A Top K Relative Outlier Detection Algorithm in Uncertain Datasets

Fei Liu, Hong Yin, and Weihong Han

College of Computer, National University of Defense Technology,
410073, Changsha, Hunan, China
fliu@cse.unsw.edu.au, huiseguiji0521@yahoo.com.cn,
hanweihong@gmail.com

Abstract. Focusing on outlier detection in uncertain datasets, we combine distance-based outlier detection techniques with classic uncertainty models. Both variety of data's value and incompleteness of data's probability distribution are considered. In our research, all data objects in an uncertain dataset are described using x-tuple model with their respective probabilities. We find that outliers in uncertain datasets are probabilistic. Neighbors of a data object are different in distinct possible worlds. Based on possible world and x-tuple models, we propose a new definition of top K relative outliers and the *RPOS* algorithm. In *RPOS* algorithm, all data objects are compared with each other to find the most probable outliers. Two pruning strategies are utilized to improve efficiency. Besides that we construct some data structures for acceleration. We evaluate our research in both synthetic and real datasets. Experimental results demonstrate that our method can detect outliers more effectively than existing algorithms in uncertain environment. Our method is also in superior efficiency.[1]

Keywords: outlier detection, uncertain dataset, relative, x-tuple.

1 Introduction

In recent years, outlier detection has been widely used, specially in network intrusion detection [1], credit card abuse analysis [2], measurement result analysis of abnormal data [2] and so on. Lots of outlier detection algorithms in deterministic dataset have been proposed, such as model-based [3], index-based [4], distance-based [5], density-based algorithms [6] and so on. In these years, research has turned into uncertain datasets. Uncertainty is inherent in data collected in various applications, such as sensor networks, marketing research, and social science [7]. Sensors in a wireless network can be at different positions at different times with different probabilities. Many datasets published are deformed to hide information for privacy protection. In this case, distance

[1] The authors work is sponsored by the National High Technology Research and Development Program (863) of China (2012AA01A401 and 2012AA01A402), the Nature Science Foundation of China (61303265).

L. Chen et al. (Eds.): APWeb 2014, LNCS 8709, pp. 36–47, 2014.

among data objects, region density and many other metric are uncertain. These properties prevent classic outlier detection methods in deterministic datasets to be used in uncertain datasets directly.

In order to detect outliers in uncertain datasets, C. C. Aggarwal et al. [8] propose a density-based δ-η algorithm to detect outliers in uncertain datasets. They estimate the density of regions. For any data object, with lower probability to be in a high density region, more likely it would be an outlier. They propose the definition of η-probability of a data object. That is defined as the probability that the uncertain data object lies in a region with (overall data) density at least η [8]. An uncertain data object X_i would be a (δ,η)-outlier, if the η-probability of X_i in some subspace is less than δ. For η-probability estimation, authors use a sampling procedure to generate values according to some distribution. For this intention, a value is obtained from uniform distribution as the input of the inverse function of some cumulative density function. However, there are some limitations. First, the data must be in some determinate distribution. But its distribution would usually be unknown in real application. In some application, users can not get complete data distribution. Besides that, the method assumes that the inverse of the distribution can be calculated efficiently. It is difficult too. Although the data is assumed to be in normal distribution in their experiments, the sampling procedure is with heavy time cost. Similarly B. Jian et al. [7] use kernel density estimation to get densities of uncertain objects and their instances. In that model, value of uncertain data is dominated by conditioning attributes. They use kernel density estimation with Gaussian kernels to estimate the probability density of a distribution. In this method, obvious conditioning attributes must be determined first. This limits its application. Wang et al. [9] introduce distance-based outlier detection into uncertain dataset for the first time. They utilize x-tuple model to describe data objects. The method enlightens our research. Nonetheless, they don't take into account data variety.

Table 1. x-tuple Model

x-tuple	tuple	probability
T_1	t_1	p_1
	t_2	p_2
T_2	t_3	p_3

Table 2. Possible Worlds

ID	possible world	probability
1	{}	(1-p_1-p_2)(1-p_3)
2	$\{t_1\}$	p_1(1-p_3)
3	$\{t_2\}$	p_2(1-p_3)
4	$\{t_3\}$	(1-p_1-p_2)(p_3)
5	$\{t_1t_3\}$	p_1p_3
6	$\{t_2t_3\}$	p_2p_3

In order to overcome above problems, we propose the concept of relative outlier and a novel distance-based outlier detection algorithm, *RPOS* algorithm, focusing on top K outlier detection in uncertain datasets. In our research, all data objects exist with independent probabilities. A data object could show various values with different probabilities. We compare every pair of data objects to find the one more like to be an outlier relative to the other one. Global distribution of data value is unnecessary in our research.

