PRIVACY PRESERVING OUTLIER DETECTION OVER VERTICALLY PARTITIONED DATA

Thesis submitted in partial fulfillment

of the requirements for the degree of

Master of Science (by Research)

in

Computer Science and Engineering

by

Madhuchand Rushi Pillutla

200907010

rushi.pillutla@research.iiit.ac.in

International Institute of Information Technology

Hyderabad - 500 032, INDIA

October 2016

International Institute of Information Technology

Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled " Privacy Preserving Outlier Detection over Vertically Partitioned Data" by Madhuchand Rushi Pillutla, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Adviser: Dr. Kannan Srinathan

To my Grandparents

Acknowledgments

I would first like to express my sincere gratitude to my thesis advisor Dr. Kannan Srinathan for the constant feedback and motivation he provided and for giving me the oppurtunity to explore new domains and ideas. I thank my co-authors Piyush Bansal and Nisarg Raval for their support and feedback. I also thank all my colleagues at C-STAR lab and my batch mates at IIIT for making my stay at IIIT memorable. Finally, I thank my family for their constant support and encouragement.

Abstract

Consider a scenario wherein a bank with a large database of customer information intends to identify the set of customers to whom a credit line could be extended. For this, the bank might need various other information, apart from its own database, such as tax information from the income tax department, criminal history of the customers from the law enforcement authorities etc. Although all the other parties might be willing to share their respective information (databases) in order for the banking organization to compute the result, they might not be willing to compromise on the privacy of their databases. To preserve the privacy of the individual data, in an idealistic world, all parties could send their data to a Trusted Third Party (TTP) which would then apply any of the data mining techniques to compute the intended result and share it with the banking organization. The bank would only get to know the final set of customers to whom a line of credit could be extended and any local information with other parties would not be revealed other than what could be computed from the final result itself. However, in the real world, in many instances it would be infeasible to find such a Trusted third party with whom all the other sources of data would be willing to share their information. To overcome such problems, Privacy Preserving Data Mining techniques have been developed which allow the computation of a function on various databases distributed across a set of players, with out either party learning anything about the data of the other parties other than that can be derived from the final result itself. In other words, these techniques simulate the functionality of a TTP.

In this thesis, we consider the problem of Privacy Preserving outlier Detection(PPOD) over vertically partitioned data. The computational and communication complexities of the proposed PPOD algorithm are sub-quadratic in the size of the dataset as opposed to the quadratic complexities of the previously known results. In order to achieve efficient algorithms in the pri-

vate setting, we first develop an approximation algorithm for outlier detection in the centralized setting and then extend it to the case of privacy preserving outlier detection.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In the current age of information explosion, tremendous amounts of data is being generated through various channels such as the Internet and mobiles etc and advances in hardware technology has facilitated the storage of these large amounts of data by various organizations. However, storage of all this data would not be of much use unless meaningful information can be extracted from it. The emergence of Data Mining as a research area has answered this need through various techniques such as Classification, Clustering etc. But in recent years, data mining is increasingly being viewed as a threat to privacy of the data due to the sensitive nature of individual data being gathered. This has led to the emergence of Privacy Preserving Data Mining (PPDM) techniques whose aim is to preserve the privacy of individual data while mining the data.

Privacy In Data Mining

The need for privacy while mining the data arises in two classical settings (although may not be limited to these two cases). The first is where some statistical data has to be released containing confidential data, such as a hospital releasing the medical history of various patients so that it could be used in medical research by other organizations. Although this research could provide very useful information, the major detriment in releasing this data is the violation of the privacy of individual data. The aim of PPDM techniques in this setting is to facilitate the release of such sensitive data in a way that meaningful research could be carried on it but at

the same time which does not reveal individual data. The other setting where need for privacy of data arises is that of data being held by two or more players who would like to mine the union of their data to discover some meaningful information. The aim of PPDM techniques in this setting is to enable the players to apply data mining algorithms on their combined data while preserving the privacy of individual data i.e., the data held by one player should not be revealed to any other players in the process of extracting patterns from their combined data. The partitioning of data among multiple players could be in two different ways namely *Horizontal Partitioning* and *Vertical Partitioning*. In the former, each player has information about all the attributes for different objects where as in the latter, each player has information about a subset of attributes for all the objects in the dataset. Both kinds of partitioning pose different set of challenges while developing distributed privacy preserving algorithms.

Privacy Preserving Data Mining Methods

Privacy Preserving Data Mining techniques developed over the years are classified into two categories based on the underlying methods used for achieving privacy namely *Data Modification* and *Secure Multi-party Computation (SMC)*.

- **Data Modification:** Privacy preserving data mining algorithms based on Data modification techniques modify the original dataset before releasing to others. Data is modified in such a way that privacy is preserved in the released dataset while maintaining high quality of data so as to enable others to perform data mining and extract useful information. The aim is to maintain a balance between the level of privacy and the level of information that could be mined from the released dataset. Data modification methods could be developed to protect the privacy of individuals or confidential underlying data or both. These techniques include *noise addition*, *data swapping*, *aggregation* and *suppression*

  Noise addition, as the name implies, adds random noise to the original data. The noise added is sufficiently large so that individual data values can no longer be recovered. *Data swapping* interchanges the attribute values among different data objects (records). Similar attribute values are interchanged with more probability. *Aggregation* may in-

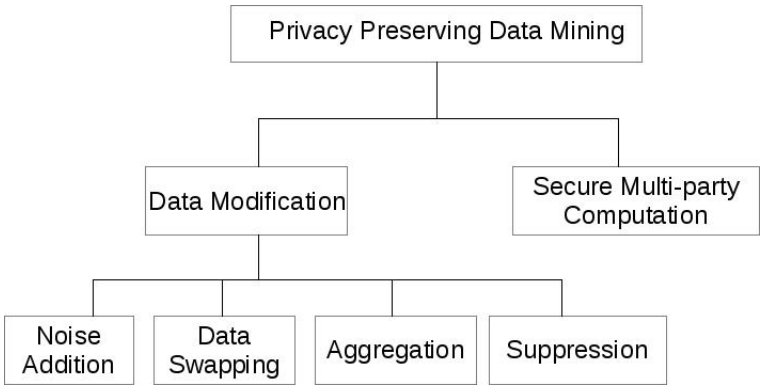


Figure 1.1 A Classification of Privacy Methods

volve either combining a few attribute values into one or grouping multiple data objects and replacing them with a group representative. *Suppression* involves replacing an attribute value in one or more objects by a missing value. Data modification based PPDM algorithms have the benefit that they are computationally efficient but suffer from the drawback that loss of information is high as the original data has been modified. Also, these methods do not provide formal guarantees for the level of privacy provided.

- **Secure Multi-party Computation:** Secure Multi-party Computation based PPDM algorithms use cryptographic tools to protect the privacy of data and are developed in the context of data partitioned across multiple players who wish to compute a function on their joint data. The aim of the *SMC* techniques is that no new information is revealed to any of the participating players other than what can be computed from the final result of the computation. These methods have the advantage that they provide formal guarantees for the level of privacy provided. But the cryptographic tools used usually involve very high computational and communication costs while performing distributed data mining.

In this work, we use *SMC* based techniques and consider the problem of Privacy Preserving Outlier Detection (PPOD) where two or more players wish to compute the outliers from the union of their individual data while preserving the privacy of the local data with each player. This problem was first studied by Vaidya *et al* in [31] where the authors use the definition for

distance based outliers provided in [18], and give PPOD algorithms for both horizontal and vertical partitioning of data. Subsequently, a PPOD algorithm using the k-nearest neighbor based definition [24] was given in [40] , considering only vertical partitioning. However, both of the above mentioned algorithms have quadratic communication and computation complexities in the database size, making them infeasible while dealing with large datasets. To the best of our knowledge, no other work in the field of PPDM based on cryptographic techniques has addressed distance based outlier detection.

Contributions of this Thesis: In this thesis, we propose approximate PPOD algorithms for the case of vertical partitioning of data based on cryptographic techniques. As opposed to the current PPOD algorithms which provide privacy for already existing (non-private) outlier detection algorithms, we develop a new outlier detection scheme for the centralized setting in order to achieve efficient algorithms in the private setting. The centralized scheme is based on the Locality Sensitive Hashing (LSH) technique [14] and could be of independent interest. We also give theoretical bounds on the level of approximation and provide the corresponding empirical evidence.

The computational complexity of our centralized algorithm is $O(ndL)$ for d dimensional dataset with n objects. The parameter L is defined as $n^{1/1+\epsilon}$, where $\epsilon > 0$ is an approximation factor. The computational complexity of the PPOD algorithm for vertically distributed data is same as that of the centralized algorithm which is a significant improvement over the previous know result of $O(n^2d)$. The communication complexity in vertically distributed setting is $O(nL)$ which is again an improvement from the quadratic complexity in dataset size.

1.1 Overview of our Approach

Our outlier detection scheme uses the definition for a distance based outlier proposed by Knorr *et al.*[18], given as below:

Definition 1 $DB(p_t, d_t)outlier$*: An object o in a dataset D is a $DB(p_t, d_t)$ outlier if at least fraction p_t of the objects in D lie at a distance greater than d_t from o.*

where $DB(p_t, d_t)$ is a shorthand notation for a Distance-Based outlier detected using parameters p_t and d_t. In our approach, we use the converse of this definition and say an object is a non-outlier if it has enough neighbors (p'_t) within distance d_t, where $p'_t = (1 - p_t) \times |D|$. Since the fraction p_t is very high (usually set to 0.9988), the modified point threshold p'_t will be very less compared to the number of objects in D. This allows us to easily detect most of the non-outliers by finding p'_t objects within distance d_t (d_t is almost comparable to the data spread).

