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

Motivation and Outline: Anne-Marie Kermarrec
Detailed Design, Writing, Tests: Erick Lavoie
Teaching Assistant: Mu Zhou
Last Updated: 2022/03/16 16:04:25 +01'00'

Due Date: 25-03-2022 23:59 CET
Submission URL: `https://cs449-submissions.epfl.ch:8083/m1`
General Questions on Moodle
Personal Questions: mu.zhou@epfl.ch

Abstract

In this project, you will progressively build a recommender system for Movies. In this Milestone, you will start by implementing a simple baseline prediction for recommendation then distribute it with Spark. You will then compare the quality of its predictions to a second *personalized* approach based on similarities and k-NN. You will measure the CPU time to develop insights in the system costs of the prediction methods.

1 Motivation: Movie Recommender

You maintain a movie recommendation platform: your goal is to automatically recommend new movies for your users that they are most likely to appreciate. While there is a range of increasingly sophisticated and accurate techniques to build recommender systems, you will first start by building a simple system based on global averages, that still take into account some personal bias in how users rate movies. This is fast and inexpensive and will therefore serve as a good baseline to compare with the more sophisticated techniques of the next milestones.

Prediction methods based on global averages are straight-forward to implement with the Resilient Distributed Dataset (RDD)[1] in combination with broadcast variables[2], so this Milestone will be a great way to assess the skills you have developed in the exercise sessions.

[1] `https://spark.apache.org/docs/latest/rdd-programming-guide.html#resilient-distributed-datasets-rdds`

[2] `https://spark.apache.org/docs/latest/rdd-programming-guide.html#broadcast-variables`

1.1 Dataset: MovieLens 100K

For this milestone, you will use the MovieLens 100K dataset [1], as a representative example of what you might collect on your recommender platform. You can download the dataset from this url: `https://grouplens.org/datasets/movielens/100k/`.

The 100K MovieLens dataset comprises 100,000 ratings from 943 users on 1,682 movies. Each user has rated at least 20 different movies, and each movie has been rated by at least one user.This dataset is small compared to the amount of RAM available in common laptops and desktops, or the IC Cluster (as of January 2021): all the exercises for this milestone should complete in a few seconds or minutes at most.

The ratings are tuples (u, i, r, t) of an (anonymized) user id $u \in \mathbb{N}$ (positive integers, starting at 1), a movie id $i \in \mathbb{N}$ (also a positive integer starting at 1)[3], a rating $r \in \{1, 2, 3, 4, 5\}$, and timestamp t (which we won't use). They are saved on the file system as tab-separated values, one line per tuple, in the file `'ml-100k/u.data'`. For example, the first line of the file records that user *196* has rated movie *242* with a rating of *3* at timestamp *881250949*:

```
196	242	3	881250949
```

All numbers are represented with ASCII characters, therefore user id 196 really represents that number (it is not a binary representation of the number). All users have made at least 20 ratings, but some movies may be rated by only one user. The dataset in that sense is "compact": all user ids and movie ids are used in at least one rating[4].

For convenience and replicability in testing different solutions, the same ratings are split randomly 80% training and 20% testing in five different folds for cross-validation[5] such that the test sets are mutually exclusives. The train and test sets are numbered according to their fold, respectively in the files `'ml-100k/uX.base'` and `'ml-100k/uX.test'` where X is 1 to 5. It may happen that some movies are not rated in one of the the training or test sets, because they only had one rating. You therefore need to be careful with any operation involving division by item-based sums or averages, as they may introduce divisions by zero.

If you were developing new algorithms, you should ensure your results are not specific to the particular test set you have chosen by doing cross-validation on all folds. However, for the sake of simplicity, you will only test on the `ml-100k/u2.test` dataset (with the corresponding `ml-100k/u2.base`).

[3]We will use the more generic *item* rather than *movie* through the rest of the document to follow the literature conventions on collaborative filtering.

[4]This won't necessarily be the case in the next milestones.

[5]`https://en.wikipedia.org/wiki/Cross-validation_(statistics)`

2 Proxy Problem: Predicting Ratings

Recommendations presented to a given user will be the top n predictions of ratings of unseen items, with typically $3 \leq n \leq 20$. For movie recommendations, it is not so important that the actual three *best* ratings are suggested but more that those actually recommended are at least above average and as good as possible. The closer the predictions are to the actual ratings on the test set, the more likely the top n predictions will be for highly rated items.

