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 work 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.

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) 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 work we address the problem of privacy preserving *Nearest Neighbor Search* using the cryptographic approach. Many data mining algorithms use nearest neighbor search as a major computational component [17]. To this end, we show how to incorporate our search algorithm into three major data mining algorithms, namely *Local Outlier Factor* (LOF) outlier detection [2], *Shared Nearest Neighbor* (SNN) clustering [6, 10], and *k Nearest Neighbor* (kNN) classification [4]. 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 [15]. 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 [6, 17]. 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 [2, 17].

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 [3, 12] 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 [20] 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.

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 called the Extended Nearest Neighbor set (which is done using techniques similar to [20]). 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. However, due to space constraints we omitted the extensions to the multiparty case, security proofs, the SNN clustering and kNN classification applications, and the complexity analysis and refer the reader to the full paper [16] for these results.

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 $y \in S$, $y \notin N_x(x) \Rightarrow d(x,n) \leq d(x,y)$, 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.

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. Also, all arithmetic is done using modular arithmetic, using 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.

2.3 Secure Multiparty Computation Primitives

In this paper we make sue of the following secure multiparty computation primitives: secure distance computations [20, 8] (resulting in random shares, the sum of which - modulo an element in a large enough field - equals the distance between the two points), secure comparison using the general solution [9] (resulting in shares of the comparison, the sum of which are 1 is the result is true and 0 otherwise, as is needed in [20]), and secure division [5].

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, making use of the composition theorem [9]. In this paper we deal with the semi-honest model. This notion of security is defined in [9], and refers to a party 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, 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 [9].

3 Nearest Neighbor Algorithm

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 $X = \{x_1, \ldots, x_n\}$, and that the other party is B, who has a set of points $Y = \{y_1, \ldots, y_{n'}\}$. What we intend to find is the set of points $nn\_set \subset X \cup Y$

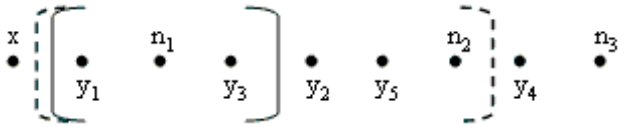

Figure 1. Example nearest neighbor scenario

of size k, such that all points in nn_set are closer to x 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. 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, 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 to be compared with the distance to a local neighbor, the secure comparison algorithm must be used (which utilizes the general solution for secure multiparty computation). 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, 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. This comparison checks if $c_i^A + c_i^B + i > k - 1$. 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.

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. 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 futher computation to be performed without releasing this intermediate information, and we intend to pursue this line of research in future work.

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 is 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 $d^A_{xz_i}$ and $d^B_{xz_i}$, where the sum of the two shares equals the distance (modulo F), and the distance between x and z_j is shared as $d^A_{xz_j}$ and $d^B_{xz_j}$. Thus to see if the distance to z_i is less than the distance to z_j, then we need to compute $d^A_{xz_i} + d^B_{xz_i} < d^A_{xz_j} + d^B_{xz_j}$, which can be accomplished using the secure comparison algorithm. 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^A_{ij} and c^B_{ij}, such that $c^A_{ij} + c^B_{ij} = 1 \mod F$ 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 ($c^A_i = \sum_j c^A_{ij}$ and $c^B_i = \sum_j c^B_{ij}$), 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 $\pi < c^A_i + c^B_i$. 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.

4 LOF Outlier Detection

LOF outlier detection [2] 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 [17], 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.

Ideally, 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. In our solution, we compute the final outlier scores as well as the nearest neighbor sets, as described above.

4.1 Protocol

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

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

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 $\delta^A_x = \sum_{n \in N_k(x)} d^A_{xn}$. Also, B can compute its share of the density of x as $\delta^B_x = \sum_{n \in N_k(x)} d^B_{xn}$. The shares of the sum of the densities of the neighbors, $\Delta^A_{N_x}$ and $\Delta^B_{N_x}$, can be computed in the same way. In order to compute the LOF score for the point x, we just need to compute

$$LOF_k(x) = \frac{density_k(x) \cdot k}{\sum_{n \in N_k(x)} density_k(n)} = \frac{\delta^A_x k + \delta^B_x k}{\Delta^A_{N_x} + \Delta^B_{N_x}}.$$

This can be computed securely by means of the secure division primitive mentioned in Section 2.3.