To efficiently find the near neighbors, we use the Locality Sensitive hashing technique. Given a set of objects, the LSH scheme hashes all the objects in such a way that all those objects within a specified distance are hashed to the same value with a very high probability. This way, all those non-outliers in the data which have many near neighbors can be identified easily, without calculating the distances to every other object in the dataset. Moreover, using LSH properties, whenever we identify a non-outlier we will be able to say most of its neighbors as non-outliers without even considering those objects separately. Thus, we obtain a very efficient pruning technique where, most of the non-outliers in the dataset can be easily pruned and a very small percent of the objects in the dataset need to be processed after pruning. The remaining points after pruning are the set of probable outliers which will contain very few non outliers. To further remove these non outliers, we use the probabilistic nature of LSH. The idea is to take the intersection of the sets of probable outliers over multiple runs, to output the final set of approximate outliers. The approach works because each set of probable outliers will contain the actual outliers with extremely high probability, but the non outliers in each set will differ with high probability.

In case of privacy preserving outlier detection over vertically distributed data, all players first communicate to obtain the LSH binning of all the objects considering all the attributes. Next, the players locally compute the probable outliers using the global LSH binning information. We consider Honest-But-Curious(HBC) adversary model in our privacy preserving algorithms.

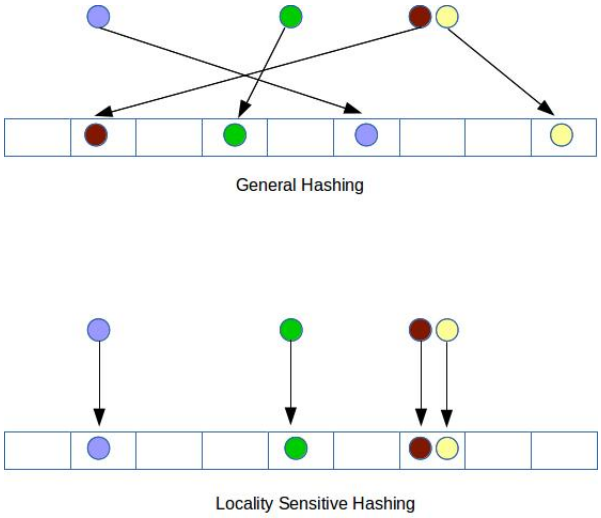


Figure 1.2 General Hashing versus Locality Sensitive Hashing

1.2 Locality Sensitive Hashing

The idea of Locality Sensitive Hashing was first introduced in [14]. Whereas in general hashing collisions are avoided, in the case of LSH such hash functions are desired in which points that are nearby are hashed to the same hash bucket with a very high probability. Mathematically, this idea is formalized as follows [14]:

Definition 2 *A family $H \equiv h : S \rightarrow U$ is called (r_1, r_2, p_1, p_2)-sensitive if for any two objects p, q in S:*

$$if\ d(p,q) \leq r_1 : Pr[h(p) = h(q)] \geq p_1 \tag{1.1}$$

$$if\ d(p,q) \geq r_2 : Pr[h(p) = h(q)] \leq p_2 \tag{1.2}$$

where $d(p,q)$ is the distance between objects p and q.

For the hashing scheme to be locality sensitive, two conditions to be satisfied are $r_2 > r_1$ and $p_2 < p_1$. In order to amplify the gap between the "high" probability p_1 and "low" probability p_2, standard practice is to concatenate several hash functions to obtain a function family $G = \{g : S \rightarrow U^k\}$ such that $g(p) = (h_1(p), h_2(p), ..., h_k(p))$; where k is the width of each hash

function and $h_i \in H$. For a hash function family G, the probabilities in Equation 1.1 and 1.2 are modified as:

$$if\ d(p,q) \leq r_1 : Pr[g(p) = g(q)] \geq p_1^k \tag{1.3}$$

$$if\ d(p,q) \geq r_2 : Pr[g(p) = g(q)] \leq p_2^k \tag{1.4}$$

During LSH, each object $o \in D$ is stored in the bins $g_j(o)$ for $j = 1, 2...L$; where each g is drawn independently and uniformly at random from G i.e., each object is hashed using L hash functions drawn from G and stored in the corresponding bins. The optimal values for the parameters k and L are computed as[14]: $k = log_{1/p_2} n$ and $L = n^\rho$ where $\rho = \frac{ln1/p_1}{ln1/p_2}$. For more detailed introduction on LSH refer [14] [10].

The LSH Scheme explained above can be used to solve the $NearestNeighbor$ problem which is defined below [10]:

Definition 3 *(Nearest Neighbor): Given a set D of objects in a d-dimensional Euclidean space R^d, preprocess D so as to efficiently answer queries by finding the object in D closest to a query object q*

The above definition could be extended to the case $K - NearestNeighbor$ search where we wish to return the K objects in the dataset that are closest to the query object. The approximate version of the $NearestNeighbor$ problem is defined as follow:

Definition 4 *(ϵ-Nearest Neighbor): Given a set D of objects in a d-dimensional Euclidean space R^d, preprocess D so as to efficiently return an object $o \in D$ for any given query object q, such that $d(p,q) \leq (1+\epsilon)d(q,D)$ where $d(q,D)$ is the distance of q to its closest object in D.*

The ϵ-Nearest Neighbor(ϵ-NN) problem can be reduced to the (R,c)-NN problem, which is a decision version of the nearest neighbor problem [7], where $c = (1+\epsilon)$. The (R,c)-NN problem is solved by using the LSH scheme on the dataset D by setting the parameter $r_1 = R$ and $r_2 = cR$. To process the query q, all the bins to which q is hashed to are searched and for each object o in those bins, if $d(o,q) \leq r_2$ YES is returned, else NO is returned.

1.3 Cryptographic Primitives

In this section, we briefly present an outline of the $SecureSum$ protocol we use as part of our privacy algorithm and also give an overview of the Adversary model we consider.

1.3.1 Secure Sum Protocol

$SecureSum$ is a cryptographic protocol which is extensively used as a building block in many privacy tasks in the areas of privacy preserving data mining, privacy preserving machine learning etc. A $SecureSum$ protocol facilitates three or more players p_1,...,p_t $t > 3$, each with a secret number v_i, to securely compute the sum of their numbers $V = \sum_{i=1}^{t} v_i$. In [5], the authors propose a secure sum protocol, along with other tools for privacy data mining like secure set union, secure scalar product and size of set intersection. Assuming that the value $V = \sum_{i=1}^{t} v_i$ to be computed lies in the range $[0, .., n]$, one player is designated as master site numbered 1. The remaining players are numbered $2, .., t$. Player 1 generates a random number R uniformly chosen from $[0, .., n]$. Player 1 adds this number to its secret value v_1 and sends the sum to player 2. For each player from 2 to n, the local secret number is added to the number received from player i-1 and passed on to the next player. Player t sends the final sum back to 1. Player 1 subtracts the random number R generated to obtain the final sum.

This algorithm considers only honest-but-curious adversaries and does not handle collusions. More sophisticated secure sum protocols, which handle collusions, have also been studied in [17][30]. Note that our PPOD algorithm uses $SecureSum$ as a black box and does not depend on its the underlying implementation.

1.3.2 Adversary Model

We give a brief description of the two adversary models namely Honest-But-Curious adversaries and Malicious adversaries.

Honest-But-Curious Adversary: Honest-But-Curious adversaries follow the protocol correctly but try to learn as much other information as possible. Security against such adversaries

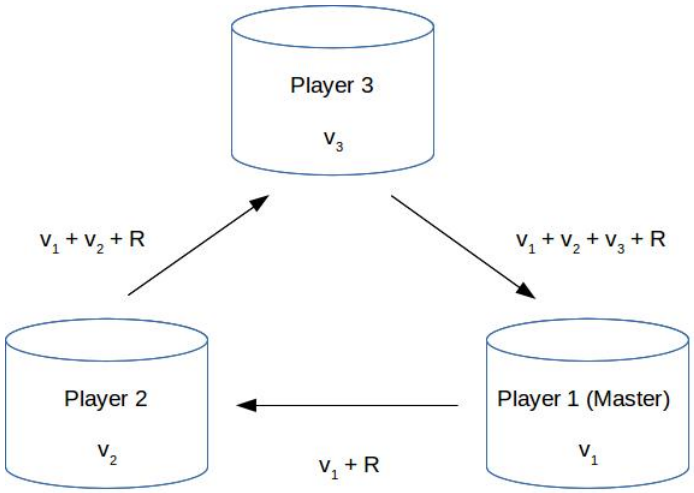


Figure 1.3 A 3-Player SecureSum protocol

is straight forward: no player or coalition of $c < n$ honest-but-curious players gain any information about the other player's private input sets other than what can be deduced from the result of the protocol. The ideal implementation for such protocol is to consider a Trusted Third Party which receives information from all players and outputs the result of the defined function. The aim for real implementation of the protocol is to simulate such a TTP where each player does not learn more information than in the ideal implementation with a very high probability. Honest-But-Curious adversary may also be called "Semi-honest" or "passive" adversary.

