Problem 1a) **Marc Gyongyosi**

For both problem a and b I use python and numba (a Python optimizer) to generate the requested information (see prob1.py). Since numba is not installed by default, you will need to install it to run these programs. Using numba, I am able to run the entire sorting in less than a minute for problem a.

The results are:
-mean of number of reviews in common: 18.8079220449
-median of number of reviews in common: 10.0
-the entire graph extends well beyond 300 reviews in common (~340) on the horizontal axis, however, that number of overlap between different users is very rare, thus, to make the general distribution more "visible", I also show one graph zoomed in on the section between 0 and 100 movies reviewed in common, Figure 1. The entire graph is shown in Figure 2.

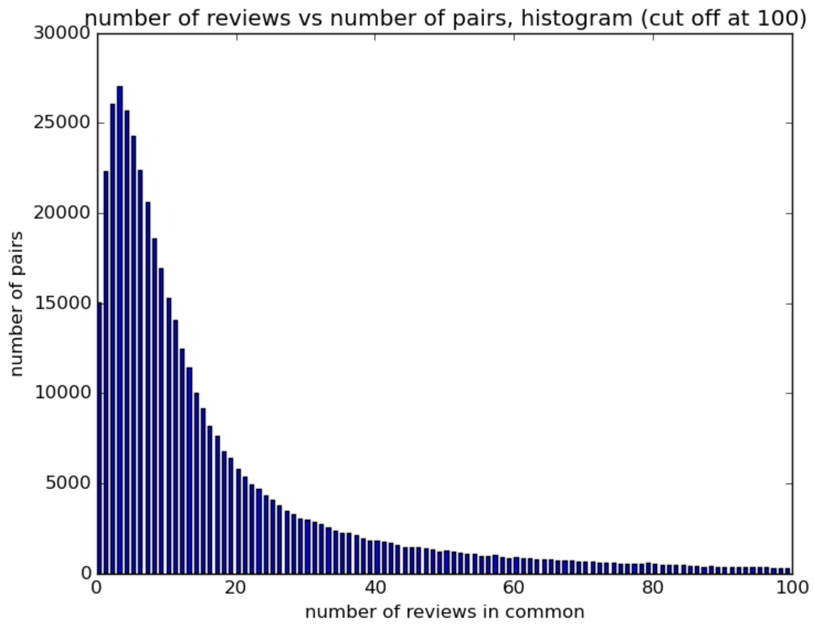

Figure 1

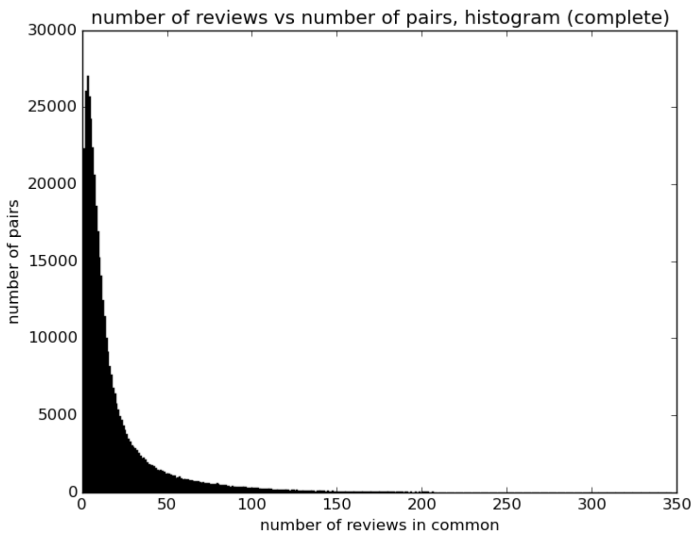

Figure 2

Problem 1b)

Maximum review count: 737 found at index: [404]
Minimum review count: 20 found at index(indices):
[18, 33, 35, 92, 142, 146, 165, 201, 241, 299, 308, 363, 417, 440, 474, 557, 570, 571, 595, 630, 635, 684, 731, 739, 808, 811, 823, 865, 872, 887, 894, 925]