5 Related Work

Much work has been done in the field of Privacy Preserving Data Mining. There are two main approaches to this field: the randomization approach, and the cryptographic approach. The randomization method was initially proposed by Agrawal and Srikant [1], 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 [7]. This randomization approach was shown to have some limitations [13], 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.

The cryptographic approach primarily makes use of Secure Multiparty Computation ideas from the field of Cryptography [9]. 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 [14], where a private computation was shown for computing information gain, allowing a secure computation of ID3 decision trees. Following this, work was done for kNN classification [3, 12, 21], association rule mining [18], k-means clustering [11, 19], and outlier detection [20].

The work which is closest to ours includes work done for kNN classification [3, 12] and outlier detection [20]. For more information about how these works differ from ours, and a fuller list of related work in both the cryptographic approach and the randomization approach, we refer the reader to the full version of our paper [16].

6 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 this algorithm could be used to compute LOF outlier scores. For future work, we aim to improve the algorithm to not reveal the intermediate neighborhood information, thus reducing the potential information leakage. Also, as this work is focused on horizontally partitioned data, another area of future work would be extending it to vertically partitioned data.

7 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 AHPCRC contract number DAAD19-01-2-0014.

References

[1] R. Agrawal and R. Srikant. Privacy-Preserving Data Mining. In *Proc. of the ACM Intl. Conf. on Management of Data*, 2000.

[2] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying Density-Based Local Outliers. In *Proc. of the ACM Intl. Conf. on Management of Data*, 2000.

[3] S. Chitti, L. Xiong, and L. Liu. Mining multiple private databased using a privacy preserving knn classifier. Georgia Tech, TR, 2004.

[4] T. Cover and P. Hart. Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[5] W. Du and M. Atallah. Privacy-Preserving Cooperative Statistical Analysis. In *Proc. of the 17th Annual Computer Security Applications Conf.*, 2001.

[6] L. Ertöz, M. Steinbach, and V. Kumar. Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data. In *Proc. of the SIAM Intl. Conf. on Data Mining*, 2003.

[7] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy Preserving Mining of Association Rules. In *Proc. of SIGKDD*, 2002.

[8] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikëinen. On Private Scalar Product Computation for Privacy-Preserving Data Mining. In *Proc. of the 7th Annual Intl. Conf. in Information Security and Cryptology*, 2004.

[9] O. Goldreich. *The Foundations of Cryptography*, volume 2. Cambridge University Press, 2004.

[10] 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.

[11] S. Jha, L. Kruger, and P. McDaniel. Privacy Preserving Clustering. In *Proc. of the 10th ESORICS*, 2005.

[12] M. Kantarcioglu and C. Clifton. Privately Computing a Distributed k-nn Classifier. In *Proc. of the 8th European Conf. on Principles and Practice of Knowledge Discovery in Databases*, 2004.

[13] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. Random Data Perturbation Techniques and Privacy Preserving Data Mining. *Knowledge and Information Systems Journal*, 2003.

[14] Y. Lindell and B. Pinkas. Privacy Preserving Data Mining. In *Proc. of Advances in Cryptology - CRYPTO*, 2000.

[15] B. Mayer, H. Rangwala, R. Gupta, J. Srivastava, G. Karypis, V. Kumar, and P. de Groen. Feature Mining for Prediction of Degree of Liver Fibrosis. Poster Presentation in the Annual Symposium of American Medical Informatics Association, 2005.

[16] M. Shaneck, Y. Kim, and V. Kumar. Privacy Preserving Nearest Neighbor Search. University of Minnesota, TR 06-014, 2006.

[17] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson Education, Inc., 2006.

[18] J. Vaidya and C. Clifton. Privacy-Preserving Association Rule Mining in Vertically Partitioned Data. In *Proc. of SIGKDD*, 2002.

[19] J. Vaidya and C. Clifton. Privacy-Preserving K-Means Clustering over Vertically Partitioned Data. In *Proc. of SIGKDD*, 2003.

[20] J. Vaidya and C. Clifton. Privacy-Preserving Outlier Detection. In *Proc. of the Fourth IEEE Intl. Conf. on Data Mining*, 2004.

[21] J. Zhan, L. Chang, and S. Matwin. Privacy Preserving K-nearest Neighbor Classification. *Intl. Journal of Network Security*, 1(1):46–51, July 2005.