Malicious Adversary: As opposed to Honest-But-Curious adversaries who follow the protocol correctly, Malicious adversaries may behave arbitrarily in order to learn private information of the honest players. Specifically, such adversaries may (1) Send arbitrary messages or drop messages they are supposed to send (2) refuse to participate in the protocol (3) prematurely abort the protocol or (4) tamper with the communication channels. Security definition under such a scenario is limited to the case where at least one of the participating players is honest.

Formal definitions for adversary models can be found in [11]. As has been mentioned, we consider Honest-But-Curious adversaries in this work.

1.4 Thesis Outline

In **Chapter 2**, we first give a brief overview of the previous work in the area of outlier detection. Next we present our approximation algorithm for outlier detection using Locality Sensitive Hashing, along with theoretical bounds on level of approximation.

In **Chapter 3**, we extend the outlier detection algorithm for centralized setting to a distributed setting with vertical partitioning of data, considering privacy. The PPOD algorithm uses p-stable distribution based LSH. A brief overview of the previous work is also presented in this chapter.

In **Chapter 4**, we give empirical evidence of the performance of the proposed algorithms by running them on various datasets.

In **Chapter 5**, we give an extension of our centralized outlier detection algorithm to a PPOD algorithm considering horizontal distribution of data. The work presented in this chapter is not a contribution of this thesis and is presented for the sake of completeness.

In **Chapter 6**, we give conclusions and a few possible extensions for the work presented in this thesis.

Chapter 2

Outlier Detection in Centralized Setting

Outlier detection has been extensively researched in the recent past due to its application in various fields such as fault detection, medical diagnosis, measuring eco system disturbances etc. Outliers are observations in a data that appear to be inconsistent with the remainder of that set of data and are present in virtually every real world data set. These outliers could be the result of various reasons such as human error or instrument error or malicious activity such as network intrusion or natural deviations in populations and so on. Outlier detection is important for mainly two reasons: (1) In many data analysis tasks, outliers need to be detected and removed before processing the data, in order to enhance system performance. (2) In some applications, outlier detection in itself is crucial since the outliers provide useful or even critical information. These applications include credit card fraud detection, network intrusion detection or identifying new molecular structures in pharmaceutical research. There exist a few definitions for outliers which are considered general enough to cope with various types of data. Hawkins [13] defines an outlier as *"an observation that deviates so much from the other observations as to arouse suspicion that it was generated by a different mechanism"*.

In the following section we give a brief overview of the previous work and present our algorithm for Outlier detection in the subsequent section.

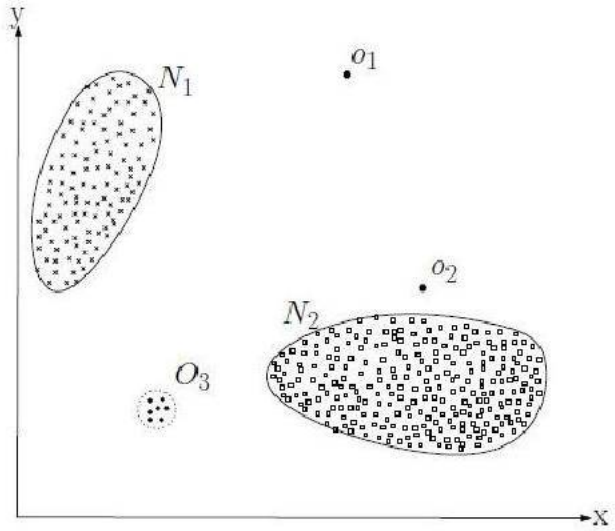


Figure 2.1 Outliers in a dataset

2.1 Previous work

Outlier detection methodologies could be broadly classified into four major categories, namely:

1. Statistics Based
2. Distance Based
3. Density Based
4. Clustering Based

We give a brief overview of each of these with more emphasis on distance based methods as the outlier detection algorithm presented in this work is a distance based approach. For a more detailed survey on outlier detection methodologies refer [4, 38].

Statistics Based: The foremost algorithms for outlier detection have been statistics based [26, 34]. These methods are further divided into two categories namely the parametric methods

and the non-parametric methods. The major difference between these two methods is that the parametric methods assume that the underlying distribution of the data is known while the non parametric methods do not assume any such knowledge. The statistical outlier detection methods are composed of two stages i.e., training stage and testing stage. A statistical model based on the given data is built in the training stage and in the testing stage a given data object is determined to be an outlier or not with respect to the model built.

The key limitation of statistical based outlier detection methods are that they do not scale well to high dimensional data. Also, particularly for the case of parametric methods, their usage to real world datasets is very limited since they assume that the data distribution is known.

Distance Based: The distance based methods for outlier detection have the advantage that they do not assume any underlying distribution of the data and also scale well to high dimensional data in comparison to the statistical based methods. Most of the metrics used for distance based outlier detection are based on the concepts of *local neighborhood* or *k nearest neighbors (kNN)*.

The first notion of distance based outliers, called $DB(p_t, d_t)outlier$ was introduced by Knorr and Ng in [18]. According to this definition, an object o in a dataset D is a $DB(p_t, d_t)$ outlier if at least fraction p_t of the objects in D lie at a distance greater than d_t from o. This definition for finding outliers is suitable where the observed distribution of the dataset does not fit any standard distribution. To calculate the percentage of data objects falling into the local neighborhood of each point, three classes of algorithms are presented [18, 19]. The first one is a nested loop algorithm and is inefficient while dealing with large datasets due to its quadratic complexity in the database size. The second is a index based algorithm and has a logarithmic complexity in the database size. However, the construction of index structures is often expensive and also the quality of the index structure is not easy to guarantee. The third algorithm is cell-based and is linear in the database size. However, it is exponential in the data dimensionality and is inefficient for datasets of high dimensions ($d > 4$).

Ramaswamy *et al.* [24] propose a slightly modified definition for distance based outliers using the distance to k^{th} nearest neighbors of each data object, denoted as D^k to rank objects

so that outliers can be more efficiently discovered and ranked. The definition for a $D^k outlier$ is given as: Given k and n, an object is a outlier if the distance to its k^{th} nearest neighbor is less than the corresponding value for no more than $n-1$ other objects in the database. Similar to the computation of $DB(p_t, d_t)outlier$, three classes of algorithms namely nested-loop algorithm, index-based algorithm and partition-based algorithm are proposed to compute D^k for each object efficiently.

Subsequent approaches[1, 2, 29] use efficient data structures and pruning techniques to reduce the complexity of outlier detection to near linear in the dataset size. An outlier detection schema based on the LSH technique was presented in [35]. Their method is based on the definition of an outlier provided in [24], whereas our approach uses the definition provided in [18]. The major drawback of distance based methods is that a majority of them are not effective to high dimensional data due to the curse of dimensionality.

Density Based: In density based outlier detection methods, instead of computing the distance to other database objects, the relative density of each object with respect to those of neighboring objects is considered. The number of objects with in a specified local region (grid region or distance based region) of a particular data object is used to define local density which can then be converted into outlier scores. In order to achieve this, the data is aggregated before outlier analysis by partitioning the data space. The data objects are then compared to the distributions in this pre-aggregated data for analysis. The first density based scheme for outlier detection was proposed in [3], which uses the concept of *Local Outlier Factor(LOF)*. Subsequently, other schemes for outlier detection using density are proposed in [23, 28]. The density based outlier detection methods are generally more efficient in finding the outliers than the distance based methods. However, in order to achieve the improved efficiency, the density based methods are more complicated and computationally expensive.

Clustering Based: Many data mining algorithms find outliers as a by product of clustering mechanism. In this method, outliers are defined as objects in a database that do not belong to any clusters. Thus, the clustering techniques implicitly define outliers as background noise of

the clusters. Examples for outlier detection schemes using clustering algorithms can be found in [9, 12, 39]. These methods are further classified into the following categories based on the clustering technique used: Partitioning clustering, Hierarchical clustering, Density-based clustering and Grid-based clustering methods. Clustering is a well researched area and there are quite a number of clustering algorithms in literature which can be leveraged for the task of outlier detection. However, one particular and obvious disadvantage of using clustering algorithms for outlier detection is that they are developed to optimize the finding of clusters in a dataset and not for finding outliers.

2.2 Outlier Detection Using Locality Sensitive Hashing

In this section, we present our approximation outlier detection algorithm for the case of centralized setting using the Locality Sensitive Hashing technique and give theoretical bounds on its performance. We use the distance based outlier definition $DB(p_t, d_t)$ where an object o in a dataset D is considered an outlier if at least fraction p_t of the objects in D lie at a distance greater than d_t from o. In our approach, we use the converse of this definition and say an object is a non-outlier if it has enough neighbors (p'_t) within distance d_t, where $p'_t = (1 - p_t) \times |D|$. Since the fraction p_t is very high (usually set to 0.9988), the modified point threshold p'_t will be very less compared to the number of objects in D. This allows us to easily detect most of the non-outliers by finding p'_t objects within distance d_t. To efficiently find the near neighbors, we use the Locality Sensitive Hashing technique.

