Research on Personalized Teaching Recommendation System Based on Big Data from Educational Platforms

Mengying Lu*, Kai Cheng
School of Computer and Artificial Intelligence, Henan University of Finance and Economics, Zhengzhou, China
**Corresponding Author*

Abstract: There are now thousands of courses available on online learning platforms. Increased options are better but can be too much to handle students. Most people find it difficult to identify exactly what suits them. To solve this problem, we created a personalized course recommendation system. Spring Boot is used as the backend, Vue.js as the frontend, and MySQL as the database in this system. Two strategies have been combined. Firstly, it identifies users with like-minded interests. Secondly, it incorporates famous courses that are well-rated by most learners. Combined, these techniques enhance relevance and coverage. The platform is compatible with various forms of learning materials such as video courses, reading lists, and case studies. Progress of users can be seen on the intuitive dashboards. They have the opportunity to connect and exchange ideas using a built-in community space. During the testing process, all new users were given helpful recommendations (100 percent coverage). The response time was kept short, with an average of less than 145 milliseconds. The system is lightweight, inexpensive, and simple to install. It is a sensible match to small and middle-sized schools.

Keywords: Online Learning; Personal Suggestions; Finding Similar Users; New User Problem; Mixed Suggestion Method; Spring Boot

1. Introduction

Online learning has grown quickly in recent years. There are now more options for courses. Students now have many choices, but they also have too much information. It's hard to compare similar courses. Many people find it hard to find the right one for them. Smaller education platforms are more affected by this problem. They don't have much money or technology. The usual ways of listing courses are no longer good enough for people's individual needs.

Personalised recommendation systems are a good solution. Research in this area has improved all over the world. International studies have evolved from basic methods [1] to sophisticated models using deep learning and knowledge graphs [2]. These approaches are better at understanding what users like. Chinese research focuses on integrating with local education systems [3,4]. It looks at ways to combine contextual awareness with knowledge tracking. The focus is still on making teaching and learning better together.

The current solutions have three main problems. High-performance systems often need expensive equipment [1,2]. They are too complicated for smaller institutions. Most recommendation models focus on video content. They also overlook other materials, such as documents and case studies [3,4]. New users are another problem because the system doesn't have any data to make useful suggestions. Also, it is often hard to understand how recommendations are made.

This research tries to solve these problems by creating a simple, reliable system that can make personalised recommendations. The most important things we have achieved are:

We developed a new way of making recommendations that combines different types of data. It combines user similarity analysis with popularity and timeliness rules. This is good for new users and works well with limited data.

A system was created to manage resources in a consistent way. It uses a MySQL database to connect courses, materials and cases in a logical way. You can now combine different types of resources and suggest them together.

The system was made simpler by using a full-stack architecture. It is built with Spring Boot, Vue.js and MySQL, and works on

standard browsers. You can use it on one computer at a time, which makes it good for smaller platforms.

Visualisation and interaction features were added. ECharts makes clear data dashboards to make things more transparent. A space where people can learn together helps students and teachers talk to each other.

The system was tested thoroughly and these tests showed that it works well. Tests showed that the software worked properly, was easy to use and was stable even when a lot of people were using it.

2. Introduction to Related Technologies

To build a simple recommendation system, this study chose a backend using Spring and a frontend using Vue.js. Two methods of recommendation were combined: finding similar users and applying basic rules.

It was important to keep the system affordable. So, they didn't use complicated systems like Hadoop. Instead, a lighter option called Spring Boot was selected.

Spring Boot simplifies installation. Simply one note will configure servers and databases automatically. Less XML configuration is required. It also has an internal Tomcat server. It allows putting the entire application into a single JAR file enabling quick deployment. The time spent on the development and maintenance of the site is reduced.

MyBatis is a tool that is utilized to process data. The SQL statements are stored separately in the XML files. This isolates the business logic with the data access. It allows simplifying and improving the SQL. MyBatis is also useful in managing complicated queries. It is capable of working with more than one condition simultaneously. The results get directly connected to Java objects and the data work more quickly.

