基于 Spark 框架的改进协同过滤算法

邹红旭,潘冠华,李 吟
(江苏自动化研究所,江苏 连云港 222006)

摘 要:随着互联网数据量的不断膨胀,单机已经无法在可接受的时间范围内计算完基于大规模数据的推荐算法,也无法存放海量的数据。利用 Spark 平台内存计算的优点,设计了一种分布式的基于项目的协同过滤算法,利用 Spark 提供的 RDD(resilient distributed dataset)算子完成算法的设计。针对由于数据稀疏而导致的相似度计算不准确的问题,提出了一种利用两项目间公共用户数目进行加权的相似度计算公式,提高了最终推荐结果的准确度。为了改善计算中涉及到的数据表等值连接操作耗时太长的问题,利用自定义的 Hash_join 函数替代 Spark 自带的连接操作算子,提高了计算效率。采用 UCI 的公用数据集 MovieLens 对算法进行测试,并分别与改进前的算法以及单机运行的算法进行对比,结果表明,改进的算法在准确度和效率方面都有更好的表现。

关键词:协同过滤;Spark;稀疏数;相似度计;等值连接
中图分类号:TP391 **文献标识码**:A **文章编号**:1673-629X(2020)05-0038-05
doi:10.3969/j.issn.1673-629X.2020.05.008

Improved Collaborative Filtering Algorithm Based on Spark

ZOU Hong-xu,PAN Guan-hua,LI Yin
(Jiangsu Automation Research Institute of CSIC,Lianyungang 222006,China)

Abstract:With the explosive growth of data,single-computer computing has been unable to meet the real-time requirements of recommendation algorithms,nor can it store massive data. A distributed item-based collaborative filtering algorithm is designed based on the advantages of memory computing in Spark platform,and the RDD (resilient distributed dataset) provided by Spark is used to complete the design of the algorithm. To solve the problem of inaccurate similarity caused by sparse data,a similarity calculation formula weighted by the number of common users between two items is proposed,which improves the accuracy of the final recommendation results. Equivalent connection of data tables is involved in the calculation. In order to reduce the time consumed by equivalent connection of data tables,the user-defined Hash_join function is used to improve the calculated performance. The performance of the algorithm based on Spark platform is tested by MovieLens dataset. Compared with the original algorithm and the one running on a single computer respectively,it is showed that the improved algorithm has better performance in accuracy and efficiency.

Key words:collaborative filtering;Spark;sparse data;similarity calculation;equivalent connection

0 引 言

在互联网产生的数据量迅速增长的今天,如何准确高效地挖掘数据中蕴藏的信息变得越来越有挑战性[1],推荐系统能有效地解决这一问题。推荐系统,是提供信息推荐的系统,以降低用户寻找有效信息的难度[2],增加用户与网络信息交互的体验感为目的的算法本体。推荐系统始于二十世纪九十年代,历经了二十多年的发展后,其理论也在不断地更新和完善[3],应用也变得越来越广泛。然而,面对当今时代爆炸式增长的数据量,推荐算法的计算效率面临很大的挑战,单机已经无法满足计算需求,因此,如何在多节点的计算机集群上高效地并行化运行推荐算法成为了重中之重[4]。

为了使协同过滤算法更能适应数据稀疏的情况,改进相似度计算公式,设计一种既能适应稀疏数据,又适合进行并行化改造的协同过滤算法,研究其在 Spark 框架下的并行化实现方案,并进行调优和验证,使其准确度和计算效率达到最大化。文献[5]研究了协同过滤算法在 Spark 上的并行化方案,但其在实现过程中采用了将数据集在集群上从单个节点进行广播的方

收稿日期:2019-06-19 **修回日期**:2019-10-21 **网络出版时间**:2020-01-10
基金项目:国家自然科学基金(61773384)
作者简介:邹红旭(1995-),男,硕士研究生,研究方向为大数据、分布式计算、数据挖掘;潘冠华,研究员,研究方向为系统工程、舰船总体方向。
网络出版地址:http://kns.cnki.net/kcms/detail/61.1450.TP.20200110.1121.036.html

