



PROJECT REPORT

TITLE:

CONTENT BASED MOVIE RECOMMENDATION SYSTEM

2025

BY

G RAMESHKANNAN

I M. Sc. COMPUTER SCIENCE

TABLE OF CONTENTS

S. No.	Title	Page Number
1.	Abstract	1
2.	Introduction	2
3.	Literature Review	3
4.	System Design	5
5.	Implementation	9
6.	Result and Testing	15
7.	Conclusion and Future Work	18
8.	References	19

1. Abstract:

In an age of overwhelming content availability, movie recommendation systems have become essential tools to enhance user experience by suggesting relevant and personalized content. This project presents a **Content-Based Movie Recommendation System (CBMRS)** that recommends similar movies based on their plot overviews using natural language processing and machine learning techniques.

The system utilizes the **TMDB 5000 Movies Dataset**, which includes metadata such as movie titles, genres, popularity, and textual overviews. The core methodology involves converting these textual descriptions into numerical feature vectors using the **TF-IDF (Term Frequency-Inverse Document Frequency)** vectorizer. This approach effectively captures the significance of words in each movie's overview, allowing the system to distinguish between different plot themes.

To identify relationships between movies, **cosine similarity** is computed on the TF-IDF matrix. This metric quantifies the similarity between movies based on the angle between their TF-IDF vectors, enabling the system to rank movies by content relevance. To enhance the accuracy and efficiency of recommendations, **KMeans clustering** is employed to group similar movies into thematic clusters. Recommendations are generated by comparing the input movie only with others in the same cluster, thereby improving contextual relevance.

The application is deployed using **Streamlit**, offering a clean and interactive user interface. Users can input any movie title from the dataset and receive a list of top five content-based recommendations within the same cluster. The system handles invalid inputs gracefully and is designed for real-time responses.

This project demonstrates how combining vector-based text analysis with clustering techniques can lead to efficient and meaningful recommendations. The modular structure of the CBMRS makes it suitable for future extensions, including hybrid recommendation strategies and user preference learning.

2. Introduction

With the explosive growth of digital content, users are often overwhelmed by the sheer number of options available on streaming platforms. Navigating through thousands of movies to find something interesting or similar to past favorites can be time-consuming and frustrating. To alleviate this, recommendation systems have become a vital component of modern media platforms. They not only improve user experience but also significantly enhance platform engagement by offering tailored suggestions. This project presents a **Content-Based Movie Recommendation System (CBMRS)** that uses natural language processing (NLP), machine learning, and clustering algorithms to provide intelligent movie recommendations based on plot similarities.

The CBMRS focuses on recommending movies that are similar in content to a given title, primarily using the textual “**overview**” of each film. The **TMDB 5000 Movies Dataset** is used, which includes metadata like title, genres, popularity, release date, vote averages, and plot overviews. However, the recommendation engine primarily relies on the "overview" field to understand and compare the narrative of each film.

To process the movie descriptions, the system uses **TF-IDF (Term Frequency-Inverse Document Frequency)** vectorization. This technique transforms the textual overviews into numerical vectors by assigning higher weights to unique and meaningful terms in each document while minimizing the importance of common words (stop words). This conversion enables the system to perform mathematical operations on text data.

Once the overviews are vectorized, **cosine similarity** is computed to measure how close each movie is to every other movie in the dataset. Cosine similarity works by measuring the cosine of the angle between two vectors, offering an effective way to compare text-based data.

To improve the performance and contextual accuracy of recommendations, the system incorporates **KMeans clustering**. This unsupervised learning algorithm groups movies into a fixed number of clusters based on their TF-IDF vectors. Each cluster tends to represent a thematic group of movies that share similar vocabulary and storytelling patterns. Recommendations are drawn from the same cluster as the selected movie, ensuring that the suggestions are contextually relevant and not arbitrarily chosen from the entire dataset.

The system is built with modularity and user experience in mind. It uses **Streamlit**, a Python-based web application framework, to deploy a simple yet effective user interface. Users can enter the name of a movie they've watched, and the system returns five content-based recommendations from the same cluster. If the movie title is not found, appropriate error messages are displayed to guide the user.

