Data Min Knowl Disc (2008) 16:349–364
DOI 10.1007/s10618-008-0093-2

Fast mining of distance-based outliers in high-dimensional datasets

Amol Ghoting · Srinivasan Parthasarathy · Matthew Eric Otey

Received: 29 January 2007 / Accepted: 14 February 2008 / Published online: 4 March 2008
Springer Science+Business Media, LLC 2008

Abstract Defining outliers by their distance to neighboring data points has been shown to be an effective non-parametric approach to outlier detection. In recent years, many research efforts have looked at developing fast distance-based outlier detection algorithms. Several of the existing distance-based outlier detection algorithms report log-linear time performance as a function of the number of data points on many real low-dimensional datasets. However, these algorithms are unable to deliver the same level of performance on high-dimensional datasets, since their scaling behavior is exponential in the number of dimensions. In this paper, we present RBRP, a fast algorithm for mining distance-based outliers, particularly targeted at high-dimensional datasets. RBRP scales log-linearly as a function of the number of data points and linearly as a function of the number of dimensions. Our empirical evaluation demonstrates that we outperform the state-of-the-art algorithm, often by an order of magnitude.

Keywords Outlier detection · High-dimensional datasets · Approximate k-nearest neighbors · Clustering

Responsible editor: Thorsten Joachims.

A. Ghoting (✉)
IBM T. J. Watson Research Center, 1101 Kitchawan Road, Yorktown Heights, NY 10598, USA
e-mail: aghoting@us.ibm.com

S. Parthasarathy
The Ohio State University, Columbus, OH, USA
e-mail: srini@cse.ohio-state.edu

M. E. Otey
Google, Inc., Pittsburgh, PA, USA
e-mail: otey@google.com

1 Introduction

A common problem in data mining is that of automatically finding outliers or anomalies in a dataset. Outliers are those points that are highly unlikely to occur given a model of the data. Since outliers and anomalies are rare, they can be indicative of bad data, faulty collection, or malicious content. Recently, researchers have applied outlier detection to tasks such as data cleaning (Gamberger et al. 1999), fraud detection (Bolton and Hand 2002), and intrusion detection (Sequeira and Zaki 2002).

There are several approaches to outlier detection. One approach is that of model-based outlier detection, where the data is assumed to follow a parametric (typically univariate) distribution (Barnett and Lewis 1994). Such approaches do not work well in even moderately high-dimensional spaces and finding the right model is often a difficult task in its own right. To overcome these limitations, researchers have turned to various non-parametric approaches that use a point's distance to its nearest neighbor as a measure of unusualness (Angiulli and Pizzuti 2002, Knorr and Ng 1999, Ramaswamy et al. 2000). The following, among others (Bay and Schwabacher 2003), is a popular definition of distance-based outliers:

- Outliers are the top n data points whose distance to their kth nearest neighbor is greatest (Ramaswamy et al. 2000).

While distance-based outlier detection has proved to be useful, the process continues to be time consuming. The nested loop (NL) algorithm for mining distance-based outliers (Knorr and Ng 1999) typically requires $O(N^2)$ time, where N is the numbers of data points. To overcome this problem, in the past few years, researchers have proposed several solutions ranging from the use of spatial index structures (KD-trees (Bentley 1975), R-trees (Guttmann 1984), or X-trees (Berchtold et al. 1996)) for fast *nearest neighbor* computation to partitioning the feature space with clustering (Ramaswamy et al. 2000). Unfortunately, these approaches do not scale well with the number of dimensions (Bay and Schwabacher 2003, Knorr and Ng 1999). Consequently, for high-dimensional datasets, solutions based on the simple NL algorithm are known to provide the best performance (Knorr and Ng 1999, Bay and Schwabacher 2003).

Bay and Schwabacher (2003), and have shown that coupled with data randomization and a simple pruning rule, the NL algorithm provides the best known performance on large, high-dimensional datasets. While the worst case complexity of their variant of the NL algorithm continues to be $O(N^2)$, their approach achieves a complexity that is sub-quadratic (but not log-linear) in the number of data points. Although their strategies do result in a significant reduction in computation, the process is still very time consuming.

In this paper, we further improve the scaling behavior of distance-based outlier detection on large, high-dimensional datasets. Specifically, we make the following contributions:

- We study the conditions under which the state-of-the-art distance-based outlier detection algorithm, ORCA (Bay and Schwabacher 2003), is unable to provide near-linear time performance.

- We present RBRP, a two-phase algorithm for fast mining of distance-based outliers. In the first phase, the dataset is pre-processed into bins such that points that are close to each other in space are likely to be placed in the same bin. In the second phase, an extension of the NL algorithm that operates over bins is used for determination of outliers. The pre-processing performed in the first phase facilitates fast convergence to a point's *approximate nearest neighbors*. As we shall see, only a point's *approximate nearest neighbors*, and not its *nearest neighbors*, are needed for fast distance-based outlier detection.
- We demonstrate that our algorithm scales well to high-dimensional datasets with millions of data points, and outperforms the state-of-the-art distance-based outlier detection algorithm, often by over an order of magnitude. Empirically as well as analytically, we show that the algorithm scales log-linearly as a function of the number of data points.

