基于 Spark 平台的并行 KNN 异常检测算法

冯贵兰[1] 周文刚[2]
(中国民航飞行学院现代教育技术中心 四川 广汉 618307)[1]
(中国民航飞行学院飞行技术学院 四川 广汉 618307)[2]

摘 要 随着大数据时代的到来,异常检测受到了广泛关注。针对传统 KNN 异常检测算法处理速度和计算资源的瓶颈,以及 Hadoop 平台上的 MapReduce 不能友好支持迭代计算和基于内存计算等问题,提出了一种基于 Spark 平台的并行 KNN 异常检测算法。该算法首先对数据集进行分区和广播,然后用 map 函数计算数据集在每个分区的 K 近邻,使用 reduce 函数归并 map 函数的输出计算全局 K 近邻得到异常度,将异常度前 n 个对象视为异常。与传统 KNN 异常检测算法相比,在保证检测精度的前提下该算法的性能与计算资源呈近似线性关系;与其他并行异常检测算法相比,该算法无需额外扩展数据,支持迭代,而且通过在内存中缓存中间结果来减少 I/O 花销。实验结果证明,该算法可以提高 KNN 算法在大规模数据上的异常检测效率。

关键词 Spark 平台,并行,K 近邻,异常检测
中图法分类号 TP311 **文献标识码** A

Spark-based Parallel Outlier Detection Algorithm of K-nearest Neighbor

FENG Gui-lan[1] ZHOU Wen-gang[2]
(Modern Education Technology Center, Civil Aviation Flight University of China, Guanghan, Sichuan 618307, China)[1]
(Institute of Flight Technology, Civil Aviation Flight University of China, Guanghan, Sichuan 618307, China)[2]

Abstract With the advent of big data era, outlier detection has attracted extensive attention. Computational resources of the traditional K-nearest neighbor outlier detection dealing with massive high dimensional data with single machine are insufficient, and the MapReduce in Hadoop cannot effectively deal with frequent iteration calculation problem. According to the above problems, this paper put forward a Spark-based parallel outlier detection algorithm of K-nearest neighbor, named SPKNN. Firstly, in the stage of map, the algorithm tries to find the local K nearest neighbors for each partition of the data in all data set. Then in the reduce stage, it determines the global K nearest neighbors according to the local K nearest neighbors of each partition. Finally, it calculates the degrees of outliers by using global K nearest neighbors and select outliers. Compared with the traditional K-nearest neighbor outlier detection, the performance of the SPKNN has an approximate linear relationship with computing resources in the premise of ensuring the detection accuracy. And compared with other outlier detection methods, it doesn't need additional extension data, support iteration calculation and can reduce I/O costs by using memory cache. Experiment results of SPKNN show that it has high efficiency and scalability for massive data sets.

Keywords Spark, Parallel, K-nearest neighbors, Outlier detection

1 引言

随着信息技术的发展,大量的数据不断涌现[1],怎样从海量的数据中挖掘出价值含量高的信息是亟待解决的问题。异常检测作为数据挖掘的重要技术之一,用于识别不规律或异常数据[2],这些异常数据常常隐藏着重要信息。例如信用卡交易数据中,异常数据意味着潜在的欺诈[3]。在海量化、复杂化、属性多的飞行数据中,异常数据意味着故障预警。因此,研究大数据集的异常检测具有重要的理论价值和实际意义。

目前,研究人员已提出一系列异常检测算法,主要包括基于统计、偏差、聚类、距离和密度等方法。基于统计和偏差的方法比较适合低维数据,对四维以上的数据集效率较低。基于聚类的方法将数据较少的类或不能被聚类的数据视为异常,异常点不能很好体现。基于距离的方法[4]易于理解和实现,已被广泛应用。基于密度的方法[5]是对基于距离的方法的改进,一般使用每个对象到第 K 个最近邻的距离大小来度量密度,其缺点是计算量大、参数鲁棒性差[6]。针对大数据规模大、维度高的特性,基于距离的异常检测算法比较合适。但传统基于距离的异常检测算法执行效率不佳,其串行计算方法的处理能力有限,已经不能满足数据的爆发式增长带来的计算需求[7]。目前的并行异常检测算法[8-10]利用 MapReduce 模型实现了算法并行,提高了效率,但大多数的并行异常检测算法不能有效处理频繁迭代的计算和基于内存的计算。针对上述问题,本文提出了一种基于 Spark 平台的并行 KNN 异

