Recommender System

0. Introduction

- Lecture: 데이터사이언스(ITE4005)
- Student: 한양대학교 컴퓨터소프트웨어학부 2017030191 이현지
- Programming Assignment #4: Predict the ratings of movies in test data by using the given training data containing movie ratings of users.

1. Main Concepts

1. Summary

1. Main Idea

   train data로부터 user가 item에 대해 평가한 정보를 가져와 user based collaborative filtering을 수행한다.

2. User-Based Collaborative Filtering
   - 두 사용자가 얼마나 비슷하게 아이템을 평가했는지를 기준으로 코사인 유사도를 사용하여 계산한다.
   - 코사인 유사도

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

, where A_i and B_i are components of vector A and B respectively.

2. Main.class

1. Get data and preprocess

```
String trainFile = args[0], testFile = args[1];
ArrayList<Integer> users = getUsers(trainFile);
ArrayList<Integer> items = getItems(trainFile);
Integer maxUserId = users.get(users.size()-1), maxItemId = items.get(items.size()-1);
ArrayList<ArrayList<Double>> trainData = getTrainData(trainFile, maxUserId, maxItemId);
```

   - trainFile, testFile: 소스 코드 실행 시 인자로 받는 파일의 이름
     - 경로를 명시하지 않으면 jar파일과 같은 경로에 존재
   - users: user id를 모두 저장한 리스트
   - items: item id를 모두 저장한 리스트
   - trainData: user id * item id = rating 으로 이루어진 리스트

2. Compute cosine similarity among users

```
ArrayList<ArrayList<Double>> userSimilarity = getUserSimilarity(trainData, users);
```

   - userSimilarity: user 사이의 코사인 유사도를 계산한 2차원 리스트

3. Get prediction

```
ArrayList<ArrayList<Double>> prediction = getPrediction(trainData, userSimilarity);
```

- prediction: userSimilarity를 활용하여 prediction을 수행한 리스트

4. Make output file

```
printPrediction(trainFile, prediction);
```

2. Details

1. getPrediction()

1. Specification
   - Paramter: ArrayList<ArrayList<Double>> trainData, ArrayList<ArrayList<Double>> userSimilarity
   - Return type: ArrayList<ArrayList<Double>>
2. Description

> 💡 user가 평하지 않은 item이 있으면 userSimilarity를 기반으로 weighted sum 값을 구한다.

3. Code

```
public static ArrayList<ArrayList<Double>> getPrediction(ArrayList<ArrayList<Double>> trainData, ArrayList<ArrayList<Double>> user
    ArrayList<ArrayList<Double>> prediction = new ArrayList<>();
    for(int i = 0 ; i < trainData.size() ; i++){
        prediction.add(new ArrayList<>(Collections.nCopies(trainData.get(0).size(), null)));
    }

    for(int i = 1 ; i < trainData.size() ; i++){
        for(int j = 1 ; j < trainData.get(i).size() ; j++){
            if(trainData.get(i).get(j) == null){
                Double numerator = 0.0, denominator = 0.0;
                for(int k = 1 ; k < userSimilarity.get(i).size() ; k++){
                    if(userSimilarity.get(i).get(k) != null && trainData.get(k).get(j) != null){
                        numerator += (userSimilarity.get(i).get(k) * trainData.get(k).get(j));
                        denominator += userSimilarity.get(i).get(k);
                    }
                }
                Double nowPrediction = numerator / denominator;
                if(denominator == 0) nowPrediction = 0.0;
                prediction.get(i).set(j, nowPrediction);
            } else {
                prediction.get(i).set(j, trainData.get(i).get(j));
            }
        }
    }

    return prediction;
}
```

2. getCosineSimilarity()

1. Specification
   - Paramter: ArrayList<Double> left, ArrayList<Double> right
   - Return type: Double
2. Description

> 💡 두 user가 공통으로 평가한 rating에 대해 cosine similarity를 계산한다.

