

Privacy Preserving Probabilistic Neural Networks for Vertically and Horizontally Partitioned Training Sets

September 22, 2009

Abstract

The Probabilistic Neural Network (PNN) is a popular and powerful neural network for mining data. This paper discusses a family of PNN algorithms that are capable of mining distributed databases that are either horizontally or vertically partitioned. It does so in a way that is privacy-preserving: that is, a query can be made to the algorithm without revealing a party's training data to any other party. These algorithms will allow the PNN to be evaluated on data sets that span multiple, geographically distributed organizations, while at the same time respecting the privacy of the records. The performance/security tradeoffs of the different privacy preserving PNN implementations are discussed, and detailed results are provided for the algorithms, evaluated over large (128K point) data sets.

1 Introduction and Motivation

The Probabilistic Neural Network (PNN) (Specht, 1990) is an effective, theoretically sound neural network architecture. It can solve a variety of classification problems by approximating the Bayes optimal classifier, the theoretically optimal classifier which minimizes the expected misclassification rate. However, to design the Bayesian classifier, one must know the class conditional probabilities involved. In the PNN, (Specht, 1990) the available data is used to estimate these class conditional probabilities.

The PNN has been shown to be well suited for a variety of classification problems, and its regression counterpart GRNN (Specht, 1991), performs well for many regression problems. PNN can perform comparably to, and often better than many other classifiers, such as backpropagation-trained neural networks (Specht and Shapiro, 1991). The PNN also remains well matched to popular modern architectures such as Support Vector Machines (SVM), often only sacrificing a minor amount of classification performance for orders of magnitude faster training time (Zhong et al., 2007). PNN and its variants have performed well in an impressive array of applications, including detecting various cancers (Kiyani and Yildirim, 2003; Lo et al., 1999; Gorunescu et al., 2005) and other bioinformatics problems (Georgiou et al., 2006), predicting concrete defects (Tam et al., 2004) and classifying power quality disturbances (Hu et al., 2007).

Because the class conditional probability density function (PDF) esti-

mator that underpins the PNN asymptotically approaches the true PDF as the number of patterns increases (Specht, 1990; Parzen, 1962), the PNN is ideally suited to take advantage of the tremendous amount of data that can be made available when combining the data sets of multiple organizations. While the testing phase of the PNN can be computationally expensive, there are several methods for speeding it up, including clustering techniques (Specht, 1992, 2006).

Considering the wide-ranging utility of the PNN, there are a number of practical scenarios where a distributed implementation of a PNN is desired. For instance, consider several hospitals who want to combine their databases to train a PNN classifier that detects breast cancer. The task involves hospitals from several different regions, each one of which has essentially the same information on different patients (a horizontal partitioning). Hospitals could also seek to merge databases from a shared set of patients, along with the data owned by dietitians, medical testing centers, and research hospitals (a vertical partitioning).

However, merging databases presents a significant challenge: sharing these databases may violate the privacy of patients, as well as privacy laws like HIPAA (the Health Insurance Portability and Accountability Act). 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 data from geographically distributed databases for useful conclusions, while

limiting information disclosure.

This paper offers the following contributions. This is the first privacy preserving Probabilistic Neural Network presented in the literature. The paper presents a comprehensive family of four conceptually similar privacy preserving algorithms for the PNN, which are practical to implement and can be immediately useful in distributed data mining applications. The algorithms are analyzed within a framework of privacy/performance tradeoffs, and this analytical methodology can be applied to other PPDM algorithms. To support the algorithms, several techniques are developed (for instance, for large scale secure sums) which can be used to make other PPDM algorithms more efficient. Finally the paper presents performance results for implementations of the algorithms and gives practical insight into PPDM algorithm development.

The organization of the paper is as follows: section 2 reviews past work in PPDM. Section 3 discusses the specifics of the PNN algorithm. Section 4 emphasizes the necessary modifications to make the PNN algorithm a privacy preserving algorithm. The experimental setup is described (section 5) and then results are presented (section 6) which demonstrate the practicality of using the four implementations of the Privacy Preserving PNN (PP-PNN) algorithm. Finally, section 7 offers conclusions and future directions.

2 Literature Review

Literature in the field of Privacy Preserving Data Mining (PPDM) concentrates on how to coherently combine and mine databases so as 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. There are data perturbation methods (e.g. (Agrawal and Srikant, 2000)), which focus on obfuscating the original data using random perturbations, done in such a way that the original distributions of the data can be easily recovered. A very different approach uses Secure Multi-party Computation (SMC) to compute the data mining functions, often in a more exact, more private way (at the expense of efficiency). Because this paper focuses on efficient implementations of SMC principles, only this area is reviewed in the interest of brevity.

2.1 Secure Multi-party Computation Techniques (SMC)

Secure Multiparty Computation (SMC) employs cryptographic techniques to ensure almost optimal privacy. SMC as a field grew out of work to solve the Millionaire's Problem (Yao, 1986): two millionaires wish to find who is richer, but neither wants to disclose his/her individual amount. The problem can securely solved by representing it as a circuit which shares random portions of the outputs. In fact, any function can be securely computed by these Yao circuits (Goldreich et al., 1987). However, representing the problem as a

circuit can be inefficient, because the circuit is typically large, especially for complex data mining algorithms. This gives rise to more specific solutions using more efficient cryptographic techniques.

One of the earliest and most basic cryptographic operations used in PPDM is the secure sum (Schneier, 1995). Suppose that parties P^1 to P^K each have a value v^k , and wish to compute the sum $v = \sum_{k=1}^K v^k$ which is limited to the range $[0...F]$. Without loss of generality, P^1 chooses a random number R from within the range $[0...F]$. P^1 then calculates $(v^1 + R) \bmod F$ and sends this to P^2 . The modulus by F is necessary to ensure that the sum is kept randomly distributed within the range $[0...F]$, therefore yielding no additional information to any observer. Parties P^2 through P^K repeat the calculations of P^1 . Finally, P^K sends its sum back to P^1 . P^1 then adds $-R$ to the sum and takes the $\bmod F$ of the sum, yielding $v = \sum_{k=1}^K v^k$. Proof of this algorithm's security can be found in (Schneier, 1995). However, to use real numbers instead of integers requires mapping them to a fixed-point notation. An illustration of how the secure sum is computed is shown in figure 1.

A technique which provides a building block for many SMC algorithms is the set of additively homomorphic encryption algorithms. By definition, a cryptosystem is additively homomorphic if the following holds true:

$$Enc(A) \otimes Enc(B) = Enc(A + B)$$

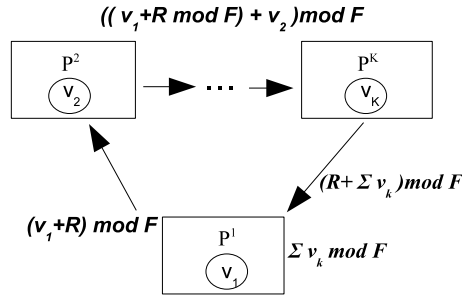


Figure 1: Diagram of the secure sum for K parties.

where Enc is the encryption function of the cryptosystem which also defines an operation \otimes . Given this operation, it is possible for two parties to participate in computation together without compromising their operands. The Paillier cryptosystem (Paillier, 1999) used in this work is a public key cryptosystem: a party can encrypt information using a public key, but a party needs a private key in order to decrypt information. One party can encrypt its data, and then have another party process the data without being able to decrypt it.

The Paillier cryptosystem's homomorphic operation is implemented in terms of a modular multiplication. Multiplying the ciphertext by itself a certain number of times is equivalent to adding it the same number of times giving the encrypted product, $Enc(A)^B = Enc(AB)$ (Paillier, 1999). These relations are the basis of the secure scalar product in (Goethals et al., 2004). In the algorithm, one party encrypts its vector, and sends it to the second party, which can then multiply by its vector using the homomorphic power relation. These element-by-element products are then added by the second

party while still homomorphically encrypted. Finally the result is sent back to the first party and decrypted.