The advantage of LSH technique is that if two data points in a high dimensional space are close, then after a projection to a lower dimension space these points remain close together with a very high probability. This advantage has been leveraged for the problem of finding nearest neighbors in large datasets with high dimensionality. Given a dataset D, all the objects are first hashed using LSH and then for any given query object its nearest neighbor can be searched with a query time sub-linear in the dataset size by searching in only the subset of dataset that hashed to the same bins as the query object [10]. In our approach, we use the LSH to first hash all the objects of a dataset but instead of finding the nearest neighbor for a object, we find all

objects that lie with in the distance threshold d_t from that object by considering the subset S of objects that has to the same bin during LSH. As opposed to the nearest neighbor search where each of the objects in this subset S has to be considered again, for the purpose of detecting outliers, we only need the cardinality of this subset. If the cardinality of this subset is more than the point threshold p'_t, then we can say that this object is not an outlier since it has enough neighbors. This is the central idea of our approach to which we add further enhancements in order to achieve a very good approximation for finding outliers.

2.2.1 Algorithm Description

The algorithm for outlier detection in centralized setting is executed in two phases. In the first phase, the LSH scheme is used to hash all the objects to generate the binning structure. In the second phase, this LSH binning is used to compute the set of approximate outliers.

The algorithm *CentralizedOD* takes as input the dataset D of size $n = |D|$, distance threshold d_t and point threshold p_t and outputs the set of approximate outliers M. In the First phase, initially the modified point threshold p'_t and the LSH distance parameter R are computed as: $p'_t = (1 - p_t) \times |D|$ and $R = r_1 = d_t/c$. The LSH scheme is then run on the dataset D with the parameter R. In the LSH scheme each object is hashed using L hash functions each of width k, as explained in Section 1.2. The output is a binning structure T with each bin having the objects hashed to that bin.

In the second phase *Pruning*, most of the objects which cannot be outliers are pruned. Initially all the objects in the dataset are marked as *not pruned*. For pruning, each object o is considered and is processed only if it has not been marked as *pruned*, in which case we find the number of neighbors o has within distance d_t. To find the neighbors, the L bins to which o is hashed during LSH are considered and the set of all objects stored in those L bins is formed (without removing duplicates). We denote this set by $Neighbors$. The objects in this set are the probable neighbors of o. More precisely, from Equation 1.4, we know that each object in the set $Neighbors$ is within the distance $r_2 = c \times r_1 = d_t$ from o, with a probability at least $(1 - p_2^k)$. To boost this probability, we consider only those objects which are repeated more than

Table 2.1 Notation

Term	Definition
D	Dataset
n	Size of dataset
L	Number of hash functions
H	Number of bins per hash function
T	Hash Table
d_t	Distance Threshold
p_t	Actual point threshold
p'_t	Modified point threshold
b_t	Bin Threshold
N_{pr}	Number of Processed Objects
M	Set of Outliers

b_t ($b_t \leq L$) times in the L bins and store only those objects in a new set $RepeatedNeighbors$. In other words, we are reducing the error probability of considering a non-neighbor as a neighbor of the object o. Here, b_t is a *bin threshold* which can be computed based on the desired false negative probability. We propose a method to compute the optimal value of b_t later in this section.

If the cardinality of the set $RepeatedNeighbors$ is greater than the modified point threshold p'_t, with a very high probability o cannot be an outlier since it has sufficient neighbors within distance d_t. Moreover, this holds true for all the objects in $RepeatedNeighbors$ because from Equation 1.4, any two objects in this set are within distance d_t, i.e., every object other than o also has more than p'_t neighbors with in the distance d_t, so it can not be an outlier (with a very high probability). Hence, all the objects in $RepeatedNeighbors$ are marked as *pruned* (non outlier). Thus we obtain a very efficient pruning mechanism where we can determine many objects to be non outliers to be non-outliers even without considering them separately.

If, on the other hand the cardinality of $RepeatedNeighbors$ is less than or equal to the point threshold p'_t, we consider the object o as a probable outlier and add it to the set of probable outliers M. This procedure is repeated till all the objects are either marked as *pruned* or as outliers. Finally, the set M of the probable outliers is returned. We denote the number of objects actually processed during *Pruning* as N_{pr} and later in sub-section 2.2.5, we give a bound for N_{pr}.

Algorithm 1 CentralizedOD

Require: Dataset D, Distance Threshold d_t, Point Threshold p_t

Ensure: Set of Outliers M

1 $p'_t = (1 - p_t) \times |D|$

2 $R = d_t/c$

3 $T = LSH(D, R)$

4 Compute b_t // Compute the optimal bin threshold

5 $M = Pruning(D, T, p'_t, b_t)$

6 Return M

The set M contains the actual outliers as well as a few false positives and extremely low false negatives. In the experiments section we show that by choosing the optimal bin threshold b_t, the false negatives can be completely eliminated at the cost of only very few false positives. In the following sub-section we give a theoretical bound on the number of false positives and false negatives and in sub-section 2.2.3 we give a method to further reduce the false positives.

If one wishes to completely remove the false positives without any scope for approximation one could instead use an additional step after the centralized algorithm where each object in the final set of approximate outliers M is again processed (the process to reduce false positives given in section 2.2.3 is not required in this case). For each object o, the distance to every object in the original database D is calculated. If the number of objects in D lying at a distance greater than the distance threshold d_t is greater than the point threshold p_t then object o is marked as an outlier. After processing each object in the M the final output would be the actual outliers in dataset D. This process is summarized in algorithm 3.

2.2.2 False Positives and False Negatives

In the context of outlier detection, a false positive (fp) is to label a non-outlier as an outlier and false negative (fn) is to label an outlier to be a non-outlier. In this section we give a bound on the probabilities of both based on the value of the bin threshold parameter b_t. Consider a LSH scheme where each object is hashed using L hash functions each of width k. The probability of two objects at a distance greater than $r_2 = r_1 \times c = d_t$, to be hashed to the

Algorithm 2 Pruning

Require: Dataset D, Hash Table T, Point Threshold p'_t, Bin Threshold b_t,

Ensure: M

$M = \{\}$
$\forall o \in D, pruned[o] = false$
for each object o in D **do**
  if $pruned[o] = false$ **then**
    $Neighbors$ = $\bigcup_{i=1}^{L} T[g_i(o)]$ // g is the hash function
    $RepeatedNeighbors$ = $\{o' \mid o' \in Neighbors$ and $occurrence(o') \geq b_t\}$
    if $|RepeatedNeighbors| > p'_t$ **then**
      $\forall o' \in RepeatedNeighbors, pruned[o'] = true$
    else
      $M = M \cup \{o\}$
    end if
  end if
end for

Algorithm 3 BruteForceOD

Require: Dataset D, Approximate outlier set M, Distance Threshold d_t, Point Threshold p_t

Ensure: Outliers M'

$M' = \{\}$
for each object o in M **do**
  $count = 0$
  for each object o' in D **do**
    if $Dist(o, o') > d_t$ **then**
      $count = count + 1$
    end if
  end for
  if $count \geq p_t$ **then**
    $M' = M' \cup \{o\}$
  end if
end for

same bin is at most p_2^k. Consider the worst case scenario where an actual outlier has exactly p_t' neighbors. Hence, counting one non-neighbor as a neighbor will lead to a false negative. In our scheme, for a non-neighbor to be counted as a neighbor of an object o, it should hash to the same bin as o for at least b_t times out of L times. The probability that this happens for exactly b_t times out of L is given by the binomial probability:

$$Pr'[fn] \leq \binom{L}{b_t} P_2^{kb_t} (1 - P_2^{kb_t})^{L-b_t} \tag{2.1}$$

and hence, the probability for a false negative is upper bounded by:

$$Pr[fn] < (L - b_t) \times \binom{L}{b_t} P_2^{kb_t} (1 - P_2^{kb_t})^{L-b_t} \tag{2.2}$$

The above formula can be still simplified by using the equality $P_2^k = 1/n$ [14], where $n = |D|$, as:

$$Pr[fn] < (L - b_t) \times \binom{L}{b_t} \frac{1}{n^{b_t}} (1 - \frac{1}{n})^{L-b_t} \tag{2.3}$$

Similarly, the probability that two objects within a distance R being hashed to two different bins is at most $(1 - p_1^k)$. The probability that this will happen at least $(L - b_t)$ times out of L times is given as:

$$Pr[fp] < b_t \times \binom{L}{L - b_t} (1 - p_1^{k(L-b_t)}) p_1^{kb_t} \tag{2.4}$$

This gives an upper bound for the probability of a false positive.

2.2.3 Reducing the False Positives

The technique to reduce the false positives is based on the probabilistic nature of the LSH scheme. As explained in Section 1.2, the LSH scheme based on p-stable distribution projects all objects onto random lines. Due to this randomization, each execution of the LSH scheme projects the objects onto different lines which in turn ensures that the output of the centralized algorithm is probabilistic. However, choosing an optimal bin threshold ensures that the actual outliers in the dataset are returned in the output of the centralized algorithm even over multiple runs. Hence, to further reduce the false positives, we run the centralized algorithm over a

fixed number of iterations $iter$, and take the intersection of the resulting sets and return this intersection set, say S as the final output. i.e., $S = \bigcap_{i=1}^{iter} M_i$. Since for a false positive to occur in the set S it should occur in each of the sets M_i, we have the modified false positive probability and false negative probability as:

$$Pr'[fp] = (Pr[fp])^{iter} \tag{2.5}$$

$$Pr'[fn] = 1 - (1 - Pr[fn])^{iter} \tag{2.6}$$

As can be seen from the above formulas, the modified probability of a false positive decreases exponentially with the number of iterations and thus we need to run only a few iterations to achieve very few false positives. We discuss more about the effect of $iter$ on the false positives and false negatives in the experiments section. Furthermore, the false positives which remain in the final output can be termed as weak non-outliers, in the sense that most of these objects have marginally greater number of neighbors than the required threshold of objects to make an object a non-outlier. We support this claim by giving empirical results in the experiments section.

