ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

PROJECT MILESTONE 1: PERSONALIZED RECOMMENDER WITH K-NN

CS-449 SYSTEMS FOR DATA SCIENCE

Irina-Madalina Bejan
Siran Li

25th March 2022

1 Motivation: Movie Recommender

Question 3

B.1 Average rating and prediction

The global average rating $\overline{r}_{\bullet,\bullet}$ is calculated as following,

$$\overline{r}_{\bullet,\bullet} = \sum_{r_{u,i}} \frac{r_{u,i}}{|U(r)|} \tag{1}$$

where $|U(r)|$ is the set of all ratings in the dataset. The results (rounded to 4 decimal places) is $\overline{r}_{\bullet,\bullet} = 3.5265$

The global average rating for users $\overline{r}_{u,\bullet}$ is calculated as following,

$$\overline{r}_{u,\bullet} = \sum_{i \in I(u)} \frac{r_{u,i}}{|I(u)|} \tag{2}$$

where $|I(u)|$ is the set of items with a rating for user u in the dataset. The average rating for user 1 ($\overline{r}_{1,\bullet}$) (rounded to 4 decimal places) is $\overline{r}_{1,\bullet} = 3.6330$

The global average rating for items $\overline{r}_{\bullet,i}$ is calculated as following,

$$\overline{r}_{\bullet,i} = \sum_{u \in U(i)} \frac{r_{u,i}}{|U(i)|} \tag{3}$$

where $|U(i)|$ is the set of users with a rating for item i in the dataset. The average rating for item 1 ($\overline{r}_{\bullet,1}$) (rounded to 4 decimal places) is $\overline{r}_{\bullet,1} = 3.8883$

The global average deviation for all users on each item is calculated and we get the average deviation for user 1 (the result is rounded to 4 decimal places) $\overline{\widehat{r}}_{\bullet,1} = 0.3027$

The predicted rating of each user for each item is calculated and we get the predicted rating of user 1 for item 1 (the result is rounded to 4 decimal places), $p_{1,1} = 4.0468$

B.2 Prediction accuracy

The prediction accuracy of previous methods ($\overline{r}_{\bullet,\bullet}$, $\overline{r}_{u,\bullet}$, $\overline{r}_{\bullet,i}$, $p_{u,i}$) is computed. The results rounded to 4 decimal places are shown in the following table.

Method	MAE
Global Average	0.9489
User Average	0.8383
Item Average	0.8207
Baseline	0.7604

TABLE 1: Table of prediction accuracy of previous methods.

B.3 Time required for computing MAE

The time required for computing MAE for all ratings in the test set with all four methods are shown in the following table and figure. They were ran locally on a Dell Inc. XPS 13 9370 that has a Intel Core i7-8550u processor with 8 cores of 1.80Ghz, a RAM of 15.4, on Ubuntu Budgie using Scala 1.6.2 and OpenJDK version 1.8.0.

Method	Average	Standard deviation
Global Average	2.6199	0.05344
User Average	17.7264	9.1902
Item Average	19.6398	4.1001
Baseline	120.7088	11.9761

TABLE 2: Average and standard deviation of the ten measurements of time for computing the predictions for each method (in microseconds).

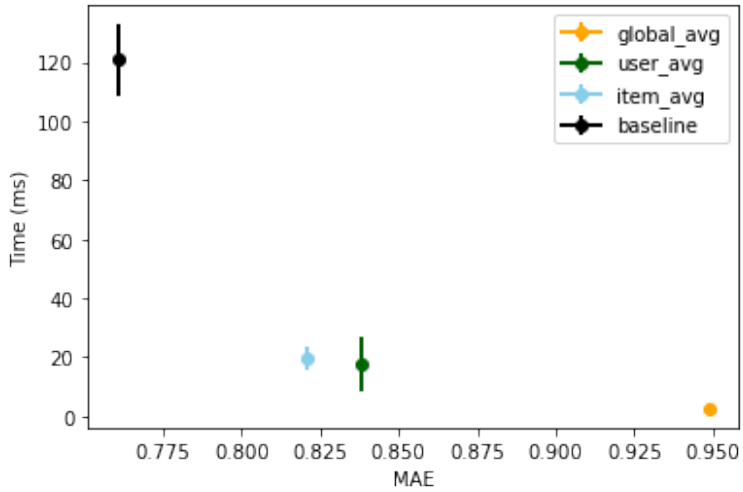

FIGURE 1: MAE for different prediction methods.

