Tejal Bhangale

R : Hands-on exercises

1. Recommender systems are a hot topic in data science companies. Recommender systems aim to predict the rating that a user will give for an item (e.g., a restaurant, a movie, a product, a Point of Interest). Surprise (http://surpriselib.com) is a Python package for developing recommender systems. To install Surprise, the easiest way is to use pip.

Open your console:
$ pip install numpy
$ pip install scikit-surprise

2. Download an experimental dataset "restaurant_ratings.txt": Files/Data/restaurant_ratings.txt

3. Load data from "restaurant_ratings.txt" with line format: 'user item rating timestamp'.

```
file_path = os.path.expanduser('D:/Sem 2/Temporal and spatial data/HW4/restaurant_ratings.txt')
reader = Reader(line_format='user item rating timestamp', sep='\t')
data = Dataset.load_from_file(file_path, reader=reader)
```

4. MAE and RMSE are two famous metrics for evaluating the performances of a recommender system.

The Mean Absolute Error is given by:

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}.$$

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

5-6-7-8. Split the data for 3-folds cross-validation, and compute the MAE and RMSE of the SVD (Singular Value Decomposition) algorithm, PMF (Probabilistic Matrix Factorization) algorithm, NMF (Nonnegative Matrix Factorization) algorithm, User based Collaborative Filtering algorithm, Item based Collaborative Filtering algorithm.

9-10-11-12. Compare the performances of SVD, PMF, NMF, UCF, ICF on fold-1, fold-2, fold-3 with respect to RMSE and MAE. Since data.split(n_folds=3)randomly split the data into 3 folds, please make sure you test the five algorithms on the same fold-1 so the results are comparable.

```
#3-fold data without shuffeling so results are comparable
kf = KFold(n_splits=3, random_state=None, shuffle=False)
```

```python
algorithms = {
    'SVD':SVD(),
    'PMF':SVD(biased=False),
    'NMF':NMF(),
    'UCF':KNNBasic(sim_options = {'user_based': True} ),
    'ICF':KNNBasic(sim_options = {'user_based': False} )
}
for name, algo in algorithms.items():
    rmse = []
    mae = []
    i = 1
    print("Evaluating RMSE, MAE of algorithm",name)
    print("========================================")
    for trainset, testset in kf.split(data):
        # train and test algorithm.
        algo.fit(trainset)
        predictions = algo.test(testset)
        # Compute and print Root Mean Squared Error and MAE
        print("fold -",i)
        i = i + 1
        rmse.append(accuracy.rmse(predictions, verbose=True))
        mae.append(accuracy.mae(predictions, verbose=True))
    print("\nRMSE Mean: %.4f" % np.mean(rmse))
    print("MAE Mean: %.4f"% np.mean(mae))
```

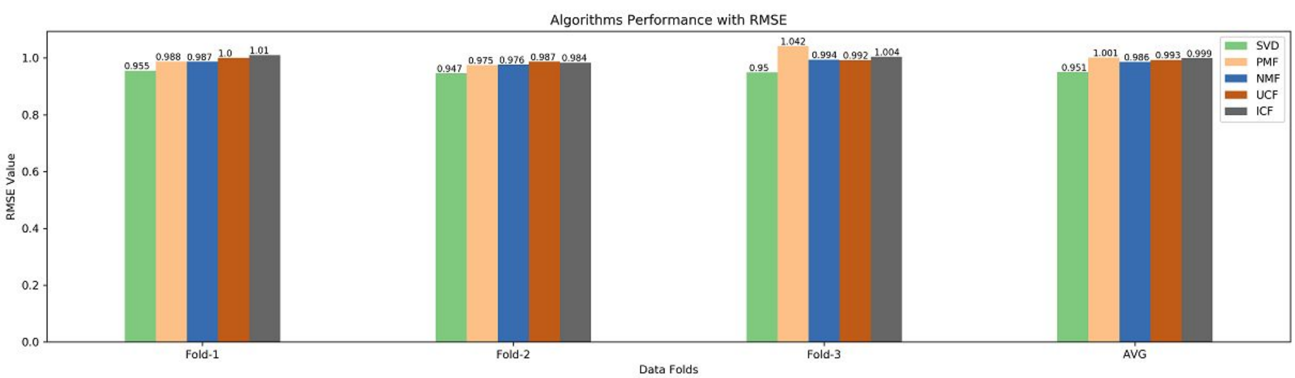
```
Evaluating RMSE, MAE of algorithm SVD
========================================
fold - 1
RMSE: 0.9554
MAE:  0.7541
fold - 2
RMSE: 0.9468
MAE:  0.7469
fold - 3
RMSE: 0.9495
MAE:  0.7502

RMSE Mean: 0.9506
MAE Mean: 0.7504
```