The frontend has been implemented using Vue.js, based on the MVVM pattern. Data updates trigger automatic screen changes. This method enables rapid updates to recommendation lists. With reusable components such as navigation bars and cards, the code will remain clean and simple to update. Communication between The frontend and backend is managed by Axios and a request interceptor ensures data exchanges are clean and uniform.

In order to create understandable data, ECharts have been introduced. They display teaching statistics in form of charts, namely pies and lines. They indicate the number of different subjects available to a student. Zooming functionality enables the administrators to view the overall trends and small particulars. Normal tables cannot represent them as accurately.

A typical challenge in education is the lack of information for new students with no background data. Therefore, a lightweight hybrid recommendation engine was developed.

User−based collaborative filtering is the main method. It looks at past actions—what users saved or rated. Similar users are found, and their liked items are recommended. To measure similarity between users, the Jaccard coefficient is used. The formula goes like this:

$$sim(u,v)_{Jaccard}=\frac{|I_u \cap I_v|}{|I_u \cup I_v|} \quad (1)$$

The formula is based on what users enjoy. To give an example, suppose that User A bookmarks both "Python Basics" and "Data Analysis" and User B bookmarks both "Python Basics" and "Machine Learning"- their similarity score would be 0.5 (sharing one of two unique courses). This score can be used to identify similar users. The system could recommend a course to User A such as Machine Learning because User B enjoyed it [5,6].

The recommendation process of new users differs. Let me give an example of a newly registered person, say Xiao Wang. No history means that the system displays popular courses such as College English Test Prep and newer ones like AI Tools Guide to him. Once he saves three courses into the system, it detects that he likes programming. It would then start recommending other courses that are similar to what users liked, e.g., Java Advanced [7].

We check how well the system is working in simple ways. First, let's look at the 'hit rate'. If 8 out of 10 recommended courses get clicked, that's 80% accuracy. Also, the quantity of the courses offered is essential. In case a user is able to love 20 programming courses and he finds only five, it means there are some issues with the coverage. Lastly, the speed of the results matters. Clicking on recommend should give the results within less than 150 milliseconds.

Caching allows speeding up things. As an example, when the homepage is opened by User Li, the system will first scan the memory cache to find popular courses such as the Photoshop Tutorials. In case they are not available, the

database cache table is checked. It implies that recommendations do not have to be recalculated every time. The system recalculates only when a user adds new bookmarks. These examples demonstrate how the system reacts to various users and remains fast and accurate.

3. System Requirement Analysis and Architecture Design

The system strictly follows the Role-Based Access Control (RBAC) model, dividing users into three roles: student, teacher, and administrator. The core functional requirements for each role are shown in Table 1.

Table 1. The Core Functional Requirements

Role	Core functional module	Functional description
Student	Front-end application subsystem	Personalized recommendation: The homepage displays "Guess You Like" courses based on a hybrid strategy. Resource retrieval: Supports multi-dimensional filtering and keyword search by course, material, and case. Learning community: Support for posting help-seeking posts and sharing learning experiences in the forum. Personal Center: Centralized management of information such as "My Favorites" and "My Postings".
Teacher	Work Subsystem	Resource Production: Publishing courses， uploading courseware materials, and submitting teaching cases. Data dashboard: View the access popularity and student interaction of published resources through ECharts charts. Content review: Oversee and manage posts within the learning community.
Administrator	Back-end Management Subsystem	User Management: Manages student and teacher accounts, supports password reset and other operations. Resource allocation: Ensure that data dictionaries such as course type and material classification are maintained to enhance the effectiveness of front-end retrieval. System management: Publishing the course information and checking the system operation logs (sys_log).

The system is also using standard RBAC permissions along with JWT-based authentication. Upon logging in, you will receive a digital token which is signed with your password. All subsequent requests should contain this token, including requests to access course materials or submit assignments. Spring Security verifies these tokens. In order to make it more secure, the passwords are encrypted by a method known as bcrypt before being saved in the database. Even in case of data breach, your initial password remains safe. All sensitive actions are logged by the system. An example might be whenever teachers alter course content or students change their passwords. Such logs allow you to ensure that you are following the regulations concerning what may be done with educational data.

