协同过滤推荐算法

作者：gongyouliu 微信号：liuq4360

来源公众号：**数据与智能**

作者在《推荐系统产品与算法概述》这篇文章中简单介绍了协同过滤算法。协同过滤算法是在整个推荐系统发展史上比较出名的算法，具备举足轻重的地位，甚至在当今还在大量使用。

本篇文章作者会详细讲解协同过滤推荐算法的方方面面，这里所讲的也是作者基于多年推荐系统研究及工程实践经验的基础上总结而成，希望对大家学习协同过滤推荐算法有所帮助，提供一些借鉴。

本文会从协同过滤思想简介、协同过滤算法原理介绍、离线协同过滤算法的工程实现、近实时协同过滤算法的工程实现、协同过滤算法应用场景、协同过滤算法的优缺点、协同过滤算法落地需要关注的几个问题等7个方面来讲述。希望读者读完本文，可以很好地理解协同过滤的思路、算法原理、工程实现方案，并且具备基于本文的思路自己独立实现一个在真实业务场景中可用的协同过滤推荐系统的能力。

在正式讲解之前，先做一个简单定义。本文用"**操作过**"这个词来表示用户对标的物的各种操作行为，包括浏览、点击、播放、收藏、评论、点赞、转发、评分等等。

一、协同过滤思想简介

协同过滤，从字面上理解，包括协同和过滤两个操作。所谓协同就是利用群体的行为来做决策(推荐)，生物上有协同进化的说法，通过协同的作用，让群体逐步进化到更佳的状态。对于推荐系统来说，通过用户的持续协同作用，最终给用户的推荐会越来越准。而过滤，就是从可行的决策(推荐)方案(标的物)中将用户喜欢的方案(标的物)找(过滤)出来。

具体来说，协同过滤的思路是通过群体的行为来找到某种相似性(用户之间的相似性或者标的物之间的相似性)，通过该相似性来为用户做决策和推荐。

现实生活中有很多协同过滤的案例及思想体现，除了前面提到的生物的进化是一种"协同过滤"作用外，我认为人类喜欢追求相亲中的"门当户对"，其实也是一种协同过滤思想的反射，门当户对实际上是建立了相亲男女的一种"相似度"(家庭背景、出身、生活习惯、为人处世、消费观、甚至价值观可能会相似)，给自己找一个门当户对的伴侣就是一种"过滤"，当双方"门当户对"时，各方面的习惯及价值观会更相似，未来幸福的概率也会更大。如果整个社会具备这样的传统和风气，以及在真实"案例"中"门当户对"的夫妻确实会更和谐，通过"协同进化"作用，大家会越来越认同这种方式。我个人也觉得"门当户对"是有一定道理的。

协同过滤利用了两个非常朴素的自然哲学思想："群体的智慧"和"相似的物体具备相似的性质"，群体的智慧从数学上讲应该满足一定的统计学规律，是一种朝向平衡稳定态发展的动态过程，越相似的物体化学及物理组成越一致，当然表现的外在特性会更相似。虽然这两个思想很简单，也很容易理解，但是正因为思想很朴素，价值反而非常大。所以协同过滤算法原理很简单，但是效果很不错，而且也非常容易实现。

协同过滤分为基于用户的协同过滤和基于标的物(物品)的协同过滤两类算法。下面我们对协同过滤的算法原理来做详细的介绍。

二、协同过滤算法原理介绍

上面一节简单介绍了协同过滤的思想，基于协同过滤的两种推荐算法，核心思想是很朴素的"物以类聚、人以群分"的思想。所谓物以类聚，就是计算出每个标的物最相似的标的物列表，我们就可以为用户推荐用户喜欢的标的物相似的标的物，这就是基于物品(标的物)的协同过滤。所谓人以群分，就是我们可以将与该用户相似的用户喜欢过的标的物的标的物推荐给该用户(而该用户未曾操作过)，这就是基于用户的协同过滤。具体思想可以参考下面的图1。

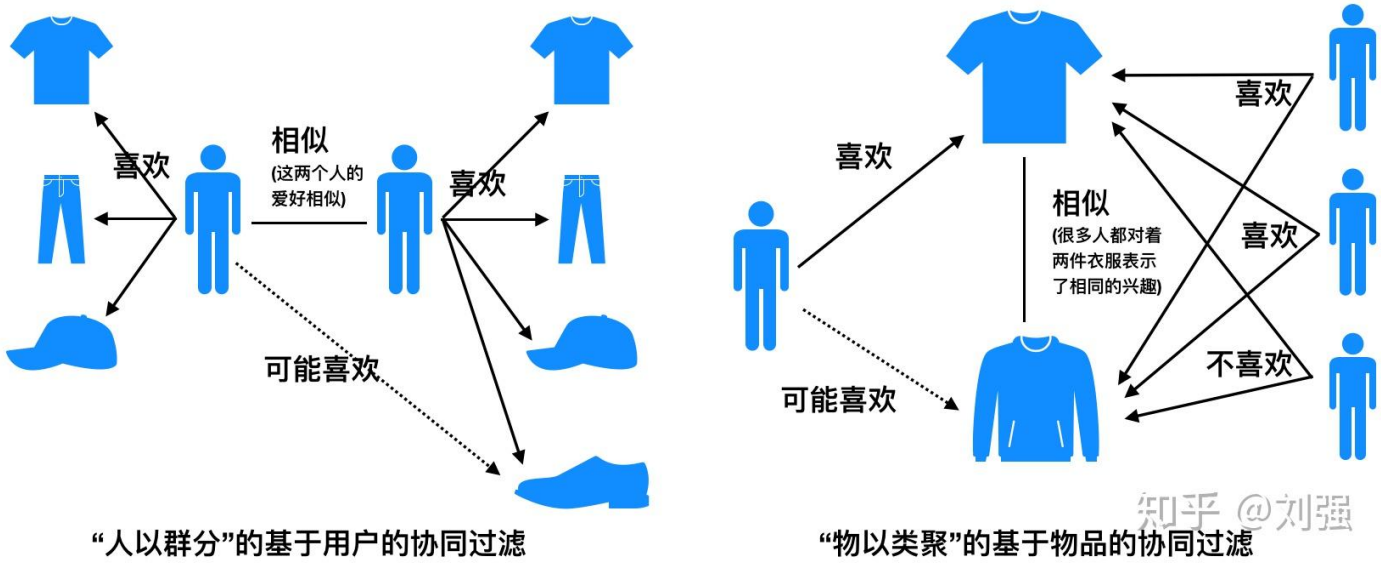


图1："物以类聚，人以群分"的朴素协同过滤推荐

协同过滤的核心是怎么计算标的物之间的相似度以及用户之间的相似度。我们可以采用非常朴素的思想来计算相似度。我们将用户对标的物的评分(或者隐式反馈，如点击、收藏等)构建如下用户行为矩阵(见下面图2)，矩阵的某个元素代表某个用户对某个标的物的评分(如果是隐式反馈，值为1)，如果某个用户对某个标的物未产生行为，值为0。其中行向量代表某个用户对所有标的物的评分向量，列向量代表所有用户对某个标的物的评分向量。有了行向量和列向量，我们就可以计算用户与用户之间、标的物与标的物之间的相似度了。具体来说，行向量之间的相似度就是用户之间的相似度，列向量之间的相似度就是标的物之间的相似度。

为了避免误解，这里简单解释一下隐式反馈，只要不是用户直接评分的操作行为都算隐式反馈，包括浏览、点击、播放、收藏、评论、点赞、转发等等。有很多隐式反馈是可以间接获得评分的，后面会讲解。如果不间接获得评分，就用0、1表示是否操作过。

在真实业务场景中用户数和标的物数一般都是很大的(用户数可能是百万、千万、亿级，标的物可能是十万、百万、千万级)，而每个用户只会操作过有限个标的物，所以用户行为矩阵是稀疏矩阵。正因为矩阵是稀疏的，会方便我们进行相似度计算及为用户做推荐。

