Adapt Learning Path by Recommending Problems to Struggling Learners

https://doi.org/10.3991/ijet.v16i20.24283

Youssef Jdidou(✉), Souhaib Aammou, Mohamed Khaldi
Abdelmalek Essaâdi University, Tetuan, Morocco
youssef.jdidou@gmail.com

Abstract—The objective of this work is the creation of a resource recommendation application in Python integrated into the code of the virtual edX platform, which appears as an additional tab in each course. By selecting this tab, learners will have access at any time to their recommended issues for this course, and so they can adapt their learning path. In this article, we present a recommendation algorithm that will be responsible for proposing these problems according to the scores obtained in the problems already performed by the learner. By calculating the similarity with the rest of the classmates, an estimate of the most practical problems for the learner will be made. We also present the different functions and parameters to implement it.

Keywords—recommender system, collaborative filter, learning path, Edx platform

1 Introduction

The MOOCs (Massive Open Online Courses) has caused a great revolution in education, a large number of universities and institutions want to offer their courses open in a massive way. However, in MOOCs it is not possible for a teacher to provide personalized help and advice due to the high number of students. Thus, the need to create automatic mechanisms such as recommenders to give this personalized help and advice to learners is obvious [1].

Some important current MOOC platforms already include recommenders, for example Coursera, however, we cannot know how it works as it is not an open source platform. The edX platform does not currently have a recommendation system.

On the other hand, recommendation systems are more and more present in our daily virtual life and, more precisely, recommendation systems applied to education are the subject of numerous studies.

The edX platform is a constantly evolving platform thanks to its open source project Open edX. Developers from all over the world are collaborating on this project, introducing new features to transform edX into a powerful and accessible platform. Being able to improve this platform thanks to a recommender which facilitates learning is the fundamental motivation of this project.

These, are the factors that have favored and allowed the creation of a tool for the platform, which is responsible for proposing the appropriate problems at the level of each learner according to their evolution throughout the course. This provides a more personalized education that adapts to different needs and provides the learner with a quality educational experience.

2 Recommendation systems

2.1 Definition

Recommendation systems, platforms, or engines are a type of information filtering systems that are responsible for predicting user preference for an item [2] or items that might be better for it. One way to make the recommendation is to look at individuals who have similar tastes as the user or at items with characteristics common to other items the user has purchased, seen, or have shown interest in.

Broadly speaking, we can talk about three main types of recommendation systems: collaborative recommendation systems, content-based recommendation systems and hybrid recommendation systems [3]. We are interested in collaborative recommendation systems.

2.2 Collaborative recommendation systems (collaborative filtering)

The main idea of collaborative recommendation approaches is to harness information about past behavior or opinions from an existing user community to predict what things the current user of the system will most likely like or be interested in.

Pure collaborative approaches take a given user-item score matrix as the sole input and typically produce the following types of output: (a) a (numerical) prediction of how much the current user will like or dislike a certain item and (b) a list of n recommended items. Such a Top N list should, of course, not contain items that the current user has already purchased [4]. Two approaches are used in this method:

- Based on the user (user-based recommendation)
- Based on the item (item-based recommendation)

User-based closest neighbor recommendation

Presentation. The first approach we are discussing here is also one of the first methods, called User-based nearest neighbor recommendation. The main idea is simply this: given a grade database and the current (active) user's ID as input, identify other users (sometimes referred to as peer users or closest neighbors) who had similar preferences to those of the formerly active user. Then, for each product p that the active user has not yet seen, a prediction is calculated based on the p scores made by the peer users. The underlying assumptions of these methods are that (a) if users had similar tastes in the past, they will have similar tastes in the future, and (b) user preferences will remain stable and consistent over time.

Better similarity and weighting measures. The basic similarity measure also does not take into account whether two users have co-assessed but only a few items (which they can agree on by chance). In fact, predictions based on ratings of neighbors with whom the active user has noted very little in common have been shown to be a poor choice and lead to poor predictions [5]. Therefore, propose to use another weighting factor, which they call significant weighting. Although the weighting scheme used in their experiments, reported by Herlocker et al. [6], is rather simple, based on a linear reduction in the similarity weight when there are less than fifty items co-evaluated, the increases in the precision of the predictions are significant. The question remains open, however, whether this weighting scheme and the heuristically determined thresholds are also useful in real-world contexts, where the scoring database is smaller and we cannot expect to find many users.

Neighborhood selection. For the calculation of the predictions, we only included those that had a positive correlation with the active user (and, of course, had noted the item for which we are looking for a prediction). If we included all users in the neighborhood, it would not only have a negative influence on the performance against the required compute time, but it would also have an effect on the accuracy of the recommendation, because the ratings of other users who do not are not really comparable would be taken into account.

