基于 Spark 的矩阵分解与最近邻融合的推荐算法①

王振军[1], 黄瑞章[1,2]

[1](贵州大学 计算机科学与技术学院, 贵阳 550025)
[2](贵州省公共大数据重点实验室, 贵阳 550025)

摘 要: 随着当前移动互联网的快速发展, 人们所面临的信息过载问题变得尤为严重, 大数据场景下对特定用户的个性化推荐面临着巨大挑战. 为了进一步提高推荐的时效性、准确度以及缓解面临的大数据量. 提出了一种矩阵分解推荐算法在大数据环境下的优化算法模型. 该模型通过在传统矩阵分解推荐算法的基础上融合了用户以及物品的相似性计算, 在训练目标函数的过程中, 即融入用户以及物品的前 k 个最近邻居的相似性计算, 增强了算法的推荐准确度. 利用 Spark 在内存计算以及迭代计算上的优势, 设计了一种 Spark 框架下的矩阵分解与最近邻融合的推荐算法. 通过在经典数据集—MovieLens 数据集上的实验结果表明, 该算法与传统的矩阵分解推荐算法相比, 可以很好的缓解数据稀疏性, 提高推荐算法的准确度, 并且在计算效率方面也优于现有的矩阵分解推荐算法.

关键词: 协同过滤; 推荐算法; 矩阵分解; 交替最小二乘法; Spark

Recommendation Algorithm Using Matrix Decomposition and Nearest Neighbor Fusion Based on Spark

WANG Zhen-Jun[1], HUANG Rui-Zhang[1,2]

[1](School of Computer Science and Technology, Guizhou University, Guiyang 550025, China)
[2](Guizhou Provincial Key Laboratory of Public Big Data, Guiyang 550025, China)

Abstract: With the current rapid development of mobile Internet, the information overload problem that people face is particularly serious, which makes it a big challenge to do particular users' personalized recommendation in the big data scenario. In order to further improve the timeliness, accuracy of recommendation and ease the problem led by large amount of data, we propose a optimized matrix decomposition recommendation algorithm under the environment of big data in this paper. This algorithm integrates users and the similarity computation of items on the basis of the traditional matrix decomposition algorithm. In the process of training objective function, we enhance the recommendation accuracy by taking in account of users and k nearest neighbors' similarity computation of items. Taking Spark's advantage on memory computing and iterative computing, we design an algorithm using matrix decomposition and nearest neighbor fusion under the Spark framework. Experiments conducted on the classical MovieLens dataset show that our proposed algorithm can deal with data sparseness well, improve recommendation accuracy to some extent, and has a better computational efficiency in the comparison with traditional matrix decomposition recommendation algorithms.

Key words: collaborative filtering; recommendation algorithm; matrix decomposition; ALS; Spark.

协同过滤技术 (Collaborative Filtering Recommendation, CF)[1]基于用户或者物品的相似性来产生推荐, 是当前推荐系统领域最成功的技术之一, 随着数据挖掘相关技术的迅速发展, 基于隐语义模型

① 基金项目:国家自然科学基金(61462011,61202089);高等学校博士学科专项科研基金(20125201120006);贵州大学引进人才科研项目(2011015);贵州省应用基础研究计划重大项目(黔科合 JZ 字[2014]2001-01)
收稿时间:2016-08-15;收到修改稿时间:2016-09-27 [doi:10.15888/j.cnki.csa.005743]

(Latent Factor Model, LFM)的推荐算法[2]是一类更加精确高效的协同过滤技术. 自Netflix Prize大赛[3]以来, LFM 算法中很重要的一部分当属矩阵分解范畴, 该方法在文献[4]中首次提到, 它与其他算法结合的混合推荐算法最终获得了 Netflix 推荐算法大赛大奖.

