http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet

Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 9 (1): 1-10, 2010
ISSN 1812-5638
© 2010 Asian Network for Scientific Information

A Privacy Preserving Neural Network Learning Algorithm for Horizontally Partitioned Databases

Li Guang, Wang Ya-Dong and Su Xiao-Hong
Department of Computer Science and Engineering, Harbin Institute of Technology,
Harbin 150001, People's Republic of China

Abstract: Ordinary data mining requires accurate input data, but privacy concerns may bar use of such techniques. Thus, privacy preserving data mining methods are needed, which can work well without opening the private data. Although, much work has been done on privacy preserving classification, to the best of our knowledge, there has not been a privacy preserving perceptron neural network learning algorithm that can work in the real world on distributed databases. To solve this problem, this study brings forward a privacy preserving Back Propagation (BP) learning algorithm for horizontally partitioned databases. In this algorithm, data nodes can privately exchange information that the original BP algorithm needs. This algorithm can obtain the same result as learning on global data using the BP algorithm without considering privacy protection and each data nodes is prevented from obtaining detailed data on other nodes in the learning process.

Key words: Horizontally distributed databases, privacy preserving data mining, BP algorithm

INTRODUCTION

In recent years, Privacy Preserving Data Mining (PPDM) has become an important issue in data mining research (Stanley and Osmar, 2004; Vassilios *et al*., 2004; Elisa *et al*., 2005). General data mining methods assume that the data can be seen directly, so they do not fit for private data that should not be opened. The PPDM technology can perform data mining without accessing the details of the original data.

One branch of PPDM is the PPDM on distributed databases. This assumes there are multiple nodes, each of which has only a part of the global data set. These nodes want to carry out data mining on the global data set, but each node does not want others nodes to know its data. For example, some hospitals may want to do data mining on their patients' information, but each hospital does not want its patients' information to be opened. PPDM on distributed databases can solve this problem. Currently, most methods of PPDM on distributed databases are based on Secure Multi-party Computation (SMC). In these methods, all of the nodes exchange information required by the mining algorithm through information exchange protocols based on SMC (Chris *et al*., 2002). These protocols can allow the information to be exchanged privately, without allowing any node to obtain the data from other nodes.

To date, much study has been done on the privacy preserving classification techniques on distributed databases, including decision tree (Emekci *et al*., 2007; Justin, 2007), kNN (Mark *et al*., 2006; Artak and Vladimir, 2007), SVM (Sven *et al*., 2006; Jaideep *et al*., 2008), Bayesian classifier (Zhiqiang and Rebecca, 2006) and boosting (Sebastien *et al*., 2007). However, to the best of our knowledge, there has not been a privacy preserving perceptron neural network learning algorithm that is advanced enough to work in the real world on distributed databases. At present, the research about privacy preserving neural networks (Jimmy *et al*., 2007; Barni *et al*., 2006; Yancheng and Chijen, 2005; Saeed and Ali, 2008; Li *et al*., 2007) has not addressed the problem of training privately a practical perceptron neural network on distributed databases. A practical perceptron neural network should have at least three levels and use sigmoid activation functions.

Jimmy *et al*. (2007) presented a privacy preserving learning algorithm for a Probabilistic Neural Network (PNN). The PNN is very different from a perceptron neural network. In fact, it is a form of Bayesian optimal classifier. Barni *et al*. (2006) mainly study how to use the perceptron neural network for private classification, but not how to learn it. In their study, there are two nodes, the server and the client. The server already has the neural network. The client has data and wants to use the server's neural

Corresponding Author: Li Guang, Department of Computer Science and Engineering, Harbin Institute of Technology, Harbin 150001, People's Republic of China Tel: 008615846591735

network to process its data privately. Yancheng and Chijen (2005) address a similar topic, with one node having the network and the other having the data. Yancheng and Chijen (2005) study both how to use and how to learn the neural network. When learning, the training samples are on the data node; and the other node uses these samples to learn a network in privacy. However, all the data resides on one node, so it is not actually a distributed database. Saeed and Ali (2008) and Li *et al.* (2007) presented privacy preserving perceptron neural network learning algorithms for distributed databases. But they all make simplifying assumptions. Saeed and Ali (2008) assumes that the network uses a linear activation function and Li *et al.* (2007) assumed that the network only has one level. Thus, these algorithms are all useless in the real world, where we often use networks with a sigmoid activation function and at least three levels.

From the existing study presented above, the perceptron neural network, an important data mining method, cannot yet be used for private data in the real world. To solve this problem, this study presents a privacy preserving BP learning algorithm for horizontally partitioned databases. In horizontally partitioned databases, every node only has part of the tuples, but every tuple is complete.

In this new algorithm, protocols are designed to let nodes privately exchange information needed by the BP algorithm. This method can obtain the same result as learning on global data using the BP algorithm without considering privacy protection. At the same time, it prevents each node from obtaining the details of data on other nodes in the learning process. The price is the added cost of computation and communication compared to the standard BP algorithm. Compared with the previous methods (Saeed and Ali, 2008; Li *et al.*, 2007), this method can train a practical perceptron neural network, which has at least three levels, using the sigmoid activation function. This algorithm gives us a chance to use the neural network method when we deal with distributed private data in the real world.

PRIVACY PRESERVING NEURAL NETWORK LEARNING ALGORITHM

This study presents a privacy preserving BP learning algorithm for horizontally distributed databases. It can train a network with any number of levels and any activation function. This algorithm gives us a chance to use the neural network method when we deal with distributed private data in the real world. As with most of the privacy preserving classification mining methods for distributed databases, this method is based on secure multi-party computation. Specifically, it uses the secure sum protocol.

