Michael Massaro

IS452

Prof. LaLone

Dec. 20, 2018

Final – IMDb File Reading

My final project is about reading in two files, the IMDb ratings file and the IMDb title file, and making two text files from that data: 1) the top 10 rated releases on IMDb with at least 10,000 ratings; 2) the lowest 10 rated releases on IMDb with at least 10,000 ratings.

IMDb input files:

- title.akas.tsv.gz (https://datasets.IMDbws.com/) – This file is used to extract the original title of all the releases on IMDb. There is more data, but it's unnecessary to collect. It is important to settle on the original title because some releases have multiple and we only need one for our output files. It also speeds up the process. Besides the original title of the release, we also must extract the titleId, which is the unique key of the release. This key is used to match a release in this file with the same release in the other input file.
- title.ratings.tsv.gz (https://datasets.IMDbws.com/) – This file is used to extract the average rating and the number of ratings of every release on IMDb. When this file is being extracted, the program only takes in values that have over 10,000 ratings. Like the other input file, there is more data, but it isn't relevant for our output files and not collecting it also helped speed up the process. Both of these files are used to print the original title, average rating, and the number of ratings for the top 10 and lowest 10 rated releases on IMDb. It's also worth mentioning that the input files are refreshed daily, so if you download them yourself and run this program, your results should be different than the results I turned in.

Output files:

- best_10_releases_IMDB.txt – This file contains the top 10 releases with the highest averages and at least 10,000 ratings. This output file contains a descriptive header and then the 10 releases on each line, numbered and in order. For example, it starts with the highest rated release, "1: Koombiyo, average rating: 9.9 with 15176 ratings."
- worst_10_releases_IMDB.txt – This file contains the worst 10 releases with the lowest averages and at least 10,000 ratings. This output file is formatted the same exact way, except it contains the worst releases.

The Process

There are four main processes in this program: 1) Extracting information from ‘title.ratings.tsv.gz’, 2) Extracting information from ‘title.akas.tsv.gz’, 3) Creating ‘best_10_releases_IMDB.txt’, 4) Creating ‘worst_10_releases_IMDB.txt’. Before the process, the program uses import sys to increase the field size limit of the csv import to the max system size. This is an important step because these are two very large input files. Without this step, PyCharm crashes before it can read both files.

1) Extracting information from ‘title.ratings.tsv.gz’ – For my final solution, I ended up using import gzip to unzip and read this tsv file so that I could make a string array filled with a row of three strings, the unique key of the release (tconst), the average rating (averageRating), and the number of ratings (numVotes). To do this, I use an array called temp_row to append onto another array called ratings_rows. The code begins by opening the file with gzip, which allows me to open the file without unzipping it myself. I then use the csv import to make a DictReader with a tab delimiter (because it’s a tsv). After these steps, the for loop goes through every row in the DictReader variable, simply called reader. As mentioned in the input file section, information is added to ratings_rows only if the release has more than 10,000 ratings. To do this, there’s an if-statement in the for loop. However, if the release does have more than 10,000 ratings, the three values are appended to temp_row, then temp_row is appended to ratings_rows. temp_rows is cleared and then the process repeats for the entire input file. This was my final solution, but I attempted many before I came to this. I attempted to understand import pandas, but realized it was complex enough to have entire books written about it, which made me want to find a different solution.

I also tried extracting the information as a dictionary, instead of an array of strings, but this confused me because I never completely understood how DictReader was different from a regular dictionary. In the end, gzip was what made extracting from a zipped file easier than my other methods and import csv's DictReader was the best way to extract values, in the form of rows, with a tab delimiter.

2) Extracting information from 'title.akas.tsv.gz' – This code is very similar to first part, but there are a few differences. Instead of only taking in values with 10,000 ratings using numVotes, we only take in the original title of the releases by using isOriginalTitle (a Boolean value). If isOriginalTitle is equal to 1, then we will repeat the process using temp_row and akas_rows.
3) Creating 'best_10_releases_IMDB.txt' – While extracting the input files takes a long time, just because the IMDb input files are huge, creating these text files takes even longer. This is because this is done with a nested for-loop that loops over every row in both of the new string arrays. Originally, I planned to do a top 100, but I never waited longer than 10 minutes for the file to finish. These nested for loops because faster after I added the if-statements in the first two parts respectively, but 100 still took an unreasonable amount of time. The first part of this process was to merely open the file and then write the header. After this step is when the nested for loop begins. The outer for loop loops through ratings_row, which is important because this is how the program sorts the highest rated releases to the top. To do this, we needed to support by the 2nd value in the row of the string array, which was the average rating. To do this, the program sorts the array in the for loop by using the method "sorted." Sorted has a feature where it lets you sort the list and designate the key. The

key we chose was lambda, which allowed us to pick the sorting index [1] (the 2nd value in the array). We also needed to reverse the sorting to get the highest rated releases (no reverse necessary for the lowest). Lambda was a feature I had no idea about until I came across it on StackOverflow, but it was a solution that worked perfectly. Before using sorted, I tried to sort it using methods I was making myself. I was also attempting to sort it during the extraction phase. I also found another import called numpy, which had a lot of sorting methods for arrays, but like pandas it was confusing to me and I never got it working. After the outer for loop, the inner for loop loops through akas_rows and doesn't need to be sorted. In the inner for loop, there's an if-statement that compares both unique keys of the two string arrays. If they match, the found_accumulator is incremented. The found_accumulator is used to number the ratings in the output file and is also the condition in which the loop breaks. If there was no condition to break the loop, the loop would output every single release on IMDb in order of highest rated (instead of just the top 10) and I'm convinced that it would take multiple days to fully compile. To write to the output file, I create a string (using the title, average rating, and number of ratings) and then write that string to the output file. The outer for loop contains the code that will break from the loop after the found_accumulator found the top 10 releases. The file is then closed.

4) Creating 'worst_10_releases_IMDB.txt' – This part is very similar to part 3. The "sorted' method is used again in the for loop, however there is no need to reverse because it's already in the right order. Besides this, and the slightly different descriptive header, the process is the same.

This project took a lot of work to get it into only four parts. Originally, this program was a lot longer than it is now, however I was able to find useful imports that made the code a lot cleaner. Originally, I had no idea how to extract information from a zipped tsv file; I thought I had to unzip manually and then work from an unzipped file, however this didn't work because of how large the files were. I also messed around with multiple csv methods before I settled on DictReader, including combining that import with itertools. I also originally tried to use DictReader to create a dictionary I could use later, but this didn't work either because of the difference in variable types. In the end, it was easiest for me just to use strings instead of a dictionary. This project took a lot of work, but I'm glad I did it because it seems like a useful skill to know for the type of job I want to have.