



PICK-A-MOVIE

(Using Machine Learning)

Name – Sarthak Agrawal

MCA 3rd Semester

Jan-2023 Batch

(Machine Learning and Artificial Intelligence)

Enrollment No. - A9929722000077 (el)

ABSTRACT

PICK-A-MOVIE which is a Movie Recommendation System, is an example of Supervised Machine Learning approach. This analysis mainly focuses on finding the similarity scores between the two contents in the content-based filtering. It helps us in finding the distance between the two vectors and their angle by the help of the **cosine similarity** formula and magnitude of their relative scores.

This model analyses the points to find the two text similarity scores by using Jupyter Notebook and distance between the two-vector model approach more effectively and precisely. In today's computer world we have lots of stuff on our Internet sources to watch and see but every single stuff available does not match our liking. We sometimes get the feed of the videos, news, clothing etc. which is not according to our liking and interest. It makes the customer interest in the application lower, and he/she further doesn't want to get through the same application again.

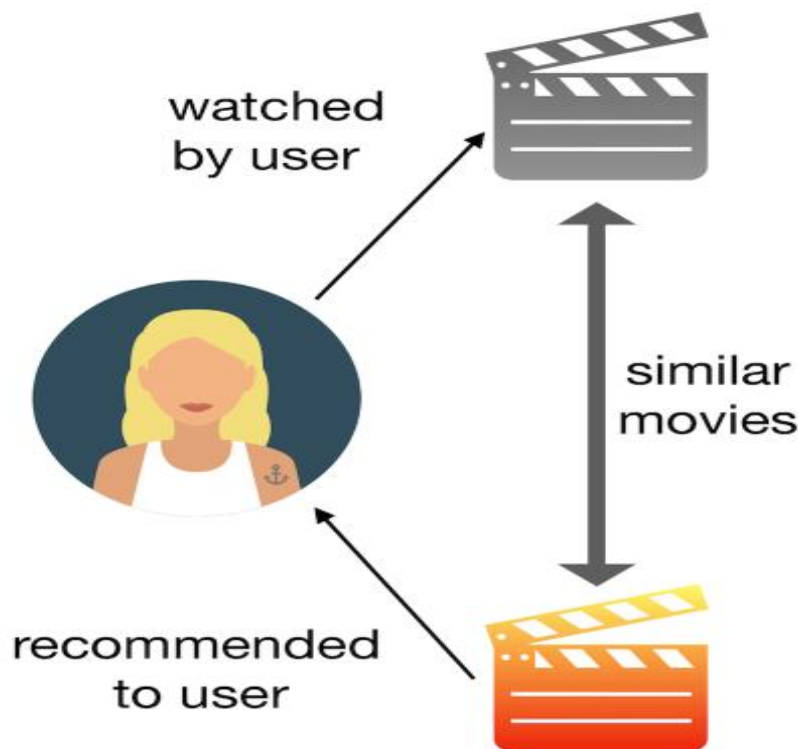
The need of the hour is to develop some code which can tell at a beginner level the matching pattern of the customer trend and recommend him with the best item of his interest level. This will help us in making the customer experience satisfactory and able to achieve good ratings and popularity as well. For example, if we tend to see a particular genre of movie more frequently, then the system recommends that genre over the others when recommending what to watch next. It gains popularity by application rating and at the same time enhances the user experience.

This policy of recommendation system is really helpful in giving optimum results to an application profitability and to make the organisation more connected. We can also see the recommendation work in online food applications such as Zomato and Swiggy which offers their customers the food restaurants which caters to their taste in food. They learn upon the behaviour of the user from the previous orders and tries to impress them with the latest additions of their favourite cuisines and stuffs. The recommendation engine implements ML algorithms that compute the similarity scores between specified features.

INTRODUCTION

Supervised Machine Learning is used when the model is getting trained on a labelled dataset.

Labelled dataset is the one which has both input and output parameters. In this type of learning both training and validation datasets are labelled. Here, we have three components such as Training Data, Test Data and features. Training data is that where data is usually split in the ratio of 80:20 i.e., 80% as training data and rest as testing data. Testing data is that when data is good to be tested. At the time of testing, input is fed from remaining 20% data which the model has never seen before, the model will predict some value and we will compare it with actual output and calculate the accuracy.



