Microsoft Engage 2022

Project Description

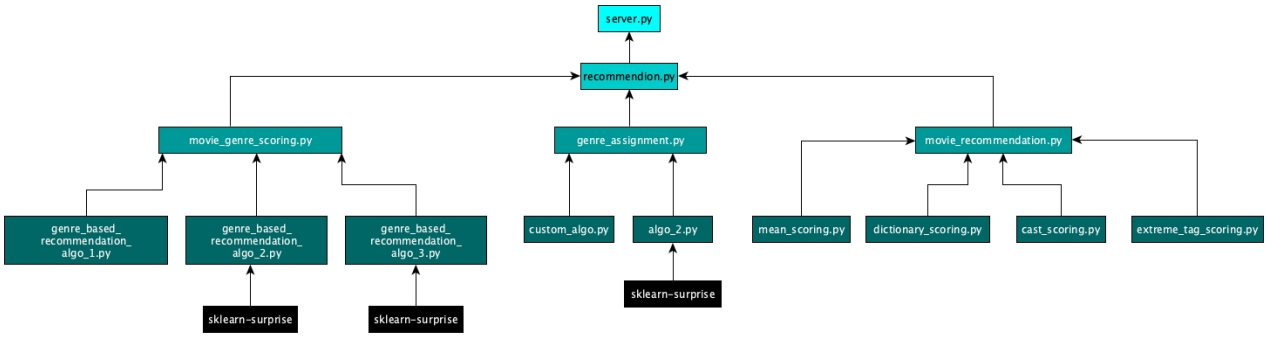
Recommendation Engine

Table of contents:

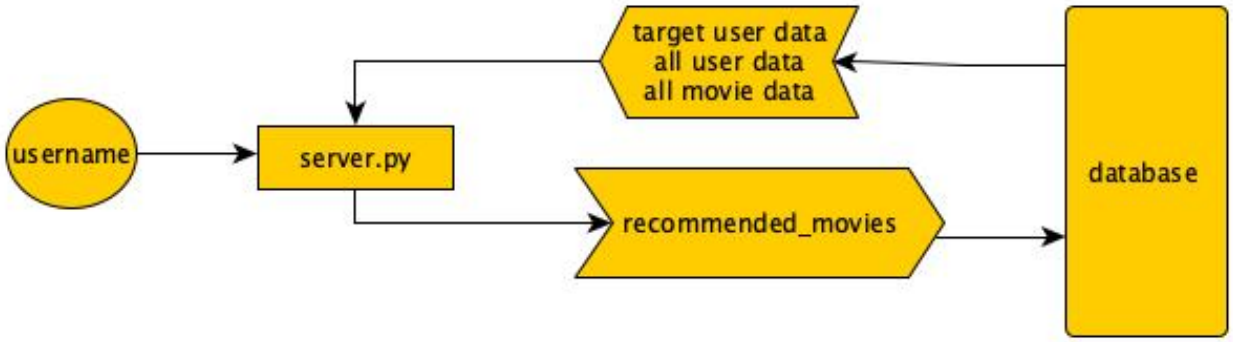
Basics of the algorithm

The algorithm is written in python because in python there are multiple libraries and supporting libraries for machine learning. Here we have used sklearn-surpirse library for the matrix factorisation based recommendation engine
And pandas for dataframe handling. No external dataset is needed as we are maintaining the datasets for our users as well as movies on own own hosted database on mongoDB atlas.

So for collaborative as well as hybrid filtering of the movies we can access our own dataset. We are using pymongo library to connect our python to our hosted database.
So the main file that handles all the other functions is server.py . Given below is the diagram of file interaction of the whole system.



Server.py takes a username as command line argument and accesses the database using pymongo library and gets the details such as liked genre and liked cast etc and details of each movie and each other person available in the database and processes the recommended movies for that user and stores it back in the database.