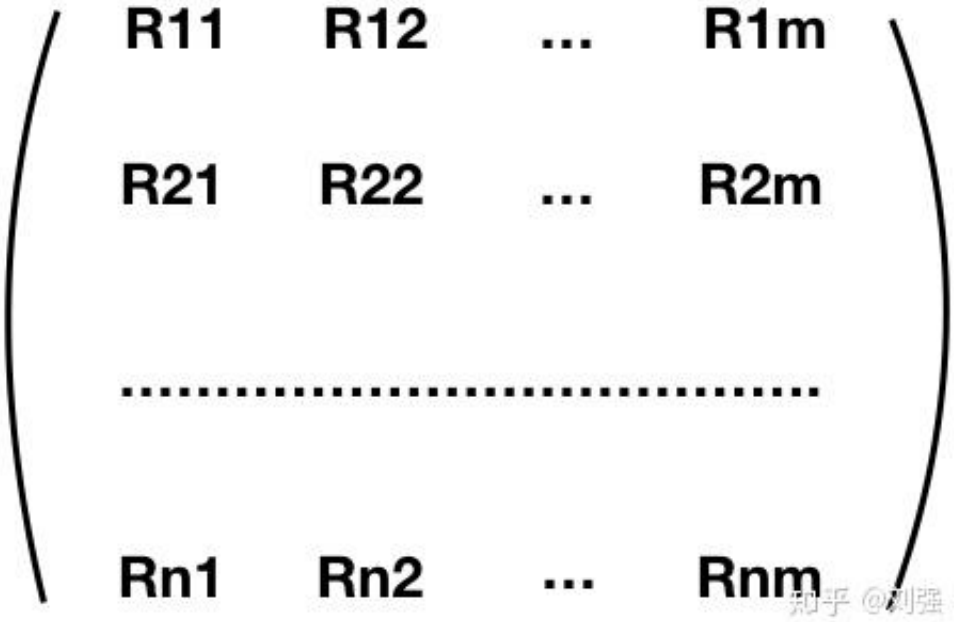


图2：用户对标的物的操作行为矩阵

相似度的计算可以采用cosine余弦相似度算法来计算两个向量

$$v_1, v_2$$

(可以是上图的中行向量或者列向量)之间的相似度：

$$sim(v_1, v_2) = \frac{v_1 * v_2}{\|v_1\| \times \|v_2\|}$$

计算完了用户(行向量)或者标的物(列向量)之间的相似度，那么下面说说怎

么为用户做个性化推荐。

1. 基于用户的协同过滤

根据上面算法思想的介绍，我们可以将与该用户最相似的用户喜欢的标的物推荐给该用户。这就是基于用户的协同过滤的核心思想。

用户u对标的物s的喜好度sim(u,s)可以采用如下公式计算，其中U是与该用户最相似的用户集合(我们可以基于用户相似度找到与某用户最相似的K个用户)，

$$score(u_i, s)$$

是用户

$$u_i$$

对标的物s的喜好度(对于隐式反馈为1，而对于非隐式反馈，该值为用户对标的物的评分)，

$$sim(u, u_i)$$

是用户

$$u_i$$

与用户u的相似度。

$$sim(u, s) = \sum_{u_i \in U} sim(u, u_i) * score(u_i, s)$$

有了用户对每个标的物的评分，基于评分降序排列，就可以取topN推荐给用户了。

1. 基于标的物的协同过滤

类似地，通过将用户操作过的标的物最相似的标的物推荐给用户，这就是基

于标的物的协同过滤的核心思想。

用户u对标的物s的喜好度sim(u,s)可以采用如下公式计算，其中S是所有用户操作过的标的物的列表，

$$score(u, s_i)$$

是用户u对标的物

$$s_i$$

的喜好度，

$$sim(s_i, s)$$

是标的物

$$s_i$$

与s的相似度。

$$sim(u, s) = \sum_{s_i \in S} score(u, s_i) * sim(s_i, s)$$

有了用户对每个标的物的评分，基于评分降序排列，就可以取topN推荐给用户了。

从上面的介绍中我们可以看到协同过滤算法思路非常直观易懂，计算公式也相对简单，并且后面两节我们也会说明它易于分布式实现，同时该算法也不依赖于用户及标的物的其他metadata信息。协同过滤算法被Netflix、Amazon等大的互联网公司证明效果也非常好，也能够为用户推荐新颖性内容，所以一直以来都在工业界得到非常广泛的应用。

三、离线协同过滤算法的工程实现

虽然协同过滤算法原理非常简单，但是在大规模用户及海量标的物的场景下，单机是难以解决计算问题的，我们必须借助分布式技术来实现，让整个

算法可以应对大规模数据的挑战。在本节，我们基于主流的Spark分布式计算平台相关的技术来详细讲解协同过滤算法的离线(批处理)实现思路，供大家参考(读者可以阅读参考文献1、2、3、4了解协同过滤算法原理及工业应用)，同时会在下一节讲解在近实时场景下怎么在工程上实现协同过滤算法。

在这里我们只讲解基于标的物的协同过滤算法的工程实现方案，基于用户的协同过滤思路完全一样，不再赘述。

为了简单起见，我们可以将推荐过程拆解为2个阶段，先计算相似度，再为用户推荐。下面分别介绍这两个步骤怎么工程实现。

1. 计算topK相似度

本步骤我们计算出任意两个标的物之间的相似度，有了任意两个标的物之间的相似度，那么我们就可以为每个标的物计算出与它最相似的K个标的物了。

假设有两个标的物

$$v_i, v_j$$

，它们对应的向量(即图2中矩阵的列向量，分别是第i列和第j列)如下，其中n是用户数。

$$v_i \to (R_{1i}, R_{2i}, ..., R_{ni}) = P_i$$
$$v_j \to (R_{1j}, R_{2j}, ..., R_{nj}) = P_j$$

那么

$$v_i, v_j$$

的相似度计算，我们可以细化如下：

$$sim(v_i, v_j) = \frac{P_i * P_j}{\|P_i\| * \|P_j\|} = \frac{\sum_{k=1}^{n} R_{ki} * R_{kj}}{\sqrt{\sum_{k=1}^{n} R_{ki} * R_{ki}} * \sqrt{\sum_{k=1}^{n} R_{kj} * R_{kj}}}$$

公式1：计算

$$v_i, v_j$$

相似度

我们仔细看一下上述公式，公式的分子就是下图矩阵中对应的i列和j列中同一行中的两个元素(红色矩形中的一对元素)相乘，并且将所有行上第i列和第j列的元素相乘得到的乘积相加(这里其实只需要考虑同一行对应的i列和j列的元素都非零的情况，如果只要第i列和第j列中有一个为零，乘积也为零)。公式中分母是第i行与第i行按照上面类似的方法相乘再相加后开根号的值，再乘以第j行与第j行按照上面类似的方法相乘再相加后开根号的值。

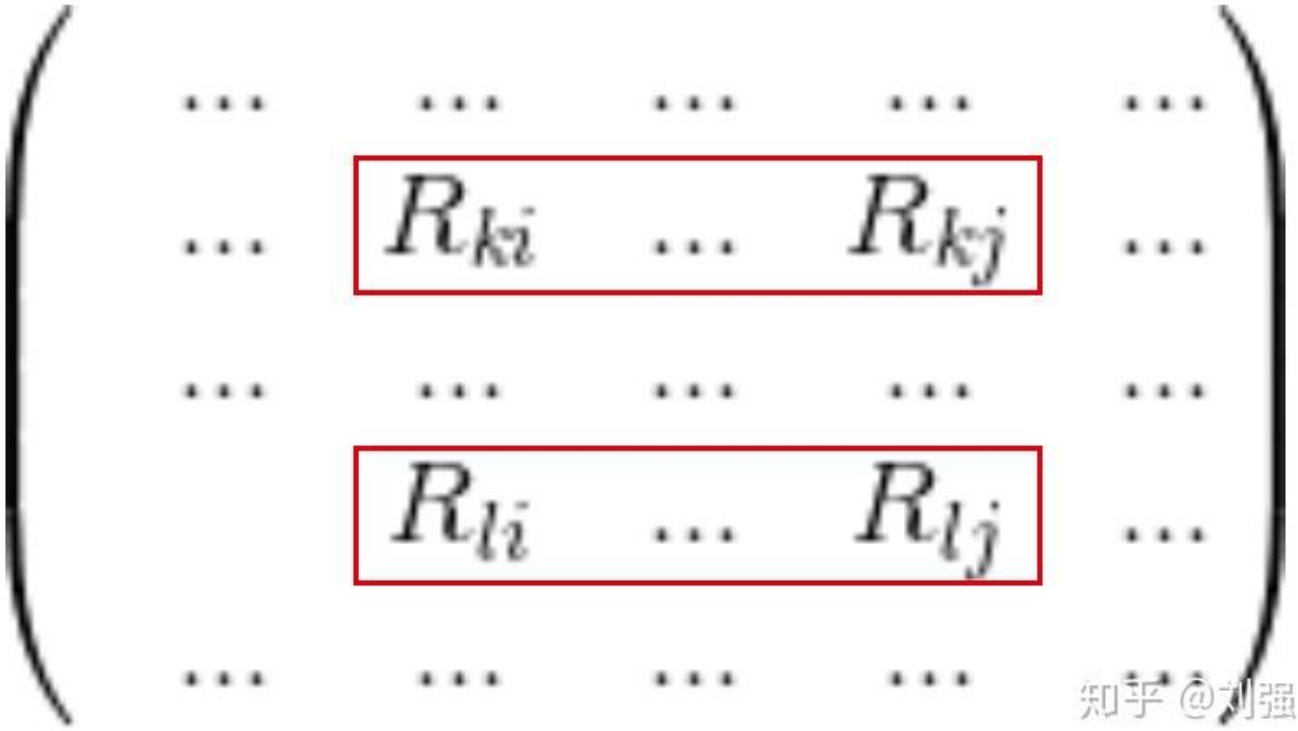


图3：计算两个列向量的cosine余弦可以拆解为简单的加减乘及开根号运算

有了上面的简单分析，就容易分布式计算相似度了。下面我们就来讲解，在Spark上怎么简单地计算每个标的物的topK相似度。在Spark上计算相似度，最主要的目标是怎么将上面巨大的计算量(前面已经提到在互联网公司，往往用户数和标的物数都是非常巨大的)通过分布式技术实现，这样就可以利用多台服务器的计算能力，解决大计算问题。

