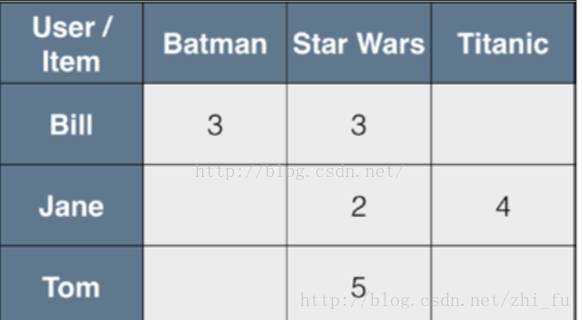
**1 推荐模型的分类**

推荐系统的研究已经相当广泛,也存在很多设计方法。最为流行的两种方法是基于内容的过滤和协同过滤。另外,排名模型等近期也受到不少关注。实践中的方案很多是综合性的,它们将多种方法的元素合并到一个模型中或是进行组合。

**1.1 基于内容的过滤**   
基于内容的过滤利用物品的内容或是属性信息以及某些相似度定义,来求出与该物品类似的物品。这些属性值通常是文本内容(比如标题、名称、标签及该物品的其他元数据)。对多媒体来说,可能还涉及从音频或视频中提取的其他属性。   
类似地,对用户的推荐可以根据用户的属性或是描述得出,之后再通过相同的相似度定义来与物品属性做匹配。比如,用户可以表示为他所接触过的各物品属性的综合。该表示可作为该用户的一种描述。之后可以用它来与物品的属性进行比较以找出符合用户描述的物品。

**1.2协同过滤**   
协同过滤是一种借助众包智慧的途径。它利用大量已有的用户偏好来估计用户对其未接触过的物品的喜好程度。其内在思想是相似度的定义。   
在基于用户的方法的中,如果两个用户表现出相似的偏好(即对相同物品的偏好大体相同),那就认为他们的兴趣类似。要对他们中的一个用户推荐一个未知物品,便可选取若干与其类似的用户并根据他们的喜好计算出对各个物品的综合得分,再以得分来推荐物品。其整体的逻辑是,如果其他用户也偏好某些物品,那这些物品很可能值得推荐。   
同样也可以借助基于物品的方法来做推荐。这种方法通常根据现有用户对物品的偏好或是评级情况,来计算物品之间的某种相似度。这时,相似用户评级相同的那些物品会被认为更相近。一旦有了物品之间的相似度,便可用用户接触过的物品来表示这个用户,然后找出和这些已知物品相似的那些物品,并将这些物品推荐给用户。同样,与已有物品相似的物品被用来生成一个综合得分,而该得分用于评估未知物品的相似度。   
基于用户或物品的方法的得分取决于若干用户或是物品之间依据相似度所构成的集合(即邻居),故它们也常被称为最近邻模型。   
最后,也存在不少基于模型的方法是对“用户−物品”偏好建模。这样,对未知“用户−物品”组合上应用该模型便可得出新的偏好。

**1.3 矩阵分解**   
Spark推荐模型库当前只包含基于矩阵分解(matrix factorization)的实现,由此我们也将重点关注这类模型。它们有吸引人的地方。首先,这些模型在协同过滤中的表现十分出色。而在Netflix Prize等知名比赛中的表现也很拔尖。

**1.3.1 显式矩阵分解**   
当要处理的那些数据是由用户所提供的自身的偏好数据，这些数据被称作显式偏好数据。这   
类数据包括如物品评级、赞、喜欢等用户对物品的评价。   
这些数据可以转换为以用户为行、物品为列的二维矩阵。矩阵的每一个数据表示某个用户对   
特定物品的偏好。大部分情况下单个用户只会和少部分物品接触，所以该矩阵只有少部分数据非   
零（即该矩阵很稀疏）。   
举个简单的例子，假设我们有如下用户对电影的评级数据：   
Tom, Star Wars, 5   
Jane, Titanic, 4   
Bill, Batman, 3   
Jane, Star Wars, 2   
Bill, Titanic, 3   
它们可转为如下评级矩阵：   
  
对这个矩阵建模，可以采用矩阵分解（或矩阵补全）的方式。具体就是找出两个低维度的矩   
阵，使得它们的乘积是原始的矩阵。因此这也是一种降维技术。假设我们的用户和物品数目分别   
是U和I，那对应的“用户−物品”矩阵的维度为U × I，   
要找到和“用户−物品”矩阵近似的k维（低阶）矩阵，最终要求出如下两个矩阵：一个用于   
表示用户的U × k维矩阵，以及一个表征物品的I × k维矩阵。这两个矩阵也称作因子矩阵。它们   
的乘积便是原始评级矩阵的一个近似。值得注意的是，原始评级矩阵通常很稀疏，但因子矩阵却   
是稠密的。这类模型试图发现对应“用户−物品”矩阵内在行为结构的隐含特征（这里表示为因子矩阵），   
所以也把它们称为隐特征模型。隐含特征或因子不能直接解释，但它可能表示了某些含义，比如   
对电影的某个导演、种类、风格或某些演员的偏好。   
由于是对“用户−物品”矩阵直接建模，用这些模型进行预测也相对直接：要计算给定用户   
对某个物品的预计评级，就从用户因子矩阵和物品因子矩阵分别选取相应的行（用户因子向量）   
与列（物品因子向量），然后计算两者的点积即可。   
而对于物品之间相似度的计算，可以用最近邻模型中用到的相似度衡量方法。不同的是，这里可以直接利用物品因子向量，将相似度计算转换为对两物品因子向量之间相似度的计算。   
因子分解类模型的好处在于，一旦建立了模型，对推荐的求解便相对容易。但也有弊端，即当用户和物品的数量很多时，其对应的物品或是用户的因子向量可能达到数以百万计。这将在存储和计算能力上带来挑战。另一个好处是，这类模型的表现通常都很出色。