2 Outlier in Uncertain Datasets

2.1 Possible World and x-tuple Model

We describe uncertain datasets using possible world semantics [10] and x-tuple model [11]. An x-tuple containing several tuples denotes a data object. A tuple containing several attributes denotes an instance. Every tuple has its own probability. Tuples in different x-tuples are independent. Tuples in the same x-tuple are mutually exclusive. Probability of an x-tuple is the product of probabilities of tuples in the x-tuple. An x-tuple may not exist if no its tuple exists. A subset of tuples from different x-tuples construct a possible world. Probability of a possible world is the product of probabilities of tuples in the possible world. An example of tuples, x-tuples, possible worlds and their probabilities can be shown in Table 1 and 2.

2.2 Relative Outlier

Based on possible world semantics and x-tuple model, we introduce the concept of distance-based outlier into uncertain datasets. In a possible world, every tuple has an outlier score as defined in Definition 1. K tuples ranked at most k according to their outlier scores in descending order are top K outliers in the possible world. If a tuple is a top K outlier in a possible world, the x-tuple containing the tuple would be a top K outlier in the possible world. The tuple's outlier score is also the x-tuple's outlier score. Since there would be many different possible worlds for data variety, top K outliers are uncertain.

Intuitively expected rank of an x-tuple in different possible worlds could be used to detect outlier. But expected rank would be easily influenced by sparse noise. For example, an x-tuple A is ranked $k{+}1$ in every possible world, another x-tuple B is ranked $10k$ in a possible world but k in others. x-tuple B is more like to be an outlier since it's ranked in front of A in most possible world. However, the expected rank of B may be smaller than that of A. So A would be considered as outlier in error. Besides that, the concept of uncertain top-k query [12], that is returning a list of k records which has the highest probability to be the top-k list in all possible worlds, is similar with our problem. But it can also be influenced by sparse data. For example, an x-tuple A is ranked $k{+}1$ in every possible world, another x-tuple B is ranked k in a possible world but $10k$ in others. Although A is not a top K outliers in any possible world, it's more like to be an outlier than B, since it's ranked in front of B in most cases.

In this paper, We propose the concept of relative outlier based on multiple comparisons. All x-tuples are compared with each other to evaluate their possibilities to be outliers. Above problems can be overcome in our method since outliers detected are more likely to have higher outlier scores than other x-tuples. Definition 2 defines the relative outlier between two x-tuples. Definition 3 defines top K relative outliers in an uncertain dataset.

Definition 1. Outlier score of a tuple is the mean distance to its n nearest neighbors [13,1,14].

Definition 2: For two x-tuples A and B in an uncertain dataset, if A is with higher probability to has higher outlier score than B, A is an outlier relative to B. Outlier score in a possible world is computed based on Definition 1.

For example, x-tuple A has higher outlier score than B with probability 0.5, x-tuple B has higher outlier score than A with probability 0.4 and A or B does not exit with probability 0.1. A would be an outlier relative to B.

Definition 3: Top K relative outliers in an uncertain dataset are those x-tuples. They are ranked top K according to the amount of x-tuples relative to which they are outliers based on Definition 2.

For example, x-tuple A is an outlier relative to another 5 x-tuples and x-tuple B is an outlier relative to another 6 x-tuple. B is more likely to be a top K outlier than A. If there are just K x-tuples who are outliers relative to at least another 6 x-tuples, B would be included in top K outliers, but A would be excluded. If there are K x-tuples who are outliers relative to at least another 7 x-tuples, both A and B would not be top K outliers.

