# COMP 120 - Problem Solving Assignment 4

## Due: Friday, April 10 @ 10:00PM PDT

## Required Background:

- Nothing new, although you should review chapter 9 (Objects and Classes) and chapter 14 (Tuples, Sets, and Dictionaries).

## Assignment Overview:

For this assignment you will write a program that implements a particular type of movie recommendation system called collaborative filtering. Collaborative filtering systems work by starting with a large set of users and a large set of movies, and gathering, for each user, the ratings that they give to the movies (from the set of movies) that they have watched. (Each user has rated only a subset of all of the movies.) This set of ratings is then used to "train" the system. The trained system then predicts the rating that a particular user (from the set of users) will give to a particular movie (from the set of movies) that they have not yet rated. A collaborative filtering system uses only its database of ratings to predict other ratings. This is the basic approach taken by, for example, Netflix, for making recommendations to its users.

To measure the effectiveness of a recommendation system, the typical approach is to divide the set of known ratings into two categories – a training set that is used to train the recommendation system, and a test set that is used to test the system. A user/movie pair from the test set (which was not used to train the system) is fed into the recommendation system, which predicts the rating the user would give to the movie. This prediction is then compared to the actual rating the user gave to the movie. The comparison is done for all ratings in the test set, and a statistical measure of the goodness of the recommendation system is computed.

(In 2006, Netflix announced a $1,000,000 prize to the first team that could come up with a recommendation system that beat their in-house system by 10% in a statistical measure of the goodness of the system. Netflix supplied the training set of ratings, and the user/movie pairs of the test set of ratings. There was much interest in the prize, and a team did win in 2009.)

For this assignment, your program will work with real movie ratings (from MovieLens.org) that have been provided to you in your assignment repository.

**Be sure to read this entire document before beginning to work on the assignment.**

## Initial Setup

Both you and your partner will need to get the starter code for your group using Git.

1. You will need your group number in what follows. Get this from the "PSA Group Numbers" file under the "PSAs" tab on Blackboard.,

2. In VS Code, open the command palette and select the "Git Clone" option.

3. When prompted for the repository URL, enter the following, with X replaced by your group number (e.g. 7 or 12).

   ssh://git@code.sandiego.edu/comp120-sp20-psa4-groupX

4. Choose the "Open Repository" option in the window that pops up in the lower-right corner of the screen. The repository should contain the following files:

   - `movie_recommendations.py`: You will put all of the code you write in this file.
   - `dummy_movies.csv`: File containing a small set of dummy movie names.
   - `dummy_training_ratings.csv`: File containing a small set of dummy training ratings for the movies in `dummy_movies.csv`.
   - `dummy_test_ratings.csv`: File containing a small set of dummy test ratings for the movies in `dummy_movies.csv`.
   - `movies.csv`: File containing the movie names.
   - `training_ratings.csv`: File containing training ratings for the movies in `movies.csv`.
   - `test_ratings.csv`: File containing test ratings for the movies in `movies.csv`.
   - `test_psa4A.py`: Code to test your program on the dummy movie files.
   - `test_psa4B.py`: Code to test your program on the actual movie files.

5. Each programming team will have its own repository that has been initialized with the same starter code, so when you sync your code, you are sharing with your partners, but with no one else in the class. (I can see your repository also.)

Remember that if you close VS Code, then when you reopen it, you should see your repository. But if you don't see it, then just select File->Open…, and then select the directory containing your repository.

We also recommend that you stage changes, commit those changes, and sync the changes periodically as you are working on the program, and certainly when you are done with a session with your programming partner. This ensures that you won't lose any of your work in case your computer gets lost or a file gets accidentally deleted.

## The problem

As said in the overview above, you will write code to implement a collaborative filtering movie recommendation system. You will implement the methods whose headers are in the `movie_recommendations.py`, in the repository. The docstring comments following each header describe how the method should work. Do not change the header or the docstring.

1. First, you should understand the contents and format of the data files you will be working with. So in VS Code, open the repository, and then open the files `movies.csv, training_ratings.csv,` and `test_ratings.csv.`

   `movies.csv` contains the names and assigned id's of all of the movies that have been rated – one movie per line. The first line is the header, which indicates that the first field on a line is the id assigned to the movie, the second field is the title of the movie, and the third field is the list of genres associated with the movie. The name of the movie actually includes the year the movie was released. For example, the title (for purposes of this program) of the movie with id 1 is "Toy Story (1995)". The reason for this is that it is not uncommon for different movies to have the same title, and adding the release year to the title helps to distinguish them from each other. The last field of each line is the list of genres that apply to the movie. We won't be using this field in this program.

