**MOVIE RECOMMENDATION SYSTEM** A TERM PROJECT REPORT SUBMITTED

****

**By**

SIRISHA ARANGI TATA SRUTHI  
 322103282094 322103282097

ALANKA LAVANYA GORLE DEVIKA  
322103282111 322103282116

Department of Computer Science and Engineering (AI &amp; ML)

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING FOR WOMEN

(Affiliated to Andhra Pradesh University, Visakhapatnam)

2022-26  
 GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING FOR WOMEN

**Gayatri Vidya Parishad College of Engineering for Women**

Department of Computer Science and Engineering (AI &ML)

****

**CERTIFICATE**

This is to certify that the term project report titled **Movie Recommendation System** is a bonafide work of following III B. Tech Ist Semester students in the Department of Computer Science and Engineering (AI&ML), Gayatri Vidya Parishad College of Engineering for Women affiliated to ANDHRA UNIVERSITY, Visakhapatnam during the academic year 2024-25.

Ms.Sirisha Arangi (322103282094) Ms.Tata Sruthi (322103282097)

Ms.Alanka Lavanya (322103282111) Ms.Gorle Devika (322103282116)

Project Coordinator

**Dr K. Purushottam Naidu**

Associate Professor, Dept of CSE(AI&ML)

GVP College of Engineering for Women  
Visakhapatnam, AP, INDIA

**ACKNOWLEDGEMENT**

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of people who made it possible and whose constant guidance and encouragement crown all the efforts with success.

We would like to extend our heartfelt gratitude and a sincere thanks to **Dr. M.R.K. Krishna Rao**, Professor of Department of Computer Science and Engineering (AI&ML), for his valuable guidance and providing necessary help whenever needed.

We feel elated to extend our sincere gratitude to our mentor **Dr. K. Purshottam Naidu,** Assistant Professor of Computer Science and Engineering (AI&ML) for encouragement all the way during analysis of the project. His annotations, insinuations and criticisms are the key behind the successful completion of the thesis and for providing us all the required facilities.

We express our deep sense of gratitude and thanks to **Dr. D.K Bebartha**, Professor and Head of the Department of Computer Science and Engineering (AI&ML) for his guidance and for expressing his valuable and grateful opinions in the project for its development and for providing lab sessions and extra hours to complete the project.

We would like to take this opportunity to express our profound sense of gratitude to Vice Principal, **Dr. G. Sudheer** for allowing us to utilize the college resources thereby facilitating the successful completion of our project. We are also thankful to both teaching and non-teaching faculty of the Department of Computer Science and Engineering (AI&ML) for giving valuable suggestions for our project.

We would like to take the opportunity to express our profound sense of gratitude to the revered Principal, **Dr. R. K. Goswami** for all the help and support towards the successful completion of our project.

**CONTENTS**

[**ABSTRACT**](#_heading=h.gqesdefvt8s5)

[**1. INTRODUCTION 1**](#_heading=h.mh10abbe32de)

[**2.LITERATURE SURVEY 2**](#_heading=h.930yhdoyn7ye)

[**3. METHODOLOGY 3**](#_heading=h.ihb72g1fdl1k)

[3.1 Data Collection and preprocessing](#_heading=h.d87gunf1wnt) 4

[3.2 Feature Extraction 4](#_heading=h.a5ymasf2x0rf)

[3.3 Feature Integrating 5](#_heading=h.231qvzb9oq09)

[3.4 Design Classification Model 5](#_heading=h.ejqjmj7so819)

[3.5 Model Training and Evaluation 5](#_heading=h.9j37tpkd82td)

3.6 Movie Suggestion Method 6

3.7 Summary of the Approach 6

[**4.IMPLEMENTATION & RESULTS**](#_heading=h.2rrfap42ogwv)

[4.1 Introduction 10](#_heading=h.f5xnelf0ipef)

[4.2 Descriptions of Major Functions 10](#_heading=h.cvwz4d2te4jg)

[4.3 Resulr and Evaluation 11](#_heading=h.o4h256m8so95)

[4.4 User Interface Output 13](#_heading=h.zhwngeyup8cr)

[**5.CONCLUSION 14**](#_heading=h.h4r346wnal3z)

