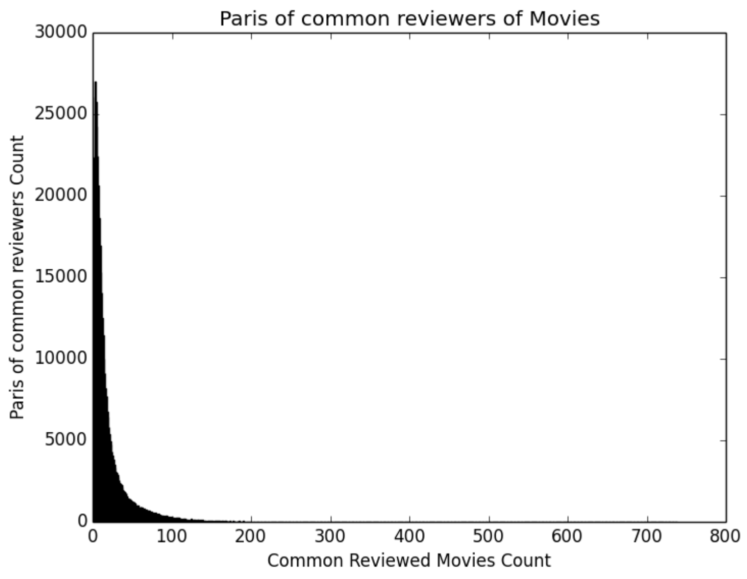
1)

A):
Mean: 18.992745
Median: 10

Paris of common reviewers distribution of each movies (sorted): See chart on the right: (the bar looks narrow since there are 1682 hist-bars on the chart)



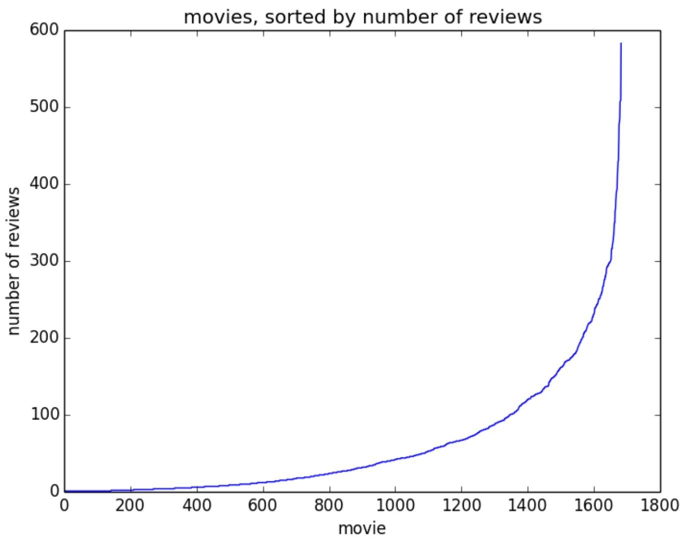
Time complexity for counting pairs is (n^2k), where n is the number of users, k is the number of items, this is lot of time. I calculated them once and stored the output as itemPairs.npy. if you want to see the counting pair function work, follow the comments in viewUsers.py, in function visualizeUsers(). It will took you about 1000 seconds.

To test the script who plot this, use this command:

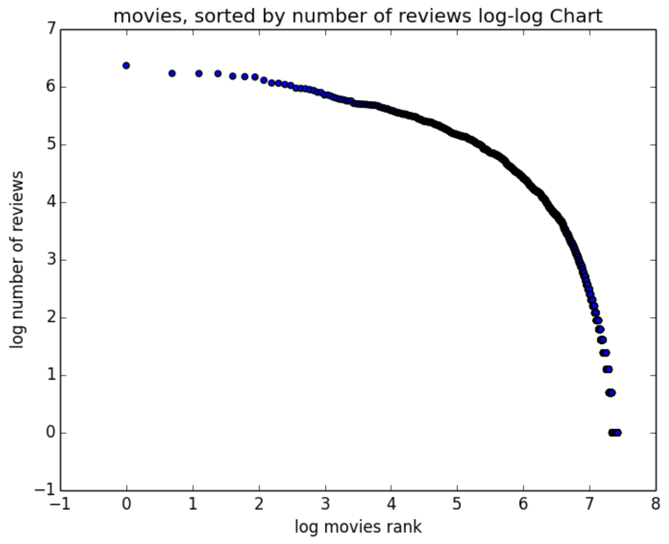
```
~ python viewUsers.py
```

B):
Movie ID who has most reviews 50, it has 583 reviews
Movie ID who has least reviews 599, it has 1 reviews

Number of reviews distribution of each Movies (sorted): See chart on the right:



I also printed the (reversed) log-log chart of movies, as the chart on the right. I guess you can not say that follow zipf's law, since the log-log chart looks not so "linear", which should be the pattern of zipf's law.