The rest of this article is organized as follows. In Sect. 2, we will describe the state-of-the-art in distance-based outlier detection. In Sect. 3, we first present the shortcomings of the state-of-the-art, and then RBRP, our two-phase outlier detection algorithm. We evaluate the performance of our algorithm in Sect. 4. Finally, we conclude in Sect. 5.

2 Distance-based outlier detection

In distance-based outlier detection, one typically looks at the local neighborhood of a data point to find its nearest neighbors. If the nearest neighbors are relatively close, then the data point is considered to be normal, otherwise it is considered to be an outlier. The key benefit of defining outliers based on their neighborhood is that no parametric distribution needs to be defined to measure unusualness. The following are three popular definitions of distance-based outliers:

- Outliers are the data points for which there are fewer than p other data points within distance d (Knorr and Ng 1999).
- Outliers are the top n data points whose distance to their kth nearest neighbor is greatest (Ramaswamy et al. 2000).
- Outliers are the top n data points whose average distance to their k nearest neighbors is greatest (Angiulli and Pizzuti 2002).

While there are several minor differences between these definitions, all these definitions use the nearest neighbor density estimate to determine if a point is an outlier. Researchers have developed a variety of approaches to find these outliers efficiently. Some approaches make use of KD-trees (Bentley 1975), R-trees (Guttmann 1984), or X-trees (Berchtold et al. 1996) to find the nearest neighbors of each candidate point. These index structures are extremely efficient in finding the k nearest neighbors of a data point. Outlier detection algorithms that make use of these index structures can potentially scale as $O(N \log N)$, if one can find the k nearest neighbors of a data point in $O(\log N)$ time. However, the time required to search through these index structures scales exponentially with the number of dimensions. Consequently, their usefulness is constrained to low-dimensional spaces. Similarly, researchers have proposed partitioning the space into regions (Knorr and Ng 1999, Ramaswamy et al. 2000), which allows for fast determination of nearest neighbors. Unfortunately, the aforementioned approaches are affected by the curse of dimensionality and do not scale to high-dimensional data (Bay and Schwabacher 2003).

The simple nested loop (NL) algorithm is known to give the best performance in high-dimensional spaces (Knorr and Ng 1999). The algorithm, in its simplest form, is presented in Algorithm 1. The main idea in the NL algorithm is that for each data point in D (step 3), we scan the dataset (step 5) and keep track of its k closest neighbors. We also maintain a cutoff threshold, c, that is the distance between the least outlying point discovered thus far and its kth closest neighbor (step 14). When a data point's kth closest neighbor has a distance that is less than the cutoff threshold, c, the data point is no longer an outlier, and we can proceed with the next data point (step 9). As we process more data points, the algorithm finds more extreme outliers, and the cutoff increases giving us improved pruning efficiency (step 14).

The state-of-the-art distance-based outlier detection algorithm, ORCA (Bay and Schwabacher 2003), uses the NL algorithm with a pre-processed dataset. ORCA randomizes the dataset (D) in linear time with constant amount of memory using a disk-based shuffling algorithm. This randomization allows the NL algorithm to process non-outlier points, which are the large majority, relatively quickly. The authors report sub-quadratic time performance in the number of data points (often well below quadratic but not log-linear) on several real and synthetic datasets.

3 Outlier detection algorithm

We will first characterize the shortcomings of ORCA and then present our outlier detection algorithm.

3.1 Shortcomings of ORCA

For expository simplicity, let us assume that we are interested in finding the top n data points whose distance to their nearest neighbor is the greatest. Let us examine the number of distance computations that are required to process a data point (say x) that is not an outlier. One can think of this problem as a set of independent Bernoulli trials where one keeps drawing instances until one has a single success (one data point within the cutoff threshold). Let $\Pi(x)$ be the probability that a randomly selected data point lies within the cutoff threshold. Let Y be a random variable representing the number of trials required until we have a single success. The probability of obtaining a success on trial y, $P(Y = y)$, is given by:

$$P(Y = y) = \Pi(x) \times (1 - \Pi(x))^{(y-1)} \quad (1)$$

Therefore, the expected number of distance computations for the data point (x) that is not an outlier is given by:

Algorithm 1 The simple nested loop

Procedure: Find outliers

```
Require: $k$, the number of nearest neighbors; $n$, the number of outliers to be returned; $D$, the set of data points.
Ensure: $O$, the set of outliers.
1: $c = 0$ ($c$ is the cutoff threshold)
2: $O = \{\}$
3: for each $d$ in $D$ do
4:   Neighbors($d$) = {}
5:   for each $b$ in $D$ such that $b \neq d$ do
6:     if |Neighbors($d$)| $< k$ or Distance($b$, $d$) $<$ Maxdist($d$, Neighbors($d$)) then
7:       Neighbors($d$)=Closest($d$, Neighbors($d$) $\cup$ $b$, $k$)
8:     end if
9:     if |Neighbors($d$)| $\geq k$ and $c >$ Distance($b$, $d$) then
10:      break
11:    end if
12:  end for
13:  $O$ = TopOutliers($O \cup b$, $n$)
14:  $c$ = MaxThreshold($O$)
15: end for
```

Note:

Maxdist(d, S) returns the maximum distance between d and an element in set S

Closest(d, S, k) returns the k nearest elements in S to d

TopOutlier(S, n) returns the top n outliers in S based on the distance to their kth nearest neighbor

MaxThreshold(S) returns the distance between the weakest outlier in S and its kth nearest neighbor

$$E[Y] = \sum_{y=1}^{N} P(Y = y) \times y = \frac{1}{\Pi(x)} \tag{2}$$

In order to achieve near-linear time scaling behavior, $E[Y]$, and hence $\Pi(x)$, must be a constant. This is the central premise behind ORCA's near-linear time performance. However, as we shall see next, this does not always hold.