首先将所有用户操作过的标的物"收集"起来，形成一个用户行为RDD，具体的数据格式如下：

$$RDD[(uid, Seq[(sid, R)])]$$

其中uid是用户唯一识别编码，sid是标的物唯一识别编码，R是用户对标的物的评分(即矩阵中的元素)。

对于

$$RDD[(uid, Seq[(sid, R)])]$$

中的某个用户来说，他操作过的标的物

$$v_i$$

和

$$v_j$$

，一定在该用户所在的行对应的列i和列j的元素非零，根据上面计算

$$v_i, v_j$$

相似度的公式，需要将该用户对应的

$$v_i, v_j$$

的评分乘起来。这个过程可以用下面的图4来说明。

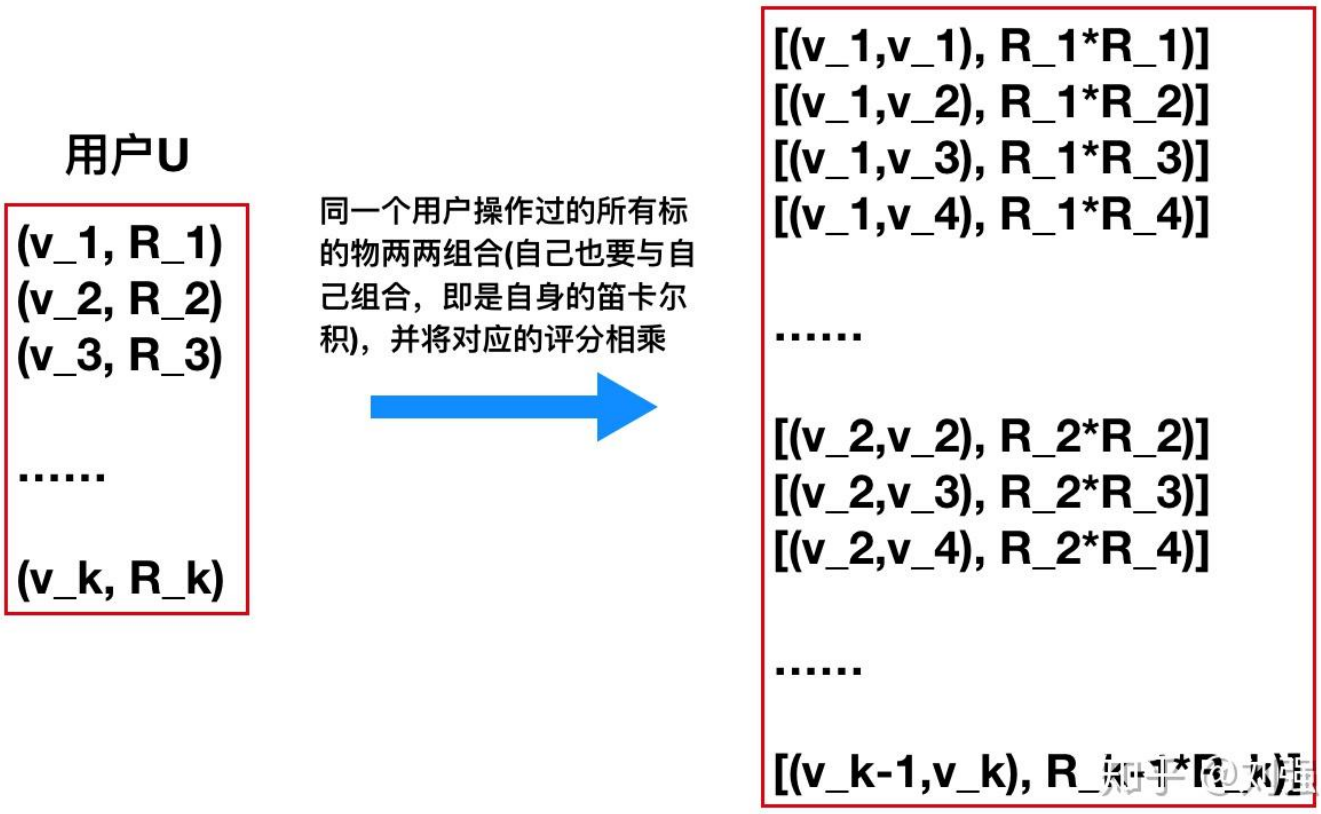


图4：对用户U来说，将他所有操作过的标的物做笛卡尔积

当所有用户都按照图4的方式转化为标的物对及得分(图4中右边的

$$R_i * R_j$$

)时，我们就可以对标的物对Group(聚合)，将相同的对合并，对应的得分相加，最终得到的RDD为：

$$S1 = RDD[((sid1, sid2), Score)]$$

这样，公式1中分子就计算出来了(上式中的Score即是公式1中的分子)。现在我们需要计算分母，这非常简单，只要从上面的RDD中将标的物sid1等于标的物sid2的列过滤出来就可以了，通过下图的操作，我们可以得到一个map

$$S_2$$

。

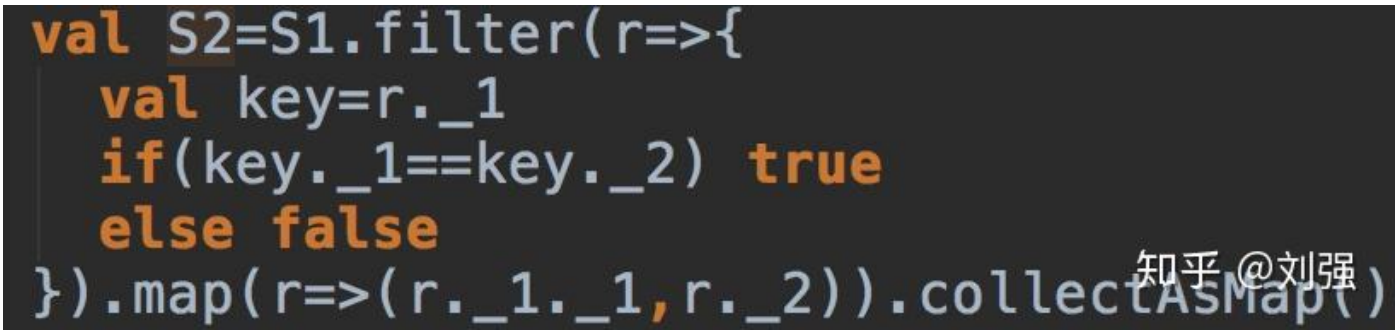


图5：从

$$S_1$$

中过滤出

$$sid1 = sid2$$

的元素，用于计算公式1中的分母

$$S_2$$

最多含有标的物的数量(m)个的元素，相对来说不大，我们可以将

$$S_2$$

广播(broadcast)出去。

$$S_2^{'} = sparkContext.broadcast(S_2)$$

方便我们按照公式1除以分母，最终得到

$$v_i, v_j$$

的相似度。

通过上面这些步骤，公式1中的分子和分母基本都很容易计算出来了，我们通过下图的代码(下面的broadcast即是

$$S_2^{'}$$

)，就可以计算出每组

$$v_i, v_j$$

对的相似度，最终得到的RDD为：

$$S = RDD[((sid1, sid2), Sim)]$$

，其中Sim为sid1和sid2的相似度。

```
val S = S1.filter(r=>{
  val key = r._1
  if(key._1==key._2) false
  else true
}).map(r=>{
  val k1= broadcast.value.get(r._1._1).get
  val k2= broadcast.value.get(r._1._2).get
  val k = r._2/Math.sqrt(k1*k2)
  ((r._1._1,r._1._2),k)
})
```


图6：计算每组

$$v_i, v_j$$

的相似度

有了上面的准备，下面我们来说明一下怎么计算每个标的物的topK最相似的标的物。

具体的计算过程可以用如下的Spark Transformation来实现。其中第三步的TopK需要我们自己实现一个函数，求

$$Seq[(sid, Score)]$$

这样的列表中评分最大的TopK个元素，实现也是非常容易的，这里不赘述。

$$
\begin{aligned}
S = RDD[((sid1, sid2), Sim)] &\overset{map}{\rightarrow} RDD[((sid1, (sid2, Sim))] \\
&\overset{groupByKey}{\rightarrow} RDD[((sid1, Seq[(sid2, Sim))]] \\
&\overset{TopK}{\rightarrow} RDD[((sid1, Seq[(sid2, Sim))]]
\end{aligned}
$$

