International Scientific Journal of Engineering and Management ISSN: 2583-6129
Volume: 03 Issue: 04 | April – 2024 DOI: 10.55041/ISJEM01577
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

RECOMMENDAITON SYSTEM ENGINE FOR MOVIES USING MACHINE LEARNING ALGORITHM (TF-IDF VECTORIZATION)

MD SUHAIB

Birla Institute of Technology, Mesra

Email: shaikmohammedsuhaib786@gmail.com

Under the Guidance of

Mr. Aruna Malik (Professor)

Abstract:

By offering tailored movie recommendations, Movie Recommendation Systems (MRS) are crucial for improving the user experience on streaming services. This research paper proposes and evaluates a Movie Recommendation System utilizing TF-IDF vectorization and cosine similarity. TF-IDF vectorization is used to analyze textual information related to movies, such as plot summaries, cast bios, and genres, in order to give users precise and pertinent suggestions. The similarity between the user's preferences and the movies in the dataset is then calculated using cosine similarity.

The results of the study show that the suggested Movie Recommendation System, which makes use of cosine similarity and TF-IDF vectorization, greatly improves user happiness and recommendation accuracy. The developed system offers an effective solution for providing personalized movie recommendations, contributing to the advancement of recommendation systems in the entertainment industry. This study provides valuable insights for streaming platforms to improve their recommendation systems and enhance user engagement.

Keywords: Movie Recommendation System, Machine Learning (ML), Natural Language Processing (NLP) TF-IDF Vectorization, Cosine Similarity, Content-Based Filtering, Personalized Recommendations, User Engagement.

1. Introduction:

Currently, there are so many online streaming services available in this day and age, there is an excessive amount of content available, which makes it harder and harder for consumers to find new movies that suit their tastes. Movie Recommendation Systems (MRS), which offer customized recommendations based on each user's tastes, are essential in tackling this problem. In this study, we present and assess a movie recommendation system that makes use of cosine similarity and TF-IDF (Term Frequency-Inverse Document Frequency) vectorization.

The objective of this research is to create a strong recommendation system that can analyze textual information about movies, including narrative summaries, actors, and genres, and then deliver users relevant and accurate recommendations. Cosine similarity is used to gauge how similar the user's tastes are to the movies in the dataset, and TF-IDF vectorization is used to extract significant features from the textual data. The recommendation system hopes to improve user happiness and recommendation accuracy by integrating various approaches.

The efficacy of the suggested method in suggesting films that correspond with the user's tastes is assessed. Metrics including accuracy, precision, and recall are part of this assessment. The study dataset, data pre-processing methods, and recommendation algorithms—which include content-based filtering utilizing TF-IDF vectorization and cosine similarity—are all covered in the methodology section.

The experimental setup and evaluation procedures are detailed, followed by the presentation and analysis of experimental results. The discussion section interprets the results, discusses the strengths and weaknesses of the proposed system, and suggests potential improvements and future research directions. The research findings indicate that the proposed Movie Recommendation System significantly enhances recommendation accuracy and user satisfaction, providing valuable insights for streaming platforms to improve their recommendation systems and enhance user engagement.

2. Literature Review:

Movie recommendation systems (MRS) have become indispensable in the digital entertainment era, where vast libraries of content make it increasingly challenging for users to discover new movies that match their preferences. In current years, various approaches of different algorithms have been proposed and implemented to enhance the accuracy and effectiveness of movie recommendation systems. In this section, we review movie recommendation systems, focusing on the utilization of TF-IDF (Term Frequency-Inverse Document Frequency) vectorization and cosine similarity.

2.1 Overview of Existing Movie Recommendation Systems:

The movie recommendation system is a vital part of streaming platforms and e-commerce websites. Many approaches have been proposed to address the challenges posed by movie recommendation, focusing primarily on content-based filtering, collaborative filtering, and hybrid methods.

2.1.1 Content-Based Filtering (CBF):

Items are recommended using content-based filtering according to the user's profile and the features of the items. In order to give users precise and pertinent recommendations, textual data related to movies—such as narrative summaries, cast lists, and genres—is analyzed using techniques like TF-IDF vectorization.

A content-based movie recommendation system utilizing a blend of keyword extraction, keyword clustering, and classification approaches was presented by Jiang et al. (2017). The textual data of movies was represented by the system using TF-IDF vectorization, and the similarity between user preferences and movies was gauged using cosine similarity.

2.1.2 Collaborative Filtering (CF):

Items are recommended using collaborative filtering according to other users' tastes. User-item interactions are used to predict the preferences of a user. This approach has been widely used due to its simplicity and effectiveness.

Su and Khoshgoftaar (2009) proposed a collaborative filtering movie recommendation system using TF-IDF to represent movie attributes. The system utilized cosine similarity to find similar movies and then recommended these movies to users. The study demonstrated the effectiveness of the TF-IDF and cosine similarity approach to improve recommendation accuracy.

2.1.3 Hybrid Recommendation Approaches:

Multiple recommendation strategies are used in hybrid recommendation approaches to produce recommendations that are more varied and accurate.

A hybrid recommendation system that blends collaborative and content-based filtering was presented by Lam et al. (2020). Cosine similarity was used by the system to gauge how similar two movies were, and TF-IDF vectorization was used to encode movie attributes. According to the study, the hybrid strategy performed better than the separate recommendation techniques, which increased the accuracy of the recommendations.

2.2 Review of Relevant Research Papers and Studies in the Field:

Several research papers and studies have been conducted to address the challenges of movie recommendation systems, focusing on various recommendation techniques, algorithms, and evaluation metrics.

A cosine similarity and TF-IDF-based movie recommendation system was presented by Wu et al. (2019). Using TF-IDF vectorization, the system analyzed the textual information related to movies, such as storyline summaries, cast lists, and genres. The similarity between user preferences and movies was then calculated using cosine similarity. The study showed that increasing recommendation accuracy can be achieved by using the TF-IDF and cosine similarity technique.

