



Shahid Beheshti University

Final Project – Code Report

Machine Learning

By

**Abolfazl Fekri
Hediyeh Talebi**

Supervisor:

Dr. Hadi Farahani

Jul 2023

Contents	Page
ABSTRACT	5
1. INTRODUCTION	7
1-1 Overview	<i>Error! Bookmark not defined.</i>
2. CLUSTERING	9
2-1 Reading Data	9
2-2 Data Preprocessing	11
2-2-1 Relationship between Days of the Week and Number of Orders	<i>Error! Bookmark not defined.</i>
2-3 Clustering Methodology	14
2-3-1 DB	<i>Error! Bookmark not defined.</i>
2-3-2 K-means	<i>Error! Bookmark not defined.</i>
3. GRADIO USER INTERFACE.....	18
3-1 Importing and Installation	18
3-2 Functions	19
3-3 Gradio Interface lunching	22
4. CONCLUSION	27

Table of figures	Page
Fig 2.1: Overview to the part of dataset.....	9
Fig 2.2: Head of dataset after removing high percentage of zero values	12
Fig 2.3: 3D visualizing PCA-transformed data by scatter plot.	15
Fig 2.4: 2D visualizing PCA-transformed data by scatter plot.	16
Fig 4.4: UI of this project gradio web-app.....	24
Fig 4.5: Representing the capability of selecting or typing favorite movies as Input.	25
Fig 4.5: Result of choosing withing typing 'Narnia' as Input, which can be scrolled to see al 5 recommendation.....	25

00

Abstract

Abstract

With the vast amount of movies available today, it can be overwhelming for related businesses to discover new movies that align with their users' interests. Movie recommender systems play a crucial role in addressing this challenge by suggesting personalized movie recommendations based on various factors such as genre, ratings, and user preferences.

This report presents a movie recommender system using the clustering method and Gradio user interface. The system leverages clustering algorithms to group movies based on their features and provides user recommendations based on their preferences. The Gradio library creates an interactive, user-friendly interface for seamless movie exploration and recommendation. Additionally, the system is deployed on Hugging Face Spaces, allowing easy access and interaction with the recommender system. Results show that recommended movies seem related to the real world, asking random users and comparing them to movie websites recommending engines.

Key Words: Movie Recommender system, Hugging Face, clustering algorithms, Gradio User Interface.

01

Introduction

1. Introduction

The movie recommender system is an essential tool in the entertainment industry to provide personalized movie recommendations to users. We created a movie recommender system that combines the power of clustering algorithms and the versatility of the Gradio library to create an efficient and intuitive platform for users to explore and discover movies.

This report will delve into the different components of our movie recommender system. Firstly, we will discuss the clustering method used to group movies based on their shared characteristics. This clustering approach enables us to identify movie similarities and patterns, facilitating the recommendation process. Next, we will explore the Gradio library, which allows us to build an interactive user interface for seamless movie browsing and recommendation. The Gradio interface empowers users to select movies, input preferences, and receive personalized recommendations in a user-friendly manner.

Furthermore, we will highlight the deployment of our movie recommender system on Hugging Face Spaces. Hugging Face Spaces is a platform for hosting and sharing machine learning models, making our recommender system accessible to a broader audience. By deploying our system on Hugging Face Spaces, we aim to provide a seamless and convenient experience for users to interact with the recommender system and explore a vast collection of movies.

02

Clustering

2. Clustering

The chosen **approach** for the **recommender system** is the **Cluster-Based** method. This method involves grouping users or items into clusters based on their similarities. Recommendations can be made by leveraging the preferences or characteristics of other users or items within the same cluster.

To implement the clustering model, several steps were taken:

2-1 Reading Data

The dataset used for this task focuses explicitly on the "**movies_metadata**" subset of the larger dataset. This subset contains a vast collection of information on **45,000 movies**, encompassing various essential details such as **budget**, **revenue**, **release dates**, **languages**, **production countries**, and more.

Reading the dataset locally involved **uploading** the data and **selecting** the **desired file** for analysis. The uploaded dataset file was then read using the **Pandas** library.