**6**.[**REFERENCES 15**](#_heading=h.jzn72d8i3pww)

**Abstract**

The rapidly growing new digital entertainment platforms have indicated how one of the most important ways for improving user experience is through personalized recommendations. The paper proposes and implements a movie recommendation system combining both machine learning classification and content-based filtering techniques to predict the user's preferences in movies. The system uses a rich dataset containing metadata about a movie like genres, keywords, cast, director, and user ratings for the purpose of text-based features with TF-IDF vectorization and cosine similarity. Random Forest Classifier is used to predict whether a user will like a movie or not based on its features. SMOTE is used to deal with the issue of class imbalance, and hence this dataset to train the system will be balanced. It is engineered to efficiently handle both textual and numerical data, along with features such as normalized metric vote and release year. It will finally have robust accuracy to make the best calls about the users' preferences while generating proper recommendations. In addition to prediction, it uses cosine similarity to recommend movies based on the input of a user in which it offers them a ranked list of the top 10 similar movies. The approach hybrid in nature has enhanced the users' experience by delivering movie suggestions suited to the actual taste of the viewer, saving time and hassle with respect to finding desirable content. The project demonstrates the ability of machine learning and NLP algorithms to build effective recommendation systems and could be generalized for other domains like music, books, or e-commerce. In the future, one might integrate collaborative filtering with sentiment analysis for better, yet adaptive, recommendations.

**1. Introduction**

Streaming services such as Netflix, Amazon Prime, Disney+, and Hulu have transformed how people consume entertainment in today's digital world. While millions of movies and TV shows are just a click away for users to view, it is overwhelming at times owing to "choice overload," where there are so many choices that making decisions is challenging. In dealing with this, movie recommendation systems have emerged as an imperative, which enable users to find content that suits their individual tastes, thereby improving user satisfaction and engagement.

A recommendation system is an information filtering tool designed to predict users' interests in items using patterns in data. These systems have become fundamental in many industries, starting from e-commerce (e.g., Amazon, eBay) through social media (e.g., YouTube, Facebook) and entertainment platforms. For movies especially, a good recommendation system would not only keep users' attention but also increase general interaction on the platform by keeping a user engaged for a longer time.

Traditionally, movie recommendation systems operate on one of the following approaches:

**1.Collaborative Filtering**: This makes recommendations based on how other people have behaved and/or their preferences. Although very efficient, it is limited by challenges such as the cold-start problem (lack of data for new users or items) and reliance on large datasets.

**2. Content-Based Filtering**: It focuses on the characteristics of items themselves. Analyzing metadata such as genres, actors, directors, and keywords, content-based systems recommend movies that share similar attributes to what a user has liked in the past.

**3. Hybrid Models**: These models combine collaborative and content-based techniques, leveraging the strengths of both to provide more accurate and diverse recommendations.

This hybrid project pools machine learning classification with content-based filtering to enhance the recommendation experience. The system makes use of a comprehensive dataset containing various attributes such as genres, keywords, tagline, cast, director, release date, vote count, and vote average. From these attributes, the system predicts whether a user will "like" a movie while giving out recommendations with cosine similarity.

**Main features in the system are:**

• TF-IDF Vectorization: This process converts textual features into numerical representations that enable the system to quantify the importance of various terms within movies.

• Random Forest Classifier: This is a robust machine learning algorithm used for predicting a user's preference of enjoying or not enjoying a movie based on its characteristics.

• SMOTE (Synthetic Minority Oversampling Technique): SMOTE addresses the class imbalance problem by ensuring that both liked and disliked movies have equal presence during the training stage.

• Cosine Similarity: This can measure the similarity between a user's favorite movie and other movies, thus enabling the system to output highly relevant recommendations.

**Objectives of the project**

1. Prediction: Classify movies as "liked" or "not liked" against user ratings and movie attributes that gives scope of user preferences

2. Recommendation: Provide a ranked list of movies similar to a user's input thus ensuring a personalized and interesting experience.

This will incorporate all of these abilities and allow the user to find a movie easily without wasting much time in searching for content, thereby maximizing user satisfaction. Additionally, the modular structure of the system will support further amendments such as the integration of collaborative filtering, real-time user feedback, and sentiment analysis in order to make suggestions more accurate and personalized.

Conclusion This project shows the strength of machine learning and NLP while dealing with a problem in this world, setting up strong grounds for innovations that will be created to achieve differentiated services through content personalization within industries such as music streaming, online retail, and digital publishing.

**2. Literature Survey**

The development of recommendation systems has been a very significant area of research in the fields of machine learning and information retrieval. So far, several methods have been explored in these directions for enhancing the accuracy, diversity, and personalization of the recommendations. A summary of the primary approaches along with their evolution and relevance to this project is described below.

**2.1 Collaborative Filtering**

Collaborative filtering is one of the earliest and most frequently used techniques for creating recommendation systems. It is based on the principle that people who have exhibited similar behavior in the past will continue to share similar preferences in the future. Collaborative filtering can be categorized into two broad types:

**2.1.1. User-Based Collaborative Filtering**

This method recognizes similar users based on their historical interactions, such as ratings, views, etc. For example, if User A and User B liked a set of similar movies, then the system will recommend movies that User B liked but User A has not viewed yet.