You will therefore evaluate the quality of different solutions according to their predictive capabilities on test sets. To ensure the solutions *generalize* to unseen items, the predictions will be made only using ratings from the train set (`u2.base`). The ratings of the test set (`u2.test`) will only be used the measure the quality of the predictions.

Multiple metrics are possible to measure the accuracy of predictions [3]. You will use the simple *Mean Absolute Error (MAE)*, i.e. average of the absolute error between the actual rating for user u of item i ($r_{u,i}$) and the predicted rating for the same user-item pair ($p_{u,i}$) for all ratings of the *Test* set[6]:

$$\textit{Mean-Absolute-Error (MAE)} = \frac{1}{|\,\textit{Test}|} \sum_{r_{u,i} \in \textit{Test}} |p_{u,i} - r_{u,i}| \tag{1}$$

3 Baseline: Prediction based on Global Average Deviation

The following baseline incorporates some user bias, in the form of a user average rating, in predictions, and then averages normalized deviations from each user's average. To understand why, observe the following two things.

First, some users tend to rate more positively than others and therefore have different average ratings over all items, which we note $\bar{r}_{u,\bullet}$. You will therefore pre-process ratings to instead express how much they *deviate* from a user's average rating ($r_{u,i} - \bar{r}_{u,\bullet}$).

Second, the average rating for a user does not necessarily sit in the middle of the rating scale $\{1, 2, 3, 4, 5\}$ and therefore maximum deviations may be *asymmetric* in the positive and negative directions. Moreover, the range of deviations, in the positive and negative directions, may differ for different users. The average of many deviations from different users may therefore result in a larger deviation than the range of some users, leading to an incorrect range, i.e. < 1 or > 5, when making predictions. We therefore normalize the deviations such that for all users, their deviations will be in the range $[-1, 1]$ with -1 corresponding to a rating of 1 (maximum negative deviation), 1 corresponding to a rating of 5 (maximum positive deviation), and 0 corresponding to the average rating for any user.

[6]The notation conventions used throughout the document are summarized in Appendix A.

The normalized deviation ($\hat{r}_{u,i}$) is therefore the following:

$$\hat{r}_{u,i} = \frac{r_{u,i} - \bar{r}_{u,\bullet}}{scale(r_{u,i}, \bar{r}_{u,\bullet})} \tag{2}$$

with a scale specific to a user's average rating:

$$scale(x, \bar{r}_{u,\bullet}) = \begin{cases} 5 - \bar{r}_{u,\bullet} & \text{if } x > \bar{r}_{u,\bullet} \\ \bar{r}_{u,\bullet} - 1 & \text{if } x < \bar{r}_{u,\bullet} \\ 1 & \text{if } x = \bar{r}_{u,\bullet} \end{cases} \tag{3}$$

The global average deviation for an item i ($\bar{\hat{r}}_{\bullet,i}$) is the average of the deviations for all users on this item (where $U(i)$ is the set of users with a rating for item i):

$$\bar{\hat{r}}_{\bullet,i} = \frac{\sum_{u \in U(i)} \hat{r}_{u,i}}{|U(i)|} \tag{4}$$

The prediction of rating for a user u on item i, which converts back the global average deviation to a numerical rating in the range $[1, 5]$, is then:

$$p_{u,i} = \bar{r}_{u,\bullet} + \bar{\hat{r}}_{\bullet,i} * scale((\bar{r}_{u,\bullet} + \bar{\hat{r}}_{\bullet,i}), \bar{r}_{u,\bullet}) \tag{5}$$

Note that if $\bar{\hat{r}}_{\bullet,i} = 0$, or there is no rating for i in the training set, then $p_{u,i} = \bar{r}_{u,\bullet}$. If u has no rating in the training set, simply use the global average, i.e. $p_{u,i} = \bar{r}_{\bullet,\bullet}$.

In the following questions, you will compare the prediction accuracy of this baseline with simpler methods based on averaging ($\bar{r}_{\bullet,\bullet}$, $\bar{r}_{u,\bullet}$, $\bar{r}_{\bullet,i}$). This will give you some intuitions about their relative efficacy[7]. You will also measure the computation time all four require. This will provide you with some intuitions about the computing cost increase that comes with better accuracy, between the four prediction methods of this section as well as those you will implement later.

