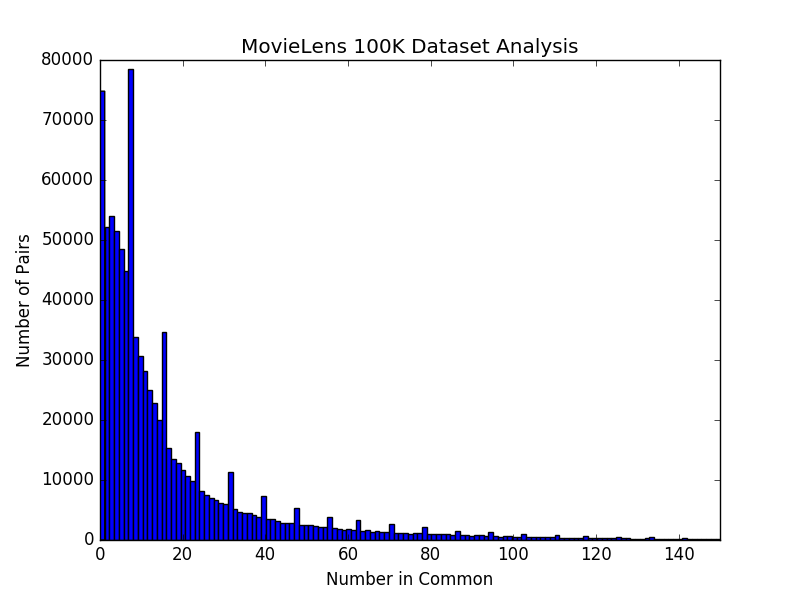
Sonia Nigam

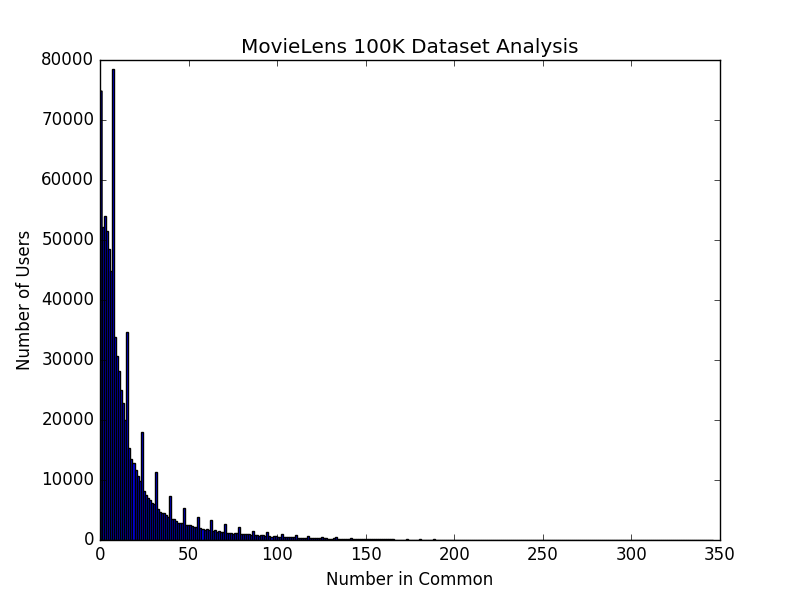
Assignment 4

**Problem 1**

**A)** Mean number of common reviewed movies: 18.8079220449

Median number of common reviewed movies: 10.0





The above numbers and graphs were generated from code found in the file understanding\_data.py. To run this file, enter in command line: “python understanding\_data.py u.data”.