3 Basic RPOS Algorithm

In this section, we propose the basic $RPOS$(Relative Probability Outlier Score) algorithm to detect the top K x-tuples most likely to be outliers. For x-tuples A and B, we compute the probability $P(A>B)$ meaning that A's outlier score is higher than B's and $P(B>A)$ meaning that B's outlier score is higher than A's. If $P(A>B)>P(B>A)$, A is considered as an outlier relative to B, and vice versa. Relative Probability Outlier Score($RPOS$) of x-tuple A relative to B is 1 in this case. It's noted as $RPOS(A\rightarrow B)$=1. At the same time, $RPOS(B\rightarrow A)$=-1. The sum of A's $RPOS$s relative to other x-tuples is $all\text{-}RPOS(A) = \sum_{X\in S, X\neq A} RPOS(A\rightarrow X)$. S is the dataset. x-tuples with top K highest $all\text{-}RPOS$s would be outliers.

Algorithm 1 gives details of $RPOS$ algorithm. It compares all x-tuples with others to calculate their $all\text{-}RPOS$s and sort all x-tuples in descending order(lines 2-7). $OutlierScore$ algorithm computes an x-tuple's relative probability outlier score with another x-tuple. $sgn(x)$ = -1,if $x<0$; 0,if $x=0$; 1,if $x>0$.

Algorithm 2 gives details of $OutlierScore$ algorithm. Because an x-tuple consists of several distinct tuples, comparison between two x-tuples is actually comparison among tuples from distinct x-tuples(lines 2-10). Unfortunately Definition 1 can not be used to compute a tuple's outlier score directly. Because all tuples' existence are uncertain, a tuple would exist in several possible worlds with distinct probabilities. So a tuple would have different neighbors in different possible worlds. In order to overcome this problem, we propose relative probability outlier score of a tuple, that is a tuple's outlier score relative

Algorithm 1. RPOS

```
Input: dataset S; outlier amount K
Output: queue of x-tuples Q
1: Q := ∅;
2: for each x-tuple T_i in S do
3:   all-RPOS(T_i) := 0;
4:   for each x-tuple T_j in S, T_j ≠ T_i do
5:     all-RPOS(T_i) += sgn(OutlierScore(T_i, T_j) - OutlierScore(T_j, T_i));
6:   end for
7:   insert Ti into Q according to all-RPOS in descending order;
8: end for
9: return Q
```

Algorithm 2. OutlierScore

```
Input: x-tuple T_1, T_2
Output: P(T_1 > T_2)
1: P(T_1 > T_2) := 0;
2: for each tuple t_i in T_1 do
3:   P := 0;
4:   for each tuple t_j in T_2 do
5:     if RelativeOutlierScore_Tuple(t_i, t_j) = 1 then
6:       P := P + Probability(t_j);
7:     end if
8:   end for
9:   P(T_1 > T_2) += Probability(t_i)·P;
10: end for
11: return P(T_1 > T_2)
```

to another tuple. *RelativeOutlierScore_Tuple* algorithm is used to calculate it. If *RelativeOutlierScore_Tuple* returns 1, the first tuple would has higher probability to get larger outlier score than the second tuple.

3.1 Relative Probability Outlier Score of a Tuple

We show details of *RelativeOutlierScore_Tuple* algorithm in this subsection. Supposing t_{1i} and t_{2j} are respective tuples of x-tuples T_1 and T_2, $score[t_{1i}]$ and $score[t_{2j}]$ are their deterministic outlier scores in a possible world and $score[t_{1i}, t_{2j}]$ is t_{1i}'s relative probability outlier score toward t_{2j}. We compute $score[t_{1i}]$ and $score[t_{2j}]$ in every possible world first. Then $score[t_{1i}, t_{2j}]$ can be computed as follows:

$score[t_{1i}, t_{2j}] \leftarrow 1$, if $P(score[t_{1i}] > score[t_{2j}]) > P(score[t_{2j}] > score[t_{1i}])$.

$P(score[t_{1i}] > score[t_{2j}])$ is the probability that t_{1i}'s deterministic outlier score is greater than that of t_{2j}. $P(score[t_{2j}] > score[t_{1i}])$ is the probability that t_{2j}'s

deterministic outlier score is greater than that of t_{1i}. $score[t_{2j}, t_{1i}]$ = -1 at the same time;

$score[t_{1i}, t_{2j}]$ = -1, if $P(score[t_{1i}]>score[t_{2j}])<P(score[t_{2j}]>score[t_{1i}])$.

$score[t_{2j}, t_{1i}]$ = 1 at the same time.

