Graph-Based Course Recommendation System

Jing Wei, Xianze Zeng, Juntong Zhang, Ao Tian

April 1, 2024

Introduction

As a Uoft student, no matter if you are in your second year (or even your third year), you may find choosing courses troublesome, painstaking, and time-consuming when trying to align with your major requirements, interests, or career aspirations. This can cause trouble entering the major or earning enough credits. **In this project, we are designing a course recommendation system for students who major in computer science, mathematics, statistics, physics, and data science, particularly in the faculty of art and science at the University of Toronto, based on the course they have already taken and other courses' review scores, program, breadth requirements, course level, and professor.** Such a system would leverage historical data from past students' course evaluations and many corresponding pieces of information for different courses to provide personalized course recommendations. With this system, we can support students with this decision-making process that is often associated with uncertainty, stress, and the high stakes of fulfilling degree requirements within the allocated time frame, Which we believe are meaningful to our community.

Datasets Used

Our system utilizes two primary datasets:

- **Course Information Dataset:** Extracted from the Faculty of Arts and Science Calendar and converted into a csv file, including course codes, names, hours needed, descriptions of the courses, prerequisites, corequisites, breadth requirements, recommended preparation, and exclusive courses. For the purpose of our project, we are only using a subset of the dataset with the following columns: course code, prerequisites, corequisites, and breadth requirements.
- **Course Reviews Dataset:** Sourced from Quercus course evaluation pages, the dataset has department and faulty course offered, course code, lecture code, first and last name of the professor, term and year the course is offered, a sequence of rating scores based on different criteria, and the number of students who rated the course. For the purpose of our project, we are only using a subset of the dataset with the following column: course code, first and last name of the professor, the sequence of ratings.

Computational Overview

The core of our recommendation system is a weighted graph representation of courses and their interrelationships. This graph-based approach aimed to represent relationships between courses, programs, professors, and relevant attributes like review scores. The recommendation system employs a weighted graph that helps calculate the similarities between the input courses of users and all the other courses provided in the faculty of art of science to deliver personalized course recommendations.

1. **Building the Graph:** Building the Graph: The graph for our course recommendation system has vertices for courses, programs, professors, course level, and breadth requirements. Edges represent the relationships and connections among these entities, such as a course being part of a program or taught by a certain professor that is associated with a review score from students. A course is also connected to its corresponding breadth number (1–5), which represents different types of courses that engage in a variety of topics for you to receive a well-rounded education, and course level. We construct the graph from the two datasets described in the "Dataset Used" section. Loading the graph involves parsing CSV files, creating vertices for each entity, and building edges that reflect their interconnections. This is mainly accomplished by the function "load graph".

2. **Calculating Similarity Scores:** For each course other than the user's input course, we compute their similarity scores based on shared attributes with the input course, like program, breadth requirement, and course level. Professors and their associated review scores are also taken into consideration. Specific weights are given to different types of connections (program, professor, breadth requirements, and course levels) to reflect their varying importance. For a set of input courses, the system aggregates scores from their similarity calculations with all other courses. Courses already taken or mentioned in the input are excluded from the recommendations. This is mainly accomplished by the function "get similarity score", its helper function, and the function associated with getting course similarity. This algorithm is affected by the weight of each similarity criteria, such as how much the review score affects a course's similarity with the input course from the user or how much the courses in the same program as the input course affect its similarity with the input course.

3. **Recommendation Generation:** Based on the computed similarity scores and additional criteria (such as students' completed courses), we generate and rank course recommendations. This step applies graph traversal algorithms to identify and prioritize courses that best align with the input courses from students that best serve their needs. This task is accomplished by the function "recommend courses." Based on the similarity scores, it ranks courses not yet taken by the student, filters out irrelevant options based on program filters (it also has the highest weight that dominates the scoring algorithm, which helps eliminate potential irrelevant recommendations), and produces a sorted list of recommendations. Specific outputs are described in the next section.

4. **Visualization:** We use Plotly, a real powerful tool to visualize our graph. The overall structure is similar to what we see in the visualization.py in ex3.

Instructions for obtaining datasets and running the program

Following are all Python libraries listed under a requirements.txt file. All the datasets are scraped from the Uoft Websites and are parsed into csv files. All the datasets will be included in the final zip file so no action is required for obtaining the datasets. The file that combines everything and shows the output is main.py. It will ask you for all the courses you have already taken, the more you input, the more recommendations you will get back. It will also ask for your interested program, then it will output three recommendations for each course you input. Run main.py, enter your name, and input course codes that you have taken before. Notice that if the course code does not align perfectly with the specific code in our database, it will ask you if you are inputting a course with a similar course code in the database, all you have to do is copy the code of the course code shown and you are all set. Example:
Enter course: CSC111
Did you mean one of the following?
CSC111H1
Enter course: CSC111H1
Notice that if you do not follow the above example and input the course code that align perfectly with the code in our database, it will not count. After you finish inputting all the courses you have taken or part of them, simply type done and it will omit the recommendations in the format of a dictionary, where each course you input being a key and they are mapped to three recommendations. For further information on pre-requisite courses and co-requisite course. We have provided a link for you to investigate yourself.
Lastly, We used plotly to visualize our graph, if you want to check out the specific visualization, all you have to do is run visualization.py. By default we are drawing the graph "dataset/review full.csv", which take a very long time to load.