To test the script who plot this, use this command:

```
~ python viewItems.py
```

2)

A):

I would prefer plan A for this one, which mean we put a 0 for every missing data.

Consider the following example: we want to find which one is more like Alex: Every name except Alex is followed by 2 number, the first number is the block distance between Alex and the other user using plan A, the other number is calculated with plan B.

	Star War	Harry Potter	The Legend of 1900	Tiny Times	Cloud Atlas
Alex		5			
Bob 16 / 6.25	3	2	5		5
David 15 / 12.5	2		4	1	3
Tom 12 / 8.75	3	4		2	4

Alex only reviewed Harry Potter, and give it 5 score, if we use plan A, we consider other 4 movies as 0, if we use plan B, we will put other 4 movies as the average score, which is 5.

Which choice is more reasonable?

If we apply plan A, which consider all other movies as 0, please notice 0 is not a valid score, so that it doesn't necessarily mean Alex doesn't like all other movie, it mean's Alex didn't review them yet. Then we may say Alex is more similar to Tom, who seems both like Harry Potter.

But if we use plan B, however, we will get a picture that Alex strongly likes all kind of movies equally, and this is hardly true. And we may think Bob is more close to Alex, but Bob only rate Harry Potter 2 points.

Consider in our dataset, one people usually don't review a lot of movie, in other words, the dataset is very sparse. In this case, using average reviews for all missing data seems not a good idea, because it will introducing too much unreasonable information. Sometime we make one choice simply because the other one is worse.

Using plan A, however, we can use the method introduced in question 3, which is called i_index, to remove the influence of some of the missing data. This may make it better a little bit.

B):

Before test, I intuitively prefer Pearson's correlation for this one. Because I think in a sparse dataset like we have, Pearson's correlation seems more reasonable, since it evaluate shape instead of actual value difference. But I'm not too confident about that before test.

Consider the same toy example here. All missing value is be considered as 3. And we try to find out which movies is more like star war.

	Star War	Harry Potter (2.6 / 0.25)	The Legend of 1900 (2.8 / -0.17)	Tiny Times (1.4 / 0.87)	Cloud Atlas (2.4 / 0.52)
Alex	(3)	5	(3)	(3)	(3)
Bob	3	2	5	(3)	5
David	2	(3)	4	1	3
Tom	3	4	(3)	2	4

We print the result on the chart, the first result is calculated with euclidean distance, the second one is calculated with Pearson's correlation.

One unfortunate fact is, both method think Tiny Times is more similar to Star War, which in fact is total opposite of that. This might be caused by the (3) we used to stuff the missing data.

If we leave that, we can see euclidean distance cannot tell much different between The Harry Potter and Cloud Atlas, but Pearson's correlation can say Cloud Atlas is more like Star War. Consider the fact that Both Cloud Atlas and Star War is science fiction movie, we may say Pearson's correlation did a better job here.

3)

item_cf.py and user_cf.py are almost using the exact same code.

The only thing need to do to switch from user-based KNN to item-based KNN is to change column and row of the matrix (using transpose function).

We can reuse manhattanDistance(), pearsonrDistance() and kNNRating() for both type of KNN.

So I put those functions to helper.py, to DRY my code.

Call the function such as:

```
~ python item_cf.py 'ml-100k/u.data' 196 242 0 10 0
```

and:

```
~ python user_cf.py 'ml-100k/u.data' 196 242 0 100 0
```

4)

A):

I will use Mean Squared Error (MSE) to evaluate performance, which is defined as:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y_i} - Y_i)^2$$

Consider following things:

1: The degree of error should be matter for evaluation, since it makes more sense to allow predicting error in here, only the degree of error should matter.

2: I want to amplify the bigger error, for example, predict 5 for actual 1 rating should be considered as "way more wrong" than predict 2 for actual 1, that's why I want to square the error, so that 5 will contribute 16 instead of 4 for MSE, while 2 will contribute 1.