3. Code

```
public static Double getCosineSimilarity(ArrayList<Double> left, ArrayList<Double> right){
    Double numerator = 0.0;
    for(int i = 0 ; i < left.size() ; i++){
```

```
        numerator += (left.get(i) * right.get(i));
    }

    Double denominator = 0.0;
    Double leftNum = 0.0;
    for(int i = 0 ; i < left.size() ; i++){
        leftNum += Math.pow(left.get(i), 2);
    }
    leftNum = Math.sqrt(leftNum);
    Double rightNum = 0.0;
    for(int i = 0 ; i < right.size() ; i++){
        rightNum += Math.pow(right.get(i), 2);
    }
    rightNum = Math.sqrt(rightNum);
    denominator = leftNum * rightNum;

    if (denominator == 0.0) return 0.0;
    return numerator / denominator;
}
```

3. getUserSimilarity()

1. Specification
   - Paramter: ArrayList<ArrayList<Double>> trainData, ArrayList<Integer> users
   - Return type: ArrayList<ArrayList<Double>>
2. Description

> 💡 모든 user에 대해 공통으로 평가한 item 리스트를 구해 cosine similarity를 계산하여 userSimilarity 리스트를 만든다.

3. Code

```
public static ArrayList<ArrayList<Double>> getUserSimilarity(ArrayList<ArrayList<Double>> trainData, ArrayList<Integer> users){
    ArrayList<ArrayList<Double>> userSimilarity = new ArrayList<>();
    for(int i = 0 ; i < trainData.size() ; i++){
        userSimilarity.add(new ArrayList<>(Collections.nCopies(trainData.size(), null)));
    }

    for(int i = 1 ; i < users.size() ; i++){
        Integer nowUser = users.get(i);
        for(int j = 1 ; j < users.size() ; j++){
            Integer nextUser = users.get(j);
            if(userSimilarity.get(nowUser).get(nextUser) == null){
                ArrayList<Double> nowUserRatings = new ArrayList<>(), nextUserRatings = new ArrayList<>();
                for(int k = 1 ; k <= trainData.size() ; k++){
                    if(trainData.get(nowUser).get(k) != null && trainData.get(nextUser).get(k) != null){
                        nowUserRatings.add(trainData.get(nowUser).get(k));
                        nextUserRatings.add(trainData.get(nextUser).get(k));
                    }
                }

                Double cosineSimilarity = getCosineSimilarity(nowUserRatings, nextUserRatings);
                userSimilarity.get(nowUser).set(nextUser, cosineSimilarity);
                userSimilarity.get(nextUser).set(nowUser, cosineSimilarity);
            }
        }
    }

    return userSimilarity;
}
```

4. getUsers()

1. Specification
   - Paramter: String trainFile
   - Return type: ArrayList<Integer>
2. Description

> 💡 trainFile에서 user id를 모두 모아 리스트로 만든다.

3. Code

```
public static ArrayList<Integer> getUsers(String trainFile){
    ArrayList<Integer> users = new ArrayList<>();
    users.add(0);

    try {
        BufferedReader reader = new BufferedReader(new FileReader(trainFile));
        String line = reader.readLine();

        while(line != null){
            StringTokenizer tokenizer = new StringTokenizer(line, "\t");

            Integer userId = Integer.parseInt(tokenizer.nextToken());
            if(!users.contains(userId)){
                users.add(userId);
            }

            line = reader.readLine();
        }
        reader.close();
    } catch (Exception e) {
        e.printStackTrace();
        System.exit(0);
    }
    Collections.sort(users);
    return users;
}
```

5. getUsers()

1. Specification
   - Paramter: String trainFile
   - Return type: ArrayList<Integer>
2. Description

> 💡 trainFile에서 item id를 모두 모아 리스트로 만든다.