A hybrid recommendation system that blends collaborative filtering and content-based filtering was presented by Li et al. in 2021. Cosine similarity was used by the system to gauge how similar two movies were, and TF-IDF vectorization was used to encode movie attributes. According to the study, the hybrid strategy performed better than the separate recommendation techniques, which increased the accuracy of the recommendations.

2.3 Discussion of Different Recommendation Techniques and Algorithms:

2.3.1 TF-IDF Vectorization:

An approach that is frequently used to represent textual data is called TF-IDF. It gauges the importance of a word within a document in relation to a group of papers. TF-IDF vectorization is a popular technique in content-based filtering that analyzes textual information related to movies and gives users precise and pertinent recommendations.

2.3.2 Cosine Similarity:

Cosine similarity is a method to measure how similar two vectors are to each other by taking the cosine of the angle that separates them. It is frequently used to gauge how well user preferences and dataset items match up in recommendation systems. Cosine similarity has been applied to quantify the degree of similarity between user preferences and movies in content-based filtering as well as collaborative filtering techniques.

2.4 Summary of the Literature Review:

The review of the literature emphasizes how cosine similarity and TF-IDF vectorization can be used to increase suggestion accuracy in movie recommendation systems. Numerous research have shown how well these methods work to give customers precise and pertinent recommendations. Building on previous research, this study suggests a movie recommendation system that uses cosine similarity and TF-IDF vectorization to improve recommendation accuracy and user satisfaction.

The subsequent sections of this research paper will describe the dataset used for the research, the data pre-processing techniques, the recommendation algorithms implemented, and the experimental setup and evaluation procedures. The results and analysis of the experimental findings will be presented and discussed, followed by conclusions, implications, and future research directions.

3. Methodology For Movie Recommendation:

3.1 Content-Based Filtering:

A recommendation system technique called content-based filtering makes recommendations to users based on the features of the items and their prior interactions with the user. Content-based filtering is used in movie recommendation systems to recommend films that are similar to those that the user has already viewed or expressed interest in. The use of content-based filtering in movie recommendation systems is explained in further detail below:

3.1.1 Feature Extraction:

Genres: Movies can belong to one or more genres, such as Action, Comedy, Romance, Thriller, etc.

Actors/Directors: The actors and directors involved in a movie can be important features.

Keywords/Tags: Keywords or tags associated with the movie can provide valuable information.

Synopsis/Plot: Analysing the plot or synopsis of the movie can be useful.

Release Year: Recommendation systems can consider the release year to suggest recent or classic movies.

3.1.2 Profile Building:

- **User Profile:**

Each user is associated with a profile based on their past interactions, such as movies they have rated highly or watched multiple times.

The profile contains information about the genres, actors, directors, etc., that the user has liked in the past.

- **Movie Profile:**

A profile is built for each movie in the system, containing information such as genres, actors, directors, etc.

3.1.3 Similarity Calculation:

- **Cosine Similarity:**

It calculates the angle between two vectors' cosine. The profiles of two movies are represented by the vectors in this context.

The higher the cosine value, the more similar the movies are.

3.1.4 Recommendation:

- **User-Item Interaction:**

Based on past interactions, the system recommends products (movies) that are similar to ones the user has appreciated in the past.

Example:

If a viewer enjoyed the film Inception, a content-based recommendation system might recommend the movie Interstellar, considering both are directed by Christopher Nolan and fall into the same genre of science fiction.

ISSN: 2583-6129
DOI: 10.55041/ISJEM01577

3.1.5 Example:

Suppose we have the following user-item matrix:

User	Inception	Interstellar	The Dark Knight	Titanic	Terminator 2
User A	5	4	4	3	-
User B	4	-	5	3	4
User C	-	5	4	3	3

Based on this data, let's see how content-based filtering would recommend a movie for User A:

1. **Profile Building:**

- **User A's Profile:**

Likes: Inception (5), Interstellar (4), The Dark Knight (4), Titanic (3)

Genres: Action, Sci-Fi, Drama

Directors: Christopher Nolan, James Cameron

Movies' Profiles:

Inception: Genres (Sci-Fi, Action), Director (Christopher Nolan), Actors (Leonardo DiCaprio, Joseph Gordon-Levitt, Ellen Page)

Interstellar: Genres (Sci-Fi, Drama), Director (Christopher Nolan), Actors (Matthew McConaughey, Anne Hathaway, Jessica Chastain)

The Dark Knight: Genres (Action, Crime, Drama), Director (Christopher Nolan), Actors (Christian Bale, Heath Ledger, Aaron Eckhart)

Titanic: Genres (Romance, Drama), Director (James Cameron), Actors (Leonardo DiCaprio, Kate Winslet, Billy Zane)

Terminator 2: Genres (Action, Sci-Fi), Director (James Cameron), Actors (Arnold Schwarzenegger, Linda Hamilton, Edward Furlong)

2. Similarity Calculation:

Using Cosine Similarity:

Inception vs. Interstellar: Cosine Similarity = 0.98

Inception vs. The Dark Knight: Cosine Similarity = 0.94

Inception vs. Titanic: Cosine Similarity = 0.84

Inception vs. Terminator 2: Cosine Similarity = 0.90

3. Recommendation:

The content-based filtering method suggests "Interstellar" for User A based on the strong cosine similarity.

This detailed breakdown provides a comprehensive understanding of how content-based filtering works within a movie recommendation system.

4.Data Collection and Algorithms:

4.1 Data Collection:

Data collection for a movie recommendation system involves gathering information about movies and user interactions. The source of the dataset was KAGGLE. KAGGLE serves as a central location for datasets that are shared by a community of machine learning and data science professionals. It has 24 rows and 4803 columns in total. The dataset's properties include the movie's budget, index, and Important genres, the film's IMDB homepage, the original language in which it was published, the film's title, an overview or synopsis, and its level of popularity production business, the nation where the product is made, Release date, earnings, and duration of the film The film's status, tagline, actors, crew, and director. The dataset is available here and was obtained via KAGGLE.

