Computing and Informatics, Vol. 27, 2008, 355–376

LEARNING K-NEAREST NEIGHBORS CLASSIFIER FROM DISTRIBUTED DATA

Ahmed M. Khedr

Mathematics Department, Faculty of Science, Zagazig University, Egypt
e-mail: amkhedr@zu.edu.eg

Manuscript received 13 November 2006; revised 2 May 2007
Communicated by Jaroslav Pokorný

Abstract. Most learning algorithms assume that all the relevant data are available on a single computer site. In the emerging networked environments learning tasks are encountering situations in which the relevant data exists in a number of geographically distributed databases that are connected by communication networks. These databases cannot be moved to other network sites due to security, size, privacy, or data-ownership considerations. In this paper we show how a k-nearest classifier algorithm can be adapted for distributed data situations. The objective of our algorithms is to achieve the learning objectives for any data distribution encountered across the network by exchanging local summaries among the participating nodes.

Keywords: Learning, k-classifier, decomposable algorithms, vertically and horizontally distributed data

Mathematics Subject Classification 2000: 68T05

1 INTRODUCTION

Recent advances in computing and communications have made it possible to gather and store large volumes of data in digital form. However, it is difficult to work with the information obtained from a different scientific community. These difficulties arise because of the large volume of information that would need to be moved around or because of the constraints imposed by the autonomy of the data collected by a particular institution (e.g., privacy constraints). Thus, the current technology

is not sufficient for the need of collaborative and interdisciplinary *e-Science*, but fortunately, new technologies are emerging with the potential to revolutionize the ability of scientists to do collaborative work, for example [5].

Privacy-preserving data mining becomes an important enabling technology for mining data from multiple private databases provided by different and possibly competing organizations. For example, many insurance companies collect data on disease incidents, seriousness of the disease and patient background. One way for the Center for Disease Control to identify disease outbreaks is to train a classifier across the data held by the various insurance companies for patterns that are indicative of disease outbreaks and use it to classify a query pattern as an outbreak or the opposite. However, commercial and legal reasons prevent the insurance companies from revealing their data. It is important and beneficial to have a distributed data mining algorithm that is capable of identifying potential outbreaks while respecting the privacy requirements of its participants.

Distributed Data Sources. A number of geographically distributed databases together form an implicitly specified global dataset that contains all the data relevant for a computation. For example, some pattern discovery tasks may require simultaneous consideration of data, parts of which reside in census databases, labor statistics databases, and employment related databases. Each of these is a huge database and resides on a different site in a different city. Consequently, it is neither desirable nor feasible to gather all of the data in a centralized location for analysis. Hence, there is a need for knowledge acquisition systems that can perform the necessary analysis of data at the locations where the data and the computational resources are available and transmit the results of analysis to the locations where they are needed [2, 9, 26].

Distributed Learning. The problem of learning from distributed data sets can be accomplished by an agent that visits the different sites to gather the information needed to generate a suitable model from the data. Alternatively, the different sites can transmit the information necessary to the learning agent situated at a *Learner* location. Consequently, the *Learner* has to rely on information (e.g., statistical summaries) extracted from the sites.

Contributions. We present a methodology for constructing a k-NN classifier across multiple private distributed databases. This methodology consists of a general model for decomposable k-NN classification and a set of algorithms for realizing this model. Our approach to learn from distributed data sets involves identifying the information requirements of existing learning algorithms, and designing efficient means of providing the necessary information to the *Learner*, while avoiding the need to transmit large quantities of data.

2 INTEGRATION OF DISTRIBUTED DATA

We consider a situation in which there are n sites of a network and each site i contains a local database, named D_i. There can be arbitrary overlaps in the schema

for these databases, that is, any two databases may share a number of attributes, and all of them together constitute the dataset $\mathcal{D}$. The set of attributes contained in D_i is represented by X_i. For any pair of relations, (D_i and D_j), the corresponding sets X_i and X_j may have a set of shared attributes given by S_{ij}.

The implicit data set $\mathcal{D}$ with which the computation is to be performed is the set of tuples generated by a *Join* operation performed on all the participating relations $D_1, D_2, \ldots, D_n$. However, the tuples of $\mathcal{D}$ cannot be made explicit at any one network site by any site. The tuples of $\mathcal{D}$, therefore, must remain implicitly specified only to one agent. This inability of an agent to make explicit the tuples of $\mathcal{D}$ is the main problem addressed in the generalized decomposition of global algorithms and is discussed in later sections. To facilitate computations with implicitly specified sets of tuples of $\mathcal{D}$, we define a set (S) that is the union of all the attribute intersection sets (S_{ij}), that is,

$$S = \bigcup_{i,j,i \neq j} S_{ij}. \tag{1}$$

The set S, thus contains the names of all those attributes that are visible to more than one agent because they occur in more than one participating D_i. We define a relation (*Shared*) containing all possible enumerations for the attributes in the set S.

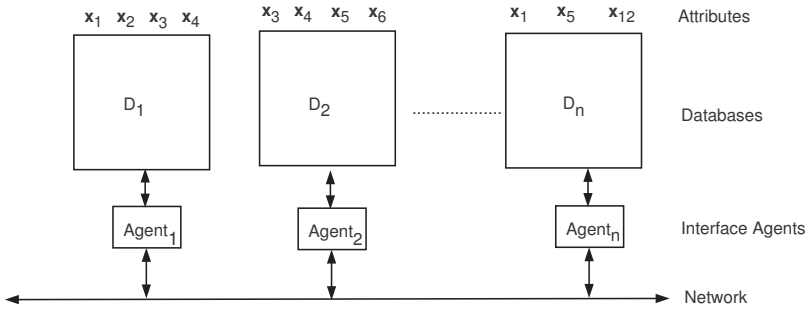

Fig. 1. Distributed data/knowledge sources

2.1 Nature of Data Distribution

There are two primary ways in which the databases, together, may be seen as forming an implicit global dataset $\mathcal{D}$.

