**Movie recommendation system**

**1. Introduction**

**What is a “Content based movie recommendation system”?**

In this machine learning project, we build a recommendation system from the ground up to suggest movies to the user based on his/her preferences. Recommendation systems are computer programs that suggest recommendations to users depending on a variety of criteria.

Our movie recommendation engine works by suggesting movies to the user based on the metadata information. The similarity between the movies is calculated and then used to make recommendations.

**Why?**

The purpose of a recommendation system basically is to search for content that would be interesting to an individual. Moreover, it involves a number of factors to create personalized lists of useful and interesting content specific to each user/individual. Recommender systems are beneficial to both service providers and users

* They help the user find items of their interest
* Helps the item provider to deliver their items to the right user
  + To identify the most relevant products for each user
  + Showcase personalized content to each user
  + Suggest top offers and discounts to the right user
* Websites can improve user-engagement
* It increases revenues for business through increased consumption

**How?**

Our recommendation system takes the title of the movie as input and gives a list of movies as recommendations. The engine suggests movies to the user based on the metadata information like cast, director, genre, etc.

**2. Literature Review**

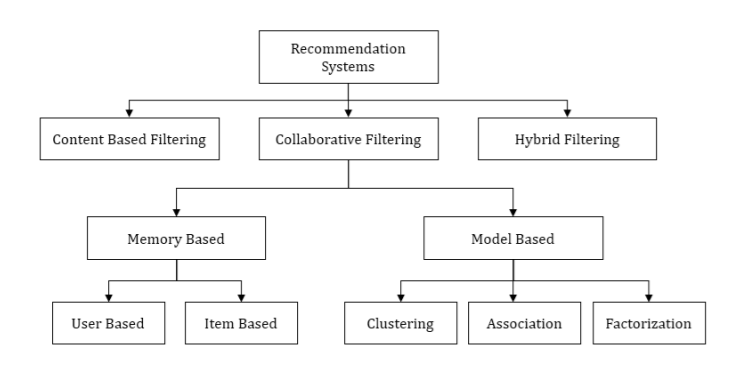
There are three techniques of recommendation system: Collaborative Filtering, Content-Based Filtering, and Hybrid Filtering.

In the Content Based recommender system, the user provides data either explicitly (rating) or implicitly (by clicking on a link). The system captures this data and generates a user profile for every user. By making use of the user profile, the recommendation is generated. In content based filtering, the recommendation is given by only watching a single user’s profile. The system tries to recommend items similar to that item based on users’ past activity.

Unlike content based, collaborative filtering finds those users whose likings are similar to a given user. It then recommends an item or any product, by considering that the given user will also like the item which other Users like because their tastes are similar. Both these techniques have their own strength and weakness so to overcome this, a hybrid technique came into the picture, which is a combination of both these techniques.

Hybrid filtering can be used in various types. We can use content based filtering first and then pass those results to the collaborative recommender (and vice-versa) or by integrating both the filters into one model to generate the result. These kinds of modifications are also used to cope up with a cold start, data sparsity, and scalability problems.

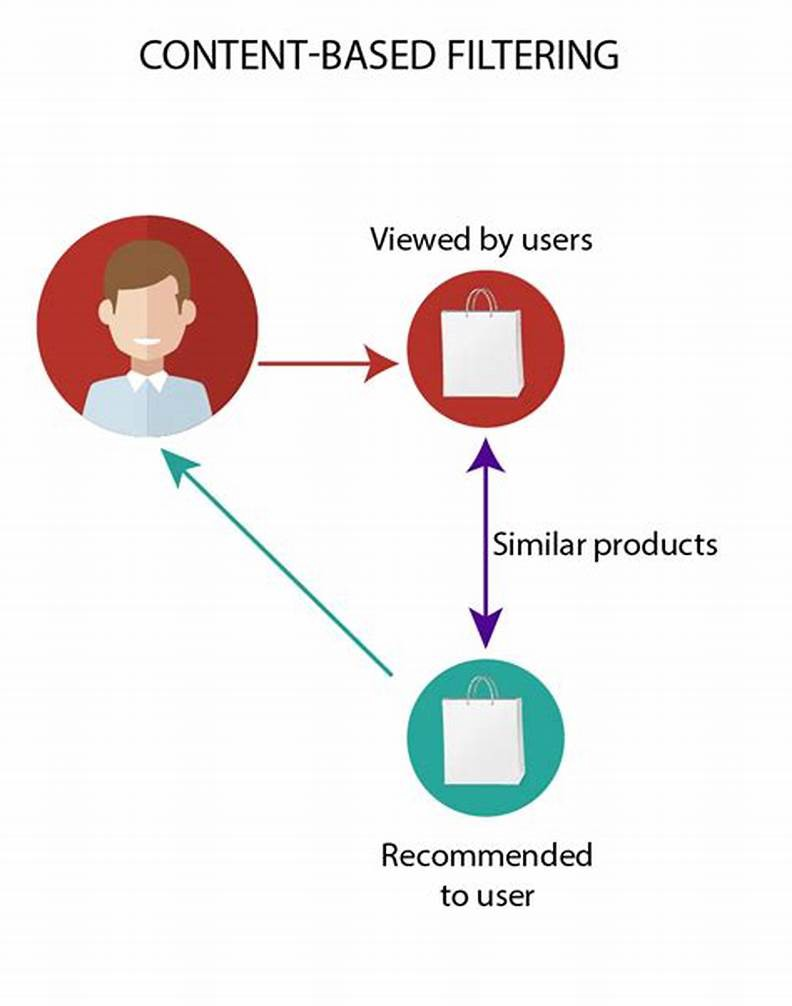
The taxonomy of the Recommender System is depicted in figure 1.