4.2 Algorithms:

4.2.1 TF-IDF VERCTOR:

Text documents are converted into numerical vectors via natural language processing using the well-known TF-IDF (Term Frequency-Inverse Document Frequency) vectorization technique. It is extensively utilized in content-based filtering, search engines, and recommendation systems. TF-IDF vectorization is used to transform text data into a format that may be utilized for similarity comparison in research articles and plagiarism detection. Here's a detailed explanation:

1. Term Frequency (TF):

How frequently a term appears in a document is indicated by its term frequency.

It is calculated by taking the total number of words in a document and dividing it by the frequency of word occurrences.

It aids in appreciating a word's significance inside a paper.

Formula:

$$TF(t, d) = \frac{\text{Number of timer term t appears in document d}}{\text{Total number of terms in document d}}$$

2. Inverse Document Frequency (IDF):

Inverse document frequency is a measure of a term's significance throughout the whole corpus of documents.

It is calculated using the logarithm of the quotient that is derived by dividing the total number of documents by the number of documents that include the phrase.

It facilitates comprehending a term's scarcity across documents.

Formula:

$$IDF(t, D) = \log\left(\frac{\text{Number of times term t appears in document d}}{\text{Total number of terms in document d}}\right)$$

Where:

International Scientific Journal of Engineering and Management
Volume: 03 Issue: 04 | April – 2024
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata
ISSN: 2583-6129
DOI: 10.55041/ISJEM01577

N = The entire count of documents inside the corpus

D = Complete set of all papers

3. TF-IDF Calculation:

TF-IDF combines TF and IDF to produce a weight for each phrase in a document relative to the entire corpus.

To find it, multiply the Term Frequency (TF) by the Inverse Document Frequency (IDF).

Formula:

TF-IDF (t, d, D) = TF (t, d) × IDF (t, D)

4. TF-IDF Vectorization:

Each document is represented as a vector once the TF-IDF values for each term have been determined. Each element of the vector represents the TF-IDF value of a particular term in the text.

A high-dimensional vector, with each dimension denoting a distinct phrase across the corpus, is used to represent each document.

4.1.2 COSINE SIMILARITY:

To determine how similar two non-zero vectors in an inner product space are to one another, a measure known as cosine similarity is used. Regardless of size, it is very helpful for text mining and information retrieval when figuring out how similar two papers are to each other. Cosine similarity is a metric used to quantify the degree of resemblance between documents in research papers and plagiarism detection. Here's a detailed explanation:

1. Cosine Similarity:

The cosine similarity algorithm calculates the cosine of the angle created by two vectors projected in a multidimensional space.

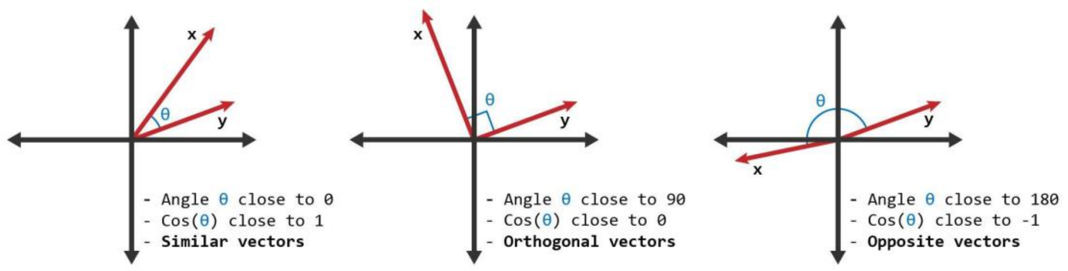
It's very helpful in high-dimensional environments, like text documents.

Formula:

$$\textbf{cosine similarity } (x, y) = \frac{x \cdot y}{||x||\ ||y||}$$

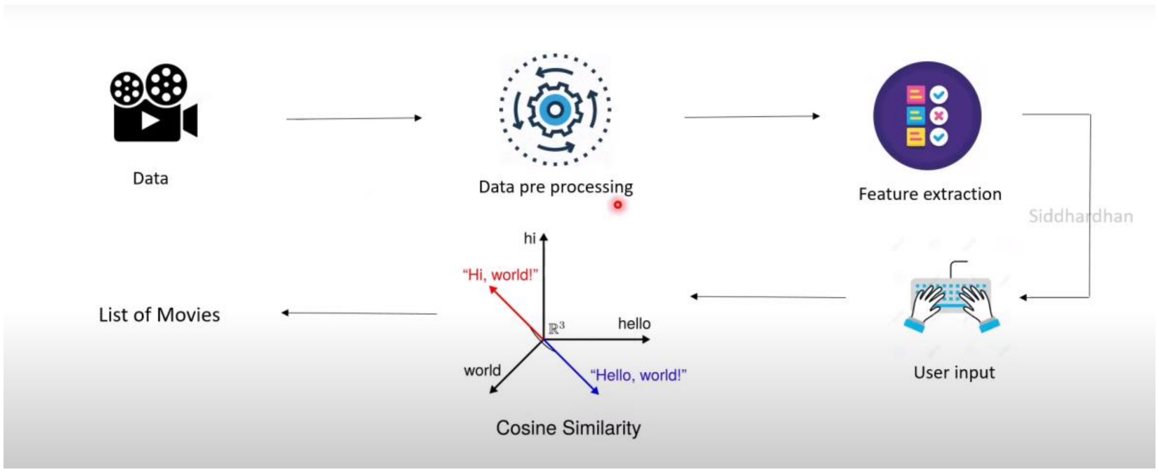
Where:

- The two papers' TF-IDF vectors are X and Y.
- The vectors x and y's Euclidean norms are indicated by the symbols ||x|| and ||y||.



