Aamleen Ahmed
2020002

CF Quiz 4

Nuclear Norm Minimization

About Nuclear Norm Minimization

Nuclear norm minimization (NNM) is an algorithm widely used in Collaborative Filtering (CF) to address matrix completion problems. CF is a popular technique employed in recommendation systems to predict user preferences or ratings for items based on similarities with other users. Matrix completion involves estimating missing values in a partially observed matrix.

The key idea behind NNM is to find a low-rank matrix that best fits the observed entries of the incomplete matrix. By minimizing the sum of singular values, known as the nuclear norm, NNM encourages solutions with low-rank structure. This promotes sparsity and facilitates the capture of underlying patterns in the data.

In the context of CF, NNM aims to predict missing ratings by leveraging the observed ratings and learning the latent factors that influence user-item preferences. The minimization of the nuclear norm helps combat overfitting and enhances generalization, resulting in accurate predictions.

The performance of NNM is typically evaluated using metrics such as the Normalized Mean Absolute Error (NMAE). Achieving a low NMAE, such as 0.21, indicates that the NNM algorithm successfully minimized prediction errors, leading to accurate recommendations with a relatively small average deviation from the true ratings.

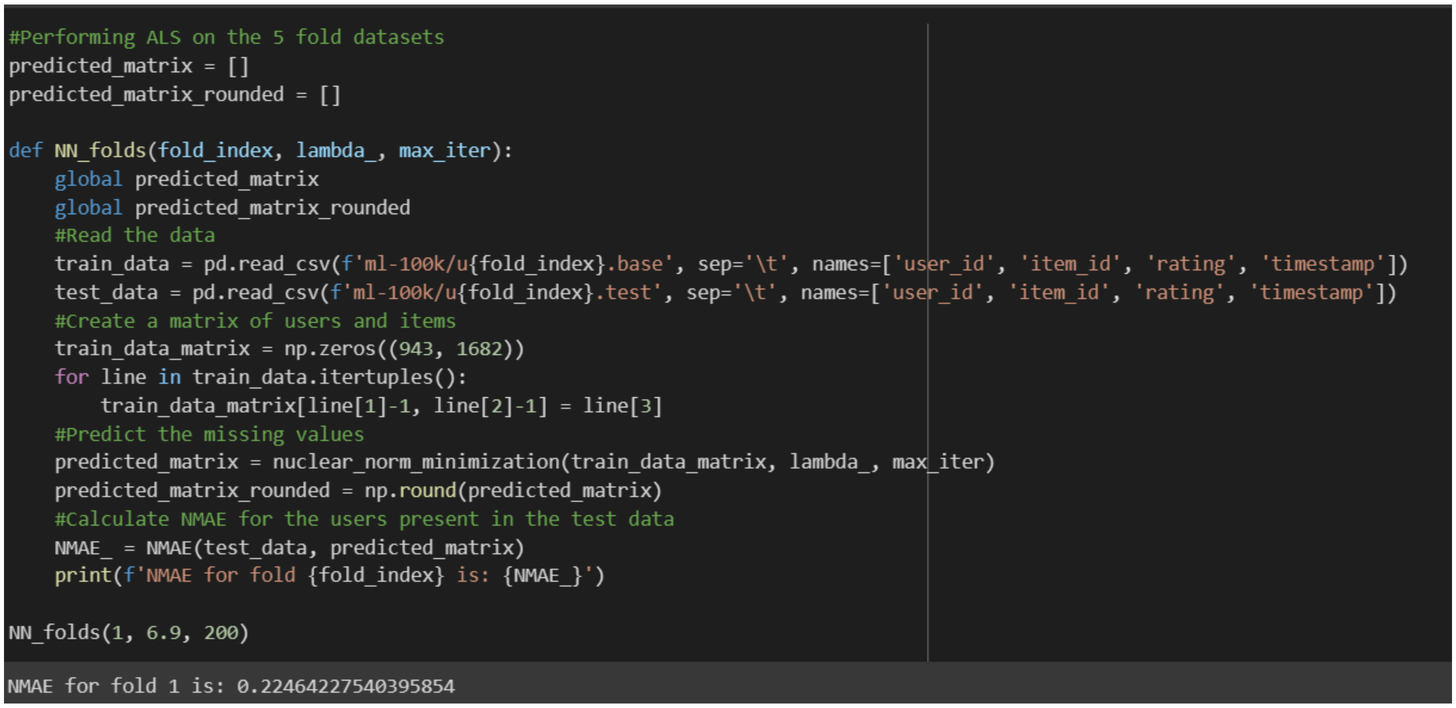
In summary, NNM is a powerful algorithm in CF for matrix completion. By exploiting the low-rank structure through nuclear norm minimization, it captures underlying patterns and produces accurate predictions, as demonstrated by its low NMAE value.

