Computing Science 1P/1PX

Laboratory Examination 2 2015-16

21-23 March 2016 **Time allowed:** 1 hour 50 minutes + advance preparation

Weight: the exam contributes 10% to the overall assessment for the CS1P course and 20% to the overall assessment for the CS1PX, respectively.

Threshold: students must attempt at least 75% of all assessments on the CS1P/1PX course in order to gain credit for the course.

Instructions

Before the exam

You should prepare a complete solution to the exam question given in this pack in advance of the exam. Put in as many hours as you need to get a properly working solution **that you fully understand**. You can of course type it into a machine and thoroughly test and debug it.

When you come into the exam, you will be able to bring nothing with you. We will provide you with another copy of the question along with some rough paper to work on. You need to be prepared to reconstruct your solution to the problem during the exam.

You may of course talk to others while preparing for the exam. Remember, however, that you will be on your own in the exam, and so you must understand your solution *thoroughly* before coming to the exam.

During the exam, the machines will be disconnected from the network and from your filestores. Note however that the Python Docs help in the Idle Help menu will be available. You should familiarise yourself with its contents, in case you get stuck during the exam with Python syntax.

There will be no access to the laboratory, other than to take the lab exam, at any time during the period **21-23 March 2016** inclusive. The lab may also be closed for short periods during the week before and on Thursday/Friday during the lab exam week to allow for preparation by the systems team.

Starting the exam

Complete the attendance slip and place it in front of you ***together with your matriculation/student ID card***.

Log in to the system using your special exam username and password as provided on the document in your examination pack. Use AMS to set up `LabExam2`. This will generate copies of the template program files in your exam account Workspace.

During the exam

As in a normal examination, communication between candidates during the laboratory exam is strictly forbidden.

A candidate who wishes to print a document during the exam should summon an invigilator, who will collect it from the printer and deliver it.

A candidate may leave early, but only after summoning an invigilator, who will ensure that submission has taken place. Any candidate who experiences a hardware or software problem during the examination should summon an invigilator at once.

At the end of the exam

Candidates should ensure that they have submitted their solution, using AMS, before the end of the examination.

Candidates must leave the laboratory quietly; the exam may still be in progress for other groups.

Lab Exam 2 Problem: An Automated Recommendation Program

Personalised recommendation services are part of many modern websites. Amazon, Youtube, and Facebook are all examples of sites that suggest potential items (e.g., products, videos, etc.) of interest to users based on their previous activity or purchases on the site. This can be handy for customers, but also very important for businesses.

The task of this exam is to write a program that makes personalised book recommendations using a similarity algorithm to predict which books a user is likely to enjoy reading based on her/his past or sample ratings.

The following sections provide further information about the main aspects of your program.

1. Input Files

Your program should read data from two input files:

`books.txt` includes a list of 55 books in an `<author>,<title>` format, one entry per line

`ratings.txt` includes usernames to represent users of the service followed by a list of 55 integers, each representing a rating for each book from `books.txt`, in the same order. The file is structured using the following format: `<user_a>\n<user_a_rating_1> … <user_a_rating_55> \n`

2. Ratings

Each integer rating should be interpreted as follows:

Rating	Meaning
-5	Hated it!
-3	Didn't like it
0	Haven't read it
1	OK
3	Liked it!
5	Really liked it!

3. The similarity algorithm

If `User A` enjoyed reading a number of books that `User B` also enjoyed reading, then when `User A` recommends another book that `User B` has not read, chances are that `User B` would like to read it too. On the other hand, if the two users always tend to disagree about books, then `User B` is not likely to read a book that the `User A` recommends.

A program can calculate how similar two users are by treating each of their ratings as a vector and calculating the dot product of these two vectors. The dot product is simply the sum of the products of each of the corresponding elements.

