CS-449 Systems for Data Science

Project Milestone 2

Due 30/04/2021

Author
DHAENE Arnaud

Professor
KERMARREC Anne-Marie

Teaching Assistants
LAVOIE Erick
XYGKIS Athanasios

General comments

Throughout the entirety of my report, I have rounded my results to the fourth decimal point.

I have decided to follow the suggested implementation, which is to use batch calculation for similarities before running the predictions. While in the first hours of tackling the problem, I chose to use an RDD-based implementation, I decided to follow the initial suggestion before it was officially overturned. As my compute times are quite low, I stuck to my Scala collections.

When running all three classes on my machine, whose technical specifications are detailed in Section 3.3 of this report, I get the following compute times:

- `similarity.Predictor` runs in 183 seconds (equivalent to 3 minutes and 3 seconds).
- `knn.Predictor` runs in 123 seconds (equivalent to 2 minutes and 3 seconds).
- `recommend.Recommender` runs in 53 seconds.

This yields a total compute time of 6 minutes and 29 seconds.

Similarity-based Predictions

2.1 Cosine-based similarity

The cosine similarity-based Mean Average Error (MAE) is 0.7489. The difference between the adjusted cosine similarity and the baseline of 0.7669 is -0.0180, indicating that the prediction accuracy is better using cosine similarity than the baseline method from Milestone 1.

2.2 Jaccard coefficient as a similarity metric

Using the Jaccard Coefficient, I used the similarity metric defined as follows:

$$j_{u,v} = \frac{|I(u) \cap I(v)|}{|I(u) \cup I(v)|} \tag{2.1}$$

The prediction accuracy I obtain using the above-defined similarity metric is 0.7634. The calculated difference between the Jaccard Coefficient and Adjusted Cosine similarity methods is 0.0145, indicating that the Jaccard Coefficient method performs worse than the Adjusted Cosine similarity method.

2.3 Computations with respect to U

Let us call the matrix of similarities S, with the entry (u, v) being $S_{u,v} = s_{u,v}$ as defined in the Milestone's guidelines.

In the worst case for any dataset of size U, if every user rated at least one item in common with every other use, the number of similarities that need to be computed follows the following equation:

$$|S| = U^2 \tag{2.2}$$

The above equation states that, for a dataset of `ml-100k`'s size (i.e. 943 users), the full matrix of similarities would have size 943x943 and thus 889249 elements.
However, this is not (at all) an optimized solution. In fact, since $s_{u,v} = s_{v,u}$ and the self-similarities $s_{u,v} \forall (u, v) | u = v$ are of no use to us, we can calculate the number of computations as a function of the amount of users using the Binomial coefficient. In fact, this coefficient in combinatorics indicates how many different subsets of k elements (in our case, $k = 2$) can be chosen from a fixed set of n elements (in our case, $n = U$). The theorem states the following:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{2.3}$$

Applying this theorem to our use-case, with $k = 2$ and $n = U = 943$, we have that

$$\binom{943}{2} = \frac{943!}{2!(941)!} = 444153 \tag{2.4}$$

To conclude, 889249 similarity computations need to be performed for our `ml-100k` dataset in the worst case described in the question. However, an optimized solution needs to compute 444153 similarity computations for our `ml-100k` dataset.

Additional note For the following section concerning Neighbourhood-Based Predictions, the computation of similarities cannot be optimized as we need to sort the top k similarities per user u. This means that in the worst case, we will need to compute $943 \cdot 942 = 888306$ similarities (for $k = 942$ as self-similarities are filtered out).

2.4 Computations required for each similarity metric

Using $|I(u) \cap I(v)|$ as the minimum number of multiplications required for each similarity $s_{u,v}$, we have the following statistics for `ml-100k/u1.base` (for the non-optimized and optimized version described above in Question 2.3 respectively):

Statistic	**value**
Minimum	0
Maximum	685
Average	12.2051
Standard deviation	18.1752

Table 1: Statistics concerning the minimum number of multiplications required for calculating the non-optimized 889249 similarities $s_{u,v}$ in `ml-100k/u1.base`.

