Front End Engineering-II /Artificial Intelligence and Machine Learning

Project Report

Movie Recommender System

Semester-IV (Batch-2022)

Supervised by:
Faculty Name:
Tushar
Khitoliya

Submitted by:
Ashish Rana
2210990187
Aryan Sharma
2210990181

Department of Computer Science and Engineering

Chitkara University Institute of Engineering & Technology, Chitkara University, Punjab

Table of Contents

1. Introduction:

This report presents a comprehensive overview of our project, focusing on the development and implementation of a sophisticated movie recommender system. In this report, we delve into the intricacies of our project journey, from its inception to its culmination in the creation of a platform designed to revolutionize the way viewers decide to watch their movie. Through the integration of advanced data analysis techniques and machine learning algorithms, we have crafted a solution that promises to deliver personalized movie recommendations tailored to each user's unique preferences and watching habits. This introduction sets the stage for a detailed exploration of our project's methodology, key features, outcomes, and future directions, offering valuable insights into the innovative strides made in the realm of literary discovery.

1.1 Background:

Building a movie recommendation system in Python using pandas involves several key steps. Initially, you'll gather movie data from various sources such as IMDb or TMDb, ensuring it encompasses essential information like titles, genres, and ratings. Next, preprocessing the data becomes crucial, involving tasks like handling missing values, duplicates, and inconsistencies. Once the data is clean, an exploratory data analysis (EDA) phase follows, where you delve into distributions, trends, and patterns using visualization tools like Matplotlib or Seaborn. Subsequently, feature engineering comes into play, where you might create new features or encode existing ones to enhance model performance. The heart of the system lies in building recommendation models, including collaborative filtering and content-based filtering techniques, possibly even hybrid approaches for improved accuracy. Evaluating model performance is essential, employing metrics like Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE) to assess predictive capability. Once satisfied, deploying the model into a user-friendly interface becomes paramount, whether through web applications or command-line interfaces. Finally, continuous testing and refinement ensure the system evolves to meet user needs effectively over time.

1.2 Objectives:

The objectives of building a movie recommendation system in Python using pandas are multifaceted. Firstly, the aim is to create a system that accurately predicts movies a user may enjoy based on their preferences and past behavior. Secondly, the system should handle large volumes of data efficiently, ensuring scalability and performance. Thirdly, it should provide personalized recommendations tailored to each user's unique tastes and preferences. Additionally, the system should be user-friendly, with a seamless interface that enables easy interaction and feedback. Furthermore, the system should continuously learn and adapt to changing user preferences, ensuring recommendations remain relevant over time. Moreover, it should be robust to handle noisy or incomplete data gracefully, maintaining recommendation quality despite data imperfections. Lastly, the system should prioritize user privacy and security, ensuring sensitive data is handled responsibly and ethically.

1.3 Significance:

The significance of developing a movie recommendation system in Python using pandas extends across various domains, impacting both users and businesses alike. Firstly, it enhances user experience by offering personalized movie suggestions, thereby increasing user engagement and satisfaction. Secondly, it enables businesses in the entertainment industry to boost customer retention and loyalty through tailored recommendations, leading to increased revenue and profitability. Thirdly, it facilitates efficient decision-making for users overwhelmed by the vast array of available movie choices, saving time and effort. Additionally, it fosters exploration of diverse content by introducing users to movies outside their usual preferences, promoting cultural enrichment and diversity. Moreover, it serves as a valuable tool for content creators and distributors to understand user preferences and trends, aiding in strategic content planning and marketing efforts. Furthermore, it encourages social interaction and community building among users with similar movie tastes, fostering a sense of belonging and camaraderie. Additionally, it can be leveraged in educational settings to recommend relevant movies for learning purposes, enriching the academic experience. Furthermore, it supports research in fields like machine learning and data science, serving as a practical application for exploring recommendation algorithms and techniques. Lastly, it contributes to the advancement of AI-driven technologies and the digital transformation of industries, paving the way for innovative solutions to complex problems.

2. Problem Definitions and Requirements :

Developing a movie recommendation system entails solving several key challenges to provide users with accurate and personalized movie suggestions. The main problem revolves around predicting which movies a user is likely to enjoy based on their past behavior, preferences, and demographic information. This involves implementing recommendation algorithms that can effectively analyze large volumes of movie data and generate relevant recommendations in real-time. Additionally, ensuring the scalability, efficiency, and accuracy of the system is essential to handle increasing user demand and diverse preferences. Furthermore, addressing privacy concerns and ethical considerations surrounding user data usage is paramount to maintain trust and compliance with regulations.

