Recommendation Systems Prediction Calculation

User_based_Recommendation_Systems_Prediction_Calculation:

```
def predict(data_matrix, similarity, type):
    if type=='user':
        mean_user_rating = data_matrix.mean(axis=1).to_numpy()
        ratings_diff = (data_matrix - mean_user_rating[:, np.newaxis])
        pred = mean_user_rating[:, np.newaxis] + similarity.dot(ratings_diff) / np.array([np.abs(similarity).sum(axis=1)]).T
    return pred
```

Explanation:
data_matrix which has user_id x movie_id matrix with rating values

```
data_matrix
```

movie_id	1	2	3	4	5	6	7	8	9	10	...	1673	1674	1675	1676	1677	1678	1679	1680	1681	1682
user_id																					
1	5.0	3.0	4.0	3.0	3.0	5.0	4.0	1.0	5.0	3.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	4.0	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	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	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	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	4.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
939	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
940	0.0	0.0	0.0	2.0	0.0	0.0	4.0	5.0	3.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
941	5.0	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	0.0	0.0	0.0	0.0
942	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
943	0.0	5.0	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	0.0	0.0

943 rows × 1682 columns

Predict() has multiple lines of code. We will see what each line is doing

1. mean_user_rating = data_matrix.mean(axis=1).to_numpy()
    a. This line calculates the average rating given by each user across all movies.
    b. axis=1 indicates that the mean is calculated across columns (i.e., for each row).

```
type(data_matrix.mean(axis=1))
```

```
pandas.core.series.Series
```

```
data_matrix.mean(axis=1)
```

```
user_id
1      0.583829
2      0.136742
3      0.089774
4      0.061831
5      0.299049
         ...
939    0.124257
940    0.219976
941    0.052913
942    0.200357
943    0.340666
Length: 943, dtype: float64
```

Here the mean returns Series but we need numpy array to subtract the mean with rating values in data matrix so we need to convert that into numpy array
Result:

```
data_matrix.mean(axis=1).to_numpy()
```

```
array([0.58382878, 0.13674197, 0.08977408, 0.06183115, 0.29904875,
       0.45600476, 0.95005945, 0.13317479, 0.05588585, 0.46016647,
       0.37277051, 0.13317479, 1.17122473, 0.23840666, 0.17776457,
       0.36028537, 0.05053508, 0.6391201 , 0.04221165, 0.08858502,
       0.28418549, 0.25505351, 0.32639715, 0.17479191, 0.18787158,
       0.18727705, 0.04815696, 0.17479191, 0.07372176, 0.09631391,
       0.08382878, 0.08085612, 0.0529132 , 0.04815696, 0.04458977,
       0.0451843 , 0.12366231, 0.26753864, 0.04934602, 0.06004756,
       0.11652794, 0.40546968, 0.48751486, 0.32758621, 0.10285375,
       0.06590287, 0.05410226, 0.14863258, 0.343044  , 0.05053508,
```

2. ratings_diff = (data_matrix - mean_user_rating[:, np.newaxis])
    a. mean_user_rating[:, np.newaxis] reshaped to a column vector (2D array)
    b. This reshaping ensures that the mean user rating can be correctly broadcasted during the calculation of ratings_diff.

Result:

```
ratings_diff
```

movie_id	1	2	3	4	5	6	7	8	9	10	...	1673	1674	1675
user_id														
1	4.416171	2.416171	3.416171	2.416171	2.416171	4.416171	3.416171	0.416171	4.416171	2.416171	...	-0.583829	-0.583829	-0.583829
2	3.863258	-0.136742	-0.136742	-0.136742	-0.136742	-0.136742	-0.136742	-0.136742	-0.136742	1.863258	...	-0.136742	-0.136742	-0.136742
3	-0.089774	-0.089774	-0.089774	-0.089774	-0.089774	-0.089774	-0.089774	-0.089774	-0.089774	-0.089774	...	-0.089774	-0.089774	-0.089774
4	-0.061831	-0.061831	-0.061831	-0.061831	-0.061831	-0.061831	-0.061831	-0.061831	-0.061831	-0.061831	...	-0.061831	-0.061831	-0.061831
5	3.700951	2.700951	-0.299049	-0.299049	-0.299049	-0.299049	-0.299049	-0.299049	-0.299049	-0.299049	...	-0.299049	-0.299049	-0.299049
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
939	-0.124257	-0.124257	-0.124257	-0.124257	-0.124257	-0.124257	-0.124257	-0.124257	4.875743	-0.124257	...	-0.124257	-0.124257	-0.124257
940	-0.219976	-0.219976	-0.219976	1.780024	-0.219976	-0.219976	3.780024	4.780024	2.780024	-0.219976	...	-0.219976	-0.219976	-0.219976
941	4.947087	-0.052913	-0.052913	-0.052913	-0.052913	-0.052913	3.947087	-0.052913	-0.052913	-0.052913	...	-0.052913	-0.052913	-0.052913
942	-0.200357	-0.200357	-0.200357	-0.200357	-0.200357	-0.200357	-0.200357	-0.200357	-0.200357	-0.200357	...	-0.200357	-0.200357	-0.200357
943	-0.340666	4.659334	-0.340666	-0.340666	-0.340666	-0.340666	-0.340666	-0.340666	2.659334	-0.340666	...	-0.340666	-0.340666	-0.340666

