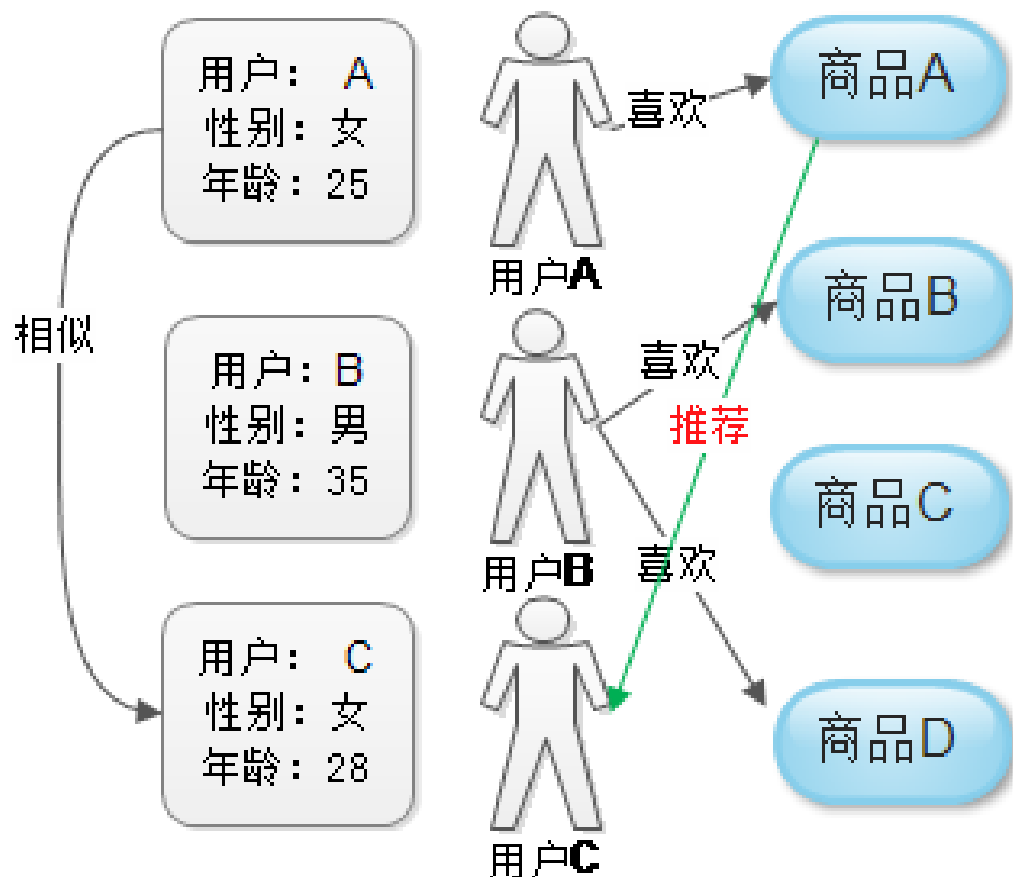


第九课 Spark ML协同过滤推荐算法

- 1、协同过滤推荐算法
- 2、ML协同过滤分布式实现逻辑
- 3、ML协同过滤源码开发
- 4、实现实例



协同过滤推荐算法，是最经典、最常用的推荐算法。通过分析用户兴趣，在用户群中找到指定用户的相似用户，综合这些相似用户对某一信息的评价，形成系统关于该指定用户对此信息的喜好程度预测。

要实现协同过滤，需要以下几个步骤：

- 1) 收集用户偏好；
- 2) 找到相似的用户或物品；
- 3) 计算推荐。

■ 从用户的行为和偏好中发现规律，并基于此进行推荐，所以收集用户的偏好信息成为系统推荐效果最基础的决定因素。用户有很多方式向系统提供自己的偏好信息，比如：评分、投票、转发、保存书签、购买、点击流、页面停留时间等。