Results

Used Nuclear Norm Minimization, and used the template of my previous assignment for importing data and printing format.

NMAE is around **0.22**

Aamleen Ahmed
2020002



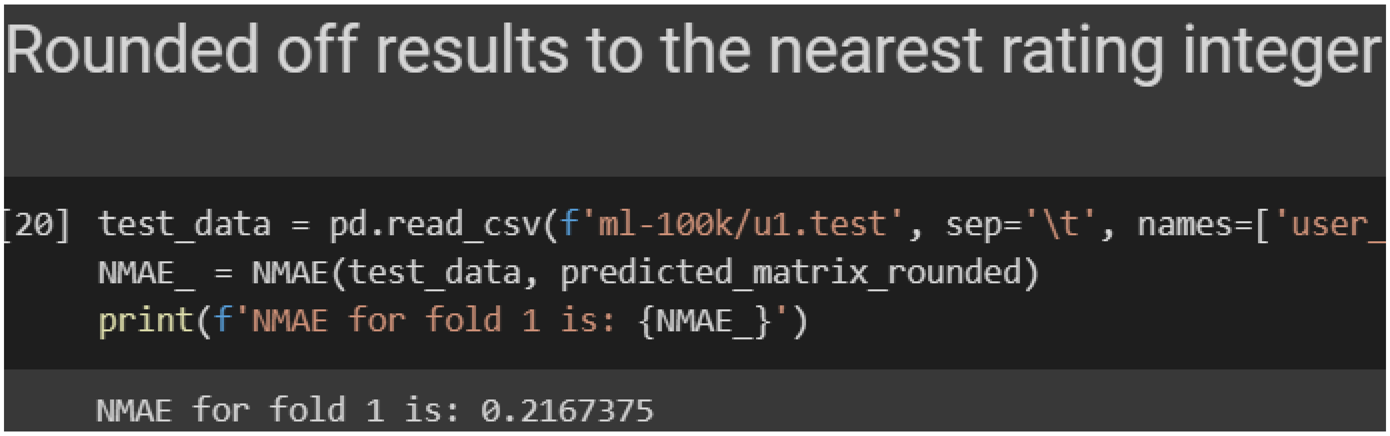
```
#Performing ALS on the 5 fold datasets
predicted_matrix = []
predicted_matrix_rounded = []

def NN_folds(fold_index, lambda_, max_iter):
    global predicted_matrix
    global predicted_matrix_rounded
    #Read the data
    train_data = pd.read_csv(f'ml-100k/u{fold_index}.base', sep='\t', names=['user_id', 'item_id', 'rating', 'timestamp'])
    test_data = pd.read_csv(f'ml-100k/u{fold_index}.test', sep='\t', names=['user_id', 'item_id', 'rating', 'timestamp'])
    #Create a matrix of users and items
    train_data_matrix = np.zeros((943, 1682))
    for line in train_data.itertuples():
        train_data_matrix[line[1]-1, line[2]-1] = line[3]
    #Predict the missing values
    predicted_matrix = nuclear_norm_minimization(train_data_matrix, lambda_, max_iter)
    predicted_matrix_rounded = np.round(predicted_matrix)
    #Calculate NMAE for the users present in the test data
    NMAE_ = NMAE(test_data, predicted_matrix)
    print(f'NMAE for fold {fold_index} is: {NMAE_}')

NN_folds(1, 6.9, 200)
```

```
NMAE for fold 1 is: 0.22464227540395854
```

NMAE is around **0.21** for fold1



Rounded off results to the nearest rating integer

```
[20] test_data = pd.read_csv(f'ml-100k/u1.test', sep='\t', names=['user_
     NMAE_ = NMAE(test_data, predicted_matrix_rounded)
     print(f'NMAE for fold 1 is: {NMAE_}')
```

```
     NMAE for fold 1 is: 0.2167375
```