The Privacy Preserving PNN presented in this work utilizes several of these techniques to make an efficient algorithm while at the same time maintaining almost ideal privacy. This work also extends several of these techniques (secure sums and secure scalar products), in ways that can likely be used in future PPDM algorithms.

2.2 Design of SMC-based Privacy Preserving Data Mining Algorithms

SMC-PPDM methods often involve the redesign of existing algorithms using many of the SMC techniques discussed in the previous section. The partitioning, that is, how the input data is distributed among the parties, will largely restrict the design and efficiency of the potential algorithm. When records are horizontally partitioned, parties have different sets of complete records (database rows). When the records are vertically partitioned, the parties have different variables (database columns) of the records.

Many machine learning algorithms have been recast into privacy preserving versions, including decision trees (Pinkas, 2002; Emekci et al., 2007), the Naïve Bayes classifier (Kantarcoglu and Vaidya, 2003), k-Means Clustering (Vaidya and Clifton, 2003b), Support Vector Machines (Yu et al., 2006), k-Nearest Neighbor (Zhan et al., 2005), and Association Rule Mining (Vaidya

and Clifton, 2002; Rozenberg and Gudes, 2006). A technique exists for computing a privacy preserving neural network (Barni et al., 2006); however, the authors only separate the network and input data, and do not train with data from multiple parties.

The Privacy-Preserving PNN will contribute to the growing body of PPDM algorithms. Adapting the PNN, a data-intensive algorithm, into a privacy preserving environment will provide practical techniques for the development of future, data-intensive, privacy preserving algorithms.

3 The PNN Algorithm

The PNN approximates the Bayesian Optimal Classifier. From Bayes' Theorem, the a-posteriori probability that an observed datum \mathbf{x} has come from class c_j is given by:

$$p(c_j|\mathbf{x}) = \frac{p(\mathbf{x}|c_j)p(c_j)}{p(\mathbf{x})} \quad (1)$$

The Bayes classifier chooses as the class that datum \mathbf{x} would have come from the class that maximizes this a-posteriori probability (this choice minimizes the misclassification error, resulting in a classifier that is optimal in the sense of minimizing this error). In order to make effective use of Bayes' formula, we must calculate the a-priori probabilities as the probability density functions of the datum \mathbf{x} , given that it comes from class c_j (i.e., $p(\mathbf{x}|c_j)$). 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 c_j . The conditional probability density functions ($p(\mathbf{x}|c_j)$) are calculated (see (Parzen, 1962)), as follows:

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

where D is the dimensionality of the input patterns (data), PT_j represents the number of training patterns belonging to class c_j , \mathbf{X}_r^j denotes the r -th such pattern, \mathbf{x} is the input pattern to be classified, and σ_i is a smoothing parameter along the i th dimension. The PNN identifies the input pattern as belonging to the class that maximizes the a-posteriori probability $p(c_j|\mathbf{x})$. It is assumed in the PNN that all dimensions in training and testing data are normalized to the range of $[0, 1]$. Figure 2 shows the neural network conceptualization of the PNN.

The pseudo-code is relatively easy to implement. Beginning on line 1, the PNN iterates through every unique class c_j . It then loops through each member of the training set \mathbf{X}_r^j which is in the current class c_j (line 2). On line 3, it calculates sum of the exponential of the distance between the testing point and the current training point. This sum is a partial calculation of the class conditional probability density function (CCPDF). In lines 4–5, it loops through and normalizes these values to accurately correspond to the conditional probability density functions, approximated by equation 2. Finally, on line 6, the class with the greatest class conditional probability

(CCP) as calculated by the CCPDF is selected as the class label for \mathbf{x} .

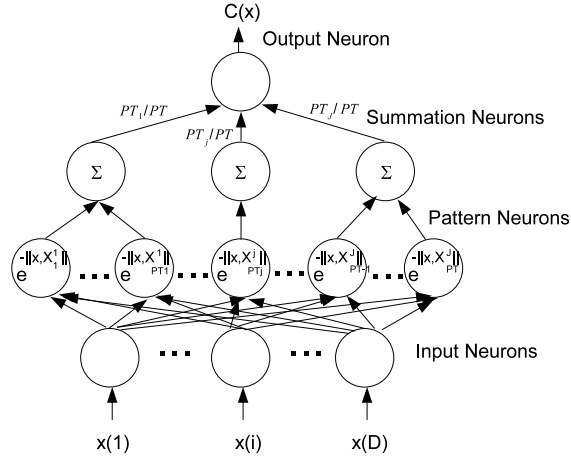


Figure 2: General architecture of the Probabilistic Neural Network

```

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

```

Figure 3: Pseudo-code for the PNN algorithm (serial version)

4 Privacy Preserving Distributed PNN Algorithm

This paper distinguishes four cases of a privacy-preserving distributed PNN, with distinction on both the type of partitioning (horizontal and vertical) and the privacy of the query. The four cases are: horizontal partitioning with a public query, horizontal partitioning with a private query, vertical partitioning with a public query, and vertical partitioning with a private query. While much of the literature does not distinguish on privacy of the query, with some algorithms having public and others private queries, having a public query can save significant computation. In the example of medical data mining, one could imagine that a private query could be used for an actual diagnosis, where the query data would need to remain secret. In contrast, one could use a public query to pose a hypothetical case to the system where there is no risk in exposing the query data. First both the model of the analysis and some assumptions are stated. Then the four related algorithms are described in turn.

4.1 Distributed Model and Assumptions

For the PP-PNN algorithm, it is assumed that at least three parties are involved. The algorithms can be empowered to work in a two-party setting with the addition of the commodity server described in (Du and Zhan, 2002). The parties are “semi-honest”. That is, they do not engage in malicious

communication or hacking to disrupt the network, but instead try to use information received from other parties to find out as much as possible about other parties' databases. Resilience to malicious adversaries is deferred to future work, but it should be noted that a protocol that is resistant to semi-honest adversaries can be made to resist malicious adversaries, albeit at the expense of significant computation and communication (Goldreich et al., 1987).

A test point query can be issued by any of the participating parties, referred to as a P^q . In the case of horizontal partitioning, the D -dimensional training data, \mathbf{X}_r in S , are divided among the parties such that each party P^k owns several D -dimensional instances of the training data, with that set designated as \mathbf{S}^k .

In the vertically partitioned case, each party P^k owns one or more variables for all of the available records, with this set denoted by \mathbf{D}^k . The class labels are also assumed to be private, with the party owning them designated as P^c . While in much of the literature, class variables are assumed to be public, this may be a poor assumption. In a medical application, the class label could be whether or not the patients have cancer: this information will certainly be restricted. It is assumed that there are no missing values, and that merging the databases from all of the different parties would yield a complete set of variables and records.

4.2 Horizontally Partitioned Databases, Public Queries

The simplest privacy-preserving PNN algorithm is the case where all data is horizontally partitioned and the query is public. In this algorithm, all of the parties calculate portions of the final set of class-conditional probabilities, which they add by secure sum. This results in very minimal communication and also allows for significant parallelism within each party.

We assume the existence of a function called *secureSum*, which takes three parameters: the party at which the secure sum will begin and end, the share that each party is contributing to the sum, and the variable where the final sum is to be stored.

The pseudocode, shown in figure 4, is written in Single Program Multiple Data (SPMD) style, similar to an MPI program. First the algorithm

```

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

```

Figure 4: The horizontally partitioned, public-query PP-PNN algorithm

calculates a secure sum to determine the number of points belonging to a

particular class; we assume that this final result is a piece of public knowledge (line 2). In lines 3–4, at every part, a partial conditional (on every class) probability density function value is calculated. These intermediate sums are, in turn, securely summed across parties on line 5, with the final sum ending up at P^q . Lines 6–9 are executed at P^q , and are identical to the calculations performed for the serial algorithm.

There is relatively minimal communication required for this version of the PP-PNN algorithm. This PP-PNN version requires $O(JK + D \max PT^k)$ time for computation and $O(JK)$ for communication, compared to $O(J + D(PT))$ operations for standard serial PNN. This PP-PNN algorithm takes advantage of the parallelism allowed by the distributed data.