The different steps of this ML approach are as follows:

- 1.) **Data Extraction and Cleaning:** This is used to extract and clean the data by using scripting languages such as Python, Shell Scripting etc. Here we extract and filter the useful data of movies dataset according to our need.
- 2.) **Build the ML Model:** Once we extract and clean the data, we start building up the model with tools such as Scikit-Learn, DiffliB, Pandas etc. We build our movie recommendation engine here which would be in form of a Python script.
- 3.) **Build Software Infrastructure:** If we want to integrate this engine into an application or website for the users, we use the ML algorithm in the form of a software by using JavaScript and other tools.

Users- They are the one who uses these services or acts as the consumer.

Items- Here these are the different sets of movies which are been recommended in a sort of zig-zag manner according to our previous searches.

There are three ways of performing the filtering. They are as follows-

- **Trending based filtering-** Here the movies are classified by the ratings and the stuff that is been liked by majority of the population.
- **Content based filtering-** Here the similar articles are recommended to the user according to his previous content search.
- **Collaborative based filtering-** Here the two similar user likings act as a recommendation criterion to each other. Like, the two users watch comedy movies so if a new comedy stuff appears and is watched by user A it will also be recommended to user B.

A recommender system, or a recommendation system (sometimes replacing 'system' with a synonym such as platform or engine), is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. They are primarily used in commercial applications. Recommender systems are utilized in a variety of areas and are most widely recognized as playlist generators for video and music services like Netflix, YouTube and Spotify, product recommenders for services such as Amazon, or content recommenders for social media platforms such as Facebook or Twitter. These systems can operate using a single input, like music, or multiple inputs within and across platforms like news, books, and search queries. Recommender systems have also been developed to explore research articles and financial services.

These both are the recommendation engines that recommend us the movies and other related stuff based on our previous searches and watched experience.

- **Classification:** It is a Supervised Learning task where output is has defined labels (discrete values).
- **Regression:** It is a Supervised Learning method where the target feature must have continuous values.

OBJECTIVE

A recommendation system or recommendation engine is a model used for information filtering where it tries to predict the preferences of a user and provide suggests based on these preferences. These systems have become increasingly popular nowadays and are widely used today in areas such as movies, music, books, videos, clothing, restaurants, food, places and other utilities. These systems collect information about a user's preferences and behaviour, and then use this information to improve their suggestions in the future.

Movies are a part and parcel of life. There are different types of movies like some for entertainment, some for educational purposes, some are animated movies for children, and some are horror movies or action films. Movies can be easily differentiated through their genres like comedy, thriller, animation, action etc. Other way to distinguish among movies can be either by releasing year, language, director etc. Watching movies online, there are several movies to search in our most liked movies.

Movie Recommendation Systems helps us to search our preferred movies among all of these different types of movies and hence reduce the trouble of spending a lot of time searching our favourable movies. So, it requires that the movie recommendation system should be very reliable and should provide us with the recommendation of movies which are exactly same or most matched with our preferences. A large number of companies are making use of recommendation systems to increase user interaction and enrich a user's shopping experience. Recommendation systems have several benefits, the most important being customer satisfaction and revenue. Movie Recommendation system is very powerful and important system. But, due to the problems associated with pure collaborative approach, movie recommendation systems also suffers with poor recommendation quality and scalability issues.

The goal of the project is to recommend a movie to the user. Providing related content out of relevant and irrelevant collection of items to users of online service providers.

LITERATURE REVIEW

(Background Study)

♦ Movie Recommendation System by K-Means Clustering And K-Nearest Neighbor

A recommendation system collect data about the user's preferences either implicitly or explicitly on different items like movies. An implicit acquisition in the development of movie recommendation system uses the user's behaviour while watching the movies. On the other hand, a explicit acquisition in the development of movie recommendation system uses the user's previous ratings or history. The other supporting technique that are used in the development of recommendation system is clustering. Clustering is a process to group a set of objects in such a way that objects in the same clusters are more similar to each other than to those in other clusters. K-Means Clustering along with K-Nearest Neighbour is implemented on the movie lens dataset in order to obtain the best-optimized result. In existing technique, the data is scattered which results in a high number of clusters while in the proposed technique data is gathered and results in a low number of clusters. The process of recommendation of a movie is optimized in the proposed scheme. The proposed recommender system predicts the user's preference of a movie on the basis of different parameters. The recommender system works on the concept that people are having common preference or choice. These users will influence on each other's opinions. This process optimizes the process and having lower RMSE.