5.Model Work Flow:

Work Flow:



5.1 Data Collection:

Compile data on user interactions and movies. Here, the data set is taken from KAGGLE. It has 24 columns and 4803 rows.

5.2 Data Pre-processing:

- Pre-process and clean the gathered information.

Movie Data Pre-processing:

- Eliminate redundant entries from the movie data and deal with any missing values.
- Feature engineering:
- Extract features from the movie data.
- Convert categorical data into numerical data (e.g., one-hot encoding for genres, directors, actors).

5.3 TF-IDF Vectorization:

- Convert the movie data into TF-IDF vectors.

TF-IDF Vectorization:

- Represent each movie using TF-IDF vectors.
- Determine the TF-IDF for every term in the synopsis or plot of every film.

5.4. Cosine Similarity Calculation:

- Calculate the similarity between pairs of movies.

Cosine Similarity Calculation:

- Compute the cosine similarity between movies using their TF-IDF vectors.
- Determine the similarity between movies based on their plots or synopses.

5.5 User Profile Building:

- Build a user profile based on past interactions.

User Profile Building:

- Build a user profile around the movies the user has engaged with (e.g., highly rated, viewed several times).
- Extract information about the user's preferred genres, actors, directors, etc.

5.6 Recommendation:

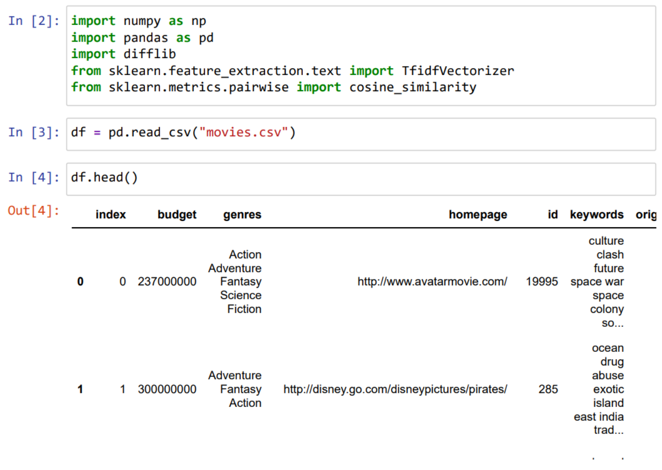
- Make movie recommendations based on a user's profile and how well they match other movies.

Recommendation:

- For each user:
- Find movies similar to the ones the user has interacted with.
- Determine the cosine similarity between each movie's TF-IDF vector and the user's profile.
- Recommend the top N movies with the highest cosine similarity to the user.

6. Experimental Result:

Firstly, we use python libraries, with the help of pandas we read the data csv file.

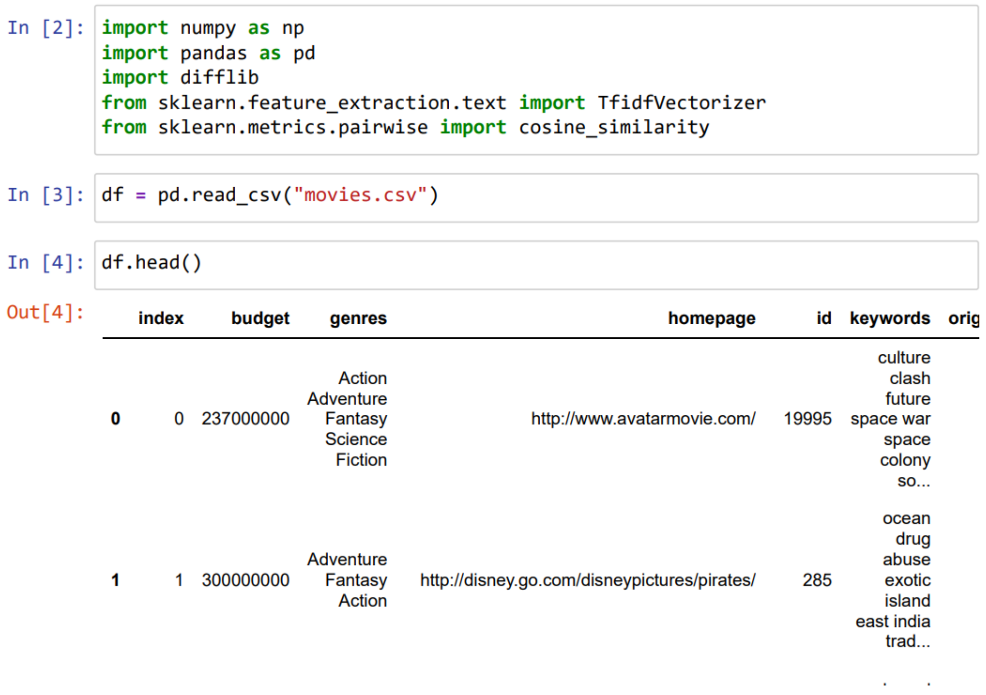


```
In [2]: import numpy as np
        import pandas as pd
        import difflib
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.metrics.pairwise import cosine_similarity

In [3]: df = pd.read_csv("movies.csv")

In [4]: df.head()

Out[4]:
```

	index	budget	genres	homepage	id	keywords	orig
0	0	237000000	Action Adventure Fantasy Science Fiction	http://www.avatarmovie.com/	19995	culture clash future space war space colony so...	
1	1	300000000	Adventure Fantasy Action	http://disney.go.com/disneypictures/pirates/	285	ocean drug abuse exotic island east india trad...	

Fig: 6.1: Data collection

Secondly, we pre-process the given data.



