



基于 Spark 平台的 ALS 加速算法研究

贾晓芳, 桑国明, 祁文凯

(大连海事大学 信息科学技术学院, 辽宁 大连 116026)

摘 要: 协同过滤推荐算法在推荐系统中发挥着重要作用,但其存在执行效率与排名精度较低的问题,交替最小二乘(ALS)算法可实现并行计算,从而提高执行效率,但是该算法数据加载与迭代收敛的时间较长。为此,将非线性共轭梯度(NCG)算法与 ALS 算法相结合,提出一种 ALS-NCG 算法,以达到加速 ALS 算法的目的。在 Spark 分布式数据处理环境中对 ALS-NCG 算法进行性能评估,实验结果表明,相比 ALS 算法,ALS-NCG 算法获取高精度推荐排名时需要的迭代次数与时间更少。

关键词: 协同过滤;推荐算法;交替最小二乘算法;非线性共轭梯度;Spark 平台

开放科学(资源服务)标志码(OSID):



中文引用格式:贾晓芳,桑国明,祁文凯. 基于 Spark 平台的 ALS 加速算法研究[J]. 计算机工程,2020,46(2):103-109.

英文引用格式:JIA Xiaofang, SANG Guoming, QI Wenkai. Research on ALS acceleration algorithm based on Spark platform[J]. Computer Engineering, 2020, 46(2): 103-109.

Research on ALS Acceleration Algorithm Based on Spark Platform

JIA Xiaofang, SANG Guoming, QI Wenkai

(School of Information Science and Technology, Dalian Maritime University, Dalian, Liaoning 116026, China)

[Abstract] Collaborative filtering algorithm plays an important role in recommendation system, but its execution efficiency and ranking accuracy are both low. Alternating Least Squares (ALS) algorithm can implement parallel computing, thus improving the execution efficiency, but the time between data loading and iterative convergence of the algorithm is a bit long. Therefore, by combining the Nonlinear Conjugate Gradient (NCG) algorithm and the ALS algorithm, this paper proposes an ALS-NCG algorithm to accelerate the ALS algorithm. The performance of the ALS-NCG algorithm is evaluated in the Spark distributed data processing environment. Experimental results show that compared with the ALS algorithm, the ALS-NCG algorithm needs less iterations and time to obtain high-precision recommended ranking.

[Key words] collaborative filtering; recommendation algorithm; Alternating Least Squares (ALS) algorithm; Nonlinear Conjugate Gradient (NCG); Spark platform

DOI:10.19678/j.issn.1000-3428.0054147

0 概述

在互联网时代,海量规模的数据信息虽然能够满足用户的多样化需求,却难以让用户在搜索时直接获取目标信息,信息处理技术的性能具有很大的提升空间。国际数据集团(IDC)2012年的报告显示,预计2020年的全球数据总量将达到35.2 ZB,是2011年的22倍^[1]。从信息匮乏到信息过载^[2-3],数据信息量产生巨大改变,但用户对信息的使用效率并未大幅提高。在这样的大数据背景下,推荐系统^[4-5]应运而生,其根据用户潜在需求和兴趣来提供

个性化的推荐服务。文献[6]中的邮件过滤系统被认为是互联网时代较早提出并被广泛应用的协同过滤系统。协同过滤推荐系统^[7-8]可以尽可能地满足用户的需求,产生与用户喜好相关且数量有限的物品推荐列表。Amazon、Netflix、淘宝等众多国内外知名网站中的推荐服务皆依赖协同过滤推荐系统,其中,用户和物品种类的数量相当庞大,这对协同过滤推荐系统的准确性、执行效率和排名精度都有非常高的要求。

协同过滤推荐算法^[9]是一种典型的大数据挖掘算法,其基于用户或物品的相似性来进行推荐。尽

基金项目:国家自然科学基金(61672122);中央高校基本科研业务费项目“大规模协作式多智能体强化学习技术研究”(3132019207)。

作者简介:贾晓芳(1994—),女,硕士研究生,主研方向为大数据应用、数据挖掘算法;桑国明(通信作者),副教授;祁文凯,硕士研究生。

收稿日期:2019-03-08 修回日期:2019-05-06 E-mail:1509887295@qq.com

管协同过滤推荐算法的应用使推荐系统获得了用户的认可,但稀疏性高、执行效率与排名精度低等问题依然存在。为此,研究人员提出一类高效的基于模型的协同过滤算法,其中,基于交替最小二乘(Alternating Least Squares, ALS)^[10]的矩阵分解(Matrix Factorization, MF)模型^[11]可实现并行计算,该特性使其在速度方面具有明显优势,但是,该模型存在数据加载时间长、矩阵分解与迭代收敛慢等问题。国内外学者先后围绕上述问题进行了优化,其中,快速矩阵分解模型 IALS^[12]、LS-WR、NALS-WR^[13]、ALS++^[14]等改进优化方法的侧重点在于缩短数据加载与矩阵分解的时间以及改变 ALS 预测模型等,但对于 ALS 迭代收敛慢的问题研究较少。

本文提出一种在 ALS 中融入非线性共轭梯度(Nonlinear Conjugate Gradient, NCG)的算法 ALS-NCG^[15],以加速 ALS 算法的收敛,提高执行效率和排名精度。