如果我们把每个标的物最相似的K个标的物及相似度看成一个列向量的话，那么我们计算出的标的物相似度其实可以看作如下矩阵，该矩阵每列K个非零元素。

$$
S_{m\times m} = \begin{pmatrix}
s_{11} & s_{12} & \dots & s_{1m} \\
s_{21} & s_{22} & \dots & s_{2m} \\
\dots & \dots & \dots & \dots \\
s_{m1} & s_{m2} & \dots & s_{mm}
\end{pmatrix}_{m\times m}
$$

图7：标的物相似度矩阵

到此为止，我们通过Spark提供的一些Transformation操作及一些工程实现上的技巧计算出了每个标的物topK最相似的标的物。该计算方法可以横向拓展，所以再大的用户数和标的物数都可以轻松应对，最多可能需要多加一些服务器。

1. 为用户生成推荐

有了1中计算出的标的物topK最相似的标的物，下面我们来说明一下怎么为用户生成个性化推荐。生成个性化推荐有两种工程实现策略，一种是看成矩阵的乘积，另外一种是根据第二节2中"基于标的物的协同过滤"中的公式来计算，这两种方法本质上是一样的，只是工程实现上不一样。下面我们分别讲解这两种实现方案。

(1) 通过矩阵相乘为用户生成推荐

上面图2中的矩阵是用户行为矩阵，第i行第j列的元素代表了用户i对标的物j

的偏好/评分，我们将该矩阵记为

$$A_{n\times m}$$

，其中n是用户数，m是标的物数。图7中的矩阵是标的物之间的相似度矩阵，我们将它记为

$$S_{m\times m}$$

，这是一个方阵。

$$A_{n\times m}$$

和

$$S_{m\times m}$$

其实都是稀疏矩阵，我们通过计算这两个矩阵的乘积(Spark上是可以直接计算两个稀疏矩阵的乘积的)，最终的结果矩阵就可以方便用来为用户推荐了：

$$R_{n\times m} = A_{n\times m} * S_{m\times m}$$

。其中的第i行

$$(r_{i1}, r_{i2}, r_{i3}, ..., r_{im})$$

代表的是用户i对每个标的物的偏好得分，我们从这个列表中过滤掉用户操作过的标的物，然后按照得分从高到低降序排列取topN就是最终给用户的推荐。

(2) 通计算公式为用户生成推荐

标的物相似度矩阵

$$S_{m\times m}$$

是稀疏矩阵，最多

$$m \times K$$

个非零元素(因为每个标的物只保留K个最相似的标的物)，一般K取几十或者上百规模的数值，m如果是十万或者百万量级，存储空间在1G左右(例如，如果m=100万，K=100，相似度为双精度浮点数，那么

$$S_{m \times m}$$

非零元素占用的空间为100万*100*8Byte=763M)，完全可以通过广播的形式将

$$S_{m \times m}$$

broadcast到每个Spark计算节点中。我们先将相似矩阵转化为下图的Map结构，再广播出去，方便利用公式计算相似度。

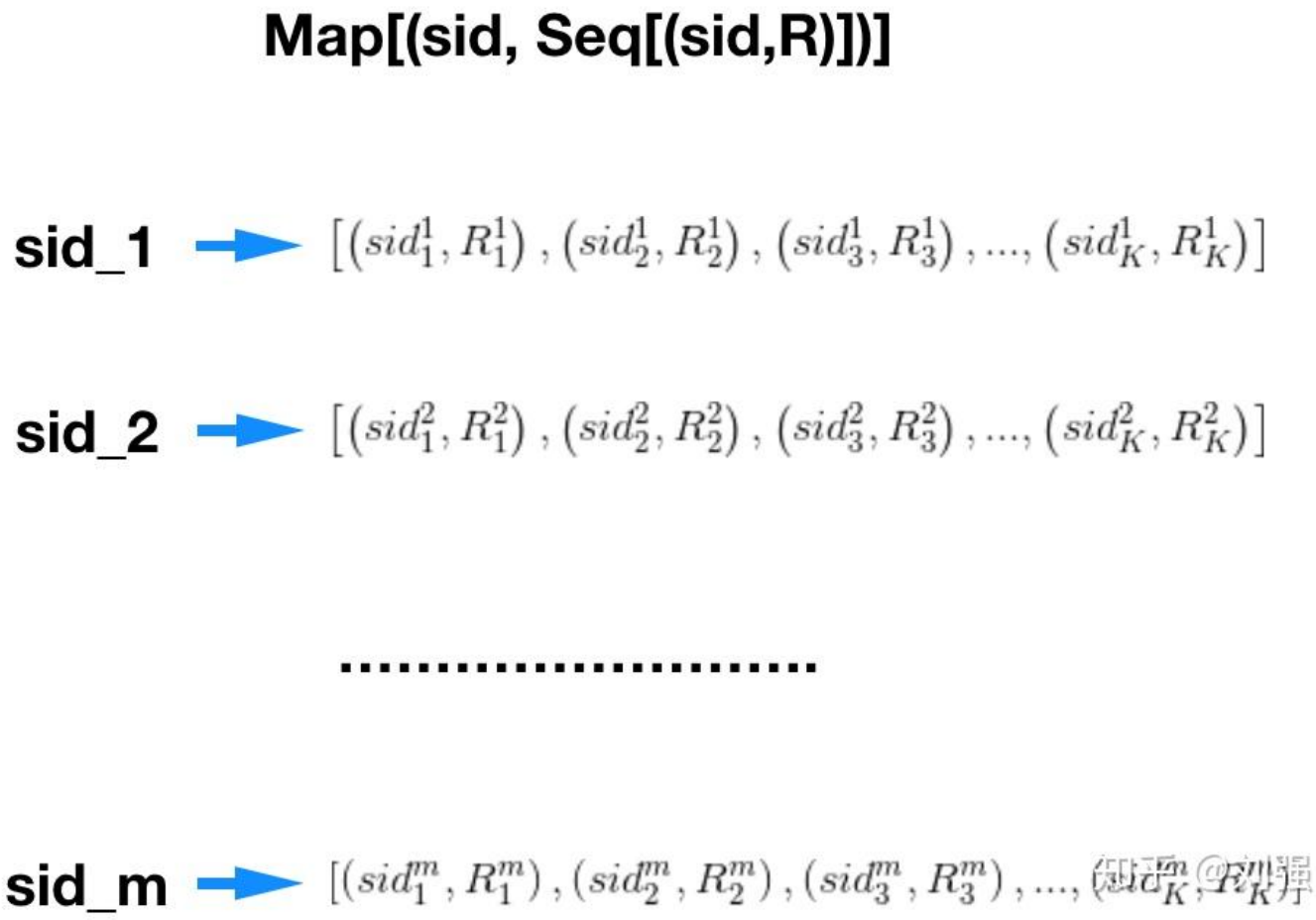


图8：标的物的topK相似列表利用Map数据结构来存储

有了标的物之间的相似度Map，为用户计算推荐的过程可以基于用户行为RDD，在每个Partition中，针对每个用户u计算u与每个标的物之间的偏好度(利用第二节2基于标的物的协同过滤中的公式)，再取topN就得到该用户的推荐结果了。由于用户行为采用了RDD来表示，所以整个计算过程可以分布式进行，每个Partition分布在一台服务器上进行计算。具体的计算逻辑可以用下面的代码片段来实现。

```
RDD[(uid,Seq[(sid,R)])].mapPartitions(partitions=> {

  val simMap = BroadcastSimMap.value //上面图8中相似节目Map
  val allObject = simMap.keySet.toArray // 所有的标的物Array

  partitions.map(r => {
    val user = r._1 //uid
    val acitonList = r._2.sortBy(e => e._1) //用户操作过的所有标的物
    val acitonAndScoreList = r._2 //用户操作过的标的物及评分List
                                  // [(sid1,score1),...,(sidk,scorek)]

    val predict = allObject.map(sid =>{
      val vidSim = simMap.get(sid).toMap//sid所有的K个相似标的物及评分Map
      var s = 0.0
      acitonAndScoreList.foreach(e=>{
        sid1  = e._1
        score = e._2
        s = s + score * vidoSim.getOrElse(sid1,0)
      })
      (sid,s)
    }) // 计算该用户对每个标的物的评分

    val rec = predict.sortBy(-_._2).take(N) // 将上面计算的评分降序排列并取topN
    (user,rec)
  })
})
```


图9：为每个用户计算topN推荐

讲到这里，基于Spark平台离线实现协同过滤算法的工程方案就讲完了。该实现方案强依赖于Spark的数据结构及分布式计算函数，可能在不同的计算平台上(比如Flink、Tensorflow等)具体的实现方式会不一样，但是基本思路和原理是一样的，有兴趣并且平时使用这些平台的读者可以在这些计算平台上独自实现一下，算是对自己的一个挑战。

