CMP2003 Term Project

Polat Öztürk(2103733)-Salih Sercan Eser(2102897)

Deadline: 6th January 2023

1 Introduction

In this project we have developed a program which predicts users' movie ratings on certain films by using 35.200 users' movie ratings on 7331 different movies. We have also developed a program which shows top 10 users who have watched the most films and top 10 movies which have been watched by the most users. This program has 3 parts. These are:

1. Taking data from CSV file and pre-processing
2. Ranking top 10 users and top 10 movies
3. Prediction of the movie ratings

2 Taking Data From CSV File and Pre-Processing Algorithm

To create the data structures to do operations on, a function named createAllList is used.

2.1 createAllList

The contents of .csv file is read and processed with fstream and stored in a string vector vector (vector⟨vector⟨string⟩⟩) named "content", in the order they are being iterated over. First, the code goes to a line, and then seperates the words using delimiter ",". Then, these words are stored in a string vector (vector⟨string⟩) called "row". Finally, every row is stored in "content".

After reading all of the contents of train.csv and storing it in "content", a new, pair vector vector (vector⟨vector⟨pair⟨float,float⟩⟩⟩) named "all" is created and initiated with empty vectors.

Then "content" is iterated over subvector-wise, (every subvector's first element is UserID, second element is ItemID, and third element is Rating).

There are two modes of createAllList, mode = 0, mode = 1.

If mode is set as 0, Every iteration, a pair consisting of (ItemID, Rating) is pushed back inside the vector at the "UserID"th index of "all". After all pair-pushback iterations end, a sorting loop over "all" is called to sort all of the sub-vectors with regard to ItemID's (this will prove to be useful in in finding Rating pairs). With the creation of "all" near-instant access to all users' every watched film and their corresponding Ratings is achieved.

If mode is set as 1, Every iteration, a pair consisting of (UserID, Rating) is pushed back inside the vector at the "ItemID"th index of "all". After all pair-pushback iterations end, a sorting loop over "all" is called to sort all of the sub-vectors with regard to UserID's (this will prove to be useful in finding Rating pairs). With the creation of "all" near-instant access to all items' every user (every user who have watched said item) and their corresponding Ratings is achieved.

With createAllList three lists are created:

- user_based_all_list_train (mode = 0)
- item_based_all_list_train (mode = 1)
- user_based_all_list_test (mode = 0)
  (structure of user_based_all_list_test: user_based_all_list_test[ID][0] = pair(UserID,ItemID).

Following the creation of these structures, two integer vectors (vector⟨int⟩) are created:

- all_users_item_count (stores all users' watch counts)
- all_items_user_count (stores all films' user counts)

2.2 Creation of common items pair array

Buckle up, we are in for a journey.

All variables and structures:

- user_based_all_list_train
  - A list created from train.csv with indices corresponding to User IDs.
- user_based_all_list_test
  - A list created from test.csv with indices corresponding to IDs (the order, from top to down).
- item_based_all_list_train
  - A list created from train.csv with indices corresponding to Item IDs.
- predictionIndex
  - predictionIndex is equal to "ID" in the test file. It is the "row number" to find which userID we are working with.
- predicted_user
  - UserID is being assigned to predicted_user.
- predicted_item
  - ItemID is being assigned to predicted_item.
- rating_pair_vector
  - Pair-vector-vector which stores pairs of ratings. For a particular film which has been watched by predicted_user, left side of the rating pair is predicted_user's rating and right side is the rating of another user who have watched predicted_item.

In a for loop predictionIndex is iterated from 0 to the size of "user_based_all_list_test". Thus, for every iteration of this loop, algorithm is creating necessary structures for rating prediction for a single user and single item.

For every iteration of this outer loop, variables; "predicted_user", "predicted_item" are re-assigned.

In another for loop which goes from 0 to size of the vector which stores all of the users who have watched the "predicted_item" ("item_based_all_list_train[predicted_item].size()"). Inside this loop, algorithm is iterating over all of the users which have watched the item (predicted_item) we are trying to predict the rating for. Iteration variable in the code is named h.

For every iteration a temporary pair vector, "temp" is created and pushed back in "rating_pair_vector" (rating_pair_vector[h].push_back(temp)). Then first element of temp is assigned as (-1, user h's rating

for the predicted_item) to feed the UBCF code with the user h's rating to the "predicted_item".

Then, a comparison algorithm is implemented. This algorithm iterates over predicted_user's items and user h's items, with a while loop. Since we have created the lists in an ordered way we can iterate over both lists with i and j variables to find the intersection of two lists in O(i+j) time.

- If the ith ItemID in predicted_user's watch list is less than jth ItemID in user h's watch films list we can increase i by one.
- If the ith ItemID in predicted_user's watch list is greater than jth ItemID in user h's watched films list we can increase j by one.
- If the ith ItemID in predicted_user's watch list is equal to jth ItemID in user h's watched films list we can add the ratings they have given to "ItemID" of both users in a pair to temporary pair vector "temp".

