协同过滤推荐算法

1. 内容

在真实的业务场景下，我们往往需要对所有商品的一个子集构建个性化推荐模型。在完成这件任务的过程中，我们不仅需要利用用户在这个商品子集上的行为数据，往往还需要利用更丰富的用户行为数据。定义如下的符号：  
U——用户集合  
I——商品全集  
P——商品子集，P ⊆ I  
D——用户对商品全集的行为数据集合  
那么我们的目标是利用D来构造U中用户对P中商品的推荐模型。

**数据说明**

本场比赛提供20000用户的完整行为数据以及百万级的商品信息。竞赛数据包含两个部分。

第一部分是用户在商品全集上的移动端行为数据（D）,表名为tianchi\_fresh\_comp\_train\_user\_2w，包含如下字段：

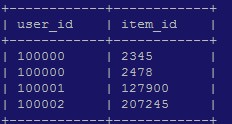
|  |  |  |
| --- | --- | --- |
| 字段 | 字段说明 | 提取说明 |
| user\_id | 用户标识 | 抽样&字段脱敏 |
| item\_id | 商品标识 | 字段脱敏 |
| behavior\_type | 用户对商品的行为类型 | 包括浏览、收藏、加购物车、购买，对应取值分别是1、2、3、4。 |
| user\_geohash | 用户位置的空间标识，可以为空 | 由经纬度通过保密的算法生成 |
| item\_category | 商品分类标识 | 字段脱敏 |
| time | 行为时间 | 精确到小时级别 |

第二个部分是商品子集（P）,表名为tianchi\_fresh\_comp\_train\_item\_2w，包含如下字段：

|  |  |  |
| --- | --- | --- |
| 字段 | 字段说明 | 提取说明 |
| item\_id | 商品标识 | 抽样&字段脱敏 |
| item\_ geohash | 商品位置的空间标识，可以为空 | 由经纬度通过保密的算法生成 |
| item\_category | 商品分类标识 | 字段脱敏 |

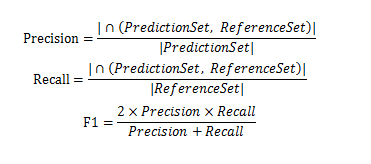
训练数据包含了抽样出来的一定量用户在一个月时间（11.18~12.18）之内的移动端行为数据（D），评分数据是这些用户在这个一个月之后的一天（12.19）对商品子集（P）的购买数据。参赛者要使用训练数据建立推荐模型，并输出用户在接下来一天对商品子集购买行为的预测结果。

**评分数据格式**  
具体计算公式如下：参赛者完成用户对商品子集的购买预测之后，需要将结果放入指定格式的数据表（非分区表）中，要求结果表名为：tianchi\_mobile\_recommendation\_predict.csv，且以utf-8格式编码；包含user\_id和item\_id两列（均为string类型）,要求去除重复。例如：



**评估指标**

比赛采用经典的精确度(precision)、召回率(recall)和F1值作为评估指标。具体计算公式如下：



## 其中PredictionSet为算法预测的购买数据集合，ReferenceSet为真实的答案购买数据集合。我们以F1值作为最终的唯一评测标准。

1. 协同过滤推荐算法原理及实现流程
2. 基于用户的协同过滤推荐算法

基于用户的协同过滤推荐算法通过寻找与目标用户具有相似评分的邻居用户，通过查找邻居用户喜欢的项目，推测目标用户也具有相同的喜好。基于用户的协同过滤推荐算法基本思想是：根据用户-项目评分矩阵查找当前用户的最近邻居，利用最近邻居的评分来预测当前用户对项目的预测值，将评分最高的N个项目推荐给用户，其中的项目可理解为系统处理的商品。其算法流程图如下图1所示。



图1基于用户的协同过滤推荐算法流程

基于用户的协同过滤推荐算法流程为：

* 构建用户项目评分矩阵

R={,……}，T：m×n的用户评分矩阵，其中r={,,……，}为用户的评分向量，代表用户对项目的评分。

* 计算用户相似度

基于用户的协同过滤推荐算法，需查找与目标用户相似的用户。衡量用户之间的相似性需要计算每个用户的评分与其他用户评分的相似度，即评分矩阵中的用户评分记录。每个用户对项目的评分可以看作是一个n维的评分向量。使用评分向量计算目标用户与其他用户之间的相似度sim(i,j)，通常计算用户相似度的方法有三种：余弦相似度、修正的余弦相似度和皮尔森相关系数。

* 构建最近邻居集

最近邻居集Neighor(u)中包含的是与目标用户具有相同爱好的其他用户。为选取邻居用户，我们首先计算目标用户u 与其他用户v的相似度 sim(u,v)，再选择相似度最大的k个用户。用户相似度可理解为用户之间的信任值或推荐权重。通常，sim(u,v)∈[1,1]。用户相似度为1表示两个用户互相的推荐权重很大。如果为-1，表示两个用户的由于兴趣相差很大，因此互相的推荐权重很小。

