iLOF*: 一种改进的局部异常检测算法①

王 飞

(复旦大学 计算机科学技术学院, 上海 201203)
(复旦大学 上海市数据科学重点实验室, 上海 201203)

摘 要: 异常检测是数据挖掘领域研究的基本问题之一, 已被广泛应用于气象预报、网络入侵检测、电信和信用卡欺诈侦察等领域. 基于密度的异常检测算法 LOF 具有较好的检测效果和适用性, 但其计算量较大, 运行效率不够高, 且在进行对象之间的距离计算时忽略了不同属性对异常值的不同影响. 针对以上不足, 本文提出了一种高效的 LOF 改进算法 iLOF*. 该算法利用网格进行数据约简, 从而提高了算法的运行效率; 同时, 在进行对象之间的距离计算时, 引入信息熵, 给不同属性赋予不同的权值, 从而提高了算法的准确率. 另外, 用 MapReduce 计算框架将 iLOF*算法并行化, 进一步提高了算法在大规模数据集上的运行效率. 最后的实验结果验证了 iLOF*算法的有效性和高效性.

关键词: 数据挖掘; 异常检测; 局部异常因子; 信息熵; 并行化

iLOF*: An Optimized Local Outlier Detection Algorithm

WANG Fei

(School of Computer Science, Fudan University, Shanghai 201203, China)
(Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 201203, China)

Abstract: Outlier detection is an important branch in the areaof data mining,It has been widely used in weather forecasting, network intrusion detection, telecommunications and credit card fraud detection,etc. LOF algorithm has good detection effect and availability, but its computation is very high, whose efficiency is not good enough,And when calculating the distance between two objects, LOF algorithm ignores the different influence of different properties.To solve above disadvantages, we put forward an improved outlier detection algorithmiLOF*. iLOF* algorithm usesgrid to reduce the data sets, so as to improve the efficiency of the algorithm; at the same time, when calculating the distance between the object, iLOF* algorithm gives different weights to different properties through the introduction of information entropy, which improve the accuracy of the algorithm.In addition, we use the parallel computing framework MapReduce to parallel iLOF * algorithm, which further improves the efficiency of algorithm on large data sets.The experimental results demonstrate the effectiveness and efficiency of the proposed algorithm.

Key words: data mining; outlier detection; local outlier factor; information entropy; parallelization

数据挖掘是从大量数据中寻找其规律的技术[1]. 异常检测是数据挖掘任务中的重要内容之一. 异常检测又称为孤立点检测、小事件检测、偏差检测等. 异常点是指在数据集中, 有些数据表现地如此与众不同, 以至于使人怀疑这些数据的出现并非是随机的, 而是有可能产生于完全不同的机制[2]. 异常数据产生的背后可能隐藏着某些重要信息, 这些信息可能对我们了解数据、分析数据有很大的帮助, 或者这些异常数据的产生可能会带来严重的后果, 例如网络入侵检测中, 异常数据(恶意访问等)会使得网络中的数据丢失、程序被篡改或者终端不能工作等. 目前异常检测已经被广泛应用于气象预报、网络入侵检测、电信和信用卡欺诈侦查等领域, 因此, 对数据的异常检测具有重要的应用背景.

① 收稿时间:2015-04-28;收到修改稿时间:2015-06-08

现有的异常检测算法通常仅给出一个点是否异常的二元属性(要不是异常点, 要么不是异常点), 然而实际生活中的数据集是复杂和多样的, 往往不能由布尔变量(非 0 即 1)来简单的表示. 为此, 研究人员提出了基于密度的局部异常因子检测算法[3]能够用来判断一个对象的异常程度. 该方法效果较好, 适应性强, 存在计算量大的不足[4], 并且在计算对象之间的距离时没有考虑不同属性上的权重, 没有体现不同属性对异常值不同的贡献.

针对现有算法计算量大和忽略了不同属性对异常值的不同影响的不足之处, 本文提出了一种新的异常检测算法 iLOF*. 在 iLOF*算法中, 用网格[5,6]进行数据约简, 筛选出高密度区的数据, 然后只需计算部分稀疏区域数据的 LOF 值, 大大上加快了异常点挖掘速度, 同时用 MapReduce 计算框架对算法并行化, 进一步提高了算法的时间效率. 在计算对象之间的距离时, 引入信息熵[7], 通过计算数据对象在每个属性上的去一划分信息熵差量 Δ, 用去一划分信息熵差量 Δ 确定属性的权重. 通过给异常程度较大的属性赋予大的权值, 来提高算法的异常检测准确率.