1 基于矩阵分解的 ALS 算法

MF 算法具有伸缩性好、灵活性高的特点^[16]。在 Netflix Prize 算法比赛中,文献[17]提出了基于 ALS 的协同过滤算法,其能很好地解决矩阵稀疏性问题。

1.1 ALS 算法

ALS 算法利用迭代求解一系列最小二乘的回归问题。对于用户-项目评分矩阵 R , 找到一个低秩矩阵 R' 来逼近原始数据评分矩阵 R , 完整过程如下:

$$R \approx R' = UV^T \quad (1)$$

其中, $U \in \mathbb{C}^{m \times d}$, $V \in \mathbb{C}^{n \times d}$, d 表示特征值个数, $r \approx \min(m, n)$, r 表示矩阵 R 的秩。

损失函数定义为:

$$L(U, V) = \sum_{i,j \in r} (R_{ij} - X_{ij})^2 = \sum_{i,j \in r} (R_{ij} - U_i \cdot V_j^T)^2 \quad (2)$$

其中, L 表示损失函数。

为防止过拟合, 对式(2)进行正则化处理, 得到优化式(3):

$$L(R, U, V) = \sum_{i,j \in r} (R_{ij} - U_i \cdot V_j^T)^2 + \lambda (\|U_i\|_2^2 + \|V_j\|_2^2) \quad (3)$$

其中, λ 表示正则化参数。

固定 V 后对 U_i 求导, 令 $\frac{\partial L(U, V)}{\partial (U_i)} = 0$, 得到 U_i 的值:

$$U_i = R_i \cdot V_{ui} (V_{ui}^T V_{ui} + \lambda n_{ui} I)^{-1}, i \in [1, m] \quad (4)$$

其中, R_i 表示用户 i 对项目的评分矩阵, V_{ui} 表示用户 i 所评价过的项目涉及的特征向量所形成的特征矩阵, n_{ui} 表示用户 i 所评价的项目数量。同理, 固定 U 后对 V_j 求导, 然后根据求导结果解出 V_j :

$$V_j = R_j^T U_{mj} (U_{mj}^T U_{mj} + \lambda n_{mj} I)^{-1}, j \in [1, n] \quad (5)$$

$$f(U, M) = \sum_{(i,j) \in I} (r_{ui} - u_i^T m_j)^2 + \lambda \left(\sum_i n_{ui} \|u_i\|^2 + \sum_j n_{mj} \|m_j\|^2 \right) \quad (6)$$

其中, R_j 表示评分过项目 j 的用户的评分向量, U_{mj} 表示评分过项目 j 的用户的特征向量所组成的特征矩阵, n_{mj} 表示评分项目 j 的用户数量。

可利用式(6)中的 $\lambda \left(\sum_i n_{ui} \|u_i\|^2 + \sum_j n_{mj} \|m_j\|^2 \right)$ 防

止过拟合问题。根据以上过程可以发现, U, M 的每列都是单独计算得到的, 适合并行计算, 利用上述过程多次迭代更新 U, M , 直到完成所有迭代或达到收敛状态。

1.2 ALS 算法存在的问题

基于 ALS 矩阵分解的协同过滤推荐算法通过对 ALS 算法的迭代, 在 Spark 集群环境下训练出最佳模型并获取最优参数, 但是 ALS 算法自身存在的一些不足无法通过模型训练、参数优化来改进。比如成本消耗问题, 有数据表明, 推荐系统利用基于 Spark 的 ALS 算法进行实时推荐, 输入数据 114 GB, 训练时间和预测时间总和超过 50 min, 是实际要求时间的 3 倍多, 超出的时间将导致巨大的资源消耗, 且推荐结果的准确性也较低。

针对 ALS 算法存在的问题, 本文将加速收敛过程以优化 ALS 模型, 并在优化 $f(U, M)$ 的过程中提高推送结果的准确性。

2 基于 NCG 的 ALS 算法

2.1 NCG 算法

NCG 算法是线性系统 CG 的扩展, 其不仅具有良好的收敛性, 而且数值表现较好^[18]。本文选择的非线性共轭梯度法是 PRP(Polak, Ribiere and Polyr)^[19]方法, 该方法克服了遇到小步长时收敛缓慢的缺点, 同时也有效地避免了连续产生小步长, 在数值表现上优于其他非线性共轭梯度法, 因此, 常被应用于处理最优化问题。利用 NCG 算法来处理 $\min_{U, M} f(U, M)$ 问题, $f(U, M)$ 等价于 $\min_{x \in \mathbb{R}^n} f(x)$ 中的 $f(x)$, 其中, 规定:

$$x^T = [U_1^T, U_2^T, \dots, U_{nu}^T, M_1^T, M_2^T, \dots, M_{nm}^T] \quad (7)$$

NCG 算法用递推关系 $x_{k+1} = x_k + \alpha_k p_k$ 从初始猜测 x_0 中生成迭代序列 $x_i, i \geq 1$, 步长参数 $\alpha_k > 0$ 是沿着搜索方向 p_k 的线搜索而确定的, 每次迭代包括计算线搜索方向 p_k 和步长因子 α_k 两部分, 搜索方向 p_k 要求为下降的方向, 沿着该方向搜索找到比 x_k 更好的点。求解 $\min_{U, M} L_\lambda(R, U, M)$ 的共轭梯度算法是根据求解线性方程组问题推广而来, 搜索方向为负梯度与上一次搜索方向的线性组合。 p_k 求解如下:

$$\begin{cases} p_{k+1} = -g_{k+1} + \beta_{k+1} p_k, & k > 0 \\ p_0 = -g_0, & k = 0 \end{cases} \quad (8)$$