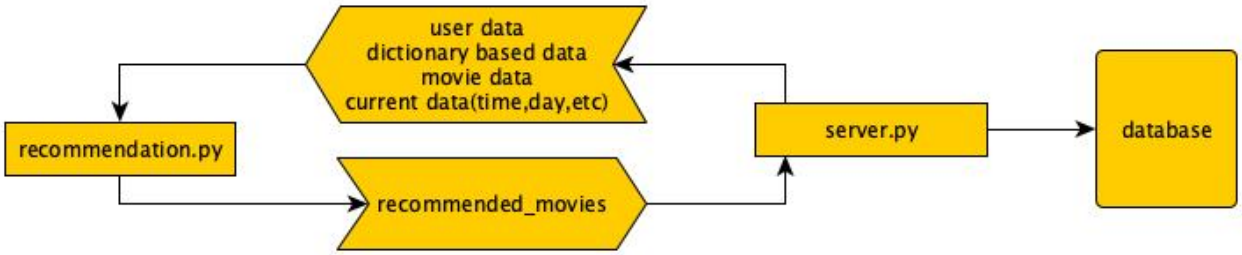


Then we come to recommendation.py it first takes the details such as other people top preferences genre/cast, movie details, current details (day, weather, time, etc). For the simulation purposes we have kept the details for current details static i.e. we send the same data every time but the effect can be seen by changing the dictionary present in the server.py containing such details.
It also takes the weather, time and day preference of the movies for other people because there is a chance that if for example most people like xyz genre when the weather is abc so if the weather today is abc there is a high possibility that the target user will also like movies with genre xyz and same goes for rest all other dictionary format data (day-genre relation eg: sunday:action,monday:comedy,etc and time-genre relation eg: nigh:romantic, evening:action, etc).

It then does 3 things:
1.Improve genre list : - it calls a function from genre_assignment.py and changes the pre existing list temporarily to provide variability in the recommendation so that the user can explore different options through recommendations but the change is based on other people's preferences (collaborative) which are similar to the target user hence the changes are most likely be accepted by the current user.

2.Recommend based on genre : - then we recommend movies based on genre purely through movie_genre_scoring.py (named as is because it scores the movies based on genre and that's how it recommends it the actual function name is recommend_movie_based_on_genre that is much more intuitive.)

3.Then we combine the result of pre watching factors such as weather, day ,etc with the result of movies recommended based on genre using movie_recommendation.py (named as is because it returns the final combined result aka "recommended_movies")



In genre_assignment.py we try to change the current top genre/cast list that the user has and change it with something which is different that what user is accustomed to but somewhat aligns with the users preference and for that we look at other people's preferences. It uses 2 algorithms for it one is custom made algorithm hence named custom_algo.py and other is algo_2.py.
Each algorithm is run in their own thread hence saving time then using ensemble method we combine the results of each algorithm and returns the newly recommended genre/cast.
Lets talk about both of the algorithms here only

1. custom_algo.py:
   steps in the algorithm are:
   1. find similar users: this is done by taking our top 5 list and compare it with their top 10 list and the person whose matching with our list is higher gets higher preference in the next step

   2. get their top genre that is not in our top genre: 5 highest ranking genre that is in their list but not in our top 5 list is then added to a list and given a score based on the persons priority from the previous step

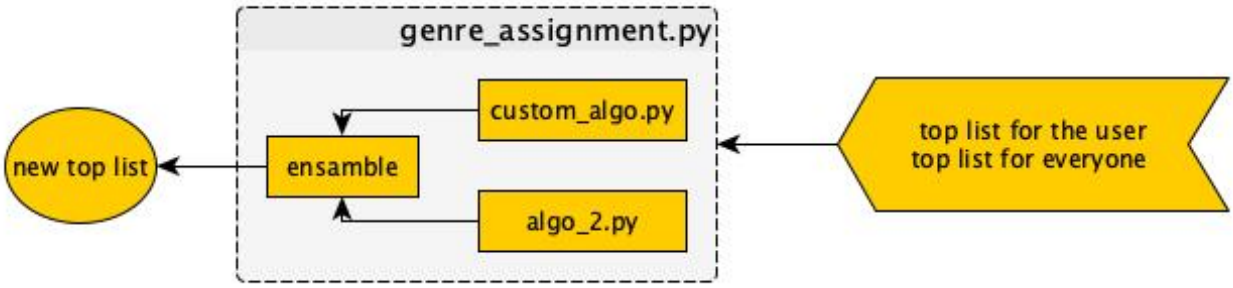
   3. score the genre based on number of occurrences: the scores are added for each person we go through