Again, for expository simplicity, let us assume that we have N uniformly distributed data points in an area of size $\sqrt{N} \times \sqrt{N}$. We seek to answer the following question: *If we randomly pick a point x in this area, what is the expected value of the cutoff threshold, c, such that* $\Pi(x)$ *will be constant?* Intuitively, for $\Pi(x)$ to be constant, the area of the circle with radius $= c$ and center $= x$, πc^2, should scale as $O(N)$. In other words, c should scale as $O(\sqrt{N})$. Thus, for the aforementioned dataset, in order for ORCA to maintain near-linear time performance, as we increase the size of the dataset (N), the cutoff threshold must also increase as $\sqrt{N}$. This indicates that as the dataset size increases, in order to maintain near-linear time performance, distance to the nearest neighbor for the true outliers must also increase. Obviously, this will not always hold true.

In summary, ORCA delivers near-linear scaling behavior only when the cutoff distance can quickly converge to a large value such that $\Pi(x)$ is a constant. This can occur only when the dataset has a large number of outlying points. When the dataset consists of a mixture of a few distributions, with not many outlying points, ORCA's complexity is near quadratic (Bay and Schwabacher 2003).

3.2 Algorithm RBRP (recursive binning and re-projection)

As pointed out in Sect. 2, in order to find distance-based outliers using the NL algorithm, one needs to find k data points that are within the cutoff threshold, c. We call these k data points *approximate nearest neighbors*. The key to fast outlier detection is to efficiently find the k approximate nearest neighbors of a data point. This goal is different from most existing approaches that attempt to find the k nearest neighbors efficiently, which is more expensive. There are several existing efforts (Arya et al. 1998, Indyk and Motwani 1998, Kleinberg 1997, Kushilevitz et al. 1998) that facilitate searching for approximate nearest neighbors of a data point. Unfortunately, to process a single lookup query, these algorithms need time that scales either exponentially in the dimensionality of the data or linearly in the size of the data. This is not acceptable for processing large, high-dimensional datasets.

We now present RBRP (Recursive Binning and Re-Projection), a *two-phase* algorithm for fast mining of distance-based outliers in high dimensional datasets. RBRP, *like* ORCA, *finds the top n outliers in the dataset whose distance to their* kth *nearest neighbor is the greatest.*

3.2.1 Phase 1

The goal of the first phase of RBRP is to partition the dataset into bins such that points that are close to each other in space are likely to be assigned to the same bin. One natural candidate to generate such bins is to cluster the data using an algorithm such as K-means (Hartigan 1975) to find a large number of small clusters. Each of the clusters can constitute a bin. However, this process requires us to specify the number of clusters, and does not guarantee equal-frequency binning, making it ineffective for our uses. Another possibility is to use a clustering algorithm such as BIRCH (Zhang et al. 1996). This is in some senses similar to the approach proposed by Ramaswamy et al. (2000). However, this approach will not scale to high-dimensional data. Related to our goal is the problem of building equi-depth multidimensional histograms (Muralikrishna and DeWitt 1988, Jagadish et al. 1998). The bucket boundaries for such histograms can be used to partition the dataset into bins with nearly equal number of points. Such an approach, however, will not necessarily result in bins that are dense (or preserve locality), making it ineffective for our uses.

Our approach to partitioning the dataset into bins is shown in Algorithm 2. It is a recursive procedure similar to divisive hierarchical clustering. At each stage in the recursion, we iteratively partition the data into k partitions. This iterative partitioning is akin to the partitioning step employed in the k-means (Hartigan 1975) algorithm. Essentially, we start with k random centers (step 1), and assign each point to its closest center, creating k partitions (steps 5–8). Next, we find k centers for these k partitions (step 10), and continue iteratively for a fixed number of iterations. Once we have finished with these iterations, for each of these partitions, we proceed recursively if

Algorithm 2 RBRP phase 1

```
Procedure: Bin

Require: $Binsize$, the maximum size of a bin; $k$, the number of partitions; $it$, no. of iterations; $D$, data
    points to be binned.
Ensure: $B$, the set of bins.
1:  $c = \{c_1, c_2, \ldots, c_k\}$ (the set of $k$ random centers)
2:  $p = \{p_1, p_2, \ldots, p_k\}$ (the set of $k$ partitions of $D$)
3:  for $it$ iterations do
4:    Empty all $k$ partitions in $p$
5:    for each $d$ in $D$ do
6:      $j$ = Closest($c$, $d$)
7:      Insert($d$, $j$)
8:    end for
9:    $c = \{\}$
10:   RecomputeCenters($c$, $p$)
11: end for
12: for each $p_i$ in $p$ do
13:   if size of $p_i > Binsize$ then
14:     Bin($Binsize, k, it, p_i$)
15:   else
16:     Reorganize data points in $p_i$, ordered as per their projection along the principal component of $p_i$
17:     Add $p_i$ to $B$
18:   end if
19: end for
```