1 相关工作

异常检测已被广泛应用于诸多领域. 现有的异常检测算法一般分为以下几类: 基于分布的异常检测算法、基于聚类的异常检测算法、基于距离的异常检测算法、基于密度的异常检测算法等.

Breunig 首次提出的局部异常因子(LOF)可以用来判断一个对象关于其局部邻域内的异常程度. LOF 算法为数据集中的每一个数据对象赋予一个表征该数据对象异常程度的因子, 即, 局部异常因子 LOF. LOF 具有较好的检测效果和适用性, 但其计算量较大, 运行效率不够高, 且在进行对象之间的距离计算时忽略了不同属性对异常值的不同影响.

Agyemang 等[8]以距离对象最近的 k 个对象的最大距离作为 k 距离, 提出了局部稀疏系数(Local sparse coefficient,LSC), 在一定程度上减少了计算次数. Papadimitriou[4] 等提出了局部关联整数(Local correlation Integral, LOCI) 的概念, 引入了多粒度的偏差因子 MDEF(Multi-Granularity Deviation Factor)来度量对象的异常程度, 通过比较对象的 k-邻域中包含的对象个数与其邻域中所有对象的 k-邻域中平均包含的对象个数得到的, 这种方法不需直接计算数据点的密度. 基于连接的离群系数(Connectivity based Outlier Factor, COF)[9]算法中, 根据给定的参数最少邻居数 k 和数据对象的连接性来确定邻域, 计算与其邻域的平均连接距离, 用平均连接)距离比作为基于连接的离群系数 COF. 以上算法在一定程度上解决了 LOF 计算量大的缺点, 运行效率比 LOF 算法要高一些, 不过总体计算量还是比较大, 在处理大规模数据时, 运行效率还不够高. 同时, 没有体现不同属性对异常值的贡献不同.

2 LOF算法基本概念

基于密度的局部异常挖掘算法不再把异常看做是一种二元属性, 而是用局部异常因子 LOF 来表示对象的异常程度. 对象 p 的局部异常因子反映了该对象的异常程度, 局部异常因子$\mathrm{LOF_k}(p)$越大, 则该对象是异常数据的可能性越大; 反之, 则该对象是异常数据的可能性越小.

LOF 算法的基本流程如下:

I)计算对象 p 的第 k 距离(k-distance), 假设数据集 D 中的两个数据对象 p 和 o, 对任意的正整数 k, 定义对象 p 的第 k 距离为 p 与 o 之间的距离, 记为 k-distance(p), 这里的对象 o 满足如下条件:

1) 至少存在 k 个对象 $o'\in D\backslash\{p\}$ 满足 $d(p,o')\le d(p,o)$;

2)至多存在 k-1 个对象$o'\in D\backslash\{p\}$满足 $d(p,o')< d(p,o)$;

其中, 对象 p 和对象 o 的距离记作 d (p,o). d (p,o)的计算公式为:

$$d(p,0)=\sqrt{\sum_{i=1}^{d}(f(p_i)-f(0_i))^2} \qquad (1)$$

d 为数据集 D 的维度, $f(p_i)$和$f(o_i)$是第 i(i=1,2,…,d)维属性的值.

II)计算对象 p 的第 k 距离邻域$N_{k-distance}(p)$, 对象 p 的第 k 距离邻域是数据集 D 中与对象 p 的距离不超过对象 p 的第 k 距离 k-distance(p)的数据对象的集合, 即:

$$N_{k-distance}(p)=\{q\mid d(p,q)\le k-distance(p)\} \qquad (2)$$

式中, $N_{k-distance}(p)$在本文中简记为$N_k(p)$.

III)计算对象 p 与其第 k 距离邻域中对象的的可达距离, 对象 p 相对于对象 o 的可达距离定义为:

$$reach-distance(p,o) = \max\{k-distance(o), d(p,o)\} \quad (3)$$

其中, $o \in N_{k-distance}(p)$.