From the above table and figure results we can get that the most expensive method to compute is the baseline method. According to the figure 1 the method with higher MAE needs longer time to compute, but the relationship between MAE and computation is not linear. Based on the above observation, we can get conclusion that more accurate prediction methods need more computing time.

4 Spark Distribution Overhead

Question 4.1

D.1 Average rating and prediction

Using Spark RDD to compute the average ratings and prediction, we get the same results as the part **B.1**:

(1) the global average rating: $\overline{r}_{\bullet,\bullet} = 3.5265$
(2) the average rating for user 1: $\overline{r}_{1,\bullet} = 3.6330$
(3) the average rating for item 1: $\overline{r}_{\bullet,1} = 3.8883$
(4) the average deviation for item 1: $\widehat{r}_{\bullet,1} = 0.3027$
(5) the predicted rating of user 1 for item 1: $p_{1,1} = 4.0468$
(6) the prediction accuracy (average MAE) of the proposed baseline: MAE = 0.7604

D.2 Computation time

We made 3 experiments (predict.Baseline, distributed.Baseline 1 worker and distributed.Baseline 4 workers) and ran each 3 times locally (for the predict.Baseline) and for the Spark experiments on the iccluster028. The local tests were performed on a Dell Inc. XPS 13 9370 that has a Intel Core i7-8550u processor with 8 cores of 1.80Ghz, a RAM of 15.4, on Ubuntu Budgie using Scala 1.6.2 and OpenJDK version 1.8.0. Below we reported the results for the runs:

To running the the computation time, we ran locally

Experiment	Number of workers	timeMs Average	timeMs Standard Deviation
predict.Baseline	-	65309.84	1259.08
distributed.Baseline	1	68884.89	5952.92
distributed.Baseline	4	23496.26	439.46

TABLE 3: Top 3 recommendations with k = 300.

If we compare the obtained results, we see that the predict.Baseline is slightly faster/in the same ballpark with the distributed.Baseline ran using only one worker, and we think that we don't benefit much from the distributed computation when using only one worker. In comparison, we have a 3x faster solution by using 4 workers, getting 23.5s.

The current version of the Scala baseline has been optimized by avoiding 'unnecessary' groupBy operations which we saw to be driving the cost of the computation and switched from using Array of tuples (Fig 2) to store the average rating per item, average rating per user and average global deviation per item to using Map (Fig 3), with more efficient lookup operation in comparison with the Scala's for comprehension and yield operation for matching key values, as illustrated below.

```
val extended_train = for {
  r: Rating <- train
  (u, avg_r) <- avg_rating_per_users
  if r.user == u
} yield (r.user, r.item, r.rating, avg_r)
```

FIGURE 2: Implementation before using Arrays.

```
def computeNormalizedDeviation(
    input: Array[Rating],
    avgRatingPerUser: Map[Int, Double]
): Array[(Int, Int, Double)] = {
  return input.map { r =>
    (
      r.user,
      r.item,
      normalize(r.rating, avgRatingPerUser.getOrElse(r.user, 0.0))
    )
  }
}
```

FIGURE 3: Implementation after using Map.

For the Spark implementation, we saw that using built-in function like .mean() was less performant than implementing from scratch the mean by using a map -> reduceByKey approach, which saved us around 5s, as shown below (Fig 4):

```scala
def maeRDD(test: RDD[Rating], predictor: (Int, Int) => Double): Double = {
  val result = test
    .map({ case r: Rating =>
      (scala.math.abs(r.rating - predictor(r.user, r.item)), 1)
    })
    .reduce((x, y) => (x._1 + y._1, x._2 + y._2))

  return result._1 / result._2
}
```

FIGURE 4: Implementation of the mean calculation

Moreover, we saw we have two options, as in the baseline to use the averages in our computation, either by a series of map join operation as illustrated in Fig 5. or by grouping and collecting the actual Map, since the amount of users/items is in the range of a few thousand (Fig. 6). We noticed the latter was faster and managed to optimize it further by broadcasting the maps to all workers and thus accessing it should be more efficient, which brought us a few more seconds.

```scala
return test_data.map{ case r: Rating => (r.user, r)}.leftOuterJoin(avg_rating_per_users)
.map{ case (user, (r, avg_rating_per_users)) => (r.item, (r, avg_rating_per_users.getOrElse(global_avg)))}
.leftOuterJoin(global_avg_deviation_per_item)
.map{case (item, ((r, r_u), r_i)) => (r.user, r.item, r.rating, r_u, r_i.getOrElse(r_u))}
.map{case (user, item, rating, r_u, r_i) => (user, item, rating, r_u + r_i * scale((r_u + r_i), r_u))}
```

