

**Building a Movie Recommendation System with Machine Learning;**

**Introduction:**

Movies hold universal appeal, connecting people of all backgrounds. Despite this unity, our individual movie preferences remain distinct, ranging from specific genres like thrillers, romance, or sci-fi to focusing on favorite actors and directors. While it’s challenging to generalize movies everyone would enjoy, data scientists analyze behavioral patterns to identify groups of similar movie preferences in society. As data scientists, we extract valuable insights from audience behavior and movie attributes to develop the “Movie Recommendation System.”

Movie recommendation systems are not just about convenience; they represent a fascinating intersection of data science, machine learning, and user experience design. These systems can make highly personalized recommendations that keep you engaged and satisfied by analyzing vast amounts of data, such as your viewing history, ratings, and even the time you spend watching certain genres. One of the most famous websites for movie recommendations is IMDB. Let’s delve into the “Movie Recommendation System” fundamentals to unlock the magic of personalized movie suggestions using machine learning algorithms.

**Learning Outcomes**:

* Gain a comprehensive understanding of recommendation algorithms, their purpose, and how they function to predict user preferences.
* Learn why recommendation engines enhance user experience, increase engagement, and drive revenue in various platforms.
* Explore different recommendation engine types, specifically content-based and collaborative filtering, including their methods, advantages, and limitations.
* Acquire practical knowledge of implementing a movie recommendation system using Python.
* Understand the application of machine learning models and machine learning algorithms to build a recommendation engine.

**What is a Recommendation System**?

A Recommendation System is a filtration program whose prime goal is to predict a user’s “rating” or “preference” toward a domain-specific item or item. In our case, this domain-specific item is a movie. Therefore, the main focus of our recommendation system is to filter and predict only those movies that a user would prefer, given some data about the user.

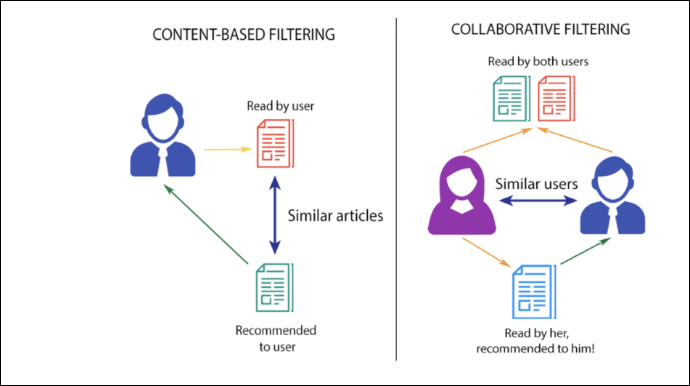
**Why Recommendation Systems?**

Recommendation Systems are essential for several reasons:

1. Recommendation Systems offer personalized suggestions based on user preferences and user ratings, ensuring that they discover content and products that are relevant and interesting to them.
2. By providing tailored recommendations, users are more likely to engage with the platform, increasing user satisfaction and retention.
3. E-commerce platforms like Amazon use recommendation engines to promote products, leading to higher sales and revenue as users discover and purchase items they might not have considered.
4. In today’s digital landscape, recommendation systems help users navigate the overwhelming amount of available content, making it easier to find a particular movie or series they seek or in any language they want, like English or Korean.
5. Recommendation algorithms expose users to new and diverse content, expanding their horizons and introducing them to items they might have overlooked.
6. Relying on past behavior and preferences, recommendation systems help users make informed decisions about complex and subjective choices such as movies, music, or books.

**Types of Recommendation Systems**

There are 2 types of recommendation systems- Content-based filtering and Collaborative Filtering. Let’s discuss them now**.**



**Content-based Filtering**

Content-based filtering is a recommendation strategy that suggests items similar to those a user has previously liked. It calculates similarity (often using cosine similarity) between the user’s preferences and item attributes, such as lead actors, directors, and genres. For example, if a user enjoys ‘The Prestige,’ the system recommends movies with Christian Bale, the ‘Thriller’ genre, or films by Christopher Nolan**.**

However, content-based filtering has drawbacks. It limits exposure to different products, preventing users from exploring a variety of items. This can hinder business expansion as users might not try out new types of products**.**

**Also Read:** Beginners Guide to Content Based Recommender System

**Collaborative Filtering**

**Collaborative filtering \_** is a recommendation strategy that considers the user’s behavior and compares it with other users in the database. It uses the history of all users to influence the recommendation algorithm. Unlike a content-based recommender system, a collaborative filtering recommender relies on multiple users’ interactions with items to generate suggestions. It doesn’t solely depend on one user’s data for modeling. There are various approaches to implementing collaborative filtering, but the fundamental concept is the collective influence of multiple users on the recommendation outcome**.**

**There are 2 types of collaborative filtering algorithms:**

User-based Collaborative filtering

User-based collaborative filtering aims to recommend items to a user (A) based on the preferences of similar users in the database. It involves creating a matrix of items rated/liked/clicked by each user, computing similarity scores between users, and then suggesting items that user A hasn’t encountered yet but similar users have liked**.**

For example, if user A likes particular movies like Batman Begins, Justice League, and The Avengers and user B like Thor, they share similar interests in the superhero genre. Therefore, it is highly likely that user A would enjoy a popular movie like Thor and user B would like The Avengers, one with the highest movie ratings.

