# Parallel algorithms for distance-based and density-based outliers

Elio Lozano University of Puerto Rico Mathematics Department Mayaguez, PR 00680 elio\_li@math.uprm.edu

#### Abstract

An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism. Outlier detection has many applications, such as data cleaning, fraud detection and network intrusion. The existence of outliers can indicate individuals or groups that exhibit a behavior that is very different from most of the individuals of the dataset. In this paper we design two parallel algorithms, the first one is for finding out distance-based outliers based on nested loops along with randomization and the use of a pruning rule. The second parallel algorithm is for detecting densitybased local outliers. In both cases data parallelism is used. We show that both algorithms reach near linear speedup. Our algorithms are tested on four real-world datasets coming from the Machine Learning Database Repository at the UCI.

# 1 Introduction

According to Hawkins [6], "An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism". Almost all the studies that consider outlier identification as their primary objective are in the field of statistics. A comprehensive treatment of outliers appears in Barnet and Lewis [1]. They provide a list of about 100 tests for detecting outliers in data following well known univariate distributions. However, Real-world data are commonly multivariate with unknown distribution.

Detecting outliers, instances in a database with unusual properties, is an important data mining task. People in the data mining community got interested in outliers after Knorr and Ng [10] proposed a non-parametric approach to outlier detection based on the distance of an instance to its nearest neighbors. Outlier detection has many applications among them: Fraud detection and network intrusion, and Edgar Acuña University of Puerto Rico Mathematics Department Mayaguez, PR 00680 edgar@cs.uprm.edu

data cleaning. Frequently, outliers are removed to improve accuracy of the estimators. However, this practice is not recommendable because sometimes outliers can have very useful information. The presence of outliers can indicate individuals or groups that have behavior very different from a standard situation.

One might think that multivariate outliers can be detected based on the univariate outliers in each feature, but this is not true. On the other hand, an instance can have values that are outliers in several features but the whole instance might not be a multivariate outlier. A basic method for detecting multivariate outliers is to observe the outliers that app ear in the distribution of the Mahalanobis square distance of the all instances. There are several methods for detecting multivariate outlier: Robust statistical-based outlier detection ([5],[15],[16]), outlier detection by clustering ([5],[9],[13]), outlier detection by neural networks [8], distance-based outlier detection[[2], [11],[12],[14]) and density-based local outlier detection[4].

Hung and Cheung (2000), proposed parallel algorithms for mining distance-based outliers as proposed in [11] and [12]. The authors claimed that their algorithm can be used to parallelize the algorithm for detecting density-based local outliers, although they did not carried out any experiment. Ruoming and Agrawal (2001) use local nearest neighbor property to parallelize KNN classifier. We had used some strategies described in Skillircon (1999) to parallelize Bay algorithm. This paper is organized as follows, in Section 2 focuses on distance-based outlier detection. Section 3 of this paper considers density local-based. The experimental results appear in section 4, and the conclusions of our work are presented in section 5.

### 2 Distance-based outlier detection

Given a distance measure on a feature space, two different definitions of distance-based outliers are the following:

1. An instance **x** in a dataset D is an outlier with parameters p and  $\lambda$  if at least a fraction p of the objects are a distance greater than  $\lambda$  from **x** [11], [12]. This definition has certain difficulties such as the determination of  $\lambda$  and the lack of a ranking for the outliers. Thus an instance with very few neighbors within a distance  $\lambda$  can be regarded as strong an outlier as an instance with more neighbors within a distance  $\lambda$ . Furthermore, the time complexity of the algorithm is O(vn<sup>2</sup>), where v is the number of features and n is the number of instances. Hence it is not an adequate definition to use with datasets having a large number of instances.

2. Given the integer numbers k and n (k<n), outliers are the top n instances with the largest distance to their k-th nearest neighbor [15]. One shortcoming of this definition is that it only considers the distance to the k-th neighbor and ignores information about closer points. An alternative is to use the greatest average distance to the k nearest neighbors. The drawback of this alternative is that it takes longer calculate.