The Movies Dataset, obtained from The Movie Database (**TMDb**) **API**, is a comprehensive resource for movie-related data. It provides valuable insights into the movie industry, allowing researchers, analysts, and developers to understand movie characteristics and trends better.

The dataset comprises a wide range of features that offer extensive information for analysis and exploration (Fig 2.1).

belongs_to_collection	budget	genres	homepage	id	imdb_id	original_language	original_title	overview	...	release_date	revenue	runtime	spoken_languages	status	tagline	title	video	vote_average	vote_count
['id': 10194, 'name': 'Toy Story Collection', ...]	300000000	['id': 16, 'name': 'Animation', 'id': 35, ...]	http://toystory.disney.com/toy-story	862	tt0114709	en	Toy Story	Led by Woody, Andy's toys live happily in his	1995-10-30	373554033.0	81.0	['iso_639_1': 'en', 'name': 'English']	Released	NaN	Toy Story	False	7.7	5415.0
NaN	65000000	['id': 12, 'name': 'Adventure', 'id': 14, ...]	NaN	8844	tt0113497	en	Jumanji	When siblings Judy and Peter discover an enche...	...	1995-12-15	262797249.0	104.0	['iso_639_1': 'en', 'name': 'English'], ['iso...	Released	Roll the dice and unleash the excitement!	Jumanji	False	6.9	2413.0
['id': 119050, 'name': 'Grumpy Old Men Collect...	0	['id': 10749, 'name': 'Romance', 'id': 35, ...]	NaN	15602	tt0113228	en	Grumpier Old Men	A family wedding reignites the ancient feud be...	...	1995-12-22	0.0	101.0	['iso_639_1': 'en', 'name': 'English']	Released	Still Yelling Still Fighting Still Ready for...	Grumpier Old Men	False	6.5	92.0
NaN	160000000	['id': 35, 'name': 'Comedy', 'id': 18, nam...	NaN	31357	tt0114885	en	Waiting to Exhale	Cheated on, mistreated and stepped on, the wom...	...	1995-12-22	81452156.0	127.0	['iso_639_1': 'en', 'name': 'English']	Released	Friends are the people who let you be yourself...	Waiting to Exhale	False	6.1	34.0
['id': 96871, 'name': 'Father of the Bride Col...	0	['id': 35, 'name': 'Comedy']	NaN	11862	tt0113041	en	Father of the Bride Part II	Just when George Banks has recovered from his	1995-02-10	76578911.0	106.0	['iso_639_1': 'en', 'name': 'English']	Released	Just When His World Is Back To Normal... He's ...	Father of the Bride Part II	False	5.7	173.0

Fig 2.1: Overview to the part of dataset

Key features include:

- **Budget:** The estimated production cost of each movie, providing insights into the financial investment involved in its creation.
- **Revenue:** The generated income for each movie, indicating its commercial success and financial performance.
- **Release Dates:** The dates on which the movies were released, enabling temporal analysis and the identification of trends over time.
- **Languages:** The languages in which the movies were produced or dubbed reflect the cultural diversity of the films.
- **Production Countries:** The countries primarily associated with each movie's production, highlighting the film industry's global nature.
- **Production Companies:** The companies involved in the production of the movies, showcasing the collaborative efforts behind the creation of each film.
- **User Ratings:** Ratings provided by a large number of users, offering an indication of audience reception and popularity.

This dataset holds excellent potential for a wide range of applications. Researchers can use it to explore correlations between budget and revenue, identify successful genres and production companies, and analyze the influence of release dates and languages on movie reception. Moreover, it provides a [foundation for building recommendation systems](#) that suggest movies based on user preferences, ratings, and other relevant factors.

2-2 Data Preprocessing

In the preprocessing phase of clustering for the movie recommendation system, a series of data preparation steps were performed to optimize the movie dataset. These steps involved cleaning, transforming, and organizing the data to enhance the clustering process's quality, relevance, and efficiency

2-2-1 Selecting Relevant Columns:

The initial step of the preprocessing involved selecting the **columns** that are **essential for the analysis**. The selected columns include 'adult', 'budget', 'genres', 'popularity', 'vote_average', 'original_language', 'release_date', 'revenue', 'runtime', 'vote_count', 'title', and 'poster_path.'

