计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(2):151−156 [doi: 10.15888/j.cnki.csa.005435]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 MapReduce 与距离的离群数据并行挖掘算法①

任　燕

(山西省特殊教育中等专业学校, 太原 030012)

摘　要: 数据挖掘技术是解决数据丰富而知识贫乏的有效途径, 离群数据挖掘是数据挖掘领域中的重要研究内容之一, 已广泛应用于网络入侵检测, 信用卡诈骗, 垃圾邮件的分析和基因突变分析等领域. 在高维海量数据中, 由于数据量大和维度高, 严重影响了离群数据挖掘的精度和效率. 本文在 KNN 基础上, 通过定义“解集”的概念, 在 MapReduce 编程环境下, 实现了一种基于距离的离群数据挖掘算法. 分别采用人工数据集和 UCI 数据集, 实验验证了该算法在不同条件下, 参数对算法性能的影响.

关键词: MapReduce; 基于距离; KNN; 离群数据挖掘

引用格式: 任燕.基于 MapReduce 与距离的离群数据并行挖掘算法.计算机系统应用,2018,27(2):151–156. http://www.c-s-a.org.cn/1003-3254/5435.html

Parallel Mining of Distance-Based Outliers Using MapReduce

REN Yan

(Shanxi Special Education Secondary School, Taiyuan 030012, China)

Abstract: Data mining technology is an effective approach to resolve the problem of abundant data and scanty information. Outlier mining is one of the main research topic in the field of data mining, and it has been widely used in network intrusion detection, line card fraud, spam analysis, gene mutation analysis, etc. In high-dimensional data, the data volume and high dimension affect the effects of outlier data mining and efficiency seriously. In view of the high dimensional data, this study adopts the KNN implementing a distance-based outlier data mining algorithms under the MapReduce programming model by defining the “solving set”. Using artificial data set and UCI data set, the influence of parameters on the algorithm performance is discussed under different conditions in the experiment.

Key words: MapReduce; distance-based; KNN; outliers data mining

1　引言

离群数据 (Outlier) 是数据集中不满足数据的一般行为或模式, 明显偏离其他数据对象的数据[1]. 通常情况下, 离群数据容易被忽视, 但其可能蕴含大量非常有价值的信息. 离群数据挖掘是数据挖掘的一项重要任务, 其目标是寻找数据集中, 行为或模式很大程度上不同于其他数据对象的数据. 目前, 离群数据挖掘已广泛应用于信用卡诈骗检测[2], 网络入侵检测[3,4], 图像匹配[5], 数据清洗[6]以及医疗诊断等领域.

现有的离群数据挖掘方法大致分为四类: 基于统计分布的方法[7,8], 基于距离的方法[9−11], 基于密度的局部离群点方法[12,13]以及基于偏差的方法[14]. 这些方法针对不同类型的数据集可以进行较为有效的数据挖掘. 经过多年的探索和研究, 离群数据挖掘虽然已有较快发展, 但是当处理海量高维数据时, 仍然会出现算法的时空复杂度太高, 效率低下等问题, 无法满足当今社会的需求. 因此, 寻找一种准确高效的离群数据挖掘方法显得尤为重要.

① 收稿时间: 2016-01-31; 采用时间: 2016-03-08

基于距离的离群数据挖掘算法是一种有效的数据挖掘方法, 最早由 Knorr 与 Ng 提出[9], 其主要思路是将离群数据定义为与数据集中大多数数据之间的距离大于某个阈值的数据. 在此定义中, 距离阈值要预先进行设定, 而对于不同的数据集而言, 距离阈值可能不同, 所以如果要挖掘离群数据就必须对数据集有一定的了解. 针对其不足, Ramaswamy 与 Kyuseok 等人提出了一种在大数据集下挖掘离群数据的方法[15], 即 KNN 算法. 将离群数据定义为前 n 个与其 k 个最近邻居的距离和最大的数据, 从而避免了基于距离的离群数据挖掘算法需要用户设定距离阈值的局限. 文献[16]提出一种基于距离的离群数据挖掘算法, 不仅可以有效挖掘离群数据, 并且可以通过“解集”来预测新加入的数据是否为离群数据. 但是此算法需要计算所有数据之间的欧氏距离, 使得算法在高维、大数据集上的运行效率较低. 随着并行与分布式计算技术的发展, Angiulli 等人[17]在文献[16]的基础上提出一种基于 MPI 编程模型的离群数据并行挖掘算法. 但是由于 MPI 编程模型较为复杂, 可靠性较差, 并且会产生通信延迟和负载不均衡等问题, 因此本文选择可靠性较强, 效率较高的 MapReduce 编程模型, 在 Hadoop 平台下, 实现了一种基于 MapReduce 与距离的离群数据并行挖掘算法, 在保证算法准确性的前提下, 有效提高了数据处理的效率.

