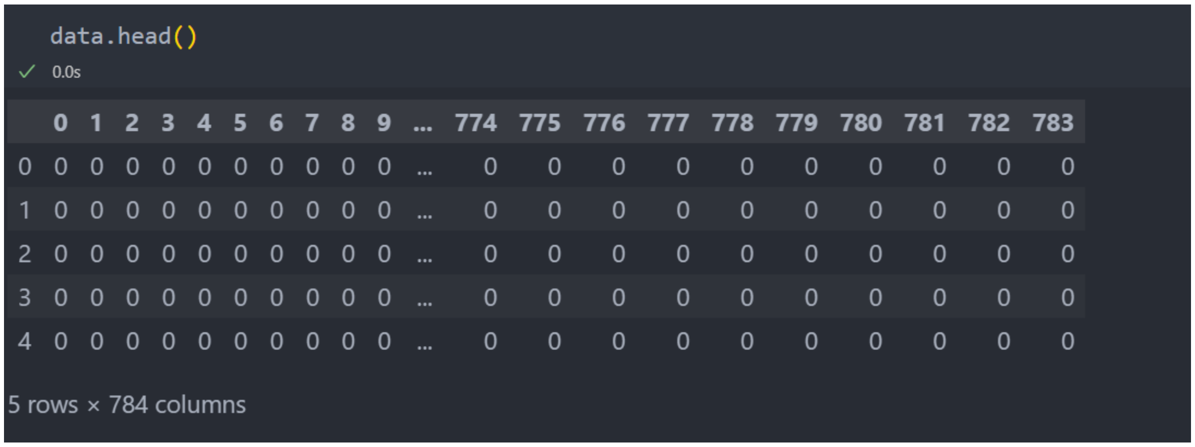
DM Models - 2

Task 1 Algorithmic Analysis K-Means Clustering with Real World Dataset

First, download a simulated dataset: kmeans_data.zip from Modules->Datasets. Then, implement the K-means algorithm **from scratch**. K-means algorithm computes the distance of a given data point pair. Replace the distance computation function with Euclidean distance, 1- Cosine similarity, and 1 – the **Generalized** Jarcard similarity (refer to: https://www.itl.nist.gov/div898/software/dataplot/refman2/auxillar/jaccard.htm).

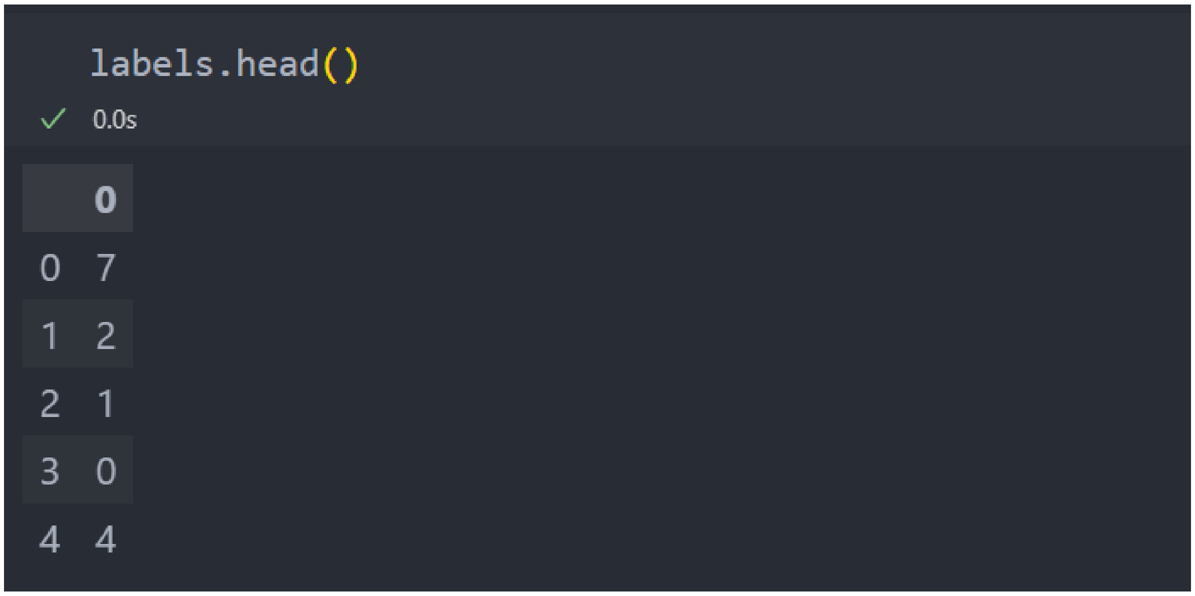
Dataset:



```
data.head()
✓ 0.0s
```

	0	1	2	3	4	5	6	7	8	9	...	774	775	776	777	778	779	780	781	782	783
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 784 columns



