

# The privacy of $k$ -NN retrieval for horizontal partitioned data — new methods and applications

Artak Amirbekyan

Vladimir Estivill-Castro

School of ICT, Griffith University,  
Meadowbrook QLD 4131, Australia  
Email: A.Amirbekyan@gu.edu.au

## Abstract

Recently, privacy issues have become important in clustering analysis, especially when data is horizontally partitioned over several parties. Associative queries are the core retrieval operation for many data mining algorithms, especially clustering and  $k$ -NN classification. The algorithms that efficiently support  $k$ -NN queries are of special interest. We show how to adapt well-known data structures to the privacy preserving context and what is the overhead of this adaptation. We present an algorithm for  $k$ -NN in secure multiparty computation. This is based on presenting private computation of several metrics. As a result, we can offer three approaches to associative queries over horizontally partitioned data with progressively less security. We show privacy preserving algorithms for data structures that induce a partition on the space; such as  $KD$ -Trees. Our next preference is our Privacy Preserving *SASH*. However, we demonstrate that the most effective approach to achieve privacy is separate data structures for parties, where associative queries work separately, followed by secure combination to produce the overall output. This idea not only enhances security but also reduces communication cost between data holders. Our results and protocols also enable us to improve on previous approaches for  $k$ -NN classification.

*Keywords:* Privacy Preserving Data Mining, Horizontally Partitioned Data, Data Structures for Associative Queries.  $k$ -NN queries.

## 1 Introduction

Never have globalization and international collaborations placed as much demand on partnerships between governments and/or corporations. Data mining has been identified as one of the most useful tools for the fight on terror and crime (Mena 2003). However, the information needed resides with many different data holders that must share their data with each other; thus, data privacy becomes extremely important. Parties may mutually not trust each other, but all parties are aware of the benefit brought by such collaboration. In the privacy preserving model, all parties of the partnership promise to provide their private data to the collaboration, but none of them wants the others or any third party to learn much about their private data.

Copyright ©2007, Australian Computer Society, Inc. This paper appeared at the Eighteenth Australasian Database Conference (ADC2007), Ballarat, Victoria, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 63. James Bailey and Alan Fekete, Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

For most data mining algorithms, the data is encoded as vectors in high dimensional space<sup>1</sup>. For these algorithms, a measure of similarity (or dissimilarity) is necessary and many times fundamental for their operation. More importantly, in large databases, attribute-vectors are the result of feature-extraction processes and constitute elements of high-dimensional spaces. These are subject to the curse of dimensionality and in such settings content-based retrieval under the vector model must typically be implemented as  $k$ -nearest-neighbour ( $k$ -NN) queries.  $k$ -NN queries produce the  $k$  items whose features most closely resemble those of the query vector according to the similarity measure. Therefore, the efficiency and scalability of  $k$ -NN methods strongly depends on that of  $k$ -NN search. And this directly impacts data mining algorithms. Thus,  $k$ -NN search is fundamental for for many data mining results.

Current database applications regularly use some form of similarity search. Well known examples include medical imaging (Korn, Sidropoulos, Faloutsos, Siegel & Proropapas 1996), and bioinformatics (Kalveci & Singh. 2001); moreover, also time-series databases (Faloutsos, Ranganathan & Manolopoulos May 1994), multimedia information systems (Subrahmanian 1999), geographical information systems (GIS) (Cheng, Dolin, Neary, Prabhakar, Ravikanth, Wu, Agrawal, El Abbadi, Freeston, Singh, Smith & Su 1997) and CAD/CAM (Berchtold 1997). These applications commonly have some distance function that acts as the similarity between two objects (among them Euclidean distance between the corresponding feature vectors). Two illustrations are the area of image databases (where the most similar images to a given image (Ankerst, Kriegel & Seidl 1998) are retrieved in this way) and molecular biology (Ankerst, Kastenmueller, Kriegel & Seidl 1999) (where features are derived from 3D shape histograms to find similar 3D proteins).

Data structures that efficiently find  $k$ -NN includes search structures include quad-trees (Samet 1984),  $KD$ -Trees (Bentley 1975), and  $R$ -Trees (Guttman 1984), and newer structures such as SR-trees (Katayama & Satoh 1997), X-trees (Berchtold, Keim & Kriegel 1996), A-trees (Sakurai, Yoshikawa, S. & Kojima 2000), and iDistance (Yu, Ooi, Tan & Jagadish 2001). The central role of these spatial indices in data mining algorithms is illustrated by  $R$ -Trees for the efficiency of the popular clustering algorithm *DBSCAN* (Ester, Kriegel, Sander & Xu 1996).

We review secure multiparty computation in Section 2 because it provides the context for our protocols. While we review some existing protocols several new extensions are provided here. In Section 3

<sup>1</sup>Attribute-vectors are the common input for learning algorithms like decision trees, artificial neural network or for clustering algorithms like  $k$ -Means or *DBSCAN*.

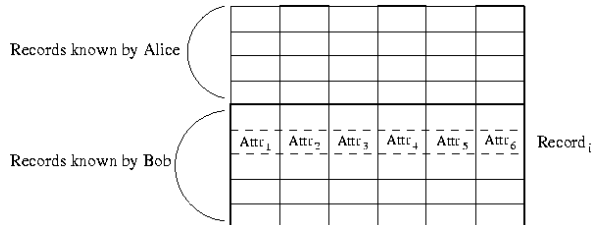


Figure 1: Horizontally partitioned data.

we introduce privacy preserving computation of some common metrics. Then, Section 4 shows privacy preserving  $k$ -NN queries. We include an analysis of what is the realistic expectation of  $k$ -NN for secure multiparty computation even in the ideal case. Section 6 demonstrates that our protocols are sufficient to produce a much better privacy preserving  $k$ -NN classifier for horizontally partitioned data. From here, we move into three models for associative queries. First, Section 7 proposes and demonstrates that the most effective approach to achieve privacy is separate data structures for parties, where associative queries work separately, followed by a secure combination protocol to produce the overall output. This idea not only enhances security but also reduces communication cost between data holders. However, most privacy preserving data mining approaches suggest a common data structure for mounting algorithms on top. The most modern and effective  $k$ -NN searcher for high dimensions with least assumptions on the metric is the *SASH* (Houle 2003b, Houle & Sakuma 2005). In Section 8, we show a privacy preserving *SASH*. We chose the *SASH* because it is neither a spatial index nor a metric index: it makes no assumptions on the nature of the database elements other than the existence of a pairwise distance measure, nor does it require the measure to satisfy the triangle inequality. Houle (Houle 2003b) has shown that for approximate  $k$ -NN queries on the largest sets, the *SASH* consistently returns a high proportion of the true  $k$ -NNs at speeds of roughly two orders of magnitude faster than sequential search. The *SASH* has been successfully applied to clustering and navigation of very large, very high dimensional text data sets (Houle 2003a), and spatial data mining of web documents (Morimoto, Aono, Houle & McCurley 2003). The fact that the *SASH* does not produce a partition of the search space adds additional security. In Section 9 we demonstrate that data structures that produce partitions (like  $R$ -Trees or  $KD$ -Trees) can be converted to the privacy preserving context but they are less secure. Section 10 offers a comparison and analysis while we conclude with final remarks in Section 11.

## 2 Privacy Preserving Computation

We study collaboration between several parties that wish to compute a function of their collective data. In fact, they are to conduct data mining tasks on the joint data set that is the union of all individual data sets. Each wants the others to find as little as possible of their own private data. To focus the discussion on privacy preserving collaboration, we will regularly use two parties we name Alice and Bob. Of course, the algorithms we describe here enable us to involve more than two parties, but for simplicity we will use only two parties. We focus on horizontally partitioned data (see Fig. 1). Some of the records are owned by Alice and the others by Bob. In the case of more than two parties, then every party will own some part (a number of records) from the database. A direct and naive use of data mining algorithms on the

union of the data requires one party to receive data (every record) from all other parties, or all parties to send their data to a trusted central place. The recipient of the data would conduct the computation in the resulting union. In settings where each party must keep their data private, this is unacceptable. Note that, for horizontally partitioned data, the more parties are involved, the more records are involved and the larger is the global database.