In a large uncertain dataset, the probability that two tuples have same deterministic outlier score is near 0. $score[t_{1i}, t_{2j}]$ can be computed as follows:

$score[t_{1i}, t_{2j}]$ = 1, if $P(score[t_{1i}]>score[t_{2j}])>0.5$;

$score[t_{1i}, t_{2j}]$ = -1, if $P(score[t_{1i}]>score[t_{2j}])<0.5$.

The intuitive method to compute $score[t_{1i}, t_{2j}]$ is to traverse all possible worlds, compute $score[t_{1i}]$ and $score[t_{2j}]$ and accumulate the probability of $score[t_{1i}]>score[t_{2j}]$. However, this is unavailable in a real application. Traversing all possible worlds would cost exponential time overhead. Suppose S is the dataset, $|S|=N$ and every x-tuple includes N_x tuples. For any tuple t, the amount of possible worlds containing t's n nearest neighbor tuples would be at least $C_N^n {N_x}^n$. In order to lower time cost, we use sampling technique to get an approximate $P(score[t_{1i}]>score[t_{2j}])$.

Tuples in an x-tuple are sampled according to their probabilities. All tuples sampled from different x-tuples composed a possible world. X_k is a random variable in the k^{th} sampling. In the possible world produced by the k^{th} sampling, if both t_{1i} and t_{2j} exist and $score[t_{1i}]\geq score[t_{2j}]$, we set X_k=1. Or else X_k=0. When all tuples' probabilities are determined, $P(score[t_{1i}]\geq score[t_{2j}])$ is determined. $P(score[t_{1i}]\geq score[t_{2j}])$ is named as p in short. We can get $P(X_k=1)=p$, $P(X_k=0)=1\text{-}p$ and $E[X_k]=1\cdot p+0\cdot(1\text{-}p)=p$. $X_1, X_2, \cdots, X_m$ are independent identical distribution random variables. m is sampling number. Variable $X=\sum X_k$, $1\leq k\leq m$. So we can get $E[X]=E(\sum X_k)=\sum E(X_k)=mp$. That means $p=E[X]/m$. We use X to estimate $E[X]$ and estimate p using X/m. In order to ensure the accuracy of estimation, sampling must satisfy some conditions.

When $p\leq 0.5$, the probability of a wrong estimation, namely $X/m>0.5$, is that:

$$P(X > 0.5m) = P(X > 0.5mp/p) = P(X > (1 + (0.5 - p)/p)mp). \quad (1)$$

According to *Chernoff bound*:

$$\begin{aligned} P(X > (1 + (0.5 - p)/p)mp) &\leq exp\{-((0.5 - p)/p)^2 mp/3\} \\ &= exp\{-(0.5 - p)^2 m/3p\}. \end{aligned} \quad (2)$$

If m is determined, 1-$exp\{-(0.5\text{-}p)^2 m/3p\}$ is the accuracy of the estimation. It's only dominated by p.

Similarly, when p>0.5, the probability of a wrong estimation is that:

$$\begin{aligned} P(X < 0.5m) &= P(X < (1 - (p - 0.5)/p)mp) \\ &< exp\{-(p - 0.5)^2 m/2p\}. \end{aligned} \quad (3)$$

1-$exp\{-(p\text{-}0.5)^2 m/2p\}$ is the accuracy of estimation. In summary, the confidence of the estimate is no less than 1-$exp\{-(0.5\text{-}p)^2 m/3p\}$. For example, when p=0.7 and m=100, accuracy of the estimation is no less than 0.94.

Algorithm 3. RelativeOutlierScore_Tuple

```
Input: tuples t1i and t2j; sampling time m
Output: outlier score of t1i relative to t2j, score[t1i,t2j]
1: score[t1i,t2j] := 0;
2: for the kth sampling, 1 ≤ k ≤ m  do
3:   score[t1i,t2j] += sgn(DistanceOutlierScore(t1i)k -
     DistanceOutlierScore(t2j)k);
4: end for
5: score[t1i,t2j] := sgn(score[t1i,t2j]);
6: return  score[t1i,t2j]
```

$DistanceOutlierScore(t)_k$ the deterministic outlier score of t in the possible world produced by the k^{th} sampling. In $DistanceOutlierScore(t)_k$, outlier score of the target tuple in a possible world is computed as Definition 1. However, it's in high time cost to detect a tuple's n nearest neighbors like classic methods e.g. $RBRP$ algorithm [15] in each sampling. In order to reduce time cost, we construct a neighbor list L_t for every tuple t. The node of L_t is a novel $Neighbor$ structure:

$Neighbor\langle t_{neighbor}, d_{neighbor}, tag\rangle$
$t_{neighbor}$: a neighbor tuple of t;
$d_{neighbor}$: distance from $t_{neighbor}$ to t;
tag: it is used to state whether $t_{neighbor}$ is selected in the sampling.