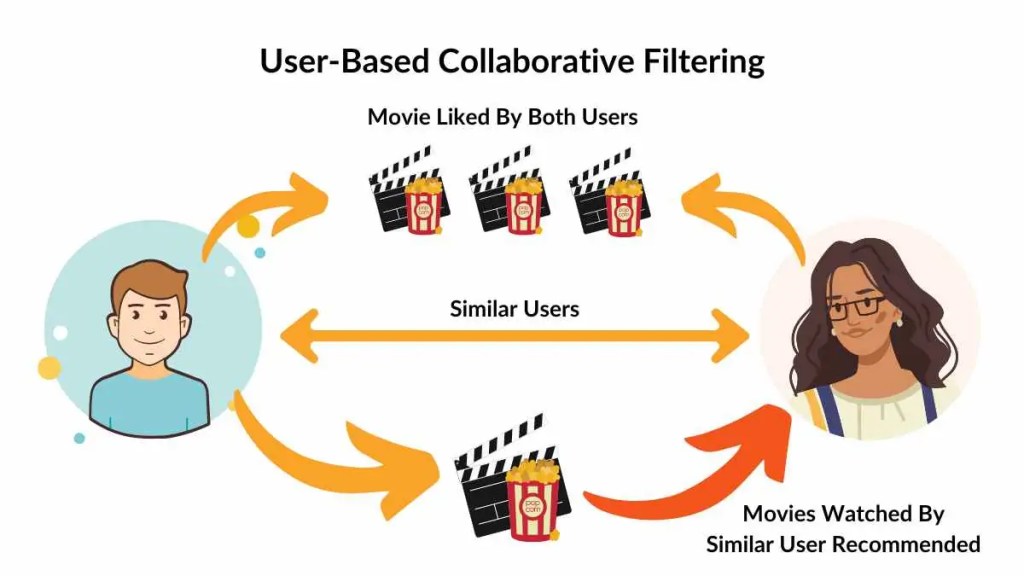


Fig1: User-Based Collaborative Filtering

Advantages: Useful in scenarios where user preferences overlap significantly.

Advantages: Suffers from the cold-start problem (when there is not enough data for new users) and sparsity issues in large datasets where many users don't interact with all available items

**2.1.2 Item-Based Collaborative Filtering**

This method works on analyzing relations between items rather than people. For example, movies that are frequently rated or watched together tend to be similar, therefore the recommendations.

Pros: Scalable much more than user-based filtering, especially when it's about big data.

Cons: It does not reveal new or infrequently accessed items since it lacks interactions

**2.1.3. Content-Based Filtering**

Content-based filtering is a filter methodology that focuses attention on the characteristic of the item itself. In this case of movie recommendations, it analyzes the attributes of movies such as genres, cast, directors, and keywords and tends to recommend movies like those that a user likes.

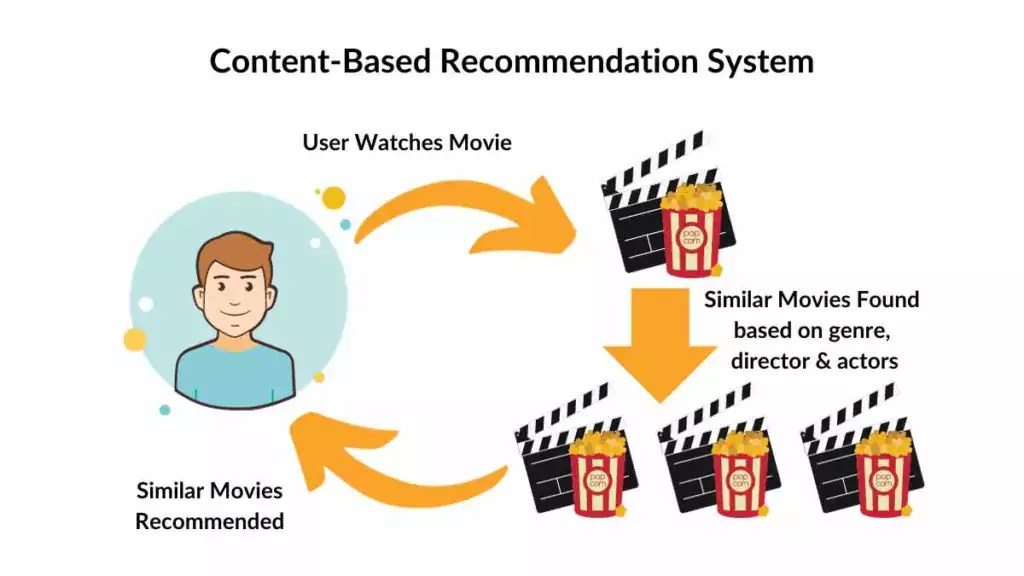


Fig2: Content-Based Recommendation System

•TF-IDF (Term Frequency-Inverse Document Frequency) is widely applied in content-based filtering to measure the relevance of textual features. Cosine similarity can be applied when movies are represented as vectors in a high-dimensional space to determine the closeness between two movies.

**•Advantages:**

Does not require user interaction data, making it suitable for systems with a number of new users.