2-2-2 Extracting Main Genre:

In this part, the 'genres' column was processed to extract the main genre of each movie. The extraction involved parsing the genre string and retrieving the first listed genre. This information was stored in a **new column called 'main_genre.'**

2-2-3 Dropping Irrelevant Columns:

After extracting the primary genre, the 'genres' column was dropped from the dataset as it was no longer needed for analysis.

2-2-4 Handling Missing Values:

The dataset was **checked for missing values using the 'isna()' function**. Any rows with missing values were dropped from the dataset.

2-2-5 Removing Columns with High Percentage of Zeros:

Columns with a high percentage of zero values were identified and removed from the dataset. These columns included 'budget' and 'revenue.' (Fig 2.2)

	adult	popularity	vote_average	original_language	release_date	runtime	vote_count	title	poster_path	main_genre
0	False	21.946943	7.7	en	1995-10-30	81.0	5415.0	Toy Story	/rhIRbceoE9IR4veEXuwCC2wARTG.jpg	Animation
1	False	17.015539	6.9	en	1995-12-15	104.0	2413.0	Jumanji	/vzmL6fP7aPKNKPRTFnZmiUfcyV.jpg	Adventure
2	False	11.712900	6.5	en	1995-12-22	101.0	92.0	Grumpier Old Men	/6ksm1sjKMFLbO7UY2li6G1ju9SML.jpg	Romance
3	False	3.859495	6.1	en	1995-12-22	127.0	34.0	Waiting to Exhale	/16XOMpEaLWkrPqSQqhTmeJuqQl.jpg	Comedy
4	False	8.387519	5.7	en	1995-02-10	106.0	173.0	Father of the Bride Part II	/e64sOI48hQXyru7naBFyssKFxVd.jpg	Comedy

Fig 2.2: Head of dataset after removing high percentage of zero values

2-2-6 Removing Zero-Valued Entries:

Rows with zero values in columns such as 'popularity', 'vote_average', 'runtime', and 'vote_count' were removed from the dataset.

2-2-7 Extracting Year from Release Date:

The 'release_date' column was processed to extract the year information. A new column called 'release_year' was created, containing only the year values. The 'release_date' column was then dropped from the dataset.

2-2-8 Categorizing Languages:

The 'original_language' column was analyzed to identify the top five languages based on their frequency which were English, French, Japanese, Italian, German. Other languages than the top five were categorized as 'Other' and stored in a new column called 'reduced_language'. The 'original_language' column was dropped from the dataset.

2-2-9 Categorizing Genres:

The 'main_genre' column was analyzed to identify the top ten genres based on their frequency which were Drama, Comedy, Action, Documentary, Horror, Crime, Thriller, Adventure, ' ', Animation,. The genre ' ' (empty string) was removed from the top genres list. Other genres than the top ten were categorized as 'Other' and stored in a new column called 'reduced_genre'. The 'main_genre' column was dropped from the dataset.

2-2-10 Dropping Additional Columns:

The 'adult' column, which mainly contained 'False' values, was dropped from the dataset as it had limited relevance for movie analysis.

2-2-11 Scaling and Encoding:

Numeric columns such as 'popularity', 'vote_average', 'runtime', 'vote_count', and 'release_year' were scaled using the StandardScaler to improve the range of their values. Categorical columns, 'reduced_language' and 'reduced_genre', were one-hot encoded using the OneHotEncoder.

2-2-12 Saving the Scaler Model:

The trained StandardScaler model was saved using the joblib library for future use.

2-2-13 Handling Poster URLs:

The columns of URLs for movie posters were removed cause the poster will be retrived from TMDB by requesting to its API.

2-2-14 Handling Movie Titles:

The movie titles were stored separately in the 'titles' variable, and the 'title' column was dropped from the dataset for now. However, we will add this column to the data after finding clusters to match them to their corresponding labels.

The dataset was cleaned and transformed by performing these preprocessing steps, preparing it for further analysis and clustering. The resulting dataset contained relevant columns, handled missing values, removed irrelevant and zero-valued entries, and performed necessary encoding and scaling operations.