2 离群数据与 MapReduce 编程模型

2.1 基本定义与概念

假设数据集 D 是给定度量空间的有限子集, 相关概念定义如下:

定义 1 (离群权值). 对于任意给定的一个数据对象 $p\in D$, D 中 p 的权值 $w_k(p, D)$ 是 p 到它的 k 个最近邻的距离之和.

定义 2 (TOP n 离群数据). 假设 T 为数据集 D 中具有 n 个数据对象的一个子集. 如果不存在对象 $x\in T$、$y\in(D\backslash T)$, 使得 $(y, D)>(x, D)$, 那么, 子集 T 就称作数据集 D 中前 n 个离群值集合. 在此基础上, 权值为 $w^*=\min x\in T\ w_k(x,D)$ 的数据为离群程度最大的前 n 个离群数据中的第 n 个离群数据, 在子集 T 中的数据为数据集 D 中前 n 个离群值.

定义 3 (离群数据检测解集). 离群数据检测解集 S 是数据集 D 中一个子集, 对于每一个数据对象 $y\in D\backslash S$, 使得 $w_k(y, S)\leq w^*$,这里的 w^*第 n 个离群数据的权值.

我们注意到解集 S 总是包含在数据集 D 中的包含 top n 离群数据的集合 T, 并且, 它具有预测离群数据的性质.

2.2 “解集”算法实现

参考文献[16], 算法实现大致如下: 首先, 从数据集 D 中随机选取一个候选集 C_j, 将候选集 C_j 中每个数据对象与数据集中所有的数据对象进行比较, 存储 C_j 中每个数据对象的最近邻. 然后对 C_j 中每个数据对象的最近邻进行排序, 求得 C_j 中每个数据对象的权值. 在 C_j 中, 比第 n 个最大权值小的候选集对象不可能属于 top n 个离群数据, 被称为不灵活对象, 而剩余的对象就被称为灵活对象. 最初, C_1 子集包含从数据集 D 中随机选取的对象, 之后的 C_2, C_3,…,C_{j-1} 子集是由在之前的计算中没有插入到 C_1,…,C_j 中的数据集中的灵活对象组成. 如果一个对象变成了不灵活对象, 便不会是离群数据, 那么在之后的计算中, 就不会被插入到以后的候选集. 随着算法产生新的对象, 更多准确的权值会被计算出来, 不灵活对象的数量会增加. 当所有的数据对象被检测完毕, 即候选集中对象全为不灵活对象时, C_j 变为空, 算法结束. 解集就是在每次循环中计算出来的子集 C_j 的并集.

算法描述如下:

```
SOLVINGSET(Data, dist, n, k, m, r)
{//初始化解集和候选集
SolvSet = ∅;
Top =∅;
Cand =RandomSelect(Data, m);
while (Cand ≠ ∅) {
//数据集的遍历
SolvSet =SolvSet ∪ Cand;
Data =Data − Cand;
//候选集中数据的遍历
for each (p in Cand)
for each (q in Cand)
{//计算距离并更新数据集距离数组
δ =dist(p, q);
UpdateMin(NN[p], <q, δ>);
vif (p ≠ q) UpdateMin(NN[q], <q, δ>);v
}
NextCand =∅;
//候选集与数据集中其他数据进行比较
for each (p in Data)
{
```

```
for each (q in Cand)
if max{Sum(NN[p]), Sum(NN[q])} ≥ Min(T op))
{//计算距离并更新距离数组
δ =dist(p, q);
UpdateMin(NN[p], <q, δ>);
UpdateMin(NN[q], <q, δ>);
}
UpdateMax(NextCand,< p,Sum(NN[p])>);
}
//选取下一次的候选集
for each (q in Cand)
UpdateMax(T op, hq, Sum(NN[q])i);
Cand =CandSelect(NextCand, Data − NextCand, r);
  }
}
```

