

文章编号:1007-5321(2018)06-0115-08

DOI:10.13190/j.jbupt.2018-028

基于粒子群优化算法的协同过滤推荐并行化研究

游思晴, 周 丽, 赵东杰, 薛 菲

(北京物资学院 信息学院, 北京 101149)

摘要: 针对常用协同过滤推荐算法存在计算性能瓶颈的问题,提出了在 Spark 上并行化实现协同过滤推荐算法 RLPSO_KM_CF. 首先,通过具备反向学习和局部学习能力的粒子群优化(RLPSO)算法寻找粒子群最优解,输出优化后的聚类中心;然后,运用 RLPSO_KM 算法对用户信息进行聚类;最后,将传统协同过滤推荐算法与 RLPSO_KM 聚类结合,从而对目标用户进行有效推荐. 实验结果显示, RLPSO_KM_CF 算法在推荐准确度方面有显著提高,具有较高的加速比,稳定性也得到了一定提升.

关 键 词: 协同过滤推荐算法; RLPSO 算法; K-means 算法; Spark

中图分类号: P315.69

文献标志码: A

Research on Parallelization of Collaborative Filtering Recommendation Algorithm Based on Particle Swarm Optimization

YOU Si-qing, ZHOU Li, ZHAO Dong-jie, XUE Fei

(School of Information, Beijing Wuzi University, Beijing 101149, China)

Abstract: In order to solve the computational performance bottleneck of the commonly used collaborative filtering recommendation algorithm, a parallel collaborative filtering recommendation algorithm RLPSO_KM_CF on Spark is proposed. Firstly, the reverse-learning and local-learning particle swarm optimization (RLPSO) algorithm is used to find the optimal solution of the particle swarm and the output clustering center is optimized. Then, the RLPSO_KM algorithm is used to cluster the user information. Finally, the traditional cooperative filtering recommendation algorithm is combined with the RLPSO_KM cluster to effectively recommend the target user. The experimental results show that the improved algorithm has a significant improvement in the recommended accuracy, and has a higher speedup and stability.

Key words: collaborative filtering recommendation algorithm; RLPSO algorithm; K-means algorithm; Spark

随着信息技术快速发展,信息化数据急剧增加. 推荐系统通过学习用户的行为,了解和掌握用户的偏好,从而可以更有针对性地向用户推荐内容. 在众多推荐算法中,协同过滤是目前应用最多的算法. 协同过滤推荐算法借助目标用户的偏好信息来计算与目标用户相似的邻域用户集,进而将有效项目推

荐给目标用户^[1]. 然而,推荐算法在推荐过程中仍然会有冷启动、数据稀疏性,以及面对大数据时存在的计算性能问题.

随着并行数据处理工具与数据处理技术不断发展,研究者开始尝试在分布式计算平台上实现协同过滤推荐算法. Xu 等^[2]通过将用户和项目信息聚

收稿日期: 2018-01-26

基金项目: 国家自然科学基金项目(71501015);北京市智能物流系统协同创新中心开放课题

作者简介: 游思晴(1982—),女,讲师, E-mail: 93028603@qq.com.

类,形成若干个用户-项目子群. Jiang 等^[3]在 MapReduce 上开发并实现了一个基于项目的可扩展协同过滤推荐算法,将运算过程中最消耗计算的 3 个步骤划分为 4 个 MapReduce 阶段. Zhao 和 Shang^[4]在 Hadoop 平台上实现了基于用户的协同过滤推荐算法,解决了可扩展性问题. 李改等^[5]提出将基于 ALS (alternating-least-squares) 的协同过滤推荐算法在 Hadoop 上进行并行化,解决了传统的基于 ALS 的协同过滤推荐算法在大规模数据集上的运算问题. 以上研究证实,利用 MapReduce 计算框架实现并行化基于邻域的协同过滤推荐算法是可行的,但是 MapReduce 仍存在一系列的缺陷:只提供 Map 和 Reduce 2 个操作,在表达力上有所欠缺,复杂计算需要大量的 job 完成,而 job 之间的依赖关系需要由开发者自己管理;中间结果存放在 Hadoop 分布式文件系统中,需要多次 IO 操作^[6]等. Winlaw 等^[7]采用并行 ALS 优化融合方法进行协同过滤,并基于 Spark 并行化实现该方法,实验结果显示该方法的加速比随着数据集的增加呈线性增长. Kupisz 和 Unold^[8]采用 Tanimoto 系数计算相似度,并通过比较基于 Hadoop 和 Spark 的运行效率,从而验证了基于 Spark 平台的推荐更有效率. Li 和 Liu^[9]提出一种基于 Spark 的混合推荐算法 PRABS (personalized recommendation algorithm based on spark),通过加权基于用户和基于内容的协同过滤推荐算法,提高了推荐效率和推荐精度. Tu 等^[10]提出一种基于 Spark 的加权正则化交替最小二乘法的协同过滤推荐算法 NALS-WR (normalized-alternating-least-squares with weighted- λ -regularization),通过对比基于 ALS-WR (alternating-least-squares with weighted- λ -regularization) 和 SVD (singular value decomposition) 的推荐系统,该算法的推荐准确率显著提高. Hammou 等^[11]提出一种基于 Spark 的专门处理大规模数据的推荐算法 APRA (approximate parallel recommendation algorithm),通过使用 Spark 基于内存的操作,大大提高了大数据环境下的推荐性能.

