Quality Assurance of Outsourced Outlier Mining

Yongjin Zhang[1], Ruilin Liu[1], Hui (Wendy) Wang[1],Anna Monreale[2], Dino Pedreschi[2],Fosca Giannotti[3], Wenge Guo [4]
Stevens Institute of Technology[1], University of Pisa[2], ISTI CNR[3], New Jersey Institute of Technology[4]
Hoboken, NJ, USA[1], Pisa, Italy [2,3], Newark, NJ, USA[4]
{rliu3,yzhang21,Hui.Wang}@stevens.edu[1],{annam, pedre}@di.unipi.it[2],fosca.giannotti@isti.cnr.it[3],wenge.guo@njit.edu[4]

ABSTRACT

Spurred by developments such as in cloud computing, there has been considerable recent interest in the paradigm of data mining-as-service. A company (data owner) lacking in expertise or computational resources can outsource its mining needs to a third-party service provider. However, as the service providers may not be fully trusted, a dishonest service provider may return inaccurate mining results to the database owner. In this paper, we study the problem of providing quality assurance for outsourced outlier mining. We propose an efficient and practical auditing approach that can verify (1) whether the service provider returns the outliers originated from the hosted database, and (2) whether the service provider returns correct and complete outlier mining results. The key of our approach is to insert a small amount of artificial tuples into the outsourced database; the mining results of the service provider will be audited by analyzing the inserted tuples in the returned results with probabilistic guarantee. Our empirical results demonstrate the effectiveness and efficiency of our method.

1. INTRODUCTION

Outlier mining has been in critical need in many real applications, such as credit card fraud detection, discovery of criminal activities, weather prediction, marketing and customer segmentation. The task of outlier mining is to find the data objects that do not comply with the general pattern of the majority. The problem of outlier detection has been widely studied in the data mining community [3, 5, 6, 10, 17]. These work has shown that detecting outliers in general is of high computational complexity [6, 10]. Such computational complexity gets even prohibitive for the datasets of high dimensionality [3]. Although a variety of methods have been proposed to improve the efficiency of outlier detection, these methods require either the availability of indexes [11] or the delicate design and implementation of mining algorithms [11, 17]. It is difficult for the data owners (e.g., credit card companies) who lack professional expertise to implement these techniques. Therefore, the data owners, especially the ones with large volume of data but limited budget for fulfilling analysis on these data, would outsource their datasets to third-party professionals. This is the data mining-as-service paradigm that grows popular recently [16, 20]. Cloud computing, an emerging trend of provisioning scalable and reliable computing services, provides a natural solution for the mining-as-service paradigm. Indeed, there has been much effort in industry to realize the paradigm. For example, Oracle has installed its data mining tools on the Amazon Cloud [2], while Microsoft has enabled data mining services in its cloud named DMCloud [1].

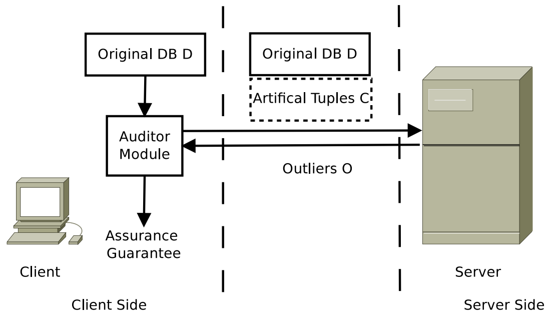


Figure 1: System Architecture

Although outsourcing is advantageous for data owners with limited abilities to achieve sophisticated analysis on its large volume of data, it triggers serious issues in the quality assurance of the mining results. The server may return wrong mining results either intentionally or accidentally for various reasons (e.g., it may only analyze part of the database to reduce the computation cost). In this paper, we focus on outlier mining. We consider three types of erroneous mining results by the server: (1) the returned outliers do not exist in the original database, (2) the returned outliers are indeed non-outliers, and (3) some of the outliers were not returned. With awareness of possibily inaccurate mining results, the client needs to ensure that the service provider indeed returns authentic, correct and complete results. By authenticity, we mean the server will return outliers that indeed exist in the hosted database. By correctness, we mean the server only returns true outliers. By completeness, we mean the server returns all true outliers.

A seemly straightforward solution to provide assurance on mining results is to allow the client to verify the outliers returned from the server against the original database. This solution has a few deficits. First, the data owner may not have sufficient computational power and expertise to accomplish such costly auditing task. Second, the client may not possess the original database after she sent the dataset to the server. Third, the client cannot verify the completeness by herself without computing all outliers. Therefore, our goal is to design practical and efficient techniques that provide assurance guarantees to the data owner, while keeping the resource requirements under control.

The architecture behind our model is illustrated in Figure 1. The data owner (client) sends her dataset to a third-party service provider (server), and requests the server to report the outliers in the dataset, if there is any. After sending the database, the client does not keep the database at her side. Instead, she maintains a small piece of auxiliary information for verification purpose. The server applies outlier mining on the hosted database and returns the resulted outliers to the client. After receiving the results, the client will feed the auditor module, which can be essentially treated as a "black box", with the returned outliers. The auditor module will provide the quantified assurance guarantee of the mining results to the client.

The core of our auditing procedure is a novel *artificial tuple injection* (ATI) technique that we proposed. ATI provide probabilistic guarantee for both completeness and correctness auditing; incorrect and incomplete answers from the server can be caught with high confidence. In particular, the auditor module constructs a set of artificial tuples that consists of two types of data, artificial outliers (AOs) and artificial non-outliers (ANOs). ATI plants both AOs and ANOs into the original database and sends them together to the service provider. The auditor module ensures that these counterfeit tuples are indistinguishable from the true ones. When validating the completeness of the outlier results, the auditor module analyzes the returned outliers against AOs and ANOs, and quantifies the probabilistic guarantee of the completeness and correctness. A nice property of our ATI techniques is that the original database is not needed for auditing; instead, only a small piece of auxiliary information (including a hash function, three constants, and the number of AOs and ANOs) will be used.

The problem of providing assurance to data-mining-as-service is initialized by [21]. However, it only focuses on frequent pattern mining outsourcing. To our best knowledge, we are the first to investigate the problem of providing assurance for outlier mining outsourcing in clouds. Our contributions are as follows.

- We identify three possible erroneous ming results, namely the fabrication error, the falseness error, and the concealment error, of outsourcing of outlier mining by the malicious server. To provide auditing mechanisms for these errors, we aim at providing assurance guaranteed for authenticity, completeness and correctness of the outlier mining results.

- We propose a signature-based scheme that can provide authenticity guarantee. The signatures also serve the purpose of verifying both completeness and correctness by distinguishing artificial tuples from the original ones efficiently.

- We propose a probabilistic approach that can provide both correctness and completeness guarantee with high confidence. We formally quantify the probability of both correctness and completeness, and show that the targeted malicious behaviors by the server can be caught with high probability with a small number of artificial tuples. We further define the α-completeness and β-correctness models that assign bounds for the completeness and correctness guarantee.

- We propose efficient approaches to construct artificial outliers and non-outliers, without validating them against the original database. We discuss how to design these artificial tuples so that: (i) they preserve the required (non)outlierness when they were put together with the original database, and (ii) they satisfy the required α-completeness and β-correctness.

- We complement our analytical results with an extensive set of experiments. The results demonstrating the efficiency and the effectiveness of our approach.

The paper is organized as following. Section 2 discusses the related work. Section 3 introduces the preliminaries of our work. Section 4 discusses our signature scheme for verification of authenticity. Section 5 introduces our probabilistic approach for verification of both completeness and correctness. Section 6 presents our solutions of efficient construction of AOs and ANOs. Section 7 introduces the verification procedure at the client side. Section 8 presents our experimental results. Section 9 concludes the paper.

2. RELATED WORK

In this section, we briefly summarize related work.

