Andrew Floyd

CS3001: Intro to Data Science

Dr. Fu

December 2nd, 2018

HW8 - RecSys

Github repo: *https://github.com/afzm4/cs3001hw8*

1. Here is a screenshot of me installing scikit-surprise (:))

```
(base) C:\Users\Andrew>conda install -c conda-forge scikit-surprise
Solving environment: done

## Package Plan ##

  environment location: C:\Users\Andrew\Anaconda3

  added / updated specs:
    - scikit-surprise


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    certifi-2018.4.16          |           py36_0         143 KB  conda-forge
    conda-4.5.11               |        py36_1000         655 KB  conda-forge
    scikit-surprise-1.0.6      |py36h452e1ab_1001         590 KB  conda-forge
    joblib-0.13.0              |            py_0          179 KB  conda-forge
    ------------------------------------------------------------
                                           Total:         1.5 MB

The following NEW packages will be INSTALLED:

    joblib:          0.13.0-py_0            conda-forge
    scikit-surprise: 1.0.6-py36h452e1ab_1001 conda-forge

The following packages will be UPDATED:

    certifi:         2018.4.16-py36_0                    --> 2018.4.16-py36_0 conda-forge
```

2. I then proceeded to download ‘restaurant_ratings.txt’ from Canvas.
3. Here I imported the dataset and using Reader and Dataset from surprise along with the os package for the process. Here is the code I used.

```
#Loading data from a file into a dataset
file_path = os.path.expanduser('restaurant_ratings.txt')
reader = Reader(line_format='user item rating timestamp',sep='\t')
data = Dataset.load_from_file(file_path=file_path, reader=reader)
```

4. MAE stands for Mean Absolute Error, which is a measure of distance between two continuous variables. Here is the formula:

$$\mathrm{ME} = \frac{\sum_{i=1}^{n} y_i - x_i}{n}$$

RMSE stands for Root Mean Squared Error and is a frequently used measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed. Here is the formula:

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{t=1}^{T} (\hat{y}_t - y_t)^2}{T}}$$

5. This step involves splitting the data into 3 folds using cross-validation and then computing the MAE and RMSE using the SVD algorithm. Below are the results and the code used:

```
Evaluating RMSE, MAE of algorithm SVD.

------------
Fold 1
RMSE: 0.9462
MAE:  0.7467
------------
Fold 2
RMSE: 0.9426
MAE:  0.7433
------------
Fold 3
RMSE: 0.9463
MAE:  0.7476
------------
------------
Mean RMSE: 0.9450
Mean MAE : 0.7459
------------
------------
        Fold 1  Fold 2  Fold 3  Mean
RMSE    0.9462  0.9426  0.9463  0.9450
MAE     0.7467  0.7433  0.7476  0.7459
```

```
#SVD Algorithm with MAE and RMSE output
algo = SVD()
perf = evaluate(algo, data, measures=['RMSE','MAE'])
print("SVD Algorithm:")
print()
print_perf(perf)
print()
```

6. Here we are running the same idea as the last step but with the PMF algorithm instead. Here are the results and the code:

```
------------
Fold 1
RMSE: 0.9620
MAE:  0.7619
------------
Fold 2
RMSE: 0.9698
MAE:  0.7650
------------
Fold 3
RMSE: 0.9647
MAE:  0.7590
------------
------------
Mean RMSE: 0.9655
Mean MAE : 0.7620
------------
------------
        Fold 1  Fold 2  Fold 3  Mean
RMSE    0.9620  0.9698  0.9647  0.9655
MAE     0.7619  0.7650  0.7590  0.7620
```

```python
#PMF Algorithm with MAE and RMSE output (PMF is the same as unbiased SVD)
algo = SVD(biased=False)
perf = evaluate(algo, data, measures=['RMSE','MAE'])
print("PMF Algorithm:")
print()
print_perf(perf)
print()
```