For this algorithm to be secure, intermediate results obtained by any party must meet the SMC definition of security. By the SMC definition of security, no party must be able to glean any information that has not either been declared public or that it could not recover by using its own data and the result of the computation.

Theorem 1. *The horizontally partitioned, public query PNN algorithm given in figure 4 is secure by the SMC definition of security.*

Proof. There are only two portions of the algorithm which require communication among parties, and therefore only two opportunities for privacy to be breached (lines 2 and 5). Because the secure sum has been proven to be secure with three or more parties (Schneier, 1995), the secure addition of the PT vector in line 2 will not reveal anything other than the overall vector of

class frequencies (PT_j), which has been declared public. While this too can be kept private, it is typically an acceptable quantity to reveal, and revealing it allows the algorithms to be simplified.

The secure sum on line 5 only exposes the unnormalized CCP to P^q . The CCP that ends up at P^q is an intermediate value to the computation, not the final value. However, the final value of the computation is the CCP vector, divided by the factor in line 8. Because party P^q could glean this information from the final result along with its own information, nothing is unduly revealed under the SMC definition of security. All other parties have only calculated their own partial CCP vectors, or have been party to a secure sum where they cannot determine any other party's CCP values, by definition of the secure sum.

By the composition theorem (see (Goldreich, 1998)) *if all components of the protocol are secure then the protocol itself is secure*. Therefore, this algorithm is secure. \square

4.3 Horizontally Partitioned Databases, Private Queries

In this case, party P^q must keep its query private while computing the horizontally partitioned PNN. In doing so, it must securely find the distance between \mathbf{x} and each $\mathbf{X}_r^{k,j}$, with the help of the secure scalar product (Goethals et al., 2004). When the second party P^k has the encrypted distances, it can share the final sum between itself and the first party, P^q , without either knowing the full value. Each party can then calculate its own CCP^k , and

securely sum them.

To more easily analyze the private query, PP-PNN algorithm, let us first define normalized versions of each vector.

$$\begin{aligned}\mathbf{x}'(i) &= \frac{\mathbf{x}(i)}{\sqrt{2\sigma_i}} \forall i = 1 \dots D, \text{ and} \\ \mathbf{X}'_r{}^j &= \frac{\mathbf{X}_r{}^j(i)}{\sqrt{2\sigma_i}} \forall i = 1 \dots D.\end{aligned}$$

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

$$p(\mathbf{x}|c_j) = \frac{1}{(2\pi)^{D/2} \left(\prod_{i=1}^D \sigma_i \right) PT_j} \sum_{r=1}^{PT_j} \exp(-dis^2(\mathbf{x}', \mathbf{X}'_r{}^j)) \quad (3)$$

where $dis(\mathbf{x}', \mathbf{X}'_r{}^j)$ is the Euclidean distance between \mathbf{x}' and $\mathbf{X}'_r{}^j$. The distance can be written as a scalar product and two summations (Haiping and Huacan, 2006):

$$dis^2(\mathbf{x}', \mathbf{X}'_r{}^j) = \sum_{i=1}^D (\mathbf{x}'(i))^2 + \sum_{i=1}^D (\mathbf{X}'_r{}^j(i))^2 - 2\mathbf{x}' \cdot \mathbf{X}'_r{}^j \quad (4)$$

The algorithm for the private query, horizontally partitioned PNN is presented in figure 5. It begins with a secure sum in lines 1–2, again used to calculate the number of training points in each class, which is then made public to all of the parties. Then the negative squared Euclidean distance between the query point \mathbf{x}' and every training point $\mathbf{X}'_r{}^{k,j}$ must be calculated. As shown in equation (4), to calculate this distance requires the sum of the squared components of both \mathbf{x}' and $\mathbf{X}'_r{}^{k,j}$, which is easy for the respective parties to obtain. To complete the distance calculation, the scalar product between \mathbf{x}' and $\mathbf{X}'_r{}^{k,j}$ is needed.

Lines 3–8 are executed on P^q , and lines 9–17 are executed on every other party. On line 4, P^q starts iterating over the set of all parties, because it must communicate with each of them. Each party must iterate through each class c_j . For every training point in every class, P^q must engage 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 .

As the algorithm enters line 6 on P^q and line 11 on the other parties, they are performing a shared scalar product over the non-querying parties' training points. However, there is a modification to the standard secure scalar product. Because each party does not want to reveal what training points are members of what classes, or how many of each class that the party has, the party must generate a spurious value for the share of the secure scalar product when the training point is not in the current party in question. The non-querying parties keep track of what shares are valid and which are not, and will filter them out later in the algorithm. Therefore, on line 13, the party will calculate the share of a particular training point properly if it owns that training point in $\mathbf{X}_r'^{k,j}$; if it does not, it will randomly make a share from a distribution that looks the same as the real values (lines 15–16). Each party P^k joins these spurious random distance shares to its distance shares, so that it has exactly PT_j shares in each class and therefore the class labels of its training points as well as their number are not revealed to P^q . On line 14–16, the parties P^k make and send these spurious random shares. Actual secure scalar products are performed at the non-querying parties on lines 12–13.

On line 7, party P^q engages in a secure scalar product not knowing whether it is between real or spurious values. In this pseudo-code, the *sharedSSP* (shared secure scalar product) is called by each party, with the parameter being the vector owned by their respective parties. On each party, it will return the respective random shares s^A and s^B to P^q and the non-querying parties respectively. The respective random shares can then be added to the respective sum of squared components (again, line 7 and line 13). Note that in practice, the transfers for the secure scalar product should be done in bulk, to avoid repeated communication latency between each party.

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}' . All of the non-querying parties also know which shares 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^q and each P^k (lines 8 and 16 for P^q and the non-querying parties respectively). On line 8, P^q also adds its own distance calculations to the share of the CCP (it is faster because it does not have to encrypt these distances). This scalar product skips operating on the spurious shares on the non-querying parties, because they are aware which values are spurious. The secure summation of all of these will then yield the CCP vector to produce the final result (lines 18–19). Finally, at P^q , the class conditional probabilities are divided by the appropriate factors and a label is found through linear search (lines 20–23).

In the algorithm, each party P^k adds spurious values to the vectors it

```

1 foreach  $c_j$  do
2    $\lfloor$  secureSum( $P^q$ ,  $PT_j^k$ ,  $PT_j$ );
3 if  $k == q$  then
4   foreach  $P^k$  do
5     foreach  $c_j$  do
6       foreach  $r = 1 \dots PT_j$  do
7          $\lfloor$   $sA_r = \exp(2 \cdot \text{sharedSSP}(\mathbf{x}') - \sum_{i=1}^D (\mathbf{x}'(i))^2)$ ;
8          $\lfloor$   $CCP_j^k += \text{sharedSSP}(sA) + \sum_{\forall \mathbf{X}_r^{q,j}} -\text{dis}^2(\mathbf{x}', \mathbf{X}_r^{q,j})$ ;
9 else
10  foreach  $c_j$  do
11    foreach  $r = 1 \dots PT_j$  do
12      if  $\mathbf{X}_r^{j,j} \in \mathbf{X}_r^{k,j}$  then
13         $\lfloor$   $sB_r = \exp(2 \cdot \text{sharedSSP}(\mathbf{X}_r^{k,j}) - \sum_{i=1}^D (\mathbf{X}_r^{k,j}(i))^2)$ ;
14        else
15           $\lfloor$   $sB_r = \exp(\text{randfloat}())$ ;
16           $\lfloor$  send( $P^q, \text{randfloat}()$ );
17         $\lfloor$   $CCP_j^k += \text{sharedSSP}(sB)$ ;
18 foreach  $CCP_j^k$  do
19    $\lfloor$  secureSum( $P^q$ ,  $CCP_j^k$ ,  $CCP_j$ );
20 if  $k == q$  then
21   foreach  $CCP_j$  do
22      $\lfloor$   $CCP_j /= (2\pi)^{D/2} PT_j \prod_{i=1}^D \sigma_i$ ;
23    $\lfloor$   $C(\mathbf{x}) = \text{argmax}_j \{CCP_j (PT_j/PT)\}$ ;

```

Figure 5: The algorithm for the horizontally-partitioned, private query PP-PNN

returns, expanding its number of values from $O(PT^k)$ to $O(PT)$ (lines 14–16). Communication for this portion can be optimized by only sending the D -dimensional operand \mathbf{x}' over once for all K parties.