♦ Movie Recommendation System Using Collaborative Filtering

Collaborative filtering systems analyse the user's behaviour and preferences and predict what they would like based on similarity with other users. There are two kinds of collaborative filtering systems; user-based recommender and item-based recommender. 1. Use-based filtering: User-based preferences are very common in the field of designing personalized systems. This approach is based on the user's likings. The process starts with users giving ratings (1-5) to some movies. These ratings can be implicit or explicit. Explicit ratings are when the user explicitly rates the item on some scale or indicates a thumbs-up/thumbs-down to the item. Often explicit ratings are hard to gather as not every user is much interested in providing feedbacks. In these scenarios, we gather implicit ratings based on their behaviour. For instance, if a user buys a product more than once, it indicates a positive preference. In context to movie systems, we can imply that if a user watches the entire movie, he/she has some likeability to it. Note that there are no clear rules in determining implicit ratings. Next, for each user, we first find some defined number of nearest neighbours. We calculate correlation between users' ratings using Pearson Correlation algorithm. The assumption that if two users' ratings are highly correlated, then these two users must enjoy similar items and products is used to recommend items to users. 2. Item-based filtering: Unlike the user-based filtering method, item-based focuses on the similarity between the item's users like instead of the users themselves. The most similar items are computed ahead of time.

RESEARCH METHODOLOGY

In order to achieve the goal of the project, the first process is to do enough background study, so the literature study will be conducted. The whole project is based on a big amount of movie data so that we choose quantitative research method. For philosophical assumption, positivism is selected because the project is experimental and testing character. The research approach is deductive approach as the improvement of our research will be tested by deducing and testing a theory. Ex post facto research is our research strategy, the movie data is already collected and we don't change the independent variables. We use experiments to collect movie data. Computational mathematics is used data analysis because the result is based on improvement of algorithm. For the quality assurance, we have a detail explanation of algorithm to ensure test validity. The similar results will be generated when we run the same data multiple times, which is for reliability. We ensure the same data leading to same result by different researchers.

Methodology Adopted - Agile Methodology

Research Design – The research design that was used in this study is both 'Descriptive' and 'Exploratory'.

Sampling Size – 4803 movie records in the data file.

1. **Collecting the data sets:** Collecting all the required data set from Kaggle website. In this project we require movies.csv file.
2. **Data Analysis:** Make sure that that the collected data sets are correct and analyse the data in the csv files. i.e., checking whether all the column fields are present in the data sets or not.
3. **Algorithms:** In our project we have only two algorithms one is cosine similarity and other is single valued decomposition are used to build the machine learning recommendation model.
4. **Training and Testing the model:** Once the implementation of algorithm is completed. we have to train the model to get the result. We have tested it several times the model is recommend different set of movies to different users.
5. **Improvements in the project:** In the later stage we can implement different algorithms and methods for better recommendation.

DATA INTERPRETATION

Content-based Filtering

Content based recommender works with data that the user provides, either explicitly (rating) or implicitly (clicking on a link). Based on that data, a user profile is generated, which is then used to make suggestions to the user. As the user provides more inputs or takes actions on the recommendations, the engine becomes more and more accurate.

Algorithm : 1. : Collect the user reviews or movie name as the input from user.

2: Extract the name from the user input.

3. Convert the dataset feature values into feature vectors.

3: Compare the extracted name from the user's input with dataset using cosine similarity.

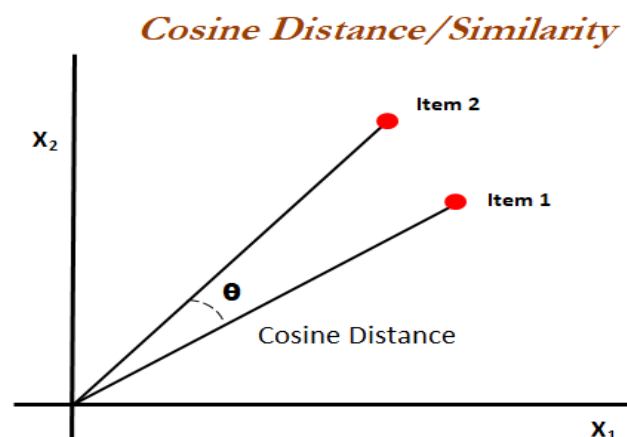
4: Learn user's profile based on the rated items and make recommendation based on top ranked items after comparing cosine values.

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them.

Formula:

$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2} \sqrt{\sum_1^n b_i^2}}$$

where, $\vec{a} \cdot \vec{b} = \sum_1^n a_i b_i = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$ is the dot product of the two vectors.



Recommendation System is a system which is used for filtering the information in the system that predicts rating for a given item. Recommended system identifies recommendations for

individual users based on past acquisition and searches, and on other users behaviour. Recommended Systems are software tools and techniques providing suggestions for items to be of use to a user. The system helps users to match with the items which they are interested in. It will support and improve the quality of the decisions which the users searching the items in the online.