Statistic	**value**
Minimum	0
Maximum	332
Average	12.1280
Standard deviation	17.8223

Table 2: Statistics concerning the minimum number of multiplications required for calculating the optimized 444153 similarities $s_{u,v}$ in `ml-100k/u1.base`.

2.5 Memory required for storing similarities

The amount of bytes needed to store a Scala `Double` is 8 (as they are 64-bit floating point values). There are 65214 zero similarity values. This would entail that the memory required to store all similarities is:

- $889249 \cdot 8 = 7113992$ bytes, for both zero and non-zero values.
- $824035 \cdot 8 = 6592280$ bytes, for only non-zero values.

For the sake of consistency, let us compute these values for the optimized case as well. In this case, there are 33607 zero similarity values. This would entail that the memory required to store all similarities is:

- $444153 \cdot 8 = 3553224$ bytes, for both zero and non-zero values.
- $411546 \cdot 8 = 3292368$ bytes, for only non-zero values.

2.6 Prediction compute time

The average time required for computing predictions with 1 Spark executor is 22.9537 seconds. The detailed statistics, rounded to the fourth decimal point, can be observed in the table below.

Task	**Minimum**	**Maximum**	**Average**	**Standard deviation**
Similarities	13.0800	15.2160	13.9905	0.7596
Total predictions	21.8248	24.0244	22.9537	0.8482

Table 3: Time required for computing predictions, in seconds.

In my implementation, I compute the similarities using batch processing, as was initially recommended in the Milestone guideline. However, in `similarity.Predictor` I calculate only the upper triangular matrix of similarities. In fact, when serving predictions, I sort the lookup key (u, v) by making sure $u > v$, which allows me to compute only 943 choose 2 = 444153 similarities, instead of the worst case of 889249.

In the previous Milestone, the average reported time for computing predictions using the baseline method was 1.4938 seconds, which is significantly less than the cosine similarity method (i.e. approximately 15 times slower). The reasons behind this huge slow-down are multiple: (1) computing the similarities for all users is computationally quite expensive, and (2) the prediction for a single user-item pair requires looking up all of said user's similarities.

2.7 Similarity compute time

The statistics related to the time it takes to compute the similarity values can be found in Table 3.

The average time per similarity computation is 30.7984 microseconds.

The average ratio between the computation of similarities and the total time required to make predictions is 0.6095. Approximately three fifths of the total prediction time is taken by computing the similarities, which is significant.

Neighbourhood-Based Predictions

3.1 Impact of k on prediction accuracy

The prediction accuracy (as the MAE on the `ml-100k/u1.test` dataset) rounded to the fourth decimal point for the different values of k given in the Milestone guidelines are reported in the table and figure below.

k	**MAE**
10	0.8418
30	0.7926
50	0.7760
100	0.7573
200	0.7496
300	0.7481
400	0.7485
800	0.7484
943	0.7489

Table 4: MAE for the different tested values of k.

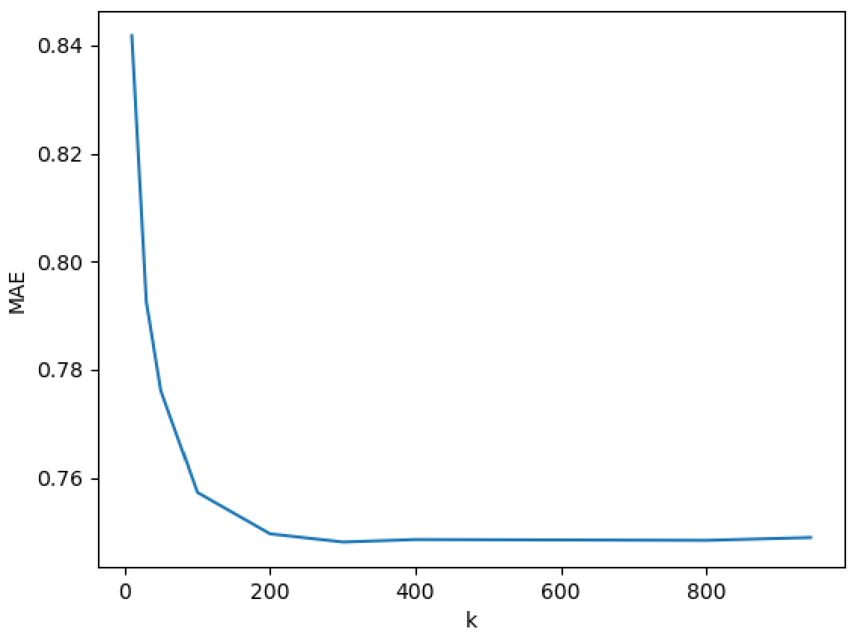

Figure 1: MAE for the different tested values of k.