本文受民航飞行数据分析研究项目(XM2852)资助。
冯贵兰(1988-),女,硕士生,工程师,主要研究领域为云计算、信息安全,E-mail:fengguilan1016@sina.com;周文刚(1981-),男,博士生,讲师,主要研究领域为网络管理、机器学习、人工智能等。

常检测算法 SPKNN(Spark-based Parallel outlier detection algorithm of K Nearest Neighbor)。SPKNN 算法首先对待检测数据集进行复用,一方面把数据集进行分区,便于并行计算,另一方面利用 Spark 的广播机制将数据集广播到各个分区;然后使用 map 函数计算每个数据对象在各个分区的局部 K 近邻,reduce 函数归并得到全局 K 近邻;最后计算异常度,输出异常度排名前 n 的数据对象。实验从准确率和参数敏感性等两个角度验证了 SPKNN 算法的可行性和有效性,并在集群环境下评估了算法的加速比和扩展性。实验表明,SPKNN 算法在集群环境中具有良好的性能,更适合海量数据的异常检测。

2 相关工作

异常检测是指搜索不符合预期的模式项目或事件,适用于故障检测、网络入侵检测等领域。在国内外学者提出的异常检测方法中,基于距离的异常检测因为其简单直观、易于实现的优点比较适合大数据集环境,该算法的核心思想是 K 近邻距离,最先由 Knorr 等[11]提出,具体是指若一数据集中,至少有 p 个对象与对象 o 的距离大于 d,则将对象 o 定义为 $DB(p,d)$异常,其缺点是:1)没有提供异常的排名;2)对参数 p 和 d 非常敏感,用户使用时需确定合适的参数。此后,Ramaswamy 等[12]对该方法进行了改进,计算所有对象的 K 近邻距离并从大到小进行排序,将距离最大的前 n 个对象视为异常。基于距离的异常检测方法在数据量小且数据维度低时,能发挥较好的作用,但随着数据量的增大,数据维度增高,算法性能下降明显。

随着云计算的发展,为提高检测效率,Subramanyam 等[8]提出了基于 MapReduce 的并行异常检测算法,通过迭代计算来更新全局候选异常点,从而找出异常点,但它却需要不断迭代,多次启动 MapReduce 过程,时间开销大。郭一鹏等[9]提出了面向混合数据的孤立点检测算法,该算法给出混合数据对象之间的相异性度量,并基于最近邻定义了对象的孤立度,实现了混合数据孤立点检测算法的并行化。该算法提高了混合型大数据的孤立点检测效率,但近邻的计算对分区情况具有很大的依赖性。苟杰等[10]提出了一种部署在 Hadoop 上的并行离群点检测算法,该算法首先采用子集扩展的方法对数据集进行划分,在子集上寻找 K 近邻并计算离群度,最后得到离群度最大的一些数据作为离群点。该算法虽然只需启动一次 MapReduce,但却要额外扩展数据才能保留数据的近邻信息,破坏了数据本身的完整性。文献[8-10]均使用了 MapReduce 模型来对数据集分区,在子集中计算数据对象的近邻,它们具有如下缺点:1)只在分区子集里计算近邻,未在数据全集中计算近邻;2)处于分区边界的数据对象会面临近邻丢失的情况,分区前后边界数据点的近邻会发生变化。这两个缺点将会影响到异常点检测的准确率。

3 SPKNN 算法的设计与实现

3.1 技术背景

3.1.1 相关概念

定义 1(距离函数) 对于 d 维空间中的数据对象 $p=\{p_1,p_2,\cdots,p_d\}$和 $q=\{q_1,q_2,\cdots,q_d\}$,p 与 q 之间的欧氏距离为:

$$dist(p,q)=\sqrt{\sum_{i=1}^{n}(p_i-q_i)^2} \tag{1}$$

定义 2(K 近邻) p 表示数据集 D 中的任意数据对象,它的 K 个最近邻记作 $KNN(p)$。计算 p 到数据集 D 中所有其他数据对象间的距离,选取距离对象 p 最近的 K 个数据对象作为 p 的 K 个最近邻。

