**Movie Recommendation System Based on Cosine Similarity**

### CSD456: Deep Learning Dr. Saurabh Shigwan

**Adeep Sharma**

### Roll No: 2210110116

### November 29, 2024

**Contents**

* 1. [Introduction](#_bookmark0) 2
  2. [Methodology](#_bookmark1) 3
     1. [Data Preprocessing](#_bookmark2) 3
     2. [Machine Learning with Cosine Similarity](#_bookmark3) 4
     3. [Deep Learning with Cosine Similarity](#_bookmark4) 5
  3. [Results and Observations](#_bookmark5) 7
  4. [Dataset Used](#_bookmark6) 10
  5. [Conclusion](#_bookmark7) 11
  6. [References](#_bookmark8) 12

# Introduction

The aim of this project is to build a movie recommendation system based on cosine similarity. Recommendation systems play a vital role in personalizing content for users, and in this case, we explore two techniques: traditional ma- chine learning (ML) and modern deep learning (DL) approaches. Both meth- ods rely on the cosine similarity metric to find similarities between movies based on their features (e.g., tags, descriptions, and embeddings). This re- port presents a detailed comparison of the two approaches using real-world movie data.

# Methodology

This section outlines the methodology followed for building the movie rec- ommendation system using both machine learning (ML) and deep learning (DL) techniques. Both approaches involve the calculation of cosine similarity between movies based on their features, but differ in the methods used to generate these features.

## Data Preprocessing

Data preprocessing is an essential step for preparing the dataset to build any recommendation system. In our project, the data comes from two main CSV files: tmdb 5000 movies.csv and tmdb 5000 credits.csv.

#### Data Cleaning

* + - * **Loading Data**: The raw movie data is loaded into pandas DataFrames using the pd.read csv() function. The movies.csv file contains vari- ous movie features such as movie id, title, tags, cast, crew, overview, etc., while credits.csv provides information on the cast and crew in- volved in each movie.
      * **Handling Missing Values**: We inspect and handle any missing or null values in the dataset to avoid biases during model training. Missing values in columns like overview or tags are filled using an imputation strategy or removed if necessary.

#### Feature Engineering

* + - * **Text Features**: Text columns like overview, tags, and genres are tokenized, cleaned (removing special characters, stop words, etc.), and transformed into vectorized form using methods such as CountVectorizer or TF-IDF. These features will later serve as the basis for cosine simi- larity calculations.
      * **Movie Descriptions**: We concatenate relevant textual information such as tags and overview to create a composite feature represent- ing the movie’s content. This helps in finding more accurate movie similarity based on content.

#### Vectorization

We use CountVectorizer or TfidfVectorizer to convert the textual data (such as movie tags) into numerical vectors that represent the frequency of words. These vectors are then used to compute cosine similarity between the movies.

The mathematical representation of cosine similarity is:

**A** *·* **B**

Cosine Similarity =



**A B**

Where:

* + - * **A** and **B** are the vectorized representations of the movie features.
      * The numerator is the dot product of the two vectors, and the denomi- nator is the product of their magnitudes.

## Machine Learning with Cosine Similarity

In this section, we focus on the machine learning approach for building the movie recommendation system. The basic machine learning approach relies on content-based filtering, where the similarity between a given movie and all others is calculated based on the cosine similarity of their vectorized features. We use traditional machine learning libraries such as scikit-learn for this approach.

#### Libraries Used

The machine learning approach uses the following libraries:

* + - * **Pandas**: A library for data manipulation and analysis, used to load and manage the movie data.
      * **NumPy**: A powerful library for handling arrays and matrices, used for numerical operations.
      * **CountVectorizer (from sklearn.feature extraction.text)**: A method used to convert text data into numerical vectors based on the frequency

of words.

* + - * **Cosine Similarity (from sklearn.metrics.pairwise)**: A function that calculates the cosine similarity between vectors, used to determine the similarity between movies.

#### Methodology

The methodology for the machine learning approach consists of the following steps:

1. **Data Loading**: We load the movie dataset using pandas. The dataset contains various movie details including movie id, title, and tags.
2. **Vectorization of Movie Tags**: We use CountVectorizer to convert the movie tags into numerical vectors. Each movie’s tags are repre- sented as a vector, where each dimension corresponds to a specific word (term) in the dataset’s vocabulary. We limit the number of features to the top 5 most frequent terms and remove stop words (common English words like ”the”, ”is”, etc.).
3. **Cosine Similarity Calculation**: The cosine similarity between all pairs of movies is computed based on their vectorized features. This produces a similarity matrix, where each entry represents the cosine similarity between two movies.
4. **Recommendation System**: For a given query movie (e.g., ”Kung Fu Panda 2”), we extract its vectorized features, compute the cosine sim- ilarity between the query movie and every other movie in the dataset, and return the top-N most similar movies.

## Deep Learning with Cosine Similarity

The deep learning approach builds on top of the traditional machine learn- ing method by using pre-trained models like **Sentence-BERT** to generate richer, more contextual embeddings for the movie features. These embed- dings are used to compute cosine similarity, just as in the machine learn- ing approach, but they capture deeper semantic relationships between the movies.

