Information Retrieval and Recommender Systems – HW2 report

Students IDs: 200320836, 301796447, 301406641

1.3 Baselines:

- Simple mean:
  For each user we calculated the mean of his rankings using Groupby() function by User. We saved the result in a Dictionary of users and their corresponding ratings mean.

- Linear regression baseline:
  For each user and item we calculated a prediction using the formula:
  $\hat{r}_{ui} = b_{ui} = mu + b_u + b_i$
  So that mu is computed by calculating the mean of all ratings and b_u and b_i by SGD that minimizes the least squares formula with regularization according to the objective function.
  At the end of this stage we saved two lists – one for user biases and one for item biases to be used in the 'knn baseline' algorithm.

  To improve the Linear regression algorithm performance we used:
  1. Clipping – making sure the prediction is between 1-5 (acceptable values) which should help with reducing the loss for items that received other values.
  2. Learning rate decay – in each epoch after the first one, we multiplied the learning rate by a factor to reduce it and help with the convergence of the model.
     First steps are 'larger' while last steps are 'smaller' in comparison.

  To improve the Linear regression efficiency:
  1. When calculating the regularization, we used vector operators on numpy arrays which are more efficient than calculating iteratively.

1.4 Neighborhood models:

- Simple Knn:
  We calculated the prediction according to the formula: $\hat{r}_{ui} = \frac{\sum_{j\ in\ N} sim(i,j) \cdot r_{uj}}{\sum_{j\ in\ N} sim(i,j)}$
  Where N is the k nearest neighbors of item i that is rated by user u.
  In the fit function we processed the correlation matrix of all items using the build_item_to_itm_corr function, which uses numpy's corrcoef() function which results in a pearson's correlation matrix. In case this process was previously done, we upload the existing CSV file that contain these values. In predict_on_pair function we save all the rankings that the user rated to the variable user_rated_items and save all the similarities that the input item has to the variable input_item_similarities. Then, we joined between the two variables so that we have all the item similarities that are correlated to items that the user rated. We sorted the data frame according to the similarity and selected the k items with highest similarity. Finally, we calculated the prediction according to the formula above.

- Baseline Knn: We calculated the predictions according to the formula: $\hat{r}_{ui} = b_{ui} + \frac{\sum_{j\ in\ N} sim(i,j) \cdot (r_{uj} - b_{uj})}{\sum_{j\ in\ N} sim(i,j)}$ using item similarities and b_{ui} that were computed in the preliminary steps.
  We calculated b_{ui} and saved it to user_input_item_bias variable. We created a data frame with all of the user details (items & rankings). We added to the data frame an additional column that calculated the user and the items bias for the items that the user had rated (b_{uj}). Then we proceed in the same manner we did in the Simple KNN algorithm, Finishing with the prediction calculation according to the formula above.

To improve both KNN algorithms performance we used:

1. Clipping – making sure the prediction is between 1-5 (acceptable values) which should help with reducing the loss for items that received other values.
2. Prediction on new items/users – in case a new item is presented we predicted by using the specific user average ratings and in case a new user is presented we predicted by using the global mean.

To improve efficiency:

1. When predicting each pair, we used arrays which allowed us to use vector operators. Because these actions occur a lot of times, the saving is substantial.
2. As requested, we upload the csv file previously created in the Simple KNN step, which saves the times of computing it again.

1.5 Matrix factorization:

In this algorithm we tried to minimize the objective function:
Min $\sum_{r_{ui}} (r_{ui} - \hat{r}_{ui})^2 + \gamma(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2)$.
In each epoch, we are updating the four main parameters: latent matrices U and V, users and items biases bu and bi according to the following update steps:
update steps:
$\mathrm{X}u \leftarrow \mathrm{X}u + \gamma(eui \cdot \mathrm{Y}i - \lambda \cdot \mathrm{X}u)$
$\mathrm{Y}i \leftarrow \mathrm{Y}i + \gamma(eui \cdot \mathrm{X}u - \lambda \cdot \mathrm{Y}i)$
$bu \leftarrow bu + \gamma(eui - \lambda * bu)$
$bi \leftarrow bi + \gamma(eui - \lambda * bi)$
Where $eui = rui - \hat{r}ui = rui - (\mathrm{X}uT\mathrm{Y}i+bu+bi+\mu)$
Then we calculated the prediction for pair of user and item with the following formula:
Prediction(u,i) = $\mathrm{X}uT\mathrm{Y}i + bu + bi + \mu$
After each epoch we calculated the rmse and the train objective to see the improvement of the model.

To improve the matrix factorization algorithm performance we used:

1. Clipping – making sure the prediction is between 1-5 (acceptable values) which should help with reducing the loss for items that received other values.
2. Prediction on new items/users – in case a new item is presented we predicted by using the specific user average ratings and in case a new user is presented we predicted by using the global mean.

To improve efficiency:

1. When predicting each pair, we used arrays which allowed us to use vector operators. Because these actions occur a lot of times, the saving is substantial.
2. In the first epochs (total number of epochs * 0.25) we calculated only the biases. This helps with the performance but also result in much less steps overall.

Summary of RMSE of each algorithm:

Algorithm	**RMSE**
Simple Mean	1.049310719086559
Linear regression	0.9752242704334725
Simple Knn	0.9657809670119639
Baseline Knn	0.9177010201511774
Matrix factorization	0.9296060921174085