

A Privacy Preserving Probabilistic Neural Network for Horizontally Partitioned Databases

Jimmy Secretan, Michael Georgiopoulos and José Castro

Abstract—In this paper, we present a version of the Probabilistic Neural Network (PNN) that is capable of operating on a distributed database that is horizontally partitioned. It does so in a way that is privacy-preserving: that is, a test point can be evaluated by the algorithm without any party knowing the data owned by the other parties. We present an analysis of this algorithm from the standpoints of security and computational performance. Finally, we provide performance results of an implementation of this privacy preserving, distributed PNN algorithm.

I. INTRODUCTION AND MOTIVATION

THE Probabilistic Neural Network (PNN), introduced by Specht [1], is an effective neural network architecture, underpinned by a theoretically sound framework. It can effectively solve a variety of classification problems. The PNN is an approximation of the theoretically optimal classifier, known as the Bayesian optimal classifier (not to be confused with the Naïve Bayes classifier or Bayesian Belief Networks). However, to compute the Bayesian optimal classification, one must know the class conditional probabilities involved. In his seminal paper, Specht uses the available data to estimate these class conditional probabilities. This approximation is based on the work by Parzen, who in his 1965 paper [2], provided a formula for the calculation of these class conditional probabilities.

The PNN has been shown to be well suited for a variety of classification problems, and its regression counterpart, GRNN has been shown very effective on a variety of regression problems. In fact, in [3], the authors show that GRNN has the highest performance for detecting malignant breast cancers of all of the other algorithms evaluated. Specht's company, R2 Technology [4], currently is utilizing the PNN algorithm in a hospital machine for detecting breast cancer. Therefore, despite the availability of many other machine learning approaches to solve classification problems, such as the multi-layer perceptron neural networks, and support vector machines, amongst others, the PNN maintains its status as a highly accurate, well understood and theoretically sound machine learning algorithm. In this paper, we show that PNN, in a privacy preserving environment, offers many advantages. Besides its solid theory and success in practice, it is highly parallelizable, easy to implement, and in certain configurations, lends itself to a high performance privacy preserving implementation.

Jimmy Secretan and Michael Georgiopoulos are with the University of Central Florida, Orlando, FL 32816, USA (jimmy@thepublicgrid.org and michaelg@mail.ucf.edu). José Castro is with the Computing Research Center, Costa Rica Institute of Technology (phone 011 506 550 2407; fax: 011 506 552 6665, email: jose.r.castro@gmail.com)

There are a number of practical scenarios where a privacy preserving implementation of a data mining algorithm, such as the PNN, is desired. For instance, consider the scenario where several hospitals want to use the data from their combined databases to train PNN to detect future instances of breast cancer. The consortium trying to accomplish this task involves hospitals from several different regions, each of which has essentially the same information on different patients (horizontal partitioning of the database). The work presented in this paper will enable this consortium to use the PNN as a predictor of future breast cancer instances by utilizing the data from all the hospital sites, while at the same time preserving the privacy of the patient information for which each hospital cares.

The “training phase” of the PNN consists only of loading the training points. This lack of a training phase makes the PNN very well suited for on-line operation, when new data points are added to the training set. In PNN's performance phase, in order for one to predict the label of a datum (e.g., whether a patient has breast cancer or not), whose label is unknown, some form of distance of this datum needs to be calculated to every data-point belonging to the training set.

Privacy preserving data mining often tries to simulate a situation where all data could be sent to a trusted third party and mined on a single system by that third party. For some mining tasks, e.g., mining of health care and criminal justice data, the transmission of this data to another party may violate privacy laws like HIPAA, the civil rights of the accused, and trade secrets. This is where privacy preserving data mining becomes most useful. Privacy preserving data mining (PPDM) borrows various techniques from disciplines like secure multiparty computation (SMC) among others in order to mine the databases for useful conclusions, without disclosing too much information.