■ 1. 将不同的行为分组

一般可以分为查看和购买，然后基于不同的用户行为，计算不同用户或者物品的相似度。

■ 2. 对不同行为进行加权

对不同行为产生的用户喜好进行加权，然后求出用户对物品的总体喜好。当我们收集好用户的行为数据后，还要对数据进行预处理，最核心的工作就是减噪和归一化。

表 14-1 用户评分表

用户/物品	物品A	物品B	物品C
用户A	0.1	0.8	1
用户B	0.1	0	0.02
用户C	0.5	0.3	0.1

- 对用户的行为分析得到用户的偏好后，可以根据用户的偏好计算相似用户和物品，然后可以基于相似用户或物品进行推荐。这就是协同过滤中的**两个分支了，即基于用户的协同过滤和基于物品的协同过滤。**

	物品1	物品2	物品3	物品4	物品5	物品6
用户A	1	1	0	0	0	1
用户B	1	0	1	1	0	0
用户C	1	1	1	0	0	0
用户D	0	0	1	1	1	0
用户E	0	0	0	1	1	1
用户F	1	0	1	0	1	0

1. 同现相似度

物品 i 和物品 j 的同现相似度公式定义：

$$w_{i,j} = \frac{|N(i) \cap N(j)|}{|N(i)|}$$

其中，分母 $|N(i)|$ 是喜欢物品 i 的用户数，而分子 $|N(i) \cap N(j)|$ 是同时喜欢物品 i 和物品 j 的用户数据。因此，上述公式可以理解为喜欢物品 i 的用户中有多少比例的用户也喜欢物品 j 。

上述公式存在一个问题，如果物品 j 是热门物品，很多人都喜欢，那么 $w_{i,j}$ 就会很大，接近 1。因此，该公式会造成任何物品都会和热门物品有很大的相似度。为了避免推荐出热门的物品，可以用如下公式：

$$w_{i,j} = \frac{|N(i) \cap N(j)|}{\sqrt{|N(i)| |N(j)|}}$$

这个公式惩罚了物品 j 的权重，因此减轻了热门物品与很多物品相似的可能性。

2. 欧氏距离 (Euclidean Distance)

最初用于计算欧几里得空间中两个点的距离，假设 x 、 y 是 n 维空间的两个点，它们之间的欧几里得距离是：

$$d(x, y) = \sqrt{\sum (x_i - y_i)^2}$$


可以看出，当 $n=2$ 时，欧几里得距离就是平面上两个点的距离。

当用欧几里得距离表示相似度时，一般采用以下公式进行转换：距离越小，相似度越大。

$$\text{sim}(x, y) = \frac{1}{1 + d(x, y)}$$

- 3. 皮尔逊相关系数 (Pearson Correlation Coefficient)
- 4. Cosine 相似度 (Cosine Similarity)
- 5. Tanimoto 系数 (Tanimoto Coefficient)

- 1. 基于用户的CF (User CF)
- 基于用户的 CF 的基本思想相当简单：基于用户对物品的偏好找到相邻的邻居用户，然后将邻居用户喜欢的推荐给当前用户。在计算上，就是将一个用户对所有物品的偏好作为一个向量来计算用户之间的相似度，找到K 邻居后，根据邻居的相似度权重及其对物品的偏好，预测当前用户没有偏好的未涉及物品，计算得到一个排序的物品列表作为推荐。图14-1 给出了一个例子，对于用户A，根据用户的历史偏好，这里只计算得到一个邻居-用户C，然后将用户C 喜欢的物品D 推荐给用户A。



用户/物品	物品A	物品B	物品C	物品D
用户A	✓		✓	推荐
用户B		✓		
用户C	✓		✓	✓

- 2. 基于物品的CF (Item CF)
- 基于物品的CF 的原理和基于用户的CF 类似，只是在计算邻居时采用物品本身，而不是从用户的角度。即基于用户对物品的偏好找到相似的物品，然后根据用户的历史偏好，推荐相似的物品给他。从计算的角度看，就是将所有用户对某个物品的偏好作为一个向量来计算物品之间的相似度，得到物品的相似物品后，根据用户历史的偏好预测当前用户还没有表示偏好的物品，计算得到一个排序的物品列表作为推荐。