Unlike collaborative filtering approaches that require user interaction history or ratings, this system relies solely on content-based features, making it highly effective even for new or unrated items. It is especially useful for cold-start scenarios where user data is unavailable.

Overall, this project showcases a scalable and efficient method for content-based recommendation by combining text mining, clustering, and real-time application deployment. The system is built to be extensible and adaptable for future enhancements such as integrating genre-based filters, user profiles, or combining with collaborative filtering for hybrid recommendations.

3. Literature Review

Recommendation systems have evolved significantly over the past few decades and are now widely used across various domains such as e-commerce, streaming platforms, social networks, and news delivery services. The primary objective of these systems is to help users discover relevant items—such as products, articles, or movies—based on their preferences or behavior. This chapter explores the different types of recommendation systems and relevant research that form the foundation of this project.

Recommendation systems are broadly categorized into three types: **collaborative filtering**, **content-based filtering**, and **hybrid approaches**. Collaborative filtering relies on user interactions, such as ratings or clicks, to identify similarities between users or items. Although effective, this method suffers from the **cold-start problem** where it struggles to make recommendations for new users or items with insufficient interaction data.

To overcome this limitation, **content-based filtering** approaches were introduced. These systems recommend items based on the attributes of the items themselves. In the context of movies, these attributes may include plot summaries, genres, cast, crew, and more. Content-based models analyze the description of a movie and compare it with other movies based on feature

similarity. This project specifically follows this method by focusing on movie plot summaries for making recommendations.

Term Frequency-Inverse Document Frequency (TF-IDF) has been a widely adopted technique in information retrieval and natural language processing. TF-IDF transforms text into numerical vectors that reflect how important a word is to a document relative to a collection of documents. Its effectiveness in distinguishing documents based on content makes it highly suitable for text-based recommendation systems. Numerous academic and industry-grade applications have leveraged TF-IDF for content analysis, such as in the works of Salton et al., which laid the foundation for vector space models in information retrieval.

Cosine similarity is another important concept frequently used in text mining and recommendation systems. It calculates the cosine of the angle between two non-zero vectors, which in this context represent movies. This metric is highly efficient in high-dimensional sparse data such as TF-IDF matrices, making it a standard choice for measuring textual similarity.

Recent advancements in recommendation systems also include the use of **clustering algorithms** to improve accuracy and reduce computational complexity. **KMeans clustering** is a commonly used unsupervised learning technique that groups similar items together based on feature similarity. By applying KMeans on TF-IDF vectors, systems can narrow down recommendations to items within the same cluster, enhancing thematic consistency. This concept is effectively employed in this project to limit comparisons to a relevant subset of movies.

Lastly, tools like **Streamlit** have enabled rapid development and deployment of machine learning applications with interactive user interfaces. Research and development efforts increasingly emphasize not just algorithmic performance but also usability and real-time responsiveness—both of which are addressed in this project.

In conclusion, this project builds upon proven methods in content-based filtering, TF-IDF, cosine similarity, and clustering, while offering a modern, interactive interface. The system benefits from established academic theories and practical tools to create a robust and user-friendly movie recommendation engine.

4. System Design

The system design of the Content-Based Movie Recommendation System (CBMRS) follows a modular and layered architecture. It ensures efficient data processing, logical separation of concerns, and scalability for future enhancements. This chapter presents the architectural flow of the system, key modules, and the interaction between components.

4.1 System Architecture

The architecture of CBMRS is divided into the following main layers:

1. Data Layer

- Handles data loading and preprocessing from the TMDB 5000 Movies dataset.
- Extracts relevant columns such as title, overview, genres, etc.
- Cleans missing values and prepares data for feature extraction and clustering.

2. Feature Engineering Layer

- Utilizes TF-IDF Vectorization to convert movie overviews into numerical vectors.
- Extracts genre metadata and applies one-hot encoding for improved clustering.
- Normalizes numerical features such as vote average and popularity.