Note:

Closest(c, d) returns the index of the nearest elements in c to d

Insert(d, j) inserts point d in jth partition in p

RecomputeCenters(c, p) inserts k centers of partitions in p into c

the size of the partition is greater than a user-defined threshold ($Binsize$) (steps 13–14). Such a binning strategy ensures that points that are close to each other in space are likely to be collocated in the same bin. In Phase 2 of RBRP, we will sequentially scan through each bin to find the approximate nearest neighbors of a data point. To facilitate fast convergence to the approximate nearest neighbors during a sequential scan, we reorganize the data points in each bin as per their order in the projection along the principal component (Jolliffe 1986) of the points in the bin (step 16). The principal component of a bin represents the axis of maximal variance. Such a reorganization within bins allows for fast convergence to approximate nearest neighbors when sequentially scanning through a bin. This is because we expect to find the approximate nearest neighbors of a data point in its neighborhood when data points are ordered as per their projection along the principal component. We also note that the task of finding the principal component scales as $O(N \times d^2)$. The process in depicted in Fig. 1.

3.2.2 *Complexity analysis for phase 1*

Assuming a two-way partitioning at each step in the recursion, the recurrence relation for Phase 1 is given by:

$$T(N) = T(N - m) + T(m) + \theta(N) \tag{3}$$

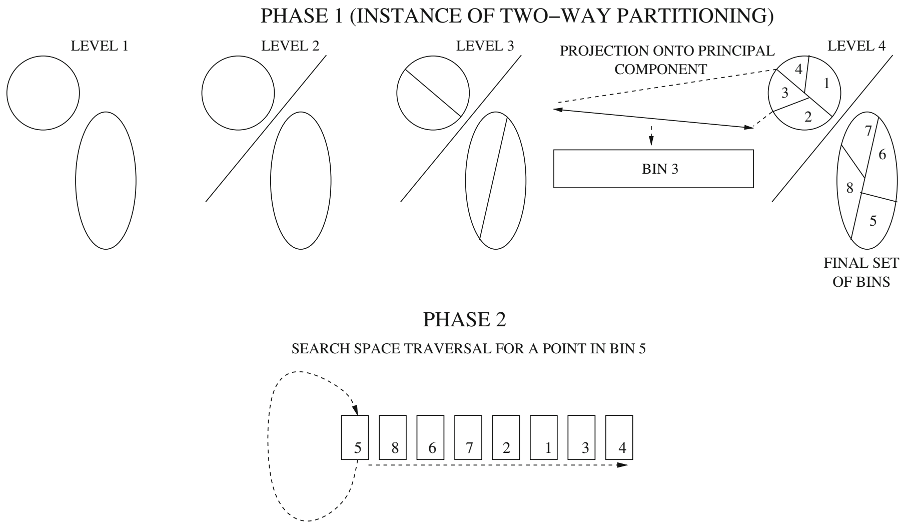

PHASE 1 (INSTANCE OF TWO–WAY PARTITIONING)
LEVEL 1
LEVEL 2
LEVEL 3
PROJECTION ONTO PRINCIPAL COMPONENT
LEVEL 4
BIN 3
FINAL SET OF BINS
PHASE 2
SEARCH SPACE TRAVERSAL FOR A POINT IN BIN 5
5 8 6 7 2 1 3 4

Fig. 1 Phase 1 and Phase 2 of RBRP

If the partitioning step is extremely unbalanced, then m is likely to be a constant (say $m = 1$). In this case:

$$\begin{aligned}
T(N) &= T(N-1) + T(1) + N \\
T(N) &= T(N-2) + 2T(1) + 2N \\
T(N) &= N \times N = \theta(N^2)
\end{aligned}$$

This results in the worst case complexity of $O(N^2)$ for phase 1. On the other hand, if the partitioning step is highly balanced, then m is likely to be $N/2$ on average. In this case:

$$\begin{aligned}
T(N) &= T(N - N/2) + T(N/2) + N \\
T(N) &= 2T(N/2) + N \\
T(N) &= 4T(N/4) + 2N \\
T(N) &= 8T(N/8) + 3N \\
T(N) &= N \log N
\end{aligned}$$

This results in the best case complexity of $O(N \log N)$ for phase 1. The exact average case complexity for phase 1 is obtained by solving the recurrence relation (assuming all potential partitions are equally likely):

$$T(N) = \sum_{m=1}^{N} \frac{1}{N}(T(m) + T(N-m)) + N$$

$$T(N) = \frac{2}{N}\sum_{m=1}^{N-1} T(m) + \frac{T(N)}{N} + N$$
$$NT(N) = 2\sum_{m=1}^{N-1} T(m) + T(N) + N^2$$

$$NT(N) - (N-1)T(N-1) = T(N-1) + T(N) + N^2 - (N-1)^2$$
$$(N-1)T(N) = NT(N-1) + 2N - 1$$
$$T(N)/N = T(N-1)/(N-1) + \frac{2N-1}{N(N-1)}$$
$$Let\ A_n = T(N)/N$$
$$A_n = A_{n-1} + \frac{2N-1}{N(N-1)}$$
$$A_n \approx \sum_{i=1}^{N} 1/i \approx \log N$$
$$\text{Therefore, } T(N) \approx N \log N$$

Therefore, the average case complexity for phase 1 is same as its best case complexity. When employing a k-way partitioning at each step in the recursion, the above results will continue to hold.