2.2.4 Bin Threshold

As seen from the equations 2.3 and 2.4, both the false negative and false positive probabilities depend on the bin threshold. Increasing the bin threshold has an effect of decreasing the false negatives at the cost of an increase in the false positives and vise versa. An optimal value for b_t would be one which would remove the false negatives at the cost of introducing minimal false positives. In our scheme, the user has the flexibility to fix b_t, based on the false negative probability desired, using equations 2.3. Given a false negative probability, equation 2.3 will be an unknown in one quantity which is b_t. Further, since equation 2.3 is a monotonically decreasing function with increase in b_t, computing the optimal value of b_t can be done easily by using binary search methods.

2.2.5 Bound on Number of Processed Objects N_{pr}

We give an upper bound on the number of objects which are processed during *Pruning*. Let the actual number of outliers in the dataset be m. Consider the worst case scenario where all the actual non-outliers in the dataset have exactly p'_t number of objects as neighbors. The number of objects processed, N_{pr}, would be maximum in the above mentioned case for which the bound is given as:

$$N_{pr} = (\frac{n - m - fp}{p'_t} + m + fp)$$

where $n = |D|$ and fp is the number of false positives. Usually, in a $DB(p_t, d_t)$ outlier detection scheme the fraction p_t is set to 0.9988, in which case p'_t will be: $p'_t = (1 - p_t)n = .0012n$. Since $n >> m$ and $n >> fp$ we can approximate $(n - m - fp)$ to n and hence, the bound for number of objects processed is: $N_{pr} < ((n/.0012n) + m + fp) < m + fp + 834$. This is the worst case bound and usually the actual number of objects processed will be much less than the above mentioned inequality. Later in the Experiments chapter, we show that for large datasets, N_{pr} is in fact less the 1% of the size of the total dataset.

2.3 Analysis

The Computational complexity of our centralized algorithm is given below:

Computational Complexity: In Algorithm 1, steps 1 and 2 are of constant complexity. The complexity for computing the LSH (step 3) is $O(ndkL)$ where n and d are the dataset size and dimensionality and k and L are the width of each has function and the number of hash functions used in LSH respectively. The complexity for the *Pruning* sub-protocol is $O(nLN_{pr})$ where N_{pr} is the number of objects processes during *Pruning*. Considering only dominating terms and since $k << n$, the total complexity of the algorithm over a constant number of iterations is $O(ndL)$.

2.4 Summary

In this Chapter, we have started out by giving an overview of the previous work in the area of outlier detection listing the various methodologies proposed in literature for the same. We then presented our work for approximate outlier detection using the Locality Sensitive Hashing technique.

We have incorporated the concept of *Bin Threshold* in the LSH framework which improved the performance of our algorithm in detecting outliers. Finally, we gave theoretical bounds on the performance as well as an analysis for computational complexity of the proposed algorithm.

Chapter 3

Privacy Preserving Outlier Detection

In this Chapter, we extend the Outlier detection algorithm for centralized setting presented in the previous chapter to a to the case of distributed setting considering privacy. In the following section, we give a brief overview of related work. Next we an overview of p-stable distribution based LSH on which our PPOD algorithm is based and then proceed to the description of the proposed PPOD algorithm.

3.1 Previous Work

Although Privacy Preserving data mining has generated a lot of interest in the research community as it has gained a lot of importance due to the increasing need for privacy, it is still relatively a very new research area and thus there is comparatively less work in literature which propose new PPDM algorithms. Privacy Preserving Outlier Detection has received even less attention and not more than handful of works consider this problem, which we mention here along with a few other works in the area of PPDM for other data mining tasks.

Privacy Preserving Data Mining was first introduced by Lindell and Pinkas in [20].In this paper, an algorithm for privacy preserving $ID3$ classification was described. privacy preserving classification has been further studied in [21][37]. PPDM algorithms for association rule mining were proposed in [16][8][25][32] while clustering was considered in [33][15]. Privacy preserving outlier detection (PPOD) was introduced by Vaidya *et al* in [31]. They use the definition for distance based outliers provided in [18], and give PPOD algorithms for both hor-

izontal and vertical partitioning of data. Subsequently, a PPOD algorithm using the k-nearest neighbor based definition [24] was given in [40] , considering only vertical partitioning. Privacy preserving density based outlier detection algorithms have been proposed in [6, 27] and an algorithm for spatial outlier detection is discussed in [36].

3.2 p-stable based LSH

Our privacy preserving algorithm uses the LSH scheme based on p-stable distribution proposed by Datar et al in [7] (Note that the centralized algorithm presented in this work is independent of underlying hash function used in LSH).

A distribution D over $\Re$ is called p-stable if there exists $p > 0$ such that for any real numbers v_1 to v_n and independent identically distributed variables X_1 to X_n with distribution D, the random variable $\sum_i v_i X_i$ has the same distribution as the variable $(\sum_i |v_i|^p)^{(1/p)} X$, where X is a random variable with distribution D. Stable distributions exist for any $p \in (0, 2]$. Special cases are:

- For $p = 2$; the Gaussian distribution $g(x) = \frac{1}{\sqrt{2\pi}} exp(-x^2/2)$ is 2-stable
- For $p = 1$; the Cauchy distribution $c(x) = \frac{1}{\pi} \frac{1}{1+x^2}$ is 1-stable

In order to build a locality sensitive hash function, the p-stable distribution is used to generate a random vector a of dimensionality d, where each entry of a is chosen independently from a p-stable distribution. For a given object v of dimensionality d, the dot product $a.v$ projects the vector v on to a real line and for any pair of vectors a, b these projections are close if $l_p(a - b)$ is small and far otherwise. Now, dividing the real line into segments of width w gives a hash function where each segment of the selected width forms a hash bucket. Formally, the hash function is given by:

$$h_{a,b}(v) = \lfloor \frac{a.v + b}{w} \rfloor$$

where b is a random real number uniformly selected from $[0, w]$ (random shift).

Now for two random vectors v_1 and v_2 and a random vector a whose entries are drawn from a p-stable distribution, $a.v_1 - a.v_2$ is distributed as cX; with X being a random variable drawn from p-stable distribution and $c = ||v_1 - v_2||_p$. The probability that the two vectors v_1, v_2 collide under this hash family is gives as:

$$p(c) = Pr_{a,b}[h_{a,b}(v_1) = h_{a,b}(v_2)] = \int_0^w \frac{1}{c} f_p(\frac{t}{c})(1 - \frac{t}{w})dt$$

Here $f_p(t)$ denotes the probability density function of the absolute value of the p-stable distribution. As per definition for LSH, the family of hash functions given above is a $(r_1, r_1 * c, p(1), p(c))$ - sensitive.

3.3 Algorithm Description

In the case of vertically distributed data, each player collects information about different attributes for the same set of objects such that the union of all the attribute subsets equals the global attributes and all the attribute subsets are disjoint. The algorithm for PPOD over vertically partitioned data is executed in two phases. In the First phase, all players engage in communication to get the LSH binning of all the objects in the dataset considering all the attributes (distributed across players). In the Second phase, using this global binning information, each player locally computes the probable outliers.

The algorithm takes as input dataset D of dimensionality d, vertically distributed across t players such that $dim(D^p) = d^p$, $|D^p| = |D| = n$, for $p = 1$ to t and $\sum_{p=1}^{t} dim(D^p) = d$. The algorithm also takes as input the parameters d_t and p_t and outputs the set of outliers M.

The proposed PPOD algorithm is based on LSH scheme using p-stable distributions[7], where each each hash function for a d-dimensional object v is computed as:

$$h_{a,b}(v) = \lfloor \frac{a.v + b}{w} \rfloor$$

Here, a is a d-dimensional vector, with each entry chosen independently from a p-stable distribution and b is a real number chosen uniformly from the range $[0, w]$. In the vertical distribution, where each player has some d^p dimensions of the total d dimensions, each player can

locally generate the respective d^p entries of the vector a and compute its local share ($a^p.v^p$) of the dot product ($a.v$). The players then compute the secure sum of their shares to get ($a.v$). Since the values of b and w can be public (can be generated by one player and then be published), all the players can build the LSH binning structure using the dot products previously computed. Using this binning structure, each player can locally invoke the *Pruning* protocol to compute the set of probable outliers. To reduce the overall computation, all this computation can be done at any one player who then publishes the final outlier set. The procedure is outlined in Algorithm 4. Now we will explain the important steps of the given algorithm.

Initially, each player p locally generates $A^p_{k\times L}$, where each element a^p is drawn from a p-stable distribution as explained before. In Steps 6-8 each player will compute their respective shares of the dot products for all the objects. In step 11, the *SecureSum* protocol is used to compute the dot product from all the shares. Steps 14-20 are carried out at any single player, say p^1. In step 14 the values of $B_{k\times L}$ and w are generated, where each element b of $B_{k\times L}$ is a random number in $[0, w]$. In steps 17-21, the previously computed dot products are used to evaluate hash functions for all the objects. The LSH binning is generated using those hash functions and then the *Pruning* protocol is invoked to compute the set of probable outliers M. Finally, this set M is then published.