7. For this step, we are once again repeating the process, but using the NMF algorithm instead. Below is the code and the results:

```
Evaluating RMSE, MAE of algorithm NMF.

------------
Fold 1
RMSE: 0.9700
MAE:  0.7609
------------
Fold 2
RMSE: 0.9737
MAE:  0.7638
------------
Fold 3
RMSE: 0.9774
MAE:  0.7691
------------
------------
Mean RMSE: 0.9737
Mean MAE : 0.7646
------------
------------
        Fold 1  Fold 2  Fold 3  Mean
RMSE    0.9700  0.9737  0.9774  0.9737
MAE     0.7609  0.7638  0.7691  0.7646
```

```python
#NMF Algorithm with MAE and RMSE output
algo = NMF()
perf = evaluate(algo, data, measures=['RMSE','MAE'])
print("NMF Algorithm: ")
print()
print_perf(perf)
print()
```

8. Once again, we run the evaluate function, this time using the User-Based Collaborative Filtering algorithm (using KNNBasic with ‘user_based’ set to True). Below are the results and the code:

```
------------
Fold 1
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 0.9939
MAE:  0.7859
------------
Fold 2
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 0.9802
MAE:  0.7748
------------
Fold 3
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 0.9903
MAE:  0.7821
------------
------------
Mean RMSE: 0.9881
Mean MAE : 0.7809
------------
------------
        Fold 1  Fold 2  Fold 3  Mean
RMSE    0.9939  0.9802  0.9903  0.9881
MAE     0.7859  0.7748  0.7821  0.7809
```

```
#User-Based Collaborative Filtering Algorithm with MAE and RMSE output
algo = KNNBasic(sim_options={'user_based':True})
perf = evaluate(algo, data, measures=['RMSE','MAE'])
print("User-Based Collaborative Filtering Algorithm: ")
print()
print_perf(perf)
print()
```

9. This time we used the User-Based Collaborative Filtering algorithm (using KNNBasic with ‘user_based’ set to False). Below are the results and the code:

```
------------
Fold 1
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 0.9922
MAE:  0.7885
------------
Fold 2
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 0.9804
MAE:  0.7746
------------
Fold 3
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 0.9854
MAE:  0.7795
------------
------------
Mean RMSE: 0.9860
Mean MAE : 0.7809
------------
------------
        Fold 1  Fold 2  Fold 3  Mean
RMSE    0.9922  0.9804  0.9854  0.9860
MAE     0.7885  0.7746  0.7795  0.7809
```

```python
#User-Based Collaborative Filtering Algorithm with Cosine, MSD and Pearson Similarities
algo = KNNBasic(sim_options={'name':'MSD','user_based':True})
perf = evaluate(algo, data, measures=['RMSE','MAE'])
print("User-Based Collaborative Filtering Algorithm (w/ MSD): ")
print()
print_perf(perf)
print()
```

10.

For these next three questions, we are comparing the five methods used in the last five steps on each fold. I made sure to use the same folds' for all five methods which can be seen below. The same code is used as above. Looking at the fold 1, it appears that SVD is the best while IBCF is the worst.

```
SVD Algorithm:

        Fold 1  Fold 2  Fold 3  Mean
RMSE    0.9498  0.9430  0.9441  0.9456
MAE     0.7498  0.7433  0.7459  0.7463
```

```
PMF Algorithm:

        Fold 1  Fold 2  Fold 3  Mean
RMSE    0.9657  0.9613  0.9649  0.9640
MAE     0.7627  0.7575  0.7623  0.7608
```

```
NMF Algorithm:

        Fold 1  Fold 2  Fold 3  Mean
RMSE    0.9781  0.9697  0.9742  0.9740
MAE     0.7692  0.7598  0.7661  0.7650
```

```
User-Based Collaborative Filtering Algorithm:

        Fold 1  Fold 2  Fold 3  Mean
RMSE    0.9922  0.9853  0.9868  0.9881
MAE     0.7852  0.7784  0.7803  0.7813
```