粒子群优化 (PSO, particle swarm optimization) 算法是由 Kennedy 和 Eberhart^[12]提出的一种基于群体智能的随机优化方法. RLPSO (reverse-learning and local-learning PSO) 算法是一种改进的 PSO 算法,该算法通过粒子种群历史位置的差分结果进行局部搜索,同时引入反向学习子粒子种群^[13]避免出现早熟收敛.

为了解决协同过滤推荐算法针对大数据带来的计算性能问题以及结合 Spark 在内存计算和迭代计算上的优势,提出了一种基于 PSO 算法的并行化协同过滤推荐算法. 该算法根据与目标用户具有相似兴趣的用户群比其他用户具有更高的参考价值,提出基于 KM (K-means) 聚类算法对评分数据进行聚类分析,从而将兴趣相似的用户聚成一类,通过类内的相似用户运用推荐算法预测目标用户对其他未评价物品的评分,进而将其推荐给目标用户. 本文算法针对 KM 聚类算法初始化因子的不确定性,通过 RLPSO 算法对其初始化因子进行优化.

笔者的主要研究工作和贡献点如下:① 针对协同过滤推荐算法面对大数据存在的计算性能问题,提出了基于 Spark 的并行化协同过滤推荐算法;② 针对协同过滤推荐算法中,与用户兴趣相似的用户具有更高的参考价值,提出了对用户进行聚类分析;③ 针对 KM 聚类算法中初始化因子的不确定性,提出了基于 RLPSO 算法对初始化因子进行优化.

1 基于 Spark 的 RLPSO_KM_CF 算法相关工作及算法实现

下面对所提出的并行化协同过滤推荐算法进行详细描述.

1.1 基于 RLPSO 的 RLPSO_KM 算法

1.1.1 RLPSO 算法^[13]

1) 粒子速度和位置更新公式为

$$v_{ij}^{t+1} = \omega v_{ij}^t + c_1 r_1 (p_{ij}^t - x_{ij}^t) + c_2 r_2 (p_{gj}^t - x_{ij}^t) \quad (1)$$

$$x_{ij}^{t+1} = x_{ij}^t + v_{ij}^t \quad (2)$$

其中: $v_{ij}^t, x_{ij}^t, p_{ij}^t$ 表示第 i 个粒子在第 j 维在第 t 代的速度、位置和最优位置, c_1 和 c_2 表示粒子个体向个体和种群历史最优解学习能力的大小, r_1 和 r_2 为 $[0, 1]$ 的随机数, ω 为惯性权重:

$$\omega^t = \omega_{\max} - (\omega_{\max} - \omega_{\min}) t / t_{\max} \quad (3)$$

其中: $t, t_{\max}, \omega_{\max}, \omega_{\min}$ 表示当前迭代次数、最大迭代次数、最大迭代次数时的惯性权重和初代惯性权重.

2) 初始化最差粒子种群 W . 算法根据粒子间的排斥半径判定该粒子是否加入最差粒子种群.

3) 局部搜索. 定义局部搜索的公式为

$$P'_{g1} = P_1 + rd_t (P_{g1} - P_{g2}) \quad (4)$$

其中: r 为 $[-1, +1]$ 的随机数,服从均匀分布; P_{g1} 和 P_{g2} 分别为粒子群的最优和次优位置; d_t 为第 t 次局域缩放因子,其取值为

$$d_{t+1} = d_t (1 - t / t_{\max}) \quad (5)$$

4) 粒子反向学习. 当算法进入停滞状态时, 根据初始化最差粒子种群并结合粒子种群中的一部分粒子进行反向学习. 同时定义第 i 个粒子进行反向学习的速度公式如下:

$$v_{ij}^{t+1} = \omega v_{ij}^t + c_3 r_3 (x_{ij}^t - w_{ij}^t) + c_4 r_3 (x_{ij}^t - w_{kj}^0) \quad (6)$$

其中: $w_{ij}^t (1 \leq i \leq n)$ 为第 i 个粒子 t 次迭代历史最差位置的第 j 维取值; c_3 和 c_4 为反向学习因子, r_3 为 $[0, 1]$ 的随机数; $w_{kj}^0 (1 \leq i \leq m)$ 为粒子群中最差粒子个体位置 w_k^0 第 j 维所取的值.