Data and pattern security in data outsourcing. Protection of data and data mining results of outsourced databases is another security issue that was caught much attention recently. A few work [8, 13, 19, 20] have been done under this theme. Wong et al. [20] consider utilizing a one-to-n item mapping together with non-deterministic addition of cipher items to protect the identification of individual items in the scenario that frequent pattern mining task is outsourced. Unfortunately, this work has potential privacy flaws; Molloy et al. [13] show how privacy can be breached in the framework of [20]. Tai et al. [19] consider the same scenario and proposed a database transformation scheme that is based on a notion of k-support anonymity. To achieve k-support anonymity, they introduced a pseudo taxonomy tree; the third party server will discover the generalized frequent itemsets instead. Giannotti et al. [8] define a similar privacy model as k-support that requires each item must be indistinguishable from the other $k-1$ items regarding their frequencies. They provide formal privacy analysis of their k-privacy model on both items and frequent patterns. Although these works focus on frequent pattern mining, their encryption techniques can be applied to our work to provide further protection on data and mining results.

Assurance of outsourced query evaluation. The issue of providing assurance for database management was initially raised in the database-as-service paradigm [9]. In this paradigm, the focus is on the evaluation of SQL queries over hosted relational databases. [9, 7, 14, 12, 15, 18, 22] study and propose a few techniques to provide assurance for SQL query evaluation. For example, [7, 14, 12] propose to use Merkle hash trees to audit the completeness of the query answers. Pang et al. [15] introduce a scheme to verify completeness by assigning signatures on a chain of paired tuples. Sion [18] proposes a *challenge token* mechanism and uses it as a probabilistic proof that the server has executed the queries over the entire database. Xie et al. [22] insert a small amount of counterfeit records (via both randomized and deterministic approaches) into the outsourced databases; the completeness and correctness can be audited by analyzing the inserted records in the query results. We follow the similar idea of using artificial tuples for quality verification of data mining results.However, our problem is more challenging as our artificial tuples should present outlierness/non-outlierness property. It is not clear how to easily adapt these aforementioned techniques to our problem.

Assurance of outsourced data mining. The problem of assuring the quality of data mining results in the outsourcing paradigm has been rarely studied. To our best knowledge, Wong et al. [21] are the first (and the only) work under this theme. In this work, they propose auditing techniques for outsourcing of frequent itemset mining. Essentially they generate a (small) artificial database such that all itemsets in the database are guaranteed to be frequent and their exact support counts are known. By hosting the artificial database with the original one and checking whether the server has returned all artificial itemsets, the data owner can verify whether the server has returned correct and complete frequent itemsets. As their focus is on frequent itemset mining, while ours is on outlier mining, their techniques cannot be directly applied to our problem.

3. PRELIMINARIES

3.1 Outlier Mining Task

A variety of definitions of outliers, including distance-based outliers [10] and density-based outliers [6], have been proposed recently. In this paper, we focus on distance-based outliers. The definition of distance-based outliers is the following [10]:

DEFINITION 3.1. **[(p, d)-outlier]** *An object O in a dataset D is a (p, d)-*outlier *if at least $p\%$ of the objects in D lies greater than distance d from O. Otherwise, O is a non-outlier with regard to (p, d) setup.*

For simplicity, we say O is a non-outlier if it is not a (p, d)-*outlier*. We assume $p\% * |D|$ always returns an integer, as it indicates the number of tuples. We use Euclidean distance to measure the distance between two tuples. In particular, given two tuples $t(a_1, \ldots, a_k)$ and $t'(a'_1, \ldots, a'_k)$, $dist(t, t') = \sqrt{\sum_{i=1}^{k}(a_i - a'_i)^2}$.

3.2 Assurance Goal

We assume the service provider (server) can return inaccruate outlier mining results. We consider the following types of erroneous behaviors that the server can conduct: (1) *Fabrication behavior*: the server returns outliers that do not exist in the hosted database; (2) *Falseness behavior*: the server returns non-outlier tuples as outliers; and (3) *Concealment behavior*: the server retains some true outliers and do not include them in the returned results.

To address these erroneous behaviors, we aim to designing an auditing environment for outsourcing of outlier mining. The auditing framework will provide the following assurance guarantees: (1) *Authenticity* guarantee that defends against the fabrication behavior; (2) *correctness* guarantee that defends against the falseness behavior; and (3) *completeness* guarantee that defends against the concealment behavior.

4. AUTHENTICITY GUARDS

Before the data owner sends out her dataset, the auditor module (at the data owner side) will sign each tuple with a cryptographic signature. The signature consists of two sub-signatures: Sig_a and Sig_t. Sig_a provides authenticity guarantee, so that any modification on the original tuples can be caught, while Sig_t is used to distinguish the true tuples from the artificial ones that will be inserted for verification of completeness and correctness. In particular, given a tuple $t(a_1, \ldots, a_n)$,

$$Sig_a = H(a_1 \oplus \ldots a_n),$$

and

$$Sig_t = \begin{cases} H(Sig_a \oplus c_1), & \text{If } t \text{ is a true tuple in } D; \\ H(Sig_a \oplus c_2), & \text{If } t \text{ is an artificial outlier;} \\ H(Sig_a \oplus c_3), & \text{If } t \text{ is an artificial non-outlier.} \end{cases}$$

The auditor module pre-defines three constants c_1, c_2, and c_3, for the true tuples, the artificial outlier tuples, and the artificial non-outlier tuples. It stores the three constants and the hash function H locally. We require that the hash function H is an efficiently computable collision-resistance hash function [4]. It takes a variable-length input and returns a fixed length binary sequence. Furthermore, in case of malicious server it should be difficult for the attacker to reverse the hash value, i.e., given $H(x)$, it is computational infeasible to compute x. Therefore, the server cannot easily modify the signature.

After the auditor module completes the computation of signatures, the data owner sends the dataset to the server. Each tuple in the dataset is associated with its two signatures Sig_a and Sig_t. These two signatures will not be taken into consideration for distance computation. When the server returns the outlier tuples to the client, he is required to return the two signatures of these tuples too. How the auditor module will use the signatures for audting will be explained in Section 7.

5. COMPLETENESS AND CORRECTNESS GUARANTEE

Overview. The core of our auditing framework is the *artificial tuple injection* (ATI) technique. ATI provides probabilistic guarantee for both completeness and correctness auditing; incorrect and incomplete answers from the service provider can be caught with a high confidence. In particular, ATI generates a set of artificial outliers (AOs) and artificial non-outliers (ANOs). ATI inserts these AOs and ANOs into the original database and sends them together to the server. Since AOs are indistinguishable from the true outliers, if the server is honest, it should return all AOs as part of the mined outliers. Thus during the auditing process, the auditor module will verify whether the outliers returned from the server contain all AOs, and obtain a probabilistic guarantee of the completeness accordingly. Similarly, to verify the correctness, the auditor module verifies whether the service provider has returned any ANO, and concludes the correctness with a probabilistic guarantee. The probabilities of both completeness and correctness are quantified below.