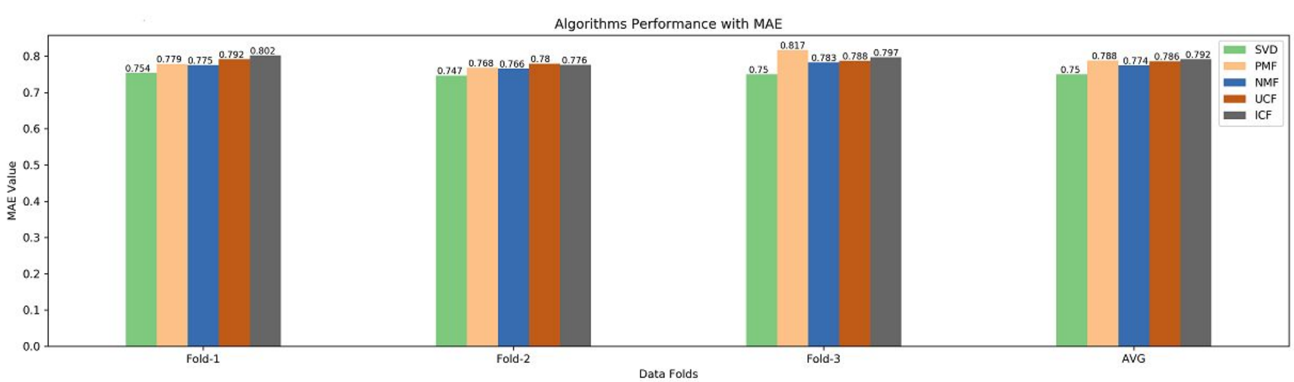
```
Evaluating RMSE, MAE of algorithm PMF
========================================
fold - 1
RMSE: 0.9875
MAE:  0.7787
fold - 2
RMSE: 0.9746
MAE:  0.7683
fold - 3
RMSE: 1.0421
MAE:  0.8171

RMSE Mean: 1.0014
MAE Mean: 0.7880
```

```
Evaluating RMSE, MAE of algorithm NMF
========================================
fold - 1
RMSE: 0.9868
MAE:  0.7752
fold - 2
RMSE: 0.9764
MAE:  0.7655
fold - 3
RMSE: 0.9942
MAE:  0.7826

RMSE Mean: 0.9858
MAE Mean: 0.7744
```

```
Evaluating RMSE, MAE of algorithm UCF
========================================
Computing the msd similarity matrix...
Done computing similarity matrix.
fold - 1
RMSE: 1.0004
MAE:  0.7915
Computing the msd similarity matrix...
Done computing similarity matrix.
fold - 2
RMSE: 0.9867
MAE:  0.7795
Computing the msd similarity matrix...
Done computing similarity matrix.
fold - 3
RMSE: 0.9923
MAE:  0.7875

RMSE Mean: 0.9931
MAE Mean: 0.7862
```

```
Evaluating RMSE, MAE of algorithm ICF
========================================
Computing the msd similarity matrix...
Done computing similarity matrix.
fold - 1
RMSE: 1.0103
MAE:  0.8019
Computing the msd similarity matrix...
Done computing similarity matrix.
fold - 2
RMSE: 0.9837
MAE:  0.7763
Computing the msd similarity matrix...
Done computing similarity matrix.
fold - 3
RMSE: 1.0038
MAE:  0.7968

RMSE Mean: 0.9993
MAE Mean: 0.7917
```





When we plot RMSE and MAE values on 3-folds of data for five algorithms, we can see that SVD algorithm gives smaller value of RMSE and MAE for all the folds and their average.

Hence, we can say that SVD performs better compared to other algorithms with lowest error value.

