

Privacy Preserving Nearest Neighbor Search

Mark Shaneck
University of Minnesota
Dept. of Computer Science
Minneapolis, MN
shaneck@cs.umn.edu

Yongdae Kim
University of Minnesota
Dept. of Computer Science
Minneapolis, MN
kyd@cs.umn.edu

Vipin Kumar
University of Minnesota
Dept. of Computer Science
Minneapolis, MN
kumar@cs.umn.edu

Abstract

Data mining is frequently obstructed by privacy concerns. In many cases data is distributed, and bringing the data together in one place for analysis is not possible due to privacy laws (e.g. HIPAA) or policies. Privacy preserving data mining techniques have been developed to address this issue by providing mechanisms to mine the data while giving certain privacy guarantees. In this paper we address the issue of privacy preserving nearest neighbor search, which forms the kernel of many data mining applications. To this end, we present a novel algorithm based on secure multiparty computation primitives to compute the nearest neighbors of records in horizontally distributed data. We show how this algorithm can be used in three important data mining algorithms, namely LOF outlier detection, SNN clustering, and kNN classification. We prove the security of these algorithms under the semi-honest adversarial model, and describe methods that can be used to optimize their performance.

1 Introduction

Privacy advocates and data miners are frequently at odds with each other. In many cases data is distributed, and bringing the data together in one place for analysis is not possible due to privacy laws (e.g. HIPAA [1]) or policies. Privacy preserving data mining techniques have been developed to address this issue by providing mechanisms to mine the data while giving certain privacy guarantees. Research in this field typically falls into one of two categories: data transformation to mask the private data, and secure multiparty computation to enable the parties to compute the data mining result without disclosing their respective inputs.

In this paper we address the problem of privacy preserving *Nearest Neighbor Search* using the cryptographic approach. The problem of finding nearest neighbors is as follows: for a given data point x , and a parameter k , find the k points which are closest to the point x . In a distributed setting, with data that is horizontally partitioned (i.e. each party has a collection of data for the same set of attributes, but for different entities), the main difference is that the points in the neighborhood may no longer be entirely located in the local data set, but instead may be distributed across multiple data sets.

Many data mining algorithms use nearest neighbor search as a major computational component [46]. In this paper, we show how to incorporate our search algorithm into three major data mining algorithms, namely *Local Outlier Factor* (LOF) outlier detection [6, 7], *Shared Nearest Neighbor* (SNN) clustering [13, 14, 25], and *k Nearest Neighbor* (kNN) classification [10]. These are an important set of data mining algorithms. For example, kNN classification is highly useful in medical research where the best diagnosis of a patient is likely the most common diagnosis of patients with the most similar symptoms [35]. SNN clustering provides good results in the presence of noise, works well for high-dimensional data, and can handle clusters of varying sizes, shapes, and densities [14, 46]. LOF outlier detection provides a quantitative measure of the degree to which a point is an outlier and also provides high quality results in the presence of regions of differing densities [7, 46].

To the best of our knowledge, this is the first work that directly deals with the issue of privacy preserving nearest neighbor search in a general way. Privacy preserving approaches for kNN classification [8, 28] also require finding nearest neighbors. However in these works, the *query point* is assumed to be publicly known, which prevents them from being applied to

algorithms such as SNN clustering and LOF outlier detection. Previous work on privacy preserving outlier detection [49] required finding the number of neighbors closer than a threshold. Although this is related to the finding of nearest neighbors, it also cannot be directly adopted to compute SNN clusters or LOF outliers as it is limited in the amount of information it can compute about the points. Since our approach directly deals with the problem of finding nearest neighbors, it can be used by any data mining algorithm that requires the computation of nearest neighbors, and thus is more broadly applicable than the previous related works. For more detail on how our work differs from these previous works, we refer the reader to Section 6.

This paper makes the following contributions: we design a novel cryptographic algorithm based on secure multiparty computation techniques to compute the nearest neighbors of points in horizontally distributed data sets, a problem which, to the best of our knowledge, has never been dealt with previously. This algorithm is composed of two main parts. The first computes a superset of the nearest neighborhood (which is done using techniques similar to [49]). The second part reduces this set to the exact nearest neighbor set. We show how to extend this algorithm to work in the case of more than two parties, and we prove the security of this algorithm. In addition, we show how this search algorithm can be used to compute the LOF outlier scores of all points in the data set, to find SNN clusters in the data, and to perform kNN classification with the given data sets taken as the training data, thus showing how the nearest neighbor search can be practically applied. We show how all of these can be done while giving guarantees on the privacy of the data. We also analyze the complexity of the algorithms and describe measures that can be taken to increase the performance. We stress that while this work makes use of existing primitives, these are extended and combined in novel ways to enable the computation of the k nearest neighbors, as well as three major data mining algorithms that rely on nearest neighbor search.

The rest of this paper is organized as follows. Section 2 includes a formal description of the problem addressed, an overview of the secure multiparty primitives used in this work, as well as a brief discussion on provable security in the secure multiparty computation setting. The secure protocol for nearest neighbor search is described in Section 3. Section 4 explains the application of this protocol for higher level data mining algorithms. Next we discuss the complexity of the scheme in Section 5, outline the areas of related work in Section 6, and we conclude and outline some areas of future work in Section 7.

2 Overview

2.1 Problem Description

The objective of this work is to find the k nearest neighbors of points in horizontally partitioned data. The basic problem of k -nearest neighbor search is as follows. Given a set of data points S , and a particular point $x \in S$, find the set of points $N_k(x) \subseteq S$ of size k , such that for every point $n \in N_k(x)$ and for every point $y \in S$, $y \notin N_k(x) \Rightarrow d(x, n) \leq d(x, y)$, where $d(x, y)$ represents the distance between the points x and y . In a distributed setting, the problem is essentially the same, but with the points located among a set of data sets, i.e. for m horizontally distributed data sets S_i ($1 \leq i \leq m$), and a particular point $x \in S_j$ (for some j , $1 \leq j \leq m$), the k -nearest neighborhood of x is the set $N_k(x) \subseteq S = \cup_{i=1}^m S_i$ of size k , such that for every $n \in N_k(x)$ and $y \in S$, $y \notin N_k(x) \Rightarrow d(x, n) \leq d(x, y)$. If a distributed nearest neighbor search algorithm is privacy preserving, then it must compute the nearest neighbors without revealing any information about the other parties' inputs (aside from what can be computed with the respective input and output of the computation). In our case, we compute some extra information in addition to the actual nearest neighbor sets. While this is not the ideal solution, we argue that this work provides an important stepping stone for further work in this area. We also provide a description of what this information reveals in Section 3.1.1. For the sake of simplicity, everything is described in the setting of two parties. For each algorithm that we describe, there is a description of what needs to be done to extend it to the more general multiparty case.

2.2 Definitions

Throughout the discussion in this work, the data is assumed to be horizontally partitioned, that is, each data set is a collection of records for the same set of attributes, but for different entities. This would be the case, for example, in medical data, where the same information is gathered for different patients, or for network data, where the same set of attributes are gathered for different IP addresses in different networks. We describe the algorithms in this work from the perspective of one of the participants, and use the term “local” to indicate data contained within the data set of the party from whose perspective we are discussing, and the term “remote” to indicate data that is not “local”. Also, all arithmetic is done using modular arithmetic in a sufficiently large field F (e.g. mod p for the field \mathbb{Z}_p). We refer to this throughout the discussion as “mod F ”. Note that in order to preserve distances, this element should be larger than the largest pairwise distance of the points in the set.

In addition to the normal notion of nearest neighbor sets, we introduce the notion of an *Extended Nearest Neighbor Set*, in the context of distributed nearest neighbor search. This term is used to represent a logical extension of the nearest neighbor set, and can be described as follows. Imagine a point x in the local data set. We can find the k nearest neighbors from the local data set easily. Select one such local neighbor n . Now we can define a Partial Nearest Neighbor Set to include the point n and all local and remote points that are closer to x than n . Thus the Extended Nearest Neighbor Set would be the smallest Partial Nearest Neighbor Set that has a size greater than or equal to k . This notion will be used in our algorithm for privately finding the k nearest neighbors.