用户/物品	物品A	物品B	物品C
用户A	✓		✓
用户B	✓	✓	✓
用户C	✓		推荐



图 14-2 基于物品的 CF 的基本原理

- 根据用户评分矩阵采用同现相似度计算物品相似度矩阵。

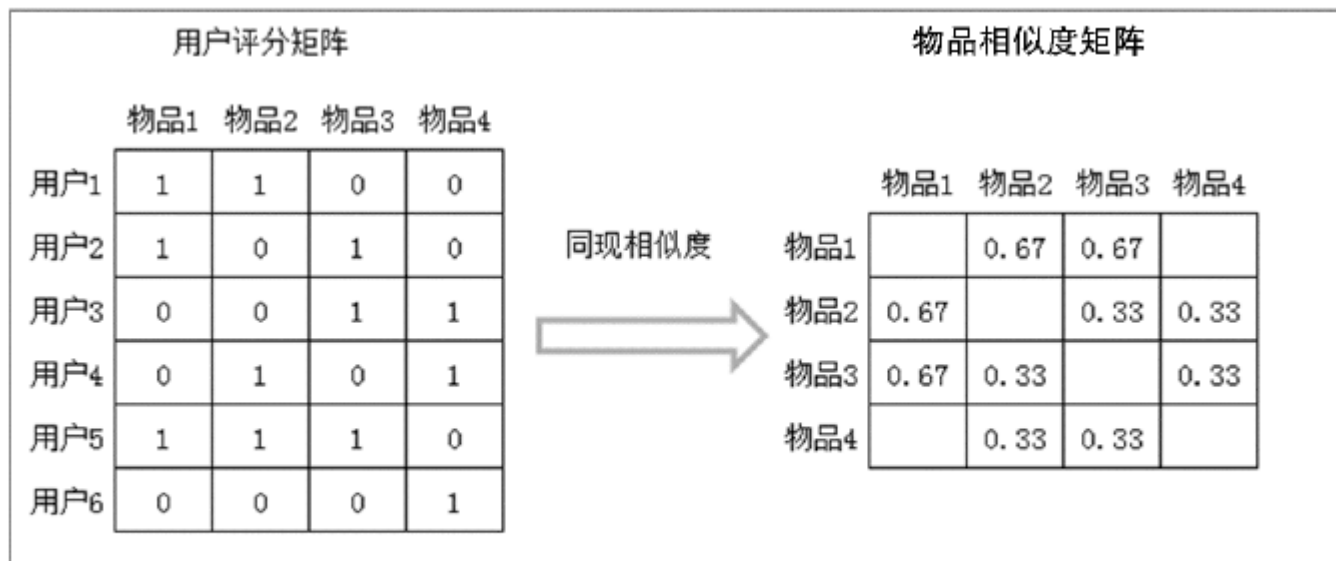


图 14-3 物品相似度矩阵

- 其相似度计算实现了分布式计算，实现过程如下：

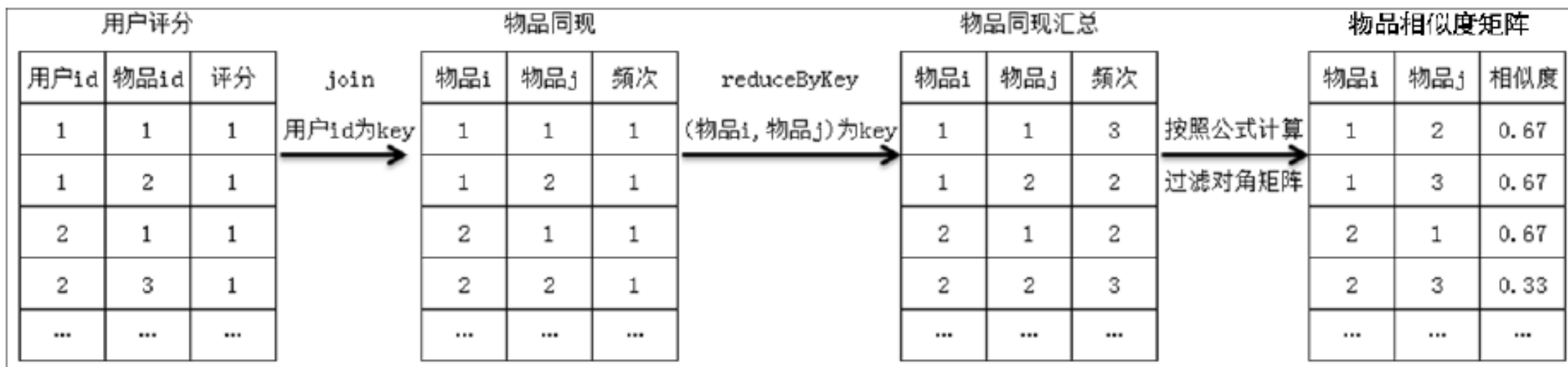


图 14-4 分布式同现相似度矩阵计算过程

协同推荐算法实现

- 对于欧氏相似度的计算，采用离散计算公式 $d(x, y) = \sqrt{\sum((x(i)-y(i)) * (x(i)- y(i))))}$ 。其中， i 只取 x 、 y 同现的点，未同现的点不参与相似度计算； $\text{sim}(x, y) = m / (1 + d(x, y))$ ， m 为 x 、 y 重叠数，同现次数