13. Examine how the cosine, MSD (Mean Squared Difference), and Pearson similarities impact the performances of User based Collaborative Filtering and Item based Collaborative Filtering.

```
similarities = {
    'UCF with MSD':KNNBasic(sim_options = {'name': 'MSD', 'user_based': True}),
    'ICF with MSD':KNNBasic(sim_options = {'name': 'MSD', 'user_based': False}),
    'UCF with cosine':KNNBasic(sim_options = {'name': 'cosine', 'user_based': True}),
    'ICF with cosine':KNNBasic(sim_options = {'name': 'cosine', 'user_based': False}),
    'UCF with pearson':KNNBasic(sim_options = {'name': 'pearson', 'user_based': True}),
    'ICF with pearson':KNNBasic(sim_options = {'name': 'pearson', 'user_based': False}),
}
for name, sim in similarities.items():
    rmse = []
    mae = []
    print("Evaluating algorithm",name)
    print("==============================================")
    for trainset, testset in kf.split(data):
        # train and test algorithm.
        sim.fit(trainset)
        predictions = sim.test(testset)
        # Compute and print Root Mean Squared Error and MAE
        rmse.append(accuracy.rmse(predictions,verbose=False))
        mae.append(accuracy.mae(predictions,verbose=False))
    print("\nRMSE Mean: %.4f" % np.mean(rmse))
    print("MAE Mean: %.4f"% np.mean(mae))
```

Finally, 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? Plot your results.

```
Evaluating algorithm UCF with MSD
========================================
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.

RMSE Mean: 0.9931
MAE Mean: 0.7862
```

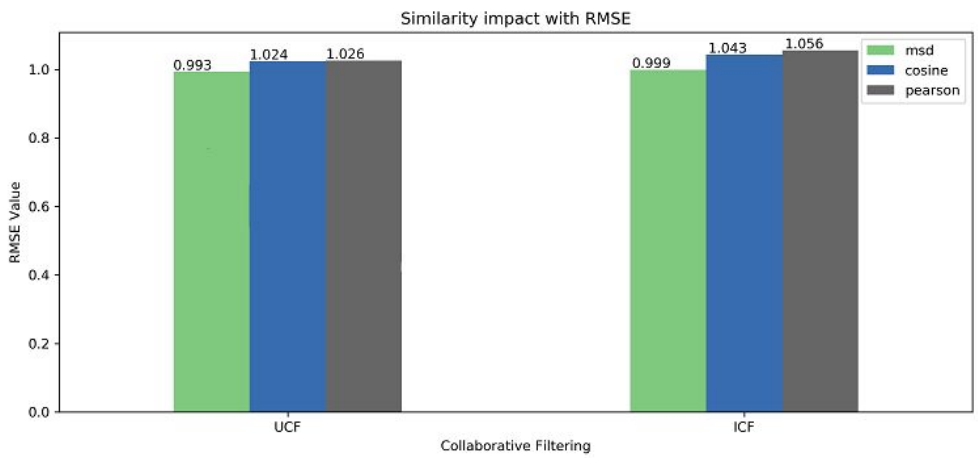
```
Evaluating algorithm ICF with MSD
========================================
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.

RMSE Mean: 0.9993
MAE Mean: 0.7917
```

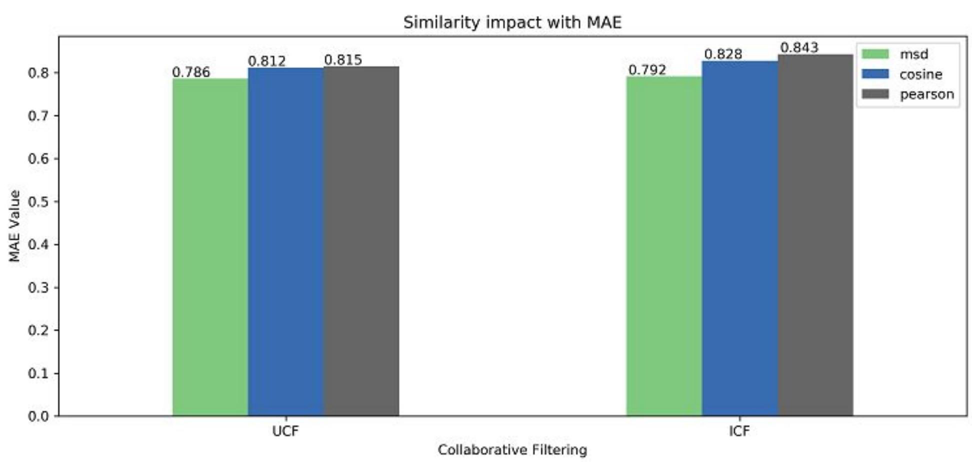
```
Evaluating algorithm UCF with cosine
==========================================
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.

RMSE Mean: 1.0240
MAE Mean: 0.8120
```

```
Evaluating algorithm ICF with cosine
==========================================
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.

RMSE Mean: 1.0434
MAE Mean: 0.8277
```

```
Evaluating algorithm UCF with pearson
===========================================
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.

RMSE Mean: 1.0262
MAE Mean: 0.8147
```

```
Evaluating algorithm ICF with pearson
===========================================
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.

RMSE Mean: 1.0555
MAE Mean: 0.8432
```





We compared User-based (UCF) and Item-based (ICF) Collaborative filtering with 3 similarity matrices, with respect to RMSE and MAE. **From the plots, we can see the impact of three metrics on UCF is consistent with the impact on ICF**. In both the cases of RMSE and MAE, MSD similarity gives lowest error value while Pearson similarity gives highest error value.