3.2.3 Phase 2

In Phase 2 (described in Algorithm 3), we use an extension of the NL algorithm to find outliers in the dataset that has been organized into bins. For each data point, we start searching for approximate nearest neighbors beginning at the next consecutive location in the bin (steps 7–14). Once the end of the bin has been reached, we wrap around to the start of the bin, and continue searching in the remainder of the bin. If the entire bin has been searched and k approximate nearest neighbors have not been discovered within this bin, we switch to the next closest bin (step 6), and continue searching for approximate nearest neighbors. This search continues iteratively until k approximate nearest neighbors are discovered. This process is depicted in Fig. 1.

3.2.4 Complexity analysis for Phase 2

The worst case time complexity of Phase 2 is $O(N^2)$. We expect such performance when the data set does not have many true outliers (for example, consider a dataset in which the distance to the kth nearest neighbor is the same for all data points). However, for datasets with true outliers, we expect the cutoff threshold to increase to a point that we can find the approximate nearest neighbors of a normal point in the same or adjacent bin (looking at only a constant number of data points). For outliers, we need to scan all of the bins, but this is expected to be a rare event, as number of desired outliers (n) is much smaller that the dataset size (N). Therefore, we expect Phase 2 to scale as $O(N \times d)$. As Phase 1 scales as $O(N \log N \times d)$, we expect RBRP to scale as $O(N \log N \times d)$.

Algorithm 3 RBRP phase 2

```
Procedure: Find outliers

Require: $k$, the number of nearest neighbors; $n$, the number of outliers to be returned; $D$, the set of data points.
Ensure: $O$, the set of outliers.
1: $c = 0$ ($c$ is the cutoff threshold)
2: $O = \{\}$
3: for each bin $b$ in $B$ do
4:   for each $d$ in $b$ do
5:     Neighbors($d$) = $\{\}$
6:     for each $t$ in $B$, ordered by increasing distance to $b$ do
7:       for each $p$ in $t$ such that $p \neq d$ do
8:         if $|$Neighbors($d$)$| < k$ or Distance($d$, $p$) $<$ Maxdist($d$, Neighbors($d$)) then
9:           Neighbors($d$)=Closest($d$, Neighbors($d$) $\cup$ $p$, $k$)
10:        end if
11:        if $|$Neighbors($d$)$| \geq k$ and $c >$ Distance($p$, $d$) then
12:          break
13:        end if
14:      end for
15:    end for
16:  end for
17:  $O$ = TopOutliers($O \cup b$, $n$)
18:  $c$ = MaxThreshold($O$)
19: end for
Note:
Maxdist($d$, $S$) returns the maximum distance between $d$ and an element in set $S$
Closest($d$, $S$, $k$) returns the $k$ nearest elements in $S$ to $d$
TopOutlier($S$, $n$) returns the top $n$ outliers in $S$ based on the distance to their $k$th nearest neighbor
MaxThreshold($S$) returns the distance between the weakest outlier in $S$ and its $k$th nearest neighbor
```

At this juncture, we would like to point out that RBRP *will always discover the exact same set of outliers as* ORCA. The key difference between RBRP and ORCA is that when processing normal points, RBRP will discover the k approximate nearest neighbors in far less time than ORCA. For outliers, both ORCA and RBRP will need to scan the entire dataset.

4 Experimental results

4.1 Setup

We evaluate our algorithm's performance on a Linux-based system with a 2.4 GHz Intel Pentium 4 processor and 1 GB of main memory. We report the wall clock time in order to capture both CPU and I/O time. All of the algorithms were implemented

Table 1 Datasets

Data set	Continuous attributes	No. of points
Corel histogram	32	68,040
Covertype	55	581, 012
KDDCup 1999	24	4,898,430
Mixed 30D	30	2,000,000
Uniform 30D	30	1,000,000
IPUMS	128	2,000,000

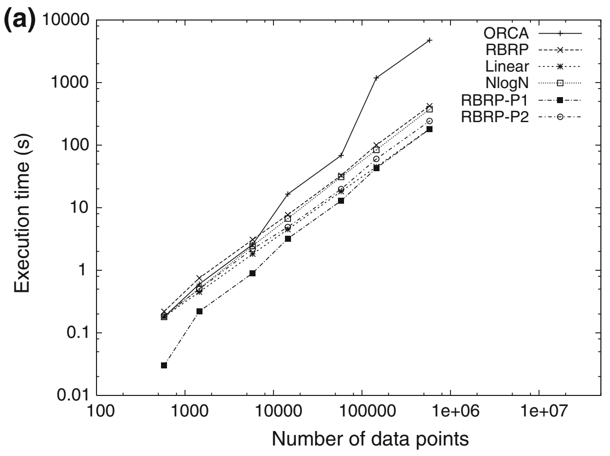
using C. We use several real and synthetic data sets for our analysis. These data sets are summarized in Table 1. They span a range of problems and have different types of features.