Common techniques for reducing the size of the neighborhood are to define a specific minimum threshold of similarity of users or to limit the size to a fixed number and take into account only the k nearest neighbors. The potential problems of either technique are discussed by [5, 7]: If the similarity threshold is too high, the neighbor size will be very small for many users, which in turn means that for many items no prediction can be made (reduced coverage). On the other hand, when the threshold is too low, the size of the neighbors is not significantly reduced.

Nearest neighbor recommendation based on item. To find similar items, a similarity measure must be defined. In item-based recommendation approaches, cosine similarity is established as the standard metric, as it has been shown to produce the most accurate results. The metric measures the similarity between two n-dimensional vectors as a function of the angle between them. This metric is also commonly used in information retrieval and text mining to compare two text documents, where the documents are represented as vectors of terms.

The similarity between two items a and b - considered as the corresponding scoring vectors a and b - is formally defined as follows:

$$sim(\vec{a}, \vec{b}) = \frac{\vec{a}.\vec{b}}{|\vec{a}|*|\vec{b}|} \quad (1)$$

Possible similarity values range from 0 to 1, where values close to 1 indicate strong similarity. The baseline cosine measurement does not take into account differences in average user scoring behavior. This problem is solved by using the fitted cosine measurement, which subtracts the user's average from the ratings. The values of the fitted cosine measure vary accordingly from -1 to +1, as in the Pearson measure.

Let U be the set of users who have evaluated the two elements a and b. The adjusted cosine measurement is then calculated as follows:

$$sim(a,b) = \frac{\sum_{u \in U}(r_{u,a} - \overline{r_u})(r_{u,b} - \overline{r_u})}{\sqrt{\sum_{u \in U}(r_{u,a} - \overline{r_u})^2}\sqrt{\sum_{u \in U}(r_{u,b} - \overline{r_u})^2}} \quad (2)$$

Formally, we can predict the score of user u for a product p as follows:

$$pred(u,p) = \frac{\sum_{i \in ratedItems(u)} sim(i,p) * r_{u,i}}{\sum_{i \in ratedItems(a)} sim(i,p)} \quad (3)$$

As in the user-based approach, the size of the considered neighborhood is also limited to a specific size - i.e., not all neighbors are taken into account for the prediction.

3 edX platform architecture

In this section the architecture of the edX platform is fully explained, it will be detailed in the following sections.

edX is made up of several components, as shown in figure 1. We know that one of its main characteristics is that it must be scalable, so it is based on a service architecture, a series of software bricks that can be run on separate machines and extended as needed.

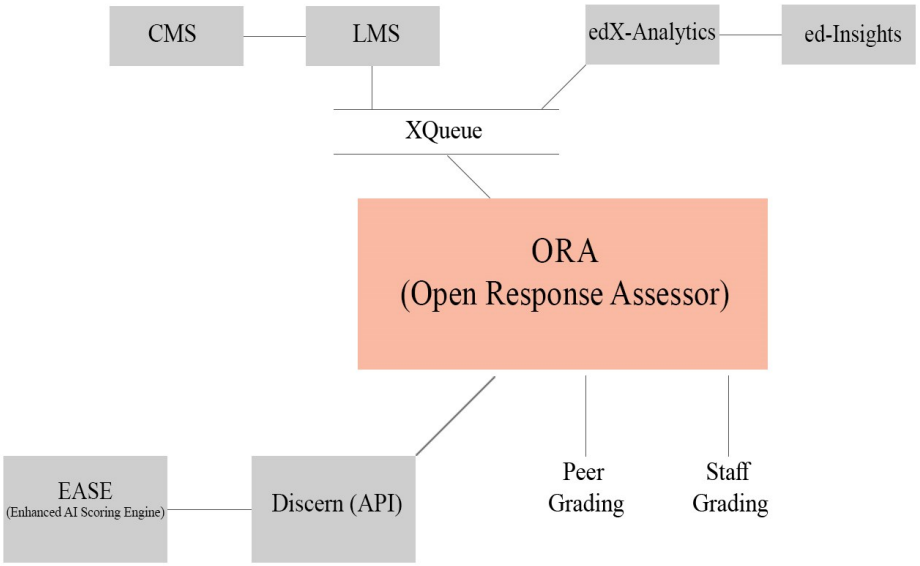

Fig. 1. Architecture of the edX platform

In addition to the above components, edX uses two database management systems:

- **MongoDB**: is a document-oriented NoSQL database system, developed according to the open source concept. Instead of saving data in tables like it is done in relational databases, MongoDB save data structures in standard JSON documents with dynamic schema, making it easier and faster to integrate data into some applications.