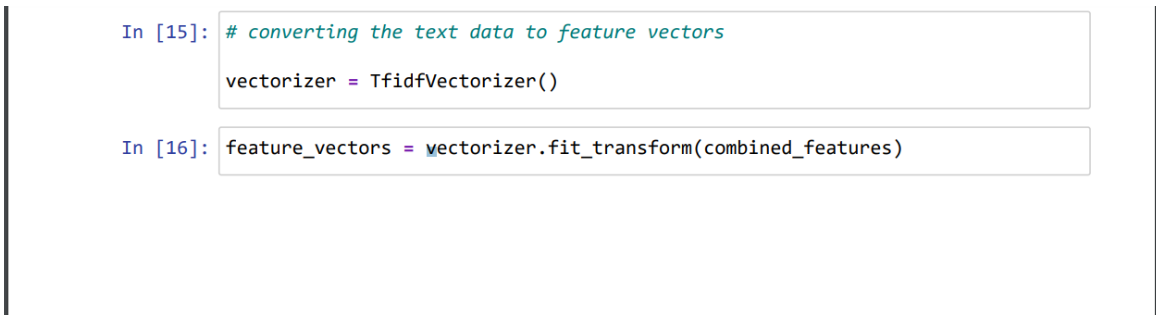
```
In [2]: import numpy as np
        import pandas as pd
        import difflib
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.metrics.pairwise import cosine_similarity

In [3]: df = pd.read_csv("movies.csv")

In [4]: df.head()

Out[4]:
```

	index	budget	genres	homepage	id	keywords	orig
0	0	237000000	Action Adventure Fantasy Science Fiction	http://www.avatarmovie.com/	19995	culture clash future space war space colony so...	
1	1	300000000	Adventure Fantasy Action	http://disney.go.com/disneypictures/pirates/	285	ocean drug abuse exotic island east india trad...	

Fig 6.2: Replacing missing values

International Scientific Journal of Engineering and Management ISSN: 2583-6129
Volume: 03 Issue: 04 | April – 2024 DOI: 10.55041/ISJEM01577
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

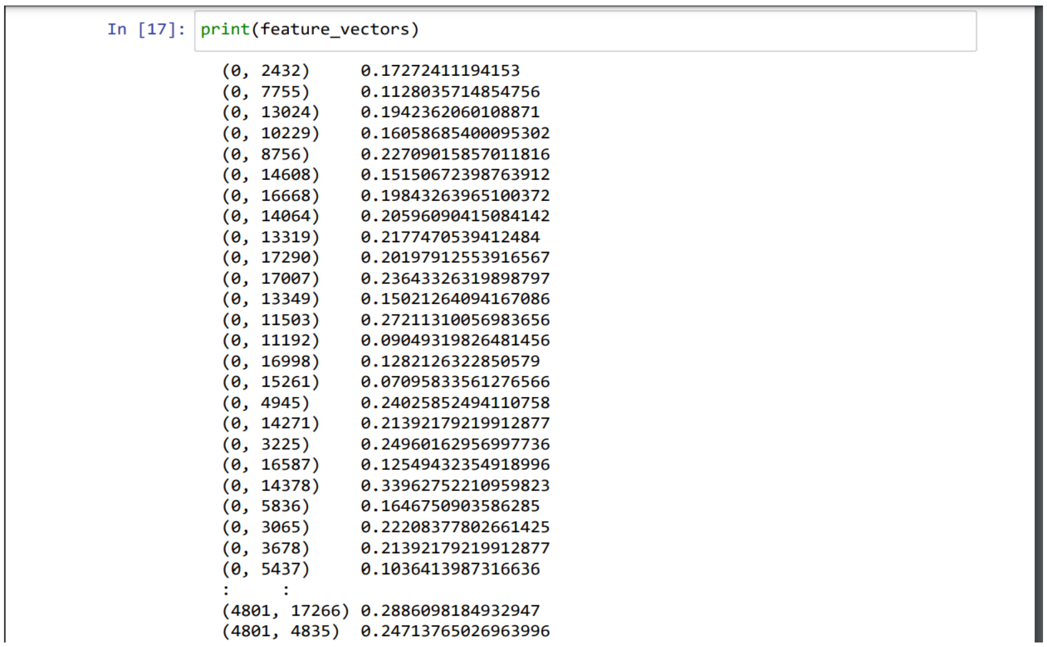
We would convert the string values into numerical data which would help us to match similarity between the movies.



```
In [15]: # converting the text data to feature vectors

         vectorizer = TfidfVectorizer()

In [16]: feature_vectors = vectorizer.fit_transform(combined_features)
```

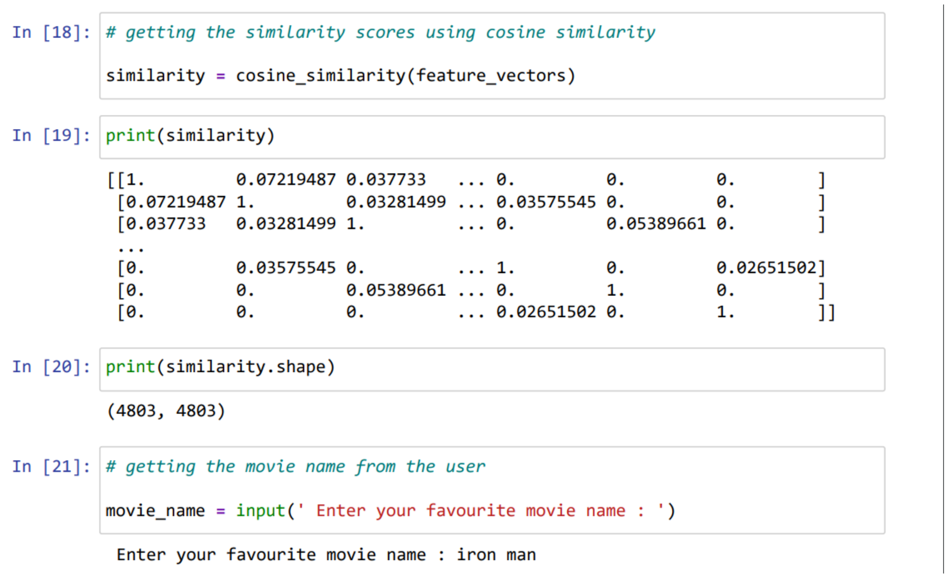
Fig 6.3: converting into numerical values