1.1.2 正则化飞行时间

为了避免粒子在最优解周围出现振荡现象, 加快粒子群的收敛时间, 加入了飞行时间因子, 从而粒子的位置更新公式调整为

$$x_{ij}^{t+1} = x_{ij}^t + H_0 (1 - t/t_{\max}) v_{ij}^{t+1} \quad (7)$$

其中 H_0 为飞行时间因子.

1.1.3 适应度函数

定义粒子的适应度函数为

$$f(x) = \sum_{j=1}^k \sum_{S_i \in C_j} \|S_i - Z_j\| \quad (8)$$

其中: S_i 为聚类 C_j 的一个数据项, Z_j 为聚类 C_j 的中心. $f(x)$ 越小, 聚类效果越好. 粒子 i 在第 $t+1$ 次迭代时, 若 $f(x_i(t+1)) < f(P_i(t))$, 则 $P_i(t+1) = f(x_i(t+1))$; 否则 $P_i(t+1) = P_i(t)$. 同样, 若 $\min(f(P_i(t+1))) < f(P'_{g1}(t))$, 则 $P'_{g1}(t+1) = \min(f(P_i(t+1)))$; 否则 $P'_{g1}(t+1) = P'_{g1}(t)$.

1.1.4 适应度方差

通过适应度方差表征粒子群的收敛度. 适应度方差定义为

$$\rho = 1/N \sum_{i=1}^N [f(x_i) - f_{\text{avg}}]^2 \quad (9)$$

其中: N 为粒子群规模, $f(x_i)$ 为粒子 i 的适应度值, f_{avg} 为所有粒子的适应度均值. ρ 越小, 整个粒子群越收敛.

1.1.5 算法描述

输入: 实验数据集 D , 聚类数目 k , 粒子群规模 N 和反向学习粒子种群规模 n , 粒子群学习因子 c_1 、 c_2 和反向学习因子 c_3 、 c_4 , 粒子群最大迭代次数 t_{\max} 和反向学习迭代次数 L_{times} , 最大惯性权重 ω_{\max} 和最小惯性权重 ω_{\min} , 扰动系数 d_0 , 飞行时间因子 H_0 和粒子最大飞行速度 v_{\max} .

输出: 优化后的 k 个聚类中心.

Begin

1 初始化粒子群. 从数据集 D 中随机选择 k 个数

据项作为粒子位置和速度每一维值的初始值, 将粒子位置的个体最优位置设为 X_i^0 , 循环执行这一过程 N 次.

2 初始化粒子群最优位置和次优位置. 根据式(8)计算粒子群中每个粒子的适应度值, 并依次选择适应度值最小和次小粒子的位置作为粒子群体最优和次优位置的初始值.

3 初始化最差粒子种群 W .

4 迭代搜索粒子.

while ($t < t_{\max} \parallel \rho < 10^{-6}$)

1) 根据式(3), 调整惯性权重;

2) 根据式(7)和式(1)分别对粒子的位置和粒子的速度进行更新;

3) 根据式(8)依次计算粒子群中每个粒子的适应度值;

4) 更新粒子的个体最优值;

5) 更新粒子群次优和最优位置;

6) 根据式(4)计算粒子群历史位置的差分结果并进行局部搜索;

7) 根据式(5)调整缩放因子权重;

8) If 满足反向学习条件 (算法局部收敛或者达到设定的阈值收敛次数), 调整粒子速度 v_{\max} ; 根据式(6)和式(7)分别更新反向学习粒子的速度和位置; 根据式(7)和式(1)更新除反向学习的剩余粒子的位置和速度;

End If

9) 根据式(9)计算适应度方差 ρ ;

10) if (适应度方差 > 阈值)

break;

11) $t++$;

End while

5 输出粒子群最优解.

6 运行 KM 聚类算法并输出优化后的聚类中心.

End

1.2 基于 RLPSO_KM 的 RLPSO_KM_CF 算法

根据与目标用户相似性更高的用户比其他用户更具有价值的参考作用, 采用 RLPSO_KM 算法对用户信息进行聚类, 然后在每个聚类内部通过运用传统基于用户的协同过滤推荐算法对目标用户进行有效推荐. 同时, 给新加入的目标用户推荐流行度最高的项目. 项目流行度公式为

$$M_i = \frac{|U_i|}{\sqrt{\sum_{i \in I} |U_i|^2}} \quad (10)$$

其中: M_i 为项目 i 流行度, 定义项目流行度与浏览过该项目的用户量成正比, U 表示用户集合 $\{u_1, u_2, u_3, \dots, u_i\}$, I 表示项目集合 $\{i_1, i_2, i_3, \dots, i_n\}$;

RLPSO_KM_CF 算法描述如下:

输入: 聚类因子 k 和聚类迭代次数 m , 用户-项目评分信息和推荐项目个数 N .

