Wisenut Programming Test
Wisenut, Inc. Saturday April 23, 2011
9: 30am -- 9: 30pm

Overview

There are **two problems** in this test. In each problem, please provide source code files and a makefile. All programs should compile in Linux at the assigned test servers. You must use **C++ or Java programming language** to solve the problems.

Evaluation

Each problem in this test has a maximum score you can get. Here is how we grade your solution. First, if your program compiles correctly by simply typing "make" at a prompt in any directory after extracting your solution files, and produces expected output without a bug, you get the maximum score on correctness. **If your Makefile does not work, you get 0 point on that problem. Please make sure that your Makefile works without a fail! If you use Java programming language, please ensure that your Makefile will work properly by specifying all the relevant paths.**

Secondly, after your program passes the correctness, we will measure the time it takes to execute on a random, fairly large input. Based on the relative speed of your program among the programs that passed the correctness test, you will receive a score from 0 to a pre-determined maximum score. **If your program fails on the correctness test, you automatically get 0 point on the speed test.**

Submission

Please use Linux tar command to archive your source codes and Makefile. No object files or executables should be included in the archive. An archive name should be **firstname.lastname.tar**, for example, **yeogirl.yun.tar**.
Please ensure you follow **the program usage** in each problem.
Please submit your solution (via email to **"yeogirl@gmail.com"**) before 9:30pm. The email **subject title** should be **Wisenut Programming Test solution.** No solution will be accepted after 9:30pm. Please note that you can submit your solution only once. Any subsequent submissions will be ignored.

Coding Policy

You are free to use standard C++ library for this test. **It is okay to use your code that you wrote beforehand for work or for preparing for this test** as

long as you submit all the code that compiles well using a simple makefile with standard C++ or Java libraries. If you were to use Boost library, please ensure that you include its source and compiles using a simple makefile command. If your program does not compile due to a third-party library that you used, your program will get no points. If you use a third-party library, you have to ensure that you include its source code such that your makefile can build a binary successfully. **It is also okay to use any code that you find in Internet during this test to expedite your coding**, as long as you provide all source code that compiles when I evaluate your solution. **However, it is prohibited that you copy or take code from other engineers taking this test. It will be a violation of Wisenut integrity principle and will result in a grave disciplinary action.**

Problem A (100 points)

In this problem, you will review the paper, ***Item-Based Collaborative Filtering Recommendation Algorithms***, and implement its main algorithm. The paper describes an efficient item-based collaborative filtering algorithm. There are three components in implementing this algorithm. The first is the similarity measure between items. As in section 3.1.3, you will use Adjusted Cosign Similarity in computing item similarities. The second is the prediction algorithm. As in section 3.2.1, you will use Weighted Sum method to compute prediction $P_{u,i.}$ The last component is the model size (item neighborhood size) for an efficient performance of the algorithm. As in section 3.3 and 4.3.3, you will use 30 as an optimal choice of neighborhood size. Please note that we will only keep positive similarities in the neighborhood. This means you will have to terminate the neighbor search if you cannot find positive similar terms any more even if the neighbors are less than 30. *By taking only positive value neighbor items, you can safely assume that both nominator and denominators in the prediction formula of section 3.2.1 will always be positive.*

Due to the possible rounding errors, it is possible that actual prediction value can be outside the item ratings range (in our problem it is [1..5]). You can simply adjust the prediction values in the predefined rating range by taking anything above 5 to 5 and anything less than 1 to 1. The standard solution employs this adjustment and shows the actual prediction values in the provided sample files to help you with debugging.

In implementing prediction algorithm of this paper, you will find that some cases do not have items of a same user rating, so that you can't produce a prediction rating. In this case, you will use the Equation (12) in Problem B with the parameter λ value of 0.7.

Your program will be given user-rating file as an input and must produce a top-N recommendation list for all of the users as in Figure 1 of the paper.

Please note that every user has rated some movies. You will have to recommend new movies that are not rated by each user using the algorithm described in this paper. The output file format will be explained in the input and output file format section.

A sample input and output file data will be provided to help you with debugging your program.

Accuracy Test (50 points)

In the accuracy test, your program will get a maximum of 50 points according to your top-N recommendation list match. In this test, we will select 10 as top-N parameter value as follows:

$$\text{Accuracy Score} = 50 * \{\Sigma(\text{total}) - \Sigma(\text{error})\} / \Sigma(\text{total})$$

For example, given a total of 100 users in an input data, if your corresponding top-N recommendation list contains 220 errors as compared to the standard solution, you will be getting (1000 – 220)/1000 = 78% of the maximum score (39 points). Normally the order of the movie recommendation is important, but to make this programming test manageable within a day timeframe and allow for a small margin of errors, we will ignore the order of movie recommendations. As long as your solution produces movies that are recommended by the standard solution, you will get a score count for each correct movie recommendation.

A simple C++ program t_grade.cc is provided to check your accuracy score. You can simply make and build t_grade executable. It takes two input files; one is your output and the other is the standard output. Given two output files, it will show accuracy percentage of your output vs. standard output.

You must pass this test with 70% accuracy (35 points) in order to be eligible for the following Performance Test. If your score is less than 35 points, you will get 0 scores in the Performance Test.