**Disadvantages:**

User-based collaborative filtering has several disadvantages:

* **Fickle User Preferences**: User preferences can change over time, making initial similarity patterns between users outdated. This can result in inaccurate recommendations as users’ tastes evolve and they start watching new movies.
* **Large Matrices**: As the number of users is typically much larger than the number of items, maintaining large matrices becomes challenging and resource-intensive. Regular recomputation is required to keep the data up-to-date.
* **Vulnerability to Shilling Attacks**: Shilling attacks involve creating fake user profiles with biased preference patterns to manipulate the recommendation system. User-based collaborative filtering is susceptible to such attacks, potentially leading to biased and manipulated recommendations

**Item-based Collaborative Filtering**

Item-based Collaborative Filtering focuses on finding similar movies instead of similar users to recommend to user ‘A’ based on their past preferences. It identifies pairs of movies rated/liked by the same users, measures their similarity across all users who rated both, and then suggests similar films based on the similarity scores**.**

For example, when comparing movies ‘A’ and ‘B,’ we analyze the ratings from users who rated both movies. If these ratings show high similarity, it indicates that ‘A’ and ‘B’ are similar movies. Thus, if someone liked ‘A’, they should be recommended ‘B’, and vice versa**.**

**Advantages over User-based Collaborative**

Filtering includes stable movie preferences, as popular movies do not change people’s tastes. Maintaining and computing matrices is more accessible as there are usually fewer items than users. Shilling attacks are also more challenging since items cannot be faked, making this approach more robust**.**

**Movie Recommendation System Code**

In this implementation, when the user searches for a movie, we recommend the top 10 similar movies using our movie recommendation system. We will be using an item-based collaborative filtering algorithm for our purpose. The training set used in this demonstration is the movie lens-small dataset.

**Getting the Data Up and Running:**

First, we need to import libraries which we’ll be using in our movie recommendation system. Also, we’ll import the dataset by adding the path of the CSV files. Then we will have a look at the movie’s dataset:

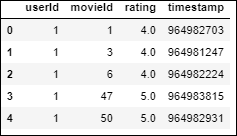
**The movie dataset has the following:**

1. movieId: Once the recommendation is made, we get a list of all similar movieIds and the title for each movie from this dataset.
2. Genres: Which are not required for this filtering approach.

**Code:**



**Output**

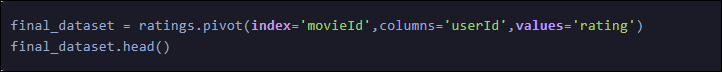


Ratings dataset has-

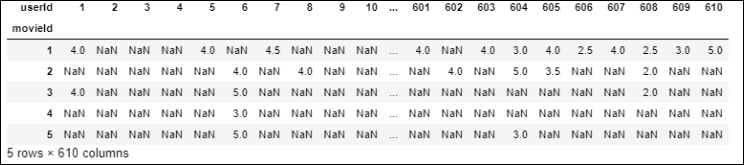
* userId: Unique for each user.
* movieId: We take the movie title from the movie dataset using this feature.
* Rating: Each user gives ratings for all the films. Using this, we are going to predict the top 10 similar films.

Here, we can see that userId one has **watched** movieId 1 and 3 and rated them 4.0 but has not rated movieId two. This interpretation is more challenging to extract from this dataframe. Therefore, to make things easier to understand and work with, we will make a new dataframe where each column represents each unique user, and each row represents each unique movie**.**

**Code**



**Output**



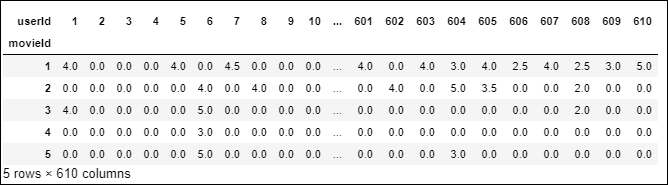
Now, it’s much easier to interpret that userId 1 has rated movieId 1& 3 4.0 but has not rated movieId 3,4,5 at all (therefore, they are represented as NaN ), and therefore, their rating data is missing.

Let’s fix this and impute NaN with 0 to make the algorithm understandable and the data more eye-soothing.