输出: 对目标用户的 Top- N 推荐.

Begin

1 If (目标用户是否是新用户)

1) 根据式(10)计算项目的流行度形成集合 W ;

2) 对集合 W 进行降序排序形成集合 W_{new} ;

3) 从集合 W_{new} 中选取前 N 个流行度最高的项目, 从而形成集合 Target;

4) 将集合 Target 推荐给目标用户;

End If

2 根据 RLPSO_KM 算法计算聚类中心.

3 根据式(11)用户相似度公式计算目标用户所属聚类.

$$\text{sim}(u_i, u_j) = \frac{\sum_{i_c \in I_{i,j}} (R_{u_i, i_c} - \bar{R}_{u_i})(R_{u_j, i_c} - \bar{R}_{u_j})}{\sqrt{\sum_{i_c \in I_{i,j}} (R_{u_i, i_c} - \bar{R}_{u_i})^2} \sqrt{\sum_{i_c \in I_{i,j}} (R_{u_j, i_c} - \bar{R}_{u_j})^2}} \quad (11)$$

其中: \bar{R}_{u_i} 和 \bar{R}_{u_j} 分别为用户 u_i 和 u_j 对所有项目的平均评分; R_{u_i, i_c} 和 R_{u_j, i_c} 分别为用户 u_i 和 u_j 对项目 i_c 的评分. 用户对项目的预测评分公式为

$$\beta(u_i, i_i) = \bar{R}_{u_i} + \frac{\sum_{u_j \in N_{u_i}} \text{sim}(u_i, u_j) (R_{u_j, i_i} - \bar{r}_{u_j})}{\sum_{u_j \in N_{u_i}} \text{sim}(u_i, u_j)} \quad (12)$$

其中 N_{u_i} 为用户 u_i 的邻域集合.

4 聚类中运用协同过滤推荐算法为目标用户进行推荐.

5 输出 Top- N 推荐列表.

End

1.3 基于 Spark 的 RLPSO_KM_CF 算法

RLPSO 算法中, 每个粒子的迭代过程相互独立且每个粒子迭代相同任务, 加大了处理的任务粒度, 而随着任务粒度增加, 计算时间也相对越长. KM 聚类算法中, 由于每个数据对象彼此独立, 计算每个数据对象所属的聚类, 随着聚类迭代次数增加, 需要不断更新聚类中心. 针对以上情况, 提出基于 Spark 的

RLPSO_KM_CF 算法, 从而有效地支持迭代运算, 提高算法效率. RLPSO_KM_CF 算法的具体程序流程如图 1 所示.

RLPSO_KM_CF 算法分为 3 个阶段.

第 1 阶段. 针对 RLPSO 算法进行并行化设计与实现. 通过将粒子群映射到 RDD 数据集, 从而将构造粒子群最优解的过程进行并行化, 将每个粒子群初始化粒子群信息、构造解、局部搜索和反向学习的过程独立成为一个并行单元. 在 Spark 分布式处理平台上, 并行处理 RDD1 至 RDD n 个粒子群信息, 从而求解每个粒子群的最优解, 最终更新整体粒子群最优解.

第 2 阶段. 在第 1 阶段输出粒子群最优解的基础上, 将粒子群最优解进行有效分解, 从而作为 RLPSO_KM 算法的初始化聚类中心. 在 Spark 上, 利用 RDD 将数据样本分布至集群中的 n 个计算节点, 并且将聚类中心借助 SparkContext 的 broadcast() 函数使其在各个计算节点间实现共享. 与此同时, 针对每个数据分片, 计算其与最近聚类中心的欧氏距离, 并根据每个聚类中心计算属于该聚类的数据之和, 对每个节点的累加数据和进行合并求和, 判断 sum 和是否小于阈值, 通过使用 map() 和 reduce() 函数完成更新聚类中心的操作, 直到算法收敛或完成迭代从而输出聚类中心和聚类结果.

第 3 阶段. 在第 2 阶段基础上, 首先对目标用户进行判断, 如果是新加入用户, 根据式(10)计算项目流行度最高的 N 个项目, 从而推荐给目标用户; 否则, 通过 map() 函数对评分信息进行排序同时执行持久化操作. 然后根据聚类中心计算用户所属聚类, 使用 groupByKey() 函数获取目标用户评分的项目信息 RDD1; 根据式(11)计算聚类内用户之间的相似度, 从而通过 filter() 函数过滤与目标用户相似度最高的用户信息 RDD2; 使用 groupByKey() 函数获取目标用户邻域的用户评价过的项目信息 RDD3; 通过将 RDD1 与 RDD3 进行 distinct() 操作, 返回目标用户未评价过的项目信息 RDD4. 最后根据式(12)计算目标用户对 RDD4 中项目信息的预测评分, 将评分最高的 N 个项目推荐给目标用户.