Horizontal Partitioned. A dataset $\mathcal{D}$ is partitioned into a set of databases $D_1, D_2, \ldots, D_n$ each of which have the same set of attributes X_i, and a subset of tuples of the original dataset $\mathcal{D}$. Each tuple of $\mathcal{D}$ is in exactly one database. The set of shared attributes S is the same as X_i for each database. The union of all databases D_i constitutes the complete dataset $\mathcal{D}$, i.e., $D_1 \cup D_2 \cup \ldots \cup D_n = \mathcal{D}$.

Vertical Partitioned. A dataset $\mathcal{D}$ is partitioned into a set of databases $D_1, D_2, \ldots, D_n$ where each database contains a subset of the attributes of the original dataset $\mathcal{D}$. In this case each component D_i, may share some attributes with other databases D_j, $j \neq i$. Each D_i may also contain some attributes not shared with any other database.

Vertically partitioned datasets are of more interest because they provide an opportunity to share knowledge across the participating nodes.

2.2 Stationary and Mobile Agents

We define *Agent* as a piece of software which performs a given task using information gleaned from its environment to act in a suitable manner so as to complete the task successfully. An agent should be able to adapt itself based on changes occurring in its environment, so that a change in circumstances will still yield the intended result. Agents are used to represent actors in a cooperative effort, and give the users of the agent system support for doing efficient negotiation, and exchange of data.

Each database D_i is represented by an agent acting on behalf of it with some degree of autonomy. This agent knows all about its underlying database and can access any part of it. Each agent also can know about other participating sites. It then determines the local computations that it needs to perform, keeping in mind the constraints of shared data with other sites; and also the local results that it needs to share with other agents in order for the global result to evolve at either one or each of participating agent. An alternative to communicating with agents at other site is that a single agent visits each of the participating sites and performs some local computation at each site when it visits.

Agents may be classified by their *mobility* i.e., by their ability to move around some network. This yields the classes of *stationary* agents as shown in Figure 1; or *mobile* agents as shown in Figure 2.

Stationary Agents. Stationary agents that stay at their respective data sites perform the local computations and send them to a *Learner* agent who applies the aggregation operation to all the local results. They interact with other similar agents at other sites for exchanging (sending/receiving) simple computational summaries. They use message passing as a communication mechanism. Each message is generally of a very small length, but the number of messages may grow very fast.

Mobile Agents. The mobile agent is free to migrate during execution from one site to another, to perform local computation at each site that they visit, and at the end to compose the gathered results for performing the global computations. Mobile agent aggregates two things: data (data collected and process states) and code (instructions that direct the behavior).

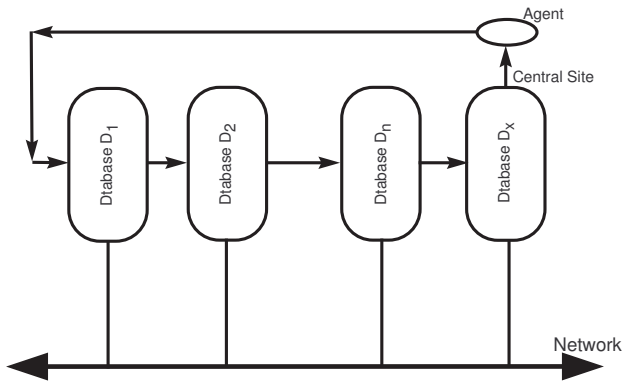

Fig. 2. Mobile agent

2.3 Agent's Decomposition Task

The objective of a *Learner* agent is to perform the global computation by communicating with other similar agents at other sites, where each agent performs some computation with its local database in the context of and as constrained by the sharing of attributes across the participating agents (see Figure 1).

Our general strategy for transforming a batch learning k-nearest classifier algorithm into a distributed learning algorithm involves identifying the information requirements of the algorithm and designing efficient means for providing the needed information to the learning agent while avoiding the need to transmit large amounts of data.

Suppose we decompose a batch learning algorithm L in terms of an information extraction operator I that extracts the necessary information from data set and a hypothesis generation operator H that uses the extracted information to produce the output of the learning algorithm L. That is,

$$L(D) = H(I(D)). \tag{2}$$

We define a distributed information extraction operator Id_i that generates the corresponding information $Id_i(D_i)$, from the data set D_i, and an operator G aggregates this information from all sites to produce $I(D)$. That is, the information extracted from the distributed data sets is the same as that used by L to infer a hypothesis from the complete dataset $\mathcal{D}$. Thus,

$$G[Id_1(D_1, S), Id_2(D_2, S), \ldots, Id_n(D_n, S)] = I(D). \tag{3}$$

Thus, we can guarantee that

$$L(D) = H(I(D)) = H(G[Id_1(D_1, S), Id_2(D_2, S), \ldots, Id_n(D_n, S)]). \tag{4}$$

The set S of shared attributes determines what explicit $\mathcal{D}$ would be generated by the individual data components. An implementation of G in Equation (3), for some S, can be engineered by a functionally equivalent formulation. That is, a local computation $Id_i(D_i, S)$ is performed by agent $agent_i$ using the database D_i and the knowledge about the attributes shared among all the data sites (S). The results of these local computations are aggregated by an agent using the operation G.