3. Code

```
public static ArrayList<Integer> getItems(String trainFile){
    ArrayList<Integer> items = new ArrayList<>();
    items.add(0);

    try {
        BufferedReader reader = new BufferedReader(new FileReader(trainFile));
        String line = reader.readLine();

        while(line != null){
            StringTokenizer tokenizer = new StringTokenizer(line, "\t");

            Integer userId = Integer.parseInt(tokenizer.nextToken());
            Integer itemId = Integer.parseInt(tokenizer.nextToken());
            if(!items.contains(itemId)){
                items.add(itemId);
            }

            line = reader.readLine();
        }
        reader.close();
    } catch (Exception e) {
        e.printStackTrace();
        System.exit(0);
    }
    Collections.sort(items);
    return items;
}
```

6. getUsers()

1. Specification

- Paramter: String trainFile, Integer maxUserId, Integer maxItemId
- Return type: AArrayList<ArrayList<Double>>rrayList<Integer>

2. Description

> 💡 trainFile에서 모든 user에 대해 모든 item을 어떻게 평가했는 지 2차원 리스트로 만든다. 없으면 null 값으로 처리한다.

3. Code

```
public static ArrayList<ArrayList<Double>> getTrainData(String trainFile, Integer maxUserId, Integer maxItemId){
    ArrayList<ArrayList<Double>> trainData = new ArrayList<>();
    for(int i = 0 ; i < maxUserId + 1 ; i++){
        trainData.add(new ArrayList<>(Collections.nCopies(maxItemId + 1, null)));
    }

    try {
        BufferedReader reader = new BufferedReader(new FileReader(trainFile));
        String line = reader.readLine();

        while(line != null){
            StringTokenizer tokenizer = new StringTokenizer(line, "\t");

            Integer userId = Integer.parseInt(tokenizer.nextToken());
            Integer itemId = Integer.parseInt(tokenizer.nextToken());
            Double rating = Double.parseDouble(tokenizer.nextToken());

            trainData.get(userId).set(itemId, rating);

            line = reader.readLine();
        }
        reader.close();
    } catch (Exception e) {
        e.printStackTrace();
        System.exit(0);
    }
    return trainData;
}
```

7. printPrediction()

1. Specification
    - Paramter: String trainFile, ArrayList<ArrayList<Double>> prediction
    - Return type: void
2. Description

> 💡 getPrediction() 함수를 통해 얻은 prediction을 형식에 맞춰 output file을 만든다.

3. Code

```
public static void printPrediction(String trainFile, ArrayList<ArrayList<Double>> prediction){
    String outputFile = trainFile.split("\\.")[0];
    try {
        BufferedWriter writer = new BufferedWriter(new FileWriter(outputFile + ".base_prediction.txt"));

        for(int i = 1 ; i < prediction.size() ; i++){
            for(int j = 1 ; j < prediction.get(i).size() ; j++){
                writer.write(i + "\t" + j + "\t" + prediction.get(i).get(j));
                writer.newLine();
            }
            writer.newLine();
        }
        writer.close();

    } catch (Exception e) {
        e.printStackTrace();
        System.exit(0);
    }
}
```

3. How to Execute

1. Environment

- OS: Mac OS Catalina 10.15.7
- Runtime: JDK 15.0.1
- IDE: IntelliJ Ultimate 2020.03