法,当数据集非常大时该方法会有明显的瓶颈现象,进行广播的节点会占用大量的计算时间,而且如果数据集大到无法在单一节点内存中存放时,该方法则无法完成计算。因此,提出了避免瓶颈问题的解决方案,采用了完全分布式的计算方法,每个节点始终只需存储和计算一小部分数据,提高了系统的可扩展性。

1　Spark 大数据计算框架

谷歌公司于 2003 年起依次发表了三篇论文:《Google 文件系统(Google file system,GFS)》[6]、《大型结构化数据表(big table)》及《大数据分布式计算模型(MapReduce)》[7-8]。这三篇论文奠定了大数据技术的基础,后来 Apache 基金会根据这三篇论文,设计了 Hadoop 分布式数据处理系统,从而将大数据的研究推向了高潮。作为 MapReduce 计算模型的替代方案,Apache Spark 弥补了 MapReduce 在实现迭代式算法时由于多次进行磁盘 I/O 而降低算法计算效率、无法满足算法实时性要求的缺陷[9]。

Spark 采用了内存计算模型,对于迭代式算法,数据和计算中间结果都存储在内存中,只需开始时从磁盘加载一次数据,之后都直接从内存中加载数据,省去了大量的磁盘 I/O 时间。Spark 的系统架构如图 1 所示,Spark 运行程序时首先由 Driver 向集群管理器申请资源,包括 CPU 和内存资源,获得集群管理器分配的资源之后,Driver 将应用程序以 task 的形式发放到各个工作节点,工作节点并行执行各个 task。

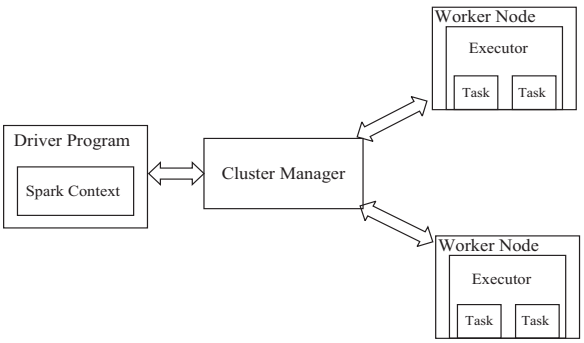

图 1　Spark 的整体架构

2　改进的 Item-Based 协同过滤算法

2.1　Item-Based 推荐算法原理

Item-Based 推荐算法[10-13]的输入为用户-项目评分数据集,该数据集由数量为 m 的用户对数量为 n 的项目的评分构成,用户集合用 $U=\{u_1,u_2,\cdots,u_m\}$ 表示,项目集合用 $I=\{i_1,i_2,\cdots,i_n\}$ 表示。评分数据集是不完善的,每个用户只评论了部分项目,推荐算法的目的就是完善该数据集,预测出所有用户对所有项目的评分。计算过程如下:

(1)计算项目间相似度,得到相似度矩阵 $\mathrm{Sim}_{n\times n}$。这里采用修正的余弦相似度计算方法,公式如下:

$$\mathrm{Sim}(i,j)=\frac{\sum_{u\in U_{ij}}(R_{ui}-\overline{R_u})\times(R_{uj}-\overline{R_u})}{\sqrt{\sum_{u\in U_{ij}}(R_{ui}-\overline{R_u})^2}\sqrt{\sum_{u\in U_{ij}}(R_{uj}-\overline{R_u})^2}} \tag{1}$$

其中,$\mathrm{Sim}(i,j)$ 表示项目 i 和 j 之间的相似度,i、$j\in I$,R_{ui} 表示用户 u 对项目 i 的评分,$\overline{R_u}$ 表示用户 u 对其所有已评分项目的评分均值。令集合 U_i 表示所有对项目 i 进行过评分的用户组成的集合,则 $U_{ij}=U_i\cap U_j$,表示由 i、j 两个项目之间的公共用户,即同时对项目 i、j 进行过评分的用户组成的集合。

