# CIS 550 – ADVANCED MACHINE LEARNING

# SPRING 2024

# FINAL PROJECT

A black background with red text

Description automatically generated A blue text on a black background

Description automatically generated

Content-Based Movie Recommender System

## 

## Professor:

**Name:** Dr. Ashok Patel

**Office:** Dion 302B

## **Email:** [apatel38@umassd.edu](mailto:apatel38@umassd.edu)

## Group Members:

|  |  |  |  |
| --- | --- | --- | --- |
| **Section** | **R.No** | **Name** | **Student ID** |
| 1 | 43 | Jai Saxena | 02129249 |
| 2 | 40 | Prerak Panwar | 02081587 |
| 1 | 13 | Yi Hsiang Hung | 02081599 |

**Introduction**

In today’s online movie platforms, personalized suggestions are important for making users happy and interested. This project is about making a smart system that suggests movies using machine learning. It looks at what users like, details about movies, and past data to predict and suggest movies each user might enjoy. This will make watching movies more fun and enjoyable for everyone.

This project aims to develop a content-based movie recommender system that leverages content similarity to recommend movies to users based on their input. We will utilize a dataset containing 5000 movie details obtained from Kaggle. The recommendation system will be built using natural language processing techniques, specifically focusing on vectorization, cosine distance calculation, stemming, and building a recommender function.

**Dataset Overview**

The dataset comprises information about 5000 movies, including attributes such as title, genre, keywords, director, cast, etc. This dataset will serve as the foundation for our content-based recommendation system.

Link: https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata

**Types of Recommendation Systems**

We will focus on developing a content-based recommendation system for this project. However, it's worth mentioning that there are other types of recommendation systems:

1. Collaborative filtering
2. Content-Based, and
3. Hybrid systems.

For this project, we will specifically emphasize Content-based recommendations.

**Project Flow**

1. **Data Import**: The first step involves importing the movie dataset from Kaggle.

**Data Preprocessing**: We will preprocess the data by handling missing values, cleaning text data, and extracting relevant features like genres, keywords, and director.

1. **Vectorization**: Text data will be converted into numerical vectors.
2. **Cosine Similarity**: Instead of using Euclidean distance, we will calculate the similarity between movies using cosine distance. This helps in capturing the semantic similarity between movie content.
3. **Stemming**: We will apply stemming to reduce words to their root form, improving the effectiveness of text processing and similarity calculations.
4. **Model Building**: We will build a content-based recommendation model using preprocessed data and cosine similarity calculations.
5. **Website Conversion:** The model will be integrated into a web application where users can input a movie and receive recommendations.

**Deployment**: Finally, the recommender system will be deployed to a web server for real-time usage.

A diagram of a computer

Description automatically generated

**Part-1: Analyzing the Dataset**

To read the dataset from the csv file and create data frame for movies and credits, we use

A screenshot of a computer code

Description automatically generated

To Display the first 5 movies information of the Data Frame **“credits”**.

A screenshot of a computer

Description automatically generated

To display the basic information of data such as datatype, columns, non-null values count

A screenshot of a computer

Description automatically generated

To Display the first 5 movies information of the Data Frame “**movies”**.

A screenshot of a computer

Description automatically generated

**Part-2: Pre-processing the Dataset**

**Data Preprocessing**

1. Load two datasets into DataFrame.
2. Combine two datasets into one.
3. Remove unnecessary columns.
4. Data cleaning on columns which contain missing value.
5. Inspection of duplicate data.
6. Applying data transformation of each column.
7. Combine all the columns into one column.

In the beginning of data preprocessing, we first load in two csv files and save it into the DataFrame we will use later.

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述

To have an efficient data preprocessing, we merge two datasets into one based on the column “title”. Before merging, the two datasets contain 20 and 4 columns. After merging two datasets, the new DataFrame contains 23 columns which is the sum of two datasets minus one duplicate columns “title”.

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述

In all 23 columns, we only needed 7 of them for our model training. We removed all the other unnecessary columns and saved the columns we needed into a new DataFrame.

一張含有 文字, 螢幕擷取畫面, 軟體, 數字 的圖片

自動產生的描述

Before doing data transformation, we inspected each column of dataset to see if there is any missing value. We found that there are 3 missing values in column “overview”. In order to avoid error in our model training, we removed those 3 data with missing values. It can be seen that the original dataset contains 4809 pieces of data and the dataset after applying data cleaning contains 4806 pieces of data.

一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

自動產生的描述

We also inspected if there are duplicate data in the dataset. The result showed that all the data are unique.