2 实验

2.1 实验数据集

实验环境为 centos7.0 设备系统服务器, 部署了

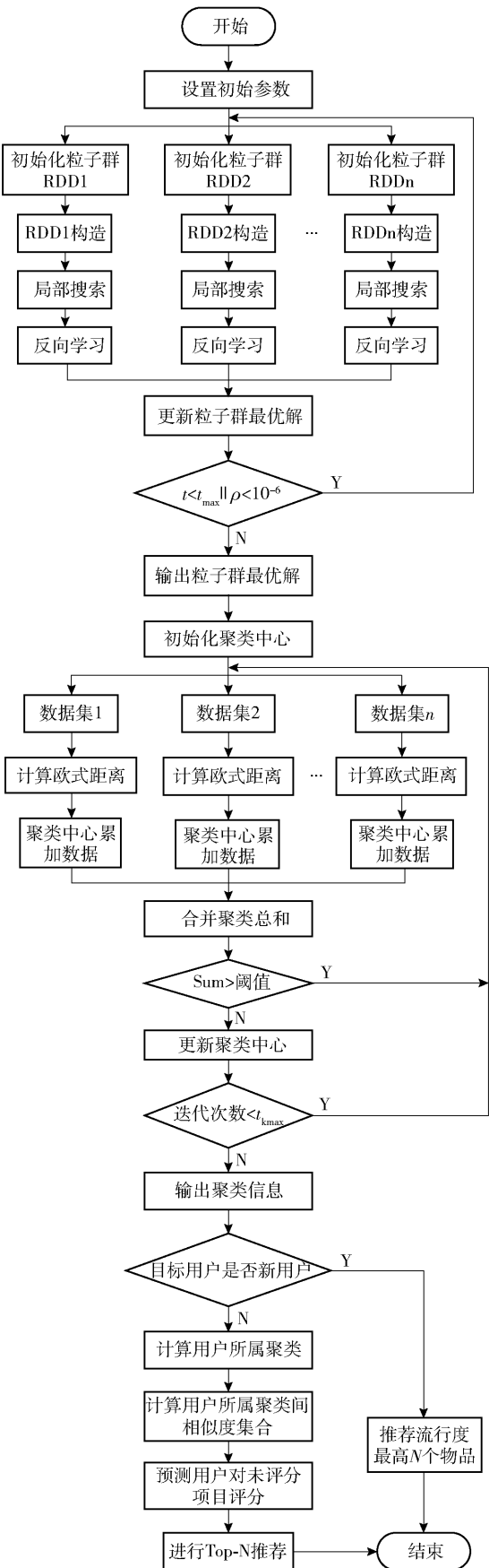


图 1 基于 Spark 的 RLPSO_KM_CF 算法流程

8 台设备,其中包含 7 个工作节点和 1 个主节点,Spark 版本为 2.0,Hadoop 版本为 2.7. 采用明尼苏达大学 Movie Lens 作为实验数据,如表 1 所示.

表 1 实验数据集

数据集	Movie Lens 数据集	数据个数	节点
A	100 K	10 000	1
B	1 M	1 000 209	4
C	10 M	10 000 054	7

2.2 评估标准

运用召回率 α ,准确率 δ ,召回率 α 与准确率 δ 的调和平均值 (F-Measure) F ,平均绝对误差 η 来评估实验结果. 定义为

$$\alpha = \frac{\sum_{i \in U} |N(i) \cap T(i)|}{\sum_{i \in U} |T(i)|} \tag{13}$$

$$\delta = \frac{\sum_{i \in U} |N(i) \cap T(i)|}{\sum_{i \in U} |N(i)|} \tag{14}$$

$$F = \frac{2\delta\alpha}{\delta + \alpha} \tag{15}$$

其中:集合 $N(i)$ 为推荐给目标用户的 N 个项目; $T(i)$ 为目标用户 u_i 喜欢的项目集合. 平均绝对误差可由下式计算:

$$\eta = \frac{\sum_{i=1}^N |p_i - q_i|}{N} \tag{16}$$

其中: $\{q_1, q_2, \dots, q_N\}$ 为实际评分集合; $\{p_1, p_2, \dots, p_N\}$ 为预测项目评分集合. 平均绝对误差越小,推荐质量越高.

2.3 比较算法及参数设定

选取了传统 User-CF 推荐算法、改进的 Top- N 聚类协同过滤推荐算法 KCF (K -means clustering collaborative filtering recommendation algorithm) 以及所提出的 RLPSO_KM_CF 算法进行对比实验. 实验中,为了能够使模型计算的复杂度降低,RLPSO 算法设定的参数采用文献[13]提出的参数设置. 同时,假定 Top- N 推荐值为 10,算法参数设置如表 2 所示.

