# Discovering Cluster Based Local Outliers

**Zengyou He*, Xiaofei Xu, Shengchun Deng**

*Department of Computer Science and Engineering, Harbin Institute of Technology, Harbin 150001, P. R. China*

Received XX; received in revised form XX

## Abstract

In this paper, we present the new definition for outlier: *cluster-based local outlier*, which is meaningful and provides importance to the local data behavior. A measure for identifying the physical significance of an outlier is designed, which is called *CBLOF (Cluster-Based Local Outlier Factor)*. We also propose the *FindCBLOF* algorithm for discovering outliers. The experimental results show that our approach outperformed the existing methods on identifying meaningful and interesting outliers.

***Keywords*:** Outlier Detection, Clustering, Data Mining.

## 1. Introduction

An outlier in a dataset is defined informally as an observation that is considerably different from the remainders as if it is generated by a different mechanism. Mining for outliers is an important data mining research with numerous applications, including credit card fraud detection, discovery of criminal activities in electronic commerce, weather prediction, marketing and customer segmentation.

Recently, some studies have been proposed on outlier detection (e.g. Knorr and Ng, 1998; Ramaswamy, et al., 2000; Breunig, et al., 2000; Aggarwal and Yu, 2001) from the data mining community. This paper presents a new definition for outlier, namely *cluster-based local outlier*, which is intuitive and meaningful. This work is motivated by the following observations.

Firstly, all existing algorithms for outlier detection involve high computation costs, which is not feasible in the environment of access to large data sets stored in secondary memory. Furthermore, algorithms presented by Knorr and Ng (1998), Ramaswamy et al.(2000), Breunig et al.(2000), etc, define outliers by using the full dimensional distances of the points from one and another, which will result in unexpected performance and qualitative costs due to the curse of dimensionality.

Secondly, clustering algorithms like ROCK (Guha, et al., 1999), $C^2P$ (Nanopoulos, et al., 2001), DBSCAN (Ester, et al., 1996) can also handle outliers, but their main concern is to find clusters, the outliers are often regarded as noises. And there are the following shortcomings in the initial work (Su, et al., 2001) that addressed clustering based outlier detection. Su, et al. (2001) only regarded *small* clusters as outliers and a measure for identifying the degree of each object being an outlier is not presented. For most of the data points in a dataset are not outliers, it is meaningful to identify only *top n* outliers. In this case, the method proposed by Su, et al. (2001) failed to fulfill this task effectively.

Finally, using the same process and functionality to solve both clustering and outlier discovery is highly desired. Such integration will be a great benefit to business users because they

* Corresponding author: Tel: +86-451-6414906 (ext. 8512). *Email address*: zengyouhe@yahoo.com (Z. He).

do not need to worry about the selection of different data mining algorithms. Instead, they can focus on data and business solution.

Based on the above observations, we present the new definition for outlier: *cluster-based local outlier.* A measure for identifying the physical significance of an outlier, namely *CBLOF (Cluster-Based Local Outlier Factor)*, is also defined. Finally, a fast algorithm for mining outliers is presented, whose effectiveness is verified by the experimental results.

Contributions of this paper are as follows:

- We propose a novel definition for outlier- *cluster-based local outlier,* which has great new intuitive appeal and numerous applications.
- A measure for identifying the degree of each object being an outlier is presented, which is called *cluster-based local outlier factor*.
- We present an efficient algorithm for mining *cluster-based local outliers* based on our definitions.

The remainder of this paper is organized as follows. Section 2 discusses previous work. In Section 3, we formalize our definition of *cluster-based local outlier*. Section 4 presents the algorithm for mining defined outliers. Experimental results are given in Section 5 and Section 6 concludes the paper.

## 2. Related Work

The statistics community conducted most of the previous studies on outlier mining (Barnett and Lewis, 1994). These studies can be broadly classified into two categories. The first category is *distribution-based*, where a standard distribution is used to fit the dataset. Outliers are defined based on the probability distribution. Yamanishi, Takeuchi and Williams (2000) used a Gaussian mixture model to present the normal behaviors and each datum is given a score on the basis of changes in the model. High score indicates high possibility of being an outlier. This approach has been combined with a supervised-based learning approach to obtain general patterns for outlier (Yamanishi and Takeuchi, 2001). The main problem with this method is that it assumes that the underlying data distribution is known a prior. However, for many applications, it is an impractical assumption. And, the cost for fitting data with standard distribution is significantly considerable.