一張含有 文字, 軟體, 字型, 數字 的圖片

自動產生的描述

After performing a data cleaning, we started on data transformation. First, we applied a transformation function to column “genres”. In this column, the original data format is “{"id": 28, "name": "Action"}”, the target value we needed in this column is “name”.

一張含有 文字, 字型, 數字, 行 的圖片

自動產生的描述

In column “keywords”, we applied the same function in column “genres”. The original formant of this column is “{"id": 1463, "name": "culture clash"}”, the target value we needed is “name”.

一張含有 文字, 軟體, 螢幕擷取畫面, 數字 的圖片

自動產生的描述

In column “cast”, it contains all the characters of each actor. The original data format is “{"cast\_id": 242, "character": "Jake Sully", "credit\_id": "5602a8a7c3a3685532001c9a", "gender": 2, "id": 65731, "name": "Sam Worthington", "order": 0}”, the target value we needed in this column is “name”, which is the name of actors. Also, in our model training, we only wanted the first three actor’s name, they are the main characters in this movie.

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述

In column “crew”, it contains all the crew members of this film. The original data format is “{"credit\_id": "52fe48009251416c750aca23", "department": "Editing", "gender": 0, "id": 1721, "job": "Editor", "name": "Stephen E. Rivkin"}”, the target value we needed is “job”. Since the director is the most discussed among the crew members of a movie, we wanted only director name.

一張含有 文字, 字型, 數字, 行 的圖片

自動產生的描述

In column “overview”, it contains the introduction to the movie, the type of the data is string. To convert its type into list, we applied a lambda function which split all the words in the string.

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述

In order to avoid confusion, we also applied a lambda function to remove all the space in all the columns.

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述

After applying all the transformations functions, we combined the values in all the columns into one column “tags”.

一張含有 文字, 螢幕擷取畫面, 數字, 字型 的圖片

自動產生的描述

We then created a new DataFrame and saved only three columns which are “movie\_id”, “movie title”, and “tags” into it.

一張含有 文字, 字型, 數字, 行 的圖片

自動產生的描述

We converted the type of the column “tags” in new DataFrame from list to string.

一張含有 文字, 字型, 行, 數字 的圖片

自動產生的描述

The last step in data preprocessing is converting all the characters into lower case.

To visualize we have used matplotlib to display “Genre Distribution in Movie Dataset”.

A screen shot of a graph

Description automatically generated

**Text Vectorization**

Text vectorization is the process of converting text data into numerical vectors that machine learning algorithms can understand. It's a crucial step in natural language processing (NLP) tasks where textual data needs to be transformed into a format suitable for mathematical analysis and modeling.

1. Bag of Words (BoW):
   * Represents text as a collection of words, disregarding grammar and word order.
   * Each document is represented by a vector where each element corresponds to the frequency of a word in the vocabulary.
   * Widely used for text classification, sentiment analysis, and document similarity tasks.
2. TF-IDF:
   * Measures word importance by considering frequency in a document and rarity across the corpus.
   * Emphasizes words specific to a document while downplaying common words.
3. Word Embeddings:
   * Maps words to dense vectors capturing semantic relationships.
   * Represents words in a continuous vector space for NLP tasks like sentiment analysis and language understanding.

**Stemming**

Stemming is like simplifying words down to their basic form, called the "stem." It chops off the ends of words to get to the root or base form. For example, if you have words like "running," "ran," and "runs," stemming will reduce them all to "run."

Here’s a simple example:

* Original Words: running, ran, runs
* Stemmed Words: run, run, run

Stemming is handy because it reduces different forms of the same word to a common base, which can help in tasks like text analysis, where you want to treat similar words the same way.

**Website Creation**

**Pickle and .pkl files**

1-**Serialization**: Pickle serializes Python objects into .pkl files for storage or transmission.

2-**Usage**: Commonly used for saving/loading machine learning models and complex data structures.

3-**File Format**: When you save an object using Pickle, it creates a binary file with the .pkl

extension, which contains the serialized data of the object.