四、近实时协同过滤算法的工程实现

上面第三节中的协同过滤工程实现方案适合做离线批量计算，比较适合标的物增长较缓慢的场景及产品(比如电商、视频、音乐等)，对于新闻、短视频这类增量非常大并且时效性强的产品(如今日头条、快手等)是不太合适的。那么我们是否可以设计出一套适合这类标的物快速增长的产品及场景下的协同过滤算法呢？实际上是可以的，下面我们来简单说一下怎么近实时实现简单的协同过滤算法。

我们的近实时协同过滤算法基于Kafka、HBase和Spark Streaming等分布式技术来实现，核心思想跟第三节中的类似，只不过我们这里是实时更新的，具体的算法流程及涉及到的数据结构见下面图10。下面我们对实现原理做简单介绍，整个推荐过程一共分为4步。

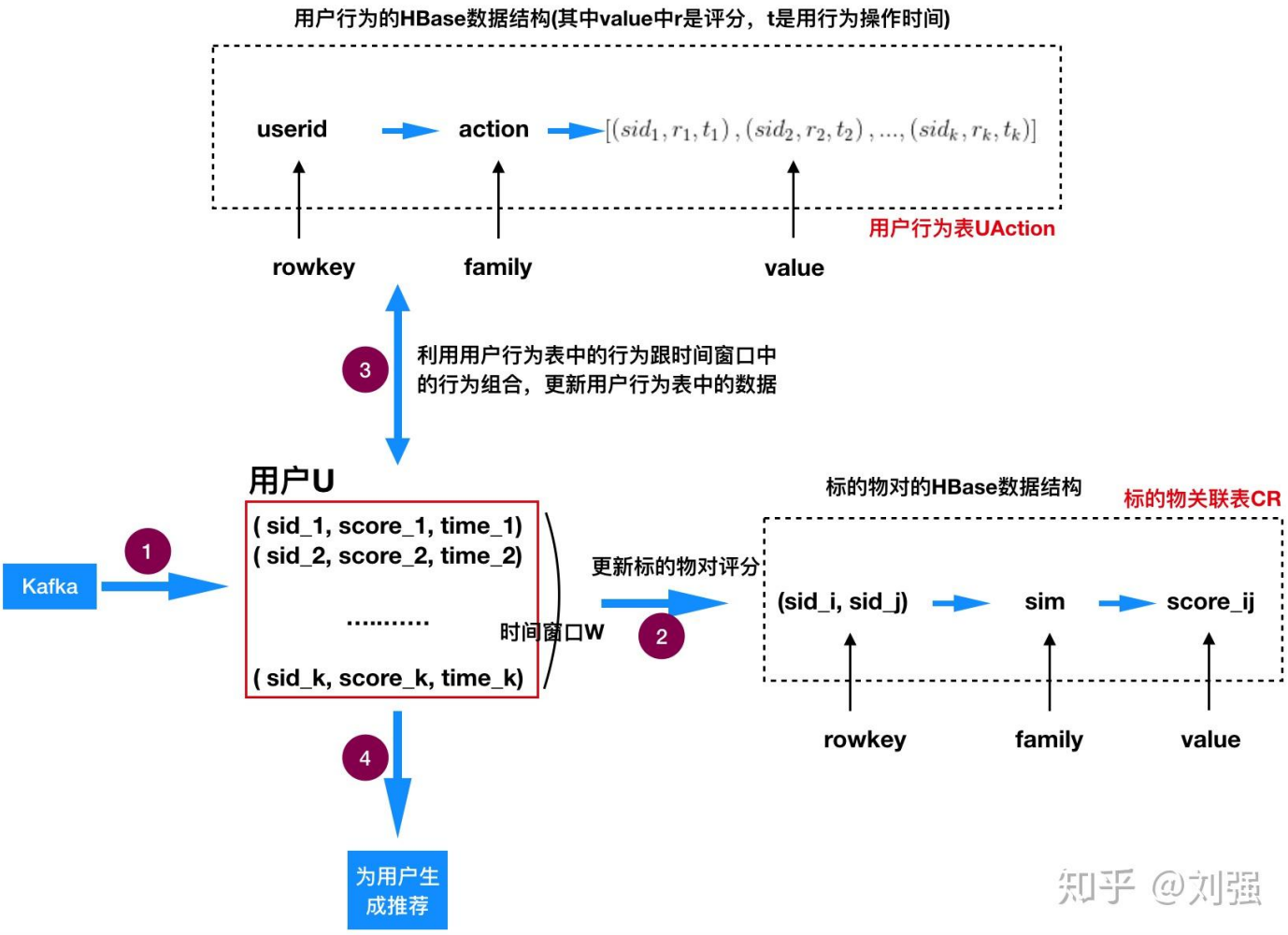


图10：近实时基于标的物的协同过滤算法流程及相关数据结构

1. 获取用户在一个时间窗口内的行为

首先Spark Streaming程序从kafka读取一个时间窗口(Window)(一般一个时间窗口几秒钟，时间越短实时性越好，但是对计算能力要求也越高)内的用户行为数据，我们对同一个用户U的行为做聚合，得到上面图中间部分的用户行为列表(用户在该时间窗口中有k次行为记录)。

顺便说一下，因为是实时计算，所以用户行为数据会实时传输到Kafka中，供后续的Spark Streaming程序读取。

1. 基于用户在时间窗口W内的行为及用户行为记录表更新标的物关联表CR

基于(1)中获取的用户行为记录，在这一步，我们需要更新标的物关联表CR，这里涉及到两类更新。首先，用户U在时间窗口W内的所有k次行为

$$[(sid_1, score_1, time_1), ..., (sid_k, score_k, time_k)]$$

，我们对标的物两两组合(自身和自身做笛卡尔积)并将得分相乘更新到CR中，比如

$$sid_1, sid_2$$

组合，它们的得分

$$score_1, score_2$$

相乘

$$score_1 \times score_2$$

更新到CR表中rowkey为

$$(sid_1, sid_2)$$

的行中。

$$(sid_1, sid_2)$$

的得分score更新为score+

$$score_1 \times score_2$$

)。其次，对于用户U在时间窗口W中的行为还要与用户行为表UAction中的行为两两组合(做笛卡尔积)采用前面介绍的一样的策略更新到CR表中，这里为了防止组合过多，我们可以只选择时间在一定范围内(比如2天内)的标的物对组合，从而减少计算量。

这里说一下，如果用户操作的某个标的物已经在行为表UAction中(这种情况一般是用户对同一个标的物做了多次操作，昨天看了这短视频，今天刷到了又看了一遍)，我们需要将这两次相同的行为合并起来，具体上我们可以将这两次行为中得分高的赋值给行为表中该标的物的得分，同时将操作时间更新为最新操作该标的物的时间。同时将时间窗口W中该操作行为剔除掉，不参上面提到的时间窗口W中的操作行为跟UAction表中同样的操作行为的笛卡尔积计算。

1. 更新用户的行为记录HBase表：UAction

基于(1)获取中的用户行为记录，当(2)处理完之后，将行为记录插入用户行为表UAction中。为了计算简单方便及保留用户最近的行为，用户行为表中我们可以只保留最近N条(可以选择的参数，比如20条)行为，同时只保留最近一段时间内(比如5天)的行为。

1. 为用户生成个性化推荐

有了上面(1)、(2)、(3)步的基础，最后一步是为用户做推荐了，我们对计算过程简单说明如下：

用户U对标的物的评分

$$R(U, s)$$

可以采用如下公式计算。

$$R(U,s) = \sum_{t \in UAction} score(U,t) * sim(t,s)$$

其中t是用户操作过的标的物，

$$score(U,t)$$

是该用户对标的物t的得分(即图10中UAction数据结构中的评分r)，

$$sim(t,s)$$

是标的物t和标的物s之间的相似度，可以采用如下公式计算，这里

$$Pair(t,s)$$

就是标的物关联表CR中(t,s)对应的得分，

$$Pair(t,t)$$

和

$$Pair(s,s)$$

类似。

$$sim(t,s) = \frac{Pair(t,s)}{\sqrt{Pair(t,t)} \times \sqrt{Pair(s,s)}}$$

当我们计算完了用户U跟所有标的物的得分之后，通过对得分降序排列取topN就可以作为U的推荐了。当标的物量很大(特别是新闻短视频类产品)时，实时计算还是压力非常大的，这时我们可以采用一个简单的技巧，我们事先从CR表中过滤出跟用户行为表中至少有一个标的物t有交集的标的物s(即标的物对