Model Implementation :-

1. First we import all the required modules in the model.

Importing the required modules

```
In [1]: import numpy as np
import pandas as pd
import difflib
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

2. After importing the required modules, we upload the data file.

Uploading Data File

```
In [2]: # Loading the data from the csv file to a pandas dataframe
movies_data = pd.read_csv('movies.csv')
```

```
In [3]: # printing the first 5 rows of the dataframe to check the CSV file
movies_data.head()
```

Out[3]:

	index	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	...	runtime
0	0	237000000	Action Adventure Fantasy Science Fiction	http://www.avatarmovie.com/	19995	culture clash future space war space colony so...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437577	...	162.0
1	1	300000000	Adventure Fantasy Action	http://disney.go.com/disneypictures/pirates/	285	ocean drug abuse exotic island east india trad...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	139.082615	...	169.0
2	2	245000000	Action Adventure Crime	http://www.sonypictures.com/movies/spectre/	206647	spy based on novel secret agent sequel mi6	en	Spectre	A cryptic message from Bond's past sends him o...	107.376788	...	148.0
3	3	250000000	Action Crime Drama Thriller	http://www.thedarkknighttrises.com/	49026	dc comics crime fighter terrorist secret ident...	en	The Dark Knight Rises	Following the death of District Attorney Harve...	112.312950	...	165.0
4	4	260000000	Action Adventure Science Fiction	http://movies.disney.com/john-carter	49529	based on novel mars medallion space travel pri...	en	John Carter	John Carter is a war-weary, former military ca...	43.926995	...	132.0

5 rows x 24 columns

3. After the data file upload, Users reviews are collected as data set. The data set is pre-processed to remove the unwanted texts and missing values and we select relevant feature to use to predict the movie name.

```
In [4]: # checking number of rows and columns in the data frame
movies_data.shape
```

```
Out[4]: (4803, 24)
```

```
In [5]: # selecting the relevant features for recommendation system

selected_features = ['genres', 'keywords', 'tagline', 'cast', 'director']
print(selected_features)

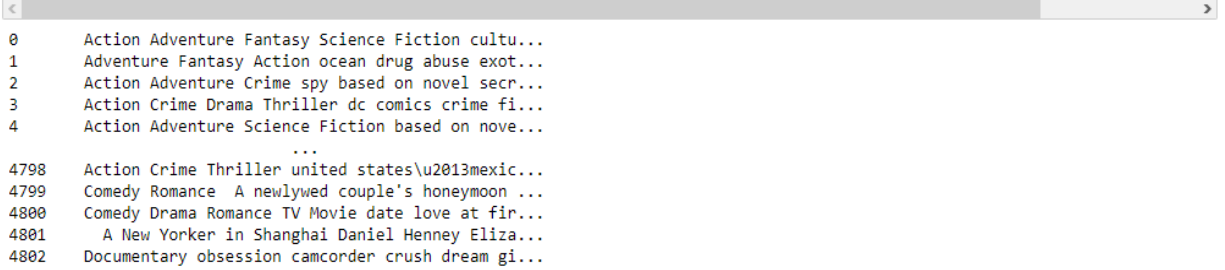
['genres', 'keywords', 'tagline', 'cast', 'director']
```

```
In [6]: # replacing the null values in the dataframe with NULL string

for feature in selected_features:
    movies_data[feature] = movies_data[feature].fillna('')
```

