ICIC Express Letters
Volume **14**, Number **6**, June **2020**

 ISSN 1881-803X
pp. 571–575

A NOTE ON ACCELERATING THE LOCAL OUTLIER FACTOR METHOD ON ONE-DIMENSIONAL DATA

Changmuk Kang

Department of Industrial and Information Systems Engineering
Soongsil University
369 Sangdo-ro, Dongjak-gu, Seoul 06978, Korea
ckang@soongsil.ac.kr

Received December 2019; accepted March 2020

Abstract. *The local outlier factor (LOF) method, which is proposed by Breunig et al. (2000), is one of the most common techniques to detect outliers or abnormal data points in a dataset. It compares the density of a data point with the densities of its k-nearest neighbors. This study presents an algorithm to perform LOF much faster than conventional methods, especially for one-dimensional data. Its worst-case time complexity is only $O(nk)$, and space complexity is $O(n)$. The performance is also computationally compared with the DMwR package, which implements Breunig et al. (2000) in R language.*
Keywords: Local outlier factor, Accelerated algorithm, One-dimensional data

1. **Introduction.** The local outlier factor (LOF) method is one of the most common techniques to detect outliers or abnormal data points in a dataset. It is proposed by Breunig et al. [1] to determine outlierness of a point with its proximity to the neighbors, rather than its relative position in a whole dataset. According to Aggarwal [2], such proximity-based models have three types: cluster-based, distance-based and density-based models. The LOF is a density-based model, which computes density of a data point as the number of points within its local region and compares it with densities of its k-nearest neighbors (k-nn). High LOF scores indicate outliers.

This study presents an idea to accelerate the LOF procedure, especially when the dataset is one-dimensional. Many practical problems, for example, demand forecasting, lead time estimation, and price provision, need to analyze one-dimensional data. Whereas outliers of these datasets are usually detected by extreme values, which are statistical tails of an underlying distribution [2], it may falsely detect outliers if the data is from multiple heterogeneous distributions. An outlier that is far from two distributions, but lies between their mode values cannot be detected. For example, the spare-part demand for an industrial machine has multiple modes depending on failure types and maintenance policies; occasional breakdowns require one or two spare parts while regular preventive maintenance requires a bunch of them. The proposed method accelerates processing the LOF procedure for such kinds of datasets.

The LOF procedure is comprised of two steps: materialization and LOF scoring, and its time complexity is constrained by the materialization step, which computes distances to the k-nearest neighbors of each data point. Where there are n data points, the complexity is $O(n \cdot [\text{time for } k\text{-nn search}])$ [1]. The complexity of the k-nn search is worst-case $O(n)$ for high-dimensional data and on average $O(\log n)$ for medium-dimensional data when using X-tree method [3]. For low-dimensional data, Mouratidis et al. [4] present a constant complexity search method.

DOI: 10.24507/icicel.14.06.571

This study suggests a much faster procedure for k-nn search, which takes k steps in the worst case, and much smaller than k on average. The trick lies in the sortable nature of one-dimensional data. In the sorted list of values, a neighborhood window can be shifted from small to large values. The shifting stops at the value that overlaps between neighborhood windows of two adjacent values, and the worst case occurs when they are totally disjoint. Thus the materialization complexity is worst-case $O(nk)$. This procedure is implemented in R language, and the performance is compared with *DMwR* package, which implements Breunig et al. [1].

2. **Brief Overview of the LOF Procedure.** The LOF score represents a relative local-region density of a data point to its neighbor points. A measure called local reachability density (LRD) is computed for each data point, and a LOF score is defined as a ratio between LRD of a point and the average LRD of its neighborhood points. As the neighborhood LRD is higher than its own, it is more likely to be an outlier.