- *Covertype*—This data set represents the type of forest coverings for 30 × 30 meter cells in the Rocky Mountain region. For each cell, the data contains the cover type, which is the dominant tree species, and additional attributes such as elevation, slope, and soil type.
- *Corel Histogram*—Each point in this data set encodes the color histogram of an image in a collection of photographs. The histogram has 32 bins corresponding to eight levels of hue and four levels of saturation.
- *IPUMS*—This data set contains the responses from the 1990 Census in the United States. It contains a variety of geographic, economic, and demographic information about individuals.
- *KDDCup 1999*—This data set contains a set of records that represent connections to a military computer network where there have been multiple intrusions by unauthorized users. The raw TCP data from the network has been processed into features such as the connection duration, protocol type, number of failed logins, and so forth.
- *Mixed 30D*—This is a synthetic data set generated from a mixture of 30-dimensional normal and uniform distributions centered on the origin. The normal distribution is centered on the origin with a covariance matrix equal to the identity matrix. This data set contains one million data points from each of the distributions.
- *Uniform 30D*—This is a synthetic data set generated from a 30 dimensional uniform distribution centered on the origin and in the range $[-1,1]$.

We obtained the Covertype, Corel Histogram, and KDDCup 1999 data sets from the UCI KDD Archive (Bay 1999). The IPUMS data set was obtained from the IPUMS repository (Ruggles and Sobek 1997).

4.2 Scalability with increasing data set size

Figures 2–4 show the total execution time to mine outliers on the six data sets as we vary the number of data points. Here, total execution time accounts for both the phases of RBRP. Each graph shows four lines. Two of these lines represent the expected



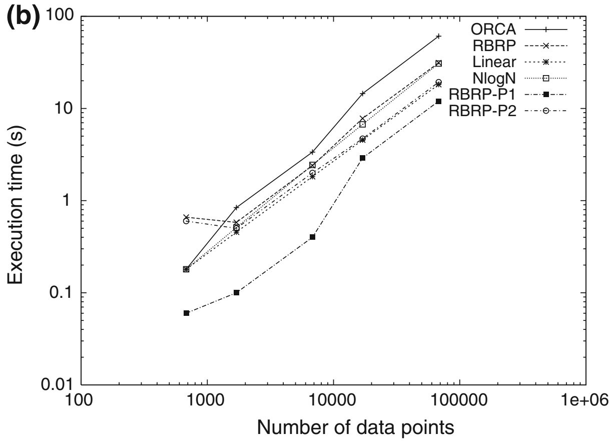
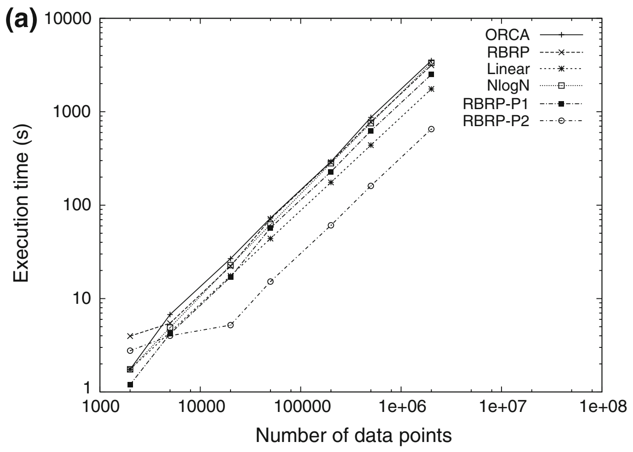


Fig. 2 (**a**) Covertype (**b**) Corel histogram

execution time to mine the data set given a linear time algorithm and an $N \log N$ time algorithm. These lines are extrapolated from the first point in the line representing ORCA's execution time. There are four remaining lines, two of which show the actual running times for RBRP and ORCA, while the remaining two lines show the running times for the individual phases of RBRP. The runs were set to mine the top 30 outliers with k set to 2.[1]

RBRP outperforms ORCA on all the considered data sets. On the Covertype, Mixed 30D, and Uniform 30D data sets, RBRP outperforms ORCA by an order of magnitude. Furthermore, it shows improved scalability with increasing data set size when compared with ORCA. We can attribute these results to the fact that while RBRP incurs an $O(N \log N)$ pre-processing overhead, it can find outliers in near constant time per data point. For data sets that have a larger number of outlying data points, the cutoff threshold is able to increase quickly, and ORCA is able to give fairly

[1] We varied k from 2 to 8 and the trends observed for $k = 2$ continue to hold with increasing k.



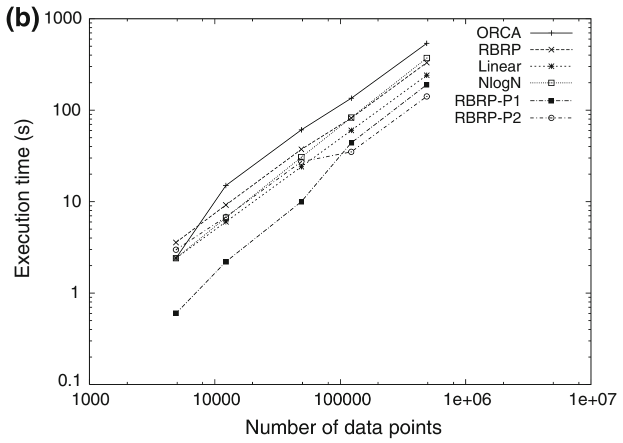
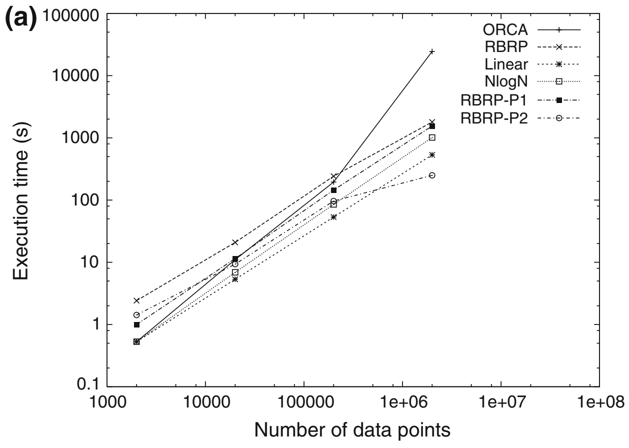