We can reduce the false positives by applying the same technique of centralized setting discussed in Section 2.2.3. That is, we run the entire Algorithm 4, $iter$ times and take the intersection of the resulting sets of probable outliers M_i for i = 1 to $iter$. Since all these runs are independent, we can execute them in parallel to lower the number of rounds.

3.4 Analysis

The Computational and Communication complexities for Algorithm Algorithm 4 along with the Security analysis is given below:

Algorithm 4 Vertical Distribution

Require: t Players, Dataset D vertically distributed among the players, Distance Threshold d_t, Point Threshold p_t

Ensure: Outliers M

for each player p = 1 to t **do**
    $p'_t = (1 - p_t) \times |D^p|$
    $R = d_t/c$
    Compute LSH parameters k and L
    Generate $A^p_{k\times L}$ where $Dim(A^p) = d^p$
    for each object o in D^p **do**
        $S^{p,o}_{k\times L} = A^p_{k\times L}$. o
    end for
end for
for each object o in D **do**
    $S^o_{k\times L} = SecureSum(S^{1,o}, S^{2,o}, .., S^{p,o})$
end for
At P^1
Generate $B_{k\times L}$ and w
for each object o in D^1 **do**
    $H^o_{k\times L} = \lfloor \frac{S^o+B}{w} \rfloor$
end for
Compute T using $H_{k\times L}$
$M = Pruning(D^p, T, p'_t, b_t)$
Publish M

Computational Complexity

We give the computational complexity from the perspective of player P^1. In Algorithm 4, steps 2-4 have constant complexity. Step 5 has a complexity of $O(kL)$. Step 7 has a complexity of $O(nd^1kL)$. Step 14 has a complexity of $O(kL)$. Steps 16 and 18 have a complexity of $O(nkL)$ and finally step 19 has a complexity of $O(nLN_{pr})$. Thus the overall complexity for player P^1 would be $O(nd^1L)$; where d^1 is the number of dimensions P^1 has. Consequently the overall complexity for Algorithm 4 would be $O(ndL)$

Communication Complexity

In Algorithm 4, communication among the players is necessary only in steps 11 and 20. Among these, step 11 is the dominating factor in terms of the communication complexity, where for each object in the dataset, we need to perform $k \times L$ *SecureSum* operations. Assuming the cost of each *SecureSum* operation to be δ, the overall communication complexity of the algorithm would be $O(nL\delta)$.

Security Analysis

In Algorithm 4, the set of outliers is published by one player in step 20 and this would not reveal any private information since this is the desired output to be made public. In step 11, the $SecureSum$ protocol is used to evaluate the dot product of all the shares with the players and hence no information about the local shares would be revealed. One thing to be noted here is that the input from each player to the $SecureSum$ protocol is not the actual data itself but the projections of the data objects onto random lines and the $SecureSum$ protocol is used to protect even this information.

Since each player knows the LSH binning considering all dimensions during the algorithm, some information about the global distribution of the dataset could be inferred. However, no information about the data projections or actual data with any particular player would be revealed(the limitation being that the number of players need to be greater than 2 due to the limitation of $SecureSum$ protocol). To elaborate more on this, consider a simplistic example

Table 3.1 Algorithm Complexity

Setting	Round	Communication	Computation
Centralized	-	-	$O(ndL)$
Vertical	2	$O(nL\delta)$	$O(nd^pL)$

with three players A, B and C each having the information about two dimensions (attributes) of a dataset having n objects. After the $SecureSum$ protocol in step 11 of algorithm 4, each player gets to know the global LSH binning structure of all objects considering all the dimensions. We give the information gained by one player, say player A, about the local data with other players. In the global binning structure, if a particular hash bucket has n' out of the total n objects hashed to it, then all these objects considering all dimensions lie with in the distance threshold d_t with a very high probability due to LSH properties. From this player A can infer that these n' objects lie with the same distance considering only the attributes of either player B or C. Players B and C also get to know the same information about other players data from the global binning structure. But for finding outliers, the distance threshold d_t is fixed very large and is comparable to the spread of the overall dataset and thus the information gained about the local data with other players is very limited.

3.5 Summary

In this chapter, we have first given a brief overview of previous work in the area of privacy preserving data mining in general and privacy preserving outlier detection in particular. A brief description of the p-stable distribution based LSH was given. Next, we presented our algorithm for privacy preserving outlier detection which used LSH bases on p-stable distributions. Finally, we gave the computational, communication and security analysis of the proposed PPOD algorithm.

Chapter 4

Experiments

Experiments are performed on datasets listed in Table 4.1. *Corel* contains image features extracted from a Corel image collection. *Landsat* database consists of the multi-spectral values of pixels in 3x3 neighborhoods in a satellite image. *Darpa* contains evaluation of various computer network intrusion detection systems. *Server* is an extract of KDD Cup 1999 Data containing the statistics of 500K network connections as explained in [29]. *Household* is a US census dataset. For implementation of LSH, we have used E2LSH package[1] as the base, which is based on the p-stable distribution [7]. For all our experiments the approximation factor ϵ is set to 2. All experiments are executed on Intel(R) Core $i7$ CPU $3.33GHz$ machine.

4.1 Centralized Setting

In this section, we provide empirical evidence on the performance of the proposed outlier detection algorithm by running it on various datasets listed in Table 4.1. The experimental re-

[1] http://www.mit.edu/∼andoni/LSH/

Table 4.1 Dataset Description

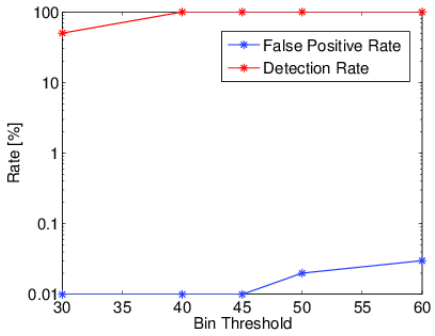
Dataset	Objects	Attributes	Source
Corel	68040	32	kdd repository
Landsat	275465	60	vision lab, ucsb
Server	494021	5	kdd repository
Darpa	458301	23	Lincoln Laboratory, MIT
Household	1000000	3	US Census Bureau

sults support various claims regarding the performance of the algorithm made in the preceding chapters.

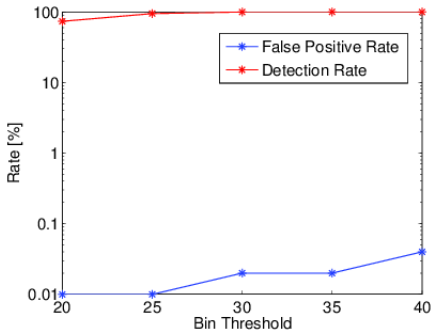
Execution Time: The execution time (in seconds) averaged over multiple runs of our centralized algorithm is tabulated in Column 2 of Table 4.2. Since all the iterations of our algorithm are independent they can be run in parallel to achieve the speed up.

Bin Threshold: We computed the optimal bin threshold b_t as described in Section 2.2.4 for mentioned datasets and ran our algorithm using the same bin threshold. The results are summarized in Table 4.2. Column 3 specifies the optimal value of b_t for each dataset and column 4 specifies the number of false positives (in $\%$) at the optimal b_t. To compute the rate of false positives and false negatives, we have first found the outliers in each data set using brute force method for the same distance and point thresholds and then compared the results with the outliers found using our algorithm. As can be seen from the result, the false positive rate is quite less proving that the approximation of our algorithm is very good. The detection rate for each dataset at the optimal b_t is 100% (i.e., no false negatives). Furthermore, for each of these false positives, we calculated the actual number of neighbors with in distance threshold d_t and listed the average number of neighbors (denoted as NN) for each false positive in column 7. For each dataset, this number NN is only marginally greater than the actual point threshold p_t (mentioned in column 6) when compared to the total dataset size. Hence, we can say that most of these false positives are having very few neighbors (close to p_t) and can be considered to be outliers or weak non-outliers.

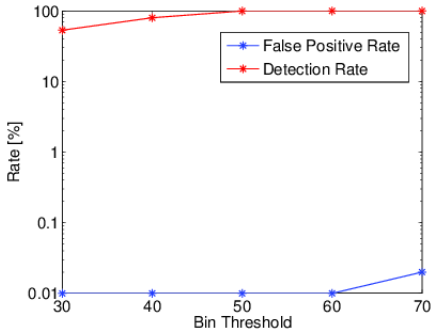
The effect of varying bin threshold on false positives and false negatives (detection rate) is shown in Figure 4.1. Both false positives and detection rate are plotted in logarithmic scale. It is evident from the figure, as we increase the value of b_t, the number of false negatives decrease (i.e., detection rate increases) at the cost of increase in the number of false positives. As seen from the in Figure 4.1 and Table 4.2, at the optimal value of b_t false negatives are eliminated at the cost of a minimal false positives for all datasets.