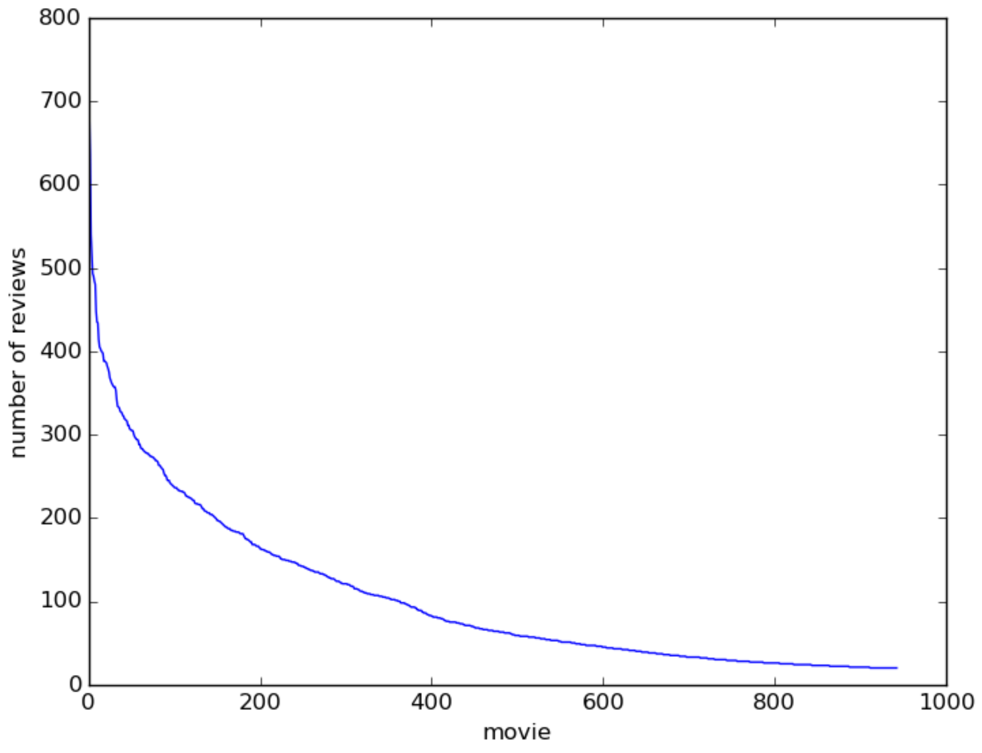

Figure 3

Figure 3 shows this "line". This is similar to most of the examples listed on the Wikipedia page. Thus, I say yes, it does follow Zipf's law.

Problem 2a)
I think approach A is better. This way, we won't skew the results by inserting a value once we do the kNN evaluation for the movie rating. In addition, once we do the kNN, we can treat a 0 as a flag for data that we should not include when averaging. Furthermore, there is no logical reason to insert the average of all rated movies into a spot, since the average of all ratings says absolutely nothing about the rating for a specific movie. For example, consider a child that likes 3 toys. Toy A it likes 5/5, whereas toy B it only likes 1/5. Now, if we don't have data for toy C, then there is absolutely no way we can tell if 3 (the midpoint in this case) would be an appropriate rating for the third toy. For all we know 3 might be the "least off" version, however if we put in three and use this in future predictions, then our predictions will be falsified by this "wrong" data.

Problem 2b)
I conjecture that if all non-existing ratings are replaced with 3s then the Manhattan distance would work properly. I think so because the Manhattan

distance would then measure equally much distance between equally “off” predictions. For the example with the toy: if we had a bunch of kids to sample from and some of them didn’t give a rating for toy C and if we then inserted a “3” into those, then looking at the mode of the k nearest items (the other two toys) as defined by Manhattan distance would give us good results on average since 3 is right in between the min and max rating.

Problem 5)
The collaborative filters are implemented in the two respective python files. To speed things up during testing, I used numba and the @autojit directive on the distance calculations. These are commented out in the provided files so you can run it on your computer, too.

In terms of the ideal number of k nearest neighbors: I got the best results with a value around 15. I believe this is because of the way the data is structured: when looking at problem 1a it is clear that the majority of pairs has approx.. 15 reviews in common. Thus, setting k to 15 allows for maximum overlap and thus the collection of the “most similar” examples.

Problem 6)

a) As an error indicator, I will actually use 2 different measures. To evaluate performance simplistically, I will first measure a mean prediction error (MPE). This mean prediction error will tell me how many I actually got right (either I am right or not, no matter how “close” I am). Then I will use a mean squared error (MSE) to quantize the amount by which the filter is off “on average” on all wrong predictions. Using the MSE and MPE together, I should be able to accurately validate the filter’s performance.

$$MPE = \frac{1}{N}\sum_{u=1}^{N}[prediction \neq truth]$$

$$MSE = \frac{1}{N}\sum_{u=1}^{N}(prediction - truth)^2 \;\; for\ all\ false\ predictions$$

b) I will use the 2 sample t-test on the MSE errors for each sample for each setting. E.g. for c, I will do a 2-sample t test with MSE_d_1 and MSE_d_0. Each of these inputs is of length 50 (the 50 samples) and each of these samples is the average of 100 tests. The null hypothesis for the test (I am using scipy.stats.ttest_ind) is that the two arrays have identical average (expected) values. I will define a 10% threshold to determine whether or not the result is statistically significant.

Note on c,d,e,f: I wrote a python script to test the parameter combinations. You can find it in getTestSets.py. I wasn’t able to compile all the data in time, however, the script writes the results for each run into two text files which are provided with the homework submission (item_cf_results, user_cf_results). It should be very easily understandable what the data in those two files means. Please consider this output of my program for the next 4 problems, even though I didn't have time to formally finish them. If I had had the time, I would have taken the MSE and MPE arrays and handed them to the statistical test.

c) No time to compile the data…. But I have the raw test results.
d) Same, but when looking at it briefly, I can see that setting "i" such that non-ratings are not included in the k nearest neighbors gives the best results. This agrees with my intuition since this way my estimate/prediction is not biased by missing data.
e) I didn't have time to vary k in the entire range, however, from experiments with smaller datasets I have seen that setting k to around 15 (i.e. in this case 16) gives the best results. This is in agreement with the way the provided data is structured: most movies have around 15 reviews in common.
f) Same as c