PDOD: TREE BASED AlGORITHM FOR OUTLIER DETECTION

Deepali Aggarwal, Pankaj Singhal , Sharanjit Kaur, Vasudha Bhatnagar
Department of Computer Science, University of Delhi, Delhi, India.
agarwal_deepali209@yahoo.co.in,pankaj_dec21@yahoo.co.in,{skaur,vbhatnagar}@cs.du.ac.in

Abstract

Outlier detection is an important data mining task. It can lead to a more meaningful discovery than the regular KDD tasks and therefore has gained considerable interest in data mining. Earlier works on outlier detection have used distance based, density based, clustering based and tree based approaches. Most of these approaches are non linear in term of dimensions and data size.

In this paper we report development of a two phases outlier detection algorithm. The algorithm is based on the idea of recursive partitioning of data on the most skewed dimension and is integrated with a distance based approach to report outliers. The algorithm is linear with respect to data size and the number of dimensions. The experimental results on synthetic and real data sets show encouraging results.

Keywords: Outlier detection, recursive partitioning, variance ratio, neighborhood

1 Introduction

Outlier detection is one of the important KDD tasks. It is concerned with detection of objects/records which are distinct from most, if not all other objects in data set. Outlier detection has wide area of applications, including credit card industry, insurance industry, information system security, clinical diagnosis etc.. In all these applications, user has knowledge of common patterns in the data and desires to know about any significant deviation from the normal patterns.

Most of the approaches used for outlier detection rely on user input for detecting the number of outliers. This is an unreasonable requirement in most practical situations since the user does not know the actual number of outliers in the data set. To address this problem most of algorithms report top N outliers where N is input by the user.

Traditional statistical approaches are parametric in nature and presume that the data follows a known distribution. Such approaches are not suitable for outlier detection in real life data sets [3]. Nested loop based approaches use a distance function to assess the extent of deviation from the normal pattern [9]. In depth based approaches, each object is assigned a depth and points having minimum depth are reported as outliers [8]. Density based approaches compute outlierability based on local outlier factor for each point [10]. All these approaches do not scale well with large, high dimensional data sets.

The essential idea behind tree based approach is to partition the data set into groups so that all data points in a group are similar to each other. Given such a partitioning, the problem of finding outliers reduces to finding outlying groups and thus the time complexity also reduces because it is now dependent on the number of leaves, which is expected to be much less than the number of objects.

In this paper, we propose a two phase algorithm for finding outliers in high dimensional and large data set using KD tree. The algorithm PDOD (**P**artitioning and **D**istance based **O**utlier **D**etection) recursively partitions the data set along the most skewed dimension in the first phase [5]. The leaves of the tree denote dense and sparse regions in the data space. In the second phase, sparse regions are inspected for outliers using a distance based approach. The salient features of the algorithm are listed below:

1. Uses variance to determine the splitting dimension
2. No need to input number of outliers to be detected
3. Linear time complexity in terms of both, data size and number of dimension

1.1 Related work

Distance based approaches categorize a point as an outlier if at least a user defined fraction of the points in the data set are further away from that point with in given radius [10]. The intuition is that if there are other objects that are close to the candidate in the feature space, then the example is probably not an outlier. If the nearest objects are substantially different, then the example is likely to be an outlier. RBRP [2] and ORCA [4], finds the top n outliers in the data set whose distance to the k^{th} nearest neighbor is the largest using two-phase approach. In the first phase, data set is recursively partitioned into groups termed as bins, where size of each bin is user defined. In second phase an extension of the nested loop algorithm is used to find outliers. EN algorithm [13] uses a simple sampling method to efficiently detect distance-based outliers in domains where distance computation is very expensive.

LOF [10] uses density based approach to mine the outliers. It assigns a outlierability degree termed as local outlying factor to each object indicating how isolated the object is with respect to the surrounding neighborhood. To overcome the limitation of computing *LOF* for all objects, [12] proposes a method which compress data in micro-clusters and constrain the search to top-n outliers only. LOCI [11] is also based upon density approach and can detect outliers and groups of outliers. It proposes an automatic, data-dictated cut-off to determine whether a point is an outlier. Instead of just an outlier-ness score, it provides a whole plot for each point that gives a wealth of information.

KD Tree based algorithm for outlier detection [5], is based on the idea of recursively partitioning the data along the most skewed dimension. The algorithm uses a space decomposition data structure called K-d tree (a binary multidimensional tree) and detects outliers from leaf cells. CD tree algorithm [7], is also based on the concept of partitioning the data space into a set of non-overlapping rectangular cells.

2 PDOD Algorithm

The proposed algorithm works in two phases. In the first phase data set is partitioned into sparse and dense regions. All the objects in a group are expected to behave similarly with respect to being outliers. In the second phase, each point in the sparse region is examined for a threshold neighborhood density (α) within a given radius (r). The points which don't have the minimum desired density are classified as outliers. Thus the large dataset is reduced to only few sparse regions for outlier detection.

