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: 2021/03/08 18:39:15 +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. For items that do not have ratings in the training set, use the global average ($\bar{r}_{\bullet,\bullet}$), instead of the item-specific average ($\bar{r}_{\bullet,i}$).

5. (7 points) Measure the time required for computing predictions for all ratings in the test set (`ml-100k/u1.text`) 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 dataset[8] and keep the n best:

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

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 dataset. As the baseline 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 dataset.

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).

[8] 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.

4.2 Tips

- Check that the prediction scores for the recommended items are indeed close to 5.
- Strictly speaking, when using the baseline method, the only required number for making predictions for yourself is your average rating on many movies. (Predictions should also only be made for items not already rated.) It may therefore seem superfluous to add your own ratings to the dataset since your ratings do not interact with those of others for predictions. "Adding your ratings to the dataset" prepares you for Milestone 2, which will require similarities between you and other users, based on ratings, to be computed prior to making predictions.

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
    src/main/scala/stats/Analyzer.scala
    src/main/scala/predict/Predictor.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`). Your archive should be around or less than 1MB. Submit to the TA using the submission URL of the first page of this Milestone.

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.

7 Updates

Since the original release of the Milestone description on February 23rd, we have made the following changes:

- Clarified question 3.1.4 to mention that the item-specific average, when no rating are in the training set, should be the global average. Prompted by Dhruti's question (`https://moodle.epfl.ch/mod/forum/discuss.php?d=53854#p110066`).
- Clarified notation for $\bar{r}_{\bullet,\bullet}$ in Appendix A. Prompted by Mustapha's question (`https://moodle.epfl.ch/mod/forum/discuss.php?d=54597#p111305`).
- Clarified measurements should be made for computing predictions for all items in the test set in question 3.5. Prompted by Raphael's question (`https://moodle.epfl.ch/mod/forum/discuss.php?d=54569#p111259`).
- Clarified Recommendation (Section 4): removed references to Train sets since we are using the entire dataset for making predictions; clarified that we can do this because we are using a subjective assessment; and clarified why you are asked to add your ratings to the dataset. Prompted by a private email from Alexander Olsson.
- Removed the `scala/` prefix for the source code in the deliverable. Prompted by a question from Karim (`https://moodle.epfl.ch/mod/forum/discuss.php?d=55151#p112334`).

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
- $r_{u,i}, r_{v,i} \in$ 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 ($\{i|r_{u,i} \in \text{Train}\}$)

CS-449 Project Milestone 2: Neighbourhood-based Personalized Recommendations with k-NN

Motivation and Outline: Anne-Marie Kermarrec
Detailed Design, Writing, Tests: Erick Lavoie
Teaching Assistant: Athanasios Xygkis
Last Updated: 2021/04/23 12:38:24 +02'00'

Due Date: 30-04-2021 23:59 CET
Submission URL: `https://cs449-sds-2021-sub.epfl.ch:8083/m2`
General Questions on Moodle
Personal Questions: athanasios.xygkis@epfl.ch

Abstract

In this milestone, you will develop empirical understanding of a simple implementation of the k-NN algorithm by using it in your recommender system from the last Milestone. You will also analyze and measure the memory consumption and CPU time used to make predictions to compare against the previous simpler baseline.

1 Motivation: *Personalized* Recommendations

While some movies are highly rated by most users, most movies might appeal only to subsets of users [1]. In effect, to best answer the tastes of a maximum of users, you would like to provide *personalized* recommendations beyond the usual blockbusters. The approach you will try in this milestone 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. Keeping all similarity values in memory, and using them for prediction, may be prohibitive in memory and computation time, so you will only keep the k most similar in a user's neighbourhood to reduce both, effectively basing your design on the k-NN algorithm.

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

1.1 Dataset: MovieLens 100K

For this milestone, you will again use the MovieLens 100K dataset [1]. Again, for the sake of simplicity, you will only test on the `ml-100k/u1.test` dataset (with the corresponding `ml-100k/u1.base`).

2 Similarity-based Predictions

The global average deviation (Milestone 1, 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):

$$s_{u,v} = \begin{cases} \frac{\sum_{i \in (I(u) \cap I(v))} \hat{r}_{u,i} * \hat{r}_{v,i}}{\sqrt{\sum_{i \in I(u)} (\hat{r}_{u,i})^2} * \sqrt{\sum_{i \in I(v)} (\hat{r}_{v,i})^2}} & (I(u) \cup I(v)) \neq \emptyset; \exists_{i \in I(u)} \hat{r}_{u,i} \neq 0; \exists_{i \in I(v)} \hat{r}_{v,i} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

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:

$$\bar{\hat{r}}_{\bullet,i}(u) = \begin{cases} \frac{\sum_{v \in U(i)} s_{u,v} * \hat{r}_{v,i}}{\sum_{v \in U(i)} |s_{u,v}|} & \exists_{v \in U(i)} s_{u,v} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The prediction equation is similar to Eq. 5 from Milestone 1 but incorporates the user-specific $\bar{\hat{r}}_{\bullet,i}(u)$ of Eq. 2 instead of $\bar{\hat{r}}_{\bullet,i}$ of Eq. 4 from Milestone 1.

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

2.1 Preprocessing Ratings

Notice that each term of the denominator of Eq. 1, 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 $\hat{r}_{u,i}$:

$$\breve{r}_{u,i} = \begin{cases} \frac{\hat{r}_{u,i}}{\sqrt{\sum_{j \in I(u)} (\hat{r}_{u,j})^2}} & \text{if } i \in I(u) \text{ and } \exists_{j \in I(u)} \hat{r}_{u,j} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

You can use that property to preprocess ratings such that only the numerator multiplication of Eq. 1 remains to be computed:

$$s_{u,v} = \sum_{i \in (I(u) \cap I(v))} \breve{r}_{u,i} * \breve{r}_{v,i} \tag{5}$$

This should make your implementation faster.