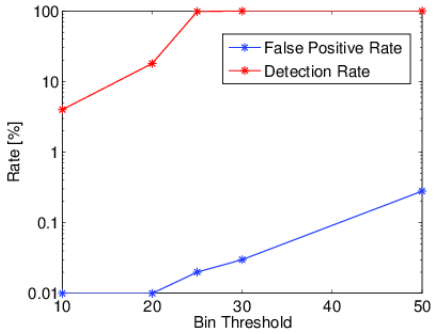
(a) Corel



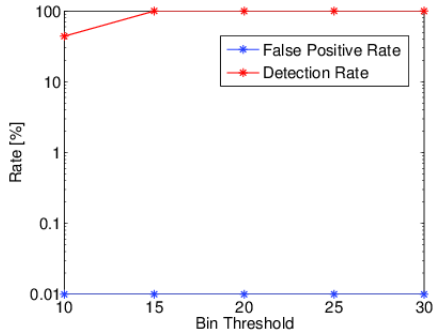
(b) Landsat



(c) Server



(d) Darpa



(e) Household

Figure 4.1 Bin Threshold Vs False Positive and Detection rate

Table 4.2 Performance of Centralized algorithm

Dataset	Time (s)	b_t	FP (%)	N_{pr} (%)	p_t	NN
Corel	30.77	45	0.01	0.62	82	147
Landsat	44.11	30	0.02	0.36	331	929
Server	37.67	50	0.01	0.07	593	1000
Darpa	50.00	30	0.03	0.68	550	1103
Household	61.63	25	0.01	0.14	1200	1326

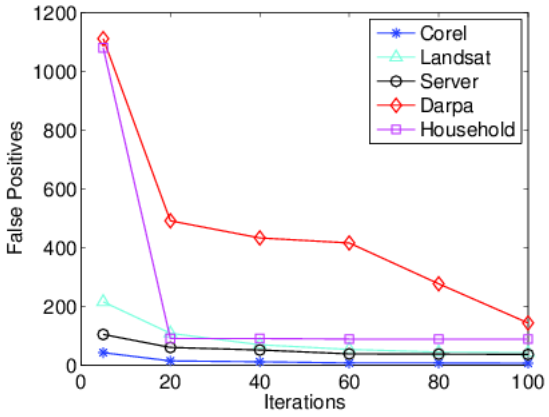


Figure 4.2 Effect of Iterations

Number of processed objects: For each dataset, the number of objects processed (N_{pr}) during *Pruning* protocol in Algorithm 1 are listed in column 5 (in %). The values in this columns shows the efficiency of our pruning technique and the number of objects processed is indeed less than 1% of the dataset size as claimed in Section 2.2.5.

Iterations: The effect of varying the number of iterations $iter$ on the false positives at the optimal bin threshold is shown in Figure 4.2 As can be seen from the figure, the number of false positives decreases exponentially with increase in $iter$. Thus the number of iterations needed to be run to achieve a very good approximation would be very less.

4.2 Distributed Setting

We discuss about the performance of the proposed privacy preserving algorithms for vertical distribution of data.

The performance of Algorithm 4 in terms of false positive and false negative rates remain the same as that of the centralized algorithm. We ran Algorithm 4 considering two players with uniform distribution of dimensions and compared the communication cost with the privacy preserving algorithm for vertical distribution given in [31]. Up to datasets having size of order 10^6, the difference in the cost is of the order 10^3, but this difference would be more considerable for larger datasets.

Chapter 5

Extension to Horizontal Partitioning

The outlier detection algorithm for centralized setting proposed in this work can also be extened to a Horizontal partitioning of data considering privacy, which is presented in this chapter. This work is in collobaration with N.Raval and is a part of his masters dissertation[22];it is presented here for the sake of completeness and is not a contribution of this thesis.

5.1 Algorithm Description

In the case of Horizontal partitioning, each player has the same attributes for a subset of the total objects. The algorithm for privacy preserving outlier detection over horizontally distributed data is executed in two phases. In the First phase, each player locally computes its own set of local probable outliers, by running *CentralizedOD* on its own dataset. These local outliers contain the global outliers as well as a few non-outliers for which enough neighbors do not exist in the respective local datasets. To prune these non-outliers, the players engage in communication in the Second phase and compute their subsets of the global probable outliers. We define the local and global outliers as follows.

Definition 5 local outlier*: given a distance threshold d_t and a point threshold p_t, an object o with player P_i is a local outlier if the number of objects in the local dataset D_i lying at a distance greater than D is at least a fraction p_t of the total dataset N.*

Definition 6 global outlier*: given a distance threshold d_t and a point threshold p_t, an object o in a dataset D is a global outlier if at least fraction p_t of the objects in D lie at a distance greater than D from o.*

We consider t players, each with dataset D^p for $p = 1$ to t and $n^p = |D^p|$ such that the size of the entire dataset D is $n = \sum_{p=1}^{t} n^p$. We assume that n is known beforehand to all the players, otherwise it could be computed using *Secure Sum* protocol. Apart from this, the algorithm takes as input distance and point thresholds. At the end of the algorithm each player has its subset of the outliers in the dataset D.

As in the case of vertical distribution, the PPOD algorithm for Horizontal distribution of data is also based on LSH scheme using p-stable distributions[7], where each hash function for a d-dimensional object v is computed as:

$$h_{a,b}(v) = \lfloor \frac{a.v + b}{w} \rfloor$$

While performing the LSH on the local datasets, in each iteration of the LSH, all the players need to use the same ranodmness. Hence, any one player say P^1 computes the LSH parameters k and L and generates the random vectors $A_{k\times L}$ and $B_{k\times L}$. Here, each element a of $A_{k\times L}$ is a d-dimensional vector whose each entry is independently drawn from a p-stable distribution and each element b of $B_{k\times L}$ is a random number in $[0, w]$. P^1 then publishes these values to all the other players. Each player then computes the modified point threshold $p'_t = (1 - p_t) \times n$ and the LSH distance parameter R. Note that the parameter p'_t is computed based on the size of the entire dataset instead of the size of the local dataset. The LSH scheme is then run on the local dataset using the above computed parameters which outputs the binning structure T^p. Here, the LSH scheme is a bit different from that of the LSH scheme run in the centralized setting in that the random vectors A and B are given as input to the LSH scheme whereas in the actual LSH scheme these values are generated with in the LSH protocol. The protocol *Pruning* is then invoked, which returns the set of local probable outliers M'^p.

In the Second phase, to form the set of global outliers, each party requires the total number of objects with all the other players that are hashed to each bin. Each player has a set of bins generated during LSH where the bin labels with each player might be different (considering

Algorithm 5 Horizontal Distribution

Require: t Players, Datasets D^p for $p = 1$ to t, Total Dataset size n, Distance Threshold d_t, Point Threshold p_t

Ensure: Outliers M^p; $p = 1$ to t

At P^1:
  Compute LSH parameters k, L and w
  Generate $A_{k\times L}$ and $B_{k\times L}$
  Publish k, L, w, $A_{k\times L}$ and $B_{k\times L}$
for each player $p = 1$ to t **do**
  $p'_t = (1 - p_t) \times n$
  $R = d_t/c$
  $T^p = LSH(D^p, R, A_{k\times L}, B_{k\times L})$
  Compute b_t
  $M'^p = Pruning(D^p, T^p, p_t, b_t)$
end for
for each player $p = 1$ to t **do**
  $BinLabels^p = \{$label of each bin in $T^p\}$
end for
$BinLabels = SecureUnion(BinLabels^p)$; $p = 1$ to t
for each player $p = 1$ to t **do**
  for $i = 1$ to $|BinLabels|$ **do**
    if $BinLabels(i) \in BinLabels^p$ **then**
      $C^p(i) = |T^p(i)|$
    else
      $C^p(i) = 0$
    end if
  end for
end for
$C = SecureSum(C^1, C^2..., C^t)$
for each player $p = 1$ to t **do**
  $\widehat{C}^p = C - C^p$
  for each object o in M'^p **do**
    $Neighbors^p = \bigcup_{i=1}^{L} T^p[g_i(o)]$

30 $RepeatedNeighbors^p = \{q \in Neighbors^p \mid occurrence(q) \geq b_t\}$

31 $req\_nn^p = p'_t - |RepeatedNeighbors^p|$

32 $ValidBins^p = \{b_i \mid \widehat{C}^p[g_i(o)] \geq req\_nn^p, i = 1, 2, ...L\ \}$

33 **if** $|ValidBins^p| \leq b_t$ **then**

34 $M^p = M^p \cup \{o\}$

35 **end if**

36 **end for**

37 Return M^p

38 **end for**

only the non-empty bins). Hence the players communicate to form the union of all the bin labels using the *SecureUnion* protocol [5]. While forming the set of binlabels, each binlabel needs to be indexed with the number of iteration during LSH (1 to L) in order to differentiate between bins of same labels created during different iterations of LSH. Each player then counts the number of objects it has in each of these bins. The *SecureSum* protocol [5] is used to compute the sum of the corresponding counts with all the players for each bin label. Each player then locally computes the sum of the other $t-1$ player's counts for all the bins. At every player, each object o in the set of local probable outliers M' is then considered to determine whether it is a global outlier. To get the number of neighbors o has in the local dataset, the cardinality of the set $RepeatedNeighbors$ for o is computed. This step is actually redundant if the value is stored in the First phase (during *Pruning*). This value would be less than p'_t since the object was considered a local probable outlier in the First phase. The number of neighbors required to make it a non-outlier is computed as: $req_{nn} = p'_t - |RepeatedNeighbors|$. To get the count of neighbors of o which are available with other players, L entries in $\widehat{C}^p$ indicated by $g_i(o)$ where $i = 1$ to L, are considered. Out of these L bins, only those bins which have object count greater than req_{nn} are said to be $ValidBins^p$. If the cardinality of $ValidBins^p$ is less than or equal to b_t, then the object o is considered as a global outlier and added to the set of global probable outliers M^p.