In edX, it stores the educational content, that is to say the content of courses and debates or discussion forums.

- **SQLite / MySQL**: in localdev environments, SQLite is used as a relational database management system, it stores user registration data, course registration, progress, status, etc. In production environments, MySQL is used.

Two other most important components of the platform are the CMS and the LMS, two applications from Django that work in both production and development environments:

- **CMS**: is the course management system (edX Studio). This is the part where teachers create and edit lessons. Communicates with the LMS through the MongoDB database.
- **LMS**: is the learning management system. This is the part that the student manages and where the content is shown (videos, problems, tutorials, etc.).

4 Recommendation process

4.1 Recommendation algorithm

Assumptions. The algorithm implemented in this project is based on collaborative filtering systems, since it makes predictions about the most appropriate problems for a learner at a certain point in the course based on the experience of similar performance models [8].

Classmates are collaborators, however, instead of sharing the same assessment models with the user to whom the recommendation is to be made, in this case, the similarity between the learner and his or her classmates is calculated by depending on the number of successfully completed match problems. To explain the algorithm in detail, we start from the following assumptions:

- We assume that we have m + 1 learners enrolled in a course and n problems in it, $\{p1, p2,\ldots, pn\}$.
- The learner l0 is the learner connected to the platform and requires a recommendation at some point in the course.
- The rest of the learners, $\{l1, l2,\ldots, lm\}$, are classmates of l0 who will play the role of collaborators.

Algorithm mechanism. We will illustrate the mechanism of the algorithm by means of an example. Table 1 shows the similarities and differences of Student $a0$ with his classmates when he uses the recommender. In our example:

- m = 15 -> the learner $a0$ to 15 classmates in the course.
- n = 15 -> There are 15 problem type modules in the course.

Table 1. Coincident problems at this point in the course

	$a0$	**$a1$**	**$a2$**	**$a3$**	**$a4$**	**$a5$**	**$a6$**	**$a7$**	**$a8$**	**$a9$**	**$a10$**	**$a11$**	**$a12$**	**$a13$**	**$a14$**	**$a15$**
$p1$																
$p2$																
$p3$																
$p4$																
$p5$																
$p6$																
$p7$																
$p8$																
$p9$																
$p10$																
$p11$																
$p12$																
$p13$																
$p14$																
$p15$																
Number of corresponding approved Problems		5	4	5	3	2	5	1	4	5	5	5	3	5	2	4
	Problem done and approved															
	Problem resolved and suspended															
	Problem unresolved															

The problems posed by each classmate are compared to the problems posed by the learner $a0$ and the number of approved problems in which they coincide is obtained. In this case, we observe that the greatest number of coincident approved issues is 5 and that there are 7 companions that coincide in 5 approved issues: {$a1$, $a3$, $a6$, $a9$, $a10$, $a11$, $a13$}. From now on, we will call them "most similar companions".

Table 2 shows the problems in which each of these classmates best corresponds to the learner $a0$ differs. Only the problems which $a0$ did not realize are taken into account, those which were executed and suspended are not considered as different problems. Since the most coincident companion who had the most problems only reached $p12$, we will limit ourselves to representing this problem.

Table 2. The most coincident and least different companions

	$a0$	**$a1$**	**$a3$**	**$a6$**	**$a9$**	**$a10$**	**$a11$**	**$a13$**
$p1$								
$p2$								
$p3$								
$p4$								
$p5$								
$p6$								
$p7$								
$p8$								
$p9$								
$p10$								
$p11$								
$p12$								
Number of different problems		2	0	0	4	2	2	2
	Problem done and approved							
	Problem resolved and suspended							
	Problem unresolved							
	Problem which is not taken into account because $a0$ has already done it although suspended							

At this point, we reject the most matching companions who do not differ in any issue since what we are looking for are partners who have issues that can be recommended. $\{a3, a6\}$ are excluded from the study because they differ by 0 problems and we will continue with the most coincident partners which differ by the fewest problems, $\{a1, a10, a11, 13\}$, from now on we will designate them as "the most coincident and least different companions". Learner $a9$ is also excluded for now.

4.2 Recommendation algorithm

The key steps in performing our algorithm to display recommended issues are as follows:

1. The MySQL database is accessible and from the 'courseware_studentmodule' table the IDs of the issues that the logged in learner (user_id) in the course (course_id) have resolved are obtained.
2. Once you have the learner issues in the course, it is calculated that they are applied and have failed. For, the score obtained and the maximum possible score for each problem are taken into account.
3. As the algorithm bases its recommendations on the similarities with the classmates, it is necessary to obtain the user ID of each of them.
4. Once we have the IDs of the classmates, we need the IDs of the issues they approved in order to calculate the similarity to the connected student.