其中, β_{k+1} 表示更新参数, 本文在更新 β_{k+1} 时使用 PRP 共轭梯度法^[20], 如式(9)所示。

$$\beta_{k+1} = \frac{\mathbf{g}_{k+1}^T (\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{g}_k^T \mathbf{g}_k} \quad (9)$$

NCG 算法伪代码描述如下:

算法1 NCG 算法

输入 \mathbf{x}_0
 输出 \mathbf{x}_k
 $\mathbf{g}_0 \leftarrow -\nabla f(\mathbf{x}_0)$;
 $\mathbf{p}_0 \leftarrow -\mathbf{g}_0$;
 $k \leftarrow 0$;
 while $\mathbf{g}_k \neq 0$ do
 Compute α_k ;
 $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$;
 $\mathbf{g}_k \leftarrow -\nabla f(\mathbf{x}_{k+1})$;
 Compute β_{k+1} ;
 $\mathbf{p}_{k+1} \leftarrow -\mathbf{g}_{k+1} + \beta_{k+1} \mathbf{p}_k$;
 $k \leftarrow k + 1$;
end

2.2 ALS-NCG 算法设计

较好的收敛特性和数值表现使得 NCG 算法在处理最优解问题时具有良好性能, 但是利用 NCG 算法单独求解 $\min L(\mathbf{R}, \mathbf{U}, \mathbf{V})$ 时效果并不好, 因此, NCG 算法不能代替 ALS 算法, 也可以认为 ALS 算法是 NCG 算法的非线性预因子, 将两者融合后可以达到加速收敛的效果。首先对 NCG 算法 β_{k+1} 和 \mathbf{g}_k 的预处理进行修改, $\bar{\mathbf{x}}$ 表示由 \mathbf{x}_k 的 ALS 算法一次迭代生成的迭代, $\bar{\mathbf{x}} = H(\mathbf{x}_k)$, H 表示 ALS 的一次迭代。通过定义由 ALS 生成的预处理梯度方向, 将该迭代整合到 NCG 算法中。将式(10)中 \mathbf{g}_k 替换成 $\bar{\mathbf{g}}_k$, 并将更新参数 β_{k+1} 重新定义为式(11) $\bar{\beta}_{k+1}$ 的形式:

$$\bar{\mathbf{g}}_k = \mathbf{x}_k - \bar{\mathbf{x}}_k = \mathbf{x}_k - H(\mathbf{x}_k) \quad (10)$$

$$\bar{\beta}_{k+1} = \frac{\bar{\mathbf{g}}_{k+1}^T (\bar{\mathbf{g}}_{k+1} - \bar{\mathbf{g}}_k)}{\bar{\mathbf{g}}_k^T \bar{\mathbf{g}}_k} \quad (11)$$

式(11)与式(9)类似, 但是并不代表 $\bar{\mathbf{g}}_k$ 可以取代任何一个 \mathbf{g}_k , 经过对替代结果的分析, 给出 $\bar{\beta}_{k+1}$ 不同形式的概述。本文后续将式(11)应用于对比实验。

ALS-NCG 算法伪代码描述如下:

算法2 ALS-NCG 算法

输入 \mathbf{x}_0
 输出 \mathbf{x}_k
 $\bar{\mathbf{g}}_0 \leftarrow \mathbf{x}_0 - H(\mathbf{x}_0)$;
 $\mathbf{p}_0 \leftarrow -\bar{\mathbf{g}}_0$;
 $k \leftarrow 0$;
 while $\mathbf{g}_k \neq 0$ do
 Compute α_k ;
 $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$;
 $\bar{\mathbf{g}}_{k+1} \leftarrow \mathbf{x}_{k+1} - H(\mathbf{x}_{k+1})$;
 Compute $\bar{\beta}_{k+1}$;

$$\mathbf{p}_{k+1} \leftarrow -\bar{\mathbf{g}}_{k+1} + \bar{\beta}_{k+1} \mathbf{p}_k;$$

$$k \leftarrow k + 1;$$

end

算法2是改进的 ALS 算法, 将初始值 \mathbf{x}_0 作为输入, 利用递推关系求解 \mathbf{x}_k 并输出。算法3是步长因子 α_k 的计算过程, 将算法2求解的 \mathbf{x}_k 作为算法3的输入, 在求解过程中, 需要注意线搜索类型的选择。线搜索类型通常可以分为精确线搜索和非精确线搜索, 参数 α_k 对线搜索类型选取至关重要, α_k 是从 \mathbf{x}_k 在沿着 \mathbf{p}_k 的方向寻找的一个最好的点, 将其作为下一个迭代点, 此时最好的点就是函数 $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$ 在该方向上数值最小的点, 但是此时属于精确线搜索状态, 其计算量太大, 实际应用时具有较高难度, 因此, 本文算法求解步长因子时选择非精确线搜索类型。c 和 τ 的取值范围为 $[0, 1]$, 本文算法3的参数选择最佳经验结果为: $c = 0.9, \tau = 0.5$ 。算法3具体描述如下:

算法3 步长因子 α_k 计算算法

输入 $\mathbf{x}_k, \mathbf{p}_k, c = 0.9, \tau = 0.5$
 输出 α_k
 $\alpha_k \leftarrow 10$;
 while $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) - f(\mathbf{x}_k) > \alpha_k \times c \times \mathbf{g}_k^T \times \mathbf{p}_k$ do
 $\alpha_k \leftarrow \tau \alpha_k$;
end

在求解线搜索方向 \mathbf{p}_k 时, 需要 ALS 算法的一次迭代, 算法4是求解 ALS 一次迭代的算法 $H(\mathbf{x}_k)$ 的伪代码, 其中, 规定 $\bar{\mathbf{x}} = H(\mathbf{x}_k)$ 。将算法2输出结果 \mathbf{x}_k 作为算法4的输入, 首先根据 \mathbf{x}_k 求解得到 \mathbf{U} 和 \mathbf{M} , 然后求解出 u_i 和 m_j , 最后根据 u_i 和 m_j 得到目标值 $\bar{\mathbf{x}}_k (H(\mathbf{x}_k))$ 。搜索方向 \mathbf{p}_k 要求为下降的方向, 标准为 $\mathbf{p}_k^T \times \nabla f(\mathbf{x}_k) < 0$, 沿着 \mathbf{p}_k 方向找到比 \mathbf{x}_k 更优的点, 这样即可以确定 \mathbf{p}_k 。

算法4 ALS 一次迭代算法 $H(\mathbf{x}_k)$

输入 \mathbf{x}_k
 输出 $\bar{\mathbf{x}}_k$
 solve \mathbf{U} and \mathbf{M} from \mathbf{x}_k ;
 for $i = 1, 2, \dots, n_u$ do
 $u_i \leftarrow \mathbf{A}_i^{-1} \mathbf{v}_j$;
 End
 for $j = 1, 2, \dots, n_m$ do
 $m_j \leftarrow \mathbf{A}_j^{-1} \mathbf{v}_j$;
 End
 solve $\bar{\mathbf{x}}_k$ by the combination of the respective u_i and m_j

2.3 ALS-NCG 算法的 Spark 并行实现

将 ALS-NCG 算法的数据结构在 Spark 弹性分布式数据集 (Resilient Distributed Dataset, RDD) 上分区, 每个分区就是一个数据集片段, 并且一个 RDD 的不同分区可以被保存到集群中不同的节点上, 从而在集群中的不同节点上进行并行计算。本文将因子矩阵 \mathbf{U} 和 \mathbf{M} 存储为单精度列块矩阵, 分别表示用户

子集的因子矢量和电影子集的因子矢量。规定单精度列块的每一块作为 RDD 的一个分区单元, R 和 R^T 的行块均以压缩的稀疏矩阵的方式进行存储。图 1 所示为 M 存储的分配方式, 将 M 区块分割成若干块, 编号为 M_1, M_2, \dots, M_{nb} , 共分解成 nb 块。

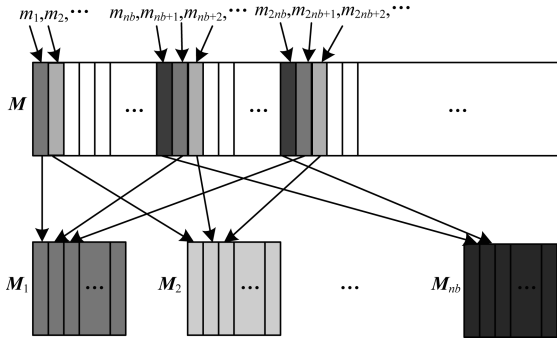


图 1 数据在 RDD 上的分区
Fig. 1 Data partition on RDD

U 的存储方式与 M 相同。 U 和 R 的 RDD 分区相同, 存储 U 的节点也可以存储 R 。在更新 U 用户块时, 需要本地评分在 R 中可用, 而本地 U 块中用户评分电影的 M 因子矢量必须混洗。由于电影因子来自不同的节点, 可通过建立路由表存储所需的本地评分数据, 避免重复发送信息。缓冲区一旦构造, 就会被混合集中到 R 的分区, 此时电影因子和本地评分数据可用来计算新的 U 子块。

路由表构建在 ALS 算法的 While 循环之前, 具体布局如图 2 所示, 每次迭代无需重新计算, 同理, 更新 M 与 U 类似。矢量 \bar{x} 、 \bar{g} 以及 P 均被存储在 2 个单独的 RDD 中, 使与其相关的分量存储在同一个 RDD, 且分割方式与 U 类似。路由表策略可以确保所有的矢量块在矢量运算中按照成分对齐, 从而实现了各数据的 RDD 存储以及 RDD 的分配运行。