定义 3(异常度) p 表示数据集 D 中的任意数据对象,P 的 K 个最近邻表示为 $KNN(p)$,设 $q_i\in KNN(p)$,则数据对象 P 的异常度定义为 p 与其 K 个最近邻对象的平均距离:

$$D^a(p)=\frac{1}{k}\sum_{i=1}^{k}dist(p,q_i) \tag{2}$$

通过式(2)可以衡量数据集中所有元素的异常程度。$D^a(p)$越大,表示 p 越远离 k-邻域内的近邻,成为异常点的可能性越大。

定义 4(异常点) 计算数据集 D 中每个数据对象的异常度 D^a,将其按从大到小的顺序降序排列,异常度最高的前 n 个点就是所求的异常点,即 Top-n 异常点。

3.1.2 Spark 云平台

Spark[13]是在 Hadoop MapReduce 基础上提出的新一代大数据分析框架,拥有 Hadoop 的所有优点。但与 Hadoop 不同的是,Spark 引入了弹性分布式数据集 RDD 的概念,它可以将任务计算结果保存到内存中,省去了大量的磁盘 I/O 操作,因此更适合数据挖掘和机器学习等需要多次迭代的算法。

Spark 运行架构如图 1 所示,SparkContext 是系统调度的总入口,Driver 程序创建了它,并对系统进行了初始化。Cluster Manager 负责分配任务运行过程中所需的资源。Executor 是 Worker 节点上的一个进程,负责具体 Task 任务的运行,并将最终结果保存到内存和磁盘上。

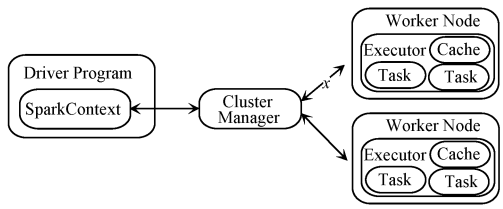

图 1 Spark 运行架构

3.2 SPKNN 算法设计

SPKNN 算法分为预处理阶段、K 近邻计算阶段和异常检测阶段等 3 个阶段。首先进行数据集的分区和广播,然后计算所有数据对象的 K 近邻,找出异常度最大的数据对象。算法流程如图 2 所示。

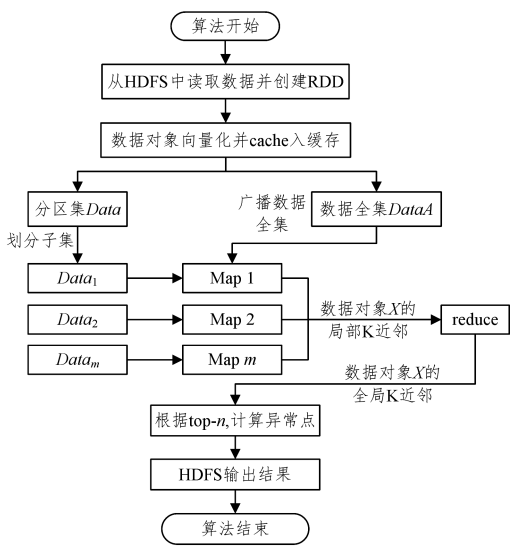

图 2 SPKNN 算法流程图

SPKNN 算法实质上也是采用 MapReduce 思想，与其他并行化设计算法[8-10]相比，其优点在于：

1）调用 RDD 的 cache 操作提高了算法的执行效率。数据对象的 K 近邻计算过程需要进行多次迭代，在 Spark 中可以调用 cache 函数把数据集生成 RDD 持久化在内存中。在任何时候，包括内存不足的时候，Spark 也不会对这个已经 cache 的 RDD 进行操作，并且其中的数据格式一直保持不变，因而迭代过程中数据集可以一直重复使用，这样就大大减少了磁盘 I/O 消耗，而在 Hadoop 平台下算法的每次迭代都需要从分布式文件系统读取数据，这使得磁盘 I/O 消耗很大，影响了算法的整体执行效率。

2）利用 Spark 提供的广播变量实现集群节点之间的数据共享。用 SPKNN 算法中待检测数据集来进行异常检测，需要将数据全集分发到所有计算节点上，利用 Spark 的广播变量能轻松完成，但在 Hadoop 中必须将数据集写到分布式文件系统上，然后由各个计算节点从分布式文件系统中读取才能实现数据共享，由于要读写文件系统，会产生磁盘 I/O 消耗，影响算法效率。

3）对待检测数据集进行复用，检测出每个数据对象在数据全集中的 K 近邻。SPKNN 算法既利用了 MapReduce 模型对数据集进行分区并行处理，又利用了 Spark 的广播机制共享数据全集，通过计算数据全集在每个分区上的局部 K 近邻，规约汇总得到全局 K 近邻，就是数据对象在数据全集中的全局 K 近邻。