3.1 Questions

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

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 5) 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 defined, otherwise the global average.*

[7]In real-life you should always ensure that simpler methods are not sufficient for the problem at hand before trying more complex methods, as they are generally much less expensive and simpler to configure.

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. 5).*

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 template provides `timingInMs` for convenience). 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?

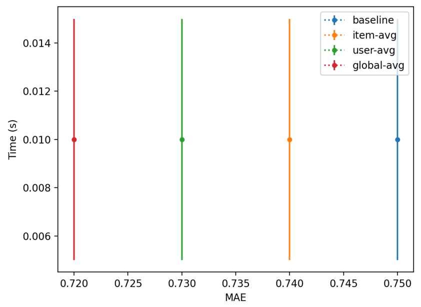

Figure 1: Example of Time vs MAE for different prediction methods. Actual position and standard deviation in final figure should be different.

3.2 Tips

Debugging numerical programs, such as those of this milestone, can be tricky because a program will not fail but will instead simply provide incorrect output. To ensure you have not made mistakes in some steps, try the followings:

- Print a few values, as well as statistics (min, max, standard-deviation, average, distribution of values in a certain range) of results and intermediary computations. Especially make sure you do not obtain *not-a-number* (nan) values.
- Check that the predictions are in the range $[1, 5]$. If not there is a problem in your code.
- The MAE for Eq. 5 should be below 0.80.

Also:

- Scala's standard library API is not backward compatible between minor versions (ex: `2.11` may not be compatible with `2.12`). Make sure to use the documentation for the version of the project, as listed in `build.sbt`.

4 Spark Distribution Overhead

Spark's Resilient Distributed Datasets (RDDs) offer an API similar to Scala's standard collections but enable distributed operations on multiple worker nodes in parallel with minimal code changes.

However, the distributed operations are subject to a base overhead that is higher than using Scala's standard library. Without proper care in first ensuring the performance benefits of distributed operations compensate for the overhead of the library, you may *slow down* your application by using Spark.

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.base` 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 5). Compute the prediction accuracy (average MAE on `ml-100k/u2.test`) of the proposed baseline ($p_{u,i}$, Eq. 5).*

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`, and compute the MAE. 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. Try obtaining results roughly in the same ballpark or faster, teams with the fastest implementations will obtain more points. Don't worry if your code is slower during some executions because the cluster is busy, we will re-run your code for grading in a less busy environment.

Try optimizing your local baseline Scala implementation by avoiding temporary objects, instead preferring the use of mutable collections 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.

4.2 Tips

- Sometimes the Java Garbage Collector, gets overwhelmed after multiple runs within the same sbt session. You may notice warnings such as:

```
[warn] In the last 10 seconds, 5.55 (57.4\%) were spent in GC.
[Heap: 3.19GB free of 4.92GB, max 7.11GB] Consider increasing
the JVM heap using ‘-Xmx‘ or try a different collector,
e.g. ‘-XX:+UseG1GC‘, for better performance.
```

  If this happens, try increasing memory by invoking sbt with 8GB of Memory, prior to running the experiments:

```
export JAVA_OPTS="-Xmx8G"; sbt
```

  You may also need to stop and restart sbt from time to time to force the memory to be cleaned up. If that still does not help, try optimizing your code to use less memory.

5 *Personalized* Predictions

While some movies are highly rated by most users, most movies might appeal only to subsets of users [8]. In effect, to best answer the tastes of a maximum of users, you would like to provide *personalized* recommendations beyond the usual blockbusters. The following approach is based on *collaborative filtering* [3], i.e. automatically identifying *similarities* in ratings between users to recommend highly rated movies from those users that are most similar. When tuned correctly, similarity approaches give higher prediction accuracy at the cost of higher computational complexity.

5.1 Similarity-based Predictions

The global average deviation (Eq. 4) gives an equal weight of $\frac{1}{n}$ to all n users that provided ratings for item i. The core insight behind similarity-based techniques is that not all users are equally useful for predictions. Giving a different weight to different users, with higher weights to *similar* users, can therefore improve prediction accuracy.

