CS-449 Project Milestone 1: Baseline Recommendation with Global Averaging

Motivation and Outline: Anne-Marie Kermarrec
Detailed Design, Writing, Tests: Erick Lavoie
Teaching Assistant: Athanasios Xygkis
Last Updated: 2021/02/23 14:49:17 +01'00'

Due Date: 19-03-2021 23:59 CET
Submission URL: `https://cs449-sds-2021-sub.epfl.ch:8083/m1`
General Questions on Moodle
Personal Questions: athanasios.xygkis@epfl.ch

Abstract

In this project, you will progressively build a recommender system for Movies, that will leverage the distributed computing capabilities of Spark. In this Milestone, you will start by implementing a simple baseline prediction model for recommendation against which you will compare those of the next milestones. You will measure its CPU time to develop insights in the system costs of various 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/u1.test` dataset (with the corresponding `ml-100k/u1.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 $5 \leq n \leq 20$. For movie recommendations, it is not so important that the actual five *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 (`u1.base`). The ratings of the test set (`u1.test`) will only be used the measure the quality of the prediction.

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:

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

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

1. (2 points) Compute and report the global average rating ($\bar{r}_{\bullet,\bullet}$). Do ratings, on average, coincide with the middle of the rating scale (3 from the scale $\{1, 2, 3, 4, 5\}$)? If not, are they higher or lower on average? By how much?

2. (3 points) Compute the average rating for each user ($\bar{r}_{u,\bullet}$). Do *all* users rate, on average, close to the global average? Check min and max for user average and assume a difference less than 0.5 is small. Do *most* users rate, on average, close to the global average? Calculate and report the ratio of users with average ratings that deviate with less than 0.5 from 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.

3. (3 points) Compute the average rating for each item ($\bar{r}_{\bullet,i}$). Are *all* items rated, on average, close to the global average? Check min and max for item average and assume a difference less than 0.5 is small. Are *most* items rated, on average, close to the global average? Calculate and report the ratio of items with average ratings that deviate with less than 0.5 from the global average.

4. (8 points) Compare the prediction accuracy (average MAE on `ml-100k/u1.test`) of the previous methods ($\bar{r}_{\bullet,\bullet}$, $\bar{r}_{u,\bullet}$, $\bar{r}_{\bullet,i}$) to the proposed baseline ($p_{u,i}$, Eq. 5). Report the results you obtained in a table. Discuss the difference(s) you observed and why you think they occur.

5. (7 points) Measure the time required for computing predictions 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. For all four methods, perform ten measurements and report in a table the min, max, average, and standard-deviation. Report also the technical specifications (model, CPU speed, RAM, OS, language version) of the machine on which you ran the tests. Which of the four prediction methods is the most expensive to compute? How much more compared to using the global average rating ($\bar{r}_{\bullet,\bullet}$)? Calculate and report the ratio between the average time for computing the baseline (Eq. 5) and the average time for computing the global average.

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: `2.12` may not be compatible with `2.13`). Make sure to use the documentation for the version of the project, as listed in `build.sbt`.

4 Recommendation

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

$$R(u,n) = top(n, [p_{u,i} | r_{u,i} \notin Train]) \tag{6}$$

To recommend movies for yourself, use id 944 (the highest user id in the dataset is 943, so that is one higher than that), and add at least 20 ratings to the training set. As the similarity and predictions are based on ratings that deviate from your rating average, make sure to provide a good number of ratings across the entire range $[1, 5]$. To make rating more convenient, we have reformatted the list of movie identifiers and titles in a single CSV file, so that you can provide your personal ratings in the third column using a spreadsheet program, such as OpenOffice Calc, Microsoft Excel, or Apple's Number. You can then process that CSV file to add your ratings to the training set.

Notice that the predictions do not take into account the popularity of a movie, i.e. total number of ratings, into account. The recommender may therefore suggests obscure movies seen by few users if they all rated the movie highly. In the following questions, you will provide your own personal recommendations and (optionally) suggest one or multiple simple modifications to take into account the popularity of a movie.