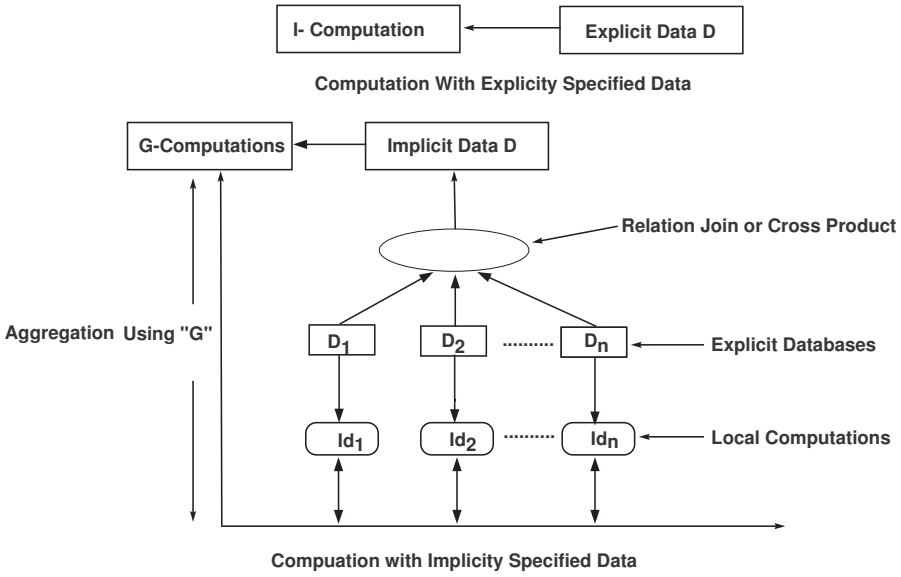

Fig. 3. Computations in explicit vs. implicit data spaces

The schematic in Figure 3 shows the process by which the agent would compute Id_i from the D_is. The component operators of a decomposition (G and Id_is), therefore, need to be dynamically determined by the agent for each instance of $L(D)$ depending on the participating nodes; the attributes contained in their native databases; and the sharing pattern of attributes.

2.4 Cost Model for Algorithmic Complexity

Traditionally, the complexity of algorithms has been measured in terms of the CPU time and the required memory. This cost model is well-suited for computations on a single computer and the closely-coupled processors model. When a number of loosely networked nodes are involved in a cooperative computation the communication cost becomes the overwhelmingly dominant component of the total cost. Complexity for distributed query processing in databases has been discussed in [25]. In our experience with the design and analysis of decomposable network algorithms,

we have found that each step of the algorithm must exchange a number of messages for evaluating the various quantitative values. Each message is generally of a very small length, but the number of messages may grow very fast. Here and in other similar works [9, 10] we have used cost models involving the number of messages exchanged and reflecting the efficiency of decomposition carried out by the network algorithm. We choose the following two cases for analyzing the complexity of our algorithm.

Stationary Agents Case. We choose the following two cost models for analyzing the complexity of our algorithm. In these cost models we count the number of messages that must be exchanged among all the participating sites in order to complete the execution of the algorithm.

- **Exchanging One Summary per Message (Unoptimized).** One message exchange includes only one local computation request at a time. The messages are exchanged in a sequential manner, that is, one site is asked for one local computation, the corresponding summary is obtained, and then the request is sent to the next participating database.
- **Exchanging All Summaries per Message (Optimized).** One message exchange includes all local computation requests which correspond to all tuples of S and receive all corresponding summaries in one message.

Mobile Agent Case. In this case, submission of a task by a user; dispatching of mobile agent to data sites; and its return with global results. In this case, the complexity can be measured in term of the number of agent hops from one site to another or in term of the number of agent visits to each site. The number of hops is independent of the size of implicit dataset $\mathcal{D}$.

3 RELATED RESEARCH

Most of the learning algorithms in the literature assume that all the relevant data are available in a single computer site. In the context of database research much work has been done towards learning from distributed databases. Among these, most of the existent algorithms work for horizontal data distributions, with a few exceptions. Many of the approaches to distributed learning come from the desire to scale up algorithms to large data sets [17, 18]. Conceptually there is a big difference between approaches to distributed learning coming from scaling up algorithms, where the data are distributed by the algorithm in order to increase the overall efficiency, and approaches that assume that data are inherently distributed and autonomous, and thus restrictions and constraints may need to be taken into account. The work in this paper falls into the second category.

Ensembles Approach to Distributed Learning. Several distributed learning algorithms have their roots in ensemble methods. [3, 16] used an ensemble of classifiers approaches to learn from horizontally distributed data, which involves

learning separate classifiers from each dataset, and combining them typically using a weighted voting scheme. In general, this combination requires gathering a subset of data from each of the data sources at a *Learner* site to determine the weights to be assigned to the individual hypotheses which is not desirable. Besides the need to transmit some subset of data to the *Learner* site, there are other potential drawbacks such that the resulting ensemble of classifiers is typically much harder to comprehend than a single classifier, and the lack of guarantees concerning generalization accuracy of the resulting hypothesis relative to the hypothesis obtained in the centralized setting.

Cooperation-based Distributed Learning. Although learning with cooperation scenarios could be very often met in real world situations, there are not many distributed learning algorithms that use the cooperation in an active way to obtain the final result, with a few notable exceptions. [19] proposed a powerful, yet practical distributed rule learning (DRL) algorithm using cooperation. They make use of several criteria to estimate the probability that a rule is correct. In [11], the authors proposed an algorithm for learning to share distributed probabilistic beliefs. As opposed to collaboration by exchanging models between learners, in [22] data could be moved from one site to another in order to fully exploit the resources of the network. One practical example of a learning algorithm that uses cooperation to exchange data is described in [7].

Learning from Vertically Distributed Data. Although most of the distributed learning algorithms assume horizontal data distribution, there are a few notable exceptions. [2] proposed algorithms for learning decision tree classifiers from vertically distributed data. [1] is a collaborative approach to concept learning from vertically distributed data. It works by computing the cardinal distribution of feature values in the individual data sets, followed by propagation of this distribution across different sites. Features with strong correlations to the concept to be learned are identified based on the first order statistical approximation to the cardinal distribution. Being based on first order approximations, this approach is impractical for problems where higher order statistics are needed. [21] proposed an ensemble approach to combine local classifiers. They used an order statistics-based technique for combining high variance models generated from heterogeneous sites. [15] observed that inter-site patterns cannot be captured by aggregating heterogeneous classifiers. To deal with this problem, at each site, they constructed a subset of the data that a particular classifier cannot classify with high confidence and ship such subsets of data at the *Learner* site, where a classifier is built. Although this approach gives better results than simply aggregating the classifiers, it requires data shipping and its performance is sensitive to the sample size.