3.2.1　预处理阶段

数据预处理的过程是将较大规模的数据集分割成多个子集。首先读取 HDFS 的文件并创建 RDD，并在本地执行 cache 操作缓存 RDD 数据，将待检测数据集分为两部分，一部分用于分区（称为分区集 *Data*），根据用户输入划分成 m 个子集 $Data_1$，$Data_2$，…，$Data_m$，另一部分用来广播（称为数据全集 *DataA*）到各个计算节点上。

3.2.2　K 近邻计算阶段

该阶段运用 MapReduce 模型完成每个数据对象的 K 近邻计算。

map 函数设计：假设数据集 *Data* 是从 HDFS 读取 RDD 对象，且用户通过 textFile 接口的第二个参数定义分区个数 m，将数据集分为 m 个分区 $Data_i$（$1 \leqslant i \leqslant m$），用 *mapPartition* 在每个分区初始化一个 KNN 搜索器，随后广播数据全集 *DataA*，每个分区同时计算数据全集 *DataA* 中每个数据对象 X 的局部 K 近邻。每个分区的结果以键值对〈*key*，*value*〉形式存在，*key* 表示每个数据对象 X 的编号，*value* 是以键值对（*idk*，*dist*）为元素的近邻数组，*idk* 和 *dist* 分别为编号为 *key* 的数据对象 X 对应的近邻编号和距离。算法 1 描述了 map 阶段的主要过程。

算法 1　knnMap 函数

输入：数据分区 $Data_i$，数据全集 DataA，近邻个数 K
输出：数据对象 X 的局部 K 近邻 localKnn
1. for x=0 to size(DataA) do
2.　$neigh_{x,i}$=KNN($Data_i$，DataA[x]，K)
3.　localKnn[x]=(〈key：x，value：$neigh_{x,i}$〉)
4. end for

reduce 函数设计：reduce 函数的任务是处理 map 函数输出的中间结果，合并具有相同属性编号 *key* 的数据。在 map 阶段结束后，对于数据对象 X 有 $m * K$ 个近邻，$neighs = \{neigh_{x,1}, neigh_{x,2}, \cdots, neigh_{x,m}\}$，其中 $neigh_{x,i} = \{(id_1, dist), (id_2, dist), \cdots, (id_k, dist)\}$，并且 *dist* 按照升序排列。主要操作是传入一个比较距离的函数作 reduceByKey 的参数，该函数比较两个近邻数组 $neigh_{x,i}$ 中距离的大小，选距离较小的数据对象加入最终的 K 近邻集合中。算法 2 描述了 reduce 阶段的主要过程。

算法 2　knnReduce 函数

输入：$neigh_1[(id_k, dist)]$，$neigh_2[(id_k, dist)]$
输出：全局 K 近邻集合 globalNeigh
1. globalNeigh=newArray[type(id)][K]；
2. $i_1=0$；$i_2=0$；
3. for j=0 to k do
4. if ($neigh_1[i_1]$.dist<$neigh_2[i_2]$.dist){
5.　globalNeigh[j].id=$neigh_1[i_1]$.id；
6.　globalNeigh[j].dist=$neigh_1[i_1]$.dist；
7.　i_1+=1；}
8. else{
9.　globalNeigh[j].id=$neigh_2[i_2]$.id；
10.　globalNeigh[j].dist=$neigh_2[i_2]$.dist；
11.　i_2+=1；}
12. end for

3.2.3　异常检测阶段

在异常检测阶段，根据式（2）求出数据对象 X 的异常度，并依据异常度对全部数据对象降序排序，输出异常度最大前 n 个数据对象。

3.3　SPKNN 算法的实现

SPKNN 算法实现的整个过程都是基于 Spark 编程模型的，其中利用到 Spark 中的 *mapPartition*，*reduceKey* 等函数，其函数伪代码如算法 3 所示。

算法 3　SPKNN 算法

输入：检测数据集、近邻个数 K、分区数 m，异常点个数 n
输出：检测到的 n 个异常点
1. val targetRDD=sc.textFile(targetSetPath，m)
2. val Data=targetRDD.map(normalize).cache()
3. val DataA=sc.broadcast(Data)
4. val localKnn=Data.mapPartition($Data_i$=>knn($Data_i$，DataA，K))
5. val gobalKnn=localKnn.reduceByKey((a，b)=>min(a，b)).collect
6. val results=calculateSum(DataA，gobalKnn)
7. val outliers=results.top(n)