近些年来, 矩阵分解在学术研究以及商业应用中越来越受到研究人员的青睐. 文献[5]提出了SVD(singular value decomposition(SVD))模型方法, 其推荐效果优于传统的基于邻域的推荐算法, 其中心思想是把用户-物品的评分矩阵进行因子分解, 很多方法[6,7]通常会忽略了矩阵中高度的缺失值导致的数据稀疏性问题, 传统的 SVD 在矩阵稀疏的时候作用是不明确的, 而且, 粗暴的处理一些相对已知的条目非常容易产生过度拟合. 文献[8]提出了一种基于矩阵分解与用户近邻模型的推荐算法, 该算法很好的实现了矩阵分解与用户近邻模型的融合, 在一定程度上提高了推荐准确度, 但是忽略了物品之间的相似性对推荐准确度的影响, 并且该算法存在模型的过拟合问题. 文献[9]提出了基于 MapReduce 的矩阵分解算法, 该算法解决了大规模评分矩阵在多节点间的高效共享问题, 实现了多次迭代计算的并行处理, 但是在基于 Mapreduce 的编程框架下, 迭代过程中过多的节点间通信以及 I/O 读写影响了算法的执行效率.

本文围绕解决上述问题展开研究, 并在已有研究的基础上, 结合经典的矩阵分解推荐模型, 融合最近邻模型的相似性计算, 提出了基于 Spark 的矩阵分解与最近邻融合的推荐算法. 首先通过评分矩阵计算用户以及物品之间的相似性, 然后分别应用用户以及物品的前 k 个近邻融合矩阵分解模型来达到预测特定用户评分的目的. 实验表明, 该算法能够有效的提高推荐算法准确度以及计算效率.

1 基础知识

1.1 矩阵分解

假设用户-物品评分矩阵为 R, 矩阵中每一个元素表示用户 u 对物品 i 的评分, 则由矩阵奇异值分解原理可知, 矩阵 R 可以分解为几个矩阵相乘的形式, 矩阵分解的目标就是把用户和物品评分矩阵映射到同一个维数为 f 的潜在因子空间, 这样用户和项目之间的关系就可以通过因子空间中的内积来建模, 如下公式:

$$R_{m\times n} = P^T Q$$

其中: m 表示用户数量, n 表示物品数量; $P\in\mathbb{R}^{f\times m}$ 表示用户的隐语义属性参数(行), $Q\in\mathbb{R}^{f\times n}$ 表示物品的隐语义属性参数(行). 那么, 对于用户 u 对物品 i 的评分的预测值 $\hat{R}(u,i)=\hat{r}_{ui}$,可以通过如下公式计算:

$$\hat{r}_{ui} = \sum_f p_{uf} q_{if}$$

其中: $p_{uf}=P(U,F)$, $q_{if}=Q(i,f)$. 那么矩阵分解的目标就是计算得到每个用户的因子向量 p_u, 每个物品对应的因子向量 q_i, 使得预测用户 u 对物品 i 的预测评分 $\hat{r}_{ui}$ 能够尽可能地接近真实评分 r_{ui}. 当给定的矩阵不完全时, 此模型容易导致过拟合, 因此, 当前的很多研究都建议对存在的评分项进行建模, 采用正则化参数避免过拟合问题, 目标函数定义如下:

$$\min_{q^*,p^*}\sum_{(u,i)\in K}\left(r_{ui}-p_u q_i^T\right)^2+\lambda\left(\left\|p_u\right\|^2+\left\|q_i\right\|^2\right)$$

其中: λ 参数用来正则化模型, 成为正则化系数; K 表示训练集中存在的评分项. 求解上面的模型, 目前常用的方法有两种: 随机梯度下降法(Stochastic Gradient Descent, SGD)和交替最小二乘法(Alternating Least Squares, ALS).

1.2 最近邻模型

最近邻模型[10]是协同过滤推荐算法中比较常用的算法模型, 基本上可以分为基于用户的近邻模型以及基于物品的近邻模型. 其中心思想是计算选取用户或者物品的前 k 个最近邻居来模拟主体的行为从而预测用户对物品的评分. 这种方法不关注用户物品之间的相互关系, 而只关注用户之间以及物品之间的相似度就能计算出某用户对某物品的兴趣评分, 从而给出推荐类目.