For simplicity, we may assume each party owns one record only, so the number  $P$  of parties is also the number  $m$  of records. Typically there would be more records than parties (as in Fig. 1 where two parties have data for 9 records). However, we consider Alice as 4 virtual parties (one for each of the records) and Bob as 5 virtual parties each controlling one of Bob’s records. This simplifies the notation in some of the algorithms (and communication between two virtual parties of the same party just does not need to occur).

The setting for our methods is privacy preserving computation originally developed under the name of “secure multiparty computation” (SMC) (Goldreich 1998). Here Alice holds one input vector  $\vec{x}$  and Bob holds an input vector  $\vec{y}$ . They both want to compute a function  $f(\vec{x}, \vec{y})$  without each other learning anything about each other’s input except what can be inferred from  $f(\vec{x}, \vec{y})$ . Yao’s Millionaires Problem (Yao 1982) provides the origin for SMC. In the millionaires, Alice holds a number  $a$  while Bob holds  $b$ . They want to identify who holds the larger value (they compute if  $a > b$ ) without neither learning anything else about the others value. The function  $f(\vec{x}, \vec{y})$  is the predicate  $f(a, b) = a > b$ . There are now many solutions (Atallah & Du 2000) improving Yao’s original solution (that required exponential complexity on the number of bits of  $(a + b)$ ; however, a recent solution (Cachin 1999) is linear on the number of bits of  $(a + b)$ ). We also adopt the *semi-honest* (Goldreich 1998) model for secure computation, which means both parties will follow the protocol since both are interested in the results. They will attempt to discover data (infer information) that they do not own, but will not supply false information in the protocol, neither will they fail to complete the protocol. To be totally precise one would have to produce proofs for the notion to “infer information  $f(\vec{x})$  from  $\vec{x}$ ”. Such definition (Goldreich 1998) involves simulated computation in polynomial time when  $\vec{x}$  is known. However, for space reasons we will not be so rigorous and we will use the following definitions.

**DEFINITION 2.1** We say that algorithm  $A$  is more secure (preferred) than algorithm  $B$  if from the output of algorithm  $A$  one can infer less information<sup>2</sup> than from algorithm  $B$ .

Note that adversaries will always know at least a set of possible values of the confidential data. We express this as a bounding box or range.

**EXAMPLE 2.1** Alice has private data  $\vec{p}$ , and after applying algorithm  $A$ , Bob can discover a bounding box  $BB_A$  where  $\vec{p}$  lies. However, after applying algorithm  $B$ , Bob can discover another bounding box  $BB_B$ . If  $BB_B \subset BB_A$ , then we say algorithm  $A$  is more secure (preferred) than algorithm  $B$ .

**EXAMPLE 2.2** Alice has private data  $\vec{p}_1, \vec{p}_2, \dots, \vec{p}_n$  and after applying algorithm  $A$ , Bob can discover bounding boxes  $BB_{A_i}$  with  $\vec{p}_i \in BB_{A_i}$ , for  $i = 1, \dots, n$ . After applying algorithm  $B$ , Bob can discover boxes  $BB_{B_j}$  with  $\vec{p}_j \in BB_{B_j}$ , for  $j = 1, \dots, n$ . If  $BB_{B_j} \subset BB_{A_j}$  (for all  $j$ ), then we say algorithm  $A$  is more secure (preferred) than algorithm  $B$ .

<sup>2</sup>Less information means not only discovering approximate values for less number of private values, but it includes also a larger bounding box for data approximations.

## 2.1 Review of the Commodity Server

Although not strictly necessary, for performance reasons, we can use the help from an extra server, the commodity server, belonging to a third party. Alice and Bob could then send request to the commodity server and receive data (called commodities) from the server, but the commodities should be independent of Alice's or Bob's private data. The purpose of the commodities is to help Alice and Bob conduct the desired computation. The third party is semi-trusted in the following sense: (1) The third party should not be trusted; therefore it should not be possible (for this third party) to derive the private information of the data from Alice or Bob; it should not learn the computation result either. (2) The third party should not collude with both Alice and Bob. (3) The third party follows the protocol correctly. Because of these attributes, we say that the third party is a semi-trusted party. In the real world, finding such a semi-trusted third party is much easier than finding a trusted third party. As we will see from our solutions, the commodity server does not participate in the actual computation between Alice and Bob, it only supplies commodities that are independent of Alice and Bob's private data. Therefore, the server can even generate independent data off-line beforehand, and sell them as commodities to the prover and the verifier (hence the name "commodity server"). The commodity server model have been used previously in the literature (Beaver 1998) for solving private information retrieval problems.

## 2.2 Review of the Scalar Product Protocol

In this protocol, Alice has a vector  $\vec{x}$  and Bob has another vector  $\vec{y}$  (both with  $n$  elements). Alice and Bob use the protocol to compute the scalar product  $\vec{x}^T \cdot \vec{y}$  between  $\vec{x}$  and  $\vec{y}$ , such that Alice gets  $V_1$  and Bob gets  $V_2$ , where  $V_1 + V_2 = \vec{x}^T \cdot \vec{y}$  and  $V_2$  is randomly generated by Bob. Namely, the scalar product of  $\vec{x}$  and  $\vec{y}$  is divided into two secret pieces, with one going to Alice and the other going to Bob. The computation is performed in the domain of the reals  $\mathfrak{R}$  and by *Theorem 4.1* in (Du & Zhan 2002) both Alice and Bob cannot learn each other's private data.

**Protocol** (Scalar Product Protocol)

1. The Commodity Server generates two random vectors  $\vec{R}_a$  and  $\vec{R}_b$  of size  $n$ , and lets  $r_a + r_b = \vec{R}_a^T \cdot \vec{R}_b$ , where  $r_a$  (or  $r_b$ ) is a randomly generated number. Then the server sends  $(\vec{R}_a; r_a)$  to Alice, and  $(\vec{R}_b; r_b)$  to Bob.
2. Alice sends  $\vec{x}' = \vec{x} + \vec{R}_a$  to Bob, and Bob sends  $\vec{y}' = \vec{y} + \vec{R}_b$  to Alice.
3. Bob generates a random number  $V_2$ , and computes  $(\vec{x}'^T \cdot \vec{y}') + (r_b - V_2)$ ; then he sends the result to Alice.
4. Alice computes  $(\vec{x}'^T \cdot \vec{y}' + (r_b - V_2)) - (\vec{R}_a^T \cdot \vec{y}') + r_a = \vec{x}^T \cdot \vec{y} - V_2 + (r_b - \vec{R}_a^T \cdot \vec{R}_b + r_a) = \vec{x}^T \cdot \vec{y} - V_2 = V_1$ .

The communication cost of this protocol is  $4n$ , which is 4 times more expensive than the optimal cost of a two-party scalar product (the optimal cost of a scalar product is defined as the cost of conducting the product of  $\vec{x}$  and  $\vec{y}$  without the privacy constraints, namely one party just sends its data in plain to the other party). The cost can be further improved to  $2n$  because the vectors  $\vec{R}_a$  and  $\vec{R}_b$  are random generated by the commodity server; therefore the commodity server can send just the seeds (numbers of constant

size) to Alice and Bob, and the seeds can be used to compute the random vector. Solutions to the scalar product protocol have been proposed before (Atallah & Du 2000, Vaidya & C. Clifton 2002) and both of these solutions achieves the security without using a third party. While the commodity server is not strictly necessary for our algorithms, in practice the communication and computation costs is more expensive than the solution without it that have a large constant under the  $O(n)$  complexity, where  $n$  is the size of the vectors.

## 2.3 Extension to the Add Vectors Protocol

The technique was introduced for manipulation of vector operations as the "permutation protocol" (Du & Atallah 2001) and is also known as the "permutation algorithm" (Vaidya & C. Clifton 2003)). In this protocol, Alice has a vector  $\vec{x}$  while Bob has vector  $\vec{y}$  and a permutation  $\pi$ . The goal is for Alice to obtain  $\pi(\vec{x} + \vec{y})$ ; that is Alice obtains the sum  $\vec{s}$  of the vectors in some sense. The entries are randomly permuted, so Alice cannot perform  $\vec{s} - \vec{x}$  to find  $\vec{y}$ . Also, Bob is not to learn  $\vec{x}$ . The solution is based on homomorphic encryption for which many implementations are possible. The protocol works as follows.