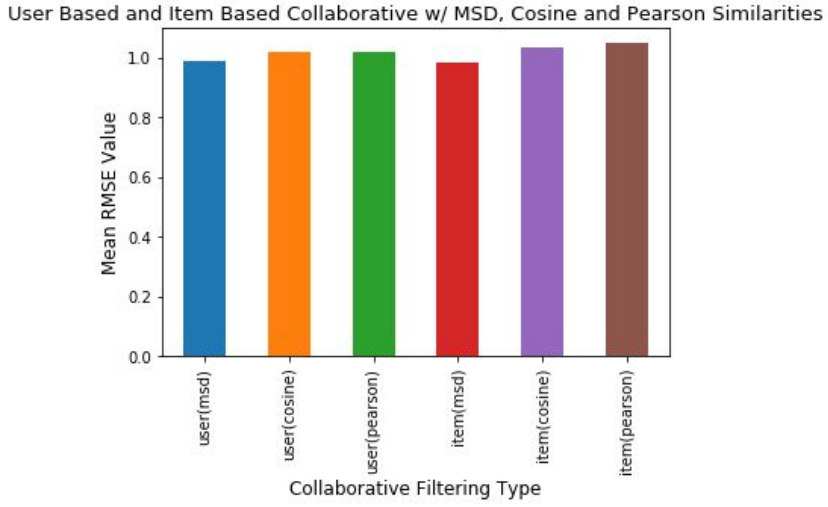
```
Item-Based Collaborative Filtering Algorithm:

        Fold 1  Fold 2  Fold 3  Mean
RMSE    0.9907  0.9794  0.9857  0.9853
MAE     0.7846  0.7745  0.7818  0.7803
```

11. Looking at fold 2, we see similar results but with UBCF as the worst while SVD is still has the best results.
12. When looking at fold 3, we see the same results as fold 2 (SVD best, UBCF worst).
13. It then makes sense, based on the last three steps, that the best mean results belong to SVD, followed by PMF, NMF, IBCF and then UBCF.
14. Here we comparing how MSD, Cosine and Pearson similarities affect User-Based Collaborative Filtering and Item-Based Collaborative Filtering. Below are the results and some code (to see all code, go to my github link at the top of the report). A plot is also included to show the results. Looking at our info below, it appears that MSD has the most positive effect on the algorithms, while cosine and pearson seem to have less of an impact. Also, based on the plot below, it appears that the impact of the three metrics seem to be fairly consistent between both algorithms.

```
User-Based Collaborative Filtering Algorithm (w/ MSD):

        Fold 1  Fold 2  Fold 3  Mean
RMSE    0.9823  0.9923  0.9898  0.9881
MAE     0.7767  0.7835  0.7839  0.7814
```

```
User-Based Collaborative Filtering Algorithm (w/ cosine):

        Fold 1  Fold 2  Fold 3  Mean
RMSE    1.0150  1.0257  1.0232  1.0213
MAE     0.8033  0.8105  0.8120  0.8086
```

```
User-Based Collaborative Filtering Algorithm (w/ Pearson):

        Fold 1  Fold 2  Fold 3  Mean
RMSE    1.0150  1.0241  1.0228  1.0206
MAE     0.8060  0.8117  0.8136  0.8104

Item-Based Collaborative Filtering Algorithm (w/ MSD):

        Fold 1  Fold 2  Fold 3  Mean
RMSE    0.9818  0.9886  0.9877  0.9860
MAE     0.7777  0.7817  0.7840  0.7811

Item-Based Collaborative Filtering Algorithm (w/ cosine):

        Fold 1  Fold 2  Fold 3  Mean
RMSE    1.0318  1.0387  1.0373  1.0360
MAE     0.8186  0.8243  0.8250  0.8226

Item-Based Collaborative Filtering Algorithm (w/ Pearson):

        Fold 1  Fold 2  Fold 3  Mean
RMSE    1.0444  1.0520  1.0520  1.0495
MAE     0.8359  0.8393  0.8434  0.8395
```

