CS 333 Algorithms in the Real World: HW5

Due 3:30 pm Tuesday, October 26

Submission. You will submit materials for homework number n on Gradescope under two separate assignments: HWn Report and HWn Code. You should turn in a pdf containing all of your written solutions for the report, and a source code file containing all of the code you wrote for the assignment. When you submit your code, the interface may mention an autograder; you can ignore this, as the assignment will not be autograded. Your grade will be recorded on the report; the source code is to show your work and for reference during grading.

Report. The pdf you submit for the report should be typed, and solutions to individual questions should be clearly labeled. Show your work or explain your reasoning for your answers. You should use LaTeX (which is free) or some other editor of your choice (Microsoft Word, Google Docs, etc.) to prepare your reports. If you use an editor like Microsoft Word, make sure to convert the final document to a pdf, confirm that the symbolic math from the equation editor is properly formatted, and submit the pdf.

Collaboration. You may complete this assignment independently or in a group of two students. If you work with another student you should submit a single report and a single code file with both of your names, and should use the group feature when submitting on gradescope to indicate that you worked as a group. Do not split up the assignment, and each only complete half of the problems. Instead, complete each portion of the assignment by working together synchronously (for example, by pair programming with screen sharing over zoom or some other similar service) or by working independently and then coming together to merge solutions and check one another's work.

Allowed Materials. You can use any standard library functions and data structures in your programming language of choice (Java or Python 3 are recommended). You may also use any slides or notes from class or reference materials posted on the course website. You may search the internet for basic definitions, terminology, and language documentation (for example, checking the syntax for array slicing in python), but you may not use anyone else's code (either another student's or from the internet), nor may you search the internet for solutions or descriptions of solutions to the homework problems.

Problem 1 (Working with Sparse Recommendation Data, 12 points). Our data are contained in `ratings.csv`. Roughly 1,000 users have rated a total of roughly 1,700 movies. However, most users have only rated a small subset of the movies. Rather than representing the input as a 1,000 by 1,700 matrix of ratings, we just have a list of ratings of the form: `user_id, movie_id, rating, timestamp`.

We will only need the first three values for this assignment; you will not need to use the timestamps. The `user_id` values are just integers corresponding to the movie watchers doing the rating. The `movie_id` values are just integers corresponding to the movies being rated. The file `movies.txt` contains metadata about the movies (in particular, the title and release date), sorted by `movie_id`. The ratings are integers on a 1 to 5 scale where 5 is the best.

In this first problem, you will read the data in and answer some general questions about the dataset. We first provide a few sample answers that you can use to ensure you are reading the data properly.

- *Toy Story* (`movie_id`=1) has been rated by 330 users and has an average rating of about 3.89.
- *Lightning Jack* (`movie_id`=1,000) has been rated by 9 users and has an average rating of 3.
- *Santa with Muscles* (`movie_id`=1,500) has an average rating of 5, but has only been rated by 2 users.

- The user with `user_id`=1 has rated 193 movies and gave an average rating of about 3.64.
- The user with `user_id`=100 has rated 49 movies and gave an average rating of about 3.06.

Once you are confident you are reading the data correctly, answer the following questions.

(a) What is the highest rated movie that has been rated by at least 100 users? You can just give the `movie_id`, but you can also look up the title in the `movies.txt` file. What is its average rating?

(b) What is the lowest rated movie that has been rated by at least 100 users? You can just give the `movie_id`, but you can also look up the title in the `movies.txt` file. What is its average rating?

(c) What is the average number of movies that a user has rated? Call this value $\bar{m}$; we will reference it in subsequent problems.

Problem 2 (k-Nearest Neighbor Search, 24 points). When discussing recommender engines, we focused on collaborative filtering wherein we make recommendations for a user based on the ratings of other users with similar ratings to that user. We find those similar users through *k-nearest neighbor search*. In this problem, you will implement efficient algorithms for k-nearest neighbor search on our sparse ratings data.

(a) Define the angular distance between two d-dimensional vectors $\mathbf{x}$ and $\mathbf{y}$ as

$$1 - \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|_2 \|\mathbf{y}\|_2} = 1 - \frac{\sum_{i=1}^{d} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{d} x_i^2} \cdot \sqrt{\sum_{i=1}^{d} y_i^2}}.$$

Implement an efficient algorithm to compute content rating distance as follows. Given two users and a minimum commonality threshold τ, if they have rated at least τ movies in common then return the angular distance between the ratings vectors of the two users on the movies they have rated in common,[1] otherwise return 1 (the maximum possible angular distance). Use $\tau = 3$ for this assignment.[2]

The angular distance between two users is just the angular distance between the ratings vectors of movies they have both rated. For example, suppose user 1 has rated movies 10, 12, and 14, and user 2 has rated movies 10, 12, 13, and 14. Only movies 10, 12, and 14 are in common, so we would take the ratings for user 1 for those movies (say `[4, 3, 5]`) and the ratings for user 2 for those movies (say `[3, 2, 5]`) and compute the angular distance according to the above formula for these vectors. In this example we would get

$$1 - \frac{(4)(3) + (3)(2) + (5)(5)}{\sqrt{4^2 + 3^2 + 5^2} \cdot \sqrt{3^2 + 2^2 + 5^2}} = 1 - \frac{43}{\sqrt{50} \cdot \sqrt{38}} \approx 0.0135.$$

Your algorithm should be efficient in that the runtime should be $O(m_1 + m_2)$ where m_1 is the number of movies that have been rated by user 1 and m_2 is the number of movies that have been rated by user 2. In other words, the average (across all users) runtime should be $O(\bar{m})$. Explain how you achieve this (note, it may require some preprocessing).