**Streamlit**

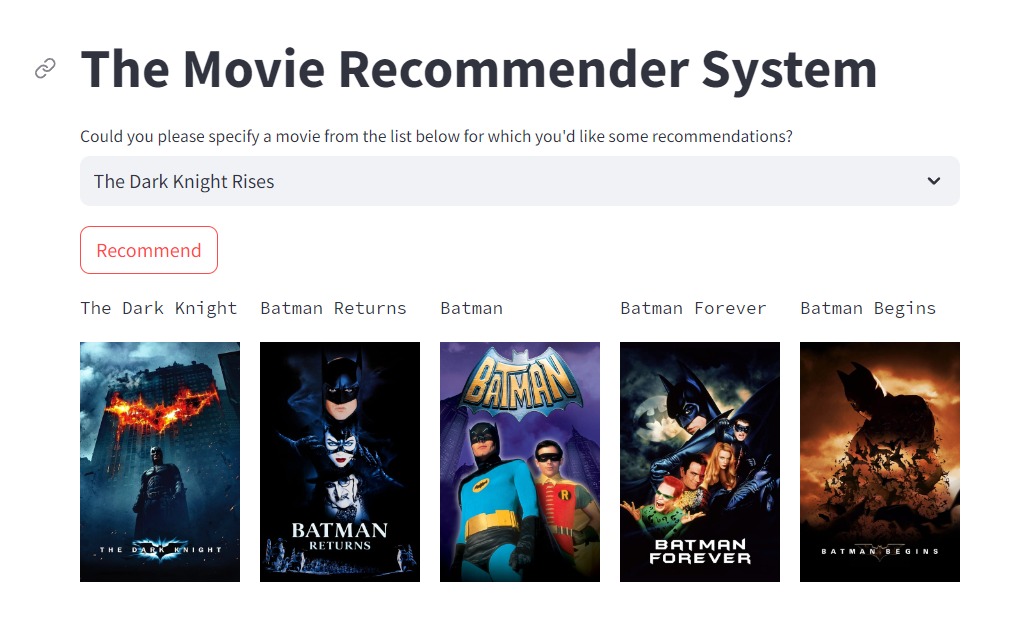
1-**Web Apps**: Streamlit creates interactive web apps directly from Python scripts.

2-**Simplicity**: Easy web app development without needing HTML/CSS/JS knowledge.

3-**Widgets**: Provides widgets for interactive elements like sliders, buttons, and plots.

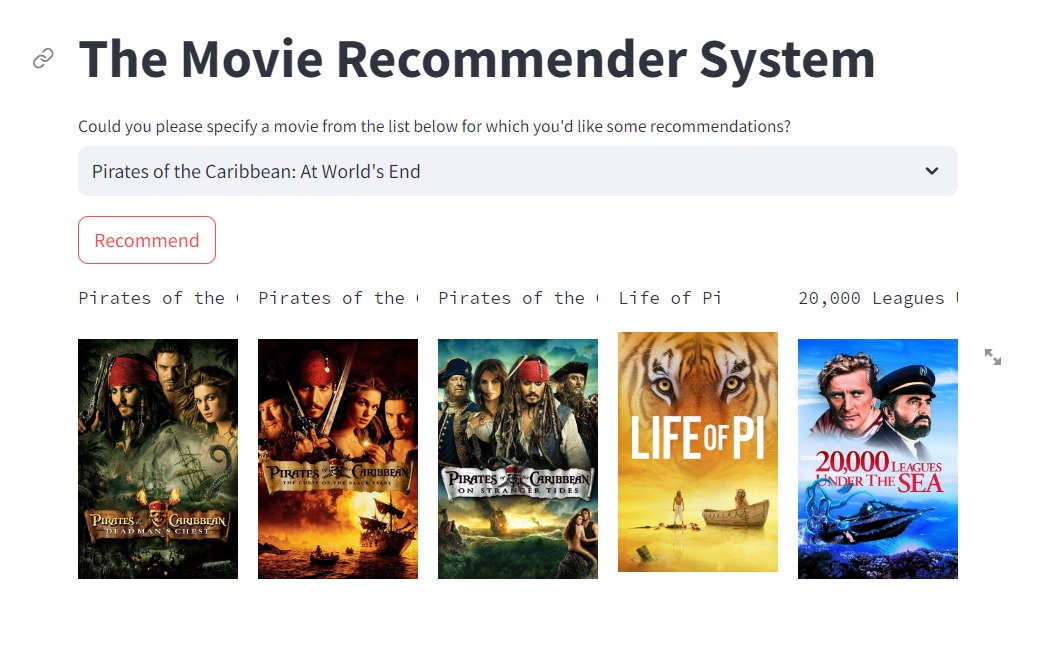
4-**Deployment**: Apps can be deployed locally, on cloud services, or via Streamlit Sharing.

Finally, we have built the movie recommender system.