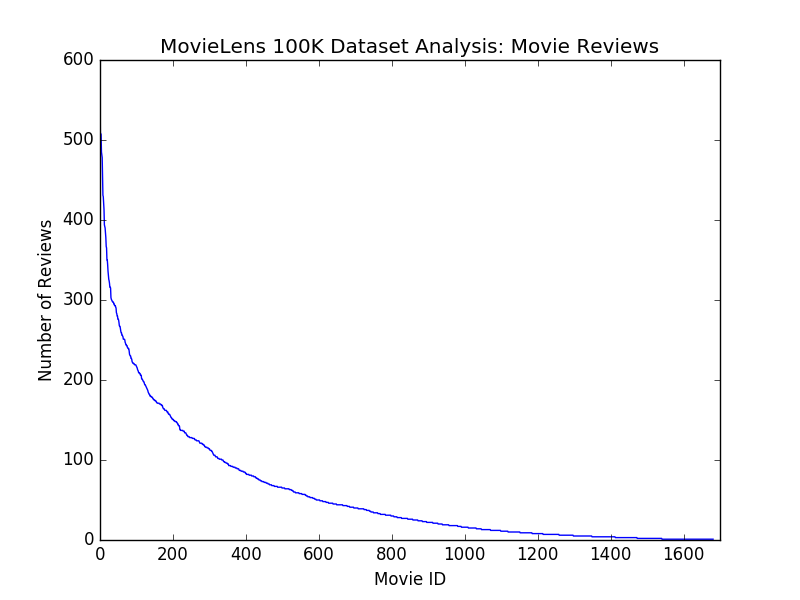
I created two histograms. The first histogram constrains the x-axis such that it focuses on the concentrated lower end of the distribution. I did this because I think it creates a more telling visual of the data, and is easier to read. The second histogram has an x-axis that goes up to 350, because there are 346 unique common quantities, and this graph covers the entire distribution. Because the right end of the distribution consists of outliers, where one given pair had a high number of common reviews, it is hard to see this data representation on the large-scale graph. That being said, it does more accurately represent the entire data representation, so I thought it was important to include it.

When designing both histograms, I set it such that the number of bins is the same as the number of individual common quantities. Meaning, each bar represents a distinct number of common movie reviews between a pair of users, and the height of the bar represents how many pairs of users had that same common quantity. I did this such that no quantities were aggregated, and that the histogram would accurately represent the distribution.

**B)** 141 movies have the minimum number of reviews (1 review). 1 movie has the maximum number of reviews (583). I think this distribution follows the general notion of Zipf’s law. There are 1,682 movies, thus the film with the maximum number of reviews is ranked 1,682nd in the frequency table. According to the law, there should be 1,682 times more movies ranked first in the frequency table. In this case, there are 141 films with the minimum number of reviews, and only one film with the maximum number of reviews. Thus, this data distribution follows the skewed trend where higher ranked terms (more reviews) have fewer occurrences (fewer movies), but does not follow the law to the exact numerical definition. This trend can be very clearly seen in the graph below.

Movie 50 had the maximum number of reviews with: 583 reviews

The following movies had the minimum number of reviews with: 1 review

['1653', '1655', '1621', '1627', '1626', '1625', '1624', '1579', '1343', '1507', '1505', '1618', '1619', '1616', '1614', '1613', '1363', '1543', '1548', '1601', '1603', '1604', '1606', '1156', '1557', '1525', '1526', '814', '1492', '1493', '1494', '1498', '1122', '1235', '1236', '1482', '1486', '1130', '1572', '1571', '1570', '1577', '1576', '1575', '1574', '830', '1582', '1583', '1580', '1581', '1586', '1373', '1584', '1457', '1458', '1587', '1595', '1596', '1593', '1599', '1447', '1452', '1453', '857', '852', '1476', '1309', '1559', '1681', '1680', '1682', '1461', '1460', '1310', '1533', '1536', '1678', '1679', '1674', '1675', '1676', '1677', '1670', '1671', '1673', '677', '1414', '1320', '1325', '1329', '1520', '711', '1669', '1668', '1667', '1666', '1665', '1663', '1661', '1660', '599', '1546', '1339', '1352', '1659', '1650', '1651', '1657', '1654', '1633', '1349', '1348', '1340', '1341', '1645', '1647', '1641', '1640', '1649', '1648', '1515', '1510', '1563', '1564', '1566', '1567', '1630', '1632', '1634', '1635', '1636', '1637', '1638', '1366', '1568', '1569', '1561', '1562', '1565', '1201', '1364'] 

The above numbers and graphs were generated from code found in the file understanding\_data.py. To run this file, enter in command line: “python understanding\_data.py u.data”.