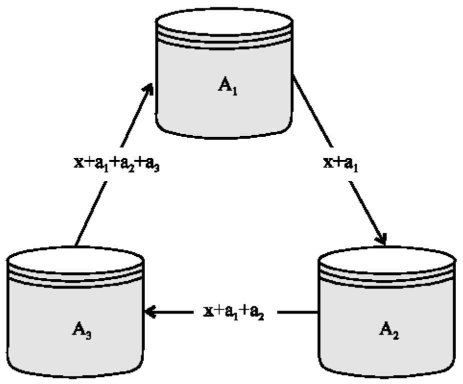

Fig. 1: A simple secure sum protocol

A simple secure sum protocol (Chris *et al.*, 2002) is shown in Fig. 1. There are three nodes: A_1, A_2, A_3, which have data a_1, a_2, a_3, respectively. They want to calculate $a_1+a_2+a_3$ in privacy. First, node A_1 generates a random number x and gives $x+a_1$ to A_2, then A_2 gives $x+a_1+a_2$ to A_3 and A_3 gives $x+a_1+a_2+a_3$ to A_1. A_1 Knows x, so it can obtain $a_1+a_2+a_3$.

This study's algorithm is based on the BP algorithm, which is the standard learning algorithm for perceptron neural networks. The BP algorithm always starts with a random weight-initialization and then uses an iterative gradient descent method to optimize these weights (Tom, 2003). In present algorithm, nodes can exchange information needed by the BP algorithm without revealing the training sample to other nodes.

Suppose there is a semi-trusted third party and all of the nodes will observe the protocol. They are interested in guessing the data on other nodes, using the information they can get legally, but all of them will never collude. This assumption is widely adopted in the research on privacy preserving data mining.

In this study's algorithm, d networks ($d>1$) are training at the same time. There are n data nodes: P_1, P_2,...,P_n. They arrange the encryption E and the cipher code K. A plaintext message M's ciphertext is E(M). This study focuses on the weights training problem; we assume the network's structure and activation functions are known. With this assumption, a network can be expressed as a vector $X = (x_1, x_2,...,x_m)$, where x_i is a specific weight. $Y = (y_1, y_2,...,y_m)$ is the update for X, where y_i is the update for x_i. All samples (including the training samples and the testing samples) are horizontally distributed and stored on all n data nodes.

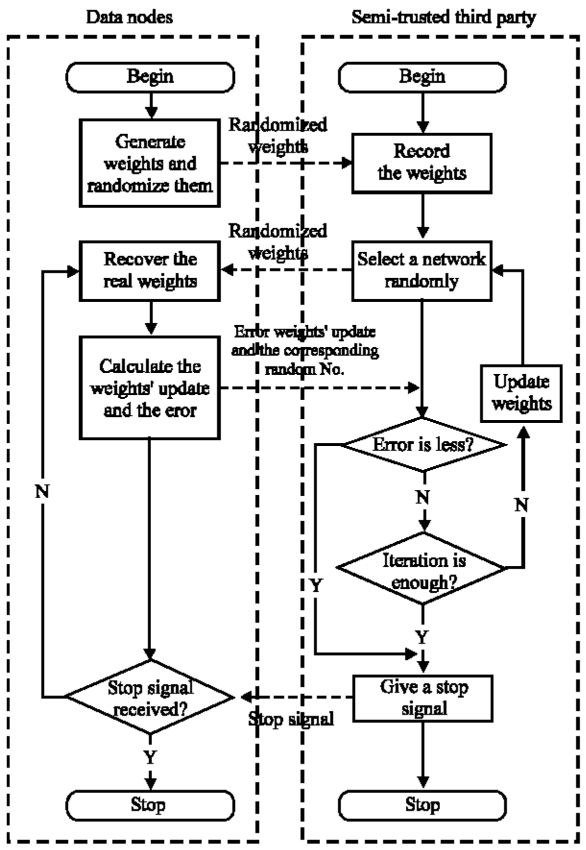

Fig. 2: The workflow of the privacy preserving neural network learning algorithm

As shown in Fig. 2, in this algorithm, data nodes randomly generate the initializing weights for networks and randomize the initializing weights and then the semi-trusted third party stores them and selects one network randomly to be trained. After data nodes calculate the weight's update and the network's error, the third party uses data nodes' result to finish the update and judge whether the training should be stop. The algorithm can be divided into two parts, namely, initialization and optimization. The optimization is also split into two parts: updating weights and judging whether optimization can stop.

Initialization: In initialization, each data node randomly generates X, which are the initializing weights for a network. Each data node also generates A and B, which are two random vectors with the same dimensions as X. X = X+A−B. X Is transferred to the semi-trusted third party Q. E(A) and E(B), the ciphertexts of A and B, are also given to Q. If there are n data nodes, Q will get n networks. One of them is selected randomly and is stored as (X, E(A), E(B), t). E(A) and E(B) are the ciphertexts of random vectors A and B. X is the randomized weight, which is the sum of the real weight and A−B. t identifies the data node that transfers X, E(A) and E(B) to Q.

Repeating this process d times, the semi-trusted third party Q will get d networks.

The detailed process of initialization is as follows:

```
for          i = 1 to d
   for    j = 1 to n

      P_j generates the i-th network's initializing weights X_ij
      and random vectors A_ij and B_ij with the same dimensions
      as X_ij:

      X_ij' = X_ij+A_ij−B_ij
      P_j gives (X_ij', E(A_ij), E(B_ij), j) to Q
   end for
   Q selects c ∈ {1, 2,...,n} randomly and stories
   U_i = (X_ic', E(A_ic), E(B_ic), c)
end for
```

Initialization never uses any samples, so it will not leak privacy.