1.3 Spark 平台

Spark[11]由美国加州大学的伯克利分校的 AMP 实验室首次提出的类 Hadoop MapReduce 的基于内存计算的大数据并行计算框架. Spark 不同于 MapReduce 的是 job 的中间结果不需要再写入本地的 HDFS, 而是直接在内存中完成, 可以更好的适用于机器学习以及数据挖掘的多次迭代的计算, 近些年来, Spark 已经被广泛应用到学术研究以及商业计算中.

Spark 最主要的创新点是提出了弹性分布式数据集[12](Resilient Distributed Dataset, RDD). RDD 本质上是一种只读的分布式共享内存模型, 它具备像 MapReduce 等数据流模型的容错特性, 并且允许开发人员在大型集群上执行内存的计算. 弹性表现在若一个 RDD 分片丢失, Spark 可以根据日志信息重构它; 分

布式表现在可以用操作本地集合的方式来操作分布式数据集. 此外, RDD 可以被缓存在内存中, 在之后的计算过程中可以直接从内存中读入, 省去了大量的磁盘存取开销, 适合需要迭代计算寻优的矩阵分解算法. 图 1 展示了 Spark 的基本工作流程, 每一个 Spark 应用程序主要由 SaprkContext 和 Executor 两部分完成, Executor 负责执行任务, 运行 Executor 的机器称为 Worker 节点, SparkContext 由用户程序启动, 通过资源调度模块和 Executor 通信. 这种运行模式有利于进行不同应用程序之间的资源调度隔离以及共享.

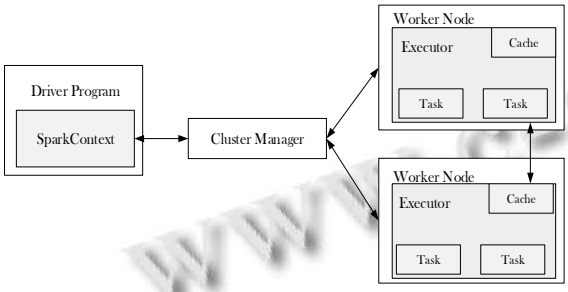

图 1　Spark 的基本工作流程

2　基于Spark的矩阵分解算法优化

2.1 问题分析

在前面介绍矩阵分解算法, 在求解隐语义向量的过程中, P 和 Q 矩阵中会丢失用户或者物品的某些信息, 如用户之间的相似性以及物品之间的相似性, 通过模型训练后得到的 P 和 Q 矩阵求得的用户或物品之间相似性(皮尔森相关系数)与由用户评分矩阵求得的相似性存在出入. 于是, 我们提出一种新的目标函数, 即融入用户和物品之间的相似性进行模型训练, 这样, 就可以保持训练前后, 它们之间相似性的一致性.

2.2 优化方案

本文称这种融合模型推荐算法为 Matrix Factori(MF-KNN), 下面是具体步骤.

1) 通过用户-物品的历史评分数据得到用户-物品评分矩阵, 经过对比研究, 这里, 我们采用皮尔森相似距离计算用户-用户, 物品-物品之间的相似度:

$$sim(i,j)=\frac{\sum_{c\in I_{ij}}\left(R_{i,C}-\bar{R}_i\right)\left(R_{j,C}-\bar{R}_j\right)}{\sqrt{\sum_{c\in I_i}\left(R_{i,C}-\bar{R}_i\right)^2}\sqrt{\sum_{c\in I_j}\left(R_C-\bar{R}_j\right)^2}}$$

2) 设计损失函数时, 为使模型训练前后的用户以及物品之间的相似性保持一致, 我们将上面计算得到的相似度数据融入目标函数中, 再迭代求解最小化目标函数, 最后得到最终的 P 和 Q 矩阵.