2.3 MapReduce 框架

MapReduce 是一个编程模型, 也是一个处理和生成超大数据集的算法模型的相关实现[18,19]. 用户首先创建一个 Map 函数处理一个基于 key/value pair 的数据集合, 输出中间的基于 key/value pair 的数据集合; 然后再创建一个 Reduce 函数用来合并所有的具有相同 key 值的 value 值. MapReduce 是基于数据划分的角度来构建并行计算模型的, 能够在大量的普通配置的计算机上实现数据的并行化处理. 这个系统在运行时只关心: 如何分割输入数据, 在大量计算机组成的集群上的调度, 集群中计算机的错误处理, 管理集群中计算机之间必要的通信. 采用 MapReduce 架构可以使没有并行计算和分布式处理系统开发经验的程序员有效利用分布式系统的丰富资源. MapReduce 框架可以运行在规模可以灵活调整的由普通机器组成的集群上: 一个典型的 MapReduce 计算往往由几千台机器组成、处理以 TB 计算的数据. 在 Google 的集群上, 每天都有 1000 多个 MapReduce 程序在执行.

MapReduce 编程模型的原理如图 1 所示.

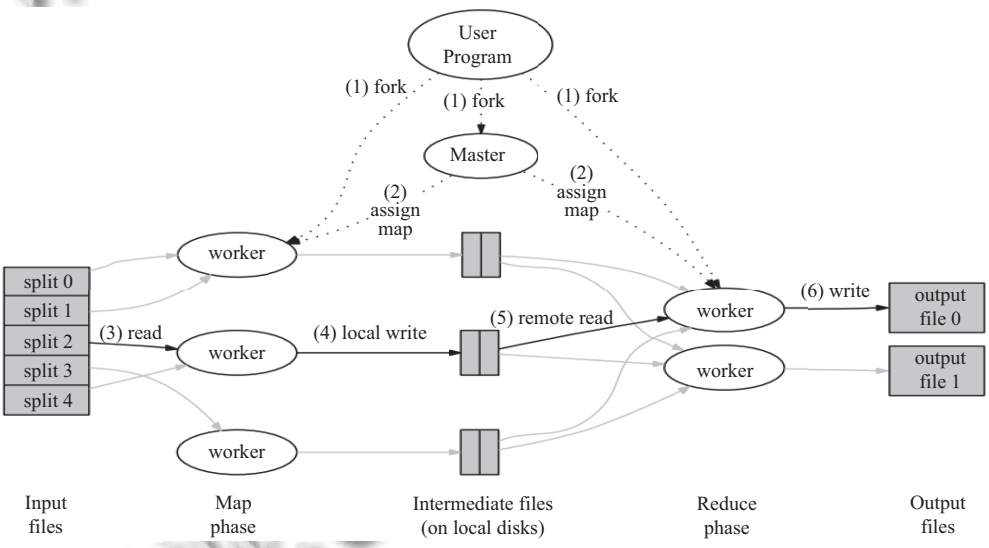

图 1 MapReduce 编程模型的原理

3 基于 MapReduce 的离群数据并行挖掘

3.1 “解集”算法的并行化

假设集群中有 1 个主节点 N_0 和 l 个从节点 $N_1,\cdots,N_i,\cdots,N_l$, 数据集 D 被分成 l 个局部数据集 $D_1,\cdots,D_i,\cdots,D_l$, 每个局部数据集被分配到一个从节点上, 目标是计算解集 S 和包含 top n 离群数据的集合 T.

主节点主要调度以下两项任务:

1) 分配任务到各个从节点, 使其同时进行核心操作运算.