### 2.1 The Bay's Algorithm.

Bay and Schwabacher [2] proposed a simple nested loop algorithm that tries to reconcile definitions 1 and 2. It gives near linear time performance when the data is in random order and a simple pruning rule is used. The performance of the algorithm in the worst case is of quadratic order. The algorithm is described in Fig. 1.

The main idea in the algorithm is that for each instance in D one keeps track of the closest neighbors found so far. When an instance's closest neighbors achieve a score lower than a cutoff then the instance is removed because it can no longer be an outlier. Bay and Schwabacher used the as score function the sum of the distances to the k neighbors, but also the average distance as well the median distance can be considered. As more instances are processed the algorithm finds more extreme outliers and the cutoff increases along with pruning efficiency.

Bay and Schwabacher showed experimentally that their algorithm is linear with respect to the number of neighbors and that is almost linear with respect to the number of instances. Using six large datasets they found a complexity of order  $O(n^{\alpha})$  where  $\alpha$  varied from 1.13 to 1.32. Although this reduction is favorable, yet it is heavy for large data sets. For this reason we propose a parallel algorithm that use Bay's algorithm.

# 2.2 Parallel Bay's Algorithm

We have constructed a parallel algorithm for the Bay's algorithm based on the Local nearest neighbors property. Once the data are distributed between processes, each process computes its local neighbors and send its results to master, which compute the global neighbors and then finds the top outliers. After that it sends the cutoff parameter to **Input:** k: number of nearest neighbors; n: number of outliers to return; D: dataset randomly ordered, B: size of blocks in which D is divided. Let distance(x,y) the Euclidean distance between x and y. Let maxdist(x,Y) maximum distance between x and an example in Y. Let Closest(x,Y,k) return the k closest examples in Y to x. begin 1.  $c \leftarrow 0 //$  set the cutoff for pruning to 0  $2.0 \leftarrow \emptyset // \text{ initialize to the empty set}$ 3.while  $B \leftarrow \text{get-next-block}(D) \{ //\text{load a block from } D \}$ 4. Neighbors(b)  $\leftarrow \emptyset$  for all b in B 5. for each d in D  $\{$ for each b in B,  $b \neq d$ 6. 7. if | Neighbors(b)| < k or distance(b,d) < maxdist(b,Neighbors(b)) { 8. Neighbors(b)  $\leftarrow$  Closest(b ,Neighbors(b)  $\cup$  d,k) 9. if (score(Neighbors(b),b)<c){ 10. remove b from B 11. }}} 12. Top $(B \cup O, n)$ //keep only the top n outliers 13.  $c \leftarrow \min(score(o))$  for all  $o \in O$ 14.} 15.return O end



Output: O, a set of outliers

each process. The proposed algorithm is given in the figure 2.

In the worst case Bay said that its algorithm run in  $O(N^2)$  in adittion to N/blocksize \* N data access. Experimentally using polynomial regression of empirical running time, he founded an exponent varyng from 1.13 to 1.32. This exponent was be found from fitting  $t = aN^b$  where t is the total time, a and b are constants. Assuming that our algorithm has simmilar complexity for our experimental dataset, we denote this complexity as  $O(N^{\alpha})$ , where  $1.13 \leq \alpha \leq 1.32$ . So the upper bound of total time of the parallel Bay algorithm is  $(N^{\alpha}p + n)t_{comp} + Npt_{I/O} + N^{\alpha}kt_{comm}$  (the sum of computation time, I/O time and communication time). Where p is the number of processes.

### **3** Density-based local outlier detection

For this type of outlier the density of the neighbors of a given instance plays a key role. Furthermore an instance is not explicitly classified as either outlier or non-outlier; instead for each instance a local outlier factor (LOF) is begin 1.  $c \leftarrow 0 //$  set the cutoff for pruning to 0  $2.0 \leftarrow \emptyset$  // initialize to the empty set 3.while  $B \leftarrow get-next-block(D){//load}$  a block from D 4. Neighbors(b)  $\leftarrow \emptyset$  for all b in B 5. for each d in LocalD { //Each process compute its local neighborhoods for each b in B,  $b \neq d$ 6. 7. if | Neighbors(b)| < k or distance(b,d) < maxdist(b,Neighbors(b)) { 8. Neighbors(b)  $\leftarrow$  Closest(b,Neighbors(b)  $\cup$  d,k) 9. if (score(Neighbors(b),b)<c){ 10. remove b from B 11. }}} Each process sends its local neighborhoods to master 12. 13. Master process { 14. Computes global neighbors from local neighbors 15. computes  $Top(B \cup O, n) //$  keeps only the top n outliers 16. computes  $c \leftarrow \min(score(o))$  for all o in O Broadcast the cutoff parameter } 17. 18.} 19.return O end

