International Research Journal of Engineering and Management Studies (IRJEMS)
Volume: 03 Issue: 03 | March -2019 e-ISSN: 2395-0126

Privacy Preserving K-NN Classification Supported By Secure SASH

A.Durga Bhavani[#1], R.Srinivas[*2]

Computer Sceince and Engineering, Jawaharlal Nehru Technological University Kakinada.
A.Durga Bhavani, Student, M. Tech., Sri Sai Aditya Inistitute of Science &Technology, Kakinada, India

[*]*R.Srinivas, M.Tech, (Ph.D)*
HOD_CSE.Dept, Sri Sai Aditya Inistitute of Science &Technology, Kakinada, India

Abstract:
The field of privacy has seen rapid advances in recent years because of the increases in the ability to store data. Besides the storing ability computers can manipulate large databases and performs many data analysis tasks using data mining techniques. Our purpose is during such analysis to preserve privacy of individuals and corporations. In this paper we address the k-NN classification along with preserving privacy to individual data. Data may be distributed across multiple sites and the owners of the data may wish to compute a common function. In such cases a variety of protocols may be used, so that secure computation is possible without reveling sensitive information.

Index Term
Privacy preserving data mining, privacy preserving classification, horizontally partitioned data, k-NN queries, SASH data structure.

Introduction

Data mining technology allows the analysis of large amounts of data. Analyses of personal data or analyses of corporate data by competitors create threats to privacy. Data mining has been identified as one of the most useful tools for the fight on terror and crime. However the information needed resides with many different data holders. Parties may mutually not trust each other, but all parties are aware of the benefit brought by collaboration. In the privacy preserving model, all parties of the partnership promise to provide their private data to the collaboration, but none of them wants the others or any third party to learn much about their private data.

In this paper we focus on the k-NN classification algorithm on horizontally partitioned data. The objective of k-NN classification is to discover k nearest neighbors for a given instance and then assign a class label to the given instance according to the majority class of the k nearest neighbors. Our goal is not only to achieve the above objective but also to preserve the privacy.

For most data mining algorithms, the data is encoded as vectors in high dimensional space. For these algorithms, a measure of similarity (or dissimilarity) is necessary, and many times fundamental for their operation. Similarity queries on multi-dimensional data are usually implemented by finding the closest attribute-vector(s) to the attribute-vector of the query data. In such settings, information retrieval under the vector model must typically be implemented as k-nearest-neighbor (k-NN) queries, whose result consists of the k items closest to the query vector according to the similarity measure. This type of query is known as a nearest neighbor (NN) query.

2. Secure Multi Party Computation

We study col l abor at i on between several parties that wish to compute a function of their collective data. In fact, they are to conduct data mining tasks on the joint data set that is the union of all individual

data sets. Each wants the others to find as little as possible of their own private data. To focus the discussion on privacy preserving collaboration, we will regularly use two parties name Alice and Bob. We focus on horizontally partitioned data(see Fig1.). Some of the records are owned by Alice and the others by Bob. Data mining algorithms on the union of the data requires one

party to receive data (every record) from all other parties, or all parties to send their data to a trusted central place. The recipient of the data would conduct the computation in the resulting union. In settings where each party must keep their data private, this is unacceptable. Note that, for horizontally partitioned data, the more par- ties are involved, the more records are involved and the larger is the global database.
In horizontal partition different sites may have different sets of records containing the same attributes.

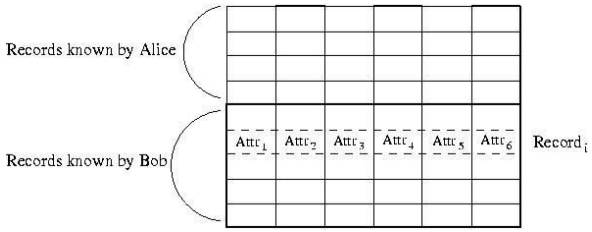

Figure 1: Horizontally partitioned data.

For simpli cit y, we may assume each party owns one record only, so the number P of parties is also the number m of records. Typically there would be more records than parties (as in Fig.1 where two parties have data for 9 records). However, we consider Alice as 4 virtual parties (one for each of the records) and Bob as 5 virtual parties each controlling one of Bob's records.

