DEWS2007 M4-4

Example-Based DB-Outlier Detection from High Dimensional Datasets

Yuan LI† and Hiroyuki KITAGAWA†,††

† Graduate School of Systems and Information Engineering
†† Center for Computational Sciences
University of Tsukuba
Tennoudai 1–1–1, Tsukuba, Ibaraki, 305–8573 Japan

Abstract Outlier detection is an important problem that has applications in many fields. High dimensional datasets are common in such applications. Among the existing outlier detection methods, Distance-Based outlier (DB-Outlier) detection is one of the most generalizable and simplest approaches. It finds outliers by calculating distances between data points. However, in high dimensional space, data distribution is sparse, so every data point becomes a good outlier candidate. It has been shown that meaningful outliers are likely to be identified by examining the behavior of the data in low dimensional projections. On the other hand, Example-Based outlier detection method is promising in discovering the hidden user view of outliers. In this paper, we present a new method to detect DB-Outliers in high dimensional datasets based on user examples. The method finds a subspace where user examples are outstanding more significantly than in any other subspaces, and reports outliers detected in this subspace.

Key words Outlier, DB-Outlier, High-dimensional Data, Example

1. Introduction

Nowadays, data mining is becoming more and more important in our life. Outlier detection is an indispensable segment of data mining in many applications such as financial analysis, fraud detection, and network robustness analysis. In such applications, outliers often contain more useful information which may lead to some serious incidents than the normal data. Therefore, outlier detection problems are attracting increasing attention.

An outlier is defined as an abnormal object which is greatly different from the rest of the data. Existing researches try to define algorithms to detect outliers based on distance or density. Examples of existing techniques for detecting outliers are as follows:

(1) *Clustering-Based Method.*
(2) *Distance-Based Method* [3].
(3) *Density-Based Method* [7].
(4) *Subspace-Based Method* [5], [9].

These approaches detect outliers by calculating the distance or density of data points. Among these methods, Distance-Based method (DB-Outlier detection) is one of the simplest and most commonly used approaches, since it only calculates the distance between two data points; moreover, distance is a commonly used standard in many knowledge and technical areas.

Many real applications process high dimensional datasets whose dimensionality may be 10, 15 or even more. It is very difficult to find abnormal objects in high dimensional spaces directly, because the distribution of data points in high dimensional spaces is not easy to be imagined. As a matter of fact, data distribution is sparse in high dimensional spaces, so it becomes complicated to distinguish objects with distance or density. Consequently, algorithms based on distance or density lose their significance when dealing with high dimensional datasets [6].

It has been shown that meaningful outliers are likely to be defined by examining the behavior of the data in low dimensional projections [5]. It implies that studying the behavior of data in subspaces is helpful for detecting outliers in high dimensional datasets.

On the other hand, most existing outlier detection methods are designed to detect outliers with parameter values decided by users in advance. These parameter values always contain some hidden user view of outliers. Actually, it is not easy for users to determine such parameter values which can help them detect outliers of their interests. However, users are often experts in their problem domains, so they usually have some outlier examples in hand, and want to find more objects that exhibit " outlier-ness " characteristics similar to these examples.

An example-Based method is shown to be promising in

discovering the hidden user view of outliers in [1]. With the help of examples, users can be released from deciding parameter values in advance. An Example-Based method detecting outliers in high dimensional datasets has also been presented to be a worthwhile approach in the paper [2]. The method proposed in this paper takes advantage of Sparsity-Based method [5] to detect outliers based on user examples. Sparsity-Based method employs meshes for deciding outliers. All points in the same cell are regarded as normal objects or outliers with the effect of meshes. Therefore the method has a problem that sometimes normal objects may be detected as outliers, and vice versa.

In this paper, we propose a new method which makes a combination of DB-Outlier detection and Example-Based algorithms for detecting outliers in high dimensional datasets with user examples.

2. Distance-Based Outlier

2.1 Basic Concept

The notion of DB-Outlier studied here has the same definition with Knorr and Ng's work [3]:

An object O in a dataset T is a DB(p, D)-outlier if at least fraction p of the objects in T lie greater than distance D from O.

It is obvious that the concept of DB-Outlier is well defined for any dimensional dataset. The parameter p is the minimum fraction of objects in a data space that must be outside an outlier's D-neighborhood. For ease to calculate, we employ another parameter M[M=N(1-p), N: data size] to represent the maximum portion of data points within an outlier's D-neighborhood. It means that an outlier needs to have less than M objects within its D-neighborhood.