For testing purposes, here are some examples:

- The content rating distance between user 1 and user 100 is about 0.057
- The content rating distance between user 200 and user 300 should be 1.0 (not enough movies rated in common)
- The content rating distance between user 200 and user 500 is about 0.072

The only thing to report for this part is your explanation of your efficient algorithm, but make sure to test that it is working.

[1] τ is a hyperparameter to the model. This is only necessary because we have incomplete information; most users have not rated most movies and we consider correlations between users who have rated fewer than τ movies in common to be spurious.

[2] Nothing magical about $\tau = 3$. Feel free to play around with the value as you experiment, but report your answers for $\tau = 3$.

(b) Implement an efficient k-nearest neighbor search algorithm. Given a user u, a set of other users S, a minimum commonality threshold τ, and a parameter k, it should return the k other users from S with minimum content rating distance to u with minimum commonality threshold τ. Make sure *not* to include u itself among the nearest neighbors, if $u \in S$. The algorithm should be efficient in that the runtime should be no more than $O(|S|\bar{m} + |S|\log(|S|))$ on average, which should follow from using your algorithm from part a and then sorting. If $k > |S|$, simply return all of the users in S.

For testing purposes, here are some examples:

- The 5-nearest neighbors of user 1 from among users 1 through 100 are users 39, 51, 66, 80, and 91 with content rating distances from user 1 of roughly 0.0, 0.0044, 0.0121, 0.0168, and 0.0169 respectively.
- The 5-nearest neighbors of user 2 from among users 100 through 110 are users 106, 108, 103, 102, and 101 with content rating distances from user 2 of roughly 0.0054, 0.0144, 0.0151, 0.0236, and 0.0370 respectively.

Nothing to report for this part, but make sure to test that your algorithm is working and working efficiently.

Problem 3 (Recommender, 32 points). In this problem, we will use our k-nearest neighbor search algorithm from the previous problem to write a simple recommender system.

(a) Implement an efficient recommend algorithm as follows. Given a user and a number r of recommendations desired, this will efficiently return an ordered list of the top r movies recommended for that user.

(i) First, run k-nearest neighbor search using your work from problem 2 part b on *all* of the users. Let N be the set of users returned. Use a minimum commonality threshold of $\tau = 3$ as before.

(ii) Efficiently find M, the set of movies rated by at least one user in N but *not* by the user for whom we are making predictions (that would be a little redundant). By efficient, this step should *not* iterate over all content items and all of N; it should only consider the content items actually rated by users in N.

(iii) Let $\rho = 3.5$ be our default rating (the average rating of movies in the dataset is about 3.5). For each movie $j \in M$, let N_j be the users in N who have rated j and let $\bar{y_j}$ be the average rating users N_j give to movie j. Our smoothed prediction for movie j will be

$$\hat{y_j} = \frac{\rho + |N_j|\bar{y_j}}{1 + |N_j|}.$$

Compute the smoothed prediction for each movie in M.

(iv) Sort M by the smoothed prediction values computed in the previous step and return the top r along with their smoothed prediction values. Congratulations, you have an efficient recommendation engine!

For example, if we use $k = 30$ and $r = 5$, the top 5 recommended movies for user 1 are:

1. Cool Hand Luke (1967) - Predicted Score: 4.625
2. Star Wars (1977) - Predicted Score: 4.55
3. Big Lebowski, The (1998) - Predicted Score: 4.5
4. Michael Collins (1996) - Predicted Score: 4.5
5. Chinatown (1974) - Predicted Score: 4.5

Nothing to report for this question, just your implementation.

(b) Using $k = 30$ and $r = 5$, provide the top 5 recommended movies for user 2 along with the smoothed predictions for those movies. As a reminder you can find the titles of a particular movie id in the `movies.txt` file.

(c) Next, add ratings for about 10 movies that *you* have seen based on how much you like them. Note that the `movies.txt` file gives the titles of all of the movies by their integer id. You will also need to give yourself a new user id, you can use 0 or a large integer like 5,000 as you prefer. Again using $k = 30$ and $r = 5$, provide the top 5 recommended movies for yourself along with the smoothed predictions for those movies. What do you think of the recommendations? Do they make sense given your ratings? Do you think you might like the movies? Hopefully this will be a fun question!