3. Clustering Layer

- Applies KMeans clustering on the TF-IDF matrix to group movies into content-based clusters.

- Each movie is assigned a cluster label which helps limit similarity comparison to relevant groups.

4. Similarity Computation Layer

- Computes cosine similarity between movie vectors.
- Builds a similarity matrix for quick lookup of similarity scores.

5. Recommendation Engine

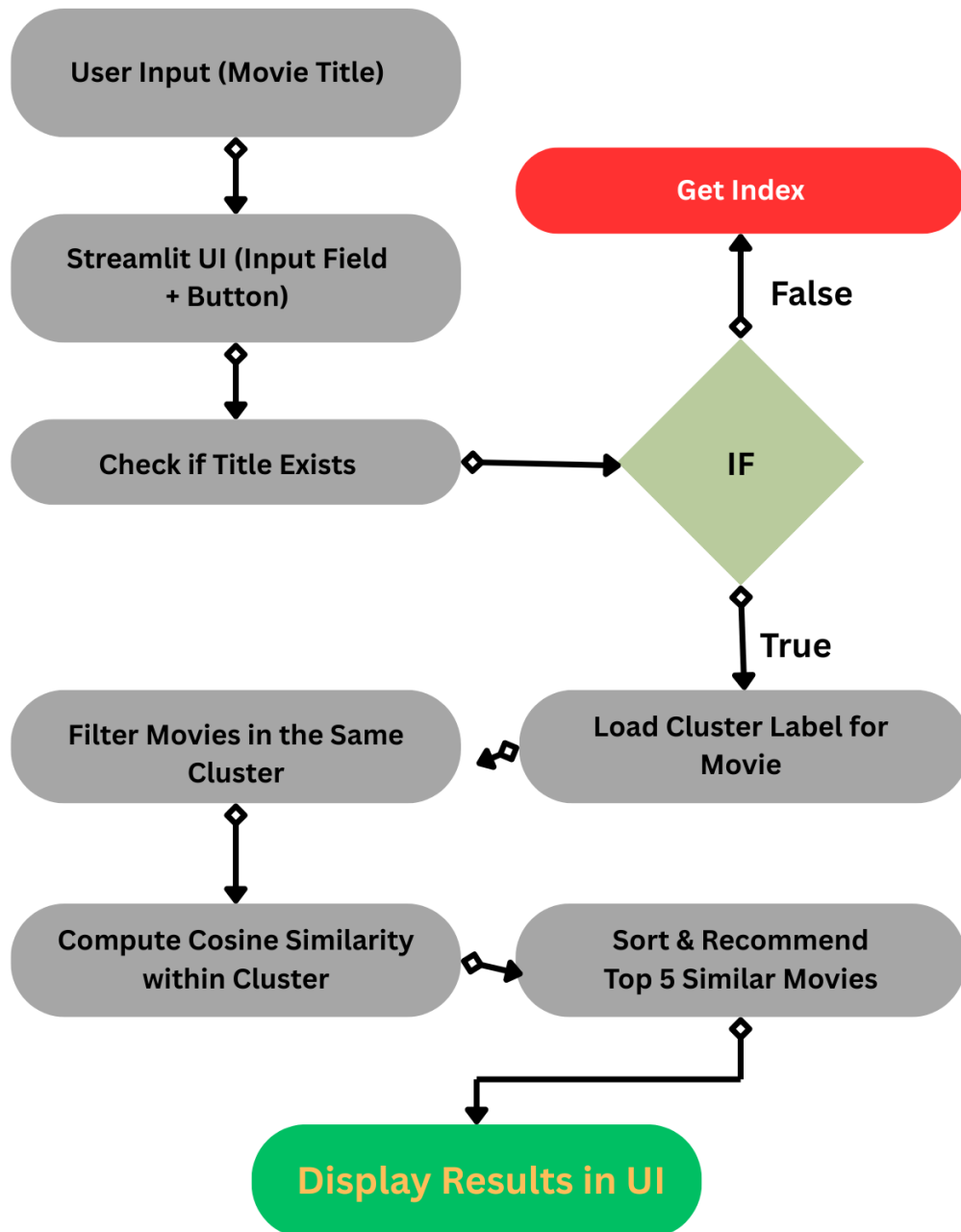
- Accepts user input (movie title).
- Identifies the cluster for the given movie.
- Filters other movies in the same cluster and ranks them based on cosine similarity.
- Returns the top 5 most similar movie titles.