2.3 Secure Multiparty Computation Primitives

In this section, we give a brief overview of the secure multiparty computation primitives used in our algorithm, along with their requirements, and some examples of existing implementations.

2.3.1 Distance

An important part of computing the nearest neighbors of a point is to find the distance between the points securely. The requirements of the protocol are that the answer should be shared between the two parties involved, such that each share is uniformly distributed over a sufficiently large field F when viewed independently of each other, and that the sum of the shares (mod F) equals the distance between the two objects. One such distance measure has been suggested [49], in which the square of the Euclidean distance is computed, by making use of a secure dot product computation [19]. Note that this formula computes the square of the distance, however this does not affect the results of the nearest neighbor algorithm, since the square of the distance preserves order and we are interested in the nearest neighbors. Also, any distance computation can be used that computes the distance securely according to the above requirements, including any similarity or non-Euclidean distance measures.

2.3.2 Comparison

Another cryptographic primitive needed in this algorithm is a protocol for secure comparison, i.e. if two parties have numbers a and b respectively, how can they determine which is larger without revealing the actual values to each other. This was originally formulated as Yao’s Millionaire Problem [51] and can be accomplished by the general circuit evaluation protocol [20], or by other more efficient protocols [18, 22, 30]. At various points in our algorithms we make use of two variations of existing solutions for this problem. In the first variation the result is given to only one of the parties in question. In the second variation the result should be split between the two parties, such that the sum equals 1 (mod F) if the answer is true, and the sum is 0 if the answer is false (this requirement is the same as is needed in [49]). We also make use of a secure equality computation. It should be clear from the context which different versions are used.

2.3.3 Division

In Algorithm 3 in Section 4.1, we need to make use of the following primitive for secure division. In the two party case, one party has r_1 and r_3 , and the other party has r_2 and r_4 and together they compute shares z_1 and z_2 , such that

$$z_1 + z_2 = \frac{r_1 + r_2}{r_3 + r_4}.$$

This problem has existing solutions for both the two party [12] and multiparty [5, 34] case.

2.4 Provable Security

Each of the algorithms presented below are privacy preserving, which is a notion that can be proven. For an algorithm to be privacy preserving, it is enough to prove that the algorithm is secure in the *Secure Multiparty Computation* sense. This notion of security is defined in [20], and we will use the notion of *semi-honest* behavior for our proofs. A semi-honest party is one that follows the protocol as specified, but may use the results and any intermediate knowledge to try to extract additional information about the other party’s data. This is a realistic model, since in the target application scenarios all parties would have a mutual interest in the correct data mining output, while at the same time would desire guarantees that the other parties cannot learn extra information about their data. This also allows us to focus on more efficient computations,

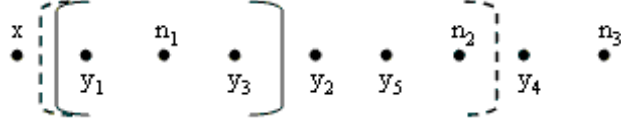


Figure 1. Example nearest neighbor scenario

since protocols that are secure under malicious adversaries require the use of expensive bit commitments and zero knowledge proofs. However, it is interesting to note that all protocols that are secure in the semi-honest model can be converted into protocols that are secure in the malicious model [20].

In addition, we make use of the composition theorem [20], which states that for functions f and g , if g is privately reducible to f and there is a protocol for privately computing f , then there is a private protocol for computing g . In order to prove that an algorithm is secure, it is sufficient to show that given the final output and the respective initial input, a party is able to simulate a valid transcript for the protocol, and that the simulated transcript is computationally indistinguishable from a transcript of a normal computation. In other words, if a party can use its own input and the final output to simulate the messages that it views during the protocol, then it is proven that the party can not learn anything extra from these messages.

3 Nearest Neighbor Algorithm

In our scenario, two parties each have a set of data points that are gathered for the same set of attributes, that is, the data is horizontally partitioned. They want to compute the nearest neighbors of their points using the union of their data sets. For a local point x , the steps to find the k nearest neighbors are as follows.

1. First, the local nearest neighbor set for x is computed.
2. The second step is to securely compute the distance from x to each of the remote points. The individual shares from this calculation are stored for use in the following steps.
3. Next, starting with the closest local neighbor of x to the farthest, the distances to all remote points are compared with the distance to the current local neighbor. Using this information, party A can count the number of total neighbors that are closer than the local neighbor.
4. This is repeated until the number of neighbors exceeds the parameter k .
5. The set of identifiers of the remote points which are closer than this local neighbor are then discovered. This set forms the Extended Nearest Neighbor Set (see definition in Section 2.2).
6. Finally, the Find Furthest Points algorithm is used to remove the remote points within this set that should not be in the actual nearest neighbor set.

The following section describes the algorithm that performs these steps. The Find Furthest Points algorithm, used in the final step, is described in Section 3.2.

3.1 Nearest Neighbor Search

In this description we will assume that we are calculating the neighborhood of a point x which is owned by the party A , A has a set of points $X = \{x_1, \dots, x_n\}$, and that the other party is B , who has a set of points $Y = \{y_1, \dots, y_{n'}\}$. What we intend to find is the set of points $nn_set \subset X \cup Y$ of size k , such that all points in nn_set are closer than all points within $(X \cup Y) \setminus nn_set$.

This is done by first computing the ordered list $local_nn$, the k nearest neighbors in the local data set X . Next the distances to all remote points are computed and stored. Then, from the closest local neighbor to the farthest, the distance to the remote points are compared with the distance to the local neighbors, in order to count the number of remote and local points that are closer than this local neighbor. This is illustrated in Figure 1. In this figure the point x and all n_i are local points, and all y_i are remote points, and $k = 3$. Thus the distance from x to each point y_i would be computed and compared first to n_1 . The

number of points in this case would be 1. Thus, the next local neighbor n_2 would need to be checked. This would result in a count of 5. Since this is larger than k , then we have found that the Extended Nearest Neighbor Set consists of all points closer than the point n_2 (including the point n_2), which is depicted by the points between the dashed braces in Figure 1. Since we know the size of the Extended Nearest Neighbor Set, we know how many points need to be excluded to get a set of the correct size, that is k . Thus, we just need to find the correct number of the furthest points from x from within the Extended Nearest Neighbor Set and remove them. In our example, the number of remote points to remove is 2, and the set of furthest points would be the set $\{y_2, y_5\}$. If these are removed (along with the furthest local neighbor n_2), then the Nearest Neighbor Set will be found, which is depicted in Figure 1 by the points within the solid braces.

In order to make this algorithm privacy preserving, we need to make use of secure distance computations and secure comparisons. When the distance between x and a remote point y is computed, as shown in step 3 of Algorithm 1, the result will be shared between the two parties, such that the sum of the shares is equal to the distance between them (modulo F). When the distance from x to y is compared with the distance to a local neighbor in step 9 of Algorithm 1, the secure comparison algorithm must be used. The result of this comparison will be shared (as c_{iy}^A and c_{iy}^B) in the same manner as the distance shares, with the sum of the shares equal to 1 (mod F) if the point y is closer than the local neighbor, and 0 if it is not. Once all points have been compared in this way, the sum of all these shares ($c_i^A = \sum_y c_{iy}^A$ and $c_i^B = \sum_y c_{iy}^B$) become shares of the number of remote points which are closer than the local neighbor n_i . This number, plus the number of local neighbors i , can be securely compared to the value k , to see if this neighbor is the final neighbor in the Extended Nearest Neighbor Set by checking if $c_i^A + c_i^B + i \geq k$. Next the identifiers of the remote points in the Extended Nearest Neighbor Set are gathered, and the entire Extended Nearest Neighbor Set is passed to the Find Furthest Points algorithm (described in Section 3.2) with the appropriate number of remote points which need to be removed from the Extended Nearest Neighbor Set, which is the size of the Extended Nearest Neighbor Set minus 1 (for the final local neighbor) minus k . This is only necessary, of course, if the size is greater than $k + 1$, since if it equals $k + 1$ then the final local neighbor is the only point that needs to be removed. Also, if it equals k , then the Nearest Neighbor Set is already found. The rest of the details of the algorithm are shown in Algorithm 1. Note in the algorithm that when a comparison is made, it is understood that the secure comparison protocol is used in the underlying implementation.