****

**Figure 1**

**Content Based Filtering:**

Content-Based Filtering is also known as cognitive filtering. This filtering recommends items to the user based on his past experience. For example, if a user likes only action movies then the system predicts only action movies similar to it. The broader explanation could suppose the user likes only politics-related content so the system suggests the websites, blogs, or the news similar to that content. Unlike collaborative filtering, content-based filtering does not face new user problems. It does not have other user interaction in it. It only deals with a particular user’s interest. Content-based filtering first checks the user preference and then suggests the movies or any other product to him. It only focuses on a single user’s ideas, thoughts and gives predictions based on his interest. So if we talk about movies, then the content-based filtering technique checks the rating given by the user. The approach checks which movies are preferred by the user by checking the genre categories in the user profile. After analyzing the user profile, the technique recommends movies to the user according to his taste. Figure 2 shows us how Content-Based Filtering works.

****

**Figure 2**

As shown in figure 2, content-based filtering the whole process is shown by giving an example of Geometric Shapes. Here in the figure, first, an Item Profile is developed based on the liking of the user. Here the user likes a circle and a triangle of blue color. Now based on the item profile, the user profile is built. This user profile is generated by getting the data from the item profile. As we can see in the item profile, the user likes a circle and a triangle of blue color so the user profile is also having a circle, a triangle, and blue color. Now we will match this user profile with the collection of different shapes available. In the shapes collection, we have a pentagon of blue color, then a circle of yellow color, and two squares of yellow color. So here the system finds which of these shapes matches the user profile. So here the blue color pentagon matches the user’s interest.

**Pros**

* There is no requirement about other users’ data to make recommendations. This makes it easier to scale to a large number of users.
* It is easy for a content-based approach to recommend new items. It provides recommendations to the user with unique tastes.
* It also provides a content feature that helps us to explain the reason for an item recommended. The model can capture the specific interests of a user, and can recommend niche items that very few other users are interested in.

**Cons**

* To find any particular feature( movies) of any specific genre sometimes becomes a problem.
* Generally, it is referred to as an overspecialization problem. The user is never recommended anything outside the user profile.
* It is easy to miss recommending an item to a user as there is not enough information about that item. Since the feature representation of the items are hand-engineered to some extent, this technique requires a lot of domain knowledge. Therefore, the model can only be as good as the hand-engineered features.

**Cosine similarity:**

Cosine similarity is the measure of similarity between two non-zero vectors in the inner product space. It measures the angle between these two vectors. A cosine of two non-zero vectors can be calculated using dot products of these two vectors:

u.v = ||u|| . ||v|| . cosθ

Cosine similarity is particularly used in positive space where the result is efficiently bounded in [0, 1]. Thus for two given vectors u and v, the cosine similarity, cosθ can be computed as the combination of dot product and magnitude of the vectors:

sim(u, v) cosine = cosθ = P u v ||u|| . ||v||

It is used to measure how similar the documents are irrespective of their size. The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance (due to the size of the document), chances are they may still be oriented closer together. The smaller the angle, the higher the cosine similarity.