(2)根据相似度进行评分预测。使用 Park 采用的预测方法,计算公式如下:

$$P_{u,i}=\overline{R_i}+\frac{\sum_{j\in \mathrm{KNN}_i}\mathrm{Sim}(i,j)(R_{uj}-\overline{R_j})}{\sum_{j\in \mathrm{KNN}_i}|\mathrm{Sim}(i,j)|} \tag{2}$$

其中,$P_{u,i}$ 表示预测的用户 u 对项目 i 的评分,$\overline{R_i}$ 表示项目 i 的已评分的均值,KNN_i 表示项目 i 的最近邻居集,即由前 k 个与项目 i 的相似度最大的其他项目组成的集合,在对 MovieLens 数据集进行测试时,取 $k=500$。

2.2　算法改进思路

如式(1)所示,相似度的计算要用到两个项目之间的公共用户,即集合 U_{ij} 里的用户 u 分别对项目 i、j 的评分,但当数据稀疏时[14-16],两个项目之间的公共用户的数目可能会很小,导致相似度的计算会出现很大的偏差。极端情况下,当 U_{ij} 中只有一个元素时,根据式(1)计算得到的相似度为 1,表示两个项目间的相似度为 100%,无论这两个项目在实际中是否相似,都将得到这个结果。而在后续的计算中,这个值为 1 的相似度由于是最大值,又必然会被选入 KNN_i 中,于是会对计算结果 $P_{u,i}$ 的准确度产生很大的干扰。

针对上述问题,提出改进的相似度计算公式,在原公式(1)的基础上乘以一个权重函数 $f(x)$,x 表示集合 U_{ij} 中元素的数目,即两项目间的公共用户数目,目的是对当 x 很小时计算得出的相似度加一个很小的权重,使其尽量不被选入 KNN_i 中,或者即使被选入 KNN_i 中,也无法对 $P_{u,i}$ 的计算产生很大的影响。另一方面,当两个项目间公共用户数目很少时,也能从一定程度上说明这两个项目间的相似度很小,所以乘以权重函数 $f(x)$ 之后得到的相似度将更加接近这两个项目间的真实相似度。$f(x)$ 的选取要满足以下两个条件:

(1) $f(x)$ 是关于 x 的增函数;

(2)当 x 足够大时 $f(x)$ 随 x 的增长要趋于收敛,因为此时根据式(1)计算得到的相似度已接近真实值,$f(x)$ 的值必须趋于稳定,才能保证加权之后对计算结果 $P_{u,i}$ 产生的干扰达到最小。

考虑以上两点,选取 $f(x)=\ln(x)$,改进后的相似度计算公式如下:

$$\mathrm{Sim}(i,j)=\frac{\sum_{u\in U_{ij}}(R_{ui}-\overline{R_u})\times(R_{uj}-\overline{R_u})}{\sqrt{\sum_{u\in U_{ij}}(R_{ui}-\overline{R_u})^2}\sqrt{\sum_{u\in U_{ij}}(R_{uj}-\overline{R_u})^2}}\times\frac{}{\ln(x)} \tag{3}$$

其中,x 表示集合 U_{ij} 中元素的数目。

3 改进的协同过滤算法在 Spark 上的并行化实现

3.1 计算资源的申请

采用的分布式环境为 Spark+Yarn 模式,Spark 作为分布式计算框架,Yarn 作为集群管理器,根据集群配置,Spark 向 Yarn 申请 10 个 executor 作为计算资源,每个 executor 包括一个 CPU 与 500 M 内存。在 Spark 中,数据被切分为若干个分区,不同的分区位于不同的 executor 中,每个 executor 只需对自己分区中的数据进行计算,从而达到分布式计算的目的。分区数目的选择是影响 Spark 计算效率的一个关键因素,分区数目太多会导致数据的混洗(shuffle)过程消耗更长的时间,而分区数目太少则会使 join 操作更加耗费时间并且降低系统容错性。在本次实验中分区数目选取为 50 个。计划好资源的申请之后就可以用如下语句启动 spark:

spark-submit --master yarn --executor-memory 500 M --executor-cores 1 --num-executors 10