LRD of a data point is an inverse of the average k-reachability distance of its neighbors. The k-reachability distance of point i from j is defined as Equation (1).

$$k\text{-reachability distance}(i,j) = \max\{k\text{-distance}(j), d(i,j)\} \tag{1}$$

where $d(i,j)$ is distance between i and j, and k-distance(j) is distance between j and its k-th nearest neighbor. Then, LRD of i for k is defined as Equation (2).

$$LRD(i,k) = 1/\left(\Sigma_{j\in N(i,k)} k\text{-reachability distance}(i,j)/\left|N(i,k)\right|\right) \tag{2}$$

where $N(i,k)$ is the set of points no farther than k-nn of i. If more than one point is equally close to i, $N(i,k)$ could have more than k points. A LOF score of i for k is computed as Equation (3).

$$LOF(i,k) = \left(\Sigma_{j\in N(i,k)} LRD(j,k)/\left|N(i,k)\right|\right)/\, LRD(i,k) \tag{3}$$

Therefore, k-nn search is necessary for computing LOF scores.

3. **One-Dimensional LOF Algorithm.** This section describes the proposed algorithm. Note that it accelerates LOF calculation by the properties of a sorted list. Henceforth, it is assumed that a dataset X of n data points is sorted in ascending order, i.e., $x_i < x_j$ if $i < j$ where $x_i, x_j \in X$. Even if X is not sorted, it does not increase the algorithm complexity, because sorting an unordered list takes $O(\log n)$ by the quick sort, and reordering the resulting LOF scores in the original order also takes $O(\log n)$ by the binary search.

The algorithm consists of two steps; k-nn search and LOF scoring. First, k-nearest neighbors and k-distance of each data point are searched. Because k is a fixed parameter, we will not specify k in all the following notations for convenience. On the ground that X is sorted, a neighborhood set of the i-th data point $N(i)$ is a consecutive subsequence including i. Thus, it is defined by $N(i) = [S(i), L(i)]$ where $S(i)$ and $L(i)$ are indices of the smallest and the largest neighbors, respectively. Then, the k-distance $K(i)$ is $\max\{x_i - x_{S(i)}, x_{L(i)} - x_i\}$. The procedures for searching the sets S, L and K are shown in Algorithm 1.

For the first data point, its neighborhood is the next k points, like step 0. The main loop is step 1. For $i > 1$, index $L(i)$ inherits $L(i-1)$, but $S(i)$ is the kth points before $L(i-1)$, like step 1.1. It is because $N(i-1)$ could be more than k points if there are duplicated values in X. The largest index $L(i-1)$ is inherited since $x_{L(i-1)}$ should also be a neighbor of x_i, whereas $x_{S(i-1)}$ may be not. In step 1.2, if distance from the smallest neighbor $x_{S(i)}$ is larger than distance from the smallest non-neighbor $x_{L(i)+1}$, the neighborhood set is shifted by one step. If they are equally distant, $S(i)$ never drops while $L(i)+1$ is included to the neighborhood set. Then the loop stops because there is no closer point after $L(i)$. If $x_i = x_i - 1$, their neighborhood sets are also the same, like step 2. Steps 3 and 4 are for duplicated values; all equally distant boundary values should be neighbors.

Algorithm 1. Given dataset X and value k,

0. Set $i = 1$, $S(1) = 1$, $L(1) = k + 1$, $K(1) = x_{k+1} - x_1$.
1. If $i > 1$ and $x_i > x_{i-1}$,
   1.1 Set $S(i) = \min\{L(i-1) - k, i\}$, $L(i) = \max\{L(i-1), i\}$.
   1.2 While $d_L \leq d_S$ where $d_L = x_{L(i)+1} - x_i$, $d_S = x_i - x_{S(i)}$,
      1.2.1. Set $L(i) = L(i) + 1$.
      1.2.2. If $d_L = d_S$, stop loop and go to 3.
      1.2.3. Else, set $S(i) = S(i) + 1$.
2. Else,
   2.1 Set $S(i) = S(i-1)$, $L(i) = L(i-1)$, $K(i) = K(i-1)$.
   2.2 Go to 6.
3. While $x_{S(i)-1} = x_{S(i)}$,
   3.1 Set $S(i) = S(i) - 1$.
4. While $x_{L(i)+1} = x_{L(i)}$,
   4.1 Set $L(i) = L(i) + 1$.
5. Set $K(i) = \max\{x_i - x_{S(i)}, x_{L(i)} - x_i\}$.
6. Update $i = i + 1$ and go to 2 until $i < n$.