In this paper, we design PNN implementations for a privacy preserving environment. We show that our privacy preserving PNN (PP-PNN) is also efficient in its use of computational resources, as compared to the standard PNN algorithm. In section II, we review past work in PPDM. In section III, we discuss the PNN algorithm in general. In section IV, we emphasize the necessary modifications to make the PNN a privacy preserving, distributed algorithm. We discuss our experimental set-up (section V) and we present results (Section VI) that attest to the efficiency of the Privacy Preserving PNN (PP-PNN) algorithm, delineated in Section IV. Finally, in Section VII we summarize our work and offer some conclusive remarks.

II. LITERATURE REVIEW

Literature in the field of Privacy Preserving Data Mining (PPDM) concentrates on how to coherently combine and mine databases in such a way to preserve the privacy of the individual parties' data. How this is accomplished, and the extent to which it is accomplished differs in the various branches of PPDM. However, because our paper relies on a secure multiparty computation (SMC) approach, we only present research related to the SMC approach.

SMC has as one of its pillars Yao's work to solve the Millionaire's Problem [5]. The Millionaire's Problem is this: two millionaires wish to find who has more money, but neither wants to disclose his/her individual amount. Yao proved that there was a secure way to solve this problem by representing it as a circuit, sharing random portions of the outputs. Yao circuits, if necessary, can serve as a basic building block for all necessary SMC operations. However, using Yao circuits is typically very inefficient, giving rise to the necessity of more efficient and more specialized cryptographic techniques.

Since Yao's work, many SMC-based adaptations of data mining algorithms have been developed, typically addressing either horizontal or vertical partitioning of the data. These include a decision tree algorithm [6], Naïve Bayes [7], k-means clustering [8], Support Vector Machines [9], and K-nearest neighbor [10] as well as several others. To the best of our knowledge a privacy preserving PNN algorithm has not appeared in the literature yet.

III. THE PNN ALGORITHM

First we state both the model of our analysis and our assumptions. Then we present the PNN classifier in its simplest form. The reader is referred to the Appendix of the paper where the notation, used throughout this paper, is presented.

A. Distributed Model and Assumptions

For the PP-PNN algorithm, we assume that we have at least 3 parties involved. The parties are "semi-honest". That is, they do not engage in malicious communication or hacking to disrupt the network, but instead they try to use the information that they have and the information received from other parties to find out as much as possible about other parties' databases. The fact that there is no involved training process for the PNN is much to its advantage in a privacy preserving environment. There is a horizontal partitioning of the D -dimensional training data, meaning that the training points, \mathbf{X}_r in S are divided among the parties, in some way, such that each party P^k owns several D -dimensional instances of the training data, designated as S^k . A test point query can be issued by any of the participating parties. The party issuing the query is always referred to as a P^{k^*} . We assume that there are no missing values, and that merging the databases from all of the different nodes would yield a complete set of variables and records.

B. The PNN Algorithm

From Bayes' Theorem, we have that the a-posteriori probability that an observed datum \mathbf{x} has come from class c_j is given by the formula:

$$p(c_j|\mathbf{x}) = \frac{p(\mathbf{x}|c_j)p(c_j)}{p(\mathbf{x})}$$

In order to make effective use of this formula, we must calculate the a-priori probabilities for each class as well as the class conditional probabilities. The a-priori probability can be estimated directly from the training data, that is, $p(c_j) = \frac{PT_j}{\sum_j PT_j}$ where PT_j designates the number of points in the training data set that are of class j . These class conditional probabilities are calculated (see [2]), as follows:

$$\frac{1}{(2\pi)^{D/2} (\prod_{i=1}^D \sigma_i) PT_j} \sum_{r=1}^{PT_j} \exp\left(-\sum_{i=1}^D \frac{(x(i) - X_r^j(i))^2}{2(\sigma_i)^2}\right)$$

where D is the dimensionality of the input patterns (data), PT_j represents the number of training patterns belonging to class j , \mathbf{X}_r^j denotes the r -th such pattern, \mathbf{x} is the input pattern to be classified, and σ_i is the smoothing parameter along the i th dimension, used by the PNN classifier. The PNN algorithm identifies the input pattern as belonging to the class that maximizes the a-posteriori probability $p(c_j|\mathbf{x})$.

