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

Name	Fedor Sergeev
Sciper	323636
Email	fedor.sergeev@epfl.ch

Name	Ivan Korovkin
Sciper	342338
Email	ivan.korovkin@epfl.ch

GitHub: github.com/TheodorSergeev/knn-movie-recommender

March 25, 2022

For convenience our comments are given in this color.

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.

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

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 ($\bar{r}_{\bullet,1}$), 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.*

$\bar{r}_{\bullet,\bullet}$	3.5264
$\bar{r}_{1,\bullet}$	3.6330
$\bar{r}_{\bullet,1}$	3.88827
$p_{1,1}$	4.0468

Table 1: Predicted ratings made by the global average, user average for user 1, item average for item 1, and baseline for user 1 item 1 on `ml-100k/u2.base`.

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).*

$\bar{r}_{\bullet,\bullet}$	0.9489
$\bar{r}_{1,\bullet}$	0.8383
$\bar{r}_{\bullet,1}$	0.8207
$p_{1,1}$	0.7604

Table 2: Mean absolute error (MAE) on dataset `ml-100k/u2.test` for various prediction methods.

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.*

Predictor	Avg, ms	Std, ms
$\bar{r}_{\bullet,\bullet}$	4.28	0.28
$\bar{r}_{1,\bullet}$	132.9	1.5
$\bar{r}_{\bullet,1}$	394	41
$p_{1,1}$	$83.4 \cdot 10^3$	$2.4 \cdot 10^3$

Table 3: Average and standard deviation over 3 runs of execution time of MAE calculation for the `ml-100k/u2.test` dataset for various predictors. Made on a local machine (see technical specifications in Table 4).

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

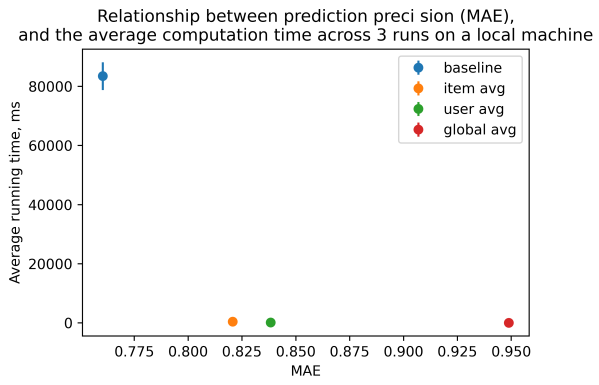
CPU	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
OS Name	Microsoft Windows 10 Home Single Language
OS Version	10.0.19044 N/A Build 19044
RAM	8107 MB
Java	Version 8 Update 321 (build 1.8.0-321-b07)
SBT	1.4.7
Scala	2.11.12

Table 4: System specification for the machine used for computations of B.1-3.

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.*

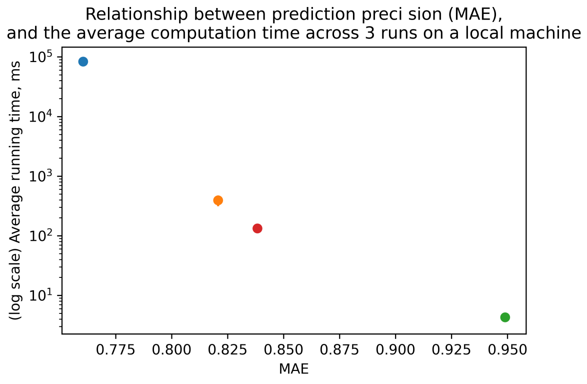
For all four methods, perform three measurements and compute the average and standard-deviation.

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?



In our case we see that the relationship between the MAE calculation and the average running time is exponential. This is likely due to the fact that our implementation is not fully optimized.

The main issue probably lies in our the `predictions/itemAvgDev` method. Initially it was written to calculate the weighted average deviation for one particular item on a given dataset. Thus it contains a filtering operation and a map that extracts all users that rated this particular movie. We



then use `itemAvgDev` to evaluate the average deviation for each item in the dataset. This obviously leads to re-filtering for each particular item, making the baseline evaluation exponentially longer, then the user-average and item-average predictors (since we have a map over all elements inside a map over all elements).

A better implementation would follow `predictions/itemUsrAvg`, which in turn was inspired by the way we work with Spark RDDs (avoid nested filterings/maps, instead replacing them by a sequence of mapping operations). We comment further on this in section D2.

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).*

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*

$\bar{r}_{\bullet,\bullet}$	3.5264
$\bar{r}_{1,\bullet}$	3.6330
$\bar{r}_{\bullet,1}$	3.8883
$p_{1,1}$	4.0468
MAE	0.7604

Table 5: Predictions made by various predictors (in distributed implementation) trained on cluster on `data/ml-100k/u2.base`, and MAE of the distributed baseline predictor on `data/ml-100k/u2.base`.

*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).*

Machine	Execution time, s (Std, s)		
	Baseline	Spark 1	Spark 4
Local (see table 4)	>30 mins	77.1 (7.2)	33.57 (2.95)
iccluster08 (see table 7)	>30 mins	96.4 (0.8)	25.38 (0.07)

Table 6: Comparison of average execution time and standard deviation for computation of MAE for predictor $p_{1,1}$ on the `data/ml-25m/r2.test` dataset.