3.4　算法时间复杂度分析

传统 KNN 异常检测算法的时间复杂度为 $O(mn^2)$，其中 m 表示维数，n 表示数据对象个数。在 SPKNN 算法中，数据集存储在 Hadoop 平台的 HDFS 中，根据数据集的大小对数据进行分块，每个块的大小为 64 MB，在 map 阶段给 1 个数据块分配 1 个 map 任务以完成 K 近邻的计算。SPKNN 算法中由原来 1 个主机处理的最耗时的 K 近邻计算被分散到了各个计算节点上，因此其时间复杂度为 $O(mn^2/C)$，其中 C 为计算节点的个数。

4　实验和结果分析

4.1　实验环境

本实验使用的 Spark 平台是由 5 台服务器搭建的真实物理集群，串行实验的运行环境和 Spark 的单服务器配置相同。

表 1 列出了每节点的硬件概要信息，表 2 列出了集群软件概要信息。

表 1 单台节点的硬件概要信息

处理器类型	CPU 数量/个	CPU 频率/GHz	硬盘容量/GB	内存/GB
Intel Xeon (R) CPU E5-2630	2	2.6	50	4

表 2 集群软件概要信息

软件	类型
操作系统	ubuntu-14.04 64 位
Java 版本	1.8.0
Hadoop 版本	2.6.5
Spark 版本	2.0.0
Scala 版本	2.11.0

4.2 评估指标

实验选取了 Aggarwal 等提出的异常点覆盖率[14]来衡量算法的有效性。选取比较常用的运行时间、加速比、可扩展性作为集群性能的评价指标[15]。

(1)异常点覆盖率(CoverageRate)，该指标指的是找到的正确异常点数量和实际异常点数量的比值。

(2)运行时间(Runtime)，该指标指的是完成异常检测所需的时间，包括读取文件占用的时间，该值越小越好。

(3)加速比(Speedup)，该指标用来衡量并行算法的高效性，理想状态下根据阿姆达尔定律，其值等于所使用的计算资源核心数，定义为：

$$speedup=\frac{base\_time}{parallel\_time} \tag{3}$$

其中，$base\_time$ 表示串行程序运行的时间，$parallel\_time$ 表示并行程序运行的时间。

(4)可扩展性(Extension)，该指标反映的是集群中节点数目变化对并行化效率的影响，用于检验并行算法的高效性。

4.3 实验分析

4.3.1 有效性分析

为验证本文算法的有效性，采用 UCI 数据集中的 Penbased 数据集，将原始数据某一类中的部分对象删除，使数据成为非平衡数据，便于实验比较分析。首先在单机环境下测试了传统 KNN 异常检测，然后保持其他参数相同在集群环境下测试 SPKNN 算法。

从表 3 可以看出，SPKNN 算法和基于单机的传统 KNN 异常检测算法的异常点覆盖率相差无几，说明 SPKNN 算法具有比传统 KNN 异常检测算法检测精度高的优点。同时 SPKNN 算法比传统 KNN 异常检测算法的运行时间短，时间优势并不明显的原因是 Penbased 数据集的记录数不够多。

表 3 两种算法的实验结果

算法	CoverageRate/%	Runtime/s
传统 KNN 异常检测	73.33	34.334
SPKNN 算法	73.33	25.935

4.3.2 参数影响分析

SPKNN 算法需设置 3 个参数，分别是近邻个数 k，分区个数 m，异常点个数 n。采用单一变量原则，保证其中两个变量不变，通过改变剩下这一变量的值，观察该变量对实验结果的影响。从 UCI 数据集中挑选了 3 个数据集进行实验，并按照 keel 数据集的格式进行输入[16]。数据集的信息如表 4 所列。

表 4 数据集信息

数据集名	样本数	特征数
Banana	5300	2
Penbased	10922	10
Poker hand	100000	10

实验 1 当 $n=30$，$m=4$ 为常量，最近邻个数 K 为变量时，算法的执行时间如表 5 所列。

表 5 不同 K 值下 SPKNN 算法的运行时间

Dataset	$K=10$	$K=20$	$K=30$	$K=40$	$K=50$
Banana	19.03	20.20	20.26	22.51	21.93
Penbased	24.23	26.85	31.92	36.58	53.05
Poker hand	536.71	676.06	790.35	753.88	801.46