4.1 Questions

1. (2 points) Report your personal top 5 recommendations using the baseline predictor (Eq. 5), including the movie identifier, the movie title, and the prediction score. If additional recommendations have the same predicted value as the top 5 recommendations, prioritize the movies with the smallest identifiers in your top 5 (ex: if the top 8 recommendations all have predicted scores of `5.0`, choose the top 5 with the smallest ids.) so your results do not depend on the sorting behaviour. Are these movies you have actually liked (but did not rate) or would like to see in the future?

2. (Bonus, 3 points) How could you modify the predictions to favour more popular movies, e.g. by smoothly decreasing the prediction score of movies with few ratings while keeping the prediction score of those with many ratings almost identical? Provide the equation(s) of your modifications, which equations of this document they are intended to replace, and the new top5 recommendations you obtain for yourself (movie identifier, movie title, prediction score).

4.2 Tips

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

5 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 report. Also, provide your source code (in Scala) and your personal movie ratings in a single archive:

```
CS449-YourID-M1/
    README.md
    report-YourID-M1.pdf
    build.sbt
    data/personal.csv
    project/build.properties
    project/plugins.sbt
    scala/src/main/scala/stats/Analyzer.scala
    scala/src/main/scala/predict/Predictor.scala
    scala/src/main/scala/recommend/Recommender.scala
```

As well as any other packages or source files you have created. Remove all other unnecessary folders (ex: `project/project`, `project/target`, and `target`). Ensure your project automatically and correctly downloads the missing dependencies and correctly compile from only the files you are providing. Ensure the `statistics.json`, `predictions.json`, and `recommendations.json` files are re-generated correctly using the commands listed in `README.md`. If in doubt, refer to the `README.md` file for more detail.

Once you have ensured the previous, remove again all unnecessary folders, as well as the dataset (`data/ml-100k` and `data/ml-100k.zip`, if present), zip your archive (`CS449-YourID-M1.zip`), and submit to the TA. Your archive should be around or less than 1MB.

6 Grading

We will use the following grading scheme:

	Points
Questions	25
Source Code Quality & Organisation	5
Bonus	3
Total	30

Any bonus point (total over 30) will be reported on future Milestones. Points for 'Source Code' will reflect how easy it was for the TA to run you code and check your answers. Grading for answers to the questions without accompanying executable code will be 0.

6.1 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 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, you should write the report and code individually. You should also compare answers *only after having attempted the best shot you can do alone*, well ahead of the deadline, and after doing your best to understand the material and hone your skills. The main reason is pedagogical: we have done our best to prepare a project that removes much of the accidental complexity of the topic, would be much more accessible than learning directly from the research literature, and would be deeper and more balanced than marketing material for the latest technologies. But for that pedagogical experience to give its fruits, you have to put enough efforts to have it grow on you.

To make grading simpler and scalable, so you will have feedback in a timely manner, we have opened the possibility to short-cutting the entire learning process and go for maximal grade with minimal effort. If you do so, you will not only completely waste a great personal opportunity to develop useful skills, you will lower the reputation of an EPFL education for all your colleagues, and you will be wasting the resources Society is collectively investing in your education. So we will be remorseless and drastically give 0 to all submissions that are copies of one another.

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
- $\bar{r}_{\bullet,\bullet}$: average over a range, with $\bullet$ representing all possible identifiers, either *users*, *items*, or both (ex: $\bar{r}_{\bullet,i}, \bar{r}_{u,\bullet}, \bar{r}_{\bullet,\bullet}$), ex: $\bar{r}_{u,\bullet} = \frac{\sum_{r_{u,i} \in \text{Train}} r_{u,i}}{\sum_{r_{u,i} \in \text{Train}} 1}$
- $\hat{r}_{u,i}$: 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
- $r_{u,i}, r_{v,i} \in$ 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
- $sorted_{\searrow}(x)$: sort the list x in decreasing order
- $[x|y]$: create a list with elements x such that y is true for each of them
- $top(n, l)$: 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 ($\{i|r_{u,i} \in \text{Train}\}$)