All $Neighbor$s in L_t are sorted according to $d_{neighbor}$ in ascending order. Tuples in the same x-tuple with t will not be in L_t. Let $|L_t|$=L. In each sampling, we traverse L_t in order and get n nearest selected tuples noted by tags. The mean distance of these n $d_{neighbor}$s is t's outlier score. When L is large enough, L_t could contain almost all n nearest neighbors of t in each sampling. While n nearest neighbors may exist in the latter part of the list with a low probability. Too large L leads to redundant memory consumption. Proper value of L should be set.

4 Pruning Strategies

In basic $RPOS$ algorithm above, every x-tuple has to be compared with all others. Its time cost is proportional to the square of a dataset's cardinality. In order to improve running speed of basic $RPOS$ algorithm, we introduce two pruning strategies.

4.1 Strategy 1

In order to detect outlier in high efficiency, we must find every tuple's neighbors quickly. Neighbors of a tuple should be close to each other. These neighbors could construct in a cluster. Using for reference from existing methods [15,16], we cluster all tuples in a dataset. Usually distances among tuples in a cluster are much less than those among tuples in different clusters. n nearest neighbors

of a tuple t would be in the same cluster with t or t's neighboring clusters. In order to construct L_t, we check tuples in the cluster containing t first, and then tuples in neighboring clusters. Other clusters would be in the end. The distance from the L^{th} tuple in L_t to t will be a threshold in following process. We name the threshold as h. When we check a following neighbor tuple, if its distance to t is larger than h, it would not be inserted into L_t. On the contrary, it would be a candidate tuple and inserted into L_t in ascending order. h is then updated. Because we check tuples near with t first, h will always be small. Tuples far away from t will be pruned soon. In order to accelerate clustering process, we first partition the dataset into several large clusters and then partition every cluster into some sub-clusters recursively.

When we check t's neighbor t', if $L_{t'}$ has be constructed with threshold h', the distance from one of t''s n nearest neighbor tuples to t would be less than $D(t,t')+h'$. $D(t,t')$ is the distance from t to t'. So that $h<D(t,t')+h'$. h is updated by $min\{h, D(t,t')+h'\}$.

Further, before checking a neighbor cluster C' of t, we compute the maximum and minimum distances from t to C' first. The maximum distance from t to C' is $MaxD(t,C')=D(t,o')+r'$, where o' is the center of C' and r' is the radius of C'. The minimum distance from t to C' is $MinD(t,C')=D(t,o')$-r'. If $MinD(t,C')>h$, distance from t to any tuple in C' can not be less than h. All tuples in C' would not be inserted into L_t. C' will be jumped over. If $MinD(t,C')\leq h$, we will check its sub-clusters. In this way, we only need to check a few tuples to construct L_t. Smaller search space leads to lower time cost.

4.2 Strategy 2

When we compute *all-RPOS* of an x-tuple, the x-tuple has to be compared with all other x-tuples. When we compare two x-tuples, we have to compare all tuples from two x-tuples respectively. Time complexity in this process is $O(N_x^2 N^2)$. N is the cardinality of the dataset. N_x is the amount of tuples in an x-tuple. But in real application outliers are in minority of the entire dataset. It's wasteful to compare all pairs of x-tuples. Suppose there are K outliers in a dataset and X x-tuples have been checked in *RPOS* algorithm. If $X>K$, we sort these X x-tuples based on their *all-RPOS*s and use the K^{th} *all-RPOS* as the threshold namely H. If expected value of an x-tuple's *all-RPOS* is less than H, it can not be an outlier. When new top K candidate outliers are detected, H is updated. In above process, expected *all-RPOS* value of an x-tuple can be computed using existed *all-RPOS* value (as that in line 5 of *RPOS* algorithm) of the x-tuple plus the number of x-tuples will be compared with it.

Further Acceleration 1: Efficiency of pruning process above would be influenced by H. Quickly H increases, more x-tuples could be pruned early. We sort all x-tuples according to expected *all-RPOS* in descending order.