*Depth-based* is the second category for outlier mining in statistics (Nuts and Rousseeuw, 1996). Based on some definition of depth, data objects are organized in convex hull layers in data space according to peeling depth, and outliers are expected to be found from data objects with shallow depth values. In theory, depth-based methods could work in high dimensional data space. However, due to relying on the computation of *k-d* convex hulls, these techniques have a lower bound complexity of $\Omega(N^{k/2})$, where $N$ is number of data objects and $k$ is the dimensionality of the dataset. This makes these techniques infeasible for large dataset with high dimensions.

*Distance-based* outlier is presented by Knorr and Ng (1998). A distance-based outlier in a dataset $D$ is a data object with *pct*% of the objects in $D$ having a distance of more than $d_{\min}$ away from it. This notion generalizes many concepts from distribution-based approach and enjoys better computational complexity. It is further extended based on the distance of a point from its $k^{th}$ nearest neighbor (Ramaswamy, et al., 2000). After ranking points by the distance to its $k^{th}$ nearest neighbor, the *top k* points are identified as outliers. Efficient algorithms for mining *top-k* outliers are given. Alternatively, in the algorithm proposed by Angiulli and Pizzuti (2002), the outlier

factor of each data point is computed as the sum of distances from its $k$ nearest neighbors. The above three algorithms define outliers by using the full dimensional distances of the points from one another. However, recent research results show that in high dimensional space, the concept of proximity may not be qualitatively meaningful (Beyer, et al., 1999). Therefore, the direct application of distance-based methods to high dimensional problems often results in unexpected performance and qualitative costs due to the curse of dimensionality.

*Deviation-based* techniques identify outliers by inspecting the characteristics of objects and consider an object that deviates these features as an outlier (Arning, et al., 1996).

Breunig, et al. (2000) introduced the concept of "*local outlie*r". The outlier rank of a data object is determined by taking into account the clustering structure in a bounded neighborhood of the object, which is formally defined as "*local outlier factor*" (*LOF*). Their notions of outliers are based on the same theoretical foundation of density-based clustering (Ester, et al., 1996). For the computation of "density" is relying on full dimensional distance between objects, in high dimensional space, problem exists in distance-based methods will be encountered again.

Clustering algorithms like ROCK (Guha, et al., 1999), $C^2P$ (Nanopoulos, et al., 2001), DBSCAN (Ester, et al., 1996) can also handle outliers, but their main concern is to find clusters, the outliers in the context of clustering are often regarded as noise. In general, outliers are typically just ignored or tolerated in the clustering process for these algorithms are optimized for producing meaningful clusters, which prevents giving good results on outlier detection. And there are the following shortcomings in the initial work (Su, et al., 2001; Yu, et al., 1999) that addressed clustering based outlier detection. Su, et al. (2001) only regarded *small* clusters as outliers and a measure for identifying the degree of each object being an outlier is not presented. For most of the data points in a dataset are not outliers, it is meaningful to identify only *top n* outliers.Hence, the method proposed by Su, et al. (2001) failed to fulfill this task effectively. Furthermore, how to distinguish small clusters form the rest is not addressed in their method. Yu, et al. (1999) introduced *FindOut*, a method based on wavelet transform, that identifies outliers by removing clusters from the original dataset.

Aggarwal and Yu (2001) discussed a new technique for outlier detection, which finds outliers by observing the density distribution of projections from the data. That is, their definition considers a point to be an outlier, if in some lower dimensional projection, it is present in a local region of abnormally low density.

The replicator neutral network (RNN) is employed to detect outliers by Harkins, et al. (2002). The approach is based on the observation that the trained neutral network will reconstruct some small number of individuals poorly, and these individuals can be considered as outliers. The outlier factor for ranking data is measured according to the magnitude of the reconstruction error.

An interesting recent technique finds outliers by incorporating semantic knowledge such as the class labels of each data point in the dataset (He, Deng and Xu, 2002). In view of the class information, a semantic outlier is a data point, which behaves differently with other data points in the same class.

## 3. Cluster-Based Local Outlier

In this section, we propose a new definition for outlier: *cluster-based local outlier*. Before formalizing the new definition for outliers, we first give an example to illustrate our basic ideas. Considering the 2-d data set in Figure 1. There are 4 clusters in this figure, $C_1$, $C_2$, $C_3$ and $C_4$.