IV)计算对象 p 的局部可达密度$ldr_k(p)$, 对象 p 与它的第 k 距离邻域$N_{k-distance}(p)$中所有对象的平均可达距离的倒数定义为对象 p 的局部可达密度, 计算公式为:

$$ldr_k(p) = \frac{1}{\sum reach-dis\tan ce(p,o)/N_k(p)} \quad (4)$$

V)计算对象 p 的局部异常因子, $LOF_k(p)$: 计算公式为:

$$iof_k(p) = \frac{\sum_{0\in N_k(p)} idr_k(0)/idr_k(p)}{|N_k(p)|} \quad (5)$$

对象 p 的局部异常因子反映了该对象的异常程度, 局部异常因子$LOF_k(p)$越大, 则该对象是异常数据的可能性越大.

3 iLOF*算法

在 LOF 算法的基础, 本文提出了一种新的异常检测算法 iLOF*. 本章介绍 iLOF*算法基本思想和步骤.

3.1 基于网格的方法快速过滤数据中非异常点

GDLOF[6]算法证明了稠密单元和稠密区域中的点不可能成为异常点. 下面简单介绍 GDLOF 算法的基本思想.

将数据集 D 的每一维进行等宽间隔划分(间隔为 σ), 形成一个网格结构. C 是其中的一个网格. P 是数据集 D 中的一个对象. 如果对象 p 的所有的 m 最近邻 q 都在网格 C 中, 并且 q 的所有的 m 最近邻 o 也在网格 C 中, 此时网格 C 就被称为稠密单元[8].

单元 $C_{i_1,i_2,\cdots,i_d}$ 和 $C_{j_1,j_2,\cdots,j_d}$ 是邻居单元. 如果满足如下条件:

$$\forall p(1 \le p \le d), |i_p - j_p| \le 1$$

这里, $i_1, i_2, \ldots, i_d$和$j_1, j_2, \ldots, j_d$分别为单元 $C_{i_1,i_2,\cdots,i_d}$ 和 $C_{j_1,j_2,\cdots,j_d}$ 的间隔标识部分的序列[8].

如果 p 的所有 m 最近邻 q 和 q 的所有的 m 最近邻 o 都在超单元集$\{C_1, C_2, \ldots, C_m\}$里($C_i$和$C_j$是邻居单元), 则这个超单元集被称为稠密区域 C-Dence[8].

根据以上思想, 用网格进行数据约简的过程如下:

输入: 数据集 D
参数 k
D 每个属性值上的间隔集$(\sigma_1, \sigma_2, \ldots, \sigma_d)$
输出: 候选异常集 E

对于 D 中的每一个点 p, 依次执行

I 根据网格间隔集将 p 映射到一个超单元格C_i中并且用$R^* - tree$节点对应于C_i.

II 如果 p 的所有 k 最近邻在C_i中, 并且 p 的所有 k 最近邻 q 也在C_i中, 则将C_i标记为稠密单元, 或者如果 p 的所有 k 最近邻 q 在超单元集$\{C_1, C_2, \ldots, C_m\}$里($C_i$和$C_j$是邻居单元), 并且 p 的所有 k 最近邻 q 也超单元集$\{C_1, C_2, \ldots, C_m\}$里, 则将$\{C_1, C_2, \ldots, C_m\}$标记为稠密区域

不断更新稠密单元和稠密区域.

iLOF*用网格的方法进行数据约简, 筛选出高密度区的数据, 只对分布在边缘的数据进行 LOF 值计算, 最后统计出具有较高 LOF 值的数据作为异常点. 只计算部分稀疏区域数据的 LOF 值, 很大程度上加快了异常点挖掘速度.

3.2 去一划分信息熵差量确定属性权值

在信息论中, 把表征信息和随机变量的不确定性定义为信息熵. 设一个随机变量 X, 集合为 S(X)为变量 X 取值的范围, 变量 X 的概率函数为 p(x), 则变量 X 的信息熵记为:

$$E(X) = \sum_{x\in s(x)} p(x)\log(p(x)) \quad (6)$$

信息熵 E(X)越大, 则随机变量 X 的不确定性就越大.