Further Acceleration 2: In order to avoid redundant comparison in pairs of x-tuples, we record all x-tuples have been compared with and the intermediate result of every x-tuple's *all-RPOS*.

5 Experiments

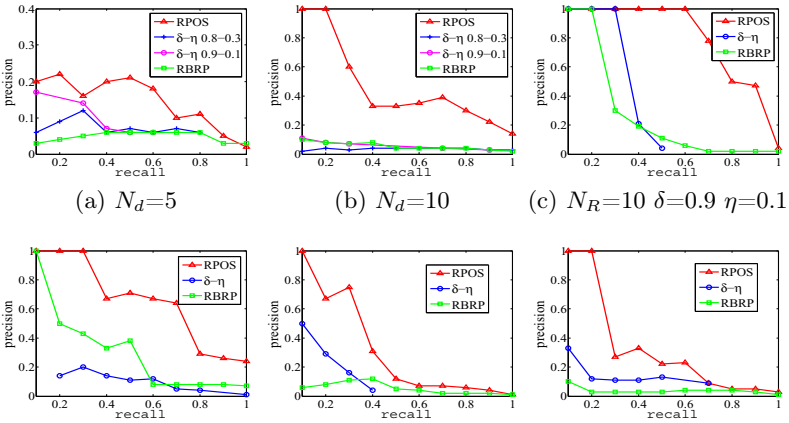
In this section, we evaluate our *RPOS* algorithm in synthetic and real datasets. All algorithms are implemented in Java. Our experiments are ran on a machine with 2 Intel Core 2 Duo E8400 3GHZ CPUs and 8.1GB RAM running Linux 3.8 Ubuntu 13.04. In our research, all attributes of data are numerical and the value of each attribute is a real number. We compare *RPOS* algorithm with *RBRP* [15] and δ-η [8] algorithms. B.Jiang's work [7] focuses on condition attributes and B. Wang's work [9] neglects data diversity. It's hard to compare them with our work. In order to use *RBRP* algorithm to detect outliers in uncertain datasets, we pretreat uncertain data for experiments. All tuples in an x-tuple are transformed into a tuple. The value of the new tuple in every dimension is the weighted mean value of all tuples of the x-tuple in the dimension. Weight of a tuple is its probability.

5.1 Dataset

Synthetic Dataset. In order to test the effectiveness and efficiency of different outlier detection algorithms, we construct several synthetic datasets. The synthetic data includes N_d attributes. Every data entity is an x-tuple including several tuples. A tuple's value in every attribute is numeric. We produce some normal regions in the N_d-dimension space. Normal tuples are allocated in these regions. On the contrary, outliers will not be in normal regions. Besides synthetic data values, we also produce the probability for every tuple. We define N_R normal regions in N_d dimensions respectively. R_i is the normal region in dimension i. R_i=[$LOW(R_i)$,$UP(R_i)$]. R'_{i*} is a sub-region of R_i. $R'_{i*} \subset R_i$. N_d sub-regions construct a N_d-dimension normal region $\langle R'_{1*}, R'_{2*} \cdots R'_{N_d*} \rangle$. $r = R'_{i*}/R_i$ determines the size of sub-region R'_{i*}. We produce N_R N_d-dimension normal regions in this way. Data objects outside of these N_R regions is abnormal. First we produce normal x-tuples in normal regions produced above. Then we produce outlier x-tuples with at least N'_x abnormal tuples in each x-tuple. In order to evaluate performance of *RPOS* algorithm, we insert some counterfeit outliers into the dataset as disturbance. A counterfeit outlier x-tuple contains several abnormal tuples. The tuples may be allocated far away from normal regions. But their quantity is small and their probabilities are smaller than those of abnormal tuples in real outliers x-tuples.

Real Dataset. We choose the real MiniBooNE dataset[2] provided by UCI. Number of entities in this dataset is 130000. Number of attributes is 50. Attribute characteristics are real. We transform the original dataset into an uncertain dataset for our experiments. Every entity in the original dataset is an x-tuple containing one tuple in the uncertain dataset. We fluctuate the value of a tuple to produce other tuples in the x-tuple. The value of a tuple in every dimension fluctuates with probability p_f. Fluctuation range is controlled by r_f. There are at most N_x tuples in an x-tuple. We add every tuple's probability as in synthetic dataset.

