CS-449 Project Milestone 1: Personalized Recommender with k-NN

Name: Emna FENDRI
Sciper: 297286
Email: emna.fendri@epfl.ch
Name: Douglas BOUCHET
Sciper: 287906
Email: douglas.bouchet@epfl.ch

March 25, 2022

The technical specifications of the machine on which we ran the tests for this project.

- Model: MacBook Pro
- CPU speed: 2,3 GHz Intel Core i5 quad-core processor
- RAM: 8 Go 2133 MHz LPDDR3
- OS: macOS Big Sur Version 11.6
- Scala language version: 2.11.12
- JVM version: 1.8.0
- sbt version: 1.4.7

1 Motivation: Movie Recommender

(No Q)

2 Proxy Problem: Predicting Ratings

(No Q)

3 Baseline: Prediction based on Global Average Deviation

3.1 Questions

Implement the previous prediction methods using Scala's standard library, without using Spark.

$$p_{u,i} = \bar{r}_{u,\bullet} + \bar{\hat{r}}_{\bullet,i} * scale((\bar{r}_{u,\bullet} + \bar{\hat{r}}_{\bullet,i}), \bar{r}_{u,\bullet}) \quad (1)$$

B.1 *Compute and output the global average rating ($\bar{r}_{\bullet,\bullet}$), the average rating for user 1 ($\bar{r}_{1,\bullet}$), the average rating for item 1 ($\bar{r}_{\bullet,1}$), the average deviation for item 1 ($\bar{\hat{r}}_{\bullet,1}$), and the predicted rating of user 1 for item 1 ($p_{1,1}$, Eq 1) using* `data/ml-100k/u2.base` *for training. When computing the item average for items that do not have ratings in the training set, use the global average ($\bar{r}_{\bullet,\bullet}$). When making predictions for items that are not in the training set, use the user average if present, otherwise the global average.*

- Global average: $\bar{r}_{\bullet,\bullet} = 3.526462$
- Average rating for user 1: $\bar{r}_{1,\bullet} = 3.633027$
- Average rating for item 1: $\bar{r}_{\bullet,1} = 3.888268$
- Average deviation for item 1: $\bar{\hat{r}}_{\bullet,1} = 0.302707$
- Predicted rating of user 1 for item 1: 4.046819

B.2 *Compute the prediction accuracy (average MAE on* `ml-100k/u2.test`*) of the previous methods ($\bar{r}_{\bullet,\bullet}$, $\bar{r}_{u,\bullet}$, $\bar{r}_{\bullet,i}$) and that of the proposed baseline ($p_{u,i}$, Eq. 1).*

- GlobalAvgMAE: 0.948910
- UserAvgMAE: 0.838340
- ItemAvgMAE: 0.820695
- BaselineMAE: 0.760446

B.3 *Measure the time required for computing the MAE for all ratings in the test set (*`ml-100k/u2.test`*) with all four methods by recording the current time before and after (ex: with* `System.nanoTime()` *in Scala). The duration is the difference between the two.*

Include the time for computing all values required to obtain the answer from the input dataset provided in the template: recompute from scratch all necessary values even if they are available after computing previous results (ex: global average $\bar{r}_{\bullet,\bullet}$). Also ensure you store the results in some auxiliary data structure (ex: `Seq[(mae, timing)]`*) as you are performing measurements to ensure the compiler won't optimize away the computations that would otherwise be unnecessary.*

For all four methods, perform three measurements and compute the average and standard-deviation.

In your report, show in a figure the relationship between prediction precision (MAE) on the x axis, and the computation time on the y axis including the standard-deviation. Report also the technical specifications (model, CPU speed, RAM, OS, Scala language version, and JVM version) of the machine on which you ran the tests. Which of the four prediction methods is the most expensive to compute? Is the relationship between MAE and computation linear? What do you conclude on the computing needs of more accurate prediction methods?

Answer B.3. We summarize in Table 1 the result of the measurements for all four methods.

Method	Average (ms)	stddev (ms)
GlobalAvg	21.496498	4.369101
UserAvg	51.785700	17.860529
ItemAvg	47.661048	16.019771
Baseline	86.745304	29.101336

Table 1: Results on 100k test set.

We can see from Figure 1 a negative correlation between the MAE and the run-time. Which is not surprising, as more complex methods can be more precise but require more computation time. From this plot, we cannot infer a linear relationship between the MAE and the computation. In fact, UserAvg method requires more computation time than ItemAvg method and results in a larger MAE.