5. We calculate the similarity between the learner and each of their classmates and we stick with the most similar classmates, that is, those who agree on the most approved problems.
6. We already have the most assorted companions, now among these the least different are in demand. The IDs of companions who additionally coincide with approved issues with the learner and differ less are recorded. For example, a companion that coincides with the learner in 4 approved problems and differs in 2 will have a greater similarity than a companion that coincides in 4 and differs in 5.
7. Once we have the IDs of the most matching and least different companions, we get the IDs of the issues in which they differ, which will be possible recommendations.
8. We are now looking for the different common issues that are most common among companions, that is, those that were approved the most often by the most similar and least different companions. We only consider approved issues as it makes no sense to recommend issues that other similar peers have failed.
9. We can now make the recommendations. We should recommend as many problems as the parameter indicates:
   (a) We start by recommending that the learner repeat the problems they have failed before continuing to move forward in the course.
   (b) If the required number of recommendations has not been reached, the most repeated problems approved by the most similar and least different companions obtained in point 8 are recommended.
   (c) If we need more recommendations, we continue to recommend issues approved by the most similar companions and a little more different than the least different. In other words, if, for example, we were dealing with more similar partners who differed in an issue, we started recommending issues that differ by more than one.
   (d) In case we have no more problems to recommend, a value of None will be assigned.

At this point, the information is returned and as many recommended issues as indicated in the number parameter of the get_recommendations (user_id, course_id, number) function, called from the application's HTML file, are displayed in the tab.

5 Implementation of recommendation algorithm

5.1 Functions

Once we have identified the necessary fields in the databases and made the connections to retrieve them, we proceed to detail the recommendation algorithm implemented for our application.

So, we define several functions to be developed with its input and output parameters and a brief description of its functionality:

- get_course_problems (course_id): Access the MongoDB database and retrieve all the problems contained in the course. Output: List of problem type modules

- get_course_problems_id (course_id): Form the identifiers of the problem type modules from its attributes. Output: List of modules module_ids
- get_display_name (module_id): Access the MongoDB database and retrieve the name of the problem whose identifier we passed as a parameter. Output: String
- get_graded_problems (user_id, course_id): Gets the IDs of the problems the student poses in the course from the MySQL courseware_studentmodule table. Output: List of modules module_ids
- get_ids (user_id, course_id): Gets the IDs of the learner's classmates. Output: List of learners user_ids
- get_passed_problems (user_id, course_id): Obtains the identifiers of the problems approved by the learner in the course, calculating whether the score obtained is greater than half of the possible score. Output: List of modules module_ids
- get_failed_problems (user_id, course_id): Gets the identifiers of the learner's failed problems according to those achieved and those approved. Output: List of modules module_ids
- get_classmates_passed_problems (user_id, course_id): Gets the problem IDs approved by each of the learner's classmates. Output: Dictionary consisting of {user_id of companions and their list of approved issues}
- get_number_of_passed_coincidences (user_id, course_id): Gets the number of approved issues in which each classmate matches the learner. Output: Dictionary consisting of {companion user_id and number of corresponding approved issues}
- get_passed_coincidences (user_id, course_id): Gets the ids of approved issues that each classmate corresponds to the learner. Output: Dictionary consisting of {user_id of companions and list of corresponding approved issues}
- get_most_coincident (user_id, course_id): Gets the IDs of the classmates that correspond to the most approved issues with the learner. Output: List of learners user_ids
- get_number_of_differences (user_id, course_id): Gets the number of issues in which the student differs from classmates with the highest number of matching approved issues. Output: Dictionary made up of {most suitable companion user_id and number of issues in which they differ}
- get_least_different (user_id, course_id): Gets the IDs of classmates with the highest number of matching approved issues and the fewest different issues. Output: List of learners user_ids
- get_recommended_problems (user_id, course_id): Gets the IDs of the classmates with the highest number of matching approved issues and the fewest approved issues and the fewest different issues and the issues in which they differ. Output: Dictionary composed of {most matching and least different companion user_id: problems where it differs with the student}
- extract_and_count (user_id, course_id): Count the number of classmates who approved each issue where they differ with the learner. Output: Dictionary composed of {different problem identifier and number of companions who approved the problem}
- get_best_recommendations (user_id, course_id d, number): Gets recommended problems for the learner based on their failed problems, repeats of problems where

they differ from their classmates, and other corresponding peer-approved problems. Output: List of modules module_ids

- set_recommendations (user_id, course_id, number): Stores in the recommender_student table the number of recommendations required for the student in the course.
- get_recommendations (user_id, course_id, number): Retrieves from the recommender_studente table the number of recommendations indicated by the number parameter. Output: List of modules module_ids

5.2 Flow diagrams

In this section, some flow diagrams show the functions implemented in the application and a brief explanation of each one.