Performance Test (50 points)

In the performance test, your program's execution time will be measured and be ranked among those programs whose accuracy is more than 70%. Your performance score is rated as follows:

$$\text{Performance Score} = 50 - 2*(\text{your program's speed rank} - 1)$$

Problem A Usage: Problem_A <input_file> <output_file>

Input File Format

Each line of the input file will have the following format. You will have to read until the end of the file to determine how many lines there are in the input file:

<user_id> <tab> <movie_id> <tab> <rating>

eg.

721	262	3
913	690	3
660	229	2

In this example input file, users 721, 913, 660 rated movies 262, 690, 229 in the ratings of 3, 3, and 2.

NOTE
All the ids are equal to or greater than 1.
<rating> has a value from 1 to 5.

It is possible and normal that a same user id will appear multiple times since the same user will rate many movies in a normal situation. However there are no duplicate ratings of the same user for the same item.

Output File Format

Each line of the output file will have the following format. The grading program will read until the end of the file to determine how many lines there are in the output file:
<user_id> <tab> <movie_id_1> <tab> … <tab> <movie_id_10> <newline>

eg.

123	1	3	32	877	887	987	990	1002	1230	1244
324	1	3	32	877	887	987	990	1002	1230	1244
234	1	3	32	877	887	987	990	1002	1230	1244

In this example output file, very unlikely, users 123, 324, and 234 have the same recommended movie ids.

NOTE
Because your program gets score when there are matches for recommended movie ids, if you make a mistake in manipulating IDs (such as starting from 0 or somehow misread or miswrite the IDs), you will probably get a very poor score. Please ensure that your ID manipulations are all correct.

You can print the user id and its recommended movies in any order as shown in the above sample output. The grading program will simply count the number of matched movie recommendations for a given user. Also note that if you somehow miss some users, you will lose all of the users' recommendation counts.

Problem B (100 points)

In this problem you will review the paper, ***Effective Missing Data Prediction for Collaborative Filtering,*** and implement its main algorithm. Unlike Problem A, this paper utilizes both user and item similarity information. It also utilizes a simple parameter, as in Section 4.2, in the similarity computation that improves the MAE (mean absolute error) quite a bit. The algorithm in this paper is about 20% more accurate in the Movie Lenz data set in the standard solution.

Just like Problem A, you will go through three steps in implementing this paper. The first step is to compute similarity matrices for both users and items as in Section 3. In this step, the paper utilizes Significance Weighting parameters γ and δ. We will use the values 30 and 25 respectively as suggested by the paper. *Please note that there is a typo in the paper. The values of Sim(a, u) and Sim(i, j) can actually be negative and is in the range of [-1..1].* The second step is to find neighbor users and items for each user and item as in Section 4.1. Unlike Problem A, this paper utilizes parameters η and θ to dynamically determine the neighbor users and items. *We will use the values 0.3 for both parameters to make it work with a smaller and sparser input data.* The last step is to do actual prediction as in Section 4.3. This paper utilizes a parameter λ to combine both user and item information in predicting a rating for an active user. We will use the parameter value of 0.7 as suggested by the paper.

Sample input and output files will be provided to help you with debugging your program.

Accuracy Test Part 1 (25 points + 25 points)

In the accuracy test, your program will get a maximum of 50 points according to the number of similar items and users that are matched by the standard solution's output. Both items neighbor test and users neighbor test have 25 scores. Unlike Problem A, this problem utilizes dynamic user and item neighborhood. If your program's output is exactly same as the standard output, you will get 25 scores, where each output file for user neighbor and item neighbor will score 25. If there are any differences, you will be deducted scores accordingly. The formula for the score is the same as in Problem A and the same t_grade program can be used for this test as well.

$$\text{Score for User Neighbor} = 25 * \{\Sigma(\text{total}) - \Sigma(\text{error})\} / \Sigma(\text{total})$$

$$\text{Score for Item Neighbor} = 25 * \{\Sigma(\text{total}) - \Sigma(\text{error})\} / \Sigma(\text{total})$$

Normally the order of the neighbor items/users is important but as in Problem A, we will ignore the order of neighbor items/users.

You must pass this test with 70% accuracy (35 points) in order to be eligible for the Accuracy Test Part 2 and Performance Test. If your score is less than 35 points, you will get 0 scores in the remainders of the tests.

Accuracy Test Part 2 (30 points)

In the accuracy test, your program will get a maximum of 30 points according to the top N recommendation output for each user as in Problem A. The same t_grade program can be used for this test.

$$\text{Accuracy Score} = 30 * \{\Sigma(\text{total}) - \Sigma(\text{error})\} / \Sigma(\text{total})$$

Performance Test (20 points)

In the performance test, your program's execution time will be measured and be ranked among those programs whose accuracy is more than 70% in the Accuracy Test Part 1. Your performance score is rated as follows:

$$\text{Performance Score} = 20 - 2 * (\text{your program's speed rank} - 1)$$

Input File Format

The input file format is the same as in Problem A

Output File Format

There are two output file formats for Problem B. The first output file format is the neighbor users and items. Both user neighbor output and item neighbor output follows the following format:

<user_or_item_id> <tab> <neighbor_1> <tab>…<tab> <neighbor_K>

Where K is determined dynamically as explained above.

The second output file format is the same as that of Problem A.

Problem B Usage: Problem_B <input_file> <user_neighbor_file>

<item_neighbor_file> <top_n_recommendation_file>