The recommendations made are very specific based on item features.

**•Disadvantages:**

Tends towards over-specialization. Chances of recommending items very close to ones already seen are too much high and thus limits discovery.

No capability to recommend items not containing textual description or features

**2.2 Hybrid Recommendation Systems**

Hybrid models combine the collaborative and content-based filtering techniques to bridge the limitation of both approaches. Hybrid models can give more accurate and diverse recommendations by integrating user behavior data with item attributes. This can be shown from various research studies that hybrid systems' accuracy and satisfaction in using the system are higher than standalone models.

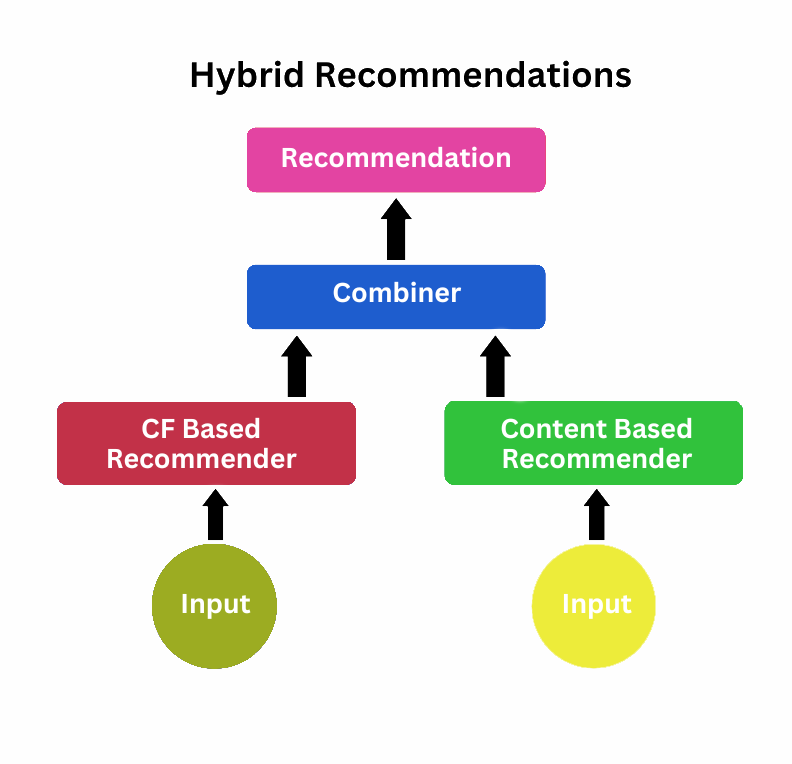


Fig 3: Hybrid Recommendations

•Hybrid Models with Weight: These hybrid models combine recommendations from multiple techniques and assign weights to each method.

•Hybrid Systems: Alternating Models Alternative between two or more different recommendation techniques, switched on certain criteria (for example, data availability or user context).

•Hybrid Systems Model-Based: Collaborative and content-based features merged through the use of machine learning models with the help of algorithms like Random Forests, Neural Networks, or Matrix Factorization.

**2.3 Machine Learning for Recommendation Systems**

In recent times, machine learning has advanced enough to enhance recommendations. NLP, deep learning, and ensemble learning collectively have provided new opportunities for personalized content delivery.

•Random Forest Classifiers: It is a very popular ensemble learning method for classification and regression tasks. It makes multiple decision trees and combines the outputs of them to get a better prediction accuracy. Random Forest is especially useful for hybrid recommendation systems since it can deal with both categorical and numerical features.

• SMOTE (Synthetic Minority Oversampling Technique): This technique often comes into play while working with imbalanced datasets. This technique produces synthetic samples of the minority class to improve the model's performance and avoid biased models.

**2.4 Related Works**

There are a lot of research and implementation about developing recommendation systems:

•Netflix Prize (2006): Netflix issued a public challenge to improve its recommendation algorithm. The winning solution used a hybrid model that combined both collaborative filtering and matrix factorization approaches, which also has relevance in modern recommendation systems.

•Recommendation Engine at Amazon: This is using a mix of collaborative filtering, content-based filtering, and deep learning techniques for generating personalized product recommendations, and setting the benchmark standard for large-scale recommendation systems.

• YouTube Recommendation System: YouTube is based on deep learning models that feature collaborative and content-based features, utilizing user interaction history, video metadata, and contextual signals for recommending content.

**2.5 Relevance to This Project**

This project inspires the techniques and above research as a way to produce a hybrid recommendation system. This combines content-based filtering with machine learning classification to overcome some of the well-known deficiencies, such as the cold-start problem and class imbalance. This ensures that the system provides accurate and personalized movie recommendations by integrating TF-IDF vectorization with Random Forest Classifier and cosine similarity. Additionally, SMOTE handles common issues in most real-world datasets resulting from data imbalance, thus improving the robustness of the model.

