used to fetch movie posters for the selected movie and its recommendations.
 - **User Interaction:** Users interact with the application by selecting a movie and clicking a button to display the recommended movies.

5.5 Deployment and Saving Models

- **Purpose:** Ensure that the application is accessible and that the machine learning models are reusable.
- **Details:**
 - The project is prepared for deployment using **Render**, a cloud service platform that supports web application hosting.
 - The preprocessed movie data and similarity matrix are saved as pickle files to allow the model to be easily loaded without needing to rerun the entire preprocessing step.

Chapter- 6

Snapshot of Portal Designed / Stimulation Performed (Till Now)

Below is a snapshot of the **Movie Recommender System** application designed using **Streamlit**. It provides an interactive interface where users can select a movie and get personalized recommendations.

6.1 Screenshot of the Movie Recommender Web Interface

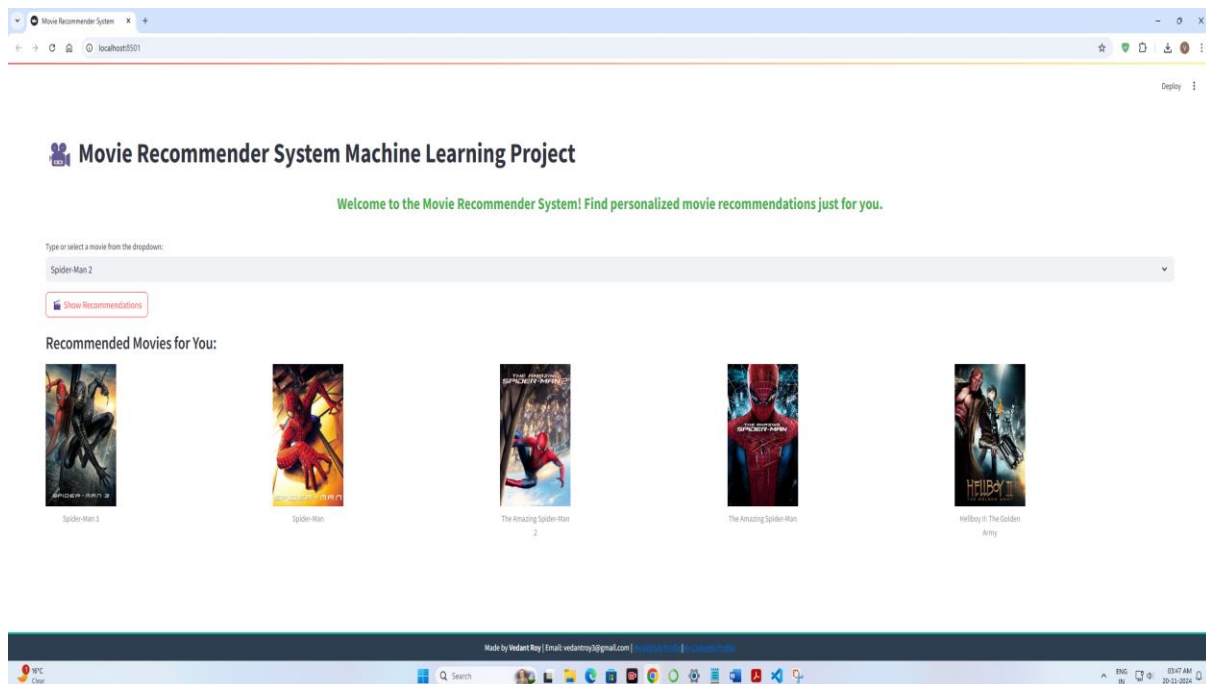
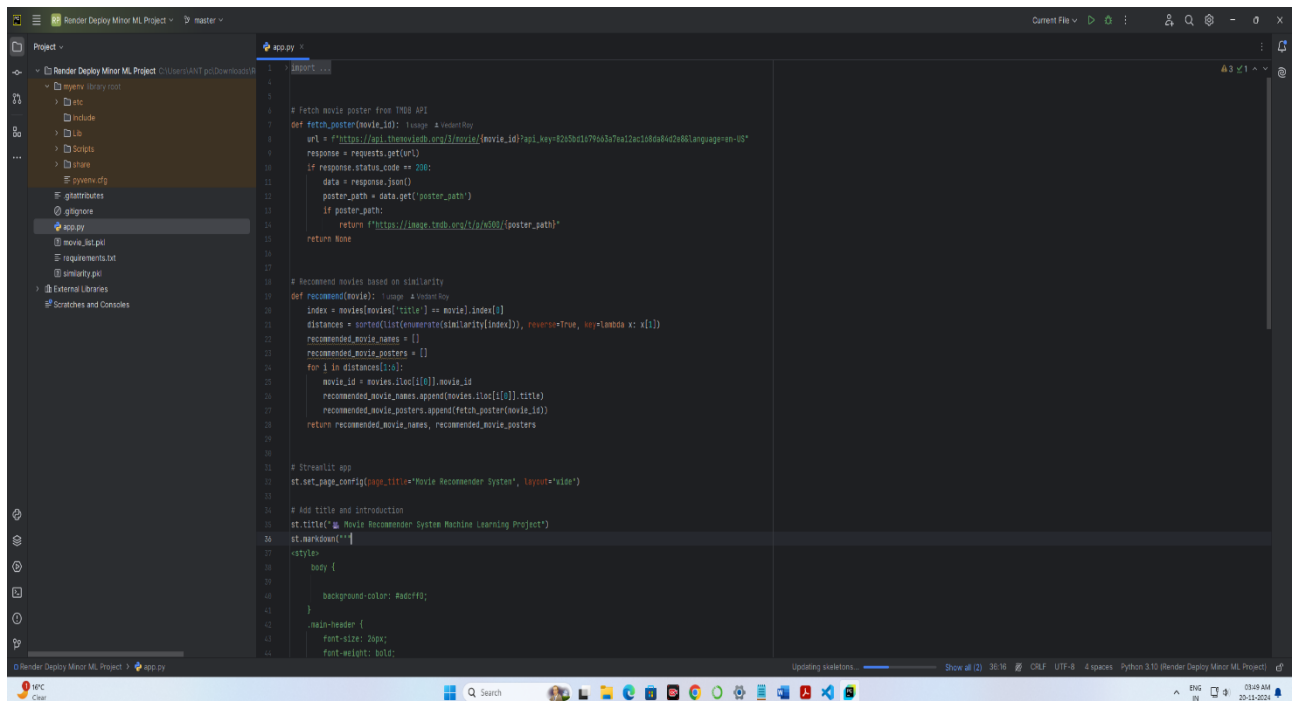


