第34卷 第12期 计 算 机 工 程 2008年6月
Vol.34 No.12 Computer Engineering June 2008
·博士论文· 文章编号：1000—3428(2008)12—0004—03 文献标识码：A 中图分类号：TP311.13

基于记忆效应的局部异常检测算法

李 健[1,2]，阎保平[1]，李 俊[1]
(1. 中国科学院计算机网络信息中心，北京 100080；2. 中国科学院研究生院，北京 100080)

摘 要： 基于密度的局部异常检测算法(LOF 算法)的时间复杂度较高，限制了其在高维数据集以及大规模数据集中的使用。该文通过分析 LOF 算法，引入记忆效应概念，提出具有记忆效应的局部异常检测算法——MELOF 算法。实验测试表明，该算法的计算结果与 LOF 算法完全相同，而且能够大大缩短运行时间。

关键词： 数据挖掘；异常检测；局部异常因子；记忆效应；MELOF 算法

Memory-effect-based Local Outlier Detection Algorithm

LI Jian[1,2], YAN Bao-ping[1], LI Jun[1]
(1. Computer Network Information Center, Chinese Academy of Sciences, Beijing 100080;
2. Graduate University of Chinese Academy of Sciences, Beijing 100080)

【Abstract】 The computational complexity of algorithm for identifying density-based local outliers (LOF algorithm) is not ideal, which affects its applications in large scale data sets, especially in high dimensional data sets. Under such circumstances, the concept of memory effect is introduced, which lays the foundation for the newly enhanced algorithm called MELOF. Experimental result shows that MELOF algorithm obtains the same result as LOF algorithm, and shortens the execution time obviously.

【Key words】 data mining; outlier detection; Local Outlier Factor(LOF); memory effect; MELOF algorithm

异常检测是数据挖掘领域的基本问题之一，用于发现数据集中与其他数据明显不同的对象。LOF(Local Outlier Factor)算法[1]在异常检测算法中占据着非常重要的地位。但是 LOF 算法的时间复杂度为 $O(n^2)$，虽然可以采用基于树的索引结构将时间复杂度降到 $O(n\log n)$，但是建立索引结构的过程复杂而且耗时，限制了该算法在大规模数据集中的应用。本文分析了 LOF 算法的特点，在邻域查询过程中充分利用邻居对象的信息，引入记忆效应的概念，提出了基于记忆效应的局部异常检测算法——MELOF 算法。

1 LOF 算法

到目前为止，异常检测算法分为基于统计的算法、基于距离的算法、基于密度的算法和基于偏差的算法等[2]。Breunig 提出基于密度的局部异常检测算法，给出了局部异常因子的概念。

邻域查询是基于密度的数据挖掘算法中最基本的概念。对于数据集 D，计算某对象 p 与 D 内所有对象之间的距离并获取其中符合一定条件的对象集合，该过程称为针对对象 p 的一次邻域查询。

假设数据集 D 包含 n 个对象，LOF 算法需要一个参数 *MinPts* 用于指定邻域中对象的最小个数。LOF 算法按以下 3 步进行[3]：

(1)对数据集 D 中每个对象进行邻域查询，计算其 *MinPts*-距离邻域，并将该对象与该邻域中每个对象的距离存入数据库。

(2)利用存储在数据库的计算结果，计算每个对象的局部异常因子。需要扫描数据库 2 次：计算每个对象的局部可达密度；计算每个对象的局部异常因子。

(3)根据用户输入的 *LOF* 阈值，将 *LOF* 值高于该阈值的对象判定为异常。

一些基本概念介绍如下：

定义 1 对象 p 的 *MinPts*-距离邻域，即所有与对象 p 的距离不超过 *MinPts-distance*(p)的对象组成的集合。形式化表示为

$$N_{MinPts\text{-}distance}(p)=\{q\in D\backslash\{p\} \mid d(p,q)\leqslant MinPts\text{-}distance(p)\}$$