Fig. 3 (**a**) IPUMS (**b**) KDDCup 1999

good performance. This behavior can be seen on the IPUMS data set. However, when the data set has a fewer number of outliers, the cut-off threshold does not grow fast. As a result, we get near quadratic scaling performance for ORCA. This can be seen on the remaining data sets. The performance of RBRP is not affected as much by the slow decay in the cutoff threshold because of its improved search space, resulting in improved performance in all cases. Furthermore, Figs. 2–4 indicate that RBRP does indeed scale as $O(N \log N)$. We note that on the IPUMS and KDDCup 1999 data sets, it appears as though RBRP scales marginally better than $O(N \log N)$. This is simply due to the errors introduced during extrapolation (Ghoting et al. 2005).

4.3 Scalability with increasing number of nearest neighbors

Figures 5–7 show the total time to mine outliers on the six data sets as the number of nearest neighbors (k) are varied. For all these experiments, we mine outliers in the entire data set. Each graph shows the actual running times for RBRP and ORCA. The runs were set to mine the top 30 outliers.



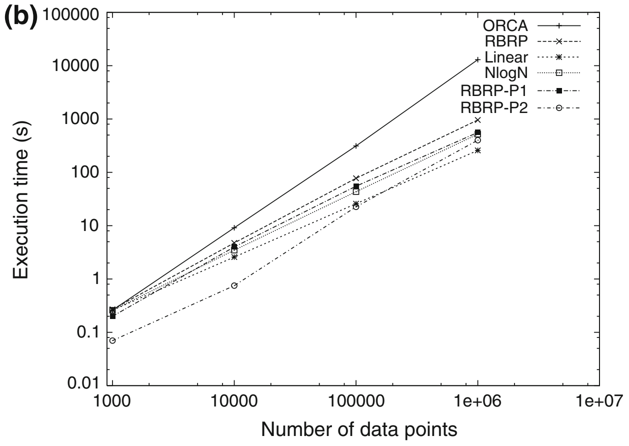
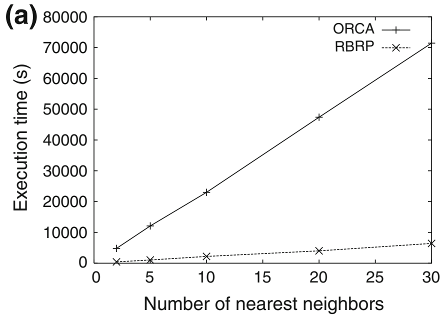


Fig. 4 (**a**) Mixed 30D (**b**) Uniform 30D



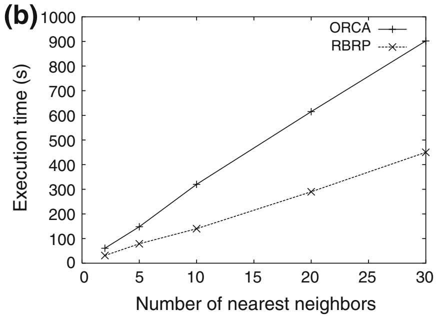
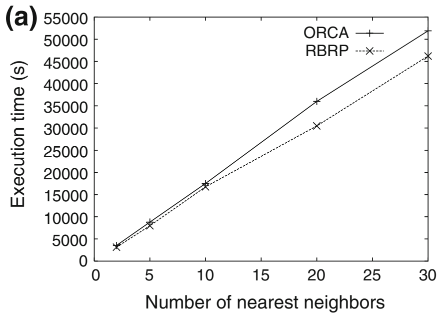


Fig. 5 (**a**) Covertype (**b**) Corel histogram



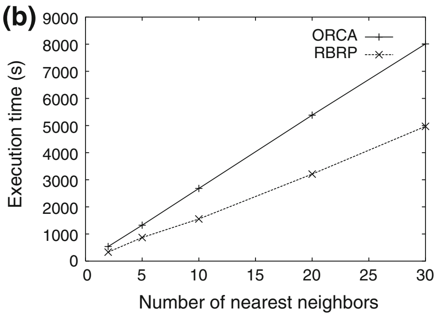
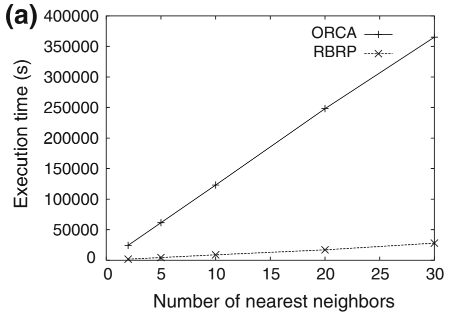