**Problem Two**

**A)** I think approach A is a more suitable solution to the issue posed by missing reviews. Users are only allowed to rate a movie from 1-5. Meaning, if the rating is set to be 0, it clearly represents a missing rating. In turn, if a user has rated a movie and another user has not, Manhattan Distance will always appropriately count this difference when measuring the distance between two users. With approach B, a missing rating will be substituted with the mean. There is a chance that this average rating is the same number as the rating of the user who actually rated the film. In turn, Manhattan Distance would mistakenly consider this as a similarity between two users, when in reality that does not accurately represent how they viewed the film. Approach B does not correctly distinguish unrated films from the pool of data, and in turn could consider erroneous similarities. Moreover, it is important to note when users have both chosen not to rate a given film, as it is telling of their profile. Approach A will correctly account for when users have both not rated a given movie, for they will both be 0.

For example, imagine the following two users rating 5 movies.

1) 1 1 3 1

2) 1 1

In this scenario, we are going to compare user 1 and user 2. With Approach A, user1 and user2’s missing reviews would be substituted with 0s and would appear like below.

1) 1 1 3 1 0

2) 1 0 1 0 0

In turn, the Manhattan Distance would be 1+2+1 = 3. This correctly captures the similarity that they did not review the fifth film. It also captures two key differences where user1 reviewed the film and user2 did not.

With Approach B, user1 and user2’s missing reviews would be substituted with the mean rating, and would appear like below.

1) 1 1 3 1 1.5

2) 1 1 1 1 1

In turn, the Manhattan Distance would be 2+.5 = 2.5. Because user2’s substituted mean rating matched user1’s rating in both cases, the algorithm interpreted these comparisons as two similarities and did not account for them in the distance. This method does not clearly distinguish unrated films, and the means of substitution can be mistaken as a matched rating. Moreover, both users were missing the 5th review, and had different substituted means. Though it truly was a similarity that they did not review a given film, this was counted as a difference within the distance.

In conclusion, Approach A ensures that unrated films have a unique numeric value; as a result, there cannot be false matches between user ratings when one user’s substitution matches the other’s true rating, and false mismatches when two users have different substituted means. As a result, this is a more accurate approach to calculating Manhattan Distance.

**B)** I think Pearson Correlation would be better for item-based collaborative filtering. Euclidean distance is helpful when you want to know the actual numeric distance between two given objects. However, in this scenario we simply want to know how similar two movies are to one another. As a result, Pearson Correlation would provide a better indicator, as it holistically analyzes trends, and produces a similarity measure. Moreover, Pearson Correlation is a great distance measures when comparing high dimensional vectors. Because each vector consists of 943 elements, Pearson Correlation would be a helpful approach.

\*\***ADD IN THE TOY EXAMPLE\*\***

**Problem 3**

Fully implemented in user\_cf.py and item\_cf.py files. They can be run using the below format:

python user\_cf.py <datafile> <userID> <movieID> <distance> <k> <i>

python item\_cf.py <datafile> <userID> <movieID> <distance> <k> <i>

**Problem 4**

**A)** I chose to use MSE (mean squared error) to assess the accuracy of a given variant. MSE takes the difference between the actual rating and predicted rating, squares it, and then averages these values across the entire sample. Meaning, if you have a sample of 50 draws, you would take the first draw, find the squared difference between the actual and predicted, and then do the same for the following 49 samples. Once these squared differences are all summed, one can divide them by 50 and find the MSE across the entire sample. In a more formulaic form, for n draws in a given sample:

MSEsample = 2

I thought it was important to not just capture how often a variant is incorrect, but to find the extent to which a variant is incorrect. Because ratings can be a range of numbers, not a binary system, one variant may be incorrect more often, but on average makes closer predictions. For example, variant1 could get a prediction right 50% of the time, but the other 50% it is off by 5 points. On the other hand, variant2 could only be right 40% of the time, but the other 60% it is only off by 1 point. MSE would be able to recognize that variant2, on average, returns closer results, even if it is not completely correct as often as variant1.