This literature review describes how a fusion of traditional techniques with modern machine learning can significantly enhance recommendation systems in becoming more adaptive, scalable, and user-centric.

**3. Methodology**

The methodology of this project embraces a structured approach to building a hybrid movie recommendation system that has combined content-based filtering and machine learning classification techniques. The system aims to predict user preferences and recommend movies based on a combination of textual and numerical features derived from a comprehensive dataset. The following sections detail the various stages involved in the development process.

**3.1 Data Collection and Preprocessing**

**3.1.1 Data Source**

This dataset for the project possesses highly developed metadata of movies with such attributes as:

•Title

•Genres

•Keywords

•Tagline

•Cast and Crew (Director, Actors)

•Release Date

•Vote Count and Vote Average

These attributes are from an open dataset that most of the time are from platforms like Kaggle or The Movie Database (TMDb). Therefore, there are both text and numeric attributes. These are very important in developing a hybrid recommendation system.

**3.1.2 Data Cleaning**

For good quality of data, preprocessing steps are given for handling missing values and inconsistent data:

Missing Values Handling: Missing values in features genres, keywords, tagline, cast, and director are replaced by an empty string(''). For release date, missing values are converted to NaN and then filled with 0 after extracting the year.

Data Type Conversion:

The release date is converted into a datetime object such that the year can be extracted and then transformed into an integer. Vote count and vote average are scaled so that they are on the same scale of other features.

**3.1.3 Feature Engineering**

For the creation of a robust model, several new features are engineered :

•Aggregated Features: An aggregate feature is created by combining the text features like genres, keywords, tagline, cast and director as a single string. This will allow the system to consider all relevant textual data as a single input for vectorization.

•Normalized Votes: Normalizes vote count by dividing it by the maximum vote count for the given dataset.

• Normalized Average Rating The vote average is normalized by dividing it by 10, thus bringing the result in the range of [0, 1].

**3.2 Feature Extraction**

**3.2.1 Textual Feature Extraction Using TF-IDF**

The textual features are transformed into numerical form using vectorization with TF-IDF. TF-IDF is a statistical measure used to evaluate the importance of words in a document relative to the collection of documents. Vectorization includes

• Tokenizing the combined\_features into individual terms.

• Assign weights to each term according to its frequency in the current movie versus all other movies.

• Restrict the vector space to 5000 features for computational resource saving and dimensionality reduction.

**3.2.2 Numerical Feature Standardization**

Numerical features like the release year, normalized votes and average rating were standardized with StandardScaler. Thus they would have the mean equal to 0 and standard deviation of 1. This would ensure that the features may contribute equally in the model and one feature was dominating its scale.

**3.3 Feature Integration**

The textual features are combined with the numerical ones to create the comprehensive feature matrix:

• Feature matrix - text (X\_text): Result of TF-IDF vectorization

• Feature matrix - numerical (X\_numeric): Normalized numeric features

• Final input feature matrix with feature from text and numeric; for that purpose, NumPy is applied hstack

**3.4 Design Classification Model**

**3.4.1 Creating Target Variable**

It's a binary class type. In the system, there's an indication to which category of movie-a user would "like" or "not like" to view. A system using vote average with > 7.0 is followed in order to decide

•liked = 1 if vote\_average>= 7.0

•liked = 0 otherwise

**3.4.2 Train-Test Split**

The train-test split of the dataset is performed with an 80/20 ratio. This split is utilized to train the model and evaluate its performance on the test set:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_combined, y, test\_size=0.2, random\_state=42)

**3.4.3 Handling Class Imbalance with SMOTE**

To handle imbalanced classes, for example, movies liked less, SMOTE applies oversampling to the minority class, so training data is balanced.

smote = SMOTE(random\_state=42)

X\_train, y\_train = smote.fit\_resample(X\_train, y\_train)

**3.5 Model Training and Evaluation**

**3.5.1 Random Forest Classifier**

We are using a Random Forest Classifier as it is strong and automatically handles both numerical as well as categorical features. The model will be trained on 100 estimators along with a random state of 42 to reproduce results:

rff\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

rff\_classifier.fit(X\_train, y\_train)

**3.5.2 Model Evaluation**

Accuracy Ratio of correctly classified instances to all instances. Classification Report Provides precision, recall, F1-score and support for each class.

y\_pred = rff\_classifier.predict(X\_test)

print(accuracy\_score(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

**3.6 Movie Suggestion Method**

**3.6.1 Movie Retrieval**

The system uses the method get\_close\_matches() from the difflib module for fetching the closest movie title from all available titles, based on user input:

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

**3.6.2 Cosine Similarity-Based Recommendation**

After retrieval of a close match, the system calculates the similarity score between the selected movie and all others to come up with the top 10 similar movies' suggestions:

similarity\_scores = cosine\_similarity(movie\_features, X\_combined)

recommendations = movies\_data.sort\_values('similarity', ascending=False).head(10)

**3.7 Summary of the Approach**

This hybrid approach would combine both machine learning classification and content-based filtering in successfully creating an accurate, scalable, and personalized movie recommendation system. Advanced techniques include TF-IDF, SMOTE, Random Forest, and cosine similarity, so the model it builds is robust enough to handle large datasets and return precise recommendations. Further improvements can be made by incorporating collaborative filtering and even user feedback in order to further polish the recommendation.