2.4 实验结果分析

为了对 RLPSO_KM_CF 算法的推荐效率和推荐准确度进行分析,采取了以下几组实验:1) 对比 RLPSO_KM_CF 算法在不同聚类下的平均绝对误差

表 2 参数设置

参数名	值
粒子群规模	N
初始化最差粒子群	$M = N/4$
学习因子	$c_1 = 1.85, c_2 = 2.0$
反向学习因子	$c_3 = 0.7, c_4 = 0.3$
反向学习迭代次数	$L_{\text{time}} = 20$
反向学习粒子群规模	$n = 0.8N$
局部搜索扰动系数	$d_0 = 0.8$
反向粒子群学习收敛条件	20
Top- N	10
H_0	1.5

值、召回率及 F-Measure 值,进而判断算法所选择的聚类参数值 K ;2) 对比 RLPSO_KM_CF 算法与传统基于用户的协同过滤推荐算法 User-CF、基于聚类的协同过滤推荐算法 KCF,进而判定本文算法的有效性;3) 对比 RLPSO_KM_CF 算法与其他改进推荐算

法的平均绝对误差值,验证 RLPSO_KM_CF 算法的推荐准确度;4) 对比基于 Spark 的 RLPSO_KM_CF 算法与基于 Hadoop 的 User-CF 与 Item-CF 算法,验证算法的推荐效率. 利用 RLPSO_KM 算法分别将用户划分成 1 ~ 8 个簇,同时,运用协同过滤推荐算法于每次迭代的每个簇类中,并且计算平均召回率和平均绝对误差值. 表 3 为 RLPSO_KM_CF 算法在聚类数分别为 1 ~ 8 时得到的推荐结果的召回率. 其中,聚类数为 1 时,RLPSO_KM_CF 算法与传统的 User-CF 算法效果相当.

图 2 所示为 Movie Lens100K 数据集下绘制的平均绝对误差曲线. 可以明显看出,在实验开始阶段随着聚类数增加,RLPSO_KM_CF 算法的平均绝对误差值下降速度最快,当聚类数为 4 时,RLPSO_KM_CF 算法的平均绝对误差值最小,此时推荐结果最好. 随着聚类数增加,平均绝对误差值也逐渐呈现先递减后递增的趋势,说明目标用户邻域集相对较少,推荐精度降低.

表 3 不同聚类数在不同迭代次数下的召回率

聚类数	$t = 5$	$t = 10$	$t = 15$	$t = 20$	$t = 25$	30	35	40	45
User-CF	0.113 911	0.113 911	0.113 911	0.113 911	0.113 911	0.113 911	0.113 911	0.113 911	0.113 911
$P = 1$	0.114 125	0.114 125	0.114 125	0.114 125	0.114 125	0.114 125	0.114 125	0.114 125	0.114 125
$P = 2$	0.114 253	0.114 568	0.115 103	0.115 458	0.115 914	0.116 102	0.116 102	0.116 102	0.116 102
$P = 3$	0.115 106	0.115 364	0.115 587	0.115 986	0.116 513	0.116 324	0.116 324	0.116 324	0.116 324
$P = 4$	0.116 332	0.116 946	0.117 136	0.117 136	0.117 136	0.117 136	0.117 136	0.117 136	0.117 136
$P = 5$	0.115 876	0.116 648	0.116 796	0.116 843	0.116 984	0.116 854	0.116 948	0.116 948	0.117 106
$P = 6$	0.115 536	0.115 874	0.115 968	0.116 205	0.116 532	0.116 314	0.116 103	0.116 002	0.116 006
$P = 7$	0.115 369	0.115 524	0.115 698	0.115 859	0.115 836	0.115 795	0.115 568	0.115 869	0.115 597
$P = 8$	0.114 863	0.114 962	0.115 164	0.115 568	0.115 502	0.115 548	0.115 256	0.115 368	0.115 568

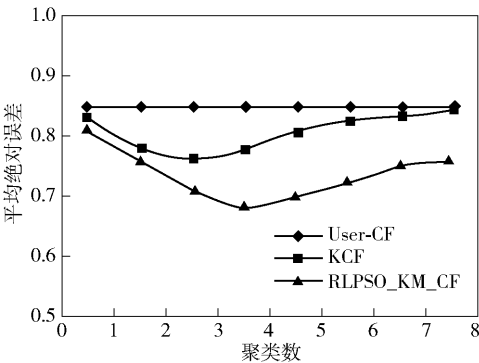


图 2 基于 Movie Lens100K 数据集

图 3 所示为 RLPSO_KM_CF 算法在不同迭代步数下推荐结果的召回率. 当迭代步数在 25 左右时,

各种情况下推荐结果的召回率基本都取得了最大值. 当聚类因子 k 为 4 并且迭代步数为 15 左右时, 算法明显收敛,此时召回率达到了 0.117 136. 相对于传统的协同过滤添加算法,RLPSO_KM_CF 算法提升了 3.2%,同时比 KCF 算法提高了 1.1%. 随着聚类因子的增加,与目标用户进行相似度计算的用户数会随之减少. 因此,算法的推荐准确度下降,推荐结果变差.