There are three key types of information that all have the same management. The first section covers such fundamentals as user identities, their roles and the dictionaries that apply to the entire system. An example of this would be stating that first-year biology students and senior faculty members are not the same. Then come teaching materials, such as course descriptions and unstructured documents, such as video lectures and case studies. We use a combination of storage solutions here. Databases store information about the files whereas the files themselves are saved in a secure online storage area. Finally, interactive data records actions performed by users, e.g., resource downloads, forum participation. Such behavior patterns influence recommendation algorithms directly.

A simple three-layer B/S structure is used in the system design. They each have their own role to play, but they all collaborate. The presentation layer uses Vue.js to build dynamic user interfaces. The course recommendation panel is one of the examples. When users add new interests, it is updated immediately, so there is no need to constantly reload the page. The business layer, fundamentally, deals with intricate logic using Spring Boot. Features such as creation of personal learning plans and implementation of access control are handled by this layer. Structured data is stored in the persistence layer using MySQL, and special

storage of media files is done using specialized systems. MyBatis allows making database queries more efficient allowing querying related tables. This design is extremely flexible. E.g. updating the recommendation engine mostly affects the business layer without affecting the rest of the components. As illustrated in Figure 1, three distinct parts of the system support different sets of people; learning portals support students, authoring platforms support teachers and monitoring consoles support administrators. All of these combine to give a well-organized educational system. Balance between security, functionality, and ease of use is maintained.

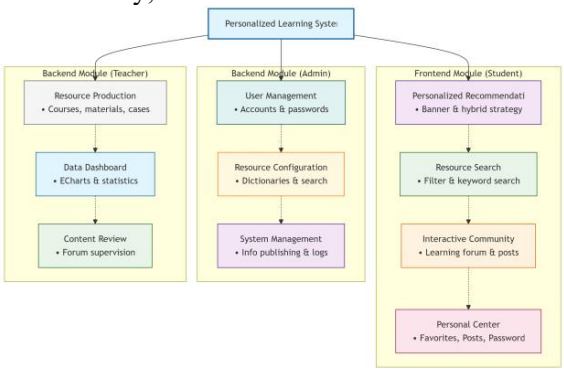

Figure 1. The Specific Architecture

3.1 Student Learning Portal

This subsystem is the primary interface of learners. It assists them in finding the courses and interacting with them. The most critical element is adaptive recommendation module. In case you are a new registrant and have not done much work on the system, the system would present you with the most popular courses as well as the latest resources. After students finish lessons or save materials, the system begins to use what is known as collaborative filtering. It uses the interests of people with similar interests to create so-called Recommended items to you. Suppose a student frequently views programming tutorials; he/she could receive recommendations of more advanced courses in algorithms that other students enjoy.

3.2 Instructor Dashboard

It is a specific workspace of teachers. It assists teachers in creating, organizing and analyzing instructional materials. Teachers may publish organized modules of courses and upload additional materials such as presentation slides and case studies. Resources are all categorized through a predetermined system of categories created by the administrator so that the data can be consistently organized. Analytics tools that are integrated convert usage statistics into straightforward images. The line graph could depict the fluctuation in the number of viewers to the video lectures over a period of weeks. Heatmaps could indicate the times at which the highest number of assignments are sent out. This is evidence-based information that facilitates the changes in the process of teaching and the most optimal use of resources.

3.3 Administrative Control Panel

The subsystem is similar to the nervous system of the platform, as it allows platform stewardship. The administrators are charged with the responsibility of ensuring that critical frameworks are maintained. They consist of assigning user roles, generating institutional taxonomies, and posting system-wide messages. Detailed audit records record significant events, e.g., a change in curriculum or revision of access policy. Specialized permission settings allow various users to access what they require: teaching assistants have access to drafts of courses, whereas only lead instructors have access to publish the final version. The three separate pieces operate in concert via shared service layers. Whenever a student posts a query regarding Calculus in the forum, the appropriate teachers receive an automatic notification. Simultaneously, the recommendation system makes use of the data to refine its recommendations in the future. Such systems form a digital learning environment when combined, which is responsive to the requirements of users.