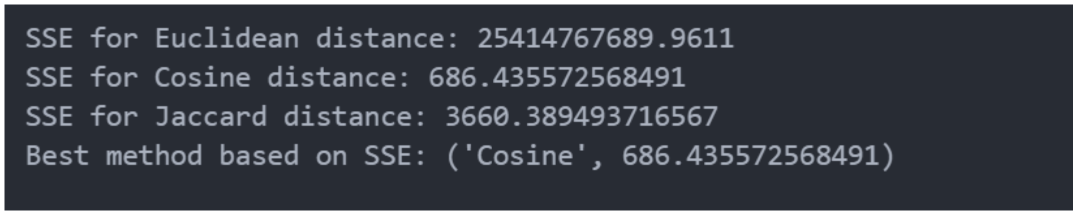
```
labels.head()
✓ 0.0s
```

	0
0	7
1	2
2	1
3	0
4	4

Q1: Run K-means clustering with Euclidean, Cosine and Jarcard similarity. Specify K= the number of categorical values of y (the number of classifications). Compare the SSEs of Euclidean-K-means, Cosine-K-means, Jarcard-K-means. Which method is better? (10 points)

Ans:

After running K-means clustering with Euclidean, Cosine, and Jaccard distance metrics, I found that Cosine distance had the lowest SSE (686.43) compared to Euclidean (25.4 billion) and Jaccard (3660.39). The minimum SSE value of cosine is because it generally works well on high-dimensional and sparse data. Cosine Distance is identified as the better method based on SSE.

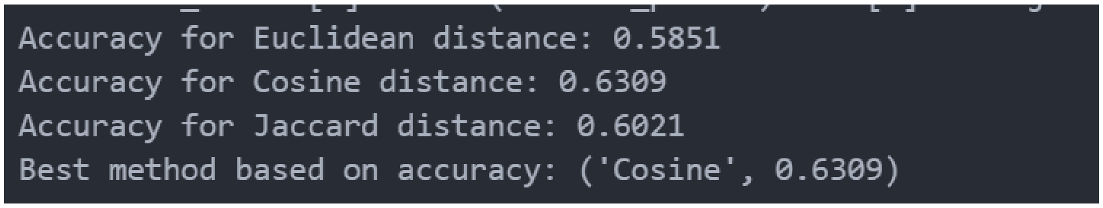


```
SSE for Euclidean distance: 25414767689.9611
SSE for Cosine distance: 686.435572568491
SSE for Jaccard distance: 3660.389493716567
Best method based on SSE: ('Cosine', 686.435572568491)
```

Q2: Compare the accuracies of Euclidean-K-means Cosine-K-means, Jarcard-K-means. First, label each cluster using the majority vote label of the data points in that cluster. Later, computethe predictive accuracy of Euclidean-K-means, Cosine-K-means, Jarcard-K-means. Which metric is better? (10 points)

Ans:

Using majority vote to assign labels to clusters, Cosine distance achieved the highest accuracy (0.6309), followed by Jaccard (0.6021) and Euclidean (0.5851). Cosine-K-means is the preferred method on accuracy too.

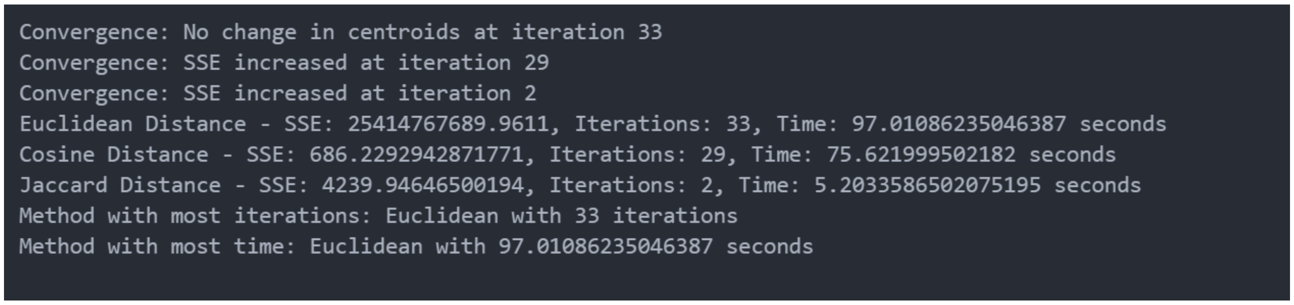


```
Accuracy for Euclidean distance: 0.5851
Accuracy for Cosine distance: 0.6309
Accuracy for Jaccard distance: 0.6021
Best method based on accuracy: ('Cosine', 0.6309)
```