Privacy Preserving Distributed Data Mining. Several approaches of distributed data mining appeared from the need to preserve the privacy of the information that is mined. In [12] the summaries of the data need to be used instead of raw data. Some work has focused on specific algorithms design in the

presence of privacy constraints: [4] introduced an algorithm for building decision trees on private data; [6, 20] dealt with privacy-preserving distributed mining of association rules from horizontally partitioned data, while [23] proposed an algorithm that works when data are vertically partitioned; [8] proposed an algorithm for computing correlations in a vertically distributed scenario while preserving privacy; [13, 14] presented algorithms for privacy preserving clustering using EM mixture modelling and generative models from horizontally distributed data, while [24] proposed a k-Means clustering over vertically partitioned data. In [26] the authors presented an algorithm for determining k-nearest neighbor tuples for a given tuple in a set of geographically distributed databases. These databases form a vertical partitioning of some implicit global database. The computation is performed by exchanging minimum number of higher level summaries.

Central to our approach is a clear separation of concerns between hypothesis construction and extraction of sufficient statistics from data. This separation makes it possible to explore the use of sophisticated techniques for query optimization that yield optimal plans for gathering sufficient statistics from distributed data sources under a specified set of constraints describing the query capabilities and operations permitted by the data sources.

4 DISTRIBUTED K-NEAREST NEIGHBORS CLASSIFIER

The k-nearest neighbors classifier (k-NNC) simply consists of two main phases (learning and classification phases). The learning phase consists of storing the data, and the information extraction is done during the classification phase. It is a simple example of instance-based learning. In the k-NNC algorithm, the nearest neighbors are defined in terms of a metric distance $d(.,.)$ between instances. If we assume that $y_t = \langle b_{1t}, \ldots, b_{dt} \rangle$, $t \in (1, 2, \ldots, l)$, and $C = \{c_1, c_2, \ldots, c_m\}$ be the set of instances, and class labels, and $D = \{(y_t, c_j) : t \in (1, 2, \ldots, n), j \in (1, 2, \ldots, m)\}$ is the set of training examples. Then the class label for a new instance $x = \langle a_1, a_2, \ldots, a_d \rangle$ is given by the most common class label among the k training examples nearest to x.

4.1 Sufficient Statistics for k-NNC

Given a new instance x to be classified, the sufficient statistics with respect to $h(x)$ ($h(x) = arg\ Max_{c \in C} \sum_{j=1}^{k} \delta(c, y_{t_j})$ where $\delta(a, b) = 1$ if $a = b$ and $\delta(a, b) = 0$ otherwise) consist of the k nearest neighbors of the new example x. Given the k nearest neighbors, the class of the new instance is determined by taking a majority vote among those examples, independent of the rest of the data. Therefore, the minimal sufficient statistics are given by the smallest k distances, and the corresponding classes. In what follows we will show how the minimal sufficient statistics can be computed, when data are horizontally and vertically distributed.

4.2 Algorithm for Horizontally Distributed Data

In the horizontally distributed setting, we compute the minimal sufficient statistics by computing the k nearest neighbors at each site i and ship the class labels corresponding to these neighbors together with the distance from each of them to the new instance x. Thus, we ship pairs $(d(x, y^i_{t_j}),\ c(y^i_{t_j}))$ for every nearest neighbor $y^i_{t_j}$ at the site i (we denote by $d^i_{t_j} = d(x, y^i_{t_j})$ the distance between x and $y^i_{t_j}$ according to the metric used by the algorithm and by $c^i_{t_j} = C(y^i_{t_j})$ the class $c^i_{t_j}$ of the example $y^i_{t_j}$. At the *Learner* site, we determine the k smallest distances among all the distances received and take a majority vote among the classes associated with those instances. The majority class will be the class of the new instance x. The pseudocode for horizontally distributed data is shown below:

At Local Sites:

1. Given a new instance x= (a_1,a_2, ..., a_d) to be classified:
2. Send x to each site i.
3. For (each data source D_i)
   (a) Find the distance between x and every tuple in D_i
   (b) Return to the *Learner* site the minimum k distances d^i_t and the corresponding class c^i_t, i.e., return the pair (d^i_t, c^i_t).

At *Learner* Site:

1. Compute the k nearest distances among all the received distances.
2. From the returned k pairs, classify the instance x to the class with maximum number of cases.

4.2.1 Complexity Computing

We show below an expression for the number of messages that need to be exchanged among the stationary and mobile agents for executing the horizontally k-nearest neighbor classifier.

Stationary Agents Implementation

One Summary Per Message (Un-Optimized). In this case the messages are exchanged in a sequential manner, that is, one site is asked for its k distances, an answer is obtained, and then the request is sent to the next participating database. We transfer only one $Id_i(D_i, S)$ summary at a time. The complexity will be n exchanged messages. The result shows that the number of messages that need to be exchanged among the sites is not dependent on the size of the database at each site. This is significant because it shows that as the sizes of the individual databases grow, the communication complexity of our algorithm would remain unaffected.

Exchanging All the Summaries in One Message (Optimized). It is possible to send a request to all agents for $Id_i(D_i, S)$ values in one request and receive all the summaries in one message. This reduces the number of messages exchanged to be only one message.

The trade-off between the two approaches is that the first one may be considered more secure for transmission over a network because each message contains only little information about the participating databases. The second alternative requires very few messages but each message contains more information about each database.

Mobile Agents Implementation. In the case of mobile agents, during a visit to a data site, it can compute the local Id_i for that site. Once all the sites have been visited, the G aggregator can be applied to the collected local results from all the sites. If there are n participating sites, the minimum k distances can be gathered during a single visit to each site. Thus, the computation of the distances can be done with one pass through the data.

On the other hand, if we move all the data to one site and then run k-NNC algorithm, the total complexity will be the communication time of moving all the data to one site. The computational resources will be needed to concatenate the data tables plus the complexity of running k-NNC algorithm with $n * N$ tuples, where N is the average number of tuples at each site.

Assertion 1. The algorithm for learning k-NNC from horizontally distributed data returns the same results with respect to the algorithm for learning k-NNC from centralized data.