1. Alice produces a key pair for a homomorphic public key system and sends the public key to Bob. We denote by  $E(\cdot)$  and  $D(\cdot)$  the corresponding encryption and decryption system.
2. Alice encrypts  $\vec{x} = (x_1, \dots, x_n)^T$  and sends  $E(\vec{x}) = (E(x_1), \dots, E(x_n))^T$  to Bob.
3. Using the public key from Alice, Bob computes  $E(\vec{y}) = (E(y_1), \dots, E(y_n))^T$  and uses the homomorphic property to compute  $E(\vec{x} + \vec{y}) = E(\vec{x}) \times E(\vec{y})$ . Then, he permutes the entries by  $\pi$  and sends  $\pi(E(\vec{x} + \vec{y}))$  to Alice.
4. Alice decrypts to obtain  $D(\pi(E(\vec{x} + \vec{y}))) = \pi(\vec{x} + \vec{y})$ .

We will extend this protocol to the case of  $P \geq 3$  vectors, that is  $P > 2$  parties are involved. In this case there is no need to permute the result, because  $D(\cdot)$  is known only by Alice, and Alice will get the value  $E(\vec{v}_2 + \dots + \vec{v}_P)$ , where  $\vec{v}_i$  is the vector owned by  $i^{th}$  party. The algorithm is as follows:

1. The 1<sup>st</sup> party (Alice), generates  $E(\cdot)$  and  $D(\cdot)$ , then sends only  $E(\cdot)$  to the other parties.
2. Then, the  $P^{th}$  party encrypts his data  $E(\vec{v}_P)$  and sends it to  $(P - 1)^{th}$  party.
3. Next, the  $(P - 1)^{th}$  party encrypts his data  $E(\vec{v}_{P-1})$  and using homomorphic encryption property computes

$$E(\vec{v}_{P-1}) \times E(\vec{v}_P) = E(\vec{v}_{P-1} + \vec{v}_P)$$

and sends this to the  $(P - 2)^{th}$  party.

4. The protocol continues until Alice (the first party) will get  $E(\vec{v}_2 + \dots + \vec{v}_{P-1} + \vec{v}_P)$  and she adds her data in the same way.
5. As Alice owns  $D(\cdot)$ , she decrypts the results and sends them to all other parties.

Note that in Step 1, the  $P^{th}$  party does not need to permute his result because the  $(P - 1)^{th}$  party does not know  $D(\cdot)$  to decrypt. In this case  $E(\cdot)$  could be as simple as adding random number (or X-OR with a random bit mask) and consequently  $D(\cdot)$  will be subtracting the random number previously added.

## 2.4 New Protocol for the Maximum Value in the Sum of Vectors.

We will look at the two party case separately from three or more parties since these require slightly different solutions.

**Two Party Case:** Consider two attribute vectors  $\vec{x}$  and  $\vec{y}$ , where only Alice knows  $\vec{x}$  and only Bob knows  $\vec{y}$ . (where  $\vec{x} = (x_1, \dots, x_n)^T$  and  $\vec{y} = (y_1, \dots, y_n)^T$ ). The goal is to obtain  $\max_i(x_i + y_i)$ . There is a variant that obtains the index  $i_0$  for which  $x_i + y_i$  is maximum. However, the protocol should never reveal both, because then the parties find a value for the other. For example, Alice would find  $\max_i(x_i + y_i) - x_{i_0} = y_{i_0}$ . Even if the maximum value is the only outcome revealed, and the protocol is ideal, each party learns the value  $x_{i_0} + y_{i_0}$  of the maximum, and thus, each party can subtract its data vector to find  $n$  values among which on value is from the other party. For example, Alice can compute  $x_{i_0} + y_{i_0} - x_i$  (for  $i = 1, \dots, n$ ) and she finds  $n$  values one of which must be  $y_{i_0}$  owned by Bob.

Bob starts by generating a random value  $r$  and using  $\vec{y} + r = (y_1 + r, \dots, y_n + r)^T$  in the add vector protocol from Section 2.3. This provides Alice with  $\pi(\vec{x} + \vec{y} + r)$ ; that is, Alice obtains the sum vector in permuted order, which will suffice for Alice to find the maximum and inform Bob. If the maximum value is the outcome sought, Alice provides only the value  $x_{i_0} + y_{i_0} + r$ . Bob will subtract  $r$  and pass  $x_{i_0} + y_{i_0}$  to Alice. If the index of where the maximum is sought, then Alice provides only the index of where she found the maximum and Bob applies  $\pi^{-1}$  to broadcast the index. Note that Alice cannot learn which of the coordinates provided the maximum value, until Bob broadcast it. Note however, only when the maximum is sought, Alice learns the random number  $r$  and all the values in the entries of  $\pi(\vec{x} + \vec{y})$ , but this is not enough to learn any of the Bobs private data.

**The Multiparty Case ( $P > 2$ ):** Here, direct use of our extension to the add vectors protocol will reveal slightly more information that we would like to; namely, Alice obtains the sum of the vectors without the effect of the permutation (see Subsection 2.3). She is able not only to find the maximum, but the index  $i_0$  which hold the maximum of  $x_i + y_i$  as well. So, Alice will know a bit more than the others. Therefore, we will improve this protocol. The proposed protocol works as follows. Assume there are  $P > 2$  parties and every party has a vector  $\vec{v}_i$ .

1. The 1<sup>st</sup> party (Alice), generates a random vector  $\vec{R} = (r_1, \dots, r_n)$  and sends it to the  $P^{th}$  party.
2. Then, the  $P^{th}$  party adds his vector  $\vec{v}_P$  to the random vector received from Alice and sends the sum to the  $(P - 1)^{th}$  party.
3. The protocol continues until Bob (the second party) receives the sum  $\vec{S}_b = \vec{v}_3 + \dots + \vec{v}_P + \vec{R}$
4. Alice and Bob use our finding the maximum value in sum (or the index) for  $P = 2$  with the vectors  $\vec{S}_a = \vec{v}_1 - \vec{R}$  owned by Alice and  $\vec{S}_b = \vec{v}_3 + \dots + \vec{v}_P + \vec{R} + r$  owned by Bob (Bob randomly generates  $r$ ), which allows Alice to obtain  $\pi(\vec{S}_a + \vec{S}_b) + r$ ; that is the sum (translated by  $r$ ) of the vectors in permuted order.
5. Alice finds and passes the maximum value to Bob who subtracts  $r$  and broadcast the result to the other parties (or alternatively, if the index is sought, Alice will pass the index to Bob who will apply  $\pi^{-1}$  and broadcasts the index to all parties).

**Lemma** Under the semi-honest model, for  $P > 2$ , the version FINDING MAXIMUM VALUE IN THE SUM OF VECTORS protocol does not allow any party to learn any party's value nor the index for where such maximum lies.

**Proof:** The lemma holds in the multiparty ( $P > 2$ ) case because of the random vector generated by Alice masks the data from the 3<sup>rd</sup> party to the  $P^{th}$ . For Alice and Bob (the first and second parties), the protocol is secure as per the two party case, except that this time, even if Alice subtracts all the values in its vector from the maximum value, she only obtains partial sums and not value from the other parties.  $\square$

A similar result applies to the version that broadcast the index but not the maximum value. In the literature, there are several algorithms to find privately not the maximum value but the index of the maximum value in a sum of vectors (Estivill-Castro 2004, Vaidya & C. Clifton 2003). Such algorithms are very costly; some require a theoretical construction hard to implement (Vaidya & C. Clifton 2003) while others (Estivill-Castro 2004) required  $P$  iterations of the add vectors protocol and a matrix of random values, plus  $m$  iterations of a division protocol. Those algorithms become slightly more efficient if they allow one party to find all the sums. Note that our index version of the algorithm does not allow this to happen (only a translation of the sums is found by Alice).