There are many similarity metrics to choose from [2, 3] to determine how similar two users are. The adjusted cosine similarity [4] works relatively well and is simple, you will therefore use it for the rest of the project (in which $I(u)$ are the items rated by user u):

$$s_{u,v} = \begin{cases} \frac{\sum_{i \in (I(u) \cap I(v))} \hat{r}_{u,i} * \hat{r}_{v,i}}{\sqrt{\sum_{i \in I(u)} (\hat{r}_{u,i})^2} * \sqrt{\sum_{i \in I(v)} (\hat{r}_{v,i})^2}} & (I(u) \cup I(v)) \neq \emptyset; \exists_{i \in I(u)} \hat{r}_{u,i} \neq 0; \exists_{i \in I(v)} \hat{r}_{v,i} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

The set intersection on the numerator selects only items that are in common. The summation in each term on the denominator normalizes the sum on the numerator. The root of a squared sum gives a higher weight to large deviations than a regular average. This similarity function ranges between $[-1, 1]$, with -1 if two users rate at the maximum of the rating scale in opposite ways on all the same items, and 1 if two users rate in exactly the same direction at the maximum of the rating scale on the same items (e.g. if the two users are actually the same). Most of the similarity values will be between those two extremes.

You can now compute the user-specific weighted-sum deviation for an item i ($\bar{\hat{r}}_{\bullet,i}(u)$), which is similar to Eq. 4 from Milestone 1 but gives a different weight to ratings of other users based on their similarity:

$$\bar{\hat{r}}_{\bullet,i}(u) = \begin{cases} \frac{\sum_{v \in U(i)} s_{u,v} * \hat{r}_{v,i}}{\sum_{v \in U(i)} |s_{u,v}|} & \exists_{v \in U(i)} s_{u,v} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

[8]This is an instance of the Long Tail distribution `https://en.wikipedia.org/wiki/Long_tail`

The prediction equation is similar to Eq. 5 but incorporates the user-specific $\bar{\hat{r}}_{\bullet,i}(u)$ of Eq. 7 instead of $\bar{\hat{r}}_{\bullet,i}$ of Eq. 5.

$$p_{u,i} = \bar{r}_{u,\bullet} + \bar{\hat{r}}_{\bullet,i}(u) * scale((\bar{r}_{u,\bullet} + \bar{\hat{r}}_{\bullet,i}(u)), \bar{r}_{u,\bullet}) \tag{8}$$

Note that if $\bar{\hat{r}}_{\bullet,i}(u) = 0$, or there is no rating for i in the training set, then $p_{u,i} = \bar{r}_{u,\bullet}$. If u has no rating in the training set, and therefore $\bar{r}_{u,\bullet}$ is undefined, simply use the global average, i.e. $p_{u,i} = \bar{r}_{\bullet,\bullet}$.

5.2 Preprocessing Ratings

Notice that each term of the denominator of Eq. 6, i.e. $\sqrt{\sum_{j \in I(u)} (\hat{r}_{u,j})^2}$, is independent of the deviation $\hat{r}_{u,i}$ for all $j \in I(u)$, the items rated by user u. By virtue of the law of multiplication for fractions ($\frac{a*c}{b*d} = \frac{a}{b} * \frac{c}{d}$), each term of the denominator can be applied independently, before the multiplication, to each $\hat{r}_{u,i}$:

$$\breve{r}_{u,i} = \begin{cases} \frac{\hat{r}_{u,i}}{\sqrt{\sum_{j \in I(u)} (\hat{r}_{u,j})^2}} & \text{if } i \in I(u) \text{ and } \exists_{j \in I(u)} \hat{r}_{u,j} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

You can use that property to preprocess ratings such that only the numerator multiplication of Eq. 6 remains to be computed:

$$s_{u,v} = \sum_{i \in (I(u) \cap I(v))} \breve{r}_{u,i} * \breve{r}_{v,i} \tag{10}$$

This should make your implementation faster.

5.3 Questions

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.2 *Using the adjusted cosine similarity (Eq. 6), 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. 8) and the prediction accuracy (MAE on `ml-100k/u2.test`) of the personalized baseline predictor.*

P.3 *Implement the Jaccard Coefficient[9]. Provide the mathematical formulation of your similarity metric in your report. Using 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. 8) 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?*

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

5.4 Tips

- The denominator of Eq. 6 and Eq. 9 should really be a sum over items, and not users. It can be easy to confuse the two with some data structures.
- The user-specific weighted-sum deviation (Eq. 7) should be computed for all users u and all items i (for all ratings to be predicted). Moreover, v is independent from u but u may also be included in $U(i)$ in some cases.