In order to clarify the definition of DB-Outlier, we give an example illustrated in Figure 1.

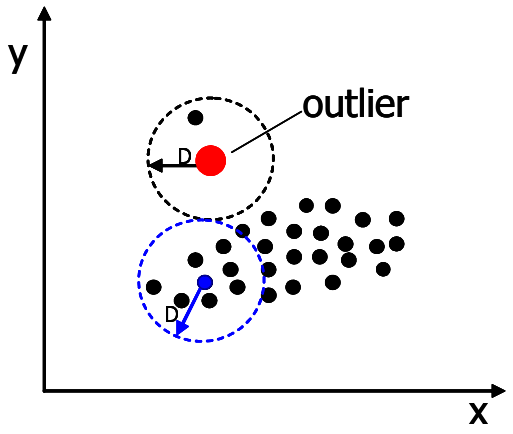

Figure 1 There are 30 points, and p=0.9, D equal to the arrow length drawn in the figure.

This example describes the data distribution of a 2-D(two dimensional) space. The dotted circles in the figure stand for D-neighborhoods. Since the parameter p equals to 0.9 and data size N is 30, if there is no more than M[M=30*(1-0.9)=3] points in a circle, the central point is decided as an outlier. Since there are only two points in the circle above, the big point is an outlier.

2.2 Detection Algorithm

The problem of detecting DB-Outliers in a data space can be solved by examining the nearest neighborhood or range query centered at each object. As soon as we find more than M data points within an object's D-neighborhood, this object will be marked as a non-outlier. There are two approaches for mining DB-Outliers in large datasets. One is the simple algorithm based on the definition of DB-Outlier, and the other is the Cell-Based algorithm [3].

- **Simple Algorithm**

This algorithm operates based on the definition of DB-Outlier. It calculates distances between the examined object and the rest. When more than M data points are found within an object's D-neighborhood, this object will be marked as a non-outlier and the examination start to check another object. If the examination finds no more than M data points within the D-neighborhood of an object after checking all the distances, this object will be reported as an outlier.

- **Cell-Based Algorithm**

The Cell-Based algorithm [3] utilizes cell structure's properties to rule out non-outliers quickly. It quantizes all the data objects into a space that has been partitioned into cells or squares. These cells or squares have a particular size which correlates with parameter D, so we need only count the number of data points in each cell. The cell structure's properties are helpful in excluding many objects from outlier candidates. The processing time of the Cell-Based method's is linear with data size. However, its performance is worse than the simple method when the dimensionality of a dataset is more than 4 [3], because it will take much quantizing time if the dimensionality is high.

3. Proposed Method

In this section, we discuss our proposed method which can detect outliers in high dimensional datasets. We try to make good use of Distance-Based and Example-Based methods for discovering a low dimensional subspace where user examples are isolated more significantly than in other subspaces. Objects which have the similar " outlier-ness " characteristics to user examples detected in this special subspace will be recognized as outliers.

There is another problem needed to be solved in searching

the most suitable subspace. Before examining all subspaces whose dimensionalities vary from 1 to total dimensions, we are not sure which one is the best. If the full dimensionality of a dataset is d, and the dimensionality of the most suitable subspace is k, $\binom{d}{k}$ combination candidates should be checked. Since we have no idea of k (the dimensionality of the most suitable subspace), we have to examine $\sum_{k=1}^{d}\binom{d}{k}$ possible combinations. It is fatiguable or even impossible to discover such desired subspace by examining all combination candidates. If the dimensionality of a dataset is very high, the brute force method which checks all combination candidates will not work.

For the above reason, we employ a GA(Genetic Algorithm) [8] to discover the most suitable subspace where user examples are outstanding significantly in a short time.

3.1 Overview of Genetic Algorithm

Genetic Algorithms simulate processes in natural systems and follow the principles of survival of the fittest like Charles Darwin's theory of evolution.

It makes a competition among a population of evolving problem solutions. Three phases in a GA, namely selection, crossover, and mutation, are combined to find the best solution of a problem. A " Fitness Value " function is used to evaluate each solution, and solutions that have a larger Fitness Value will generate more copies. It selects some solutions as parents, and makes a crossover to bring forth better children which have larger Fitness Values. Mutation creates new solutions that may be superior to their parents by randomly reversing the values of some positions in chromosomes. The algorithm undergoes the three steps repeatedly until it reaches the convergence criterion.

