CS504 Fall 2023 -- Final Project Option 2 – Coding KNN and CF (v4)

**Due Date - December 8th, 2023 11:59 PM**

In this project, you will explore one of the MovieLens datasets and develop software to analyze and make predictions. This project is to be programmed in Python. However, if you feel more comfortable using MATLAB or something else, **you MUST RECEIVE permission from me first.**

**Summary**

The goal of this work will be to utilize two techniques: k-nearest neighbors (KNN) and collaborating filtering (CF) to build a simple recommender system for movies using dataset from the MovieLens website (original dataset name is ml-latest-small at <http://grouplens.org/datasets/movielens/>) that contains 100,000 ratings for ≈ 9,000 movies. You can also google other sources.

You still need to take some caution. There will be a few examples in the CF portion where the user does not have any other movies that also have a similarity score. In this case, you should predict a score of “0.0”.

**Methods**

KNN – K Nearest Neighbors

Use k-nearest-neighbors (KNN) to predict movie ratings for users. For a given user *u*, find its *k* nearest neighbors (that have also rated movie *m*) using the Euclidian distance metric. You will evaluate TWO methods of calculating a ratings prediction:

Unweighted Prediction Method

Each predicted rating is the average of its nearest neighbor ratings:

**NOTE**: For some test cases there may be a tie for the kth with respect to distance. E.g., the kth neighbor may have distance 12.3, and the k+1 neighbor may also have distance 12.3 (and the k+2, etc). Each of these users may have different ratings. In these cases, you MUST EXPAND k such that it includes all of the ties.

Weighted Prediction Method

Each predicted rating is the **weighted** average (Di) of its nearest neighbor ratings:

|  |  |
| --- | --- |
| if i = 1 |  |
| otherwise |  |

The notation used in the formulas is referenced below:

* *Pu,m* is the predicted value for user *u*  on movie *m*
* *r*i,m = rating for user *i* and movie *m*
* *x*' is the vector of ratings for user *u* (the user we are currently predicting)
* d(x,y) is the Euclidian distance between two vectors x and y
* *xiNN* is the ratings vector for the ith nearest neighbor

**NOTE**: Since this method assigns 0 weight to the kth neighbor, the problem listed above with ties does not impact this method. This method also assumes that the neighbors are ordered from closest (neighbor 1) to furthest (neighbor k).

**NOTE**: It is possible that for some test cases you will not have *k* examples. In this situation (and only these situations), you reduce *k* to the actual number of examples (taking care to divide by this new value of *k*).

**Output**:

For each prediction method, use the ratings\_training\_90.csv file for your example data and make predictions for ALL entries in the ratings\_test\_10.csv file for 3 different values of *k* (3,5,10). Thus, you will generate 6 prediction files that must be named as outlined in the deliverables section 3 (a-f). The "uw" in these file names is used for the unweighted prediction method and the "wt" for the weighted prediction method. The files **must** be formatted as follows (with each field (except the last) separated by a comma): userID, movieID, realRating, predictedRating (ALL PREDICTIONS ROUNDED TO ONE DECIMAL POINT). The order within each file must match the ratings\_test\_10.csv file (so your results can easily be compared).

Collaborative Filtering (CF)

You now have the choice of using the original CF method described in the project (now referred to as the Sarwar/Karypis method), or the Ullman method (the one presented in the slides). I will tell you that the Ullman method does ALLOW you to use the Pearson’s correlation coefficient routine for calculating similarity and it is FAST. I have updated the slides and included a supplement explaining step by step how these two methods work. This change required me to change the training/test sets. In your report, you MUST state which method you developed.