1. Code checking: python-ta =2.7.0
2. Web scraping: selenium==4.9.0 (Also download chromedriver so selenium could be run properly)
3. requests==2.31.0
4. beautifulsoup4==4.12.3
5. plotly
6. networkx

ChromeDriver is a separate executable that Selenium WebDriver uses to control Chrome. Follow these steps to setup your tests for running with ChromeDriver:
Ensure Chromium/Google Chrome is installed in a recognized location ChromeDriver expects you to have Chrome installed in the default location for your platform. You can also force ChromeDriver to use a custom location by setting a special capability. Download the ChromeDriver binary for your platform under the downloads section of this site Help WebDriver find the downloaded ChromeDriver executable (Python only) include the path to ChromeDriver when instantiating webdriver.Chrome Specific sample are in the website if you want to check out the example:
https://chromedriver.chromium.org/getting-started

Changes to your project plan

Since our initial project proposal, our plan for the course recommendation system has undergone several refinements and adjustments. Originally, we planned to use simple lists and similarity calculations based on courses previously taken by the students and their declared majors. However, after deep consideration and examination of the data available to us, we recognized the importance of considering a wider range of attributes, so we shifted towards a more comprehensive graph-based model. Where we build relationships between courses, programs, professors (with review scores), course level, and breadth requirements. These additional factors enhance the personalization of our recommendations. Prerequisite and correquisite are abandoned because they are not really helpful in our recommendation system. We tried it at first, then we realized if certain courses required a prerequisite that was not in the input list of courses from the user, it would prioritize that prerequisite. And if the prerequisite of that prerequisite is also not taken, it will prioritize the prerequisite of that prerequisite. This goes on forever and sometimes makes the recommendations really ugly. So the best thing to do is recommend a list of more meaningful and relevant courses and leave the rest for students to look up on their own. This effectively eliminates the problem that students don't have to input all the courses they have taken throughout the school year. (Imagine the number of courses a fourth-year student has to input.)

Discussion

We have built up a concrete course recommendation system for students in the faculty of art and science at the University of Toronto. Our system's core functionality—the calculation of similarity scores based on courses' shared attributes—has proven effective in achieving our project's primary goal. By considering various factors such as program, breadth requirements, course levels, and review scores, we have been able to provide recommendations that are not only relevant but also fit well with students' specific interests and needs. We have encountered several obstacles throughout the process of doing this project. The first obstacle is trying to scrape all the data from the Uoft websites. The amount of data is just humongous, and there is no way we can simply record them ourselves and type our own dataset, so we wrote a scrap course and a scrap review, which helped us parse all the data into CSV files. One headache-inducing problem is that code cannot press a button. When trying to load data from the website, the website only shows part of the data; to see more, you have to press the specific view more button. It took a long time until we came up with a solution for this. The second obstacle is the overall workflow. All group members start at the same time and are responsible for different crucial parts of the project. This means someone needs to write the scoring algorithm before the data is parsed into the CSV files, load the graph without knowing the specific structure of the data, or even start with the visualization before the graph is fully functional. This meant making assumptions about data formatting and availability, which could lead to significant revisions later. And in the final stage, we made a huge number of revisions; the whole previous scoring algorithm was abandoned because it did not align with our final graph structure and idea. Different parts of the project do not work together because they are completed separately and therefore need to be modified almost completely. The third obstacle is learning how to actually use Plotly. Although examples are presented in Ex3 and Ex4, generating the graph that suits our needs and blending it into the whole project takes quite some time. Taking into account the limitations in our course recommendation system, the most obvious one is that some data is a little bit outdated, and the variability in course evaluations could affect the relevance of recommendations. While the graph-based recommendation offers a concrete framework for analyzing course relationships, the algorithms still need to be optimized for better efficiency. The current implementation focuses on backend algorithms. We didn't put too much effort into the interface of the project, which might limit user engagement and accessibility. Developing an interactive and user-friendly interface would significantly enhance the usability of our system. We have also included a lot of factors when computing the recommendations; however,

due to a lack of data, we didn't have the opportunity to include individual student preferences such as workload tolerance and interest levels. Lastly, talking about some next steps for further exploration, there are a few points mentioned before. The first is further expanding the dataset to include more diverse and comprehensive course evaluations across different faculties so that we can effectively enhance the system's applicability. The scoring also needs to be optimized for better efficiency, along with a more interactive user interface and the integration of personal preferences. In conclusion, our graph-based course recommendation system is a significant step forward in academic planning. While the current implementation provides a solid foundation, there is still a lot of room for improvement and innovation. Overall, the challenges we faced throughout the process were valuable learning opportunities, and we really got to gain a deeper understanding of the abstract data structure.

Reference

1. "Academic Calendar." 2023-24 Academic Calendar — Academic Calendar, artsci.calendar.utoronto.ca/. Accessed 1 Apr. 2024.
2. Quercus Course Evaluation Pages.
3. "Webdriver for Chrome - Getting Started." ChromeDriver, chromedriver.chromium.org/getting-started. Accessed 1 Apr. 2024.