Proof. If $\min_k(T)$ returns the smallest k distances in a set of distances T corresponding to the set of examples $\mathcal{D}$, and a new instance x to be classified, $\min_k(T_i)$ returns the smallest k distances in a set of distances T_i corresponding to the set of examples D_i and an instance x to be classified. Since $\mathcal{D} = D_1 \bigcup D_2 \bigcup \ldots \bigcup D_n$, from the observation

$$\min_k(T) = \min_k(\min_k(T_1) \bigcup \min_k(T_2) \bigcup \ldots \bigcup \min_k(T_n)). \tag{5}$$

The set of class labels used for majority vote in the distributed case is the same as the set of class labels used for majority vote in the centralized case. □

4.3 Algorithm for Vertically Distributed Data

To compute the minimal sufficient statistics in the vertically distributed setting, we compute the k-nearest neighbors and the corresponding class labels from a new instance x by calling the distributed k nearest neighbors (DkNN) algorithm. The k-NNC pseudocode for vertically distributed data is shown below:

At Local Sites:

1. Given a new instance x to be classified.
2. Call DkNN(x, $(D_1, D_2, \ldots, D_n)$) to compute the k-nearest neighbors from vertically distributed data.

At *Learner* Site:

1. From the returned k pairs ⟨distance, class⟩, classify the instance x to the class with maximum number of cases.
2. End Algorithm.

The nearest k neighbors will be determined by (DkNN) algorithm and then at the *Learner* site the instance x will be classified into the class with maximum number of cases.

In the following subsection, we introduce the DkNN algorithm for finding the nearest k points from a given one in vertically distributed databases.

4.3.1 The Distributed k-Nearest Neighbors (DkNN) Algorithm

The objective of this algorithm is to find the k-nearest neighbors from a given instance q in vertically distributed databases. We use Euclidean distance as the metric for determining the distance between a pair of tuples p_1, p_2. If there are d attributes, the implicit tuples can be interpreted as points in R^d, and the distance between p_1, p_2 is given by

$$d(p_1, p_2) = \sqrt{\sum_{t=1}^{d} (x_t - y_t)^2}, \tag{6}$$

where p_1=$(x_1, x_2, \ldots, x_d)$, and p_2= $(y_1, y_2, \ldots, y_d)$. Since each tuple includes two types of attributes – *Shared*, and *Unshared* – Equation (6) can be rewritten as:

$$d(p_1, p_2) = \sqrt{\sum_{shared} (x_t - y_t)^2 \quad + \sum_{unshared} (x_t - y_t)^2}. \tag{7}$$

Definition 1. Let *Shared* be the indexed shared relation; we define *Shared* l as the shared tuple with index l, and index l as set of all implicit tuples corresponding to l.

Definition 2. In Equation (7), we define the first summation as the *shared distance* that can be computed at the *Learner* site, where the *Shared* relation and the given point q are known. The second summation is defined as the *unshared distance* that can be computed by finding the minimum unshared distances inside each index at each local site (*Local Computation*), and then we aggregate the results at the *Learner* site to get the global minimum unshared distance for each index (*Global Computation*).

DkNN $(q,\ D_i, D_2, \ldots, D_n)$ **Outlines**

Step 1: Shared Relation Computing. We create a relation called *PreShared* at the *Learner* site, that contains all possible enumerations for the attributes in the set S. Then we generate the relation *Shared* from *PreShared* by removing the tuples with zero count.

Step 2: Local and Global Computations.

1. **Data Structure.** A table called *Distance* resides at the *Learner* site. It has five attributes: the *Index* attribute stores the index of the *Shared* tuples, the *Shared-Distance* attribute stores the distance between the *Shared* attributes (the first summation in Equation (7)), and the *Unshared-Distance* attribute stores the distance between the *Unshared* attributes (the second summation in Equation (7)). The sum and the square root of Shared and Unshared distances will be stored in the *Total-Distance* attribute. The *Class* attribute contains the corresponding class labels.
2. Index the *Shared* relation beginning with zero.
3. **Local Computation.** For every *Shared* tuple l, every site D_i computes the set of ordered pairs $UnsharedDist^i_{(l)} = \{\langle$*unshared distance* (the distances between the *Unshared* attributes in a tuple corresponding to shared l at D_i, and the *Unshared* attributes in q), *the corresponding class label*$\rangle\}$. The detailed steps will be as follows:

   (a) for every *Shared* l do

      (i) for every participating site D_i in database do

         A. Select all tuples that belong to *Shared* l
         B. for each selected tuple v create the ordered pair which consists of
            - the distance $d_v = \sum_{(Unshared)}(x_j - y_j)^2$, where x_j is the value of *Unshared* attribute in v, and y_j is the corresponding value of the *Unshared* attribute in the given point q,
            - the corresponding class label c_v, if D_i contains the *class* label attribute, otherwise we set c_v to empty character (' '),
         C. store all the ordered pairs in a set $UnsharedDist^i_{(l)}$,
         D. arrange $UnsharedDist^i_{(l)}$ in increasing order according to the distance value,
         E. send the $UnsharedDist^i_{(l)}$ back to the *Learner* site.

      (ii) end for

   (b) end for

4. **Global Computation.** This computation will be executed at the *Learner* site. For every *Shared* l, we compute the distance in Equation (7) as follows:

(a) for every *Shared l*

(i) The shared distance can be directly computed as the distance between the *Shared* attributes in the *Shared l* and the corresponding values in the given point q using $\sum_{(Shared)}(x_t - y_t)^2$.

(ii) The unshared distance can be computed by creating the matrix $UnshardMatrix_{(l)}[m][n+1]$ from the following matrices:
(1) $UnshardMatrix_{(l)}[m][n]$ the cross product of the distances from the n sites,
(2) $UnshardMatrix_{(l)}[m][1]$ the corresponding class label of each cross product by taking the concatenation of the second part of the ordered pairs. Therefore, the distance table will be filled as follows:

A. for $i = 1$ to m

- *sum-unshared*$[i] = 0$
- for $j = 1$ to n
  - *sum-unshared*$[i]$ = *sum-unshared*$[i] + UnsharedMatrix_{(l)}[i][j]$
- end for j
- *Distance*$[l]$[*Unshared-Distance*]= *sum-unshared[i]*
- *Distance*$[l]$[*class*] $= UnsharedMatrix_{(l)}[i][n+1]$

B. end for i

C. *Distance*$[l]$[*Shared-Distance*] = *sum-shared*

D. *Distance*$[l]$[*Total-Distance*]= the square root of the sum values in columns *Shared-Distance* and *Unshared-Distance*.