$$\begin{aligned}f(U,M)=&\sum_{(i,j)\in I}\left(r_{ij}-u_i m_j^T\right)^2\\&+\sum_{i\in I_j}\left(u_{ki}-\frac{\sum_{u_p\in KNN(u_i)}sim_{u_iu_p}u_{kp}}{\sum_{u_p\in KNN(u_i)}sim_{u_iu_p}}\right)^2\\&+\sum_{j\in I_i}\left(m_{kj}-\frac{\sum_{m_q\in KNN(m_j)}sim_{m_im_p}m_{kp}}{\sum_{m_q\in KNN(m_j)}sim_{m_im_p}}\right)^2\\&+\lambda\left(\|p_u\|^2+\|q_i\|^2\right)\end{aligned}$$

上述公式中, $(i,j)\in I$ 训练集中存在的评分项, I_j 表示用户集合, I_i 表示物品集合, $KNN(u_i)$表示用户 u_i 的前 K 个最近邻居集合, $KNN(m_j)$表示物品 m_j 的前 K 个最近邻居集合, $sim_{u_iu_p}$ 代表用户之间的相似度, $sim_{m_im_p}$ 代表物品之间的相似度, k 表示某一个特征(或隐含因子).

为求解上述目标函数, 我们采用 Spark 计算方式来求解特征矩阵 U 和 M, 与上一节中提到求解传统矩阵分解模型的方法类似, 这里我们采用最小二乘法(ALS), 首先固定 U 求解 M, 然后固定 M 求解 U, 这样迭代求解. 具体如下:

$$\text{令}\ \frac{1}{2}\frac{\partial f}{\partial u_{ki}}=0,\forall i,k\quad\Rightarrow\sum_{j\in I_i}\left(u_i^T m_j-r_{ij}\right)m_{kj}+\left(u_{ki}-\frac{\sum_{u_p\in KNN(u_i)}S_{u_iu_p}u_{kp}}{\sum_{u_p\in KNN(u_i)}S_{u_iu_p}}\right)+\lambda n_{ui}u_{ki}=0,\forall i,k$$

$$\Rightarrow\sum_{j\in I_i}m_{kj}m_j^T u_j+(\lambda+1)u_{kj}=\sum_{j\in I_i}m_{kj}m_j^T r_{ij}+\frac{\sum_{u_p\in KNN(u_i)}S_{u_iu_p}u_{kp}}{\sum_{u_p\in KNN(u_i)}S_{u_iu_p}},\forall i,k$$

$$\Rightarrow\left(M_{I_I}M_{I_I}^T+\left(\lambda n_{ui}+1\right)E\right)u_i=M_{I_i}R^T(i,I_i)+\frac{\sum_{u_p\in KNN(u_i)}S_{u_iu_p}u_{kp}}{\sum_{u_p\in KNN(u_i)}S_{u_iu_p}},\forall i$$

$$\Rightarrow u_i=\left(M_{I_I}M_{I_I}^T+\left(\lambda n_{ui}+1\right)E\right)^{-1}\times\left(M_{I_i}R^T(i,I_i)+\frac{\sum_{u_p\in KNN(u_i)}S_{u_iu_p}u_{kp}}{\sum_{u_p\in KNN(u_i)}S_{u_iu_p}}\right),\forall i$$

以上, 每次迭代求取 u_i, 同理, 对于 m_j, 有:

$$m_j = \left(M_{I_I} M_{I_I}^T + (\lambda n_{ui} + 1)E\right)^{-1} \times \left(M_{I_i} R^T(j, I_j) + \frac{\sum_{m_q \in KNN(m_j)} S_{m_j m_q} m_p}{\sum_{m_q \in KNN(m_j)} S_{m_j m_q} m_p}\right)$$

我们这样设计是因为考虑融合相似性信息进损失函数中, 如果全面考虑计算 U 和 M 中用户或者物品的相似性, 然后与由评分矩阵计算的到的相似性进行对比时, 偏导数的求解很复杂, 而且计算量过大, 于是设计为计算用户或者物品的K近邻这种方法简单处理. 可以很好的中和 K 近邻模型的相似性进损失函数, 以期望在矩阵分解后, 能减少用户或者物品信息的丢失, 进而提高算法在评分预测场景下的准确度.