The choice of the σ parameter has an effect on PNN's classification accuracy. In this paper, we assume that σ parameters have been appropriately chosen, and we are only concerned in loading the training data into memory, prior to the initiation of the PNN's performance phase. Once the loading is complete, a set of class conditional probabilities (one for each class) is computed for each testing point. The label of the testing point is determined by the highest class a-posteriori probability. Figure 1 shows the pseudo-code of the PNN algorithm.

```

foreach  $c_j$  do
  foreach  $\mathbf{X}_r^j$  do
     $CCP_j =$ 
     $CCP_j + \exp\left(-\sum_{i=1}^D (\mathbf{x}(i) - \mathbf{X}_r^j(i))^2 / (2\sigma_i^2)\right)$ 
    ;
  foreach  $CCP_j$  do
     $CCP_j = CCP_j / ((2\pi)^{D/2} PT_j \prod_{i=1}^D \sigma_i)$  ;
   $C(\mathbf{x}) = \operatorname{argmax}_j \{CCP_j (PT_j / PT)\}$ ;

```

Fig. 1. Pseudo-code for the PNN algorithm (serial version)

IV. PRIVACY PRESERVING PNN ALGORITHM

A. Distributed PNN

In this paper, we consider two different algorithms, one that allows the test point query be public, and one that keeps the test point query private.

1) *Public Queries*: To begin with, we assume the existence of a function called *secureSum*, capable of summing together a matrix across many parties, without any individual party knowing the intermediate terms. Such an algorithm is presented in [11]. The secure sum works as follows: The first party, takes its matrix and adds to it a random matrix R , of the same size. All of the elements of R are taken from some field of numbers F , whose maximum value is designated as $|F|$. This needs to be chosen to be large enough to accommodate the group sum. The first party then passes this new matrix to the next party, and the next party adds (modulus of $|F|$) its matrix to the result. This process continues until every party has added its matrix to the matrix being circulated amongst the parties. Finally, the originating party receives the result, subtracts from it the random matrix that it has added, thus obtaining the final summed matrix result. This process does not disclose to any party the individual matrices that resulted in this summed matrix. This function, as used in the pseudo-code, takes two parameters. The first is the share that each party is contributing to the sum, and the second is where the final sum is to be stored.

When the query is public, we use an algorithm similar to the one presented in [12] where the parties calculate individual class conditional probabilities (CCPs) and sum them. The pseudocode, illustrating this in figure 2 is written in Single Program Multiple Data (SPMD) style, similar to an MPI program.

```

secureSum( $PT_j^k$ ,  $PT_j$ );
foreach  $c_j$  do
  foreach  $\mathbf{X}_r^{k,j}$  do
     $CCP_j^k = CCP_j^k +$ 
     $\exp\left(-\sum_{i=1}^D (\mathbf{x}(i) - \mathbf{X}_r^{k,j}(i))^2 / (2\sigma_i^2)\right)$ ;
secureSum( $CCP_j^k$ ,  $CCP_j$ );
if  $k == k^*$  then
  foreach  $CCP_j$  do
     $CCP_j = CCP_j / ((2\pi)^{D/2} PT_j \prod_{i=1}^D \sigma_i)$ ;
     $C(\mathbf{x}) = \operatorname{argmax}_j \{CCP_j (PT_j / PT)\}$ ;

```

Fig. 2. The PP-PNN algorithm for the public query case

There is relatively little communication required for this version of the PP-PNN algorithm. The first round of communication to determine the number of members in each class requires that a vector of size J needs to be communicated between all parties resulting in $O(JK)$ time. Thereafter, each party must find the squared difference between the components of all $\mathbf{X}_r^{k,j}$'s and the test point \mathbf{x} , take the exponential of them, and sum them together. This is similar to what needs to be done with the serial PNN algorithm, and therefore takes $O(D \max PT^k)$ time, as it can be done in parallel. Each party must use the secure matrix summation to add the partial class conditional probability vectors together.