3.2 项目间相似度的计算

Spark 主要通过把数据集抽象成 RDD(resilient distributed dataset,弹性分布式数据集)对象来操作数据,通过对 RDD 的操作来间接操作数据。相似度的计算可以分为以下四个步骤:

(1)从 HDFS(Hadoop distributed file system,Hadoop 分布式文件系统)中读取包含训练数据的文件。需要用到数据文件中 userId、itemId、rate 这三个字段,分别表示用户编号、项目编号、用户对项目的评分。将数据利用 textFile()方法读入内存,并转化为格式为(userId,(itemId,rate))的元组组成的 $\mathrm{RDD}_{\mathrm{data}}$。

(2)依次利用 map、reduceByKey、mapValues 算子对 $\mathrm{RDD}_{\mathrm{data}}$ 操作得到 $\mathrm{RDD}_{\mathrm{avgrate}}$,该 RDD 代表用户评分均值,其元素的格式为(userId,averageRate)。因为后面要多次用到该 RDD,为了避免 Spark 重复计算该 RDD,利用 persist 操作将其持久化到内存中。

(3)用 join 算子对 $\mathrm{RDD}_{\mathrm{data}}$ 和 $\mathrm{RDD}_{\mathrm{avgrate}}$ 进行等值连接,将得到的结果再次与 $\mathrm{RDD}_{\mathrm{data}}$ 进行等值连接,得到 $\mathrm{RDD}_{\mathrm{join}}$,其元素的格式为(userId,(itemId1,rate1,itemId2,rate2,averageRate)),itemId1、itemId2 表示该用户共同评价过的两个项目,rate1、rate2 表示评分。

(4)对 $\mathrm{RDD}_{\mathrm{join}}$ 进行 map 操作,将其元素的格式转换为(itemId1_itemId2,rate1,rate2,averageRate),然后对其进行 combineByKey 操作得到 $\mathrm{RDD}_{\mathrm{similarity}}$,其元素的格式为(itemId1_itemId2,similarity),similarity 表示 itemId1 和 itemId2 之间的相似度。

相似度计算过程中用到的 RDD 的谱系图如图 2 所示。

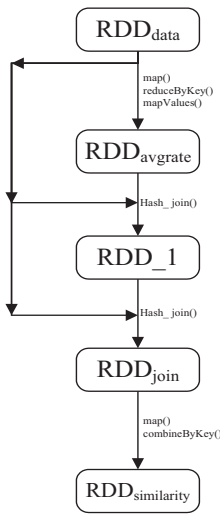

图 2 相似度计算过程中的 RDD 谱系图

3.3 评分预测

评分的预测可以分为以下几个步骤:

(1)对 $\mathrm{RDD}_{\mathrm{data}}$ 进行 combineByKey 操作,得到 $\mathrm{RDD}_{\mathrm{itemList}}$,其元素的格式为(userId,ArrayBuffer[(itemId,rate)]),其中 ArrayBuffer 表示 scala 语言中的变长数组,ArrayBuffer[(itemId,rate)]表示该用户评价过的所有项目和对应评分组成的数组。

(2)从 HDFS 中读取测试数据集 $\mathrm{RDD}_{\mathrm{test}}$,并与 $\mathrm{RDD}_{\mathrm{itemList}}$ 进行 join 操作,再对结果进行 map 操作得到 $\mathrm{RDD}_{\mathrm{testList}}$,其元素的格式为(itemId1,(userId,ArrayBuffer[(itemId2,rate)])),其中 itemId1、userId 分别表示要进行预测评分的用户和项目,需要注意的是 itemId1 来自测试集而 itemId2 来自训练集。最后再对 $\mathrm{RDD}_{\mathrm{testList}}$ 进行 combineByKey 聚合操作,使其键保持不变,值聚合到一个 ArrayBuffer 中,方便后续计算。