In the following questions, you will measure the effect of the previous similarity method on prediction accuracy, as well as analyze and measure its additional CPU and memory usage.

2.2 Questions

1. Compute the prediction accuracy (MAE on `ml-100k/u1.test`) of (Eq. 3). Report the result. Compute the difference between the Adjusted Cosine similarity and the baseline (*cosine* − *baseline*). Is the prediction accuracy better or worst than the baseline (Eq. 5 from Milestone 1)? (If you are reusing some of your code of Milestone 1, use your previous baseline MAE. Otherwise, use a baseline MAE of 0.7669.)

2. Implement the Jaccard Coefficient[2]. Provide the mathematical formulation of your similarity metric in your report. Compute the prediction accuracy and report the result. Compute the difference between the Jaccard Coefficient and the Adjusted Cosine similarity (Eq. 1, *jaccard* − *cosine*) Is the Jaccard Coefficient better or worst than Adjusted Cosine similarity?

3. In the worst case and for any dataset, how many $s_{u,v}$ have to be computed, as a function of the size of U (the set of users), if every user rated at least one item in common with every other users? (Provide the formula in your report) How many would that represent if that was the case in the 'ml-100k' dataset? (Compute the answer in your code from the number of users in the input dataset)

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

4. Compute the minimum number of multiplications required for each possible $s_{u,v}$ on the `ml-100k/u1.base`.[3] (Tip: This is the number of common items, $|I(u) \cap I(v)|$, between u and v.) What are the min, max, average, and standard deviation of the number of multiplications? Report those in a table.

5. How much memory, as a function of the size of U, is required to store all possible $s_{u,v}$, both zero and non-zero values, assuming both require the same amount of memory? How many bytes are needed to store only the non-zero $s_{u,v}$ on the 'ml-100k' dataset, assuming each non-zero $s_{u,v}$ is stored as a double (64-bit floating point value)? (Tip: Do not include memory usage for intermediate computations, only for storing the final results.)

6. Measure the time required for computing predictions (with 1 Spark executor), including computing the similarities $s_{u,v}$. (Tip: If you compute the similarities in batch prior to predictions, include the time for computing them all. If you compute similarities on a by-need basis, possibly with caching, include the time for all those that were actually computed.) Provide the min, max, average, and standard-deviation over ~~ten~~ **five** measurements. Discuss in your report whether the average is higher than the previous methods you measured in Milestone 1 (Q.3.1.5)? If this is so, discuss why.

7. Measure only the time for computing similarities (with 1 Spark executor). (Tip: If you compute the similarities in batch prior to predictions, include the time for computing them all. If you compute similarities on a by-need basis, possibly with caching, include the time for all those that were actually computed.) Provide the min, max, average and standard-deviation over ~~ten~~ **five** measurements. What is the average time per $s_{u,v}$ in microseconds? On average, what is the ratio between the computation of similarities and the total time required to make predictions? Are the computation of similarities significant for predictions? (Tip: To lower your total running time, you can combine this measurement in with that of the previous question in the same runs.)

2.3 Tips

- Using equal similarities, such as $s_{u,v} = 1$ for all users, should result in a MAE for Eq. 3, equal to that of Eq.5 of Milestone 1. Once, you have ensured this, the only reason why you may not observe an improvement compared to the baseline is that your similarity computation (Eq. 1) is incorrect.

[3] This is a lower bound. As an alternative, you could choose to implement the numerator of Eq. 1 with a matrix multiplication or dot product, by setting to 0 the value of ratings not provided by either user. This would increase the number of multiplications but may still result in a faster implementation than performing a set intersection beforehand.

- The denominator of Eq. 1 and Eq. 4 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. 2) 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.

3 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. 2) are not considered.

Keeping only the k nearest neighbours, according to a similarity metric (ex: Eq. 1), 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. You will also investigate the reduction in memory usage possible, and the capabilities of modern hardware to support large number of users.

3.1 Questions

1. What is the impact of varying k on the prediction accuracy? Provide the MAE (on `ml-100k/u1.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 method (Eq. 5 of Milestone 1)? How much lower? (Use a baseline MAE of 0.7669, and compute $lowestk - baseline$)

2. What is the minimum number of bytes required, as a function of the size of U, to store only the k nearest similarity values for all possible users u, i.e. top k $s_{u,v}$ for every u, for all previous values of k (with the `ml-100k` dataset)? Assume an ideal implementation that stored only similiarity values with a double (64-bit floating point value) and did not use extra memory for the containing data structures (this represents a lower bound on memory usage). Provide the formula in the report. Compute the number of bytes for each value of k in your code.

3. Provide the RAM available in your laptop. Given the lowest k you have provided in Q.3.1.1, what is the maximum number of users you could store in RAM? Only count the similarity values, and assume you were storing values in a simple sparse matrix implementation that used 3x the memory

than what you have computed in the previous section (2 64-bit integers for indices and 1 double for similarity values).

4. Does varying k has an impact on the number of similarity values ($s_{u,v}$) to compute, to obtain the exact k nearest neighbours? If so, which? Provide the answer in your report.

5. Report your personal top 5 recommendations with the neighbourhood predictor (Eq. 3) with $k = 30$ and $k = 300$. How much do they differ between the two different values of k? How much do they differ from those of the previous Milestone?

