Fast Mining of Distance-Based Outliers in High-Dimensional Datasets*

Amol Ghoting, Srinivasan Parthasarathy, and Matthew Eric Otey
Department of Computer Science and Engineering
The Ohio State University, Columbus, OH 43210, USA.
Contact email: srini@cse.ohio-state.edu

Abstract

Defining outliers by their distance to neighboring data points has been shown to be an effective non-parametric approach to outlier detection. Existing algorithms for mining distance-based outliers do not scale to large, high-dimensional data sets. In this paper, we present RBRP, a fast algorithm for mining distance-based outliers, particularly targeted at high-dimensional data sets. 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, often by an order of magnitude.

Keywords: Outlier detection, high-dimensional data sets, approximate k-nearest neighbors, clustering.

1 Introduction

A common problem in data mining is that of automatically finding outliers or anomalies in a data set. 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.

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 [2]. 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 [1, 10, 11]. The following (among others [3]) is a popular definition of distance-based outliers:

- Outliers are the top n data points whose distance to the k^{th} nearest neighbor is greatest [11].

While distance-based outlier detection has proven to be useful, the process continues to be time consuming. The nested loop (NL) algorithm for mining distance-based outliers [10] 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 [4], R-trees [7], or X-trees [5]) for fast *nearest neighbor* computation to partitioning the feature space with clustering [11]. Unfortunately, these approaches do not scale well with the number of dimensions [3, 10]. Consequently, for high dimensional data sets, solutions based on the simple NL algorithm are known to provide the best performance [10, 3].

The algorithm, in its simplest form, is presented in Table 1. The main idea in the NL algorithm is that for each data point in D, we keep track of its k closest neighbors as we scan the data set. When a data point's k^{th} 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. As we process more data points, the algorithm finds more extreme outliers, and the cutoff increases giving us improved pruning efficiency. The state-of-the-art distance-based outlier detection algorithm, ORCA [3], uses the NL algorithm with a preprocessed data set. ORCA randomizes the data set (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 data sets.

In this paper, we further improve the scaling behavior of distance-based outlier detection on large, high-dimensional data sets. Specifically, we make the following contributions. First, we study the conditions under which the state-of-the-art distance-based outlier detection algorithm, ORCA (due to Bay and Schwabacher [3]), is unable to provide near-linear time performance. Second, we present RBRP, an algorithm for fast mining of distance-based outliers. The algorithm 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 efficient distance-based outlier detection. Finally,

*This work is supported in part by NSF grants CAREER-IIS-0347662 and NGS-CNS-0406386. Please refer to [6] for a detailed version of this paper.

```
Procedure: Find Outliers
Input: $k$, the number of nearest neighbors;
       $n$, the number of outliers to be returned;
       $D$, the set of data points.
Output: $O$, the set of outliers.
begin
  $c = 0$ ($c$ is the cutoff threshold)
  $O = \{\}$
  for each $d$ in $D$
    Neighbors($d$) = {}
    for each $b$ in $D$ such that $b \neq d$
      if $|$Neighbors($d$)$| < k$ or
      Distance($b, d$) < Maxdist($d$, Neighbors($d$))
        Neighbors($d$) = Closest($d$, Neighbors($d$) $\cup$ $b$, $k$)
      endif
      if $|$Neighbors($d$)$| \geq k$ and $c >$ Distance($b, d$))
        break
      end if
    end for
    $O$ = TopOutliers($O \cup b$, $n$)
    $c$ = MaxThreshold($O$)
  end for
end
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
```

Table 1: The Simple Nested Loop Algorithm

we demonstrate that our algorithm scales well to high-dimensional data sets 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, we show that the algorithm scales log-linearly as a function of the number of data points, and linearly as a function of the number of dimensions.

2 Outlier Detection Algorithm

Shortcomings of ORCA**:**
For expository simplicity, let us assume that we are interested in finding the top n data points whose distance to the 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:

$$P(Y = y) = \Pi(x) \times (1 - \Pi(x))^{(y-1)} \quad (2.1)$$

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

$$E[Y] = \sum_{y=1}^{N} P(Y = y) \times y = \frac{1}{\Pi(x)} \quad (2.2)$$

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 $O(\sqrt{N})$. The cutoff threshold in unlikely to converge to such a large value quickly, not just for a uniformly distributed data set, but for any arbitrary data set. In summary, ORCA delivers near-linear scaling behavior only when the cutoff distance can quickly converge to a large value. This can occur only when the data set has a large number of outlying points. When the data set consists of a mixture of a few distributions, with not many outlying points, ORCA's complexity is near quadratic [3].

