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:

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

where m is the number of users that have rated both items a and b. The barred $\overline{r_a}$ is the average of all m ratings for item a, and $\overline{r_b}$ is the average for item b. 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

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`.

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.

Part 1

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.

To Turn In: In your 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

In a file called `similarity.py`, write a program that computes, for each movie, the movie in the dataset that it is most similar to (i.e., the movie with a similarity score closest to 1.0). Write the output to a `.csv` file with columns for the base movie id, the movie id it is most similar to, the similarity score, and the number of common ratings between the two movies. If a movie

does not have a match with enough common ratings, then you can set certain entries in the DataFrame to NaN. Your code should roughly look like:

```
def ...

def ...

def compute_similarity(input_file, output_file, user_threshold):
    """
    Function to compute similarity scores

    Arguments
    ---------
    input_file: str, path to input MovieLens file
    output_file: str, path to output .csv
    user_threshold: int, optional argument to specify
    the minimum number of common users between movies
    to compute a similarity score. The default value
    should be 5.
    """

if __name__ == "__main__":
    input_file = "path/to/input"
    output_file = "path/to/output"
    compute_similarity(input_file, output_file)
```

Developing your programs using functions will 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 two functions. These functions are indicated by the `def ...` in the code block above, and they should be called from the `compute_similarity` function, which is what a user would invoke to compute the similarity scores. The block that starts with

```
if __name__ == "__main__":
```

simply says: if this script is run directly, then run this block too. If elements of this script (like functions) are imported by another program, then don't

run this block.

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.

Some hints and guidance:

- The easiest way to write your program is with many nested for-loops, extracting data from the movie DataFrame during each iteration. But your solution will take too long to run. Still, it might help to write this simple-but-slow algorithm first to make sure you understand the task, and test it out on your `test.data`. For your final algorithm, you should aim for something that runs in less than 1 minute on the full dataset.
- When optimizing your algorithm, you may want to modify the way the data is stored. Key-value lookups are very fast, so a dictionary might be helpful.
- Instead of (or in addition to) a dictionary, a helpful data structure might be a matrix (numpy array) with the M rows representing the movies, and N columns representing the users, with the entry at row i and column j representing the rating given to movie i by user j.

To Turn In: Make sure you repository contains the following:

1. `similarity.py` with your algorithm implementation
2. `test.data`
3. A CSV containing first 20 lines from your movie similarity output file. You can create this abridged DataFrame using `df_abridged = df.head(20)`.