3.2 Procedure of Proposed Method

The procedure of our proposed method consists of the following three steps:

(1) **Detecting Most Suitable Subspace with GA**
At first, a Genetic Algorithm is used to find a subspace where user examples are isolated significantly than in any other subspaces.

(2) **Parameter Selection**
In the most suitable subspace, some pairs of parameters p and D are selected to separate examples from normal data points. With these pairs of (p, D) we can detect objects that have similar " outlier-ness" characteristics to user examples.

(3) **Outlier Report**
With different pairs of (p, D), we may get different outliers in the same subspace. Therefore, we do not only report outliers detected in the most suitable subspace, but also report the " outlier-ness " degree of each outlier.

3.3 Detecting Most Suitable Subspace with GA

With the help of GA, we can discover the best subspace in shortter time.

(1) **Calculation of Fitness Value**

A. Fitness Value

Since users want to find objects that have similar " outlier-ness "characteristics to their examples, most user examples should be recognized as outliers in right subspaces. In these right subspaces, user examples (outliers) show different appearances from the normal objects. For the normal objects, as distance D become longer, the number of data points within their D-neighborhood will increase quickly, because the density of a normal data is not low. In contrast, the number of neighbors within an outlier's D-neighborhood grows slowly before the distance D reaches large groups of data points. Let Nm denote the number of data points within an object's D-neighborhood. When we draw the Distance-Nm lines of objects, user examples and normal objects are clearly separated in right subspaces as shown in Figure 2.

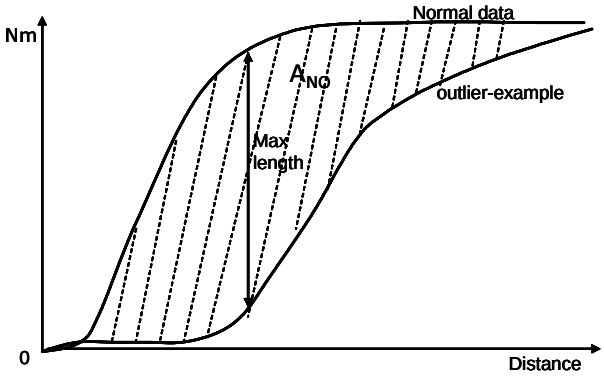

Figure 2 Different appearance between user examples and normal objects.
A_{NO}: Area circled by D-Nm lines of user examples and normal objects.

In other words, the more user examples are isolated significantly, the larger the area surrounded by D-Nm(Distance-Nm) lines of user examples and normal objects (denoted by A_{NO}) is. Actually, we want to find a subspace of lower dimension in which user examples show abnormal behavior. Consequently, we define Fitness Value of a subspace as follows:

$$f = \frac{A_{NO}}{k}$$

A_{NO}: the area surrounded by D-Nm lines of user examples and normal objects.
k: dimensionality of a dataset.

Since users may provide several examples, there are some D-Nm lines of user examples in one D-Nm space. In order

to make sure that every example can be recognized as an outlier in right subspaces, we only use the maximum value of these D-Nm lines of examples (Figure 3).

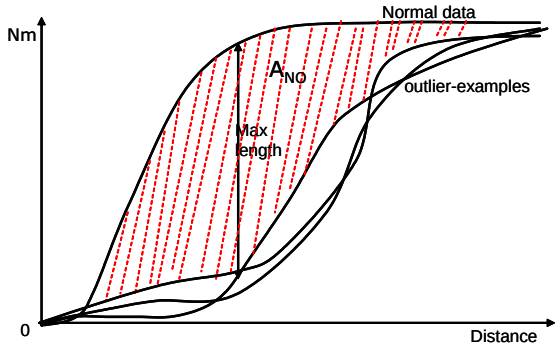

Figure 3 Selection of Nm lines of user examples

B. Actual Calculation

In a dataset, different attributes have different value scope, so we need to normalize each attribute of a dataset to a certain range such as 0∼1. In a normalized subspace whose dimensionality is k, the distance between two data points is by far $\sqrt{k}$, as we take a metric distance function ($distance = \sqrt{(x_1-y_1)^2+(x_2-y_2)^2+...+(x_n-y_n)^2}$). We uniformly select the same number of distance points in each subspace solution, and the sum of gaps between D-Nm lines of normal data and user examples at each distance point is used to calculate the value of A_{NO}. If the Nm value of a normal data is smaller than user examples, the gap value at this distance point is 0. Figure 4 shows an example to explain the calculation. In this example, $A_{NO} = gap_{D1} + gap_{D2} + gap_{D3} + gap_{D4} + ... + gap_{Dm-2} + gap_{Dm-1} + gap_{Dm} = 0 + gap_{D2} + gap_{D3} + gap_{D4} + .. + gap_{Dm-2} + 0 + 0$.

