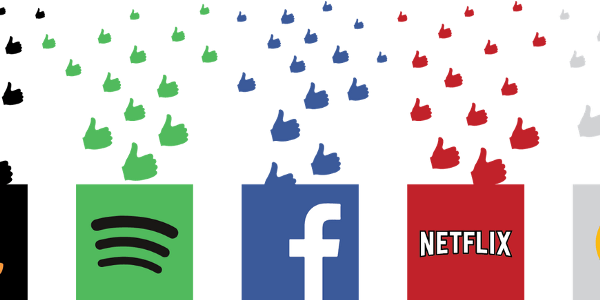
### **Building a Collaborative-Filtering Recommendation system for recommending classes to students**

**In this article, we will build a collaborative-filtering recommendation system that will use the ratings from similar users to recommend them, various classes. We will also build a web application that will show us the recommendations given a user ID.**

A Recommendation System, in simpler terms, is basically a filtering system that aims to predict a rating or preference that a user would give to a particular item, which in our case are classes.

There are various types of algorithms used in building Recommendation systems such as Rank Based, Content-Based, Collaborative Filtering, Matrix Factorization, and many more. Content-Based and Collaborative Filtering are the most common ones among these. In this article, we will look into collaborative filtering to recommend classes to the students.

*If you want to jump straight to code, check out the GitHub repository for this code here:* [*course-recommender-app*](https://github.com/fenago/blog/tree/main/Collaborative-Filtering-Recommender)

# Exploratory Data Analysis

First, let's create a dummy dataset. We will read course names from a text file and assign a random rating to it by every user.

Text

Description automatically generated

Table

Description automatically generated

Let's first start by conducting EDA on our data to get a better understanding of what we are dealing with.

Here we have two separate data frames. The first data frame contains the rating a user gave to a class. The rating varies from 1–10 and 0 shows that the user hasn't attended that class. We have 30 users and 175 classes, thus a total of 5250 rows in the data frame.

Let's group the data by ratings and see the average ratings that the users give. We will use the following data for that.



Chart, bar chart

Description automatically generated

# User-Based Collaborative Filtering

## There are different types of collaborative filtering algorithms, mainly item-based and user-based. In this article, we are going to be using user-based collaborative filtering.

## User-Based Collaborative filtering basically utilizes the assumption that generally, people with similar characteristics share similar interests. In our case, it would mean that if two students have highly rated similar classes in the past, we can recommend them classes that are highly rated by one student to the other as it makes sense that they will highly rate those classes in the future as well.

## Creating User Item Matrix

For this, we will first create a user-item matrix (in our case student-class) where the rows and columns will be the student and class ids respectively. If a student has given a class a rating, we will place **‘the rating’** there and **‘0’** if they have not.

Text

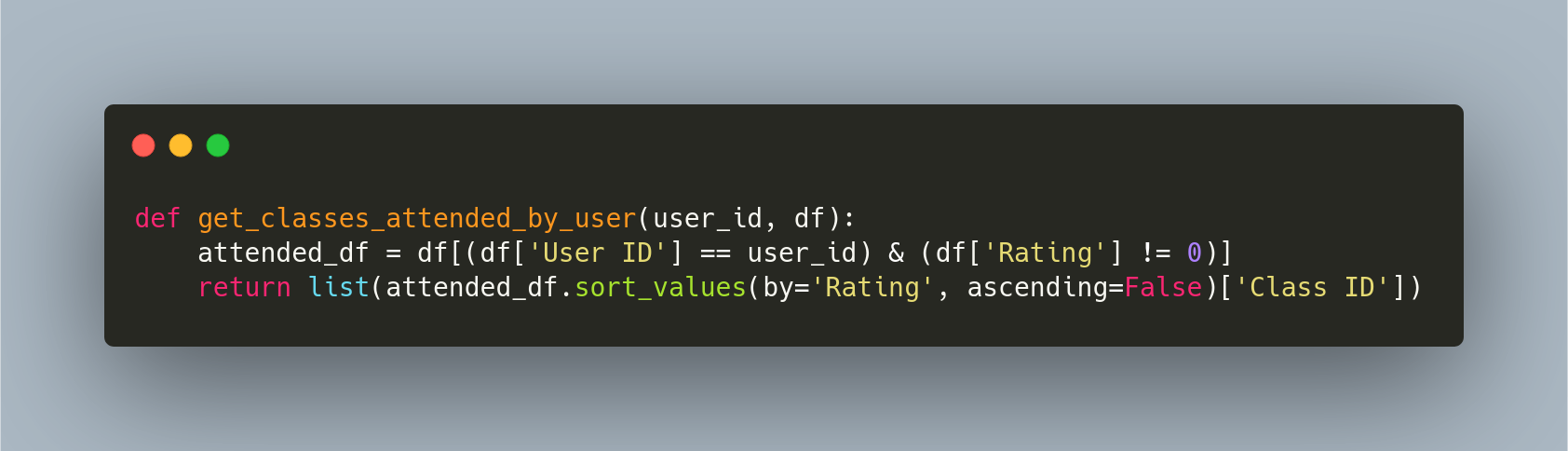
Description automatically generated

A picture containing text

Description automatically generated

## Get Classes attended by Users

Now, we will write a function that will return the classes attended by a user and sort them by the ratings. We do this because after getting similar users, we will recommend the classes attended by the most similar user with the highest rating to that particular user.