Calculate predictions using each pair of training/test sets. Name the result files:

|  |  |  |
| --- | --- | --- |
| **Training filename** | **Test Filename** | **Predictions file (you create)** |
| ratings\_training\_60.csv | ratings\_test\_40.csv | ratings\_predictions\_40\_cf.csv |
| ratings\_training\_70.csv | ratings\_test\_30.csv | ratings\_predictions\_30\_cf.csv |
| ratings\_training\_80.csv | ratings\_test\_20.csv | ratings\_predictions\_20\_cf.csv |
| ratings\_training\_90.csv | ratings\_test\_10.csv | ratings\_predictions\_10\_cf.csv |

Below is the outline from the original/Sarwar/Karypis method:

**Sarwar/Karypis method**

Use the collaborative filtering (CF) item-item recommendation system discussed in class to predict movie ratings. The similarity between two items/movies (*i,*j) is defined as follows:

where:

* -1.0 ≤ *sim(i,j)* ≤ 1.0
* *i* refers to movie *i*  and *j* refers to movie *j*
* ru,i refers to the rating for user *u* on movie *i*
* *U* is set of users who have rated both movie *i* and movie *j*
* is the average rating assigned by user *u* (do not include movies that user *u* has not rated)

**NOTE**: The full similarity matrix *S* has dimensions *M*x*M* (where *M* is the total number of movies for which we have ratings, thus, *M* ≈ 9,216). Computing this matrix can take a significant amount of time (on my PC, this takes about 5 hours). An alternate approach to computing this matrix all at once is to compute it on demand. That is, allocate the matrix and populate it with some special value (maybe -2). Then, when you need a value from the matrix, call a function to retrieve it. If the function finds a -2 in that position, the function calculates the similarity and stores it in the matrix for future use. However, if the function finds some other value, it uses the other value (since it must have been computed by some earlier call). Given the sparsity of the matrix, this provides a very significant runtime improvement (the entire run takes less than 15 minutes on my PC).

To calculate a prediction, use the weighted sum technique discussed in class:

where:

* r\*u,i = prediction for user *u* for movie *i*
* sim(i,j) is the similarity between movie *i* and movie *j*
* *N* is the set of movies that are similar to movie *i* (that is, given a movie *i*, find the set of movies *N* such that for each j ∈ N, sim(i,j) > 0)

**Ullman method**

This section describes the method and highlights the differences between it and the Sarwar/Karypis method.

The similarity between two items/movies (*i,*j) is defined as follows:

Notice that this similarity uses the average of the MOVIE (and not the average of a USER). See notes below on calculating the average and how to zero center the ratings/utility matrix.

Since this similarity includes all users (unlike the Sarwar/Karypis method that only includes users that co-rated movies), the **corrcoef** function within numpy CAN BE USED to calculate the similarity matrix in a single call. **PLEASE NOTE**  this requires the ratings matrix to be in MxU (movies by users) form and to be zero-centered when calling the function. It also introduces the fact the “NAN” (not a number) value being assigned. This can happen in a number of scenarios, one being when a movie only has one rating. In this case, when you zero centered the data, this value turned to zero, and this in terms leads to one of the terms in the denominator being zero, and thus, the entire denominator is zero. You can either change the similarity score to zero (since we avoid these values) or check for it using numpy’s isnan function.

**Notes on Centering Data**

The zero centering of the ratings/utility matrix has many benefits (as we discussed in class). One of these benefits is that missing scores (0s) are now the user’s average rating (where before this, a zero showed a strong distaste for the movie). In order to “preserve” these missing values as zero, you only calculate the average from the available values.

Lets look at for both the Sarwar/Karypis and Ullman methods and what it means.

Sarwar/Karypis

This method uses the user’s average rating. So, lets calculate the average for user X. We have a total of 10,000 movies, but user X has only rated 50 of them. To calculate the average for user X, we add up the ratings for these 50 movies and divide by 50. We can then use this average to “zero center” the ratings for user X. To do this, we subtract user X’s average from each of his ratings. **WARNING**: Do not subtract user X’s average from movies that he/she has not rated (in this case, the other 9,950 entries for user X). If you want to double check that you have done this correctly, after zero centering the data, you can add up all the ratings for user X. If they do not total zero, you have a problem.