图 14-5 分布式欧氏距离相似度矩阵计算过程

- 根据物品相似度矩阵和用户评分计算用户推荐列表，计算公式是 $R=W*A$ ，取推荐计算中用户未评分过的物品，并且按照计算结果倒序推荐给用户。

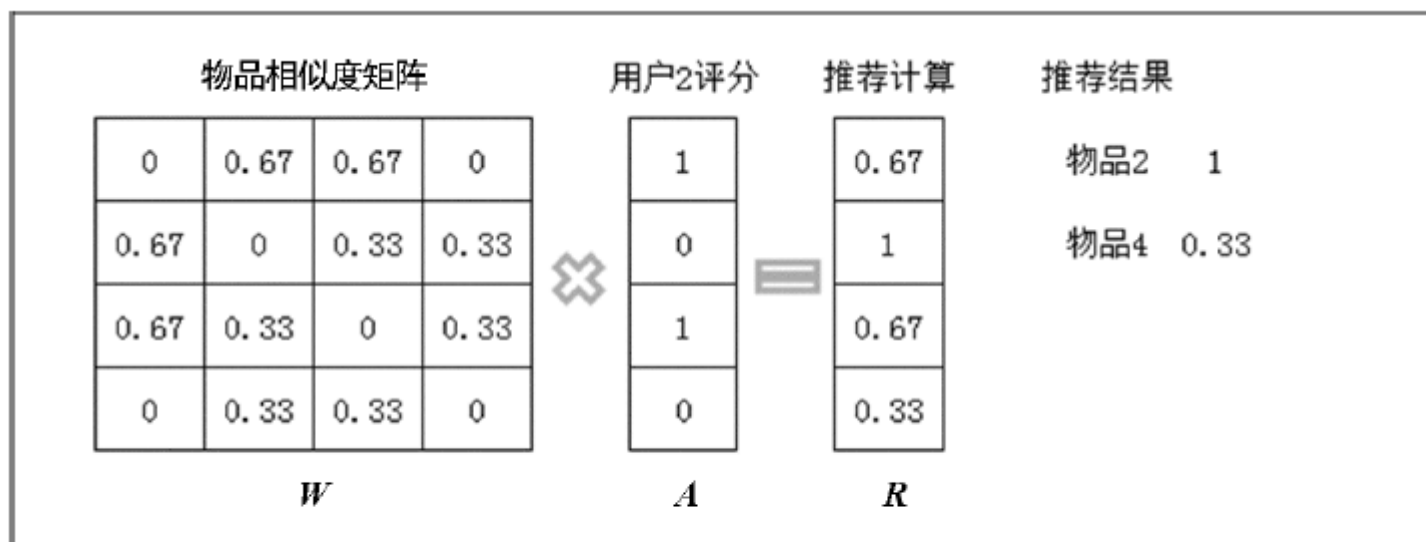


图 14-6 协同推荐计算

协同推荐算法实现

- 其推荐计算实现了分布式计算

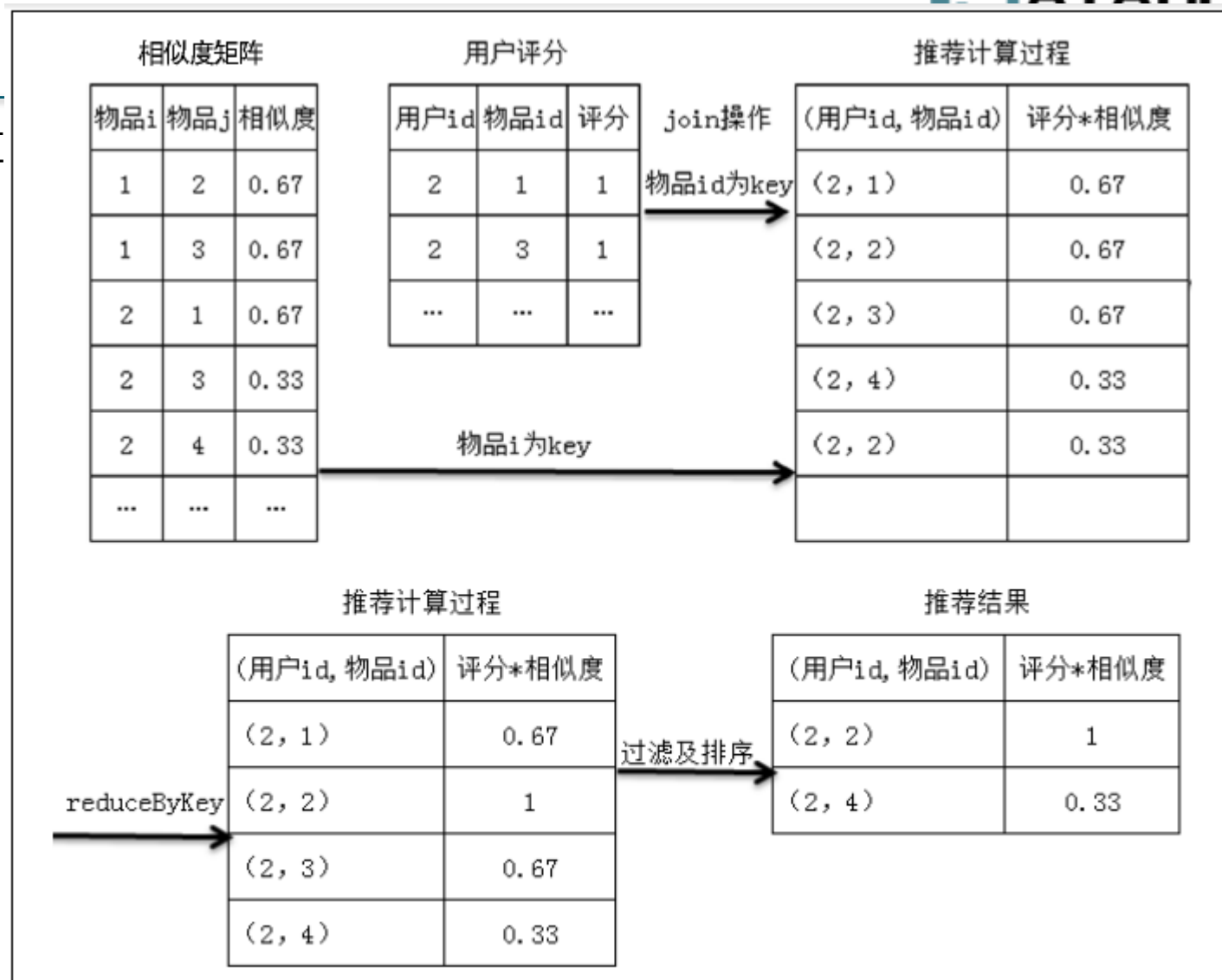


图 14-7 分布式协同推荐计算过程

ML 源码实现

```
/**  
 * 用户评分.  
 * @param userid 用户  
 * @param itemid 评分物品  
 * @param pref 评分  
 */  
case class ItemPref(  
    val userid: String,  
    val itemid: String,  
    val pref: Double)
```



```
/**  
 * 相似度.  
 * @param itemidI 物品  
 * @param itemidJ 物品  
 * @param similar 相似度  
 */  
case class ItemSimi(  
    val itemidI: String,  
    val itemidJ: String,  
    val similar: Double)
```

```
/**  
 * 推荐结果.  
 * @param userid 用户  
 * @param itemid 物品  
 * @param pref 得分  
 */  
case class UserRecomm(  
    val userid: String,  
    val itemid: String,  
    val pref: Double)
```

```
/**
```

```
 * 同现相似度矩阵计算.
```

```
 *  $w(i,j) = N(i)nN(j)/\sqrt{N(i)*N(j)}$ 
```

```
 * @param user_rdd 用户评分
```

```
 * @param RDD[ItemSimi] 返回物品相似度
```

```
 *
```

```
 */
```

```
def CooccurrenceSimilarityV1(user_ds: org.apache.spark.sql.Dataset[ItemPref]):
```

```
org.apache.spark.sql.Dataset[ItemSimi] = {
```

```
// 0 数据做准备