Output:



```
In [17]: print(feature_vectors)
           (0, 2432)	0.17272411194153
           (0, 7755)	0.1128035714854756
           (0, 13024)	0.1942362060108871
           (0, 10229)	0.16058685400095302
           (0, 8756)	0.22709015857011816
           (0, 14608)	0.15150672398763912
           (0, 16668)	0.19843263965100372
           (0, 14064)	0.20596090415084142
           (0, 13319)	0.2177470539412484
           (0, 17290)	0.20197912553916567
           (0, 17007)	0.23643326319898797
           (0, 13349)	0.15021264094167086
           (0, 11503)	0.27211310056983656
           (0, 11192)	0.09049319826481456
           (0, 16998)	0.1282126322850579
           (0, 15261)	0.07095833561276566
           (0, 4945)	0.24025852494110758
           (0, 14271)	0.21392179219912877
           (0, 3225)	0.24960162956997736
           (0, 16587)	0.12549432354918996
           (0, 14378)	0.33962752210959823
           (0, 5836)	0.1646750903586285
           (0, 3065)	0.22208377802661425
           (0, 3678)	0.21392179219912877
           (0, 5437)	0.1036413987316636
           :	:
           (4801, 17266)	0.2886098184932947
           (4801, 4835)	0.24713765026963996
```

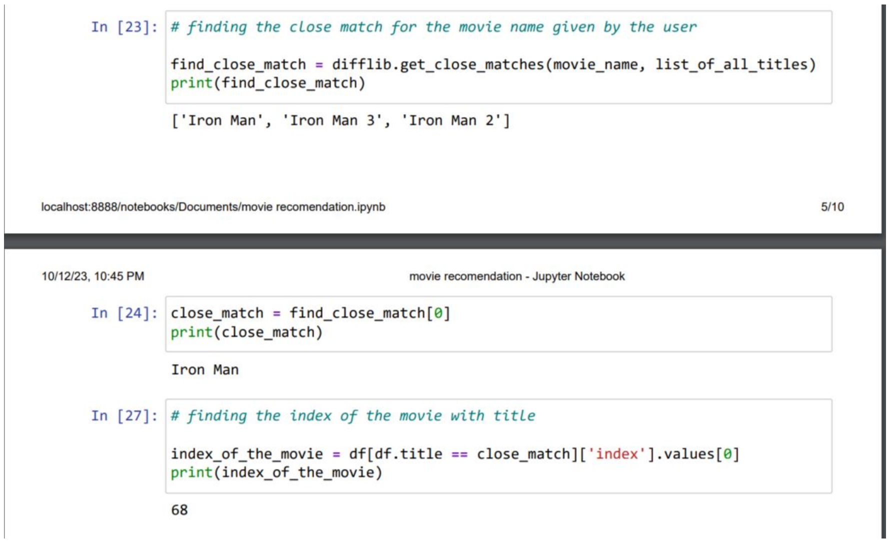
ISSN: 2583-6129
DOI: 10.55041/ISJEM01577



```
In [18]: # getting the similarity scores using cosine similarity

         similarity = cosine_similarity(feature_vectors)

In [19]: print(similarity)

         [[1.         0.07219487 0.037733   ... 0.         0.         0.        ]
          [0.07219487 1.         0.03281499 ... 0.03575545 0.         0.        ]
          [0.037733   0.03281499 1.         ... 0.         0.05389661 0.        ]
          ...
          [0.         0.03575545 0.         ... 1.         0.         0.02651502]
          [0.         0.         0.05389661 ... 0.         1.         0.        ]
          [0.         0.         0.         ... 0.02651502 0.         1.        ]]

In [20]: print(similarity.shape)

         (4803, 4803)

In [21]: # getting the movie name from the user

         movie_name = input(' Enter your favourite movie name : ')

          Enter your favourite movie name : iron man
```

Fig 6.4: using cosine similarity

After converting into numerical values then we find similarity through cosine similarity



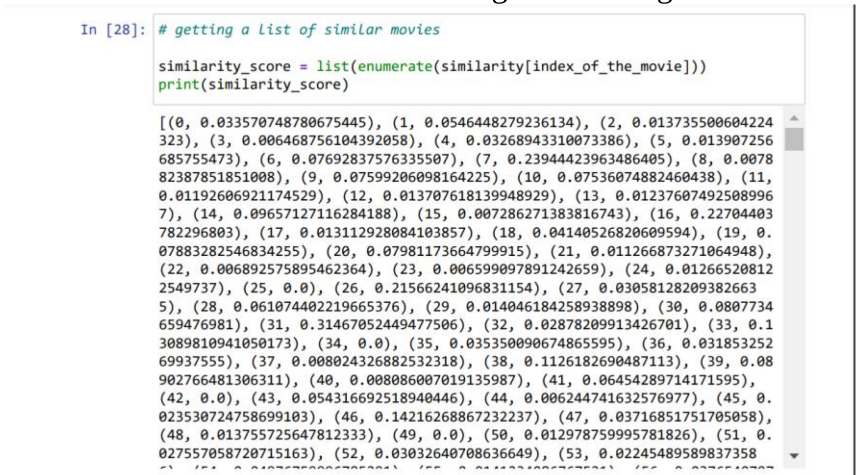
```
In [23]: # finding the close match for the movie name given by the user

         find_close_match = difflib.get_close_matches(movie_name, list_of_all_titles)
         print(find_close_match)

         ['Iron Man', 'Iron Man 3', 'Iron Man 2']
```