   4. sort them: we sort them based on the scores

   5. return the new genres: and take the top n items

2. algo_2.py
   We use KNN with means from sklearn surprise (we could also have used SVD and NMF algorithms form sklearn surprise but due to less computing time and manageable RMSE we opted for KNN).

First we give each genre a score according to their order in the list for each user then we try to predict the rating for each unique genre for our user and then sort it and return the top n required.



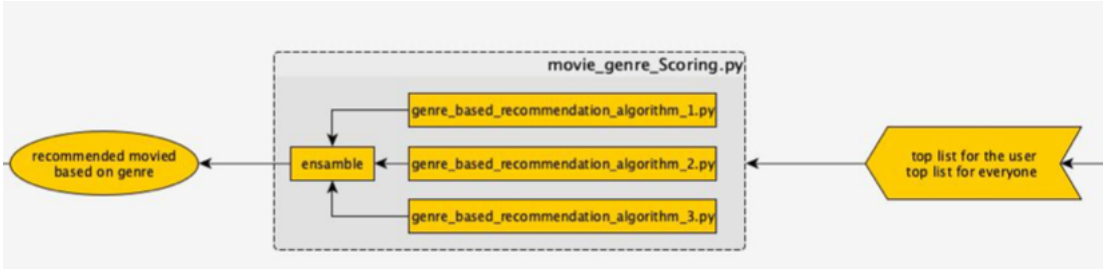
In movie_genre_scoring.py we take the “improved” genre list and predict movies based on genre only and for that we use 3 different algorithms.

1. In the first algorithm we find in each movie how many common tag it has with the user then we score it based on it and after that we try to normalise it according to logit function to accommodate the movies whole spectrum. For example there is a movie that only has one tag romance and there is a movie that has many tags romantic, drama, action, adventure, fantasy, etc so there will be an inert advantage to the movies with larger number of tags so for that only we normalise by computing the completeness factor (how many tag out of total are checked out) and using logit function.

2. In second algorithms first we find similar person to our target person using cosine similarity in scoring a movie with pre existing genre list for each person then using the same scoring for each movie we make a dictionary containing each person-movie-score pair, then we use sklearn-surprise library to predict the score that our user will give to a movie and then this way we rate each movie and then sort and return the top result.

3. In the third algorithm we find similar users using cosine similarity same as second algorithm but while creating a dictionary for sklaern-surpise library we map the top genre from the closest users to each movie and score those movies based on a single genre, each genre has a priority score and while scoring a movie it is taken into consideration. So the mapping is genre-movie-score and then we use the genre from out user and predict the movie scores and then sort and return.

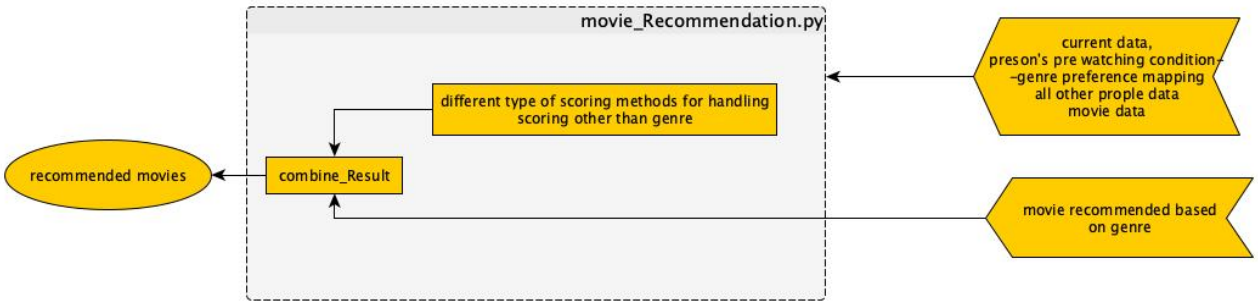
Key points:
1. Even if the all people item list contains the list of the current user too it doesn’t affect the user much because in also 2 and 3 we are using KNN and in that that row is just one voting condition so it doesn’t matter much and also 1 is totally different.
2. Algo 1 is made from scratch because it provides more flexibility to tailor make recommended movie list for the user to throw into the ensemble engine to get considered.



