Report for the practical assignment

Part 2: ITEM-ITEM

INFDTA01-1 – Data Science (Recommender systems) – 2015/16

**Important**: you can work in groups of *two* students, but the final report is *individual*. The *code* can be the same for the two students working in pair, but *explanations/answers* must be different from each other. The code must be readable (by placing comments and naming well the variables).

**Delivery:** Fill this document with your answers (code, explanations, etc…) and upload a zipped folder (containing *this document* and *all your code*) on N@tschool *before* the deadline specified in the modulewijzer.

**Evaluation:** Each of the numbered sections below gives up to 1 point. The total number of points for this part of the assignment is, thus, 11. The final grade of the whole practical assignment is obtained combining the points of Part 1 and Part 2 (see formula in the modulewijzer).

Student name: Geddy Schellevis

Student number: 0883374

Class (3A/B/C): 3C

Language (Java, C#, …): Java

[ If code was developed in pairs, name of the other student: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ]

# Computing deviations

1. public HashMap<Integer, HashMap<Integer, Deviations>> calculate(HashMap<String, UserPreference> userRatings){  
     
    HashMap<Integer, HashMap<Integer,Integer>> freg = new HashMap<>();  
    HashMap<Integer, HashMap<Integer,Deviations>> dev = new HashMap<>();  
     
    // loop trough all the users  
    for (Map.Entry<String,UserPreference> user : userRatings.entrySet()) {  
     
    // loop trough all ratings of a user  
    for(Map.Entry<Integer,Double> targetRating : user.getValue().getRatings().entrySet()){  
    // loop trough all items again for compare them  
    for(Map.Entry<Integer,Double> compareRating : user.getValue().getRatings().entrySet()){  
    // if rating != targetRating calculate slope one  
    if(targetRating != compareRating){  
     
    // create the hashmap or get the right hashmap  
    HashMap<Integer, Integer> temp;  
    HashMap<Integer, Deviations> tempDev;  
    int amount = 1;  
    double rate = 0.0;  
    if (freg.size() == 0 || !freg.containsKey(targetRating.getKey())) {  
    temp = new HashMap<>();  
    tempDev = new HashMap<>();  
    }else{  
    temp = freg.get(targetRating.getKey());  
    if(temp.containsKey(compareRating.getKey())) {  
    amount = temp.get(compareRating.getKey()) + 1;  
    }  
    tempDev = dev.get(targetRating.getKey());  
    if(tempDev.containsKey(compareRating.getKey())) {  
    rate = tempDev.get(compareRating.getKey()).getRating();  
    }  
    }  
     
    // Calculate the slope one ans update the hashmap  
    rate += targetRating.getValue() - compareRating.getValue();  
    temp.put(compareRating.getKey(),amount);  
     
    // create Deviations object  
    Deviations deviation = new Deviations(compareRating.getKey(),rate,amount);  
     
    // update the hashmap  
    tempDev.put(compareRating.getKey(),deviation);  
    dev.put(targetRating.getKey(),tempDev);  
    freg.put(targetRating.getKey(),temp);  
     
   // }  
    }  
    }  
    }  
    }  
     
    // loop trough the hashmap we just creates for calculating the right deviation  
    // depending on the amount of ratings  
    HashMap<Integer, HashMap<Integer,Deviations>> deviationMap = new HashMap<>();  
    for (Map.Entry<Integer, HashMap<Integer,Deviations>> deviationItem : dev.entrySet())  
    for (Map.Entry<Integer, Deviations> devItem : deviationItem.getValue().entrySet()) {  
    HashMap<Integer, Deviations> tempDev;  
     
    if (deviationMap.size() == 0 ||  
    !deviationMap.containsKey(deviationItem.getKey())) {  
    tempDev = new HashMap<>();  
    }else {  
    tempDev = deviationMap.get(deviationItem.getKey());  
    }  
     
    double a = devItem.getValue().getRating() / freg.get(deviationItem.getKey()).get(devItem.getKey());  
     
    Deviations deviation = new Deviations(devItem.getKey(), a, devItem.getValue().getAmountOfRatings());  
    tempDev.put(devItem.getKey(), deviation);  
    deviationMap.put(deviationItem.getKey(), tempDev);  
    }  
    return deviationMap;  
    }

# Predicting ratings and the top recommendations

public List<Recommendation> predictRating(UserPreference targetUser,HashMap<Integer, HashMap<Integer,Deviations>> slopeOne ,  
 int numberOfRecommendations){  
 HashMap<Integer,Double> targetUserRatings = targetUser.getRatings();  
  
 HashMap<Integer,Integer> amountOfRaters = new HashMap<>();  
 HashMap<Integer,Double> tempRecommendations = new HashMap<>();  
  
 for(Map.Entry<Integer,Double> targetUserRating : targetUserRatings.entrySet()){  
 for (Map.Entry<Integer,HashMap<Integer,Deviations>> deviations : slopeOne.entrySet()){  
 if(!targetUserRatings.containsKey(deviations.getKey()) && deviations.getValue().containsKey(targetUserRating.getKey())){  
 Deviations dev = deviations.getValue().get(targetUserRating.getKey());  
 int amoutOfRatings = 0;  
 if(amountOfRaters.containsKey(deviations.getKey())){  
 amoutOfRatings = amountOfRaters.get(deviations.getKey());  
 }  
 amoutOfRatings += dev.getAmountOfRatings();  
 amountOfRaters.put(deviations.getKey(),amoutOfRatings);  
 double rating = 0.0;  
 if(tempRecommendations.containsKey(deviations.getKey())){  
 rating = tempRecommendations.get(deviations.getKey());  
 }  
  
 rating += (dev.getRating() + targetUserRating.getValue()) \* dev.getAmountOfRatings();  
 tempRecommendations.put(deviations.getKey(),rating);  
 }  
  