**3. Data Description**

Data source:  **TMDB 5000 Movie Dataset**: [Kaggle](https://www.kaggle.com/tmdb/tmdb-movie-metadata)

* The dataset contains two CSV files named movies and credits with 4803 rows.
* In total there are 23 features:

|  |  |  |
| --- | --- | --- |
| movie\_id  title  cast  crew  budget  genres  homepage  id | keywords  popularity  production\_companies  production\_countries  release\_date  overview  original\_language  original\_title | revenue  spoken\_languages  Status  tagline  title  vote\_average  vote\_count  runtime |

* There are 10 numerical features and 13 categorical features.
* The features that are useful to recommend movies are :

movie\_id, genres, title, overview, cast, crew, keywords

**4. Method Description**

**Preprocessing of data:**

The dataset contains two CSV files, credits, and movies. The credits file contains all the metadata information about the movie and the movie file contains the information like name and id of the movie, budget, languages in the movie that has been released, etc. First, Merge both the data frames on the column ‘title’. From all the available features we only need movie\_id, genres, title, overview, cast, crew, keywords, so trim the data frame to these required features.

Add a column named ‘tags’ which contains strings that are formed by concatenating the metadata information.



After processing, the data frame looks like this:

**Building the Movie Recommendation system:**

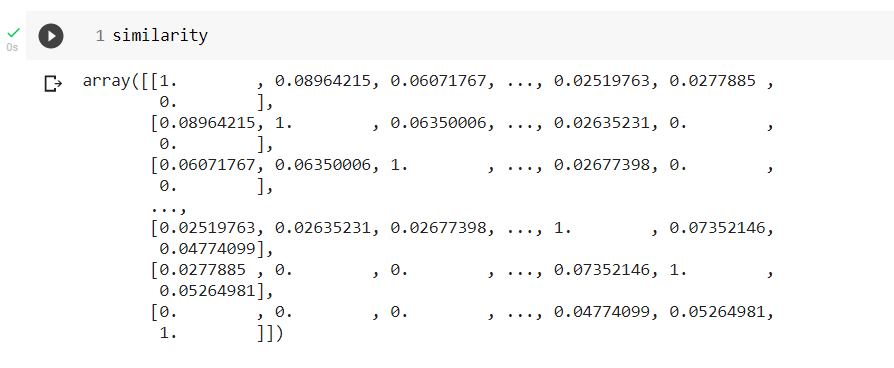
Our movie recommendation engine works by suggesting movies to the user based on the metadata information. The similarity between the movies is calculated and then used to make recommendations. For that, our text data should be preprocessed and converted into a vectorizer using the CountVectorizer. As the name suggests, CountVectorizer counts the frequency of each word and outputs a 2D vector containing frequencies.

We don’t take into account the words like a, an, the (these are called “stopwords”) because these words are usually present in higher amounts in the text and don’t make any sense.

There exist several similarity score functions such as cosine similarity, Pearson correlation coefficient, etc. Here, we use the cosine similarity score as this is just the dot product of the vector output by the CountVectorizer.

Create a Similarity matrix by applying the cosine similarity function to the output of the CountVectorizer function. Similarity Matrix contains similarity scores of all the movies with respect to the movie indexed to the particular row.

The similarity matrix looks like this:



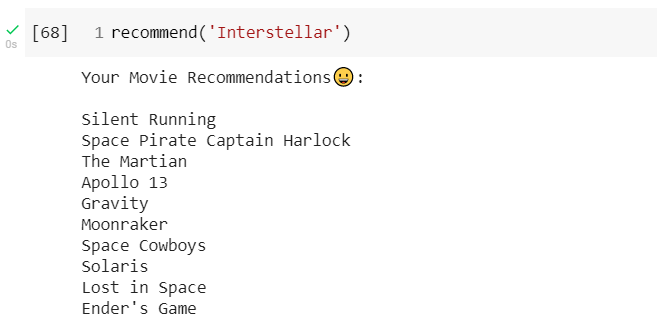
Let’s have a brief look at it: all the numbers on the diagonal are 1 because, of course, every movie is identical to itself. The matrix is also symmetrical because the similarity between A and B is the same as the similarity between B and A.

The recommend() function takes the title of the movie and the similarity function as input. It follows the below steps to make recommendations.

* Get the index of the movie using the title.
* Get the list of similarity scores of the movies concerning all the movies.
* Enumerate them (create tuples) with the first element being the index and the second element being the cosine similarity score.
* Sort the list of tuples in descending order based on the similarity score.
* Get the list of the indices of the top movies from the above-sorted list. Exclude the first element because it is the title itself.
* Map those indices to their respective titles and return the movies list.

**5. Result**

The model recommends a list of movies to the user based on the users’ favorite movie.



[Link to code of the project](https://colab.research.google.com/drive/1zBnoJpB0c9_PUrGHjE18fU8g9FZjKlrM?usp=sharing)

[Link to the website](https://recommenderapp-dm.herokuapp.com/)