Get_recommended_problems function. The get_recommended_problems (user _id,course_id) function (see figure 2) is responsible for selecting the problems that can be recommended to the learner with the user_id identifier, that is, the problems approved by their most popular classmates. similar and less different and which the learner has not yet completed.

Set_recommendations function. The set_recommendations (user_id, course_id, number) function (see figure 3) establishes a connection with the MySQL database and stores in the 'recommender_student' table the recommendations for the learner with user_id identifier in the course with Course_id identifier indicated by the parameter number.

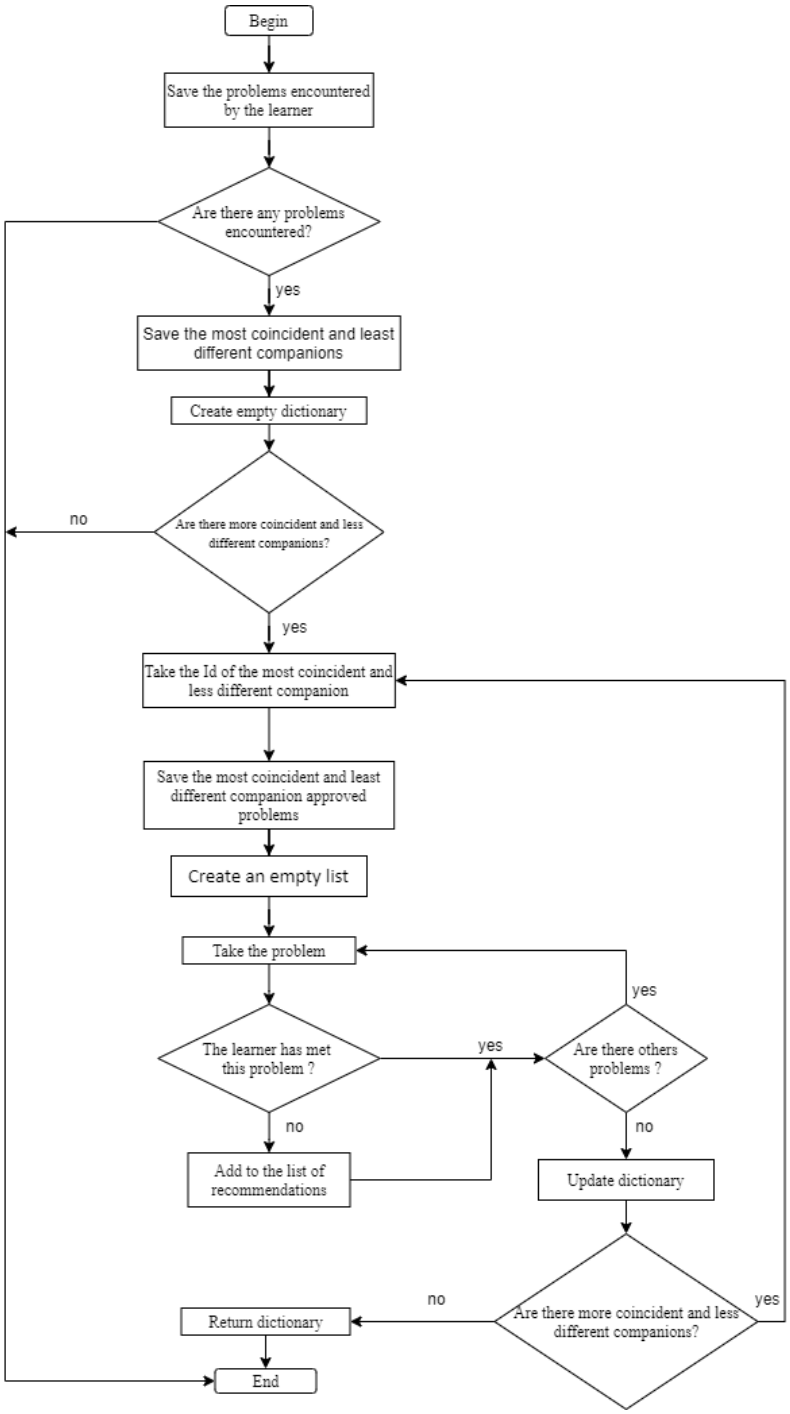

Fig. 2. Flow diagrams of the get_recommended_problems (user_id, course_id) function