After the completion of all these steps we have supplied the necessary rating pairs of all of the users for the implementation of UBCF.

3 Ranking Top 10 Users and Top 10 Movies Algorithm

3.1 Top 10 Users

The algorithm iterates over the list "all_user_based_list" and finds the length of every subvector and stores them in an "integer vector" named "all_users_item_count". At every index of "all_users_item_count", indices corresponding to an userID (Index 1 for userID = 1), the user's total watch count is stored.

"all_users_item_count" is then traversed sequentially to find the 10 users who have watched the most films.

The algorithm creates a pair array called "max_10_users". Initially, first 10 userIDs and values of "all_users_item_count" is assigned to first 10 values of "max_10_users" as (userID, watchCount). Then the pair array is sorted with regard to users' watch counts in ascending order with insertion sort. After that, the algorithm iterates over every element of "all_users_item_count" and in every iteration "all_users_item_count[i]" is compared with "max_10_users[0].second" (smallest element with regard to watch count).

- If "all_users_item_count[i]" is smaller than or equal to "max_ 10_users[0].second" nothing happens (Code continues).
- If "all_users_item_count[i]" is larger than "max_10_users[0].second" then "max_10_users[0]" is assigned as the pair: (i, all_users_item_count[i]). Then the array is again sorted to make sure the smallest element is always contained in the first index, index 0.

After iterating over the whole "all_users_item_count" vector, the top 10 users are found and stored in ascending order. To show the user who have watched the most films on the top of the list "max_10_users" is reversed.

Finally, every element of max_10_users is printed using a for loop.

3.2 Top 10 Movies

The algorithm iterates over the list "all_item_based_list" and finds the length of every subvector and stores them in a "integer vector" named "all_items_user_count". At every index of "all_items_user_count", indices corresponding to an itemID (Index 1 for itemID: 1), the item's total watch count is stored.

"all_items_user_count" is then traversed sequentially to find the 10 items which have been watched

by the most users.

The algorithm creates a pair array called "max_10_items". Initially, first 10 itemIDs and values of "all_items_user_count" is assigned to first 10 values of "max_10_items" as (itemID, watchCount). Then the pair array is sorted with regard to items' watch counts with insertion sort. After that, the algorithm iterates over every element of "all_items_user_count" and in every iteration "all_items_user_count[i]" is compared with "max_10_items[0].second" (smallest element with regard to watch count).

- If "all_items_user_count[i]" is smaller than or equal to "max_10_items[0].second" nothing happens (Code continues).
- If "all_items_user_count[i]" is larger than "max_10_users[0].second" then "max_10_items[0]" is assigned as the pair: (i, all_items_user_count[i]). Then the array is again sorted to make sure the smallest element is always contained in the first index, index 0.

After iterating over the whole "all_items_user_count" vector, the top 10 users are found and stored ordered increasingly. To show the user who have watched the most films on the top of the list "max_10_items" is reversed.

Finally, every element of max_10_items is printed using a for loop.

```
/Users/sercaneser/CLionProjects/list_of_all/cmake-build-debug/untitled16
UserID: 190 Number of films watched: 1633
UserID: 1475 Number of films watched: 1436
UserID: 591 Number of films watched: 1388
UserID: 19 Number of films watched: 1330
UserID: 52 Number of films watched: 1087
UserID: 24 Number of films watched: 964
UserID: 703 Number of films watched: 935
UserID: 1758 Number of films watched: 906
UserID: 1814 Number of films watched: 882
UserID: 942 Number of films watched: 836

MovieID: 118 Number of users watched: 16000
MovieID: 198 Number of users watched: 13145
MovieID: 11 Number of users watched: 11993
MovieID: 135 Number of users watched: 11249
MovieID: 49 Number of users watched: 10533
MovieID: 27 Number of users watched: 10009
MovieID: 8 Number of users watched: 9724
MovieID: 72 Number of users watched: 8779
MovieID: 30 Number of users watched: 8174
MovieID: 61 Number of users watched: 7799
Run-time of code excluding reading the csv file (in microseconds): 3123

Process finished with exit code 0
```

4 Prediction Algorithm

In this part we use the data from pre-processing part in our functions to find similarities between users and using these similarities, we predict users' ratings.

4.1 Functions

To accurately predict users' movie ratings, 5 different functions are used. These are:

4.1.1 Function 1: qualityVector

This function takes 4 inputs, these are:

- rating_pair_vector: A vector-vector of pairs had come from pre-processing of raw data.
- size: An integer variable which stores the number of users whom watched the predicted movie.
- requiredQuality: An integer variable which stores the minimum number of common films between predicted user and other users.
- minimum_user_size: An integer which stores the minimum amount of users added to "data" for similarity calculations to begin.