Obviously, the data points in both $C_1$ and $C_3$ should be regarded as outliers and captured by proposed definitions. Intuitively, we call data points in $C_1$ and $C_3$ outliers because they did not belonge to the cluster $C_2$ and $C_4$. Thus, it is reasonable to define the outliers from the point of view of clusters and identify those data points that don't lie in any *large* clusters as outliers. Here, the number of data points in $C_2$ and $C_4$ are dominant in the data set. Furthermore, to capture the spirit of "*local*" proposed by Breunig, et al. (2000), the cluster-based outliers should satisfy that they are "*local*" to specified clusters. For example, data points in $C_1$ are "*local*" to $C_2$.

To identify the physical significance of the definition of an outlier, we assign to each object an outlier factor, namely *CBLOF (Cluster-Based Local Outlier Factor)*, which is measured by both the size of the cluster the object belongs to and the distance between the object and its closest cluster (if the object lies in a *small* cluster).

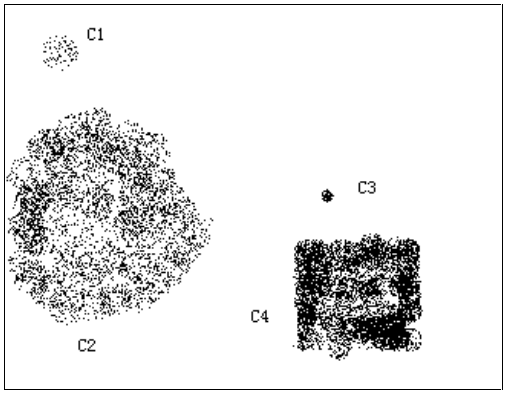


**Fig. 1.** 2-d data set DS1

Before we present the concept of cluster-based local outliers and design the measure for outlier factor, let's first look at the concept about clustering. Throughout the paper, we use $|S|$ to donate the size of $S$, where $S$ in general a set containing some elements.

**Definition 1:** Let $A_1, \ldots, A_m$ be a set of attributes with domains $D_1,\ldots, D_m$ respectively. Let the dataset $D$ be a set of records where each record $t$: $t \in D_1 \times \ldots \times D_m$. The results of a clustering algorithm executed on $D$ is denoted as: $C = \{C_1, C_2, \ldots, C_k\}$ where $C_i \cap C_j = \phi$ and $C_1 \cup C_2 \cup \ldots, \cup C_k = D$. The number of cluster is $k$.

Here, the clustering algorithm used for partitioning the dataset into disjoint sets of records can be chosen freely. The only requirement for the selected clustering algorithm is that it should have the ability to produce good clustering results.

A critical problem that must be solved before defining the *cluster-based local outlier* is how to identify whether a cluster is *large* or *small*. This problem is discussed in definition 2.

**Definition 2:** (*large* and *small* cluster) Suppose $C = \{C_1, C_2, \ldots, C_k\}$ is the set of clusters in the sequence that $|C_1| \geq |C_2| \geq \ldots \geq |C_k|$. Given two numeric parameters $\alpha$ and $\beta$, we define $b$ as the *boundary* of *large* and *small* cluster if one of following formulas holds.

$$\begin{cases} (|C_1|+|C_2|+ \ldots +|C_b|) \geq |D| * \alpha & (1) \\ |C_b|/|C_{b+1}| \geq \beta & (2) \end{cases}$$

Then, the set of *large* cluster is defined as: $LC = \{C_i \mid i \leq b\}$ and the set of *small* cluster is defined as: $SC = \{C_j \mid j > b\}$.

Definition 2 gives quantitative measure to distinguish *large* and *small* clusters. Formula (1) considers the fact that most data points in the data set are not outliers. Therefore, clusters that hold a large portion of data points should be taken as *large* clusters. For example, if $\alpha$ is set to 90%, we intend to regard clusters contain 90% of data points as *large* clusters. Formula (2) considers the fact that *large* and *small* clusters should have significant differences in size. For instance, it is easy to get that, if we set $\beta$ to 5, the size of any cluster in *LC* is at least 5 times of the size of the cluster in *SC*.

**Definition 3:** (*Cluster-Based Local Outlier Factor*) Suppose $C = \{C_1, C_2, \ldots, C_k\}$ is the set of clusters in the sequence that $|C_1| \geq |C_2| \geq \ldots \geq |C_k|$ and the meanings of $\alpha$, $\beta$, $b$, LC and SC are the same as they are formalized in Definition 2. For any record *t*, the *cluster-based local outlier factor of t* is defined as:

$$CBLOF(t) = \begin{cases} |C_i| * min(distance(t, C_j)), \text{ where } t \in C_i, C_i \in SC \text{ and } C_j \in LC \text{ for } j=1 \text{ to } b \\ |C_i| * (distance(t, C_i) \text{ where } t \in C_i \text{ and } C_i \in LC \end{cases} \quad (3)$$

From Definition 3, the *CBLOF* of a record is determined by the size of its cluster, and the distance between the record and its closest cluster (if this record lies in *small* cluster) or the distance between the record and the cluster it belongs to (if this record belongs to *large* cluster), which provides importance to the local data behavior.