## 3 Privacy Preserving Metrics

Here Alice has again a vector  $\vec{x}$  while Bob has vector  $\vec{y}$ . We introduce here secure computation of the Euclidean distance between these vectors. Alice replaces each component  $x_i$  with three components  $x_i^2, -2x_i, 1$  while Bob replaces each  $y_i$  component with  $1, y_i, y_i^2$ . The dot product for these three components will then be  $x_i^2 - 2x_i y_i + y_i^2 = (x_i - y_i)^2$ . In general

$$\sum_i (x_i - y_i)^2 = \left( \sum_i x_i^2, -2x_1, \dots, -2x_n, 1 \right) \cdot \left( 1, y_1, \dots, y_n, - \sum_i y_i^2 \right),$$

and thus the Euclidean distance between two feature vectors can also be expressed as a scalar product of two vectors. Hence one could use the secure scalar product protocol (Du & Zhan 2002) (see Section 2.2), to compute a secure Euclidean distance. The result is two pieces of information  $V_1$  and  $V_2$ , with  $V_1$  going to Alice,  $V_2$  going to Bob and  $\sum_i (x_i - y_i)^2 = V_1 + V_2$ .

It is known that the Euclidean distance is not robust in higher dimensions due to the many terms involved in the sum and the potential that a few values are magnified by squaring them. In other words, distinguishing what is close and what is far becomes very difficult. This is called the *curse of dimensionality*. To ameliorate this unpleasant fact, other metrics may be needed. We explore here the chessboard distance which is defined as

$$L^\infty(\vec{x}, \vec{y}) = \max(|x_1 - y_1|, \dots, |x_n - y_n|).$$

Note that

$$(|x_1 - y_1|, \dots, |x_n - y_n|)^T = |\vec{x} - \vec{y}| = |\vec{x} + (-\vec{y})|.$$

In order to compute the chessboard distance securely we can use our protocol for finding the maximum value in a sum of vectors for two parties. Note that Alice learns  $\pi(|x_i - y_i|)$  which is not sufficient to learn values from Bob.

Other metrics that weight attributes differently and used in  $k$ -NN classification can also be computed securely using a strategy similar to the one above (for example, variants like  $(\vec{x} - \vec{y})^T W (\vec{x} - \vec{y}) = \sum_i w_i (x_i - y_i)^2$  where  $W$  is a diagonal matrix of weights known to all parties). It should now be clear that it is also possible to compute securely the cosine metric  $\vec{x}^T \cdot \vec{y} / (\|\vec{x}\| \|\vec{y}\|)$  as  $(\vec{x} / \|\vec{x}\|)^T \cdot (\vec{y} / \|\vec{y}\|)$ .

#### 4 Privacy Preserving $k$ -NN Queries

This section describes our privacy preserving  $k$ -NN algorithm. Later, we will show that with this operation we can build a privacy preserving *SASH*.

For the PP- $k$ -NN protocol we are given a set of vectors

$$\vec{v}_1^T = (v_{11}, \dots, v_{1n}), \dots, \vec{v}_m^T = (v_{m1}, \dots, v_{mn}) \quad (1)$$

and a vector  $\vec{q}^T = (q_1, q_2, \dots, q_n)$ , where  $m$  is the number of records/vectors involved in the computation. The goal is to find  $NN(\vec{q}, k)$  where  $NN(\vec{q}, k)$  is the set of indices of the  $k$  nearest neighbours to the vector  $q$ . Assume the query vector  $\vec{q}$  and the first  $l < m$  vectors are owned by Alice while the other  $m - l$  vectors are owned by Bob. In the case of the Euclidean distance, the distance values between  $\vec{q}$  and  $\vec{v}_i$ , be partially distributed between Alice and Bob. Alice will have

$$V_{q, v_{l+1}}^1, \dots, V_{q, v_m}^1 \quad (2)$$

and Bob will have

$$V_{q, v_{l+1}}^2, \dots, V_{q, v_m}^2, \quad (3)$$

where  $dist(\vec{q}, \vec{v}_i) = V_{q, v_i}^1 + V_{q, v_i}^2$ ,  $l < i \leq m$ . Of course, Alice will have also the distance values for her own data. At this stage, Alice and Bob can perform the ADD VECTOR PROTOCOL with the values that determine  $dist(\vec{q}, \vec{v}_i) = V_{q, v_i}^1 + V_{q, v_i}^2$ ,  $l < i \leq m$ . Alice receives these values shuffled by the permutation  $\pi$  that Bob knows. Alice finds among these values and her own  $dist(\vec{q}, \vec{v}_i)$ ,  $1 < i \leq l$ , the  $k$  smallest. If any came from Bob's, she lets know the indexes  $j$  to Bob and Bob returns  $\pi^{-1}(j)$  to Alice. Then, Alice broadcasts the indexes of all  $k$ -NN. Note that Alice learns all the distances from  $\vec{q}$  to data points of Bob. We will discuss this issue later in Section 7.

**Theorem 4.1** *The PP- $k$ -NN protocol does not allow either Alice or Bob to learn each other's private data/vectors.*

**Proof:** In the first step of the PP- $k$ -NN protocol, Alice obtains (2) and Bob obtains (3) as a result of the secure scalar product. The next step applies the secure add vector protocol (see Section 2.3) which allows Alice to learn the distance values. But, because the distances obtained by Alice were shuffled by Bob, Alice cannot learn the values in List (3). Clearly, Bob only learns the list of values in List (3) and the indexes of the  $k$ -NN. This, of course, is not enough to disclose Alice's private data.  $\square$

This protocol is efficient, because the secure scalar product can be implemented very efficiently in linear time on the dimension  $m$  of the data. The secure scalar product is executed at most  $m$  times for  $O(mn)$  time. The secure add vectors depends on homomorphic encryption (which can be implemented with RSA). Because the add vector protocol for PP- $k$ -NN operates here with at most  $m$  values, the complexity of this phase only depends on  $m$  and not the dimension of the data. The total complexity is  $O(mn)$  time, which is linear in the input as per List (1).

#### 5 The Ideal Case For Privacy Preserving $k$ -NN queries

In this section we analyse what is the best possible security we can expect for a  $k$ -NN query. Assume Alice has a database  $D_1 = \{\vec{a}_1, \dots, \vec{a}_m\}$  and query vector  $\vec{q}$ , while Bob has database  $D_2 = \{\vec{b}_1, \dots, \vec{b}_m\}$ . They want to compute privately

$$k\text{-NN}(\vec{q}, D_1, D_2) := \langle z_1, \dots, z_k \rangle_{D_1 \cup D_2},$$

where  $z_1, \dots, z_k$  are the indexes of the vectors, which are  $k$ -NNs to the query point  $\vec{q}$ .

The literature of SMC has an ideal theoretical solution for this problem. In fact, it is always possible to securely compute  $f(\vec{x}, \vec{y})$  for a polynomial-time  $f$  using private input  $\vec{x}$  from Alice and private input  $\vec{y}$  from Bob so that Alice learns nothing about  $\vec{y}$  except what can be computed from  $f(\vec{x}, \vec{y})$  and similarly Bob learns nothing about  $\vec{x}$  except what can be inferred from  $\vec{y}$  (Goldreich, Micali & Wigderson 1987). There is still a lot of interest in protocols for SMC because (1) the general solution requires  $f$  to be explicitly represented as a Boolean circuit of polynomial size and (2) the constants involved are not small, once the circuit is described the parties enter into a protocol holding shares of the inputs to gates and shares of the outputs of gates. In data mining settings this becomes impractical because even if represented as a circuit of polynomial size in its input, the input would represent the entire data sets of all the parties. Moreover, the community in the field keeps discovering more efficient solutions for special cases of  $f$ .

If such an ideal (theoretically possible) protocol for  $k$ -NN were constructed from describing the function as a circuit, then every party obtains only the indexes  $z_1, \dots, z_k$ . However, they also obtain what can be inferred from this. Assume  $z_{i_1}, \dots, z_{i_l}$  are indexes of vectors that belongs to Alice, and  $z_{i_{l+1}}, \dots, z_{i_k}$  are the those of vectors that belongs to Bob. If Bob constructs the  $k - (l + 1)^{th}$  order Voronoi diagram (Chazelle & Edelsbrunner 1987, Lee 1982) with his data, he can discover a cell/bounding box (BB) for the query vector  $\vec{q}$ . These cells are not regular, and in particular, the one for  $\vec{q}$  could be extremely small even if the number  $k - (l + 1)$  is small. Naturally, the more of Bob's vectors among the  $k$ -NNs of  $\vec{q}$ , the more likely a more accurate cell/BB. Thus, in the ideal (theoretically possible) case, even Bob is able to discover a cell/BB where the query point  $\vec{q}$  lies.