$$(t,s)$$

得分不为零)，只针对这部分标的物计算

$$R(u, s)$$

，再从这些标的物中选择得分最大的topN推荐给用户。为什么可以这么做呢？因为如果某个标的物s与用户行为标的物集合无交集，那么根据计算

$$R(U, s)$$

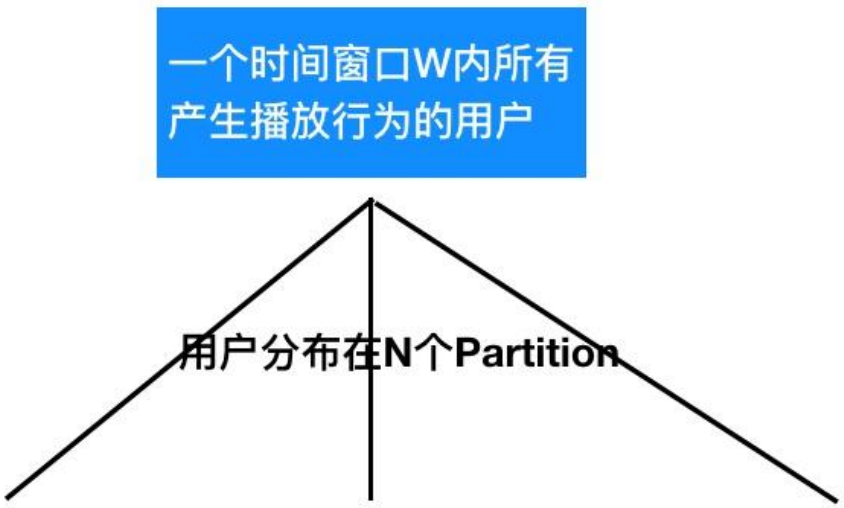
的公式，

$$sim(t, s)$$

一定为0，这时计算出的

也一定为0。

上面针对一个用户怎么实时计算协同过滤做了讲解，那么在一个时间窗口W中有若干个用户都有操作行为，这时可以将用户均匀分配到不同的Partition中，每个Partition为一批用户计推荐。具体流程可以参考下面图11。为每个用户计算好推荐后，可以插一份到HBase中作为一个副本，另外还可以通过Kafka将推荐结果同步一份到CouchBase集群中，供推荐Web服务为用户提供线上推荐服务。



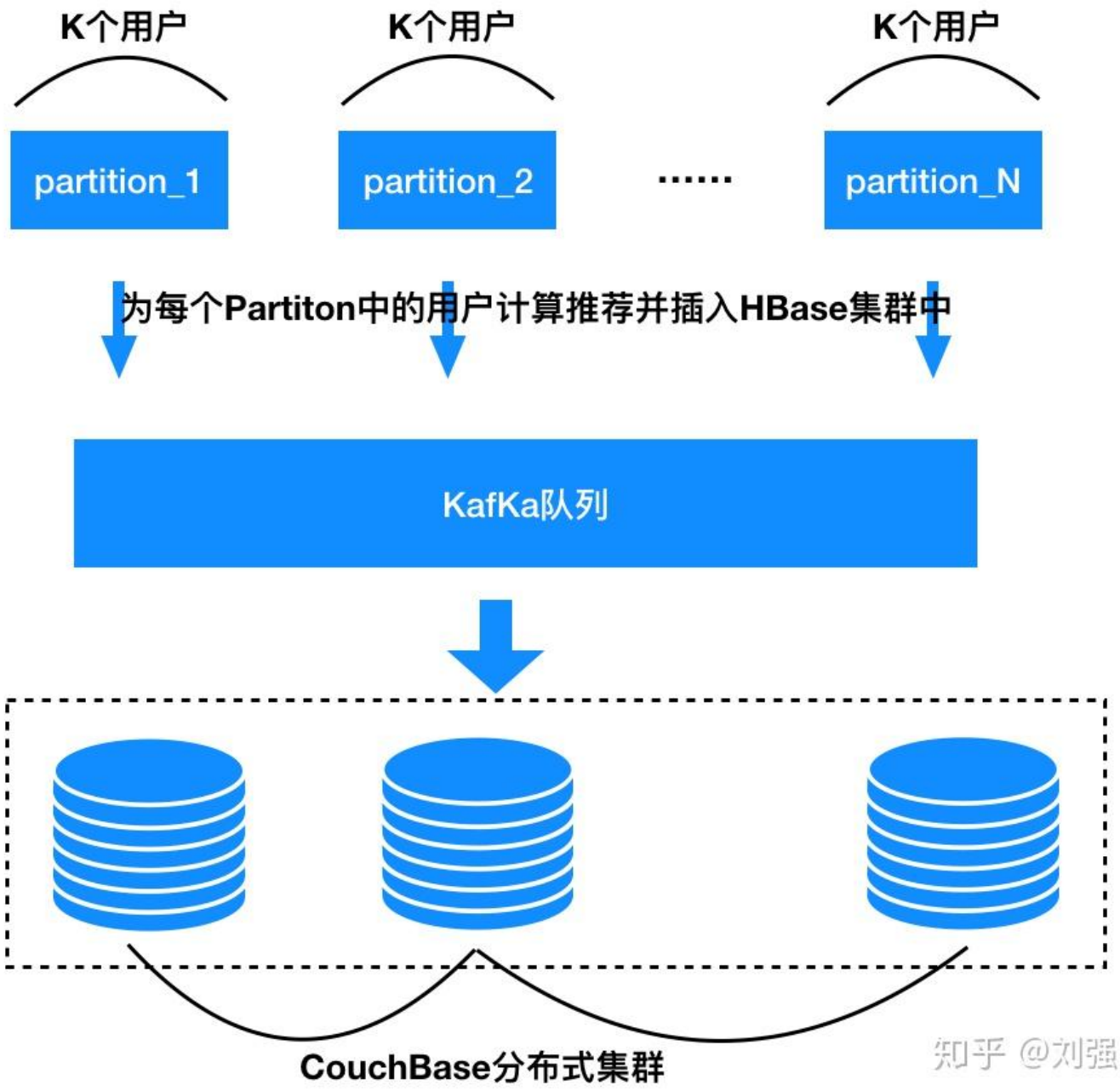


图11：在同一时间窗口W中为多个用户生成个性化推荐

近实时的协同过滤主要用于对时效性要求比较高的产品形态，比如新闻、短视频等应用。这些应用标的物更新快，用户消耗一个标的物(读一篇文章、看一段短视频)所花的时间较短，这类应用一般是用于填补用户的碎片化时间的。而对于电商、视频等产品，近实时的协同过滤不是必须的。

上面我们讲解的只是近实时协同过滤的一种实现方案，其实近实时协同过滤

有很多可行的实现方案，我们的实现方案跟参考文献6中的covisitation counts方案思路本质上是一致的。读者也可以阅读参考文献5，腾讯给出了另外一个利用Storm来实时实现协同过滤的方案，思路是非常值得借鉴的。另外参考文献6中Google实现了一个新闻的协同过滤算法，通过MinHash算法基于用户行为来近实时计算用户相似度，最终通过类似基于用户的协同过滤的算法来为用户推荐。参考文献7、8也对怎么增量做协同过滤给出了独特的方法和思路。

五、协同过滤算法的应用场景

协同过滤是非常重要的一类推荐算法，我们在第三、第四节介绍了批处理(离线)协同过滤和近实时协同过滤的工程实现方案，相信大家对怎么基于Spark及HBase技术实现协同过滤有了比较清晰的认知。那么协同过滤算法可以用于哪些推荐业务场景呢？它主要的及延伸的应用场景有如下3类：

1. 完全个性化推荐(范式)

完全个性化推荐是为每个用户推荐不一样的标的物推荐列表，我们在第二节中所讲解的两类协同过滤算法即是完全个性化推荐的方法，所以协同过滤可以用于该场景中。我们在第三、第四节中也非常明确地给出了从工程上实现完全个性化推荐的思路。