Because of the stark contrast in performance between standard arithmetic and the encrypted arithmetic that must be done in this algorithm, a new constant E is introduced to take into account the time required by an encrypted operation, compared to $O(1)$ for the corresponding standard operation. The computational complexity of a private query, horizontally partitioned PP-PNN algorithm is equal to $O(KED(\max PT^k) + EK(PT) + D \max PT^k + JK)$. The communication complexity is equal to $O(JK + KD + KPT)$.

Theorem 2. *The horizontally partitioned, private query PNN algorithm given in figure 5 is secure by the SMC definition of security.*

Proof. During the secure sum in lines 1–2, P^q receives the total number of training points in each class, which as before is considered acceptable public knowledge. By prior security proof of the secure sum (Schneier, 1995), it is also known that the secure sum computation itself conforms to the SMC security definition.

By lemma 3, lines 3–7 on P^q and lines 9–16 on all other parties are secure. The secure scalar product which disregards spurious shares, taking place on lines 8 and 17 for P^q and P^k respectively, is secure by lemma 4.

In lines 18 and 19, P^q receives the unnormalized CCP_j values, which can be generated from the final probabilities and PT_j , which is public knowledge. Therefore, this revelation is also secure by the SMC definition.

Again, by the composition theorem, because all sections of this algorithm are private with respect to the SMC definition, the algorithm itself is secure.

□

Lemma 3. *The actions performed in lines 3–7 and lines 9–16 in the horizontally partitioned, private query PNN are secure by the SMC definition of security.*

Proof. This lemma and others depend on a *proof by simulation*. Proof by simulation states that if each party can simulate each part of the algorithm through using its own knowledge and public knowledge, then the algorithm is secure. The proof can be further simplified by concentrating only on simulating the messages received, as each party’s local calculations and messages sent can be generated from the local knowledge and the received messages.

On line 7, P^q receives a share of a secure scalar product or a spurious value. Because the existing security proofs for the secure scalar product (Goethals et al., 2004) emphasize that no additional knowledge about the total is gained from this share, P^q can simulate these with random numbers drawn from the same field. The same process can simulate the spurious values. In line 13, the data received from P^q during the secure scalar product is entirely encrypted and indecipherable by the other parties and can be simulated using random encrypted numbers.

□

Lemma 4. *The secure scalar product performed on lines 8 and 17 of the horizontally partitioned, private query PNN are secure by the SMC definition*

of security

Proof. The secure scalar product performed between P^q and each P^k on lines 8 and 17 of figure 5 is similar to the one in (Goethals et al., 2004); the difference is that, because P^k knows which of those shares are spurious, it knows not to include them in the final result. A proof by simulation will show that this secure scalar product reveals nothing additional.

To begin with, P^q transmits PT encrypted distances to P^k . Because the size of the vector PT is public knowledge, and the distances are encrypted with only P^q owning the private key, P^k can simulate this message by generating PT random encrypted numbers.

The homomorphically-encrypted operations are then performed on P^k . The number of valid shares owned by P^k can be in the range of $[0, PT]$. Therefore, the range of the resultant share ($[P^k e^{-D}, PT]$) is the same for every party P^k , and is well known by all parties. P^q can simulate the final share it receives by simply drawing a random number from that same range. Because all information received can be simulated, this portion of the algorithm is secure. \square

4.4 Vertically Partitioned Databases, Public Queries

As stated before, classes are kept privately by party P^c in the vertically partitioned case. Because the PNN's calculations center around computing values by class, this fact dictates much of the structure of the algorithm. For

all parties, the set \mathbf{D}^k represents the vertical partitions or columns owned by P^k . The vertically partitioned, public query PNN is presented in figure 7.

First, P^q generates its key pair (lines 1–2) and broadcasts it to the other parties (lines 3–4). On lines 5–7, a secure sum calculates each PT_j and the result is broadcast, because these final values are assumed to be public. In lines 8–9, all parties iterate over their training points $\mathbf{X}_r^{k,j}$, calculating $\sum_{i \in D^k} (\mathbf{X}_r^{k,j}(\mathbf{D}^k(i)) - x(i))^2$ for each, and the result is kept in \mathbf{Y}_r^k .

Lines 10–13 then add these \mathbf{Y}_r^k arrays using a secure sum, and share that secure sum between different parties through secret sharing ((Shamir, 1979; Blakley, 1979)). The final result of the secure sum is split in such a way that adding their new shares yields the final sum, but neither party knows the full sum.

To split the result of the secure sum, a function *sharedSecureSum*, is introduced. It is similar to the previously presented secure sum algorithm except that instead of returning the full sum to the final party, it shares the sum between two parties. It takes four parameters. The first two are the two parties that will share the sum. The third parameter is the partial value that will be added in the sum, and the fourth parameter is the variable where the sum shares will end up on the two appropriate parties.

One such way to implement this functions is as follows. As in the standard secure sum, each party P^k has a share v^k of the sum. All of the parties split their respective shares into two positive shares v_1^k and v_2^k , which can be done by multiplying v^k by a random number $R \in [0, 1]$ and $(1 - R)$. After this,

```

1 if  $k == q$  then
2    $(sk, pk) = \text{GenerateKeyPair}()$ ;
3 broadcast( $pk$ );
4 broadcast( $x'$ );
5 foreach  $c_j$  do
6    $\text{secureSum}(P^q, PT_j^k, PT_j)$ ;
7   broadcast( $PT_j$ );
8 foreach  $\mathbf{X}_r^{k,j}$  do
9    $\mathbf{Y}_r^k = \sum_{i \in D^k} (\mathbf{X}_r^{k,j}(\mathbf{D}^k(i)) - x'(i))^2$ ;
10 if  $c == q$  then
11    $\text{sharedSecureSum}(P^q, P^q, Y_r^k, Y_r)$ ;
12 else
13    $\text{sharedSecureSum}(P^q, P^c, Y_r^k, Y_r)$ ;
14 if  $c == q$  and ( $k == q$  or  $k == g$ ) then
15   foreach  $c_j$  do
16     if  $k == q$  then
17        $CCP_j = \text{SSP}(\text{pad}(\exp(-\mathbf{Y}_r), c_j))$ ;
18     else if  $k == g$  then
19        $\text{SSP}(\exp(-\mathbf{Y}_r))$ ;
20 else
21   foreach  $c_j$  do
22     if  $k == q$  then
23        $CCP_j = \text{SSP}(\exp(-\mathbf{Y}_r))$ ;
24     else if  $k == c$  then
25        $\text{SSP}(\exp(-\mathbf{Y}_r))$ ;
26 if  $k == q$  then
27   foreach  $CCP_j$  do
28      $CCP_j /= (2\pi)^{D/2} PT_j \prod_{i=1}^D \sigma_i$ ;
29    $C(\mathbf{x}) = \text{argmax}_j \{CCP_j (PT_j / PT)\}$ ;

```

Figure 6: The PP-PNN Algorithm for the vertically partitioned public query case

two separate secure sums are conducted, one beginning and ending at P^A and one beginning and ending at P^B .

There are two options with sharing this sum. If the $q \neq c$, that is, if the querying party and the party with the classes are different, it is most efficient to share the distances between them (lines 12–13). If the party originating the query and owning the classes are the same, $q = c$, then the distances are shared with another party, P^g which is randomly selected, but is neither q nor c (lines 10–11). If this were to remain unshared between different parties, then P^g would be able to fully decrypt all of the distance values, violating privacy.

Now, the requisite distances have been calculated for each point in $\mathbf{X}_r'^{k,j}$, and they are shared between two parties, with each share in the variable Y_r on the two parties. The sum of the exponential of these distances must now be calculated to yield the class conditional probabilities (lines 14–25). In this pseudo-code, it is assumed that functions apply to each element of the vector. Again, different paths are chosen if the querying party and the party holding the classes are the same party.