Algorithm RBRP (Recursive Binning and Re-Projection): As pointed out in Section 1, 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. We now present RBRP (Recursive Binning and Re-Projection), a *two-phase* algorithm for fast mining of distance-based outliers in high dimensional data sets.

Phase 1: The goal of the first phase of RBRP is to partition the data set 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 [8] 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 [12]. This is in some senses similar to the approach proposed by Ramaswamy *et al.* [11]. However, this approach will not scale to high-dimensional data.

Our approach to partitioning the data set into bins is shown in Table 2. It is a recursive procedure known as 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 [8] algorithm. Essentially, we start with k random centers, and assign each point to its closest center, creating k partitions. Next, we find k centers for these k partitions, 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 the size of the partition is greater than a user-defined threshold ($Binsize$). 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 [9] of the points in the bin. 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.

Complexity analysis: We expect Phase 1 to scale as $O(NlogN \times d)$ in the average case [6].

Phase 2: In Phase 2, we use an extension of the NL algorithm to find outliers in the data set 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. 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, and continue searching for approximate nearest neighbors. This search continues iteratively until k approximate nearest neighbors are discovered.

Complexity analysis: The worst case time complexity of Phase 2 is $O(N^2)$. However, we expect to find the approximate nearest neighbors of a normal point in the very same bin. 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 data set size (N). Therefore, we expect Phase 2 to scale as $O(N \times d)$. As Phase 1 scales as $O(NlogN \times d)$, we expect RBRP to scale close to $O(NlogN \times d)$.

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 data set.

3 Experimental Results

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 using C. We use several real and synthetic data sets for our analysis. These data sets are summarized in Table 4. They span a range of problems and have different types of features. Please refer to [6] for more details on the data sets and implementations.

Scalability with increasing data set size: Figures 1-2 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 execution time to mine the data set given a linear time algorithm and an $NlogN$ time algorithm. These lines are extrapolated from the first point in the line representing ORCA's execution time. The two remaining lines show the actual running times for RBRP and ORCA. The runs were set to mine the top 30 outliers with k set to 2.

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 $O(NlogN)$ 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 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, Figures 1-2 indicate that RBRP does indeed scale as $O(NlogN)$. We note that on the Ipums and KDDCup 1999 data sets, it appears as though RBRP scales marginally better than $O(NlogN)$. This is simply due to the errors introduced during extrapolation [6].

Scalability with increasing number of nearest neighbors: Figures 3-4 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

```
Procedure: Bin
Input: $Binsize$, the maximum size of a bin;
        $k$, the number of partitions; $it$, no. of iterations;
        $D$, data points to be binned.
Output: $B$, the set of bins.
begin
  $c = \{c_1, c_2, ..., c_k\}$ (the set of $k$ random centers)
  $p = \{p_1, p_2, ..., p_k\}$ (the set of $k$ partitions of $D$)
  for $it$ iterations
    Empty all $k$ partitions in $p$
    for each $d$ in $D$
      $j$ = Closest($c$, $d$)
      Insert($d$, $j$)
    end for
    $c = \{\}$
    RecomputeCenters($c$, $p$)
  end for
  for each $p_i$ in $p$
    if size of $p_i > Binsize$
        Bin($Binsize$, $k$, $it$, $p_i$)
    else
        Reorganize data points in $p_i$, ordered as per their
          projection along the principal component of $p_i$
        Add $p_i$ to $B$
    end if
  end for
end
Note:
Closest($c$, $d$) returns the index of the nearest elements in $c$ to $d$
Insert($d$, $j$) inserts point $d$ in $j$th partition in $p$
RecomputeCenters($c$, $p$) inserts $k$ centers of partitions
   in $p$ into $c$
```