6. User Interface Layer (Streamlit App)

- Built using Streamlit to provide an interactive UI.
- Accepts user input and displays recommendations dynamically.
- Shows error messages for invalid inputs or missing titles.

4.2 Data Flow Diagram:

FLOW DIAGRAM



4.3 Module Overview

Module Name	Functionality Description
data_loader.py	Loads and preprocesses the TMDB dataset
feature_extraction.py	Converts movie overviews into TF-IDF vectors
feature_extraction.py	Converts movie overviews into TF-IDF vectors
clustering.py	Applies KMeans and returns cluster labels
similarity.py	Computes pairwise cosine similarity for all movie vectors
recommender.py	Core logic for fetching and sorting similar movies based on similarity
app.py (Streamlit)	Integrates all modules, builds UI, and handles user interaction

4.4 Advantages of Modular Design

- Scalability: Each module can be upgraded independently (e.g., swap TF-IDF with BERT).
- Readability & Maintainability: Clear separation of logic improves collaboration and debugging.
- Performance: Clustering reduces unnecessary computations across the entire dataset.

This modular and layered design makes the CBMRS system both efficient and extensible for future improvements such as hybrid models, real-time feedback, or collaborative filtering integration.

5. Implementation

The implementation of the Content-Based Movie Recommendation System (CBMRS) is centered around processing textual information from a movie database, transforming it into numerical representations, clustering similar movies, and delivering personalized recommendations through a user-friendly web interface. The project is implemented using Python due to its rich ecosystem of libraries for data science and web application development. This chapter outlines the systematic construction of the system, from data loading to recommendation delivery.

5.1 Tools and Technologies Used

- **Python:** The core language used for development.
- **Pandas:** For data loading, manipulation, and cleaning.
- **NumPy:** For handling numerical operations.
- **Scikit-learn:** For vectorization (TF-IDF), similarity computation (cosine_similarity), and clustering (KMeans).
- **Streamlit:** For building a lightweight, interactive web interface.
- **VS Code:** Used during initial development and testing phases.

These tools offer flexibility, efficiency, and integration necessary for a content-based filtering system.

5.2 Data Loading and Preprocessing

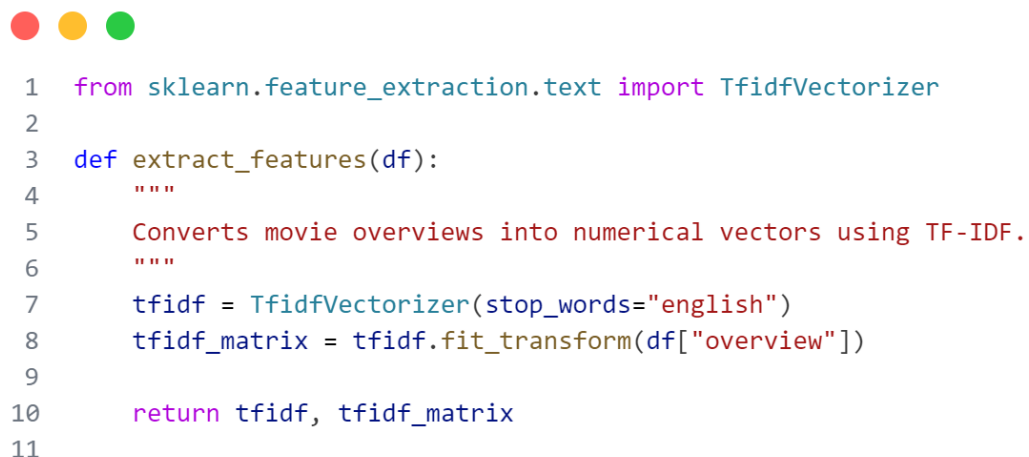
The system uses the TMDb 5000 Movies Dataset, which contains metadata such as movie overviews, genres, cast, and popularity metrics. Initially, a function named `load_data()` is used to extract core features like `id`, `title`, and

`overview`. All missing values are filled with empty strings to prevent processing errors.