简写为 $N_{MinPts}(p)$，其中，*MinPts-distance*(p)表示对象 p 的 *MinPts*-距离。当某对象 o 满足如下条件：

(1)至少存在 *MinPts* 个对象 $s\in D\backslash\{p\}$，使得 $d(p,s)\leqslant d(p,o)$；

(2)至多存在 *MinPts*-1 个对象 $s\in D\backslash\{p\}$，使得 $d(p,s)<d(p,o)$。

则对象 p 与 o 的距离 $d(p,o)$ 记为 *MinPts-distance*(p)，简写为 *MinPts-d*(p)。

定义 2 对象 p 的局部可达密度 $lrd_{MinPts}(p)$表示如下：

$$lrd_{MinPts}(p)=1/\left(\frac{\sum_{s\in N_{MinPts}(p)} reach-dist_{MinPts}(p,s)}{|N_{MinPts}(p)|}\right)$$

其中，$reach\text{-}dist_{MinPts}(p,s)$表示对象 p 相对于 s 的可达距离，定义为 $reach\text{-}dist_{MinPts}(p,s)=\max\{MinPts\text{-}d(s),d(p,s)\}$。

定义 3 对象 p 的局部异常因子，其计算公式为

$$LOF_{MinPts}(p)=\frac{\sum_{s\in N_{MinPts}(p)}\frac{lrd_{MinPts}(s)}{lrd_{MinPts}(p)}}{|N_{MinPts}(p)|}$$

可以看出，LOF 算法的时间复杂度取决于邻域查询操作。

作者简介： 李 健(1979－)，男，博士研究生，主研方向：网络安全，网络管理，数据挖掘；阎保平、李 俊，研究员

收稿日期： 2007-08-17 **E-mail：** lijian@cstnet.cn

因为每次邻域查询的时间复杂度是 $O(n)$，所以全部邻域查询导致 LOF 算法时间复杂度为 $O(n^2)$。虽然可以采用基于树的索引结构将时间复杂度降到 $O(n\log n)$，但是建立索引结构的过程复杂而且耗时，不适用于高维数据。

2 记忆效应

局部异常因子 LOF 用于表征数据集中每个数据对象的异常程度，并且这种异常是局部的，与所求数据对象一定范围内的邻居分布有关。但是 LOF 算法并没有将这一思想应用到计算局部异常因子的实际操作中。在 LOF 算法的邻域查询过程中，对象 p 的邻域查询信息仅仅用于处理当前对象 p，该邻域查询结束后这些信息被彻底放弃。实际上，这些信息对于 $N_{MinPts}(p)$中对象的邻域查询是非常有用的。

MELOF 算法与数据对象的维数无关，适用于任意维数的数据集。本文以二维数据为例，如图 1 所示，以对象 p 为圆心，假设圆 1 半径为 ε，圆 3 半径为 $3\times\varepsilon$。圆 2 是以对象 d 为圆心，半径 $2\times\varepsilon$ 的范围。假设参数 *MinPts* 取值为 4，由图 1 可知，$N_{MinPts}(p)=\{a,b,c,d\}$。按照 LOF 算法，获得 $N_{MinPts}(p)$与相关的距离之后，对象 p 的邻域查询过程结束，应该选取另外一个点进行邻域查询。实际上，对象 a, b, c, d 的邻域查询范围只需要在圆 3 的区域内进行，不必在整个数据集中查询，这可以显著减少计算量。

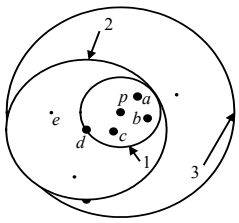

图 1 MELOF 算法示意图

将对象 p 的邻域查询获得的信息用于缩小 $N_{MinPts}(p)$中对象的邻域查询范围，从而减少运行时间，本文称为记忆效应(memory effect)。尽可能利用已知信息来改进随后的操作，这就是记忆效应的精髓。