3.1.1 Security Analysis

In this algorithm, A learns the nearest neighbors and extended nearest neighbors of all its points, even if it includes points from the other party's set. Thus, for example, if A computes the Extended Nearest Neighborhood of its point x , and the resulting neighborhood has n_i as the furthest local neighbor, then A learns that the remote points in the set are closer than the point n_i . Also, A knows which points are not in the Nearest Neighbor Set, and thus knows that those points are further from the next closest local neighbor n_{i-1} . Thus, the points not in the Nearest Neighbor Set (but in the Extended Nearest Neighbor Set) lie in a particular hypershell constrained by the distances from x to n_{i-1} and n_i . This hypershell is most constrained when the distances from x to n_{i-1} and n_i are close together, which allows A to infer some information regarding the distribution of B 's points (i.e. A could discover that B has many points at a distance between the distances to these two local neighbors). This ability to estimate the distribution of B 's points decreases greatly in areas where A 's points are sparse, and also greatly decreases as the number of dimensions increases. Thus the information that is leaked (from the specified output) does not give an unreasonable amount of information to the other party.

Given that the nearest neighborhood is the specified output, then this information leakage is expected. Note that the information in the Nearest Neighbor Set does not include any information about the points themselves, just an identifier (that is, an ID that can be used to reference a particular point in the set). However, since these identifiers can be used to infer some extra information about the distribution of the other party's data set (as described above), it would be better if this information could be securely shared. This would allow further computation to be performed without releasing this intermediate information, and we intend to pursue this line of research in future work.

If the Extended Nearest Neighbor Set is included as part of the output for party A , along with the Nearest Neighbor Set, then it can be proven that this algorithm is privacy preserving.

Theorem 1 *The Nearest Neighbor calculation (Algorithm 1) privately computes the Nearest Neighbor Set and the Extended Nearest Neighbor Set of a point x in the semi-honest model.*

Proof. The private inputs of each party are their respective sets of points, with one point in one of the sets as the point whose neighborhood is being calculated, and the output is the Nearest Neighbor Set (with the Extended Nearest Neighbor Set). As can be seen from the algorithm, all communication takes place in steps 3, 9, 12, 21, and 30. In step 3 the distance is

Require: A local point x

Require: A local data set X and a remote data set Y

Require: A field F and a global parameter k

- 1: $local_nn \leftarrow$ sorted k nearest neighbors locally (from X)
- 2: **for all** $y \in Y$ **do**
- 3: Compute (and store) $distance(x, y)$ s.t. A gets d_{xy}^A and B gets d_{xy}^B where $d_{xy}^A + d_{xy}^B = distance(x, y) \pmod{F}$
- 4: **end for**
- 5: **for** $i = 1, \dots, k$ **do**
- 6: $n_i \leftarrow local_nn[i]$
- 7: $d_{xn_i} \leftarrow distance(x, n_i)$
- 8: **for all** $y \in Y$ **do**
- 9: Compute $d_{xy}^A + d_{xy}^B < d_{xn_i}$ where A gets c_{iy}^A and B gets c_{iy}^B such that $c_{iy}^A + c_{iy}^B = 1 \pmod{F}$ if $d_{xy}^A + d_{xy}^B < d_{xn_i}$, and 0 otherwise
- 10: **end for**
- 11: A computes $c_i^A = \sum_y c_{iy}^A$ and B computes $c_i^B = \sum_y c_{iy}^B$ where $c_i^A + c_i^B$ is the number of remote points closer than n_i
- 12: $g \leftarrow c_i^A + c_i^B + i \geq k$ where only A gets g
- 13: **if** g is TRUE **then**
- 14: $I \leftarrow i$
- 15: Break out of loop
- 16: **end if**
- 17: **end for**
- 18: $nn_set \leftarrow \{n_1, \dots, n_j\}$ for $j \leq I$
- 19: $d_{xn_I} \leftarrow distance(x, n_I)$
- 20: **for all** $y \in Y$ **do**
- 21: $g \leftarrow d_{xy}^A + d_{xy}^B < d_{xn_I}$ such that A gets g
- 22: **if** g is TRUE **then**
- 23: $nn_set \leftarrow nn_set \cup \{y\}$
- 24: **end if**
- 25: **end for**
- 26: **if** $|nn_set| = k + 1$ **then**
- 27: $nn_set \leftarrow nn_set \setminus \{n_I\}$
- 28: **else if** $|nn_set| > k + 1$ **then**
- 29: $\pi \leftarrow |nn_set| - 1 - k$
- 30: $nn_far \leftarrow FindFurthestPoints(x, \pi, nn_set)$
- 31: $nn_set = nn_set \setminus nn_far$
- 32: **end if**
- 33: Return nn_set

Algorithm 1: Nearest Neighbor Search

calculated, with each party receiving a random share of the distance, where each share is uniformly distributed over the field when viewed independently of each other. Thus these messages can be simulated with a value randomly selected from the field. In step 9, the distance from the point x to a remote point is compared to the distance between x and a local neighbor. This procedure is secure, and the results are again random shares, which can also be simulated as before. In step 12, the shared count is compared to the value k . Only A gets the result of this comparison, and thus can be simulated, since A knows from the Extended Nearest Neighbor Set which n_i is the furthest local neighbor, and thus when to output true and stop the comparisons. In step 21, A simply gets the identifiers of the remote points which are in the Extended Nearest Neighbor Set, which is part of its output. Finally the Extended Nearest Neighbor Set is passed to the Find Furthest Points algorithm in step 30, with the results being the points which are not in the Nearest Neighbor Set, which A can compute since it knows both the Extended Nearest Neighbor Set and the Nearest Neighbor Set. Since the Find Furthest Points algorithm is secure, as will be shown in 3.2, then applying the composition theorem to this algorithm, the Find Furthest Points subroutine, the secure distance protocol, and the secure comparison protocol proves that the nearest neighbor computation is secure in the semi-honest model. ■

3.2 Find Furthest Points

In this portion of the algorithm, we start with a local point x , and a set of points Z which contains n remote points, and we need to find the set $Z_{far} \subset Z$ of size π such that all the points in Z_{far} are further than all the points which are in $Z_{close} = Z \setminus Z_{far}$. In other words we need to find the π furthest remote points from x , but without revealing any extra information that cannot be derived from the output, such as the ordering of the points. Note that initially Z would contain both remote and local points, but the local points can be easily filtered out by A to match the description above, and thus we can assume without loss of generality that all points in Z are remote.

This can be done in the following way. First we test each point, to see if it should be placed in the set Z_{close} or Z_{far} . Since we have already computed the distances from x to each point in Z , we can compare the distance to a given point in Z with the distances to all other points in the set. If it turns out that the point is closer than enough other points, then it can be added to Z_{close} , otherwise it should be added to Z_{far} . Since the size of the set Z is n , and we are trying to determine which π points are furthest, then a given point must be closer than at least π other points. All points which pass this test will be added to Z_{close} , and those which do not pass are added to Z_{far} .

In order to make this algorithm privacy preserving, the following must be done. In this case, the distances are shared between the two parties, such that the distance to z_i from x is shared as $d_{xz_i}^A$ and $d_{xz_i}^B$, where the sum of the two shares equals the distance (modulo F), and the distance between x and z_j is shared as $d_{xz_j}^A$ and $d_{xz_j}^B$. Thus to see if the distance to z_i is less than the distance to z_j , then we need to compute $d_{xz_i}^A + d_{xz_i}^B < d_{xz_j}^A + d_{xz_j}^B$. Note that, without loss of generality, this algorithm assumes that there are no two points of equal distance away from x . The case of equal distance can be handled by breaking ties by means of the point identifiers.