Method of updating the weights: How to update the weights without seeing the samples directly is the core of our algorithm. To do that, two problems have to be solved.

- How to calculate the weights' update without seeing the samples directly

This problem is solved by making use of the additivity of weights' update. Let Y_d be the weights' update corresponding to training sample d. Y is the weights' update corresponding to the whole training samples set D. D_i is the training samples set on node P_i. Then $Y=\sum_{d\in D} Y_d=\sum_{i=1}^{n}(\sum_{d\in D_i} Y_d)$ will be obtained, which is called the additivity of weights' update. In this formula, $\sum_{d\in D_i} Y_d$ is the weights' update corresponding to node P_i and can be calculated by P_i itself using its local data.

- If a node knows the detailed training process of a network, it can build some equations of the samples by using the weights' update on each iteration. Because the number of iterations is large, there will be many equations, which constitute a threat to privacy protection because the nodes can obtain a lot of information about the samples from these equations. Thus, it is better not to let any node know the details of the training process

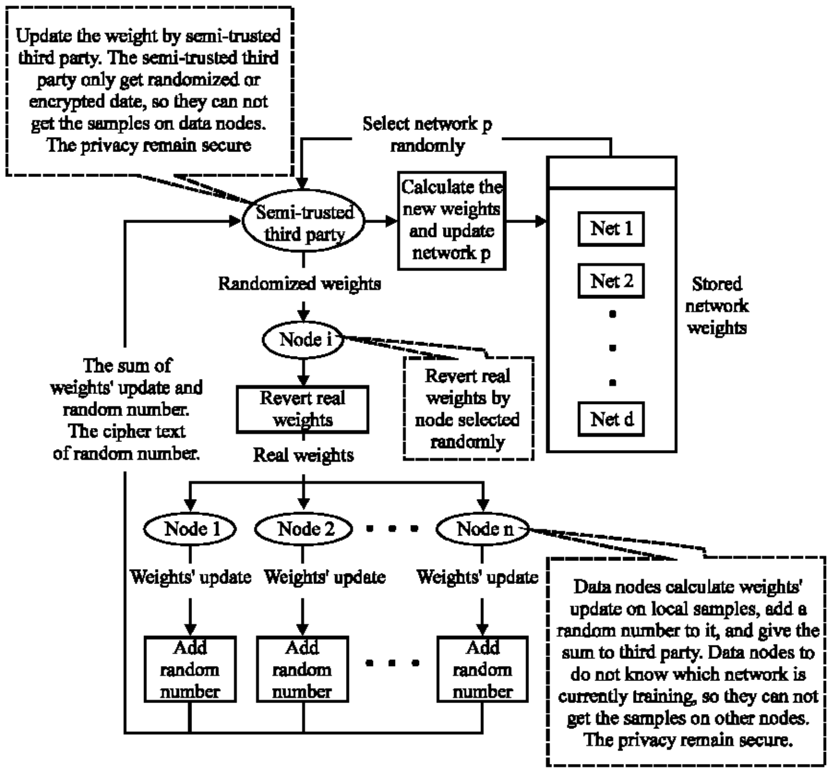
Update the weight by semi-trusted third party. The semi-trusted third party only get randomized or encrypted date, so they can not get the samples on data nodes. The privacy remain secure
Select network p randomly
Semi-trusted third party
Calculate the new weights and update network p
Net 1
Net 2
Net d
Stored network weights
Randomized weights
Node i
Revert real weights
Revert real weights by node selected randomly
The sum of weights' update and random number. The cipher text of random number.
Real weights
Node 1
Node 2
Node n
Weights' update
Weights' update
Weights' update
Add random number
Add random number
Add random number
Data nodes calculate weights' update on local samples, add a random number to it, and give the sum to third party. Data nodes to do not know which network is currently training, so they can not get the samples on other nodes. The privacy remain secure.

Fig. 3: The training process

This problem is solved by training more than one network (d networks). In each loop, semi-trusted third party Q decides which network is trained. Then, the data nodes calculate the update of this network, randomize it and give it to the third party Q, who then updates the network. In this process, data nodes do not know which network is trained in each loop; therefore, data nodes cannot link the information from different iterations and they cannot set up many equations. If data nodes cannot get what they want after one iteration, they also cannot get it after the full training process has been run. On the other hand, the third party can only get the randomized or encrypted data, which is not of any help for guessing the private information. In summary, no nodes can obtain information about the samples on other nodes and thus privacy remains protected. The process of training is shown in Fig. 3.

At the beginning, all the d networks are stored on third party Q. Q selects one network randomly and transfers it, which is stored as (X, E(A), E(B), t), to P'. P' is selected from $\{P_1, P_2,...,P_n\} - \{P_t\}$ randomly.

Then, P' gets A and B from E(A) and E(B) and calculates real weights X = X−A+B. P' transfers real weights X to all the data nodes. Each data node calculates the weights' update corresponding to its local data and randomizes the weights' update. If X is the real weights, Y_i is X's update related to data stored on node P_i. P_i generates random vector B_i. Q then calculates $\sum_{i=1}^{n}(Y_i + B_i)$ by a secure sum protocol.

After that, a data node P_l is selected randomly to calculate $B = \sum_{i=1}^{n} B_i$ by a secure sum protocol. Then, P_l gives E(B) and its own index l to Q. P_l generates random vector A_l with the same dimension as X, calculates $X_l = X + A_l$ and gives $E(A_l)$ and X_l to Q.

