CME 211: Homework 2

Due: Friday, October 14, 2022 at 5:00pm (Pacific Daylight Time)

This assignment was designed by Patrick LeGresley and modified for the purposes of this course.

Background

Recommendation systems are a class of algorithms in *machine learning* that attempt to predict the preferences of a user. For example Amazon and Netflix present recommendations to customers about what products they might like to buy or what movies to watch. The recommendations could be computed using a wide variety of data including demographics (gender, age, location, etc.), purchase or viewing history, *item similarity* (e.g. you have rated movies categorized as comedies very highly, so other comedies are recommended), and *user similarity*.

The similarity of two items can be quantified by finding all of the users that have rated both items. If the two items have similar ratings from those users they have a high degree of similarity. One measure of item similarity is the adjusted cosine similarity for items a and b:

$$P_{a,b} = \frac{\sum_{i=1}^{m}\left(r_{a,i} - \bar{r_a}\right)\left(r_{b,i} - \bar{r_b}\right)}{\sqrt{\sum_{i=1}^{m}\left(r_{a,i} - \bar{r_a}\right)^2 \sum_{i=1}^{m}\left(r_{b,i} - \bar{r_b}\right)^2}}$$

where m is the number of users that have rated both items a and b. The barred $\bar{r_b}$ is the average of all ratings for item b. Note that the cosine similarity defined in this way is equivalent to the Pearson correlation-based similarity measure (a quick Google search will teach you the potential difference). Subtracting the average rating helps account for the observation that different users use a $1-5$ star rating system in different ways (e.g. some are binary and give everything either a 1 or a 5, some are skewed towards giving higher or lower ratings in general, etc.). A similarity coefficient of 1.0 means users give the same ratings to both items, a value of -1.0 means the users give the opposite ratings to the two items, and a value of 0.0 indicates no relation. The coefficient may need to be adjusted depending on the number of users common to both items. For example, if only one or two users have rated the same item the similarity coefficient may be unreliable given the minimal amount of common data. So we will want to set a threshold value for the minimum number of common users m needed to make a comparison at all. You will also need to handle cases where the denominator is zero.

Assignment (65 functionality, 20 design, 15 writeup, 5 bonus)

The overall goal for the assignment is to compute the similarities of the movies from a large collection of movie ratings. We will be using a dataset of movie ratings from the MovieLens project. There are several datasets available but we will be using the smallest data set of $100,000$ ratings from $1,000$ users for $1,700$ movies. You can find the data at http://files.grouplens.org/datasets/movielens/ml-100k.zip. A complete description of the dataset is available in the `README` http://www.grouplens.org/system/files/ml-100k-README.txt.

You can download and unzip the files directly on the `rice` machines using the following commands in the terminal window (note that anything following the `$` with a space after is the user input):

```
$ wget http://files.grouplens.org/datasets/movielens/ml-100k.zip
...
$ ls
ml-100k.zip
$ unzip ml-100k.zip
Archive:  ml-100k.zip
   creating: ml-100k/
  inflating: ml-100k/allbut.pl
  inflating: ml-100k/mku.sh
  inflating: ml-100k/README
```

```
...
...
$ ls
ml-100k  ml-100k.zip
```

Note that in this assignment you will only be using the `u.data` file. Within that file you will need the first three columns containing the **user id**, **movie id**, and **movie rating** respectively and can ignore the fourth column that contains a timestamp.

Here are the first few lines of `u.data`:

```
$ head -n 5 ml-100k/u.data
196 242 3   881250949
186 302 3   891717742
22  377 1   878887116
244 51  2   880606923
166 346 1   886397596
```

Preparation

This assignment requires that you have completed CME211 HW0. If you have not completed HW0, please do so (even if it is past the HW0 deadline). Part of HW0 is the creation of your homework repository, which will be used throughout the quarter. You will not be able to complete HW2 (or any future CME211 homework) without completing HW0.