因子分解类模型也存在某些弱点。相比最近邻模型，这类模型在理解和可解释性上难度都有   
所增加。另外，其模型训练阶段的计算量也很大。

**1.3.2 隐式矩阵分解**   
上面针对的是评级之类的显式偏好数据，但能收集到的偏好数据里也会包含大量的隐式反馈数据。在这类数据中，用户对物品的偏好不会直接给出，而是隐含在用户与物品的交互之中。   
二元数据（比如用户是否观看了某个电影或是否购买了某个商品）和计数数据（比如用户观看某电   
影的次数）便是这类数据。   
处理隐式数据的方法相当多。MLlib实现了一个特定方法，它将输入的评级数据视为两个矩   
阵：一个二元偏好矩阵P以及一个信心权重矩阵C。   
举例来说，假设之前提到的“用户−电影”评级实际上是各用户观看电影的次数。其中，矩阵P表示用户是否看过某些电影，而矩阵C则以观看的次数来   
表示信心权重。一般来说，某个用户观看某个电影的次数越多，那我们对该用户的确喜欢该电影   
的信心也就越强。

隐式模型仍然会创建一个用户因子矩阵和一个物品因子矩阵。但是，模型所求解的是偏好矩   
阵而非评级矩阵的近似。类似地，此时用户因子向量和物品因子向量的点积所得到的分数也不再   
是一个对评级的估值，而是对某个用户对某一物品偏好的估值（该值的取值虽并不严格地处于0   
到1之间，但十分趋近于这个区间）。

**1.3.3 最小二乘法**   
最小二乘法（Alternating Least Squares，ALS）是一种求解矩阵分解问题的最优化方法。它   
功能强大、效果理想而且被证明相对容易并行化。这使得它很适合如Spark这样的平台。在本书   
写作时，它是MLlib唯一已实现的求解方法。   
ALS的实现原理是迭代式求解一系列最小二乘回归问题。在每一次迭代时，固定用户因子矩   
阵或是物品因子矩阵中的一个，然后用固定的这个矩阵以及评级数据来更新另一个矩阵。之后，   
被更新的矩阵被固定住，再更新另外一个矩阵。如此迭代，直到模型收敛（或是迭代了预设好的   
次数）。

**2 提取有效特征**

从MovieLens 100k数据集提取特征   
从Spark主目录启动Spark shell。启动时保证内存分配充足：

>./bin/spark-shell –driver-memory 4g

* 1

val rawData = sc.textFile("/PATH/ml-100k/u.data")

val rawRatings = rawData.map(\_.split("\t").take(3))

import org.apache.spark.mllib.recommendation.Rating

val ratings = rawRatings.map { case Array(user, movie, rating) =>

Rating(user.toInt, movie.toInt, rating.toDouble) }

ratings.first()

* 1
* 2
* 3
* 4
* 5
* 6

输出   
…   
took 0.003752 s   
res8: org.apache.spark.mllib.recommendation.Rating = Rating(196,242,3.0)

**3 训练推荐模型**

现在可以开始训练模型了,所需的其他参数有以下几个。   
θ rank:对应ALS模型中的因子个数,也就是在低阶近似矩阵中的隐含特征个数。因子个数一般越多越好。但它也会直接影响模型训练和保存时所需的内存开销,尤其是在用户和物品很多的时候。因此实践中该参数常作为训练效果与系统开销之间的调节参数。通常,其合理取值为10到200。   
θ iterations:对应运行时的迭代次数。ALS能确保每次迭代都能降低评级矩阵的重建误差,但一般经少数次迭代后ALS模型便已能收敛为一个比较合理的好模型。这样,大部分情况下都没必要迭代太多次(10次左右一般就挺好)。   
θ lambda:该参数控制模型的正则化过程,从而控制模型的过拟合情况。其值越高,正则化越严厉。该参数的赋值与实际数据的大小、特征和稀疏程度有关。和其他的机器学习模型一样,正则参数应该通过用非样本的测试数据进行交叉验证来调整。   
作为示例,这里将使用的rank、iterations和lambda参数的值分别为50、10和0.01:

val model = ALS.train(ratings, 50, 10, 0.01)

model.userFeatures.count

model.productFeatures.count

* 1
* 2
* 3

14/03/30 13:10:45 INFO SparkContext: Job finished: count at :26, took 5.068255   
s res16: Long = 943   
14/03/30 13:15:21 INFO SparkContext: Job finished: count at :26,   
took 0.030044 s   
res21: Long = 1682

每个用户和每部电影都会有对应的因子数组（分别含943个和1682个因子）。

**使用隐式反馈数据训练模型**   
MLlib中标准的矩阵分解模型用于显式评级数据的处理。若要处理隐式数据,则可使用trainImplicit函数。其调用方式和标准的train模式类似,但多了一个可设置的alpha参数（也是一个正则化参数,lambda应通过测试和交叉验证法来设置）。   
alpha参数指定了信心权重所应达到的基准线。该值越高则所训练出的模型越认为用户与他所没评级过的电影之间没有相关性。