2-3 Clustering Methodology

The clustering methodology employed in this study involved the utilization of the K-means algorithm to group movies based on their features. The following steps outline the process:

2-3-1 K-means Clustering:

The K-means algorithm was implemented using the KMeans function from the scikit-learn library. The algorithm was configured with 13 clusters and 10 initializations. By fitting the algorithm to the preprocessed movie dataset, cluster assignments were obtained for each movie. Choosing 13 cluster is based on trying different clusters from 1 to 15 and selection the best one.

2-3-2 Dimensionality Reduction:

Principal Component Analysis (PCA) was applied to reduce the dimensionality of the data. Two visualizations were created: one in three dimensions and another in two. These visualizations provided a simplified representation of the data while capturing the essential characteristics of the clusters.

2-3-3 Visualization in 3D:

The PCA-transformed data in three dimensions was visualized **using a scatter plot**. Each point in the plot represented a movie, with its coordinates determined by the first three principal components. The points were color-coded based on their assigned cluster, providing insights into the clustering results.

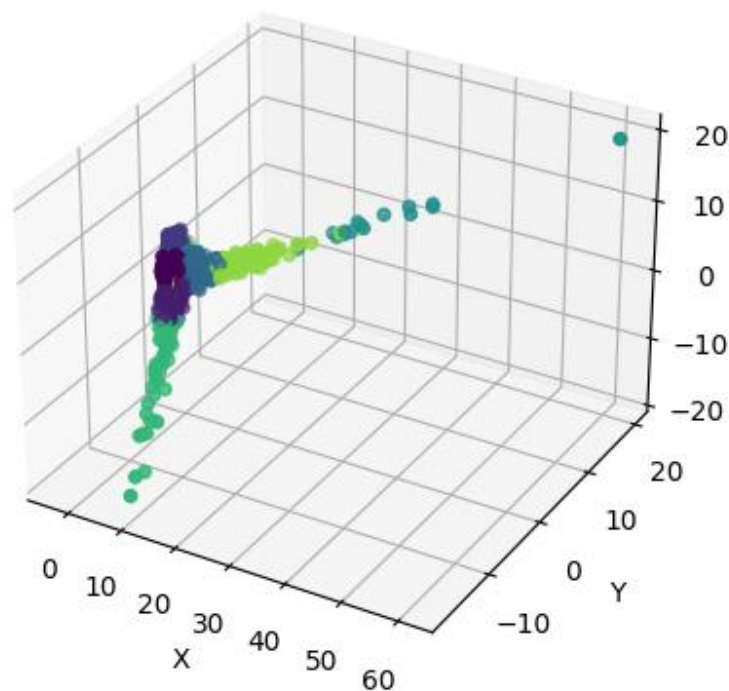


Fig 2.3: 3D visualizing PCA-transformed data by scatter plot.

2-3-4 Visualization in 2D:

Similarly, a two-dimensional scatter plot was generated using the PCA-transformed data. Each point represented a movie, positioned according to the first two principal components. The points were colored based on the assigned cluster, offering an alternative perspective on the clustering outcomes.