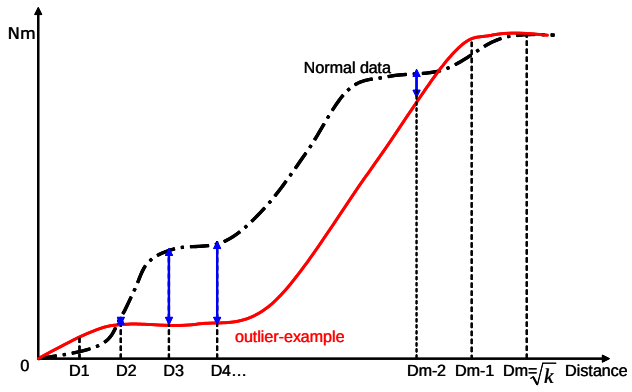

Figure 4 In a k dimensional subspace, unformly select m distance points. Only the double arrow lengths need to be added for calculating A_{NO}.

For the purpose of picking out normal data, we randomly select g objects from a dataset and calculate A_{NO} for each selected object. If an outlier is included in the selected g objects, the A_{NO} of this infiltrated data must be much smaller than the rest in right subspaces. As the number of outliers is very small, the probability of randomly selecting an outlier is actually low. Therefore, we just use the middle value of A_{NO} among the selected g objects in order to avoid such special cases. For confirmation, we do the same work q times, and calculate the average A_{NO}. Thus, the Fitness Value Function can be reformed as follows:

$$f = \frac{\overline{A_{NO}}}{k} = \frac{\sum_{n=1}^{q} A_{NO}(n)}{qk}$$

(2) **Selection**

A solution standing for a subspace is like a chromosome. Every solution is represented by a binary code string, and each position of a solution is defined as a dimension of a dataset. A subspace is composed of those dimensions whose position values are 1 in the solution string. For example, a code string 011000 denotes a 2-D subspace which is constructed by the second and the third dimensions of a 6-D dataset.

Let the population size be Z. In this process, we randomly select Z solutions for the next generation. A rank selection mechanism is used to choose some parents that have better Fitness Value for crossover, since the rank selection mechanism is often more stable [2]. All the chosen solutions are ranked in descending order by comparing their Fitness Values. We can draw a line whose length equals to the sum of the Z solutions'Fitness Values, so each solution occupies a part of the line and the length of this part corresponds to the solution's Fitness Value. The length of the Fitness Value line can be calculated by this function: $L = f_1 + f_2 + \cdots + f_Z$. It is clear that a solution which has a bigger Fitness Value will take a longer section in this line. Except the first step, the mechanism moves along the line with equivalent steps whose value equal to $\frac{L}{Z+1}$. The first step size is a random number less than the step size. That is just to make sure that solutions with better Fitness Value will be selected easily. The mechanism stops at a certain section with each step and the solution standing for this section will be selected as a parent. The selection mechanism operates as shown in Figure 5.

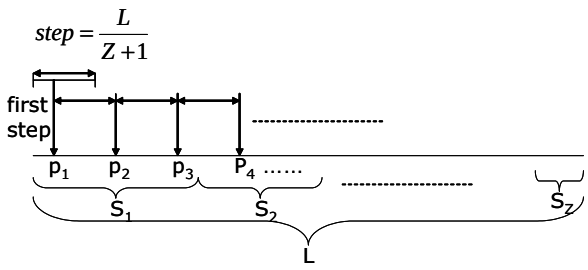

Figure 5 Description of selection mechanism

(3) **Crossover**

Crossover is a momentous phase of GAs in creating a better child by combining parent solutions. Sometimes GAs have unsuitable performances for their worse combinations. Therefore, we use an optimized crossover to bring birth only better children.