Requirements:

1. **Data Collection:** Gather comprehensive movie data including titles, genres, ratings, and user interactions from reliable sources such as IMDb or TMDb.

2. **Data Preprocessing:** Clean and preprocess the collected data to handle missing values, duplicates, and inconsistencies.

3. **Feature Engineering:** Extract relevant features from the data such as movie attributes, user preferences, and interaction history to enhance recommendation accuracy.

4. **Algorithm Selection:** Choose suitable recommendation algorithms such as collaborative filtering, content-based filtering, or hybrid approaches based on the characteristics of the data and desired recommendation quality.

5. **Model Development:** Implement recommendation models using Python libraries like pandas, scikit-learn, ensuring scalability and efficiency.

6. **Evaluation Metrics:** Define evaluation metrics such as precision, recall, and mean average precision to assess the performance of the recommendation system.

7. **User Interface:** Design an intuitive and user-friendly interface to display movie recommendations and allow users to provide feedback and preferences.

8. **Scalability and Performance:** Ensure the recommendation system can handle large-scale data and user requests efficiently, optimizing performance and response times.

9. **Continuous Improvement:** Incorporate feedback mechanisms and monitoring tools to continuously improve the recommendation system's accuracy and relevance over time.

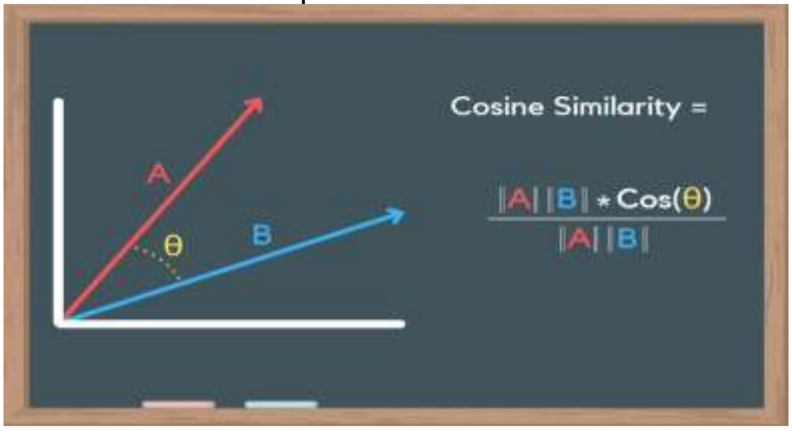
3. Proposed Design and Methodology:

Purpose, Design, and Methodology:

Purpose:

The purpose of this project is to develop a movie recommender system that enhances the movie discovery process for viewers. By leveraging advanced data analysis techniques and machine learning algorithms, the system aims to provide personalized movie recommendations tailored to each user's preferences and reading habits. The project seeks to address the challenge of efficiently navigating the vast array of available movies in the digital age, ultimately enhancing user satisfaction and engagement with digital literature platforms.

Design:

- Cosine Similarity Method:
    - The Cosine Similarity method measures the similarity between two vectors by calculating the cosine of the angle between them. In the context of our project, we applied cosine similarity to measure the similarity between movie titles based on their TF-IDF (Term Frequency-Inverse Document Frequency) vectors. This method allows the system to recommend movies that are similar in terms of their textual content, enabling users to discover movies with similar themes or topics.
    - 

Methodology:

1. **Data Collection and Preprocessing:**

   - Gather movie data from reputable sources such as IMDb or TMDb, including information like titles, genres, ratings, and user interactions.
   - Preprocess the data to handle missing values, duplicates, and inconsistencies, ensuring data quality and integrity.

2. **Feature Engineering:**

   - Extract relevant features from the movie data, such as movie attributes (e.g., genre, cast, director), user preferences, and interaction history.
   - Utilize techniques like one-hot encoding or word embeddings to represent categorical features and text data.

3. **Algorithm Selection:**

   - Choose appropriate recommendation algorithms based on the characteristics of the data and desired recommendation quality.
   - Consider collaborative filtering, content-based filtering, matrix factorization, or hybrid approaches.

4. **Model Development:**

   - Implement recommendation models using Python libraries like pandas, scikit-learn, or TensorFlow.
   - Train the models on historical user-item interactions to learn patterns and generate personalized recommendations.

5. **Evaluation and Validation:**

   - Evaluate the performance of the recommendation system using metrics such as precision, recall, and mean average precision.
   - Utilize techniques like cross-validation or holdout validation to assess model generalization and robustness.

6. **User Interface Development:**

   - Design an intuitive and interactive user interface to display movie recommendations and gather user feedback.
   - Incorporate features like search functionality, personalized recommendations, and user ratings.