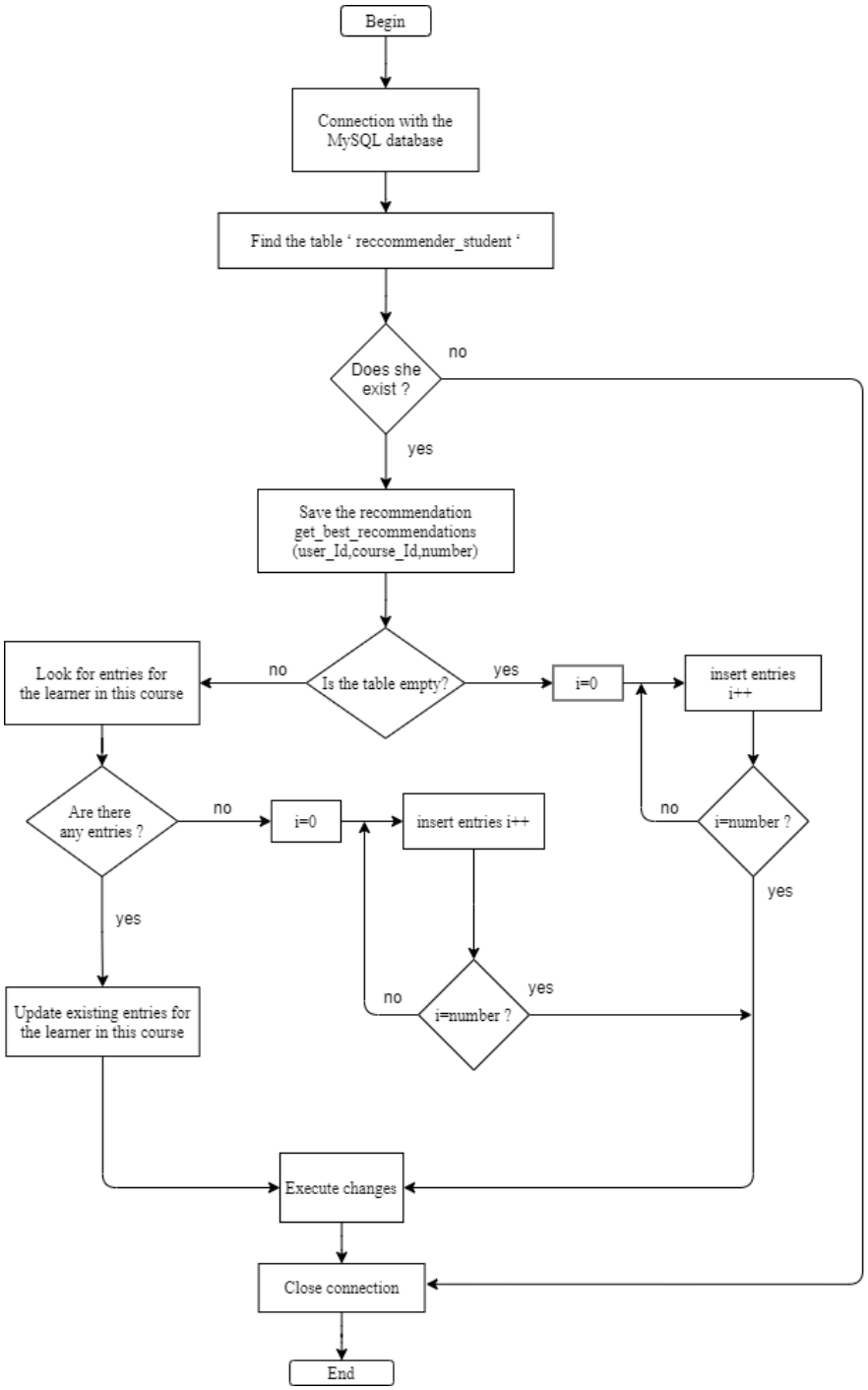

Fig. 3. Flow diagrams of the get_best_recommendations (user_id, course_id, number) function

6 Results

6.1 Only one learner registered for the course (case 1)

In case of being the only learner registered in the course and still not having made a problem, it is not possible to recommend problems to their classmates or problems that they did and failed, therefore, the learner sees the message in figure 4 in the Recommend Me tab!

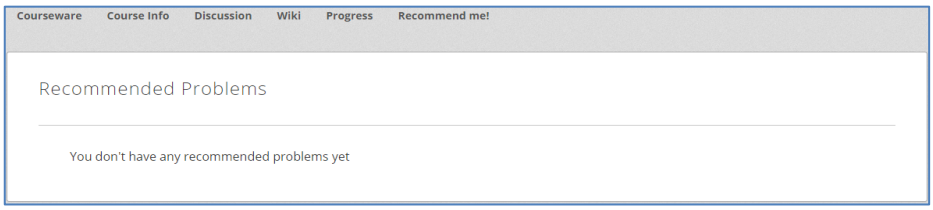

Fig. 4. Warning message to the learner

In the console we get the information displayed in figure 5:

Fig. 5. Case 1 console messages

6.2 Several registered learners. Most advanced backlog

We choose another learner, who is later than the learner we are following so far, and see which problems are recommended in Figure 6.