On the other hand, if Alice computes

$$d_{min}^a = \min_{a_i \in D_1 \text{ and } a_i \notin \{a_{z_{i_1}}, \dots, a_{z_{i_l}}\}} (dist(\vec{q}, \vec{a}_i)),$$

then Alice will know that all Bob's vectors that were included in  $k$ -NN are in the hyper-sphere centred by  $\vec{q}$  with radius  $d_{min}^a$ .

#### 6 Applications to Classification

Learning classifiers is a common machine learning or data mining task where from a training set of labelled cases, a classifiers is constructed to find the class of new instances. Famous approaches for this include Artificial Neural Networks, Decision Trees, Decision Lists and Support Vector Machines (Witten & Frank 2000). Recently, a solution for privately constructing  $k$ -NN classifiers in horizontally partitioned data has been proposed (Kantarcioglu & Clifton 2004). This solutions is limited because it requires a non-colluding untrusted third party and is computationally costly —  $O(P^2 k^2)$  where  $P$  is number of parties.

Let us now see how we can use our PP  $k$ -NN protocol to solve this same problem far more efficiently and without a third party. We assume there are  $P > 2$  parties and the query point  $\vec{q}$  belongs to the first party. This is also an improvement from the previous solution that required the query point to be public (in our protocol, let the party that owns the query point be the first party). So, we are given  $D_1, \dots, D_P$  databases, where  $D_i = \vec{p}_1^i, \dots, \vec{p}_n^i$  are the attribute vectors for the  $i$ -th party. The  $t$  possible classes are identified by a set  $L = l_1, \dots, l_t$  of labels. Each point in  $D_1 \cup \dots \cup D_P$  has a label in  $L$  attached. The task is to find which of  $l_j \in L$  corresponds to the query point  $\vec{q}$  by taking a majority vote in the labels associated to the  $k$ -NNs of  $\vec{q}$ . Now, using our PP  $k$ -NN protocol (see Section 4), the parties can compute the  $k$ -NN of the query vector  $\vec{q}$  (although Alice will also know all the distances). What we need now is to classify  $\vec{q}$ . The problem setting consists now of each party holding his/her training data vectors that are included in the global  $k$ -NN of the query point  $\vec{q}$ . Consequently, they know labels (classification) associated with their vectors. We want to compute which of the labels is the most present in the overall  $k$ -NN vectors. Assume the labels among the  $k$ -NN vectors are

$$L^{k\text{-NN}} = \langle (1)_{i_1}, \dots, l_{i_{p_1}}^{(1)}, l_{i_1}^{(2)}, \dots, l_{i_{p_2}}^{(2)}, \dots, l_{i_1}^{(m)}, \dots, l_{i_{p_m}}^{(m)} \rangle$$

where  $p_1 + p_2 + \dots + p_P = k$  and  $l_{i_1}^{(j)}, \dots, l_{i_{p_j}}^{(j)}$  are the labels of the vectors that the  $j$ -th party contributes to the  $k$ -NN. Then label for  $\vec{q}$  will be

$$l^q = \text{Majority}(L^{k\text{-NN}}). \quad (4)$$

Hence, the challenge is how to compute (4) privately.

In fact, this is equivalent to finding the row with largest sum where each party owns a column, because  $l_{i_1}^{(j)}, \dots, l_{i_{p_j}}^{(j)}$  are the labels of the vectors that are known by the  $j$ -th party, he can compute sums of every label present in  $l_{i_1}^{(j)}, \dots, l_{i_{p_j}}^{(j)}$ , if there are labels that were not present, just put 0. This is the column known by the  $i$ -th party. For example, assume the  $j$ -th party labels are  $(l_1, l_2, l_2, l_1, l_4, l_4, l_1)$ , then he will have  $(3, 2, 0, 2, 0, \dots, 0)$  as a representative column.

Hence, every party will know its representative vector with length  $t$ . The protocol must add them and find the index of the maximum value. This will represent the class label that will be assigned to  $\vec{q}$ . We now use our index-version of the FINDING MAXIMUM IN THE SUM OF VECTORS protocol (see Section 2.4) to add all representative vectors, and then Alice (the first party) can find the maximum value and the permuted index where the maximum lies. Because Bob was the author of permutation  $\pi$ , then Bob can apply  $\pi^{-1}$  to discover the original index where the maximum value lies. This will represent the label's index in  $L$ , so  $\vec{q}$  will be labelled.

**Example:** Assume there are three parties and after computation of representative vectors they have  $R^{(1)} = (2, 0, 0, 3, 2, 0)$ ,  $R^{(2)} = (0, 1, 2, 0, 4, 0)$ ,  $R^{(3)} = (0, 0, 1, 3, 1, 1)$ . When we apply the secure add vectors protocol Alice gets a permutation of  $R = (2, 1, 3, 6, 7, 1)$ , so the id that represents the maximum value here is 5, then Bob needs to find the real id of the maximum and broadcast it. Thus, label assigned for  $\vec{q}$  will be  $l_{\pi^{-1}(5)}$ .

Our solution is less costly than the solution requiring a third party.

## 7 Private Combination of Local Associative Queries

Because the data is horizontally partitioned, every party owns part (a number of records) of the database. Our algorithm in Section 4 allows Alice to learn all the distances from its data point  $\vec{q}$  to the data points of Bob. In this section we discuss the rationale for this. First, note that if  $\vec{q}$  were public (Kantarcioglu & Clifton 2004), then each party could compute local  $k$ -NN and each would have a set of  $k$  distance values. The problem then reduces to find the  $k$ -richest millionaires and is solvable by repeated application of Yao's Millionaires Problem.

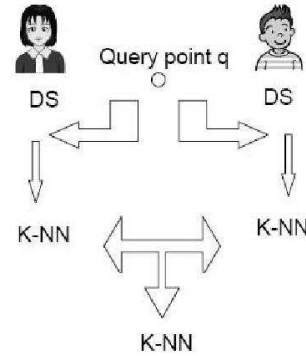


Figure 2: How local results combined to obtain result for joint database.

Figure 2 illustrates how the combining proceeds for two parties, but it is also applicable when more parties involved in the computation.

1. Alice and Bob use data structures (DS) to organise their data locally.
2. When the public query point  $\vec{q}$  comes, both of them calculate the local  $k$ -NN for their data using their own data structures.
3. Then,  $k$  repetition of Yao, Millionaires problem determines the  $k$  smallest values for  $\text{dist}(\vec{q}, \vec{v})$  among the parties.

Now, if we assume that  $\vec{q}$  is private and owned by the first party, Alice, we have some complications if Bob learns  $\text{dist}(\vec{q}, \vec{v})$  for vectors  $\vec{v}$  in Bob's database (even when distances between the query point and Bob's vectors can be calculated using PP metrics as in Section 3). When the query point  $\vec{q}$  is owned by Alice and Bob receives the results of the distance calculations, then Bob has the possibility to locate the query point  $\vec{q}$ . For instance, say Bob knows  $\text{dist}(\vec{q}, \vec{v}_1) = d_1$  and  $\text{dist}(\vec{q}, \vec{v}_2) = d_2$ . From the first fact, Bob will know that  $\vec{q}$  is in the hyper-sphere centred at  $\vec{v}_1$  and with radius  $d_1$ . From the second fact, again  $\vec{q}$  is in the hyper-sphere centred at  $\vec{v}_2$  and with radius  $d_2$ . Hence,  $\vec{q}$  is in the intersections of this two hyper-spheres. With a third point  $\vec{v}_3$ , Bob will learn  $\vec{q}$  is at the intersection of 3 hyper-spheres. This may provide significant information to identify values for  $\vec{q}$ , or very small regions where it lies. When the query point  $\vec{q}$  owned by Alice and Alice receives the results of distance calculations, Alice learns only that Bob's vectors are in the hyper-sphere centred at  $\vec{q}$  and the radius depending on the distances from  $\vec{q}$ . Even, if Alice knows all the distances from the query point  $\vec{q}$  and indexes for Bob's vectors, Alice is unable to find the exact location of any Bob's vectors. Because all the hyper-spheres are centred at  $\vec{q}$ . These hyper-spheres either coincides with each other or do not intersect each other. That is the rationale why we allow the owner of the query point to learn the distance values.