There are three kinds of positions in each pair of parents. Type 1: both parent strings have 1 in the same positions, Type 2: both parent strings have 0 in the same positions, and the remainder is type 3. It is obvious that child string must keep the same value as parents in positions of types 1 and 2. Hence our objective is to find an ideal child among the possible combinations of positions in type 3. As the Fitness Value f is decreasing with dimension k(namely the larger the dimensionality of a solution is the smaller the Fitness Value is), we begin to examine child strings from combinations of low dimensionality. It is similar to the case treated in the paper [2]. We give the algorithm of their optimized crossover shown in Figure 6.

```
Input:
  Two parent strings: p1, p2
Output:
  A child string: child
Algorithm:
begin
  T1 := Set of positions where p1 = p2 = 1;
  T2 := Set of positions where p1 ≠ p2;
  s := Solution(T1);
  while s is worse than p1 and p2 do begin
    for each V_i ∈ T2
      Q_i := Solution(V_i ∪ T1);
    s := Best solution in all the Q_i;
    T2 := T2-(1positions(s)-T1);
    T1 := 1positions(s);
    end;
  child :=s;
  return child;
end
Algorithm: Solution(T)
begin
  return A solution whose values of positions
    in T are 1 and 0 for the rest.
end
Algorithm: 1positions(s)
begin
  return A set of positions where the values are 1 in s;
end
```

Figure 6 Optimized crossover algorithm [2].

(4) **Mutation**

We employ a simple uniform mutation algorithm to make the mutation. In this algorithm every position of a child string has a probability of being inversed. For example, if the probability is 0.001, we can select a random number from 1∼1000, and invert the value of a position if the random number is 1, otherwise the position will keep its original value.*(We also set the probability 0.001 in our experiments.)*

(5) **Convergence Criterion**

The GA repeats processes of selection, crossover, and mutation until it achieves the convergence goal of the population. The convergence criterion is defined as the state that $y\%$ of the population has the same Fitness Value. The solution that has the largest Fitness Value in the convergent population indicates the most suitable subspace in our proposed method.

3.4 Parameter Selection

As soon as we discover the most suitable subspace, we select some pairs of parameters (p,D) to separate outliers from normal objects. A pair of parameters (p,D) can decide a point in the D-Nm[in this selection of parameters Nm=M=N(1-p)] space. According to the definition of DB-Outlier, objects whose D-Nm lines pass under this point detected as outliers based on this pair of parameters (p,D). Otherwise, the objects are normal data. We randomly select some parameters D between the two max/2 lines(half of the max length shown in Figure 7), and then trace out vertical lines with the selected Ds in the D-Nm space. We choose a random point on every vertical line between D-Nm lines of user examples and normal objects. Since we can calculate p by formula: $p = 1 - Nm/N$, every random point on these vertical lines represents a pair of (p,D). The selection of parameters (p,D) is carried out like Figure 7. In this figure, objects whose D-Nm lines pass under the black point are outliers with respect to parameters $(p1,D1)$. With different pairs of parameters (p,D), we will detect different outliers.

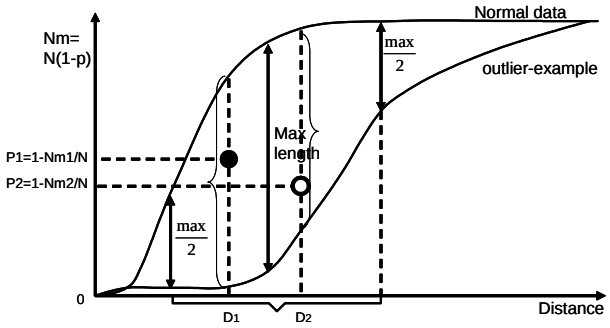

Figure 7 Selection of parameters(p,D)

With these selected parameters we can detect DB-Outliers in the most suitable subspace.

3.5 Outlier Report

At last we report outliers and their " outlier-ness " degrees in the most suitable subspace. As we have already explained, with different parameters we may get different outliers in the same subspace. Thus, we do not only report the detected outliers but also report their " outlier-ness " degrees. To ex-

plain the " outlier-ness " degree, we give an example shown in Figure 8. 3 pairs of (p,D) are selected in this most suitable subspace. Outlier1 is detected as an outlier three times with these parameters, outlier2 is detected as an outlier only one time, and outlier3 is detected two times. Therefore, the " outlier-ness " degrees of outlier1, outlier2, and outlier3 are 100%, 33.33%, and 66.67%, respectively.