从表 5 中可以看到，随着 K 值的增加，运行时间一直在增加。K 的数值决定了 KNN 中 K 的个数，影响着异常度的计算，因此过小的 K 值不能完整地反映近邻信息，而过大的 K 值会增加运算量，因此应选择一个合适的 K 值来达到较好的检测效果。

实验 2 当 $K=10$，$m=4$ 为常量，n 为变量时，算法的执行时间如表 6 所列。

表 6 不同 n 值下 SPKNN 算法的运行时间

Dataset	$n=20$	$n=40$	$n=60$	$n=80$	$n=100$
Banana	19.47	19.32	21.44	22.53	21.09
Penbased	25.66	26.12	27.52	29.21	32.43
Poker hand	660.54	698.01	717.87	712.24	718.53

从表 6 中可以看到，随着 n 值的增加，运行时间一直在增加。n 的取值和异常点的数量密切相关，n 值太小无法选出全部的异常点，n 值太大会导致运行时间增加，因此需根据用户的需求选择合适的 n 值。

实验 3 当 $K=3$，$n=30$ 为常量，m 为变量时，算法的执行时间如表 7 所列。

表 7 不同 m 值下 SPKNN 算法的运行时间

Dataset	$m=2$	$m=4$	$m=6$	$m=8$	$m=10$
Banana	18.44	19.06	25.28	24.49	26.64
Penbased	26.61	29.53	31.33	34.26	36.39
Poker hand	547.70	534.95	490.71	373.71	289.96

从表 7 中可知，当数据量比较少时，随着 m 的增加，运行时间线性增加，因为对于较少的数据量，m 的增加会导致通信开销变大，时间线性增加。当数据量比较大时，随着 m 的增加，运行时间减少，因为并行度的增加有利于分布式处理，但随着并行度的增加，效率增长会慢慢达到某个饱和点，在这点上集群各个节点的通信时间远远大于计算时间，算法的性能趋于平缓甚至变弱。

4.3.3 集群性能分析

本实验采用机器学习中的 Poker hand，该数据约有 100 万条记录，包含 10 个数值型属性。实验中需要 3 种大小的数据集，从中选择 3 个不同规模数据集的样本，样本如表 8 所列。

表 8 实验样本数据情况

数据集名	记录数	样本大小/MB
Dataset1	100000	2.24
Dataset2	150000	3.36
Dataset3	200000	4.48

(下转第 366 页)

[12] 韩风英,李云. 利用复合混沌系统的图像加密方案研究与设计[J]. 电脑知识与技术,2010,6(13):3450-3452.
[13] 黄婵,曾小龙. 基于可逆记忆 CA 融合时间延迟的图像加密算法研究[J]. 科学技术与工程,2014,14(3):86-92.
[14] 孙倩,胡苏. 基于改进 cat 映射与混沌系统的彩色图像快速加密算法 [J]. 计算机应用研究,2017,34(1):233-237.
[15] 马俊明,叶瑞松. 对"基于改进广义 cat 映射的彩色卫星图像加密算法"的安全性分析 [J]. 汕头大学学报(自然科学版),2016,31(2):50-58.
[16] LI G D,WANG L L. Double chaotic image encryption algorithm based on optimal sequence solution and fractional transform[J]. Visual Computer,2018,1(1):1-11.

(上接第 352 页)

实验 4 分别选择 1,2,3,4,5 这 5 个节点参与运算,处理 3 个数据集,考察在运算节点逐渐增加的情况下,Spark 集群完成任务的时间,计算 SPKNN 算法的加速比。经过多次运行测试,实验结果如图 3 所示。

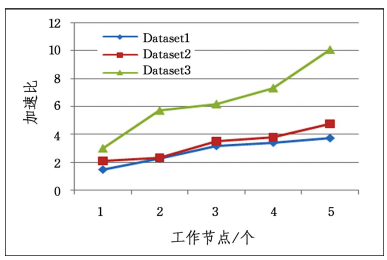

图 3 加速比实验结果

从图 3 中可以看出,随着节点的增加,SPKNN 算法的加速比是接近线性的,并且随着数据规模的增大,算法的加速比性能越来越好。这说明 SPKNN 算法具有良好的加速比。