Here the top list is the new improved version

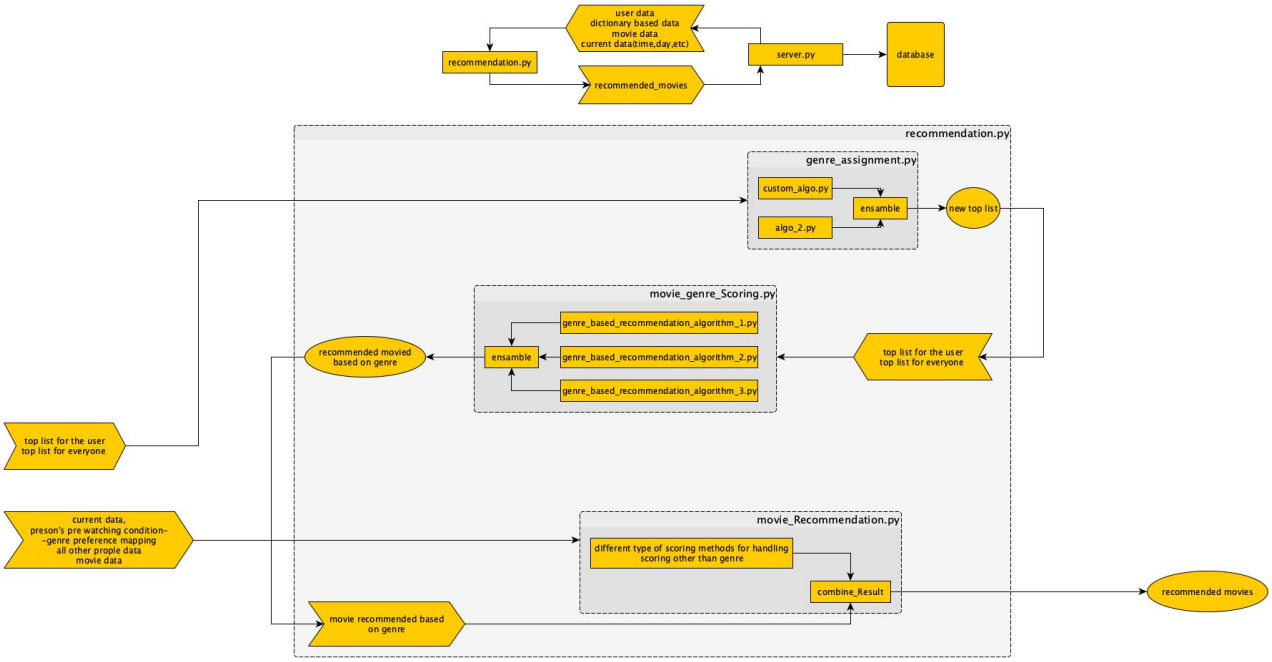
In movie_recommendation.py we predict movies based on the current conditions such as time, weather, day, length of movies, release year, cast and extreme tags. Time, weather and day are stored in a dictionary format hence we use the same function for scoring them, for the length and release year we use mean scoring method, for extreme tags we use a custom-made method and for cast scoring we do the same thing that we did for genre scoring.
in mean scoring we just take the average of the difference and then score accordingly, in dictionary scoring we take the current information that is available to us such as weather, time and day and then take the personal preference of the person in the form of a mapping that is whether-genre preference then we score the movies according to it (eg: if current weather is rainy and in person's user data the rainy weather is mapped to romantic genre then we give priority to the movies with romantic tag).
In the extreme tag scoring if a person has genre in their exclusion list and that genre is found in the movie too we reduct the score exponentially hence preventing the movie to get a competing chance.
Then we combine the result of pre_watching_based_Recommended_movies with genred_based_recommended movies using combine_result.py .