Ullman:

This method requires the average rating for each movie. When calculating the average for a movie, you only include user’s who have rated the movie. So, if we have 671 users, and 40 of them have rated movie Y, then we add up those 40 ratings (avoiding all the other entries) and then divide by 40. When then subtract the average from those 40 ratings only. **WARNING**: Do not subtract movie Y’s average from user’s that did not rate movie Y. If you want to double check that you have done this correctly, after zero centering the data, you can all up all the ratings for movie Y (or any movie). If they do not total zero, you have a problem.

**Honor Code**

You are to work on this project **alone and not in groups**. All projects will be checked for plagiarism against each other and against the Internet using software. Suspicious projects will be sent to the honor committee for further review.

**Deliverables:**

Submit to BB in a tarball (or zip file) named lastName\_cs504\_final\_project (for example, my file would be named gang\_cs504\_final\_project.zip). The following files must be included:

1. For each method (KNN and CF) submit all source code. Organize the source code files used for each step in its own directory (cs504final/knn, cs504final/cf).
2. A professionally organized single PDF document. **Any document not in PDF format will automatically lose 10% of the potential grade**. This document should include:
   1. Brief introduction to the problem being address (predicting movie ratings). You can discuss a little bit on recommendation systems in general.
   2. A METHODS section that outlines EACH technique (KNN, CF) and any observations. Each method should be discussed in a separate paragraph.
   3. A RESULTS section, with a subsection/paragraph for KNN and CF. You must show the run times (wall clock is OK as long as nothing else major is running on your PC) and accuracy of each method and compare how changing the values in your model (*k*  and the prediction formulas for KNN and the training/testing set sizes for CF) impacted your results (in terms of run time and in terms of accuracy). Again, I expect explanations in words accompanied by PLOTS of your results (you need **BOTH plots and words**). For each training test set, show a plot and a table of the RMSE.

RMSE =

Where *N* is the number of predictions, *R*i is the real answer for the ith prediction and Pi is the predicted value. That is, for KNN, the plot needs to show the RMSE for each value of *k* and for each prediction formula (6 values). For CF, show the RMSE for each training/test set (4 values). Comment on which model performed the best for KNN, for CF, and overall. Make a hypothesis as to why the model had the best performance in terms of performance (run time) and accuracy (lowest RMSE).

* 1. A small conclusion paragraph with any final thoughts on your findings.

1. Include the 10 output prediction files from your program. These files should be placed in a subdirectory called prediction\_results and named:
   1. ratings\_predictions\_10\_knn\_uw\_k3.csv
   2. ratings\_predictions\_10\_knn\_uw\_k5.csv
   3. ratings\_predictions\_10\_knn\_uw\_10.csv
   4. ratings\_predictions\_10\_knn\_wt\_k3.csv
   5. ratings\_predictions\_10\_knn\_wt\_k5.csv
   6. ratings\_predictions\_10\_knn\_wt\_10.csv
   7. ratings\_predictions\_10\_cf.csv
   8. ratings\_predictions\_20\_cf.csv
   9. ratings\_predictions\_30\_cf.csv
   10. ratings\_predictions\_40\_cf.csv

**Sample Output Data to Verify Your Program Results**

I will post 5 files to BB in the coming weeks:

* ratings\_predictions\_10\_knn\_uw\_k3\_sample.csv
* ratings\_predictions\_10\_knn\_wt\_k3\_sample.csv
* ratings\_predictions\_10\_knn\_uw\_k10\_sample.csv
* ratings\_predictions\_10\_knn\_wt\_k10\_sample.csv
* ratings\_predictions\_20\_cf\_ sarwar\_sample.csv
* ratings\_predictions\_20\_cf\_ullman\_sample.csv

These files each contain the output for the first 25 test cases for the KNN unweighted and weighted with k=3 and k=10, and the CF case using the 80% / 20% training/test split. These files **ALLOW YOU TO VERIFY YOUR PROGRAM's OUTPUT** on a subset of the data. If your answers do not match this limited set, **you should be concerned**.