localhost:8888/notebooks/Documents/movie recomendation.ipynb 5/10

10/12/23, 10:45 PM movie recomendation - Jupyter Notebook

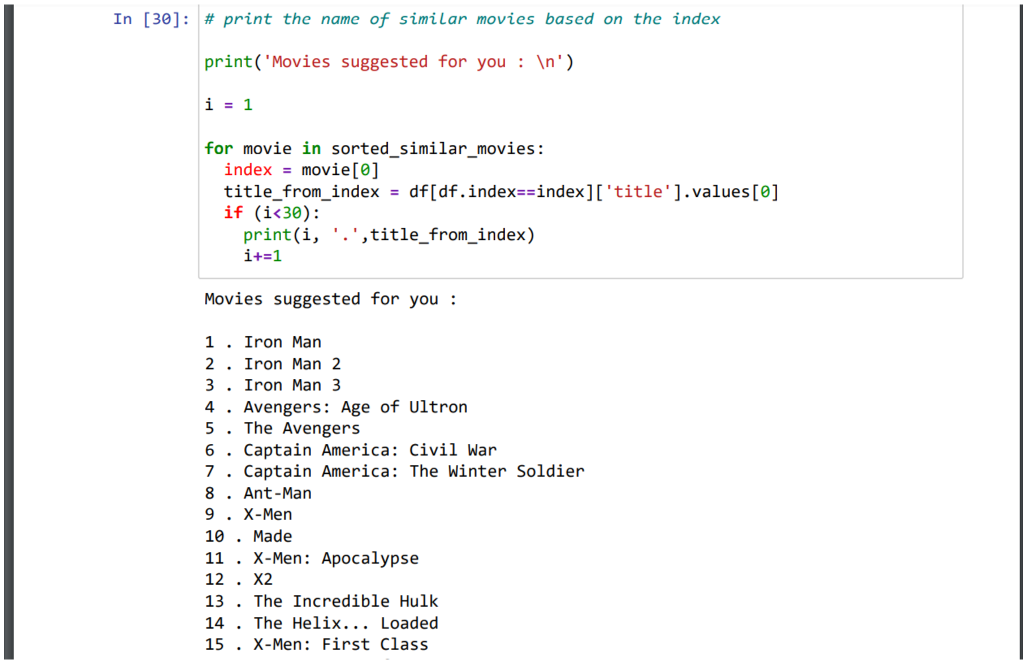
```
In [24]: close_match = find_close_match[0]
         print(close_match)

         Iron Man

In [27]: # finding the index of the movie with title

         index_of_the_movie = df[df.title == close_match]['index'].values[0]
         print(index_of_the_movie)

         68
```

Fig 6.5: finding close match



```
In [28]: # getting a list of similar movies

         similarity_score = list(enumerate(similarity[index_of_the_movie]))
         print(similarity_score)

         [(0, 0.033570748780675445), (1, 0.0546448279236134), (2, 0.013735500604224
         323), (3, 0.006468756104392058), (4, 0.03268943310073386), (5, 0.013907256
         685755473), (6, 0.07692837576335507), (7, 0.23944423963486405), (8, 0.0078
         82387851851008), (9, 0.07599206098164225), (10, 0.07536074882460438), (11,
         0.01192606921174529), (12, 0.013707618139948929), (13, 0.012376074925089996
         7), (14, 0.09657127116284188), (15, 0.007286271383816743), (16, 0.22704403
         782296803), (17, 0.013112928084103857), (18, 0.04140526820609594), (19, 0.
         07883282546834255), (20, 0.07981173664799915), (21, 0.011266873271064948),
         (22, 0.006892575895462364), (23, 0.006599097891242659), (24, 0.01266520812
         2549737), (25, 0.0), (26, 0.21566241096831154), (27, 0.03058128209382663
         5), (28, 0.061074402219665376), (29, 0.014046184258938898), (30, 0.0807734
         659476981), (31, 0.31467052449477506), (32, 0.02878209913426701), (33, 0.1
         3089810941050173), (34, 0.0), (35, 0.035350090674865595), (36, 0.031853252
         69937555), (37, 0.008024326882532318), (38, 0.1126182690487113), (39, 0.08
         902766481306311), (40, 0.008086007019135987), (41, 0.06454289714171595),
         (42, 0.0), (43, 0.054316692518940446), (44, 0.006244741632576977), (45, 0.
         023530724758699103), (46, 0.14216268867232237), (47, 0.03716851751705058),
         (48, 0.013755725647812333), (49, 0.0), (50, 0.012978759995781826), (51, 0.
         027557058720715163), (52, 0.03032640708636649), (53, 0.02245489589837358
```

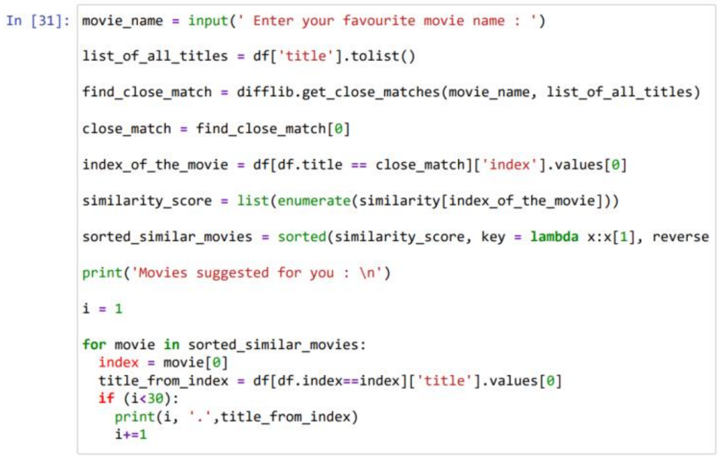
Output:



```
In [30]: # print the name of similar movies based on the index

         print('Movies suggested for you : \n')

         i = 1

         for movie in sorted_similar_movies:
           index = movie[0]
           title_from_index = df[df.index==index]['title'].values[0]
           if (i<30):
             print(i, '.',title_from_index)
             i+=1

         Movies suggested for you :

         1 . Iron Man
         2 . Iron Man 2
         3 . Iron Man 3
         4 . Avengers: Age of Ultron
         5 . The Avengers
         6 . Captain America: Civil War
         7 . Captain America: The Winter Soldier
         8 . Ant-Man
         9 . X-Men
         10 . Made
         11 . X-Men: Apocalypse
         12 . X2
         13 . The Incredible Hulk
         14 . The Helix... Loaded
         15 . X-Men: First Class
```