Finally, Q updates the network's weights. Q replaces (X, E(A), E(B), t), which is the currently-trained state of the network, with $((X + A_l) - \sum_{i=1}^{n}(Y_i + B_i)), E(A_l), E(\sum_{i=1}^{n} B_i), l)$.

Optimization termination: A network is trained successfully when its error on testing sample set T:

$$J = (1/2)\sum_{d \in T}\sum_{j \in out}(t_{jd} - z_{jd})^2$$

is small enough. out is the set of output units and t_{jd} and z_{jd} are the expected and practical output of the j-th output unit for sample d. In the new algorithm, a threshold value δ is set up before learning. For the error to be small enough, it would have to be smaller than δ. The testing samples are horizontally distributed and stored on all the data nodes.

If a network has been trained successfully, training will stop and this network will become the final result. The third party Q keeps a record of the number of iterations. If every network's number of iterations is larger than a threshold value M, training will also stop and the training will be deemed to have failed. In other circumstances, Q updates the network and selects a network randomly to begin the next iteration.

In this study, each data node calculates the error using its own local testing samples. A random number is added to this error to protect it. Then the third party Q obtains the error on the whole testing sample set by a secure sum protocol. In the same way, data nodes generate $k-1$ pseudo errors and transfer them to the third party Q. After that, Q will get k errors, only one of which is real. Q arranges them in randomly order and then transfers them to data nodes to calculate the differences between them and the threshold value δ. The signs of these differences will be given to Q, which will judge whether the error is small enough.

The detailed process is as follows. Data nodes divide δ randomly. Node P_i gets δ_i and $\sum_{i=1}^{n}\delta_i = \delta$. When getting the real network's weights, P_i calculates:

$$J_i = (1/2)\sum_{d\in Ti}\sum_{j\in out}(t_{jd} - z_{jd})^2$$

where, T_i is the set of testing samples on node P_i.

After that, P_i generates random number C_i. Then, the third party Q calculates $\Gamma_1 = \sum_{i=1}^{n}(J_i + C_i)$ by using a secure sum protocol. After that, P_i generates random numbers $J_{i1}, J_{i2},...,J_{i(k-1)}$ and Q calculates $\Gamma_2 = \sum_{i=1}^{n}J_{i1}, \Gamma_3 = \sum_{i=1}^{n}J_{i2}$ and $\Gamma_k = \sum_{i=1}^{n}J_{i(k-1)}$ by a secure sum protocol. π is a random permutation. Q uses π to arrange the order of $\Gamma_1, \Gamma_2,...\Gamma_k$. $\pi(\Gamma_1, \Gamma_2,...\Gamma_k)$ is denoted as $\Gamma'_1, \Gamma'_2,...\Gamma'_k$. Q divides each Γ'_j into n parts $\gamma_{j1}, \gamma_{j2},...\gamma_{jn}$, randomly. $\sum_{i=1}^{n}\gamma_{ji} = \Gamma'_j$. Q generates n vectors $E_1, E_2,...,E_n$, where $Ei = \{\gamma_{1i}, \gamma_{2i},...\gamma_{ki}\}$. Q gives E_i to P_i. P_i calculates $\gamma'_{ji} = \gamma_{ji} - \delta_i - C_i$, where $j = 1, 2,...,k$ and $i = 1, 2,...,n$. A data node P is selected randomly from data nodes. P calculates $\sum_{i=1}^{n}\gamma'_{ji}$ by using a secure sum protocol and gives $(\Gamma''_1, \Gamma''_2,...\Gamma''_k)$ to Q, where $\Gamma''_j = \mathrm{sgn}(\sum_{i=1}^{n}\gamma'_{ji})$. sgn() is the sign function: sgn() = 1, if $x>0$, sgn() = 0 if $x = 0$ and sgn() = -1 if $x<0$. If the permutation π turns Γ_1 into Γ'_j, Γ''_j is equal to $\mathrm{sgn}(J-\delta)$ and denotes whether the error is small enough. Q knows $(\Gamma''_1, \Gamma''_2,...\Gamma''_k)$ and π, so it can determine whether or not the network has been trained successfully.

In this process, Q gets the error plus a random number. Because Q does not know the random number, it also will not know the real value of the error and will not get information about testing samples. Q also gets the signs of the differences between errors and the threshold value δ. Getting the signs is useless to guess the private data. The data nodes can get k errors, including the real one, but they do not know which one is real. So, they also cannot get information that is helpful to guess detailed sample data on the other data nodes. In summary, the process of judging whether the network is trained successfully is secure.

The detailed process of optimization: The detailed process of optimization is as follows.

Q generates a d-dimensional vector NUM = (0, 0,..,0) to keep a record of the number of iterations for every network..

$P_1, P_2,..,P_n$ decide the threshold value δ. If the error $J \le \delta$, then the network is trained successfully.

$P_1, P_2,..,P_n$ decide constant k. When judging whether $J \le \delta$, data nodes generate $k-1$ pseudo errors.

```
      while Q does not issue the stop signal
        SelectNetwork ( ) // select a network randomly and
        recover the real weights.
        Error ( ) // calculate the network's error and judge if
        it is small enough.
         $\Delta$ denotes whether the error is small enough.
if       $\Delta$ = -1//The error is small enough.
      Q gives the stop signal

      The training has been successful. The training
network in this iteration is the final result.

else if NUM[i]$\ge$M for every i $\in$ {1, 2,...,d}
      Q gives the stop signal. The training has failed.
    else
      Update () // The network is updated.
    end if
  end if
end while
```