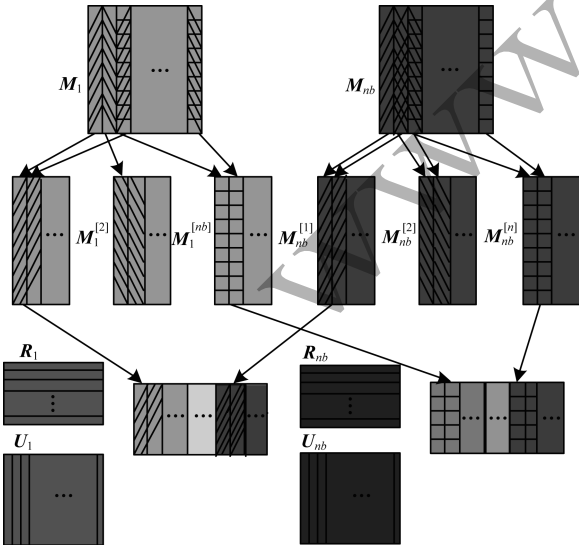


图 2 RDD 路由表策略
Fig. 2 RDD routing table strategy

3 实验结果与分析

3.1 实验数据集

本文采用 MovieLens 上 10 M 大小的数据集进行实验, 该数据集包括 71 567 个用户、10 681 部电影和 10 000 054 条评分, 为更好地比较算法性能, ALS 与 ALS-NCG 均使用相同的数据集。在实际应用中, 并非所有数据都满足建立实验数据集的要求, 因此, 本文找到原始完整数据集中按照每个用户评分数目进行排序的中位数, 用 c 表示, 舍弃对电影评分数目与中位数值相差较大的用户, 保留评分数目与中位数相近的用户, 将这些数据组成实验数据集。

3.2 Spark 运行环境

Spark 是一种类 Hadoop MapReduce 的基于内存计算的大数据并行计算框架^[21], Spark 较 Hadoop 的优势之一是提出了 RDD。RDD 的高容错性表现在 2 个方面: 1) 若其中一个 RDD 分片丢失, 则 Spark 可以根据日志信息重构 RDD; 2) 在内存中缓存 RDD 后, 只需从内存中读取数据即可计算, 这样节省了大量的磁盘存取开销, 适合迭代计算的矩阵分解算法。因此, 本文搭建 Spark 集群作为实验平台。

本文将 Spark 计算集群搭建在 Hadoop 分布式平台上, 以 Hadoop 中的 HDFS 作为集群的分布式文件存储系统, 选择 Spark 原生语言 Scala^[22] 作为编程语言, 该语言比较简洁完善, 是 Spark 领域较为常用的程序设计语言。本文实验的集群环境配置如表 1 所示。

表 1 实验环境配置

Table 1 Experimental environment configuration	
软硬件环境	版本设置
Spark 版本	Spark-2.7.0
操作系统	Ubuntu16.0.4
JDK	JDK-1.8
Scala 语言	Scala-2.10.7
Hadoop	Hadoop-2.9.0
Master 节点 (1 个)	四核 CPU、4 GB 内存
Worker 节点 (3 个)	四核 CPU、4 GB 内存
IntelliJ IDEA	IdeaIC-14.0.2

3.3 ALS-NCG 算法与 ALS 算法时间性能比较

在本次实验中, 算法循环的终止条件是期望的收敛值 $\frac{\|g_k\|}{N}$ 小于等于给定值, $N = n_f \times (n_u + n_m)$ 。

实验 1 进行 ALS-NCG 算法与 ALS 算法的时间性能比较, 采用若干不同规格的数据集在 Spark 平台上反复实验, 以验证 ALS-NCG 算法是否能达到预期目标。分别统计 ALS-NCG 与 ALS 2 种算法达到期望收敛值 $\frac{\|g_k\|}{N} \leq 10^{-6}$ 和 $\frac{\|g_k\|}{N} \leq 10^{-3}$ 时各自所需的时间, 用 n_u 表示用户数量, n_m 表示电影数量, 时间对比结果如表 2、表 3 所示。其中, 加速倍数表示 ALS 与 ALS-NCG 的时间比值。

表 2 收敛值为 10^{-6} 时 2 种算法的时间性能对比Table 2 Time performance comparison of two algorithms when convergence value is 10^{-6}

评分矩阵规模	时间/s		加速倍数
	ALS	ALS-NCG	
100 × 20	9.08	1.98	4.59
200 × 40	22.56	5.80	3.89
400 × 80	54.14	12.02	4.50
800 × 160	162.20	47.55	3.41
1 600 × 320	390.55	142.71	2.74
3 200 × 640	960.82	303.72	3.16
6 400 × 1 280	2 275.98	859.93	2.65

表 3 收敛值为 10^{-3} 时 2 种算法的时间性能对比Table 3 Time performance comparison of two algorithms when convergence value is 10^{-3}

评分矩阵规模	时间/s		加速倍数
	ALS	ALS-NCG	
100 × 20	1.31	0.42	3.12
200 × 40	3.67	1.51	2.43
400 × 80	7.74	2.49	3.11
800 × 160	23.21	10.41	2.23
1 600 × 320	65.10	35.43	1.84
3 200 × 640	136.98	56.71	2.42
6 400 × 1 280	397.03	181.11	2.19