Fig. 6 (**a**) IPUMS (**b**) KDDCup 1999



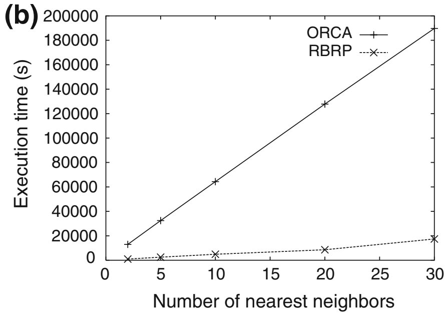


Fig. 7 (**a**) Mixed 30D (**b**) Uniform 30D

Both RBRP and ORCA exhibit linear scalability on all the considered data sets. Moreover, RBRP exhibits better scalability than ORCA with increasing k. This is attributed to the localized search for approximate nearest neighbors employed by RBRP. As k increases, for each normal point, we expect to see a constant increase in the number of bins that need to be searched. Unlike ORCA, RBRP is not affected by the slow decay in the cutoff threshold that occurs on most data sets. This is evident on all data sets except the IPUMS data set. On the IPUMS data set, the cutoff threshold converges to a large value relatively quickly. Therefore ORCA and RBRP have comparable scaling performance on this data set.

5 Conclusion

In this paper, we presented RBRP, a two phase distance-based outlier detection algorithm targeted at high-dimensional datasets. RBRP improves upon the scaling behavior of the state-of-the-art by employing an efficient pre-processing step that allows for fast determination of approximate nearest neighbors. We provide theoretical arguments as to why RBRP is expected to scale as $O(N \log N \times d)$ on d-dimensional datasets with N data points. We validated its scaling behavior on several real and synthetic datasets. Our empirical results on real data back the above claim realizing a significant speedup over ORCA, often by over an order of magnitude, while returning the same set of outliers as ORCA.

Acknowledgements This work is supported in part by NSF grants CAREER-IIS-0347662 and NGS-CNS-0406386. This work was done while Amol Ghoting and Matthew Eric Otey were at The Ohio State University.

References

Angiulli F, Pizzuti C (2002) Fast outlier detection in high dimensional spaces. In: Proceedings of the international conference on principles of data mining and knowledge discovery

Arya S, Mount DM, Netanyahu NS, Silverman R, Wu AY (1998) An optimal algorithm for approximate nearest neighbor searching fixed dimensions. J ACM 45(6):891–923

Barnett V, Lewis T (1994) Outliers in statistical data. John Wiley and Sons

Bay S (1999) The UCI KDD archive. University of California, Department of Information and Computer Science, Irvine, CA

Bay S, Schwabacher M (2003) Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In: Proceedings of the international conference on knowledge discovery and data mining

Bentley J (1975) Multidimensional binary search trees used for associative searching. Commun ACM 18:509–517

Berchtold S, Keim D, Kreigel H (1996) The X-tree: an index structure for high dimensional data. In: Proceedings of the international conference on very large data bases (VLDB)

Bolton R, Hand D (2002) Statistical fraud detection: a review. Stat Sci 17:235–255

Gamberger D, Lavrac N, Groselj C (1999) Experiments with noise filtering in the medical domain. In: Proceedings of the international conference on machine learning

Ghoting A, Parthasarathy S, Otey M (2005) Fast mining of distance-based outliers in high dimensional datasets. Technical report TR71, Department of Computer Science and Engineering, The Ohio State University

Guttmann R (1984) A dynamic index structure for spatial searching. In: Proceedings of the international conference on management of data (SIGMOD)

Hartigan J (1975) Clustering algorithms. John Wiley and Sons

Indyk P, Motwani R (1998) Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the symposium on theory of computing (STOC), pp 604–613

Jagadish H, Poosala V, Koudas N, Sevcik K, Muthukrishnan S, Suel T (1998) Optimal histograms with guarantees. In: Proceedings of the international conference on very large databases (VLDB)

Jolliffe I (1986) Principal component analysis. Springer-Verlag

Kleinberg JM (1997) Two algorithms for nearest-neighbor search in high dimensions. In: Proceedings of the symposium on theory of computing (STOC), pp 599–608

Knorr E, Ng R (1999) Finding intensional knowledge of distance-based outliers. In: Proceedings of the international conference on very large data bases (VLDB)

Kushilevitz E, Ostrovsky R, Rabani Y (1998) Efficient search for approximate nearest neighbor in high dimensional spaces. In: Proceedings of the symposium on theory of computing (STOC)

Muralikrishna M, DeWitt D (1988) Equi-depth histograms for estimating selectivity factors for multidimensional queries. In: Proceedings of the international conference on management of data (SIGMOD)

Ramaswamy S, Rastogi R, Shim K (2000) Efficient algorithms for mining outliers from large datasets. In: Proceedings of the international conference on management of data

Ruggles S, Sobek M (1997) Integrated public use microdata series: version 2.0

Sequeira K, Zaki M (2002) ADMIT: anomaly-based data mining for intrusions. In: Proceedings of the international conference on knowledge discovery and data mining

Zhang T, Ramakrishnan R, Livny M (1996) BIRCH: an efficient data clustering method for very large databases. In: Proceedings of the international conference on management of data (SIGMOD)