从图 4 中的 F-Measure 值来看,假定不考虑随机因素的影响,RLPSO_KM_CF 算法和 KCF 算法优于传统的 User-CF 算法,而且 RLPSO_KM_CF 算法更甚于 KCF 算法. 同时也可以明显看到,随着聚类

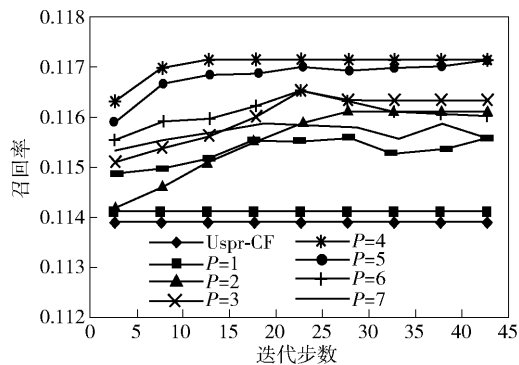


图3 RLPSO_KM_CF 算法不同聚类数下不同迭代步数下的召回率

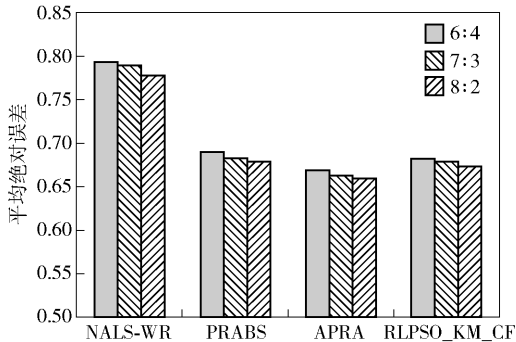


图5 Movie Lens10M 不同训练集与测试集比例下各算法平均绝对误差对比

数的增加,KCF 算法和 RLPSO_KM_CF 算法的 F-Measure 值都逐渐呈现递减趋势.从总体上来说,本文算法提高了推荐的准确率.

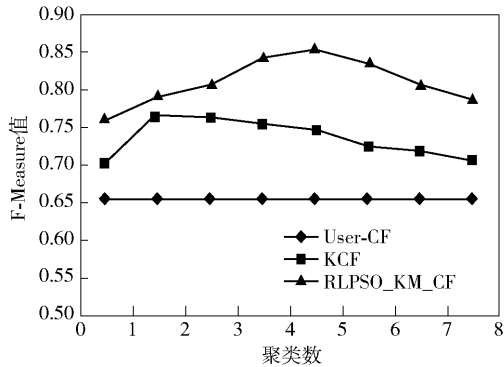


图4 不同聚类数下各算法的 F-measure 值比较

统计了 RLPSO_KM_CF 算法与 NALS-WR、PRABS、APRA 算法在数据集为 Movie Lens10M 时,不同比例训练集与测试集下的平均绝对误差值.由图 5 可知,随着训练集数据不断增加,几种算法的平均绝对误差值都呈现下降的趋势,相比于 NALS-WR 算法,RLPSO_KM_CF 算法在推荐准确度上有显著提高.对于 PRABS 算法,RLPSO_KM_CF 算法在训练集与测试集比例为 8:2 时,平均绝对误差值降低了 1.07%,相比于 ARPA 算法,RLPSO_KM_CF 算法的优势不太明显.

通过采用不同大小的实验数据集进行实验,在 Spark 环境下(4 个节点)并行化执行 RLPSO_KM_CF 算法.从图 6 可以看出,数据集为 100 K 和 1 M 时,并行算法优势并不明显.但是随着数据集规模不断增大,Spark 并行化优势逐渐变得明显,这充分体现了 Spark 基于内存计算模式在时间开销上的巨大优势.同时,对比了本文算法与 Hadoop-ItemCF^[8] 和 Hadoop-UserCF^[9] 算法的

加速比,在数据集为 10 M 时,加速比达到了其他算法的 10 倍.这是由于 Spark 是基于内存的分布式计算框架,在处理初始阶段,通信节点间的通信使得时间消耗增加,而随着数据集的增加,加速比则逐渐提高,这充分体现了 Spark 基于内存的计算的优势.