(b) end for

5. Arrange *Distance* table in increasing order according to the *Total-Distance*.
6. Return to the *Learner* site the first k- distances and their corresponding class type ($\langle distance, class \rangle$).

End Algorithm

4.3.2 Example Scenario

We show here an example execution of vertically 5-classifier algorithm, where three databases exist at three different network sites across a wide area network. The local databases from the three sites are shown in Table 1. The three databases together implicitly define a global database $\mathcal{D}$ consisting of points in a 6-dimensional space. The algorithm's objective here is to classify the instance $q = (1, 2, 2, 3, 4, 5)$.

- From Table 1, the *Shared* attributes are a, b and c, and the *Shared* values are $\{1, 2\}$, $\{1, 2, 3\}$ and $\{1, 2\}$ for a, b, and c, respectively. According to step 1, the relation *PreShared* will be as in Table 2.

Site1		
a	b	e
1	1	2
1	3	4
3	2	2
2	1	4
3	1	5
2	2	2
1	1	1

Site2		
b	c	f
1	1	1
3	2	4
2	2	9
7	2	8
4	1	6
1	1	2
1	2	8

Site3			
a	c	d	**class**
2	2	3	0
1	1	9	1
2	1	8	0
1	1	10	0
2	1	11	1
2	2	4	0
2	1	4	1

Table 1. Explicit component databases at local sites

- For every tuple in *PreShared*, we compute its *count* by multiplying its counts at the three sites; the relation *Shared* will be as in Table 3.

PreShared		
a	b	c
1	1	1
1	1	2
1	2	1
1	2	2
1	3	1
1	3	2
2	1	1
2	1	2
2	2	1
2	2	2
2	3	1
2	3	2

Table 2. The *PreShared* relation

Index	a	b	c
0	1	1	1
1	2	1	1
2	2	1	2

Table 3. The indexed Shared relation

- **Local Computation.** For every *Shared* tuple l, every site i will compute the unshared distance set $UnsharedDist^i_{(l)}$. Below we execute the local computation for *Shared* 0.

  – At Site1: the *Unshared* attribute e has two values 2,1, and has the value 4 at the instance q, and then the ordered pairs set of unshared distances and the

corresponding class labels $UnsharedDist^1_{(0)}$= {⟨4,' '⟩, ⟨9,' '⟩}, where d_1=4, and d_2=9.

- At Site2: The *Unshared* attribute f has two values 2,1, and has the value 5 at the instance q, and then $UnsharedDist^2_{(0)}$ = {⟨9,' '⟩, ⟨16,' '⟩}, where d_1=16, and d_2=9.
- At Site3: The *Unshared* attribute d has two values 9, 10, and has the value 3 at the instance q, and then $UnsharedDist^3_{(0)}$ = {⟨36, '1'⟩, ⟨49, '0'⟩}, where d_1 = 36, and d_2=49.

- **Global Computations.** The global unshared distance matrix $UnshardMatrix_l(m, n)$ will be computed as the Cartesian product of all received $UnsharedDist^i_{(l)}$s. Below we execute the global computation for *Shared* 0.

  - **Unshared Distances Computing.** Since the count of tuples that meet the *Shared* 0 is 8, i.e, $m = 8$, and the number of participating sites is 3, i.e, $n = 3$.

    $UnsharedMatrix_{(0)}(8, 4)$=

$$\begin{pmatrix} 4 & 9 & 36 & 1 \\ 4 & 9 & 49 & 0 \\ 4 & 16 & 36 & 1 \\ 4 & 16 & 49 & 0 \\ 9 & 9 & 36 & 1 \\ 9 & 9 & 49 & 0 \\ 9 & 16 & 36 & 1 \\ 9 & 16 & 49 & 0 \end{pmatrix}$$

    The sum of distances in each row gives the total of unshared distances for a tuple in *Shared* 0 and the corresponding class label (column 4).

    sum-unshared =

$$\begin{pmatrix} 49 & 1 \\ 62 & 0 \\ 56 & 1 \\ 69 & 0 \\ 54 & 1 \\ 67 & 0 \\ 61 & 1 \\ 74 & 0 \end{pmatrix}$$

  - **Shared Distances Computing.** The shared distance for shared 0 will be $Sum - Shared = 2$, where the *Shared* attributes a, b, c take the values 1, 1, 1 in *Shared* 0 and the values 1, 2, 2 in q.

By executing the algorithm with every *Shared* tuple, the ordered *Distance* table will be as in Table 4. From the first five rows, the class type of q will be 0.

Index	Shared Distance	Unshared Distance	Total Result	Class
2	2	9	3.32	0
2	2	10	3.46	0
1	3	10	3.61	1
1	3	17	4.47	1
1	3	34	6.08	0
1	3	41	6.63	0
0	2	49	7.14	1
0	2	54	7.48	1
0	2	56	7.62	1
0	2	61	7.94	1
0	2	62	8.00	0
0	2	67	8.31	0
0	2	69	8.43	0
0	2	74	8.72	0
1	3	73	8.72	1
1	3	80	9.11	1

Table 4. The *Distance* table after updating and ordering

4.3.3 Complexity Computing

We show below an expression for the number of messages that need to be exchanged among the stationary and mobile agents for running the k-NNC from vertically distributed data. Let us say there are: n relations reside at n different network sites, r tuples in *PreShared* relation, and t tuples in relation *Shared* ($r \leq t$).