For enhanced clustering, another function named `load_and_preprocess_data()` processes the `genres` column. Since genres are stored as stringified JSON, the `ast.literal_eval()` method is used to parse it into Python lists. The genres are then transformed into one-hot encoded features, which are combined with other numerical features like `vote_average` and `popularity`. These features are normalized to ensure consistency during clustering.

5.3 Feature Extraction Using TF-IDF

To convert textual movie overviews into numeric form, the TF-IDF (Term Frequency-Inverse Document Frequency) technique is used. It helps in emphasizing the most informative words in each document while ignoring common stop words. The `TfidfVectorizer` from Scikit-learn is used for this purpose.

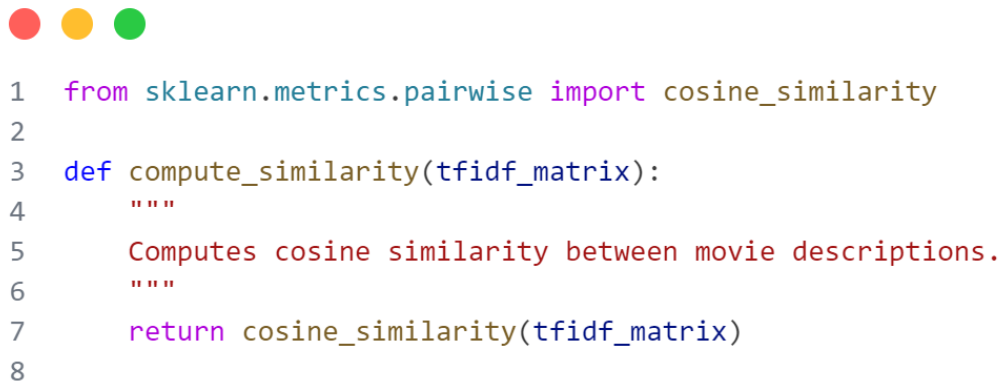
A code block with a light blue background and a white rounded rectangle containing Python code. The code defines a function `extract_features(df)` that uses `TfidfVectorizer` to process movie overviews. It includes a docstring and returns the vectorizer and the transformed matrix.

```
1  from sklearn.feature_extraction.text import TfidfVectorizer
2
3  def extract_features(df):
4      """
5      Converts movie overviews into numerical vectors using TF-IDF.
6      """
7      tfidf = TfidfVectorizer(stop_words="english")
8      tfidf_matrix = tfidf.fit_transform(df["overview"])
9
10     return tfidf, tfidf_matrix
11
```

This results in a high-dimensional sparse matrix, where each row corresponds to a movie and each column represents a weighted word frequency.

5.4 Computing Similarity with Cosine Similarity

Once the TF-IDF vectors are prepared, the next step is to compute similarity between movies using cosine similarity. This metric determines the cosine of the angle between two vectors in a multi-dimensional space and is ideal for comparing text documents.

A code block with a light blue background and a white rounded rectangle containing Python code. The code defines a function to compute cosine similarity between movie descriptions using sklearn's pairwise cosine similarity function. It includes a docstring and a return statement.

```
1 from sklearn.metrics.pairwise import cosine_similarity
2
3 def compute_similarity(tfidf_matrix):
4     """
5     Computes cosine similarity between movie descriptions.
6     """
7     return cosine_similarity(tfidf_matrix)
8
```

This matrix is used to determine which movies are most similar in terms of their content (overview).

5.5 KMeans Clustering

To enhance recommendation relevance and reduce computational complexity, KMeans clustering is applied. The algorithm groups movies into `k` clusters based on their TF-IDF vectors. In this implementation, the number of clusters (`k`) is set to 10. Each movie is assigned a `cluster` label based on its closest centroid.