943 rows × 1682 columns

3. pred = mean_user_rating[:, np.newaxis] + similarity.dot(ratings_diff) / np.array([np.abs(similarity).sum(axis=1)]).T
    a. The `similarity.dot(ratings_diff)` operation performs a matrix multiplication between the similarity matrix and the `ratings_diff` matrix

Similarity result:

```
user_similarity
```

```
array([[1.33226763e-15, 8.33069016e-01, 9.52540457e-01, ...,
        8.51383057e-01, 8.20492117e-01, 6.01825261e-01],
       [8.33069016e-01, 0.00000000e+00, 8.89408675e-01, ...,
        8.38515222e-01, 8.27732187e-01, 8.94202122e-01],
       [9.52540457e-01, 8.89408675e-01, 0.00000000e+00, ...,
        8.98757435e-01, 8.66583851e-01, 9.73444131e-01],
       ...,
       [8.51383057e-01, 8.38515222e-01, 8.98757435e-01, ...,
        0.00000000e+00, 8.98358201e-01, 9.04880419e-01],
       [8.20492117e-01, 8.27732187e-01, 8.66583851e-01, ...,
        8.98358201e-01, 0.00000000e+00, 8.17535338e-01],
       [6.01825261e-01, 8.94202122e-01, 9.73444131e-01, ...,
        9.04880419e-01, 8.17535338e-01, 0.00000000e+00]])
```

Rating_diff:

```
ratings_diff
```

movie_id	1	2	3	4	5	6	7	8	9	10	...	1673	1674	1675
user_id														
1	4.416171	2.416171	3.416171	2.416171	2.416171	4.416171	3.416171	0.416171	4.416171	2.416171	...	-0.583829	-0.583829	-0.583829
2	3.863258	-0.136742	-0.136742	-0.136742	-0.136742	-0.136742	-0.136742	-0.136742	-0.136742	1.863258	...	-0.136742	-0.136742	-0.136742
3	-0.089774	-0.089774	-0.089774	-0.089774	-0.089774	-0.089774	-0.089774	-0.089774	-0.089774	-0.089774	...	-0.089774	-0.089774	-0.089774
4	-0.061831	-0.061831	-0.061831	-0.061831	-0.061831	-0.061831	-0.061831	-0.061831	-0.061831	-0.061831	...	-0.061831	-0.061831	-0.061831
5	3.700951	2.700951	-0.299049	-0.299049	-0.299049	-0.299049	-0.299049	-0.299049	-0.299049	-0.299049	...	-0.299049	-0.299049	-0.299049
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
939	-0.124257	-0.124257	-0.124257	-0.124257	-0.124257	-0.124257	-0.124257	-0.124257	4.875743	-0.124257	...	-0.124257	-0.124257	-0.124257
940	-0.219976	-0.219976	-0.219976	1.780024	-0.219976	-0.219976	3.780024	4.780024	2.780024	-0.219976	...	-0.219976	-0.219976	-0.219976
941	4.947087	-0.052913	-0.052913	-0.052913	-0.052913	-0.052913	3.947087	-0.052913	-0.052913	-0.052913	...	-0.052913	-0.052913	-0.052913
942	-0.200357	-0.200357	-0.200357	-0.200357	-0.200357	-0.200357	-0.200357	-0.200357	-0.200357	-0.200357	...	-0.200357	-0.200357	-0.200357
943	-0.340666	4.659334	-0.340666	-0.340666	-0.340666	-0.340666	-0.340666	-0.340666	2.659334	-0.340666	...	-0.340666	-0.340666	-0.340666

943 rows × 1682 columns

After multiplying the matrix the result is,

```
user_similarity.dot(ratings_diff)
```

```
array([[1053.46456048,  106.99918332,   32.77730441, ..., -135.27488255,
        -135.66005564, -135.85741105],
       [1236.14249108,  187.96477188,   45.1764139 , ..., -171.10760287,
        -169.99820267, -169.8412708 ],
       [1409.18485925,  197.62870362,   57.03606449, ..., -187.29854641,
        -185.62705133, -185.45885449],
       ...,
       [1245.81782545,  180.04311388,   39.90412677, ..., -178.35011786,
        -177.72089084, -177.5907325 ],
       [1258.24647345,  159.77486735,   58.68675582, ..., -163.6732682 ,
        -163.19833111, -162.87221498],
       [1128.30290113,  104.69887603,   33.37003481, ..., -145.99356249,
        -146.42195908, -146.33422024]])
```