It can be observed in Table 4 that the lowest k such that the MAE is lower than for the baseline method (i.e. 0.7669) is $k = 100$ for an MAE of 0.7573. The MAE is 0.0096 lower than the baseline method for the aforementioned value of k.

3.2 Memory requirements

The minimum number of bytes required, indicated as M, as a function of the size of U, to store only the k nearest similarity values for all possible users, is as follows in Equation 3.1. We assume an ideal implementation for which only similarity values are stored with 64-bit floating point values.

$$M(U, k) = U \cdot 8 \cdot k \tag{3.1}$$

The equation stems from the fact that each 64-bit floating point value takes up 8 bytes of storage, and k similarity values are stored for each user $u \in U$.

3.3 Potential users in RAM

The technical specifications of my personal computer are the following:

- **Model**: MacBook Pro (Retina, 13-inch, Early 2015)
- **OS**: macOS Big Sur version 11.1
- **Processor**: 2.9 GHz Dual-Core Intel Core i5
- **Memory**: 16 GB 1867 MHz DDR3
- **Graphics**: Intel Iris Graphics 6100 1536 MB
- **Language**: English

The lowest k I have provided in Q.3.1.1 is $k = 100$. The maximum number of users I could store in my RAM assuming I was storing 3 64-bit values per similarity can be calculated as explicited in Equation, with RAM in bytes. A RAM of 16 GB is equivalent to 17179869184 bytes.

$$U(RAM, k) = \frac{RAM}{8 \cdot 3 \cdot k} = \frac{17179869184}{8 \cdot 3 \cdot 100} = 7158278.8267 \tag{3.2}$$

When applying the equation, we find that we can store the similarities for 7158278 users (7.1583 M) in memory.

3.4 Impact of k on number of similarity computations

Varying the parameter k does not have an impact on the number of similarity values ($s_{u,v}$) to compute as we need to sort all similarity values for each user u in order to select the top k.

3.5 Personal top 5 recommendations

My personal top 5 recommendations using the baseline predictor from Milestone 1, and the k-Nearest-Neighbor predictor using $k = 30$ and $k = 300$, are respectively:

Movie	**ID**	**Predicted rating**
Great Day in Harlem	814	5.0
They Made Me a Criminal (1939)	1122	5.0
Prefontaine (1997)	1189	5.0
Marlene Dietrich: Shadow and Light (1996)	1201	5.0
Star Kid (1997)	1293	5.0

Table 5: My personal recommendations for Milestone 1 using the baseline predictor.

Movie	**ID**	**Predicted rating**
Citizen Ruth (1996)	236	5.0
Kolya (1996)	242	5.0
Pillow Book	253	5.0
Marvin's Room (1996)	287	5.0
Paradise Lost: The Child Murders at Robin Hood Hills (1996)	320	5.0

Table 6: My personal recommendations with $k = 30$.

Movie	**ID**	**Predicted rating**
Perfect Candidate	850	5.0
Year of the Horse (1997)	884	5.0
Prefontaine (1997)	1189	5.0
Marlene Dietrich: Shadow and Light (1996)	1201	5.0
The Deadly Cure (1996)	1358	5.0

Table 7: My personal recommendations with $k = 300$.

The intersection of both k-NN recommendations is void. When I serve predictions, I sort them following the movie ID before sorting them with regards to their predicted rating in order to get consistent results. In my opinion, the first recommendations (i.e. using $k = 30$) contain many movies with a predicted rating of 5.0, which indicates that looking at only the top 5 recommendations might not give a full picture of the predictions.

On the other hand, the recommendations made with $k = 300$ seem to be more relevant. In fact, two out of five recommendations are also in the recommendations from the baseline implementation from Milestone 1.