2) 整合处理每个从节点完成运算后返回结果.

主节点同步处理从节点数据, 并对从节点返回结果进行整合, 来产生新的全局候选节点集, 这些候选集被用于接下来的迭代, 也用于产生它们每一个最近邻距离的真实列表 (顺序), 用来计算新的下界.

从节点的主要运算包含以下步骤:

(1) 接收当前候选集中的数据对象及前 n 个离群数据的下界, 并将下界作为 w^*.

(2) 将候选集中数据对象与本地数据对象进行比较.

(3) 求得新的本地候选对象和关于解集的本地最近邻的数据对象列表.

(4) 最后确定本地灵活对象的数目, 向主节点提交结果.

在局部数据集 D_i 中, 用 k 最近邻来计算候选集中每个数据对象的权值, 找前 n 个离群数据, 输出解集 *DSS* 和在数据集 D 中包含 top n 离群数据的集合 OUT. 在算法执行的开始时, DSS 和 OUT 被初始化为空集, 候选集 C 被初始化为从数据集 D 中随机选取的 m 个对象, 当候选集 C 变为空时结束主循环. 此刻属于候选集 C 的数据点被添加到集合 DSS. 在每个循环开始时, 候选集 C 被传递给运行在每个节点上的 NodeComp 比较程序, 最后由主节点汇总计算结果.

3.2 MapReduce 编程模型下的“分布式解集”算法实现

为了将该并行算法映射到 MapReduce 编程模型中, 本文将定义两个 Map 函数和两个 Reduce 函数. 在前一个 Map 函数中, 将局部数据集 D_i 中候选集 C_i 中的对象 p(object) 作为 Key 值, 计算 p 与候选集中所有对象 $q1$, $q2$, $q3$,..., ql 的欧式距离 dist($q1$), dist($q2$), dist($q3$),...,dist(ql), 以键值对的形式输出结果, 即: Map<p, dist($q1$)>, Map<p, $dist$($q2$)>, Map<p, dist($q3$)>,..., Map<p, dist(ql)>. 当 dist<ql>小于距离数组的最大值时, 更新每个对象的距离数组. Reduce 函数将 Map 函数输出的键值对进行整理 (shuffle) 和按键分组, 将 p 与候选集中对象的距离 dist($q1$), dist($q2$), dist($q3$),..., dist(ql) 进行降序排序, 并筛选前 k 个距离. 再通过 Map 函数计算 p 对应的前 k 个数据对象的距离和 sum(p), 即 p 权值, 将 Map 函数<p, sum(p)>形式的输出作为 Reduce 函数的输入, 最后通过 Reduce 函数将数据对象的 p 权值作为 Key 值, p 作为 Value 值, 并将 Key 值降序排序, 求得前 n 个离群数据. 具体实现如图 2 所示.