Again, this requires $O(JK)$ operations. Finally, the last node must search through the CCP vector to find the class with the largest probability, needing $O(J)$ operations. Therefore, this PP-PNN version requires $O(JK + D \max PT^k)$ operations. In this case, the PP-PNN algorithm takes advantage of the parallelism allowed by the distributed data.

2) *Private Queries*: To keep the query private, the party that is performing the query must interact with each party, computing the distance between \mathbf{x} and each $\mathbf{X}_r^{k,j}$, but doing so in a way that does not reveal the distance to either party. Revealing the full distance could easily allow one of the parties to solve for the query point or allow the querying party to solve for some of the other data points. Therefore, we utilize a commonly used technique in SMC, where the result is shared randomly between the parties. If we can randomly share the result between the parties, each party can then calculate its own CCP^k , and securely sum then together in a way that is similar to how the public query version of the algorithm operates.

To securely find the distance between \mathbf{x} and each $\mathbf{X}_r^{k,j}$, we use as our primary component the secure scalar product protocol from [13]. This protocol is built around the additively homomorphic cryptosystem. By definition, an encryption algorithm is additively homomorphic if the following holds true:

$$Enc(A) * Enc(B) = Enc(A + B)$$

where *Enc* is the encryption function of the cryptosystem. Given this, it is possible for two parties to participate in computation together without compromising their operands. For our application, we use the cryptosystem provided by Paillier [14].

We rewrite the algorithm from [13] here, assuming that, two parties, P^A and P^B each have the vectors \mathbf{x}^A and \mathbf{x}^B respectively. We present the secure scalar product (SSP) algorithm in figure 3.

```

Input: Vectors  $\mathbf{x}^A$  and  $\mathbf{x}^B$  on parties  $P^A$  and  $P^B$  respectively.
Output: Random shares  $s^A$  and  $s^B$  on parties  $P^A$  and  $P^B$  respectively.
if  $k == A$  then
   $(sk, pk) = \text{GenerateKeyPair}()$ ;
  send( $P^B, pk$ );
  foreach  $i \in (1, \dots, D)$  do
     $\mathbf{cx}(i) = Enc_{pk}(\mathbf{x}^A(i))$ ;
  send( $P^B, \mathbf{cx}$ );
   $s^A = Dec_{sk}(\text{receive}(P^B))$ ;
if  $k == B$  then
   $\mathbf{cx} = \text{receive}(P^A)$ ;
  send( $P^A, \prod_{i=1}^D \mathbf{cx}(i)^{\mathbf{x}^B(i)} \cdot Enc_{pk}(-s^B)$ )

```

Fig. 3. Secure Scalar Product (SSP) algorithm.

P^A generates a public/private key pair, encrypts all the elements of \mathbf{x}^A and sends them to P^B . P^B puts each

encrypted element of $\mathbf{c}\mathbf{x}(i)$ to the power of the corresponding element of $\mathbf{x}^B(i)$, and then takes the product of all of these numbers. Because of the properties of the homomorphic encryption system, this is equivalent to multiplying the corresponding elements together and adding them. However, while P^B performs this computation, it cannot decrypt this value, because only P^A has the private key. Despite this, P^B does subtract a random share s^B , which only P^B knows. He sends the remainder back to P^A where P^A can only decrypt the value of the scalar product minus the random share that P^B owns. Therefore, neither party knows the full value of the scalar product, but each party has a random share of it.

To more easily look at the private query, PP-PNN algorithm in terms of the secure scalar product, let us first define normalized versions of each vector.

$$\mathbf{x}'(i) = \frac{\mathbf{x}(i)}{\sqrt{2\sigma_i}} \forall i = 1 \dots D, \text{ and}$$

$$\mathbf{X}_r'^j = \frac{\mathbf{X}_r^j(i)}{\sqrt{2\sigma_i}} \forall i = 1 \dots D.$$