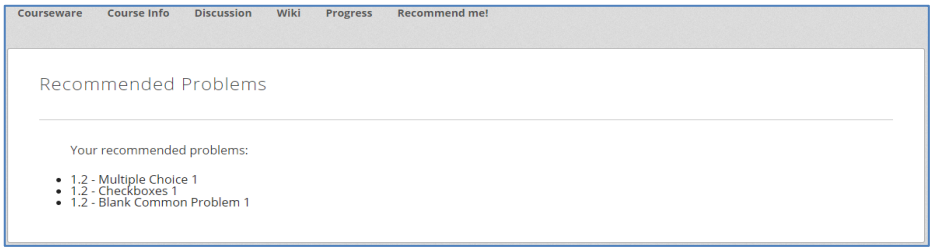

Fig. 6. Problems recommended to learner

In the console we get the information displayed in figure 7:

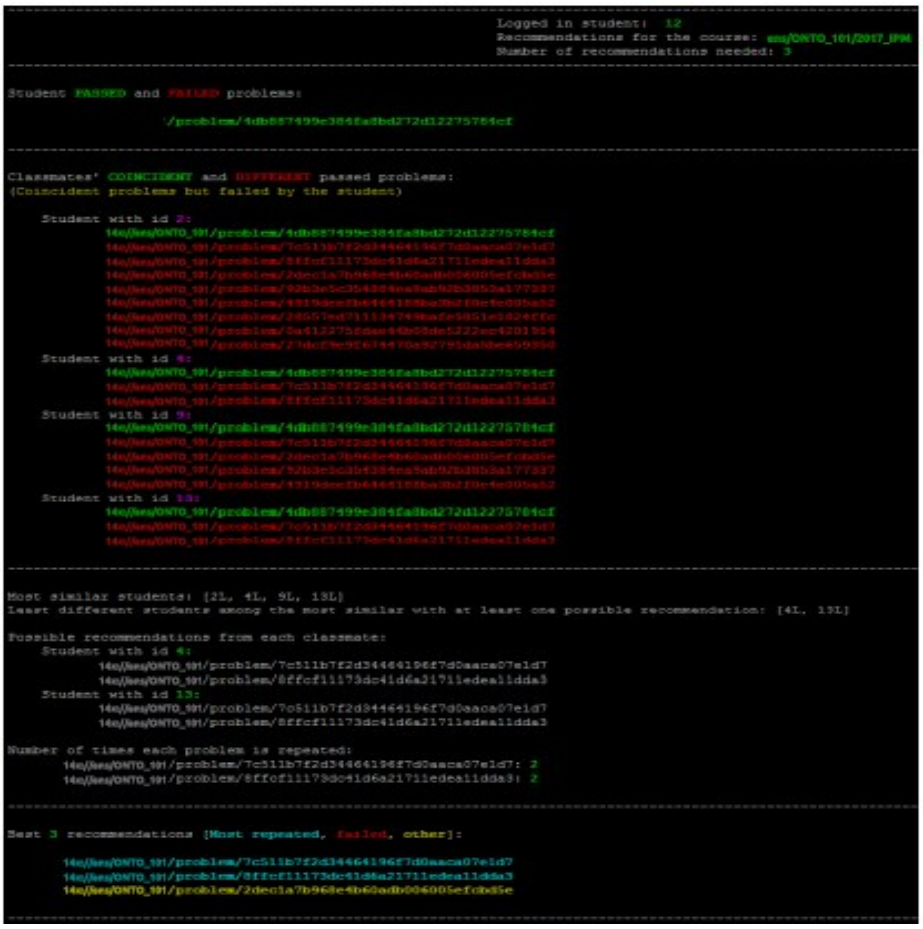

Fig. 7. Case 2 console messages

We analyze the information obtained in the console:

- In this case, we study the learner with user_id = 12 and we see that he has only completed and approved one problem.
- Problems in which it coincides (green) and in which it differs (red) with each classmate at this time are indicated.
- In this case, everyone agrees on a problem (the only one they have done) but with some it differs less than with others. The least different learners are chosen from among the most coincident (4 and 13).
- The times each possible recommendation is repeated (the most coincident and least different problems in red) are counted and the most repeated are recommended.