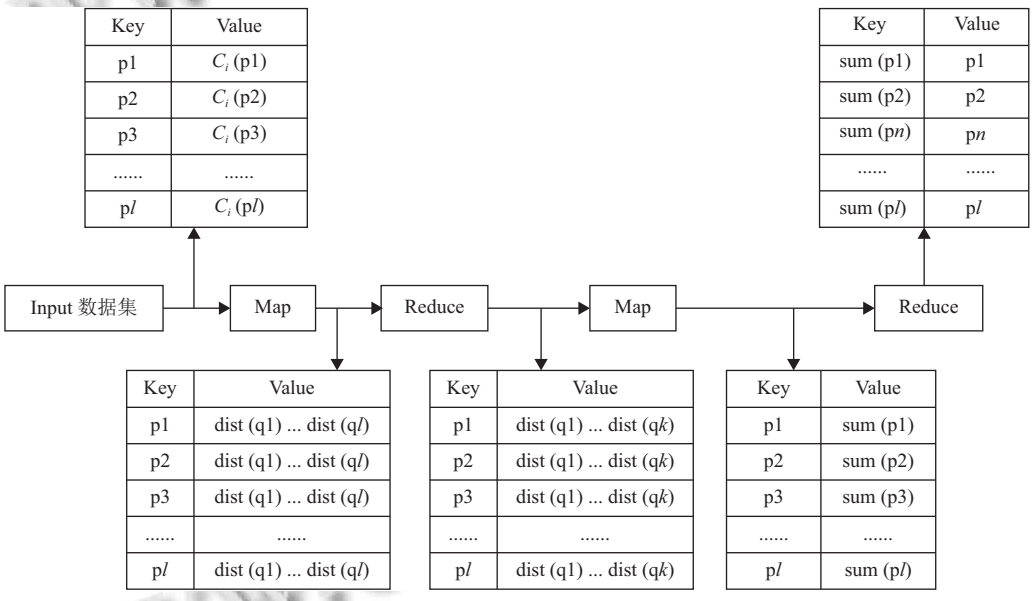

图 2 算法在 MapReduce 编程模型下实现

4 实验结果及分析

实验环境: 伪分布环境为, 1 台 Dell 笔记本 (Core i7 CPU, 8 G 内存); 全分布环境为, 由 10 台相同性能节点构成的集群 (Intel Core i7- 820M CPU, 16 G 内存) 操作系统为 ubuntu Linux 12.04; 实验平台: Jdk 1.6.0_37,Hadoop1.1.1, Eclipse. 采用 Java 语言作为开发工具. 分别在 MPI 和 MapReduce 并行环境下实现了该算法.

实验数据集: 1) 人工数据集: 参照文献[10]中的方式, 分别生成了 50、100、150 和 200 维的 30 000 条、60 000 条、90 000 条和 120 000 条, 共 16 组人工数据集. 在每一组人工数据集中, 包含有 30 条离群数据. 2) UCI 数据库中 Wholesale customers data 数据集.

4.1 人工数据集

选取参数如下: KNN 中 K=10, 节点个数 N=6, 数据量 D=30000 条, 在 MapReduce 框架下, 实验分析数据维度对挖掘性能的影响. 如图 3 所示.

当数据集的数量不变时, 随着数据维度的增加, 其

挖掘效率降低, 其主要原因是: 随着数据集维度的增加, 在进行 KNN 距离比较的时候计算复杂度略高于线性, 相应的挖掘效率要降低.

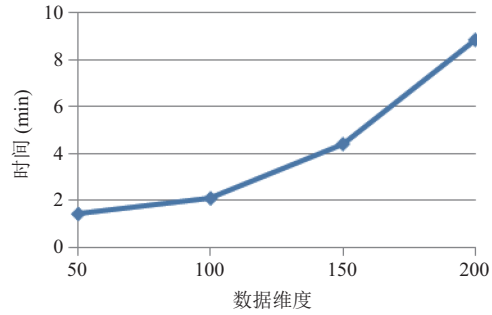

图 3 数据维度对算法性能的影响

选取参数如下: KNN 中 $K=10$, 数据量 D=30000 条, 数据维度 d=100, 在 MapReduce 框架下, 实验分析节点对挖掘性能的影响. 如图 4 所示.

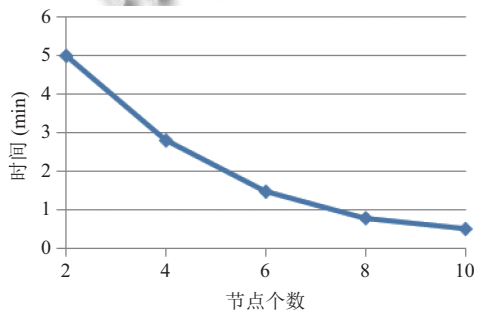

图 4 节点个数对性能的影响

当数据量一定时, 随着集群计算结点个数的增加, 算法的挖掘耗时基本按计算结点比例减小, 体现了算法具有较好的并行程度. 其主要原因是: 首先候选集中各个数据对象与本地数据对象进行比较计算过程完全可以并行化, 各计算结点的数据对象个数, 可以按计算结点比例分配.

选取参数如下: KNN 中 K=10, 节点个数 N=6, 维度 d=100, 分别在 MPI 和 MapReduce 框架下, 实验分析两种编程模型在不同数据量的情况下对挖掘性能的影响. 如图 5 所示.