* 预测评分计算

用户a 对项目i的预测评分 p(a,i)为邻居用户对该项目评分的加权评分值。显然，不同用户对于目标用户的影响程度不同，所以在计算预测评分时，不同用户有不同的权重。计算时，我们选择用户相似度作为用户的权重因子，计算公式如下：

*p(a,i)*=+ (1)

基于用户的协同过滤推荐算法实现步骤为：

* 实时统计user对item的打分，从而生成user-item表（即构建用户-项目评分矩阵）；
* 计算各个user之间的相似度，从而生成user-user的得分表，并进行排序；
* 对每一user的item集合排序；
* 针对预推荐的user，在user-user的得分表中选择与该用户最相似的N个用户，并在user-item表中选择这N个用户中已排序好的item集合中的topM；
* 此时的N\*M个商品即为该用户推荐的商品集。

1. 基于项目的协同过滤推荐算法

基于项目的协同过滤推荐算法依据用户-项目评分矩阵通过计算项目之间的评分相似性来衡量项目评分相似性，找到与目标项目最相似的n个项目作为最近邻居集。然后通过对目标项目的相似邻居赋予一定的权重来预测当前项目的评分，再将得到的最终预测评分按序排列，将评分最高的N个项目推荐给当前用户，其中的项目可理解为系统处理的商品。其算法流程如下图2所示。



图2基于项目的协同过滤推荐算法流程

基于项目的协同过滤推荐算法流程为：

首先，读取目标用户的评分记录集合；然后计算项目i与中其他项目的相似度，选取k个最近邻居；根据评分相似度计算公式计算候选集中所有项目的预测评分；最后选取预测评分最高的N个项目推荐给用户。

基于项目的协同过滤推荐算法预测评分与其他用户评分的加权评分值相关，不同的历史评分项目与当前项目i的相关度有差异，所以在进行计算时，不同的项目有不同的权重。评分预测函数p(u,i)，以项目相似度作为项目的权重因子，得到的评分公式如下：

*p(u,i)*=+ (2)

基于项目的协同过滤推荐算法实现步骤为：

* 实时统计user对item的打分，从而生成user-item表（即构建用户-项目评分矩阵）；
* 计算各个item之间的相似度，从而生成item-item的得分表，并进行排序；
* 对每一user的item集合排序；
* 针对预推荐的user，在该用户已选择的item集合中，根据item-item表选择与已选item最相似的N个item；
* 此时的N个商品即为该用户推荐的商品集。

1. 基于用户的协同过滤推荐算法与基于项目的协同过滤推荐算法比较

基于用户的协同过滤推荐算法：

可以帮助用户发现新的商品，但需要较复杂的在线计算，需要处理新用户的问题。

基于项目的协同过滤推荐算法：

准确性好，表现稳定可控，便于离线计算，但推荐结果的多样性会差一些，一般不会带给用户惊喜性。

1. 项目实现

针对移动推荐，我们选择使用基于用户的协同过滤推荐算法来进行实现。

1. 数据模型及其实体类

用户行为数据：(user.csv)

user\_id,item\_id,behavior\_type,user\_geohash,item\_category,time

10001082,285259775,1,97lk14c,4076,2014-12-08 18

10001082,4368907,1,,5503,2014-12-12 12

10001082,4368907,1,,5503,2014-12-12 12

10001082,53616768,1,,9762,2014-12-02 15

10001082,151466952,1,,5232,2014-12-12 11

10001082,53616768,4,,9762,2014-12-02 15

10001082,290088061,1,,5503,2014-12-12 12

10001082,298397524,1,,10894,2014-12-12 12

10001082,32104252,1,,6513,2014-12-12 12

10001082,323339743,1,,10894,2014-12-12 12

商品信息：(item.csv)