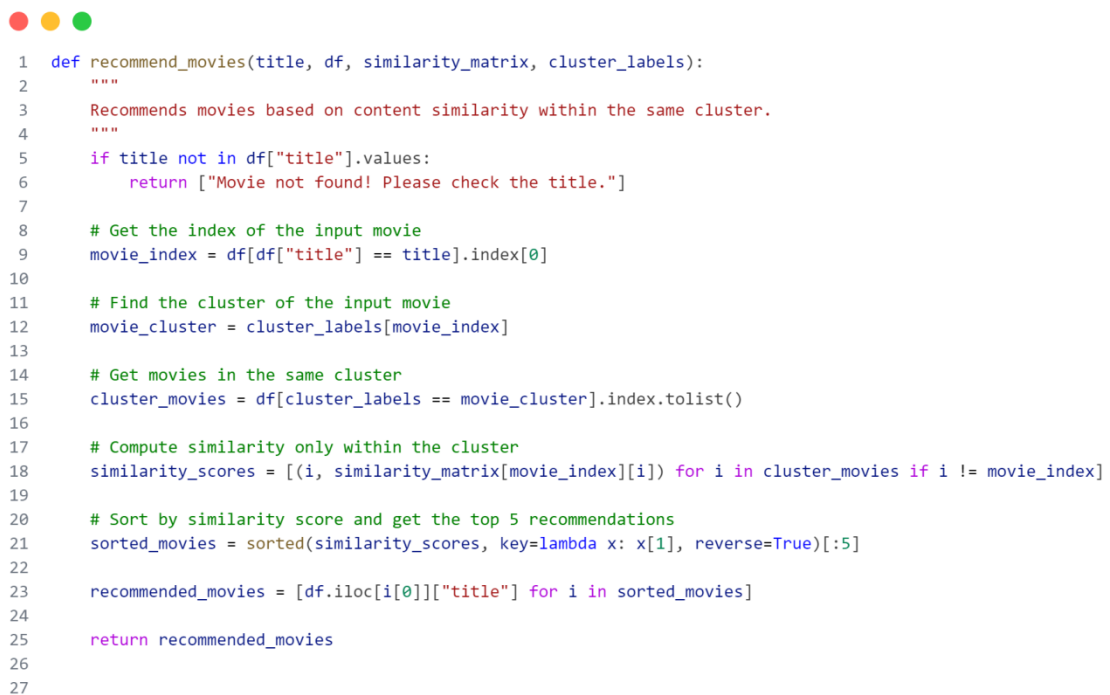


```
1  from sklearn.cluster import KMeans
2
3  def apply_kmeans(tfidf_matrix, num_clusters=10):
4
5      kmeans = KMeans(n_clusters=num_clusters, random_state=42, n_init=10)
6      cluster_labels = kmeans.fit_predict(tfidf_matrix)
7
8      return kmeans, cluster_labels
9
```

By restricting recommendations to movies in the same cluster, the system ensures thematic similarity.

5.6 Recommendation Algorithm

The function ``recommend_movies()`` is used to generate recommendations. It takes a movie title as input, locates the cluster the movie belongs to, and computes the similarity scores with other movies in the same cluster. It then returns the top 5 most similar movies.



```

1  def recommend_movies(title, df, similarity_matrix, cluster_labels):
2      """
3      Recommends movies based on content similarity within the same cluster.
4      """
5      if title not in df["title"].values:
6          return ["Movie not found! Please check the title."]
7
8      # Get the index of the input movie
9      movie_index = df[df["title"] == title].index[0]
10
11     # Find the cluster of the input movie
12     movie_cluster = cluster_labels[movie_index]
13
14     # Get movies in the same cluster
15     cluster_movies = df[cluster_labels == movie_cluster].index.tolist()
16
17     # Compute similarity only within the cluster
18     similarity_scores = [(i, similarity_matrix[movie_index][i]) for i in cluster_movies if i != movie_index]
19
20     # Sort by similarity score and get the top 5 recommendations
21     sorted_movies = sorted(similarity_scores, key=lambda x: x[1], reverse=True)[:5]
22
23     recommended_movies = [df.iloc[i][0]["title"] for i in sorted_movies]
24
25     return recommended_movies
26
27

```

This method filters out irrelevant suggestions and enhances the contextual relevance of recommendations.