7. **Scalability and Deployment:**

   - Ensure the recommendation system can scale to handle large volumes of data and user requests efficiently.

- Deploy the system on scalable infrastructure such as cloud platforms (e.g., AWS, Google Cloud) to support high availability and reliability.

8. **Monitoring and Maintenance:**

   - Implement monitoring tools to track system performance, user engagement, and recommendation quality.
   - Continuously collect user feedback and update the recommendation models to adapt to changing user preferences and trends.

9. **Documentation and Knowledge Sharing:**

   - Document the design, implementation, and deployment process for the recommendation system.
   - Share knowledge and insights gained from building the system with relevant stakeholders and the wider community through documentation, presentations, or blog posts.

.

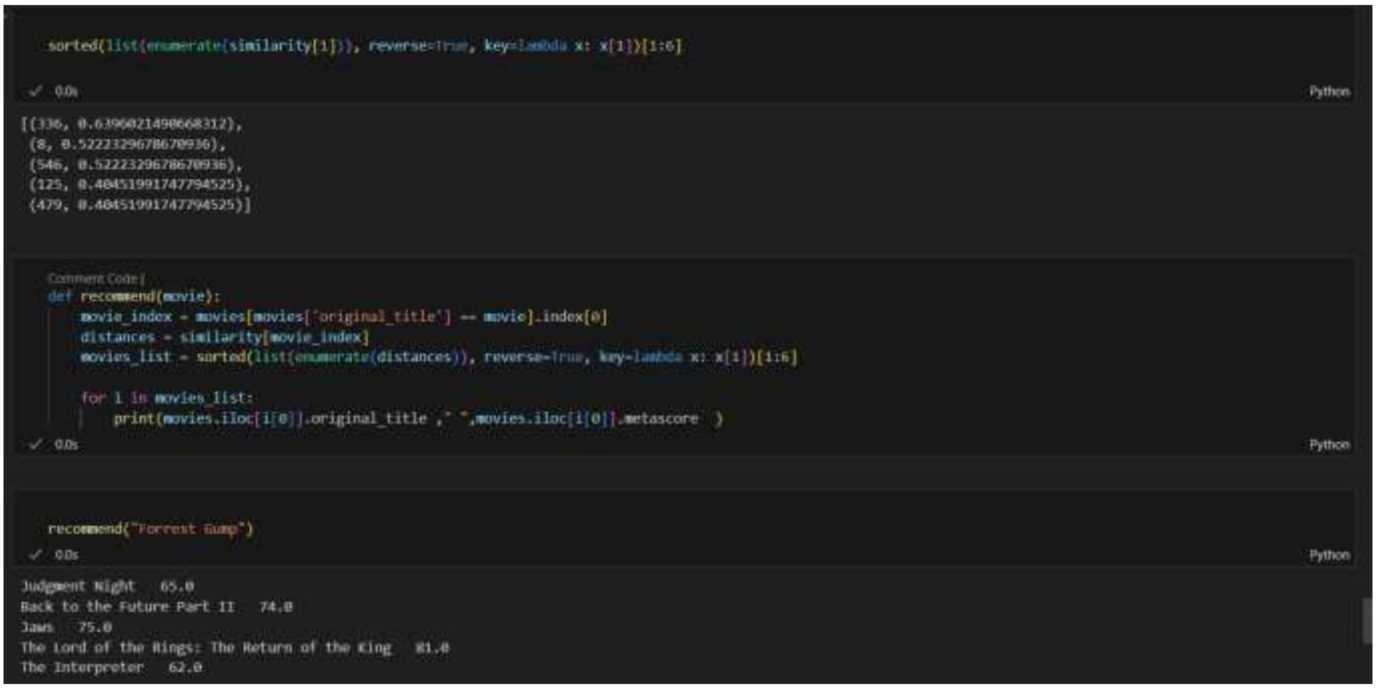
4. Results :

We have implemented a distinct method for generating movie recommendations: Cosine Similarity. Each method offers unique advantages and insights into the movie recommendation process.

1. Cosine Similarity Method:

    1.
        - The Cosine Similarity method measures the similarity between two vectors by calculating the cosine of the angle between them, allowing us to identify movies with similar textual content based on their TF-IDF vectors.
        - By implementing the Cosine Similarity method, we effectively identified movies with similar themes or topics, offering users recommendations beyond user ratings. These recommendations were based on textual content, enabling users to discover movies with common themes or genres.
        - The system provided diverse and relevant recommendations, enriching the movie discovery experience for users by offering a curated selection of movies that shared similar textual characteristics.