Figure 2. Parallel Bay's Algorithm

computed which will give an indication of how strongly an instance can be considered an outlier. Breuning et al. [4], show through an example the weakness of the distancebased method in identifying certain type of outliers.

The following definitions are needed in order to formalize the algorithm to detect density-based local outliers:

k-distance of an instance x. For any positive integer k, the k-distance of an instance x, denoted by k-distance(x), is defined as the distance d(x,y) between x an instance y ∈ D such that:

(i) for at least k instances  $y'\in D{-}\{x\}$  it holds that  $d(x,y')\leq d(x,y),$ 

(ii) for at most k-1 instances  $y' \in D-\{x\}$  it holds that d(x,y') < d(x,y).

- 2. *k-distance neighborhood of an instance x*. Given the k-distance of x, the k-distance neighborhood of x contains every instance whose distance from x is not greater than the k-distance; i.e.  $N_{k-distance(x)}(x) = \{q \in D \{x\} : d(x,q) \le k distance(x)\}$
- 3. *reachability distance of an instance x w.r.t. object y.* Let k be a positive integer number. The reachability distance of an instance x with respect to the instance y is defined as

 $reach - dist_k(x, y) = max\{k - distance(y), d(x, y)\}$ 

4. *local reachability density of an instance x*. It is given by

$$lrd_{Minpts}(x) = \left[\frac{\sum_{o \in N_{Minpts}(x)} reach - dist_{Minpts}(x, o)}{|N_{Minpts}(x)|}\right]^{-1}$$

The *lrd* is the average *reachability distance* based on the *MinPts*-nearest neighbor of the instance x.

5. *local outlier factor of an instance x*. The local outlier factor of x is defined as

$$LOF_{Minpts}(x) = \frac{\sum_{o \in N_{Minpts}(x)} \frac{lrd_{Minpts}(o)}{lrd_{Minpts}(x)}}{|N_{Minpts}(x)|}$$

The density-based local algorithm to detect outliers requires only one parameter, MinPts, which is the number of nearest neighbors used in defining the local neighborhood of the instance. The LOF measures the degree to which an instance x can be considered as an outlier. Breuning et al.[4] show that for instances deep inside a cluster their LOF's are close to 1 and should not be labelled as a local outlier. Since LOF is not monotonic, they recommended finding the LOF for each instance of the datasets using MinPts-nearest neighbor, where MinPts assumes a range of values from MinPtsLB to MinPtsUB. They also suggested MINPtsLB=10 and MinPtsUB=20. Having determined MInPtsLB and MinPtsUB, the LOF of each instance is computed within this range. Finally all the instances are ranked with respect to the maximum LOF value within the specified range. That is, the ranking of an instance x is based on:

Max{LOF<sub>MinPts</sub>(x) / MinPtsLB≤MinPtsUB}

The LOF algorithm to detect density-based local outliers is shown in Fig. 3. Breuning et al. discusses in detail the time complexity of the LOF algorithm.

The serial LOF's algorithm appears in figure 3

### 3.1 Parallel LOF

The major task to be carried out in the serial LOF algorithm relies on the computation of KDNeighbors (which is the matrix that contains the elements of D with its respective k-neighbors). For this reason we attempt to parallelize in that step. Each process computes its respective KDNeighbors matrix, and then send its result to the master process, which collects the results and then