**B)** Null Hypothesis: There is no significant difference between the two variants’ error measures.

**C)** The following parameters were kept constant across the two comparisons:

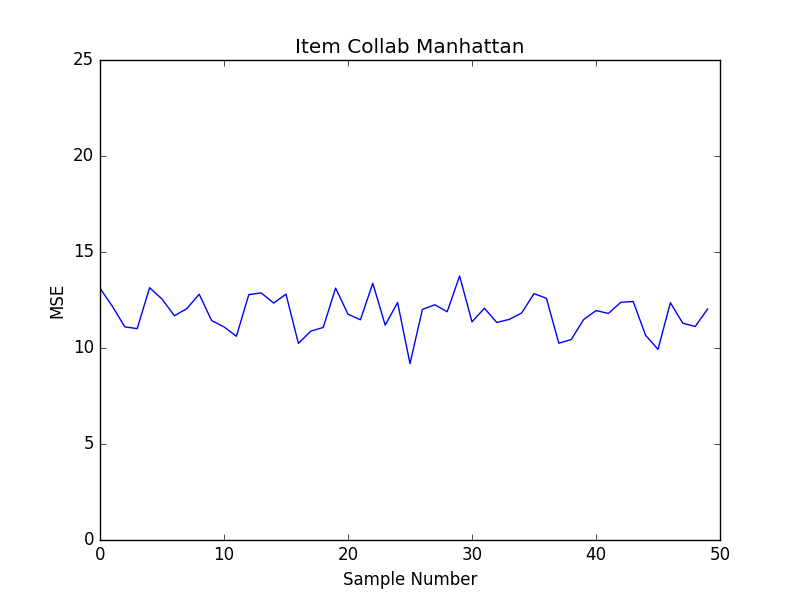
datafile = u.data (samples are serialized), k = 5, iFlag = 1

iFlag was set to 1 for both variants so that the eligible dataset members were kept constant, and the entire testing set could be used within each variant. I thought allowing all data members to be used within the variant, including those with missing reviews, would provide more accurate, raw results of each approach. Filtering the data skews it such that it always has neighbors with reviews, and doesn’t necessarily represent a variant’s effectiveness with the entire, unfiltered dataset. I also chose to do k=5 neighbors, because I hoped to provide a sufficient sampling of “similar” data points, while maintaining computational speed. I did not want to overfit my data and choose too low of a k, but I also did not want to extend the runtime or underfit the dataset.

**Item Collab where Distance =1 (Manhattan):**

This experiment was conducted by running: python item\_experiment.py u.data 1 5 1

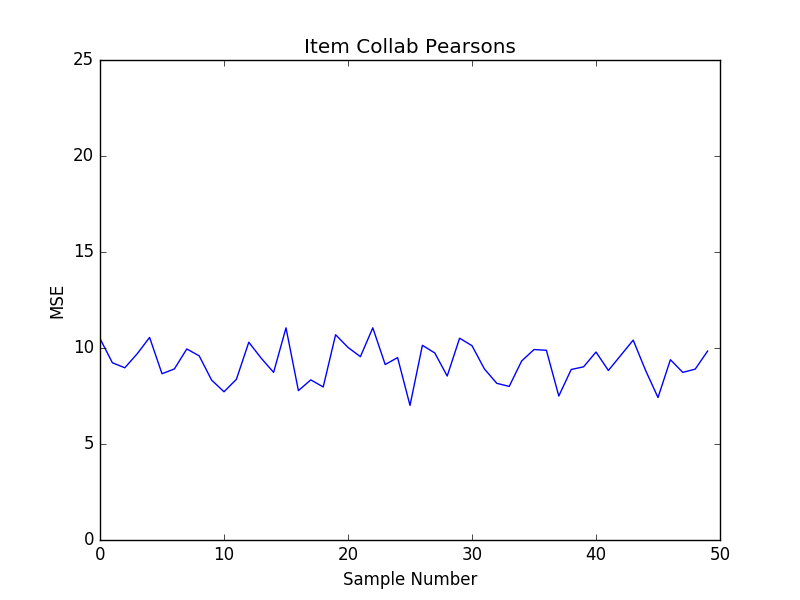
The average MSE across the 50 samples was **11.794**.



**Item Collab where Distance = 0 (Pearson):**

This experiment was conducted by running: python item\_experiment.py u.data 0 5 1

The average MSE across the 50 samples was **9.2296**. In the figure below, each individual sample’s MSE is plotted against the iteration number. As you can see, the MSE per sample was pretty steady, and there were not any extreme outliers that skewed the overarching average MSE. In turn, I think this average MSE seems to be quite representative of this variant.