4. Implementation Architecture and Deployment

4.1 System Deployment Framework

To address the resource limitations common to middle-sized schools and colleges, the architecture uses a unified, single-node deployment scheme. The Vue.js front-end app, Spring Boot back-end services, and MySQL database exist on a single consolidated server. External communication is controlled by the Nginx reverse proxy server which serves static asset requests directly and routes dynamic API calls to backend processors.

The core business logic and recommendation algorithms are wrapped into the Spring Boot runtime environment, which runs on an

embedded Tomcat instance. Relational tables hold structured academic records, and multimedia records like instructional videos are kept as filesystem objects with indexed metadata pointers. This combined solution helps to decrease infrastructure costs but remains responsive in normal loads of usage. Benchmarking shows that it performs consistently with more than 400 simultaneous users on a standard cloud setup.

4.2 Operation of the Hybrid Recommendation Engine

The recommendation engine, which is the intelligent core of the platform, uses a context-sensitive dual-mode approach. Initial queries will cause checking of cache to validate existing valid recommendations. Without a cache, behavioural analytics decides what recommendation mode should be used. Users with less than five documented interactions have their suggestions generated using heuristic rules. They are the combination of global trending articles, which are measured by viewership, and temporally new articles. More seasoned users trigger collaborative filtering processes. The system calculates the similarity coefficients by comparing the history of resource engagement between the users. The students who follow the same interest trajectories (e.g. consecutive enrollment in data science coursework) get higher similarity scores. Then, recommendations are created depending upon the preferred resources of the user’s nearest behavioural neighbors and are constantly updated via further monitoring of the interaction.

The similarity between users is measured using something called the Jaccard coefficient (see equation 1 in Section 2). First, the system identifies the top K users with the highest similarity scores as the target user's neighbours. Next, it brings together all the courses that these neighbours have liked, but which the target user has not yet tried. Each candidate course is given a predicted interest score. This is calculated as the sum of the similarity weights of all the neighbours who have liked that course. Finally, a personalised recommendation list is created by ranking these courses based on their predicted scores.

Finally, the list is delivered to the front end in JSON format. The front end then dynamically renders the list within the 'Guess You Like' module.

4.3 Core Database Structure

Our database is designed to manage a variety of learning resources. It also keeps track of how users interact with these resources. It was designed following standard relational database principles.

The table designed to record user activities is called 'useraction'. Each record in the table has specific info: a unique behaviour ID, the student's ID, the ID of the resource that was used, and the resource type. The resource type tells you if it's a course, study material, or a case exercise. We also keep a record of the action that's been done, like a click, a save or a rating. Every action gets a weight score and a record of the time it happened.

For example, if a student saves a Python course, the system logs it as resource type 1 (course) with the action type "collect." If the same student opens a PDF lecture note, it's logged as type 2 (material) with the action type "click". If you log your actions like this, it helps the system work out what kind of content each user is interested in.

To make recommendations faster, we added a recommendation cache table. This table stores pre-computed suggestion lists so they can be retrieved quickly. Each cache entry includes the student's ID, the recommended course ID and the algorithm that generated the suggestion. It also explains why the course was recommended, e.g. "similar users liked this" or "this is currently trending." We set an expiration time for each entry to keep suggestions up to date.

The courses and materials are really easy to find and use. Each course entry has info like the title, description and difficulty level. Courses are linked to a category tree, so a topic like "Data Science" can have subtopics such as "Machine Learning" and "Statistics." Teaching materials are organised in the same way – PDFs, slides and videos are all sorted consistently.

4.4 Building Key Features

The front end of the website contains tabs and filters that allow easy navigation of the site. The <el-tabs> component makes various sections of courses, materials and cases available. Using the el-tree component, users may explore further topics. As an illustration, they can start with a topic on Programming and then move to Python and then to Web Development.