   `training_ratings.csv` contains the ratings of movies by users. The first line is the header, which indicates that the first field on a line is the id of the user (no names are associated with users, to protect privacy), the second field is the id of a movie, the third field is the rating that the user gave to the movie, and the fourth field is a timestamp associated with the rating (we won't use this field). The movie id's are the same as are assigned in `movies.csv.` The ratings in this file will be used to "train" your recommendation system. The smallest allowable rating is 0.5, and the maximum is 5.0.

   `test_ratings.csv` also contains the ratings of movies by users, but these ratings (which are not included in `training_ratings.csv),` are held back to test the goodness of your recommendation system. Your recommendation system will predict the rating that a user will give to a movie, and that prediction can be compared to their actual rating that is in this file. The format of this file is the same as for `training_ratings.csv.`

   The approach of dividing user ratings into a training set that trains the system, and a test set that tests it, is a feature of collaborative filtering systems.

   As the extensions of the data files suggest, they are csv files, short for comma separated value. This means that commas separate the fields of a line, as you can see by looking at the file contents. But what happens when a field contains a comma? For example, the movie whose id is 11 has the title "American President, The (1995)". Notice that the title contains a comma. How can we distinguish this comma as part of the field, and not a field separator? The answer is that the creator of the file puts the entire field inside double quotes. So notice that for movie id 11, the title is in double quotes, whereas the previous titles are not. Those double quotes indicate that everything inside them is to be treated like a single field, even if a comma is inside.

   This could complicate your reading of the movie file, since you would probably use the split method of the string class to break a line into its fields. But split does not know about the double quote convention describe above. Thankfully, Python has a

module called csv (which must be imported) that makes reading csv files straightforward. Just open the file, and pass the file object to the constructor of a csv reader. Then iterate over the lines of the file:

```
f = open(filename)

csv_reader = csv.reader(f, delimiter = ‘,’, quotechar = ‘”’)

for line in csv_reader:

      ….
```

The difference here is that the line variable in the for loop is already a list of strings. The csv reader, knowing what character separates the fields, separates each line into a list of strings for you.

2. Next, understand the algorithms that you will implement as part of the methods you write.

   First, a key concept is the idea of the similarity between two movies. We'll say that two movies are similar if in general, users who view both movies give similar ratings to them, and two movies are not similar if users who view both movies do not give them similar ratings. We will quantify the similarity of two movies with a number between 0 and 1, with a 0 indicating not similar at all, and a 1 indicating very similar.

   First, if we have two movies, and there are no users who have viewed both movies, then we will say the similarity of the movies is 0. (So we are not going to go down the transitivity road – if A is similar to B, and B is similar to C, then A and C must be similar, even if no one has seen both A and C.)

   Assuming that at least one user has viewed the two movies (call them A and B), here is how you will compute their similarity. First, for all users who have rated both A and B, find the absolute value of the difference between the user's rating of A and their rating for B. Then, compute the average of these differences. The smallest this average can be is 0 (if all users rate A and B exactly the same), and the largest the average can be is 4.5 (since the lowest rating is 0.5 and the highest rating is 5).

   We want the similarity to be 0 when A and B are least similar (when the average difference is 4.5), and 1 when A and B are most similar (when the average is 5.0). So we'll define a simple linear equation that is 0 when the average difference is 4.5 and 1 when the average difference is 0. Here is the equation:

   ```
   similarity = 1 – diff/4.5
   ```

   Here, `diff` is the average difference between ratings.

   So this is how you will compute the similarity between two movies, remember that the similarity is 0 if no one has watched both movies. Remember that it is the ratings in the training ratings file that you will use in computing similarities.

3. Using movie similarities, you will predict the rating of a movie for a particular user. Here is how you will do that: suppose that you want to predict the rating user i will give to movie A. You find all of the movies that user i has rated. For each of these movies, find the similarity that it has with movie A, and multiply that by the rating user i gives that movie. Then add these products up for all movies that A has seen, and divide that by the sum of the similarities of these movies with movie A. This is the predicted rating of movie A by user i. (You can think of this as a weighted average of the ratings of all movies seen by user i, but weighted by the similarities of those movies with movie A).