For example, considering the 50 samples * 100 ratings experiment we are going to do:

For every sample, I will calculate the MSE from 100 ratings from that sample.

And then, I will use the Mean value of the 50 as a way to evaluate the performance.

B):

I will first generate a 50 * 100 matrix, containing all the ratings index from each sample, then we store those value to sampleSet.npy.

In every test, we use sampleSet.npy matrix to generate the testing set and training set. To ensure that after we draw the samples, we will keep using the same samples for every test.

Before the test, I intuitively think user based method would be better, considering there are less users than items in our dataset, so we can get a slightly more data for each vector if we use users based method. I'm not very confident about that, however, before the experiment.

For the null hypothesis, I assume each pairs of output is similar, I used a p-value threshold of 0.05 to decide if we can reject this assumption. The p-value of each pair of results is calculated below:

If you want to test how p-value is calculated in follow experiments, go to my HW directory and enter python console:

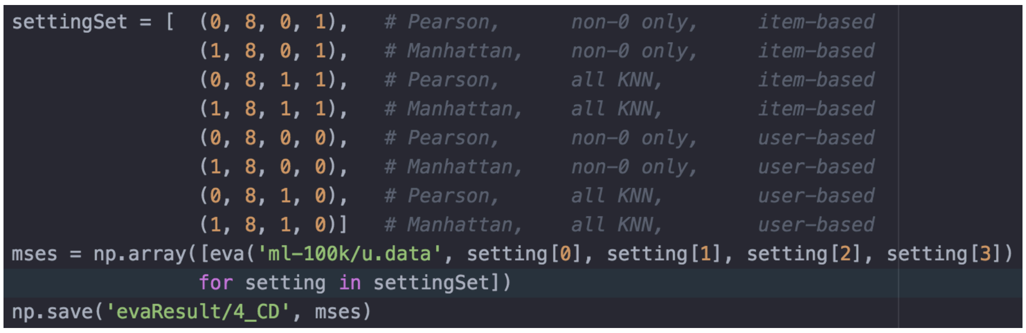
```
~ python
>>> from eva import *
>>> pValue('evaResult/4_CD.npy', 0, 1)
>>> p-value: 3.38077060065e-08
```

This command will read saved data and calculate the p value between 2 columns.

C & D):

For questions 4-C and 4-D, I planned to do 8 tests, each tests will run through 50 samples, calculate the MSE of each of their 100 ratings, and form a box chart to evaluate our result, you can test the eva.py by calling:

~ python eva.py

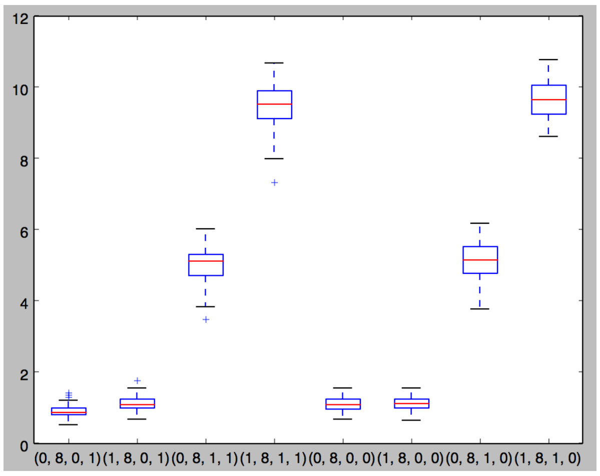


```
settingSet = [  (0, 8, 0, 1),    # Pearson,        non-0 only,      item-based
                (1, 8, 0, 1),    # Manhattan,      non-0 only,      item-based
                (0, 8, 1, 1),    # Pearson,        all KNN,         item-based
                (1, 8, 1, 1),    # Manhattan,      all KNN,         item-based
                (0, 8, 0, 0),    # Pearson,        non-0 only,      user-based
                (1, 8, 0, 0),    # Manhattan,      non-0 only,      user-based
                (0, 8, 1, 0),    # Pearson,        all KNN,         user-based
                (1, 8, 1, 0)]    # Manhattan,      all KNN,         user-based
mses = np.array([eva('ml-100k/u.data', setting[0], setting[1], setting[2], setting[3])
                for setting in settingSet])
np.save('evaResult/4_CD', mses)
```

In those 8 tests, we set k unchanged (otherwise it will be too many). And change the other parameters, to test on both item-based and user-based method. After I calculated the result, which should be organized as 8 * 50 matrix, I store that results to evaResult/4_CD. And we will read those data later to evaluate them.