– In this case, there are no suspended issues (which would be recommended first), then only the issues are recommended by classmates. In this case there are only two issues per repeat, so the rest of the issues will be taken from the issues approved by another of the more similar companions (yellow).

7 Conclusions

In order to draw reliable conclusions, it is necessary to test the recommender with real learners interacting in a course created with different resources.

The objective of this research, the development of a resource recommendation tool for the edX platform, was achieved. For this, a recommendation algorithm was designed from the scores obtained in the problems by the rest of the classmates. This recommendation allows learners to know the problems to be solved.

Regarding the recommendation algorithm, we can say that it has a weakness since it is based on the most common problems among the most similar learners, there might be some problems that are never recommended. This can happen, for example, with problems with a high level of difficulty, because in these cases the success rate is very low, so their popularity index will be close to zero and they will not be offered.

8 References

[1] Jdidou Y. and Khaldi M. 2018. Using Recommendation Systems in MOOC: An Innovation in Education That Increases the Profitability of Students. In *Enhancing Knowledge Discovery and Innovation in the Digital Era*, 176-190. IGI Global. https://www.igi-global.com/chapter/using-recommendation-systems-in-mooc/196511, https://doi.org/10.4018/978-1-5225-4191-2.ch010

[2] Souabi, S., Retbi, A., Idrissi, M. K., & Bennani, S. (2021). Towards an Evolution of E-Learning Recommendation Systems: From 2000 to Nowadays. International Journal of Emerging Technologies in Learning, 16(6). https://doi.org/10.3991/ijet.v16i06.18159

[3] Chen, B., & Wu, J. (2019). Promotive Effect of Psychological Intervention on English Vocabulary Teaching Based on Hybrid Collaborative Recommender Technology. International Journal of Emerging Technologies in Learning, 14(15). https://doi.org/10.3991/ijet.v14i15.11185

[4] Gao, X., Huang, W. X., Wang, N., Yang, Y. C., & Yan, Y. (2016). A top-N algorithm-based personalized learning recommendation system for digital library. International Journal of Emerging Technologies in Learning (iJET), 11(11), 55-59. https://doi.org/10.3991/ijet.v11i11.6256

[5] Herlocker J. L., Konstan J. A., Borchers A. and Riedl J. 2017. An algorithmic framework for performing collaborative filtering. In *ACM SIGIR Forum* (Vol. 51, No. 2, pp. 227-234). New York, NY, USA: ACM. https://doi.org/10.1145/3130348.3130372

[6] Herlocker J., Konstan J. A. and Riedl, J. 2002). An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Information retrieval*, *5*(4), 287-310. https://doi.org/10.1023/a:1020443909834

[7] Mobasher, B. and Anand, S.S. eds., 2005. Intelligent Techniques for Web Personalization: IJCAI 2003 Workshop, ITWP 2003, Acapulco, Mexico, August 11, 2003, Revised Selected Papers (Vol. 3169). Springer Science & Business Media.

[8] Jdidou Y. and Khaldi M. 2016. Increasing the Profitability of Students in MOOCs using Recommendation Systems. International Journal of Knowledge Society Research (IJKSR), 7 (4), 75-85. https://www.igi-global.com/article/increasing-the-profitability-of-students-in-moocs-using-recommendation-systems/174402, https://doi.org/10.4018/ijksr.2016100107

9 Authors

Youssef Jdidou is currently the President of the Association of Scientific Research, Innovation and Technology. Founder of Tetuan International Conference on Education and Technology. He is a PhD candidate in Computer Science at Abdelmalek Essaâdi University, Faculty of Science, LIROSA Laboratory. In research, his current interests include E-learning, Adaptive Hypermedia Systems, MOOCs, and RECOMMENDATION SYSTEMS. He has been involved in several projects like MOOCMAROC, SMARTER® and Chess for everyone.

Souhaib Aammou is a Professor at Ecole Normale Supérieure, Abdelmalek Essaadi University, Tetuan, Morocco; member of research group of Computer sciences and university educational engineering. His research interests include Knowledge representation and reasoning, Semantic networks, Educational Recommendation Systems, Human computer interaction (HCI) theory and educational technologies for learning. Author and co-author of more than 20 publications in international peer-reviewed journals. Reviewer in several refereed journals (IRRODL, iJIM …)

Mohamed Khaldi is a full Professor at Ecole Normale Supérieure, Abdelmalek Essaâdi University, Tetuan, Morocco. Member of research group of Computer sciences and university educational engineering. His research interests include educational technologies for learning, MOOCs, Adaptive Hypermedia Systems. Author and co-author of more than 50 publications in international peer-reviewed journals.

Article submitted 2021-05-26. Resubmitted 2021-06-28. Final acceptance 2021-07-07. Final version published as submitted by the authors.