The SelectNetwork () function is used to select a network randomly to train. First, the third party selects a network randomly and transfers this choice to a data node. The ciphertexts of the related random numbers are also transferred to that data node. After that, the data node recovers the network's real weights and broadcasts them to other data nodes.

```
   SelectNetwork () {
      Q selects $c_1 \in \{1, 2,...,d\}$, letting
                              U = (X', E(A), E(B), t) = $U_{c1}$.
      P' is selected randomly from $\{P_1, P_2,..,P_n\} - \{P_t\}$.
      Q gives U to P'.
      P' gets A and B from E(A) and E(B) and calculates
      X = X'−A+B.
      P' broadcasts X to other data nodes.
}
```

The Error () function calculates a network's error on the testing set and judges whether it is smaller than a threshold value. Each data node calculates the network's error on its own local testing samples. Then, the third party gets both the real error and some pseudo errors. The third party arranges the order of these errors randomly and gives them to data nodes to judge whether each of them is smaller than the threshold value.

```
Error () {
    $P_1, P_2,..,P_n$ divide $\delta$ randomly. $P_i$ gets $\delta_i$. $\sum_{i=1}^{n}\delta_i = \delta$
    $P_i$ calculates $J_i$, that is X's error on $T_i$, which is the set of $P_i$'s local testing samples
    $P_i$ generates random numbers $C_i$ and $J_{ij}$, $j = 1, 2,...,k-1$
    Q calculates $\Gamma_1 = \sum_{i=1}^{n}(J_i + C_i), \Gamma_j = \sum_{i=1}^{n} J_{i(j-1)}$ , $j = 2, 3, \cdots, k$ , by a secure sum protocol
    Q arranges the order of $\Gamma_j$, $j = 1, 2,...,k$, by a random permutation $\pi$
    $\pi(\Gamma_1, \Gamma_2,...,\Gamma_k)$ is denoted as $\Gamma'_1, \Gamma'_2,...,\Gamma'_k$
    Q divides $\Gamma'_j$ to $\gamma_{j1}, \gamma_{j2},...\gamma_{jn}$ randomly. $\sum_{i=1}^{n}\gamma_{ji} = \Gamma'_j$. $j = 1, 2, \cdots, k$
    Q gives $Ei = (\gamma_{1i}, \gamma_{2i},...\gamma_{ki})$ to $P_i$
    $P_i$ calculates $\gamma'_{ji} = \gamma_{ji} - \delta_i - C_i$, where $j = 1, 2,...,k$ and $i = 1, 2,..,n$
    P calculates $\sum_{i=1}^{n}\gamma'_{ji}$ by a secure sum protocol, where P is selected randomly
    P gives $(\Gamma''_1, \Gamma''_2,...\Gamma''_k)$ to Q, where $\Gamma''_j = \mathrm{sgn}(\sum_{i=1}^{n}\gamma'_{ji})$
    Q knows $\pi$, so it can get $\Delta = \mathrm{sgn}(J - \delta)$
}
```

The Update () function updates the currently trained network. In this function, the data nodes calculate the network's update and randomize it. The randomized update, the randomized network's weights and the ciphertexts of related random numbers are given to the third party. The third party will then update that network.

```
  Update () {
    $P_i$ calculates $Y_i$, X's update on $D_i$, where $i = 1, 2,..,n$. $D_i$ is the set of training samples on data node $P_i$. X is the weight vector of the currently trained network.

    $P_i$ generates random vector $B_i$ with the same dimension as X
    Q calculates $\sum_{i=1}^{n}(Y_i + B_i)$ by a secure sum protocol
    $P_l$ is selected from $P_1, P_2,...P_n$ randomly
    $P_1, P_2...P_n$ calculate $B = \sum_{k=1}^{n} B_k$ using a secure sum protocol and $P_l$ gets the result B
    $P_l$ gives its index $l$ and E(B) to Q
    $P_l$ generates random vector $A_l$ with the same dimension as X and calculates $X_l = X + A_l$
    $P_l$ gives $E(A_l)$ and $X_l$ to Q
    Q calculates $X_l' = X_l - \sum_{i=1}^{n}(Y_i + B_i)$
    Q replaces $U_{c1}$, which is the currently trained network, with $(X'_l, E(A_l), E(B), l)$
    $NUM[c_l] = NUM[c_l]+1$
}
```

ALGORITHMI ANALYSIS

Privacy preserving data mining has three requirements: efficiency, accuracy and security. Efficiency means that the algorithm should have low computational and communication complexity. Accuracy means that the algorithm can produce a good result. That is to say, the privacy preserving method should produce similar results to the general data mining method that does not consider privacy preservation. Security means that after running the algorithm, no node can get the data on other nodes.

Efficiency: Suppose that there are n data nodes, that d networks are trained, that each network has m weights, that the time complexity of learning network by the standard BP algorithm on global data without considering privacy protecting is O(F), that encrypting and decrypting a number take O(E) and O(G) time, that the computation and communication complexity of calculating a secure sum for a number on all nodes are O(S) and O(W) and that M is a threshold value for the largest iteration times.

In initialization, the loop will run d times. In every loop, each node generates constant m-dimensional vectors and does addition and encryption of them. All nodes can work concurrently. This will take O(dmE) time.

In initialization, every node will transmit constant m-dimensional vectors in each loop. So, the communication complexity of initialization is O(dnm).