Table 2: RBRP Phase 1

```
Procedure: Find Outliers
Input: $k$, the number of nearest neighbors;
        $n$, the number of outliers to be returned;
        $D$, the set of data points.
Output: $O$, the set of outliers.
begin
  $c = 0$ ($c$ is the cutoff threshold)
  $O = \{\}$
  for each bin $b$ in $B$
    for each $d$ in $b$
      Neighbors($d$) = {}
      for each $t$ in $B$, ordered
      by increasing distance to $b$
        for each $p$ in $t$ such that $p \neq d$
            if |Neighbors($d$)| $< k$ or
            Distance($d, p$) $<$ Maxdist($d$, Neighbors($d$))
              Neighbors($d$) =
                Closest($d$, Neighbors($d$) $\cup$ $p$, $k$)
            endif
            if |Neighbors($d$)| $\geq k$ and $c >$ Distance($p, d$))
                break
            end if
        end for
      end for
    end for
    $O$ = TopOutliers($O \cup b$, $n$)
    $c$ = MaxThreshold($O$)
  end for
end
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
```

Table 3: RBRP Phase 2

Data set	Continuous Attributes	No. of Points
Corel Histogram	32	68,040
Covertype	55	5,81,012
KDDCup 1999	24	4,898,430
Mixed 30D	30	2,000,000
Uniform 30D	30	1,000,000
Ipums	128	2,000,000

Table 4: Data sets

runs were set to mine the top 30 outliers.

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.

4 Conclusion

In this paper, we presented RBRP, a two phase distance-based outlier detection algorithm targeted at high-dimensional data sets. 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. RBRP is expected to scale as $O(NlogN \times d)$ on d-dimensional data sets with N data points. We validated its scaling behavior on several real and synthetic data sets. RBRP consistently outperforms ORCA, the state-of-the-art distance-based outlier detection algorithm, often by an order of magnitude.

References

[1] F. Angiulli and C. Pizzuti. Fast outlier detection in high dimensional spaces. In *PKDD*, 2002.

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

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

[4] J. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 1975.

[5] S. Berchtold, D. Keim, and H. Kreigel. The X-tree: an index structure for high dimensional data. In *VLDB*, 1996.

[6] A. Ghoting, S. Parthasarathy, and M. Otey. Fast mining of distance-based outliers in high dimensional datasets. Technical report, TR71, CSE, The Ohio State University, 2005.

[7] R. Guttmann. A dynamic index structure for spatial searching. In *SIGMOD*, 1984.

[8] J. Hartigan. *Clustering Algorithms*. John Wiley and Sons, 1975.

[9] I. Jolliffe. *Principal Component Analysis*. Springer-Verlag, 1986.

[10] E. Knorr and R. Ng. Finding intensional knowledge of distance-based outliers. In *VLDB*, 1999.

[11] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large datasets. In *SIGMOD*, 2000.

[12] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *SIGMOD*, 1996.