This distance comparison is done in such a way that the answer is randomly split between the two parties A and B as c_{ij}^A and c_{ij}^B , such that $c_{ij}^A + c_{ij}^B = 1 \pmod F$ if z_i is closer to x than z_j and 0 otherwise. Once all the points have been compared, each party can sum all their shares ($c_i^A = \sum_j c_{ij}^A$ and $c_i^B = \sum_j c_{ij}^B$), such that the sum of these two shares is equal (mod F) to the number of points which are farther from x than z_i . If this number is greater than π then it belongs in Z_{close} , which can be computed by comparing $\pi < c_i^A + c_i^B$. Otherwise the point is added to the set Z_{far} . Then this process is repeated for all $z_i \in Z$. Once this loop completes, Z_{far} contains the π points which are furthest from x , and Z_{close} contains the rest. The pseudocode for this algorithm is found in Algorithm 2.

3.2.1 Security Analysis

Theorem 2 *The Find Furthest Points calculation (Algorithm 2) privately computes the π furthest points from x in the set Z in the semi-honest model.*

Proof. In this algorithm, the private inputs are the local point x and the set of remote points Z , along with the distance shares from x to each point in Z , and π , the number of furthest points to find. The output is the set Z_{far} of π furthest points from the set Z . The only communication in this algorithm takes place in steps 5 and 9. In step 5, two distances are compared, using the secure comparison protocol. The answer is returned as two uniformly distributed shares in the field F , such that they sum to 1 if the answer is true, and 0 if the answer is false. Since these shares are uniformly distributed when viewed independently of each other, they can also be simulated with random values from the field F . The other communication takes

Require: $x \leftarrow$ basis (local) point
Require: π is number of farthest points to return
Require: $Z = \{z_0, \dots, z_n\}$, (remote points) with $n \geq \pi$
Require: A and B share $d_{xz_i}^A$ and $d_{xz_i}^B$, for $1 \leq i \leq n$, such that $d_{xz_i}^A + d_{xz_i}^B = \text{distance}(x, z_i)$

- 1: $Z_{close} \leftarrow \{\}$
- 2: $Z_{far} \leftarrow \{\}$
- 3: **for** $i \leftarrow 1$ to n **do**
- 4: **for** $j \leftarrow i$ to $n, i \neq j$ **do**
- 5: Compute $d_{xz_i}^A + d_{xz_j}^B < d_{xz_j}^A + d_{xz_i}^B$, where A gets c_{ij}^A and B gets c_{ij}^B , such that $c_{ij}^A + c_{ij}^B = 1 \pmod F$ if $d_{xz_i}^A + d_{xz_j}^B < d_{xz_j}^A + d_{xz_i}^B$ and 0 otherwise
- 6: **end for**
- 7: A computes $c_i^A = \sum_j c_{ij}^A$
- 8: B computes $c_i^B = \sum_j c_{ij}^B$
- 9: $g \leftarrow \pi < c_i^A + c_i^B$, where only A gets g
- 10: **if** g is TRUE **then**
- 11: add z_i to Z_{close}
- 12: **else**
- 13: add z_i to Z_{far}
- 14: **end if**
- 15: **end for**
- 16: Return Z_{far}

Algorithm 2: Find Furthest Points

place in step 9, which is another secure comparison. The results of this comparison can be simulated since Z_{far} is the output and the answer will be true if the point is in the set $Z_{close} = Z \setminus Z_{far}$ and false if it is in the final set Z_{far} . Thus applying the composition theorem to this algorithm and the two invocations of the secure comparison protocol, this algorithm is secure in the semi-honest model. ■

3.3 Extension to the Multiparty Case

So far, we have concentrated on the case where two parties are involved in the protocol. In many applications, however, this algorithm would be more useful if it was extended to the case when more than two parties are involved. This is straightforward for all computations which involve at most two parties (e.g. distance computations) and thus can be performed serially, except for the cases where comparisons involve all parties. One such comparison occurs in step 12 in Algorithm 1 where the counts are shared across all parties. The other case where multiparty comparison occurs is in the Find Furthest Points algorithm (Algorithm 2), where party A has the point x and all the other parties have sets of points. In this case, the comparisons in steps 5 and 9 can involve multiple parties, and the result of the comparison (in step 5) needs to be shared. However the general solution, which is used to compute the comparisons, is applicable for the multiparty case, and thus could be used for the multi-party comparisons.

4 Applications

In this section we discuss ways in which the privacy preserving nearest neighbor search can be used, by showing three data mining algorithms that utilize nearest neighbor search.

4.1 LOF Outlier Detection

LOF outlier detection [6, 7] is a method of outlier detection that relies on relative densities of data points. This approach works well in cases where there are clusters of points with differing densities, and provides a measure of the degree to which a point can be considered an outlier. Note that in our algorithm we compute the simplified version of LOF described in [46], which computes as the LOF score the ratio of the density of a point to the average density of its neighbors. The original LOF

is a more complicated function that takes the distance to all k nearest neighbors as the distance to the k -th nearest neighbor (i.e. the reachability distance for each point in the k -distance neighborhood is simply the k -distance), and allows for variable sized nearest neighbor sets (when there are multiple points whose actual distance is equal to the k -distance). Extending our algorithm for simplified LOF to the original LOF calculation is relatively straightforward, but is omitted due to space constraints.

Before we discuss the privacy preserving algorithm for LOF outlier detection, we must first discuss what the requirements for a solution are, and how a distributed version of LOF outlier detection should work. To do this, let us first recall how the local LOF algorithm works. The LOF algorithm computes as the outlier score of a point x the ratio of the density of x times the parameter k , and the sum of the densities of the k nearest neighbors (i.e. the ratio of the density of x and the average densities of its nearest neighbors). In a distributed version of LOF, the computation of the outlier score, and thus also the computation of the densities of the points, must take into account the fact that the k nearest neighbors may not be contained within the local data set.

For an ideal secure computation (that is, one in which all inputs are sent to a trusted third party who performs the calculations locally, and returns each party's respective output), the only information that would be revealed would be the final outlier scores for each point. Also, since each party does not need to know the outlier scores of the other party's points, only the scores for the local points should be in the output for each party. Since each party can run the algorithm locally, and then compare the scores from the secure computation to the local computation, it is important to examine what information would be leaked from the secure computation. By comparing the two computed scores, a participant can infer some information about the locations of the other party's points. For example, if the other party has a dense cluster of points within the neighborhood of a local point, and the local neighborhood of that point has a relatively low density, then the outlier score could increase. If that dense cluster is very close to the local point, and it is denser than the local neighborhood, then the outlier score can decrease. However this leakage is unavoidable, since this information is leaked even in the case of the ideal computation.

4.1.1 Protocol

For the simplified version of LOF that we are using, the LOF score of a point x is

$$LOF_k(x) = \frac{density_k(x) \cdot k}{\sum_{n \in N_k(x)} density_k(n)} \quad (1)$$

where $density_k(x) = \sum_{n \in N_k(x)} distance(x, n)$. Since the distance between x and each of its neighbors is shared between the parties A and B , then shares of the density of the point x can be computed by simply summing the shares of the distances. In other words, A can compute its share of the density of the point x as $\delta_x^A = \sum_{n \in N_k(x)} d_{xn}^A$. Also, B can compute its share of the density of x as $\delta_x^B = \sum_{n \in N_k(x)} d_{xn}^B$. The shares of the sum of the densities of the neighbors (the numerator in Equation 1), $\Delta_{N_x}^A$ and $\Delta_{N_x}^B$, can be computed in the same way. In order to compute the LOF score for the point x , we just need to compute

$$LOF_k(x) = \frac{density_k(x) \cdot k}{\sum_{n \in N_k(x)} density_k(n)} = \frac{\delta_x^A k + \delta_x^B k}{\Delta_{N_x}^A + \Delta_{N_x}^B}.$$

This can be computed securely by means of the secure division primitive discussed in Section 2.3.3. The full details of this algorithm are found in Algorithm 3.