   - If you are using your `personal.csv` file from last Milestone, modify line 177 to use the updated title `The Good the Bad and the Ugly`, without quotes or comma, if that was not already the case. The previous versions in Template 1 instead used `"Good, the Bad and the Ugly The"` which tripped up some simple CSV parsers. The empty `personal.csv` file from the second template already incorporates the change.
   - Please ensure the `personal.csv` file with your ratings does use commas (`','`) and not semi-colons (`';'`) as separators after being saved, as sometimes Excel does. The simplest way is simply to open the file in a text editor, such as Notepad, and add your ratings at the end of the line.

3.2 Tips

- Check that the prediction scores for the recommended items are indeed close to 5.
- You may be tempted to avoid storing all similarities, and instead keep only the top k as they are computed to make your code efficient. Remember to always make your code *correct* first, and *efficient* later: this way you can compare your optimized version against a correct one to ensure you are not breaking anything.
- You don't need to recompute all similarities to answer for smaller K values, only to select a smaller subset. You can also compute the results from the largest K to the smallest, by selecting the top k from the smaller set each time. This may significantly speed up your computations.

4 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-M2/-/archive/master/cs449-Template-M2-master.zip`

- Git Repository:

  ```
  git clone

  https://gitlab.epfl.ch/sacs/cs-449-sds-public/project/cs449-Template-M2.
  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-M2` 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) and your personal movie ratings in a single archive:

```
CS449-YourID-M2/
    README.md
    report-YourID-M2.pdf
    build.sbt
    data/personal.csv
    project/build.properties
    project/plugins.sbt
    src/main/scala/similarity/Predictor.scala
    src/main/scala/knn/Predictor.scala
    src/main/scala/recommend/Recommender.scala
```

Add 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 `similarity.json`, `knn.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. You archive should be around or less than 1MB.

5 Grading

We will use the following grading scheme:

	Points
Questions	25
Source Code Quality & Organisation	5
Total	30

Points for 'Source Code' will reflect how easy it was for the TA to run your code and check your answers. We will enforce both the location of input files (`personal.csv`), the minimal project structure of the previous section, as well as the JSON output format: deviations will cost points. If you really need to change any of those, please ask on the Moodle forum first.

Grading for answers to the questions without accompanying executable code will be 0.

Ensure your code takes less than 10 minutes to produce all answers on your machine, we will kill scripts that take more than that amount of time when grading. Unanswered questions will obtain 0.

5.1 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.

6 Updates

Since the original release of the Milestone description on April 8th, we have made the following changes:

- Explicitly put the definition of $\hat{r}_{u,i}$ from Milestone 1 in the Notation section to remove any ambiguity. Prompted by Raphael and Sorin-Sebastian

questions: https://moodle.epfl.ch/mod/forum/discuss.php?d=57655#p117259.

- Added clarification that a user's self-similarity should not be included in k-NN. Prompted by Dhruti's question: https://moodle.epfl.ch/mod/forum/discuss.php?d=57768#p117495.
- The number of repetitions used to make measurements has been lowered from 10 to 5, following Hugo's suggestion in a private email.
- Added clarifications that answers for questions 2.3.3 to 2.3.5 are about *all possible* similarity values, as this represents a worst case and helps understand the costs and benefits of k-NN of the next section; Added additional clarifications for 2.3.6-2.3.7 that time for making similarity computations may include *only those required for predictions* is this number may be lower (those computing all similarities prior to making predictions should include the time for all of them). Prompted by Tiannan's question: https://moodle.epfl.ch/mod/forum/discuss.php?d=58262#p118427.
- Added clarification that memory usage estimation should not include that needed by intermediate computations for $s_{u,v}$. Prompted by Pauline's question: https://moodle.epfl.ch/mod/forum/discuss.php?d=58659#p119185.
- Added clarification that stats in 2.3.4 are computed on the number of multiplications, not the values of similarities computed. Prompted by Dhruti's question: https://moodle.epfl.ch/mod/forum/discuss.php?d=58533#p118958.
- Added clarification on edge cases that may induce division by zero in equations 1, 2, and 4. Prompted by questions from Mateo and Stephan: https://moodle.epfl.ch/mod/forum/discuss.php?d=58082#p118109.
- Added clarification that you can reuse your baseline MAE from Milestone 1, if using the same prediction code. Prompted by a question from Maina: https://moodle.epfl.ch/mod/forum/discuss.php?d=58444#p118800.
- Added tip for speeding up computations for k-NN. Prompted by Tiannan's question: https://moodle.epfl.ch/mod/forum/discuss.php?d=57772#p119169.
- Removed the suggested implementations in Scala, as students have found an RDD-based implementation to be faster and more in line with their previous exercise sessions: the suggestion was more confusing than helpful.

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: $\bar{r}_{u,\bullet} = \frac{\sum_{r_{u,i} \in \text{Train}} r_{u,i}}{\sum_{r_{u,i} \in \text{Train}} 1}$
- $\hat{r}_{u,i}$: normalized deviation from the average $\bar{r}_{u,\bullet}$:

$$\hat{r}_{u,i} = \frac{r_{u,i} - \bar{r}_{u,\bullet}}{scale(r_{u,i}, \bar{r}_{u,\bullet})} \tag{6}$$

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

$$scale(x, \bar{r}_{u,\bullet}) = \begin{cases} 5 - \bar{r}_{u,\bullet} & \text{if } x > \bar{r}_{u,\bullet} \\ \bar{r}_{u,\bullet} - 1 & \text{if } x < \bar{r}_{u,\bullet} \\ 1 & \text{if } x = \bar{r}_{u,\bullet} \end{cases} \tag{7}$$

- $r_{u,i}$: 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 ($\{i|r_{u,i} \in \text{Train}\}$)
- $I(u) \cap I(v)$): Intersection of (common) items rated by both u and v

CS-449 Project Milestone 3: Optimizing, Scaling, and Economics

Motivation and Outline: Anne-Marie Kermarrec
Detailed Design, Writing, Tests: Erick Lavoie
Teaching Assistant: Athanasios Xygkis
Last Updated: 2021/05/17 17:04:57 +02'00'

Due Date: 04-06-2021 23:59 CET
Submission URL: `https://cs449-sds-2021-sub.epfl.ch:8083/m3`
General Questions on Moodle
Personal Questions: athanasios.xygkis@epfl.ch

Abstract

In this milestone, you will parallelize the computation of similarities by leveraging the Breeze linear algebra library for Scala, effectively using more efficient low-level routines. You will also measure how well your Spark implementation scales when adding more executors. You will finally compute economic ratios to help you choose the most appropriate infrastructure for your needs.

1 Motivation: Growing Pains

Your movie recommendation service is growing in number of users and movies rated, beyond the hobby side project you had started with. As your service is growing, the time to compute the *k-nearest neighbours* is growing also. If left unchecked, this may exceed the time your users are willing to wait to obtain high-quality suggestions.

In this last Milestone you are going to keep computation time within reasonable bounds in two steps: (1) you will first optimize your implementation for a single executor, aiming for a speedup of at least 10x compared to Milestone 2 (Section 3); (2) you will parallelize the execution of both the *k-NN* implementation and the predictions with Spark, using simple parallelization methods you have already used in the exercises (Section 4). The optimization will keep your technical requirements lower, and the scaling will enable you follow the growth of your users easily.

As your service is growing, so does the cost of the infrastructure you are using. In order to plan for future growth, in the most affordable way possible,

you will compute a few simple economic ratios to compare some currently available options for the supporting hardware infrastructure, either in a private or public cloud, to select one that provides you with the best bang for your buck (Section 5).

After completing this Milestone and the project, you will have had a basic but complete experience in implementing useful data analytics algorithms, measuring and analyzing their computing time and memory usage, optimizing their performance both on a single machine and in a distributed fashion, and planning required resources for future growth. You should therefore have a good foundation for pursuing a research career in distributed systems, doing more advanced development on real-world infrastructure, as well as leading a technical team in a startup.

2 Dataset

For the optimization part of this milestone, you will again use the MovieLens 100K dataset [3]. Again, for the sake of simplicity, you will only test on the `ml-100k/u1.test` dataset (with the corresponding `ml-100k/u1.base`). This will enable you to evaluate performance gains compared to your implementation of Milestone 2.

For the scaling part of this milestone, you will use a larger dataset in order to simulate a larger computation load on your service. Using the techniques of this Milestone, you could easily scale to 10M or 25M ratings while retaining a computation time that is within the bounds of a real-world service. However, to make your (busy) student life easier, you will test with the smaller 1M MovieLens dataset, which should take at most a minute to process (once optimized). You you can find it here:

`https://grouplens.org/datasets/movielens/1m/`

However, the 1M dataset does not provide pre-split training and test sets, as the 100K, so you will use the scripts of the 10M dataset to split the ratings into a `ml-1m/ra.train` and `ml-1m/ra.test` datasets. Instructions for downloading the dataset, copying the scripts, and performing the split are provided in the template. As for the 100K dataset, in a real-life setting you would test on multiple splits of the dataset to ensure your results are not specific to one split and most likely generalize. But for the sake of the pedagogical exercise, it will be sufficient to only test on the `ml-1m/ra.train` and `ml-1m/ra.test` split.

3 Optimizing with Breeze, a Linear Algebra Library

Before throwing more computing resources at the problem, e.g. Spark Executors running on additional cluster machines, it is always worth verifying whether a single machine would actually be sufficient for the task. In some cases, as is the case for your recommender, a drastic reduction in computation time can be

achieved with slightly different, but mathematically equivalent, implementation techniques.

In the case of the recommender of this project, the performance of the same prediction algorithm in Python/Numpy shows that there is major room for improvement: we obtained computation times of 0.05s for computing all similarities and k-nearest neighbours on the `ml-100k/u1.base` dataset and 0.15s for computing all predictions on the `ml-100k/u1.test` dataset. Compared to students' reported execution times in Scala/Spark for Milestone 2, that were in the order of 20s of seconds for computing similarities and close to a minute for computing predictions, this represents potential for speedups of $400x$. The major difference with Python/Numpy compared to using Spark's RDDs and DataFrame APIs, is the use of mature linear algebra libraries with efficient multi-core vectorized implementations.

As far as we are aware of, Scala does not offer complete and efficient equivalents to the Numpy libraries yet. The Breeze library[1] does however come somewhat close on a subset of operations and datatypes. Breeze is the recommended interface for Scala/Spark to the underlying netlib-java library[2], itself a wrapper for low-level linear algebra libraries such as BLAS and LAPACK, that is used internally by the Spark mllib library. In this section, you will therefore use Breeze directly to reimplement the prediction algorithm of Milestone 2 and measure the speedup you can obtain. With relatively simple implementation techniques, you should obtain at least a $10x$ reduction in execution time. This can translate in at least a $10x$ reduction in the number of executors required to obtain the same computation time, potentially more depending on the overhead of RDD and Dataframe libraries and their parallization behaviour.

Note that to best leverage a linear algebra library, your implementation should compute *all* k-nearest neighbours prior to making predictions, instead of lazily based on which predictions to make, as the RDD API might have encouraged you to do in Milestone 2. You will notice that the speed up you obtain this way stays quite large even if you are actually computing more similarities than previously.

3.1 Overview of some Breeze Operations

Unfortunately, the breeze library has been on minimal maintenance mode for a couple of years[3] and the documentation is somewhat lacking[4]. To get you started, here is an overview of some operations we found useful in our reference implementation.

For all of them you should first import:

[1] `https://github.com/scalanlp/breeze`

[2] `https://github.com/fommil/netlib-java`

[3] `https://github.com/scalanlp/breeze/commits/master`

[4] This is no fault of the author, as the library does quite a lot as a labor of (volunteer) love, and as such is already quite an accomplishment. This highlights the issue of funding to maintain core open source infrastructure. If you ever work for a company that makes money using open source software, do remember to pay the benefits forward and sustain the open source commons by funnelling some funding towards the projects you depend on.