- **Which of the four prediction methods is the most expensive to compute?** As we can see from the table above, the baseline method was the most expensive to compute as the average of 3 measurements is roughly 86.74ms. Which is not surprising, as the baseline method requires a specific computation for every prediction i.e. for every $(user, item)$ pair.
- **What do you conclude on the computing needs of more accurate prediction methods?** The highest MAE (0.9489) is obtained using the **GlobalAvg** method, which is not surprising as this is a very generic method and consists in using the global average rating of the training set as the prediction for any $(user, item)$ pair i.e. always outputs the prediction.
  The **UserAvg** and **ItemAvg** methods give a better MAE (0.838 and 0.820 respectively), which is due to the fact that they are user oriented and item oriented respectively and are therefore less generic.

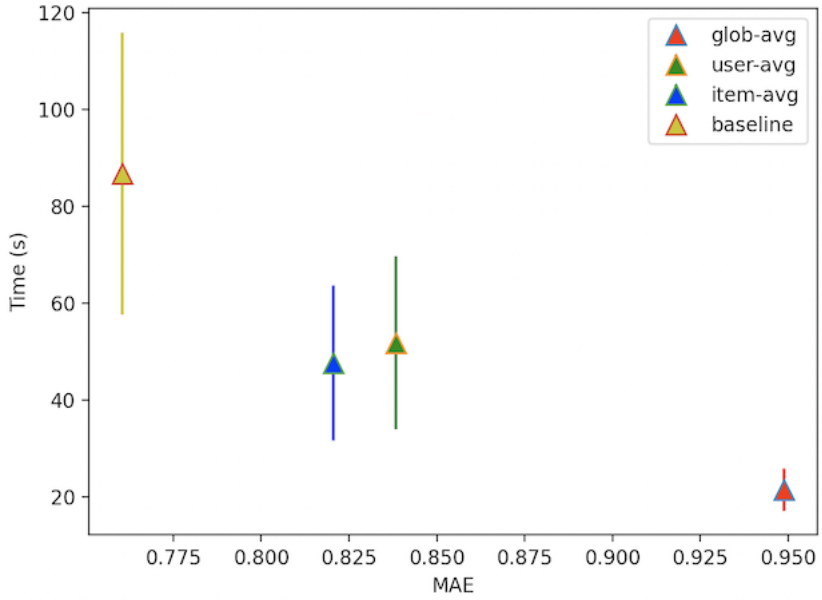

Figure 1: Relationship between MAE and the computation time including the standard-deviation.

The Baseline method gives the smallest MAE (0.760) as it takes into account the specific $(user, item)$ pair in order to compute the prediction formula. This is done by using in this formula the average rating of the given user and the global average deviation for the given item. We therefore conclude that, in order to have more accurate predictions, we need to do some specific computations about the user and the item to extract some underlying information about the given pair.

4 Spark Distribution Overhead

4.1 Questions

Implement $p_{u,i}$ using Spark RDDs. Your distributed implementation should give the same results as your previous implementation using Scala's standard library. Once your implementation works well with `data/ml-100k/u2.base` and `data/ml-100k/u2.test`, stress test its performance with the bigger `data/ml-25m/r2.train` and `data/ml-25m/r2.test`.

D.1 *Ensure the results of your distributed implementation are consistent with* ***B.1*** *and* ***B.2*** *on* `data/ml-100k/u2.train` *and* `data/ml-100k/u2.test`*. Compute and output the global average rating* ($\bar{r}_{\bullet,\bullet}$)*, the average rating for*

user 1 ($\bar{r}_{1,\bullet}$), the average rating for item 1 ($\bar{r}_{\bullet,1}$), the average deviation for item 1 ($\bar{\hat{r}}_{\bullet,1}$), and the predicted rating of user 1 for item 1 ($p_{1,1}$, Eq 1). Compute the prediction accuracy (average MAE on `ml-100k/u2.test`) of the proposed baseline ($p_{u,i}$, Eq. 1).