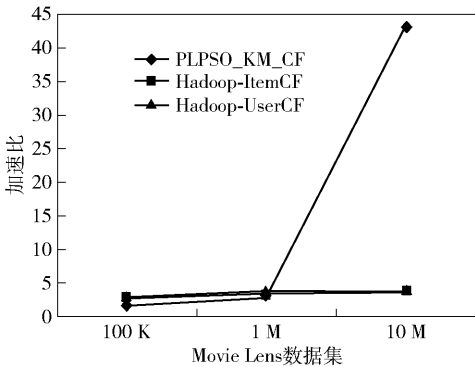


图6 在不同实验数据集下各算法的加速比

3 结束语

传统协同过滤推荐算法用户邻域的选择参考了所有用户的评分信息.然而当为一个目标用户进行推荐时,与其相似性更高的用户显然比其他用户具有更有价值的参考作用.笔者提出在 Spark 上并行化实现了基于粒子群优化算法的协同过滤推荐算法.通过 RLPSO 算法寻找粒子群最优解,输出优化后的聚类中心;运用 RLPSO_KM 算法对用户信息进行聚类,从而将传统协同过滤推荐算法与 RLPSO_KM 聚类结合,对目标用户进行有效推荐.笔者只选取了 KM 聚类算法进行研究,在今后的研究中,面对冷启动问题和评分矩阵稀疏性问题,可以考虑选取一些适合稀疏性矩阵的聚类算法与传统的协同过滤推荐算法相结合进行研究.

参考文献:

- [1] Ricci F, Rokach L, Shapira B. Introduction to recommender systems handbook [C] // Ricci F, Rokach L, Shapira B, et al. Recommender Systems Handbook. Berlin: Springer, 2011: 1-35.
- [2] Xu Bin, Bu Jiajun, Chen Chun, et al. An exploration of improving collaborative recommender systems via user-item subgroups [C] // Proceedings of the 21st International Conference on World Wide Web. New York: ACM, 2012: 21-30.
- [3] Jiang Jing, Lu Jie, Zhang Guangquan, et al. Scaling-up item-based collaborative filtering recommendation algorithm based on Hadoop [C] // IEEE World Congress on Services. New York: IEEE Press, 2011: 490-497.
- [4] Zhao Zhidan, Shang Mingsheng. User-based collaborative-filtering recommendation algorithms on Hadoop [C] // Third International Conference on Knowledge Discovery and Data Mining. New York: IEEE Press, 2010: 478-481.
- [5] 李改, 潘嵘, 李章凤, 等. 基于大数据集的协同过滤算法的并行化研究[J]. 计算机工程与设计, 2012, 33(6): 2437-2441.
- Li Gai, Pan Rong, Li Zhangfeng, et al. Research on parallelization of collaborative filtering algorithms based on big data sets [J]. Computer Engineering and Design, 2012, 33(6): 2437-2441.
- [6] 戚荣志, 王志坚, 黄宜华, 等. 基于 Spark 的并行化组合测试用例集生成方法[J]. 计算机学报, 2017, 40(16): 1-18.
- Qi Rongzhi, Wang Zhijian, Huang Yihua, et al. Spark-based parallel combination test case generation method [J]. Chinese Journal of Computers, 2017, 40(16): 1-18.
- [7] Winlaw M, Hynes M B, Caterini A, et al. Algorithmic acceleration of parallel ALS for collaborative filtering: speeding up distributed big data recommendation in Spark [C] // 21st International Conference on Parallel and Distributed Systems. New York: IEEE Press, 2015: 682-691.
- [8] Kupisz B, Unold O. Collaborative filtering recommendation algorithm based on Hadoop and Spark [C] // IEEE International Conference on Industrial Technology. New York: IEEE Press, 2015: 1510-1514.
- [9] Li Zeng, Liu Yu. Research on personalized recommendation algorithm based on Spark [C] // 2nd International Conference on Advances in Materials, Machinery, Electronics (AMME). MELVILLE: AIP, 2018: 040073.
- [10] Tu Xiaohan, Liu Siping, Li Renfa. Improving matrix factorization recommendations for problems in big data [C] // IEEE 2nd International Conference on Big Data Analysis. New York: IEEE Press, 2017: 198-202.
- [11] Hammou B A, Lahcen A A, Mouline S. APRA: an approximate parallel recommendation algorithm for big data [J]. Knowledge-Based Systems, 2018(157): 10-19.
- [12] Kennedy J, Eberhart R. Particle swarm optimization [C] // Proceedings of the IEEE International Conference on Neural Networks. New York: IEEE Press, 1995(4): 1942-1948.
- [13] 夏学文, 刘经南, 高柯夫, 等. 具备反向学习和局部学习能力的粒子群算法[J]. 计算机学报, 2015, 38(7): 1397-1407.
- Xia Xuewen, Liu Jingnan, Gao Kefu, et al. Particle swarm optimization with inverse learning and local learning capability [J]. Chinese Journal of Computers, 2015, 38(7): 1397-1407.