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: 2021/04/29 14:14:07 +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] Do not consider (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 (excluding self-similarity of a user with themselves), 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 $k = 10, 30, 50, 100, 200, 300, 400, 800, 942$. 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 $lowestk - baseline$). Do not include self-similarity in the k-nearest neighbours.

2. What is the number of bytes required to store the k nearest similarity values for all users, i.e. top k $s_{u,v}$ for every u? Assume an ideal implementation that stored only similarity 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 to plan hardware capacities). In your report, provide a formula as a function of the size of U, assuming all users have exactly k neighbours. In your code, compute the total number of bytes for each value of k for the actual number of neighbours (i.e. $\leq k$) of each user. Do not include self-similarity (the similarity of a user with themselves).

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 ~~ten~~ **25** minutes to produce all answers on your machine, we will kill scripts that take more than that amount of time when grading. The grading machine is a high-end machine so should take less time than you obtain on your machine. Scripts that take longer may receive 0 for unanswered questions.

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.
- Increased allotted time per submission to 25 minutes, instead of the original 10 minutes, following preliminary results shared by students.
- Added additional clarifications on k-NN: (1) not count self-similarity in question 1 and 2 (Section 3); (2) formula in report for question 2 (Section 3), should be an upper bound assuming all users have exactly k neighbours; (3) code in same question should compute for the actual number of neighbours of each user. We will accept answers that include self-similarity (ex: for $k = 943$) because submissions have already been received prior to that clarification. Prompted by a question from Saoud: `https://moodle.epfl.ch/mod/forum/discuss.php?d=59185#p120124`.

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