In optimization, it is assumed that the iteration runs c times. C is bigger than dM and is not a definite value. Considering concurrent computation, on each iteration, the algorithm generates, encrypts and decrypts constant m-dimensional vectors and does addition between m-dimensional vectors a constant number of times. On each iteration, the algorithm runs the secure sum protocol for m-dimensional vectors a constant number of times. These operations take O(m(G+S+E)) time. The algorithm also needs to calculate the error and weights' update. In the worst case, this takes O(cF/M) operations. In the worst case, the samples concentrate on one data node, so there will be no concurrent computation. Each iteration takes as much time as the standard BP algorithm. O(F) is also the worst case for the standard BP algorithm, which means that the training has failed and the number of iterations is M. Thus, one iteration takes O(F/M) time and our algorithm, which has c iterations, takes O(cF/M) operations. To summarize, the time complexity of optimization is:

$$O(c(m(G+S+E)+(F/M)))$$

On each iteration, the algorithms need to transmit O(n) m-dimensional vectors and run the secure sum protocol for m-dimensional vectors a constant number of times. So, the communication complexity of optimization is O(c(nm+W)).

To summarize, the total time complexity is:

$$O(c(m(G+S+E)+(F/M)))$$

The total communication complexity is O(c(nm+W)). Because the expected value of c is O(dM) (M>>d), the expectation of the total computation complexity is:

$$O(dMm(G+S+E)+dF)$$

The expectation of the total communication complexity is:

$$O(dM(nm+W))$$

The number of data nodes, samples and networks are the three main factors that influence the time and traffic cost. Of these three factors, the number of networks is a parameter set by the user. The more networks desired, the more time and traffic cost, but the more difficult it will be for nodes to guess the samples. If there are d networks, each node only has 1/d probability of successfully guessing which network is training.

Using experiments, we show how the time and traffic cost change when these three factors change. The experiments use the data from (Rakesh and Ramakrishnan, 2000). These data have 9 attributes and 5 class functions and experiments are performed on all 5 class functions. For each function, 20 networks are trained. The time and traffic cost are then averaged.

Experiment 1 uses 5,000 samples, of which 5% are testing samples. The number of networks is 5. All the samples are distributed evenly and randomly on data nodes. Figure 4a and b show the result of this experiment, showing the time and traffic cost when the number of data nodes changes. The number of data nodes does not appear in the computational complexity, because the computational complexity corresponds to the worst case. However, it is always an important factor that influences the time and traffic cost.

The experiment shows that when the number of data nodes increases, the learning process will take less time but generate more traffic.

Less time is used because for some work, for example, calculating the weights' update and error, the load only depends on the number of samples and the cost is thus split among all the nodes. In present experiment, the samples are distributed evenly. When the number of data nodes increases, there are more nodes to share in the work evenly, so the time cost is decreased. It is can be seen from Fig. 4 that with ever-increasing number of nodes, the decrease in time cost slows down. This is because some of the work cannot be divided among the nodes. Consistent with Amdahl's Law, this study takes most of the time when the number of nodes becomes big enough. More nodes also cause more communication, so the traffic cost will increase with the number of nodes.

Experiment 2 uses 8 nodes and 5 networks. Figure 5a and b show the results of this experiment, showing the time and traffic cost when the number of samples changes. All the samples are distributed evenly and randomly among the data nodes and 5% of samples are testing samples.