3 MELOF 算法

与 LOF 算法类似，MELOF 算法也分为 3 步进行，其中步骤(2)、步骤(3)完全相同。MELOF 算法将记忆效应的思想应用在邻域查询过程中，这是与 LOF 算法的主要区别。参考图 1，由于 $N_{MinPts}(p)$中已经至少包含 *MinPts* 个对象，即圆 1 区域已经包含至少 *MinPts*+1 个对象(含对象 p)，因此 $N_{MinPts}(p)$中任一对象 s 在其半径 $2\times$*MinPts-d*(p)的范围内已经包含圆 1 的区域，这就可以保证该范围内至少包含 *MinPts*+1 个对象，而且是距离该对象最近的 *MinPts*+1 个对象。因此，只需在该范围内进行邻域查询就可以确定 $N_{MinPts}(s)$。为了确定适合 $N_{MinPts}(p)$中所有对象的查询范围，考虑边界对象的情况。例如图 1 中，以对象 d 为圆心，$2\times$*MinPts-d*(p)为半径构造出圆 2。由此可以确定出以对象 p 为圆心，$3\times$*MinPts-d*(p)为半径构成的圆 3 就是适合 $N_{MinPts}(p)$中所有对象的邻域查询范围。

与 LOF 算法类似，MELOF 算法也需要参数 *MinPts*，用于指定邻域中对象的最小个数。MELOF 算法中邻域查询伪代码如下：

```
for(int i=0;i< DataSet ->size;i++) {
  p= DataSet ->points[i];
if(p.status==0)        //判断是否已处理
  minPtsNei(DataSet,minNei,minNeiList,neighbour,temp,p,MinPts);
  }
```

其中，*DataSet* 表示整个数据集，函数 *minPtsNei* 的伪代码如下：

```
void minPtsNei(DataSet, minNei, minNeiList, neighbour, temp, p, MinPts){
  pointNeigh(DataSet, minNei, minNeiList, neighbour, p, MinPts);
  arraN(neighbour, temp，minNei[MinPts-1].distance) ;
  min3PtsNei(DataSet,minNei, minNeiList,temp,MinPts);
  return ;
}
```

其中，*pointNeigh* 函数主要完成以下功能：

(1)获取 $N_{MinPts}(p)$和相关的距离值，存放于 *minNeiList*；

(2)将 $N_{MinPts}(p)$中的对象 ID 以及与 p 对象的距离值分别放入 *minNei* 中备用；

(3)将与对象 p 的距离值不大于 $3\times$*MinPts-d*(p)的所有对象都存放到 *neighbour* 队列中。

(4)修改 *point* 对象的 *status* 属性为 1，表明该对象已经处理过了。

pointNeigh 函数无法在最初就确定 *MinPts-d*(p)的精确值，该函数只能将不大于 3 倍当前 *MinPts-d*(p)值的所有对象都送入队列。在 *pointNeigh* 函数完成后，*MinPts-d*(p)值已经确定，因此，将 *neighbour* 队列中的对象重新整理，将其中不符合条件的对象重新去掉，适当缩小范围，这是 *arraN* 函数的功能。

min3PtsNe 函数则是对 $N_{MinPts}(p)$中的对象 s(存放于 *minNei* 中)在 *temp* 队列存放的对象范围内进行邻域查询，同时注意修改对象 s 的 *status* 属性。当然如果对象 s 已经处理过则可以直接跳过，处理 *minNei* 中的下一个对象。

4 实验测试

本节从计算结果和运行效率等方面测试与分析 MELOF 算法的优点。本实验的测试平台为：P4 3.0 GHz CPU, 512MB 内存，操作系统为 Linux，编程语言为 C++。