从表 2、表 3 可以看出, ALS-NCG 达到目标收敛值较快, 相对 ALS 的加速效果明显。

图 3 表示不同规模的评分矩阵下 2 种算法在迭代次数相同时所用时间比值的变化趋势, 可以看出, ALS-NCG 在时间消耗上比 ALS 小很多, 随着矩阵规模的增大, ALS-NCG 的加速效果更明显, 性能更加优越。在不同收敛值时, ALS 与 ALS-NCG 的时间性能比值变化趋势近似一致, 可见, 将改进的组合算法应用于大数据环境中, 即使收敛值扩大到 10^{-3} , 也可以达到同样的加速效果。

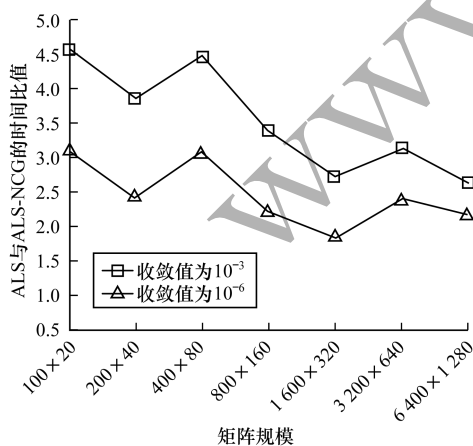


图 3 不同收敛值时 ALS 与 ALS-NCG 的时间比值对比

Fig. 3 Time ratio comparison between ALS and ALS-NCG with different convergence values

3.4 ALS-NCG 算法与 ALS 算法排名精度比较

由于实验条件限制, 本文仅对排名前 20 的电影进行准确度实验, 使用电影排名矢量转化所需的成对互换次数作为度量。本次实验借助 Kendall-Tau 距离来计算向量之间的差异, 将距离归一化范围限制在 $[0, 1]$ 之间, 再求取所有用户的距离平均值。

若 $p_1 = [6, 3, 1, 4, 2, 5]$, $p_2 = [3, 4, 5, 2, 6, 1]$ 是用户 u 对 2 个不同电影的排名, 设置 $t=4$ 表示对排名前 4 的电影进行排名精度计算。 p_1 中排名第 1 的电影 6 在 p_2 中排名第 5, 若要将 p_2 中电影 6 排在第 1 位, 需要交换 4 次位置, 此时产生第 1 次迭代排名顺序 $p_{21} = [6, 3, 4, 5, 2, 1]$, p_1 中排名第 2 位的是电影 3, 在 p_{21} 中电影 3 同样排在第 2 位, 此时不需要转换, 即 $p_{22} = p_{21}$, 同理可得 $p_{23} = [6, 3, 1, 4, 5, 2]$, 转换 3 次, $p_{24} = p_{23}$, 转换 0 次。匹配 p_2 到 p_1 所需成对互换的距离总数 $s_i = 4 + 0 + 3 + 0 = 7$ 。若匹配时相同位置的電影相同, 则不需要转换, 否则将会发生转换, 若匹配与被匹配的序列相反, 则会出现最大交换次数, 在第 1 次匹配发生 n_{m-1} 次交换, 第 2 次匹配发生 n_{m-2} 次交换, 以此类推, 最大交换次数可表示为:

$$s_{\max} = (n_m - 1) + (n_m - 2) + \cdots + (n_m - t) = \frac{1}{2}(2n_m - t - 1)$$

根据最大交换次数公式可以推导出排名精度指

标为: $p_i = 1 - \frac{s_i}{s_{\max}}$, 将该计算指标应用于接下来的实验中。

实验 2 选取 400×80 和 800×160 两种规模的评分矩阵, 2 种算法都以 20 个不同的随机起始值进行迭代运行求解, 达到给定的不同排序精度, 记录时间数据, 结果如表 4、表 5 所示。从表 4、表 5 可以看出, 在排序精度为 70% 时, ALS 对于 2 种规模矩阵的收敛速度相对较快, 而在排序精度大于 70% 时, ALS 收敛时间较长, ALS-NCG 的收敛时间比 ALS 算法小很多, 其效率较高。

表 4 400×80 矩阵规模下 2 种算法的时间对比Table 4 Time comparison of two algorithms under 400×80 matrix scale

排序精度/%	时间/s		加速倍数
	ALS	ALS-NCG	
70	0.36	0.65	0.55
75	3.17	1.63	1.94
80	7.84	2.79	2.81
85	19.20	3.62	5.30
90	37.02	6.91	5.36
95	60.20	9.43	6.38
100	101.32	14.10	7.19

表 5 800×160 矩阵规模下 2 种算法的时间对比
Table 5 Time comparison of two algorithms under 800×160 matrix scale

排序精度/%	时间/s		加速倍数
	ALS	ALS-NCG	
70	0.75	1.44	0.52
75	5.32	4.89	1.09
80	19.78	12.62	1.57
85	56.15	20.12	2.79
90	103.89	33.27	3.12
95	224.33	53.22	4.22
100	309.91	60.10	5.16

图 4、图 5 分别呈现矩阵规模为 400×80 、 800×160 时 2 种算法的运行时间趋势。从中可以看出,随着排序精度的提高,ALS-NCG 算法的时间消耗变化较为平缓,更符合实际需求,而 ALS 算法需要更长的时间才能达到目标精度。