```
In [7]: # combining all the 5 selected features

combined_features = movies_data['genres']+' '+movies_data['keywords']+' '+movies_data['tagline']+' '+movies_data['cast']+' '+movies_data['director']
print(combined_features)
```

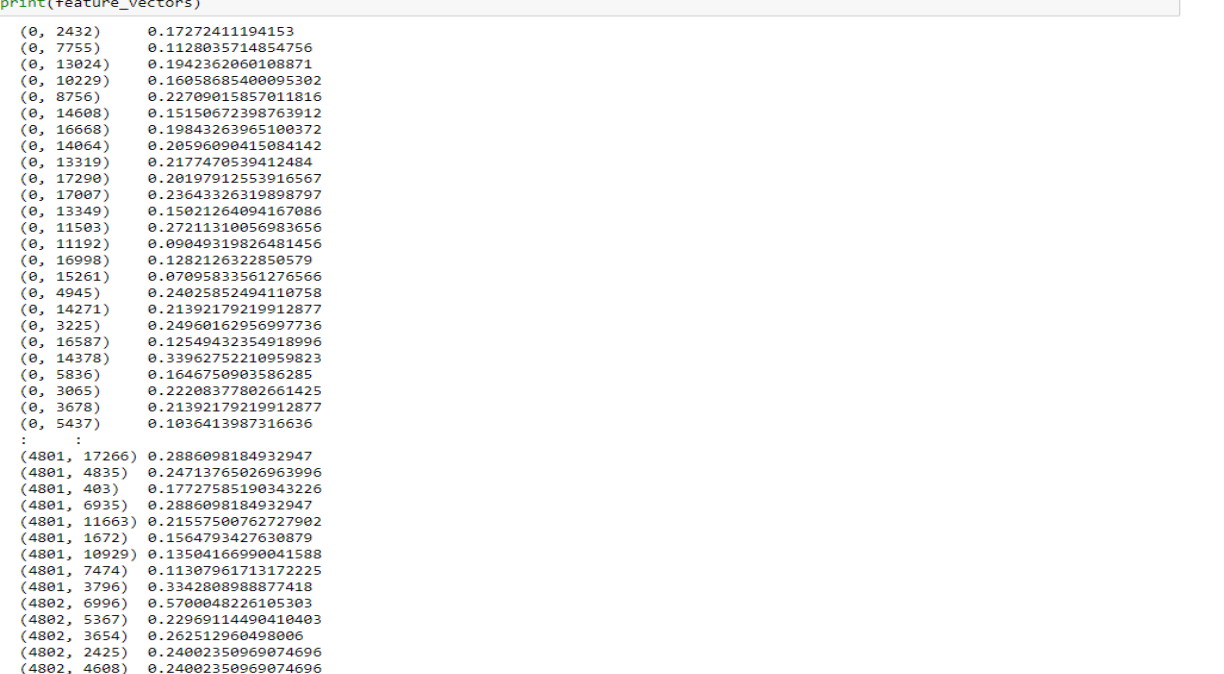


```
0      Action Adventure Fantasy Science Fiction cultu...
1      Adventure Fantasy Action ocean drug abuse exot...
2      Action Adventure Crime spy based on novel secr...
3      Action Crime Drama Thriller dc comics crime fi...
4      Action Adventure Science Fiction based on nove...
...
4798    Action Crime Thriller united states\u2013mexic...
4799    Comedy Romance  A newlywed couple's honeymoon ...
4800    Comedy Drama Romance TV Movie date love at fir...
4801      A New Yorker in Shanghai Daniel Henney Eliza...
4802    Documentary obsession camcorder crush dream gi...
Length: 4803, dtype: object
```

4. Now, we convert the 'combined_features' data to feature vectors.

```
In [8]: # converting the text data to feature vectors

vectorizer = TfidfVectorizer()
feature_vectors = vectorizer.fit_transform(combined_features)
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
(4801, 403)   0.17727585190343226
(4801, 6935)  0.2886098184932947
(4801, 11663) 0.21557500762727902
(4801, 1672)  0.1564793427630879
(4801, 10929) 0.13504166990041588
(4801, 7474)  0.11307961713172225
(4801, 3796)  0.3342808988877418
(4802, 6996)  0.5700048226105303
(4802, 5367)  0.22969114490410403
(4802, 3654)  0.262512960498006
(4802, 2425)  0.24002350969074696
(4802, 4608)  0.24002350969074696
(4802, 6417)  0.21753405888348784
```

5. Now, we apply cosine similarity function to get cosine values in shape of a matrix.

Using Cosine Similarity

```
In [9]: # getting the similarity scores using cosine similarity

similarity = cosine_similarity(feature_vectors)

print(similarity)

[[1.          0.07219487 0.0377733 ... 0.          0.          0.          ]
 [0.07219487 1.          0.03281499 ... 0.03575545 0.          0.          ]
 [0.0377733  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 [10]: print(similarity.shape)

(4803, 4803)
```

- Now, we take the movie name input from the user and find a close match in the list of all available movie titles.