```
import breeze.linalg._
import breeze.numerics._
```

3.1.1 Dense Vector and Matrix

If all the data can fit in memory, the most efficient numerical operations are usually based around the `DenseVector` and `DenseMatrix` data types. There is a fairly comprehensive library for linear algebra, which supports efficient slicing, element-wise operations, and matrix multiplication. The set of operations and their equivalence in other numerical languages is listed here: `https://github.com/scalanlp/breeze/wiki/Linear-Algebra-Cheat-Sheet`.

When working with the `ml-100k/u1.base` dataset, a `DenseMatrix` can easily fit all the ratings in memory, even if they are sparse compared to $users * items$. Prefer a sparse matrix anyway since it makes it easier to try larger datasets with the same implementation.

3.1.2 Sparse Matrix

If a matrix, e.g. user-item ratings, does not fit in memory using a `DenseMatrix`, as is necessarily the case for the largest MovieLens datasets, you can use a `SparseMatrix` representation. The Breeze library only supports the Compressed Sparse Column (CSC) format[5]. Here is a summary of operations you might find useful.

Creation

You can efficiently create a `CSCMatrix` with the `CSCMatrix.Builder` as follows:

```
val builder = new CSCMatrix.Builder[Double](rows=943, cols=1682)
builder.add(row, col, value)
...
val x = builder.result()
```

The template already does this for the ratings, both for training and testing.

Slicing

You can obtain a `SliceVector` of rows i to j of column m with `x(i to j, m)` or a `SliceMatrix` of row i to j of columns m to n with `x(i to j, m to n)`. Note that if you slice to obtain a single row i between columns m and n you will obtain a "transposed" vector, which you may have to transpose again for some operations as they may not support the transposed version. Note also that the `::` operator (ex: `x(i, ::)`), which provides all elements along the corresponding dimension, is not supported on `SparseMatrix`. You have to explicitly provide a range `i to j` for that dimension.

[5] See Wikipedia for an overview of other formats: `https://en.wikipedia.org/wiki/Sparse_matrix`

Iteration

You can efficiently iterate through all non-zero elements of a `CSCMatrix` matrix x with:

```
for ((k,v) <- x.activeIterator) {
  val row = k._1
  val col = k._2
  // Do something with row, col, v
}
```

Matrix multiplication

You can multiply matrix `x` by matrix `y` (both can also be slices with compatible dimensions) with `x * y`. You can use it to implement the $s_{u,v}$ computation with pre-processed ratings of Milestone 2:

$$s_{u,v} = \sum_{i \in (I(u) \cap I(v))} \breve{r}_{u,i} * \breve{r}_{v,i} \tag{1}$$

Operating on slices of two CSCMatrix (or the same) is however slower than performing a matrix-vector multiplication, see below.

Matrix-Vector multiplication

After a fair amount of experimentation, it seems that a `CSCMatrix-DenseVector` multiplication in Breeze is significantly faster than operating on `SliceMatrix` of a `CSCMatrix`, even when taking into account the conversion of a `SliceVector` to a `DenseVector` prior to the multiplication and the explicit iteration through rows/columns. I therefore recommend you organize the main *k-NN* computation, as defined in Eq. 1, around this primitive. To still keep a little bit of the fun in experimenting with different possible implementations, you will have to figure out yourself whether it is best to assign users to rows and items to columns of a `CSCMatrix` (or the opposite!) when preparing the $\breve{r}_{u,i}$ sparse matrix.

Reduction

The reduction along a given dimension, such as performing a `sum(x, Axis._1)` is not supported. You can either explicitly create a `DenseVector` with the correct size for that dimension then iterate through all active (non-zero) elements as shown above. You can also do a matrix multiplication: for a matrix X, you can reduce the columns by multiplying $X_{m,n} * 1_{n,1}$ or the rows by multiplying $1_{1,m} * X_{m,n}$, where $1_{1,n}$ is an `n-by-1` matrix of all 1 and $1_{1,m}$ is a `1-by-m` matrix of all 1. Either options are useful for pre-processing the ratings, i.e. computing $\breve{r}_{u,i}$, prior to computing $s_{u,v}$, you will have to measure which one is fastest.

top-k elements

You can find the indices of the top `k` elements of a (non-transposed) `SliceVector` `s` with `argtopk(s, k)` and iterate through them with:

```
for (i <- argtopk(s,k)) {
  // do something with s(i)
}
```

It can sometimes be faster to call `argtopk` with a `DenseVector` instead of a `SliceVector`. You will have to measure whether this is significant.

Element-wise operations

Many element-wise operations and boolean operations from `https://github.com/scalanlp/breeze/wiki/Linear-Algebra-Cheat-Sheet` are supported on `CSCMatrix`. Unsupported operations will result in compilation errors. You can quickly and interactively identify and test those that are supported with `sbt console` on a small test `CSCMatrix`, which you may create as such:

```
> val x = CSCMatrix[Double]((1.0, 2.0, 3.0), (4.0, 5.0, 0.0))
// x.<tab> for autocompletion of supported operations
```

Others

See `https://github.com/scalanlp/breeze/blob/06b23cfa837c53e025bb70f0f4bc1f241986d0ba/math/src/test/scala/breeze/linalg/CSCMatrixTest.scala` for example usage of other supported operations on `CSCMatrix`. If you find success with additional operations, please share examples as above on the forum (but without giving away a complete solution) to help others.

3.2 Questions

1. Reimplement the `Predictor` of Milestone 2 using the Breeze library and without using Spark. Test your implementation on the 'ml-100k/u1.base' as a training set, and the 'ml-100k/u1.test' as a test dataset, and $k = 200$. Ensure your implementation gives a MAE of 0.7485 (within 0.0001). Output the MAE for $k = 100$ and $k = 200$ (We will use both for grading). (Make sure to make self-similarities equal to 0, i.e. $s_{u,u} = 0$, prior to computing the k-nearest neighbours.)

2. Measure the time for computing all *k-nearest neighbours* (even if not all are used to make predictions) for all users on the `ml-100k/u1.base'` dataset. Output the min, max, average, and standard-deviation over 5 runs in your implementation.

3. Measure the time for computing all 20,000 predictions of the 'ml-100k/u1.test'. Output the min, max, average, and standard-deviation over 5 runs in your implementation.

4. Compare the time for computing all *k-nearest neighbours* of 'ml-100k/u1.base' to the one you obtained in Milestone 2 (Q.2.2.7). What is the speedup of your new implementation (as a ratio $\frac{t_{old}}{t_{new}}$)? Why do you think that is the case? (Ensure you have obtained at least a 10x decrease in execution time

for the *k-nearest neighbours* and predictions compared to Milestone 2. If you did not compute all *k-NN* in Milestone 2, use the time your reported for computing similarities.).

3.3 Tips

1. For any kind of optimization work, always make sure to first have a correct (and often simpler) implementation to compare against, to ensure your optimizations do not affect the correctness of the results. As you develop, compare the answers for both and ensure they answers stay the same. Your implementation of Milestone 2 can serve for this.

2. Prior to optimizing, make sure to measure how long each part of your program takes. Our intuitions of what is fast and slow is (most) often wrong, so always measure first to focus you attention on the parts that matter most.

3. Always tackle the slowest parts first, then move to the others. As you optimize, the bottlenecks, i.e. the parts that slow your program the most, will change.

4. Optimization work can be addictive. It also suffers from diminishing returns, i.e. the biggest gains can be obtained with relatively little work but further gains will take increasingly larger amounts of work. For the sake of the project, we are aiming for a $10x$ reduction in computation time for both the *k-NN* computations and the prediction time. For a more precise reference see below.

5. As a reference, without using Spark and running on the `iccluster041.iccluster.epfl.ch`, we have obtained a time of $\approx$ 1s for computing all *k-NN*s on `ml-100k/u1.base`, prior to making predictions, and less than 4s for making all 20,000 predictions. You should be able to obtain similar results with a reasonable and systematic effort. If in doubts, share your running times on the forum.

4 Scaling with Spark

In a real-world setting, your number of users may outgrow the capabilities of a single machine, even after extensive optimization work. In that case, distributing the execution on multiple machines allows your implementation to grow with the number of users. The two main parts worth parallelizing are *k-NN* and making predictions.

For this Milestone, you will use a simple parallelization strategy in which every worker (Spark executor) will obtain a copy of the input data and they will compute different parts of the output. The distribution of the shared input data is done efficiently with Spark's *broadcast variables*.

For *k-NN*, workers will compute the top k users for different subsets of users. The results are then reassembled on the driver node in a sparse *k-NN* `CSCMatrix`. A high-level presentation of the parallelization strategy is given in Algorithm 1. A similar pattern can be used to make predictions by parallelizing over pairs of user-items (u, i) to output a tuple user-item-prediction $(u, i, p_{u,i})$ and storing them in a `CSCMatrix`. The MAE is then simply `sum(abs(pred-test))/test.activeSize`.

Algorithm 1 Parallel k-NN Computations with Replicated Ratings (Spark)

Input $r_{\bullet,\bullet}$ (ratings), *sc* (SparkContext), k (number of nearest neighbours)
$\breve{r}_{\bullet,\bullet} \leftarrow$ preprocess($r_{\bullet,\bullet}$) ▷ Similar to Milestone 2
$br \leftarrow sc$.broadcast($\breve{r}_{\bullet,\bullet}$)
procedure *topk*(u)
    $\breve{r}_{\bullet,\bullet} \leftarrow br$.value ▷ Retrieve broadcast variable
    $s_{u,\bullet} \leftarrow similarities(u, \breve{r}_{\bullet,\bullet})$ ▷ Using Matrix-Vector multiplication
    return $(u, argtopk(s_{u,\bullet}, k).map(v \rightarrow (v, s_{u,v})))$ ▷ Compute k-NN for user u
end procedure
$topks \leftarrow sc.parallelize(0$ **to** $nb\_users).map(topk).collect()$
$knn \leftarrow knnBuilder(topks)$ ▷ Using the CSCMatrix.Builder
return *knn*

4.1 Questions

In the following questions, you will ensure this distributed version gives the same results you obtained previously and assess its scalability and resource usage.

1. Test your spark implementation with the `ml-1m/ra.train` as a training set, and the `ml-1m/ra.test` as a test dataset, and $k = 200$ (the command to do so is provided in the README of the template). Output the MAE you obtain.

2. Manually run your implementation on the cluster 5 times and compute the average. Compare the average *kNN* and *prediction* time to your optimized (non-Spark) version of Section 3 on the `ml-1m/ra.train` dataset. For the Spark version of this section, use a single executor, i.e. `--master "local[1]"` when using `spark-submit`. Are they similar? If not, how much overhead does Spark add? Answer both questions in your report.

3. Measure and report *kNN* and *prediction* time when using 1, 2, 4, 8, 16 executors on the IC Cluster in a table. Perform each experiment 3 times and report the average, min, and max for both *kNN* and *predictions*. Do you observe a speedup? Does this speedup grow linearly with the number of executors, i.e. is the running time X times faster when using X executors compared to using a single executor? Answer both questions in your report.

4.2 Tips

1. Use the following command to vary the number of executors on the IC-Cluster (once connected on the `iccluster041.iccluster.epfl.ch` driver node): `spark-submit --master yarn --num-executors X ...` where `X` is the number of executors to use.

5 Economics

You have seen that the optimizations of Section 3 may translate in lower needs for additional hardware. In this section, you will quantify the economics of buying/renting hardware to execute your recommender. You will, in the process, obtain insights into the economic benefits of optimization as well as develop an ability to estimate the running costs of a given implementation.

These days, you have many choices of hardware infrastructure. The landscape and particular tradeoffs of each point, are too broad to cover in this project. But since it may affect some design decisions, let's quickly review three representative examples of currently available hardware, summarized in Table 1, and provide some context in which they may be useful.