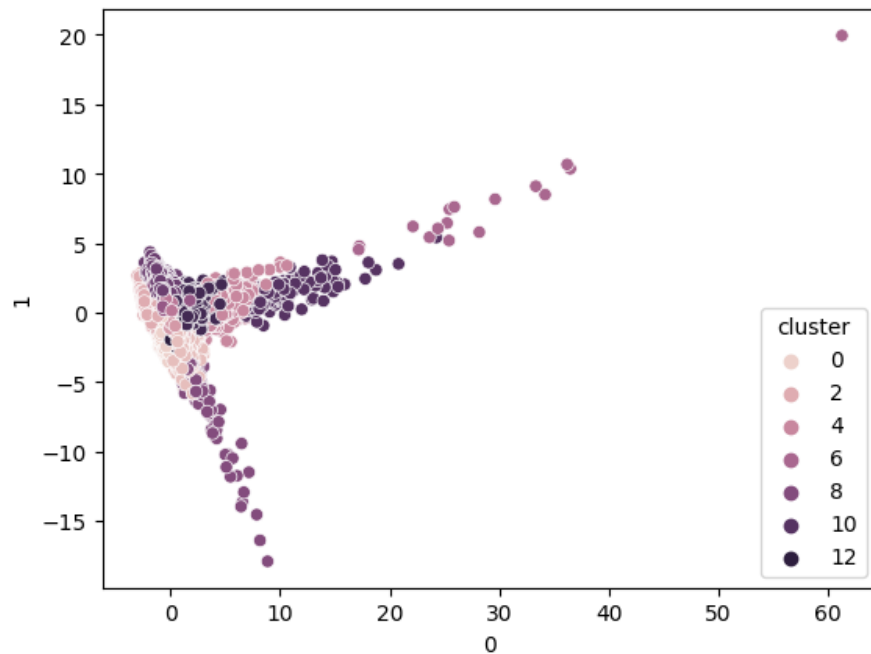


Fig 2.4: 2D visualizing PCA-transformed data by scatter plot.

2-3-5 Cluster Analysis:

Further analysis was conducted on the resulting clusters. A data frame incorporated movie titles and cluster labels. The distribution of movies within each cluster was examined to identify potential anomalies or outliers by checking if any cluster contained fewer than six movies.

2-3-6 Output Storage:

The **augmented** dataset was **saved** as separate **CSV** files, including assigned clusters, movie titles, and visualizations. This facilitated additional analysis and served as a reference for future research.

2-3-7 Model Persistence:

The **model** was **saved using** the **pickle** library to enable future use of the trained K-means clustering model without retraining.

03

Gradio for User Interface

3. Gradio User Interface

Gradio is a Python library that allows us to create interactive interfaces for machine-learning models. In this project, we utilized Gradio to develop an interface for our Movie Recommender System. The interface enables users to select their favorite movies or enter (type) movie names and provides recommendations based on their choices. We implemented a set of functions for both handling new movies and movies from the clustered dataset. Some of these functions are common for both scenarios and some for specific scenarios tasks such as generating different recommendations or fetching movie posters and genre features from the OMDB API.

3-1 Importing and Installation

Installation:

To ensure the proper installation of the packages required for our Movie Recommender System on Hugging Face, which will be in the requirements.txt file later, in the Hugging Face repository, we used the !pip install module to remember the task. So this file contains a comprehensive list of the necessary modules and their specific versions. Utilizing the pip package manager, we executed the installation command, guaranteeing that all the required packages were downloaded and configured correctly.

While some modules may have already been pre-installed in the Colab environment, this action, ensures a consistent and reproducible setup for our project. This approach aids in managing dependencies and ensures that all the essential packages are readily available, enabling our code to function effectively.

Imports

After completing the installation process, we imported the required libraries for our project. This included importing Gradio, the main library that enables us to create interactive interfaces for machine-learning models. Additionally,

we imported other relevant libraries, such as `requests`, `Images from PIL`, `load_dataset from datasets`, etc., which were necessary for the functionality and implementation of our Movie Recommender System.

3-2 Reading data

We created a dedicated dataset for our movie recommender system within the Hugging Face environment. We prepared the dataset by uploading a CSV file containing our clustering model's output. This file included essential information about the movies, such as their titles, features, and assigned clusters.

We employed the `load_dataset` function from the Hugging Face datasets module to access and utilize this dataset.

We **transformed** the retrieved dataset **into** a `pandas DataFrame` using the `to_pandas` method to facilitate further analysis and model development. This allowed us to work with the data more conveniently and leverage the rich functionality of pandas for data manipulation and exploration.

3-3 Functions

1. `fetch_list_posters(movie_names):`

This function **fetches the posters of multiple movies using the TMDB API**. It takes a list of movie names as input and returns a list of poster URLs.

2. `fetch_movie_poster(movie_name):`

This function **fetches the poster of a single movie using the TMDB API**. It takes a movie name as input and returns the movie poster as an image.

3. `fetch_movie_feature(movie_name, feature):`

This function **fetches a specific feature of a movie using the TMDB API**. It takes a movie name and a feature name as inputs and **returns the corresponding**

feature value. Because of time limitations, we only used **'genre'** features in this case.