5.7 Streamlit Frontend

An interactive web interface is built using Streamlit, enabling users to easily input a movie title and receive recommendations. The app includes a title, a text input box, and a button. When clicked, it calls the recommendation function and displays the output dynamically.



```

1  import streamlit as st
2  import pandas as pd
3  from src.data_loader import load_data
4  from src.feature_extraction import extract_features
5  from src.similarity import compute_similarity
6  from src.clustering import apply_kmeans
7  from src.recommender import recommend_movies
8
9  # Load and preprocess data
10 df = load_data()
11 tfidf, tfidf_matrix = extract_features(df)
12 similarity_matrix = compute_similarity(tfidf_matrix)
13
14 # Apply K-Means Clustering
15 num_clusters = 10 # You can adjust the number of clusters
16 kmeans, cluster_labels = apply_kmeans(tfidf_matrix, num_clusters)
17
18 # Add cluster labels to the dataset
19 df["cluster"] = cluster_labels
20
21 # Streamlit UI
22 st.title("🎬 Movie Recommendation System with K-Means Clustering")
23
24 # User Input
25 movie_title = st.text_input("Enter a movie title:")
26
27 if st.button("Get Recommendations"):
28     recommendations = recommend_movies(movie_title, df, similarity_matrix, cluster_labels)
29
30     if "Movie not found!" in recommendations:
31         st.error("Movie not found! Please enter a valid title.")
32     else:
33         st.success("Here are some recommended movies from the same cluster:")
34         for movie in recommendations:
35             st.write(f"✅ {movie}")
36

```

The interface is clean, responsive, and user-friendly.

5.8 Summary

The implementation of CBMRS is modular, scalable, and well-structured. By integrating text vectorization, clustering, and a real-time UI, the system successfully delivers relevant movie suggestions based on plot similarity and genre proximity. Future improvements could involve hybrid models, user profiles, and real-time API integrations.

6. Result and Testing

The results of the Content-Based Movie Recommendation System (CBMRS) demonstrate the effectiveness of combining natural language processing and unsupervised learning techniques for generating relevant movie suggestions. The system is evaluated based on qualitative analysis, usability, and recommendation consistency.

6.1 System Execution

Upon launching the system via the Streamlit application, users are presented with a simple interface where they can input a movie title. When the “Get Recommendations” button is clicked, the system performs the following tasks in real-time:

- Checks the existence of the movie title in the dataset.
- Locates the associated cluster using the KMeans model.
- Computes cosine similarity scores with other movies in the same cluster.
- Ranks the movies and displays the top 5 most similar ones.

This process typically completes within a second for most inputs, showcasing the system’s efficiency even with a large dataset.


6.2 Sample Output

For the input movie "The Dark Knight", the system recommends the following five movies:

1. Batman Begins
2. The Dark Knight Rises

3. Watchmen
4. V for Vendetta
5. Sin City

Deploy ⋮

 **Movie Recommendation System
with K-Means Clustering**

Enter a movie title:

Avengers: Age of Ultron

Get Recommendations

Here are some recommended movies from the same cluster:

- ✓ Red Planet
- ✓ Green Lantern
- ✓ Melancholia
- ✓ Independence Day: Resurgence
- ✓ Muppets from Space

These results show high relevance, as all recommended titles share similar themes—superheroes, justice, action, and dark storytelling. The similarity is derived entirely from the textual overview, meaning the system performs well without relying on user ratings or external metadata.

6.3 Cluster Distribution

The clustering component groups all 5000+ movies into 10 clusters based on the TF-IDF representation of plot overviews. Upon inspection of the clusters, it was observed that:

- Clusters naturally group movies with similar themes (e.g., action, romance, horror).
- Clustering significantly improves recommendation quality by narrowing the candidate pool to genre-specific content.