4.1 计算结果

由第 3 节可知，图 1 中的圆 3 已经包含 $N_{MinPts}(p)$中所有对象的 *MinPts*-距离邻域，因此，2 种算法的计算结果应该完全相同。这一论断在实验中得到了很好的验证。实验分别采用了模拟数据集和 SEQUIOA 2000[4]的真实数据集。实验结果表明，针对相同的参数 *MinPts* 值，MELOF 算法与 LOF 算法使得每个对象都取得完全相同的局部异常因子。

4.2 运行效率

MELOF 算法难以从理论上衡量其时间复杂度，因此，本节采用比较权威的数据集进行测试，从实验的角度进行对比。为了验证 MELOF 算法的性能优势，性能测试的数据集采用的是 SEQUIOA 2000 数据库中的点数据，包含大约 62 500 条记录。本实验使用点数据来对比 MELOF 与 LOF 算法的时间复杂度，分别使用了多组 *MinPts* 参数值。为便于分析，这里只统计步骤(1)的运行时间。实验结果转换为线性图，如图 2 所示，其中算法名称中的数字用于表明 *MinPts* 的值。

由于 LOF 算法中每个对象都要进行一次邻域查询，因此它的运行时间仅与数据集大小有关，与参数值 *MinPts* 无关。这在实验中也得到了证明，因此，图 2 中的 LOF 时间线实际上是不同 *MinPts* 值的多条 LOF 时间线重合，而且不表明 *MinPts* 值。

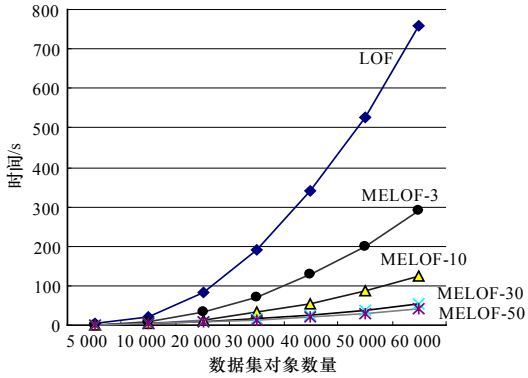

图 2　LOF 与 MELOF 算法运行时间对比

MELOF 算法的运行时间与 *MinPts* 密切相关。由图 2 可以看出，随着 *MinPts* 参数的逐步增大，运行时间逐渐减少。图 2 显示，MELOF 算法的运行效率明显优于 LOF 算法。当 *MinPts* 取值 50 时，对于 60 000 条数据的处理，MELOF 算法只用了 LOF 算法约 5.4%的时间就得到了同样的结果。当 *MinPts* 取值 30 或者 50 时，2 条时间线的差距已经非常细微，表明时间提升的潜力已经比较小了。

按照 MELOF 算法的处理过程，假设处理某对象 p 时，$N_{MinPts}(p)$中所有对象都尚未进行邻域查询，在 $3\times MinPts\text{-}d(p)$ 范围内对 $N_{MinPts}(p)$中所有对象进行邻域查询的时间忽略不计。在这种极端情况下，相当于每 *MinPts*+1 个对象中只有 1 个对象进行 1 次邻域查询。MELOF 算法的运行时间将变为 LOF 算法的 1/(*MinPts*+1)。通过以上分析可以看出，对于特定的参数值 *MinPts*，MELOF 算法存在一个特定的运行时间下限，该下限是实际操作过程中无法达到的。这一结论可以由图 2 的数据得到验证。

5　结束语

LOF 算法在异常检测领域发挥着重要作用，但其时间复杂度不理想，不适用于高维数据集及大规模数据集。本文定义了“记忆效应”的概念，提出 MELOF 算法。该算法大大减少了邻域查询的计算量，同时获得了与 LOF 算法完全相同的效果。通过对模拟数据集和 SEQUIOA 2000 数据集的实验可以看出，数据集规模越大，越能显示出 MELOF 算法在运行性能上的优势，使得运行时间与计算结果都比较理想。今后的研究方向是将记忆效应的思想应用到基于密度的其他数据挖掘算法中，研究如何更有效地确定参数 *MinPts* 的合理值，减少用户输入参数对运行结果的影响。

参考文献

[1] Breunig M, Kriegel H P, Ng R, et al. LOF: Identifying Density-based Local Outliers[C]//Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Dalles, TX, USA: [s. n.], 2000.