item\_id,item\_geohash,item\_category

100002303,,3368

100003592,,7995

100006838,,12630

100008089,,7791

100012750,,9614

100014072,,1032

100014463,,9023

100019387,,3064

100023812,,6700

**package** entity;

**public** **class** Item {

**private** String itemId;

**private** String itemGeoHash;

**private** String itemCategory;

**public** String getItemId() {

**return** itemId;

}

**public** **void** setItemId(String itemId) {

**this**.itemId = itemId;

}

**public** String getItemGeoHash() {

**return** itemGeoHash;

}

**public** **void** setItemGeoHash(String itemGeoHash) {

**this**.itemGeoHash = itemGeoHash;

}

**public** String getItemCategory() {

**return** itemCategory;

}

**public** **void** setItemCategory(String itemCategory) {

**this**.itemCategory = itemCategory;

}

}

**package** entity;

**public** **class** Score **implements** Comparable<Score> {

**private** String userId; // 用户标识

**private** String itemId; // 商品标识

**private** **double** score;

**public** String getUserId() {

**return** userId;

}

**public** **void** setUserId(String userId) {

**this**.userId = userId;

}

**public** String getItemId() {

**return** itemId;

}

**public** **void** setItemId(String itemId) {

**this**.itemId = itemId;

}

**public** **double** getScore() {

**return** score;

}

**public** **void** setScore(**double** score) {

**this**.score = score;

}

@Override

**public** **int** compareTo(Score o) {

**if** ((**this**.score - o.score) < 0) {

**return** 1;

}**else** **if** ((**this**.score - o.score) > 0) {

**return** -1;

}**else** {

**return** 0;

}

}

}

**package** entity;

**public** **class** User **implements** Comparable<User> {

**private** String userId; // 用户标识

**private** String itemId; // 商品标识

**private** **int** behaviorType; // 用户对商品的行为类型,可以为空,包括浏览、收藏、加购物车、购买，对应取值分别是1、2、3、4.

**private** String userGeoHash; // 用户位置的空间标识

**private** String itemCategory;// 商品分类标识

**private** String time; // 行为时间

**private** **int** count;

**private** **double** weight; // 权重

**public** String getUserId() {

**return** userId;

}

**public** **void** setUserId(String userId) {

**this**.userId = userId;

}

**public** String getItemId() {

**return** itemId;

}

**public** **void** setItemId(String itemId) {

**this**.itemId = itemId;

}

**public** **int** getBehaviorType() {

**return** behaviorType;

}

**public** **void** setBehaviorType(**int** behaviorType) {

**this**.behaviorType = behaviorType;

}

**public** String getUserGeoHash() {

**return** userGeoHash;

}

**public** **void** setUserGeoHash(String userGeoHash) {

**this**.userGeoHash = userGeoHash;

}

**public** String getItemCategory() {

**return** itemCategory;

}

**public** **void** setItemCategory(String itemCategory) {

**this**.itemCategory = itemCategory;

}

**public** String getTime() {

**return** time;

}

**public** **void** setTime(String time) {

**this**.time = time;

}

**public** **int** getCount() {

**return** count;

}

**public** **void** setCount(**int** count) {

**this**.count = count;

}

**public** **double** getWeight() {

**return** weight;

}

**public** **void** setWeight(**double** weight) {

**this**.weight = weight;

}

@Override

**public** **int** compareTo(User o) {

**return** (**int**)((-1) \* (**this**.weight - o.weight));

}

}

1. 工具类

文件处理工具:

package util;

public class FileTool {

public static FileReader fr=null;

public static BufferedReader br=null;

public static String line=null;

public static FileOutputStream fos1 = null,fos2 = null,fos3 = null;

public static PrintStream ps1 = null,ps2 = null,ps3 = null;

public static int count = 0;

/\*\*

\* 初始化写文件器(单一指针)

\* \*/

public static void initWriter1(String writePath) {

try {

fos1 = new FileOutputStream(writePath);

ps1 = new PrintStream(fos1);

} catch (FileNotFoundException e) {

e.printStackTrace();

}

}

/\*\*

\* 关闭文件器(单一指针)

\* \*/

public static void closeRedaer() {

try {

br.close();

fr.close();

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* 关闭文件器(单一指针)

\* \*/

public static void closeWriter1() {

try {

ps1.close();

fos1.close();

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* 初始化写文件器(双指针)

\* \*/

public static void initWriter2(String writePath1,String writePath2) {

try {

fos1 = new FileOutputStream(writePath1);

ps1 = new PrintStream(fos1);

fos2 = new FileOutputStream(writePath2);

ps2 = new PrintStream(fos2);

} catch (FileNotFoundException e) {

e.printStackTrace();

}

}

/\*\*

\* 关闭文件器(双指针)

\* \*/

public static void closeWriter2() {

try {

ps1.close();

fos1.close();

ps2.close();

fos2.close();

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* 初始化写文件器(三指针)

\* \*/

public static void initWriter3(String writePath1,String writePath2,String writePath3) {

try {

fos1 = new FileOutputStream(writePath1);

ps1 = new PrintStream(fos1);

fos2 = new FileOutputStream(writePath2);

ps2 = new PrintStream(fos2);

fos3 = new FileOutputStream(writePath3);

ps3 = new PrintStream(fos3);

} catch (FileNotFoundException e) {

e.printStackTrace();

}

}

/\*\*

\* 关闭文件器(三指针)

\* \*/

public static void closeWriter3() {

try {

ps1.close();

fos1.close();

ps2.close();

fos2.close();

ps3.close();

fos3.close();

} catch (IOException e) {

e.printStackTrace();