当计算节点不变时, 随数据量的增加, 其挖掘效率降低, 其主要原因是: 随数据量的增加, 数据对象的个数增多, 各节点上分配的对象个数基本按计算节点比例线性增加; 每个对象对应的 KNN 查询的时间也随之增加. 总之, 随着数据量的增加, 其挖掘效率曲线的倾斜程度要略高于线性. 从图中看出, MapReduce 编程模型比 MPI 编程模型效率高, 主要原因是 MapReduce 比 MPI 容错能力强, 并且负载均衡策略更为合理.

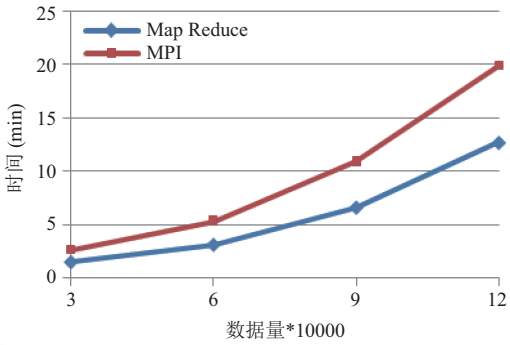

图 5 两种编程模型在不同数据量的情况下对挖掘性能的影响

4.2 UCI 数据集

Wholesale customers data 数据集, 分别在伪分布环境的 MapReduce 编程模型和 MPI 编程模型下, 实验分析 k 值对挖掘精度的影响.

由图 6 可以看出, k 值对两种编程模型的影响几乎相同. 当 k 值较小时, 算法挖掘准确率较低, 当 k 值增大到一定值时, 挖掘准确率会上升到一个比较高的值, 大约 80%, 主要原因是当 k 值增大到一定值时, 候选集中数据对象的最近邻已经能够较好地体现数据的疏密特征或分布趋势, 随着 k 继续增大, 由 KNN 得到的离群数据可能会出现较小误差.

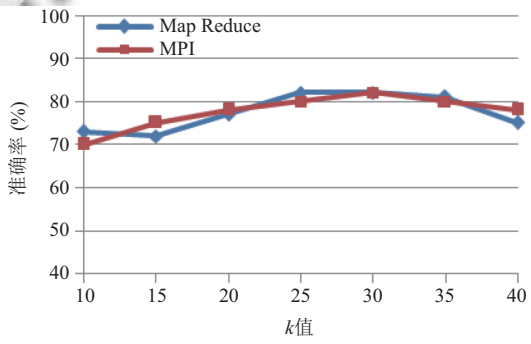

图 6 两种编程模型的 k 值对挖掘精度的影响

选取 UCI 数据集中 DOCC (Default of Credit Card Clients) 数据集, WCD (Wholesale Customers Data) 数据集, ad 数据集和 adult 数据集, 分别在 MapReduce 和 MPI 两种编程模型下, 实验分析两种编程模型对数据

挖掘性能的影响. 实验结果如图 7 所示.

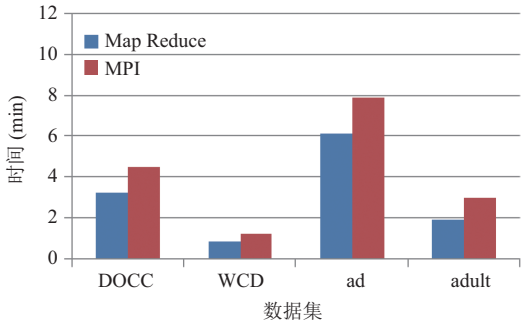

图 7 两种编程模型性能比较

由图中可以看出, 在 4 个不同的 UCI 数据集下, MapReduce 编程模型均比 MPI 编程模型的效率高, 且数据量越大, MapReduce 优势越明显. 因为 MapReduce 隐藏了并简化了并行计算的细节, 还提拱了备份冗余, 本地优化以及负载均衡的机制.