2.3 Spark 框架下的矩阵分解最最近邻融合的并行化算法

输入: 用户评分矩阵 *Ratings*

输出: 矩阵分解模型

```
#Ratings 为用户评分矩阵, U 与 M 为用户及物品个数, F 为隐语义属性个数
1) partitions = N
2) sc = SparkContext()
3) 读入数据生成 Rating:RDD,#Rating 是序列<User,Item,rating>
4) 随机生成 ms 和 us 矩阵
   W = matrix(rand(m,F))
   U = matrix(rand(u,F))
   Rb = sc.broadcast(R)
   Mb = sc.broadcast(M)
   Ub = sc.broadcast(U)
5) #相似性计算获得最近邻居
   user_sims = Rb.map(lambda \
   x:calcSim(x[0],x[1])).map(lambda \
   x:keyOnFirstUser(x[0],x[1]))groupByKay().ma
   p(lambda x:nearestNeighbors(x[0],x[1],20))
   item_sims = Rb.map(lambda\
   x:calcSim(x[0],x[1])).map(lambda\
   x:keyOnFirstItem(x[0],x[1]))groupByKay()\
    .map(lambda
   x:nearestNeighbors(x[0],x[1],20))
6) #按照设定的迭代值进行迭代
   for i in range(ITERATIONS):
       #固定 U, 并行化更新 M
       M = sc.parallelize(range(m),partisons)\
       .map(lambda                              \
       x:update(x,Mb.value[x,:],Ub.value,Rb.val
       ue)) \
       .collect()
       M = matrix(np.array(M)[:,:,0])
       Mb = sc.broadcast(M)
       #固定 M, 并行化更新 U
       U = sc.parallelize(range(u),partition) \
       .map(lambda
       x:update(x,Ub.value[x,:],Mb.value,Rb.val
       ue.T)).collect()
       U = matrix(np.array(U)[:,:,0])
       Ub = sc.broadcast(U)
       error = rmse(R,M,U)
   sc.stop()
```

如上述算法所示: 基于 Spark 的矩阵分解与最近邻融合算法的寻优方式和矩阵分解类似, 算法主要是模型不断迭代寻优的过程, 其中固定 U 更新 M, 然后固定 M 更新 U, 依次循环迭代, 直到在训练集上计算得到 rmse 值达到近似收敛, 即模型达到了最优, 最后求得的 *Ub* 和 *Mb* 分别为用户和物品的特征矩阵.

3 实验分析

3.1 实验环境

基于上述研究, 本实验采用 3 台搭载 CentOS 6.5 操作系统的服务器, 在部署了Hadoop2.6.0基础上部署了 Spark-1.5.0 分布式集群, 利用 Pycharm 作为开发工具.

3.2 数据集

为了验证算法效果, 本实验采用国际经典数据集—MovieLens 数据集, 数据集包括了 943 个用户对大约 1500 部电影的评分数据, 数据条数为 100000. 用户对电影的评分数值为从 1 到 5. 其中, 1 表示最低, 5 表示最高, 0 代表未评论.

实验过程将原始数据集以 8:2 的比例划分为训练集和测试集.

3.3 评测标准

本实验将采用推荐系统领域最常用的均方根误差(Root Mean Squared Error, RMSE)作为评测标准. 通过

计算预测的用户评分与实际的用户评分之间的误差来反映评分预测的准确性, RMSE 越小, 评分预测准确度就越高.

对于测试集中的每一个用户 u 和物品 i, 令 T 表示用户 u 对物品 i 的评分集合, r_{ui} 是用户 u 对物品 i 的实际评分, 而 $\hat{r}_{ui}$ 是推荐算法给出的预测评分, RMSE 的定义公式如下:

$$RMSE=\sqrt{\frac{\sum_{u,i\in T}\left(r_{ui}-\hat{r}_{ui}\right)^2}{|T|}}$$

3.4 实验结果与分析

本文通过实验对比 MF-KNN 算法在评分预测问题中的性能. 在该算法中, 重要的模型参数有 3 个: 隐含特征个数 F, 近邻模型邻居 k 的个数以及正则化参数 *lambda*. 通过多轮试验发现, 隐含特征个数以及近邻模型邻居 k 的个数对算法性能影响最大. 算法参数选择: 正则化参数 *lambda* 通过交叉验证决定.