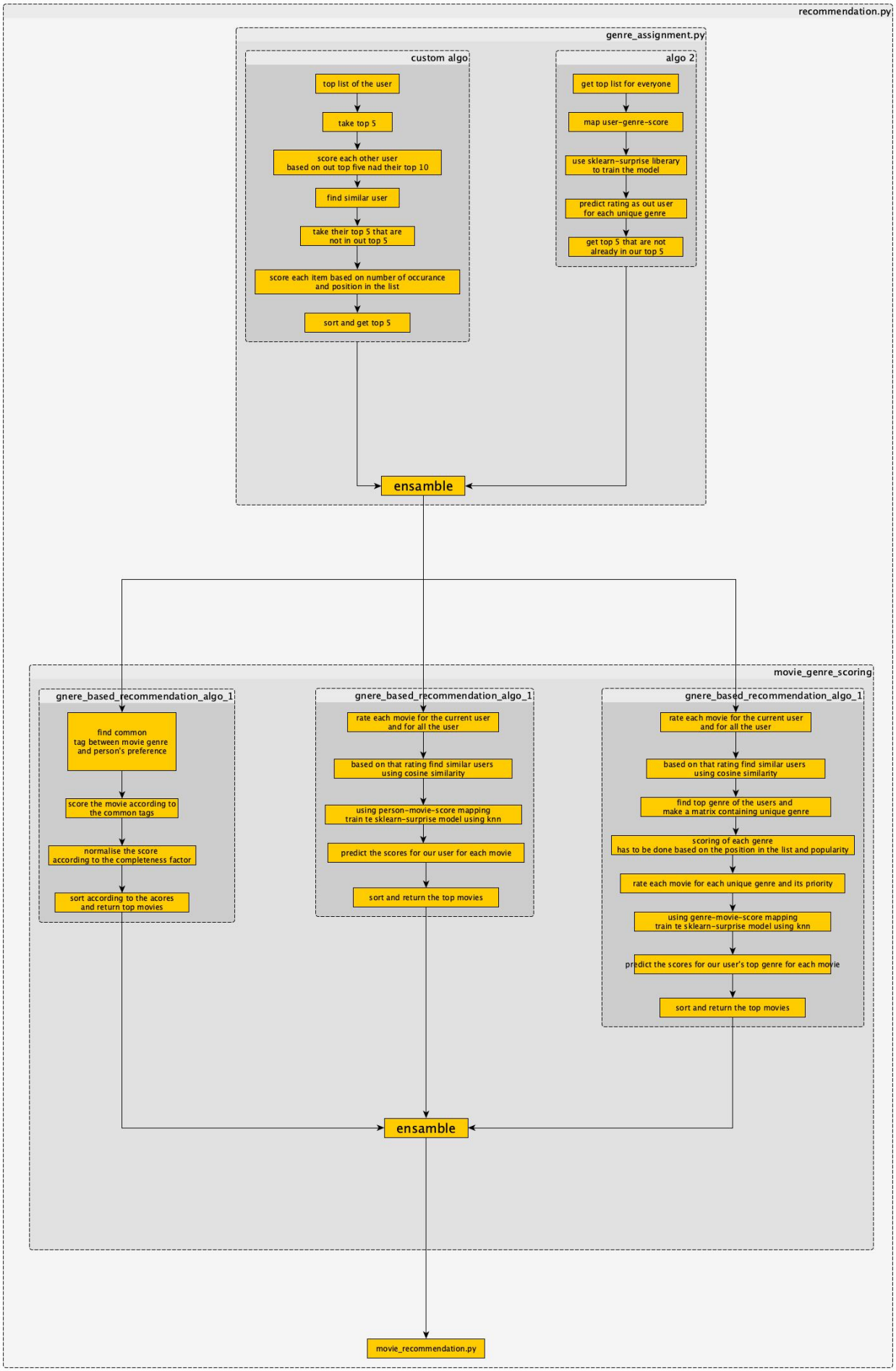
Note:
As the front doesn't have actual videos and we didn't use any weather apis we cannot actually get real data on length of movie, extreme tags, etc hence they are added statically to the persons data in server.py but the code is there already in place to handle it if we had the actual data. Hence there is a scope of improvement in this field "collecting actual real time data from the user".