Axios is used to retrieve data enabling real-time updates. Charts that provide visual information assist users in seeing patterns. An example would be a bar chart that shows the number of students who completed various courses last month, or a line chart that shows how many people accessed the forum every week. ECharts draws the charts and updates them as they are drawn. The learning forum has its own database tables. It allows asking questions, responding to others, and reading discussions. All the items they have saved are displayed in their profile under the heading of My Collections which consists of courses, articles and videos. The list of their forum posts can also be seen under the heading of My Posts. The back-end, behind the scenes, has implemented MyBatis to create smart queries. For instance, when a student chooses Computer Science and writes database, it will search all the same resources in various courses, materials, and cases. All the findings are presented in one list that is readable.

5. Experiment and Result Analysis

In order to test the effectiveness of our hybrid recommendation strategy, we conducted some thorough experiments. We also tested how well the system functions under high load. Each of the tests was performed using a controlled simulation environment.

The test environment comprised the following key components: an an Intel Core i7-12700H processor, 16GB of RAM, and a 512GB SSD. The system was run on Windows 11, JDK 17, and MySQL 8.0. To determine whether or not the interfaces were working properly, we used Postman. Jmeter assisted us in determining how the system performed when many users were logged in simultaneously.

It was hard to obtain the real user information, so we created our own dataset. We created 200 fictitious student accounts. To each person, we assigned a grade level and a major. Then, we included 500 learning resources. They were courses, study materials, and practice cases. All the resources were labeled with category labels.

In order to be more realistic, we fabricated 2,000 user actions. They involved favorites, clicks, and ratings. They were distributed randomly. This resulted in a map of a user-resource interaction. It was as though it were the history of actual users. To observe how the system reacts, we came up with three test situations. Firstly, a new user registers. We will name this user A. On login, A is presented with eight popular courses in the Guess You Like area. There are examples of these courses such as Java Basics and Data Structures. The recommendation coverage in this case is 100%. It means that the fallback rules, which depend on popularity and novelty, do actually assist the new users to start.

Next, we tested an engaged user. User B saves three bookmarks about front-end development. After refreshing the page, the list of recommendations now includes titles like "Practical Vue.js" and "Advanced CSS3." The system explains this by saying: "Recommended based on your bookmarking habits." It clearly learns from what users explicitly choose.

Thirdly, we looked at a user who had not been active. User C only clicked one item. The recommendations that appear are a hybrid list. There are two suggestions that are connected to that single click. The other six are general popular courses. This shows that the hybrid approach can work even when there is not much data. It gives you personal tips and reliable suggestions.

Finally, we compared our hybrid method with basic collaborative filtering. We measured two things: how many users we could help and how quickly the system replied. The hybrid strategy gave suggestions to everyone, even new users. The basic method didn't work for newcomers. The response time was quick, at less than 150 milliseconds. You can find all the results in Table 2.

Our analysis shows clear results. A basic approach to collaborative filtering is entirely ineffective for new users. It doesn't cover these cases at all. Our hybrid strategy, on the other hand, works in all situations. This is possible thanks to its rule-based fallback mechanism.

We also looked at how the system performs over time. We did a seven-day monitoring experiment. Each day, 50 test users carried out normal learning activities. They made around 500 new behaviour records every day. During the week, the system got between 82% and 85% of recommendations right. There was no significant drift.

After the third day, the number of recommendations that were actually used was still more than 70% of the time. This shows that the system learns what users like. Memory usage increased by only 8% over the baseline. This shows that you can manage your resources

well and that you don't have any memory leaks.

The recommendation cache table is very important. This design makes the hybrid strategy respond much faster than pure collaborative filtering. For regular users, the average response time drops by about 31%. The performance in real time gets much better, but it stays accurate.