After I get the result, I use plot4_CD() function defined in eva.py to generate this chart. you can use the comments in last screenshot to map the meaning of the parameters.

From this Chart, we can see that (0, 8, 0, 1) and (1, 8, 0, 0) runs the best results respectively for item-based CL and user-based CL. In both level, using non-0 input only (i = 0) is better than using all KNN. With (i = 0) Pearson's correlation is better than Manhattan distance for item-based CL, and Pearson's correlation is slightly worse than Manhattan distance for user-based CL.



Which means:

For question 4-C:

For item-based CL, Pearson's correlation shows better result than Manhattan distance, comparing column (0, 8, 0, 1) and (1, 8, 0, 1) on the chart. The p-value between those two column is 3.3807706006450175e-08 < 0.05, so we can say that those two methods caused different result towards the input samples.

For question 4-D:

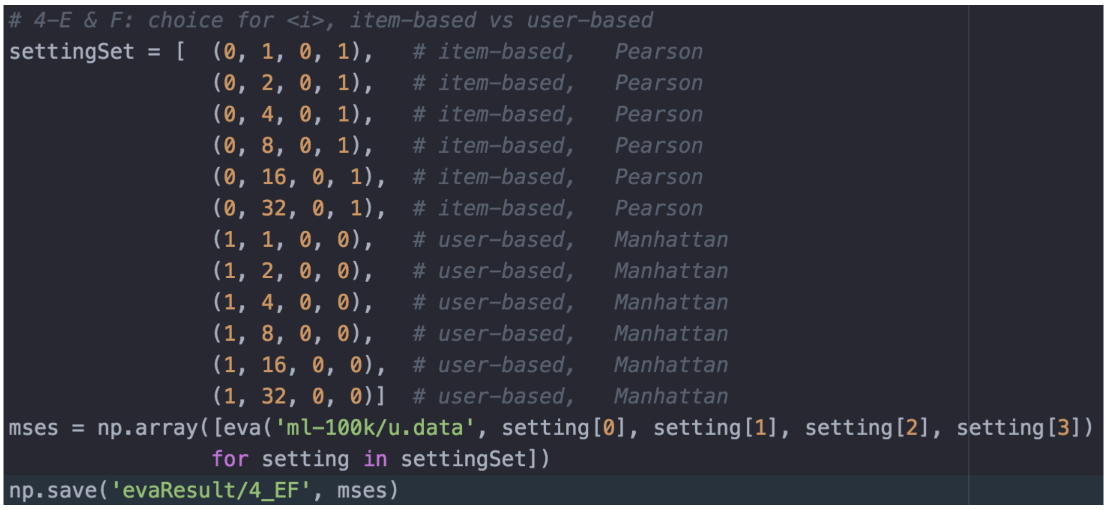
For user-based CL, the result with i = 0 is hugely better than the result with i = 1.

Comparing the last 4 boxes on the chart, (0, 8, 0, 0) and (1, 8, 0, 0) did with i = 0, (0, 8, 1, 0) and (1, 8, 1, 0) did with i = 1.

And the p-value also shows that: p-value 5.4282496870267857e-70 < 0.05, indicating that those two output is hugely influenced by two different methods.