FIGURE 5: Implementation using joins

```scala
val avgRatingPerUsersMap = computeAvgRatingPerUsersMapRDD(train)
val globalAvgDevPerItem = computeGlobalDevPerItemRDD(train, avgRatingPerUsersMap)

val broadcaster_avgRatingPerUsersMap =
  train.sparkContext.broadcast(avgRatingPerUsersMap)
val broadcaster_avgDevPerItem =
  train.sparkContext.broadcast(globalAvgDevPerItem)

(u: Int, i: Int) => {
  if (broadcaster_avgRatingPerUsersMap.value.contains(u)) {
    val r_u = broadcaster_avgRatingPerUsersMap.value.getOrElse(u, globalAvg)
    val r_i = broadcaster_avgDevPerItem.value.getOrElse(i, 0.0)
```

FIGURE 6: Implementation using broadcasts of maps

5 Personalized Predictions

Question 5.3

P.1 Prediction accuracy with uniform similarities

Using uniform similarities of 1 between all users means that we treat different users for each items with the same weight. So the predicted rating of user 1 for item 1 is the same as the result from baseline, $p_{1,1} = 4.0468$ and the prediction accuracy is also the same as previous baseline result $MAE = 0.7604$ (MAE on ml-100k/u2.test).

P.2 Cosine similarity

Following the functions provided in the requirements, we build a matrix for the preprocessed rating $\breve{r}_{u,i}$. The row index of the matrix represents (user_id - 1) and the column index represents (item_id - 1) since both the user_id and the item_id count from 1. After changing the collections to the matrix, we can also change the equation for computing similarities to matrix convolution as following

$$s_{u,v} = \sum_{i \in (I(u) \bigcap I(v))} \breve{r}_{u,i} * \breve{r}_{v,i} \implies S_{uv} = \breve{R}_{ui} \circledast \breve{R}^T_{vi} \tag{4}$$

where S_{uv} represents the matrix of similarities between different users, the row index and the column index represent (user_id - 1) of two users respectively, $\breve{R}_{ui}$ represents the preprocessed rating ($\breve{r}_{u,i}$) matrix, and $\breve{R}^T_{vi}$ represents the transposed matrix of preprocessed rating ($\breve{r}_{v,i}$), and $\circledast$ represents convolution.

Then for the user-specific weighted-sum deviation we also use matrix for calculation. For the following equation,

$$\overline{\widehat{r}}_{\bullet,i}(u) = \begin{cases} \frac{\sum_{v \in U(i)} s_{u,v} * \widehat{r}_{u,i}}{\sum_{v \in U(i)} |s_{u,v}|} & \exists_{v \in U(i)} s_{u,v} \neq 0 \\ 0 & otherwise \end{cases} \tag{5}$$

first we compute the numerator $\sum_{v \in U(i)} s_{u,v} * \breve{r}_{u,i}$ and denominator $\sum_{v \in U(i)} |s_{u,v}|$ separately. the functions are changed as following:

$$\sum_{v \in U(i)} s_{u,v} * \widehat{r}_{u,i} \Longrightarrow S_{uv} \circledast \widehat{R}_{vi} \tag{6}$$

where $\widehat{R}_{vi}$ represents the normalized deviation $\widehat{r}_{vi}$ matrix, and

$$\sum_{v \in U(i)} |s_{u,v}| \Longrightarrow S_{uv} \dot{\circledast} \widehat{I}_{vi} \tag{7}$$

where $\widehat{I}_{vi}$ represents matrix using $\widehat{R}_{vi}$ by replacing nonzero values to 1 (where $\widehat{r}_{vi}$ exists we set the corresponding value in $\widehat{I}_{vi}$ 1), and $\dot{\circledast}$ represents the convolution of absolute values.

