The recommender system we wanted was the kind to recommend a list of movies to a user based on their movie taste.

Out of two methods that we could use for developing our recommender system we chose the second one , and used content-based filtering.

The idea behind this method is that if someone likes an specific movie ( or anything in general) , there’s a really high possibility that they’d like movies that are similar to that one . This method made more sense to us than the others.

There were several things we should have done :

1. to know what are each user’s favorite movies
2. what are the similar characteristics of movies
3. And based on those similarities , suggest a bunch of movies to each user

For achieving our goals we should have done three steps :

1 – Data Analysis

2 – Building the actual system

3 – test to see if it works the way it should

We also installed the libraries we needed and loaded the data files (movies\_metadata , credits ) as Step 0 .

**Step 1 : Data Analysis**

We started with taking a quick look at our data frames to see what we really have in our hands to work with .

For a better understanding we used “df.head()” command to see just a few first rows of our data frame.

After that , it was the time to merge two data frames that we had . At first we tried to use the pd.merge function , using the common column “id” , but then we got stuck in an error because apparently the said column had different types in each data frame . In one of them it was and an int64 and then in another it was an object .

As a result of that , we decided to concatenate these two data frames instead . So , in order to being able to delete the duplicated columns “ id “ , we first changed the name of this column to the “movie\_id “in the “credits” data frame . After concatenating , we deleted the “movie\_id” column to omit the duplication , using “df.drop()”.

Before moving to the next stage, we realized that for this method we’re using we also need the info in “keywords” file . So we loaded that one as well and concatenate it with those two( using the very same method explained above).

**Step 2 : Build our recommender system**

In this step we’re gonna use the output of the previous step , which is a data frame that contains all of the info we need .

First convert the data of columns : cast , keywords , genres ; into a form that can be used later , using “literal\_eval” .

The reason that we’re only doing this process for these three columns is that they’re chosen to be representative of each movie’s content and characteristic.

This converting stage continues with applying the lowercase function and space-eliminating , just for easier use . For which we have defined a “data-cleaning” function that is applied on each instance we have using a loop.

In this stage we define a “soup-creating” function to be used later . This function gets a dictionary as an input and gives a string as an output . The output string contains the concatenation of the values of selected columns (cast , keywords , genres).

In the next part , we create a MinHashLSHForest object with 128 permutations for each row of our data frame . In other words we create a “soup” object for each row using the function we have and then create a MinHash object . After that , we iterate over each word of our “soup” object and update the MinHash with the encoded word . In the end , the MinHash and its index are added to our MinHashLSHForest object .

The aim of this whole process is to make the search of similar items in our data frame more efficient . That’s the reason why we’re using MinHashLSHForest data structure , so we can easily find movies similar to each other based on their features .

**Step 3 : test and get recommendation**

In this step we first get inputs from user and then generate MinHash objects for each of the given movies . Then we search for the 10 similar items in our data frame . We also sort these similar items based on their occurrences in the descending order . These processes are followed by getting the top 10 of recommended movies and then the final list of recommendation is printed for the user .

How does it work for each user ? The user gives their id to the system , system searches for the user’s favorite movies list (movies that the user has rated at least 4). After finding the list the system searches for movies similar to each of the movies that user likes , and finally prints the result .

Now our brief report has come to an end , thank you for your time and attention .

**The End**