设一个对象集 D, 对象集 S 是对象集 D 的一个子集, 并且$|S| > 1$, 对象 q 是对象集 S 中的一个对象, 对象集 S 被对象 q 划分为$\{q\}$和 $S - \{q\}$两个部分,

记为$\hat{C} = \{C_1, C_2\}$, 得到:

$$E(\hat{C}) = \sum_{k=1,2}\left(\frac{|C_k|}{|D|}(C_k)\right)) \quad (7)$$

则 $E(S) - E(\hat{C})$ 为对象集 S 的去一划分信息熵差量[10], 记为 $\Delta(q,s)$

$$\Delta(q,s) = E(S) - E(\hat{C}) \quad (8)$$

在对象集 S 明确的情况下, 下文中简记为 $\Delta(q)$.

从去一划分信息熵差量的定义可知, 对象集 S 的去一划分信息熵差量 $\Delta(q)$表征的是对象集 S 被划分前后, 其信息熵的变化量, 此处信息熵减少的值就是数据集 S 中消除的不确定性[7,10]. 从统计学的角度来看, 异常检测就是要检测出在数据集中占比小的数据对象, 因为在子集 S 中 $\Delta(q)$值大的对象恰恰是占比小的数据对象, 因此, 这些对象更有可能是异常对象. 在对象集中, 对于某个具体的对象来说, 每一个属性对

这个对象的异常的贡献度的大小, 就是看这个对象在这一属性值上的去一划分信息熵差量的大小, 即 Δ(q). 所以, 我们用这个 Δ(q)要表示属性的权值.

根据对象 q 在各个属性的去一划分信息熵差量 $\Delta(q_i)$作为计算对象 q 与其他对象距离时该属性上的权值, 重新计算数据集 D 中两个对象之间的距离.

定义 1 对象 p 和对象 q 的距离 d(p ,q): 假设数据集 D 的维度为 d, p 和 q 是数据集 D 中的两个数据对象, f(p)和 f(q)是第 i(i=1,2,…,d)维属性的值, 则 p 和 q 的 d 维属性的加权距离为:

$$d(p,o,w_j)=\sqrt{\sum_{i=1}^{d} w_j(f(p)-f(p))^2} \quad (9)$$

w_j 是对象 p 的第 j 维属性的权值, $0\le w_j\le 1$(这里对 w_j 做了归一化处理).

iLOF*算法

为了解决 LOF 算法计算量大的问题, 用网格进行数据约简, 筛选出稠密单元和稠密区域中的数据, GDLOF 算法已经证明了稠密单元和稠密区域中的点不可能成为异常点. 因此, 我们只需计算部分稀疏区域数据的 LOF 值, 这大大上加快了异常点挖掘速度. 另外, 我们用 MapReduce 计算框架对算法并行化, 进一步提高了算法的时间效率.

为了提高 LOF 算法的检测准确率, 在计算对象之间的距离时, 引入信息熵[7], 通过计算数据对象在每个属性上的去一划分信息熵差量 Δ, 用去一划分信息熵差量 Δ 确定属性的权重. 通过给异常程度较大的属性赋予大的权值, 提高了算法的检测准确率.

iLOF*算法的基本步骤如下:

iLOF*算法

输入: 数据集 D

输出: 异常点集合 S

I 使用网格的方法进行数据筛选, 检测出 D 中的稠密单元和稀疏区域(GDLOF 算法证明了稠密单元和稠密区域中的点不可能成为异常点), 剩余的稀疏区域中的数据组成候选异常集合 E

II 并行计算 E 中的 LOF 值, 并将其按照 LOF 降序, 输出前 n 个 LOF 值最大的对象组成异常点集合 S

3.4 iLOF*算法的并行化

Hadoop 是一个用来处理大规模数据的分布式系统, 其核心组件是分布式数据处理模型 MapReduce[11] 和分布式文件系统 HDFS[12]. MapReduce 处理分为两个阶段: Map 阶段和 Reduce 阶段. 每个阶段以 key-value 键值对作为输入和输出.

为了将计算 LOF 值的任务基于 MapReduce 模型对算法进行并行化, 我们需要将 LOF 值的计算任务适应 MapReduce 编程模型. 考虑到 MapReduce 的工作机制, 设 d 维的数据集 D 作为输入, 数据集 E 为用网格的方法过滤后得到的位于系数区域的数据的集合(E 是 D 的子集), 对数据集 E 中的对象计算 LOF 值的过程主要分成 4 个阶段.