Fig. 6.1: Screenshot of the Movie Recommender Web Interface

- The portal features a **dropdown menu** where users can type or select a movie title from a list of movies in the dataset.
- Upon selection, users can click on a button labeled "Show Recommendations" to view a set of 5 recommended movies with their posters.
- The **movie posters** are fetched dynamically using the TMDb API, ensuring a seamless user experience.
- A **watermark footer** provides credits and contact information.
- The image depicts a code editor window displaying a React project's file structure and code content.

6.2 Screenshot of Front-End Code of Movie Recommender System



```
1 # Fetch movie poster from TMDB API
2
3 # Fetch movie poster from TMDB API
4 def fetch_poster(movie_id):
5     url = "https://api.themoviedb.org/3/movie/{movie_id}?api_key={api_key}&language=en-US"
6     response = requests.get(url)
7     if response.status_code == 200:
8         data = response.json()
9         poster_path = data.get('poster_path')
10         if poster_path:
11             return "https://image.tmdb.org/t/p/w500/{poster_path}"
12         return None
13
14 # Recommend movies based on similarity
15 def recommend(movie):
16     index = movies[movies['title'] == movie].index[0]
17     distances = sorted(list(enumerate(similarity[index])), reverse=True, key=lambda x: x[1])
18     recommended_movie_names = []
19     recommended_movie_posters = []
20     for i in distances[1:3]:
21         movie_id = movies.iloc[i[0]].movie_id
22         recommended_movie_names.append(movies.iloc[i[0]].title)
23         recommended_movie_posters.append(fetch_poster(movie_id))
24     return recommended_movie_names, recommended_movie_posters
25
26 # Streamlit app
27 st.set_page_config(page_title="Movie Recommender System", layout="wide")
28
29 # App title and introduction
30 st.title("Movie Recommender System Machine Learning Project")
31 st.markdown("")
32
33 # Style
34 body =
35     body {
36         background-color: #f0f0f0;
37     }
38 # Main-Header
39 st.header(
40     st.header(
41         Font-size: 20px;
42         Font-weight: bold;
```

Fig. 6.2: Screenshot of Front-End Code of Movie Recommender System

The image depicts a code editor window displaying a Machine Learning project's file structure and code content that uses streamlit for casting it on website

6.3 User Interaction Process

1. **Movie Selection:** Users select a movie from the dropdown list.
2. **Recommendation Generation:** After clicking the "Show Recommendations" button, the system processes the input and generates a list of recommended movies.
3. **Displaying Recommendations:** The recommendations are displayed in a grid with movie posters and names, providing an easy-to-read format for users to explore similar films.

References

Here are the references used in the development of the **Movie Recommender System**:

1. The Movie Database (TMDb). (n.d.). *TMDb API documentation*. Retrieved from <https://developers.themoviedb.org/3>

Description: The TMDb API was used to fetch metadata for movies, such as their posters, titles, and unique IDs. This integration enabled an enhanced user experience by displaying visual information for the recommended movies.

2. Kaggle. (n.d.). *TMDB 5000 Movies Dataset*. Retrieved from <https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata>

Description: This dataset provided the foundational data for the recommendation system, containing essential movie details such as genres, keywords, cast, and crew. It served as the primary resource for building the feature set.

3. Streamlit. (n.d.). *Streamlit documentation*. Retrieved from <https://docs.streamlit.io/>

Description: Streamlit was the framework utilized to design the web-based interactive interface. The documentation served as a reference for implementing features like dropdowns, buttons, and displaying movie posters in the application.

4. ResearchGate. (2016). *Cosine similarity and its application in recommender systems*. Retrieved from https://www.researchgate.net/publication/311795257_Cosine_Similarity_and_Its_Application_in_Recommender_Systems

Description: The cosine similarity concept was implemented to calculate the similarity scores between movies based on their feature vectors. This reference provided an understanding of its relevance and application in content-based recommender systems.

5. Scikit-learn. (n.d.). *CountVectorizer documentation*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

Description: The CountVectorizer was used to process textual data (tags) and

create feature vectors for the movies. The documentation helped in configuring the tool effectively, especially with parameters like `max_features` and `stop_words`.

6. Koren, Y., Bell, R., & Volinsky, C. (2009). *Matrix factorization techniques for recommender systems*. *ACM Computing Surveys (CSUR)*, 42(1), 30. Retrieved from

Description: This paper provided insights into collaborative filtering and matrix factorization techniques. While the implemented system was primarily content-based, the knowledge from this reference enhanced the overall understanding of recommender system methodologies.

These references provided crucial insights into building and deploying a content-based recommendation system and integrating web technologies like Streamlit for user interaction.