4. `to_numpy_images_from_urls(image_urls):`

This function **converts a list of image URLs to a list of NumPy arrays representing the images**. It takes a list of poster URLs as input and returns a list of NumPy arrays.

5. `get_label_by_name(movie_name):`

This function **retrieves the cluster label of a movie, using its title**. It takes a movie name as input and **returns the movie label**.

6. `find_samelabel_movie(label):`

This function **finds a random movie with the same cluster label**. It takes a label as input and **returns a movie name**.

7. `movie_existence(movie_name):`

This function **checks if a movie exists in the dataset**. It takes a movie name as input and **returns a boolean input (True or False)**.

8. `recommend_based_on_new_movie(movie_name):`

This function **recommends movies based on a new movie, not in the dataset**.

The function begins by **extracting the genre information** of the new movie using the `fetch_movie_feature` function. It then **compares the genre IDs of the new movie** with a predefined dictionary that maps genre IDs to their corresponding labels, sets based on preprocessing phase. This comparison helps to find common genres between the new movie and the dataset.

If there are shared genres, the function selects random movies from the dataset that belong to those genres.

In cases of no common genres, the function selects random movies categorized as the **"Other"** genre from the dataset.

It takes a movie name as input and returns a list of recommended movie titles.

9. `recommend_movie(movie_list):`

This function recommends movies based on a list of selected movies or the name of a movie entered in a text box, provided that it exists in the dataset.

It iterates through the selected movie list or `temp_list` (for text input) and retrieves the label associated with each movie. Using this label, it searches the dataset for movies that share the same label.

So, it takes a list of movie titles as input and returns a list of 5 recommended movie titles.

10. `gradio_function(selected_movies , text):`

It takes two parameters as input: `selected_movies` (a list of selected movie titles in Checkbox Group) and `text` (any movie title typed by the user).

The function checks if `text` exists in the dataset if it is provided. If it does, it retrieves a list of recommended movies based on that input using the `recommend_movie` function. The function then fetches the movie posters for the recommended movies using the `fetch_movie_poster` function. These posters are converted into NumPy arrays and stacked vertically to represent the recommended movies visually. The array of posters is returned as the output.

If `text` is not found in the dataset, the function generates recommendations based on the entered movie title using the `recommend_based_on_new_movie` function. The posters for the recommended movies are fetched, converted to numpy arrays, and vertically stacked. This array is then returned as the output.

If the selected movie length exceeds 5, the function returns a message asking the user to select fewer than 5 movies.

If the length of `selected_movies` is less than 6 (but not empty), the function generates recommendations based on the selected movies using the `recommend_movie`

`function`. The movie posters for the recommended movies are fetched, converted to numpy arrays, and vertically stacked. This array is returned as the output.

If no movies are selected (i.e., both `text` and `selected_movies` are empty), the function returns a message stating that no movies were selected.

`Set_movie_options()`

This function **generates a list of 26 movie options** (2 movies from each cluster) based on predefined labels. The interface utilizes these options, enabling users to select movies from the provided choices.

3-4 Gradio Interface lunching

We created a Gradio interface using the `gr.Interface` function. The interface allows users to select **up to** five movies from a checkbox group or enter a movie name in a textbox. Based on the selected movies or entered movie names, the interface calls the `gradio_function` to generate recommendations. The recommended movie posters are displayed as output in the interface.

Using Gradio, we have created an interactive Movie Recommender System interface that enables users to explore and discover new movies based on their preferences.