At line 14, it is checked if $c = q$. If so, the algorithm makes sure that security of the distance shares is not violated by giving the full distances to P^c/P^q . For each class c_j , P^c makes a vector of size PT . For training points that are members of the particular class c_j , their shares of $\exp(-\mathbf{Y}_r)$ are placed into the respective positions. Zeros are placed in all other positions by the *pad* function. The *pad* function takes two parameters: the vector to

pad, and the class to preserve (not pad with zeros). Then the secure scalar product is executed. In this, the padded vector is then encrypted and sent to P^g . P^g then performs a secure scalar product between this vector and its own share vector $exp(-\mathbf{Y}_r)$. It sends the final result back to P^q , where P^q can decrypt it. The scalar product is calculated between the proper shares because the zeros at all other positions preclude the respective elements from being included in the scalar product. The secure scalar product, SSP , is not a shared scalar product but gives P^q the entire answer.

If P^q and P^c are not the same party, as in lines 20–25, then the class conditional probabilities are relatively easy (and more efficient) to calculate. Each party takes the negative exponential of their respective shares of the calculated distance values. Then, iterating over each class, a secure scalar product is calculated and given back to P^q . This secure scalar product takes place only between corresponding shares in the same class. Through this secure scalar product function, P^c receives all of the shares of \mathbf{s}_r^A from P^q , discriminating between valid and invalid shares.

In lines 26–28, P^q divides the CCP values it has received by the factors necessary to correct the calculation. Finally, in line 29, the class with the largest class conditional probability is selected for a testing point.

Overall, this algorithm requires $O(JK + PTmax|D^k| + EK(PT))$ time for computation as well as $O(JK + KPT)$ total communication.

Theorem 5. *The vertically partitioned, public-query PNN algorithm given in figure 6 is secure by the SMC definition of security.*

Proof. In lines 3–4, all parties besides P^q receive both the public key and the test point query, which are considered public. The secure sum of PT_j values (lines 6–7) has already been proven secure. The result of that computation, the PT_j values received by all parties, are acceptably revealed because these overall quantities have been declared public.

For lines 10–13, there are two different cases, with two possible secure sums. As is shown in lemma 6, the final shares reveal no additional information, and therefore, these lines are secure. Lemma 7 furthermore shows that lines 14–25 are secure.

By the composition theorem, because all of the portions of the algorithm are secure, the overall algorithm is also secure. \square

Lemma 6. *The function `sharedSecureSum` is secure by the SMC definition of security.*

Proof. At the beginning of the algorithm, before each share of the vector \mathbf{Y}_r^k is split, it is known that the sum $\sum_k Y_r^k = Y_r$ is in the range $0 \leq Y_r \leq D$. Splitting each element into two nonnegative shares is equivalent to splitting the final sum into two nonnegative shares. Because the split is done randomly, each share, we will call them $A(Y_r)$ and $B(Y_r)$, are in the range $0 \leq A(Y_r), B(Y_r) \leq D$. Therefore, the shares which will be used in each of the independent secure sums, conform to the constraints of the non-shared secure sum.

The shared secure sum is therefore secure by the composition theorem (Goldreich, 1998), because it is composed of two secure sums, which have

already been shown secure. □

Lemma 7. *The actions in lines 14–25 of the vertically partitioned, public-query PNN algorithm in figure 6 are secure*

Proof. In lines 14–19, there are two sets of messages received. In participating in a secure scalar product, party P^q receives the quantity in CCP_j (line 17) which is only different from the final answer by a publicly known quantity (the divisor in line 28). Therefore by using this final answer and public information, it can effectively simulate receiving this quantity. In participating in the secure scalar product, party P^g receives only encrypted quantities, which it cannot decrypt (line 19) and can therefore simulate by random strings. In lines 20–25, a similar situation is encountered, wherein P^q as before receives something different from the final answer only by factor of the divisor in line 28: therefore P^q can easily simulate that message using that publicly known divisor and the final answer. Additionally, party P^c receives only encrypted strings during the secure scalar product, which it can easily simulate. Therefore, by simulation, these lines are secure. □

4.4.1 Vertically Partitioned Databases, Private Queries

The vertically partitioned, private query PNN is similar to the public query case, but with additional encrypted operations. The algorithm is given in figure 7.

As before, in lines 1–3, the public and private key are generated by P^q and the public key is broadcast to all of the parties. In line 4, the encrypted

```

1 if  $k == q$  then
2    $(sk, pk) = \text{GenerateKeyPair}()$ ;
3 broadcast  $(pk)$ ;
4 broadcast  $(\text{Enc}(\mathbf{x}'))$ ;
5 foreach  $c_j$  do
6    $\text{secureSum}(P^q, PT_j^k, PT_j)$ ;
7   broadcast  $(PT_j)$ ;
8 foreach  $\mathbf{X}_r'^{k,j}(\mathbf{D}^k(i))$  do
9    $\text{Enc}(Y_r^k) =$ 
10   $\text{Enc}\left(\sum_{i \in \mathbf{D}^k} - \left(\mathbf{X}_r'^{k,j}(i)\right)^2\right) \otimes \left(\prod_{i \in \mathbf{D}^k} \text{Enc}(\mathbf{x}'(i)) \mathbf{X}_r'^{k,j}(i)\right)^2$ ;
11 if  $k == q$  then
12    $\text{Enc}(Y_r^k) = \text{Enc}(Y_r^k) \otimes \text{Enc}(\sum_{i \in \mathbf{D}^k} - (x'(i)^2))$ ;
13 if  $c == q$  then
14    $\text{sharedEncryptedSecureSum}(P^q, P^g, \text{Enc}(\mathbf{Y}_r^k), \mathbf{Y}_r)$ ;
15 else
16    $\text{sharedEncryptedSecureSum}(P^q, P^c, \text{Enc}(\mathbf{Y}_r^k), \mathbf{Y}_r)$ ;
17 if  $c == q$  and  $(k == q \text{ or } k == g)$  then
18   foreach  $c_j$  do
19     if  $k == q$  then
20        $CCP_j = \text{SSP}(\text{pad}(\text{exp}(\mathbf{Y}_r, c_j)))$ ;
21     else if  $k == g$  then
22        $\text{SSP}(\text{exp}(\mathbf{Y}_r))$ ;
23 else
24   foreach  $c_j$  do
25     if  $k == q$  then
26        $CCP_j = \text{SSP}(\text{exp}(\mathbf{Y}_r))$ ;
27     else if  $k == c$  then
28        $\text{SSP}(\text{exp}(\mathbf{Y}_r))$ ;
29 if  $k == q$  then
30   foreach  $CCP_j$  do
31      $CCP_j /= (2\pi)^{D/2} PT_j \prod_{i=1}^D \sigma_i$ ;
32    $C(\mathbf{x}) = \text{argmax}_j \{CCP_j (PT_j / PT)\}$ ;

```

Figure 7: The PP-PNN Algorithm for the vertically partitioned private query case

query is broadcast. Again, in lines 5–7, the PT_j values are calculated, and broadcast.

Lines 8–11 pertain to calculating the square of the Euclidean distance. In lines 8–9, every party calculates the sum of the squares of the distance components over each of its available dimensions (the operator \amalg signifies repeated application of the operation \otimes). Specifically, in the private query version of the vertically partitioned algorithm, the negative distance is calculated, because of the limitations of the Paillier cryptosystem (difficulty multiplying by a -2). To complete the distance sum, P^q must add the encrypted sum of the query point \mathbf{x}' on lines 10–11.