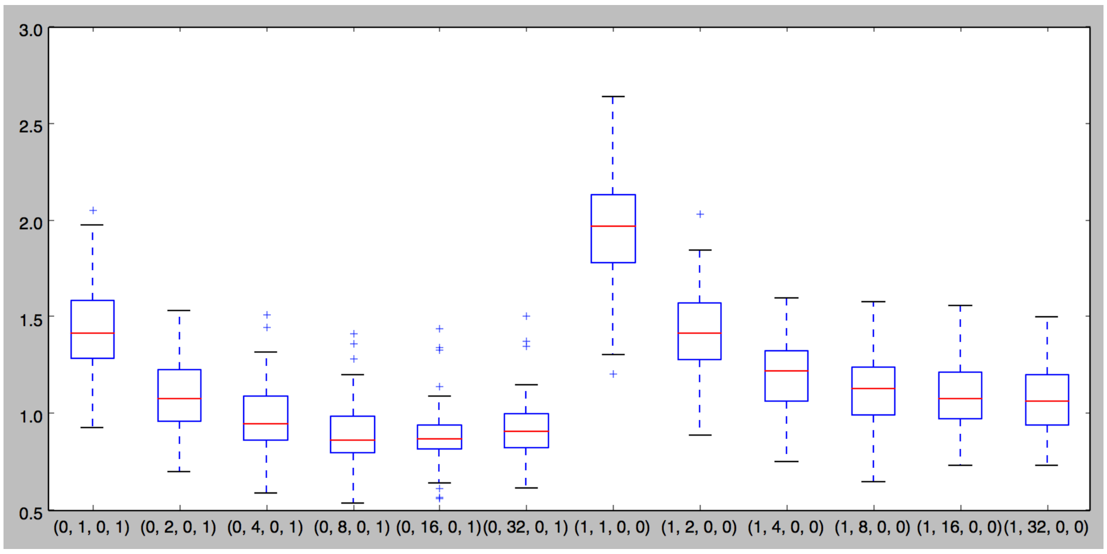
E & F):

With the best <i> and <distance> parameters learned from 4-C & D, I set up 12 experiment for question 4-E & F, trying to figure out which <K> is better for both item-based and user-based CL, similar to previous experiment, I am going to run trough those experiment and store the result to evaResult/4_EF.npy for later evaluation.



```
# 4-E & F: choice for <i>, item-based vs user-based
settingSet = [  (0, 1, 0, 1),   # item-based,   Pearson
                (0, 2, 0, 1),   # item-based,   Pearson
                (0, 4, 0, 1),   # item-based,   Pearson
                (0, 8, 0, 1),   # item-based,   Pearson
                (0, 16, 0, 1),  # item-based,   Pearson
                (0, 32, 0, 1),  # item-based,   Pearson
                (1, 1, 0, 0),   # user-based,   Manhattan
                (1, 2, 0, 0),   # user-based,   Manhattan
                (1, 4, 0, 0),   # user-based,   Manhattan
                (1, 8, 0, 0),   # user-based,   Manhattan
                (1, 16, 0, 0),  # user-based,   Manhattan
                (1, 32, 0, 0)]  # user-based,   Manhattan
mses = np.array([eva('ml-100k/u.data', setting[0], setting[1], setting[2], setting[3])
                for setting in settingSet])
np.save('evaResult/4_EF', mses)
```

The result is shown as follow: the 6 chart on the left is item-based CL, the 6 chart on the right is user-based CL.



Based on the result, we can get the answer for question 4 E & F

For question 4-E:

Item-based CL get the best average result (MSE) when k = 8, the second best result is when k = 16, however, two column has p-value = 0.8512855645846934, indicating k = 16 or k = 8 actually makes not too much different.

The user-based CL get best average result when k = 32. The p-value bewtten k = 32 and k = 16 (second best for user-based CL) is 0.79451818537111452, indicating those two method are actually similar.

For question 4-F:

Item-based CL is slightly better than user-based CL. This is different with my assumption. But I am still cannot be too confident about this result, considering the data is sparse and the data set is small, and we didn't do cross validation for that either.

The p-value between column(0, 8, 0, 1) and column(1, 32, 0, 0) is 1.5426224237688518e-06 < 0.05, indicating that this two method is actually making significant different.

If you want to test how eva.py works, go to my HW directory and call

```
~ python eva.py
```

They will calculate the result and save the result as .npy document for later evaluation, you can change the parameters you want to compare and storage place in batchEva(), which can be found in eva.py

You can enter the python console and plot the box_chart used to illustrate my answer by calling plot4_CD() and plot4_EF(), which will use the stored data and plot the box chart for the results coming from different methods.

```
~ python
>>> from eva import *
>>> plot4_CD()
>>> plot4_EF()
```

Please notice: The experiments in question 4 is based on the result using average k value (instead of mode k value), because I think it makes more sense at the beginning, please see more in helper.py. Which described how prediction is made. I changed it back after the experiment, since question 3 is asking for mode value. But for question four, even though there are might be slightly different with the results of other students, but there would not be conceptual difference. So please don't grade me down for this.

If you want to see how it goes with mode value for question four, the eva.py is fully commented which can guide you to run the data with mode knn and generate a new data. Running through those data in my machine cost about 4 hours, so I didn't do that again.