(3)依次利用 map、reduceByKey、mapValues 算子对 $\mathrm{RDD}_{\mathrm{data}}$ 操作得到 $\mathrm{RDD}_{\mathrm{itemavg}}$,代表项目评分均值,其元素的格式为(itemId,itemAverageRate)。

(4)对 $\mathrm{RDD}_{\mathrm{similarity}}$ 和 $\mathrm{RDD}_{\mathrm{itemavg}}$ 进行 join 操作,再对

结果进行 flatMap、combineByKey 操作得到 RDD_{knn},代表项目的最近邻居集,其元素的格式为(itemId1, ArrayBuffer[(itemId2,similarity,itemAverageRate)])。

(5)对 $RDD_{itemavg}$、RDD_{knn}、$RDD_{testList}$ 进行 join 操作,再对结果进行 mapValues、flatMap 操作得到最终结果 RDD_{result},其元素格式为(itemId,userId,result),result 表示最终预测的评分。

评分预测过程中用到的 RDD 的谱系图如图 3 所示。

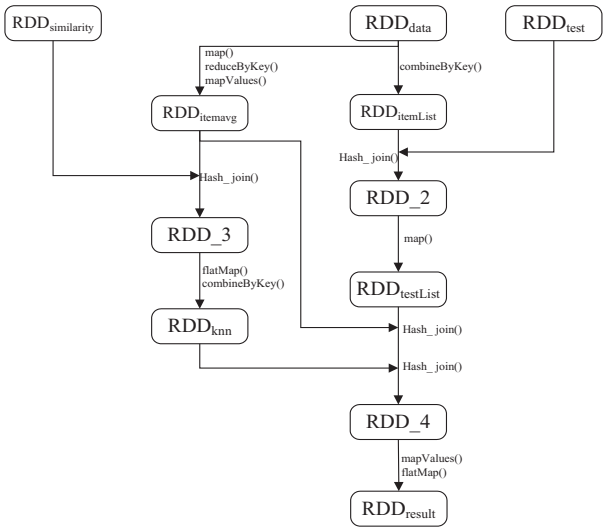

图 3　评分预测过程中的 RDD 谱系图

3.4　等值连接操作的优化

根据上述步骤可知,计算过程中涉及到很多的 join 操作,而 join 操作会消耗很多的计算资源[17-18],这主要是因为 join 操作会进行多次的迭代来匹配相同键值的元素,因此有必要对其进行优化。假设要对 RDD1 和 RDD2 进行连接操作,其元素数目分别为 x_1 和 x_2,如果采用 join 算子会进行共 $x_1 * x_2$ 次迭代。设计自定义的 Hash_Join 函数替代 join 操作,Hash_Join 函数原理如下:

假设 RDD1 的元素格式为(Key1,Value1),RDD2 的元素格式为(Key2,Value2),首先对两个 RDD 进行 map 操作,利用自定义的 Hash 函数对两个 RDD 的 Key 进行转换,使其映射到集合 Buckets,原有的 Key 放入 Value 中。两个 RDD 中相同的 Key 会被映射到相同的 Bucket 中。Key 集合与 Buckets 集合是多对一的关系,目的是为了将多个 Key 对应的内容聚集到一个 Bucket 中。然后再对这两个 RDD 进行 join 操作,把两个 RDD 中相同 Bucket 的内容匹配到一起,最后再把每个 Bucket 中相同 Key 的内容匹配到一起。Hash_Join 函数原理如图 4 所示。