Now, we can rewrite the class conditional probability equation as:

$$p(\mathbf{x}|c_j) = \frac{1}{(2\pi)^{D/2} \left(\prod_{i=1}^D \sigma_i \right) PT_j} \sum_{r=1}^{PT_j} \exp(-dis(\mathbf{x}', \mathbf{X}_r'^j))$$

where $dis(\mathbf{x}', \mathbf{X}_r'^j)$ gives the Euclidean distance between the query (testing point) and the corresponding training point. At this point, we can write the distance in terms a scalar product and two summations, as follows:

$$dis(\mathbf{x}', \mathbf{X}_r'^j) = \sum_{i=1}^D (\mathbf{x}'(i))^2 + \sum_{i=1}^D (\mathbf{X}_r'^j(i))^2 - 2\mathbf{x}' \cdot \mathbf{X}_r'^j$$

As pointed out previously in [15], a secure scalar product can be utilized for this problem.

Now, the private query algorithm begins with a secure sum, again used to calculate the number of training points in each class. This is shared among all of the parties. P^{k^*} then engages in a scalar product calculation between the query (testing) point \mathbf{x}' and every training data-point $\mathbf{X}_r'^{k,j}$ residing at party P^k . To complete the distance calculation, the summation of the squares of the components are added along with the scalar product shares. Each party P^k now owns a random share of the distance between each of its training points $\mathbf{X}_r'^{k,j}$ and the test point \mathbf{x}' . Each party P^k adds some spurious random distance shares to its distance shares, so that it has exactly PT_j shares in each class. This conceals the exact number of training points of each class that a party P^k owns. Each party P^k keeps track of which shares are valid and which are spurious. All parties then perform the exponential of their respective shares. Finally, a scalar product is performed on the exponential of these shares, sharing the results of this between P^{k^*} and each P^k . This scalar product skips operating on the spurious shares. The secure summation of all of these will then yield the CCP vector to produce the final result. Finally, at P^{k^*} , the class conditional probabilities are divided by the appropriate factors and a label is found through linear search.

The algorithm is presented in figure 4. In this pseudo-code, the *SSP* (secure scalar product) is meant to be called by each party, with the two parameters being the vector owned by P^A and P^B respectively. On each party, it will return the respective random shares s^A and s^B .

```

secureSum( $PT_j^k, PT_j$ );
if  $k == k^*$  then
  foreach  $P^k$  do
    foreach  $c_j$  do
      foreach  $\mathbf{X}_r'^{k,j}$  do
         $sA_r =$ 
         $\exp(2SSP(\mathbf{x}', \mathbf{X}_r'^{k,j}) - \sum_{i=1}^D (\mathbf{x}'(i))^2);$ 
         $CCP_j^k = CCP_j^k + SSP(sA, sB);$ 
      else
        foreach  $c_j$  do
          foreach  $\mathbf{X}_r'^{k,j}$  do
             $sB_r =$ 
             $\exp(2SSP(\mathbf{x}', \mathbf{X}_r'^{k,j}) - \sum_{i=1}^D (\mathbf{X}_r'^{k,j}(i))^2);$ 
             $CCP_j^k = CCP_j^k + SSP(sA, sB);$ 
         $secureSum(CCP_j^k, CCP_j);$ 
        if  $k == k^*$  then
          foreach  $CCP_j^k$  do
             $CCP_j = (CCP_j^k) \frac{1}{(2\pi)^{D/2} \prod_{i=1}^D \sigma_i PT_j};$ 
           $C(\mathbf{x}) = \operatorname{argmax}_j \{CCP_j (PT_j/PT)\};$ 

```

Fig. 4. The PP-PNN Algorithm for the private query case

The performance of this algorithm is inferior compared to the public query case. The additional time needed by this version of the PP-PNN algorithm is mostly needed to compute the secure scalar products. Because each party P^k adds spurious values to the vectors it returns, each party communicates with P^{k^*} , $O(PT)$ times, yielding $O(K(PT))$ operations for this step. Each P^k must also calculate the summation of the components of each of its $\mathbf{X}_r'^{k,j}$. However, these calculations can proceed in parallel, so this adds $O(D \max PT^k)$ operations to the process. A scalar product must be computed between every party in order to get the CCP shares, thus requiring $O(K(PT))$ operations. Finally, these must be summed together, taking $O(JK)$. Consequently, the final complexity of a private query PP-PNN algorithm ends up being equal to $O(K(PT) + D \max PT^k + JK)$.

