**REPORT ON**

**MOVIE RECOMMENDATION SYSTEM**

Submitted in partial fulfilment of the requirements for the award of the degree of

**BACHELOR OF TECHNOLOGY**

**IN**

**ELECTRONICS AND COMMUNICATION**

                                             Submitted by

**Radhika Sharma**

**(T4-01715602822),**

**Mahak Bisht**

**(T4-02215602822)**

**&**

**Mehak Aggarwal**

**(T4-00615602822)**

To

# Dr. Surender Dhiman (HOD-ECE)



**Dr. Akhilesh Das Gupta Institute of Technology & Management (Affiliated to Guru Gobind Singh Indraprastha University, Dwarka, Delhi)**

**ACKNOWLEDGEMENT**

We would like to extend our sincere gratitude to **Dr Surender Dhiman**, our HOD, for his valuable guidance, continuous support, and encouragement throughout this project. His insights and advice have been invaluable in shaping the direction of this research.

We are also grateful to our mentors and fellow peers, whose discussions and feedback helped refine our approach. Lastly, we would like to express our appreciation to **ADGIPS**, where we honed our technical skills and gained experience in **AI/ML technologies**.

**Abstract**

Movie recommendation systems have become an essential tool in the entertainment industry, helping users discover films based on their preferences. This project presents a **Flask-based, machine learning-powered movie recommendation system** that suggests similar movies and lists films directed by the same filmmaker.

The system uses **TF-IDF vectorization** and **cosine similarity** to determine similarity between movies based on attributes like **genres, keywords, taglines, cast, and director**. A **Flask web application** is integrated to provide an interactive user interface. The system also ensures accurate recommendations by employing **string-matching techniques (difflib)** to correct minor spelling errors in user input.

This project demonstrates the **real-world application of AI and machine learning** in content-based filtering. Future improvements could include **deep learning, collaborative filtering, and real-time movie recommendations** based on user preferences.

**Introduction**

### **Overview**

With the increasing number of available movies, finding relevant recommendations based on user preferences has become essential. Our project aims to build a **movie recommendation system using Python, Flask, and machine learning techniques**. The system suggests movies similar to the user’s favourite movie and provides a list of films directed by the same filmmaker.

Project Link: <https://github.com/mehakagg03/Movie-Recommendation-System/tree/main>

### **Objective**

The primary objectives of this project are:

* To build a **Flask-based web application** for movie recommendations.
* To **implement content-based filtering** using **TF-IDF and cosine similarity**.
* To suggest **movies similar to user input** based on textual similarity.
* To provide **recommendations based on the same director**.

**Technologies Used**

* **programming Language:** Python
* **Libraries & Frameworks:**
  + Flask – for building the web application
  + pandas – for data manipulation
  + scikit-learn – for **TF-IDF vectorization** and **cosine similarity**
  + difflib – for **close string matching**
  + requests – for handling HTTP requests (if extended to an API)

**DATASET**

The project uses a **CSV file** containing information about various movies, including:

* **Title:** Name of the movie.
* **Genres:** Categories like Action, Drama, Comedy, etc.
* **Keywords:** Descriptive words related to the movie.
* **Tagline:** A short tagline describing the movie.
* **Cast:** List of main actors.
* **Director:** The filmmaker of the movie.

## ****Methodology****

### **Step 1: Load the Dataset**

The dataset is loaded using pandas, and missing values are filled with an empty string.

python

CopyEdit

import pandas as pd

import difflib

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

movies = pd.read\_csv('movies\_for\_project.csv')

movies.fillna('', inplace=True)

### **Step 2: Feature Engineering (TF-IDF Vectorization)**

A new feature is created by combining multiple attributes like **genres, keywords, tagline, cast, and director**.

python

CopyEdit

combined\_features = movies['genres'] + ' ' + movies['keywords'] + ' ' + movies['tagline'] + ' ' + movies['cast'] + ' ' + movies['director']

vectorizer = TfidfVectorizer()

features\_vectors = vectorizer.fit\_transform(combined\_features)

### **Step 3: Compute Cosine Similarity**

The similarity between movies is computed using **cosine similarity**.

python

CopyEdit

similarity = cosine\_similarity(features\_vectors)

### **Step 4: Finding the Closest Match**

The **difflib** library is used to handle minor spelling errors in user input.

python

CopyEdit

movie\_name = input("Enter your favourite movie name: ")

list\_of\_all\_titles = movies['title'].tolist()

find\_close\_match = difflib.get\_close\_matches(movie\_name, list\_of\_all\_titles)

if find\_close\_match:

close\_match = find\_close\_match[0]

### **Step 5: Movie Recommendation System**

The **30 most similar movies** are retrieved and displayed based on similarity scores.

python

CopyEdit

index\_of\_the\_movie = movies[movies.title == close\_match].index[0]

similarity\_score = list(enumerate(similarity[index\_of\_the\_movie]))

sorted\_similar\_movies = sorted(similarity\_score, key=lambda x: x[1], reverse=True)

for i, movie in enumerate(sorted\_similar\_movies[:30], 1):

print(f"{i}. {movies.iloc[movie[0]]['title']}")

### **Step 6: Fetching Movies by the Same Director**

The system also recommends movies directed by the same filmmaker.

python

CopyEdit

director\_name = movies[movies.title == close\_match]['director'].values[0]

movies\_by\_director = movies[movies.director == director\_name]['title'].tolist()

for j, movie in enumerate(movies\_by\_director, 1):

print(f"{j}. {movie}")

## ****FLASK INTEGRATION****

The system is deployed as a **Flask web application** for user-friendly interaction.

python

CopyEdit

from flask import Flask, request, render\_template

app = Flask(\_\_name\_\_)

@app.route('/', methods=['GET', 'POST'])

def recommend\_movies():

if request.method == 'POST':

movie\_name = request.form['movie\_name']

find\_close\_match = difflib.get\_close\_matches(movie\_name, list\_of\_all\_titles)

if find\_close\_match:

close\_match = find\_close\_match[0]

index\_of\_the\_movie = movies[movies.title == close\_match].index[0]

similarity\_score = list(enumerate(similarity[index\_of\_the\_movie]))

sorted\_similar\_movies = sorted(similarity\_score, key=lambda x: x[1], reverse=True)[:10]

recommended\_movies = [movies.iloc[movie[0]]['title'] for movie in sorted\_similar\_movies]

director\_movies = movies[movies.director == movies.iloc[index\_of\_the\_movie]['director']]['title'].tolist()

return render\_template('index.html', recommendations=recommended\_movies, director\_movies=director\_movies)

return render\_template('index.html')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Results & Discussion**

The system successfully provides two types of recommendations:

1. **Similarity-Based Recommendations** using **cosine similarity**.
2. **Director-Based Recommendations** listing other movies by the same filmmaker.

**Conclusion**

This project demonstrates **machine learning and Flask** integration in building an **AI-powered movie recommendation system**. Future enhancements could include:

* **Deep learning models for better accuracy**.
* **Collaborative filtering for personalized recommendations**.
* **Integration of real-time user reviews and ratings**.