We see that on the cluster increasing the number of workers 4 times indeed speeds up the execution time 4 times. On a local machine (a laptop) we only see speed up of about 2 times. This is probably due to the fact that Spark in this case is limited by the hardware it has, and cannot create two fully functional workers.

The baseline implementation takes an extremely long time to compute. This is not a surprise, as we mentioned before, our implementation can be further optimized. Currently it is exponentially slower than the user average and item average predictors, which on a large dataset means that this predictor is too slow to be practically useful. We discuss the way to improve the implementation below.

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.

Our implementation runs in 25 seconds matching the intended solution.

Try optimizing your local Scala implementation by avoiding temporary objects, instead preferring the use of mutable collations and data structures.

CPU	Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz
OS Name	Ubuntu 20.04.3 LTS
OS Kernel	Linux 5.4.0-96-generic
RAM	264035 MB
Java	Version 8 1.8.0-312
SBT	1.6.1
Scala	2.12.15

Table 7: System specification for iccluster028.iccluster.epfl.ch used for computations of D.1-2.

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.

The main differences of our implementation in Spark is that Spark does not allow to nested RDDs or performing Spark actions inside of transformations (see SPARK-5603). This made us rethink out implementation, and come up with a version that uses a sequence of map operations rather then nested map operations.

We used the new approach when re-implementing some of the methods for calculation of initial predictors (see for example `predictions/userAvgMap` and `predictions/itemAvgMap`). This simplified and sped up execution of all predictor methods.

However, we did not have the time to deal with the main issue in calculating the baseline prediction — the `predictions/itemAvgDev` method. We discussed the issue with this method above. Rewriting it in a fashion similar to, for example `userAvgMap`, would probably speed up the calculations significantly.

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.*

$p_{1,1}$	4.0468
MAE	0.7604

Table 8: Personalized prediction with uniform similarities made for user 1 item 1 (trained on dataset `ml-100k/u2.base`). Mean absolute error on dataset `ml-100k/u2.test`.

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.*

$s_{1,2}$	0.0730
$p_{1,1}$	4.0870
MAE	0.7372

Table 9: Adjusted cosine similarity for users 1 and 2, and personalized prediction with adjusted cosine similarities made for user 1 item 1 (trained on dataset `ml-100k/u2.base`). Mean absolute error on dataset `ml-100k/u2.test`.

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?*

The Jaccard Coefficient for two sets A and B is given by

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| + |A \cap B|}.$$

In our case the sets will represent the sets of items rated by users. So for the training set T, a set of items rated by user u will be

$$A_u = \{i \ : \ \exists r_{u,i} \in T\},$$

and thus the Jaccard Similarity Coefficient for users u and v is

$$J(u,v) := J(A_u, A_v).$$

The predictor based on Jaccard Coefficient performs worse than the one using the Adjusted Cosine Similarity. At the same time the computation of the Jaccard Coefficent is simpler to implement and performs faster in our case.

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

$s_{1,2}$	0.03187
$p_{1,1}$	4.0982
MAE	0.7556

Table 10: Jaccard similarity for users 1 and 2, and personalized prediction with Jaccard similarities made for user 1 item 1 (trained on dataset `ml-100k/u2.base`). Mean absolute error on dataset `ml-100k/u2.test`.

The intuition behind Adjusted Cosine Similarity performing better then Jaccard Coefficient is as follows. Jaccard Coefficient draws similarity between users based on the similarity of movies that they rated. Thus it does not take into account the ratings of these movies that the users gave. In this case, two users who have watched the same movies will receive a high similarity score, although one could have given everything a rating of 1, when the other gave every movie a 5. On the contrary, the Adjusted Cosine Similarity takes into account not only on which movies the users watched, but also on the rating the users gave.

Jaccard Coefficient is a better choice of a similarity metric for the recommender if the aim is to provide a movie that is more likely to be "clicked" rather than provide a movie that is more likely to be highly rated by the user.

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`.*

user 1 and itself	0.0
user 1 and 864	0.242
user 1 and 886	0.0
$p_{1,1}$	4.319

Table 11: Similarities and prediction for user 1, item 1 for knn method with k=10.

N.2 *Report the MAE on `data/ml-100k/u2.test` for* $k = 10, 30, 50, 100, 200, 300, 400, 800, 943$. *What is the lowest* k *such that the MAE is lower than for the baseline* *(non-personalized) method?* The lowest k such that MAE is lower than the baseline is k = 100.

K	MAE
10	0.82872
30	0.78101
50	0.76181
100	0.74669
200	0.74005
300	0.73915
400	0.73912
800	0.73923
943	0.83663

Table 12: Mae values for different k.

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.*

Most of the efforts were spent to speed up similarity calculations. First, we calculated full list of all pairs which took us twice longer. Second, we implemented a pairing algorithm to avoid such long computations of unneeded variables. Many functions were taken out of similarity calculations. For an example, implementation with preprocArray function inside similarity computation took almost one hour for only one user.

The problem of long computations in this point is the calculating of all similarities between users for knn processing, which takes about 9 minutes. After that MAE computation takes only 30 secs. We expect much better results with probable implementation of cosine similarity via Spark in the next milestone.

Method	Time, s
mean	526.8
std	14.6

Table 13: Mae values for different k.

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 (*$p_{1,1}$*).*

$p_{1,1}$	4.132

Table 14: Prediction for user 1, item 1 on augmented dataset with k=10.

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.*

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

Table 15: Top 3 recommendations for user 944 with lowest id's first.