4.1.2 Security Analysis

As in the case of all the applications of the basic method, the entire algorithm is only provably secure if the Nearest Neighbor Set and the Extended Nearest Neighbor Set are included in the specified output. In this case, the Nearest Neighbor Set needs to be in the output of both parties.

Theorem 3 *The Privacy Preserving LOF algorithm (Algorithm 3) privately computes the LOF scores of each party's points, as well as the Nearest Neighbor Sets and Extended Nearest Neighbor Sets, in the semi-honest model.*

Proof. The private inputs in this case are the respective sets of data points, and the output is the outlier score of each point in the respective sets of points. In addition, the output contains the Nearest Neighbor Set along with each point's Extended Nearest Neighbor Set. The communication that takes place in this algorithm occurs in steps 2, 5, 10, and 12. In step 2 and 5,

Require: Two sets of points X and Y owned by the two parties A and B respectively

- 1: **for all** $x \in X$ **do**
- 2: Compute the nearest neighbors for x , using Algorithm 1.
- 3: Using the distance shares from the last step, A and B compute shares of the density of x : δ_x^A and δ_x^B respectively.
- 4: **end for**
- 5: Repeat steps 1-4 for all $y \in Y$
- 6: **for all** $x \in X$ **do**
- 7: $N_x \leftarrow k$ nearest neighbors for x (for which density has already been computed)
- 8: A computes $\Delta_{N_x}^A = \sum_i \delta_{n_i}^A$ for all $n_i \in N_x$
- 9: B computes $\Delta_{N_x}^B = \sum_i \delta_{n_i}^B$ for all $n_i \in N_x$
- 10: $LOF_x \leftarrow divide(\delta_x^A k, \delta_x^B k, \Delta_{N_x}^A, \Delta_{N_x}^B)$
- 11: **end for**
- 12: Repeat steps 6-11 for all $y \in Y$

Algorithm 3: Privacy Preserving LOF Outlier Detection

the nearest neighbors are being computed for each point. This was shown to be a secure computation, and the output of this protocol is the Nearest Neighbor Set (and the Extended Nearest Neighbor Set) for each point, which is part of the specified output. In steps 10 and 12, the LOF score for a point is computed using the secure division algorithm. The output of these steps is the LOF score of the point, which is part of the output. Thus applying the composition theorem to this algorithm, the density calculation, and the secure division protocol, this algorithm is secure in the semi-honest model. ■

4.1.3 Extension to Multiparty Case

To extend this algorithm to the multiparty case, all that is needed is a multiparty version of the division algorithm. The nearest neighbor detection algorithm can be extended to the multiparty case, as was shown above. Once the neighbors are found using this method, the density will be shared among multiple parties, and thus in order to get the LOF score, all that is needed is a multiparty case of the division algorithm, where there are m parties, the i -th party holds s_i and t_i , and the LOF score is

$$\frac{s_1 + \dots + s_m}{t_1 + \dots + t_m}.$$

This can be computed in the three party case [34] and in the more general m -party case [5].

4.2 SNN Clustering

SNN Clustering is a technique for clustering that defines similarity based on the number of nearest neighbors that two points share. The shared nearest neighbor similarity for two points x and y is defined to be the number of neighbors that appear in the nearest neighbor lists for both x and y , if x and y are in each other's nearest neighbor lists. If they are not in each other's nearest neighbor lists, then their similarity is 0. A graph that uses this notion of similarity is constructed and is called the SNN graph. This graph can be constructed based on information computed using our nearest neighbor search algorithm if the parties share their nearest neighbor lists (as was required for LOF outlier detection in Section 4.1).

A description of SNN Clustering techniques can be found in [46]. There are two main methods of SNN Clustering, the Jarvis-Patrick method [25] and SNN density based clustering [13, 14]. The Jarvis-Patrick clustering method is to compute the nearest neighbors of each point, find the similarity between each point, extrapolate this into a sparse graph where points are connected if their similarity exceeds a certain threshold, and designate as clusters the connected components of this graph. For SNN density based clustering, a notion of density is computed for each point that is based on the nearest neighbor similarity of the point, and then a DBSCAN [15] like algorithm is run to identify connected components of high density points. In both of these algorithms, no additional private information is required beyond what is needed to construct the similarity graph. Also, both of these can be extended to the case of more than two parties since the nearest neighbor search algorithm was shown to be extensible to the multiparty case.

4.3 kNN Classification

The problem of kNN classification [10] is as follows. Given a set of labeled records, and a given query instance, find the k nearest neighbors, and output as the predicted class label the majority class label of those neighbors. In a distributed setting, the nearest neighbors can be discovered among all the various sites. Previous approaches to solving this problem in a privacy preserving way [8, 28] assume that the query point is public knowledge for all the sites involved. However, it would be more desirable if the query point was only known to the party to which it was submitted. Using our basic nearest neighbor search algorithm, these goals can be achieved.

4.3.1 Protocol

Given the k nearest neighbors of a point x , our task in this algorithm is to find which class is represented by the most neighbors. Once the k nearest neighbors have been determined, each party knows which of its points are in the nearest neighbor set, and thus knows the class labels for these points. Thus party B , for example, knows how many of the neighbors it owns are members in each class c_i (where $1 \leq i \leq m$, and m is the number of class labels). So each party has a list of class labels with the number of neighbors in each class. In order to find the majority class, the sum of the number of neighbors in each class needs to be compared to the sum of the number of neighbors in all the other classes. The class for which the sum is greater than all other sums is the majority class.

In order to accomplish this, the only hurdle that needs to be overcome is the comparison. In the two party case this is fairly straightforward. Using the two party comparison protocol described in Section 2.3.2, a sum can be tested to see if it is the largest by comparing it to each other sum individually. If the result is true for each comparison, then it is the largest and thus represents the majority class. This can be repeated for each class label until the largest is found¹. The answers to these comparisons are shared securely between the parties, and the sum is compared for equality after all comparisons are made (the sum of the shares should equal $m - 1$). Also, the answers to the equality tests should be received only by the party who owns the point in question, so that the other parties do not learn the predicted class label of the point. This algorithm also assumes that no two classes have the same counts, and as mentioned in 3.2, ties can be broken using class identifiers. The pseudocode for this algorithm is found in Algorithm 4.

Require: A query point x given to party A
Require: A has a set of labeled data points X
Require: B has a set of labeled data points Y
Require: m is the number of total class labels

- 1: Run Algorithm 1 to find the nearest neighbors for the point x
- 2: For each class c_i , sum locally the number of neighbors in each class as s_i^A and s_i^B for party A and B respectively
- 3: **for all** class c_i **do**
- 4: **for all** class c_j , where $j \neq i$ **do**
- 5: Compute $s_i^A - s_j^A \geq s_j^B - s_i^B$, where A gets r_{ij}^A and B gets r_{ij}^B such that $r_{ij}^A + r_{ij}^B = 1 \pmod{F}$ if $s_i^A + s_i^B \geq s_j^A + s_j^B$, and 0 otherwise.
- 6: **end for**
- 7: A computes $r_i^A = \sum_j r_{ij}^A$
- 8: B computes $r_i^B = \sum_j r_{ij}^B$
- 9: Compute $r_i^A - m + 1 = r_i^B$ such that only A gets the answer
- 10: **end for**
- 11: For one of the classes, A will have received an answer of true for the equality test, and this can be output as the majority class

Algorithm 4: Privacy Preserving kNN Classification

¹If the answer of true is obtained before the last class is checked, the party A should not terminate this loop, as it will reveal the identity of the majority class to the other party.

4.3.2 Security Analysis

As in the case of all the applications of the basic method, the entire algorithm is provably secure if the Nearest Neighbor Set and the Extended Nearest Neighbor Set are included in the specified output.