**Code**



**Output**



**Removing Noise from the Data**

In the real world, ratings are very sparse, and data points are mainly collected from Noisetop-rated movies and highly engaged users. We wouldn’t want movies rated by a few users because it’s **not credible** enough. Similarly, users who have rated only a **handful of films** should also not be considered.

Considering all that and some trial-and-error experimentation, we will reduce the noise by adding filters to the final dataset.

* To qualify for a movie, a minimum of **10** users should hoist a film. The user should have voted for at least 50 movies to qualify users.

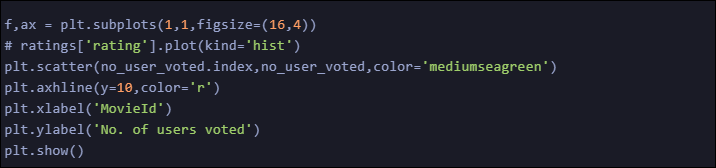
**Let’s Visualize How These Filters Look Like**

Aggregating the number of users who voted and the number of movies who voted.

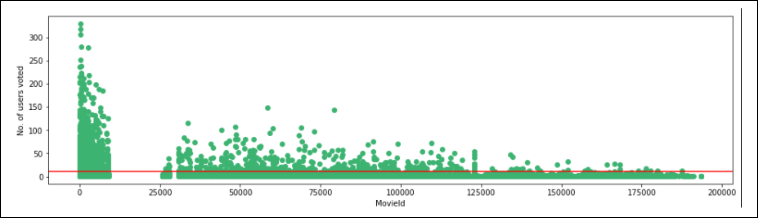
**Code**



Let’s visualize the number of users who voted with our threshold of 10.



**Output**

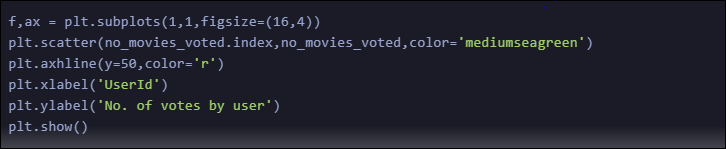


Making the necessary modifications as per the threshold set.

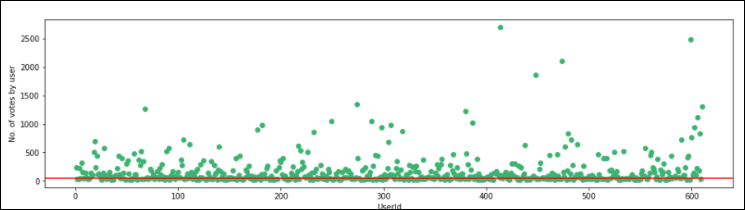
**Code**



Let’s visualize the number of votes by each user with our threshold of 50.

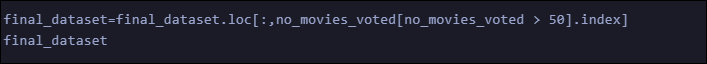


**Output**

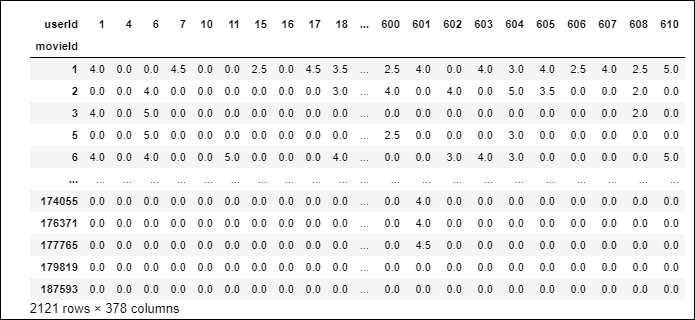


Making the necessary modifications as per the threshold set.

**Code**



**Output**

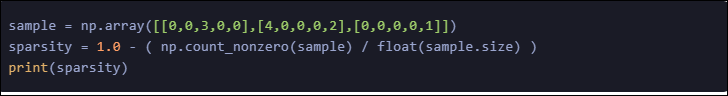


**Removing Sparsity**

Our final dataset has dimensions of **2121 \* 378**, and most of the values are sparse. We are using only a small dataset, but for the original large dataset of movie lenses, which has more than **100,000** features, our system may run out of computational resources when fed to the model. We use the csr\_matrix function from the scipy library to reduce the sparsity.

I’ll give an example of how it works :

**Code**



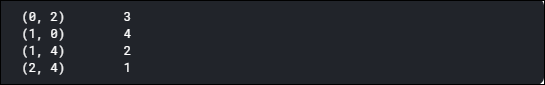
**Output**



**Code**

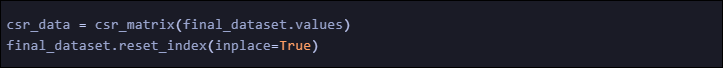


**Output**



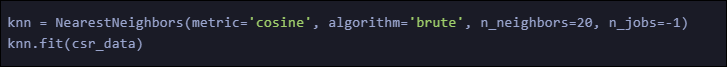
As you can see, there is no sparse value in the csr\_sample, and values are assigned as rows and column indexes. For the 0th row and 2nd column, the value is 3.