```
In [11]: # getting the movie name from the user

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

Enter your favourite movie name : batman
```

```
In [12]: # creating a List with all the movie names given in the dataset

list_of_all_titles = movies_data['title'].tolist()
print(list_of_all_titles)
```

['Avatar', 'Pirates of the Caribbean: At World's End', 'Spectre', 'The Dark Knight Rises', 'John Carter', 'Spider-Man 3', 'Tangled', 'Avengers: Age of Ultron', 'Harry Potter and the Half-Blood Prince', 'Batman v Superman: Dawn of Justice', 'Superman Returns', 'Quantum of Solace', 'Pirates of the Caribbean: Dead Man's Chest', 'The Lone Ranger', 'Man of Steel', 'The Chronicles of Narnia: Prince Caspian', 'The Avengers', 'Pirates of the Caribbean: On Stranger Tides', 'Men in Black 3', 'The Hobbit: The Battle of the Five Armies', 'The Amazing Spider-Man', 'Robin Hood', 'The Hobbit: The Desolation of Smaug', 'The Golden Compass', 'King Kong', 'Titanic', 'Captain America: Civil War', 'Battleship', 'Jurassic World', 'Skyfall', 'Spider-Man 2', 'Iron Man 3', 'Alice in Wonderland', 'X-Men: The Last Stand', 'Monsters University', 'Transformers: Revenge of the Fallen', 'Transformers: Age of Extinction', 'Oz: The Great and Powerful', 'The Amazing Spider-Man 2', 'TRON: Legacy', 'Cars 2', 'Green Lantern', 'Toy Story 3', 'Terminator Salvation', 'Furious 7', 'World War Z', 'X-Men: Days of Future Past', 'Star Trek Into Darkness', 'Jack the Giant Slayer', 'The Great Gatsby', 'Prince of Persia: The Sands of Time', 'Pacific Rim', 'Transformers: Dark of the Moon', 'Indiana Jones and the Kingdom of the Crystal Skull', 'The Good Dinosaur', 'Brave', 'Star Trek Beyond', 'WALL-E', 'Rush Hour 3', '2012', 'A Christmas Carol', 'Jupiter Ascending', 'The Legend of Tarzan', 'The Chronicles of Narnia: The Lion, the Witch and the Wardrobe', 'X-Men: Apocalypse', 'The Dark Knight', 'Up', 'Monsters vs Aliens', 'Iron Man', 'Hugo', 'Wild Wild West', 'The Mummy: Tomb of the Dragon Emperor', 'Suicide Squad', 'Evan Almighty', 'Edge of Tomorrow', 'Waterworld', 'G.I. Joe: The Rise of Cobra', 'Inside Out', 'The Jungle Book', 'Iron Man 2', 'Snow White and the Huntsman', 'Maleficent', 'Dawn of the Planet of the Apes', 'The Lovers', '47 Ronin', 'Captain America: The Winter Soldier', 'Shrek Forever After', 'Tomorrowland', 'Big Hero 6', 'Wreck-It Ralph', 'The Polar Express', 'Independence Day: Resurgence', 'How to Train Your Dragon', 'Terminator 3: Rise of the Machines', 'Guardians of the Galaxy', 'Interstellar', 'Inception', 'Shin Godzilla', 'The Hobbit: An Unexpected Journey', 'The Fast and the Furious', 'The Curious Case of Benjamin Button', 'X-Men: First Class', 'The Hunger G

```
In [13]: # 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)

['Batman', 'Batman', 'Catwoman']
```

```
In [14]: close_match = find_close_match[0]
print(close_match)
```

Batman

7. Find the index of the movie that was the closest match and arrange the similarity scores in descending order.

```
In [15]: # finding the index of the movie with title
```

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

```
1359
```

```
In [16]: # getting a List of similar movies
```

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