Apart form these main files there are few extra ones like sort.py , unique_scored.py, ensemble_result.py these are mostly utility files used for sorting, combining matrices into a list ,etc . They are made available to all other files to promote DRY principle in coding and the remaining files are for updating and handling requests from the server (node.js) that will discussed in the next section.

Given below is the full overview of the architecture of the recommendation engine.




recommendation.py
genre_assignment.py
custom algo
top list of the user
take top 5
score each other user based on out top five nad their top 10
find similar user
take their top 5 that are not in out top 5
score each item based on number of occurance and position in the list
sort and get top 5
algo 2
get top list for everyone
map user-genre-score
use sklearn-surprise liberary to train the model
predict rating as out user for each unique genre
get top 5 that are not already in our top 5
ensamble
movie_genre_scoring
gnere_based_recommendation_algo_1
find common tag between movie genre and person's preference
score the movie according to the common tags
normalise the score according to the completeness factor
sort according to the acores and return top movies
gnere_based_recommendation_algo_1
rate each movie for the current user and for all the user
based on that rating find similar users using cosine similarity
using person-movie-score mapping train te sklearn-surprise model using knn
predict the scores for our user for each movie
sort and return the top movies
gnere_based_recommendation_algo_1
rate each movie for the current user and for all the user
based on that rating find similar users using cosine similarity
find top genre of the users and make a matrix containing unique genre
scoring of each genre has to be done based on the position in the list and popularity
rate each movie for each unique genre and its priority
using genre-movie-score mapping train te sklearn-surprise model using knn
predict the scores for our user's top genre for each movie
sort and return the top movies
ensamble
movie_recommendation.py

Working of the server and frontend

The main server file is server.js that handles all the fetch requests from the frontend and this is the file that handles the communication with python that deals with backend.

Calling python from node.js:

We create a child process and we spawn a command line instruction that calls the python files with correct command line arguments. This is why the dependencies of python needs to be present globally (at the root).

The server basically runs on node.js with express to help with linking files.

The frontend calls fetch function to request data from the backend and we handle the request by calling the python file that handles the logic as well as connecting with mongoDB.

Each call is listed below with its handling and respective python file and arguments (all the links below are excluding base url).

1. callPythonUpdate/:name/:movie —here we call the python file handle_update.py and pass the parameters of name and movie that we get from the url. In handle_update.py. this request is gendered when the user clicks on a movie card. After the click we assume the user has watched the movie (for simulation purposes) and now we have to update the user's genre, cast, etc preferences. This is done in the following way : first we get the priority list form the database for that user and then assign each genre a score form 1 to 10 based on the order we received them (they are always stored in the db in the correct priority order) then we simply add the score if the genre is already present in the list and if not we add the genre to the list and give them a constant score. This constant score decides how much we want to weigh in the most recent watches into the decision then the list is sorted and stored back into the database. Same is done for cast preference also.

2. /name/:name/:version — this is called when ever the home page is refreshed or loaded for the first time. We get the user details by calling get_user.py and it internally calls a function that calls server.py which runs the whole recommendation algorithm to change the recommended movies and whenever the page is refreshed again the changes are reflected.

3. /getMovies/:version — this is simply to get the movie details , it calls get_movies.py that gets the detail of the movies from the mongoldb hosted db.

