CSC110 Project Proposal: Secondary School Course Enrolment during the COVID-19 Pandemic

Billy Guo and Sridhar Sairam

Tuesday, 14 December 2021

Problem Description and Research Question

Ontario secondary schools typically run on a semester system, where the first semester is from September to January and the second semester is from February to June. During each semester, students typically take 4 courses; students can drop (unenroll from) a course before the middle of the semester if they no longer wish to take it.

The COVID-19 pandemic started around March 2020, when students started learning from home for the remainder of the year. Since this was before the middle of the semester, students still had the opportunity to drop courses. However, due to the pandemic, the Ontario government said that final grades for semester 2 courses could not be lower than students' grades before the March break (when the learning from home started). In addition, the number of hours per week of learning a teacher could assign was drastically reduced, causing some courses to be unable to achieve the same amount of learning it would otherwise have.

With experience for each course going to be very different from usual and considering that their mark would not decrease, would students still drop courses that they did not want to take? If many students dropped courses due to the pandemic, we would see a significant drop in overall course enrolments for that year, when compared to previous years. If course enrolments were similar when compared to other years, it would mean that the pandemic seemingly had not immediately impacted course enrollment.

Being students that experienced this situation, we wondered whether other students dropped courses. Given this, the following question arose: **Did the COVID-19 pandemic have an immediate effect on the 2019 - 2020 school year's course enrollment numbers in a significant manner?**

Dataset Description

The datasets used are a collection of .txt and .xlsx files. Each school year has its own pair of .txt and .xlsx files. Since the data in both file formats are the same, just formatted differently, we opted with using the .txt files.

The datasets were found on the Canadian government's Open Government section. The datasets themselves originate from the Ontario government's Data Catalogue.

The .txt file is organized in a similar manner to a CSV file, but with a delimiter of a vertical bar, "|". The courses that the dataset contains are courses defined by the Ontario Ministry of Education, for grades 9 - 12. Each row contains the following information about one specific course: course code, course description (course title), grade level, pathway or destination, and the number of students in that course. We use all of the columns of data in our project, however, most of the information regarding the course is mainly used just for labelling purposes.

Computational Overview

The code was broken into 3 files: `functions_and_classes`, `command_line_interface`, and `main`. Computationally, `main` doesn't really do that much. It just imports and runs the necessary functions to run the program.

`functions_and_classes` is the basis of all of the computations and analysis. Within the module, The classes and functions have been divided into parts: "Classes", "Importing .txt files", "Text File Data Extraction", "Course Searching and Related", and "Statistical Analysis". The "Classes" section of the module contained all of the classes that were used to represent the data from the database files. For this, a `Course` class was created. This represented an Ontario course. There is also a class attribute that keeps a collection of all `Course` objects; this ensures that there are no duplicate `Course` objects (as in objects that are equivalent, value wise, but are stored in different memory locations). Building on top of the `Course` object, we have the `CourseEnrollment` object. This encapsulates a `Course` object, and its respective enrollment values, for a given school year. The `SchoolYearAllCourseEnrollments` class encapsulates all `CourseEnrollment` for a given school year. Then, the most important class, the `CourseEnrollmentHistory` class. This object contains a collection of `CourseEnrollment` objects for a specific courses, where each `CourseEnrollment` object represents a different school year. This object will allow us to see how a course's enrollment numbers change throughout its history (throughout the school years).

The "Importing .txt files" and "Text File Data Extraction" help sanitize the input from the database files, then group everything together. There are many helper functions that all end up in the grand result of `get_course_enrollment_history_for_all_courses_in_folder`. Using this function, we can input a directory path, and using the helper functions, it is able to go through each database file, create a respective `SchoolYearAllCourseEnrollments` for each file, and then merge them all together to get a master list of `CourseEnrollmentHistory` objects. This master list is a huge list that contains the enrollment history for all of the courses in the database folder.

The "Course Searching and Related" functions intended to be used by the functions in the `command_line_interface` module, but due to issues (that I will discuss in later section), these functions ended up not being used.

The "Statistical Analysis" functions are used to plot the graphs, displaying a course's enrollment history. Here, two functions are used to plot 2 graphs. One function displays the actual course data with its polynomial regression alongside it, and the second one displays the residual/difference graph for the enrollment data and polynomial regression.