THEOREM 5.1. **(Completeness Guarantee) :** Given a database D, let AO be the set of artificial outliers that is inserted into D. Let O be the set of outliers that is returned by the service provider. Let m be the number of

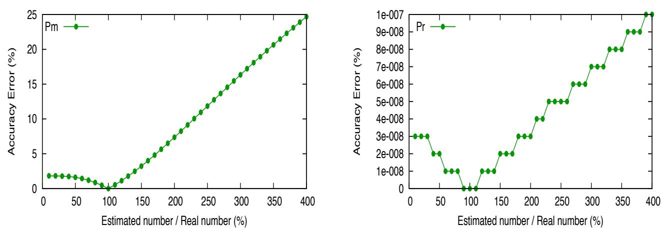


Figure 2: Accuracy error by estimated # of outliers

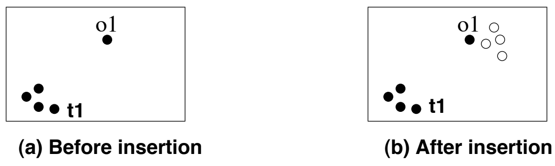


Figure 3: An example to show that insertion of new tuples can change outlierness; Black and white circles represent original and inserted tuples.

true outliers, and k be the number of true outliers that are retained by the server. Then the probabilistic guarantee p_m that the server will be caught when he returns incomplete outlier mining result is

$$p_m = \begin{cases} 1 & \text{If } AO \not\subseteq O \\ 1 - \frac{\binom{m}{k}}{\binom{|AO|+m}{k}} & \text{Otherwise;} \end{cases}$$

Proof: The service provider will get caught if at least one of the k deleted outliers is faked. Thus the probability $= 1 - Prob(all\ k\ deleted\ outliers\ are\ true) = 1 - \frac{\binom{m}{k}}{\binom{m+|AO|}{k}}$. ∎

Similarly,

THEOREM 5.2. **(Correctness Guarantee) :** Given a database D, let ANO be the set of artificial non-outliers that is inserted into D. Let O be the set of outliers that is returned by the service provider. Let k be the number of true non-outliers that are returned as outliers, and m be the number of true outliers. Then the probabilistic guarantee p_r that the service provider will be caught when he returns incorrect answers is

$$p_r = \begin{cases} 1 & \text{If } ANO \cap O \neq Null \\ 1 - \frac{\binom{|D|-m}{k}}{\binom{|ANO|+|D|-m}{k}} & \text{Otherwise;} \end{cases}$$

where k is the number of true non-outliers that are returned as outliers, and m is the number of true outliers.
Proof: The service provider will get caught if at least one of the returned outliers is indeed an artificial non-outlier. Thus the probability $= 1 - Prob(all\ k\ returned\ non-outliers\ are\ true) = 1 - \frac{\binom{m}{k}}{\binom{m+|AO|}{k}}$. ∎

In both Theorem 5.1 and 5.2, it is essential to value the number of outliers in the original database. We assume the data owner has prior estimation of the number of outliers. Our empirical study shows that the inaccuracy of the estimation does not affect the probabilistic guarantee significantly. For example, Figure 2 shows that for the *KDDCUP* dataset [1] that contains $494K$ tuples and 100 outliers (according to our (p, d) configuration), the estimated number of outliers (varying from 10% to 400% of the real number) only incur at most 20% accuracy error in P_m, and neglible error in P_r (at most 0.00000001%).

In general, there exists a trade-off between the probabilistic guarantee that can achieve and the number of artificial tuples that have to be inserted to achieve the required guarantee. Better probabilistic guarantee can be achieved by inserting more artificial tuples, which may lead to higher overhead. To address this issue, we require that the probabilistic guarantee of both correctness and completeness should be no less than a given threshold. To address this requirement, we define *α-completeness and β-correctness.*

DEFINITION 5.1. [**α-completeness & β-correctness**] *Given a database D, let F be the set of artificial tuples that are inserted into D. We say that F can verify α-*completeness *$(0 \leq \alpha \leq 1)$ if $p_m > \alpha$, and can verify β-*correctness *$(0 \leq \beta \leq 1)$ if $p_r > \beta$.*

Challenges. Inserting tuples (as verifiers) into databases may change the (non)outlierness of some tuples in the original dataset. Example 5.1 gives more details.

EXAMPLE 5.1. Consider the dataset shown in Figure 3 (a) that contains 5 tuples in total. The tuple o_1 is a $(50\%, d)$-outlier in the dataset, and the tuple t_1 is not. However, after inserting four artificial tuples into the database (shown in Figure 3 (b)), o_1 is not a $(50\%, d)$-outlier anymore, while t_1 turns to a $(50\%, d)$-outlier now.

The change of (non)outlierness can happen on both original and artificial tuples. Preserving the (non)outlierness of original and artificial tuples is essential to integrity auditing. We say the artificial tuples are *valid* if the (non)outlierness of both original and artificial tuples are well preserved after inserting artificial tuples into the original database.

Therefore, our goal is that given a dataset D, generate a set of valid AOs and ANOs that can verify α-completeness and β-correctness.

6. THE QUALITY ASSURANCE METHOD

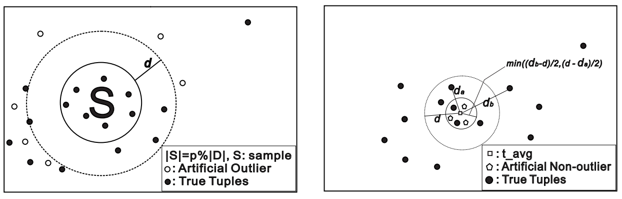
6.1 Artificial Outlier Construction

A naive way to construct artificial outliers is that the auditor module randomly creates tuples, and checks whether these artificial tuples are outliers. This can be prohibitively expensive. Our goal is to construct artificial tuples that are guaranteed to be (p, d)-outliers with low computational cost. In the following, we discuss the details of how to efficiently construct such tuples.

Our construction procedure is based on the definition of *distant* tuples.

DEFINITION 6.1. [**Distant Tuple**] *Given the database D and a set of tuples $S \subseteq D$, let n be number of attributes of S (and D), and min_i and max_i the minimum and maximum value of the i-th $(1 \leq i \leq n)$ attribute of S. Then for any*

[1]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html



(a) AO construction (b) ANO construction

Figure 4: Artificial Tuple Construction

tuple $o(a_1, \ldots, a_n) \notin S$, if it has k ($1 \leq k \leq n$) attributes such that on each attribute A_i, $a_i < (min_i - \frac{d}{\sqrt{k}})$ or $a_i > (max_i + \frac{d}{\sqrt{k}})$, then we call o a distant *tuple of S.*

Next, we show that (p, d)-outliers can be generated from the distant tuples.

Theorem 6.1. Given the database D, let S be a subset of D such that S takes $p\%$ tuples of D. Then any distant node o of S must be a (p, d)-outlier in $D \cup \{o\}$, and any distant node o of D must be a $(100\%, d)$-outlier in $D \cup \{o\}$. **Proof.** First, we prove that o is a $(100\%, d)$-outlier to S. Let $t'(a'_1, \ldots, a'_n)$ be a tuple in S. We prove that $dist(t, o) > d$ always holds. We have:

$$dist(t, o) = \sqrt{\sum_{i=1}^{n} (a_i - a'_i)^2}$$

Since there are k attributes A whose value is out of the boundary (i.e., less than the minimum or greater than the maximum of values on A). On the i-th attribute of A, $a'_i < min_i - d/\sqrt{k}$ or $a'_i > max_i + d/\sqrt{k}$, then it must be true that $(a_i - a'_i)^2 > d^2/k$. Thus

$$dist(t, o) > \sqrt{k * d^2/k} = d.$$

Then we prove that o is a (p, d)-outlier to D. This is straightforward as o is of distance at least d to all tuples of S. Since S takes $p\%$ tuples of T, o must be a (p, d)-outlier in T. It is straightforward that if $S = D$, then o must be a $(100\%, d)$-outlier. Then the theorem follows. ▮

Based on Theorem 6.1, we design the AO construction procedure as follows. First, the auditor module randomly picks $p\%$ of tuples from D as a sample S. Second, the auditor module treats S as an n-dimension hypercube $\mathcal{H}$. In $\mathcal{H}$, the edge at the i-th dimension represents the range $[min_i, max_i]$ of the data values in S. Then the auditor module randomly picks k dimensions (possibily $k = 1$) of $\mathcal{H}$. Last, the auditor module expands the k dimensions by $\frac{d}{\sqrt{k}}$ (i.e., change the minimum and maximum value of the i-th attribute to be $min_i - \frac{d}{\sqrt{k}}$ and $max_i + \frac{d}{\sqrt{k}}$). Let the expanded hypercube be $\mathcal{H}'$. Then any tuple O that is created out of $\mathcal{H}'$ must be a (p, d)-outlier of D. Figure 4(a) illustrates the construction procedure in a 2-dimension.