6 Neighbourhood-Based Predictions

The similarity method of the previous section lowers the weight of some ratings such that they become much less significant for the final prediction. The insight of neighbourhood method is that the least significants can actually be ignored, with limited negative, and sometimes positive, impact on predictions. Formally, that means we nullify the similarity values for a majority of pairs of users ($s_{u,v} = 0$) and therefore the deviations (ratings) multiplied by a null similarity (in Eq. 7) are not considered.

Keeping only the k nearest neighbours (excluding self-similarity of a user with themselves), according to a similarity metric (ex: Eq. 6), gives us the *k-NN* algorithm which has the added benefit of lowering memory usage, data transfers, and prediction time. The actual impact of the neighbourhood size k on the prediction accuracy is dependent on the dataset, similarity metric, and prediction methods. This needs to be investigated empirically for every specific problem, which you will do in the following questions.

6.1 Questions

N.1 *Implement the k-NN predictor using the adjusted cosine similarity. Do not include self-similarity in the k-nearest neighbours. Using $k = 10$ and `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`.*

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?*

N.3 *Measure the time required for computing predictions (without using Spark) and the MAE 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.*

7 Recommendation

To provide recommendations of new movies to user u, you can now simply compute predictions for which there are no ratings in the dataset[10] and keep the n best:

$$R(u,n) = top(n, [p_{u,i} | r_{u,i} \notin Ratings]) \tag{11}$$

To recommend movies for a new user, use id 944 for that user (the highest user id in the dataset is 943, so that is one higher than that), and add at least 20 ratings to the dataset. As the baseline predictions are based on ratings that deviate from the rating average, make sure there are a good number of ratings across the entire range $[1, 5]$.

For convenience and grading, the template already provides ratings in `data/personal.csv` but you can still change them for your own preferences to see what recommendations you would personally get!

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

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.*

7.2 Tips

- Check that the prediction scores for the recommended items are indeed close to 5.

[10]We are assessing the quality of recommendations *subjectively* rather than *objectively* ("ground truth" is our personal judgement), therefore we are using the entire dataset for predictions. If we were assessing the quality of recommendations using an objective metric, we would again use disjoint train and test sets.

8 Deliverables

You can start from the latest version of the template:

- Zip Archive: `https://gitlab.epfl.ch/sacs/cs-449-sds-public/project/cs449-Template-M1/-/archive/master/cs449-Template-M1-master.zip`
- Git Repository:

```
git clone

https://gitlab.epfl.ch/sacs/cs-449-sds-public/project/cs449-Template-M1.
git
```

We may update the template to clarify or simplify some aspects based on student feedback during the semester, so please refer back to `https://gitlab.epfl.ch/sacs/cs-449-sds-public/project/cs449-Template-M1` to see the latest changes.

Provide answers to the previous questions in a **pdf** report (if your document saves in any other format, export/print to a pdf for the submission). Also, provide your source code (in Scala) in a single archive:

```
report.pdf
src/*
```

Your archive should be around or less than 1MB. Submit to the TA using the submission URL of the first page of this Milestone.

9 Grading

You will be graded both on the question answers, and source code quality as well as organization. Points for source code will reflect how easy it was for the TA to run your code and check your answers. Grading for answers to the questions without accompanying executable code will be 0.

9.1 Collaboration vs Plagiarism

You are encouraged to help each other better understand the material of the course and the project by asking questions and sharing answers. You are also very much encouraged to help each other learn the Scala syntax, semantics, standard library, and idioms and Spark's Resilient Distributed Data types and APIs. It is also fine if you compare numerical answers to the questions before submitting your report and code for grading. The dynamics of peer learning can enable the entire class to go much further than each person could have gone individually, so it is very welcome.

However, each team should write their own original report and code. We will give 0 to all submissions that are copies of one another from this year, and also 0 to copies of previous years' submissions. You are also incentivized to keep the fastest implementation techniques private to your team, as the fastest implementations will receive higher grades. This is done to reward the teams that will spend more time exploring different implementation techniques.

10 Updates

Since the original release of the Milestone description on February 25th, we have made the following changes:

- Fixed inconsistency between **N.2** and template repo so both use 943 as the last k value to test.