}

}

public static List readFileOne(String path,boolean isTitle,String token,String pattern) throws Exception {

List<Object> ret = new ArrayList<Object>();

fr = new FileReader(path);

br = new BufferedReader(fr);

int count = 0,i = 0;

if (isTitle) {

line = br.readLine();

count++;

}

while((line = br.readLine()) != null){

String[] strArr = line.split(token);

switch (pattern) {

case "item":

ret.add(ParseTool.parseItem(strArr));

break;

case "user":

ret.add(ParseTool.parseUser(strArr));

break;

case "score":

ret.add(ParseTool.parseScore(strArr));

default:

ret.add(line);

break;

}

count++;

if (count/100000 == 1) {

i++;

System.out.println(100000\*i);

count = 0;

}

}

closeRedaer();

return ret;

}

public static void makeSampleData(String inputPath,boolean isTitle,String outputPath,int threshold) throws Exception {

fr = new FileReader(inputPath);

br = new BufferedReader(fr);

initWriter1(outputPath);

if (isTitle) {

line = br.readLine();

}

int count = 0;

while((line = br.readLine()) != null){

ps1.println(line);

count++;

if (count == threshold) {

break;

}

}

closeRedaer();

}

public static List<String> traverseFolder(String dir) {

File file = new File(dir);

String[] fileList = null;

if (file.exists()) {

fileList = file.list();

}

List<String> list = new ArrayList<String>();

for(String path : fileList){

list.add(path);

}

return list;

}

public static Map<String, List<Score>> loadScoreMap(String path,boolean isTitle,String token) throws Exception {

fr = new FileReader(path);

br = new BufferedReader(fr);

if (isTitle) {

line = br.readLine();

}

Map<String, List<Score>> scoreMap = new HashMap<String, List<Score>>();

while((line = br.readLine()) != null){

String[] arr = line.split(token);

Score score = ParseTool.parseScore(arr);

List<Score> temp = new ArrayList<Score>();

if (scoreMap.containsKey(score.getUserId())) {

temp = scoreMap.get(score.getUserId());

}

temp.add(score);

scoreMap.put(score.getUserId(), temp);

}

closeRedaer();

return scoreMap;

}

public static Map<String, List<String>> loadPredictData(String path,boolean isTitle,String token) throws Exception {

fr = new FileReader(path);

br = new BufferedReader(fr);

if (isTitle) {

line = br.readLine();

}

Map<String, List<String>> map = new HashMap<String, List<String>>();

while((line = br.readLine()) != null){

String[] arr = line.split(token);

String userId = arr[0];

String itemId = arr[1];

List<String> temp = new ArrayList<String>();

if (map.containsKey(userId)) {

temp = map.get(userId);

}

temp.add(itemId);

map.put(userId, temp);

count++;

}

closeRedaer();

return map;

}

public static Map<String, List<String>> loadTestData(Map<String, List<String>> predictMap, String dir, boolean isTitle, String token) throws Exception {

List<String> fileList = traverseFolder(dir);

Set<String> predictKeySet = predictMap.keySet();

Map<String, List<String>> testMap = new HashMap<String, List<String>>();

for(String predictKey : predictKeySet){

if (fileList.contains(predictKey)) {

List<String> itemList = loadTestData(dir + predictKey, isTitle, token);

testMap.put(predictKey, itemList);

}

}

return testMap;

}

public static List<String> loadTestData(String path, boolean isTitle, String token) throws Exception {

fr = new FileReader(path);

br = new BufferedReader(fr);

if (isTitle) {

line = br.readLine();

}

List<String> list = new ArrayList<String>();

Set<String> set = new HashSet<String>();

while((line = br.readLine()) != null){

String[] arr = line.split(token);

set.add(arr[1]);

count++;

}

closeRedaer();

for(String item : set){

list.add(item);

}

return list;

}

public static Map<String, Double> loadUser\_ItemData(String path,boolean isTitle,String token) throws Exception {

fr = new FileReader(path);

br = new BufferedReader(fr);

if (isTitle) {

line = br.readLine();

}

Map<String, Double> map = new HashMap<String, Double>();

while((line = br.readLine()) != null){

String[] arr = line.split(token);

String itemId = arr[1];

double score = Double.valueOf(arr[2]);

if(map.containsKey(itemId)){

double temp = map.get(itemId);

if (temp > score) {

score = temp;

}

}

map.put(itemId, score);

}

closeRedaer();

return map;

}

public static Map<String, Set<String>> loadTestUser(String path,boolean isTitle,String token) throws Exception {

fr = new FileReader(path);

br = new BufferedReader(fr);

int count = 0,i = 0;

if (isTitle) {

line = br.readLine();

count++;

}

Map<String, Set<String>> map = new HashMap<String, Set<String>>();

while((line = br.readLine()) != null){

String[] arr = line.split(token);