Q3: Set up the same stop criteria: “when there is no change in centroid position OR when the SSE value increases in the next iteration OR when the maximum preset value (e.g., 500, you can set the preset value by yourself) of iteration is complete”, for Euclidean-K-means, Cosine-K- means, Jarcard-K-means. Which method requires more iterations and times to converge? (10 points)

Ans:

Implemented three stopping criteria: no change in centroids, SSE increase, and a maximum iteration limit. Euclidean distance took the longest to converge (33 iterations, 97 seconds), while Jaccard distance converged fastest (2 iterations, 5.2 seconds). Cosine distance converged in 29 iterations (75 seconds). These results show that Euclidean distance requires more time and iterations due to the high dimensionality, while Cosine and Jaccard converge more efficiently.



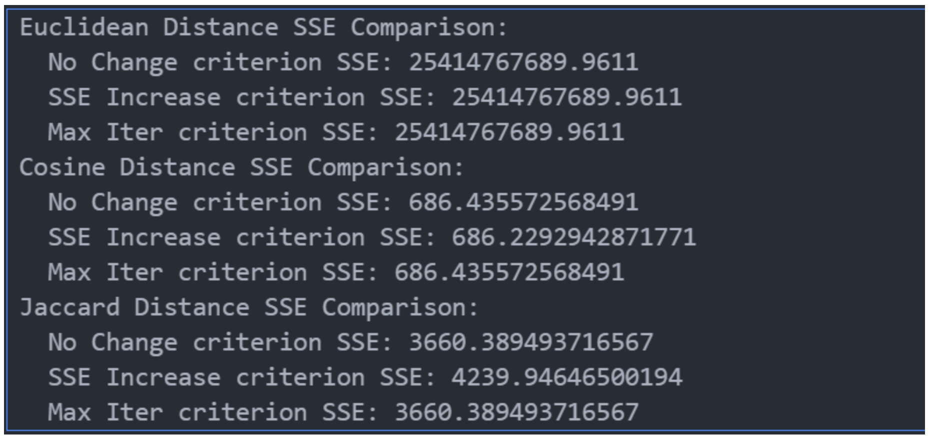
```
Convergence: No change in centroids at iteration 33
Convergence: SSE increased at iteration 29
Convergence: SSE increased at iteration 2
Euclidean Distance - SSE: 25414767689.9611, Iterations: 33, Time: 97.01086235046387 seconds
Cosine Distance - SSE: 686.2292942871771, Iterations: 29, Time: 75.621999502182 seconds
Jaccard Distance - SSE: 4239.94646500194, Iterations: 2, Time: 5.2033586502075195 seconds
Method with most iterations: Euclidean with 33 iterations
Method with most time: Euclidean with 97.01086235046387 seconds
```

Q4: Compare the SSEs of Euclidean-K-means Cosine-K-means, Jarcard-K-means with respect tothe following three terminating conditions: (10 points)

- when there is no change in centroid position
- when the SSE value increases in the next iteration
- when the maximum preset value (e.g., 100) of iteration is complete

Ans:

When comparing SSEs across the three stopping criteria, the cosine distance consistently generated the lowest SSE under all criteria, with a little improvement when using the "SSE increase" criterion (686.23). This illustrates that Cosine distance functions well independent of the stopping condition, while early termination due to SSE growth can help achieve optimal clustering more quickly. In contrast, Euclidean and Jaccard distances demonstrated no sensitivity to stopping criterion, with consistently high SSEs.



```
Euclidean Distance SSE Comparison:
  No Change criterion SSE: 25414767689.9611
  SSE Increase criterion SSE: 25414767689.9611
  Max Iter criterion SSE: 25414767689.9611
Cosine Distance SSE Comparison:
  No Change criterion SSE: 686.435572568491
  SSE Increase criterion SSE: 686.2292942871771
  Max Iter criterion SSE: 686.435572568491
Jaccard Distance SSE Comparison:
  No Change criterion SSE: 3660.389493716567
  SSE Increase criterion SSE: 4239.94646500194
  Max Iter criterion SSE: 3660.389493716567
```

Q5: What are your summary observations or takeaways based on your algorithmic analysis? (5points)

Ans:

- The optimal measure for high-dimensional, sparse data is cosine similarity, which outperforms other metrics in terms of accuracy and SSE.
- The "SSE increase" criterion for Cosine distance improved clustering performance marginally. However, all three criteria had limited effect on Euclidean and Jaccard outcomes.
- Euclidean distance needed more iterations and time, while Jaccard converged faster but had a higher SSE.
- Cosine similarity's robustness to feature scale makes it an acceptable choice for variable data, while Euclidean and Jaccard may have limitations in such circumstances.