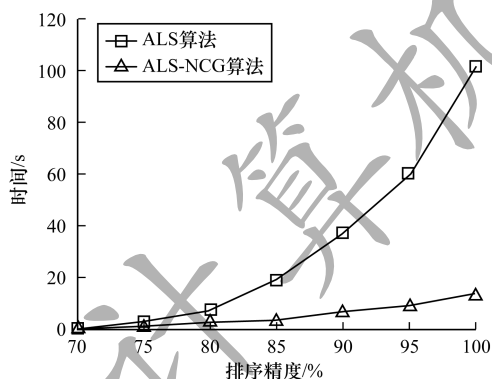


图 4 矩阵规模为 400×80 时的算法时间趋势对比

Fig. 4 Comparison of algorithm time trend when matrix scale is 400×80

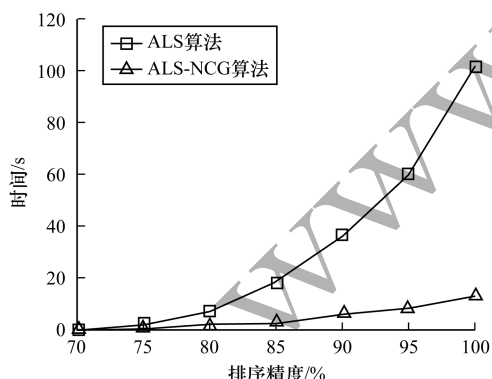


图 5 矩阵规模为 800×160 时的算法时间趋势对比

Fig. 5 Comparison of algorithm time trend when matrix scale is 800×160

3.5 ALS-NCG 算法与 ALS 算法性能指标比较

本文采用推荐系统领域常用的均方根误差 (Root Mean Squared Error, RMSE) 来度量算法的性

能。RMSE 是预测值与真实值偏差的平方与预测值的比值的平方根,其值越小说明测量的精确度越高。

ALS-NCG 算法在执行速度与排序精度上都优于 ALS 算法,实验 3 将在 ALS 训练的最优参数模型的基础上对 2 种算法的 RMSE 值进行求解,其中,ALS 算法最佳迭代次数为 25,实验结果如表 6 所示。从表 6 可以看出,ALS-NCG 算法达到与 ALS 算法相近的 RMSE 值仅需迭代 7 次,迭代次数明显减少。

表 6 2 种算法的 RMSE 值对比
Table 6 RMSE comparison of two algorithms

正则化 参数值	RMSE 值		
	ALS (迭代 25 次)	ALS-NCG (迭代 7 次)	ALS-NCG (迭代 25 次)
0.050	0.804 8	0.804 5	0.790 8
0.055	0.803 7	0.803 5	0.793 3
0.060	0.802 8	0.802 7	0.792 9
0.065	0.803 1	0.803 3	0.792 8
0.070	0.804 6	0.804 1	0.794 2

3.6 ALS-NCG 算法与其他算法性能指标比较

将 ALS-NCG 算法与受限玻尔兹曼机 (RBM)、 $k=21$ 时的 k 近邻 (kNN) 算法、ALS 算法、ALS + kNN + RBM 组合方法以及均值模型的 RMSE 值进行比较,结果如表 7 所示。从表 7 可以看出,ALS-NCG 算法具有明显优势,它相对其他算法的 RMSE 改善率最高可达 25.17%。

表 7 不同方法的 RMSE 值比较
Table 7 RMSE comparison between different methods

对比方法	RMSE 值	ALS-NCG 相对该 算法的 RMSE 改善率/%
RBM	0.854 4	7.20
kNN	0.860 9	7.90
ALS	0.802 8	1.23
ALS + kNN + RBM	0.802 6	1.21
均值模型	1.059 6	25.17
ALS-NCG	0.792 9	—

ALS 算法虽然应用于推荐系统时效果较好,但是其迭代时间过长,影响推荐系统性能。NCG 算法数值表现良好,并且具有较高的收敛性,可以加速 ALS 算法的收敛,提高推荐效果。虽然 ALS-NCG 算法的每一步迭代复杂度较高,但是算法整体收敛程度增大,使得总迭代次数减少,在 Spark 环境中,其收敛速度明显高于 ALS 算法,当两者的 RMSE 值接近时,ALS-NCG 算法的迭代次数仅为 ALS 算法的 1/3 左右。

4 结束语

基于矩阵分解的 ALS 算法应用于推荐系统时, 执行速率与排名精度均较低。本文分析 ALS 算法自身存在的问题, 提出一种融合 NCG 与 ALS 的协同过滤推荐算法 ALS-NCG, 并在 Spark 集群环境中进行实验, 结果表明, ALS-NCG 算法在收敛速度和排名精度上都优于 ALS 算法。下一步将通过卷积神经网络对用户评论中的文本特征进行学习, 并将提取的文本特征引入到 ALS-NCG 算法中, 以进一步改善推荐效果。