String userId = arr[0];

String itemId = arr[1];

Set<String> set = new HashSet<String>();

if (map.containsKey(userId)) {

set = map.get(userId);

set.add(itemId);

}

map.put(userId, set);

count++;

if (count/100000 == 1) {

i++;

System.out.println(100000\*i);

count = 0;

}

}

closeRedaer();

return map;

}

}

解析工具:

package util;

public class ParseTool {

//判断是否为数字

public static boolean isNumber(String str) {

int i,n;

n = str.length();

for(i = 0;i < n;i++){

if (!Character.isDigit(str.charAt(i))) {

return false;

}

}

return true;

}

public static Item parseItem(String[] contents) {

Item item = new Item();

if (contents[0] != null && !contents[0].isEmpty()) {

item.setItemId(contents[0].trim());

}

if (contents[1] != null && !contents[1].isEmpty()) {

item.setItemGeoHash(contents[1].trim());

}

if (contents[2] != null && !contents[2].isEmpty()) {

item.setItemCategory(contents[2].trim());

}

return item;

}

public static User parseUser(String[] contents) {

User user = new User();

int n = contents.length;

if (contents[0] != null && !contents[0].isEmpty()) {

user.setUserId(contents[0].trim());

}

if (contents[1] != null && !contents[1].isEmpty()) {

user.setItemId(contents[1].trim());

}

/\*

// 2.调用CountFileTest需放开，其它需注释

if (contents[2] != null && !contents[2].isEmpty()) {

user.setBehaviorType(Integer.valueOf(contents[2].trim()));

}

// 2.调用CountFileTest需放开，其它需注释

if (contents[n-1] != null && !contents[n-1].isEmpty()) {

user.setCount(Integer.valueOf(contents[n-1].trim()));

}

\*/

// 3.调用PredictTest需放开，其它需注释

if (contents[n-1] != null && !contents[n-1].isEmpty()) {

user.setWeight(Double.valueOf(contents[n-1].trim()));

}

/\*

// 1.调用SpliteFileAndMakeScoreTable需放开，其它需注释

if (contents[3] != null && !contents[3].isEmpty()) {

user.setUserGeoHash(contents[3].trim());

}

if (contents[4] != null && !contents[4].isEmpty()) {

user.setItemCategory(contents[4].trim());

}

if (contents[5] != null && !contents[5].isEmpty()) {

user.setTime(contents[5].trim());

}

\*/

return user;

}

public static Score parseScore(String[] contents) {

Score score = new Score();

if (contents[0] != null && !contents[0].isEmpty()) {

score.setUserId(contents[0].trim());

}

if (contents[1] != null && !contents[1].isEmpty()) {

score.setItemId(contents[1].trim());

}

if (contents[2] != null && !contents[2].isEmpty()) {

score.setScore(Double.parseDouble(contents[2].trim()));

}

return score;

}

}

1. 数据处理模块:

package service;

public class DataProcess {

public static final double[] w = {0,10,20,30};

public static void output(Map<String, Map<String, List<User>>> userMap,String outputPath) {

for(Entry<String, Map<String, List<User>>> entry : userMap.entrySet()){

FileTool.initWriter1(outputPath + entry.getKey());

Map<String, List<User>> temp = entry.getValue();

for(Entry<String, List<User>> tempEntry : temp.entrySet()){

List<User> users = tempEntry.getValue();

int count = users.size();

for(User user : users){

FileTool.ps1.print(user.getUserId() + "\t");

FileTool.ps1.print(user.getItemId() + "\t");

FileTool.ps1.print(user.getBehaviorType() + "\t");

//FileTool.ps1.print(user.getUserGeoHash() + "\t");

//FileTool.ps1.print(user.getItemCategory() + "\t");

//FileTool.ps1.print(user.getTime() + "\t");

FileTool.ps1.print(count + "\n");

}

}

}

FileTool.closeWriter1();

}

public static void output(Map<String, Map<String, Double>> scoreTable, String outputPath, Set<String> userSet, Set<String> itemSet, String token) {

FileTool.initWriter1(outputPath);

for(String itemId: itemSet){

FileTool.ps1.print(token + itemId);

}

FileTool.ps1.println();

for(String userId : userSet){

FileTool.ps1.print(userId + token);

Map<String, Double> itemMap = scoreTable.get(userId);

for(String itemId: itemSet){

if(itemMap.containsKey(itemId)){

FileTool.ps1.print(itemMap.get(itemId));

}else {

//FileTool.ps1.print(0);

}

FileTool.ps1.print(token);

}

FileTool.ps1.print("\n");

}

}

public static void outputUser(List<User> userList) {

for(User user : userList){

FileTool.ps1.println(user.getUserId() + "\t" + user.getItemId() + "\t" + user.getWeight());