For the computation of distance between the record and the cluster, it is sufficient to adopt the similarity measure used in the clustering algorithm.

# 4. Algorithm for Detecting Cluster-Based Local Outliers

With the outlier factor *CBLOF*, we can determine the degree of a record's deviation. In this section, we will describe our algorithm for detecting outliers according to Definition 3.

To compute *CBLOF (t)*, we need a clustering algorithm first. In this paper, the clustering algorithm used is the *Squeezer* algorithm (He, Xu and Deng, 2002), which can produce good clustering results and at the same time deserves good salability. For the process of mining outliers is tightly coupled with the clustering algorithm, we give an introduction on *Squeezer* algorithm.

## 4.1 The *Squeezer* Algorithm

Let $A_1, \ldots, A_m$ be a set of categorical attributes with domains $D_1, \ldots, D_m$ respectively. Let the dataset *D* be a set of tuples where each tuple *t*: $t \in D_1 \times \ldots \times D_m$. Let *TID* be the set of unique ID

of every tuple. For each $tid \in TID$, the attribute value for $A_i$ of corresponding tuple is represented as $tid.A_i$.

**Definition 4:** (*Cluster*) *Cluster* = $\{tid \mid tid \in TID\}$ is subset of *TID*.

**Definition 5:** Given a *Cluster C*, the set of different attribute values on $A_i$ with respect to *C* is defined as: $VAL_i\ (C) = \{tid.A_i \mid tid \in C\}$ *where* $1 \le i \le m$.

**Definition 6:** Given a *Cluster C*, let $a_i \in D_i$, the support of $a_i$ in *C* with respect to $A_i$ is defined as: $Sup\ (a_i) = |\{tid \mid tid.A_i = a_i\}|$.

**Definition 7:** (*Summary*) Given a *Cluster C*, the *Summary* for *C* is defined as:

$$Summary = \{VS_i \mid 1 \le i \le m\} \text{ where } VS_i = \{(a_i, Sup\ (a_i)) \mid a_i \in VAL_i\ (C)\}.$$

Intuitively, the *Summary* of a *Cluster* contains summary information about this *Cluster*. In general, each *Summary* will consists of *m* elements, where *m* is number of attributes. The element in *Summary* is the set of pairs of attribute values and their corresponding supports.

**Definition 8:** (*Cluster Structure, CS*) Given a *Cluster C*, the *Cluster Structure (CS)* for *C* is defined as: *CS = {Cluster, Summary}*.

**Definition 9:** Given a *Cluster C* and a tuple *t* with $tid \in TID$, the *similarity* between *C* and *tid* is defined as:

$$Sim\ (C, tid) = \sum_{i=1}^{m} \left( \frac{Sup(a_i)}{\sum_{a_j \in VAL_i(C)} Sup(a_j)} \right) \text{ where } tid.A_i = a_i.$$

From the definition 9, it is clear that the similarity used here is statistics based. In other words, if the similarity between a tuple and an existed cluster is large enough, it means that the probability of the tuple belongs to this cluster is larger. In the *Squeezer* algorithm, this measure is used to determine whether the tuple should be put into the cluster or not.

The *Squeezer* algorithm has *n* tuples as input and produce clusters as final results. Initially, the first tuple in the database is read in and a *Cluster Structure (CS)* is constructed with $C = \{1\}$. Then, the consequent tuples are read iteratively.

For every tuple, by the similarity function, we compute its similarities with all existing clusters, which are represented and embodied in the corresponding *CSs*. The largest value of similarity is selected out. If it is larger than the given threshold, donated as *s*, the tuple will be put into the cluster that has the largest value of similarity. The *CS* is also updated with the new tuple. If the above condition does not hold, a new cluster must be created with this tuple.

The algorithm continues until it has traversed all the tuples in the dataset. It is obvious that the *Squeezer* algorithm only makes one scan over the dataset, thus, highly efficient for disk resident datasets where the I/O cost becomes the bottleneck of efficiency.

The *Squeezer* algorithm is presented in Figure 2. It accepts as input the dataset *D* and the value of the desired similarity threshold. The algorithm fetches tuples from *D* iteratively.