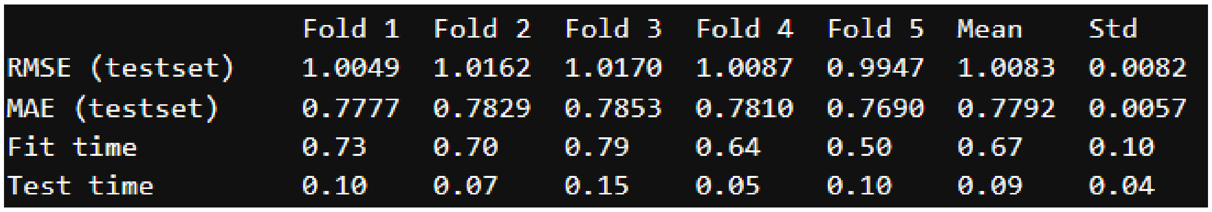
These findings highlight the necessity of selecting the right distance metric and stopping criteria based on data properties and clustering objectives.

Task 2, Machine Learning with Matrix Data for Recommender Systems

1. Recommender systems are a hot topic. Recommendation systems can be formulated as a task of matrix completion in machine learning. Recommender systems aim to predict the rating that a user will give for an item (e.g., a restaurant, a movie, a product).
2. Download the movie rating dataset from: https://www.kaggle.com/rounakbanik/the-movies-dataset. These files contain metadata for all 45,000 movies listed in the Full MovieLens Dataset. The dataset consists of movies released on or before July 2017. Data points include cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts and vote averages. This dataset also has files containing 26 million ratings from 270,000 users for all 45,000 movies. Ratings are on a scale of 1-5 and have been obtained from the official GroupLens website.
3. Building a small recommender system with the matrix data: “ratings small.csv”. You can use the recommender system library: Surprise (http://surpriselib.com), use other recommender system libraries, or implement from scratches.
    a. Read data from “ratings small.csv” with line format: 'userID movieID rating timestamp'.
    b. MAE and RMSE are two famous metrics for evaluating the performances of a recommender system. The definition of MAE can be found via: https://en.wikipedia.org/wiki/Mean_absolute_error. The definition of RMSE can be found via: https://en.wikipedia.org/wiki/Root-mean-square_deviation.
    c. Compute the average MAE and RMSE of the Probabilistic Matrix Factorization (PMF), User based Collaborative Filtering, Item based Collaborative Filtering, under the 5-folds cross-validation (10 points)

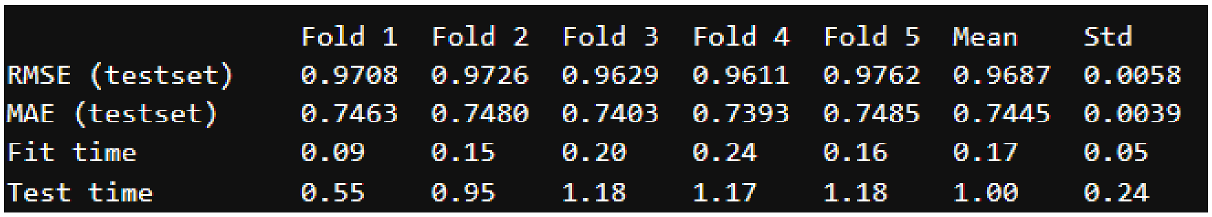
    Ans:

    MAE and RMSE of PMF:



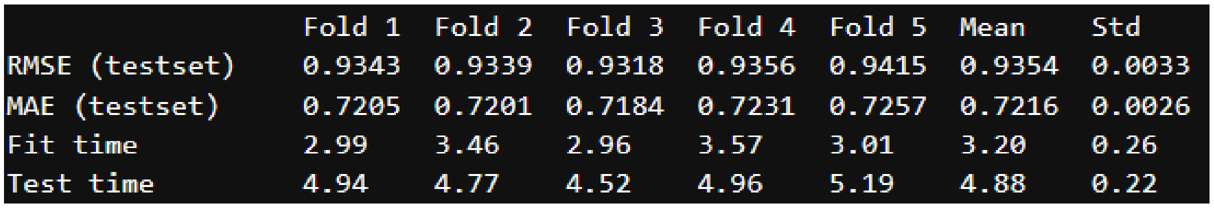
```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.0049  1.0162  1.0170  1.0087  0.9947  1.0083  0.0082
MAE (testset)     0.7777  0.7829  0.7853  0.7810  0.7690  0.7792  0.0057
Fit time          0.73    0.70    0.79    0.64    0.50    0.67    0.10
Test time         0.10    0.07    0.15    0.05    0.10    0.09    0.04
```

MAE RMSE of User-Based:



```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9708  0.9726  0.9629  0.9611  0.9762  0.9687  0.0058
MAE (testset)     0.7463  0.7480  0.7403  0.7393  0.7485  0.7445  0.0039
Fit time          0.09    0.15    0.20    0.24    0.16    0.17    0.05
Test time         0.55    0.95    1.18    1.17    1.18    1.00    0.24
```

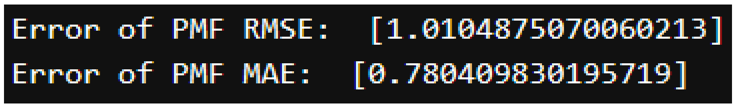
MAE RMSE of Item Based:



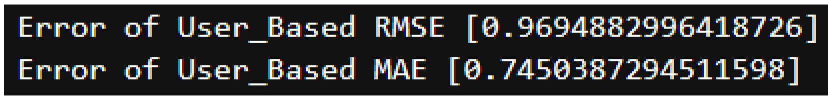
```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9343  0.9339  0.9318  0.9356  0.9415  0.9354  0.0033
MAE (testset)     0.7205  0.7201  0.7184  0.7231  0.7257  0.7216  0.0026
Fit time          2.99    3.46    2.96    3.57    3.01    3.20    0.26
Test time         4.94    4.77    4.52    4.96    5.19    4.88    0.22
```

d. Compare the **average (mean)** performances of User-based collaborative filtering, item-based collaborative filtering, PMF with respect to RMSE and MAE. Which ML model is the best in the movie rating data? (10 points)

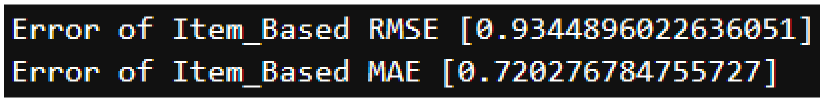
Ans:



```
Error of PMF RMSE:  [1.0104875070060213]
Error of PMF MAE:  [0.780409830195719]
```



```
Error of User_Based RMSE [0.9694882996418726]
Error of User_Based MAE [0.7450387294511598]
```



```
Error of Item_Based RMSE [0.9344896022636051]
Error of Item_Based MAE [0.720276784755727]
```

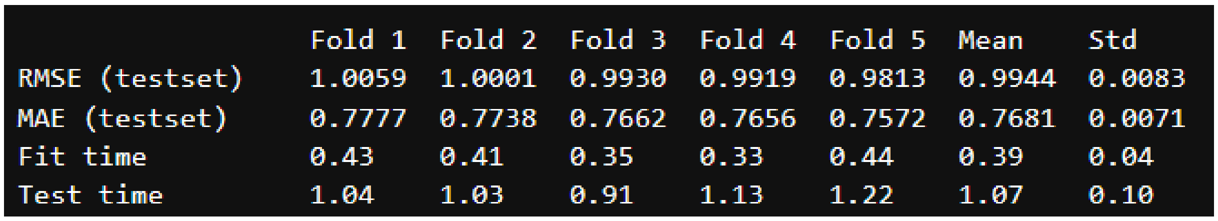
Therefore, Item-Based collaborative filtering is the best in the movie rating data.

e. Examine how the cosine, MSD (Mean Squared Difference), and Pearson similarities impact the performances of User based Collaborative Filtering and Item based Collaborative Filtering. Plot your results. Is the impact of the three metrics on User based Collaborative Filtering consistent with the impact of the three metrics on Item based Collaborative Filtering? (10 points)

Ans:

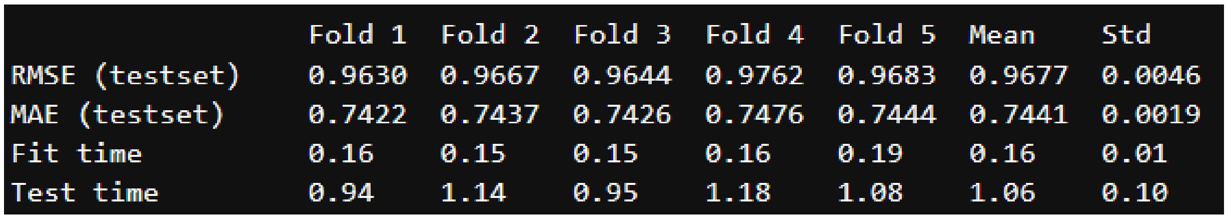
Performances:

User-Based Cosine:



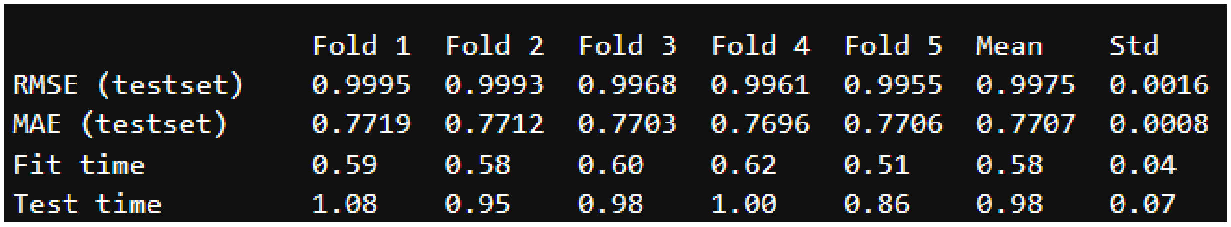
```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.0059  1.0001  0.9930  0.9919  0.9813  0.9944  0.0083
MAE (testset)     0.7777  0.7738  0.7662  0.7656  0.7572  0.7681  0.0071
Fit time          0.43    0.41    0.35    0.33    0.44    0.39    0.04
Test time         1.04    1.03    0.91    1.13    1.22    1.07    0.10
```

User-Based MSD:



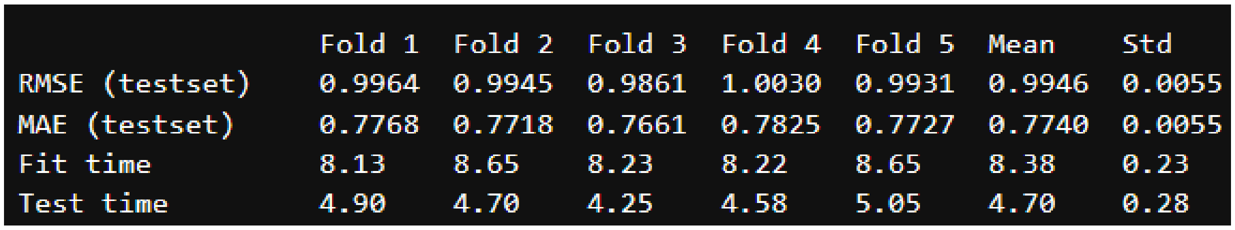
```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9630  0.9667  0.9644  0.9762  0.9683  0.9677  0.0046
MAE (testset)     0.7422  0.7437  0.7426  0.7476  0.7444  0.7441  0.0019
Fit time          0.16    0.15    0.15    0.16    0.19    0.16    0.01
Test time         0.94    1.14    0.95    1.18    1.08    1.06    0.10
```

User-Based Pearson Baseline:



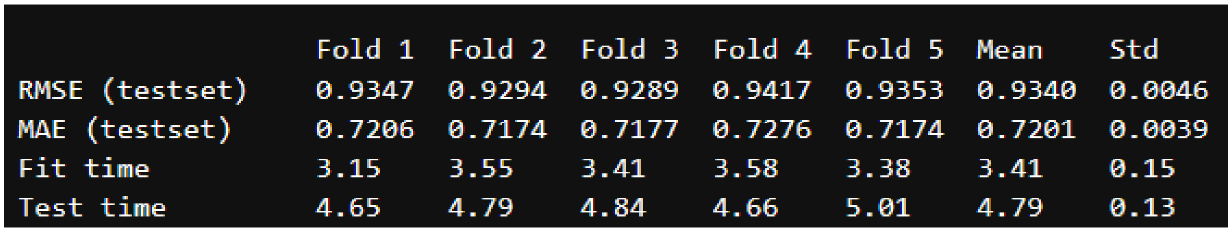
```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9995  0.9993  0.9968  0.9961  0.9955  0.9975  0.0016
MAE (testset)     0.7719  0.7712  0.7703  0.7696  0.7706  0.7707  0.0008
Fit time          0.59    0.58    0.60    0.62    0.51    0.58    0.04
Test time         1.08    0.95    0.98    1.00    0.86    0.98    0.07
```