The `command_line_interface` consists of functions intended to communicate with the user, so that they can choose what courses they'd like to run. Originally, this was intended to be a section for a GUI, using `tkinter`, but for unforseen reasons (that I'll get into in a later section), I had to create a CLI instead. Although I've broken the module into 2 parts, the second part, "Main Loop", is just a function that uses all of the helper functions. Speaking of, the "Command Line Interface Helper" breaks down each step of the interactive process. When interacting with the user, we want to make sure that 1 misinput didn't end the whole program, so adding some input sanitization, we grouped each step with functions. The major computational procceses here are: validating input, raising and catching possible errors, and repeatedly asking for a valid input until the user provides a valid input.

Instructions: Obtaining Data Sets and Running the Program

These instructions are also included in the `main.py` file.
Instructions:

- A folder named `database_files` needs to be created in the same folder the 3 program files are in.
- All database (.txt) files that need to be downloaded must be placed in this newly made folder (`database_files`).
- (including .txt files in the `database_files` folder that are not a part of the database may cause problems)
- The specific database files to be downloaded can be found in the module description of `main.py`, or below, formatted as a Python list of URL strings.

```
["https://files.ontario.ca/opendata/mdc_enrol_1112_en_supp_2.txt",
    "https://files.ontario.ca/opendata/mdc_enrol_1213_en_supp_2.txt",
    "https://files.ontario.ca/opendata/mdc_enrol_1314_en_supp_2.txt",
    "https://files.ontario.ca/opendata/mdc_enrol_1415_en_supp_2.txt",
    "https://files.ontario.ca/opendata/mdc_enrol_1516_en_supp.txt",
    "https://files.ontario.ca/opendata/mdc_enrol_1617_en_supp.txt",
    "https://files.ontario.ca/opendata/mdc_enrol_1718_en_supp.txt",
    "https://data.ontario.ca/dataset/83b53104-19f9-4a77-8e38-a56c5575ef1c/resource/
    f4e0fef6-61f9-47cd-ae69-353b90fe0d7e/download/mdc_enrol_1819_en_supp.txt"]
```

- After this is done, the `main.py` file can be run.
- Via interaction with the console, the user will be able to plot graphs (that will open in a new window).

What to Expect:

- You should see an introductory mesasge, followed by more text that checks for the files.

- Given that you have placed the files in the correct place, you will be able to now search for courses. Here, you will give 2 numbers (indices), that represent a range of courses.
- Typing two numbers (separated by a space) will output the list of courses with numbers within that respective range!
- Once you are done searching through the courses, you can input an empty string to select a specific course to graph.
- Using the same course numbers from before, you will input a course number for the course you'd like to graph.
- After asking how many years to predict, and the degree of the polynomial regression, it will graph the course enrollment data, the polynomial regression, and the residuals (differences) graph (for the polynomial and the actual data)
- Then, the program will end.

Changes Made between the Project Proposal and Final Submission

Originally, we intended to use matplotlib to graph plots and tkinter to create a GUI to open these plots. Unfortunately, do to an unforseen problem, we weren't able to do that. We ended up creating a CLI, where the user now interacts via the console, to search and select a course to see its course enrollment history, and see how well the polynomial regression predicts future enrolments. This was the only major change that occured.

The bug was one that was really odd and unexpected. When we started to write the GUI using tkinter, I started to map out a general interface. Using `grid`, I had an `Entry` section on the top, which had a `Button` beside it that would take in a file path (the file path of the data set folder). Using this, we would show a `Listbox`, and have a `Scrollbar` so that one could scroll through all of the courses, and select the one they'd like to graph. Then, they could use buttons on the bottom to show the graphs.

It seems like there wouldn't be a problem. It started off smooth sailing, but then, creating a `StringVar` instance to get the text from the `Entry` is where the problem occured. I had to use functions from the `functions_and_classes` module, so I used the wildcard import. This also imported `matplot.pyplot`. When I tried to get the string data from `Entry` using `StringVar.get`, I was always getting an empty string! However, when I removed the import statement, the `StringVar.get` method would return the correct string. After lots of trial and error debugging, I found out that when I imported `matplotlib`, the `StringVar.get` method was not working. After some digging, my final guess is due to the fact that the `matplotlib` library has a top level function called `get`, which may be interfering the `StringVar.get`. This being a guess, I'm not 100% certain about this.

Discussion

discussssssssssss

References

Education, & Government of Ontario. (2020). *Course enrolment in secondary schools.* Retrieved from https://open.canada.ca/data/en/dataset/83b53104-19f9-4a77-8e38-a56c5575ef1c.

Education, & Government of Ontario. (2021). *Second language course enrolment.* Retrieved from https://open.canada.ca/data/en/dataset/4546116e-e67f-4dcc-b2d7-9a701bd3be2f.