- **Stationary Agents Implementation**
  - **One Summary Per Message (Un-Optimized)**
    Exchanged Messages for Computing Shared Relation. If there are n participating agents then one agent would be sending one request to each agent to get all shared values for computing the relation *PreShared*; then for each tuple in *PreShared*, one agent would be sending one request to each agent to check the tuple count. Therefore, amounting to a total of $n+r*n = n(1+r)$ messages are required for computing the *Shared* relation.
    Exchanged Messages for Computing $UnsharedDist^i_{(l)}$*:* For each *Shared* l, we need n messages for computing $UnsharedDist^i_{(l)}$, so $t*n$ exchanged messages are required.
    Therefore, the total number of exchanged messages of our algorithm will be:

$$Total\ Exchanged\ Messages = n(t + r + 1). \tag{8}$$

  - **Exchanging all the Summaries in one Message (Optimized)**
    It is possible to send a request to an agent for all $Id_i(D_i, S)$ values, that is, the values corresponding to all tuples $cond_j$ of S in one request, and receive

all the summaries in one message. This reduces the number of messages exchanged to be

$$Total\ Exchanged\ Messages = 4n. \tag{9}$$

The first approach may be considered more secure for transmission over a network because each message contains only little information about the participating databases. The second alternative requires very few messages but each message contains more information about each database.

- **Mobile Agents Implementation**
  Exchanged Messages for computing Shared Relation. The local results for computing, the shared values sets at each site can be gathered during a single visit to each site and then the local results are aggregated by finding the cross product. We need to visit each site once again for computing the *Shared* relation from *PreShared*.
  Exchanged Messages for computing $UnsharedDist^i_{(l)}$. For each *Shared* l, $UnsharedDist^i_{(l)}$ can be gathered during a single visit to each site. Then, the number of exchanged messages to perform this step will be t. Therefore, the total number of exchanged messages for our algorithm will be

$$Total\ Exchanged\ Messages = t + 2. \tag{10}$$

Assertion 2. The algorithm for learning k-NN classifiers from vertically distributed data returns the same results with respect to the algorithm for learning k-NN classifiers from centralized data.

Proof. It is obvious from Equation (7) that the distances $d(p_1, p_2)$ computed in the distributed case are the same as the distances computed in the centralized case. Then their corresponding sets of class labels are identical, i.e., the set of class labels used for majority vote in the centralized case is the same as the set of class labels used for majority vote in the distributed case. □

4.3.4 Simulation Results

We have performed a number of tests to demonstrate that the k-NN classifier can be run in a distributed knowledge environment without moving all the databases to a single site. These tests have been carried out on a network of workstations connected by a LAN and tested against a number of databases of different sizes. The algorithms have been tested on both test data and real life databases. We have implemented the algorithm using Java, RMI (Remote Method Invocation), and JDBC (Java Database Connectivity) to interface with the databases. This has been done to provide a standard interface and platform independence.

Figure 4 shows how the time taken to compute k-NN classifier in an implicit database $\mathcal{D}$ changes with the size of the individual databases. As we can see, when

we exchange one summary per message, the time taken to run the k-NN classifier varies exponentially as the size of the database increases. However, when we use the optimized method the time taken to run the k-NN classifier reduces considerably and depends on the number of participating nodes.

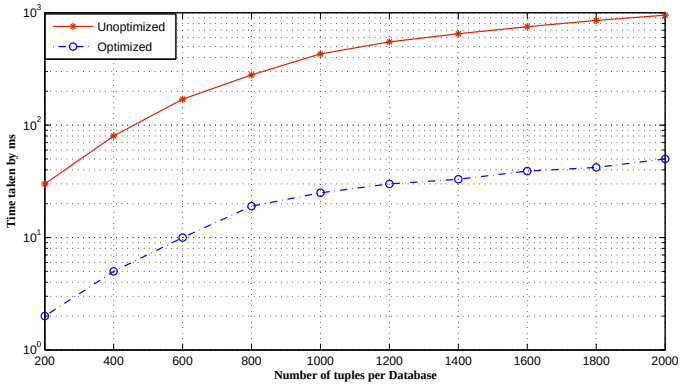

Fig. 4. Time taken to run k-NN classifier on vertically distributed databases

Figure 5 shows how the number of messages exchanged between the *Learner* site and the remote sites varies with the number of tuples in the database. It can be easily seen that the number of messages exchanged varies exponentially with the size of the database when we send one summary per message. The result validates the expression for the total number of messages exchanged as given above. However, in the optimized version when we receive all the summaries in a single message, the number of messages exchanged was a constant depending upon the total number of participating nodes.

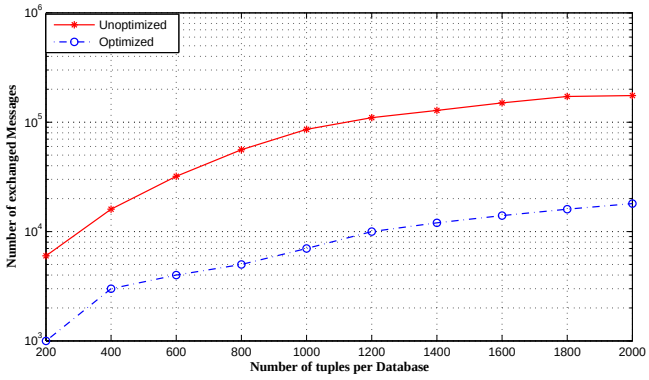

Fig. 5. Number of messages exchanged to run k-NN classifier on vertically distributed databases

5 CONCLUSION

In this paper, we have defined the problem of learning from distributed data, and presented a general strategy for transforming algorithms for learning from centralized data into algorithms for learning from distributed data. This strategy is based on the decomposition of an algorithm into information extraction and hypothesis generation components. The information extraction from distributed data entails decomposing each statistical query posed by the information extraction component of the *Learner* into local computations that can be performed by the individual data sources, and a procedure for combining the results of local computations into an answer for the original query. We have applied this strategy to design algorithms for learning the k-NN classifiers from horizontally and vertically distributed data.