[2] `http://archive.ics.uci.edu/ml/datasets`



(a) N_d=5 (b) N_d=10 (c) N_R=10 δ=0.9 η=0.1

(d) N_R=20 δ=1 η=0.3 (e) N_R=70 δ=0.9 η=0.1 (f) N_R=80 δ=0.9 η=0.1

Fig. 1. Effectiveness of different algorithms

5.2 Effectiveness Evaluation

In this sub-section, we show the evaluation result of different algorithms' effectiveness to detect outliers. Because no instance in real datasets is labeled as an outlier, effectiveness evaluation is performed only in synthetic datasets. Some parameters are modified to produce different datasets.

In order to test the influence of the data dimensionality, we perform experiments in 5 and 10 dimensions respectively. We set N_R=100, N_x=5. Amount of outliers $N_{outlier}$=0.01N. Amount of counterfeiters $N_{counterfeiter}$=0.05N. N is the cardinality of a dataset. For *RPOS* algorithm we set parameters L=200 and m=100 (see section 3). For δ-η algorithm we do experiments in two situations. First, parameter δ is set to be 0.8 and η is set to be 0.3, and then δ is set to be 0.9 and η is set to be 0.1. Space lack for more parameter setting. But they do not influence experimental results. In Figure 1(a), dimensions amount is N_d=5. N_d=10 in Figure 1(b). Precision and recall ratio are two test indexes. As shown in these figures, in various amount of dimensions, *RPOS* algorithm always performs best in three algorithms. With same recall rate, *RPOS* algorithm detects outliers in high precision.

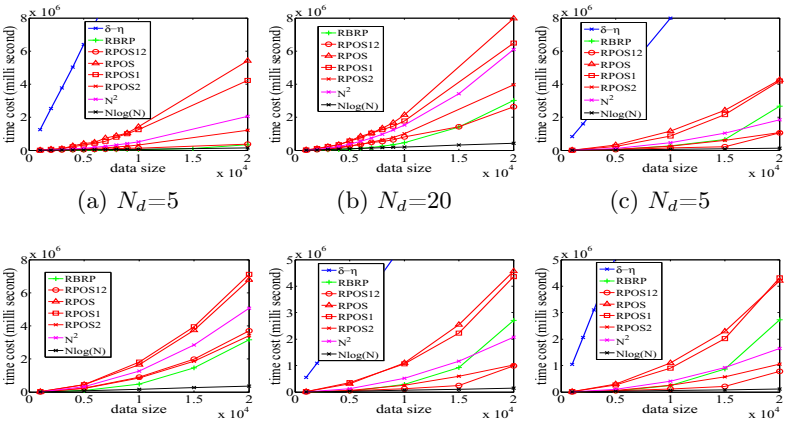
We then change parameter N_R to produce different uncertain synthetic datasets to evaluate the effectiveness of different algorithms. Data is processed in these experiments with N_d=5. N_R is set to be different values with r=0.5. δ and η are set arbitrarily. Results can be found in Figure 1(c-f). As shown in above figures, *RPOS* algorithm performs best in almost all experiments.

5.3 Efficiency Evaluation

In this sub-section, we show evaluation results of different algorithms' time cost. Our experiments perform in both synthetic and real datasets. We implement

basic *RPOS* algorithm (*RPOS*), *RPOS* with pruning strategy 1 (*RPOS*1), *RPOS* with pruning strategy 2 (*RPOS*2), *RPOS* with both pruning strategies 1 and 2 (*RPOS*12), δ-η algorithm and *RBRP* algorithm in every experiment. Time complexity of $O(NlogN)$ and $O(N^2)$ are shown for comparison.

Synthetic datasets are produced with N_R=100, r=0.5, N_x=5. We change N_d in different experiments. In *RPOS* algorithm, the rate K/N=r_k=0.02. For instance, in a dataset containing 10000 x-tuples, K is 200. r_k is set to be a constant to avoid its influence. Similarly, we set the rate between amount of clusters and dataset cardinality r_C=0.25 to ensure stability of pruning effect. In δ-η algorithm, we set δ=0.06 and η=0.5. Experiments results in 5 and 20 dimensions are shown in Figure 2(a,b). Data size increases in X-axis and corresponding time cost is shown in Y-axis.