## 8 The SASH Data Structure

In this section we show a strategy by which we limit the number of distances that one party learns when the  $k$  nearest neighbours to one of its data points is performed. The idea comes from approaches where the parties share a common data structure (Du & Zhan 2002, Amirbekyan & Estivill-Castro 2006). Not that when parties share a data structure, they share the nodes and the pointers among the nodes, but not the information stored at the nodes. If we want to share a search data structure, but preserve privacy we recommend the *SASH*. In this multi-level search structure queries are processed by recursively locating approximate neighbours within the sample, and then using the pre-established connections to discover neighbours within the remainder of the data set. We describe our privacy preserving version of the algorithms of the *SASH* data structure. We consider  $n$  objects for which a similarity measure  $dist(u, v)$  exists between any two objects  $u$  and  $v$ . Since the *SASH* assumes only a metric  $dist$ , we can use any of the metrics between attribute-oriented vectors for which we have presented SMC protocols.

The directed edge-weighted graph that constitutes the *SASH* is shared by the parties. While each database object corresponds to a unique node, only the party that owns that record (vector) will know the data and the index of that vector, the others will only know who is the owner of the node. Thus, the following makes no distinction between a node and the database object to which it corresponds. Nodes are organised into a hierarchy of levels, ranging from a bottom level containing  $\lfloor n/2 \rfloor$  nodes (the leaves), to a top level containing a single node (the root). With the possible exception of the top level, each level contains half as many nodes as the level below it, rounded down. The levels of the *SASH* are numbered from 1 (the top level) to  $h$  (the bottom level). Edges within the *SASH* link nodes from consecutive levels. Each node can have edges directed to at most  $p$  parent nodes as the level above it, and to at most  $c$  child nodes as the level below it. Every node except for the root must have at least one parent. In the discussion below, we will assume that each party stores (with each edge  $(u, v)$ ) the data that enables them to compute securely  $k$ -NN regarding the distance  $dist(u, v)$  (for example, in the case of Euclidean distance, the value  $dist(u, v)$  may be partitioned among the parties as a sum of values add into to  $dis(u, v)$ ). Every node  $v$  (other than the root) has an edge directed to one parent  $g(v)$  that is designated as its guarantor. The guarantor of  $v$  must have  $v$  as one of its children;  $v$  is called the dependent of  $g(v)$ . The requirement that every node have a guarantor ensures that every node is reachable from the root.

The edges of the *SASH* heuristically minimise the distances between their endpoints. During the construction, each new node is attached to a small number of its near neighbours from the level above it. *SASH* edges do not induce a partition of the objects as lower levels. At the start of construction, the *SASH* is empty, and the parties pick a random and uniform order on the totality of the data to insert the database objects. As a result of this, the parties alternate being the party that owns the query point  $\vec{q}$  and because the internal  $k$ -NN queries in the *SASH* are only for levels with restricted number of nodes, the party does not get to learn distances to all data points of the other parties.

Let  $SASH_i$  denote the graph induced by the nodes from level 1 through  $i$ , for  $1 \leq i \leq h$ .  $SASH_i$  is a *SASH* in itself. The construction of the entire *SASH* (that is,  $SASH_h$ ) proceeds by iteratively constructing  $SASH_1, SASH_2, \dots, SASH_h$  in order.

The following algorithm shows how to build securely  $SASH_l$  given  $SASH_{l-1}$ , for  $1 \leq l \leq h$ , by adding edges between nodes of the current last two levels.

*Algorithm Privacy Preserving ConnectSASHLevel(l):*

1. If  $l = 2$ , then every node of level 2 will have the root node as its sole parent and guarantor, and the root node will have all nodes of level 2 as its children and dependents. This completes the construction of  $SASH_2$ .
2. Otherwise, for the remaining steps, we have  $l > 2$ . For each node  $v$  of level  $l$ , the parties choose a set of up to  $p$  near neighbours  $P_i(v, p)$  from among the nodes of each level for  $i = 1$  to  $l$ :
  - (a) If  $i = 1$ , then  $P_i(v, p)$  consists of a single node, the root.
  - (b) Otherwise,  $i > 1$ . Let  $P'_i(v)$  be the set of distinct children of the nodes of  $P_{i-1}(v, p)$ . Set  $P_i(v, p)$  to be the  $p$  nodes of  $P'_i(v)$  closest to  $v$ , according to the measure  $dist$  using our privacy preserving  $k$ -NN operation. If  $|P'_i(v)| < p$ , then set  $P_i(v, p) = P'_i(v)$ .
3. Set the parents of  $v$  to the nodes in  $P_{l-1}(v, p)$ . Each element  $v$  at level  $l$  has up to  $p$  distinct parents associated with points in its vicinity.
4. Create the child edges for the nodes of level  $l - 1$ , as follows:
  - (a) For each node  $u$  of level  $l - 1$ , determine the list of distinct nodes  $C(u)$  of level  $l$  that have chosen  $u$  as a parent.
  - (b) Use our Privacy Preserving  $k$ -NN to find the  $c$  closest points to  $u$  among those in  $C(u)$ .
  - (c) Set the children of  $u$  to be these  $c$  nearest neighbours.
5. For each node  $v$  of level  $l$ , determine whether it was accepted as a child of any node at level  $l - 1$ . If yes, then the closest node that accepted it as a child becomes the guarantor  $g(v)$  of  $v$ , and  $v$  becomes a dependent of  $g(v)$ . Otherwise, label  $v$  as an orphan node.
6. For each orphan node  $v$  at level  $l$ , a node at level  $l - 1$  is needed to act as its guarantor. The node should be close as possible to  $v$  (in terms of the distance measure), and must be unencumbered; that is it must have fewer than the maximum allowed number  $c$  of children. Find a guarantor for  $v$  by successively doubling the size of the candidate parents set as follows:
  - (a) Set  $i = 1$
  - (b) Compute  $P_{l-1}(v, 2^i p)$  as in Step 2.
  - (c) If  $P_{l-1}(v, 2^i p)$  has no unencumbered node, increment  $i$  and go to 6b.
  - (d) Otherwise, choose as the guarantor  $g(v)$  the unencumbered node of  $P_{l-1}(v, 2^i p)$  that is closest to  $v$ . Add  $v$  as a child and dependent of  $g(v)$ , and replace the parent of  $v$  furthest from  $v$  by  $g(v)$ .

This completes the construction of the privacy preserving  $SASH_l$ . Note that the information shared by the parties is all the edges (parent/child relationships) and results of  $k$ -NN queries that identify only owners of vectors but do not reveal the data associated with those vectors. It may be necessary to demonstrate to all other parties that all the local data is involved in the process. The *SASH* does not partition the search

space, but a *KD-Tree* or an *R-Tree* does. Our case study is *KD-Trees* but the conclusions apply to *R-Trees*, since in fact, a node in an *R-Tree* is a bounding box for all data below that node.

## 9 Privacy Preserving *KD-Trees*

Multidimensional binary search trees (*K-Dimensional search trees* or *KD-Trees*)<sup>3</sup> are a generalisation of binary search trees for multidimensional points. A *KD-Tree* for a set of *K*-dimensional records is such that:

1. Each node contains a *K*-dimensional record and has an associated discriminant  $j \in \{1, 2, \dots, K\}$ .
2. For every node with key  $\vec{x}$  and discriminant  $j$ , any record in the left subtree with key  $\vec{y}$  satisfies  $y_j < x_j$  and any record in the right subtree with key  $\vec{y}$  satisfies  $y_j > x_j$ .
3. The root node has depth 0 and discriminant 1. All nodes at depth  $d$  have discriminant  $(d \bmod K) + 1$ .

A *KD-Tree* can be built by successive insertion into an initially empty *KD-Tree*. When inserting a key  $\vec{x}$ , we compare the key to be inserted with some key  $\vec{y}$  at the root of some subtree: if  $\vec{y}$  is at level  $j$ , we compare  $x_{(j \bmod K)+1}$  and  $y_{(j \bmod K)+1}$ , and recursively continue the insertion in the left or the right subtree of  $\vec{y}$ , until a leaf (empty subtree) is found.