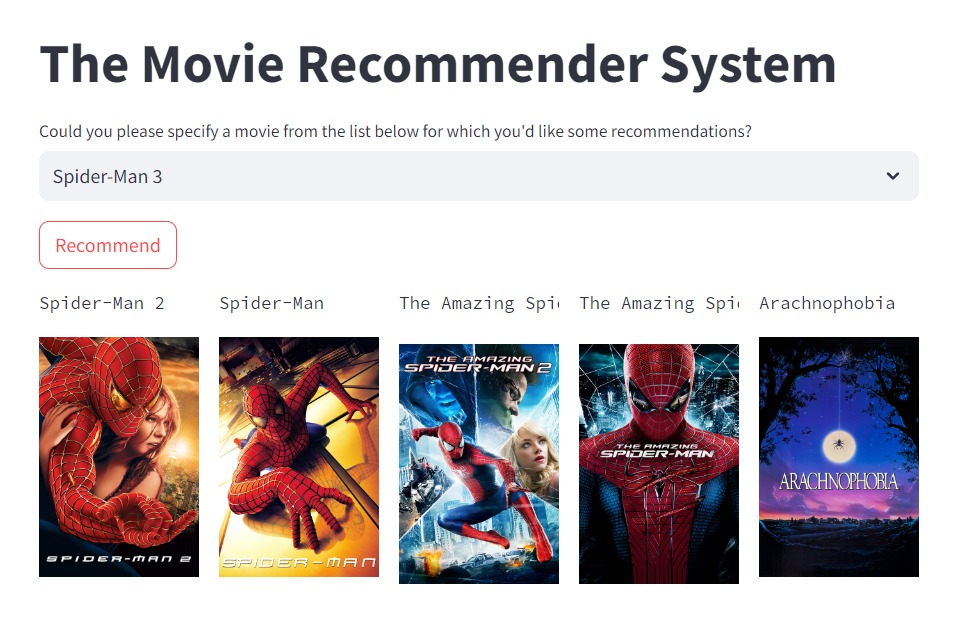


The above searched movie name gave similar recommendations to other Batman movies.

Similarly, in below screenshot, searching for pirates of Caribbean not only gave the other movies in the trilogy as recommendation but also gave other Sea related movies which shows that the recommender system not only picks up movies from the trilogy of a series but also recommends other movies of same relevance like in this case all recommended movies are “**Sea”** based movies.



Similarly, by searching for Spiderman-3, it not only recommended Spider man movies but also a spider-based movie.



**CONCLUSION SUMMARY**

In this project, we learned how to create a content-based movie recommender system using advanced machine learning techniques. Our aim was to develop a system that suggests movies to users based on their preferences and the content of movies they like. Here's what we've learned and achieved:

1. **Understanding Movie Recommendations**: We learned that personalized recommendations are crucial for online movie platforms to keep users engaged and satisfied. By analyzing past data and user preferences, we can predict what movies a user might enjoy.
2. **Content-Based Recommendation System**: We focused on building a content-based recommendation system. This system suggests movies to users based on the similarity of movie attributes such as genre, keywords, and director.
3. **Data Processing and Preprocessing**: We worked with a dataset of 5000 movies, handling missing values, cleaning text data, and extracting relevant features like genres, keywords, and directors. We merged two datasets, removed unnecessary columns, and transformed the data to prepare it for analysis.
4. **Text Vectorization and Stemming**: We converted text data into numerical vectors using techniques like Bag of Words (BoW) and TF-IDF. Additionally, we applied stemming to reduce words to their root form, improving the efficiency of text processing.
5. **Model Building and Integration**: We built a recommendation model based on cosine similarity calculations. This model suggests movies to users based on their input. We also integrated the model into a web application where users can input a movie and receive recommendations in real-time.
6. **Serialization and Web App Development**: We used Pickle to serialize Python objects and Streamlit to create an interactive web application. This application allows users to easily interact with the recommendation system.

In conclusion, we've created an effective content-based movie recommender system that enhances user experience on movie platforms by providing personalized and relevant movie suggestions. This system demonstrates the power of machine learning in understanding user preferences and recommending suitable content.

REFERENCES:

1. DATASET: <https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata>
2. <https://www.imdb.com/chart/top/?ref_=nv_mv_250> – for movie references
3. https://scikit-learn.org/stable/modules/generated/sklearn.feature\_extraction.text.CountVectorizer.html
4. <https://jupyter.org/> **-** JupyterLab for coding
5. [**https://www.jetbrains.com/pycharm/**](https://www.jetbrains.com/pycharm/) **-** pycharm for application code
6. <https://docs.streamlit.io/develop/api-reference>  **-** streamlit for website