03

Deploying in hugging face

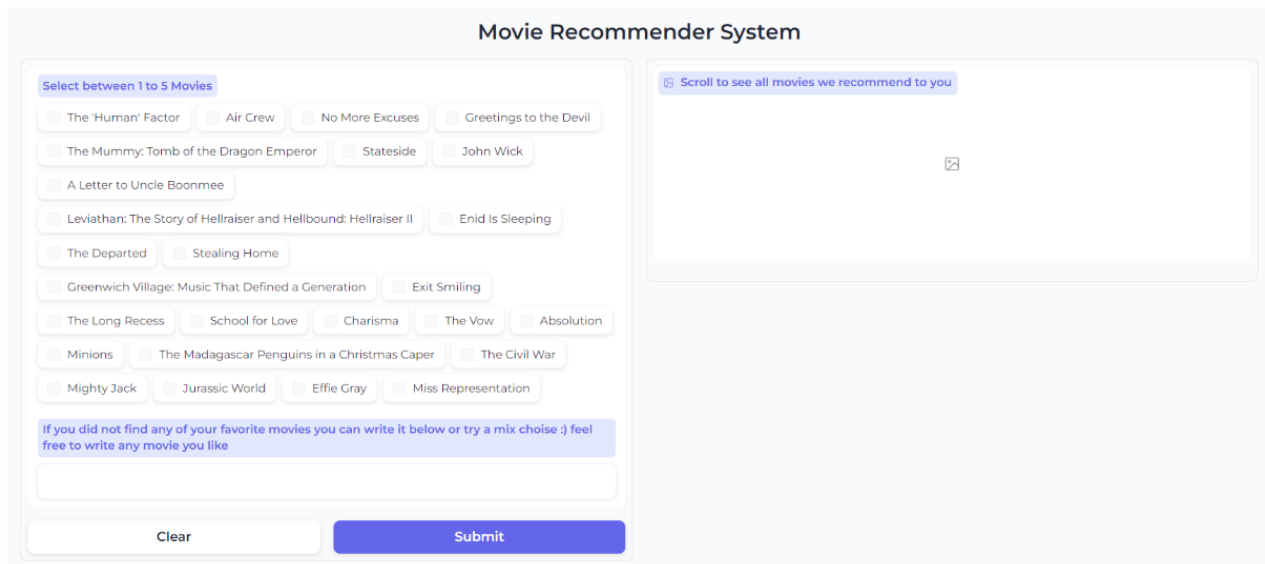
4. Deploying in hugging face

Hugging Face is a **platform** and community focused on natural language processing (NLP) and machine learning. It provides various tools and resources for working with NLP models, including model training, fine-tuning, and deployment.

We **deployed** our movie recommender system **using Hugging Face Spaces** in our project. We uploaded the Gradio code that powers our interactive interface. The code is in the **app.py** file, which serves as our model in this context.

To ensure that the necessary dependencies are installed, we included a section at the top of the code that installs the required packages. This section uses the **subprocess module** to run the command **pip install -r requirements.txt**, which **installs** the **packages** listed in the **requirements.txt** file.

By uploading our Gradio code as a model on Hugging Face Spaces, we make it easily accessible to others. Users can interact with our movie recommender system by utilizing the deployed interface and receiving personalized movie recommendations based on their selections (Fig 4.1 to 4.3).



The screenshot displays the 'Movie Recommender System' web application. On the left, a panel titled 'Select between 1 to 5 Movies' contains a grid of movie titles, each with a radio button for selection. The movies listed are: 'The Human Factor', 'Air Crew', 'No More Excuses', 'Greetings to the Devil', 'The Mummy: Tomb of the Dragon Emperor', 'Stateside', 'John Wick', 'A Letter to Uncle Boonmee', 'Leviathan: The Story of Hellraiser and Hellbound: Hellraiser II', 'Enid Is Sleeping', 'The Departed', 'Stealing Home', 'Greenwich Village: Music That Defined a Generation', 'Exit Smiling', 'The Long Recess', 'School for Love', 'Charisma', 'The Vow', 'Absolution', 'Minions', 'The Madagascar Penguins in a Christmas Caper', 'The Civil War', 'Mighty Jack', 'Jurassic World', 'Effie Gray', and 'Miss Representation'. Below this grid is a text input field with a placeholder message: 'If you did not find any of your favorite movies you can write it below or try a mix choice :) feel free to write any movie you like'. At the bottom of this panel are 'Clear' and 'Submit' buttons. To the right of the selection panel is a large rectangular area with a header 'Scroll to see all movies we recommend to you' and a placeholder image icon, intended for displaying the recommended movies.

Fig 4.4: UI of this project gradio web-app.