The construction of a shared *K*-dimensional search tree enables one party to learn bounding boxes for the other party's data. Without loss of generality, we will construct a secure solution for two parties and assume that  $K = 2$  (we are in 2D). So, for the moment, the number  $P$  of parties is 2 (as well as the dimension  $m$  of the attribute vectors). Extensions to more parties and cases where each party holds more than two attributes are easy to infer once we present the initial case. So, in what follows we have a setting where the data points look like  $\vec{p} = (p_1, p_2)$  where both  $p_1$  and  $p_2$  are known by only one party. Because we assumed two parties and the data is horizontally partitioned into two-dimensional domains for each party, we take it that the first party (Alice) knows a fraction of the total number of vectors while the other one (Bob) knows the other part (some number of vectors). All parties will know the structure of the tree; that is, all parties will know how many nodes are at each level and what records fall to the left or to the right on each internal node. The data at a node is only known by the party that owns the vector at that node. The other party only knows they are not the owner. As the tree is constructed, both parties hold an empty tree. A random order is jointly decided on the totality of the data. Both parties insert the first point, one will be the owner. Later points are included by the recursive insertion. As a data point arrives, there are two possibilities, the data point and the root of the subtree are owned by the same party. Then, that party performs the comparison and announces the result without any other part discovering here anything about the data values of the record being inserted or of the subtree's root. However, because data is partitioned horizontally, there would be cases where the vector being inserted and the subtree root node are owned by different parties. In this case, they can use Yao's comparison protocol (Yao 1982) to compare the two values. Thus, parties never announce their values, only the outcome of comparisons. Once this secure comparison of levels is set in the *KD-Tree*, all other algorithms for associative queries can be set up in the privacy preserving *KD-Tree*.

<sup>3</sup>Here  $K = m$  the dimension of the vectors, while  $k$  means the number of nearest neighbours.

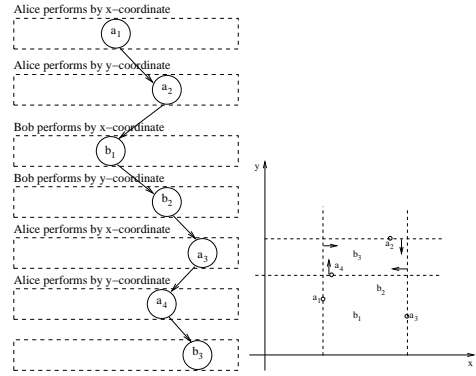


Figure 3: Example of partial 2D-tree structure and how Alice obtains range where  $\vec{b}_3$  lies.

The *KD-Tree* is a sufficient data structure on small data sets that fit in RAM (main memory) and for which we are not performing many deletions and insertions of records (or locks for multi-user access or transactions) as it would be the case of a multi-dimensional database. Although large databases that use concurrent access and use external disks (I/O to store on disk besides RAM) require *R-Trees* or other partition-based data structure, we hope that the *KD-Tree* illustration shows how this data structures could be shared with some level of privacy for associative queries. However, as we mentioned before, when the data structure is shared, the tree structure is known to all parties involved. The question is: How much can be inferred from this kind of information? Unfortunately, quite rapidly one party can learn bounding boxes on the data of the other party. It is not uncommon that on average, with a tree of depth 7, one party would have compared values with 4 of its data points. All data of other parties that fall in the bounding box determined by those four points are known to be inside such bounding box. According to the algorithm's output both Alice and Bob know the structure shown in Figure 3, but they do not know the values of the each others points. Now let us see what Alice can infer from the output and her data about Bob's vector  $\vec{b}_3$ . Figure 3 shows how Alice obtains the approximate(range where the vector lies) location of  $\vec{b}_3$ . Here, because the *KD-Tree* constructed by arranging the values to the left or to the right, according to the secure comparison, Alice using her values and already constructed *KD-Tree* (she has as an output) detects the approximate location of the vector owned by Bob.

Of course with vectors of higher dimensions (more than 2) and more parties (more than 2), the level of privacy is higher. With  $D$  dimensions one party constructs a bounding box after having  $D \times 2$  points in a path to some leaves.

## 10 Comparison and Analysis

While it is more effective and efficient that each party compute the  $k$ -NN on its data and then to merge them as in our privacy preserving  $k$ -NN algorithm, the *SASH* allows the parties to monitor the participation of other parties. We have assumed the semi-honest model, but if each party uses their own data structure to answer locally its  $k$ -NN query, in reality, one party may chose to keep away always some different sections of their data as they see fit. With a shared data structure, this is not possible. The data in the shared structure will be involved in many associative queries and many  $k$ -NN, perhaps the entire process of classification or clustering. Clearly, the methods are equivalent if one party decides from the

beginning never to involve some set of records, but then, the results will not be accurate for the union of all records anyways.

We have shown that even in the ideal case,  $k$ -NN queries tend to disclose some information about the data of the parties involved. Even in the ideal case, bounding boxes or bounding regions are found for some of the data owned by other parties. From the perspective of Definition 2.1, the separate data structures and secure  $k$ -NN is the most secure option, but parties have less certainty that all others have contributed their entire data sets. The secure *SASH* allows parties to learn slightly more about the data values of other parties, but provides more assurances that all parties have contributed their data (otherwise the shared *SASH* can not be constructed) or that the data placed into the shared data structure is the effective union of the data. It does not enable to infer directly bounding boxes for other party's records. Partition data structures, like *KD*-Trees (or *R*-Trees) are the least private as they enable one party to immediately learn bounding boxes for the other's data. All these options incur a communication overhead. However, this is insignificant if we consider the overhead for the transmission of all data to a trusted server which performs the desired analysis on the join data.

### 10.1 Time Complexity Analysis

The complexity, of course, depends of the data structures, that every party uses locally and the number of distances calculated for local  $k$ -NN, plus after combining of the local  $k$ -NN query results the communication cost for sending the overall result to all parties. The cost of communication in our algorithms is much less than the communication cost when all parties migrate their data to a trusted party. For example, in the  $PP$ - $k$ -NN protocol when  $\vec{q}$  is public, one can easily see that the cost of communication is only applied to the vectors which are candidates for the  $k$ -NN query (see Figure 2), which is clearly cheaper than bringing all the data together.

In terms of CPU-time overhead, we have shown that all the algorithms induce only a linear time overhead. In all dictionary data structures, when the data is high-dimensional, the CPU-time costs are mainly those associated with metric evaluation. We have chosen *SASH*, because this data structure is very efficient in this manner. The approximate  $k$ -NN queries proceeds by choosing  $P_i(q, k)$  at every level of the *SASH*, then combing them and choosing  $k$  closest to the query vector  $\vec{q}$ . In the *SASH* (Houle & Sakuma 2005, Houle 2003b) there are two strategies for queries: uniform and geometric. Their paper suggests that the geometric pattern improves both accuracy and search time. In geometric search instead of finding at every level  $P_i(q, k)$ , one looks for  $P_i(q, k_i)$ , where  $k_i = \max\{k^{1-\frac{h-i}{\log_2 n}}, \frac{1}{2}pc\}$ , for all  $1 \leq i \leq h$ . The upper bounds on the number of distance computations for the *SASH* (Houle & Sakuma 2005, Houle 2003b) are as follows.

*SASH* construction:  $pcn \log 2n$

Approx.  $k$ -NN query (uniform):  $ck \log 2n$

Approx.  $k$ -NN query (geom.):  $\frac{k^{1+\frac{1}{\log 2n}}}{k^{\frac{1}{\log 2n}} - 1} + 2p^3 \log 2n$

Note that the cost of the geometric pattern query is less than uniform query. The cost of privacy would then add a multiplicative constant to these costs.

## 11 Final remarks

The development of privacy preserving versions of *SASH* and *KD*-Trees reveals that if the structure is shared between parties, it provides some unnecessary information (see Figure 3) and the information leak is worse as the data structure is organised around partitions of the space. That is, although the construction data structures is secure, its operation can disclose some private information. While these may seem unsatisfactory, the fact remains that the protocols and algorithms presented here are the most practical know to our knowledge. They allow some level of protection at essentially affordable cost (the CPU-time is affected only by a constant factor). Other methods are essentially theoretical because the calculations we want to perform here cannot be written into a small circuit with as many inputs as all the data points.

We have presented several privacy preserving metrics and based on this, an algorithms to obtain  $k$ -NN with some level of preserving privacy. This provide some practical methods for applications in classification and clustering with considerations to privacy.