The next step is LOF scoring. First, the LRDs are calculated, and then LOF scores are obtained. For LRD calculation, we must examine whether x_j, which is a neighbor of x_i, also has x_i as its neighbor. If so, k-reachability distance(i, j) is $K(j)$, otherwise, it is $d(i, j)$, as noted in Equation (1). The proposed method uses Lemma 3.1 to shorten this step.

Lemma 3.1. *If x_i is a neighbor of x_j, it is also a neighbor of x_h where $i < h < j$ or $j < h < i$.*

Proof: Let $i < h < j$ (the proof is the same for the case of $j < h < i$). Suppose x_i is not a neighbor of x_h; $i < S(h)$. Where $L(h) \leq L(j)$, the points from $S(h)$ to $L(h)$ are all neighbors of x_j. Because there are at least k points in $[S(h), L(h)]$, point x_i, which is smaller than $x_{S(h)}$, cannot be one of the k-nearest neighbors of x_j. It leads to a contradiction. □

By Lemma 3.1, if x_i is a neighbor of x_j, other points x_h, $i < h < j$ or $j < h < i$, are also neighbors of x_j; no further examination is required. The procedure for LOF scoring is shown in Algorithm 2.

Algorithm 2. Given dataset X, index sets S, L and k-distance set K,

0. Set $i = 1$ and $LRD(i) = 0$ for all i's.
1. Set $p = S(i)$, $q = L(i)$.
2. For $j = S(i), S(i) + 1, \ldots, i - 1$,
   2.1 If $L(j) < i$, update $LRD(i) = LRD(i) + (x_i - x_j)$.
   2.2 Else, update $p = j$ and stop loop.
3. For $j = L(i), L(i) - 1, \ldots, i + 1$,
   3.1 If $S(j) > i$, update $LRD(i) = LRD(i) + (x_j - x_i)$.
   3.2 Else, update $q = j$ and stop loop.
4. Update $LRD(i) = \{L(i) - S(i)\} \big/ \big\{LRD(i) + \Sigma_{j\in[p,q]\setminus\{i\}} K(j)\big\}$.
5. Update $i = i + 1$ and go to 1 until $i < n$.
6. For $i = 1 \ldots, n$, $LOF(i) = \big\{\Sigma_{j\in[S(i),L(i)]\setminus\{i\}} LRD(j)\big\} \big/ LRD(i) \big/ \{L(i) - S(i)\}$.

For each i, variables p and q are starting and ending indices of the mutual neighbors, respectively. In step 2, check whether x_j is a mutual neighbor of x_i from the smallest one,

and if it is, let $p = j$ and stop the loop. Likewise, in step 3, let $q = j$ and stop the loop if x_j is the largest mutual neighbor. The neighbors j from p to q, $K(j)$'s are summed up in step 4. Steps 4 and 6 follow Equations (2) and (3), respectively.

The time complexity of Algorithm 1 is $O(nk)$ for the worst case. The main loop is iterated n times, and step 1 repeats at most k times if adjacent points have totally disjoint neighborhood sets. In the best case, step 1 repeats only once if the neighborhood window never moves, and it repeats constant times regardless of the value of k. Steps 3 and 4 do not repeat if there is no duplicated point in X. Thus, the average complexity of Algorithm 1 is $O(n)$.

The time complexity of Algorithm 2 is $O(nk)$ in every case since the summation of k-distance values and LRD values of k neighbors are required for each data point. Nonetheless, it becomes more efficient by eliminating determination of reachability distance between k-distance and real distance.

The space complexity of the whole procedure is $O(n)$ since all the sets S, L, K, LRD and LOF are the size of n.

4. **Computational Experiment.** The performance of the proposed procedure is computationally compared with the `lofactor` function in the *DMwR* package of the *R* language [5]. This package implements Breunig et al. [1], and faster than other *R* implementations such as *Rlof* and *dbscan* packages.