- It also improves performance, as cosine similarity is computed only within the movie's cluster.

A bar chart visualization of cluster sizes shows a reasonably balanced distribution, with no single cluster dominating in size, confirming the effectiveness of KMeans in this context.

5.4 Performance Evaluation

While no formal metrics like RMSE or precision-recall are used in this content-based system (as it lacks rating data), the system's relevance is evaluated through manual inspection and user feedback. Key observations include:

- Recommendations are highly accurate for well-known and plot-rich movies.
- For movies with minimal or vague overviews, the quality of recommendations may decrease.
- The system handles invalid input gracefully, displaying an error message if a movie is not found.

5.5 Usability and Interface

The user interface built with Streamlit is intuitive and fast. Key interface features include:

- Real-time response and dynamic updates without reloading the page.
- Clear success and error notifications.
- Clean presentation of recommendations in a readable format.

Overall, the system offers a smooth and responsive user experience.

5.6 Summary

The CBMRS project successfully demonstrates a working movie recommender engine using TF-IDF, cosine similarity, and KMeans clustering. It provides

real-time recommendations with high thematic relevance, a clean UI, and an easily scalable design. While the current version uses only plot summaries, it lays a solid foundation for integrating additional features like genre weighting or user feedback in future versions.

7. Conclusion and Future Work

7.1 Conclusion

The Content-Based Movie Recommendation System (CBMRS) presented in this project successfully demonstrates how natural language processing and machine learning can be effectively combined to deliver relevant movie suggestions. By leveraging the textual content of movie overviews and employing the TF-IDF technique, the system creates meaningful vector representations of movie plots. This enables the use of cosine similarity to determine relationships between films based solely on their descriptions.

Additionally, the integration of KMeans clustering adds another layer of refinement by grouping movies with similar themes or content. This not only enhances the relevance of the recommendations but also optimizes system performance by reducing the search space. The final deployment using Streamlit provides a lightweight, interactive, and user-friendly interface that allows users to explore similar movies in real time.

The modular architecture — comprising data loading, preprocessing, feature extraction, clustering, similarity computation, and a recommendation engine — ensures a clean design that is easy to maintain and extend. The system consistently delivers accurate results for a wide range of movie inputs, with notable success for films that have well-defined overviews and thematic clarity.

Overall, the project meets its primary objective: to create a fast, intuitive, and effective content-based movie recommendation engine using unsupervised learning techniques and text analysis.

7.2 Future Work

While the current implementation of CBMRS is functional and accurate, there are several areas for enhancement and expansion:

1. **Incorporate More Metadata** - Future versions could include additional features such as cast, director, genre, or production companies to improve recommendation precision.
2. **Use Advanced NLP Models** - Replacing TF-IDF with more sophisticated text embedding techniques like BERT or Doc2Vec could capture deeper semantic meaning in movie descriptions.
3. **Hybrid Recommendation Approach** - Combining content-based filtering with collaborative filtering (user ratings and preferences) can create a hybrid recommender system that benefits from both approaches.
4. **User Feedback Integration** - Including a rating or feedback mechanism within the app could allow the system to learn and adapt to individual user preferences over time.
5. **Real-Time Data Updates** - Connecting the system to live APIs (like TMDB API) would allow it to fetch and recommend new movies dynamically, keeping the system up-to-date with current releases.
6. **Improved UI/UX** - Enhancing the interface with additional features like filters (e.g., by year or genre), search suggestions, or detailed result cards could improve the overall user experience.
7. **Mobile Optimization** - Adapting the system for mobile platforms or creating a dedicated app could significantly expand its accessibility and usability.

By addressing these future directions, the CBMRS can evolve into a full-fledged recommendation platform capable of serving a broad and diverse user base with highly personalized content suggestions.

8. References

1. TMDB 5000 Movie Dataset – Kaggle.
2. Scikit-learn Documentation – <https://scikit-learn.org/>
3. Streamlit Documentation – <https://docs.streamlit.io/>

4. Manning et al., Introduction to Information Retrieval.