As in the centralized setting, the whole algorithm is run over multiple iterations and the intersection of all the global probable outlier sets is returned as the final set of outliers at

each player. Since the procedure to find the global outliers is a bit different from that of the original centralized scheme, the false positive rate increases slightly in comparison to that of the centralized algorithm. Experiments show that, at the optimal value of b_t, the increase in rate of false positives is only about 0.02% [22] where as the false negatives still remain zero.

5.2 Analysis

Computational Complexity: We give the computational complexity from the perspective of player P^1. The computational complexity of step 3 is $O(kL)$. Steps 8 and 10 have computational complexities of $O(n^1dkL)$ and $O(n^1N^1_{pr}L)$ respectively. Step 13 has a constant complexity. The computational complexity of steps 17-23 depends on the average number of non-empty bins N_b created during each iteration of LSH. In the worst case where each object in the dataset is hashed to a different bin, the number of bins would be equal to the dataset size. However, the number of bins would be much less than the total data size in the average case. Thus the average case complexity for steps 17-23 is $O(N_bL)$; where $N_b << n$. The computational complexity for steps 28-36 is $O(m'^1L)$; where $m' = |M'^1|$. Considering the dominating terms, the overall computational complexity for player P^1 is $O(n^1dL)$.

Communication Complexity: In Algorithm 5, communication among the players happens in steps 4, 15 and 25. Step 4 has a communication complexity of $O(kL)$. The average case communication complexity for step 15 is $O(N_bL)$; where $N_b << n$. The corresponding average case communication complexity for step 25 is $O(N_blognL)$. Thus the overall communication complexity for Algorithm 5 is $O(N_blognL)$.

Security Analysis: In Algorithm 5, the values communicated in step 4 are public values and do not reveal any private information. After executing steps 15 and 25, each player has number of objects in the entire database that are hashed to each bin. From this, each player can infer about the distribution of the objects of all the other players but cannot infer about the distribution of any single player (since *SecureUnion* and *SecureSum* are used). However, in most cases where the objects with each player come from the same distribution, this information is usually known before hand and thus there is no extra information revealed.

Chapter 6

Conclusions

In this thesis, we have considered the problem of privacy preserving outlier detection for the case of vertically partitioned data and have come up with an algorithm which avoided the quadratic communication and computational complexities of the previously known algorithms. First, we have presented an approximation algorithm for the case of centralized data where in all the data is available with a single party. This algorithm uses the technique of Locality sensitive hashing through which a very effective pruning process was developed. The idea of *Bin Threshold* was developed in order to improve the level of approximation of the algorithm. We have provided theoretical bounds for the level of approximation of the proposed outlier detection algorithm along with empirical evidence for the same. The computational complexity of this algorithm is $O(ndL)$ where n and d are the size and dimensionality of the dataset and L is the LSH parameter.

Next, we have extend this algorithm to the case of vertical distribution of data considering privacy and gave an algorithm with a computational complexity same as that of the centralized setting which is a significant improvement from the known $O(n^2d)$ complexity. The communication complexity of the algorithm is $O(nL)$ which is again an improvement from the quadratic complexity in dataset size.

6.1 Future Work

An extension of the proposed centralized algorithm to the case of Horizontal distribution of data was already discussed although not a contribution of this thesis. One line of further improvement could be to try and extend it to the case of hybrid partitioning of data where each player can have a subset of the total objects or a subset of the dimensions of the dataset.

One other line of work could be to try and utilize the proposed LSH framework for other outlier detection methods such as density based outlier detection etc. Also, the advantage of classifying the objects of a dataset using LSH technique could be considered for other data mining tasks such as clustering and explore the possibility of developing efficient privacy preserving algorithms for those tasks.

Publications

- Madhuchand Rushi Pillutla, Nisarg Raval, Piyush Bansal, Kannan Srinathan, C.V. Jawahar, *LSH Based outlier Detection and its application in disbtributed setting*, 20^{th} ACM International Conference on Information and Knowledge Management, *CIKM* 2011, pages 2289-2292.
  (http://dl.acm.org/citation.cfm?id=2063948)

- Nisarg Raval, Madhuchand Rushi Pillutla, Piyush Bansal, Kannan Srinathan, C.V. Jawahar, *Privacy Preserving Outlier Detection using Locality Sesnsitive Hashing*, 11^{th} IEEE International Conference on Data Mining Workshops, *ICDMW* 201, pages 674-681.
  (http://dl.acm.org/citation.cfm?id=2119566)

Bibliography

[1] F. Angiulli and F. Fassetti. Very efficient mining of distance-based outliers. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, CIKM '07, pages 791–800, New York, NY, USA, 2007. ACM.

[2] S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 29–38, New York, NY, USA, 2003. ACM.

[3] M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGOD International Conference on Management of Data*, pages 93–104. ACM, 2000.

[4] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[5] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Y. Zhu. Tools for privacy preserving distributed data mining. *SIGKDD Explor. Newsl.*, 4(2):28–34, Dec. 2002.

[6] Z. Dai, L. Huang, Y. Zhu, and W. Yang. Privacy preserving density-based outlier detection. *Communications and Mobile Computing, International Conference on*, 2010.

[7] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG 2004*.

[8] Y. Duan, J. Canny, and J. Zhan. Efficient privacy-preserving association rule mining: P4p style. In *Computational Intelligence and Data Mining, 2007. CIDM 2007. IEEE Symposium on*, pages 654 –660, 1 2007-april 5 2007.

[9] M. Ester, H. peter Kriegel, J. S, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[10] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[11] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[12] S. Guha, R. Rastogi, and K. Shim. Cure: an efficient clustering algorithm for large databases. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, SIGMOD '98, pages 73–84, New York, NY, USA, 1998. ACM.

[13] D. M. Hawkins. *Identification of outliers*. Chapman and Hall, 1980.

[14] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[15] S. Jha, L. Kruger, and P. McDaniel. Privacy preserving clustering. In *Proceedings of the tenth European Symposium on Research in Computer Science*, pages 397–417, 2005.

[16] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *Knowledge and Data Engineering, IEEE Transactions on*, 16(9):1026 – 1037, sept. 2004.

[17] H. Kargupta, K. Das, and K. Liu. A game theoretic approach toward multi-party privacypreserving distributed data mining. Technical report, In In Communication, 2007.

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

[19] E. M. Knorr, R. T. Ng, and V. Tucakov. Distance-based outliers: algorithms and applications. *The VLDB Journal*, 8:237–253, February 2000.

[20] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Advances in Cryptology CRYPTO 2000*.

[21] O. L. Mangasarian, E. W. Wild, and G. M. Fung. Privacy-preserving classification of vertically partitioned data via random kernels. *ACM Trans. Knowl. Discov. Data*, 2(3):12:1–12:16, Oct. 2008.

[22] N.Raval. Private outlier detection and content based encrypted search. In *Dissertatoin at International Institute of Information Technology, Hyderabad*, 2013.

[23] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos. Loci: Fast outlier detection using the local correlation integral. *Data Engineering, International Conference on*, 0:315, 2003.

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

[25] S. J. Rizvi and J. R. Haritsa. Maintaining data privacy in association rule mining. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 682–693. VLDB Endowment, 2002.

[26] P. Rousseeuw and A.M.Leroy. Robust regression and outlier detection. In *John Wiley and Sons*, 1997.

[27] M. Shaneck, Y. Kim, and V. Kumar. Privacy preserving nearest neighbor search. *ICDM Workshops 2006*.

[28] J. Tang, Z. Chen, A. W.-C. Fu, and D. W.-L. Cheung. Enhancing effectiveness of outlier detections for low density patterns. In *Proceedings of the 6th Pacific-Asia Conference*

on Advances in Knowledge Discovery and Data Mining, PAKDD '02, pages 535–548, London, UK, UK, 2002. Springer-Verlag.

[29] Y. Tao and X. Xiao. S.: Mining distance-based outliers from large databases in any metric space. In *In: KDD (2006*. ACM Press.

[30] S. Urabe, J. Wong, E. Kodama, and T. Takata. A high collusion-resistant approach to distributed privacy-preserving data mining. In *Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference: parallel and distributed computing and networks*, PDCN'07, pages 326–331, Anaheim, CA, USA, 2007. ACTA Press.

[31] J. Vaidya and C. Clifton. Privacy-preserving outlier detection. In *ICDM 2004.*

[32] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 639–644, New York, NY, USA, 2002. ACM.

[33] J. Vaidya and C. Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 206–215, New York, NY, USA, 2003. ACM.

[34] V.Barnett and T.Lewis. Outliers in statistical data. In *John Wiley*, 1994.

[35] Y. Wang, S. Parthasarathy, and S. Tatikonda. Locality sensitive outlier detection: A ranking driven approach. In *Proc. ICDE 2011*, 2011.

[36] A. Xue, X. Duan, H. Ma, W. Chen, and S. Ju. Privacy preserving spatial outlier detection. In *ICYCS 2008*.

[37] Z. Yang, S. Zhong, and R. N. Wright. Privacy-preserving classification of customer data without loss of accuracy. In *Proceedings of the 5th SIAM International Conference on Data Mining*, 2005.

[38] J. Zhang. Advancements of outlier detection: A survey. *EAI Endorsed Transactions on Scalable Information Systems*, 13(1), 2 2013.

[39] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. *SIGMOD Rec.*, 25:103–114, June 1996.

[40] Z. Zhou, L. Huanrg, Y. Wei, and Y. Yun. Privacy preserving outlier detection over vertically partitioned data. In *EBISS '09*.