**D)** The following parameters were kept constant across the two comparisons:

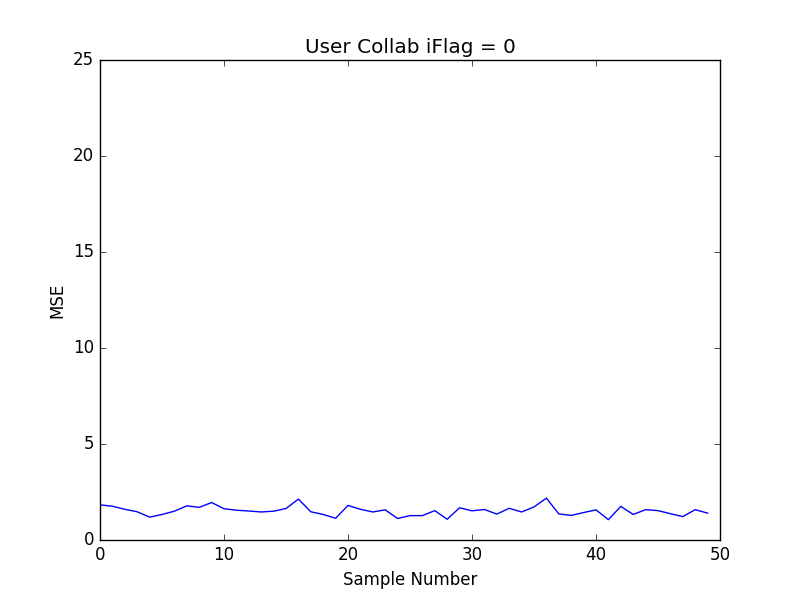
datafile = u.data (samples are serialized), distance = 0 (Pearson’s), k = 5

Pearson’s distance was used for both variants so that the distance measure was kept constant, and vectors were compared in the same way. If the distance was Pearson for one, and Manhattan for the other, results may be skewed depending on which measure is better suited for the dataset. Computing two vector’s Pearson’s relationship is faster than finding Manhattan distance. Because each test required 50\*100 = 5,000 iterations, I chose to expedite it in this way. Also, since it’s highly dimensional data, I assume these will return better results, though that is not grounded in tangible evidence yet. I also chose to do k=5 neighbors, because I hoped to provide a sufficient sampling of “similar” data points, while maintaining computational speed. I did not want to overfit my data and choose too low of a k, but I also did not want to extend the runtime or underfit the dataset.

**User Collab where iFlag = 0:**

The experiment was conducted by running: python experiment.py u.data 0 5 0

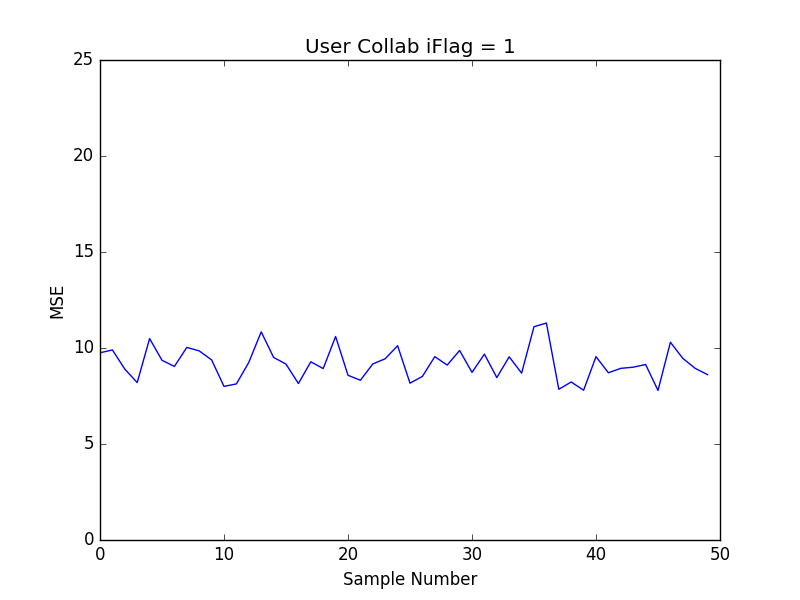
The average MSE across the 50 samples was **1.5154**. In the figure below, each individual sample’s MSE is plotted against the iteration number. As you can see, the MSE per sample was pretty steady, and there were not any extreme outliers that skewed the overarching average MSE. In turn, I think this average MSE seems to be quite representative of this variant.