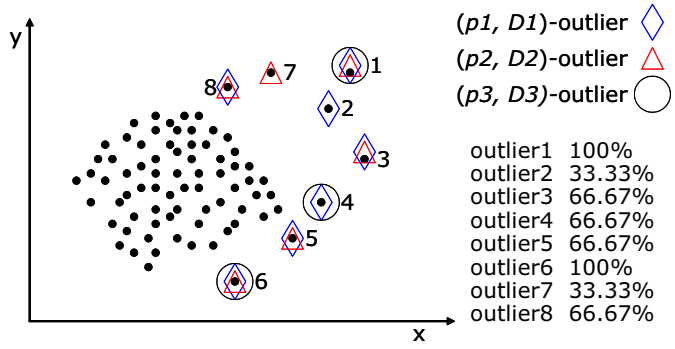

Figure 8 Calculation of "outlier-ness" degree

4. Experiment and Results

We test our proposed method on both synthetic and real datasets. The real dataset contains abalone data obtained from the UCI machine learning repository [4]. In the synthetic dataset, we implant a group of outliers in the 1^{st}-2^{nd} subspace(see the detail in Table 1).

Table 1 Description of synthetic and real datasets. Dim denotes dimensionality of the dataset

Dataset	Dim	Description
Synthetic Data	16	20,012 data which are uniformly distributed in 14 dimensions and holding one set of outliers in the remainder 2-dimensional subspace.
Abalone Data	8	Abalone data, obtained from the UCI machine learning repository, 4177 examinations of abalones with 8 attributes.

We choose 3(g=3) objects to calculate A_{NO}, and do the same calculation 5(q=5) times. Then we get an average value of A_{NO}. After we discover the most suitable subspace with GA, we select 10 pairs of parameters (p,D) to detect objects that have similar " outlier-ness " characteristics to user examples. Since the Cell-Based method is by far the better for k$\leq$4 [3], we take advantage of the Cell-Based method to detect outliers if the dimensionality of the most suitable subspace is lower than 5, otherwise we use the simple method.

The distribution of data points with user examples in the 1^{st}-2^{nd} subspace is shown in Figure 9. The same examples give normal appearances in other subspaces such as Figure 10 and Figure 11. Our proposed method detects the most suitable subspace (the 1^{st}-2^{nd} subspace) correctly in most of trials and report outliers detected in the most suitable subspace. The result is shown in Figure 12.

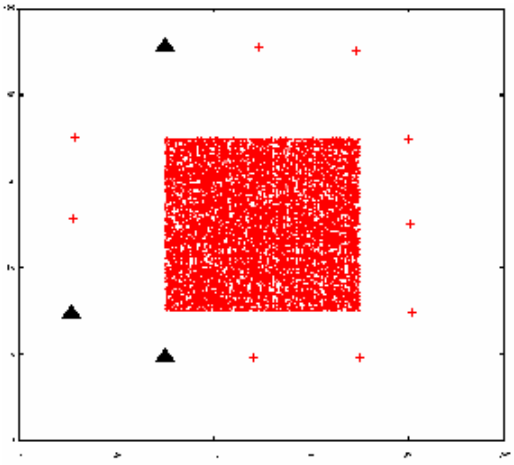

Figure 9 Synthetic data: Distribution of data points with user examples in 1^{st}-2^{nd} subspace

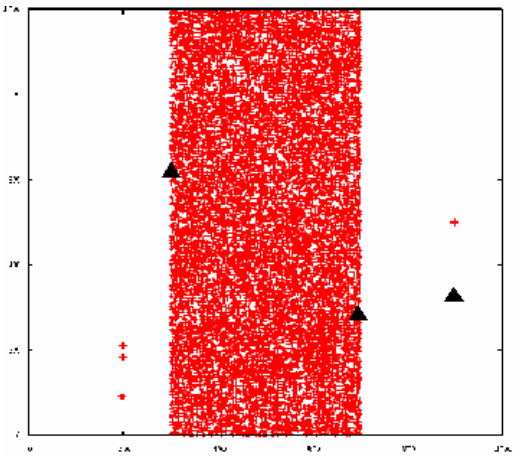

Figure 10 Synthetic data: Distribution of data points with user examples in $2^{nd} - 3^{rd}$ subspace

We find the most suitable subspace of the abalone dataset only seven times after doing the same experiment ten times. However, the failed results are correlated with the correct one. The dimensionality of the failed results contain the dimensionality of the correct subspace. It implies that there are some reasons causing the results inaccurate. The first one is the definition of Fitness Value function. The definition of the Fitness Value function may be not appropriate enough to evaluate solutions. The second reason is that user examples we used in our experiments are not only isolated in the most suitable subspace, but also in other subspaces. The data distribution of the abalone dataset is shown in Figure 13, Figure 14 and Figure 15.