Initially, the first tuple is read in, and the sub-function *addNewClusterStructure()* is used to establish a new *Clustering Structure*, which includes *Summary* and *Cluster* (Step 3-4).

For the consequent tuples, the similarity between an existed *Cluster C* and each tuple is computed using sub-function *simComputation()* (Step 6-7). We get the maximal value of similarity (donated by *sim_max*) and the corresponding index of *Cluster* (donated by *index*) from the above computing results (Step 8-9). Then, if the *sim_max* is larger than the input threshold *s*, sub-function *addTupleToCluster()* will be called to assign the tuple to selected *Cluster* (Step 10-11). If it is not the case, the sub-function *addNewClusterStructure()* will be called to construct

a new *CS* (Step 12-13). Finally, the clustering results will be labeled on the disk (Step 15).

```
Algorithm Squeezer (D, s)
Begin
1.   while (D has unread tuple){
2.       tuple = getCurrentTuple (D )
3.       if (tuple.tid = = 1){
4.            addNewClusterStructure (tuple.tid) }
5.       else{
6.            for each existed cluster C
7.                 simComputation(C,tuple)
8.            get the max value of similarity: sim_max
9.            get the corresponding Cluster Index: index
10.           if sim_max >= s
11.                addTupleToCluster(tuple, index)
12.           else
13.                addNewClusterStructure (tuple.tid) }
14.  }
15.  outputClusteringResult ()
End
```

**Fig. 2.** *Squeezer* Algorithm

Choosing the *Squeezer* algorithm as the background clustering algorithm for outlier detection in this literature is based on the consideration that this algorithm has the following novel features:

- ✓ It achieves both high quality of clustering results and scalability.
- ✓ Its ability for handling high dimensional datasets effectively.
- ✓ It does not require the number of desired clusters as an input parameter and it can produce more natural clusters with significant different sizes. This feature is undoubtedly important for discovering outliers defined in Section 3.

## 4.2 The *FindCBLOF* Algorithm

The algorithm *FindCBLOF* for detecting outliers is listed in Figure 3.

The algorithm *FindCBLOF* first partitions the dataset into clusters with *Squeezer* algorithm (Step 2-3). The sets of *large* clusters and *small* clusters, *LC* and *SC*, are derived using the parameters according to definition 2 (Step 4). Then, for every data point in the data set, the value of *CBLOF* is computed with definition 3 (Step 5-11).

**Algorithm** *Find**CBLOF***

**Input:** $D(A_1, \ldots, A_m)$, // the data set
The parameter $\alpha$ and $\beta$ // parameters

**Output:** *The values of CBLOF for all records* //indicates the degree of deviation

```
01 begin
02      Clustering the dataset D (A1, ..., Am,) using Squeezer algorithm
03      /* Produced clusters: C = {C1, C2, ..., Ck} and |C1|>=|C2|>= ...>=|Ck| */
04      Get LC and SC with the 2 parameters    //get the set of large and small clusters
05      foreach record t in the dataset do begin
06          if t belongs to Ci and Ci belongs to SC do begin
07              CBLOF= |Ci |*min (distance (t, Cj))      // Cj belongs to LC
08          else
09              CBLOF = |Ci |*distance(t, Ci)            // Ci belongs to LC
10          return CBLOF
11      end
12  end
```

**Fig. 3.** The *FindCBLOF* Algorithm

Algorithm *FindCBLOF* has two parts: 1) Clustering the dataset and 2) Computing the value of *CBLOF* for each record. The *Squeezer* determines the cost of part 1. From the execution process of *Squeezer*, it needs only one scan over the dataset. Thus, the cost of part 1 should be $O(N)$, where $N$ is number of the records in the dataset. As to the part 2, one scan over the dataset is also required. Therefore, the overall cost for the *FindCBLOF* algorithm is $O(N)$.

From the above analysis, we can see that linear scalability is achieved with respect to the size of dataset, which makes the *FindCBLOF* algorithm qualified for handling large dataset.

# 5. Experimental Results

A comprehensive performance study has been conducted to evaluate our algorithm. In this section, we describe those experiments and their results. We ran our algorithm on both real-life datasets obtained from the UCI Machine Learning Repository (Merz and Merphy, 1996) and synthetic datasets. Our algorithm was implemented in Java. All experiments were conducted on a PentiumⅢ-600 machine with 128 M of RAM and running Windows 2000 Server.