14. Examine how the number of neighbors impacts the performances of User based Collaborative Filtering or Item based Collaborative Filtering? Plot your results.

Here, I have specified the setting of k number of neighbors from k = 1 to 100.

```python
cf_types = ["UCF", "ICF"]
for cf in cf_types:
    print("Evaluating RMSE of algorithm", cf)
    print("========================================")
    print("RMSE Mean for 100 values of k:")
    rmse_mean = []
    for k in list(range(1,100)):
        algo = KNNBasic(k=k, sim_options = {'name':'msd', 'user_based': cf == "UCF"})
        rmse = []
        for trainset, testset in kf.split(data):
            # train and test algorithm.
            algo.fit(trainset)
            predictions = algo.test(testset,verbose=False)
            # Compute and print Root Mean Squared Error
            rmse.append(accuracy.rmse(predictions,verbose=False))
        rmse_mean.append("%.4f" % np.mean(rmse))
    print(rmse_mean)
    print()
```

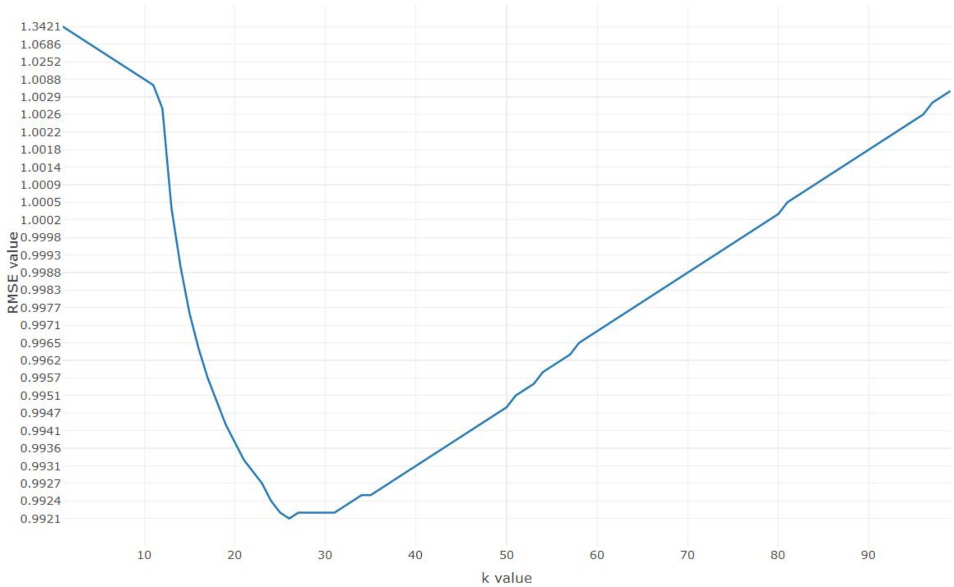
Identify the best K for User/Item based collaborative filtering in terms of RMSE. Is the the best K of User based collaborative filtering the same with the best K of Item based collaborative filtering?

Below are the RMSE values for k = 1 to 100, for UCF and ICF as well.