Theorem 4 *The Privacy Preserving kNN Classifier (Algorithm 4) privately computes the majority class of the query point, as well as the Nearest Neighbor Set and Extended Nearest Neighbor Set, in the semi-honest model.*

Proof. The private inputs in this case are the respective sets of data points, and the query point (which is known only to the local party A). The output of the algorithm is the majority class of the query point, along with the Nearest Neighbor Set and the Extended Nearest Neighbor Set of the query point. The nearest neighbors are found securely (as was shown in Section 3) in step 1. The output of this step can be simulated with the nearest neighbor set, which is part of the output. The next communication takes place when the classes are compared to each other in step 5. The result of these comparisons are shares that are uniformly distributed when viewed independently, and thus can be simulated with a uniform random number from the field F . Finally, the sums of the shares are compared for equality. The output of this (from A 's perspective) will be true for the majority class (which is part of A 's output) and false for all the others. A can simulate the answers from these comparisons with the output. B on the other hand, gets no information about the comparisons, including which class causes the loop to terminate, since the loop will not be terminated early. Thus, by applying the composition theorem to this algorithm, the nearest neighbor algorithm, and the secure comparison primitives, this algorithm is secure in the semi-honest model. ■

4.3.3 Extension to Multiparty Case

In the case of two parties, the total sum of neighbors for a given class is shared between two parties. Thus the two sums can be compared using each party's local shares as shown in step 5 of Algorithm 4. However, in the case of three or more parties, the sums are spread across all parties, and thus the two-party comparison cannot be used. In this case a multi-party comparison algorithm could be used to compute the comparison. Alternatively, this comparison can be accomplished in the following manner, using a two-party comparison protocol. Let us take the case of 3 parties, A , B , and C . We will also consider 2 classes, C_1 and C_2 . The total counts for these two classes will be c_1 and c_2 , respectively. Now the count c_1 is shared among the three parties as c_1^A , c_1^B , and c_1^C , so that $c_1^A + c_1^B + c_1^C = c_1$. Similarly, the count c_2 is shared among the three parties as c_2^A , c_2^B , and c_2^C , so that $c_2^A + c_2^B + c_2^C = c_2$. To perform the comparison, C simply chooses at random two values r_1 and r_2 from the field F , and sends r_1 and r_2 to A , and sends $c_1^C - r_1 \pmod F$ and $c_2^C - r_2 \pmod F$ to B . At this point A and B can compare the counts to see if $c_1 > c_2$ by computing if $c_1^A + c_1^B + (c_1^C - r_1) + r_1 \pmod F > c_2^A + c_2^B + (c_2^C - r_2) + r_2 \pmod F$. This can also be extended to an arbitrary number of parties by picking one of the parties to play the role of B (A would be the party with the query instance), and having each other party in turn play the role of C . Thus the parties playing the role of A and B would accumulate random shares of each other parties counts and at the end would be able to compute the comparison. In this case the comparison results would be shared between A and B , and thus step 9 of the algorithm would remain unchanged.

This extension maintains the security of the kNN Classification protocol. Since the parties playing the role of C select the r value uniformly at random from the field, the shares that A and B get of C 's values are uniformly distributed when viewed independently. Thus the shares give no information as to C 's individual counts. Also, since all parties have a share of the total (even if a party has 0 neighbors for that class), no information is leaked as to how many neighbors each party has within each class. Thus the multiparty extension is as secure as the two party version.

5 Complexity Analysis

The overall complexity of the algorithm is dominated by the secure multiparty computation protocols (nearest neighbor search in general involves $O(n^2)$ computation and this algorithm involves $O(n^2)$ secure multiparty computation protocols). There are two major components to the complexity of these protocols: communication and computation. If we assume that we use homomorphic encryption [19] as the underlying primitive for the distance computations, we can concretely analyze the computational and communication complexity of this part of the protocol. For this analysis we will assume that only two parties are involved, and that each party has n data records, for a total of $2n$ records. Let us also assume that each data point is of dimension d , that each record contains 32 bit integers, and thus the field in which we perform the arithmetic is the integers modulo a $64 + d$ bit integer (in order to contain the largest possible dot product, where each of the d components in

the two vectors are 2^{32}). Also, the homomorphic encryption scheme that we use is Paillier’s encryption scheme [42], with a 1024 bit RSA modulus, and thus the encryption involves 2048 bit exponentiation and multiplication. Finally, we assume that Paillier’s encryption scheme has the appropriate values precomputed (the amount of time required for the precomputation will be discussed later in the section).

Now a total of n^2 pairwise distances need to be computed as the first step in the algorithm. This involves the computation of n^2 dot products. As per the dot product algorithm in [19], each component must be encrypted by the first party and sent to the second party. The second party computes the n^2 dot products and sends back encrypted random shares, which the first party must decrypt. Thus, the first party must compute dn encryptions, and send these dn values to the second party. For each dot product, the second party must compute d exponentiations and d multiplications, followed by an encryption and a multiplication. Finally, these n^2 encrypted values must be sent back to the first party, who must decrypt them all. Thus, the total computation required by the first party is dn encryptions and n^2 decryptions. If we ignore all multiplications and divisions in the encryption and decryption routines, we have the first party computing $dn + n^2 \cdot 2048$ bit exponentiations. The first party then must send $2048dn$ bits to the second party. The second party must then compute $(d + 1)n^2$ exponentiations. These $n^2 \cdot 2048$ bit values must be sent back to the first party, who must compute n^2 exponentiations. Thus, the total bandwidth required is $2048(n^2 + dn)$ bits, and the total number of 2048 bit exponentiations is $dn + n^2 + (d + 1)n^2 + n^2 = dn + (d + 3)n^2$. However, it is interesting to note that these encryptions can be highly parallelized, and thus can be greatly sped up through the use of high powered computers with multiple processors.

In addition to this, there are $O(n^2)$ comparisons that need to be computed. It is not entirely straightforward what the most efficient mechanism is to perform this computation, and thus providing a concrete implementation for this protocol is left for future work. It is important that the end solution is not only as efficient as possible, but also easy to implement.

5.1 Optimizations

We can see from the previous section that the algorithm is somewhat heavyweight, in that it is quadratic in terms of secure multiparty computation primitives, and may be difficult to execute in the case when the number of points is very large. Unfortunately, this is the case of many Privacy Preserving Data Mining solutions based on secure multiparty computation. Thus, in this section, we propose a few mechanisms that can be used to speed up the execution of the protocol, as a tradeoff in terms of the accuracy of the results.

The first is the use of sampling, which is a common mechanism used in data mining. The parties involved could select a subset of their data that will be used to find the neighbors. Thus, the neighbors can be found for each point, but the neighbors will only be located in the sample². Thus, if the sample is of size m , with $m \ll n$, then the protocols would run in time $O(mn)$ for outlier detection and clustering and time $O(m)$ for classification.

The next possibility to increase performance of the algorithm is to only run it for a restricted portion of the dataset. This would not be applicable for the classification protocol, since there is only one query point for which neighbors are found. However, the outlier detection could be run only on the local points that are anomalous locally. Since points that are in dense clusters would be both locally and jointly non-anomalous, finding the joint anomaly score for these points would give no extra information. Similarly, the clustering could be run on points that are not a part of dense clusters, since those points would likely remain in those clusters even with the joint data. It is difficult to quantify how much accuracy would be lost in this way, however it would enable the parties involved to be able to run the algorithm and receive some results, hopefully resulting in better results than if the algorithm was only run locally. Related to this idea would be the use of preclustering in order to prearrange the data, so that neighbors are only searched for in the closest cluster. This would affect the accuracy of the results in a somewhat unpredictable way, but would greatly reduce the complexity of the algorithm.

A final possibility to increase performance, which is left as an area of future work, is to investigate the field of approximate nearest neighbor search. Much work has been done in this field [4, 21, 32], and applying those principles to the field of privacy preserving nearest neighbor detection could result in faster algorithms that privately find the approximate nearest neighbor sets.