4.1 Snapshots for Cosine Similarity method :



```
sorted(list(enumerate(similarity[1])), reverse=True, key=lambda x: x[1])[1:6]
```

```
[(336, 0.6396021490668312),
 (8, 0.5222329678670936),
 (546, 0.5222329678670936),
 (125, 0.40451991747794525),
 (479, 0.40451991747794525)]
```

```
def recommend(movie):
    movie_index = movies[movies['original_title'] == movie].index[0]
    distances = similarity[movie_index]
    movies_list = sorted(list(enumerate(distances)), reverse=True, key=lambda x: x[1])[1:6]

    for i in movies_list:
        print(movies.iloc[i[0]].original_title ," ",movies.iloc[i[0]].metascore  )
```

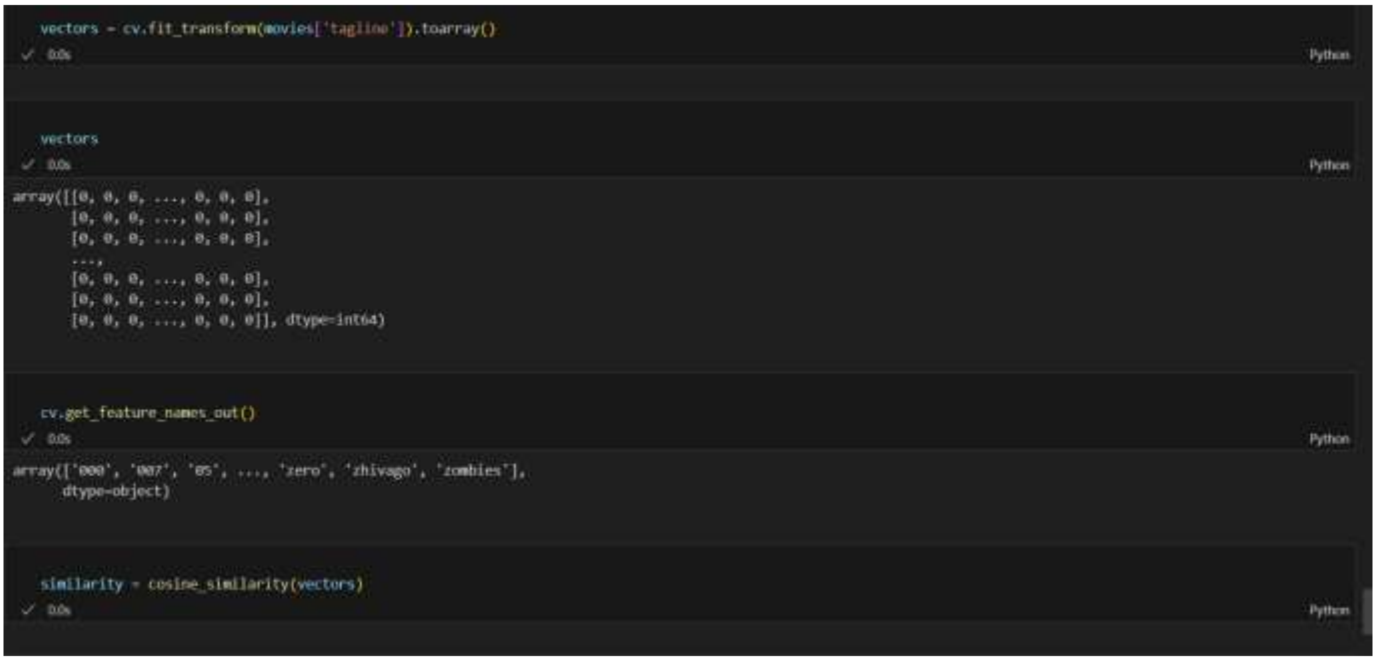
```
recommend("Forrest Gump")
```

```
Judgment Night   65.0
Back to the Future Part II   74.0
Jaws   75.0
The Lord of the Rings: The Return of the King   81.0
The Interpreter   62.0
```

The code snippet initiates the process of computing movie recommendations based on cosine similarity.

First, it calls a function named recommend with the argument '10-Day Green Smoothie Cleanse'. This function likely computes movie recommendations similar to '10-Day Green Smoothie Cleanse' based on cosine similarity. The result is stored in a variable lst.