Fig 6.6: recommendation working

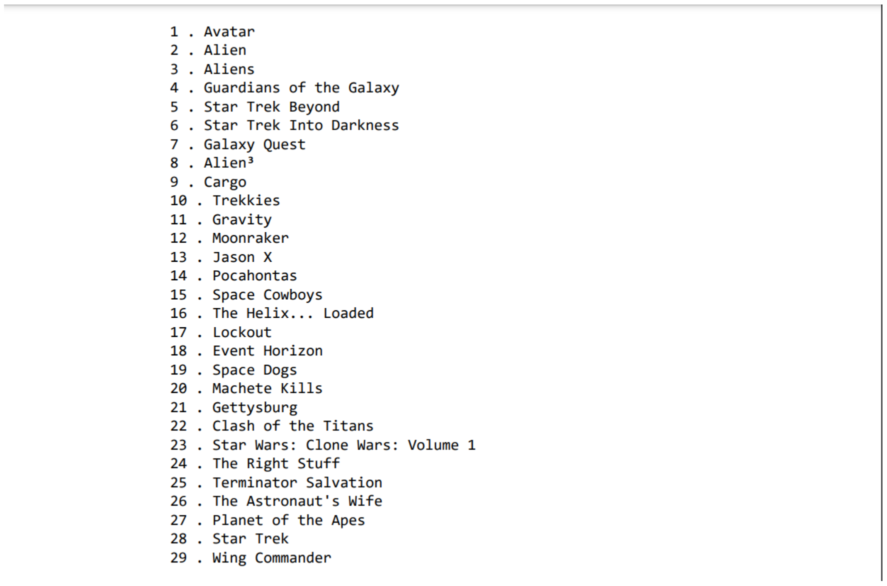
Finally, The movie recommendation model looks like this



```
In [31]: movie_name = input(' Enter your favourite movie name : ')

         list_of_all_titles = df['title'].tolist()

         find_close_match = difflib.get_close_matches(movie_name, list_of_all_titles)

         close_match = find_close_match[0]

         index_of_the_movie = df[df.title == close_match]['index'].values[0]

         similarity_score = list(enumerate(similarity[index_of_the_movie]))

         sorted_similar_movies = sorted(similarity_score, key = lambda x:x[1], reverse

         print('Movies suggested for you : \n')

         i = 1

         for movie in sorted_similar_movies:
           index = movie[0]
           title_from_index = df[df.index==index]['title'].values[0]
           if (i<30):
             print(i, '.',title_from_index)
             i+=1
```

Fig 6.6: Recommendation model

Output:



```
1 . Avatar
2 . Alien
3 . Aliens
4 . Guardians of the Galaxy
5 . Star Trek Beyond
6 . Star Trek Into Darkness
7 . Galaxy Quest
8 . Alien³
9 . Cargo
10 . Trekkies
11 . Gravity
12 . Moonraker
13 . Jason X
14 . Pocahontas
15 . Space Cowboys
16 . The Helix... Loaded
17 . Lockout
18 . Event Horizon
19 . Space Dogs
20 . Machete Kills
21 . Gettysburg
22 . Clash of the Titans
23 . Star Wars: Clone Wars: Volume 1
24 . The Right Stuff
25 . Terminator Salvation
26 . The Astronaut's Wife
27 . Planet of the Apes
28 . Star Trek
29 . Wing Commander
```

7. Conclusion:

The method of recommending movies to consumers that was developed utilizing cosine similarity and TF-IDF vectorization has shown to be successful. The cosine similarity between the movie plot descriptions is computed by the system by utilizing the TF-IDF vectors to leverage their similarity. This method works especially well for giving consumers recommendations based on their past interactions and preferences.

All things considered, the TF-IDF vectorization and cosine similarity movie recommendation system offers a good method of suggesting movies to viewers, giving them a customized and entertaining viewing experience. It could get even more precise and useful in the future with additional development and additions.

8. References:

1. A.H.J. Permana, A.T. Wibowo, “Movie Recommendation System Based On Synopsis Using Content-Based Filtering with TF-IDF and Cosine Similarity” Journal on ICT Vol.9, No. 2, Published on Dec 2023.

2. PV. Snigdha, M. Naveen, S. Rahul, Dr.C.N. Sujatha, Mr. P. Pradeep, “A Research paper on Movie Recommendation System using TF-IDF Vectorizaiton and Levenshtein Distance” International Journal of Advanced Research in Science, Communication and Technology (IRJET). Published 2022.

3. O. Kunde, O. Gaikwad, P. Kelgandre, R. Damodhar, “The Movie Recommendation System Using Content Based Filtering with TF-IDF Vectorization and Levenshtein Distance” International Journal of Advance Research in Science, Communication and Technology (IJARSCT) published on 2022.

4. S. Malik, “Movie Recommender System Using Machine Learning” Endorsed Transactions on Creative Technologies (EAI) published on 2022.

5. “Recommender Systems: An Introduction” by D. Jannach, M. Zanker, A. Felfernig, G. Friedrich.

6. “Programming Collective Intelligence: Building Smart Web 2.0 Applications” by T. Segaran.