val user_ds_i = user_ds.withColumnRenamed("itemid", "itemidI").withColumnRenamed("pref",
"prefI")
val user_ds_j = user_ds.withColumnRenamed("itemid", "itemidJ").withColumnRenamed("pref",
"prefJ")
// 1 (用户 : 物品) 笛卡尔积 (用户 : 物品) => 物品:物品组合
val user_ds1 = user_ds_i.join(user_ds_j, "userid")
val user_ds2 = user_ds1.withColumn("score", col("prefI") * 0 + 1).select("itemidI",
"itemidJ", "score")
// 2 物品:物品:频次
val user_ds3 = user_ds2.groupBy("itemidI", "itemidJ").agg(sum("score").as("sumIJ"))
// 3 对角矩阵
val user_ds4 = user_ds3.where("itemidI = itemidJ")
// 4 非对角矩阵
val user_ds5 = user_ds3.filter("itemidI != itemidJ")
```

```
// 5 计算同现相似度 ( 物品1 , 物品2 , 同现频次 )
val user_ds6 = user_ds5.join(user_ds4.withColumnRenamed("sumIJ", "sumJ").select("itemidJ",
"sumJ"), "itemidJ")

val user_ds7 = user_ds6.join(user_ds4.withColumnRenamed("sumIJ", "sumI").select("itemidI",
"sumI"), "itemidI")

val user_ds8 = user_ds7.withColumn("result", col("sumIJ") / sqrt(col("sumI") *
col("sumJ")))

// 6 结果返回
val out = user_ds8.select("itemidI", "itemidJ", "result").map { row =>
    val itemidI = row.getString(0)
    val itemidJ = row.getString(1)
    val similar = row.getDouble(2)
    ItemSimi(itemidI, itemidJ, similar)
}
```

```
/**
 * 计算推荐结果.
 *  $w(i,j) = N(i)nN(j)/\sqrt{N(i)*N(j)}$ 