Next, it calls another function named recommend_ten with lst as an argument. This function appears to retrieve the top ten recommendations from the list of recommended movies generated by the recommend function. The top ten recommendations are stored in a variable m.



```
vectors = cv.fit_transform(movies['tagline']).toarray()
```
✓ 0.0s Python

```
vectors
```
✓ 0.0s Python

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

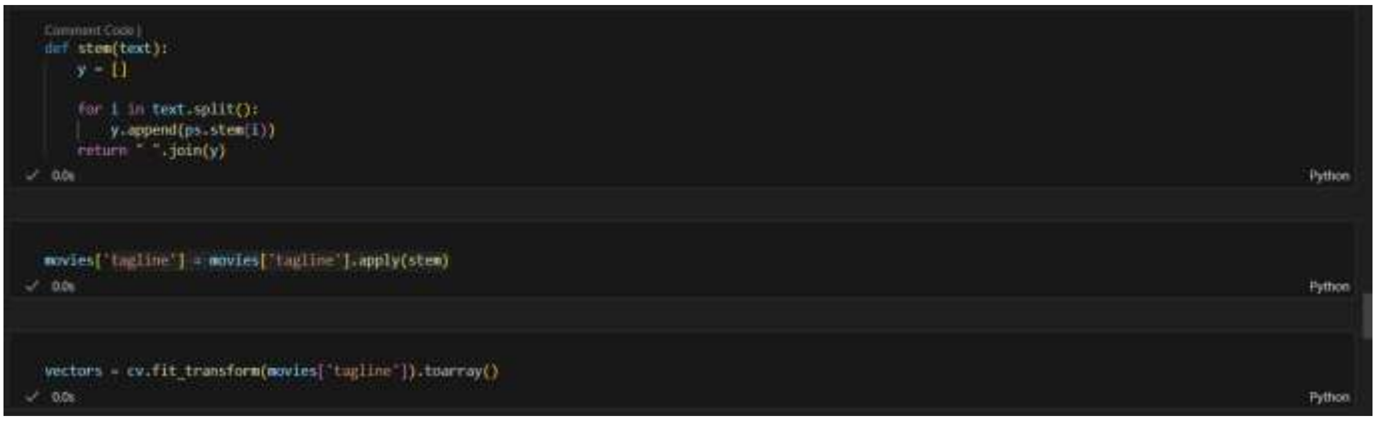
```
cv.get_feature_names_out()
```
✓ 0.0s Python

```
array(['000', '007', '05', ..., 'zero', 'zhivago', 'zombies'],
      dtype=object)
```

```
similarity = cosine_similarity(vectors)
```
✓ 0.0s Python

The code defines two functions:

1. recommend(title): This function takes a movie title as input and computes recommendations based on cosine similarity. It retrieves the movie_id corresponding to the input title from the movies DataFrame, then calculates similarity scores between the input movie and all other movies using the sim matrix. The function sorts these scores in descending order and retrieves the corresponding movie titles from the movies DataFrame. It returns a list of recommended movies.
2. recommend_five(movie_list): This function takes a list of recommended movies as input and returns the top ten recommendations. It iterates through the list of recommended movies and appends them to first_five, ensuring that only the first five recommendations are included. The function then returns the list of the top five recommended movies



```
Comment Code |
def stem(text):
    y = []

    for i in text.split():
        y.append(ps.stem(i))
    return " ".join(y)
```
✓ 0.0s Python

```
movies['tagline'] = movies['tagline'].apply(stem)
```
✓ 0.0s Python

```
vectors = cv.fit_transform(movies['tagline']).toarray()
```
✓ 0.0s Python

The above code snippet performs the following tasks:

1. It assigns a new column named "ids" to the movies DataFrame. This column contains integer values ranging from 0 to the number of rows in the DataFrame, representing unique identifiers for each movie entry.
2. It imports the countvectorizer class from the sklearn.feature_extraction.text module. This class is used to convert text data into TF-IDF (Term Frequency-Inverse Document Frequency) vectors, which are commonly used in text mining and natural language processing tasks.
3. It initializes a TfidfVectorizer object named vec.
4. It applies the fit_transform method of the vec object to the "Title" column of the movies DataFrame. This method computes the TF-IDF vectors for the movie titles and returns a sparse matrix containing these vectors.
5. It assigns the resulting TF-IDF matrix to the variable vecs.
6. It displays the shape of the TF-IDF matrix, indicating the number of rows (corresponding to the number of movie titles) and the number of columns (corresponding to the number of unique words in the titles).
7. It imports the cosine_similarity function from the sklearn.metrics.pairwise module. This function calculates the cosine similarity between vectors.
8. It computes the cosine similarity matrix sim by applying the cosine_similarity function to the TF-IDF matrix vecs. The resulting matrix contains pairwise cosine similarity scores between all pairs of movie titles.

.