Here Alice holds one input vector $\boldsymbol{x}$ and Bob holds an input vector $\boldsymbol{y}$. They both want to compute a function $f(\boldsymbol{x},\boldsymbol{y})$ without each other learning anything about each other's input except what can be inferred from $f(\boldsymbol{x},\boldsymbol{y})$. Yao's Millionaires Problem (Yao 1982) provides the origin for SMC. In the millionaires, Alice holds a number $\boldsymbol{a}$ while Bob holds $\boldsymbol{b}$. They want to identify who holds the larger value (they compute if $a > b$) without neither learning anything else about the others value. The function $f(\boldsymbol{x},\boldsymbol{y})$ is the predicate $f(a, b) = a > b$. There are now many solutions improving Yao's original solution. We also adopt the *semi-honest* model for secure computation, which means both parties will follow the protocol since both are interested in the results.

DEFINITION 2.1 We say that algorithm A *is more secure (preferred) than algorithm* B *if from the Output of algorithm* A *one can infer less information than from algorithm* B.

Commodity Server: The Trusted third party

The commodity server is a third party. Alice and Bob can send requests to the commodity server and receive data (called commodities) from the server, but the commodities should be independent of Alice's or Bob's private data. The purpose of the commodities is to help Alice and Bob conduct the desired computation.

Scalar Product Protocol

In this protocol, Alice has a vector X and Bob has another vector Y. The scalar product of X and Y is divided into two secret pieces, with one going to Alice and the other to Bob.
The steps involved in this protocol are:

1. The commodity server generates two random vectors R_a and R_b of size n and lets $r_a + r_b = R_a . R_b$ where r_a or r_b is a randomly generated number. Then the server sends $(R_a; r_a)$ to Alice and

$(R_b; r_b)$ to Bob.

2. Alice sends $x' = x + R_a$ to Bob, and Bob sends $y' = y + R_b$ to Alice.
3. Bob generates a random number V_2, and computes $(x' \cdot y)+(r_b - V_2)$ and sends the result to Alice.
4. Alice computes $(x' \cdot y+(r_{ba} - V_2))-(R \cdot y')+ra = x\,y - V_2 + (r_b - R_a \cdot R_b + r_a) = x \cdot y - V = V_2$ 1.

Extension to Add Vectors Protocol

In this protocol Alice has a vector x while Bob has a vector y and a permutation π. Alice gets $\pi(x+y)$, that is he gets the sum S in some sense. The entries are randomly permuted, so Alice cannot performs S-x to get y. Likewise Bob also cannot learn x.

The protocol works as follows:

1. Alice produces a key pair for a homomorphic public key system and sends the public key to Bob. We denote by $E(\cdot)$ and $D(\cdot)$ the corresponding encryption and decryption system.
2. Alice encrypts $x = (x1, \cdots, xn)^T$ and sends $E(x) = (E(x1), \cdots, E(xn))^T$ to Bob.
3. Using the public key from Alice, Bob computes $E(y) = (E(y1), \cdots, E(yn))$ and uses the homomorphic property to compute $E(x+y) = E(x) \times E(y)$. Then, he permutes the entries by π and sends $\pi(E(x+y))$ to Alice.
4. Alice decrypts to obtain $D(\pi(E(x+y))) = \pi(x+y)$.

A Protocol to find the Maximum Value in the Sum of Vectors

Two Party Case: Consider two attribute vectors x and y, where only Alice knows x and only Bob knows y. The goal is to obtain $\max i\ (xi + yi)$.

T he protocol should never reveal index $i0$ for which $xi + yi$ is maximum, because then the parties find a value for the other. For example, Alice would find $\max i\ (xi + yi) - xi0 = yi0$.