Both the secure sum and the homomorphic additions and multiplications can be expensive operations, because of the computational complexity of cryptographic operations and the relatively high latency and low bandwidth of Internet connections. However, a tree arrangement can make secure operations faster (Vaidya and Clifton, 2003a). Any function that can be represented as $y = f(x_1, \dots, x_k) = x_1 \otimes x_2 \otimes \dots \otimes x_k$ with \otimes being an associative operation can be securely computed with the tree structure. Again, it is assumed that the parties are numbered from $P^1 \dots P^K$ with their respective operands $v^1 \dots v^K$ without loss of generality. The leaf parties (P^{2^w} through P^K) begin by performing the secure operation \otimes with their parents ($P^{2^{w-1}}$ through $P^{2^{w-1}}$). The result of this associative operation should end up on those parent parties and then the parent parties themselves participate in another three-way operation. These three-way operations continue up the

tree until the results reach parties P_1 , P_2 and P_3 .

In lines 12–15, a shared encrypted secure sum is performed. The function *sharedEncryptedSecureSum*, is a tree structured operation for efficiency. It takes four parameters. The first two are the parties that will share the final plaintext value. The third parameter is the encrypted operand and the final parameter is where the result will be stored. During this function, each party receives the encrypted operands from its two children parties, and adds them homomorphically (Paillier, 1999) to its own before transmitting that value to its parent. To maintain security, the host with the private key pk (P^q here), must remain a leaf of the tree. If, for instance, a party with the public key were on the second level of the tree, it would be able to completely decrypt information received from a leaf node. The root node is assumed to be the other party specified in the function (P^c or P^g). When the encrypted operand reaches the root node, it subtracts a random share as in the secure scalar product (Goethals et al., 2004), and sends the encrypted share to P^q where it is decrypted.

Again, as in the public query case, we must make a distinction between the cases where $q = c$. If $q = c$, then the result of the secure sum must be shared with P^g instead. From this point on (lines 16–31), the algorithm is identical to the public query case with the exception that the values of \mathbf{Y}_r on lines 19, 21, 25 and 27 are not negated.

The shared encrypted secure sum takes $O(E(PT)\log(K))$ computation and $O(PT\log(K))$ communication (lines 12–15), because it makes use of a

tree structure. The overall algorithm requires $O(JK + E(PT)(\max|D^k| + \log(K)) + EK(PT))$ computation and $O(JK + PT\log(K) + KPT)$ for communication.

Theorem 8. *The vertically partitioned, private-query PNN algorithm given in figure 7 is secure by the SMC definition of security.*

Proof. On line 3, all of the parties received the public key, which is of course, public knowledge. The parties also receive the encrypted query. P^q already knows this query, and by definition of the cryptosystem, it is unable to be understood by the other parties.

Again, it is known by the secure sum proof that the secure sum on line 6 is secure, and its result, the quantities PT_j which are broadcast, are considered public knowledge.

Lines 10 and 11 involve only local operations, and because of the homomorphic cryptosystem used, do not reveal additional information. According to lemma 9, lines 12–15 of the algorithm are secure. The security of lines 16–31 were already proven for the vertically partitioned, public query algorithm (lemma 7) and are omitted for brevity.

By the composition theorem, because all sections of the algorithm are secure, the overall algorithm is secure. \square

Lemma 9. *The actions in lines 12–15 of the vertically partitioned, private-query PNN algorithm in figure 7 are secure*

Proof. This lemma will be proven by simulation. During the shared encrypted secure sum, only parties without the private key receive messages, because the only party with the private key, P^q is a leaf node of the secure tree, and therefore receives no message during the tree-based communication. All other parties receive only encrypted operands during this phase, which can be easily simulated by random strings.

Because random shares of the final sum are split between either P^q and P^g or alternatively between P^q and P^c , neither party is able to obtain any additional information about the final sums. Therefore, because all parties know the limits $[e^{-D}, D]$ of the sum, both parties can simulate their respective shares with a vector of random numbers chosen from the same range. \square

5 Experimental Setup

5.1 Implementation

The algorithms were implemented in C++ using the MPICH (mpi, 2007) implementation of the MPI communication standard. To implement the Paillier encryption scheme, the Number Theory Library (NTL) (ntl, 2007) was used along with code adapted from (Liu, 2007).

One practical issue that must be dealt with when using the Paillier cryptosystem is the fact that cannot naturally encrypt floating-point numbers. Floating-point numbers must be converted to a fixed-point representation. This is done by multiplying them by a large constant C and then truncating

the result to an integer. In these experiments, $C = 100000$. Other methods (Fouque et al., 2002) can also be used.

Due to the unavailability of a coordinated multi-organizational set of servers for this experiment, a cluster of servers was used for the testing, with 64 quad-core, Intel Xeon 3GHz nodes, each with 8GB of RAM, connected to a frontend and centralized storage by gigabit Ethernet switches.

5.2 Test Description

Because this paper’s objective is not to assess the classification accuracy of the PNN but to evaluate the security and efficiency of the algorithms, an artificially generated database is used, consisting of 2-class Gaussian functions with a 15% overlap on one of the dimensions. For the interested reader, there is a significant amount of literature that demonstrates the classification performance of the PNN (Specht and Shapiro, 1991; Zhong et al., 2007; Specht, 2006, 1992).

The performance test are intended to evaluate the scaling of the four algorithms, with separate tests for the horizontally and vertically distributed algorithms. In the horizontal tests, the number of parties is varied from 3 to 16, with each party owning 8,000 training points. Therefore, the total number of training points varies from 24,000 to 128,000. The dimensionalities of the training points are also varied using values of 8,16,32, and 64 dimensions.

In the vertical tests, the number of parties is also varied from 3 to 16. The dimensionality of the training points is varied using values of 8,16,32, and 64

dimensions. A constant size of 128,000 training points is used to closely simulate a large scale data mining task. The dimensions of the 128,000 points are split evenly among the parties, with the last party also receiving the labels of the training points. Both the private and public query algorithms are evaluated for the time that it takes for them to execute a single test point query. 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, no effort was expended to optimize the sigma parameter values.

6 Results

In the graph for the horizontally partitioned, public query PNN (figure 8), scaling was quite moderate for both increased dimensionality and increased number of parties in the computation. In this graph, as in the others, some minor fluctuations can be seen, attributable to background loads on the cluster computer.

Clearly, of the four algorithms presented, the horizontally partitioned, public query PNN has the best performance and is best suited for fast operation in a distributed environment. This is because it avoids the use of cryptography, provides data parallelism and minimizes communication.

In figure 9, the performance of the horizontally partitioned, private query algorithm is shown. The time appears to scale quadratically with the number

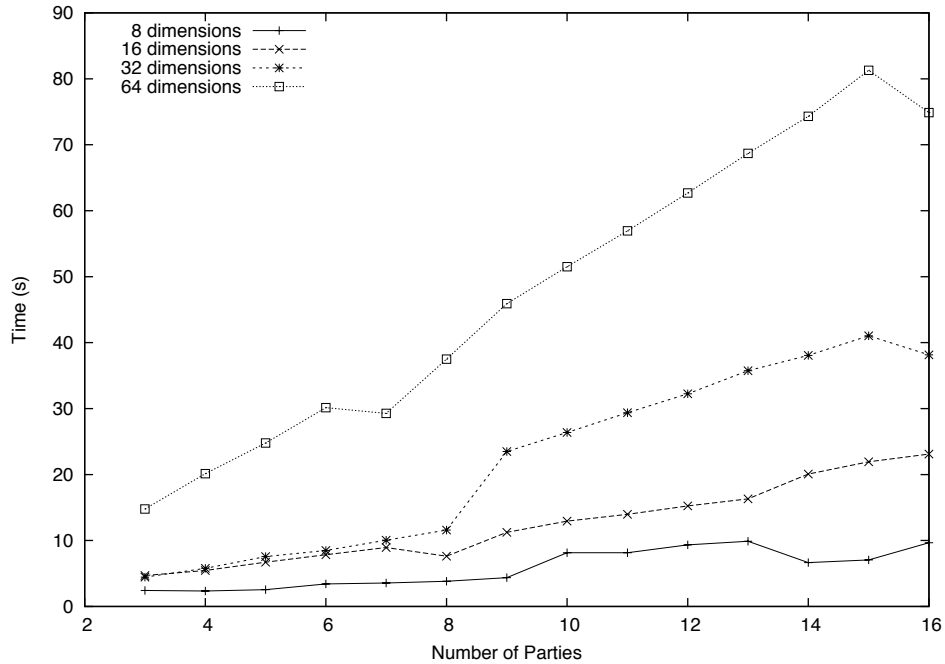


Figure 8: Performance of the horizontally partitioned, public query PNN. The time taken scales gradually with the number of parties, and is low in absolute terms. The scaling is moderate because there are no expensive encrypted operations, and much of the work is done in parallel by the parties. The time taken scales with the dimensionality as in the standard serial algorithm.