These instructions assume that you are logged into `rice.stanford.edu` and have cloned your CME211 homework repository to your Farmshare user directory. First, navigate to your homework repository:

```
$ cd /farmshare/user_data/[sunet_id]/cme211-[github_user]
$ ls
README.md  STUDENT  hw0  hw1
```

In the above `cd` command, `[sunet_id]` must be replaced with your SUNetID and `[github_user]` must be replaced with your GitHub username.

At this point, it is good to check if your local repository is clean and up-to-date with the remote repository on GitHub. This is achieved by running:

```
$ git status
On branch main
Your branch is up-to-date with 'origin/main'.

nothing to commit, working directory clean
```

If you get a different message, make sure to commit (or revert) all modified files or perform a `$ git pull` to retrieve remote changes. Now, create a `hw2` directory (it must be lower case):

```
$ mkdir hw2
$ ls
README.md  STUDENT  hw0  hw1  hw2
```

Now, create a basic `README` inside of `hw2`, commit it to the repository, and push to GitHub:

```
$ nano hw2/README
# add some basic info to the README
# check status of the repo with $ git status
$ git add hw2
$ git commit -m "add hw2 readme"
# ... output omitted ...
```

```
$ git push origin main
# ... output omitted ...
```

All HW2 files must go inside of the `hw2` directory you just created. So, `$ cd hw2` and you are ready to go.

Part 1 (10 points functionality for the test data creation, 5 points writeup)

Use your favorite text editor to manually create a small dataset with around 3 movies and 10 users. Put the data into a file called `test.data` using the same format used for the real data. You can make up a timestamp by using 0 or similar.

Writeup In your HW2 `README`, please briefly answer the following questions:

1. What were your considerations when creating this test data?
2. Were there certain characteristics of the real data and file format that you made sure to capture in your test data?
3. Did you create a reference solution for your test data? If so, how?

Part 2 (55 points functionality, 20 points design, 10 points style)

In the file `similarity.py` write a Python program to find for each movie the one other movie that is most similar (i.e. similarity coefficient closest to 1.0). Note that the most similar movie may have a similarity coefficient less than zero. Also, you might want to think about the case when the denominator is equal to zero. What happens to the numerator in this case and what does it imply for the co-relationship? Your program should obtain the name of the input and output files, and an optional specification of the threshold value, via the command line arguments as shown below. When invoked without the required command line arguments, the program should provide a useful usage message and exit:

```
$ python3 similarity.py
Usage:
  $ python3 similarity.py <data_file> <output_file> [user_thresh (default = 5)]
```

The command line arguments for `similarity.py` are:

- `<data_file>`: input MovieLens data filename
- `<output_file>`: output filename to write similarity scores with format described below
- `[user_thresh (default = 5)]`: optional integer command line argument to specify the minimum number of common users between movies to compute a similarity score. The default value is `5`.

While processing a dataset, the program should summarize what it is doing by printing output **in a format identical to what is shown here** for the MovieLens dataset:

```
$ python3 similarity.py ml-100k/u.data similarities.txt
Input MovieLens file: ml-100k/u.data
Output file for similarity data: similarities.txt
Minimum number of common users: 5
Read 100000 lines with total of 1682 movies and 943 users
Computed similarities in 69.747 seconds
```

Make sure it works on your test data first!

File format Regarding the output file `similarities.txt`. This is a sample excerpt from a `similarities.txt` file showing the results for movie id 40 with no similar movie, movie id 41 that is most similar to movie id 67, and movie id 42 with no similar movie.

```
...
40
41 (67,0.94,7)
```

```
42
...
```

The parentheses enclose the data for the single most similar movie: the movie id of the similar movie, the similarity coefficient for the two movies, and the number of common users that rated both movies. The lines should be in movie id order as shown here. *Note that these are not the real results for the MovieLens 100k dataset and are only shown for describing the file format.* We remark that we desire the output file to be sorted in ascending order of movie-id.

Writeup In your HW2 `README` file, include a command line log of using your program to compute similarities on the `ml-100k/u.data` data file. Also include the first 10 lines of the output similarity file.