Bob generates a random value r and computes $y + r = (y1 + r, \ldots, yn + r)$ in the add vector protocol. This provides Alice with $\pi(x + y + r)$; which will suffice for Alice to find the maximum and inform Bob. If the maximum value is the outcome sought, Alice provides only the value $xi0 + yi0 + r$. Bob will subtract r and pass $xi0 + yi0$ to Alice. If the index of where the maximum is sought, then Alice provides only the index of where she found the maximum and Bob applies π^{-1} to broadcast the index. Note that Alice cannot learn which of the coordinates provided the maximum value, until Bob broadcast it. Note however, only when the maximum is sought, Alice learns the random number r and all the values in the entries of $\pi(x+y)$, but this is not enough to learn any of the Bobs private data.

Privacy Preserving Metrics

Associative queries are the core retrieval operations for many data mining algorithms including k-NN search. Associative queries are based on several metrics and discussion about k-NN cannot be separated from proper discussion of the metrics.

3.1 Euclidian metric

Here Alice has again a vector x while Bob has vector y. We introduce here secure computation of the Euclidean distance between these vectors. Alice replaces each component xi with three components x_i^2, $-2xi$, , 1.

While Bob replaces each yi component with 1, yi , y_i^2 . The dot product for these three

components will then be $x_i^2 - 2 x_i y_i + y_i^2 = (x_i - y_i)^2$. In general $(\quad)^2 = (\quad)$.

and thus the Euclidean distance between two feature vectors can also be expressed as a scalar product of two vectors. Hence one could use the secure scalar product protocol, to compute a secure Euclidean distance. The result is two pieces of information V1 and V2, with V1 going to Alice, *V*2 going to Bob.

Privacy Preserving k-NN Algorithm This section describes our privacy preserving *k*-NN algorithm. Later, we will show that with this operation we can build a privacy preserving *SASH*. For the PP-*k*-NN protocol we are given a set of vectors

$$V_1^T = (v_{11}, \bullet\bullet\bullet, v_{1n}), \bullet\bullet\bullet, V_m^T = (v_{m1}, \bullet\bullet\bullet, v_{mn}) \quad (1)$$

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

$$V^1_{q,v_{l+1}}, \bullet\bullet\bullet, V^1_{q,vm} \quad (2)$$

and Bob will have

$$V^2_{q,v_{l+1}}, \bullet\bullet\bullet, V^2_{q,vm}, \quad (3)$$

where $\text{dist}(q, v_i) = V^1_{q,vi} + V^2_{q,vi}$, $1 < i \leq m$. Of course, Alice will have also the distance values for her own data. At this stage, Alice and Bob can perform the add vector protocol with the values that determine $\text{dist}(q, v_i) = V^1_{q,vi} + V^2_{q,vi}$, $1 < i \leq m$. Alice receives these values shuffled by the permutation π that Bob knows. Alice finds among these values and her own $dist(q,v_i)$, $1 < i \leq l$, the k smallest. If any came from Bob's, she lets know the indexes j to Bob and Bob returns $\pi^{-1}(j)$ to Alice. Then, Alice broadcasts the indexes of all *k*-NN. Note that Alice learns all the distances from q to data points of Bob.

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

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

The Ideal Case For Privacy Preserving k-NN Queries

Here we analyze what is the best possible security we can except for a k-NN query. Assume Alice has a database $D_1=\{a_1,...,a_m\}$ and query vector q, while Bob has database $D_2=\{b_1,...,b_m\}$. They want to compute privately

$$\text{k-NN}(q,D_1,D_2) := (z_1,..,z_k)\ D_1 \cup D_2$$

where $z_1,..,z_k$ are the indexes of vectors, which are k-NNs to q.

Assume $z_{i1},..,z_{il}$ are indexes of vectors that belongs to Alice and $z_{i1+1},..,z_{ik}$ are indexes of vectors that belongs to Bob. If Bob can discover a cell/bounding box for the query vector q, the cell for q could be extremely

small. Naturally the more of Bob's vectors among the k-NNs of q, the more likely a more accurate cell/BB will be found. Thus even in the ideal case Bob is able to discover a cell/BB where the query vector lies.

The *SASH* Data Structure

In this section we show a strategy by which we limit the number of distances that one party learns when the k nearest neighbors to one of its data points is performed. If we want to share a search data structure, but

preserve privacy we recommend the *SASH*. We consider n objects for which a similarity measure $dist(u, v)$ exists between any two objects u and v. Since the *SASH* assumes only a metric *dist*, we can use any of the metrics between attribute-oriented vectors for which we have presented SMC protocols.