We tested our proposed method on both synthetic and

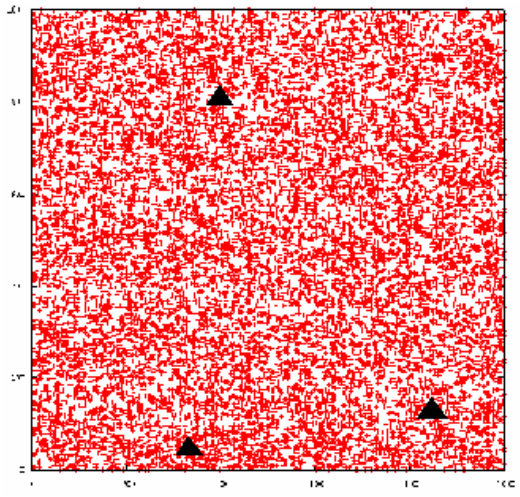

Figure 11 Synthetic data: Distribution of data points with user examples in $7^{th} - 10^{th}$ subspace

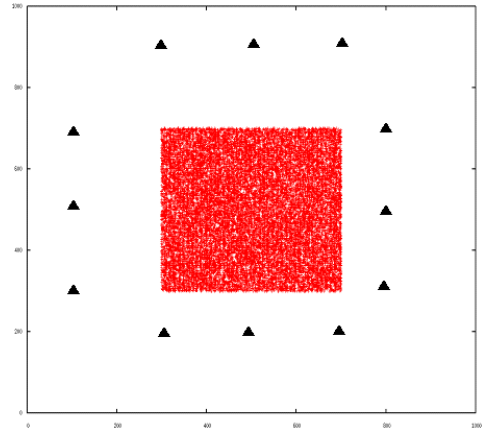

Figure 12 Synthetic data: Outliers ("outlier-ness" degree over 70%) in the most suitable subspace.

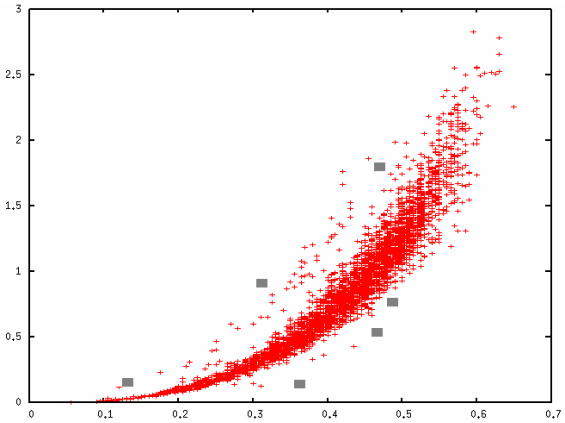

Figure 13 Abalone data: Distribution of data points with user examples in Diameter-Whole Weight subspace subspace

real datasets 10 times separately. The summary of our experiments is shown in Table 2. *p_Size* stands for the size of population, and *Acc* denotes the accuracy of the method to discover the most suitabel subspace within 10 times. Convergence means the convergence criterion of each dataset used in experiments, and Time shows the average time of finding the most suitable subspace. All of our experiments were run on a Microsoft Windows XP machine having 1GB of main memory.