## References

- Amirbekyan, A. & Estivill-Castro, V. (2006), Privacy preserving DBSCAN for vertically partitioned data, in 'IEEE Int. Conf. on Intelligence and Security Informatics, ISI 2006', Springer Verlag LNCS 3975, San Diego, CA, USA, pp. 141–153.
- Ankerst, M., Kastenmueller, G., Kriegel, H. & Seidl, T. (1999), Nearest neighbor classification in 3D protein databases, in 'In Proc. 7th Int. Conf. on Intelligent Systems for Molecular Biology (ISMB-99)', pp. 34–43.
- Ankerst, M., Kriegel, H. & Seidl, T. (1998), 'A multi-step approach for shape similarity in image databases', *IEEE Transactions on Data Engineering* **10**(6), 996–1004.
- Atallah, M. & Du, W. (2000), Secure multi-party computational geometry, in 'Proc. of the 7th Int. Workshop on Algorithms and Data Structures', LNCS 2125, Springer Verlag, pp. 165–179.
- Beaver, D. (1998), Server-assisted cryptography., in 'NSPW '98: Proc. of the 1998 workshop on New security paradigms', ACM Press, New York, NY, USA, pp. 92–106.
- Bentley, J. (1975), 'Multidimensional binary search trees used for associative retrieval.', *Communications of the ACM* **18**(9), 509–517.
- Berchtold, K. H. P. (1997), Similarity search in CAD database systems, in 'Proc. of ACM SIGMOD Int. Conf. on Management of Data', Tucson, Arizona, pp. 564–567.
- Berchtold, S., Keim, D. A. & Kriegel, H.-P. (1996), The X-tree: An index structure for higher dimensional data, in 'Proc. 22th VLDB Conf.', pp. 28–39.
- Cachin, C. (1999), Efficient private bidding and auctions with an oblivious third party, in 'Proc. of the 6th ACM Conf. on Computer and Communications Security', SIGSAC, ACM Press, Singapore, pp. 120–127.
- Chazelle, B. & Edelsbrunner, H. (1987), 'An improved algorithm for constructing k-th order voronoi diagrams', *IEEE Transactions of Computers* **C**(36), 1349–1354.

- Cheng, X., Dolin, R., Neary, M., Prabhakar, S., Ravikanth, K., Wu, D., Agrawal, D., El Abbadi, E., Freeston, M., Singh, A., Smith, T. & Su, J. (1997), Scalable access within the context of digital libraries, in 'Proc. of the Int. Conf. on Advances in Digital Libraries, ADL', Washington, D.C., pp. 70–81.
- Du, W. & Atallah, M. (2001), Privacy-preserving cooperative statistical analysis, in 'Proc. of the 17th Annual Computer Security Applications Conf. (ACSAC)', ACM SIGSAC, IEEE Computer Society, New Orleans, Louisiana, pp. 102–110.
- Du, W. & Zhan, Z. (2002), Building decision tree classifier on private data, in V. Estivill-Castro & C. Clifton, eds, 'Privacy, Security and Data Mining', IEEE ICDM Workshop Proc., Volume 14 in the Conf. in Research and Practice in IT Series, ACS, Sydney, Australia, pp. 1–8.
- Ester, M., Kriegel, H., Sander, S. & Xu, X. (1996), A density-based algorithm for discovering clusters in large spatial databases with noise, in E. Simoudis, J. Han & U. Fayyad, eds, 'Proc. of the 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD-96)', AAAI, AAAI Press, Menlo Park, CA, pp. 226–231.
- Estivill-Castro, V. (2004), Private representative-based clustering for vertically partitioned data, in R. Baeza-Yates, J. Marroquin & E. Chávez, eds, 'Fifth Mexican Int. Conf. on Computer science (ENC 04)', SMCC, IEEE Computer Society Press, Colima, Mexico, pp. 160–167.
- Faloutsos, C., Ranganathan, M. & Manolopoulos, Y. (May 1994), Fast subsequence matching in time-series databases., in 'In Proc. ACM SIGMOD Int. Conf. on Management of Data', Minneapolis, pp. 419–429.
- Goldreich, O. (1998), 'Secure multi-party computation', Working draft, Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel. [www.citeseer.ist.psu.edu/goldreich98secure.html](http://www.citeseer.ist.psu.edu/goldreich98secure.html).
- Goldreich, O., Micali, S. & Wigderson, A. (1987), How to play any mental game (extended abstract), in A. Aho, ed., 'Proc. of the 19th ACM Annual Symposium on Theory of Computing', ACM Press, New York, pp. 218–229.
- Guttman, A. (1984), R-trees: a dynamic index structure for spatial searching, in 'Proc. ACM SIGMOD Int. Conf. on Management of Data', pp. 47–57.
- Houle, M. (2003a), Navigating massive data sets via local clustering, in 'KDD '03: Proc. of the ninth ACM SIGKDD Int. Conf. on Knowledge discovery and data mining', ACM Press, New York, NY, USA, pp. 547–552.
- Houle, M. (2003b), SASH: a spatial approximation sample hierarchy for similarity, Technical Report RT-0517, IBM Tokyo Research Laboratory. 16 pages.
- Houle, M. & Sakuma, J. (2005), Fast approximate similarity search in extremely high-dimensional data sets, in '21st Int. Conf. on Data Engineering ICDE', IEEE Computer Society, Tokyo, Japan, pp. 619–630.
- Kahveci, T. & Singh, A. K. (2001), An efficient index structure for string databases, in 'VLDB Conf.', Rome, Italy, pp. 351–360.
- Kantarcioğlu, M. & Clifton, C. (2004), Privately computing a distributed  $k$ -nn classifier, in J.-F. Boulicaut, ed., '8-th European Conf. on Principles and Practice of Knowledge Discovery in Databases PKDD', Vol. 3202, Springer Verlag LNCS, pp. 279–290.
- Katayama, N. & Satoh, S. (1997), The SR-tree: an index structure for high-dimensional nearest neighbour queries, in 'ACM SIGMOD Conf. on Management of Data', Tucson, USA, pp. 369–380.
- Korn, F., Sidropoulos, N., Faloutsos, C., Siegel, E. & Proropapas, Z. (1996), Fast nearest neighbor search in medical image databases, in 'VLDB', Mumbai, India, pp. 215 – 226.
- Lee, D. (1982), 'On  $k$ -nearest neighbors voronoi diagrams in the plane.', *IEEE Transactions of Computers* **C(31)**, 478–487.
- Mena, J. (2003), *Investigative Data Mining for Security and Criminal Detection*, Butterworth-Heinemann, US.
- Morimoto, Y., Aono, M., Houle, M. & McCurley, K. (2003), Extracting spatial knowledge from the web, in 'Int. Symp. on Applications and the Internet (SAINT)', Orlando, USA, pp. 326–333.
- Sakurai, Y., Yoshikawa, M., S., U. & Kojima, H. (2000), The A-tree: an index structure for high-dimensional spaces using relative approximation, in '26th VLDB Conf.', pp. 519–526.
- Samet, H. (1984), 'The quad-tree and related hierarchical data structures', *ACM Computing Surveys* **16(2)**, 187–260.
- Subrahmanian, V. (1999), *Principles of multimedia database systems*, Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA.
- Vaidya, J. & C. Clifton, C. (2002), Privacy preserving association rule mining in vertically partitioned data, in 'The Eighth ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining', SIGKDD, ACM Press, Edmonton, Canada, pp. 639–644.
- Vaidya, J. & C. Clifton, C. (2003), Privacy-preserving  $k$ -means clustering over vertically partitioned data, in 'Proc. of the SIGKDD-ACM Conf. of Data Mining', ACM Press, Washington, D.C., US, pp. 206–215.
- Witten, I. & Frank, E. (2000), *Data Mining — Practical Machine Learning Tools and Technologies with JAVA implementations*, Morgan Kaufmann Publishers, San Mateo, CA.
- Yao, A. (1982), Protocols for secure computation, in 'IEEE Symposium of Foundations of Computer Science', IEEE Computer Society, pp. 160–164.
- Yu, B., Ooi, C., Tan, K. & Jagadish, H. (2001), Indexing the distance: an efficient method to KNN processing, in '27th VLDB Conf.', Rome, Italy, pp. 421–430.