Also, in your HW2 `README` briefly explain in no more than one paragraph the decomposition of your program in terms of functions.

Design considerations

A common mistake when developing data structures and code to process something like the MovieLens dataset is that you focus on a solution that only works for the particular characteristics of that particular dataset. For example, you may notice that the movie and user ids in the MovieLens data are consecutive integers starting at 1. If you utilize that assumption in your design you can of course get a program to work but only for datasets with those characteristics. Sometimes it is little to no work to avoid some assumptions and make a more general program that is much more flexible and can accommodate a range of dataset characteristics.

For purposes of this assignment you should not assume that the movie and user ids would always be consecutive integers. This means your program should be designed in such a way that it can efficiently handle the use of any integer movie and user ids, even if the ids are not consecutive and there are large gaps in the numbers.

Developing your programs using functions may make it more concise, easier to develop/test/debug, and result in components of the original code that can easily be reused in other similar programs. **Think about these considerations and implement your program in terms of at least three functions.**

Performance

On the small dataset which you generate by hand, your program should run in (less than) a second. On the `ml-100k` dataset, a reasonable solution can run 30 seconds. We will start considering deductions if your program is more than 5x slower (i.e. takes *several minutes* to run), and we will “kill” programs which run longer for more than 10 minutes.

Modules Disallowed

Please *do not use* any modules from the following set: `numpy`, `pandas`, `tabular`, ,`csv` or `statistics`. These modules facilitate too much of the solution.

Style Guidlines

This homework, we will look for students to follow a few new style guidelines that may not have been called to your attention prior.

1. Modules should be imported in lexicographic (alphabetic) order.

   This is simply so that a reader can quickly find whether you are using a particular module (or not). If the modules are unsorted, we have to inspect each one to answer this curiosity.

2. Each line of code (including comments) shall not exceed 80 characters.

This is simply so that a reader can view your code on a single terminal display. Some of you may have wide screen monitors, but we will start penalizing if lines of code exceed the 80 character margin by a significant amount.

Checklist & submission

The following files must be present in the `hw2` directory of your CME211 GitHub homework repository by the deadline:

- `README`: text document with command line logs and answers to questions from the assignment. You may also call this file `README.txt` or `README.md` (if you want to use Markdown formatting).
- `test.data`: test data file from Part 1
- `similarity.py`: code implemented for Part 2

We will use a Git tag to mark the commit that you want graded. A Git tag is essentially a bookmark to a particular commit. After you have committed the version of the code you would like graded, create a hw2 tag with the following command:

$ git tag -am “HW2 Submission” hw2

The quotes contain a tag message. In the above command hw2 is the actual tag. You can push commits along with tags to the remote repo with:

$ git push –tags origin main

It is a good idea to check if the tags show up on GitHub. These appear in the “release” tab on the web interface.

It is possible to change your tag later, too, if you discover a bug after pushing:

$ git tag -d hw2 # delete the hw2 tag pointing to old commit

$ git tag -am “My HW2” hw2 # create hw2 tag pointing to current commit

$ git push –tags origin main # push changes to code and tags

We will use Git time-stamps on tags and the associated commits to check deadlines. Be careful with overwriting tags – you don’t want to lose the submission tag.

Notes:

- **Do not commit `u.data` or any MovieLens data file to your repository. We will deduct points if extraneous data files are present in your repository.**
- Take care to follow the file and directory names exactly. Everything is case sensitive.
- Please avoid committing other data or temporary files to the repository. We only want to see your `README`, Python code, and manually constructed data.
- Your homework is not complete until you have pushed the above files to GitHub.
- Do regular commits when you have finished little pieces of the puzzle. Don’t do a commit for small individual changes, but also don’t make only one huge commit. There’s no set rule for when to commit. Whenever you feel the need to ‘save’ your (good) progress, think about committing.
- When you do a commit, type a sensible commit message such that your repository’s commit log makes sense to an outsider, for tracking what you have done. This is very important if you work with a team on the same code.