The pseudo code of our AO construction procedure is shown in Algorithm 1. The complexity of the algorithm is $O(nk)$, where n and k are the number of the tuples and the number of attributes of the dataset.

To construct valid AOs (i.e., with outlierness of AOs and true outliers well preserved when AOs are put into the database), we require that the distance between every two AOs must be less than d. We also construct a few $(100\%, d)$-outliers (i.e., the distant tuples of the database D) that take k% of AOs. More details of how these requirements can achieve validity of AOs will be in Section 6.3. We also will show that k must be greater than p in Section 6.3. It is true that the attacker is able to identify $(100\%, d)$-outliers as AOs. To achieve the same degree of integrity guarantee, we can add $|AO| * k\%$ artificial outliers that are (p, d)-outliers, so the total number of AOs that are (p, d)-outliers does not decrease.

Algorithm 1 AO Construction

```
Require: The database $D$, the parameters $p, d$ for $(p, d)$-outliers;
Ensure: Construct $m$ artificial $(p, d)$-outliers
 1: $Output = \{\}$; $o = \{\}$.
 2: Randomly pick $p\%$ of tuples of $D$. Let the picked tuples be $S$.
 3: Randomly pick $k$ attributes of $D$.
 4: repeat
 5:   for all the $i$-th picked attribute $A_i$ of $S$ do
 6:     Find $min$ and $max$, the min and max values of $A$ in $S$.
 7:     if Coin tossing returns positive then
 8:       randomly generate a value $v_i$ such that $v_i < (min_i - d/\sqrt{k})$ or $v > (max_i + d/\sqrt{k})$;
 9:       Store $min$ and $max$ in synopsis.
10:     else
11:       randomly generate a value $v_i \in [min_i, max_i]$;
12:     $o[i] = v_i$;
13:   if $\exists o' \in Output$ s.t. $dist(o, o') \leq d$ then
14:     Add $o$ to $Output$;
15: until $|Output| = m$
```

6.2 Artificial Non-Outlier Construction

Our ANO construction procedure is based on the concept of *close* tuples.

Definition 6.2. **[Close Tuple]** *Given the database D and a tuple $t \in D$, let $t_a \in D$ be a tuple whose distance to t is the largest out of all tuples whose distance to t is less than d, and $t_b \in D$ be a tuple whose distance to t is the smallest out of all tuples whose distance to t is greater than d. Let d_a and d_b be the distance between t and t_a and the distance between t and t_b. Let P be a partition that uses t as the centroid, and $r = min(\frac{d-d_a}{2}, \frac{d_b-d}{2})$ as the radius. Then we call any tuple $t \in P$ a* close *tuple to t.*

Next, we prove that the close tuples always have the same distance property as t.

Lemma 6.1. Given a tuple t and its close tuple t_c, for each tuple $t' \neq t$, it must be true that:

1. if $dist(t, t') > d$, then $dist(t_c, t') > d$;

2. if $dist(t, t') < d$, then $dist(t_c, t') < d$;

Proof: We have $dist(t, t') \leq d_a < d$, and $dist(t, t_c) < \frac{d-d_a}{2}$, leading to $dist(t', t_c) < dist(t, t') + dist(t', t_c) < d_a + \frac{d-d_a}{2} = \frac{d+d_a}{2}$. Since $d_a < d$, then it must be true that $dist(t', t_c) < d$. Similarly, we have $dist(t, t') \geq d_b > d$, and $dist(t, t_c) \geq dist(t, t') - dist(t', t_c) \geq \frac{d_b-d}{2}$. Thus,

$dist(t_c, t') \geq dist(t, t') - dist(t, t_c) \geq d_b - (d_b - d)/2 = (d_b + d)/2$. Since $d_b > d$, then it must be true that $dist(t', t_{no}) > d$. ▮

Based on Lemma 6.1, we show that the close tuples of tuple t always have the same (non)-outlierness property as t.

THEOREM 6.2. Given a tuple t, if it is a (non)outlier, all of its close tuples must be (non)outliers.
Proof: Let t_c be a close tuple of t. First, we prove that if t is a non-outlier, then t_c is a non-outlier, i.e., there exist at least $(1 - p\%) * |D|$ tuples in D whose distance to t_c is less than d. Since t is a non-outlier, there exist at least $(1 - p\%) * |D|$ tuples T in D whose distance to t is less than d. Now we prove each tuple $t' \in T$ whose distance to t_c is also less than d. This is straightforward from Lemma 6.1. ▮

We make use of Theorem 6.2 to construct the non-outlier tuples. We pick a *seed* tuple t_{seed} that is a non-outlier, and create its close tuples as ANOs. Figure 4(b) illustrates the construction procedure of ANOs. We require that the distance between each pair of ANOs must be less than d. The reasoning behind these requirements will be explained in Section 6.3.
Seed Tuple. A naive approach to identify a seed tuple is to repeatly pick a tuple from the database randomly, and verify its non-outlierness against the original database, until a non-outlier tuple is reached. Such approach can be computationally complex and time costly. Therefore, we aim at constructing a non-outlier tuple, instead of picking a tuple from the original database, as the t_{seed}. In particular, we compute a tuple $t_{avg}(a_1, \ldots, a_n)$, where $a_i (1 \leq i \leq n)$ is valued as the average of the data values at attribute A_i in the original database. To compute the average, we can use the arithmetic mean for numerical values, and the mode for categorical values.

We evaluated the non-outlierness of t_{avg} by using the Letter dataset from UCI machine learning dataset repository [2]. The dataset contains $20K$ tuples. We used $p = 50\%$, 60%, $70\%, 80\%$ and 90%. For each p value, we decide appropriate d values that return 0.1%, 0.5%, 1%, 5%, and 10% tuples of the original database as outliers. Our results show that for these 25 setting of (p, d) values, t_{avg} is always a non-outlier. This gives us more confidence of using t_{avg} as the seed for the construction of ANOs.

The computation of t_{avg} needs to traverse the whole database once. The computation of the radius of partition P needs another traversal of the database. Therefore, we can construct ANOs with two passes of the original databases.

To construct valid ANOs, we require that the distance between every two ANOs must be less than d, and the distance between every pair of AO and ANO must exceed d. The former requirement can be easily achieved when constructing AOs and ANOs. To achieve the latter requirement, we slightly modify the construction procedure of AOs as following: the sample S in the AO construction procedure will include all tuples in the partition P used in the ANO construction procedure. By doing this, as AOs are guaranteed to be of the distance greater than d to all tuples in P, these AOs must be of the distance greater than d to all ANOs.

[2]http://archive.ics.uci.edu/ml/datasets/Letter+Recognition

6.3 Preservation of (non)outlierness

Our construction procedures in Section 6.1 and 6.2 only ensure the (non)outliers of the artificial tuples if they are inserted into the original database alone. Next, we discuss how to satisfy the validity requirement, i.e., the (non)outlierness is still well preserved for both artificial and real tuples after the artificial tuples are inserted together into the original database.
Preservation of Outlierness of AOs. To preserve the outlierness of AOs, we have:

THEOREM 6.3. Given a database D, let AO and ANO be the set of AOs and ANOs that are inserted into D. Then if $|ANO| \geq \frac{p\% - k\%}{1 - p\%} * |AO|$, each outlier in AO is still an outlier in $D \cup AO \cup ANO$.
Proof: For each artificial outlier $o \in AO$, we define three types of tuples in $D \cup AO \cup ANO$.