We used three real life datasets to demonstrate the effectiveness of our algorithms against other algorithms. For all the experiments, the two parameters needed by *FindCBLOF* are set to 90% and 5 separately. For the KNN algorithm (Ramaswamy, et al., 2000), the results were obtained using the *5-nearest-neighbour*; the results didn't change significantly when the parameter $k$ is specified to alternative values. Since the *Squeezer* algorithm operates on categorical dataset, the Annealing dataset are disretized using the automatic discretization functionality provided by the CBA (Liu, et al., 1998) software.

## 5.1 Annealing Data

The first dataset used is the Annealing data set, which has 798 instances with 38 attributes. The data set contains a total of 5 (non-empty) classes. Class 3 has the largest number of instances. The remained classes are regarded as rare class labels for they are small in size. The corresponding class distribution is illustrated in Table 1.

**Table 1** Class Distribution of Annealing Data Set

| Case | Class codes | Percentage of instances |
|---|---|---|
| Commonly Occurring Classes | 3 | 76.1% |
| Rare Classes | 1, 2, 5, U | 23.9% |

As pointed out by Aggarwal and Yu (2001), one way to test how well the outlier detection algorithm worked is to run the method on the dataset and test the percentage of points which belong to the rare classes. If outlier detection works well, it is expected that the rare classes would be over-represented in the set of points found. These kinds of classes are also interesting from a practical perspective.

Table 2 shows the results produced by the *FindCBLOF* algorithm against the KNN algorithm (Ramaswamy, et al., 2000). Here, the *top ratio* is ratio of the number of records specified as *top-k* outliers to that of the records in the dataset. The *coverage* is ratio of the number of detected rare classes to that of the rare classes in the dataset. For example, we let *FindCBLOF* algorithm find the *top 175* outliers with the top ratio of 25%. By examining these 175 points, we found that 105 of them belonged to the rare classes. In contrast, when we ran the KNN algorithm on this dataset, we found that only 58 of 175 top outliers belonged to rare classes.

**Table 2** Detected Rare Classes in Annealing Dataset

| Top Ratio (Number of Records) | Number of Rare Classes Included (Coverage) | |
|---|---|---|
| | ***FindCBLOF*** | ***KNN*** |
| 10% (80) | 45 (24%) | 21 (11%) |
| 15%(105) | 55 (29%) | 30 (16%) |
| 20%(140) | 82 (43%) | 41 (22%) |
| 25%(175) | 105 (55%) | 58 (31%) |
| 30%(209) | 105 (55%) | 62 (33%) |

From Table 2, the performance of the *FindCBLOF* algorithm outperformed that of the KNN algorithm in all the five cases, especially, when the *top ratio* is relative small, the *FindCBLOF* algorithm worked much better.

## 5.2 Lymphography Data

The second dataset used is the Lymphography data set, which has 148 instances with 18 attributes. The data set contains a total of 4 classes. Classes 2 and 3 have the largest number of instances. The remained classes are regarded as rare class labels for they are small in size. The corresponding class distribution is illustrated in Table 3.

**Table 3** Class Distribution of Lymphography Data Set

| Case | Class codes | Percentage of instances |
|---|---|---|
| Commonly Occurring Classes | 2, 3 | 95.9% |
| Rare Classes | 1, 4 | 4.1% |

Table 4 shows the results produced by the *FindCBLOF* algorithm against the KNN algorithm. In this experiment, the *FindCBLOF* algorithm can find all the records in rare classes when the *top ratio* reached 20%. Moreover, it can find majority of the records in the rare classes even the *top ratio* is set to relative small. In contrast, the performance of KNN algorithm is not satisfied.

**Table 4:** Detected Rare Classes in Lymphography Dataset

| Top Ratio (Number of Records) | Number of Rare Classes Included (Coverage) | |
|---|---|---|
| | ***FindCBLOF*** | ***KNN*** |
| 5% (7) | 4 (67%) | 1 (17%) |
| 10%(15) | 4 (67%) | 1 (17%) |
| 15%(22) | 4 (67%) | 2 (33%) |
| 20%(30) | 6 (100%) | 2 (33%) |

## 5.3 Wisconsin Breast Cancer Data

The third dataset used is the Wisconsin breast cancer data set, which has 699 instances with 9 attributes. Each record is labeled as *benign* (458 or 65.5%) or *malignant* (241 or 34.5%). We follow the experimental technique of Harkins, et al. (2002) by removing some of the *malignant* records to form a very unbalanced distribution; the resultant dataset had 39 (8%) *malignant* records and 444 (92%) *benign* records[1].

For this dataset, our aim is to test the performance of our algorithm with the KNN algorithm and the RNN based outlier detection algorithm (Harkins, et al., 2002). The results of RNN based outlier detection algorithm on this dataset are reported from Harkins, et al. (2002).