**User Collab where iFlag = 1:**

The experiment was conducted by running: python experiment.py u.data 0 5 1

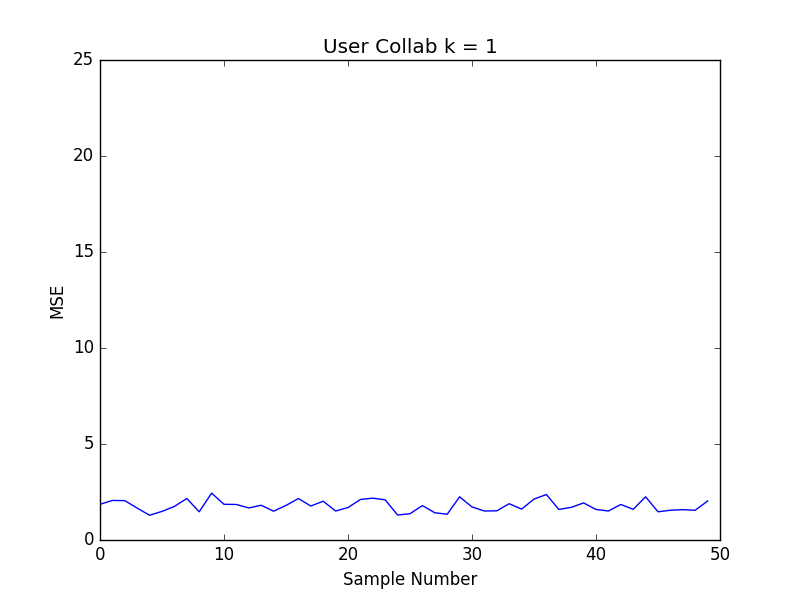
The average MSE across the 50 samples was **9.1886.**



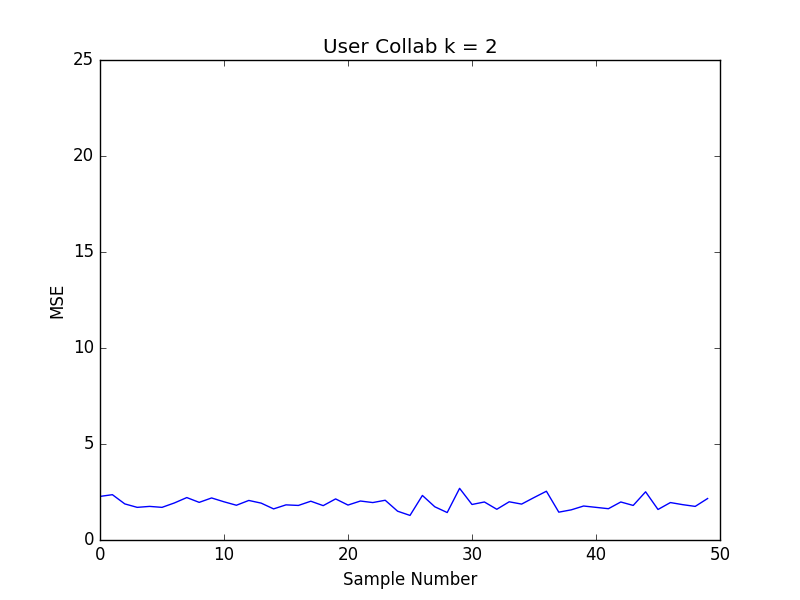
**D)** Based on the above tests, the best settings are: distance = 0 (Pearson’s), iFlag = 0

**User Collab**

K=1: the average MSE across the 50 samples was: **1.7734**



K=2: the average MSE across the 50 samples was: **1.9098**



K=4: the average MSE across the 50 samples was:

**Item Collab**

K=1: the average MSE across the 50 samples was: 1.4004