}

}

public static void outputScore(List<Score> scoreList) {

for(Score score : scoreList){

FileTool.ps1.println(score.getUserId() + "\t" + score.getItemId() + "\t" + score.getScore());

}

}

public static void outputRecommendList(Map<String, Set<String>> map) {

for(Entry<String, Set<String>> entry : map.entrySet()){

String userId = entry.getKey();

Set<String> itemSet = entry.getValue();

for(String itemId : itemSet){

FileTool.ps1.println(userId + "," + itemId);

}

}

}

public static void output(Map<String, Set<String>> map) {

for(Entry<String, Set<String>> entry : map.entrySet()){

String userId = entry.getKey();

Set<String> set = entry.getValue();

for(String itemId : set){

FileTool.ps1.println(userId + "\t" + itemId);

}

}

}

public static Map<String, Map<String, List<User>>> mapByUser(List<User> userList,Set<String> userSet,Set<String> itemSet) {

Map<String, Map<String, List<User>>> userMap = new HashMap<>();

for(User user: userList){

Map<String, List<User>> tempMap = new HashMap<String, List<User>>();

List<User> tempList = new ArrayList<User>();

if (!userMap.containsKey(user.getUserId())) {

}else {

tempMap = userMap.get(user.getUserId());

if (!tempMap.containsKey(user.getItemId())) {

}else {

tempList = tempMap.get(user.getItemId());

}

}

tempList.add(user);

tempMap.put(user.getItemId(), tempList);

userMap.put(user.getUserId(), tempMap);

userSet.add(user.getUserId());

itemSet.add(user.getItemId());

}

return userMap;

}

public static Map<String, Map<String, Double>> makeScoreTable(Map<String, Map<String, List<User>>> userMap) {

Map<String, Map<String, Double>> scoreTable = new HashMap<String, Map<String,Double>>();

for(Entry<String, Map<String, List<User>>> userEntry : userMap.entrySet()){

Map<String, List<User>> itemMap = userEntry.getValue();

String userId = userEntry.getKey();

Map<String, Double> itemScoreMap = new HashMap<String, Double>();

for(Entry<String, List<User>> itemEntry : itemMap.entrySet()){

String itemId = itemEntry.getKey();

List<User> users = itemEntry.getValue();

double weight = 0.0;

int maxType = 0;

for(User user : users){

if (user.getBehaviorType() > maxType) {

maxType = user.getBehaviorType();

}

}

int count = users.size();

if (maxType != 0) {

weight += w[maxType-1];

}

weight += count;

itemScoreMap.put(itemId, weight);

}

scoreTable.put(userId, itemScoreMap);

}

return scoreTable;

}

public static double calculateWeight(int behaviorType, int count) {

double weight = w[behaviorType-1] + count;

return weight;

}

public static List<User> reduceUserByItem(List<User> userList) {

List<User> list = new ArrayList<User>();

Map<String, User> userMap = new LinkedHashMap<String, User>();

for(User user : userList){

String itemId = user.getItemId();

if (!userMap.containsKey(itemId)) {

double weight = calculateWeight(user.getBehaviorType(), user.getCount());

user.setWeight(weight);

userMap.put(itemId, user);

list.add(user);

}else {

User temp = userMap.get(itemId);

if (temp.getBehaviorType() < user.getBehaviorType()) {

double weight = calculateWeight(user.getBehaviorType(), user.getCount());

user.setWeight(weight);

userMap.put(itemId, user);

list.add(user);

}

}

}

userMap.clear();

return list;

}

public static void sortScoreMap(Map<String, List<Score>> scoreMap) {

Set<String> userSet = scoreMap.keySet();

for(String userId : userSet){

List<Score> temp = scoreMap.get(userId);

Collections.sort(temp);

scoreMap.put(userId, temp);

}

}

public static Map<String, Set<String>> predict(Map<String, List<Score>> scoreMap, List<String> fileNameList, String userDir,int topNUser,int topNItem) throws Exception {

Map<String, Set<String>> recommendList = new HashMap<String, Set<String>>();

for(Entry<String, List<Score>> entry : scoreMap.entrySet()){

String userId1 = entry.getKey();

List<Score> list = entry.getValue();

int countUser = 0;

Set<String> predictItemSet = new LinkedHashSet<String>();

for(Score score : list){

String userId2 = score.getItemId();

if(fileNameList.contains(userId2)){

List<User> userList = FileTool.readFileOne(userDir + userId2, false, "\t", "user");

int countItem = 0;

for(User user : userList){

predictItemSet.add(user.getItemId());

countItem++;

if (countItem == topNItem) {

break;

}

}

countUser++;

}

if (countUser == topNUser) {

break;

}

}

recommendList.put(userId1, predictItemSet);