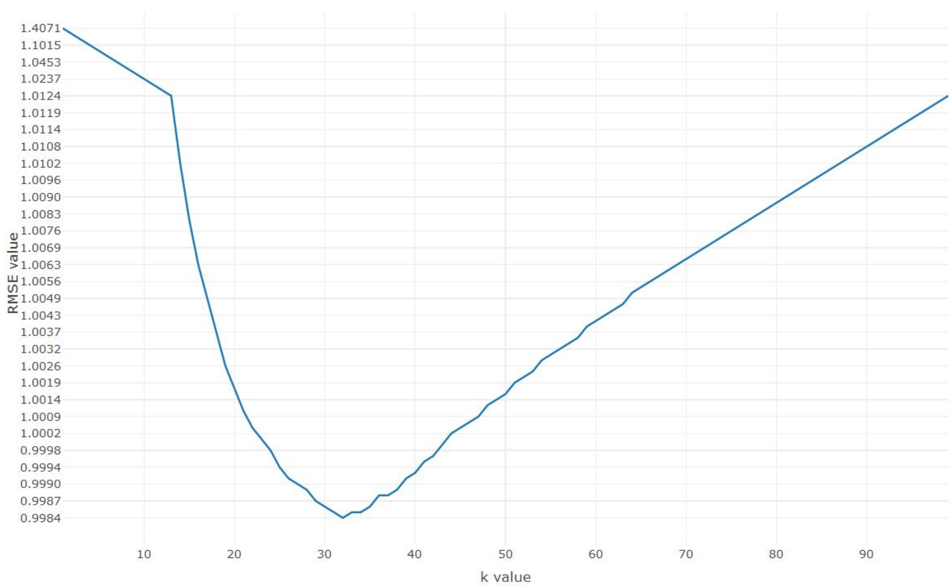
```
Evaluating RMSE of algorithm UCF
========================================
RMSE Mean for 100 values of k:

['1.3421', '1.1659', '1.1014', '1.0686', '1.0482', '1.0351', '1.0252', '1.0179', '1.0129', '1.0088',
'1.0055', '1.0027', '1.0004', '0.9989', '0.9975', '0.9964', '0.9957', '0.9950', '0.9943', '0.9937',
'0.9932', '0.9929', '0.9927', '0.9924', '0.9922', '0.9921', '0.9922', '0.9922', '0.9922', '0.9922',
'0.9922', '0.9923', '0.9924', '0.9925', '0.9925', '0.9926', '0.9927', '0.9928', '0.9929', '0.9931',
'0.9932', '0.9934', '0.9936', '0.9937', '0.9939', '0.9941', '0.9943', '0.9945', '0.9947', '0.9949',
'0.9951', '0.9954', '0.9955', '0.9958', '0.9960', '0.9962', '0.9963', '0.9965', '0.9968', '0.9969',
'0.9971', '0.9973', '0.9975', '0.9977', '0.9979', '0.9981', '0.9983', '0.9985', '0.9987', '0.9988',
 '0.9989', '0.9991', '0.9993', '0.9994', '0.9996', '0.9998', '0.9999', '1.0001', '1.0002', '1.0003',
 '1.0005', '1.0006', '1.0008', '1.0009', '1.0011', '1.0012', '1.0014', '1.0015', '1.0017', '1.0018',
 '1.0020', '1.0021', '1.0022', '1.0023', '1.0025', '1.0026', '1.0028', '1.0029', '1.0030']
```

```
Evaluating RMSE of algorithm ICF
========================================
RMSE Mean for 100 values of k:

['1.4071', '1.2163', '1.1416', '1.1015', '1.0757', '1.0581', '1.0453', '1.0364', '1.0292', '1.0237',
 '1.0190', '1.0154', '1.0124', '1.0102', '1.0081', '1.0063', '1.0049', '1.0037', '1.0026', '1.0018',
 '1.0011', '1.0005', '1.0001', '0.9998', '0.9994', '0.9991', '0.9990', '0.9989', '0.9987', '0.9986',
 '0.9985', '0.9984', '0.9985', '0.9985', '0.9986', '0.9988', '0.9988', '0.9989', '0.9991', '0.9993',
 '0.9995', '0.9997', '0.9999', '1.0002', '1.0005', '1.0007', '1.0009', '1.0012', '1.0014', '1.0017',
 '1.0019', '1.0022', '1.0024', '1.0027', '1.0029', '1.0032', '1.0034', '1.0036', '1.0038', '1.0040',
 '1.0043', '1.0046', '1.0048', '1.0051', '1.0053', '1.0056', '1.0058', '1.0061', '1.0063', '1.0066',
 '1.0068', '1.0069', '1.0072', '1.0074', '1.0076', '1.0079', '1.0081', '1.0083', '1.0085', '1.0088',
 '1.0090', '1.0092', '1.0094', '1.0096', '1.0098', '1.0100', '1.0102', '1.0104', '1.0106', '1.0108',
 '1.0110', '1.0112', '1.0114', '1.0116', '1.0118', '1.0119', '1.0121', '1.0123', '1.0124']
```

KNN impact on User based Collaborative Filtering:



KNN impact on Item based Collaborative Filtering:



From these graphs, we can see that the best k for User based collaborative filtering is **k = 26**, which gives lowest RMSE value of 0.9921. While the best k for Item based collaborative filtering is **k = 32**, which gives lowest RMSE value of 0.9984.

Hence, the best k of User based collaborative filtering is not the same with the best k of Item based collaborative filtering.