Mean_user_rating:

```
mean_user_rating[:, np.newaxis]
```

```
array([[0.58382878],
       [0.13674197],
       [0.08977408],
       [0.06183115],
       [0.29904875],
       [0.45600476],
       [0.95005945],
       [0.13317479],
       [0.05588585],
       [0.46016647],
       [0.37277051],
       [0.13317479],
       [1.17122473],
```

Absolute sum of similarity of users

```
np.abs(user_similarity)
```

```
array([[1.33226763e-15, 8.33069016e-01, 9.52540457e-01, ...,
        8.51383057e-01, 8.20492117e-01, 6.01825261e-01],
       [8.33069016e-01, 0.00000000e+00, 8.89408675e-01, ...,
        8.38515222e-01, 8.27732187e-01, 8.94202122e-01],
       [9.52540457e-01, 8.89408675e-01, 0.00000000e+00, ...,
        8.98757435e-01, 8.66583851e-01, 9.73444131e-01],
       ...,
       [8.51383057e-01, 8.38515222e-01, 8.98757435e-01, ...,
        0.00000000e+00, 8.98358201e-01, 9.04880419e-01],
       [8.20492117e-01, 8.27732187e-01, 8.66583851e-01, ...,
        8.98358201e-01, 0.00000000e+00, 8.17535338e-01],
       [6.01825261e-01, 8.94202122e-01, 9.73444131e-01, ...,
        9.04880419e-01, 8.17535338e-01, 0.00000000e+00]])
```

```
np.abs(user_similarity).sum(axis=1)
```

```
array([711.08099519, 760.07331717, 825.95402725, 827.39224279,
       779.66510396, 716.76574749, 713.40474397, 777.18367345,
       844.95785479, 738.25941741, 759.6083914 , 787.5837141 ,
       689.77352349, 749.23289325, 767.9488538 , 730.66594402,
       794.99966418, 725.35033113, 838.87922505, 790.62608759,
       787.57614319, 770.40422666, 737.33790467, 753.67929565,
       762.77562448, 711.92729357, 830.83431315, 766.29113247,
       817.83967763, 790.00049674, 867.79453678, 776.19443727,
       821.53541302, 876.69262601, 850.58533255, 879.44009825,
```

```
np.array([np.abs(user_similarity).sum(axis=1)]).T
```

```
array([[711.08099519],
       [760.07331717],
       [825.95402725],
       [827.39224279],
       [779.66510396],
       [716.76574749],
       [713.40474397],
```

Conclusion:

The predict method is designed for collaborative filtering to predict ratings based on user-user similarity:

1. It first computes mean_user_rating, which calculates the average rating for each user across all movies.

2. ratings_diff is then computed by subtracting mean_user_rating from each user's ratings, reshaping it to align with the ratings matrix using np.newaxis.
3. The main prediction step similarity.dot(ratings_diff) computes a weighted sum of these differences using the similarity matrix similarity.
4. The division by np.array([np.abs(similarity).sum(axis=1)]).T normalizes this weighted sum by the absolute sum of similarities, ensuring that predictions are appropriately scaled.
5. Finally, pred contains the predicted ratings matrix where each entry represents the predicted rating for each user-movie pair based on their similarities and deviations from their mean ratings. This method is effective for personalized recommendations based on user behavior similarities.

Item based Prediction:

```
def predict(data_matrix, similarity, type):
    if type=='item':
        pred = ratings.dot(similarity) / np.array([np.abs(similarity).sum(axis=1)])
    return pred
```

This line performs the prediction calculation:

- ratings.dot(similarity): This performs a matrix multiplication (dot product) between the ratings matrix and the similarity matrix. Assuming ratings is a matrix where each row represents a user and each column represents an item, and similarity is a matrix where each element represents the similarity between items, this operation computes a weighted sum of the ratings based on item similarities.
- np.abs(similarity).sum(axis=1): This computes the sum of the absolute values of the similarity matrix along axis 1 (summing each row). It creates a 1D array where each element is the sum of the absolute similarities for a given item.
- np.array([...]): This wraps the 1D array from the previous step in another array, converting it into a 2D array with a single row.
- ratings.dot(similarity) / np.array([np.abs(similarity).sum(axis=1)]): This divides the result of the dot product by the 2D array, normalizing the predicted ratings by the sum of the absolute similarities.