```
[(0, 0.02531512269737111), (1, 0.04983293064399152), (2, 0.013599520029326722), (3, 0.20438773732168222), (4, 0.0249297267235
26918), (5, 0.11533013884014888), (6, 0.0), (7, 0.005521938648290493), (8, 0.03093493520052642), (9, 0.0897616996163815), (1
0, 0.11240850198526249), (11, 0.011808001891564552), (12, 0.03825857693295372), (13, 0.01326374702786566), (14, 0.12815026535
769386), (15, 0.010432758242497996), (16, 0.005235971636780309), (17, 0.01601669810036477), (18, 0.04153881551585253), (19,
0.02968626711977731), (20, 0.025210054010122055), (21, 0.004591239318101996), (22, 0.01792276874908081), (23, 0.0248704381665
94427), (24, 0.04186550745628901), (25, 0.0), (26, 0.005699120907668437), (27, 0.013106673998149269), (28, 0.0143895813353007
13), (29, 0.005723805699010054), (30, 0.11336219425554919), (31, 0.02290308453698018), (32, 0.0899394677673743), (33, 0.00548
7629188043741), (34, 0.0), (35, 0.025792416651174273), (36, 0.04323488875573723), (37, 0.022073100012780275), (38, 0.01673525
1293804827), (39, 0.03733968312911064), (40, 0.029604400185067402), (41, 0.10650154604688718), (42, 0.07244022042935871), (4
3, 0.023689369843914104), (44, 0.005896865705087829), (45, 0.004962729367343318), (46, 0.05119279389322904), (47, 0.006056967
859003828), (48, 0.01680183887529584), (49, 0.007462062943836138), (50, 0.024473647577403737), (51, 0.004490688090626269), (5
2, 0.03018245470107544), (53, 0.024852760467374575), (54, 0.038282186098375935), (55, 0.005755274570173456), (56, 0.006136094
904267454), (57, 0.009917061156672187), (58, 0.025146268236415702), (59, 0.013588616309692061), (60, 0.0), (61, 0.01635015002
611376), (62, 0.0054166345344265655), (63, 0.01833889030024455), (64, 0.028404894035231252), (65, 0.1775581506611392), (66,
0.01794874160703813), (67, 0.0), (68, 0.005130123069684274), (69, 0.01618329334393077), (70, 0.005371331822946504), (71, 0.02
1555882140030593), (72, 0.12266571244563478), (73, 0.04415940718548016), (74, 0.005495141965586583), (75, 0.02205711969120782
4), (76, 0.005516189273301243), (77, 0.008716823552559781), (78, 0.012046810653425794), (79, 0.029113928626152034), (80, 0.02
148467123933299), (81, 0.02794528136793173), (82, 0.00576827305044206), (83, 0.019994216200751452), (84, 0.01330888670622219
6), (85, 0.017148934252787445), (86, 0.009327381057907095), (87, 0.0), (88, 0.014079698000279712), (89, 0.02571541018199715
5), (90, 0.0000000000000000), (91, 0.0000000000000000), (92, 0.0000000000000000), (93, 0.0000000000000000), (94, 0.0000000000000000), (95, 0.0000000000000000), (96, 0.0000000000000000), (97, 0.0000000000000000), (98, 0.0000000000000000), (99, 0.0000000000000000)
```

```
In [17]: len(similarity_score)
```

```
Out[17]: 4803
```

```
In [18]: # sorting the movies based on their similarity score
```

```
sorted_similar_movies = sorted(similarity_score, key = lambda x:x[1], reverse = True)
print(sorted_similar_movies)
```

```
[(1359, 1.0), (428, 0.4311643836232694), (210, 0.25737999820859625), (3, 0.20438773732168222), (119, 0.19262528757150407), (6
5, 0.1775581506611392), (1512, 0.14705162654306442), (813, 0.14414128303962467), (2530, 0.13737322473729185), (1017, 0.137132
81929528845), (473, 0.13217075714794208), (753, 0.13216136203404205), (278, 0.12996260715124025), (14, 0.12815026535769386),
(72, 0.12266571244563478), (2313, 0.1183433298876972), (2381, 0.11752380293327759), (5, 0.11533013884014888), (2858, 0.114136
52180839844), (4183, 0.11353876531321921), (30, 0.11336219425554919), (2655, 0.11270953916613297), (10, 0.11240850198526249),
(1035, 0.11197582745207195), (2685, 0.10980421440315517), (870, 0.10722005407914785), (41, 0.10650154604688718), (1296, 0.105
84478077207024), (438, 0.10277392559139559), (1477, 0.10199709558322863), (299, 0.10154632458465614), (1474, 0.10078843818576
026), (1002, 0.1005262424012352), (1085, 0.10023066759543668), (2753, 0.09951168410001496), (303, 0.09883937789831629), (280
```

8. Finally display the name of top 30 movies that are similar to the name entered by the user.

```
In [20]: # 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 = movies_data[movies_data.index==index]['title'].values[0]
    if (i<30):
        print(i, '.',title_from_index)
        i+=1
```

```
Movies suggested for you :
```

```
1 . Batman
2 . Batman Returns
3 . Batman & Robin
4 . The Dark Knight Rises
5 . Batman Begins
6 . The Dark Knight
7 . A History of Violence
8 . Superman
9 . Beetlejuice
10 . Bedazzled
11 . Mars Attacks!
12 . The Sentinel
13 . Planet of the Apes
14 . Man of Steel
15 . Suicide Squad
16 . The Mask
17 . Salton Sea
18 . Spider-Man 3
19 . The Postman Always Rings Twice
20 . Hang 'em High
21 . Spider-Man 2
22 . Dungeons & Dragons: Wrath of the Dragon God
23 . Superman Returns
24 . Jonah Hex
25 . Exorcist II: The Heretic
26 . Superman II
27 . Green Lantern
28 . Superman III
29 . Something's Gotta Give
```

Model at a Glance

Movie Recommendation System