For example, suppose we had 3 books in our database and `User A` rated them [5, 3, -5], `User B` rated them [1, 5, -3], `User C` rated them [5, -3, 5], and `User D` rated them [1, 3, 0]. The similarity between `Users A` and `B` is calculated as: (5 x 1) + (3 x 5) + (-5 x -3) = 5 + 15 + 15 = 35. The similarity between `Users A` and `C` is: (5 x 5) + (3 x -3) + (-5 x 5) = 25 - 9 - 25 = -9. The similarity between `Users A` and `D` is (5 x 1) + (3 x 3) + (-5 x 0) = 5 + 9 + 0 = 14. We see that if both people like a book (rating it with a positive number) it increases their similarity and if both people dislike a book (both giving it a negative number) it also increases their similarity.

Once you have calculated the pair-wise similarity between `User A` and every other user, you can then identify whose ratings are most similar to `User A`'s. In this case, `User B` is most similar to `User A`, so we would recommend to `User A` the top books from `User B`'s list that `User A` hasn't already read.

4. The complete program

Write a program that takes users' book ratings and makes recommendations to them using the similarity algorithm above. If the user for whom we want to provide recommendations has already rated books in the database (i.e., the username already exists), then similarity with other users should be calculated based on her/his existing ratings. Otherwise, the program should ask the user to rate about 20% of the titles selected at random (from `books.txt`) and then calculate similarity and make recommendations based on the user's input.

The program should return a number of recommendations (specified by the user) of books that the user has not already read, and other user(s) with a high similarity score have rated positively (i.e., with 3 or 5). If the required number of recommendations cannot be compiled from the single user with the highest similarity score (because, e.g., both users have read the same books already), then the list should be completed with recommendations from user(s) with the next highest similarity score and so on.

5. User Input and Output

You are NOT required to provide a Graphical User Interface for this exam. A Command Line Interface should suffice. If you want, however, the extra challenge of providing a GUI environment you are allowed to do so.

The program should ask for a username and the number of recommendations required. The default should be 10. If the username does not already exist in the database, then a random sample of ratings should be requested from the user as specified above. The list of recommendations should include titles rated positively by other user(s) with high similarity score(s).

The program should print the list or recommended titles, *also indicating which titles have been recommended by which user*.

Finally, this recommendation list should also be written to a file.

6. Testing

The AMS folder will include the two input files (`books.txt; ratings.txt`) and a single placeholder (`.py`) file for you to write your code in. A placeholder output (`.txt`) file will be included for you to write an indicative list of 10 recommendations for a user, as a sample output of your program.

You should perform extensive error checking for you program (such as, e.g., incorrect input, non-existing files, invalid ratings, etc.).

Specific Instructions

What you must do and what you must submit

You **may** use any standard Python modules, functions and methods. For example, the `split` function from the `string` module (alternatively, the `split` method of a string) and functions from the `random` module are likely to be useful.

You **must** do your work in a single file: **`recommendations.py`**

(if you choose to implement a GUI, then you are allowed to create a second file **`gui.py`**.)

The following files will be available under your AMS setup and have also been made available on Moodle for you to start work on and check your solution:

- `recommendations.py`: Placeholder for the main program
- `books.txt`: Input file with the list of book titles
- `ratings.txt`: Input file with the list of user ratings
- `output.txt`: Placeholder for your sample output. **IMPORTANT**: this file **must** include the following information after submission:
    - The name of the user for whom these recommendations are made. For testing purposes, you should choose **one of the existing users** from `ratings.txt`
    - The list of 10 recommendations **CLEARLY** indicating **which books** are recommended by **which other user**.

Suggestions on how to progress

1. Choose appropriate data structure(s) to use and code/function(s) to process the input files
2. Think carefully about the design of your program. Use the necessary functions for your code to be readable, your overall implementation efficient, and your error checking adequate.
3. Think carefully how many data structures you will use and how to manipulate them in order to produce the desired result.
4. If you choose to implement the GUI, carefully plan the interaction between `gui.py` and `recommendations.py` in terms of function calls and parameter passing

Marking guidelines

The following aspects of the program will be taken into account in marking:

- use of appropriate and correct algorithms and Python control structures
- program efficiency
- use of appropriate data structures
- correct Python syntax
- good programming style, including layout, use of explanatory comments, choice of identifiers, and appropriate use of functions
- error checking
- correctness of the implementation on unseen test input