下图是电视猫电影猜你喜欢推荐，这是一类完全个性化推荐范式，这类推荐可以基于协同过滤算法来实现。

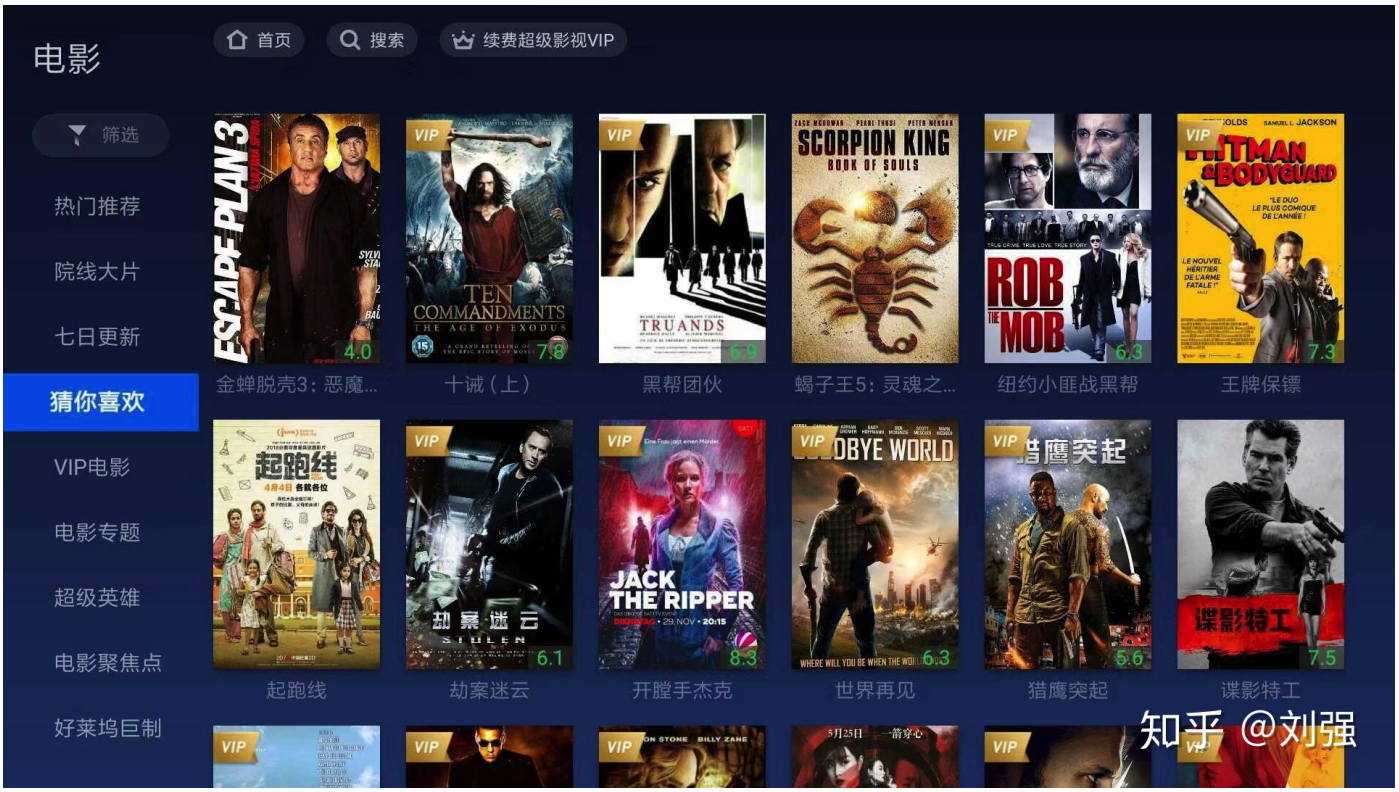


图12：电视猫完全个性化推荐：电影猜你喜欢

1. 标的物关联标的物推荐(范式)

虽然第二节没有直接讲标的物关联标的物的算法，但是讲到了怎么计算两个标的物之间的相似度(即图2中评分矩阵的列向量之间的相似度)，我们利用该相似度可以计算某个标的物最相似的K个标的物(在第三节1中我们给出了实现标的物相似性的工程实现，在第四节4中我们也给出了近实时计算标的物相似度的实现方案)。那么这K个最相似的标的物就可以作为该标的物的关联推荐。

下图是电视猫相似影片推荐，是一类标的物关联标的物推荐范式，这类推荐可以基于协同过滤算法中间过程中的标的物topN相似度计算来实现。

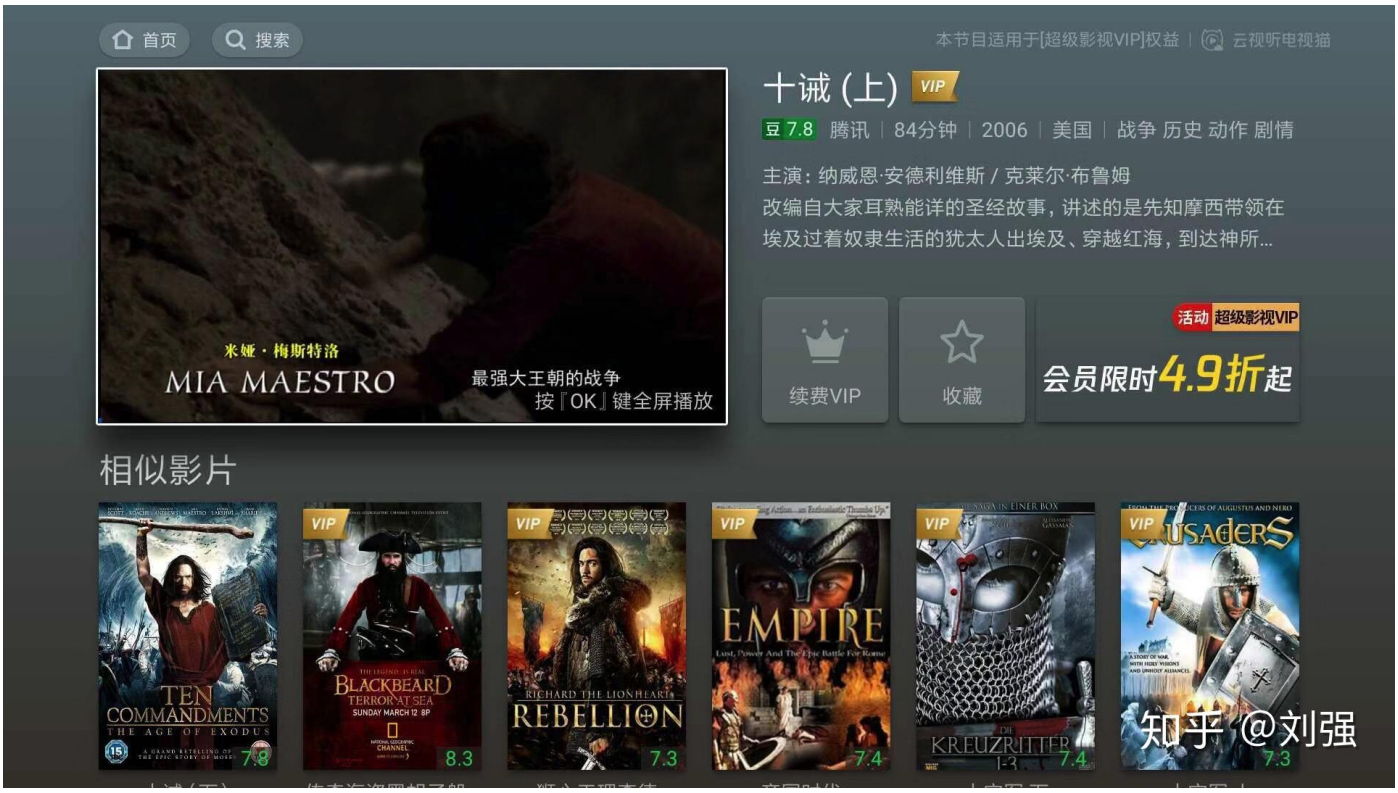


图13：电视猫标的物关联标的物推荐：相似影片

1. 其他应用形式及场景

在协同过滤算法的讲解中，我们可以将用户或者标的物用向量表示(用户行为矩阵中的行向量和列向量)，有了用户和标的物的向量表示，我们就可以对用户和标的物做聚类了。

对用户聚类后，当然可以用于做推荐，将同一类中其他用户操作过的标的物推荐给该用户就是一种可行的推荐策略。同时，用户聚类后，也可以用于做lookalike类的商业化(如广告)尝试。

对标的物聚类后，也可以用于做标的物关联推荐，将同一类中的其他标的物作为关联推荐结果。另外，标的物聚类后，这些类可以作为专题供编辑或者运营团队来作为一种内容分发的素材。

六、协同过滤算法的优缺点

前面对协同过滤算法做了比较完备的讲解，也提到了协同过滤算法的一些特点，这里我们简单罗列一些协同过滤算法的优缺点，方便大家更进一步深入了解协同过滤算法。

1. 优点

协同过滤算有很多优点，总结下来最大的优点有如下几个：

(1) 算法原理简单、思想朴素

从前面的几节讲解中不难看出，协同过滤算法的实现非常简单，只要懂简单的四则混合运算，了解向量和矩阵的基本概念就可以理解算法的原理。估计在整个机器学习领域，没有比这个算法更直观简单的算法了。

协同过滤的思想是简单的"物以类聚"、"人以群分"的思想，相信大家都可以理解，正因为思想朴素，所以算法原理简单。

(2) 算法易于分布式实现、可以处理海量数据集

我们在第三、第四节分别讲解了协同过滤算法的离线和实时工程实现，大家应该很容易看到，协同过滤算法可以非常容易利用Spark分布式平台来实现，因此可以通过增加计算节点很容易处理大规模数据集。

(3) 算法整体效果很不错

协同过滤算法是得到工业界验证过的一类重要算法，在Netflix、Google、Amazon及国内大型互联网公司都有很好的落地和应用。

(4) 能够为用户推荐出多样性、新颖性的标的物

前面讲到协同过滤算法是基于群体智慧的一类算法，它利用群体行为来做决策。在实践使用中已经被证明可以很好的为用户推荐多样性、新颖性的标的

物。特别是当群体规模越大，用户行为越多，推荐的效果越好。

(5) 协同过滤算法只需要用户的行为信息，不依赖用户及标的物的其他信息

从前面的算法及工程实践中大家可以知道，协同过滤算法只依赖用户的操作行为，不依赖具体用户相关和标的物相关的信息就可以做推荐，往往用户信息和标的物信息都是比较复杂的半结构化或者非结构化的信息，处理起来很不方便。这是一个极大的优势，正因为这个优势让协同过滤算法在工业界大放异彩。