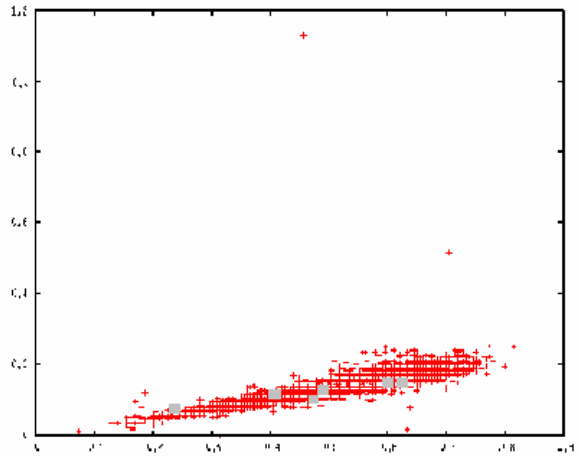

Figure 14 Abalone data: Distribution of data points with user examples in Length-Height subspace

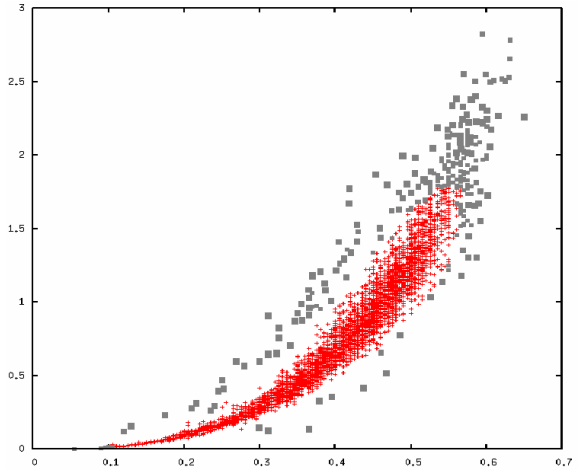

Figure 15 Abalone data: Outliers ("outlier-ness" degree over 70%) in the most suitable subspace.

Table 2 Performance of the proposed method.

Dataset	p_Size	Trials	Accu	Convergence	Time(s)
Synthetic Data	120	10	100%	80%	1070
Abalone Data	50	10	70%	90%	121

5. Conclusions and Future Work

We discussed a new technique for detecting outliers in high dimensional datasets based on user examples. Most existing methods are designed to detect outliers with parameters that

include outliers ' hidden view, and such parameters must be decided by users in advance. Our proposed method takes advantage of user examples, so it releases users from deciding parameters beforehand.

A Genetic Algorithm is employed in our proposed method to detect the most suitable subspace where user examples are isolated significantly, since the GA can save much processing time in detecting an optimum solution from a large number of candidates. In this subspace, objects which are also outstanding greatly will be reported as outliers.

We tested our proposed method on both synthetic and real datasets. The result indicates that our proposed method works well in detecting the most suitable subspace based on users' standpoints. In spite of that, there are some factors which can lead to inaccurate results. Therefore, we try to define a more appropriate Fitness Value function in the futrue work, and we will continue to make efforts on our research to improve our proposed method.

In the future, we plan to compare the processing time and quality between our proposed method using the GA and without the GA. We will also compare our method with the method proposed in [2] in terms of processing time and quality.

Acknowledgements

This research has been supported in part by the Grant-in-Aid for Scientific Research from JSPS(#18200005) and MEXT(#18049005).

References

[1] C. Zhu, H. Kitagawa, S. Papadimitriou, and C. Faloutsos. OBE: Outlier By Example. *Proc. PAKDD 2004, LNAI 3056*, pp.222-234, 2004.

[2] C. Zhu, H. Kitagawa, and C. Faloutsos. Example-Based Outlier Detection for High Dimensional Datasets. *IPSJ Transactions on Databases*, Vol. 46, No. SIG5, pp.120-129, 2005.

[3] E. M. Knorr and R. T. Ng. Algorithms for Mining Distance-Based Outliers in Large Datasets. *Proc. VLDB*, pp.392-403, 1988.

[4] http://www.ics.uci.edu/~mlearn/MLRepository.html.

[5] C. C. Aggarwal and P. S. Yu. Outlier Detection for High Dimensional Data. *Proc. SIGMOD Conf.*, pp.37-46, 2001.

[6] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is Nearest Neighbors Meaningful? *Proc. Int. Conf. Database Theory*, pp.217-235, 1999.

[7] M. M. Breuning, H. P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying Density-Based Local Outliers. *Proc. SIGMOD Conf.*, pp.93-104, 2000.

[8] D. E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. *Addison Wesley*, 1989.

[9] C. C. Aggarwal and P. S. Yu. An Effective and Efficient Algorithm for High-dimensional Outlier Detection. *The VLDB Journal*, Vol. 14, No. 2, pp.211-221, 2005.