3) *Security Analysis:* The PP-PNN versions discussed above reveal the class conditional probabilities for each class. We could use a secure circuit protocol in order to obscure these class conditional probabilities. However, there are certain applications where it is useful to know the CCPs for each class. The PP-PNN versions also reveal the number of training points in each class, but as this is an aggregate statistic, it is acceptable. We now analyze the data received and sent by each of the parties.

First, we consider the public query case. Here, we have

two different types of parties involved in the computation. We separately consider party P^{k^*} , which originates the query, and P^k for $k = 1 \dots K, k \neq k^*$. We intend to show that in both cases, the parties discover only limited information which they could not obtain from their own data and the final model. Because the secure sum has been proven to be secure with 3 or more parties, the secure addition of the PT vector will not reveal any party’s individual class frequencies, but will reveal the final class frequency vector. Each party then calculates its partial CCP vector based on that query and its own data points, which requires no further communication and therefore exposes no additional information. Again, a secure sum adds the individual CCP vectors, only exposing the final CCP, which is the result of our computation. The final portion takes place only on P^{k^*} , so it can therefore expose no additional information. By the composition theorem [16] *if all components of the protocol are secure then the protocol itself is secure*. Therefore, this algorithm is secure.

Now, we show the security of the private query version as well. In this case, we also have the same secure sum to find the PT vector and this is provably secure. Furthermore, in this case, a secure scalar product, proved secure in [13], is performed between P^{k^*} and every other party P^k . Because each has random shares, nothing of value is revealed. Taking the exponentials of these shares is a completely local computation and cannot expose any information. Another secure scalar product takes place between the exponentials of these shares. Finally, the same secure sum, as before, is used to add together the class conditional probabilities. All of the aforementioned algorithms are secure, used within their constraints. Eventually, processing completely local to P^{k^*} is performed to compute the label, revealing no additional information. Once more, because all portions of the algorithm are secure, the entire algorithm is secure.

V. EXPERIMENTAL SETUP

A. Implementation

In order to simplify the evaluation of these algorithms, a C++ implementation with the MPICH [17] implementation of the MPI standard for communication was used. This approach worked well because our distributed data mining environment is simulated on a locally connected cluster. To implement the Paillier encryption scheme, the Number Theory Library (NTL) [18] was used along with code adapted from [19]. The cluster of servers consisted of nodes with Opteron processors connected by gigabit Ethernet. While this is certainly faster than will be encountered in most WAN connections, Internet connections approaching these speeds are not unheard of for large organizations.

B. Experimental Database

Because our objective is not to assess the classification accuracy of the PNN, but to instead evaluate the scaling and efficiency of the PP-PNN algorithms, we use an artificially generated database for our experiments. An artificial database also allows us to generate data of arbitrary size,

and dimensionality so as to evaluate the scale-up of the PP-PNN algorithms. The artificially generated database consists of a 16-dimensional, 2-class Gaussian functions, with a 15% overlap.

C. Experimental Description

Our privacy preserving algorithm preserves the integrity of the serial PNN algorithm. Because of lack of space, we save the proof for future work. Therefore, for the experiments with PP-PNN, the only concern is performance and scaling. We vary the number of parties from 3 to 8, giving each party 8,000 training points. Both the private and public query algorithms are evaluated for the time that it takes for them to execute 5 test point queries and an average is taken. A nominal σ value of 0.5 over all dimensions and classes was chosen, because it worked well in the initial experimentation with the Gaussian dataset. Because this paper only focuses on the computational efficiency of the PP-PNN algorithms and not their classification performance, no effort was expended to optimize the sigma parameter values for the PNN.

VI. RESULTS

Here we present a table showing the average time to label a test point query x . The number of parties is varied from 3 to 8. Each party has a constant (8,000) number of training points, thereby varying the total number of training points from 8,000 to 64,000.