6 Related Work

Much work has been done in the field of Privacy Preserving Data Mining. There are two main approaches to this field: the data modification approach, and the cryptographic approach. The data modification approach consists of two major angles

²This technique has been used quite effectively to identify anomalous net-flows in large collections of network data using the LOF scheme [11].

as well, one being the randomization method and the other being k -anonymization. The randomization method was initially proposed by Agrawal and Srikant [3], with their work on reconstructing approximations of distribution of the original dataset from randomly perturbed values. The randomization approach has also been applied to association rule mining [16, 43] and clustering [39, 40, 41], in addition to further work on decision trees [2]. This randomization approach was shown to have some limitations [29], where under certain circumstances, the original data points were able to be recovered with fairly high accuracy, thus greatly reducing the privacy guarantees of some randomization methods. Algorithms have also been developed to compute clusters by exchanging higher level information, such as generative models [36], without exchanging the actual data and without relying on randomization.

The k -anonymization approach was largely established by the works of Sweeney and Samarati [44, 45]. The idea of k -anonymization is simply to modify the records of the data such that for every record, there are $k - 1$ other records that are indistinguishable from it. This is effective at preventing re-identification, which is the process of using fields that do not obviously identify an individual (e.g. zip code or data of birth), and combining them with some external information (e.g. a phone book), to link the identity to the record, thus breaking the privacy of that individual. The main technique to obtain a k -anonymous data set is generalization and suppression. Various techniques have been proposed in [44, 45, 23], and they essentially work by reducing the granularity of the data, making the attributes less specific. This is typically done on a selected subset of the attributes that are deemed potentially identifying, as achieving optimal k -anonymization was found to be NP-hard [37]. Anonymization has also been used for IP Addresses [17, 38].

The cryptographic approach primarily makes use of Secure Multiparty Computation ideas from the field of Cryptography [20, 51]. In this setting, two or more parties want to jointly compute the function from the combination of their inputs, such that they learn the output, yet receive no more information about the other’s input than they can discern from the output alone. Much work has been done in this area, starting with [33], where a private computation was shown for computing information gain, allowing a secure computation of ID3 decision trees. Following this, much work was done for general tools for privacy preserving data mining (including secure set operations [9, 31] and secure sum and scalar product [9]), kNN classification for horizontally partitioned data [8, 28] and vertically partitioned data [52], association rule mining [27, 47], k -means clustering [24, 26, 48], outlier detection [49], and further work on decision trees [50].

The previous work on kNN classification for horizontally partitioned data [8, 28] deserves some extra discussion, as the key step in kNN classification is to compute the k nearest neighbors of the given point to be classified. In both of these works [8, 28], the authors assumed that the query point was public knowledge (including the values for all attributes). For both works the first step for each party is to compute the k closest points from the local data sets. From this point, Chitti et al. [8] make use of a distributed randomized algorithm *PP-TopK* to compute the k smallest distances to the query point. This is followed by a secure sum algorithm to compute the totals for each class, from which the majority class can be found. In Kantarcioglu and Clifton [28], once each party computes the k closest points in their respective data sets, the distance from all mk points to the query point are compared securely to each other (where m is the number of parties involved), and the results are sent to a semi-trusted third party (after scrambling to prevent the third party from learning which party contributed which points) along with the class values for each point (encrypted using the querier’s public key). The third party can then compute the k nearest points from the mk points, and engages in a secure computation with the querier to compute the majority class label.

Since in both of these works the query point is given to each party involved, neither of these schemes can be used for LOF outlier detection and SNN clustering. The reason is that each of the data points themselves are the “query” points, and releasing them would release the entire data set. For kNN classification, both of these schemes, as well as our scheme, are potentially applicable. In our work however, the query point is assumed to be private in addition to the data sets, which is a requirement in certain applications. For example, if the query point is a patient record, that patient’s information should be kept private, in addition to the records of the patients in the other data sets. Note also that in Chitti et al., once the neighbors are found the secure sum algorithm is used to sum up the total counts for each class label. This effectively releases the counts for each class label, and discloses the class distribution among the nearest neighbors. In our extension for kNN classification however, only the majority class is found, and the additional class information is not leaked. Finally, for Kantarcioglu and Clifton a semi-trusted third party is required to find the nearest neighbors and the majority class label. This assumption is not required in our construction.

To the best of our knowledge, the only other work that has addressed outlier detection was by Vaidya and Clifton [49]. In that work they describe an algorithm to compute distance-based global outliers, where a point is declared an outlier if the distance to all other points (or a certain percentage p of the other points) is greater than some given threshold. As this requires finding all neighbors closer than a threshold, their kernel and our kernel are very similar. Both run in quadratic time and are capable of detecting the nearest neighbors within a certain distance from the point in question. In their algorithm, that

distance is dt , a predefined static distance. In our scheme however, the distance is the distance to the k -th nearest neighbor, which varies for each point. This difference is key, as it enables the computation of much more information about the points. Vaidya and Clifton's kernel allows for the computation of a binary classification for outliers - either a point is an outlier or it is not. Our kernel, however, allows for the computation of the LOF score of each point, ranking each point in its degree of outlier-ness. It also allows for outliers and clusters to be computed based on local density. Thus our algorithm is more suitable to be used as a subroutine to provide private computations of algorithms that require the computation of nearest neighbors sets, which we show in our applications to LOF outlier detection, SNN clustering, and kNN classification.

7 Conclusion

In conclusion, we have shown a protocol for privately computing the k nearest neighbors of points in a horizontally partitioned data set. We described this algorithm in the two party case and proved security for each of the parts of the algorithm. In addition, we showed how to extend the nearest neighbor computation to the multiparty case. We also showed how this algorithm could be used to compute LOF outlier scores, SNN Clustering, and kNN Classification, including security proofs and extensions of each to the multiparty case. Finally, we analyzed the complexity of the scheme and suggested some mechanisms to improve the performance of the protocol at the expense of accuracy.

Since this algorithm is most likely to be used as a step in a more complicated process, it would be desirable to hide the identities of the neighbors (or share this information securely to enable further computation), so that the end result is obtained without leaking this information. This would require a solution more tightly coupled with the information that is needed from the neighborhood (i.e. it would be highly application-dependent), and is left for future work. Also, this work is focused on horizontally partitioned data, and another area of future work would be extending it to vertically partitioned data. Finally, one last area of work would be to investigate the area of approximate nearest neighbor search, in order to develop faster private computations of the neighbors.

8 Acknowledgements

We would like to thank Nicholas Hopper for his comments on the security proofs, and Karthikeyan Mahadevan, Vishal Kher, Peng Wang, Shyam Boriah, and Varun Chandola for their helpful discussions. Portions of this work were supported by NSF Grant IIS-0308264 and NSF ITR Grant ACI-0325949, and the Army High Performance Computing Research Center under the auspices of the Department of the Army, ARL cooperative agreement number DAAD19-01-2-0014. The content of this work does not necessarily reflect the position or policy of the government and no official endorsement should be inferred. Access to computing facilities was provided by the AHPCRC and the Minnesota Supercomputing Institute.

References

- [1] 1996 Health Insurance Portability and Accountability Act. <http://www.hhs.gov/ocr/hipaa/>.
- [2] Dakshi Agrawal and Charu C. Aggarwal. On the Design and Quantification of Privacy Preserving Data Mining Algorithms. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2001.
- [3] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-Preserving Data Mining. In *Proceedings of the ACM International Conference on Management of Data*, 2000.
- [4] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An Optimal Algorithm for Approximate Nearest Neighbor Searching Fixed Dimensions. *Journal of the ACM*, 45(6):891–923, 1998.
- [5] Mikhail Atallah, Marina Bykova, Jiangtao Li, Keith Frikken, and Mercan Topkara. Private Collaborative Forecasting and Benchmarking. In *Proceedings of the Workshop on Privacy in the Electronic Society*, 2004.
- [6] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. OPTICS-OF: Identifying Local Outliers. In *Proceedings of the Third European Conference on Principles of Data Mining and Knowledge Discovery*, 1999.
- [7] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying Density-Based Local Outliers. In *Proceedings of the ACM International Conference on Management of Data*, 2000.