Item-Based Cosine:



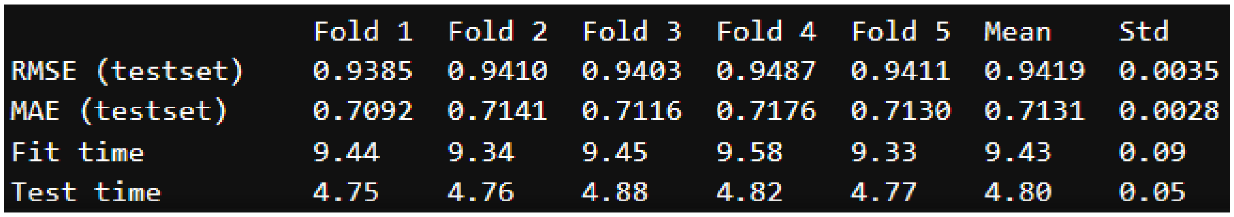
```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9964  0.9945  0.9861  1.0030  0.9931  0.9946  0.0055
MAE (testset)     0.7768  0.7718  0.7661  0.7825  0.7727  0.7740  0.0055
Fit time          8.13    8.65    8.23    8.22    8.65    8.38    0.23
Test time         4.90    4.70    4.25    4.58    5.05    4.70    0.28
```

Item-Based MSD:

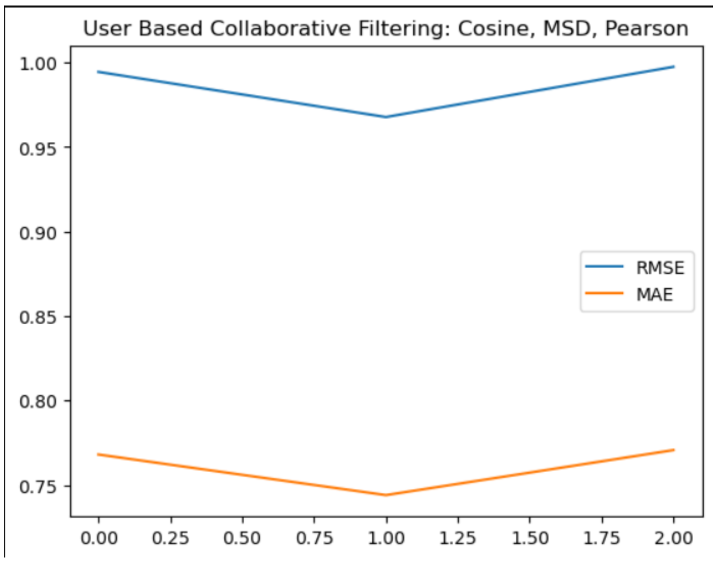


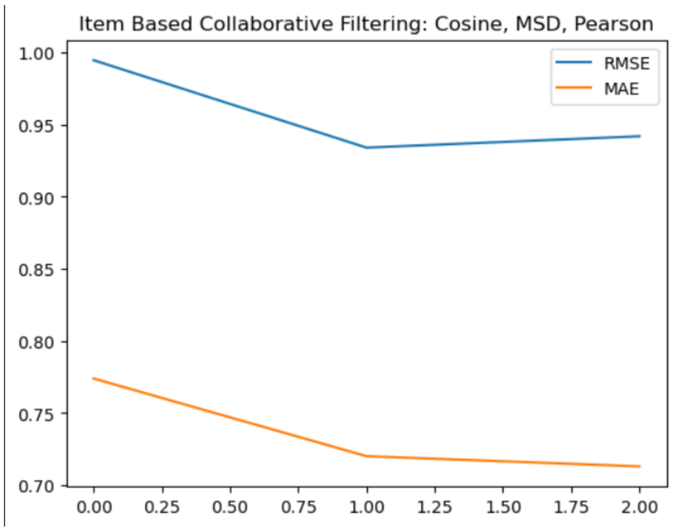
```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9347  0.9294  0.9289  0.9417  0.9353  0.9340  0.0046
MAE (testset)     0.7206  0.7174  0.7177  0.7276  0.7174  0.7201  0.0039
Fit time          3.15    3.55    3.41    3.58    3.38    3.41    0.15
Test time         4.65    4.79    4.84    4.66    5.01    4.79    0.13
```