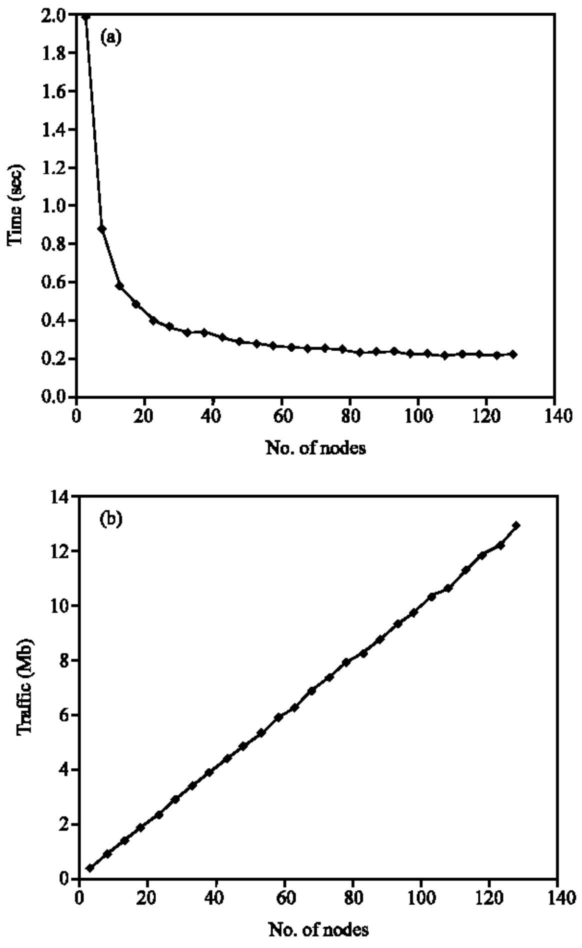

Fig. 4: (a) The time and (b) traffic cost with different number of nodes

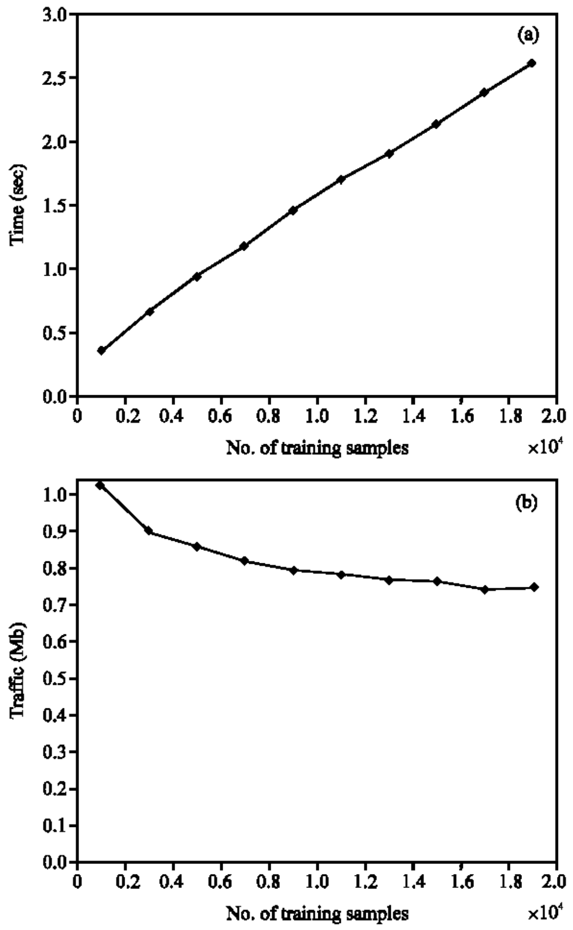

Fig. 5: (a) The time and (b) traffic cost with different number of samples

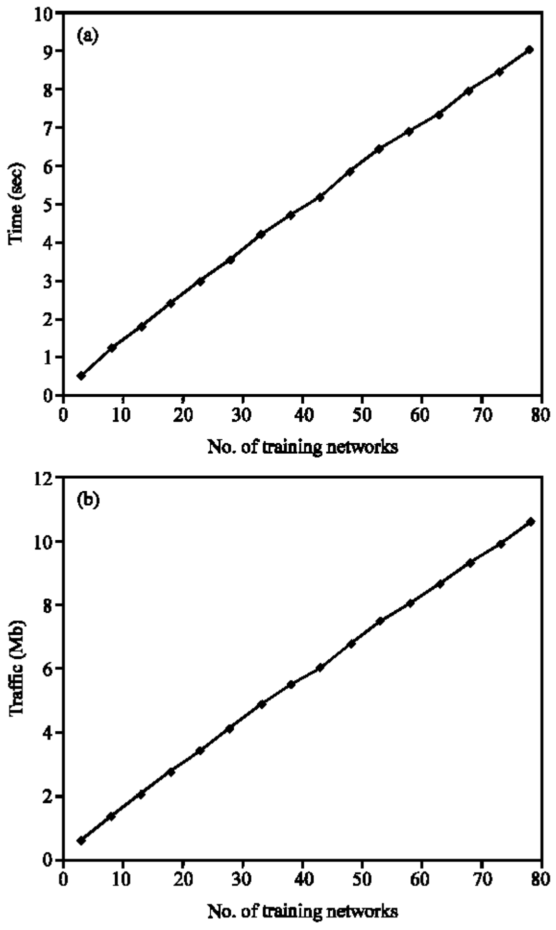

Fig. 6: (a) The time and (b) traffic cost with different number of networks

The experiment shows that when the number of samples increases, the learning process will cost more time and generate less traffic.

The more samples we have, the greater the load of calculating weights' update and network's error. Thus, the algorithm will take more time. However, more samples also make for a smaller number of iterations, because more optimization will be done on each iteration. This causes less traffic cost. Fewer iterations also requires less time, but the increasing load caused by increasing samples have greater effect.

Experiment 3 uses 5,000 samples, of which 5% are testing samples. The number of data nodes is 8. Figure 6a and b show the results of this experiment, giving the time and traffic cost when the number of networks changes. All the samples are distributed evenly and randomly on data nodes.

The experiment shows that when the number of networks increases, the learning process will take more time and generate more traffic.

The more networks there are, the more work should be done, so the algorithm will require more time and generate more traffic.

Accuracy: If the result obtained by the privacy preserving algorithm is similar to the result obtained by the general algorithm, then we can say the algorithm has sufficient accuracy.

The essence of present algorithm is exchanging information that the BP algorithm needs, in privacy. The learning process is the same as the BP algorithm. Running our algorithm on the horizontally distributed database is equivalent to running the standard BP algorithm on global data without privacy considerations. Given the same argument, they will obtain the same result. Thus, present algorithm is accurate.

Security: The semi-trusted third party can obtain six kinds of data. They are as follows:

- The network's weights plus random numbers
- The weights' update plus random numbers
- The network's error plus random numbers
- The ciphertexts of the random numbers that are added to the network's weights or weights' update
- The pseudo errors
- Some data nodes' indices and the signs of the differences between errors and the threshold value

In these six kinds of data, the first three kinds are all masked by the addition of a random number and these random numbers change in every loop. The ciphertexts of these random numbers are the fourth kind of data. With a proper encryption method, the third party cannot obtain the random numbers from their ciphertexts, so it will not be able to determine the real value of the first three kinds of data. The last two kinds are secure data. That means that they do not contain information about the detailed values of samples. To sum up, the semi-trusted third party cannot discover the detailed value of samples.

Every data node receives three kinds of data. They are:

- The training network's weights
- The information received in the secure sum protocol
- The shared values of errors

In these three kinds of data, the network's weights are the riskiest. But it is the third party, who decides which network is trained in every loop and the data nodes cannot determine which network is currently being trained. Data nodes cannot link together the information from different iterations. They cannot set up equations about samples on other data nodes. Thus, the first kind of data cannot help data nodes determine the detailed data on other nodes. With the right secure sum protocol, the second kind of data also remains secure for privacy. The shared values of errors are divided randomly by the third party. Data nodes cannot know the value of errors from the third kind of data. Thus, the third kind of data is secure for privacy.

In addition, some special data nodes can get other information.