本文统一采用 *lambda*=0.005 进行实验, 在训练集上迭代次数选用 40 次. 控制邻域范围 k, 通过实验证明 k 值越大, 算法精度越高, 因此需要在算法精度与计算代价之间取得平衡.

实验 1 我们首先在 Spark 环境下分别对优化后的矩阵分解与最近邻融合的推荐算法(MF-KNN)、传统的矩阵分解推荐算法(MF)以及基于物品的邻域模型(ItemNgbr)进行对比实验, 计算两者的评分预测准确度指标 RMSE 值.

对于上组实验中的的每种模型, 选用最优化的参数, 对于基于物品的邻域模型, 我们选取邻居大小为 50, 而对于矩阵分解与最近邻融合的模型, 我们分别选取用户以及物品邻居大小集合为 20. 实验结果如图 2 所示, 可以得出矩阵分解模型可以很好的优于基于邻域的模型, 进而得出, 矩阵分解与最近邻融合模型可以在很大程度提高矩阵分解的推荐准确度, 另外, 从图中可以看出, 随着 F 值的增加, RMSE 值下降速度明显变缓, 这也从侧面表明了算法每次计算得出的结果都是最显著的特征向量.

其次, 我们进一步考虑最近邻居 k 的个数对算法性能的影响. 实验 2 我们采用固定 F=50, 等差增加 k 值, 观测算法迭代耗费时间以及预测误差的变化.

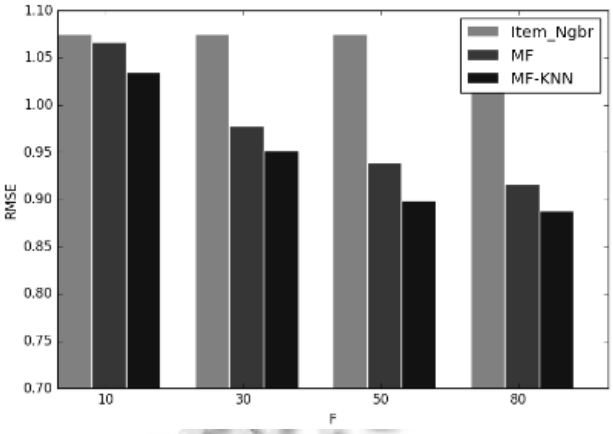

图 2 不同 F 值各模型在 MovieLens 数据集下的 RMSE 值

表 1 随着最近邻居 k 的增加所获得的预测误差

F 值	k 值	迭代耗费时间(s)	RMSE
F=50	10	49.36	0.9212
	20	62.29	0.9072
	30	85.36	0.9053
	40	106.52	0.9020

表 1 可以得出, 当固定隐特征数量 F=50, 迭代收敛的耗费时间随最近邻居 k 值的增加而上升, 当达到 k=20 时, *RMSE* 值变化幅度微小. 主要是因为, 在融合最近邻居相似度信息的时候, 前 k 个最近邻居可以基本代表整个主体行为信息. 因此, 针对于不同的应用场景, 经过多次实验, 才可以获得本场景下最佳近邻模型的 k 值.

最后, 为验证算法的计算效率, 实验 3 在 Spark 环境下和非分布式环境下分别计算本文提出的基于 Spark 的矩阵分解与最近邻融合的推荐算法运行时间, 比较其在相同用户数量, 相同硬件条件下的计算效率. 图 3 表明基于 Spark 的矩阵分解与最近邻融合的推荐算法对于同等规模的数据集, Spark 框架下的计算方式要比非分布式的计算方式节约更多的运行时间, 并且随着 Spark 节点数量的增加, 此算法在 Spark 框架下的耗时增长更趋于平滑, 这体现了 Spark 基于内存计算的实时性优势. 当用户数=200 时, 一个 Saprk 节点与非分布式的计算时间相比耗时要多一些, 这说明, Spark 开始计算时, 要启动整个计算框架分片, 要耗费一定时间.