- The set of tuples D' in D whose distance to o is greater than d. Let $m = |D'|$. It is straightforward that $m/|D| \geq p\%$.
- The set of tuples $AO' \subseteq AO$ whose distance to o is greater than d. Since we require that the distance between each pair of AOs that are not $(100\%, d)$-outliers is less than d, $|AO'| = k\% * |AO|$, where k is the percentage of $(100\%, d)$-outliers that are AOs, because according to our construction procedure, the distance between AOs that are not $(100\%, d)$-outliers and $(100\%, d)$-outliers must be greater than 0.
- The set of tuples $ANO' \subseteq ANO$ whose distance to o is greater than d. Since we require that the distance between o and any ANO always exceeds d, $|ANO'|$ $=|ANO|$.

Therefore in $D \cup AO \cup ANO$, there are $x * |D| + k * |AO| + |ANO|$ number of tuples whose distance to o is greater than d. To help explanation, we denote $|D| = n$, $|D'| = m$, $|AO| = f_1$, and $|ANO| = f_2$. Our goal is to make

$$\frac{m + f_2}{n + f_1 + f_2} \geq p\%. \tag{1}$$

It is straightforward that

$$\frac{m + f_1 * k\% + f_2}{n + f_1 + f_2} \geq p\%$$

$$\rightarrow (1 - p\%) f_2 \geq n * p\% - m + (p\% - k\%) * f_1$$

To ensure Formula 1 holds even in the worst case that $n * p\% = m$, it must be true that $|ANO| > |AO| * \frac{p\% - k\%}{1 - p\%}$. ▮

Preservation of Outlierness of True Outliers. To preserve the outlierness of true outliers, we have:

THEOREM 6.4. Given a database D, let AO and ANO be the set of AOs and ANOs that are inserted into D. Then if $|ANO| \leq \frac{(k\% - p\%)|AO|}{p\%}$, each (p, d)-outlier in D is still a (p, d)-outlier in $D \cup AO \cup ANO$.
Proof: Given a true outlier $o \in D$, we define three types of tuples:

- The set of tuples D' in D whose distance to o is greater than d. Since t is a (p, d)-outlier in D, $|D'|/|D| \geq p\%$.

- The set of tuples in AO whose distance to o is greater than d. We assume it takes $x\%$ of tuples in AO. Since there are $k\%$ tuples of AO that are $(100\%, d)$-outliers, $k\% \le x \le 100\%$.

- The set of tuples in ANO whose distance to o is greater than d. We assume it takes $y\%$ of tuples in ANO.

Therefore in $D \cup AO \cup ANO$, there exist $|D'| + x\%|AO| + y\%|ANO|$ of tuples whose distance to o is greater than d. We denote $|D| = n$, $|D'| = m$, $|AO| = f_1$ and $|ANO| = f_2$.

We discuss two possible cases of the tuple o. Let t_{avg} be the seed tuple that is used for ANO construction.

1. If $dist(o, t_{avg}) \ge d$. In this case, following Lemma 6.1, the distance between all ANOs and t must be greater than d. Therefore, $y = 100\%$. Thus we need to ensure that $\frac{m+x\%f_1+f_2}{n+f_1+f_2} \ge p\%$. We can infer that it must be true that $(1-p)f_2 \ge np - m + (p-x)f_1$ even for the worst case that $n*p\% = m$ and $x = k\%$, the percentage of $(100\%, d)$-outliers in AOs. Then we can infer that $|ANO| \ge \frac{(p\%-k\%)|AO|}{1-p\%}$.

2. if $dist(o, t_{avg}) < d$. In this case, following Lemma 6.1, the distance between all ANOs and o must be less than d. Therefore, $y = 0\%$. Thus we need to ensure that $\frac{m+x\%f_1}{n+f_1+f_2} \ge p\%$ even for the worst case that $n*p\% = m$. Then we can infer that $|ANO| \le \frac{(k\%-p\%)|AO|}{p\%}$.

If the condition of Case 2 holds, then $k > p$. Thus, the condition of Case 1 is also automatically satisfied. Therefore, the result follows. ∎

Theorem 6.4 shows that it must be true that $k > p$, i.e., the percentage of $(100\%, d)$-outliers out of AOs must exceed $p\%$. This confirms the need of $(100\%, d)$-outliers.

Preservation of non-outlierness of ANOs. To preserve the non-outlierness of ANOs, we have:

THEOREM 6.5. Given a database D, let AO and ANO be the set of AOs and ANOs that are inserted into D. Then if $|ANO| > \frac{(1-p\%)|AO|-1}{p\%}$, each non-outlier in ANO is still a non-outlier in $D \cup AO \cup ANO$.

Proof: For each artificial non-outlier $t \in ANO$, we define three types of tuples in $D \cup AO \cup ANO$.

- the set of tuples D' in D whose distance to t is greater than d. Since t is not a (p, d)-outlier in D, it must be true that $|D'|/|D| < p\%$.

- the set of tuples $AO' \subseteq AO$ whose distance to t is greater than d. Following our artificial non-outlier and outlier construction procedure, the distance between t and any tuple in AO always exceeds d, i.e., $|AO'| = |AO|$.

- the set of tuples $ANO' \subseteq ANO$ whose distance to t is greater than d. Since we require that the distance between each pair of artificial non-outliers is less than d (Section 6.2), $|ANO|' = 0$.

In $D \cup AO \cup ANO$, there are $|D'| + |ANO|$ number of tuples whose distance to f is greater than d. To help explanation, we define $|D| = n$, $|D'| = m$, $|AO| = f_1$, and $|ANO| = f_2$. Note that $m < n * p\%$. Our goal is to make $\frac{m+f_1}{n+f_1+f_2} < p\%$ hold. It is straightforward that

$$\begin{aligned} & \frac{m+f_1}{n+f_1+f_2} < p\% \qquad (2) \\ \rightarrow \quad & m + f_1 < p\% * (n + f_1 + f_2) \\ \rightarrow \quad & m - p\% * n + (1-p\%) * f_1 < p\% * f_2 \end{aligned}$$

Since $m/n < p\%$, it must be true that $(m - p\% * n)_{max} = -1$ (we assume that $p\% * n$ always returns an integer, as it indicates the number of tuples). Thus $m - p\% * n + (1 - p\%) * f_1 \le (1-p\%) * f_1 - 1$. Then if $p\% * f_2 > (1-p\%) * f_1 - 1$ always holds, then the inference in Equation 2 must be true. It is easy to infer that $f_2 > \frac{(1-p\%)*f_1-1}{p\%}$. ∎

Preservation of non-outlierness of true non-outliers. To preserve the non-outlierness of true non-outliers, we have:

THEOREM 6.6. Given a database D, let AO and ANO be the set of artificial outliers and non-outliers that are inserted into D. Then if $|ANO| \in (\frac{(1-p\%)|AO|-1}{p\%}, \frac{(p\%-1)|AO|+1}{1-p\%})$, each tuple that is not a (p, d)-outlier in D is not a (p, d)-outlier in $D \cup AO \cup ANO$. ∎

Proof: Given a true non-outlier $t \in D$, we define three types of tuples:

- The set of tuples D' in D whose distance to t is greater than d. Since t is not a (p, d)-outlier in D. $|D'|/|D| < p\%$.

- The set of tuples in AO whose distance to t is greater than d. We assume it takes $x\%$ of tuples in AO.

- The set of tuples in ANO whose distance to t is greater than d. We assume it takes $y\%$ of tuples in ANO.

Therefore in $D \cup AO \cup ANO$, there exist $|D'| + x\%|AO| + y\% * |ANO|)$ of tuples whose distance to t is grater than d. We denote $|D| = n$,$|D'| = m$, $|AO| = f_1$ and $|ANO| = f_2$.

We discuss two possible cases of the tuple t. Let t_{avg} be the seed tuple that is used for ANO construction.