4. / — returns the login page

5. /index — returns the home page

Here in some api call we added a version parameter in the url so that we can handle that if needed. (Future implementation)

The client side script fetches the information and first validates the user if the username and password matches from that of the user then it renders the home page where it requests movie and user data again and renders the page accordingly.
It first injects the roes for each top genre , the recommended movie row will always be present and then each movie is injected into its respective row as a card containing name and the genre of the movie. The UI is very simple because I don't major in frontend but it is intuitive and easy to use and understand.
To watch a movie you click on the **empty space** on a card and that's how the watching is simulated. After the click is registered the subsequent logic is handled.

Database

The database used is mongoDB and is hosted on mongoldb atlas. NoSQL (mongodb) is used because it can scale easily and can accommodate the changing needs in the future as it doesn't have fixed relations. Its integration with python was very easy and in future it can very easily be directly be connected to Node.js . Its currently not being connected to node because of logic handling and libraries (sklearn). It was easier and more efficient to connect mongo with python or else we would have to send very big command line arguments and handling them would be difficult.

How to run the project

Note: the project was developed and tested in macOS

Compilers and package managers required:
Python version : 3.8.3
Node version : v16.13.0
pip version : 21.2.4
Npm version : 8.1.0

A system with multithreading support (in python multithreading is used) is needed.

First install all the required python packages globally.
For this use requirements.txt provided inside recommendation_engine folder
Use the command "pip install -r requirements.txt"
After that in terminal run the command " pip install 'pymongo[srv]' "

Then come inside newfrontend folder and type the command “npm i” or ”npm install”
This will install the required packages for node
Then in the newfrontend folder only type the command “npx nodemon server.js” or “nodemon server.js”.

Then go to a browser and search for “localhost:3000” the login page should appear and then using the username-password json provided login to the page and if a user is logged in for the first time their recommended list will be empty so refresh the page to get the movies in recommended movie row if that is the case.

Note: if the page doesn’t render after login wait for a few seconds because if everything is loading for the first time it would be slower. Same goes foe home page if its blank wait for a few second to fetch the data from the database.

Scope of improvement and project’s future

More algorithms can be added to make the recommendation even more better.
The code is designed in such a way that adding an algorithm is as simple as adding a thread to a thread list and rest all is taken care of and due to multithreaded nature of the system the whole system is as slow as the slowest algorithm and not the cumulative computing time.

The UI can be a lot better and eye catching and additional data like current weather, time, day can be captured and sent to the recommendation engine to make better choices.
If we had actual videos then we can calculate actual watch time and that can also be given to the recommendation engine.

In actual scenario like in case of Netflix every movie is reviewed before uploading so in that time only tags such as murder, blood, sex, nudity, profanity, etc can be captured in a movie and can be stored as “extreme tags” (tag-timestamp pairs) and when a user is watching a movie and leaves a movie whenever these tags come up we can assume that the user doesn’t want to see those and hence this data can be utilised .(currently all this data is statically being used to simulate a real world scenario)

In frontend a way to register a new user should also be added but due to lack of time it wasn't. (But for simulation purposes 13 users are already created with real data (survey was conducted) to support collaborative and hybrid filtering)
After these changes the project can be deployed and can be made public and with time as more and more people register and refine the data the model will become more accurate.

Challenges faced

The biggest challenge faced was lack of time to learn more and implement it. The competition started in the middle of my semester's final exam so most of my time was divided into studying for them also.
Then my domain is DSA, ML and AI with python and java and I had a very little experience in web development hence I had to explore a lot about web development , node.js, express, api, backend, mongoDB are few of the things I had to learn form scratch , even though I have good experience in mySQL but none with noSQL like mongoDB but the project demands for scalable and flexible database solution hence I opted for it.

What I learned

The thing that attracted me towards the third track was implementing and developing algorithms as I like DSA and actively try to make things on my own here I got a chance to learn about different libraries available in python to implement recommendation engines (I used sklearn-surprise here).
In web development I learned how to locally host a website , server side scripts and client side scripts and hot they communicate with each other through apis, database connectivity, etc.
I also interacted with my colleagues working to solve a similar problem and learned different perspectives of solving similar problems.
Through mentors and sessions taken I learned about corporate life and Microsoft work culture.
Overall the experience was very positive and I became better at coding and related fields than I was before.