of parties and training points, likely because of number of spurious values that must be sent in order not to reveal each parties' number of training points in each class. While the private query case remains practical to perform, there is a significant difference in performance between the private query and the public query cases, further supporting the need for differentiating on the privacy of the query. Code profiling of the private query case reveals that 90% of the time is spent in the large integer calculation functions needed by

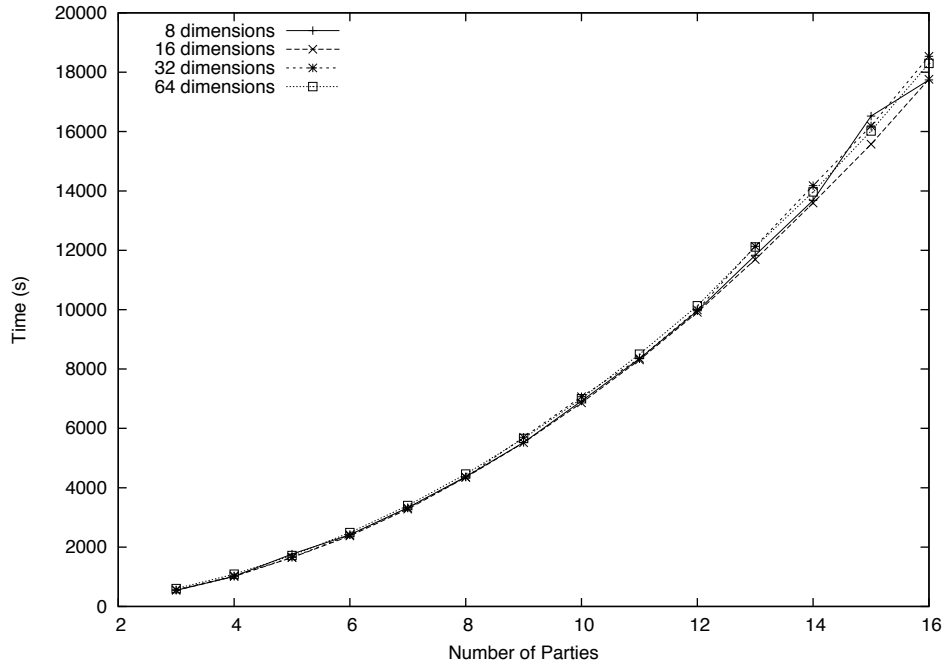


Figure 9: Performance of the horizontally partitioned, private query PNN. The algorithm scales quadratically with the number of points. Dimensionality does not affect it significantly because the time is dominated by the encrypted operations for the distance calculations, which is constant across dimensionalities in this algorithm.

encrypted operations.

In figures 10 and 11, the performance graphs are shown for the vertically partitioned public and private query, respectively. Both algorithms scale moderately with increased dimensionality, and stay mostly constant for different numbers of parties. Both are shown to be practical to use. However, the public query version again has a significant performance advantage over the private query version, for the same reasons stated for the horizontally

partitioned algorithms: the private query version incurs additional encrypted operations, which are computationally intensive. Again, this reinforces the case for differentiating algorithms on the basis of query privacy.

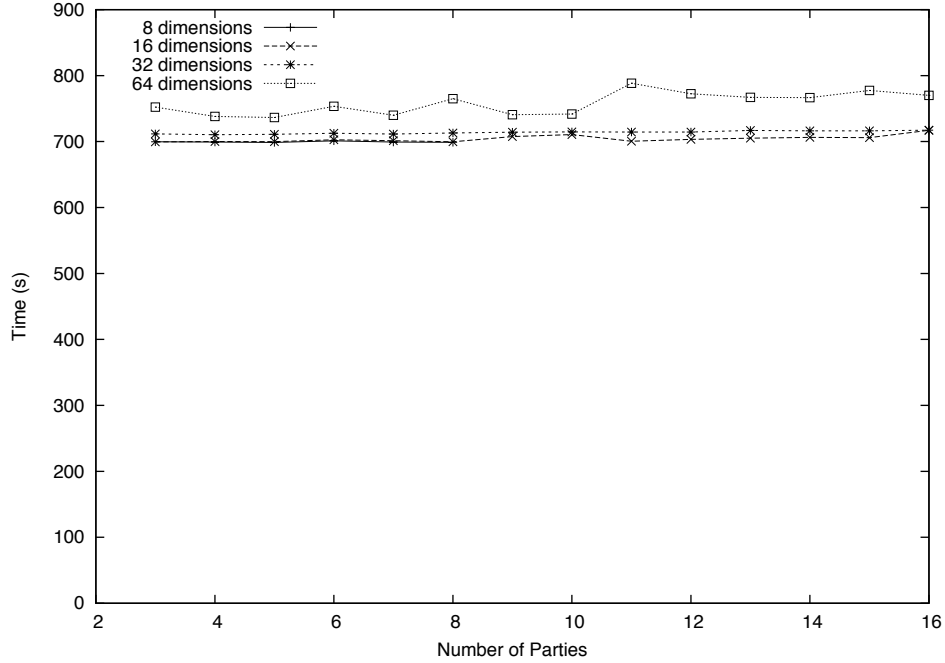


Figure 10: Performance of the vertically partitioned, public query PNN. The performance remains relatively constant over the dimensionality and number of parties, because the number of demanding encrypted operations is on the order of the number of points in the training set.

7 Summary, Conclusions and Future Work

In this paper, we have shown four practical algorithms for implementing the PNN in a privacy preserving distributed environment, and evaluated their

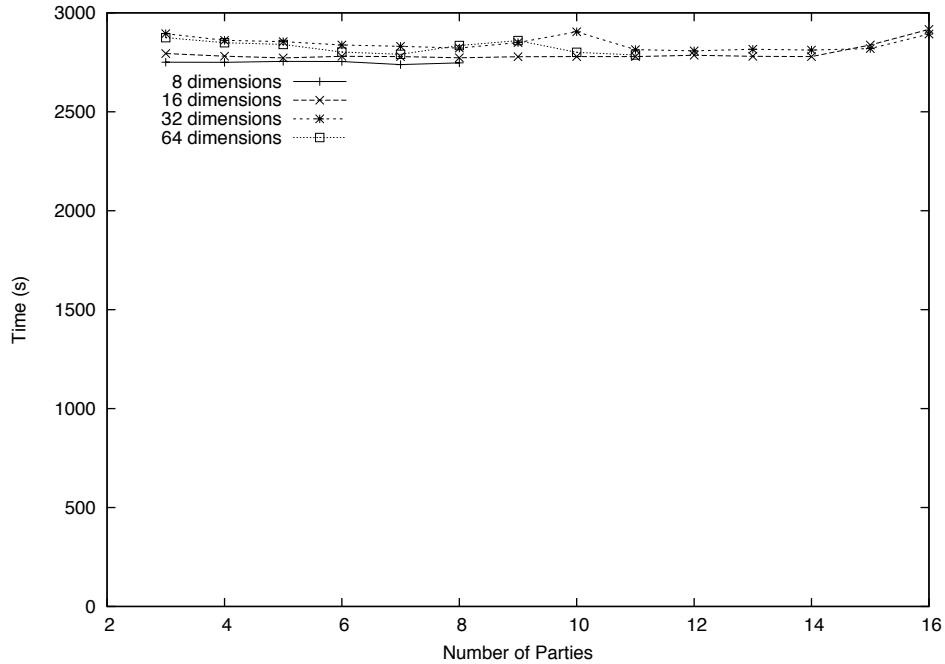


Figure 11: Performance of the horizontally partitioned, private query PNN. The behavior is similar to the public query version, remaining relatively constant for number of parties and dimensionality, again because the number of encrypted operations is on the order of the number of training points.

performance on a large scale data mining task. While both PP-PNN algorithms (public-query and private-query) remain practical, clearly the public query algorithms have a performance advantage over the private query ones. This gain shows that it is sensible to differentiate on the basis of query-privacy. As a rule, whenever one is allowed to expose a greater amount of knowledge, one can use that fact to potentially increase the algorithm’s efficiency. It has also become clear that where possible, one should avoid the expense incurred by encrypted operations. We have also developed several

procedures (e.g. the shared encrypted secure sum) which should be useful for implementing other data-intensive privacy preserving mining algorithms.