(d) For the parameter settings and data used above, which of steps (i) through (iv) in the recommendation algorithm outlined in part (a) would you expect to dominate the running time? Briefly explain your answer referencing the algorithm. You are welcome to check empirically, but you must also provide an explanation referencing the algorithm.

(e) Our recommendations were based off of the *smoothed* predictions $\hat{y_j}$ calculated in 3.a.iii above. Why do you think we used these smoothed predictions instead of just $\bar{y_j}$, the average score given to the movie by the nearest neighbors?

Problem 4 (Evaluation, 18 points). In this problem, we will evaluate our collaborative filtering model against a simple baseline. In particular, for a number of user-item pairs, we will measure the root-mean-square error (RMSE) between the true ratings y and our model's predicted ratings $\hat{y}$. RMSE for length d vectors is

$$RMSE(y, \hat{y}) = \sqrt{\sum_{j=1}^{d} \frac{(y_j - \hat{y_j})^2}{d}}.$$

(a) Use the collaborative filtering model described in the previous problem to predict scores $\hat{y}$ for the first 1,000 user-item pairs on the first 1,000 lines of `ratings.csv`[3]. That is, follow the directions from Problem 3, part a, step i to get the nearest neighbors of a given user and then use step iii to calculate the smoothed prediction for the given content item (note that if none of the nearest neighbors have rated a content item then the smoothed prediction is just the default rating ρ). Use the same hyperparameters of $k = 30$, $\tau = 3$, $\rho = 3.5$.

Time how long it takes your model to make these 1,000 predictions and report the runtime in seconds[4]. Calculate and report the RMSE of these predictions. Also calculate and report the RMSE of the simple baseline that simply predicts the default value $\rho = 3.5$ for each user-item pair.

(b) Repeat what you did in part (a) but for the new user-item pairs located in `test_ratings.csv` (same format as `ratings.csv`). That is, your model should only use information in `ratings.csv` to make predictions about the new unseen user-item pairs in `test_ratings.csv`. Report the RMSE of your model predictions as well as the RMSE of the simple baseline that predicts $\rho = 3.5$ for every pair.

(c) Based on your reported errors, do you think the hyperparameter k (the number of nearest neighbors used in the collaborative filtering) should be decreased or increased to improve the model's performance, especially on the unseen test data? Explain your reasoning referring to the algorithm (you are welcome to run experiments to check, but you must also explain your results referring to the algorithm).

Problem * (Optional Extension, 0 points). This section is not required and is not worth any points, but provides some ideas if you would like to explore the subject of this assignment more deeply. There are several directions to explore in improving your collaborative filtering model:

- With respect to running time, you can improve the query time by preprocessing the data with a clustering. For example, in class we discussed using k-means clustering where the distance is defined

[3]Note that because these user-item pairs are in our training dataset, we know their true values; Nevertheless you can use the model to predict scores to calculate how well the model captures the training data itself. We only use the first 1,000 lines so the code runs reasonably quickly. If you wish you can leave your self-reported ratings from problem 3 part c in the collaborative filtering model; it should not have a major impact on your aggregate results

[4]An efficient implementation should take at most few seconds to compute all 1,000 on a low-end commodity CPU core, certainly not more than 30 or so seconds.

by the content rating distance defined in problem 2 part a. You will need to randomly initialize the cluster centers, for example, by choosing a random rating from 1 to 5 for every possible content item (or another similar scheme, as the standard initialization of choosing a random data point will not work well because of the sparsity of user rating data).

Once you have clustered the users, to predict or make recommendations for a given user, determine which cluster center they are closest to in distance and then only conduct k-nearest neighbor search within that cluster. Implement such a preprocessing and then study how the running time of prediction / recommendation queries varies with the number of clusters you use, as well as how the runtime performance of adding more clusters trades off with the RMSE of prediction accuracy.

- With respect to predictive accuracy, there are several relatively simple modifications you can explore for the collaborative filtering model. For example:
  - Rather than a constant default rating $\rho = 3.5$ in the model, you can calculate a baseline x_i for each user i (their average rating across all content items they have rated) and a baseline z_j for each content item j (the average rating of the item by all users that have rated it). Then you can use a default rating custom to each user-item pair, $\rho(i, j) = \frac{x_i + z_j}{2}$.
  - Rather than weighting each of the k-Nearest Neighbors equally (when computing the average score in problem 3 part a item iii), it may be advantageous to use a weighted average that places higher weight on nearer neighbors (perhaps proportional to the distances, though in general you could treat the weights just as additional hyper-parameters to be tuned empirically).
- A more fundamental extension to improve performance is to use a matrix factorization model that embeds content items and users into a low dimensional latent space. Can you build a matrix factorization model (or a combined model) that outperforms the collaborative filtering model alone? This will require you to perform gradient descent optimization, for which you might consider a standard library implementation like Pytorch's autograd for calculating gradients. This extension is nontrivial.