## Getting Similar Users

Now we will work on getting users similar to a particular user. This is done by comparing the ratings given by the two users to a class attended by both of them. We will calculate the total ratings and return the users with the highest total ratings.



There’s a lot going on in this function. Let’s break it down a bit:

1. It starts with computing the dot product of the user with user\_id with all other users. A higher dot product means that the users gave similar high ratings to the same classes.
2. user\_interactions contain data regarding the number of classes each user has attended.
3. We merge the similar\_users with user\_interactions and sort it first by the similarity and if it's the same, then use the one with more interactions.

This is what we get when we run this function for user\_id 1:

Graphical user interface, application

Description automatically generated

## Table Description automatically generated with low confidence

## Providing Recommendations

Finally, we will combine all of this to recommend classes to users.

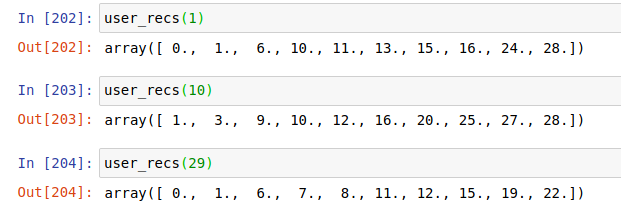
Text

Description automatically generated

The first function just returns the course names given a list of course ids. The user\_recs function executes as follows:

1. We get the most similar users based on ratings on classes using the function get\_similar\_users
2. We get the classes attended by the current user.
3. We iterate through the ids of similar users and append the classes not yet attended by the current user to the total recommendations.
4. If we reach the number of recommendations we want, we exit the loop

Let’s test this function:Graphical user interface, text

Description automatically generated

As we can see, the function successfully recommends the top 10 classes the user is most likely to give a higher rating to.

# Web Application:

Now, we will build a simple web application that will accept the user ID and the number of recommendations they want. And recommend those to them. For this purpose, we will use Flask.

The Flask code is simple and quite straight-forward. Let’s get right to it.



Let’s break it down:

1. First, we import the necessary modules and initialize our Flask app.
2. In a Flask app (or any web application for that matter), you have to define the routes. In our case, it’s just the home page route ‘/’. We will define two methods, GET method (when the user wants to see the form) and the POST method (when the user enters in the form and wants to see the recommendations)
3. When the method posts, we first extract the user ID and the number of recommendations from the form using its ids (userID and nRecs respectively).
4. Then we call our method for recommending that we built above with the user id and the number of recommendations
5. Finally, we render the page by passing it to the HTML file and the recommendations.
6. In the case of the GET method, we just return the HTML file which displays the form.
7. Lastly, we run the app on our localhost.

Now, we will look into the index.html file defined above.