并行计算 E 中对象的 LOF 值的过程如下:

输入: 数据集 D, 候选异常数据集 E

参数 k

异常点个数 n

输出: 异常集 S

I 计算数据集 D 中各个维度上的每个出现的值出现的次数<key1,value1>(key1 为在该维度上出现的一个值 value, value1 为在该维度上 key1 出现的次数), 同时计算出数据集 D 上各个维度上的信息熵 E(i)(i=1,2,…,d).

II 找到数据集 E 中每个对象的 k 距离邻域.

III 计算数据集 E 中每个对象的 ldr 值.

IV 计算数据集 E 中每个对象的 lof 值.

段, 输入为数据集 D(数据集 E 为 D 的一个子集, 用一个布尔型的变量标志), 输出为数据集 D 中各个维度上的每个出现的值出现的次数<key1,value1>和数据集 D 上各个维度上的信息熵 E(i)(i=1,2,…,d), 结果存储在 HDFS 文件中. Map 阶段输入为数据集 D, 输出为 $<d_i, value_i\_count_i>$, 其中 d_i 为维度数(i=1,2,…,d), $value_i$为该 Map 中输入对象中在d_i维度上出现的一个值, $count_i$为该$value_i$值在d_i维度上出现的次数. 在 Reduce 阶段, 输入为在 Suffle 阶段重排的键值对 key-values, Reduce 中会将 $value_i\_count_i$ 解析为 $<value_i, count_i>$, 然后统计出在该维度上出现的所有值和该值出现的次数, 同时, 也会计算出该维度上的信息熵 E(i). 这主要是为了避免在后面计算去一划分信息熵差量是的重复计算.

在第 II 阶段, 输入为数据集 D(数据集 E 为 D 的一个子集, 用一个布尔型的变量标志), 输出为数据集 E 中每个对象与其 k 距离邻域中的每个对象的局部可达距离. Map 阶段输入为数据集 E, 在每个 Map 中, 根据公式(8)计算的每个对象在各个维度上的去一划分信息

熵差量 Δ(在计算的过程中要读入第一阶段中计算出的在对应维度上的<$value_i$, $count_i$>和信息熵 E(i))(详见 2.2 节), 然后根据公式(9)计算对象和其他对象的距离(详见 3.1 节), 数据集 D 中的一个对象 p 与其他所有对象的距离会在一个 Map 中计算, 计算对象 p 的第 k 距离 k-distance(p)和 k 距离邻域$N_{k-distance}(p)$, 这里输出有两种, 一种是对象 p 的 k 距离 k-distance(p), 另一种为对象 p 与其 k 距离邻域$N_{k-distance}(p)$中对象的距离 $d(p,q)$($q \in N_{k-distance}(p)$). 在 Reduce 阶段, 根据公式(3)计算出对象 p 的 k 距离邻域$N_{k-distance}(p)$中对象的局部可达距离 $reach-distance(p,q)$ ($q \in N_{k-distance}(p)$).

在第 III 阶段, 输入为对象 p 其 k 距离邻域 $N_{k-distance}(p)$ 中对象的局部可达距离 $reach-distance(p,q)$ ($q \in N_{k-distance}(p)$), 输出为对象 p 的局部可达密度$ldr_k(p)$. 在 Map 阶段, 不做计算处理. 在 Reduce 阶段, 根据公式(4)计算对象 p 的局部可达密度$ldr_k(p)$, 并输出.

在第 IV 阶段, 输入为对象 p 其 k 距离邻域 $N_{k-distance}(p)$中对象的局部异常因子$LOF_k(p)$($q \in N_{k-distance}(p)$), 输出为对象 p 的局部可达密度$ldr_k(p)$. 在 Map 阶段, 不做计算处理. 在 Reduce 阶段, 根据公式(5)计算对象 p 的局部异常因子$LOF_k(p)$, 并输出.

四步计算出了对象集 E 中每个对象的 LOF 值, 然后将 LOF 值按照降序排序, 最终将具有较高 LOF 值得数据输出, 作为异常点.