This function compares the size of "common movies vector" with requiredQuality. If it is bigger than requiredQuality, function adds this vector to the new "vector⟨vector⟨pair⟩⟩" which is named "data". If the size of "data" is less than minimum_user_size than function recursively calls itself with requiredQuality decreased by 10. And finally returns this "data" vector-vector pair.

4.1.2 Function 2: fundamentalFunction

This function takes a "vector⟨vector⟨pair⟩⟩" ("data") as an input and calculates the average rating of each user*. Then the calculated average for each user is subtracted from all of their ratings, in order to center the ratings.

Table 1: Raw Data

-1	4.5
1.5	3.0
4.5	5.0
2.0	3.5
1.0	2.5

Table 2: Centered Data

-1	1.0
-0.75	-0.5
2.25	1.5
-0.25	0.0
-1.25	-1.0

Finally, this centered ratings are supplied to cosineSimilarity function.

*: (During calculations of average rating, first row of every sub-vector is excluded. Because first row is used for delivering compared user's rating for the film we are trying to predict.)

4.1.3 Function 3: cosineSimilarity

This function takes 2 inputs. These are:

- data: The "vector⟨vector⟨pair⟩⟩" which was centered by the fundamentalFunction function.
- special average: The average of the user we are going to predict about.

In this function we use cosine similarity formula for the similarity calculations between two users.

$$cossim(U_1, U_2) = \frac{\sum_{i=1}^{n} U_{1_i} \times U_{2_i}}{\sqrt{\sum_{i=1}^{n} {U_{1_i}}^2} \times \sqrt{\sum_{i=1}^{n} {U_{2_i}}^2}} \tag{1}$$

Cosine Similarity Function

After these calculations we use the equation below to predict user's movie rating.

$$r_p = U_p avr + \frac{\sum_{i=1}^{n}[cossim(U_p, U_i) \times r_{ip} \times s]}{\sum_{i=1}^{n}[|cossim(U_p, U_i)| \times s]}$$

$r_p : rating\ predicted,\ U_p avr : average\ of\ predicted\ user's\ movie\ ratings,$
$r_i p : rating\ of\ other\ user's\ compared\ movie,\ s : number\ of\ common\ movies\ between\ these\ users$
Rating Prediction Function (2)

And this function returns this value as a float variable finalresult.

4.1.4 Function 4: predictor

It is a function which create better data and predicted movie rate by using qualityVector function and then use this data in the fundamentalFunction. It has 4 inputs these are inputs of fundamentalFunction and qualityVector functions which are we explain them before. And return the rating prediction as a float variable finalresult.

4.1.5 Function 5: finalFunction

This function takes 3 inputs. These are:

- rate_pair_vector: Raw data from preprocessing, still not centered.
- size: An integer variable which stores the number of users who watched the predicted film.
- mode: Modes for making predicted ratings accurate, predictor function can some times return the prediction as nan or negative. To avoid this, two modes are used this function:
  - mode 0: In this mode minimum_user_size is 40 and predictor function starts from requiredQuality 50. If there are nan or negative values, function decrements the requiredQuality (to 40-30-20-10-5 after each failed try.). If after the last decrement there is still nan or negative values, result (predicted rating) is assigned 3.5.
  - mode 1: In this mode function starts from minimum_user_size 100 and requiredQuality 55. Rest is same with mode 0.

To get much more accurate ratings in main function we calculate mode 0 and mode 1. After that we take their average.

5 Printing Results to the Submission CSV File

In this part we collect our results in vector structure and then by using fstream we write this results to the CSV file. We chose vector because its is implemented as a dynamically allocated array in C++ so we can access the results very quickly.

6 Running Time of Algorithms

6.1 Ranking Top 10 Users and Movies

Running time of the this part of the program is 3123 microseconds.
Output:

```
Run-time of code excluding reading the csv file (in microseconds): 3123
```

6.2 Pre-Processing and Prediction

In this part we have three situations. These are:

1. When only mode 0 of the finalFunction is used to predict ratings.

   - In this mode, algorithm assigned minimum_user_size as 40, requiredQuality as 50. It is the least accurate mode of this program. The running time is: 96746103 microseconds.

   Output:

```
(base) polatozturk@Polat-MacBook-Air CMP2003Project_Final % ./main00
96746103 microseconds
```

2. When only mode 1 of the finalFunction is used to predict ratings.

   - In this mode, algorithm assigned minimum_user_size as 100, requiredQuality as 55. It is the most accurate mode and (somehow) a little bit faster than mode 0. Running time of this algorithm is: 96724061 microseconds.

   Output:

```
(base) polatozturk@Polat-MacBook-Air CMP2003Project_Final % ./main22
96724061 microseconds
```

3. When the average of mode 0 and mode 1 of the finalFunction is used.

   - Running time of this algorithm is: 121825404 microseconds.

   Output:

```
(base) polatozturk@Polat-MacBook-Air CMP2003Project_Final % ./main33
121825404 microseconds
```