The directed edge-weighted graph that constitutes the S A S His shared by the parties. While each database object corresponds to a unique node, only if the party that owns that record (vector) will know the data and the index of that vector, the others will only know who is the owner of the node. Nodes are organized into a hierarchy of levels, ranging from a bottom level containing $bn/2c$ nodes (the leaves), to a top level containing a single node (the root). The levels of the *SASH* are numbered from 1 (the top level) to h (the bottom level). Edges within the *SASH* link nodes from consecutive levels. Each node can have edges directed to at most p parent nodes, and to at most c child nodes. Every node v (other than the root) has an edge directed to one parent $g(v)$ that is designated as its guarantor. The guarantor of v must have v as one of its children; v is called the dependent of $g(v)$.

During the construction, each new node is attached to a small number of its near neighbors from the level above it. At the start of construction, the *SASH* is empty, and the parties pick a random and uniform order on the totality of the data to insert the database objects. As a result of this, the parties alternate being the party that owns the query point q and because the internal k-NN queries in the *SASH* are only for levels with restricted number of nodes, the party does not get to learn distances to all data points of the other parties.

Let *SASH i* denote the graph induced by the nodes from level 1 through i, for $1 \le i \le h$. *SASH i* is a *SASH* in itself. The construction of the entire *SASH* (that is, *SASHh*) proceeds by iteratively constructing *SASH*1 , *SASH*2 , ..., *SASHh* in order.

The following algorithm shows how to build securely *SASHl* given *SASHl*−1 , for $1 \le l \le h$, by adding edges between nodes of the current last two levels.

Algorithm Privacy Preserving ConnectSASHLevel(l):

1. If $l = 2$, then every node of level 2 will have the root node as its sole parent and guarantor, and the root node will have all nodes of level 2 as its children and dependents. This completes the construction of *SASH*2.
2. Otherwise, for the remaining steps, we have $l > 2$. For each node v of level l, the parties choose a set of up to p near neighbors $Pi(v, p)$ from among the nodes of each level for $i = 1$ to l:

(a) If $i = 1$, then $Pi(v, p)$ consists of a single node, the root.

(b) Otherwise, $i > 1$. Let $P_i(v)$ be the set of

distinct children of the nodes of P_{i-1} (v,p). Set Pi (v, p) to be the p nodes of P_i (v) closest to v, according to the measure *dist* using our privacy preserving k-NN operation.

3. Set the parents of v to the nodes in P_{l-1} (v, p). Each element v at level l has up to p parents associated with points in its distinct vicinity.
4. Create the child edges for the nodes of level $l - 1$, as follows:

(a) For each node u of level $l - 1$, determine the list of distinct nodes C (u) of level l that have chosen u as a parent.

(b) Use our Privacy Preserving k-NN to find the c closest points to u among those in C (u).

(c) Set the children of u to be these c nearest neighbors.

5. For each node v of level l, determine whether it was accepted as child of any node at level l − 1.

If yes, then the closest node that accepted it as a child becomes the guarantor $g(v)$ of v, and v becomes a dependent of $g(v)$. Otherwise, label v as an orphan node.

6. For each orphan node v at level l, a node at level $l - 1$ is needed to act as its guarantor. The node should be close as possible to v (in terms of the distance measure), and must be unencumbered; that is it must have fewer that the maximum allowed number c of children. Find a guarantor for v by successively doubling the size of the candidate parents set as f ollows:

(a) Set $i = 1$

(b) Compute $P_{l-1}(v, 2p)$ as in Step2.

(c) If Pl 1 $(v, 2^i p)$ has no unencumbered node, increment i and go to 6b.

(d) Otherwise, choose as the guarantor $g(v)$ the unencumbered node of $Pl-1$ $(v, 2 p)$ that is closest to v. Add v as a child and dependent of $g(v)$, and replace the parent of v furthest from v by $g(v)$.