5 结束语

数据的并行与分布式处理逐渐被用于数据挖掘领域. 而 MapReduce 框架大大降低了编程的复杂性, 提高了编程的效率, 可以有效避免 MPI 编程模型的不足之处. 本文基于 MapReduce 与距离, 实现了一种离群数据并行挖掘算法, 提高了离群数据挖掘的效率, 并且使用人工数据集和 UCI 数据集, 实验验证了算法在不同条件下, 参数对性能的影响.

参考文献

1 Knorr EM, Ng RT. Algorithms for mining distance-based outliers in large datasets. Proceedings of the 24th International Conference on Very Large Data Bases. San Francisco, CA, USA. 1998. 392–403.

2 Aggarwal CC. Outlier analysis. Aggarwal CC. Data Mining. Cham: Springer, 2015: 237–263.

3 Hubert M, Rousseeuw PJ, Segaert P. Multivariate functional outlier detection. Statistical Methods & Applications, 2015, 24(2): 177–202.

4 Lu W, Shen YY, Chen S, *et al*. Efficient processing of k nearest neighbor joins using MapReduce. Proceedings of the VLDB Endowment, 2012, 5(10): 1016–1027. [doi: 10.14778/2336664]

5 Zhao HF, Jiang B, Tang J, *et al*. Image matching using a local distribution based outlier detection technique. Neurocomputing, 2015, 148: 611–618. [doi: 10.1016/j.neucom.2014.07.002]

6 Roth V. Kernel fisher discriminants for outlier detection. Neural Computation, 2006, 18(4): 942–960. [doi: 10.1162/neco.2006.18.4.942]

7 Radovanović M, Nanopoulos A, Ivanović M. Reverse nearest neighbors in unsupervised distance-based outlier detection. IEEE Transactions on Knowledge and Data Engineering, 2015, 27(5): 1369 –1382. [doi: 10.1109/TKDE.2014.2365790]

8 Li Y, Nitinawarat S, Veeravalli VV. Universal outlier hypothesis testing. IEEE Transactions on Information Theory, 2014, 60(7): 4066 –4082. [doi: 10.1109/TIT.2014.2317691]

9 Marghny M H, Taloba AI. Outlier detection using improved genetic K-means. International Journal of Computer Applications, 2011, 28(11): 33–36. [doi: 10.5120/ijca]

10 Kriegel HP, Kröger P, Schubert E, *et al*. Outlier detection in arbitrarily oriented subspaces. Proceedings of the 12th International Conference on Data Mining. Brussels, Belgium. 2012. 379–388.

11 Bhattacharya G, Ghosh K, Chowdhury AS. Outlier detection using neighborhood rank difference. Pattern Recognition Letters, 2015, (60-61): 24–31.

12 Breunig MM, Kriegel HP, Ng RT, *et al*. LOF: Identifying density-based local outliers. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. Dallas, TX, USA. 2000. 93–104.

13 Murugavel P, Punithavalli M. Performance evaluation of density-based outlier detection on high dimensional data. International Journal on Computer Science and Engineering, 2013, 5(2): 62–67.

14 Sarawagi S, Agrawal R, Megiddo N. Discovery-driven exploration of OLAP data cubes. In: Schek HJ, Alonso G, Saltor F, *et al*. eds. Advances in database technology — EDBT'98. Berlin Heidelberg: Springer, 1998. 168–182.

15 Ramaswamy S, Rastogi R, Shim K. Efficient algorithms for mining outliers from large data sets. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. New York, NY, USA. 2000. 427–438.

16 Angiulli F, Basta S, Pizzuti C. Distance-based detection and prediction of outliers. IEEE Transactions on Knowledge and Data Engineering, 2006, 18(2): 145 –160. [doi: 10.1109/TKDE.2006.29]

17 Angiulli F, Basta S, Lodi S, *et al*. Distributed strategies for mining outliers in large data sets. IEEE Transactions on Knowledge and Data Engineering, 2013, 25(7): 1520–1532. [doi: 10.1109/TKDE.2012.71]

18 Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation. San Francisco, CA, USA. 2004.

19 周品. Hadoop 云计算实战. 北京: 清华大学出版社, 2012.