}

return recommendList;

}

public static void prediction(Map<String, List<String>> predictMap,int predictN, Map<String, List<String>> referenceMap, int refN) {

int count = 0;

for(Entry<String, List<String>> predictEntity : predictMap.entrySet()){

String userId = predictEntity.getKey();

if (referenceMap.containsKey(userId)) {

List<String> predictList = predictEntity.getValue();

for(String itemId : predictList){

if (referenceMap.get(userId).contains(itemId)) {

count++;

}

}

}

}

double precision = (1.0 \* count / predictN) \* 100;

double recall = (1.0 \* count / refN) \* 100;

double f1 = (2 \* precision \* recall)/(precision + recall);

System.out.println("precision="+precision+",recall="+recall+",f1="+f1);

}

}

1. 计算模块

package service;

public class CalculateSimilarity {

public static double EuclidDist(Map<String, Double> userMap1,

Map<String, Double> userMap2, Set<String> userSet,

Set<String> itemSet) {

double sum = 0;

for (String itemId : itemSet) {

double score1 = 0.0;

double score2 = 0.0;

if (userMap1.containsKey(itemId) && userMap2.containsKey(itemId)) {

score1 = userMap1.get(itemId);

score2 = userMap2.get(itemId);

} else if (userMap1.containsKey(itemId)) {

score1 = userMap1.get(itemId);

} else if (userMap2.containsKey(itemId)) {

score2 = userMap2.get(itemId);

}

double temp = Math.pow((score1 - score2), 2);

sum += temp;

}

sum = Math.sqrt(sum);

return sum;

}

public static double CosineDist(Map<String, Double> userMap1,

Map<String, Double> userMap2, Set<String> userSet,

Set<String> itemSet) {

double dist = 0;

double numerator = 0; // 分子

double denominator1 = 0; // 分母

double denominator2 = 0; // 分母

for (String itemId : itemSet) {

double score1 = 0.0;

double score2 = 0.0;

if (userMap1.containsKey(itemId) && userMap2.containsKey(itemId)) {

numerator++;

score1 = userMap1.get(itemId);

score2 = userMap2.get(itemId);

} else if (userMap1.containsKey(itemId)) {

score1 = userMap1.get(itemId);

} else if (userMap2.containsKey(itemId)) {

score2 = userMap2.get(itemId);

}

denominator1 += Math.pow(score1, 2);

denominator2 += Math.pow(score2, 2);

}

dist = ((1.0 \* numerator) / (Math.sqrt(denominator1) \* Math

.sqrt(denominator2)));

return dist;

}

public static double execute(Map<String, Double> userMap1,Map<String, Double> userMap2,Set<String> userSet,Set<String> itemSet) {

double dist = EuclidDist(userMap1, userMap2, userSet, itemSet);

double userScore = 1.0 / (1.0 + dist);

// double userScore = CosineDist(userMap1, userMap2, userSet, itemSet);

return userScore;

}

//由于数据已经量化处理并进行脱敏处理，因此这里选用欧式距离计算相似度即可

public static void execute(String userId,Map<String, Map<String, Double>> scoreTable,

Set<String> userSet, Set<String> itemSet) {

for (Entry<String, Map<String, Double>> userEntry : scoreTable.entrySet()) {

String userId2 = userEntry.getKey();

Map<String, Double> userMap2 = userEntry.getValue();

double dist = EuclidDist(scoreTable.get(userId), userMap2, userSet, itemSet);

double userScore = 1.0 / (1.0 + dist);

// double userScore = CosineDist(userMap1, userMap2, userSet, itemSet);

FileTool.ps1.println(userId + "\t" + userId2 + "\t" + userScore);

}

}

public static void execute(Map<String, Map<String, Double>> scoreTable,

Set<String> userSet, Set<String> itemSet) {

List<Score> similarList = new ArrayList<Score>();

for (Entry<String, Map<String, Double>> userEntry1 : scoreTable.entrySet()) {

String userId = userEntry1.getKey();

execute(userId, scoreTable, userSet, itemSet);

}

}

}

1. 脚本

生成userset和itemset:

package script;

public class MakeSet {

public static void main(String[] args) throws Exception {

String inputDir = args[0];

String outputDir = args[1];

Set<String> userSet = new HashSet<String>();

Set<String> itemSet = new HashSet<String>();

List<String> pathList = FileTool.traverseFolder(inputDir);

for(String path : pathList){

String inputPath = inputDir + path;

List<User> list = FileTool.readFileOne(inputPath, false, "\t", "user");

for(User user : list){

userSet.add(user.getUserId());

itemSet.add(user.getItemId());

}

}

FileTool.initWriter1(outputDir+"userSet");

for(String userId : userSet){

FileTool.ps1.println(userId);

}

FileTool.closeWriter1();