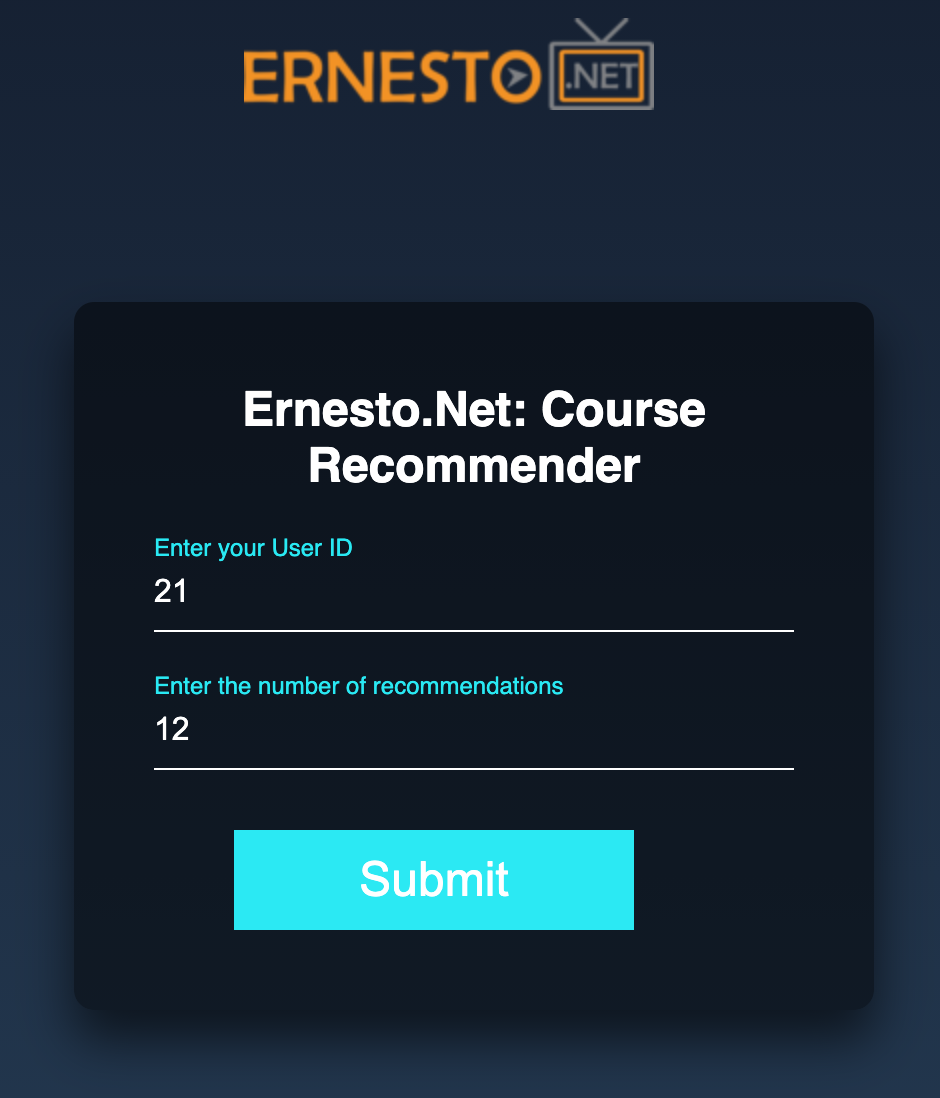
Text

Description automatically generated

The file executes as follows:

1. It starts by checking if there are any recommendations and shows those recommendations in a list
2. If there are no recommendations, it shows the form. Note the ids are similar to what we used above to get these values.

Make sure you create this HTML file inside a templates folder, otherwise Flask doesn’t recognize it. Let’s test the application and see the output. The application will be live on localhost with port 80

Text

Description automatically generated

We successfully built an application that shows the recommendations to a user.

# Docker:

To make our app more useful and easily accessible to people, we would containerize it using Docker. We will create a Dockerfile in the same folder as the index.py as shown in the image below:

Graphical user interface, text

Description automatically generated

Let’s now examine the Dockerfile.Text, chat or text message

Description automatically generated

The commands are executed one by one here.

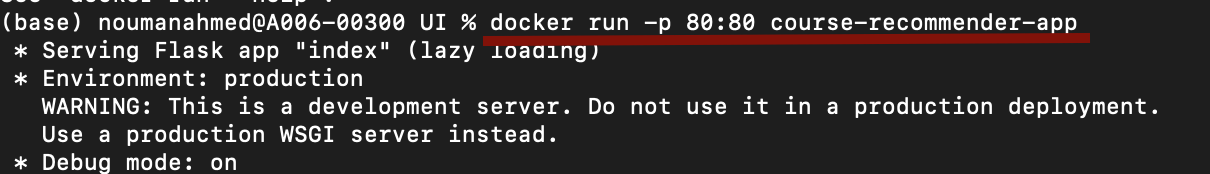
1. First, we install python and create a new working directory and copy all the contents from the current directory to this one.
2. Then, we run the commands to upgrade pip and install all the necessary libraries for this.
3. Finally, we run the index.py file which runs the server of our app.

To make an image, first, go inside the folder where your Dockerfile exists, then run the following command:

Text, chat or text message

Description automatically generated

This will take some time to execute. After it’s done you can run your app as follows:



-p tag defines the port that you want the application to run on. You can view the image running on the docker desktop and also view the application in the browser.

I have also pushed this docker image to docker hub so you can download and play around with this application here: [course-recommender-app-docker](https://hub.docker.com/r/fenago/course-recommender-app)

# Conclusion

In this article, we looked into collaborative-filtering based recommendation systems:

1. We saw the importance of creating a user-item matrix and using the dot product of ratings as a means to calculate the similarity between different users
2. We built a recommendation system using the information we had regarding the rating users give to various classes they had attended
3. We calculated the similar users and recommended the highest-rated classes not yet attended by the current user.
4. We created a Flask web that uses this recommendation system
5. Finally, we built a docker image so that anyone can use this application.