Input:
klb and kub the lower and upper bounds of k-distance
neighborhoods.
D a set of examples.
The number of top outliers
Output: lof a vector with local density factors
Let kdis-neighbors(D,k) return a matrix that containts
the k-distances neighborhoods and their k-distances.
Let reachability(KDNeighbors) return the local reachability
density of each p in D
Begin
$1.lof \leftarrow NULL$
2.for each k in {klb,, kub} {
3. KDNeighbors $\leftarrow$ kdis-neighbors(D,k)
4. lrddata ← reachability(KDNeighbors,k)
5. for each p in KDNeighbors
6. templof[i] $\leftarrow$ sum(lrddata[o \in N(p)])/lrddata[i])/ $ N(p) $
7. lof $\leftarrow$ max{lof, templof}}
8.return top(lof)
End
Output: lof

#### Figure 3. The LOF Algorithm

computes the reachability and local outlier factor. The proposed parallel algorithm is given in the figure 4.

1.lof ←NULL 2.for each k in  $\{klb, ..., kub\}$ 3. Each process computes its respective KDNeighbors and sends it to master 4. Master process collects the partial KDNeighbors and finds the KDNeighbors matrix 5. Master process {  $lrddata \leftarrow reachability(KDNeighbors,k)$ 6. 7. for each p in KDNeighbors { templof[i]  $\leftarrow$  sum(lrddata[o \in N(p)])/lrddata[i])/ |N(p)|8. 9.  $lof \leftarrow max{lof, templof}$ 10. } 11.return lof



Parallel LOF uses the master slave paradigm. Master process sends the data set to slaves, to calculate distances between its parts. Then master process collects the results and find the LOF factor for each observation.

The total upper bound of computation time of LOF algorithm is  $Nvt_{I/O} + (kub - klb)((N^2v + 2Nk)t_{comp})$ , where N is the number of instances, klb and kub are the lower and upper bounds, v the number of variables, k is the k-distance neighborhoods. Therefore, the parallel algorithm has a total upper bound of  $Nvt_{I/O} + (kup - klb)((N(\frac{N}{p}v + 2Nk)t_{comp} + Nkt_{comm})))$ , where p is the number of processes.

# **4** Experimental results

We had tested our algorithms in the following environment.

# 4.1 Cluster Description

Our computer environment consists of a cluster of 4 nodes HP Itanium 2 6M zx6000 (IA64 Architecture). Each node has 2 processors running at 1.5 GHz with 4 GB of main memory. Each running Red Hat Advanced Workstation 2.1. We had implemented our algorithms in C++ (gcc version 3.2.3) using some libraries of LAM MPI version 7.0 [14], under Red Hat Advanced Workstation 2.1.

#### 4.2 Data sets

Four well-known Machine Learning datasets, *Landsat*, *Census, Shuttle*, and *Covtype* are used to show the performance of our parallel algorithms. These datasets are available on the Machine Learning database repository at the University California, Irvine [6].

- Landsat. It consists of 4435 instances. each instance represents a satellite image with 36 features. The data is classified in six classes.
- Shuttle. The NASA Shuttle (Catlett, 1991), contains 43,500 training instances, with 9 continuous attributes. Therea are seven clases.
- Census. This dataset consists of 14 features measured on 32561 subjects. The features try to classify the subjects in two classes: "income < 50k", and "income ≥ 50k". A 7 percent of subjects have incomplete information and we have deleted them.
- Covtype. This dataset obtained from the US Forest Service (USFS). It contains measures of seven types of soils with 581,012 instances, and 54 attributes.

#### 4.3 Description of Input and Output Parameters

In the Bay's algorithm, we setup with big numbers (of order  $10^6$ ) the initial k distance to the k neighbors for each  $b \in B$  in Bay's algorithm (see line 4). Also we use the following parameters: Block size = 1000, number of neighbors = 10, number of top outliers = 10. In The LOF algorithm,

we set up k-lower bound = 10, k- upper bound = 20, number of top outliers = 10. For both algorithms we detect outliers only in the first class.

### 4.4 Runtime and Speedup

In Figures 5, 6, 7 and 8 we show overall runtime for our parallel algorithms, and its corresponding speedups, for different number of processors and different datasets. Our algorithms overall shows good speedup results. The slowdown observed in Figure 6 for Covtype is mainly due to the high communication cost in the phase of exchange the neighborhoods, this occurs because the block size is small for the choice of our parameters. For Landsat occurs too this slowdown because the data set is small. But for the other datasets, the speedup is almost linear. The speedup of parallel LOF (Figure 8) reaches linear speedup for all datasets.