TABLE I
AVERAGE TIME FOR TEST POINT QUERY

K	Public Query	Private Query
3	0.004s	02min 31s
4	0.004s	05min 04s
5	0.004s	08min 38s
6	0.004s	13min 03s
7	0.004s	18min 04s
8	0.004s	23min 02s

VII. CONCLUSIONS AND FUTURE WORK

While both PP-PNN algorithms (public-query and private-query) remain practical, clearly the public query algorithm has a performance advantage over the private query one. The public query case maximizes parallelism for the PNN, while minimizing network traffic. In addition, much of the private query algorithm’s time is spent doing the expensive encryptions and decryptions that are involved in the secure scalar product. The public query algorithm could easily scale up to more parties, all the while keeping efficient queries.

Future work will consider implementing intra-party parallelism. Especially in the public query case, the distances between the query and the training points can be calculated by parallel machines at each party site, thus rapidly accelerating the queries and making this suitable for a production environment. In the private query case, further effort will be made to optimize the software from a cryptographic standpoint. There is also the possibility of using specialized hardware to increase the speed of cryptographic computations.

Here we summarize relevant notation from the paper.

TABLE II
NOTATIONS FOR ANALYSIS

Notation	Description
S	The set of training data for the classification problem.
D	The dimensionality of data points in the classification problem, agreed upon by all parties.
PT	The number of points in the training set S .
c_j	The j th class of the classification problem of interest; $j = 1 \dots J$.
J	The number of classes in the classification problem.
\mathbf{X}_r^j	The r th training point from S that is of class c_j .
\mathbf{x}	A D dimensional test point in the classification problem of interest.
$\mathbf{x}(i)$	The i th component of the training point \mathbf{x} ; $i = 1 \dots D$
K	The number of parties in the distributed environment.
P^k	The k th party of the distributed environment.
P^{k*}	The party that generates the query requesting the prediction of the class of test point \mathbf{x} ; k^* can be any one of the indices from the set $\{1, 2, \dots, K\}$
S^k	The portion of the training set S owned by party P^k in the distributed environment, $k = 1 \dots K$
PT_j^k	The number of points in training sub-set set S^k of class c_j .
$\mathbf{X}_r^{k,j}$	The r th training point from the set S^k that is part of class c_j .
$\mathbf{X}_r^j(i)$	The i th component of the training point \mathbf{X}_r^j ; $i = 1 \dots D$
$C(\mathbf{x})$	The class that PNN algorithm predicts for the test point \mathbf{x} .
$p(c_j \mathbf{x})$	The a-posteriori probability that an observed point \mathbf{x} comes from class c_j
$p(\mathbf{x} c_j) = CCP_j$	The class conditional probability that an observed point \mathbf{x} is of class c_j
CCP_j^k	The class conditional probability of test pattern \mathbf{x} , given that it has come from class c_j , which can be computed at party P^k .
CCP_j	The class conditional probability of test pattern \mathbf{x} , given that it has come from class c_j ; $CCP_j = \sum_{k=1}^K CCP_j^k$.
$p(c_j)$	The a-priori probability that a datum in the classification problem is of class c_j .
$\sigma = (\sigma_1, \sigma_2, \dots, \sigma_D)$	A vector of parameter values used by the PNN algorithm to estimate the class conditional probabilities for the classification problem of interest.
sk	A private (secret) cryptographic key.
pk	A public cryptographic key.
Dec_{sk}	A decryption function utilizing the private key.
Enc_{pk}	An encryption function utilizing the public key.
\mathbf{cx}	Vector of encrypted text.
$secureSum$	The function for performing the secure summation across 3 or more parties.
SSP	The function which calculates a secure scalar product between two parties, shared randomly between the two calling parties. The function takes two arguments, the vectors \mathbf{x}^A , and \mathbf{x}^B , belonging to parties P^A and P^B , respectively, whose scalar product is to be calculated.
sA, sB	These designate random distance shares generated by the secure scalar product. In the private query algorithm, they represent vectors of such shares.