Answer D.1. Our distributed implementation is consistent with **B.1** and **B.2** on `data/ml-100k/u2.train` and `data/ml-100k/u2.test`. We obtain the same results:

- Global average: $\bar{r}_{\bullet,\bullet} = 3.5264625$
- Average rating for user 1: $\bar{r}_{1,\bullet} = 3.633027$
- Average rating for item 1: $\bar{r}_{\bullet,1} = 3.888268$
- Average deviation for item 1: $\bar{\hat{r}}_{\bullet,1} = 0.302707$
- Predicted rating of user 1 for item 1: 4.046819
- BaselineMAE: 0.760446

D.2 *Measure the combined time to (1) pre-compute the required baseline values for predictions and (2) to predict all values of the test set on the 25M dataset, `data/ml-25m/r2.train` and `data/ml-25m/r2.test`. Compare the time required by your implementation using Scala's standard library (**B.1** and **B.2**) on your machine, and your new distributed implementation using Spark on `iccluster028`. Use 1 and 4 executors for Spark and repeat all three experiments (predict.Baseline, distributed.Baseline 1 worker, distributed.Baseline 4 workers) 3 times. Write in your report the average and standard deviation for all three experiments, as well as the specification of the machine on which you ran the tests (similar to B.3).*

As a comparison, our reference implementation runs in 44s on the cluster with 4 workers. Ensure you obtain results roughly in the same ballpark or faster. Don't worry if your code is slower during some executions because the cluster is busy.

Try optimizing your local Scala implementation by avoiding temporary objects, instead preferring the use of mutable collations and data structures. Can you make it faster, running locally on your machine without Spark, than on the cluster with 4 workers? Explain the changes you have made to make your code faster in your report.

Answer D.2. We mention below the results of our distributed implementation on the 25M dataset. We report in Table 2 the time required by our implementations using Scala's standard library on the machine and the new distributed implementation using Spark on both the machine and `iccluster028`.

- Global average: $\bar{r}_{\bullet,\bullet} = 3.543195$
- Average rating for user 1: $\bar{r}_{1,\bullet} = 3.821428$
- Average rating for item 1: $\bar{r}_{\bullet,1} = 3.900366$

- Average deviation for item 1: $\bar{\hat{r}}_{\bullet,1} = 0.284758$
- Predicted rating of user 1 for item 1: 4.157036
- BaselineMAE: 0.834001

Implementation	Average (ms)	stddev (ms)
Standard baseline implementation	146072.144595	9867.5228626
Distributed implementation (1 worker , locally)	163974.3027	3291.8256
Distributed implementation (4 worker , locally)	49606.77529	1476.24114
Distributed implementation (1 worker , on iccluster)	40785.5029	2605.4936
Distributed implementation (4 worker , on iccluster)	35791.2091	1449.6239

Table 2: Results on 25M set.

- **Compare the time required by your implementation using Scala's standard library (B.1 and B.2) on your machine, and your new distributed implementation using Spark on `iccluster028`.**

  We can first notice that the distributed implementation with Spark is faster that the standard implementation using Scala's standard library. Secondly, we can see that running the distributed implementation on the cluster is faster than running it locally. In addition, another interesting thing to note is that the optimization is "more efficient" locally than on the cluster. In fact, multiplying by 4 the number of workers results roughly in dividing by 4 the computation time locally (from 163 to 49 seconds). Nevertheless, this does not apply for the cluster. So we can suppose that our computer makes somehow a more efficient use of its workers when applied to Spark.

- **Can you make it faster, running locally on your machine without Spark, than on the cluster with 4 workers? Explain the changes you have made to make your code faster in your report.**

  At the beginning, our code was taking an outrageous amount of time to run, which was due to our implementation. We mention below the most significant modifications and decisions we had to make:

  - **Avoid copying maps**: One big issue we had in our initial implementation was storing intermediate map before applying more operations/reductions on them. So we tried to use suitable methods such as a "foldLeft" from the trainSet to compute the values. (Sometimes using "groupBy", depending on the situation). This allows us to run the program, without memory errors.
  - **Cache values:** Originally, running the program locally was taking too much time (based on our computation it would have been 8 hours long). We spot that the access to our map with pre computed values

of items average deviation were really long (this is strange as normally access in hash maps are O(1)). We created a small "cache" map, on which each time we get the *item average deviation of an item* from our pre-computed map, we store this value inside the cache (if the value wasn't inside our "cache"). This might seems a bit surprising, but this allows us to get from a run time of 8 hours to $\sim$ 13 seconds.