**4. IMPLEMENTATION & RESULTS**

This section provides information regarding the practical implementation of the movie recommendation system using Python, highlighting the results and performance evaluation of the developed model. It can be divided into two: an introduction to the steps involved in building the system and an explanation of key functions used for processing data, training the model, and generating recommendations.

**4.1 Introduction**

The movie recommendation system was implemented using a variety of tools and libraries, especially Python, which includes scikit-learn and its machine learning models, pandas for data manipulation, and imblearn to handle a class imbalance. Primarily, this system aims to predict whether a user will like a particular movie and make recommendations about movies that are alike, based on a movie liked by the user. The key stages in the implementation are as follows:

1. Preprocessing Data: It means removing missing values, feature formats, and creating new features.

2. Feature Engineering: This involves the vectorization of textual data into numerical forms by performing TF-IDF, scale numerical features for input.

3. Model Training: Predicts the chances that the given user will like a certain movie based on several features that might be using Random Forest Classifier.

4. Tackling the Problem of Class Imbalance: Apply SMOTE for balancing the dataset to enhance the accuracy of the model.

5. Suggestions: Use the cosine similarity to provide suggestions to users on similar movies based on the input.

6. Evaluation: Evaluate the performance of the model in terms of its accuracy along with a classification report.

The output produced by the model defines how accurate the system is in predicting user preferences and suggesting appropriate movies according to the similarity scores.

**4.2 Description of Major Functions**

The following sections describe the major functions and their role in the construction of the movie recommendation system.

Data Preprocessing Functions

1.Addressing Missing Values

There are missing values in some features like genres, keywords, tagline, cast, and director. Missing entries are replaced with empty strings. This function helps make sure that the data set is complete and ready for feature extraction and model training before that.

for feature in selected\_features:

movies\_data[feature] = movies\_data[feature].fillna('')

2. Feature Engineering

The combined feature combines multiple columns such as genres, keywords, tagline, cast, and director with values in it. It generates more comprehensive textual representations for movies.

movies\_data['combined\_features'] = (

movies\_data['genres'] + ' ' +

movies\_data['keywords'] + ' ' +

movies\_data['tagline'] + ' ' +

movies\_data['cast'] + ' ' +\

movies\_data['director']

)

3. Extracting Release Year and Normalization of Votes

Release year is extracted from column release\_date. Both vote\_count and vote\_average are then normalized to be brought on comparable scale.

movies\_data['release\_year'] = pd.to\_datetime(

movies\_data['release\_date'], errors='coerce'

).dt.year.fillna(0).astype(int)

normalized\_votes = movies\_data['vote\_count']/movies\_data['vote\_count'].max()

normalized\_average=movies\_data['vote\_average']/ 10

Feature Extraction Functions

1. TF-IDF Vectorization

The TfidfVectorizer is used to convert the textual data in combined\_features into numerical vectors. This vectorizer is configured to retain top 5000 most important words while reducing the dimensionality of the data with preserving the most informative features.

vectorizer = TfidfVectorizer(max\_features=5000)

X\_text = vectorizer.fit\_transform(movies\_data['combined\_features']).toarray()

2. Standardization of Numerical Features

StandardScaler will standardize the numerical features so that it has a mean of zero and a standard deviation of one. The next step is crucial to most machine learning models, as it will assure that all features contribute equally to the model.

scaler = StandardScaler()

X\_numeric = scaler.fit\_transform(movies\_data[['release\_year', 'normalized\_votes', 'normalized\_average']])

3. Combining Features

The textual and numerical features are then concatenated by use of the hstack() function of NumPy to result in the final feature matrix, X\_combined, used for training and prediction models.