- If $dist(t, t_{avg}) \ge d$. In this case, following Lemma 6.1, the distance between all ANOs and t must be greater than d. Therefore, $y = 100\%$. Thus we need to ensure that $\frac{m+x\%f_1+f_2}{n+f_1+f_2} < p\%$. We can infer that it must be true that $(1-p)f_2 < n * p\% - m + (p-x)f_1$ even for the worst case that $p\% * n - m = 1$ (we assume that $p\% * n$ always returns an integer, as it indicates the number of tuples) and $x = 100\%$. Then we can infer that $|ANO| < \frac{(p\%-1)|AO|+1}{1-p\%}$.

- if $dist(t, t_{avg}) < d$. In this case, following Lemma 6.1, the distance between all ANOs and t must be less than d. Therefore, $f_2 = 0$. Thus we need to ensure that $\frac{m+x\%f_1}{n+f_1+f_2} < p\%$ even for the worst case that $n*p\% - m = 1$ (we assume that $p\% * n$ always returns an integer, as it indicates the number of tuples) and $x = 100\%$. Then we can infer that $|ANO| > \frac{(1-p\%)|AO|-1}{p\%}$. ∎

By further inference on Theorem 6.6, to ensure that $\frac{(p\%-1)|AO|+1}{1-p\%} > 0$, it must be true that $|AO| > \frac{1}{1-p\%}$.

Putting all together. Based on Theorem 6.3, 6.4, 6.5, and 6.6, we can quantify the relationship between the number of

AOs and $ANOs$. In particular, to preserve the (non)outlierness of AOs and ANOs when they are put together, their sizes must satisfy the following:

Based on Theorem 6.3, 6.4, 6.5, and 6.6, we can quantify the relationship between the number of AOs and $ANOs$. In particular, we have the following theorem:

THEOREM 6.7. Given a database D, let AO and ANO be the AOs and ANOs that are inserted into D. Let k be the percentage of AOs that are $(100\%, d)$-outliers. Then if: (1) $|AO| > \frac{1}{1-p\%}$, (2) $k \geq p$, and (3) $|ANO| \in (\frac{(1-p\%)|AO|-1}{p\%}, \frac{(p\%-1)|AO|+1}{1-p\%})$, the (non)-outlierness of all tuples, including tuples in D and tuples in AO and ANO, is preserved in $D \cup AO \cup ANO$.

Proof: Condition (1) comes from Theorem 6.6. Condition (2) comes from Theorem 6.4. The lowerbound of $|ANO|$ in Condition (3) is straightforward from Theorem 6.5 and 6.6. To infer the upperbound of $|ANO|$, we compare $\frac{(k\%-p\%)*|AO|}{p\%}$ (in Theorem 6.4) and $\frac{(p\%-1)|AO|+1}{1-p\%}$ (in Theorem 6.6). As $|AO| \geq \frac{1}{1-p\%}$ and $k > p$, we can easily prove that $\frac{1}{1-p\%} > \frac{p\%}{k\%(1-p\%)}$. Therefore, it must be true that $\frac{(k\%-p\%)*|AO|}{p\%} > \frac{(p\%-1)|AO|+1}{1-p\%}$. The results then follow. ▌

6.4 α-completeness and β-correctness

To satisfy α-complete, it must be true that $1 - \frac{\binom{m}{k}}{\binom{x+m}{k}} > \alpha$, where m is the number of true outliers in D, x is the number of AOs, and k is the number of true outliers that are not included in the returned answer by the service provider. Obviously $k \leq m$. Now we want to quantify the value of x, the number of AOs that are needed to satisfy α-completeness. It is equivalent to

$$(1-\alpha)\binom{x+m}{k} > \binom{m}{k}. \tag{3}$$

Define

$$g(x) = (1-\alpha)\binom{x+m}{k} - \binom{m}{k}.$$

Obviously, $g(x)$ is an strictly increasing function with respect to x. Then, we define

$$x^* = min\{x : g(x) > 0\}.$$

Thus, the range of x value satisfying Equation 3 should be all positive integers that are greater than or equal to x^*.

In the following, we discuss how to calculate x^*. Define $b_i = \frac{m+i-k}{m+i}$ for each positive integer i. Thus, when $x = 1$, Equation 3 is equivalent to $1 - \alpha > \frac{m+1-k}{m+1} = b_1$. For the worst case that $k = m$ (i.e., the service provider returns none of the true outliers), $b_1 = \frac{1}{m+1}$. When $x = 2$, Equation 3 is equivalent to $1-\alpha > \frac{(m+2-k)(m+1-k)}{(m+2)(m+1)} = b_1 b_2$. Again, for the worst case that $k = m$, $\frac{2}{m+2} = b_2$. Generally, when $x = j$, Equation 3 is equivalent to $1 - \alpha > \Pi_{i=1}^{j} b_i$. Therefore, x^* satisfies the following

$$x^* = min\{j : \Pi_{i=1}^{j} b_i < 1 - \alpha\}. \tag{4}$$

It is easy to see that x^* in Equation 4 can be calculated iteratively.

Similarly, we can compute the number y of ANOs that are needed to satisfy β-correctness, i.e.,

$$(1-\beta)\binom{|D|-m+y}{k} > \binom{|D|-m}{k}. \tag{5}$$

To make the explanation easier, we use n to denote the size of the original database.

We define

$$g(y) = (1-\beta)\binom{y+n-m}{k} - \binom{n-m}{k}.$$

As $g(y)$ is an strictly increasing function with respect to y, we define

$$y^* = min\{y : g(y) > 0\}.$$

Thus, the range of y value satisfying Equation 5 should be all positive integers that are greater than or equal to y^*.

To calculate y^*, we define $b_i = \frac{n-m+i-k}{n-m+i}$ for each positive integer i. Similar to the reasoning of α-completeness, we have

$$y^* = min\{j : \Pi_{i=1}^{j} b_i < 1 - \beta\}. \tag{6}$$

Obviously, y^* in Equation 6 can be calculated iteratively.

7. VERIFICATION AT CLIENT SIDE

Auxiliary information at client Side. The auditor module will maintain the following auxiliary information for integrity auditing: (1) the size of the database D, (2) the hash function H and the three constants c_1, c_2, and c_3 used to construct the signatures, and (3) the number of AOs and ANOs. It is straightforward that the space overhead of these auxiliary information is negligible.

Verification procedure. After the service provider returns the outlier mining results to the data owner, the auditor module will run the 2-phase verification procedure as following:

Phase-1. For each returned outlier tuple t, the auditor module will re-compute the signature Sig_a by applying the stored hash function H (Section 4) on t. If the computed signature does not match the one that is attached to t, the auditor module will conclude that the server has returned tuples that do not exist in the hosted database, and thus fails to pass authenticity auditing.

Phase-2. If the server passed the phase-1 auditing, the auditor module will further compute three signatures: $Sig_t^1 = H(Sig_a \oplus c_1)$, $Sig_t^2 = H(Sig_a \oplus c_2)$, and $Sig_t^3 = H(Sig_a \oplus c_3)$, by using the three constants c_1, c_2, and c_3 that the module has used and stored locally. The auditor module will compare these three signatures with the attached Sig_t, and identify whether the returned tuple is a true tuple (a true outlier), an artificial outlier, and an artificial non-outlier. After the auditor module identified all returned tuples, it will compute the probabilistic guarantee of both completeness and correctness as explained in Section 5.

8. EXPERIMENTAL EVALUATION

We ran an extensive set of experiments to evaluate both the assurance guarantee and performance of our approach. In particular, we measured: (1) the completeness and correctness guarantee, (2) the verification overhead at the client side, including the construction time and space of AOs and ANOs, and (3) the mining overhead at the server side.

8.1 Setup

All of our experiments are evaluated on a desktop with a 3GHz Intel Core i7 CPU and 8GB RAM running Windows 7. We implemented the algorithm in Java. For each experiment that measured the time performance, we ran 5 times and took the average.