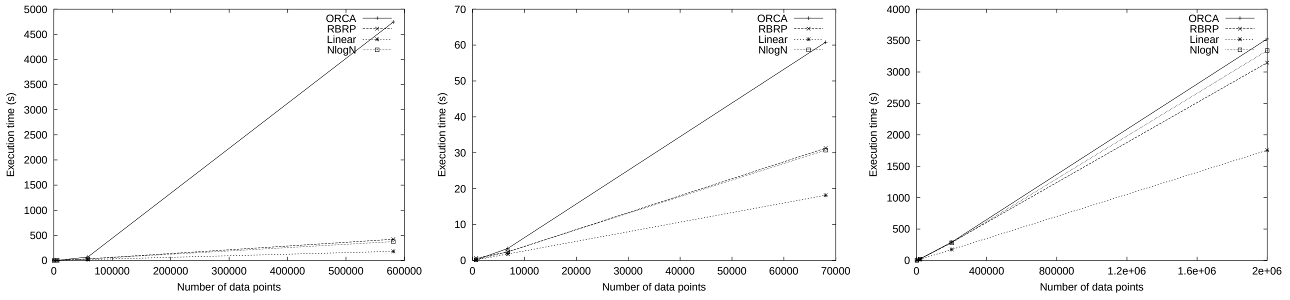

Figure 1: (a) Covertype (b) Corel histogram (c) Ipums

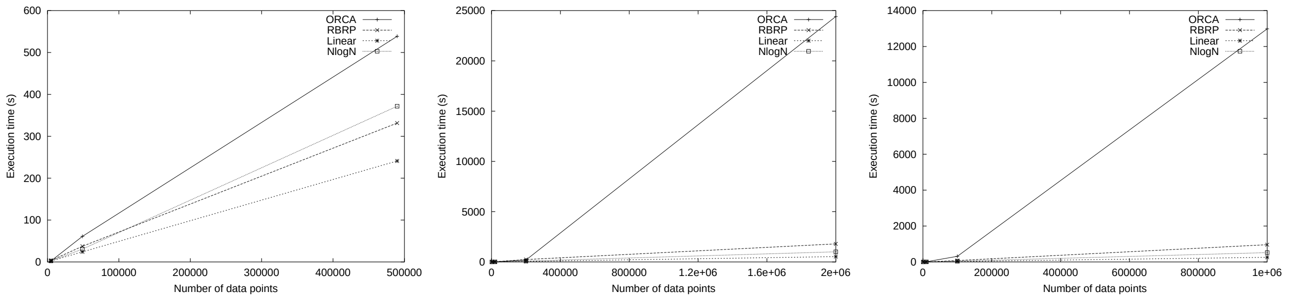

Figure 2: (d) KDDCup 1999 (e) Mixed 30D (f) Uniform 30D

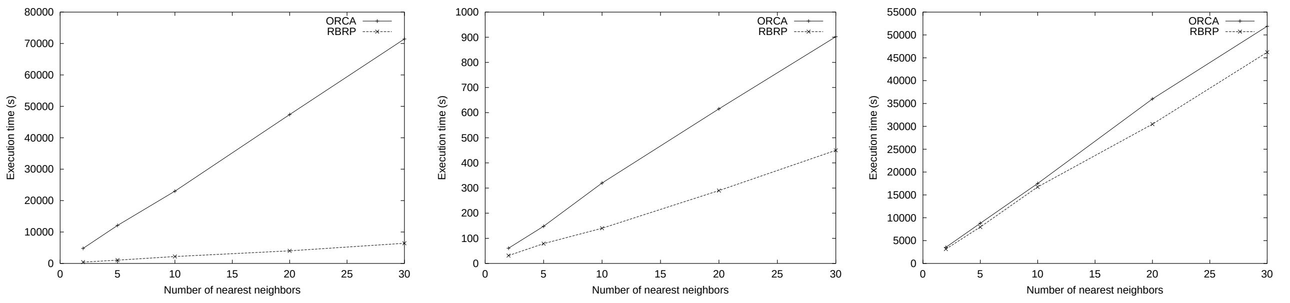

Figure 3: (a) Covertype (b) Corel histogram (c) Ipums

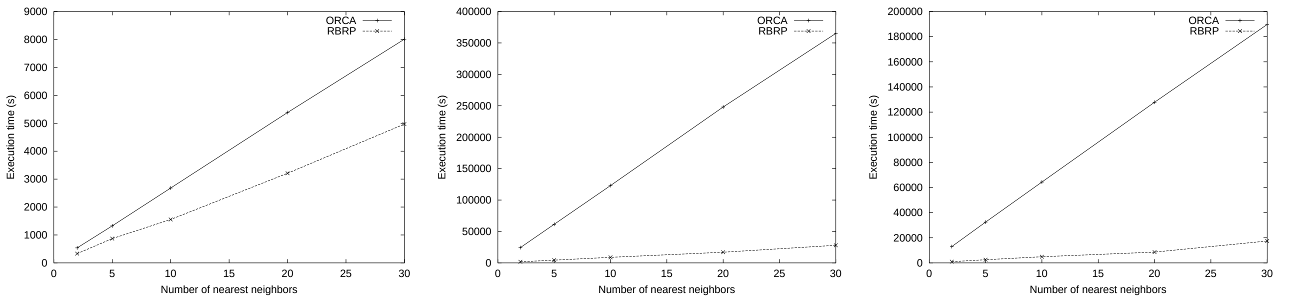

Figure 4: (d) KDDCup 1999 (e) Mixed 30D (f) Uniform 30D