The data node that is selected to recover the real network's weights can get the random numbers related to the trained network. Also, these random numbers are generated or calculated by a data node that is selected as the master node in that network's last update. In present algorithm, we make sure that these two nodes are different. Thus, they cannot use this special data to know which network is currently being trained and this special data remains secure for privacy.

The data node that calculates the signs of differences between errors and the threshold value can obtain the values of these differences. However, only one of the errors is a real error and the order of the errors is arranged by the third party randomly. Therefore, this node cannot determine the real error and so this special data also remains secure for privacy.

To summarize, this algorithm ensures that the privacy cannot be leaked. Neither the semi-trusted third party nor the data nodes can obtain the data on the other nodes through the learning process.

CONCLUSIONS

Privacy Preserving Data Mining (PPDM) is a hot topic in data mining research today. An important task of PPDM is to do mining when the original private data cannot be seen directly. The perceptron neural network is an important data mining method. However, to the best of our knowledge, there is currently not a good enough privacy preserving learning algorithm for it. Thus, the perceptron neural network method cannot yet be used on private data in the real world. To solve this problem, this study brings forward a privacy preserving BP learning algorithm for horizontally distributed databases. In this algorithm, the information exchange method is designed to let nodes exchange information that the BP algorithm needs, while maintaining privacy. This algorithm can obtain the same result as the standard BP learning algorithm on the whole data set without privacy considerations, while the algorithm maintains the property that no node can obtain the detailed data on other nodes through the learning process. The price of this privacy is additional time and traffic cost.

ACKNOWLEDGMENTS

The authors would like to thank National High-Tech Research and Development Plan of China, under Grant No. 2007AA02Z329 and 2007AA02Z302, for funding this research.

REFERENCES

Artak, A. and E.C. Vladimir, 2007. The privacy of k-NN retrieval for horizontal partitioned data: New methods and applications. Proceedings of the 18th Conference on Australasian Database, Jan. 29-Feb. 2, Australian Computer Society, Ballarat, Victoria, Australia, pp: 33-42.

Barni, M., C. Orlandi and A. Piva, 2006. A privacy-preserving protocol for neural-network-based computation. Proceedings of the 8th Workshop on Multimedia and Security, Sept. 26-27, ACM Press, Geneva, Switzerland, pp: 146-151.

Chris, C., K. Murat, V. Jaideep, L. Xiadong and Y.Z. Michael, 2002. Tools for privacy preserving distributed data mining. ACM SIGKDD Explorations Newslett., 4: 28-34.

Elisa, B., N.F. Igor and P.P. Loredana, 2005. A framework for evaluating privacy preserving data mining algorithms. Data Mining Knowledge Discovery, 11: 121-154.

Emekci, O.F., D. Sahin, D. Agrawal and A. El Abbadi, 2007. Privacy preserving decision tree learning over multiple parties. Data Knowledge Eng., 63: 348-361.

Jaideep, V., Y. Hwanjo and J. Xiaoqian, 2008. Privacy-preserving SVM classification. Knowledge Inform. Syst., 14: 161-178.

Jimmy, S., G.. Michael and C. Jose, 2007. A privacy preserving probabilistic neural network for horizontally partitioned databases. Proceedings of the 2007 International Joint Conference on Neural Networks, Aug. 12-17, Orlando, Florida, USA., pp: 1554-1559.

Justin, Z., 2007. Using homomorphic encryption for privacy-preserving collaborative decision tree classification. Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Data Mining, April 1-5, Honolulu, Hawaii, pp: 637-645.

Li, W., K.N. Wee, H. Shuguo and C.S.L. Vincent, 2007. Privacy-preservation for gradient descent methods. Proceedings of the 13th ACM SIGKDD International Conference and Knowledge Discovery and Data Mining, Aug. 12-15, ACM Press, San Jose, California, pp: 775-783.

Mark, S., K. Yongdae and K. Vipin, 2006. Privacy preserving nearest neighbor search. Proceedings of the 6th IEEE International Conference on Data Mining-Workshops, Dec. 18-22, Hong Kong, China, pp: 1-545.

Rakesh, A. and S. Ramakrishnan, 2000. Privacy-preserving data mining. ACM SIGMOD Record, 29: 439-450.

Saeed, S. and M. Ali, 2008. Privacy-preserving protocols for perceptron learning algorithm in neural networks. Proceedings of the 4th International IEEE Conference on Intelligent Systems, Sept. 6-8, Golden Sands Resort, Varna, Bulgaria, pp: 1065-1070.

Sebastien, G., K. Balazs and A. Esma, 2007. Privacy-preserving boosting. Data Mining Knowledge Discovery, 14: 131-170.

Stanley, R.M.O. and R.Z. Osmar, 2004. Toward standardization in privacy preserving data mining. Proceedings of the 3rd Workshop on Data Mining Standards, WDMS'2004, USA., pp: 7-17.

Sven, L., L. Helger and M. Taneli, 2006. Cryptographically private support vector machines. Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Aug. 20-23, Philadelphia, USA. ACM Press, pp: 618-624.

Tom, M.M., 2003. Machine Leaning. China Machine Press, Beijing, China.

Vassilios, S.V., B. Elisa, N.F. Igor, P.P. Loredana, S. Yucel and T. Yannis, 2004. State-of-the-art in privacy preserving data mining. SIGMOD Record, 33: 50-57.

Yancheng, C. and L. Chijen, 2005. Oblivious polynomial evaluation and oblivious neural learning. Theor. Comput. Sci., 341: 39-54.

Zhiqiang, Y. and N.W. Rebecca, 2006. Privacy-preserving computation of bayesian networks on vertically partitioned data. IEEE Trans. Knowledge Data Engin-Eering., 18: 1253-1264.