假设 Key 的数量是 Bucket 数量的 a 倍,则每个 Bucket 含有 a 条元素,优化后的迭代次数如下:

$$y = \frac{x_1}{a} \times \frac{x_2}{a} + \mathrm{MIN}\left(\frac{x_1}{a}, \frac{x_2}{a}\right) \times a^2 \tag{4}$$

在对 MovieLens 数据集进行测试时,进行连接操作的数据集规模大小为百万级,此时取 a 的大小为 200,可以看出,优化后的迭代次数明显小于 $x_1 * x_2$。

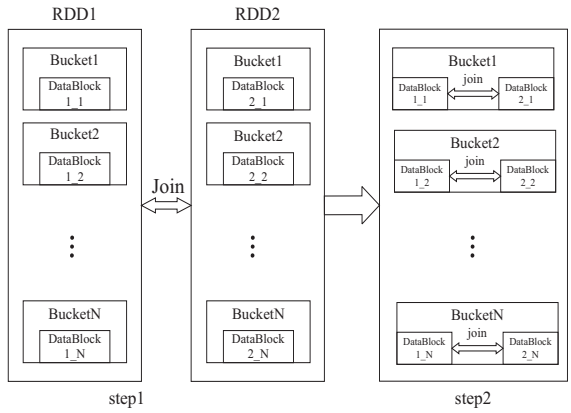

图 4　Hash_Join 原理

4　实验结果分析

实验使用 UCI 的公用数据集 MovieLens 对算法进行测试,该数据集有三种不同大小的数据,其数据规模分别为 100 k、1 M 和 10 M,实验中采用规模为 1 M 的数据集,其中包含了 6 040 位用户对 3 900 部电影的评分,评分记录一共有 1 000 209 条。实验用到的 Spark 集群部署在 5 台服务器上,使用的集群管理器为 Yarn 管理器,通过向 Yarn 申请 executor 来获取计算资源,实验中申请的 executor 数目为 10 个,为每个 executor 分配一个 CPU,CPU 型号为 Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40 GHz,每个 executor 分配的内存为 500 M。

(1)算法准确度测试。

为了衡量预测评分的准确性,采用平均绝对误差(MAE)来计算预测结果的误差。先后把数据集分成不同比例的训练集和测试集,分别用改进前的算法与改进后的算法对测试集进行预测,并分别对预测结果进行 MAE 的计算,结果如表 1 所示。

表 1　不同的训练集和测试集比例下两种算法的 MAE 值

指标	9∶1	8∶2	7∶3	6∶4
改进前 MAE	0.700 6	0.708 9	0.719 7	0.731 3
改进后 MAE	0.679 8	0.683 3	0.688 0	0.693 1
MAE 差值	0.020 8	0.025 6	0.031 7	0.038 2

从结果中可以看出,改进后的算法预测结果的 MAE 要小于改进前的算法,说明改进后的算法提高了预测结果的准确度,并且可以看到,随着训练数据集的比例逐渐减小,即训练数据越发稀疏时,两种算法的

MAE 的差值逐渐增大,说明改进后的算法对预测准确度的提升愈加明显,从而说明了改进后的算法更加适合数据稀疏的情况。

(2)算法执行时间测试。

在 Spark 集群上分别运行未优化的算法和优化等值连接操作后的算法,并与单机运行算法的时间进行对比,取训练集与测试集比例为 9 ∶ 1,数据集规模逐渐增加,实验结果如表 2 所示。

表 2 运行时间对比

数据规模/条	单机运行时间/s	未优化集群算法/s	优化集群算法/s
10 000	37	18	20
100 000	175	31	36
200 000	253	46	49
400 000	375	79	72
600 000	572	117	96
800 000	813	202	132
1 000 000	1 134	292	168

从结果中可以看出,当数据规模较小时,集群算法的运行时间并没有比单机算法短很多,加速比并不是很高,这主要是因为 Spark 启动作业、分配任务等系统操作以及集群上各节点之间的通信、互相传递数据等操作占用了额外的运行时间,但随着数据规模的扩大,加速比逐渐提高并趋于稳定。