Jimmy Secretan acknowledges the support of an NSF Graduate Research Fellowship. This work was supported in part by the NSF grants CRCD: 0203446, CCLI: 0341601, DUE: 05254209, and IIS: 0647120. The authors would also like to acknowledge Advanced Micro Devices, Inc. (AMD) for their generous donation of equipment used in this research.

REFERENCES

- [1] D. F. Specht, "Probabilistic neural networks," *Neural Networks*, vol. 3, pp. 109–118, 1990.
- [2] E. Parzen, "On estimation of probability density function and mode," *Annals of Mathematical Statistics*, vol. 33, pp. 1065–1073, 1962.
- [3] T. Kiyani and T. Yildirim, "Breast cancer diagnosis using statistical neural networks," in *International XII. Turkish Symposium on Artificial Intelligence and Neural Networks – TAINN 2003*, 2003.
- [4] "R2 home," <http://www.r2tech.com>, 2007.
- [5] A. Yao, "How to generate and exchange secrets," in *Proc. 27th Annual Symposium on Foundations of Computer Science*, 1986, pp. 162–167.
- [6] W. Du and Z. Zhan, "Building decision tree classifier on private data," in *IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining*, vol. 14, Maebashi City, Japan, December 2002, pp. 1–8.
- [7] J. Vaidya and C. Clifton, "Privacy preserving naive bayes classifier for vertically partitioned data," in *2004 SIAM International Conference on Data Mining*, Orlando, FL, May 2004.
- [8] —, "Privacy-Preserving K-Means Clustering over Vertically Partitioned Data," in *The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, DC, August 2003. [Online]. Available: <http://www.cs.purdue.edu/homes/jsvaidya/pub-papers/vaidya-kmeans.pdf>
- [9] H. Yu, J. Vaidya, and X. Jiang, "Privacy-preserving svm classification on vertically partitioned data," in *Proceedings of PAKDD '06*, ser. Lecture Notes in Computer Science, vol. 3918. Springer-Verlag, jan 2006, pp. 647–656.
- [10] L. Xiong, S. Chitti, and L. Liu, "k nearest neighbor classification across multiple private databases," in *Proceedings of the ACM Fifteenth Conference on Information and Knowledge Management*, 5-11 November 2006.
- [11] H. Yu and J. Vaidya, "Secure matrix addition," University of Iowa, Tech. Rep. UIOWA-CS-04-04, 2004. [Online]. Available: <http://hwanjoyu.org/paper/techreport04-04.pdf>
- [12] J. Secretan, M. Georgiopoulos, I. Maidhof, P. Shibly, and J. Hecker, "Methods for parallelizing the probabilistic neural network on a beowulf cluster computer," in *World Congress on Computational Intelligence*, Vancouver, B.C., Canada, 2006.
- [13] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikäinen, "On private scalar product computation for privacy-preserving data mining," in *Information Security and Cryptology - ICISC 2004, 7th International Conference, Seoul, Korea, December 2-3, 2004, Revised Selected Papers*, ser. Lecture Notes in Computer Science, C. Park and S. Chee, Eds., vol. 3506. Springer, 2004, pp. 104–120.
- [14] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology – EUROCRYPT '99*, ser. Lecture Notes in Computer Science, J. Stern, Ed., vol. 1592. Springer-Verlag, 2-6 May 1999, pp. 223–238.
- [15] W. Haiping and H. Huacan, "Tools for privacy preserving kernel methods in data mining," in *AIA'06: Proceedings of the 24th IASTED international conference on Artificial intelligence and applications*. Anaheim, CA, USA: ACTA Press, 2006, pp. 388–391.
- [16] O. Goldreich, "Secure multi-party computation," (Sep. 1998) Working Draft.
- [17] "Mpich2 home page," <http://www-unix.mcs.anl.gov/mpi/mpich2/>, 2007.
- [18] "Ntl: A library for doing number theory," <http://www.shoup.net/ntl/>, 2007.
- [19] K. Liu, "Paillier java code," <http://www.csee.umbc.edu/~kunliu1/research/Paillier.java>, 2007.