实验 5 对比当 Spark 集群的工作节点个数不同时,集群处理不同规模数据时的运行时间,实验结果如图 4 所示。

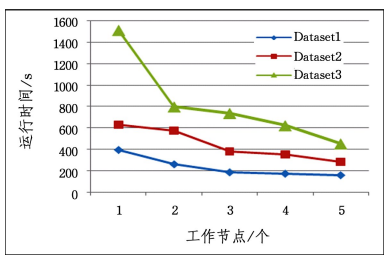

图 4 扩展性实验结果

由图 4 可知,随着 Spark 分布式集群工作节点数量的增加,3 个样本的运行时间呈明显下降趋势,然后趋于稳定,这是因为节点间通信开销和运算能力的增加达到相对平衡,整个集群的效率趋于稳定。同时可以看出:数据规模越大,下降趋势越明显。实验表明,该算法在大数据集上具有良好的可扩展性。

结束语 传统 KNN 异常检测算法在处理大数据集时存在两个主要问题:运行时间和内存消耗。Spark 平台可以为 KNN 异常检测算法的并行提供简单、透明、有效的环境。本文结合传统 KNN 异常检测算法的特点和 Spark 基于内存迭代计算的优点,实现了基于 Spark 平台的并行 KNN 异常检测算法,在保证异常检测精度的前提下,提高了 KNN 异常检测算法在处理大数据集的计算效率。实验表明:基于 Spark 平台的并行 KNN 异常检测算法具有良好的加速比和扩展性,能够有效应对数据集的快速增长。接下来将进一步研究基于 Spark 平台的飞行数据异常检测,把本文算法应用到真实应用场景中。

参 考 文 献

[1] 陈运文,吴飞,吴庐山,等. 基于异常检测的时间序列研究[J]. 计算机技术与发展,2015(4):166-170.
[2] HODGE V J,AUSTIN J. A survey of outlier detection methodologies[J]. Artificial Intelligence Review,2004,22(2):85-126.
[3] 邹云峰,张昕,宋世渊,等. 基于局部密度的快速离群点检测算法[J]. 计算机应用,2017,37(10):2932-2937.
[4] KNORR,EDWIN M,NG,et al. Distance-based outliers:algorithms and applications[J]. Vldb Journal,2000,8(3-4):237-253.
[5] BREUNIG M M. LOF:identifying density-based local outliers [J]. ACM Sigmod Record,2000,29(2):93-104.
[6] 辛丽玲. 基于密度差异的离群点检测研究[D]. 北京:北京交通大学,2015.
[7] PAN Y,ZHANG J. Parallel Programming on Cloud Computing Platforms[J]. Journal of Convergence Volume,2012,3:23-28.
[8] SUBRAMANYAM R B V,SONAM G. Map-Reduce Algorithm for Mining Outliers in the Large Data Sets using Twister Programing Model[J]. International Journal of Computer Science and Electronics Engineering,2015,3(1):81-86.
[9] 郭一鹏,梁吉业,赵兴旺. 基于 MapReduce 的混合数据孤立点检测算法[J]. 小型微型计算机系统,2014,35(9):1961-1966.
[10] 苟杰,马自堂,张喆程. PODKNN:面向大数据集的并行离群点检测算法[J]. 计算机科学,2016,43(7):251-254.
[11] KNORR E M,NG R T. A Unified Notion of Outliers:Properties and Computation[C]// International Conference on Knowledge Discovery & Data Mining. 1997:219-222.
[12] RAMASWAMY S,RASTOGI R,SHIM K. Efficient algorithms for mining outliers from large data sets[C]// ACM SIGMOD International Conference on Management of Data. ACM,2000:427-438.
[13] 高彦杰. Spark 大数据处理技术、应用与性能优化[M]. 北京:机械工业出版社,2014.
[14] AGGARWAL C C,YU P S. Outlier Detection for High Dimensional Data[J]. ACM Sigmod Record,2001,30(2):37-46.
[15] ZHANG X,GONG K,ZHAO G. Parallel K-Medoids algorithm based on MapReduce[J]. Journal of Computer Applications,2013,33(4):1023-1005.
[16] ALCALÁ-FDEZ J,FERNÁNDEZ A,LUENGO J,et al. KEEL Data-Mining Software Tool:Data Set Repository,Integration of Algorithms and Experimental Analysis Framework[J]. Journal of Multiple-Valued Logic & Soft Computing,2011,17:255-287.