 * @param items_similar 物品相似矩阵
 * @param user_prefer 用户评分表
 * @param RDD[UserRecomm] 返回用户推荐结果
 *
 */
def Recommend(items_similar: org.apache.spark.sql.Dataset[ItemSimi],
               user_prefer: org.apache.spark.sql.Dataset[ItemPref]):
org.apache.spark.sql.Dataset[UserRecomm] = {
```

```
// 0 数据准备
val items_similar_ds1 = items_similar
val user_prefer_ds1 = user_prefer
// 1 根据用户的item召回相似物品
val user_prefer_ds2 = items_similar_ds1.join(user_prefer_ds1, $"itemidI" === $"itemid",
"inner")
// user_prefer_ds2.show()
// 2 计算召回的用户物品得分
val user_prefer_ds3 = user_prefer_ds2.withColumn("score", col("pref") *
col("similar")).select("userid", "itemidJ", "score")
// user_prefer_ds3.show()
// 3 得分汇总
val user_prefer_ds4 = user_prefer_ds3.groupBy("userid",
"itemidJ").agg(sum("score").as("score")).withColumnRenamed("itemidJ", "itemid")
// user_prefer_ds4.show()
```

```
// 4 用户得分排序结果
val user_prefer_ds5 = user_prefer_ds4
//      user_prefer_ds5.show()
// 5 结果返回
val out1 = user_prefer_ds5.select("userid", "itemid", "score").map { row =>
    val userid = row.getString(0)
    val itemid = row.getString(1)
    val pref = row.getDouble(2)
    UserRecomm(userid, itemid, pref)
}
//      out1.orderBy($"userid", $"pref".desc).show
out1
```