References

[1] Harper, F. M., and Konstan, J. A. The MovieLens datasets: History and context. ACM Transactions on Interactive Intelligent Systems 5, 4 (Dec. 2015), 19:1–19:19.

[2] Herlocker, J., Konstan, J. A., and Riedl, J. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. Information retrieval 5, 4 (2002), 287–310.

[3] Karydi, E., and Margaritis, K. Parallel and distributed collaborative filtering: A survey. ACM Comput. Surv. 49, 2 (Aug. 2016).

[4] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th international conference on World Wide Web (2001), pp. 285–295.

A Notation

- u and v: *users* identifiers
- i and j: *items* identifiers
- $\bar{r}_{\bullet,\bullet}$: global average (over all users and items) ($\frac{1}{|\text{Ratings}|} \sum_{r_{u,i} \in \text{Ratings}} r_{u,i}$)
- $\bar{r}_{u,\bullet}$: average rating for user u, over all items ($\frac{1}{|I(u)|} \sum_{i \in I(u)} r_{u,i}$)
- $\bar{r}_{\bullet,i}$: average rating for item i, over all users ($\frac{1}{|U(i)|} \sum_{u \in U(i)} r_{u,i}$)
- $\hat{r}_{u,i}$: deviation from the average $\bar{r}_{u,\bullet}$
- $r_{u,i}$: rating of user u on item i, (u is always written before i)
- $p_{u,i}$: predicted rating of user u on item i

- $|X|$: number of items in set X
- $*$: scalar multiplication
- $r_{u,i}, r_{v,i} \in \text{Train}$: both $r_{u,i}$ and $r_{v,i}$ are elements of $Train$ for the same i
- 1_x: indicator function, $\begin{cases} 1 & \text{if } x \text{ is true} \\ 0 & \text{otherwise} \end{cases}$
- $u, v \in U$: shorthand for $\forall u \in U, \forall v \in U$
- $R(u, n)$: top n recommendations for user u as a list
- $\mathit{sorted}_{\searrow}(x)$: sort the list x in decreasing order
- $[x|y]$: create a list with elements x such that y is true for each of them
- $\mathit{top}(n, l)$: return the highest n elements of list l
- U: set of users
- I: set of items
- $U(i)$: is the set of users with a rating for item i ($\{u | r_{u,i} \in \text{Train}\}$)
- $I(u)$: is the set of items for which user u has a rating ($\{i | r_{u,i} \in \text{Train}\}$)

B Pedagogical Notes

In this appendix, we explain why we structured the Milestones as they are. This is not necessary to complete the Milestone successfully but might be of interest for future revisions and curious students. Among other goals, the entire project aims to:

- Develop the skills required to implement and optimize variations of a data science algorithms locally and in a distributed manner;
- Understand how k-NN fits in a recommender application, and how it relates in performance and accuracy to other similar methods;
- Understand the methodology to build reliable distributed applications in a step-wise fashion;
- Compare the benefits and costs of distributed and non-distributed approaches to find where each is most applicable;
- Develop the skills to choose the most affordable infrastructure for different operation contexts.

Moreover, for the previous sections, here are specific goals:

B.1 Baseline

Progression	: Stepping stone towards personalized methods
Skills	: Get familiar with Scala and Movielens using simple algorithms
Methodology	: Establish a baseline that more complex algorithms should improve upon in accuracy
Knowledge	: Relationship between accuracy and computing cost
Evaluate	: Scala programming skills

B.2 Spark Distribution Overhead

Progression	: Get familiar with Spark with simple algorithms
Skills	: Use simpler (non-distributed) implementations to verify correctness
Methodology	: Compare the performance gains of distribution against an optimized non-distributed version
Knowledge	: When to distribute implementations, when to optimize
Evaluate	: Spark programming abilities

B.3 Personalized Predictions

Progression : Intermediate stage between non-personalized and k-NN

Evaluate : Translation of similarity metric from mathematical definition to code, and comparison against baseline

B.4 Neighbourhood-based Predictions

Progression : Stepping stone towards distributed k-NN implementations

Skills : Implement k-NN on a single machine

Knowledge : Impact of the choice of k on accuracy

Evaluate : Programming skills for fast code

B.5 Recommender

Progression : Understand how k-NN can be used in an actual application, ties all previous skills in the concrete use case of the project