The experiment is conducted for different numbers of data points. The number n increases from 1,000 to 10,000 by 500. The data points are randomly generated from the uniform distribution between 0 and 1. The parameter k is fixed with 100 in every case. In order to stabilize performance fluctuation by the dataset randomness, 30 different datasets are generated for each n. The proposed procedure (Algorithms 1 and 2) implemented in *R* language and the *DMwR* function are applied to each dataset, and their average of 30 replications are compared.

The result is shown in Figure 1. The *DMwR* exponentially increases computation time for large n's, whereas the proposed procedure linearly increases it. Moreover, the increasing rate is much lower for the proposed one. For $n = 10{,}000$, the proposed procedure takes only 0.45 seconds while *DMwR* takes almost 12 seconds.

5. **Conclusions.** This study proposes a method to accelerate LOF calculation for one-dimensional data. The k-nn search and LOF scoring procedures are much more efficient

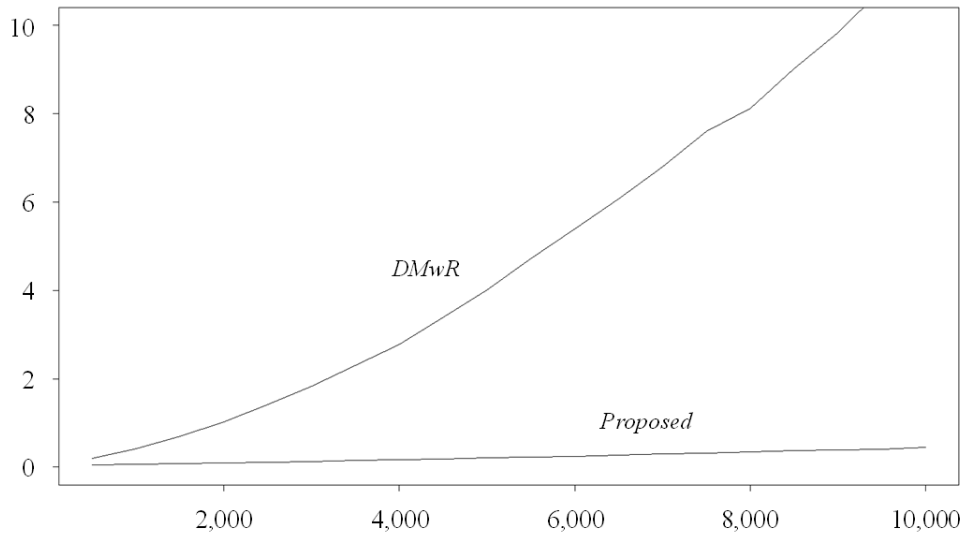

FIGURE 1. Computation time comparison (in seconds)

for a sorted list. The performance is computationally compared with the *DMwR* package in *R*, and the proposed procedure outperforms it. Whereas one-dimensionality is a special case of the outlier detection, many datasets for industrial operations, such as demand, price, processing time, etc, are one-dimensional and can be much more efficiently processed by the proposed method.

Acknowledgment. This work was supported by the Ministry of Education of the Republic of Korea and the National Research Foundation of Korea (NRF-2017R1D1A1B0303217 6).

REFERENCES

[1] M. Breunig, H. Kriegel, R. Ng and J. Sander, LOF: Identifying density-based local outliers, *Proc. of the 2000 ACM SIGMOD International Conference on Management of Data*, Dallas, TX, USA, pp.93-104, 2000.

[2] C. C. Aggarwal, *Outlier Analysis*, Springer, New York, USA, 2015.

[3] S. Berchthold, D. A. Keim and H.-P. Kriegel, The X-tree: An index structure for high-dimensional data, *Proc. of the 22nd International Conference on Very Large Data Bases*, Bombay, India, pp.28-39, 1996.

[4] K. Mouratidis, D. Papadias and M. Hadjieleftheriou, Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring, *Proc. of the 2005 ACM SIGMOD International Conference on Management of Data*, pp.634-645, 2005.

[5] L. Torgo, Functions and data for "Data Mining with R", *R Package Version 2.1*, 2015.