[2] Han J, Kamber M. Data Mining: Concepts and Techniques[M]. [S. l.]: Morgan Kaufmann, 2000.

[3] 杨风召, 朱扬勇, 施伯乐. IncLOF: 动态环境下局部异常的增量挖掘算法[J]. 计算机研究与发展, 2004, 41(3): 477-484.

[4] Stonebraker M, Frew J, Gardels K, et al. The Sequoia 2000 Storage Benchmark[C]//Proc. of ACM SIGMOD Int'l Conference on Management of Data. Washington, D. C., USA: [s. n.], 1993.

(上接第 3 页)

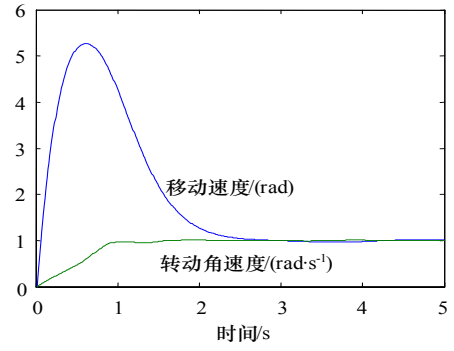

图 4　倒立摆移动速度和转动角速度响应曲线

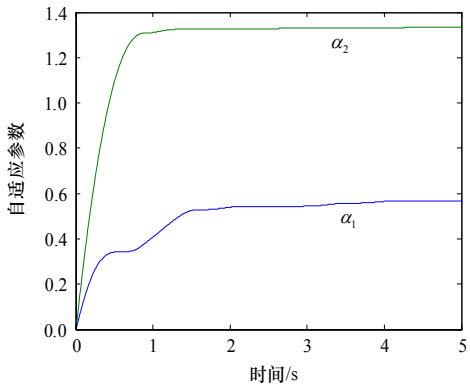

图 5　自适应参数

自适应模糊滑模控制实际上是对不确定边界层厚度的自适应。无需知道系统不确定的边界，就能保证系统的稳定性。

5　结束语

本文研究基于 T-S 模型的不确定非线性系统的跟踪问题。充分利用了 T-S 模型特点，将该系统转换成 3 个组成部分。针对 3 个部分分别进行控制器设计，系统的稳定性分析简单，无需求公共正定矩阵，就能够保证系统稳定性。

参考文献

[1] Tanaka K, Sugeno M. Stability Analysis and Design of Fuzzy Control Systems[J]. Fuzzy Sets and Systems, 1992, 45(1): 135-156.

[2] Tanaka K, Hori T, Wang H. A Multiple Lyapunov Function Approach to Stabilization of Fuzzy Control Systems[J]. IEEE Transactions on Fuzzy Systems, 2003, 11(4): 582-589.

[3] Wang Yuye, Feng Yong, Yu Xinghuo, et al. Terminal Sliding Mode Control of MIMO Linear Systems with Unmatched Uncertainties, Industrial Electronics Society[C]//Proceedings of the 29th Annual Conference on Industrial Electronics Society. [S. l.]: IEEE Press, 2003: 1146-1151.

[4] 高为炳. 变结构控制的理论及设计方法[M]. 北京: 科学出版社, 1996: 119-126.

[5] Tao C W, Taur J S, Chan Meilang. Adaptive Fuzzy Terminal Sliding Mode Controller for Linear Systems with Mismatched Time-varying Uncertainties[J]. IEEE Transactions on Systems, Man and Cybernetics, 2004, 34(1): 255-262.