 }  
  
 }  
 ArrayList<Recommendation> recommendations = new ArrayList<>();  
 for(Map.Entry<Integer,Double> rec : tempRecommendations.entrySet()) {  
 double rating = rec.getValue() / amountOfRaters.get(rec.getKey());  
 Recommendation recommendation = new Recommendation(rec.getKey(), rating);  
 recommendations.add(recommendation);  
 }  
  
 recommendations.sort(new recommendationComparator());  
 if(recommendations.size() > numberOfRecommendations){  
 return recommendations.subList(0,numberOfRecommendations);  
 }  
 return recommendations;  
}

# Computing the top recommendations

*See code above*

# Updating deviations

public HashMap<Integer, HashMap<Integer,Deviations>> update(HashMap<Integer, HashMap<Integer,Deviations>> deviations,  
 Pair<Integer,Double> updateItem,  
 UserPreference targetUser){  
 // loop trow the x as of items  
 for (Map.Entry<Integer, HashMap<Integer,Deviations>> deviationItem : deviations.entrySet()) {  
  
 if (updateItem.getKey() == deviationItem.getKey()) {  
 // loop trough the y as  
 for (Map.Entry<Integer, Deviations> devItem : deviationItem.getValue().entrySet()) {  
 if (updateItem.getKey() != devItem.getKey() && targetUser.getRatings().containsKey(devItem.getKey())){  
 Deviations dev = devItem.getValue();  
 Double newDev = ((dev.amountOfRatings\*dev.rating)+(updateItem.getValue() - targetUser.getRating(devItem.getKey())))/ (dev.amountOfRatings + 1);  
 deviationItem.getValue().put(devItem.getKey(),new Deviations(devItem.getKey(),newDev,dev.amountOfRatings + 1));  
 }  
 }  
 }else{  
 for (Map.Entry<Integer, Deviations> devItem : deviationItem.getValue().entrySet()) {  
 if (updateItem.getKey() == devItem.getKey() && targetUser.getRatings().containsKey(deviationItem.getKey())){  
 Deviations dev = devItem.getValue();  
 Double newDev = ((dev.amountOfRatings\*dev.rating)+( targetUser.getRating(deviationItem.getKey()) - updateItem.getValue()))/ (dev.amountOfRatings + 1);  
 deviationItem.getValue().put(devItem.getKey(),new Deviations(devItem.getKey(),newDev,dev.amountOfRatings + 1));  
 }  
 }  
 }  
 }  
 return deviations;  
}

# Comment on updating deviations

*Which* deviations do you have to update when inserting a new item rating? All of them or only a subset? Why?

You only have to update a subset. If you see the data set after the deviations in a table you have to update the rows of the updated value and the collums of the updated values in all the other rows.

# Results of predicting ratings

Apply the *Slope One* algorithm to the small dataset imported at the beginning of Part 1 (*userItem.data*) and compute the predicted ratings for user**7**:

Predicted rating for item 101: 2.61

Predicted rating for item 103: 2.17

Predicted rating for item 106: 3.8

Compute then the predicted ratings for user **3**:

Predicted rating for item 103: 2.17

Predicted rating for item 105: 2.34

# Reflection on predicted ratings

Using the Item-Item technique, is it possible that the result of a predicted rating is higher than 5 (i.e., the maximum rating)? Why? Make a simple example.

It’s possible to get a value higher than the maximum rating if a user rate an item with a 5 and we have a positive deviation ( 1 in the example)we can get a rating higher then 5

Sum((Rating + deviation) \* freq) / sum(freg)

((5+1) \*2)/2 = 6

# Results of predicted ratings after updating deviations

Suppose now that user **3** rates item **105** with **4.0**. Update the deviations and compute again the predicted ratings for user **7**:

Predicted rating for item 101: 2.4

Predicted rating for item 103: 2.17

Predicted rating for item 106: 3.06

# Comment on results

Which of the three predicted ratings (items 101, 103, 106) change and which stay the same? Explain *why* that happens.

User 3 didn’t rate the movie 103 that’s why 103 didn’t change after the update. The other 2 (101,106) change because we compare each of this ratings for the deviation.

# 10) Results of top recommendations

Apply the Slope One algorithm to the MovieLens100k dataset and compute the 5 top recommendations for user 186.

Recommendation 1: Someone Else's America (1995) with predicted rating 5.6

Recommendation 2: Aiqing wansui (1994) with predicted rating 5.38

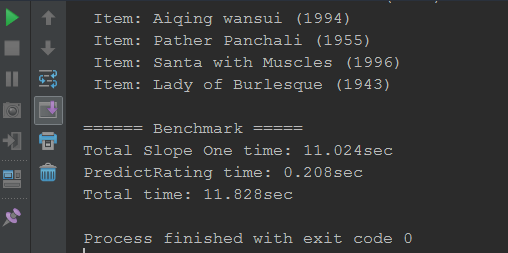
Recommendation 3: Pather Panchali (1955) with predicted rating 5.23

Recommendation 4: Santa with Muscles (1996) with predicted rating 5.21

Recommendation 5: Lady of Burlesque (1943) with predicted rating 5.19

# 11) Performance of the system

How much time (in seconds) does it take to create a recommendation for a user (*excluding* the computation of the deviations)?

**

Do you think that the performance of your system is good? If not, which parts of your code would you change (and how) to improve the performance?

I think my code Is pretty fast. I think the way for a faster result by a greater dataset is too use spark. That’s because I think it fast because I use objects instead of hashmap. All the movies are in a separated map so I can reach it with the key.