8.2 Dataset

We experiment with two datasets, *Letter* dataset [3] from UCI MLC++ Library that contains 20k tuples, and *KDDCUP* dataset [4] that contains 494k tuples. Letter dataset has 16 numerical (integer) attributes, and KDDCUP dataset contains 41 (numerical or categorical) attributes. In our experiments, we used all 16 attributes of Letter dataset, and used only five numerical attributes, including `duration`, `dst_bytes`, `flag`, `count`, and `serror_rate`, of KDDCUP dataset, when we measured the distances for outlier mining.

8.3 Probablistic Guarantee

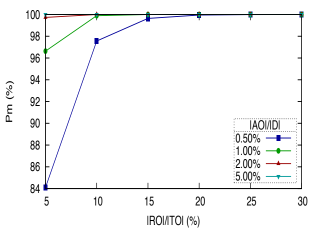
Completeness Guarantee. Figure 5 (a) and (b) show that even if the service provider removes a small portion of original outliers from the result, there is a high probability to catch him/her with a small number of artificial outliers needed. For example, removing 5% of true outliers from Letter dataset can be caught with 82% belief probability by inserting artificial outliers that take 0.5% of the original database. The same observation also holds for *KDDCUP* dataset that is of larger size; the action of removing 0.1% of true outliers can be caught with more than 95% probability by inserting AOs whose size takes 1% of the original database. For both datasets, inserting more AOs will increase the completeness probability. After AOs take 2% of the original dataset (for both Letter dataset and KDDCUP dataset), our approach can reach 100% probability of completeness guarantee.

Correctness Guarantee. We also measured the correctness guarantee probability P_r. Figure 5 (c) and (d) illustrate the results. We observe that even if the service provider inserts a small portion of non-outliers as outliers, we still have a high probability to catch the cheating behavior by injecting a small portion of ANOs. For example, for *KDDCUP* dataset, inserting non-outliers that take 10% of true outliers can be caught with 87% probability by inserting ANOs that take 2% of the original database. The correctness guarantee probability increases when inserting more ANOs. This trend holds on both datasets. When the number of ANOs reaches 5% of original databases for both Letter and KDDCUP datasets, we can archieve 100% probability of correctness guarantee.

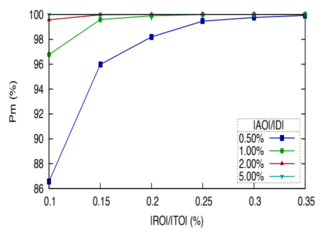
8.4 Cost Analysis at Client Side

Construction Time. We first measure the construction time of the AOs at the client side. Figure 6 (a) and (b) illustrate the *AO* construction time (excluding the time to calculate the number of outliers to satisfy α-completeness guarantee) regarding various percentages of AOs compared

[3]http://www.sgi.com/tech/mlc/db/letter.all
[4]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

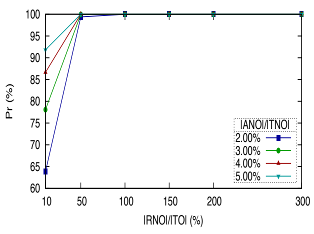


(a) Letter dataset

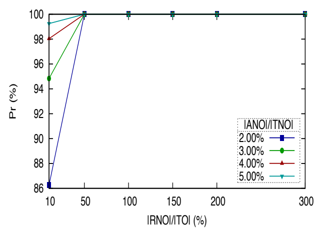


(b) KDDCUP dataset

Completeness guarantee



(c) Letter dataset



(d) KDDCUP dataset

Correctness guarantee

Figure 5: Probablistic Guarantee (RO: removed outliers; TO: true outliers; TNO: true non-outliers (as outliers); D: original database)

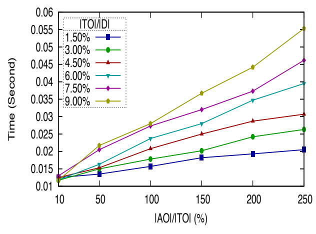
with the number of true outliers. The results show that the construction time increases as the number of AOs increases. However, even if the number of AOs is large ($111K$ as the maximum in our experiments), the construction time is still less than 0.25 seconds.

Next, we measure the construction time of the ANOs at the client side. Figure 6 (c) and (d) illustrate the performance results. The observation is similar to AO construction; the construction time increases with larger ANOs. However,the construction time is always negligible (within 1.3 seconds) even the ANOs size is very large ($120K$ as maximum).

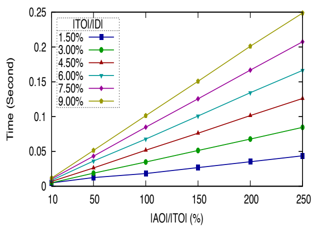
α-completeness and β-correctness. We also measured the time to generate AOs and ANOs to achieve given α-completeness and β-correctness requirements. It turned out that the generation time is always negligible; for $\alpha = 0.95$, the generation time of AOs for both Letter dataset and KDDCUP dataset is 0.15 seconds and 1.3 seconds, while for $\beta = 0.95$, the generation time of ANOs is at most 0.29 seconds and 2.9 seconds for Letter dataset and KDDCUP dataset respectively. We omit the results due to the space limit.

Next, we measured the number of AOs and ANOs according to various α and β requirements. Figure 7 illustrates the number of AOs that are needed to achieve α-completeness for both datasets. We varied p and d values and repeated the measurement. The results are similar. We observe that higher α values requires more AOs. However, even for the largest α value 0.95, the number of needed AOs only takes at most 0.4% of the original database. This shows that our auditing approach is lightweight, as it does not require large amounts of AOs to achieve high α completeness guarantee.

Figure 8 illustrates the number of ANOs needed to satisfy β-correctness. Similar to the case of AOs, higher β values need more ANOs. We also observe that the needed ANOs is significantly larger than that of AOs, given α and β being assigned the same values. This is because the number of true non-outliers (that the attacker can pick and pretend as outliers) are dramatically larger than the true outliers; there-

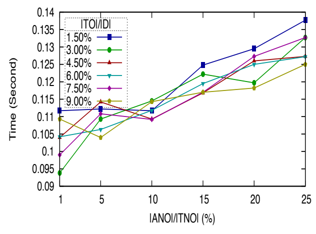


(a) Letter dataset

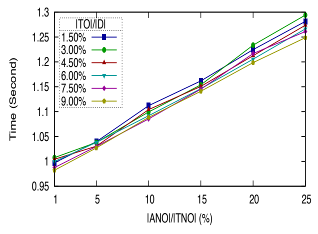


(b) KDDCUP dataset

AOs



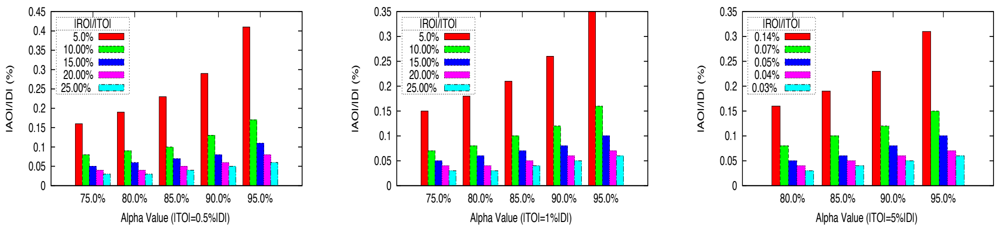
(c) Letter dataset



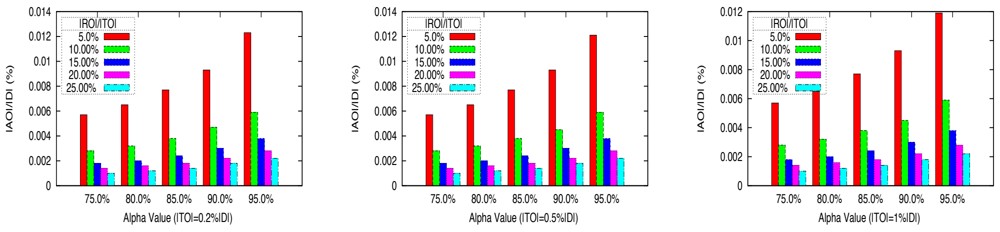
(d) KDDCUP dataset

ANOs

Figure 6: Construction Time (RO: removed outliers; TO: true outliers; TNO: true non-outliers (returned as outliers); D: original database)