First, you may choose not to distribute your similarity computations and instead replace your re-purposed desktop machine, with a more powerful one, i.e. with increased RAM and better processor performance. One representative example is a powerful server. For example, the server in the IC IT Cluster with the largest amount of main memory, at 1.5TB of RAM, is listed in Table 1. The main advantage of this approach is that you don't have to change your software implementation, which sometimes is not possible because your particular algorithms are not easily distributed, and other times might be too complex for the skills or development time you currently have. However, the main drawback is that higher-end machines are usually more expensive because they are produced in lower volumes and sold at higher margins. The running costs at idle are also higher because all the added circuitry needs to be powered, even if not used.

Second, if your algorithms can be distributed, which fortunately is the case for k-NN [5, 4], you may elect to distribute them on virtualized cloud infrastructure. Today's cloud providers have many virtualization offerings, with one example based on containers, i.e. isolated processes running within the same operating system. Virtualization makes available the underlying hardware at a finer granularity, e.g. per CPU per second, as listed in the second line of Table 1. The pricing structure for the IC IT cluster suggests that the same hardware is rented in both cases, the previous case being equivalent to renting upfront all the capabilities of the hardware for an entire day. The main advantages of virtualized cloud offerings are potentially reduced maintenance costs, as the maintenance of the hardware is centralized and shared between multiple

[6] From EUR to CHF, as listed here (Jan. 20th 2021) `https://buyzero.de/products/compute-module-4-cm4?variant=32090358612070&src=raspberrypi`

[7] Electricity (3W (idle) - 15W (max) `https://www.raspberrypi.org/documentation/hardware/raspberrypi/power/README.md`, 0.15CHF/kWh)

Hardware	**RAM**	**CPU**	**Buying Costs**	**Operating Costs**
ICC.M7	24x64GB DDR4 -2666: 1.5 TB	2x Intel Xeon Gold 6132 (Skylake) 2x 14 cores, 2.6 GHz, 19.25 MB L3 cache	–	20.40CHF/day
Containers	Unspec.	Unspec.	–	1 GB-s: $1.5e^{-7}$CHF/s (0.012CHF/day) 1 vCPU-s: $1.02e^{-6}$CHF/s (0.092CHF/day) 1 vGPU-s: $6.583e^{-5}$CHF/s (5.68CHF/day)
RPi 4 Compute Module	8GB LPDDR4 -3200 SDRAM	Broadcom BCM2711 quad-core Cortex-A72 ARMv8 64-bit SoC@1.5GHz	94.83CHF[6]	0.0108CHF/day - 0.054CHF/day[7] + Maintenance

Table 1: Technical specifications and cost of IC IT cloud offerings, compared to the highest performing Raspberry Pi available in 2021, a cheap and widely available consumer computing device.

cloud users, and offloading the risks associated with under- or over-provisioning hardware infrastructure compared to actual business needs (you only pay what you use). However, if the actual maintenance costs for some applications are low, and computing needs are stable and foreseeable, the cost of acquiring hardware may actually be recouped in a few years. Moreover, if user data is subject to stringent privacy limitations, it may sometimes not be possible to process it in a cloud (although there are plenty of ongoing research and industrial developments aiming to provide better privacy guarantees). Also, the electronic waste, carbon emissions, and energy usage, of cloud providers may be higher than what you could obtain with privately-operated infrastructure [2].

Third, you may distribute your algorithms on hardware you acquire and maintain yourself, by favouring cheap, energy-efficient, and widely available commodity hardware, an approach that was pioneered by Google in 1998. For example, the highest performing Raspberry Pis are increasingly closing the gap with Desktop and Server performance at a really low price point. The specification and costs, as of January 2021, are listed in the third line of Table 1. While there is still a significant gap in performance with the most powerful hardware of today, RaspberryPi 4 Compute Modules are on par or more powerful than the servers used by Google in 2005, are more energy efficient by a factor of 4-5x, and probably less expensive by a factor at least 4-5x [1]! They provide the same advantages as Hardware-as-a-Service, in having full access to the hardware and their operating costs (when only taking electricity into account) are lower than equivalent virtualized offerings for the same performance. Moreover, private data never leaves the organization, the hardware can be used for longer than the typical 2-3 years turn-around time of cloud servers (lowering e-waste and grey energy), and they can be setup to operate on renewable energy at reasonable costs [2]. However, setting up the infrastructure and maintaining the software up-to-date requires labor time, which may more than offset the lower operating costs. They also have higher initial costs compared to cloud offerings, but, given the low price of individual devices, that cost may be spread over time and engaged as computing needs grow.

The following questions will help you analyze how technical capabilities relate to application needs, and acquire or rent the right amount of hardware resources for your current and future needs.

5.1 Questions

For the following questions, make the following assumptions[8]:

- Throughput of 1 single Intel CPU core $\approx$ 4 Cortex-A72 (16 cores total) [9]

[8] For real deployments, you would have to empirically verify these assumptions, and revise them as the application, the landscape of hardware offerings, and your software implementation evolve.

[9] Rough estimate based on the performance of the more recent Intel Xeon E3-1230 v6, which has 4 cores instead of 14, on the following benchmark: `https://truebench.the-toffee-project.org/`. Note that the Intel cores have hyper-threading, which doubles some of hardware to obtain twice the number of virtual CPUs and is clocked almost twice

- Containers use the same CPU and RAM as the ICC.M7 machine
- The ICC.M7 costs 35,000 CHF (based on a 39,000$USD quote from the Dell website in the US, created on Jan. 21st 2021)
- Users rate on average 100 movies
- A good k for k-NN, to obtain a significant gain in accuracy compared to average global deviation, is 200
- The nodes should hold in memory both (1) the similarity values obtained from k-NN and (2) all movie ratings for all users, by distributing them roughly equally between many nodes, while still leaving 50% of memory free for computation and other tasks
- Use floats (32-bit floating point numbers) for similarity values $s_{u,v}$ and 8-bit unsigned ints for ratings ($r_{u,i}$)
- 1 GB = 10^9 Bytes (manufacturers sometimes use 1GB = 1024^3)
- Assume a year is always 365 days (no leap years)

Answer the following questions:

1. How many days of renting does it take to make buying the ICC.M7 less expensive? How many years does that represent?
2. What is the daily cost of an IC container with the same amount of RAM and CPUs (1 vCPU = 1 core) as the ICC.M7 machine? What is the ratio of $\frac{ICC.m7(CHF/day)}{Container(CHF/day)}$? Is the container cheaper (consider a difference of less than 5% to be negligible)?
3. Idem, but compared to 4 Raspberry Pis (to obtain the same CPU throughput)? What is the ratio $\frac{4RPis(CHF/day)}{Container(CHF/day)}$ at max power for the RPi? at min power for RPi? Is the container cheaper than the 4RPis (consider a difference of less than 5% to be negligible)?
4. After how many days of renting a container, is the cost higher than buying and running 4 Raspberry Pis? (1) Assuming optimistically no maintenance at minimum power usage, and (2) no maintenance at maximum power usage, to obtain a likely range. (Round up days)
5. For the same buying price as an ICC.M7, how many Raspberry Pis can you get? Assuming perfect scaling, would you obtain a larger overall throughput and RAM from these? If so, by how much (ratio)?
6. How many users can be held in memory per GB? Per RPi? Per ICC.M7?
7. Based on your answers, which would be your preferred option of the three? Would you buy or rent? Why?

higher, so the actual performance per ARM core is actually much better than it may seem at first sight.

5.2 Tips

1. Write down the full equation with units for every term when calculating ratios.

6 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-M3/-/archive/master/cs449-Template-M3-master.zip`
- Git Repository:

  ```
  git clone

  https://gitlab.epfl.ch/sacs/cs-449-sds-public/project/cs449-Template-M3.git
  ```

We may update the template to clarify or simplify some aspects based on student feedback during the next weeks, so please refer back to `https://gitlab.epfl.ch/sacs/cs-449-sds-public/project/cs449-Template-M3` 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). Use the sub-section numbers of this document to group your answers, i.e. Section 3.2 and 5.1. Also, provide your source code (in Scala):

```
CS449-YourID-M3/
    report-YourID-M3.pdf
    optimizing/build.sbt
    optimizing/src/main/scala/Predictor.scala
    scaling/build.sbt
    scaling/src/main/scala/Predictor.scala
    scaling/project/plugins.sbt
    economics/build.sbt
    economics/src/main/scala/Economics.scala
```

Your executables should output their numerical answers and timing measurements in the JSON format, following the conventions for keys and data types provided in the template.

Add 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 `optimizing.json`, `scaling-100k.json`, `scaling-1m.json`, and `economics.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 datasets (`data/ml-100k`, `data/ml-100k.zip`, `data/ml-1m`, `data/ml-1m.zip`, and `data/ml-10M100K`, `data/ml-10m.zip`, if present), zip your archive (`CS449-YourID-M3.zip`), and submit to the TA. You archive should be around or less than 1MB.

7 Grading

We will use the following grading scheme:

	Points
Questions	35
Source Code Quality & Organisation	5
Total	40

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. We will enforce the minimal project structure of the previous section, as well as the JSON output format: deviations will cost points. If you really need to change any of those, please ask on the Moodle forum first.

7.1 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.

8 Updates

None yet!

References

[1] Google machine. `https://google-services.blogspot.com/2006/07/google-machine.html`, 2006. Accessed: 2021-01-21.

[2] De Decker, K. Low-tech magazine: Solar-powered website. `https://solar.lowtechmagazine.com/about.html`, 2019. Accessed: 2021-01-21.

[3] 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.

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

[5] Schelter, S., Boden, C., and Markl, V. Scalable similarity-based neighborhood methods with mapreduce. In *Proceedings of the Sixth ACM Conference on Recommender Systems* (New York, NY, USA, 2012), RecSys '12, Association for Computing Machinery, p. 163–170.

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 ($\{i|r_{u,i} \in \text{Train}\}$)