2.1 Construction of Tree

Given a data set $D = \{p_1, p_2, p_3, \ldots, p_N\}$ with d dimensional data points. Each attribute A_i in the set $A = \{A_1, A_2, \ldots, A_d\}$ has a numeric domain $Dom(A_i)$. Each data point p is of the form $< v_1, v_2, \ldots, v_d >$, such that $v_i \in Dom(A_i)$. For attribute A_i, let l_i denote the minimum value and h_i denote the maximum value in the domain. The length r_i of region along i^{th} dimension, is given by $(h_i - l_i)$. The complete d-dimensional data space R is defined by $(l_1, h_1) \times (l_2, h_2) \times (l_3, h_3), \ldots, \times (l_d, h_d)$.

Initially, root node r represents the region R and the data set D. At each node, data is partitioned on the basis of a dimension i, also known as *splitting dimension* (Section 2.1.1). After selecting the splitting dimension, a cut point is decided for partitioning the region of the node into two disjoint regions (Section 2.1.2). The algorithm keeps on partitioning the re-

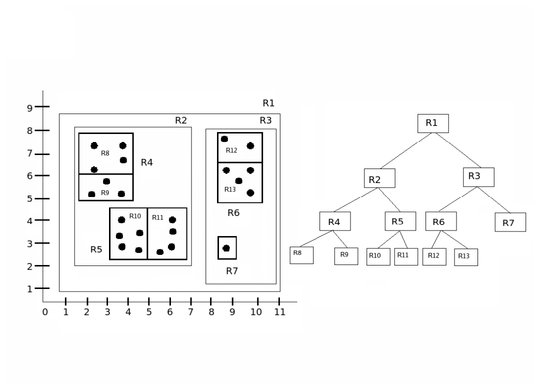

Figure 1: KD tree structure

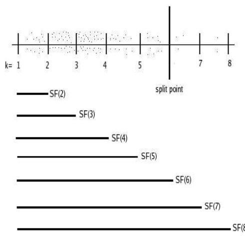

Figure 2: Finding cut point of a splitting dimension

gion R in two regions, R_l and R_r till either (i) the number of points in R is less than a threshold λ or (ii) the volume of hyper rectangle formed is less than hyper sphere of radius r (Section 2.1.3). This procedure results into a KD Tree structure as shown in Figure 1 (for 2D dataset).

2.1.1 Selection of splitting dimension

The splitting dimension for a node(region) is selected on the basis of variance. The variance is computed along all dimensions, and the dimension with maximum variance is selected for splitting the node. Intuitively data points along a dimension with low variance are very similar to each other and hence do not contribute to the outlierability. Therefore dimension with highest variance is selected as the splitting dimension.

2.1.2 Splitting the node

The next task is to choose a suitable cut point in the splitting dimension that partitions the region into two disjoint sub-regions. The cut point represents the partitioning hyperplane which separates the dense region and the sparse region.

To find the cut point, the range of the splitting dimension is divided into n equal sized intervals $[l_i^x, a_{i1}), [a_{i1}, a_{i2}), ..., [a_{in-1}, h_i^x)$. Let the data in node x have the range (l_i^x, h_i^x) along dimension i. We compute the local variance for i^{th} dimension in the j^{th} cumulative interval $(\sigma_{ij}^2(x))$ using all the data points falling up to j^{th} interval along dimension i i.e.

$$\sigma_{ij}^2(x) = variance\ of\ cumulative\ interval\ (l_i^x, a_{ij}^x) \tag{1}$$

Global variance, $\sigma_i^2(x)$ is the variance of all the data point in the node x along dimension i. We compute the ratio $R_{ij}(x)$ of local to global variance as follows:

$$R_{ij}(x) = \frac{\sigma_{ij}^2(x)}{\sigma_i^2(x)} \tag{2}$$

$R_{ij}(x)$ gives a low value when the local variance is small as compared to global variance. This corresponds to the situation when all the data points are densely located on that interval. The end point of the cumulative interval (i.e. a_{ij}) with maximum ratio is selected for partitioning the node x. This point separates sparse and dense regions in x along dimension i (Figure 2).

$$Cutpoint(x, i) = max_{j=1}^{n-1} R_{ij}(x) \quad \forall j \tag{3}$$

The intuition behind splitting the node region using this concept is that densely located interval has low variance over sparsely located interval (Figure 2) and hence need not to be observed for outliers.

2.1.3 Additional Stopping Criteria for splitting of a node

Though the number of points in a node is used as the stopping criterion for node split, size of the region can be used to further optimize the tree construction i.e. to reduce the splitting and the depth of the tree. If the node represents a significantly small region in data space, then it is not useful to split it further.