REFERENCES

[1] Aronis, J.—Kolluri, V.—Provost, F.—Buchanan, B.: The WoRLD: Knowledge Discovery from Multiple Distributed Databases. Technical Report ISL-96-6, Intelligent Systems Laboratory, Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, 1996.

[2] Bhatnagar, R.—Srinivasan, S.: Pattern Discovery in Distributed Databases. In Proceedings of the Fourteenth AAAI Conference, Providence, RI, AAAI Press/The MIT Press, pp. 504–508, 1997.

[3] Domingos, P.: Knowledge Acquisition from Examples via Multiple Models. In Proceedings of the Fourteenth International Conference on Machine Learning, Nashville, TN, pp. 98–106, 1997.

[4] Du, W.—Zhan, Z.: Building Decision Tree Classifier on Private Data. The 2002 IEEE International Conference on Data Mining (ICDM '02), Maebashi City, Japan, December 2002.

[5] Hendler, J.: Science and the Semantic Web. Science, 299, January 2003.

[6] Kantarcioglu, M.—Clifton, C.: Privacy-Preserving Distributed Mining of Association Rules on Horizontally Partitioned Data. In Proceedings of ACM SIGMOD Workshop on Research Issues on DMKD '02, June 2002.

[7] Kargupta, H.—Park, B. H.—Hershberger, D.—Johnson, E.: Collective Data Mining: A New Perspective Toward Distributed Data Mining. In Advances in Distributed and Parallel Knowledge Discovery. MIT Press, 1999.

[8] Kargupta, H.—Liu, K.—Ryan, J.: Random Projection and Privacy Preserving Correlation Computation from Distributed Data. In Proceedings of High Performance, Pervasive, and Data Stream Mining 6^{th} International Workshop on High Performance Data Mining: Pervasive and Data Stream Mining (HPDM:PDS '03), 2003.

[9] Khedr, A. M.—Bhatnagar, R.: A Decomposable Algorithm for Minimum Spanning Tree. Distributed Computing-Lecture Notes in Computer Science, Springer-Verlag Heidelberg, Vol. 2918, pp. 33–44, 2004.

[10] Khedr, A. M.—Bhatnagar, R.: Agents for Integrating Distributed Data for Complex Computations. Computing and Informatics, Vol. 26, 2007, pp. 149–170.

[11] Leckie, C.—Kotagiri, R.: Learning to Share Distributed Probabilistic Beliefs. In Proceedings of The Nineteenth International Conference on Machine Learning (ICML 2002), Sydney, Australia, July 2002.

[12] Lindell, Y.—Pinkas, B.: Privacy Preserving Data Mining. Journal of Cryptology, Vol. 15, 2002, No. 3, pp. 177–206.

[13] Lin, X.—Clifton, C.—Zhu, M.: Privacy Preserving Clustering with Distributed EM Mixture Modeling. Knowledge and Information Systems, Vol. 8, 2005, No. 1, pp. 68–81.

[14] Merugu, S.—Ghosh, J.: Privacy-Preserving Distributed Clustering Using Generative Models. In Proceedings of the Third IEEE International Conference on Data Mining (ICDM '03), Melbourne, FL, November 2003.

[15] Park, B.—Kargupta, H.: Constructing Simpler Decision Trees from Ensemble Models Using Fourier Analysis. In Proceedings of the 7th Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD 2002), pp. 18–23, Madison, WI, ACM SIGMOD, June 2002.

[16] Prodromidis, A. L.—Chan, P.—Stolfo, S. J.: Meta-Learning in Distributed Data Mining Systems: Issues and Approaches. In Proceedings of Advances of Distributed Data Mining, AAAI Press, 2000.

[17] Provost, F.: Distributed Data Mining: Scaling Up and Beyond. In Proceedings of Advances in Distributed Data Mining, MIT Press, 2000.

[18] Provost, J.—Kolluri, V.: A Survey of Methods for Scaling Up Inductive Algorithms. Data Mining and Knowledge Discovery, Vol. 3, 1999, No. 2, pp. 131–169.

[19] Provost, F.—Hennessy, D.: Scaling Up: Distributed Machine Learning with Cooperation. In Proceedings of the Thirteenth National Conference on Artificial Intelligence, 1996.

[20] Schuster, A.—Wolf, R.—Gilburd, B.: Privacy-Preserving Association Rule Mining in Large-Scale Distributed Systems. In Proceedings of Cluster Computing and the Grid (CCGrid), 2004.

[21] Tumer, K.—Ghosh, J.: Robust Order Statistics Based Ensembles for Distributed Data Mining. In Advances in Distributed and Parallel Knowledge Discovery, pp. 185–210, MIT, 2000.

[22] Turinsky, A.—Grossman, R. L.: A Framework for Finding Distributed Data Mining Strategies That Are Intermediate Between Centralized Strategies and In-Place Strategies. In Proceedings of KDD, 2000.

[23] Vaidya, J.—Clifton, C.: Privacy Preserving Association Rule Mining in Vertically Partitioned Data. In The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, Canada, July 2002.

[24] Vaidya, J.—Clifton, C.: Privacy-Preserving k-Means Clustering Over Vertically Partitioned Data. In The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, August 2003.

[25] Wang, C.—Chen, M.: On the Complexity of Distributed Query Optimization. IEEE Transactions on Knowledge and Data Engineering, Vol. 8, 1996, No. 4, pp. 650–662.

[26] Young, B.—Bhatnagar, R.: Secure K-NN Algorithm for Distributed Databases. Proceedings of the International Conference on Privacy Security and Trust, pp. 485–490, 2006.

Ahmed M. Khedr received his B. Sc. degree in mathematics in June 1989 and the M. Sc. degree in the area of optimal control in July 1995 both from Zagazig University, Egypt. In March 2003 he received his Ph. D. degree in computer science from University of Cincinnati, Ohio, USA. From March 2003 to January 2004, he was a research assistant professor at ECECS Department University of Cincinnati, USA. From January 2004 till now he is working as faculty at the Mathematics Department, Faculty of Science, Zagazig University, Egypt. He has coauthored 21 works in journals and conferences relating with optimal control, decomposable algorithms, and wireless sensor networks.