2. Structure

execution - recommender.jar

ㄴ u1.base

ㄴ u1.test

ㄴ u2.base

ㄴ u2.base

ㄴ u3.test

ㄴ u3.base

ㄴ u4.base

ㄴ u4.test

ㄴ u5.base

ㄴ u5.test

project - src - cse.ds - Main.class

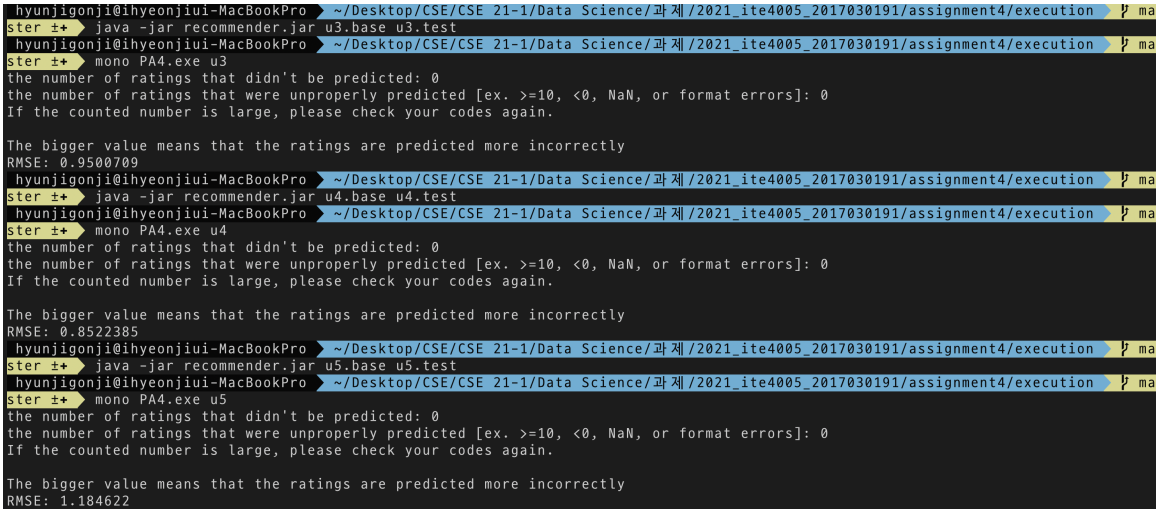
ㄴ out - artifacts - recommender.jar

3. Usage

```
 hyunjigonji@ihyeonjiui-MacBookPro  ~/Desktop/CSE/CSE 21-1/Data Science/과제/2021_ite4005_2017030191/assignment4/execution  master ±+  ll
total 94736
-rw-------@ 1 hyunjigonji  staff   8.0K  5 23  2019 PA4.exe
-rw-r--r--  1 hyunjigonji  staff   4.2K  6  9 00:39 recommender.jar
-rw-r--r--@ 1 hyunjigonji  staff   1.5M  3  8  2001 u1.base
-rw-r--r--  1 hyunjigonji  staff    37M  6  9 00:39 u1.base_prediction.txt
-rw-r--r--@ 1 hyunjigonji  staff   383K  3  8  2001 u1.test
-rw-r--r--@ 1 hyunjigonji  staff   1.5M  3  8  2001 u2.base
-rw-r--r--@ 1 hyunjigonji  staff   386K  3  8  2001 u2.test
-rw-r--r--@ 1 hyunjigonji  staff   1.5M  3  8  2001 u3.base
-rw-r--r--@ 1 hyunjigonji  staff   387K  3  8  2001 u3.test
-rw-r--r--@ 1 hyunjigonji  staff   1.5M  3  8  2001 u4.base
-rw-r--r--@ 1 hyunjigonji  staff   388K  3  8  2001 u4.test
-rw-r--r--@ 1 hyunjigonji  staff   1.5M  3  8  2001 u5.base
-rw-r--r--@ 1 hyunjigonji  staff   388K  3  8  2001 u5.test
 hyunjigonji@ihyeonjiui-MacBookPro  ~/Desktop/CSE/CSE 21-1/Data Science/과제/2021_ite4005_2017030191/assignment4/execution  master ±+  java -jar recommender.jar u1.base u1.test
 hyunjigonji@ihyeonjiui-MacBookPro  ~/Desktop/CSE/CSE 21-1/Data Science/과제/2021_ite4005_2017030191/assignment4/execution  master ±+  mono PA4.exe u1
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format errors]: 0
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 1.005456
 hyunjigonji@ihyeonjiui-MacBookPro  ~/Desktop/CSE/CSE 21-1/Data Science/과제/2021_ite4005_2017030191/assignment4/execution  master ±+  java -jar recommender.jar u2.base u2.test
 hyunjigonji@ihyeonjiui-MacBookPro  ~/Desktop/CSE/CSE 21-1/Data Science/과제/2021_ite4005_2017030191/assignment4/execution  master ±+  mono PA4.exe u2
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format errors]: 0
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 1.045179
```



- execution 폴더의 clustering.jar 파일로 실행 가능

```
% java -jar recommender.jar {train_file} {test_file}

ex) java -jar recommender.jar u1.base u1.test
```