- [8] Subramanyam Chitti, Li Xiong, and Ling Liu. Mining multiple private databased using a privacy preserving knn classifier. Technical report, Georgia Tech, 2004.
- [9] Chris Clifton, Murat Kantarcioglu, Jaideep Vaidya, Xiaodong Lin, and Michael Zhu. Tools for Privacy Preserving Data Mining. In *SIGKDD Explorations*, 2002.
- [10] T.M. Cover and P.E. Hart. Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.
- [11] Paul Dokas, Levent Ertöz, Vipin Kumar, Aleks Lazarevic, Jaideep Srivastava, and Pang-Ning Tan. Data Mining for Network Intrusion Detection. In *Proceedings of the NSF Workshop on Next Generation Data Mining*, 2002.
- [12] Wenliang Du and Mikhail Atallah. Privacy-Preserving Cooperative Statistical Analysis. In *Proceedings of the 17th Annual Computer Security Applications Conference*, 2001.
- [13] Levent Ertöz, Michael Steinbach, and Vipin Kumar. A New Shared Nearest Neighbor Clustering Algorithm and its Applications. In *Workshop on Clustering High Dimensional Data and its Applications, Proceedings of Text Mine '01, First SIAM International Conference on Data Mining*, 2001.
- [14] Levent Ertöz, Michael Steinbach, and Vipin Kumar. Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data. In *Proceedings of the SIAM International Conference on Data Mining*, 2003.
- [15] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 1996.
- [16] Alexandre Evfimievski, Ramakrishnan Srikant, Rakesh Agrawal, and Johannes Gehrke. Privacy Preserving Mining of Association Rules. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*, 2002.
- [17] Jinliang Fan, Jun Xu, Mostafa H. Ammar, and Sue B. Moon. Prefix-Preserving IP Address Anonymization: Measurement-Based Security Evaluation and a New Cryptography-Based Scheme. *The International Journal of Computer and Telecommunications Networking*, 46(2):253–272, 2004.
- [18] Marc Fischlin. A Cost-Effective Pay-Per-Multiplication Comparison Method for Millionaires. In *RSA Security 2001 Cryptographer's Track and Lecture Notes in Computer Science*, volume 2020, pages 457–471, 2001.
- [19] Bart Goethals, Sven Laur, Helger Lipmaa, and Taneli Mielikäinen. On Private Scalar Product Computation for Privacy-Preserving Data Mining. In *Proceedings of the 7th Annual International Conference in Information Security and Cryptology*, 2004.
- [20] Oded Goldreich. Secure Multiparty Computation, manuscript. <http://www.wisdom.weizmann.ac.il/~oded/PSBookFrag/prot.ps>, 2003.
- [21] Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Symposium on Theory of Computing*, 1998.
- [22] Ioannis Ioannidis and Ananth Grama. An Efficient Protocol for Yao's Millionaire's Problem. In *Proceedings of the Hawaii International Conference on System Sciences*, 2003.
- [23] Vijay Iyengar. Transforming Data to Satisfy Privacy Constraints. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*, 2002.
- [24] Geetha Jagannathan and Rebecca N. Wright. Privacy-Preserving Distributed k -Means Clustering over Arbitrarily Partitioned Data. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*, 2005.
- [25] R. A. Jarvis and E. A. Patrick. Clustering Using a Similarity Measure Based on Shared Nearest Neighbors. *IEEE Transactions on Computers*, C22(11):1025–1034, 1973.

- [26] Somesh Jha, Louis Kruger, and Patrick McDaniel. Privacy Preserving Clustering. In *Proceedings of the 10th European Symposium On Research In Computer Security*, 2005.
- [27] Murat Kantarcioglu and Chris Clifton. Privacy-preserving Distributed Mining of Association Rules on Horizontally Partitioned Data. In *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2002.
- [28] Murat Kantarcioglu and Chris Clifton. Privately Computing a Distributed k-nn Classifier. In *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2004.
- [29] Hillol Kargupta, Souptik Datta, Qi Wang, and Krishnamoorthy Sivakumar. Random Data Perturbation Techniques and Privacy Preserving Data Mining. *Knowledge and Information Systems Journal*, 2003.
- [30] Eike Kiltz, Ivan Damgaard, Matthias Fitz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally Secure Constant Round Multi-Party Computation for Equality, Comparison, Bits and Exponentiation. In *Proceedings of the third Theory of Cryptography Conference*, 2006.
- [31] Lea Kissner and Dawn Song. Privacy Preserving Set Operations. In *Proceedings of Advances in Cryptology - CRYPTO*, 2005.
- [32] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient Search for Approximate Nearest Neighbor in High Dimensional Spaces. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, 1998.
- [33] Yehuda Lindell and Benny Pinkas. Privacy Preserving Data Mining. In *Proceedings of Advances in Cryptology - CRYPTO*, 2000.
- [34] Wenjun Luo and Xiang Li. A Study of Secure Multi-party Elementary Function Computation Protocols. In *Proceedings of the ACM International Conference on Information Security*, 2004.
- [35] Benjamin Mayer, Huzefa Rangwala, Rohit Gupta, Jaideep Srivastava, George Karypis, Vipin Kumar, and Piet de Groen. Feature Mining for Prediction of Degree of Liver Fibrosis. Poster Presentation in the Annual Symposium of American Medical Informatics Association, 2005.
- [36] Srujana Merugu and Joydeep Ghosh. Privacy-preserving Distributed Clustering using Generative Models. In *Proceedings of The Third IEEE International Conference on Data Mining*, 2003.
- [37] Adam Meyerson and Ryan Williams. On the Complexity of Optimal KAnonymity. In *Proceedings of the Twenty-third ACM Symposium on Principles of Database Systems*, 2004.
- [38] Greg Minshall. TCPdpriv Command Manual, 1996.
- [39] Stanley Oliveira and Osmar Zaïane. Privacy Preserving Clustering By Data Transformation. In *Proceedings of the 18th Brazilian Symposium on Databases*, 2003.
- [40] Stanley Oliveira and Osmar Zaïane. Achieving Privacy Preservation When Sharing Data For Clustering. In *Proceedings of the International Workshop on Secure Data Management in a Connected World*, 2004.
- [41] Stanley Oliveira and Osmar Zaïane. Privacy-Preserving Clustering by Object Similarity-Based Representation and Dimensionality Reduction Transformation. In *Proceedings of the Workshop on Privacy and Security Aspects of Data Mining*, 2004.
- [42] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Proceedings of Eurocrypt*, 1999.
- [43] Shariq Rizvi and Jayant Haritsa. Maintaining Data Privacy in Association Rule Mining. In *Proceedings of 28th International Conference on Very Large Data Bases*, 2002.
- [44] Pierangela Samarati and Latanya Sweeney. Protecting Privacy when Disclosing Information: k-Anonymity and Its Enforcement through Generalization and Suppression. Technical report, SRI International, 1998.

- [45] Latanya Sweeney. k-Anonymity: A Model for Protecting Privacy. *International Journal on Uncertainty, Fuzziness, and Knowledge-based Systems*, 2002.
- [46] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Pearson Education, Inc., 2006.
- [47] Jaideep Vaidya and Chris Clifton. Privacy-Preserving Association Rule Mining in Vertically Partitioned Data. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*, 2002.
- [48] Jaideep Vaidya and Chris Clifton. Privacy-Preserving K-Means Clustering over Vertically Partitioned Data. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*, 2003.
- [49] Jaideep Vaidya and Chris Clifton. Privacy-Preserving Outlier Detection. In *Proceedings of the Fourth IEEE International Conference on Data Mining*, 2004.
- [50] Jaideep Vaidya and Chris Clifton. Privacy-Preserving Decision Trees over Vertically Partitioned Data. In *Proceedings of the IFIP WG 11.3 International Conference on Data and Applications Security*, 2005.
- [51] A. C. Yao. How to Generate and Exchange Secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, 1986.
- [52] Justin Zhan, LiWu Chang, and Stan Matwin. Privacy Preserving K-nearest Neighbor Classification. *International Journal of Network Security*, 1(1):46–51, July 2005.