FileTool.initWriter1(outputDir+"itemSet");

for(String itemId : itemSet){

FileTool.ps1.println(itemId);

}

FileTool.closeWriter1();

}

}

map文件构建user-item评分矩阵并计算user间的相似度生成user-user的得分表:

package script;

public class SpliteFileAndMakeScoreTable {

public static void main(String[] args) throws Exception {

//String inputDir = "data/fresh\_comp\_offline/";

//String outputDir = "data/fresh\_comp\_offline/sample/";

//String inputDir = "data/fresh\_comp\_offline/sample/";

//String outputDir = "data/fresh\_comp\_offline/sample/out/";

String inputDir = args[0];

String outputDir = args[1];

//String userPath = inputDir + "tianchi\_fresh\_comp\_train\_user.csv";

String userPath = inputDir + args[2];

String outputPath = args[3];

//String outputPath = outputDir + "user.csv";

//FileTool.makeSampleData(userPath, true, outputPath, 10000);

//List<Object> itemList = FileTool.readFileOne(itemPath, true, ",", "item");

//List<User> userList = FileTool.readFileOne(userPath, false, ",", "user");

List<User> userList = FileTool.readFileOne(userPath, false, ",", "user");

Set<String> userSet = new HashSet<String>();

Set<String> itemSet = new HashSet<String>();

Map<String, Map<String, List<User>>> userMap = DataProcess.mapByUser(userList,userSet,itemSet);

userList.clear();

DataProcess.output(userMap, outputDir);

//生成userToItem的打分表

Map<String, Map<String, Double>> scoreTable = DataProcess.makeScoreTable(userMap);

//DataProcess.output(scoreTable, outputDir + "scoreTable.csv" , userSet, itemSet, ",");

userMap.clear();

FileTool.initWriter1(outputPath);

CalculateSimilarity.execute(scoreTable, userSet, itemSet);

FileTool.closeWriter1();

}

}

reduce文件:

package script;

public class ReduceFileTest {

public static void main(String[] args) throws Exception {

//String inputDir = "data/fresh\_comp\_offline/";

//String outputDir = "data/fresh\_comp\_offline/sample/";

//String inputDir = "data/fresh\_comp\_offline/sample/";

//String outputDir = "data/fresh\_comp\_offline/sample/out/";

String inputDir = args[0];

String outputDir = args[1];

//String userPath = inputDir + "tianchi\_fresh\_comp\_train\_user.csv";

//String itemPath = inputDir + args[2];

//String userPath = inputDir + args[3];

List<String> pathList = FileTool.traverseFolder(inputDir);

for(String path : pathList){

List<User> userList = FileTool.readFileOne(inputDir+path, false, "\t", "user");

List<User> list = DataProcess.reduceUserByItem(userList);

userList.clear();

FileTool.initWriter1(outputDir + path);

Collections.sort(list);

DataProcess.outputUser(list);

FileTool.closeWriter1();

list.clear();

}

}

}

为用户进行推荐，生成预测列表:

package script;

public class PredictTest {

public static void main(String[] args) throws Exception {

//String inputDir = "data/fresh\_comp\_offline/";

//String outputDir = "data/fresh\_comp\_offline/sample/";

//String inputDir = "data/fresh\_comp\_offline/sample/";

//String outputDir = "data/fresh\_comp\_offline/sample/out/";

String inputDir = args[0];

String outputDir = args[1];

//String userPath = inputDir + "tianchi\_fresh\_comp\_train\_user.csv";

String inputPath = inputDir + args[2];

String outputPath = inputDir + args[3];

String userDir = args[4];

Map<String, List<Score>> scoreMap = FileTool.loadScoreMap(inputPath, false, "\t");

DataProcess.sortScoreMap(scoreMap);

List<String> fileNameList = FileTool.traverseFolder(userDir);

Map<String, Set<String>> predictMap = DataProcess.predict(scoreMap, fileNameList, userDir, 5, 5);//我选择推荐前5个user的前5个item

FileTool.initWriter1(outputPath);

DataProcess.outputRecommendList(predictMap);

FileTool.closeWriter1();

scoreMap.clear();

}

}

计算准确率、召回率、F测度值:

package script;

public class MatchTest2 {

public static void main(String[] args) throws Exception {

String inputDir = args[0];

String inputPath1 = inputDir + args[1];

String userDir = args[2];

Map<String, List<String>> predictMap = FileTool.loadPredictData(inputPath1, false, ",");

int predictN = FileTool.count;

System.out.println(predictN);

FileTool.count = 0;

Map<String, List<String>> referenceMap = FileTool.loadTestData(predictMap, userDir, false, "\t");

int referenceN = FileTool.count;

System.out.println(referenceN);

DataProcess.prediction(predictMap, predictN, referenceMap, referenceN);

}

}