参考文献

- [1] GANTZ J, REINSEL D. The digital universe in 2020: big data, bigger digital shadows, and biggest growth in the far east [EB/OL]. [2019-02-10]. <https://www.speicherguide.de/download/dokus/IDC-Digital-Univers-Studie-iView-11.12.pdf>.
- [2] ISINKAYE F O, FOLAJIMI Y O, OJOKOH B A. Recommendation systems: principles, methods and evaluation [J]. Egyptian Informatics Journal, 2015, 16(3): 261-273.
- [3] KOREN Y, BELL R, VOLINSKY C. Matrix factorization techniques for recommender systems[J]. Computer, 2009, 42(8): 30-37.
- [4] LIU Yingnan, XIE Jinkui, ZHANG Jiali, et al. Recommendation algorithm based on trust in social network [J]. Journal of Chinese Computer Systems, 2015, 36(6): 1165-1170. (in Chinese)
刘英南, 谢瑾奎, 张家利, 等. 社交网络中基于信任的推荐算法[J]. 小型微型计算机系统, 2015, 36(6): 1165-1170.
- [5] BOKDE D, GIRASE S, MUKHOPADHYAY D. Matrix factorization model in collaborative filtering algorithms: a survey[J]. Procedia Computer Science, 2015, 49(1): 136-146.
- [6] MENG Xiangwu, LIANG Bi, DU Yulu, et al. Study on the utility evaluation of location-based mobile recommendation system[J]. Chinese Journal of Computers, 2019, 42(12): 2695-2721. (in Chinese)
孟祥武, 梁弼, 杜雨露, 等. 基于位置的移动推荐系统效用评价研究[J]. 计算机学报, 2019, 42(12): 2695-2721.
- [7] HERNANDO A, BOBADILLA J, ORTEGA F. A non negative matrix factorization for collaborative filtering recommender systems based on a Bayesian probabilistic model [J]. Knowledge-Based Systems, 2016, 97(4): 188-202.
- [8] DU Dongfang, XU Tong, LU Yanan, et al. User rating prediction based on trust-driven probabilistic matrix factorization[J]. Journal of Software, 2018, 29(12): 3747-3763. (in Chinese)
杜东舫, 徐童, 鲁亚男, 等. 基于信任机制下概率矩阵分解的用户评分预测[J]. 软件学报, 2018, 29(12): 3747-3763.
- [9] KOREN Y. Collaborative filtering with temporal dynamics[J]. Communications of the ACM, 2010, 53(4): 447-456.
- [10] DESHPANDE M, GEORGE K. Item-based top-n recommendation algorithms [J]. ACM Transactions on Information Systems, 2004, 22(1): 143-177.
- [11] WANG Jianfang, MIAO Yanling, HAN Pengfei, et al. Probabilistic matrix factorization algorithm of collaborative filtering based on trust mechanism [J]. Journal of Chinese Computer Systems, 2019, 40(1): 31-35. (in Chinese)
王建芳, 苗艳玲, 韩鹏飞, 等. 一种基于信任机制的概率矩阵分解协同过滤推荐算法[J]. 小型微型计算机系统, 2019, 40(1): 31-35.
- [12] CONDIE T, MINEIRO P, POLYZOTIS N, et al. Machine learning on big data [C]//Proceedings of 2013 IEEE International Conference on Data Engineering. Washington D. C., USA: IEEE Press, 2013: 1242-1244.
- [13] TU Xiaohan, LIU Siping, LI Renfa. Improving matrix factorization recommendations for problems in big data [C]//Proceedings of 2017 IEEE International Conference on Big Data Analysis. Washington D. C., USA: IEEE Press, 2017: 193-197.
- [14] THAN H A, RACHSUDA J T. Alternating least squares with incremental learning bias [C]//Proceedings of 2015 International Joint Conference on Computer Science and Software Engineering. Washington D. C., USA: IEEE Press, 2015: 297-302.
- [15] STERCK H D, WINLAW M. A nonlinearly preconditioned conjugate gradient algorithm for rank-R canonical tensor approximation [J]. Numerical Linear Algebra with Applications, 2015, 22(3): 410-432.
- [16] YE Hanmin, ZHANG Qiuling, BAI Xue. A new collaborative filtering algorithm based on modified matrix factorization [C]//Proceedings of 2017 IEEE Advanced Information Technology Electronic and Automation Control Conference. Washington D. C., USA: IEEE Press, 2017: 147-151.
- [17] ZHOU Y H, WILKINSON D, SCHREIBER R, et al. Large-scale parallel collaborative filtering for the netflix prize [C]//Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management. Berlin, Germany: Springer, 2008: 337-348.
- [18] QI Changxia. Study on several fusion nonlinear conjugate gradient methods [D]. Qinhuangdao: Yanshan University, 2017. (in Chinese)
齐昌霞. 几种融合非线性共轭梯度法的研究 [D]. 秦皇岛: 燕山大学, 2017.
- [19] LIU Zhengrong. Several new nonlinear conjugate gradient methods [D]. Nanning: Guangxi University, 2016. (in Chinese)
刘峥嵘. 几种新的非线性共轭梯度法 [D]. 南宁: 广西大学, 2016.
- [20] ZAHARIA M, CHOWDHURY M, DAS T, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computers [C]//Proceedings of USENIX Conference on Networked Systems Design and Implementation. San Diego, USA: USENIX Association, 2012: 141-146.
- [21] WU Xindong, JI Shengwei. Comparative study on MapReduce and Spark for big data analytics [J]. Journal of Software, 2018, 29(6): 1770-1791. (in Chinese)
吴信东, 嵇圣巍. MapReduce 与 Spark 用于大数据分析之比较 [J]. 软件学报, 2018, 29(6): 1770-1791.