Future work will consider implementing intra-party parallelism. Especially in the horizontally partitioned, public query algorithm, the distances between the query and the training points can be calculated by parallel machines at each party site, thus rapidly accelerating the queries. All of the algorithms may benefit from using specialized hardware to increase the speed of cryptographic computations. It is also hoped that the future will bring stronger primitives, such as more capable homomorphic cryptosystems, with which to develop privacy preserving data mining algorithms.

References

- (2007). Mpich2 home page. <http://www-unix.mcs.anl.gov/mpi/mpich2/>.
- (2007). Ntl: A library for doing number theory. <http://www.shoup.net/ntl/>.
- Agrawal, R. and Srikant, R. (2000). Privacy-preserving data mining. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 439–450. ACM Press.
- Barni, M., Orlandi, C., and Piva, A. (2006). A privacy-preserving protocol for neural-network-based computation. In *MM&Sec '06: Proceeding of the 8th workshop on Multimedia and security*, pages 146–151, New York, NY, USA. ACM Press.
- Blakley, G. R. (1979). Safeguarding cryptographic keys. In *National Computer Conference*, volume 48, pages 313–317.
- Du, W. and Zhan, Z. (2002). A practical approach to solve secure multi-party computation problems. In *Proceedings of New Security Paradigms Workshop*, Virginia Beach, virginia, USA.

- Emekci, F., Sahin, O., Agrawal, D., and Abbadi, A. E. (2007). Privacy preserving decision tree learning over multiple parties. *Data & Knowledge Engineering*, 63(2):348 – 361.
- Fouque, P.-A., Stern, J., and Wackers, G.-J. (2002). Cryptocomputing with rationals. In *6th International Conference on Financial Cryptography*, volume 2357 of *Lecture Notes in Computer Science*, pages 136–146. Springer-Verlag.
- Georgiou, V. L., Pavlidis, N. G., Parsopoulos, K. E., Alevizos, P. D., and Vrahatis, M. N. (2006). New self-adaptive probabilistic neural networks in bioinformatic and medical tasks. *International Journal on Artificial Intelligence Tools*, 15(3):371–396.
- Goethals, B., Laur, S., Lipmaa, H., and Mielikäinen, T. (2004). On private scalar product computation for privacy-preserving data mining. In Park, C. and Chee, S., editors, *Information Security and Cryptology - ICISC 2004, 7th International Conference, Seoul, Korea, December 2-3, 2004, Revised Selected Papers*, volume 3506 of *Lecture Notes in Computer Science*, pages 104–120. Springer.
- Goldreich, O. (1998). Secure multi-party computation. (Sep. 1998) Working Draft.
- Goldreich, O., Micali, S., and Wigderson, A. (1987). How to play any mental game. In *STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 218–229, New York, NY, USA. ACM Press.
- Gorunescu, F., El-Darzi, E., and Gorunescu, S. (2005). An evolutionary computational approach to probabilistic neural network with application to hepatic cancer diagnosis. In *CBMS '05: Proceedings of the 18th IEEE Symposium on Computer-Based Medical Systems*, pages 461–466, Washington, DC, USA. IEEE Computer Society.
- Haiping, W. and Huacan, H. (2006). Tools for privacy preserving kernel methods in data mining. In *AIA '06: Proceedings of the 24th IASTED international conference on Artificial intelligence and applications*, pages 388–391, Anaheim, CA, USA. ACTA Press.
- Hu, W.-B., Li, K.-C., and Zhao, D. (2007). A novel probabilistic neural network system for power quality classification based on different wavelet

- transform. In *International Conference on Wavelet Analysis and Pattern Recognition*, volume 2, pages 746–750.
- Kantarcoglu, M. and Vaidya, J. (2003). Privacy preserving naive bayes classifier for horizontally partitioned data. In *IEEE ICDM Workshop on Privacy Preserving Data Mining*, pages 3–9, Melbourne, FL.
- Kiyan, T. and Yildirim, T. (2003). Breast cancer diagnosis using statistical neural networks. In *International XII. Turkish Symposium on Artificial Intelligence and Neural Networks – TAINN 2003*.
- Liu, K. (2007). Paillier java code. <http://www.csee.umbc.edu/~kun-liu1/research/Paillier.java>.
- Lo, J., Land, W., and Morrison, C. (1999). Application of evolutionary programming and probabilistic neural networks to breast cancer diagnosis. In *International Joint Conference on Neural Networks*, volume 5, pages 3712–3716 vol.5.
- Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In Stern, J., editor, *Advances in Cryptology – EUROCRYPT ’99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer-Verlag.
- Parzen, E. (1962). On estimation of probability density function and mode. *Annals of Mathematical Statistics*, 33:1065–1073.
- Pinkas, B. (2002). Cryptographic techniques for privacy-preserving data mining. *SIGKDD Explor. Newsl.*, 4(2):12–19.
- Rozenberg, B. and Gudes, E. (2006). Association rules mining in vertically partitioned databases. *Data & Knowledge Engineering*, 59(2):378 – 396.
- Schneier, B. (1995). *Applied Cryptography*. John Wiley & Sons, 2nd edition.
- Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613.
- Specht, D. (1991). A general regression neural network. *IEEE Transactions on Neural Networks*, 2(6):568–576.

- Specht, D. (1992). Enhancements to probabilistic neural networks. In *International Joint Conference on Neural Networks*, volume 1, pages 761–768.
- Specht, D. (2006). Grnn with double clustering. In *Neural Networks, 2006. IJCNN '06. International Joint Conference on*, pages 5074–5079.
- Specht, D. and Shapiro, P. (1991). Generalization accuracy of probabilistic neural networks compared with backpropagation networks. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume i, pages 887–892 vol.1.
- Specht, D. F. (1990). Probabilistic neural networks. *Neural Networks*, 3:109–118.
- Tam, C. M., Tong, T. K. L., Lau, T. C. T., and Chan, K. K. (2004). Diagnosis of prestressed concrete pile defects using probabilistic neural networks. *Engineering Structures*, 26(8):1155–1162.
- Vaidya, J. and Clifton, C. (2002). Privacy preserving association rule mining in vertically partitioned data. In *In The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
- Vaidya, J. and Clifton, C. (2003a). Leveraging the "multi" in secure multi-party computation. In *WPES '03: Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, pages 53–59, New York, NY, USA. ACM Press.
- Vaidya, J. and Clifton, C. (2003b). Privacy-Preserving K-Means Clustering over Vertically Partitioned Data. In *The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, DC.
- Yao, A. (1986). How to generate and exchange secrets. In *Proc. 27th Annual Symposium on Foundations of Computer Science*, pages 162–167.
- Yu, H., Vaidya, J., and Jiang, X. (2006). Privacy-preserving svm classification on vertically partitioned data. In *Pan-Asia Conference on Knowledge Discover and Data Mining (PAKDD)*, pages 647–656, Singapore.
- Zhan, J., Change, L., and Matwin, S. (2005). Privacy preserving k-nearest neighbor classification. *International Journal of Network Security*, 1(1).

Zhong, M., Coeggeshall, D., Ghaneie, E., Pope, T., Rivera, M., Georgiopoulos, M., Anagnostopoulos, G., Mollaghasemi, M., and Richie, S. (2007). Gap-based estimation: Choosing the smoothing parameters for probabilistic and general regression neural networks. *Neural Computation*, 19:2840–2864.