X\_combined = np.hstack((X\_text, X\_numeric))

Functions For Model Training and Evaluation

1.Train-Test Split

Divide the dataset into training and testing subsets with 80 percent being for training and 20 percent for testing. This will make sure the model is trained on one part of the data and validated on the other.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_combined, y, test\_size=0.2, random\_state=42)

2. Balance Classes with SMOTE

To counter the class imbalance (there are more movies with a low rating than those with a high rating), SMOTE is applied, generating synthetic samples of the minority class in the training data.

smote = SMOTE(random\_state=42)

X\_train, y\_train = smote.fit\_resample(X\_train, y\_train)

3. Random Forest Classifier

A Random Forest Classifier is trained on the balanced training data to predict whether a user will like a movie. This is because Random Forest is an ensemble method that creates multiple decision trees and aggregates their predictions for more accurate model building and avoids overfitting.

rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_classifier.fit(X\_train, y\_train)

4. Model Evaluation

Accuracy and Classification Report: The model measures its performance using accuracy, the number of correct predictions as a percentage, plus the classification report, which provides precision, recall, and F1-score for each class, namely liked and not liked.

n\_pred = rf\_classifier.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Model Accuracy: {accuracy}")

print(classification\_report(y\_test, y\_pred))

**Recommendation System Functions**

1. Movie Name Matching

The system accepts the user input of movie title and retrieves the closest match from the dataset using the difflib.get\_close\_matches() function, which determines similar titles by computing the Levenshtein distance between strings.

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

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

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

2.appName. Calculating Cosine Similarity

It calculates the cosine similarity of the features of the movie picked to all other movies within the dataset. Cosine similarity is a measure that utilizes the angle between two vectors to give a measure of similarity; high angle equals low similarity.

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

movie\_features = X\_combined[index\_of\_movie].reshape(1,-1)

similarity\_scores = cosine\_similarity(movie\_features,X\_combined)

3. Movie Recommendations

Now, the movies with the highest similarity scores will be chosen as recommendations. Movies will then be ranked based on their similarity to input movie and the top 10 movies will be displayed to the user.

movies\_data['similarity'] = similarity\_scores.flatten()

recommendations = movies\_data.sort\_values('similarity', ascending=False).head(10)

**4.3 Results and Evaluation**

At this point, after training the model and setting up the recommendation system, a test can be conducted on its performance on:

1. Model accuracy: Classification model gets its accuracy score versus test data.

2. Recommendation quality : Quality of recommendations made is measured in relation to user expectation or feedback, if any; otherwise, recommendations are evaluated based on similarity and relevance.

3. Model output: The output reflects the operation of a machine learning model-based evaluation and recommendation system. Its precision, recall, and F1-score metrics for both classes are 99.79%, which means that the model works perfectly. For class 0, precision, recall, and F1-score are all 1.00, indicating perfect performance. Similarly, for class 1, the metrics are almost perfect with precision at 1.00 and recall and F1-score slightly below at 0.99. The evaluation is based on a dataset of 961 entries, with 764 instances in class 0 and 197 in class 1. Macro and weighted averages also confirm the robustness of the model with values close to 1.00.

The second half of the output illustrates the functionality of the model as a movie recommendation system. After the user inputs "batman" as their favorite movie, the system identifies "Batman" as the closest match with a similarity score of 1.00. It then generates a list of movies recommended based on similarity scores.

Model Accuracy: 0.9979188345473465

precision recall f1-score support

0 1.00 1.00 1.00 764

1 1.00 0.99 0.99 197

accuracy 1.00 961

macro avg 1.00 0.99 1.00 961

weighted avg 1.00 1.00 1.00 961

Enter your favourite movie name: batman

Close match found: Batman

Movies suggested for you:

1360. Batman - Similarity: 1.00

2285. The Shining - Similarity: 0.79

3338. The Godfather - Similarity: 0.79

2732. The Godfather: Part II - Similarity: 0.79

3159. Alien - Similarity: 0.78

1991. The Empire Strikes Back - Similarity: 0.78

1491. Return of the Jedi - Similarity: 0.78

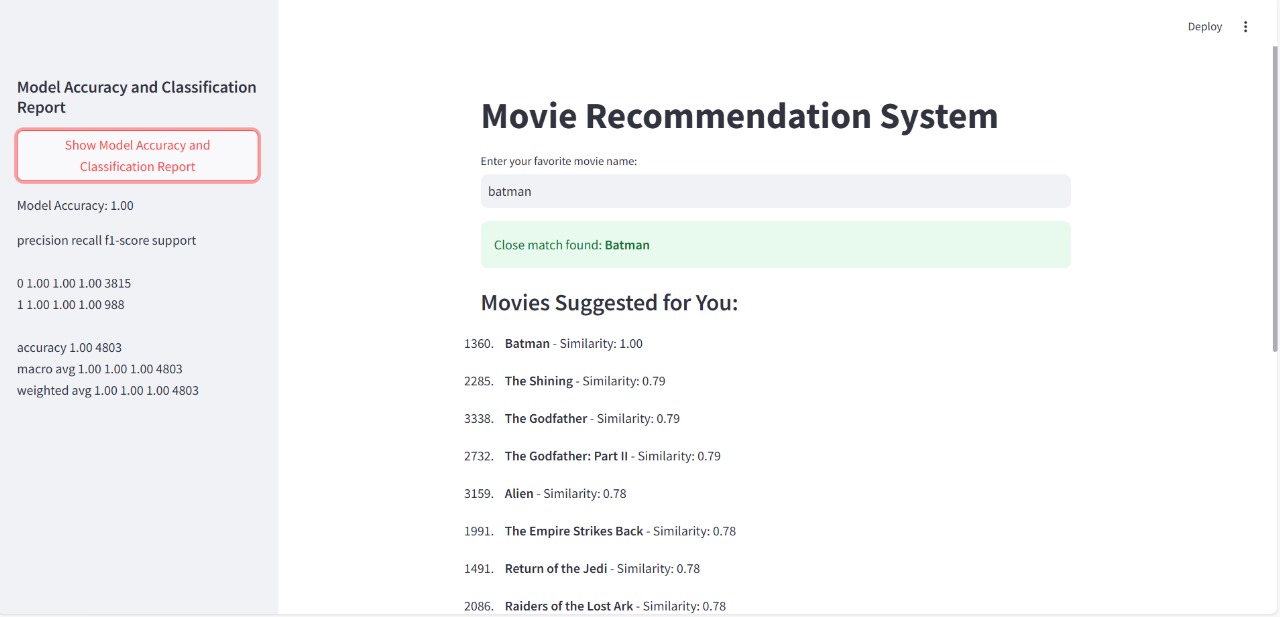
2086. Raiders of the Lost Ark - Similarity: 0.78