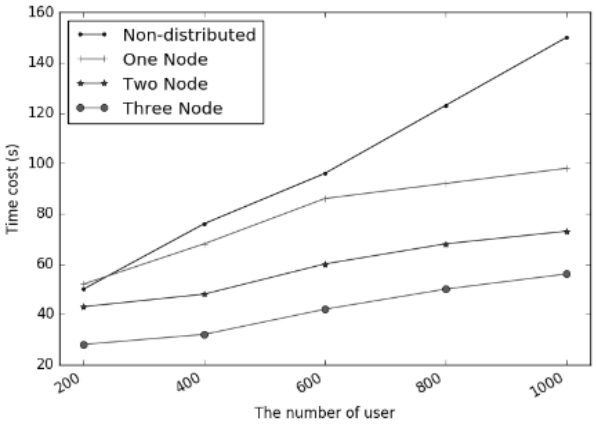

图 3 不同用户数量并行化运行时间的比较

综上可以得出, 基于 Spark 的矩阵分解与最近邻融合的推荐算法无论是在推荐准确度还是在计算效率方面都要优于当前大部分的推荐算法.

4 结语

最近几年, 矩阵分解推荐算法是推荐系统领域中的一个非常热门的研究课题. 本文针对传统矩阵分解推荐算法面临的推荐准确度以及计算效率方面的问题, 结合传统的最近邻算法以及当前新兴并行化计算框架, 提出了基于 Spark 的矩阵分解与最近邻融合的推荐算法. 该算法充分利用显性反馈数据, 提高了推荐准确度的同时, 也很大程度上提高了传统矩阵分解算法的计算效率.

最后, 本文设定的主要应用场景为只有显式反馈, 虽然这种场景比较普遍, 但在现实生活中, 显式反馈与隐式反馈并存, 如何整合这两种资源以提高推荐质量有待进一步研究.

参考文献

1 Schafer JB, Dan F, Herlocker J, et al. Collaborative filtering recommender systems. The Adaptive Web, Methods and Strategies of Web Personalizationk, 2010: 46–45.

2 Koren Y. Factorization meets the neighborhood: A multifaceted collaborative filtering model. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM. 2008. 426–434.

3 Bennett J, Lanning S, Netflix N. The Netflix Prize// Kdd Cup and Workshop in Conjunction with Kdd. 2009.

4 Koren Y, Bell R,Volinsky C. Matrix factorization techniques for recommender systems. Computer, 2009, 42(8): 30–37.

5 Vozalis MG, Margaritis KG. Applying SVD on item-based filtering. International Conference on Intelligent Systems Design and Applications. 2005. 464–469.

6 Kim D, Yum BJ. Collaborative filtering based on iterative principal component analysis. Expert Systems with Applications, 2005, 28(4): 823–830.

7 Herlocker JL, Konstan JA, Borchers A, et al. An algorithmic framework for performing collaborative filtering. SIGIR'99: Proc. of the International ACM SIGIR Conference on Research and Development in Information Retrieval. August 15–19, 1999. Berkeley, Ca, Usa. 1999. 230–237.

8 杨阳,向阳,熊磊.基于矩阵分解与用户近邻模型的协同过滤推荐算法.计算机应用,2012,32(2):395–398.

9 张宇,程久军.基于 MapReduce 的矩阵分解推荐算法研究.计算机科学,2013,40(1):19–21.

10 Su X, Khoshgoftaar TM. A survey of collaborative filtering techniques. Advances in Artificial Intelligence, 2009, (12).

11 Zaharia M, Chowdhury M, Franklin MJ, et al. Spark: Cluster computing with working sets. Usenix Conference on Hot Topics in Cloud Computing. USENIX Association. 2010. 1765–1773.

12 Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Usenix Conference on Networked Systems Design and Implementation. 2012. 141–146.