4. To test the goodness of your movie recommendation system, you will do many rating predictions (for the user/movie combinations in the test ratings files). From these ratings you will create a list of the predicted ratings, plus a corresponding list of the actual ratings (which are also in the test ratings files). So you have two lists that are the same length, and the hope is that the values in the predicted ratings list are close to the values in the actual ratings list. There are different ways to measure how close one list of numbers is to another. One such measure is called the Pearson Correlation Coefficient, which is a number between -1 and 1. 1 indicates a strong correlation between the numbers in the lists, 0 means no correlation at all, and -1 means a strong negative correlation. We hope for a positive correlation between our predictions and the actual ratings. The method `correlation` is already written for you. Just pass in a list of predicted ratings and a list of actual ratings, and it returns the Pearson correlation between them. The method requires a module called scipy, which is already imported, but you may need to install this module. If the module cannot be found, then install it by typing the following in a terminal window (for Windows users, type just python and not python3):

   ```
   python3 -m pip install -U scipy
   ```

5. To help you understand the algorithms described above, you will apply them by hand to a small set of dummy data that is included in your repository. Go to the dummy training ratings file, `dummy_training_ratings.csv`, and make a table (users on the rows, movies on the columns) of the ratings. Some entries will not have a value. Then, using the procedure described above in item 2, compute the similarity between all movies, showing how you compute them. Put the values in a table (movies by movies). Then using the procedure described above in item 3, compute the predicted rating for all empty entries in the ratings table, showing how you compute them. (If you ask for help on this program, I will ask to see this work.)

6. Next, understand the methods that you will need to implement. Open the `movie_recommendations.py` file and study the methods of the `Movie_Recommendations` class and of the `Movie` class that you will need to implement. If anything is unclear, post a question to Piazza (#psa4).

7. Now, you're ready to start coding.

8. First, write the constructors of the `Movie_Recommendations` class and the `Movie` class. The headers and docstrings of the constructors explain what the instance

variables should be, what the instance variables mean, and other comments to help you initialize them correctly.

Note that your program should not compute all movie similarities at the start of your program. There are many combinations of movies. Rather, computer them on an as-needed basis. When you need a similarity, then compute it, and once you've computed it, save it so the next time you need it (if there is a next time), you don't have to recompute it.

9. Next, test the code that you've written so far. Run the program `test_psa4A.py`, and make sure the first two tests say pass. The first should say "Testing movie_dict of Movie_Recommendation constructor" and the second should say "Testing movie_dict of Movie_Recommendation constructor." If anything wrong is detected, you will get a message, and you can go to your code to look for the problem. Make sure those first two tests report passed before proceeding.

10. Next, write the `compute_similarity` method of the `Movie` class. Item 2 above described how to compute the similarity between two movies, and in item 5 you should have hand computed similarities for the dummy data set. The header and docstrings tell you how the method should work.

11. Next, test `compute_similarity` by running `test_psa4A.py`, and this time the first three tests should pass, the last of these three being "Testing compute_similarity method of Movie class". If anything wrong is detected, you will get a message, and you can go to your code to look for the problem. Make sure those first three tests report passed before proceeding.

12. Next, write the `get_similarity` method of the `Movie` class. The header and docstrings tell you how the method should work. Remember, the method should check to see if the similarity has already been computed and saved in the movie dictionary, and if it has been, just return that computed. If it hasn't been, use the `compute_similarity` method to compute it. Before returning, be sure to save the computed value the movie dictionary (for both movies).

13. Next, test `get_similarity` by running `test_psa4A.py`, and this time the first four tests should pass, the last of these three being "Testing get_similarity method of Movie class". If anything wrong is detected, you will get a message, and you can go to your code to look for the problem. Make sure those first four tests report passed before proceeding.

14. Next, write the `predict_rating` method of the `Movie_Recommendation` class. The header and docstrings tell you how the method should work. Item 3 above explains how to predict a rating, and in item 4 above you computed a predicted rating by hand.

15. Next, test `predict_rating` by running `test_psa4A.py`, and this time the next three tests should pass (so the first seven total), these being "Testing predict_rating with valid user id and valid movie id," "Testing predict_rating with valid user id and

valid movie id that the user has already rated," and "Testing predict_rating with invalid user id and invalid movie id." If anything wrong is detected, you will get a message, and you can go to your code to look for the problem. Make sure those first seven tests report passed before proceeding.

16. Next, write the `predict_ratings` method of the `Movie_Recommendation` class. The header and docstrings tell you how the method should work. This involves reading a file containing test ratings, and predicting the rating for each test rating.

17. Next, test `predict_ratings` by running `test_psa4A.py`, and this time the eighth and ninth tests should pass, this being "Testing predict_ratings," and "Testing overall correctness by computing correlation for predictions of test ratings." If anything wrong is detected, you will get a message, and you can go to your code to look for the problem. Make sure that all eight tests report passed before proceeding.