4 实验结果与分析

本文用 LOF 算法作为基准算法, 设计了几组对比实验, 对本文所提 iLOF*算法的执行效率、检测准确率进行验证.

实验环境:

实验平台配置: 9 台相同配置的 PC 机(通过局域网连接), CPU 为 Intel(R) Xeon(R) CPU E5-2609 v2 @ 2.50GHz, 内存 128G, 操作系统为 CentOS release 6.5. Hadoop 版本信息: hadoop2.0, 一个主节点 master, 8 个分节点 slavers. 编程语言为 java 语言, eclipse 编译环境.

实验数据集:

实验数据为 UCI 提供的公共数据集, 包括网络入侵检测数据集 KDD-CUP1999[13] 和 UCI KDD ARCHIVE FOREST COVERTYPE 数据集[14], KDD-CUP1999 数据集中包含 5 个大类, 包括正常连接、dos、u2r、r21、probe 入侵和攻击等. 在我们的实验中对 KDD-CUP1999 数据集进行一定的预处理, 使得攻击数据(即异常点)在总数据集的占比为 2%. 本实验选择了其中的 20 个连续属性维度.

4.1 算法的准确率比较

我们使用准确率来评估异常检测精度的性能, 准确率度量公式如下:

$$准确率 \frac{正确找到的异常点个数}{异常点总数目}$$

如图 1 所示横坐标为数据集和算法, 纵坐标为准确率. 图 1 是 LOF 算法和 iLOF* 算法分别在 KDD-CUP1999 数据集和上的实验结果. 实验中做了算法分别加入网格约简和信息熵对属性赋权值等情况的准确率对比. LOF 表示基本的居于异常因子算法; iLOF*(无信息熵)表示加入了网格约简, 没有加入信息熵对属性赋予不同权值; iLOF*(无网格约简)表示加入了加入信息熵对属性赋予不同权值, 没有网格约简; iLOF*表示同时加入了网格约简和信息熵对属性赋予不同权值.

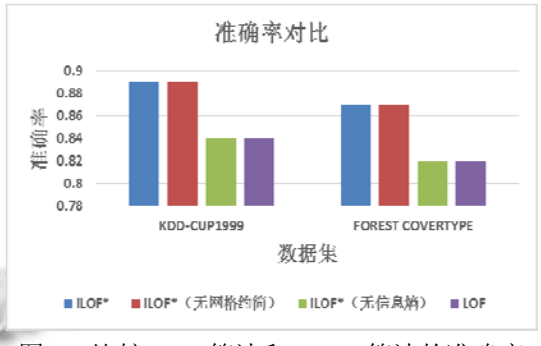

图 1 比较 LOF 算法和 iLOF*算法的准确率

从图中可以看出, 用网格对数据约简对准确率没有影响, 引入信息熵对属性赋予不同权值后, 在 KDD-CUP1999 数据集和 UCI KDD ARCHIVE FOREST COVERTYPE 数据集上准确率都有提升. 例如, 在 KDD-CUP1999 数据集上, iLOF*的准确率为 89%, iLOF*(无网格约简)的准确率为 89%, iLOF*的准确率为 89%, iLOF*(无信息熵)的准确率为 84%, LOF 算法的准确率为 84%, 前两者准确率相同, 后两者相同, 前两者要优于后两者.

在 UCI KDD ARCHIVE FOREST COVERTYPE 数据集上, 也类似.

4.2 算法的时间效率比较

时间效率一般用运行时间来衡量, 也就是在实验数据集上从读取参数后启动程序到数据实验结果所用的时间, iLOF*算法是工作在 Hadoop 集群上的, LOF 算法是在单机上工作的. 实验数据为: KDD-CUP1999. 图 2 是在 KDD-CUP1999 数据集上, LOF 算法和 iLOF*算法分别在单机上和集群上的运行效率对比.

如图 2 所示, 横坐标为数据集中数据的条数, 单位为百万条, 横坐标大小依次从 5 十万增长到 5 百万. 纵坐标为运行时间, 单位为 100 秒. 从图 2 中可以看出, 刚开始时, iLOF*算法在单机上的运行小路并没有比 LOF 算法的效率高, 这主要是因为 iLOF*算法需要用去一划分信息熵差量计算每个属性上的权值. 随着数据的增长逐渐体现出 iLOF*算法比 LOF 算法运行效率要高, 这主要是因为, iLOF*算法本身并没有计算全部数据集的 LOF 值, 而是先用网格对数据约简, 排除掉大量高密度区域中的非异常点, 只需计算稀疏区域中数据的 LOF 值, 这大大降低了 iLOF*算法的计算量.