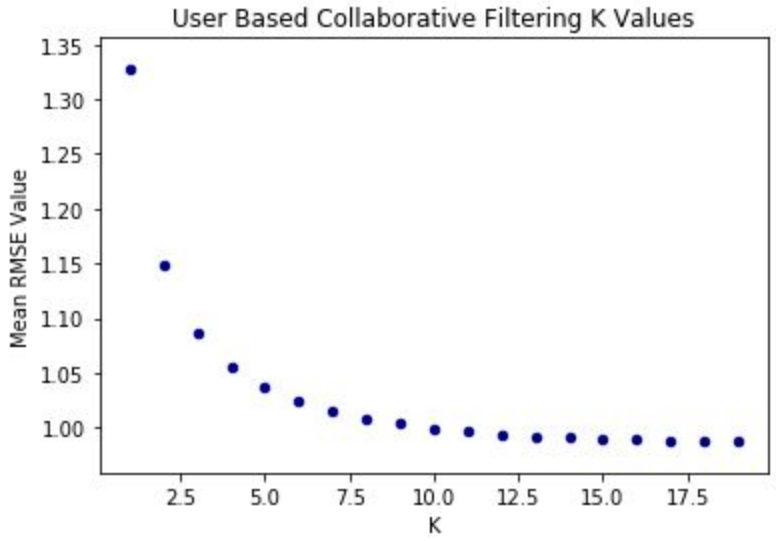


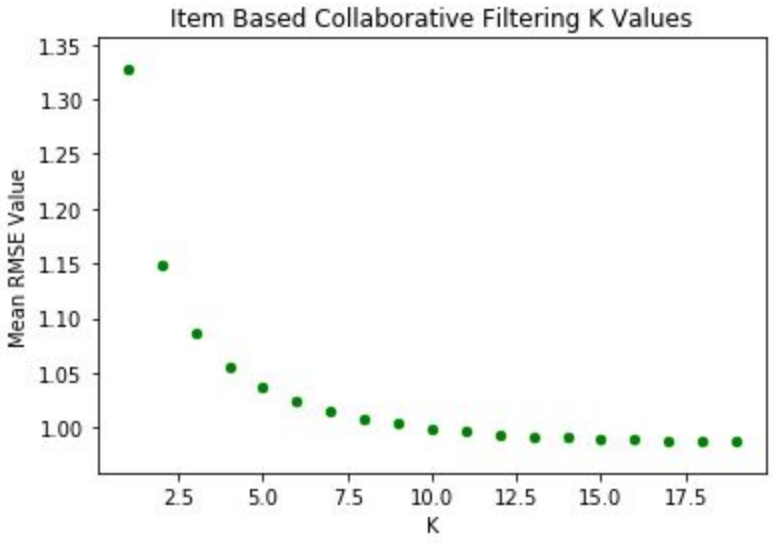

```python
#User-Based Collaborative Filtering Algorithm with Cosine, MSD and Pearson Similarities
algo = KNNBasic(sim_options={'name':'MSD','user_based':True})
perf = evaluate(algo, data, measures=['RMSE','MAE'])
print("User-Based Collaborative Filtering Algorithm (w/ MSD): ")
print()
print_perf(perf)
print()
```

```
#Item-Based Collaborative Filtering Algorithm with Cosine, MSD and Pearson Similarities
algo = KNNBasic(sim_options={'name':'MSD','user_based':False})
perf = evaluate(algo, data, measures=['RMSE','MAE'])
print("Item-Based Collaborative Filtering Algorithm (w/ MSD): ")
print()
print_perf(perf)
print()
```

```
plotc = cf.plot.bar(x='filtering_type', y='rmse_mean', legend=False)
plotc.set_title('User Based and Item Based Collaborative w/ MSD, Cosine and Pearson Similarities',
plotc.set_ylabel('Mean RMSE Value', size=12)
plotc.set_xlabel('Collaborative Filtering Type', size=12)
plotc.legend(False)
```

15. For this final step, I looked at how the number of neighbors impacted the performance of User-based Collaborative Filtering and Item-based Collaborative Filtering. I used a loop to run the algorithms with many different K values and then plotted the means based on different K's. In terms of RMSE, it appears that for both algorithms the lower the K, the lower the mean RMSE value. The best K value that I tested for both seemed to be lower than that of the plots below. When I raised the K value to around 55, I seemed to get the least amount of error. After increasing the value around 60 and 70, the amount of error seemed to increase. The plots are below along with some code:






```
#User-Based Collaborative Filtering Algorithm with varying K values
for i in range(0,20):
    algo = KNNBasic(k=i, sim_options={'name':'MSD','user_based':True})
    perf = evaluate(algo, data, measures=['RMSE'])
    print("User-Based Collaborative Filtering Algorithm (k=", i)
    print()
    print_perf(perf)
    print()
```

```
plot1 = df.plot.scatter(x='k', y='mean', c='DarkBlue')
plot1.set_title('User Based Collaborative Filtering K Values')
plot1.set_ylabel('Mean RMSE Value')
plot1.set_xlabel('K')
```