18. Now test your program on the full, real, movie database by running `test_psa4B.py.` You will see that the correlation between the predicted ratings and the real ratings for the user/movie pairs in the test ratings file is 0.47 – not bad for a pretty straightforward recommendation system.

19. (If you have the time and inclination, you can add your ratings of some movies to the ratings file (give yourself a new userid), say 25 or 30 of them, and then you can open a REPL and see predicted ratings for you for movies you have not seen.)

20. Before submitting you should have at least passed all of the tests in `test_psa4A.py,` and hopefully (but not required) all of those in `test_psa4B.py.` There will be up to 10 point penalty if you don't pass all tests in `test_psa4B.py.`

## Requirements

Some of these requirements are mentioned above, but are repeated here so that you can treat this as a checklist to check before you submit.

1. Use descriptive variable names in your program, and use all lowercase letters with underscores separating words. You should appropriately comment your program. It should have a header (this is started for you - be sure to add your names, the date you started it, and a description). Any helper functions you write should have appropriate docstrings with them. (Follow the model of the docstrings given to you.) Up to 5 points off for not following this.

2. You should also comment blocks of code within your functions, explaining what the code is doing. How much commenting to add is a judgement call - you don't want too much, or too little. If you have a block of code (say up to 10 lines long), put a brief comment before it saying what is about to happen. Then put blank lines between the blocks of code. Don't comment individual lines of code, unless they are doing something that the reader might not see right away. Up to 5 points off for not following this.

3. No functions/methods should be longer than 30 lines of code, not counting blank lines and lines with just a comment on them. You can write helper functions if need need to break up the logic of a function or method.

## Pair programming requirement

As described in the syllabus, you should write your program using pair programming. Recall that in pair programming, you and your partner work together at one computer, with one of you typing code (the driver), and the other managing (the navigator). To encourage this from you and your partner, when you are the driver for your team, you should be working on your own computer. When you switch roles, the driver should sync her code to the repository, and her partner should then sync onto his computer, and then become the driver. Remember that you should be switching roles every half hour or so.

So when you follow this approach in writing your program, I should see syncs from both of you, with significant differences between the code synced. If I don't see syncs from all of you, there will be a 10 point penalty on your final grade for the assignment.

## Submission Instructions

**Important**: To be safe, you should run your final code on both you and your partner's computers. This will ensure that you are not relying on any special setup on your own computer for the code to work.

To submit your code, you will need to synchronize it using Git. To make sure your changes are saved and synchronized, follow these steps.

1. Open the "Source Control" menu, either by clicking on the 3rd icon on the left (right under the magnifying glass) *or* by going to "View" and "SCM".

2. Your `movie_recommendations.py` file should show up under the "Changes" section. Click on the "+" icon to the right of the name(s) of the file(s) that you changed to "stage" the changes. This should move the file to a section named "Staged Changes."

3. Commit your changes by typing in a descriptive message (e.g. "finished problem 1") into the "Message" textbox (above the Staged Changes area). After entering the message, click the checkmark icon above the message box to perform the commit. This should remove the changed files from the Staged Changes section.

4. Then, Sync your commit(s) to the server by clicking the "…" (to the right of the checkmark from the last step) and select the "Sync" option. This will likely result in a pop-up notifying you that the sync will do a push and a pull. Select "OK" (or "OK and don't ask again") to complete the sync.

   If you run into problems, check Piazza and ask a new question there if the answer doesn't already exist.

   To make sure that your changes were synced correctly, have your partner do the final step above (namely "…" and then "Sync"). This should fetch the changes you made. You

can then test on their computer to make sure it works exactly the same as on your computer. If your partner has trouble accessing and/or running the file, it is likely that the grader will also have problems and your grade will be negatively impacted.

5. When you have finished the assignment, go to Piazza, and post to the `psa4_submit` folder a message saying that you are ready for grading of psa4.

## Grading

When I grade the problems, I will go down the requirements listed above, and up to 10 points will be deducted for each item where you have not met the requirements.

## Late Penalties

1. If you commit by the due date, no penalty. Else,
2. if you commit by 10PM on Tuesday, April 14, 10 points late penalty. Else,
3. if you sync by 10PM on Friday, April 17, 20 points total late penalty. Else,
4. if you sync by 10PM on Tuesday, April 21, 30 points total late penalty. Else,

## Academic Integrity

Please review the portion of the syllabus that talks about academic integrity with regard to writing programs. In summary, do not share or show your code to any other team in class, and do not turn in any code that you did not write yourself. Don't look at the code from another team, or from any other source. The point of these programs is for you to develop your coding skills.