Table 5 shows the results produced by the *FindCBLOF* algorithm against both the KNN algorithm and the RNN based outlier detection algorithm.

One important observation from Table 5 was that our algorithm performed the best for all cases and never performed the worst. That is to say, the *FindCBLOF* algorithm is more capable to effectively detect outliers than the other two algorithms.

Another important observation was that the *FindCBLOF* algorithm found all the *malignant* records with the *top ratio* at 16%. In contrast, for the RNN based outlier detection algorithm, it achieved this goal with the *top ratio* at 28%, which is almost the twice for that of *FindCBLOF*.

In summary, the above experimental results on the three datasets show that the *FindCBLOF* algorithm can outliers more efficiently than existing algorithms, from which we can confidently assert that the new concept of *cluster-based local outlier* is promising in practice.

[1] The resultant dataset is public available at: http://research.cmis.csiro.au/rohanb/outliers/breast-cancer/

**Table 5** Detected Malignant Records in Wisconsin Breast Cancer Dataset

| Top Ratio (Number of Records) | Number of Malignant Included (Coverage) | | |
|---|---|---|---|
| | ***FindCBLOF*** | ***RNN*** | ***KNN*** |
| 0% (0) | 0 (0.00%) | 0 (0.00%) | 0 (0.00%) |
| 1%(4) | 4 (10.26%) | 3 (7.69%) | 4 (10.26%) |
| 2%(8) | 7 (17.95%) | 6 (15.38%) | 4 (10.26%) |
| 4%(16) | 14 (35.90%) | 11 (28.21%) | 9 (23.80%) |
| 6%(24) | 21 (53.85%) | 18 (46.15%) | 15 (38.46%) |
| 8%(32) | 27 (69.23%) | 25 (64.10%) | 20 (51.28%) |
| 10%(40) | 32 (82.05%) | 30 (76.92%) | 21 (53.83%) |
| 12%(48) | 35 (89.74%) | 35 (89.74%) | 26 (66.67%) |
| 14%(56) | 38 (97.44%) | 36 (92.31%) | 28 (71.79%) |
| 16%(64) | 39 (100.00%) | 36 (92.31%) | 28 (71.79%) |
| 18%(72) | 39 (100.00%) | 38 (97.44%) | 28 (71.79%) |
| 20%(80) | 39 (100.00%) | 38 (97.44%) | 28 (71.79%) |
| 25%(100) | 39 (100.00%) | 38 (97.44%) | 28 (71.79%) |
| 28%(112) | 39 (100.00%) | 39 (100.00%) | 28 (71.79%) |

## 5.4 Scalability Tests

In this section, we evaluate the performance of the computation of *CBLOF*. The datasets were generated using a data generator, in which all possible values are produced with (approximately) equal probability, and the number of attribute values for each attribute set to 10. To find out how the number of records and attributes affect the algorithm, we ran a series of experiments with increasing number of records and attributes. The number of records varied from 0.1 million to 0.5 million, and the number of attributes varied from 10 to 40. The elapsed time measured is shown in Figure 4.

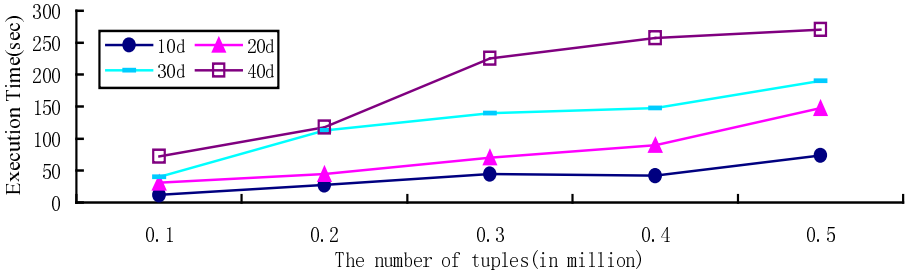


**Fig. 4.** Runtime for the computation of *CBLOFs* with different dataset sizes and different number of attributes

From this figure, we can see that linear scalability is achieved with respect to the number of records and attributes. This property qualifies our algorithm for discovering outliers in very large databases.

## 6 Conclusions

In this paper, we present a new definition for outlier: *cluster-based local outlier*, which is intuitive and provides importance to the local data behavior. A measure for identifying the physical significance of an outlier, namely *CBLOF (Cluster-Based Local Outlier Factor)*, is also defined. Furthermore, we propose the *FindCBLOF* algorithm for discovering outliers. The experimental results show that our approach outperformed existing methods on identifying meaningful and interesting outliers.