This completes the construction of the privacy pre- serving *SASHl*. Note that the information shared by the parties is all the edges (parent/child relationships) and results of k-NN queries that identify only owners of vectors but do not reveal the data associated with those vectors. It may be necessary to demonstrate to all other parties that all the local data is involved in the process. The *SASH* does not partition the search space, but a *KD* -Tree or an *R*-Tree does. Our case study is *KD* -Trees but the conclusions apply to *R*- Trees, since in fact, a node in an *R*-Tree is a bounding box for all data below that node.

Comparison and Analysis

ven it is more efficient that each party compute the k-NN on their data and then merge them as in privacy preserving k-NN algorithm, the *SASH* allows the parties to monitor the participation of other parties. But if each party uses their own data structure to answer locally its k-NN query, in reality, one party may always keep away some different sections of their data. With a shared data structure, this is not possible. The data in the shared structure will be involved in the entire process of classification or clustering. Clearly, the methods are equivalent if one party decides from the beginning never to involve some set of records, but then, the results will not be accurate for the union of all records anyways.

Even in the ideal case, k-NN queries disclose some information about the data of the parties involved. And bounding boxes or bounding regions are found for some of the data owned by other parties. From the perspective of

Definition 2.1, the separate data structures and secure k-NN is the most secure option, but parties have less certainty that all others have contributed their entire data sets. The secure SASH provides more assurances that all parties have contributed their data (otherwise the shared SASH cannot be constructed). Partition data structures, like KD -Trees (or R-Trees) are the least private as they enable one party to immediately learn bounding boxes for the other's data. All these options incur a communication overhead. Considering the overhead for the transmission of all data to a trusted server which performs the desired analysis on the join data is insignificant.

Time Complexity Analysis

The complexity, depends on the data structures, that every party uses locally and the number of distances calculated for local k-NN, plus the communication cost for sending the overall result to all parties.

In terms of CPU-time overhead, the algorithms induce only a linear time overhead. In all dictionary data structures, when the data is high-dimensional, the CPU-time costs are mainly associated[1] with metric evaluation. The SASH data structure is very efficient in terms of CPU[2]-time costs. The approximate k-NN queries proceeds by choosing Pi (q, k) at every level[3] of the SASH , then combing them and choosing k closest to the query vector *q*. In the SASH there are two strategies for queries: uniform and geometric. Their paper suggests that the geometric pattern improves both accuracy and search time.

SASH construction: $pcn \log 2n$
Approx. *k*-NN query (uniform): $ck \log 2n$
Approx. *k*-NN query (g ~~eom.):~~ $\log 2n1+ 2p \log 2n$

Conclusion

Privacy preserving data mining emerged in response to two equally important and seemingly desperate needs: data analysis in order to deliver better services and ensuring the privacy rights of the data owners

Although the construction data structure is secure, its operation can disclose some private information. While these may seems un-satisfactory, the fact remains that the protocols and algorithms presented here are the most practical. They allow some level of protection at essentially affordable cost (the CPU-time is affected only by a constant factor). Other methods are essentially theoretical.

Several privacy preserving metrics are presented here and based on this, an algorithm to obtain *k*-NN with some level of preserving privacy. This provides some practical methods for applications in classification and clustering with considerations to privacy.

References

Rogress Pressman Software Engineering 5th Edition Pearson Education 2005.

Herbert Schildt The Complete Reference 7th Edition Tata McGraw-Hill 2007.

V. Estivill-Castro, L. Brankovic, and D. L. Dowe. Privacy in data mining. *Privacy – law & Policy Reporter,* 6(3):33-35, September 1999.

[4] R. Agrawal and R. Srikant, "Privacy-preserving data mining," in *Proceedings of the 2000 ACM SIGMOD Conference on Management of Data*. Dallas, TX: ACM, May 14-19 2000, pp. 439–450. [Online]. Available: http://doi.acm.org/10.1145/342009.335438.

[5] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally

partitioned data. In ACM SIGMOD *Workshop on Research Issues in Data Mining and Knowledge Discovery, Madison, Wisconsin,* USA, 2002.

[6] J. Han and M Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000, ISBN: 1-55860-489-8.

[7] D.T. Lee. On k-nearest neighbors voronoi diagrams in the plane. *IEEE Transactions of Computers*, C(31):478–487, 1982.