Let the $v(x)$ be the volume of the $hyper-rectangle$ represented by a node x where all the points fall in (l_i^x, h_i^x) for dimension i.

$$v(x) = \prod_{i=1}^{d} |h_i^x - l_i^x| \tag{4}$$

If r is the user parameter used to compute the neighborhood density of a point (Section 2.2), then $S = r^d$ gives the volume of the hyper sphere in d dimension space. If $v(x) < S$, then the hyper rectangle formed by the node fits into the hyper sphere and there is no need to split the node further.

The complete algorithm for construction of tree is given below where $split(x, p)$ is used to split the region and cutpoint is the function used to find cutpoint.

```
Tree(x, λ)
Input : Node x, density threshold λ
Output: Tree T
1: if points in a node < $\lambda || v(x) < r^d$ then
2:     return
3: end if
4: for $i = 1$ to $d$ do
5:     $temp = \sigma_i^2(x)$
6:     if $temp < maxvar$ then
7:         $maxvar = temp$
8:         $splitdim = i$
9:     end if
10: end for
11: $\psi = Cutpoint(x, splitdim)$ // as per eq 3
12: for all $p$ in $x$ do
13:     if $v_{splitdim} < \psi$ then
14:         $split(x->left, p)$
15:     else
16:         $split(x->right, p)$
17:     end if
18: end for
19: $Tree(x->left, \lambda)$
20: $Tree(x->right, \lambda)$
```

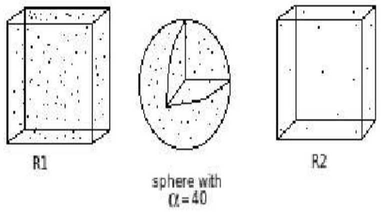

Figure 3: Comparing hyper rectangle and hyper sphere

2.2 Mining outliers from leaf cells

This phase uses sparse regions for outlier detection which are identified by number of points in node. For each leaf cell c, $Sparse(c)$ is computed as follows

$$Sparse(c) = \frac{v(c)}{n_c} \tag{5}$$

where $v(c)$ is the volume (computed as in eq 4) and n_c is number of points of the cell c. Lower sparsity means that the cell is densely packed with data.

Sparsity of the *hyper-sphere* with radius r by

$$Sphere(r, \alpha) = \frac{r^d}{\alpha} \tag{6}$$

If $Sparse(c) \geq Sphere(r, \alpha)$ then the probability of detecting outliers from cell c is more, else it is expected that it will not have outliers. In Figure 3, hyper rectangle R_1 is more densely packed than R_2 whereas both are equal in volume. So it is expected that R_1 cannot contain an outlier but same may not be true for R_2.

Once sparse cells have being identified, their points are examined. For each point in the sparse cell, its neighbors are determined. If a data point has got α neighbors with in the radius r in the same cell, then it is certainly not an outlier. Otherwise adjacent leaf cells are searched for neighbors. As soon as α neighbors are found, search is stopped for that data point. This procedure is repeated for all data points in leaf cell. If a data point x has neighbors less then α

even after searching entire data set then x can safely termed as outlier.

Tree structure helps in fast searching for neighbors because adjacent leaf cells in tree can be considered more similar than others because two adjacent leaf cells must be having same parent or grand parent at some point during the construction of the tree. So adjacent cells must be siblings or cousins and therefore must be similar to each other in some respect.

2.3 Time Complexity

Tree construction takes $O(\rho N d)$ time, where ρ is depth of tree, N is total data size and d is number of dimensions. Time complexity for the second phase is $O(N \times m)$ where $m = N/l$ is the average number of data points per leaf cell and l is the number of leave cells. Hence total time taken by PDOD is $O(\rho N d) + O(N \times m)$, which is linear in terms of both N and d.

3 Experimental Study

We implemented our algorithm using C++ language and compiled using g++ with no optimization. All the experiments were performed with no other user process running on a Linux based system with a 3.03 GHZ Intel Pentium 4 processor and 512 MB of main memory. We compare PDOD with ORCA [4]. We used following data sets for experiments:

1. Synthetic data sets generated by ENCLUS data generator [6] : We generated 10 different data sets for experiment purpose, each having 10 numeric attributes. The number of outliers in each data set are proportional to the numbers of records, so as to make results comparable.

2. KDD Cup 1999 data set : It has 4,94,021 tuples each of 42 dimensions belonging to any of 23 classes. Out of 42 dimensions, only 10 commonly used numeric dimensions are selected. Class 0 represent the normal class and rest are attack classes. For experimentation purpose, we created 22 files corresponding to each attack class. Each file has 97,278 tuples of class 0 and at most 10 tuples from one attack class.