(a) N_d=5 (b) N_d=20 (c) N_d=5

(d) N_d=20 (e) p_f=0.3, r_f=0.3 (f) p_f=0.7, r_f=0.7

Fig. 2. Time cost in synthetic datasets(a,b) and real datasets(c-f)

As shown in Figure 2(a), pruning strategies 1 and 2, especially strategy 2 can accelerate *RPOS* algorithm. Time cost of basic *RPOS* algorithm and *RPOS*1 are higher than $O(N^2)$. The main time consumption is from comparison among tuples and x-tuples. Time cost of *RPOS*2 and *RPOS*12 are lower than $O(N^2)$ but higher than $O(NlogN)$. Pruning strategies can improve basic *RPOS* algorithm obviously. Speedup using pruning strategies in 20 dimensions is similar as in 5 dimensions.

Time cost are also evaluated in real datasets. First we set parameters p_f=0.5, r_f=0.5. Other parameters keep consistent with those in synthetic datasets. Experiments are implemented in 5 and 20 dimensions respectively. Results of time cost can be shown in Figure 2(c,d). Basic *RPOS* algorithm runs with higher time complexity than $O(N^2)$. pruning strategy 2 can improve *RPOS* algorithm greatly.

We then change p_f and r_f to produce different uncertainty with N_d=20. Time cost is shown in Figure 2(e,f). We can find that results are similar in different experiments. With various values of p_f and r_f, $RPOS12$ performs better than others in time cost.

References

1. Eskin, E., Arnold, A., Prerau, M., Portnoy, L., Stolfo, S.: A geometric framework for unsupervised anomaly detection. In: Applications of Data Mining in Computer Security, pp. 77–101. Springer (2002)
2. Kriegel, H.P., Kroger, P., Schubert, E., Zimek, A.: Outlier detection in arbitrarily oriented subspaces. In: 12th International Conference on Data Mining (ICDM), pp. 379–388. IEEE (2012)
3. Rousseeuw, P.J., Leroy, A.M.: Robust regression and outlier detection, vol. 589. Wiley.com (2005)
4. Han, J., Kamber, M., Pei, J.: Data mining: concepts and techniques. Morgan Kaufmann (2006)
5. Aggarwal, C.C., Yu, P.: An effective and efficient algorithm for high-dimensional outlier detection. The VLDB Journal 14(2), 211–221 (2005)
6. Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: Lof: identifying density-based local outliers. ACM Sigmod Record, 93–104 (2000)
7. Jiang, B., Pei, J.: Outlier detection on uncertain data: Objects, instances, and inferences. In: ICDE, pp. 422–433. IEEE (2011)
8. Aggarwal, C.C., Yu, P.: Outlier detection with uncertain data. In: SDM, pp. 483–493 (2008)
9. Wang, B., Xiao, G., Yu, H., Yang, X.C.: Distance-based outlier detection on uncertain data. In: CIT, pp. 293–298. IEEE (2009)
10. Dalvi, N., Suciu, D.: Efficient query evaluation on probabilistic databases. The VLDB Journal 16(4), 523–544 (2007)
11. Parag, A., Benjelloun, O., Sarma, A.D., Hayworth, C., Nabar, S., Sugihara, T., Widom, J.: Trio: A system for data uncertainty and lineage. In: VLDB (2006)
12. Hua, M., Pei, J., Zhang, W.J., Lin, X.M.: Efficiently answering probabilistic threshold top-k queries on uncertain data. In: ICDE, vol. 8, pp. 1403–1405 (2008)
13. Angiulli, F., Pizzuti, C.: Outlier mining in large high-dimensional data sets. IEEE Transactions on Knowledge and Data Engineering 17(2), 203–215 (2005)
14. Bay, S.D., Schwabacher, M.: Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In: Proceedings of the Ninth ACM SIGKDD, pp. 29–38. ACM (2003)
15. Ghoting, A., Parthasarathy, S., Otey, M.E.: Fast mining of distance-based outliers in high-dimensional datasets. Data Mining and Knowledge Discovery 16(3), 349–364 (2008)
16. Vu, N.H., Gopalkrishnan, V.: Efficient pruning schemes for distance-based outlier detection. In: Buntine, W., Grobelnik, M., Mladenić, D., Shawe-Taylor, J. (eds.) ECML PKDD 2009, Part II. LNCS, vol. 5782, pp. 160–175. Springer, Heidelberg (2009)