- When we tried running the standard baseline on the 25M dataset with 1 measurement, we often end up with average run time of roughly 13 seconds. However when running with 3 measurements, we actually get 146 seconds. (Note that in each of this case, we were also running the 3 others predictors: globalAverage, itemAvgBased, and userAvgBased. This may be causing some ram limitation which could induce a big loss of performances.)

5 *Personalized* Predictions

5.1 Questions

$$s_{u,v} = \begin{cases} \frac{\sum_{i \in (I(u) \cap I(v))} \hat{r}_{u,i} * \hat{r}_{v,i}}{\sqrt{\sum_{i \in I(u)} (\hat{r}_{u,i})^2} * \sqrt{\sum_{i \in I(v)} (\hat{r}_{v,i})^2}} & (I(u) \cup I(v)) \neq \emptyset; \exists_{i \in I(u)} \hat{r}_{u,i} \neq 0; \exists_{i \in I(v)} \hat{r}_{v,i} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$p_{u,i} = \bar{r}_{u,\bullet} + \bar{\hat{r}}_{\bullet,i}(u) * scale((\bar{r}_{u,\bullet} + \bar{\hat{r}}_{\bullet,i}(u)), \bar{r}_{u,\bullet}) \tag{3}$$

P.1 *Using uniform similarities of 1 between all users, compute the predicted rating of user 1 for item 1 ($p_{1,1}$) and the prediction accuracy (MAE on `ml-100k/u2.test`) of the personalized baseline predictor.*

Answer P.1.

Using a uniform similarity of 1 for all users is equivalent to the original Baseline method, in fact this makes $\bar{\hat{r}}_{\bullet,i}$ and $\bar{\hat{r}}_{\bullet,i}(u)$ equivalent. We therefore obtain the same results as in part **B**:

- PredUser1Item1: 4.046819
- OnesMAE: 0.760446

P.2 *Using the the adjusted cosine similarity (Eq. 2), compute the similarity between user 1 and user 2 ($s_{1,2}$), the predicted rating of user 1 for item 1 ($p_{1,1}$ Eq. 3) and the prediction accuracy (MAE on `ml-100k/u2.test`) of the personalized baseline predictor.*

Answer P.2.

- AdjustedCosineUser1User2: 0.073037
- PredUser1Item1: 4.087027

– AdjustedCosineMAE": 0.737231

P.3 *Implement the Jaccard Coefficient*[1]*. Provide the mathematical formulation of your similarity metric in your report. User the jaccard similarity, compute the similarity between user* 1 *and user* 2 *($s_{1,2}$), the predicted rating of user 1 for item 1 ($p_{1,1}$ Eq. 3) and the prediction accuracy (MAE on* `ml-100k/u2.test`*) of the personalized baseline predictor. Is the Jaccard Coefficient better or worst than Adjusted Cosine similarity?*

Answer P.3.

For this question, we implement the Jaccard Coefficient as our metric similarity between two users as follows:

$$sim(u, v) = \frac{|I(u) \cap I(v)|}{|I(u) \cup I(v)|}$$

Where $I(u)$ is the set of items for which user u has a rating.

– JaccardUser1User2: 0.031872
– PredUser1Item1: 3.565011
– JaccardPersonalizedMAE: 0.755613

Is the Jaccard Coefficient better or worst than Adjusted Cosine similarity?

We obtain a lower MAE using the Adjusted Cosine Similarity (0.737) than using the Jaccard Coefficient (0.755). From this result we can say that the Adjusted Cosine Similarity gives better results. The Jaccard coefficient measures the similarity between 2 users solely based on the number of rated items that they have in common, yet, 2 users can rate the exact same set of items while giving opposite ratings (e.g. first user rates all items with a 5 and the other user rates them with a 1). This example shows a similarity of 1 between 2 users that have opposite tastes, for this reason the Jaccard Coefficient may lead to a less better accuracy.

6 Neighbourhood-Based Predictions

6.1 Questions

N.1 *Implement the k-NN predictor. Do not include self-similarity in the k-nearest neighbours. Using* $k = 10$*,* `data/ml-100k/u2.base` *for training output the similarities between: (1) user* 1 *and itself; (2) user* 1 *and user* 864*; (3) user* 1 *and user* 886*. Still using* $k = 10$*, output the prediction for user 1 and item 1 ($p_{1,1}$), and make sure that you obtain an MAE of* 0.8287 ± 0.0001 *on* `data/ml-100k/u2.test`*.*

Answer N.1. Results for k = 10:

[1] `https://en.wikipedia.org/wiki/Jaccard_index`

- Similarity user 1 and 1: 0
- Similarity user 1 and 864: 0.242323
- Similarity user 1 and 886: 0
- prediction user 1 item 1: 4.319093
- MAE on *u2.test*: 0.828727

N.2 *Report the MAE on* `data/ml-100k/u2.test` *for* $k = 10, 30, 50, 100, 200, 300, 400, 800, 942$. *What is the lowest k such that the MAE is lower than for the baseline (non-personalized) method?*
Answer N.2. We obtained the following MAE:

- k = 10: 0.828727
- k = 30: 0.781015
- k = 50: 0.761817
- k = 100: 0.746693
- k = 200: 0.740057
- k = 300: 0.739156
- k = 400: 0.739127
- k = 800: 0.739236
- k = 943: 0.737231

What is the lowest k such that the MAE is lower than for the baseline (non-personalized) method?

We can see that the MAE is strictly decreasing as a function of k (except for $k = 400$ and 800). The baseline method has an MAE of 0.760446. So the lowest k such that KNN-MAE is lower than the baseline MAE is between $\mathbf{k = 50}$ and $\mathbf{k = 100}$, so we compute the error for these values and conclude that this is true for $\mathbf{k = 53}$.

N.3 *Measure the time required for computing predictions (without using Spark) on* `data/ml-100k/u2.test`. *Include the time to train the predictor on* `data/ml-100k/u2.base` *including computing the similarities* $s_{u,v}$ *and using* $k = 300$. *Try reducing the computation time with alternative implementation techniques (making sure you keep obtaining the same results). Mention in your report which alternatives you tried, which ones were fastest, and by how much. The teams with the correct answer and shortest times on a secret test set will obtain more points on this question.*

Answer N.3.
The time required based on 3 samples is:

- average timing: 21.686 seconds
- stddev: 7.794 seconds

To obtain this results we used the following optimization:

- When computing the predictions, we give as an argument to our function an array of distinct users instead of the complete training data. This contributed to speedup the **mae computation** by a factor 4
- We first try to compute the similarity between all users and store it in a Map. We however change it to a more practical approach by only computing the similarity between two users when needed, but still store the results inside a Map to avoid recomputing values. The training time have been greatly reduced as we don't compute similarity during training but rather when making the predictions for the test Set. Overall this implementation was ~ 1.5 times faster.
- We also decided to "cache" the nearest neighbors of a user upon computing them. They will be the same for the same user, no matter the given item we want to predict. (Of course, the user weighted sum average depends on the item, but this computation only requires the k nearest neighbors). This increased the speed of predictions by a factor of 2.

Significant note on the accuracy of KNN: At the beginning we misunderstood the way we should adapt eq(7) to knn, and we were only selecting k neighbors of the given userID on the set of users that have rated the given itemID. We ended up making a prediction by considering exactly k-neighbors (for whom it is ensured that they rated the given item). This was giving us an MAE on the test set of ~ 0.73, so slightly better that what was computed in **N.1**.

7 Recommendation

7.1 Questions

R.1 *Train a k-NN predictor with training data from* `data/ml-100k/u.data`*, augmented with additional ratings from user "*944*" provided in* `personal.csv`*, using adjusted cosine similarity and* $k = 300$*. Report the prediction for user 1 item 1 (*$p_{1,1}$*).*

Answer R.1. Prediction user 1 item 1: 4.132180

R.2 *Report the top 3 recommendations for user "*944*" using the same k-NN predictor as for* ***R.1****. Include the movie identifier, the movie title, and the prediction score in the output. If additional recommendations have the same predicted value as the top 3 recommendations, prioritize the movies with the smallest identifiers in your top 3 (ex: if the top 8 recommendations all have predicted scores of* `5.0`*, choose the top 3 with the smallest*

ids.) so your results do not depend on the initial permutation of the recommendations.

Answer R.2.
Top 3 recommendations for user 944:

- Movie 1:
  - Movie id: 119
  - Movie title: Maya Lin: A Strong Clear Vision (1994)
  - Prediction score: 5.0
- Movie 2:
  - Movie id: 814
  - Movie title: Great Day in Harlem
  - Prediction score: 5.0
- Movie 3:
  - Movie id: 1189
  - Movie title: Prefontaine (1997)
  - Prediction score: 5.0

Note that there was more than 3 movies with rating 5, so we sort them by id as stated.