Item-Based Pearson Baseline:



```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9385  0.9410  0.9403  0.9487  0.9411  0.9419  0.0035
MAE (testset)     0.7092  0.7141  0.7116  0.7176  0.7130  0.7131  0.0028
Fit time          9.44    9.34    9.45    9.58    9.33    9.43    0.09
Test time         4.75    4.76    4.88    4.82    4.77    4.80    0.05
```

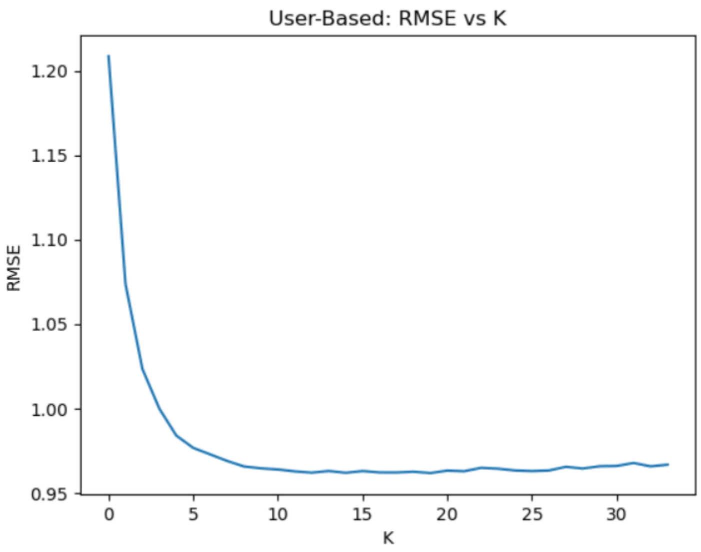


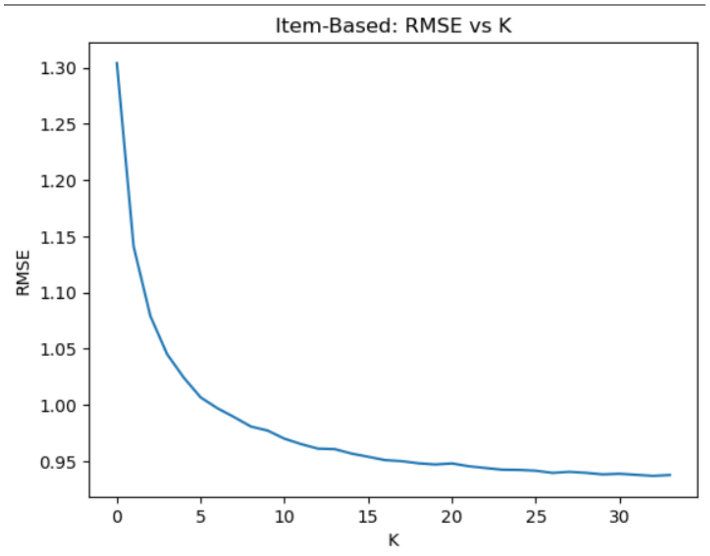


From the above two charts, we can conclude that the effect of the three metrics on User-based Collaborative Filtering is similar to the effect of the three metrics on Item-based Collaborative Filtering

f. Examine how the number of neighbors impacts the performances of User based Collaborative Filtering and Item based Collaborative Filtering? Plot your results.(10 points)

Ans:





When there are fewer neighbors, user-based collaborative filtering works better. But when there are more neighbors, item-based collaborative filtering works better.

g. Identify the best number of neighbor (denoted by K) for User/Item based collaborative filtering in terms of RMSE. Is the best K of User based collaborative filtering the same with the best K of Item based collaborative filtering? (10 points)

Ans:

The best K value for User-based collaborative filtering differs from that of Item-based collaborative filtering. For User-Based, the best K is 19, while for Item-Based, the best K is 32.

GitHub Links:

Task – 1:

https://github.com/renujakkampudi/Data-Mining-HW-Tasks/blob/main/HW3_Task1.ipynb

Task - 2:

https://github.com/renujakkampudi/Data-Mining-HW-Tasks/blob/main/HW3_Task2.ipynb