Table 1: Execution time over synthetic data

Data size in lacs	Total outliers	Time taken by PDOD(in secs)	Time taken by ORCA(in secs)
1	100	40	335
2	200	85	1098
3	300	130	2282
4	400	165	3886
5	500	210	5912
6	600	250	8358
7	700	300	11225
8	800	345	14513
9	900	390	18222
10	1000	440	22352

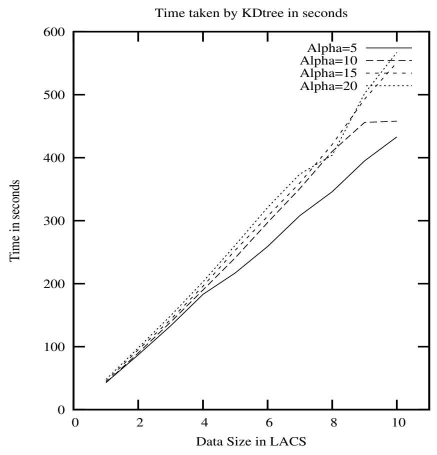

Figure 4: Scalability with data size

3.1 Scalability with data size

We tested the scalability of PDOD algorithm using ENCLUS data in terms of execution time and results are compared with ORCA. The algorithm is repeatedly executed with $\lambda = .5$ and $r = .25$, for different values of neighborhood threshold α. We find that for all values of α, the time varies linearly with increase in data size. Figure 4 shows the linear relationship between time to mine outliers and data size. Table 1 shows that PDOD takes linear time but ORCA has quadratic time complexity (theoretically $O(n^2)$).

Table 2: Detection rate for KDD Cup Data

Attack type	Accuracy by ORCA(%)	Accuracy by by PDOD(%)
smurf	10	100
portsweep	0	40
nmap	10	90
back	0	0
bufferoverflow	0	0
multihop	0	0
neptune	30	100
land	0	100
spy	0	100
warezclient	0	0
guesspasswd	20	40
rootkit	0	10
ipsweep	0	0
teardrop	0	0
ftpwrite	0	25
loadmodule	0	45
pod	0	0
satan	10	70
warezmaster	0	0
imap	0	90
perl	0	0
phf	0	0

3.2 Quality comparison

Both the algorithms PDOD and ORCA detected all the actual outliers and gave 100% accuracy for synthetic data. We used KDD cup dataset to compare the quality of output of ORCA and PDOD. Although PDOD is not able to detect certain attack types, it manages to detect the most of the attack types and has performed better than ORCA in detecting attacks Table 2.

Acknowledgment: We are thankful to Naveen Kumar (DU) and Sona jhariaminz (JNU), their comments on our work and constant encouragement.

4 CONCLUSION

In this paper we have proposed PDOD algorithm for outlier detection. This two phase algorithm integrates partitioning approach and distance base approach. The Objective is to integrate all the positive points of both approaches while minimizing the drawbacks of each approach. We have done experiments with real and synthetic datasets. Results reveal that tree partitioning approach with distance approach detect outliers efficiently.

References

[1] D. Aggarwal and P. Singhal. *Tree based Outlier Detection.* University of Delhi, New Delhi, 2006.

[2] S. P. Amol Ghoting and M. E. Otey. Fast mining of distance-based outliers in high dimensional datasets. In *Proc. SDM*, 2006.

[3] V. Barnett and T. Lewis. *Outliers in Statistical Data.* John Wiley, 1994.

[4] S. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proc. ACM*, 2003.

[5] A. Chaudhary, A. S. Szalay, and A. W. Moore. Very fast outlier detection in large multidimensional data sets. In *Proc. DMKD*, 2002.

[6] A. W.-C. F. Chung Heng Cheng and Y. Zhang. http://www.cse.cuhk.edu.hk/ kdd/clustering/enclus.html, 1999.

[7] Y. B. Huanliang Sun and et. al. Cd-tree: An efficient index structure for outlier detection. In *Proc. WAIM*, 2004.

[8] T. Johnson, I. Kwok, and R. T. Ng. Fast computation of 2-dimensional depth contours. In *Proc. Knowledge Discovery and Data Mining*, pages 224–228, 1998.

[9] K. E. M. *Outliers and Data Mining: Finding Exceptions in Data.* PhD thesis, Dept. of Computer science, University of British Columbia, 2002.

[10] H.-P. K. Markus M. Breunig and et. al. Lof: Identifying density-based local outliers. In *Proc. ACM SIGMOD*, pages 93–104, 2000.

[11] P. G. S. Papadimitriou, H. Kitagawa and C. Faloutsos. Loci: Fast outlier detection using the local correlation integral. In *Proc. 19th ICDE*, pages 351–326, 2003.

[12] A. K. H. T. Wen Jin and J. Han. Mining top-n local outliers in large databases. In *Proc. KDD*. ACM, 2001.

[13] M. Wu and C. Jermaine. Outlier detection by sampling with accuracy guarantees. In *Proc. KDD*. ACM, 2006.