```
In [20]: 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)

close_match = find_close_match[0]

index_of_the_movie = movies_data[movies_data.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 = True)

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

i = 1

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

Enter your favourite movie name : Avengers
Movies suggested for you :

- 1 . The Avengers
- 2 . Avengers: Age of Ultron
- 3 . Captain America: The Winter Soldier
- 4 . Captain America: Civil War
- 5 . Iron Man 2
- 6 . Thor: The Dark World
- 7 . X-Men
- 8 . The Incredible Hulk
- 9 . X-Men: Apocalypse
- 10 . Ant-Man
- 11 . Thor
- 12 . X2
- 13 . X-Men: The Last Stand
- 14 . Deadpool
- 15 . X-Men: Days of Future Past
- 16 . Captain America: The First Avenger
- 17 . The Amazing Spider-Man 2
- 18 . The Image Revolution
- 19 . Iron Man
- 20 . Iron Man 3
- 21 . Man of Steel
- 22 . The Spirit
- 23 . Superman II
- 24 . X-Men: First Class
- 25 . Guardians of the Galaxy
- 26 . Batman v Superman: Dawn of Justice
- 27 . Serenity
- 28 . Spawn
- 29 . Teenage Mutant Ninja Turtles: Out of the Shadows

RECOMMENDATIONS AND CONCLUSION

• Conclusion

In this project we have implemented and learn the following things such as- • Building a Movie Recommendation System • To find the Similarity Scores and Indexes. • Compute Distance Between Two Vectors • Cosine Similarity • To find Euclidian Distance and many more ML related concepts and techniques.

Since our project is movie recommendation system, one can develop a movie recommendation system by using either content based or collaborative filtering or combining both. In our project we have used the content-based approach. This approach is quite straight forward and easy to implement. Content-based approach have its own advantages and disadvantages. In content-based filtering is based on the user ratings, only such kind of movie will be recommended to the user.

Advantages: it is easy to design and it takes less time to compute.

Dis-advantages: the model can only make recommendations based on existing interests of the user. In other words, the model has limited ability to expand on the users' existing interests.

• Recommendations

There are plenty of way to expand on the work done in this project. Firstly, the content-based method can be expanded to include more criteria to help categorize the movies. The most obvious ideas are to add features to suggest movies with common actors, directors or writers. In addition, movies released within the same time-period could also receive a boost in likelihood for recommendation. Similarly, the movies total gross could be used to identify a user's taste in terms of whether he/she prefers large release blockbusters, or smaller indie films. However, the above ideas may lead to overfitting, given that a user's taste can be highly varied, and we only have a guarantee that 20 movies (less than 0.2%) have been reviewed by the user. In addition, we could try to develop hybrid methods that try to combine the advantages of both content-based methods and collaborative filtering into one recommendation system. Mood-detection system can also be combined with the model as sometimes user may tend to watch a particular movie if he/she is in a particular movie. For example, if one is angry or upset an action may be a better fit but since our model only recommends based on name and not considers other factors, the prediction may not satisfy the customer.

BIBLIOGRAPHY AND REFERENCES

- [1] Hirdesh Shivhare, Anshul Gupta and Shalki Sharma (2015), “Recommender system using fuzzy c-means clustering and genetic algorithm based weighted similarity measure”, IEEE International Conference on Computer, Communication and Control.
- [2] Manoj Kumar, D.K. Yadav, Ankur Singh and Vijay Kr. Gupta (2015), “A Movie Recommender System: MOVREC”, International Journal of Computer Applications (0975 – 8887) Volume 124 – No.3.
- [3] RyuRi Kim, Ye Jeong Kwak, HyeonJeong Mo, Mucheol Kim, Seungmin Rho, Ka Lok Man, Woon Kian Chong (2015), “Trustworthy Movie Recommender System with Correct Assessment and Emotion Evaluation”, Proceedings of the International MultiConference of Engineers and Computer Scientists Vol II.
- [4] Zan Wang, Xue Yu*, Nan Feng, Zhenhua Wang (2014), “An Improved Collaborative Movie Recommendation System using Computational Intelligence”, Journal of Visual Languages & Computing, Volume 25, Issue 6.
- [5] Debadrita Roy, Arnab Kundu, (2013), “Design of Movie Recommendation System by Means of Collaborative Filtering”, International Journal of Emerging Technology and Advanced Engineering, Volume 3, Issue 4.