1. 缺点

除了上面介绍的这些优点外，协同过滤算法也存在一些不足的方面，具体来说，在下面这些点，协同过滤算法存在软肋，有提升和优化的空间。

(1) 冷启动问题

协同过滤算法依赖用户的行为来为用户做推荐，如果用户行为少(比如新上线的产品或者用户规模不大的产品)，这时就很难发挥协同过滤算法的优势和价值，甚至根本无法为用户做推荐。这时可以采用基于内容的推荐算法作为补充。

另外，对于新入库的标的物，由于只有很少的用户操作行为，这时相当于用户行为矩阵中该标的物对应的列基本都是零，这时无法计算出该标的物的相似标的物，同时，该标的物也不会出现在其他标的物的相似列表中，因此无法将该标的物推荐出去。这时，可以采用人工的策略将该标的物在一定的位置曝光，或者强行以一定的比例或者概率加入推荐列表中，通过收集该标的物的行为解决该标的物无法被推荐出去的问题。

在第七节我们会更加详细介绍协同过滤的冷启动解决方案。

(2) 稀疏性问题

对于现代的互联网产品，用户基数大，标的物数量多(特别是新闻、UGC短视频类产品)，一般用户只对很少量的标的物产生操作行为，这是用户操作行为矩阵是非常稀疏的，太稀疏的行为矩阵计算出的标的物相似度往往不够精准，最终影响推荐结果的精准度。

七、协同过滤算法落地到业务场景需要关注的问题

协同过滤算法虽然简单，但是在实际业务中，为了让它有较好的效果，最终对业务产生较大的价值，我们在使用该算法时需要注意如下问题。

1. 是采用基于用户的协同过滤还是采用基于标的物的协同过滤

在互联网产品中一般会采用基于标的物的协同过滤，因为对于互联网产品来说，用户相对于标的物变化更大，用户是增长较快的，标的物增长相对较慢(这也不是绝对的，像新闻、短视频类应用标的物也是增速巨大的)，利用基于标的物的协同过滤算法效果更稳定。

1. 对时间加权

一般来说，用户的兴趣是随着时间变化的，越是久远的行为对用户当前的兴趣贡献越小，基于该思考，我们可以对用户的行为矩阵做时间加权处理。将用户评分加上一个时间惩罚因子，对久远的行为进行一定的惩罚，可行的惩罚方案可以采用指数衰减的方式。例如，我们可以采用如下的公式来对时间做衰减，我们可以选择一个时间作为基准值，比如当前时间，下式中的n是标的物操作时间与基准时间相差的天数(n=0时，w(0)=1)。

1. 关于用户对标的物的评分

在真实业务场景中，用户不一定对标的物评分，可能只有操作行为。这时可

以采用隐式反馈的方式来做协同过滤，虽然隐式反馈的效果可能会差一些。

但同时，我们是可以通过一些方法和技巧来间接获得隐式反馈的评分的，主要有如下两类方法，通过这两类方法获得评分，是非常直观的，效果肯定比隐式反馈直接用0或者1好。

虽然很多时候用户的反馈是隐式的，但用户的操作行为是多样化的，有浏览、点击、点赞、购买、收藏、分享、评论等等，我们可以基于用户这些隐式行为的投入度(投入的时间成本、资金成本、社交压力等，投入成本越大给定越高的分数)对这些行为人为打分，比如浏览给1分，点赞给2分，转发给4分等等。这样我们就可以针对用户不同的行为生成差异化的评分。

对于像音乐、视频、文章等，我们可以记录用户在消费这些标的物上所花的时间来计算评分。拿视频来说，如果一个电影总时长是100分钟，如果用户看了60分钟就退出了，那么我们就可以给用户打6分(10分制，因为用户看了60%，所以打6分)。

1. 相似度计算

我们在前面讲解协同过滤算法时需要计算两个向量的相似度，本文前面采用的是cosine余弦相似度。其实，计算两个向量相似度的方式非常多，cosine余弦是被证明在很多场景效果都不错的一个算法，但并不是所有场景cosine余弦都是最好的，需要针对不同场景做尝试和对比。在这里，我们简单罗列一些常用的相似度计算的方法，供大家参考。

(1) cosine余弦相似度

前面已经花了很大篇幅讲解了cosine的计算公式，这里不赘述。需要提一点的是，针对隐式反馈(用户只有点击等行为，没有评分)，向量的元素要么为1要么为0，直接用cosine余弦公式效果不是很好，参考文献5针对隐式反馈给出了一个更好的计算公式(见下面图14)，其中

是用户u对标的物p的评分(对于隐式反馈，评分是0或者1，但是参考文献5针对用户不同的隐式反馈给出了不同的评分，而不是一律用1，比如浏览给1分，收藏给3分，分享给5分等，

取用户u对标的物p所有的隐式反馈行为中得分最高的)。

$$sim(i_p, i_q) = \frac{\sum_{u \in U} \min\left(r_{u,p}, r_{u,q}\right)}{\sqrt{\sum r_{u,p}} \sqrt{\sum r_{u,q}}}$$

图14：一种优化后的计算隐式反馈相似度的公式，类似cosine余弦公式

(2) 皮尔森相关系数(Pearson correlation coefficient)

皮尔森相关系数是一种线性相关系数。皮尔森相关系数是用来反映两个变量线性相关的程度的统计量。具体计算公式如下面图15，其中

和

是两个向量，

和

是这两个向量的均值。参考文献9中有对怎么利用皮尔逊相关系数做协同过滤的介绍，感兴趣的读者可以参考学习。

$$r = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}}$$

图15：皮尔逊相关系数的计算公式

(3) Jaccard coefficient

Jaccard系数用于计算两个集合之间的相似度，也比较适合隐式反馈类型的用户行为，假设两个标的物

，操作过这两个标的物的用户分别为：

和

，

那么Jaccard系数的计算公式如下：

1. 冷启动问题

前面在讲协同过滤算法的缺点时讲到协同过滤算法会存在严重的冷启动问题，主要表现在如下3个方面：

(1) 用户冷启动

所谓用户冷启动就是新用户没有太多的行为，我们无法为他计算个性化推荐。这时可行的推荐策略是为这类用户推荐热门标的物、通过人工编排筛选出的标的物。或者用户只有很少的行为，协同过滤效果也不好，这时可以采用基于内容的推荐算法补充。

(2) 标的物冷启动

所谓标的物冷启动就是新的标的物加入系统，没有用户操作行为，这时协同过滤算法也无法将该标的物推荐给用户。可行的解决方案有三个：

首先，这类标的物可以通过人工曝光到比较好的推荐位(如首页)上，在尽短的时间内获得足够多的用户行为，这样就可以“启动”协同过滤算法了。这里有个比较大的问题是，如果该标的物不是主流的标的物、不够热门的话，放在好的位子不光占用资源同时对用户体验还不好。

其次，在推荐算法上做一些策略，可以将这类新的标的物以一定的概率混杂在用户的推荐列表中，让这些标的物有足够多的曝光，在曝光过程中收集用户行为，同时该方法也可以提升用户推荐的多样性。

最后，这类标的物也可以通过基于内容的推荐算法来分发出去，作者在《基于内容的推荐算法》中已经讲过内容推荐，这里不再赘述。

(3) 系统冷启动

所谓系统冷启动，就是该产品是一个新开发不久的产品，还在发展用户初期阶段，这时协同过滤算法基本无法起作用，最好采用基于内容的推荐算法或者直接利用编辑编排一些多样性的优质内容作为推荐备选推荐集。

总结

至此，协同过滤推荐算法基本讲完了，在最后我们做一个简单总结。本文对协同过滤算法原理、工程实践进行了介绍，在工程实践上既讲解了批处理实现方案，同时也讲解了一种近实时实现方案。并且对协同过滤的产品形态及应用场景、优缺点、在落地协同过滤算法中需要注意的问题进行了介绍。希望本文可以帮助读者更深入地了解协同过滤推荐算法。参考文献中的材料从学术上、工业界都对协同过滤算法原理、实践从不同视角及场景进行了论述，具有非常大的参考价值，值得大家阅读学习。

参考文献

1. Item-based collaborative filtering recommendation algorithms
2. item-based top-n recommendation algorithms
3. Collaborative filtering for implicit feedback datasets
4. http://Amazon.com reecommendations: Item-to-item collaborative filtering
5. TencentRec- Real-time Stream Recommendation in Practice
6. Google news personalization:Scalable online collaborative flitering
7. Forgetting mechanisms for incremental collaborative filtering
8. Scalable collaborative filtering using incremental update and local link prediction
9. GroupLens：An Open Architecture for Collaborative Filtering of Netnews

作者：gongyouliu 微信号：liuq4360

来源公众号：**数据与智能**