For future work, we will integrate the *FindCBLOF* algorithm more tightly with clustering algorithms to make the detecting process more efficient. The designing of effective *top-k* outliers' detection algorithm will be also addressed.

## Acknowledgements

The comments and suggestions from the anonymous reviewers greatly improve this paper. The authors wish to thank Fabrizio Angiulli for sending us their PKDD'02 paper. Special acknowledgement goes to Mbale Jameson for his help on English editing. The National Nature Science Foundation of P. R. China (No. 60084004) and the IBM SUR Research Fund supported this research.

## References

Aggarwal，C., Yu， P.，2001. Outlier detection for high dimensional data. In: Proceedings of SIGMOD'01, Santa Barbara, CA, USA, pp. 37-46.

Angiulli, F., Pizzuti, C., 2002. Fast outlier detection in high dimensional spaces. In: Proceedings of PKDD'02.

Arning, A., Agrawal, R., Raghavan, P., 1996. A linear method for deviation detection in large databases. In: Proceedings of KDD'96, Portland OR, USA, pp. 164-169.

Barnett, V., Lewis, T., 1994. Outliers in statistical data. John Wiley and Sons, New York, 1994.

Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U., 1999. When is "nearest neighbors" meaningful? In: Proceedings of ICDT'99, Jerusalem, Israel, pp. 217-235.

Breunig, M. M., Kriegel, H. P., Ng, R. T., Sander, J., 2000. LOF: identifying density-based local outliers. In: Proceedings of SIGMOD'00, Dallas, Texas, pp. 427-438.

Ester, M., Kriegel, H. P., Sander, J., Xu, X., 1996. A density-based algorithm for discovering clusters in large spatial databases". In: Proceedings of KDD'96, Portland OR, USA, pp. 226-231.

Guha, S., Rastogi, R., Kyuseok, S., 1999. ROCK: A robust clustering algorithm for categorical attributes. In: Proceedings of ICDE'99, Sydney, Australia, pp. 512-521.

Harkins, S., He, H., Willams, G. J., Baster, R. A., 2002. Outlier detection using replicator neural networks. In: Proc. of the 4th International Conference on Data Warehousing and Knowledge Discovery, Aix-en-Provence, France, pp. 170-180.

He, Z., Deng, S., Xu, X., 2002. Outlier detection integrating semantic knowledge. In: Proc. of the 3th Int'l Conf. on Web-Age Information Management, Beijing, China, pp.126-131.

He, Z., Xu, X., Deng, S., 2002. Squeezer: An efficient algorithm for clustering categorical data. Journal of

Computer Science and Technology, 17(5), 611-625.

Jiang, M. F., Tseng, S. S., Su, C. M., 2001. Two-phase clustering process for outliers detection. Pattern Recognition Letters, 22(6/7), 691-700.

Knorr, E. M., Ng, R. T., 1998. Algorithms for mining distance-based outliers in large datasets. In: Proceedings of VLDB'98, New York, USA, pp. 392-403.

Liu, B., Hsu, W., Ma, Y., 1998. Integrating classification and association rule mining. In: Proceedings of KDD'98, New York, USA, pp. 80-86.

Merz, C. J., Merphy, P., 1996. UCI Repository of Machine Learning Databases. URL: Http://www.ics.uci.edu/~mlearn/MLRRepository.html.

Nanopoulos, A., Theodoridis, Y., Manolopoulos, Y., 2001. $C^2P$: Clustering based on closest pairs. In: Proceedings of VLDB'01, Rome Italy, pp. 331-340.

Nuts, R., Rousseeuw, P., 1996. Computing depth contours of bivariate point clouds. Journal of Computational Statistics and Data Analysis, vol.23, 153-168.

Ramaswamy, S., Rastogi, R., Kyuseok, S., 2000. Efficient algorithms for mining outliers from large data sets. In: Proceedings of SIGMOD'00, Dallas, Texas, pp. 93-104.

Yamanishi, K., Takeuchi, J., 2001. Discovering outlier filtering rules from unlabeled data-combining a supervised learner with an unsupervised Learner. In: Proceedings of KDD'01, pp. 389-394.

Yamanishi, K., Takeuchi, J., Williams, G., 2000. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. In: Proceedings of KDD'00, Boston, MA, USA, pp. 320-325.

Yu, D., Sheikholeslami, G., Zhang, A., 1999. FindOut: Finding out outliers in large datasets. Technique Report, State University of New York at Buffalo, 1999.