然后我们对比 iLOF*算法在单机上的运行效率和在集群上的运行效率. 从图 2 中可以看出, 刚开始时, iLOF*算法在集群上的效率没有 iLOF*算法在单机上的效率高, 这主要是因为并行化的 iLOF*算法在集群上运行时, 其各个节点的启动、任务初始化等需要一定的时间. 当数据量不是很大时, 其效率并不明显, 但是, 当实验数据量较大时, 其效率就明显高于其在单机上运行的了, 并且随着数据量的增大, 并行的 iLOF*算法在的高效性也越来越明显.

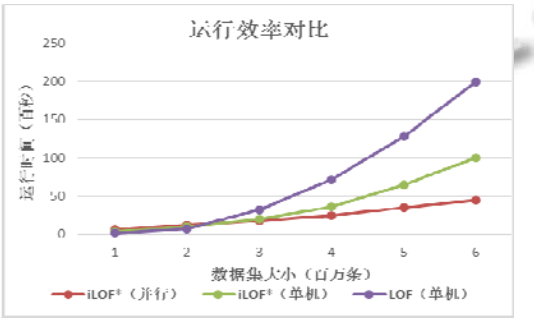

图 2 算法执行效率对比

5 结语

本文基于 LOF 算法, 提出了一种改进的异常检测算法 iLOF*. 并用实验验证了算法的有效性和高效性. 在 iLOF*算法中, 计算效率的提升受实验的参数(如: 数据属性的间隔距离值和 k 距离值)的影响. 对于参数的处理, 我们当前的做法还是根据实验来做参数的选择. 如何减少参数的使用或能自动地决定出参数的值, 是下一步要研究的问题.

参考文献

1 朱扬勇,熊赟.数据学:Dataology and data science.上海:复旦大学出版社,2009.
2 Hawkins DM. Identification of outliers. London: Chapman and Hall, 1980.
3 Breunig MM, Kriegel HP, Ng RT, Sander J. LOF: identifying density-based local outliers. 2000.
4 Papadimitriou S, Kitagawa H, Gibbons PB, et al. Loci: Fast outlier detection using the local correlation integral. Proc. of the 19th International Conference on Data Engineering 2003. IEEE, 2003: 315–326.
5 Ma Y, Shi H, Wang M. Adaptive local outlier probability for dynamic process monitoring. Chinese Journal of Chemical Engineering, 2014, 22(7): 820–827.
6 李存华,孙志挥.GridOF:面向大规模数据集的高效异常检测算法.计算机研究与发展,2004,40(11):1586–1592.
7 Shannon CE. A mathematical theory of communication. ACM SIGMOBILE Mobile Computing and Communications Review, 2001, 5(1): 3–55.
8 Agyemang M, Ezeife CI. Lsc-mine: Algorithm for mining local outliers. Proc. of the 15th Information Resource Management Association (IRMA) International Conference. New Orleans. 2004, 1: 5–8.
9 Tang J, Chen Z, Fu AWC, Cheung DW. Enhancing effectiveness of outlier detections for low density patterns. In Dvances in Knowledge Discovery and Data Mining. Springer Berlin Heidelberg.2002. 535–548.
10 Jiang F, Sui Y, Cao C. An information entropy–based approach to outlier detection in rough sets. Expert Systems with Applications, 2010, 37(9): 6338–6344.
11 Dean J, Ghemawat S. MapReduce: a flexible data processing tool. Communications of the ACM, 2010, 53(1): 72–77.
12 Shvachko K, Kuang H, Radia S, et al. The hadoop distributed file system. 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). IEEE, 2010: 1–10.
13 KDD-CUP1999 数据集: http://kdd.ics.uci. du/databases/ddcup99/kddcup99.html
14 UCI KDD ARCHIVE FOREST COVERTYPE 数据集: http://kdd.ics.uci.edu/databases/covertype/covertype.html