Deploying in hugging face

Movie Recommender System

Select between 1 to 5 Movies

☐ The Human Factor ☐ Air Crew ☐ No More Excuses ☐ Greetings to the Devil

☐ The Mummy: Tomb of the Dragon Emperor ☐ Stateside ☐ John Wick

☐ A Letter to Uncle Boonmee

☐ Leviathan: The Story of Hellraiser and Hellbound: Hellraiser II ☒ Enid is Sleeping

☐ The Departed ☐ Stealing Home

☒ Greenwich Village: Music That Defined a Generation ☐ Exit Smiling

☐ The Long Recess ☒ School for Love ☐ Charisma ☐ The Vow ☐ Absolution

☐ Minions ☐ The Madagascar Penguins in a Christmas Caper ☐ The Civil War

☐ Mighty Jack ☐ Jurassic World ☐ Effie Gray ☐ Miss Representation

If you did not find any of your favorite movies you can write it below or try a mix choice 3 feel free to write any movie you like

Narnia

Fig 4.5: Representing the capability of selecting or typing favorite movies as Input.

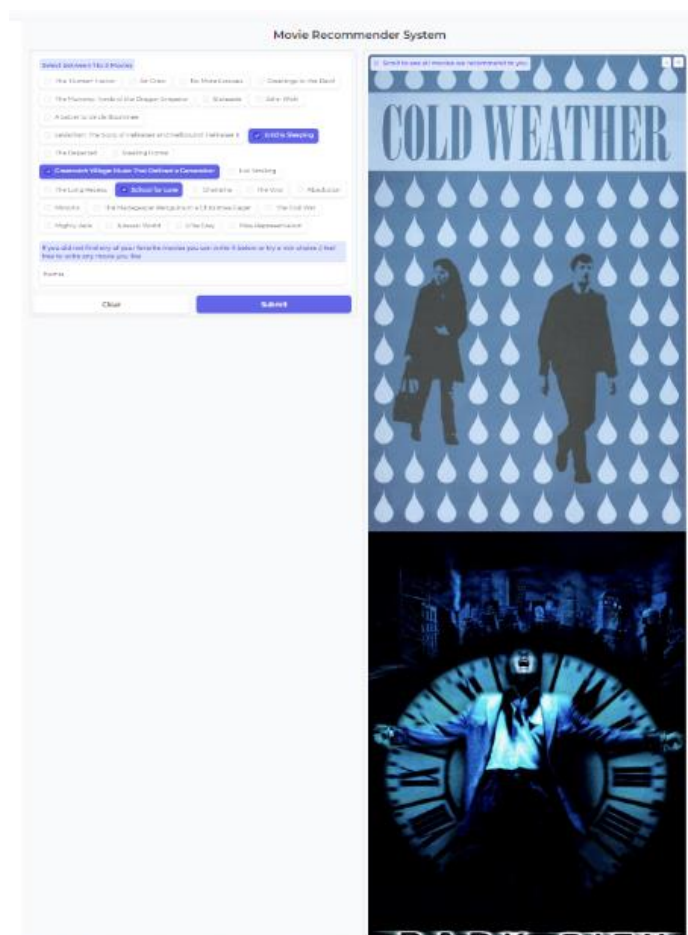


Fig 4.5: Result of choosing withing typing ‘Narnia’ as Input, which can be scrolled to see al 5 recommendation

04

Conclusion

5. Conclusion

A movie recommender system was developed using the clustering method and the Gradio user interface to provide personalized movie recommendations based on user preferences and movie features.

The clustering method was employed to group movies based on their shared characteristics, allowing for the identification of movie similarities and patterns. The K-means algorithm was utilized for clustering, and dimensionality reduction techniques such as PCA were applied to visualize the clusters in 3D and 2D scatter plots.

A user-friendly interface for seamless movie exploration and recommendation was created with the Gradio library. Users could select their favorite movies from a checkbox group or type any favorite movie names in a text box. Based on the selected movies or entered movie names, the system generated recommendations using various functions, including fetching movie posters and genre features from the TMDB API.

The movie recommender system was deployed on Hugging Face Spaces, making it easily accessible to a broader audience. Users could interact with the system through the deployed interface and receive personalized movie recommendations based on their preferences.

As expected, results show that recommended movies seem related to the real world, asking random users and comparing them to movie websites recommending engines.