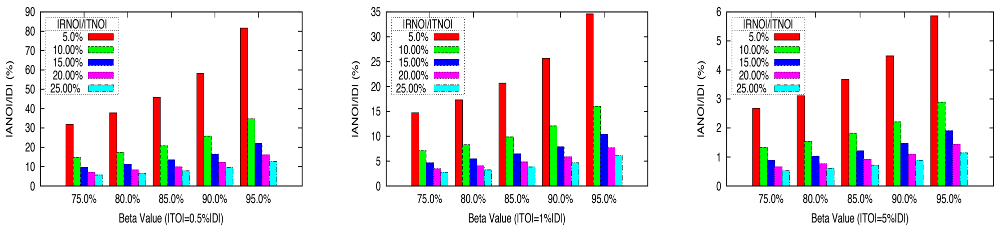


(a) Letter dataset

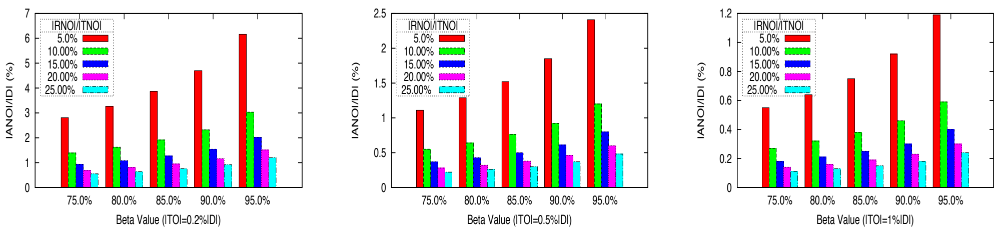


(a) KDDCUP dataset

Figure 7: Number of AOs and ANOs to satisfy α-completeness (RO: removed outliers; TO: true outliers; TNO: true non-outliers (returned as outliers); D: original database)

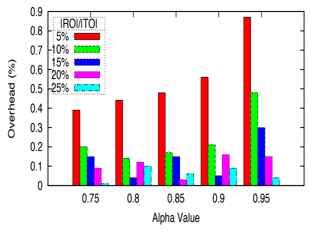


(a) Letter dataset

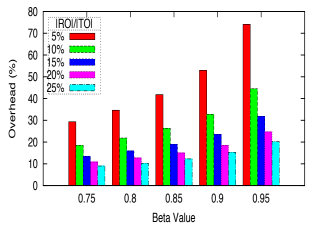


(a) KDDCUP dataset

Figure 8: Number of AOs and ANOs to satisfy β-correctness (RO: removed outliers; TO: true outliers; TNO: true non-outliers (returned as outliers); D: original database)



(a) Various α values



(b) Various β values

Figure 9: Mining overhead

fore verifying correctness with the same amount of guarantee needs more ANOs.

8.5 Overhead at Server Side

In our experiment, we gauged the additional time for the service provider to apply outlier mining on the dataset D' that contains both AOs and ANOs. In this experiment, we only used Letter dataset. The overhead is measured as $|T_{D'} - T_D|/T_D$, where $T_{D'}$ and T_D are the time of mining outliers from the original database D and the dataset D'. Figure 9 (a) shows that it takes at most additional 0.9% of the original mining time for datasets that satisfy α-completeness requirement, with α varies from 0.75 to 0.95. On the other hand, Figure 9 (b) shows that the time overhead for datasets that satisfy β-correctness is considerably large; it takes at most additional 70% of the original mining time when β is equal to 0.95. Since such overhead occurs only at the server side, it will not bring burden to the framework, especially to the data owner.

9. CONCLUSION

Outsourcing of outlier mining arises serious concern on quality assurance of mining results. In this paper, we proposed a lightweight signature scheme that can verify the authenticity of the returned outlier tuples. We also proposed a practical scheme that can verify the correctness and completeness of the outlier mining results with probabilistic guarantee. We designed efficient approaches of constructing artificial tuples for verification purpose, and discussed how to achieve both requirements of validity of the artificial tuples and bounded completeness and correctness guarantee.

In the future, we will work on updates on the databases. We will also extend our work to cover other types of outliers, e.g., density-based outliers [6].

10. REFERENCES

[1] Microsoft inc., data mining in dmcloud. http://www.microsoft.com/azure/mining.mspx.

[2] Oracle inc., oracle data mining in the cloud. http://www.oracle.com/technetwork/database/options/odm/odm-education-101260.html.

[3] C. C. Aggarwal and P. S. Yu. Outlier detection for high dimensional data. In *SIGMOD*, 2001.

[4] S. Bakhtiari, R. Safavi-naini, J. Pieprzyk, and C. Computer. Cryptographic hash functions: A survey. Technical report, University of Wollongong, 1995.

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

[6] M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *SIGMOD*, 2000.

[7] P. T. Devanbu, M. Gertz, C. U. Martel, and S. G. Stubblebine. Authentic third-party data publication. In *Proceedings of the IFIP TC11/ WG11.3 Fourteenth Annual Working Conference on Database Security: Data and Application Security, Development and Directions*, 2001.

[8] F. Giannotti, L. V. Lakshmanan, A. Monreale, D. Pedreschi, and H. Wang. Privacy-preserving mining of association rules from outsourced transaction databases. In *The workshop on Security and Privacy in Cloud Computing (SPCC2010).*

[9] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *SIGMOD*, 2002.

[10] E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB*, 1998.

[11] E. M. Knorr, R. T. Ng, and V. Tucakov. Distance-based outliers: Algorithms and applications. In *VLDB Journal*, volume 8, pages 237–253, 2000.

[12] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic authenticated index structures for outsourced databases. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD, 2006.

[13] I. Molloy, N. Li, and T. Li. On the (in)security and (im)practicality of outsourcing precise association rule mining. In *ICDM*, 2009.

[14] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. *Trans. Storage*, 2, May 2006.

[15] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan. Verifying completeness of relational query results in data publishing. In *SIGMOD*, 2005.

[16] L. Qiu, Y. Li, and X. Wu. Protecting business intelligence and customer privacy while outsourcing data mining tasks. *Knowledge Information System*, 17(1), 2008.

[17] S. Ramaswamy, R. Rastogi, K. Shim, and Aitrc. Efficient algorithms for mining outliers from large data sets. In *SIGMOD*, 2000.

[18] R. Sion. Query execution assurance for outsourced databases. In *VLDB*, 2005.

[19] C.-H. Tai, P. S. Yu, and M.-S. Chen. k-support anonymity based on pseudo taxonomy for outsourcing of frequent itemset mining. In *SIGKDD*, 2010.

[20] W. K. Wong, D. W. Cheung, E. Hung, B. Kao, and N. Mamoulis. Security in outsourcing of association rule mining. In *VLDB*, 2007.

[21] W. K. Wong, D. W. Cheung, B. Kao, E. Hung, and N. Mamoulis. An audit environment for outsourcing of frequent itemset mining. In *PVLDB*, volume 2, pages 1162–1172, 2009.

[22] M. Xie, H. Wang, J. Yin, and X. Meng. Integrity auditing of outsourced data. In *VLDB*, 2007.