Table 2. The Evaluation Metrics

Strategy Type	Test Scenario	Coverage Rate	Coverage Rate	Remarks
Pure Collaborative Filtering (User-CF)	New User (No Historical Behavior)	0%	120 ms	Completely ineffective, with severe cold start problem
Pure Collaborative Filtering (User-CF)	Regular users (with rich behaviors)	85%	210 ms	While the performance is good, the computation time increases with the volume of data
Mixed strategy in this article	Full scenario (new/existing users)	100%	145 ms	fast response, no blind spots

We also did stress tests using something called JMeter. The main screen was tested with 100 users using it at the same time. These users got into the system within 5 seconds. The results show that the system can respond to an average of 145 milliseconds and handle 450.2 requests every second. The number of errors stayed the same at 0.00%.

The server's resources were used at a steady level. The CPU was used at an average of around 35% of its capacity. Memory consumption showed no unusual spikes. These findings show that the lightweight standalone deployment can handle the typical concurrency needs of small to mid-sized education platforms. The hybrid algorithm uses caching, so response times stay under one second even when there is a lot of traffic. There were no big drops in performance or timeouts. Overall, the system is stable and always available.

As well as objective metrics, we gathered user opinions. Thirty real users tested the system for one week. They gave their feedback using a 5-point Likert scale. On average, users gave 4.2 out of 5 for the recommended content. The 'Relevance of recommended content' section received the highest rating of 4.5 points. "Interface response speed" was the second most important, with 4.3 points.

It is interesting to note that new users liked the recommendations they received when they first joined. The first thing they did was to try out some popular courses, which turned out to be a great way to get started.

6. Conclusion

Research shows that personalised teaching recommendation systems are really useful. These systems leverage data from learning platforms. They help to match students with the right resources more efficiently.

Our tests showed excellent results. Every user got recommendations, whether they were new or returning. The system achieved full coverage. Response times came quickly, with an average of less than 145 milliseconds between each one. The performance stayed the same even when we tested it during busy periods with lots of users online at the same time.

The recommendation model we built is easy to use and can be adapted to suit different needs. It mostly uses something called 'collaborative filtering' to find patterns among users. When there is not much data, rule-based backups are used. For example, a new student might see the most popular courses first. If you save maths content, you might get suggestions for related topics like statistics or logic later on.

All learning materials are managed through one interface. Teachers and students can see how well things are going with the help of these dashboards. Charts can show which courses are popular or how students are doing in different parts of the course.

But there are still some limitations. The current recommendations mainly focus on clear actions. If a student rates a course or adds it to their favourites, the system will notice.However, more refined actions, such as watching a video again or taking additional time to complete a quiz, are not fully utilized. The system does not also indicate a change in a students understanding over time. Future revisions may comprise models of knowledge tracing. Consider the case of a student beginning with simple algebra. With experience the system can gradually introduce calculus exercises. In the future, we have a few excellent choices. Techniques such as deep learning would allow us to discover connections between various

fields. Graph networks represent how topics are related to one another in various courses. Large language models may turn learning into a conversation. When a student is uncertain about a physics principle, they can pose questions in plain language and be directed to explanations. Lastly, combining various forms of learning data may result in a more detailed image. Forum discussions, video watch patterns, and assignment scores may all be combined to promote the individualized way forward of every student.

References

[1] Walls C. Spring Boot in Action. Translated by Ding Xuefeng. Beijing: People's Post and Telecommunications Press, 2016.

[2] Schafer J B, Frankowski D, Herlocker J, et al. Collaborative filtering recommender systems. //The Adaptive Web. Springer, Berlin, Heidelberg, 2007: 291-324.

[3] Wang Guoxia, Liu Heping. Overview of Personalized Recommendation System. Computer Engineering and Application, 2012, 48(07): 1-4.

[4] Adomavicius G, Tuzhilin A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(6): 734-749.

[5] Xiang Liang. Recommender Systems in Practice. Beijing: People's Post and Telecommunications Press, 2012.

[6] Huang Xiaodong. Design and Implementation of Database Access Layer Based on MyBatis. Information Technology and Informatization, 2021(08): 120-122.

[7] Zhang Xue, Sun Zhihui. Collaborative Filtering Recommendation Algorithm Based on Hybrid Strategy. Computer Science, 2019, 46(02): 23-28.