So the results of the two functions 6 and 9 are two matrices, which have the same size $U * I$, where U represents the number of users and I represents the number of items. Then we divide the element of 6 by the corresponding element of 9, and get the results of 5, $\overline{\widehat{R}}_{\bullet,i}(u)$.

The prediction value is calculated by the following equation

$$p_{u,i} = \overline{r}_{u,\bullet} + \overline{\widehat{R}}_{\bullet,i}(u) * scale((\overline{r}_{u,\bullet} + \overline{\widehat{R}}_{\bullet,i}(u)), \overline{r}_{u,\bullet}) \tag{8}$$

Following the above description we can get the similarity between user 1 and user 2, $s_{1,2} = 0.07303$, and the predicted rating of user 1 and item 1 $p_{1,1} = 4.0870$ and the prediction accuracy (MAE on ml-100k/u2.test) of the personalized baseline predictor $MAE_{cosine\_sim} = 0.7372$

P.3 Jaccard similarity

The Jaccard Coefficient is defined as follows:

$$j_{u,v} = \frac{i \in I(u) \cap I(v) |U(i)|}{j \in I(u) \cup I(v) |U(j)|} \tag{9}$$

where U(i) is the set of users with a rating for item i. Then we use the same prediction funtion with the jaccard similarity and get similarity between user 1 and user 2, $s_{1,2} = 0.03187$, and the predicted rating of user 1 and item 1 $p_{1,1} = 4.0982$ and the prediction accuracy (MAE on ml-100k/u2.test) of the personalized baseline predictor $MAE_{jaccard\_sim} = 0.7556$

Compared the Jaccard Coefficient similarity and the Adjusted Cosine similarity, the prediction accuracy MAE of consine similarity is lower than the method with Jaccard Coefficient. The Adjusted cosine similarity performs better than Jaccard Coefficient because cosine similarity keeps the information of rating values while the Jaccard similarity only keeps the number of the set and can not present their rating values correlation.

6 Neighbourhood-Based Predictions

Question 6.1

N.1 k-NN predictor

Implementing the k-NN predictor using the adjusted cosine similarity and using $k = 10$, we can get the following output:

(1) similarity between user 1 and it self: $s_{1,1} = 0$
(2) similarity between user 1 and user 864: $s_{1,864} = 0.2423$
(3) similarity between user 1 and user 886: $s_{1,886} = 0$
(4) prediction for user 1 and item 1: $p_{1,1} = 4.3191$

N.2 k-NN predictor MAE

The prediction accuracy MAE on data/ml-100k/u2.test rounded to the fourth decimal point for $k = 10, 30, 50, 100, 200, 300, 400, 800, 942$ are reported in the following table.

k	MAE
10	0.8287
30	0.7810
50	0.7618
100	0.7467
200	0.7401
300	0.7392
400	0.7390
500	0.7385
943	0.7372

TABLE 4: MAE for the different k values.

From 1 we know that the MAE for the baseline (non-personalized) method is 0.7604. Compared with the MAE from baseline, the MAE lower than it in 4 is 0.7467 with the lowest k, $k = 100$. We notice from the table above that an accurate prediction requires many data points as neighbours (all of them in our case) and we believe it is because the dataset is small or it is possible the similarity measure is not a good fit for our prediction.

N.3 Time required for computing predictions

The time in Ms for running the KNN on the 100k dataset and a k = 300 is 3.7s. Our initial version of the code followed a similar approach as the baseline (relying of map and groupBy operations) which was very inefficient, taking 72s. Thus, we derived the matrix multiplication formulation we presented above to do the computation and optimized different iterations using forEach closures whenever possible to facilitate parallel computation, as well as .par.

7 Recommendation

Question 7.1

R.1 k-NN predictor for the potential user

For the augmented dataset with additional ratings from user "994' provided in personal.csv, we train the k-NN predictor using adjusted cosine similarity and $k = 300$, the prediction for user 1 item 1 is $p_{1,1} = 4.1322$ (the result is rounded to the fourth decimal point.

R.2 Top 3 recommendations

The top 3 recommendations for user "994" using the k-NN predictor with $k = 300$ are shown in the following table:

ID	Movies	Predicted rating
119	Maya Lin: A Strong Clear Vision (1994)	5
814	Great Day in Harlem	5
1189	Prefontaine (1997)	5

TABLE 5: Top 3 recommendations with k = 300.