Applying the csr\_matrix method to the dataset:



Making the Movie Recommendation System

We will be using the KN algorithm to compute similarity with metric which is very fast and more preferable than peoarson Conffient.



Making the Recommendation Function

The working principle is very simple. We first check if the movie name input is in the database. If it is, we use our recommendation system to find similar movies, sort them based on their similarity distance, and output only the top 10 movies with their distances from the input movie.

def get\_movie\_recommendation(movie\_name):

n\_movies\_to\_reccomend = 10

movie\_list = movies[movies['title'].str.contains(movie\_name)]

if len(movie\_list):

movie\_idx= movie\_list.iloc[0]['movieId']

movie\_idx = final\_dataset[final\_dataset['movieId'] == movie\_idx].index[0]

distances , indices = knn.kneighbors(csr\_data[movie\_idx],n\_neighbors=n\_movies\_to\_reccomend+1)

rec\_movie\_indices = sorted(list(zip(indices.squeeze().tolist(),distances.squeeze().tolist())),key=lambda x: x[1])[:0:-1]

recommend\_frame = []

for val in rec\_movie\_indices:

movie\_idx = final\_dataset.iloc[val[0]]['movieId']

idx = movies[movies['movieId'] == movie\_idx].index recommend\_frame.append({'Title':movies.iloc[idx]['title'].values[0],'Distance':val[1]})

df = pd.DataFrame(recommend\_frame,index=range(1,n\_movies\_to\_reccomend+1))

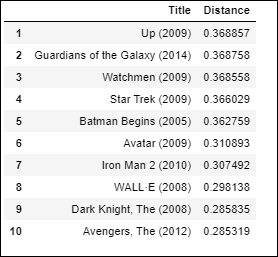
return df

else:

return "No movies found. Please check your input"

**Finally, Let’s Recommend Some Movies!**

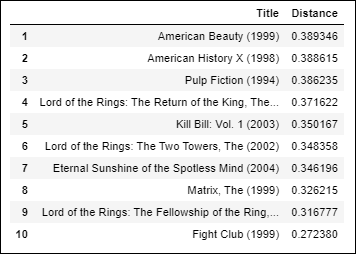




I personally think the results are pretty good. All the movies at the top are superhero or animation movies, which are ideal for kids, as is the input movie “Iron Man”.

Let’s try another one:





All the movies in the top 10 are serious and mindful, just like Memento itself; therefore, I think the result is also good in this case.

Our model works quite well- a movie recommendation system based on user behavior. Hence, we conclude our collaborative filtering here. You can get the complete implementation notebook [here.](https://github.com/So-ham/Movie-Recommendation-System)

Also Read:[*Movies Recommendation System using Python*](https://www.analyticsvidhya.com/blog/2022/08/movies-recommendation-system-using-python/)

Conclusion

In conclusion, building a movie recommendation system with machine learning significantly enhances user experience by providing personalized movie suggestions based on individual preferences. Utilizing algorithms such as collaborative filtering and content-based filtering, these systems leverage user behavior data to deliver accurate and relevant recommendations. With the integration of [Python](https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-learn-data-science-python-scratch-2/) and robust datasets like MovieLens, developers can create sophisticated recommendation engines akin to Netflix’s or Amazon Prime’s, using techniques like cosine similarity and matrix factorization. Embracing these technologies enriches user engagement and drives retention and satisfaction, showcasing the transformative power of data science in personalized content delivery.

Key Takeaways:

* Content-based filtering recommends items similar to those a user has liked before, while collaborative filtering suggests items based on similarities between users’ behaviors and preferences.
* Learn about the challenges such as fickle user preferences, large matrices, and vulnerability to shilling attacks in user-based collaborative filtering and how item-based collaborative filtering can mitigate some of these issues.
* Understand the importance of data preparation, including removing noise and handling sparsity, to ensure the recommendation system functions effectively and efficiently.
* Gain insights into the practical steps and Python code required to build and test a movie recommendation system, including data importing, matrix manipulation, and model training.
* Recognize the real-world impact of recommendation systems, such as those used by Netflix, and how they improve user satisfaction by providing personalized and relevant content suggestions.

Join the [Blackbelt Plus program](https://blackbelt.analyticsvidhya.com/plus) to enhance machine-learning skills and build cutting-edge movie recommendation systems. Propel your career in data science and make a difference in the world of personalized movie recommendations. Enroll in the program today and step into the future of movie suggestions!