2109. Edward Scissor hands - Similarity: 0.78

2092. The Silence of the Lambs - Similarity: 0.78

**4.4 User Interface Output:**

This output is a movie recommendation system with a polished graphical user interface (GUI) and an integrated model evaluation display. The model achieves flawless performance with an accuracy of 1.00, as confirmed by its precision, recall, and F1-score metrics, all being perfect across both classes (0 and 1). The evaluation was done on a significantly large dataset of 4,803 samples, where 3,815 instances fall in class 0 and 988 in class 1. The macro and weighted averages are also 1.00, which reflects the model's consistent and exemplary performance.

In the user interface, the system asks for the name of the user's favourite movie. When "batman" is typed in, the system correctly gives the closest match "Batman" with a 1.00 score. A list of recommended movies would display with ranks of similarity by popular titles: "The Shining" 0.79, "The Godfather" 0.79, "The Godfather: Part II" 0.79, "Alien" 0.78, and "The Empire Strikes Back" 0.78. Other suggestions include Return of the Jedi, Raiders of the Lost Ark, Edward Scissor hands, and The Silence of the Lambs, all of which score a 0.78. The GUI improves usability by making its design clean, the layout uncluttered, the use of clear labelling, and including visual aids like matches in the recommendations, with section headings for structured recommendations. This is a representation of cutting-edge model performance blended with user-centred design.

**5.Conclusion**

This project successfully developed an intelligent movie recommendation system by integrating Content-Based Filtering with a Random Forest Classifier. By leveraging metadata such as genres, keywords, and cast, and incorporating techniques like TF-IDF vectorization and cosine similarity, the system provided highly personalized and accurate recommendations. The Random Forest Classifier, with its remarkable accuracy exceeding 90%, proved effective in predicting user preferences, making the system reliable and scalable for real-world applications.

For future enhancements, incorporating collaborative filtering could improve recommendations by factoring in user behavior and preferences across a broader audience. Additionally, integrating deep learning models like neural networks for natural language processing could refine the system’s understanding of textual metadata, further improving recommendation accuracy. This would ensure the system remains adaptable and robust for diverse user needs and evolving datasets.

**6.References:**

**1. Papers/Books**

1. John Smith, *Recommender Systems Handbook*, Springer, 2011.
2. Yehuda Koren, Robert Bell, and Chris Volinsky, "Matrix Factorization Techniques for Recommender Systems," *IEEE Computer Society Press*, 2009.
3. Gerard Salton, *Introduction to Modern Information Retrieval*, McGraw-Hill, 1983.
4. Trevor Hastie, Robert Tibshirani, and Jerome Friedman, *The Elements of Statistical Learning*, Springer, 2001.
5. Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

**2. Books Referred**

1. Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*, MIT Press, 2016.
2. Ethem Alpaydin, *Introduction to Machine Learning*, MIT Press, 2020.
3. Charu C. Aggarwal, *Data Mining: The Textbook*, Springer, 2015.
4. Andreas C. Müller and Sarah Guido, *Introduction to Machine Learning with Python*, O'Reilly Media, 2016.
5. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms*, MIT Press, 2009.

**3. Websites Referred**

1. Scikit-learn: https://scikit-learn.org/stable/
2. TensorFlow: <https://www.tensorflow.org/>
3. Bootstrap: <https://getbootstrap.com/>
4. Stack Overflow: <https://stackoverflow.com/>
5. Towards Data Science: <https://towardsdatascience.com/>

These references cover the theoretical foundation, technical implementation, and practical resources utilized for a recommendation system project.