#### Libraries Used

The deep learning approach uses the following libraries:

* + - * **Pandas**: Used to load and manage the movie dataset.
      * **Sentence-Transformers**: A library for generating sentence embed- dings using pre-trained models like Sentence-BERT.
      * **Cosine Similarity (from sklearn.metrics.pairwise)**: As in the machine learning approach, cosine similarity is used to measure the similarity between movie embeddings.

#### Methodology

The deep learning approach follows these steps:

1. **Data Loading**: The movie dataset is loaded using pandas, as in the ML approach.
2. **Sentence-BERT Embedding Generation**: We initialize the \*\*Sentence- BERT\*\* model (paraphrase-MiniLM-L6-v2) from the SentenceTransformers library. This model generates contextual embeddings for the movie

tags. These embeddings capture the semantic meaning of the movie tags, as opposed to the simple word counts used in the ML approach.

1. **Cosine Similarity Calculation**: Similar to the ML approach, we compute the cosine similarity between the query movie’s embedding and all other movie embeddings. The result is a similarity score for each movie.
2. **Recommendation System**: Given a query movie (e.g., ”Kung Fu Panda 2”), we extract its embedding, compute the cosine similarity with all other movies, and return the top-N most similar movies.

# Results and Observations

The results from the machine learning approach are decent but show limita- tions in terms of capturing the deeper semantic relationships between movies. The deep learning-based recommendations, on the other hand, tend to pro- vide more accurate results as they capture nuanced meanings, even when movies share similar tags or descriptions.

For instance, as illustrated in Figures 1 and 2, the same movie title, **Kung Fu Panda**, was used for comparison. This adventurous animated film revolves around a panda and an imaginative kung fu saga. The ML- based approach primarily recommends other animated movies, which are similar in genre but lack deeper contextual understanding. In contrast, the DL-based approach provides recommendations that not only include similar animated movies but also extend to films with comparable storylines, such as those involving kung fu, ancient Chinese lore, and myths. This highlights the ability of the DL approach to capture nuanced meanings and context behind movie tags and titles.

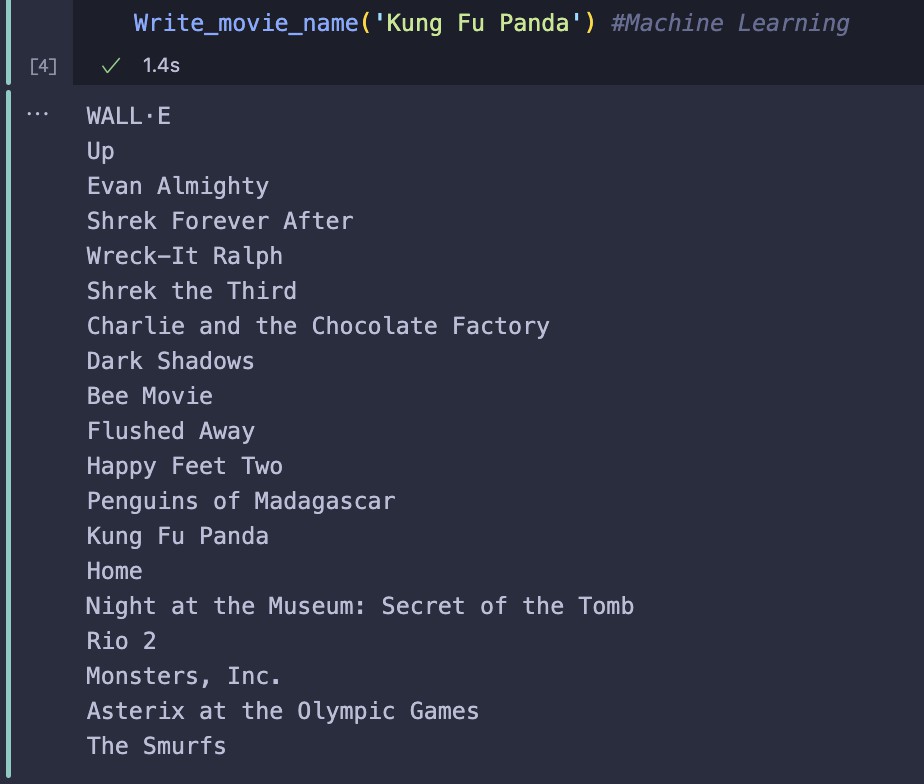


Figure 1: Results for Machine Learning approach



Figure 2: Results for Deep Learning approach

# Dataset Used

The dataset consists of the following files:

* tmdb 5000 movies.csv: Contains information about 5000 movies, in- cluding movie id, title, tags, etc.
* tmdb 5000 credits.csv: Contains the cast and crew information for each movie.

# Conclusion

In conclusion, both the machine learning and deep learning approaches pro- vide useful movie recommendations based on cosine similarity. The machine learning approach is simpler and faster but does not capture the semantic nuances as effectively as the deep learning approach. The deep learning ap- proach, while more computationally intensive, provides more accurate and contextually relevant recommendations. Thus, for a larger and more complex dataset, the deep learning approach is preferable.

# References

1. [Scikit-learn Documentation](https://scikit-learn.org/stable/)
2. [Sentence-Transformers Documentation](https://huggingface.co/sentence-transformers)
3. [Kaggle Dataset](https://www.kaggle.com/datasets)
4. [Research paper - Netflix Movies Recommendation System](https://www.ijisrt.com/netflix-movies-recommendation-system)