Figure 5. Runtime for Parallel Bay's Algorithm

# 5 Conclusion

There are plenty of algorithm for detecting outliers. Two of them are the Bay's and LOF algorithms, which have good performance in runtime and outlier detection. We have constructed parallel version of both algorithms. Parallel Bay's algorithm based in the nearest neighbor property, reaches good performance, in which each processor runs the same Bay's algorithm but locally in its own dataset. After that it sends its local neighborhoods to the master process. The Master process receives the partial neighbors and computes the cutoff and sends it to the slaves for the next iteration. This parallel Bay's algorithm reaches almost linear speedup. On the other hand the heavy work in the LOF



Figure 6. Speedup for Parallel Bay's Algorithm

algorithm resides in the computation of the k-distance nearest neighborhoods for each observation. We proposed a new parallel LOF algorithm. The master process sends the entire data set to slaves, and assigns their task. Each slave computes its respective kdistance nearest neighborhoods of its respective data, and sends it back to the master. Once that the master receives the results from the slaves, it computes the local reachability and LOF factor for each observation. This algorithm gives good speedup, because there is a few communication. All our algorithms are tested and validated with four real datasets.



Figure 7. Runtime for Parallel LOF Algorithm

ACKNOWLEDGMENT



Figure 8. Speedup for Parallel LOF Algorithm

This research was partially supported by grant N00014-00-1-0360 from ONR. The authors want to thank Caroline Rodriguez for her suggestions on programming the algorithms.

# References

- V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley, New York, 1994.
- [2] S. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.
- [3] C. Blake and C. Mertz. Uci repository of machine learning databases. [http://www.ics.uci.edu/mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 1998.
- [4] M. Breuning, H. Kriegel, R. Ng, and J. Sander. LOF: Identifying density-based local outliers. ACM SIGMOD International Conference on Management of Data, 2000.
- [5] J. Hardin and D. Rocke. Outlier detection in the multiple cluster setting using the minimum covariance determinant estimator. *Computational Statistics and Data Analysis*, 44:625–638, 2004.
- [6] D. Hawkins. *Identification of Outliers*. Chapman and Hall, London, 1980.
- [7] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22:85–126, 2004.
- [8] E. Hung and D. Cheung. Parallel Mining of Outliers in Large Database. *Distributed and Parallel Databases. Kluver Academic Publishers.*, (12):5–26, 2004.
- [9] L. Kaufman and P. Rousseeuw. Finding Groups in Data: An Introduction to Cluster Analysis. Wiley, New York, 1990.

- [10] E. Knorr and R. Ng. Algorithms for mining distance-based outliers in large datasets. *In Proc. 24th Int. Conf. Very Large Data Bases VLDB*, pages 392–403, 1998.
- [11] E. Knorr, R. Ng, and V. Tucakov. Aistance-based outliers: Algorithms and applications. *VLDB Journal: Very Large Data Bases*, (8(3-4)):237–253, 2000.
- [12] R. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. Proc. 20<sup>th</sup> Int. Conf. on Very Large Databases. Morgan and Kaufmann Publishers, San Francisco, (8(3-4)):44–155, 1994.
- [13] P. Pacheco. Parallel Programming with MPI. Morgan Kauffmann Publishers Inc., 1997.
- [14] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 427–438, 2000.
- [15] D. Rocke and D. Woodruff. Computational connections between robust multivariate analysis and clustering. In COMP-STAT 2002 Proc. in Computational Statistics, Wolfgang H}.
- [16] P. Rousseeuw and A. Leroy. Robust Regression and Outlier Detection. MJohn Wiley, 1987.
- [17] J. Ruoming and G. Agrawal. A Middleware for developing parallel data mining applications. Proc. of the First SIAM Conference on Data Mining, 2001.
- [18] D. Skillicorn. Strategies for Parallel Data Mining. IEEE Concurrency, 4(7):36–35, 2001.