经过等值连接优化的算法在数据规模较小时表现并不突出,因为在用自定义的 Hash 函数把两个 RDD 的 Key 转换为 Bucket 时要付出额外的计算成本,但随着数据规模的扩大,其性能表现逐渐超过了未优化的算法,并且数据规模越大,其相比未优化的算法节省的计算时间越多。

5 结束语

设计了一种在 Spark 平台上运行的改进的 Item-Based 协同过滤算法,使其更适合数据稀疏的情况,并对算法中涉及的等值连接操作进行了优化,提高算法效率。用 MovieLens 数据集对算法进行测试后得到的结果表明,算法在准确度和效率方面都有更好的表现,说明了算法既能更准确地得到预测结果,又能适应分布式计算平台,更快地得到计算结果。

参考文献:

[1] 王源龙,孙卫真,向 勇. 基于 Spark 的混合协同过滤算法改进与实现[J]. 计算机应用研究,2019,36(3):855-860.

[2] RICCI F,ROKACH L,SHAPIRA B,et al. Recommender systems handbook[M]. Boston,MA:Springer,2011.

[3] 杨志伟. 基于 Spark 平台推荐系统研究[D]. 合肥:中国科学技术大学,2015.

[4] 廖 彬,张 陶,国冰磊,等. 基于 Spark 的 ItemBased 推荐算法性能优化[J]. 计算机应用,2017,37(7):1900-1905.

[5] 陆俊尧,李玲娟. 基于 Spark 的协同过滤算法并行化研究[J]. 计算机技术与发展,2019,29(1):85-89.

[6] GHEMAWAT S,GOBIOFF H,LEUNG S T. The Google file system[C]//Proceedings of the 19th ACM symposium on operating system principles. New York:ACM,2003:29-43.

[7] DEAN J, GHEMAWAT S. MapReduce:simplified data processing on large clusters[C]//OSDI 2004:proceedings of the 2004 conference on operating system design and implementation. New York:ACM,2004:137-150.

[8] 廖 彬,张 陶,于 炯,等. MapReduce 能耗建模及优化分析[J]. 计算机研究与发展,2016,53(9):2107-2131.

[9] KARAU H. Spark 快速大数据分析[M]. 北京:人民邮电出版社,2015.

[10] 邓爱林,左子叶,朱扬勇. 基于项目聚类的协同过滤推荐算法[J]. 小型微型计算机系统,2004,25(9):1665-1670.

[11] KIM B M,LI Q,PARK C S,et al. A new approach for combining content-based and collaborative filters[J]. Journal of Intelligent Information System,2006,27:79-91.

[12] 邓爱林,朱扬勇,施伯乐. 基于项目评分预测的协同过滤推荐算法[J]. 软件学报,2003,14(9):1621-1628.

[13] TIAN X H. PCIB:a new algorithm for item-based collaborative filtering recommendations[C]//Proceedings of 2014 international conference on artificial intelligence and industrial application. Hong Kong,China:Advanced Science and Industry Research Center,2014:9.

[14] SU Xiaoyuan,KHOSHGOFTAAR T M. Collaborative filtering for multi-class data using belief nets algorithms[C]//18th IEEE international conference on tools with artificial intelligence. Arlington,VA,USA:IEEE,2006:497-504.

[15] 黄 迪. 基于协同过滤算法数据稀疏性问题的研究[D]. 绵阳:西南科技大学,2018.

[16] 许智宏,蒋新宇,董永峰,等. 一种基于 Spark 的改进协同过滤算法研究[J]. 计算机应用与软件,2017,34(5):247-254.

[17] 张子栋,郑延斌. 基于 Spark 的两表等值连接过程优化[J]. 计算机应用研究,2019,36(2):486-489.

[18] ARMBRUST M,XIN R S,LIAN C,et al. Spark SQL:relational data processing in Spark[C]//Proceedings of ACM SIGMOD international conference on management of data. New York:ACM,2015:1383-1394.