ML 实例

```
// 读取hive表数据
val hql1 = "select ... where"
val userDS1 = spark.sql(hql1)
userDS1.show(10)

val userDS2 = userDS1.map { row =>
  val itemidI = row.getString(0)
  val itemidJ = row.getString(1)
  val pref = row.getDouble(2)
  ItemPref(itemidI, itemidJ, pref)
}

val user_ds = userDS2
user_ds.show(10)
user_ds.cache()
user_ds.count()
```

user_ds: org.apache.spark.sql.Dataset[ItemPref] = [userid: string, itemid: string ... 1 more field]

userid	itemid	pref
000000008461832	135	0.673
000000013326178	183	0.673
000000013656319	141	0.9057
00000003554056	375	1.8114
000000038904512	183	0.9057
00000005824495	135	1.0

```
// 相似度计算v2
val startTime1 = System.currentTimeMillis()
val I2I = new ItemSimilarity(spark)
val items_similar = I2I.CooccurrenceSimilarityV2(user_ds)
items_similar.columns
//
items_similar.cache()
items_similar.count
val endTime1 = System.currentTimeMillis()
val computeTime1 = (endTime1 - startTime1).toDouble / 1000 / 60
```

```
startTime1: Long = 1500376430962
I2I: ItemSimilarity = ItemSimilarity@52992a77
items_similar: org.apache.spark.sql.Dataset[ItemSimi] = [itemidI: string, itemidJ: string ... 1 more field]
res10: Array[String] = Array(itemidI, itemidJ, similar)
res11: items_similar.type = [itemidI: string, itemidJ: string ... 1 more field]
res12: Long = 156
endTime1: Long = 1500376444966
computeTime1: Double = 0.2334
```

```

val id2name = udf { (itemid: String) =>
    val name = item_map(itemid)
    name
}
// 推荐结果计算
val startTime2 = System.currentTimeMillis()
val user_predict = I2I.Recommend(items_similar,user_ds)
user_predict.columns
val endTime2 = System.currentTimeMillis()
val computeTime2 = (endTime2 - startTime2).toDouble / 1000 / 60
user_predict.withColumn("Iname", id2name($"itemid")).orderBy($"userid".asc, $"pref".desc).show(100)

```

computeTime2: Double = 1.9609166666666666

userid itemid		pref		Iname
+gsr	373	0.1708830387852367		
+gsr	141	0.15337203544527345		
+gsn:	375	0.1525476849819516		
+gsr	183	0.11218752940820041		
+gs	331	0.11015248611711412		
+gs	255	0.10302986155740383		
+gs	371	0.10220212426540397		
+gs	181	0.1017828961481279		
+gs	369	0.09367638355283121		
+gsr	341	0.08278612192196945		
+gsn	251	0.0708352466592993		
+gsn	135	0.06959097815945206		
+gsn	373	0.12274383779421227		
+gsr	141	0.11376412282184747		

Thanks

FAQ时间