Privately Computing a Distributed k-nn Classifier[⋆]

Murat Kantarcıoğlu and Chris Clifton

Purdue University, Department of Computer Sciences
250 N University St
West Lafayette, IN 47907-2066 USA
+1-765-494-6408, Fax: +1-765-494-0739
{kanmurat,clifton}@cs.purdue.edu

Abstract. The ability of databases to organize and share data often raises privacy concerns. Data warehousing combined with data mining, bringing data from multiple sources under a single authority, increases the risk of privacy violations. Privacy preserving data mining provides a means of addressing this issue, particularly if data mining is done in a way that doesn't disclose information beyond the result. This paper presents a method for privately computing $k - nn$ classification from distributed sources without revealing any information about the sources or their data, other than that revealed by the final classification result.

1 Introduction

The growing amount of information stored in different databases has lead to an increase in privacy concerns. Bringing data from multiple sources under one roof may improve processing and mining of the data, but it also increases the potential for misuse. Privacy is important, and concerns over privacy can prevent even the legitimate use of data. For example, the *Data-Mining Moratorium Act* introduced in the U.S. Senate would have forbid *any* "data-mining program" within the U.S. Department of Defense. Privacy concerns must be addressed, or such over-reaction may prevent beneficial uses of data mining.

Consider the case of a physician who wants to learn the most likely diagnosis for a patient by looking at diagnoses of similar symptoms at other hospitals. Specifically, the physician wants to use a *k-nearest neighbor* (k-nn) classifier to predict the disease of the patient. Revealing the patients particular test results may not be a threat to privacy (if only the physician knows the identity of the patient) but privacy of the different hospitals may be at risk. If this procedure is implemented naïvely, the researcher may learn that two patients with the same medical test results are diagnosed with different diseases in different hospitals. This could damage the reputations of the hospitals. The possibility of such incidents may prevent hospitals from participating in such a diagnostic tool. The

⋆ This material is based upon work supported by the National Science Foundation under Grant No. 0312357.

obvious question is, can this be done without revealing anything other than the final classification? The answer is yes: This paper presents an efficient method with provable privacy properties for k-nn classification.

This work assumes data is horizontally partitioned, i.e., each database is able to construct its own k-nearest neighbors independently. The *distributed* problems are determining which of the local results are the closest globally, and finding the majority class of the global k-nearest neighbors. We assume that attributes of the instance that needs to be classified are not private (i.e., we do not try to protect the privacy of the query issuer); we want to protect the privacy of the data sources. The approach makes use of an untrusted, non-colluding party: a party that is not allowed to learn anything about any of the data, but is trusted not to collude with other parties to reveal information about the data.

The basic idea is that each site finds its own k-nearest neighbors, and encrypts the class with the public key of the site that sent the instance for classification (querying site). The parties compare their k-nearest neighbors with those of all other sites – except that the comparison gives each site a random share of the result, so no party learns the result of the comparison. The results from all sites are combined, scrambled, and given to the untrusted, non-colluding site. This site combines the random shares to get a comparison result for each pair, enabling it to sort and select the global k-nearest neighbors (but without learning the source or values of the items). The querying site and the untrusted, non-colluding site then engage in a protocol to find the class value. Each site learns nothing about other sites (the comparison results appears to be randomly chosen bits.) The untrusted site sees $k * n$ encrypted results. It is able to totally order the results, but since it knows nothing about what each means or where it comes from, it learns nothing. The querying site only sees the final result.

Details of the algorithm are given in Section 3, along with a discussion of the privacy of the method. Computation and communication costs are given in Section 4. First, we discuss related work and relevant background.

2 Related Work

Finding the k-nearest neighbors of a multidimensional data point q [1] and building k-nn classifiers [2] have been well studied, but not in the context of security.

Interest has arisen in privacy-preserving data mining. One approach is to add "noise" to the data before the data mining process, and use techniques that mitigate the impact of the noise on the data mining results. The other approach is based on protecting the privacy of distributed sources. This was first addressed for the construction of decision trees. This work closely followed the secure multiparty computation approach discussed below, achieving "perfect" privacy, i.e., nothing is learned that could not be deduced from one's own data and the resulting tree. The key insight was to trade computation and communication cost for accuracy, improving efficiency over the generic secure multiparty computation method. Methods have since been developed for association rules, K-means and EM clustering, and generalized approaches to reducing the number of "on-line" parties required for computation. For a survey of this area see [3]. This paper falls in the latter class: privacy preserving distributed data mining work. The goal is to provably prevent disclosure of the "training data" as much as possible, disclosing only what is inherent in the classification result.

To better explain the concept of provable privacy of distributed sources, we give some background on Secure Multiparty Computation. Yao introduced a solution with his millionaire's problem: Two millionaires want to know who is richer, without disclosing their net worth[4]. Goldreich proved there is a secure solution for *any* functionality[5]. The idea is that the function to be computed is represented as a combinatorial circuit. The idea is based on computing random shares of each wire in the circuit: the exclusive-or of the shares is the correct value, but from its own share a party learns nothing. Each party sends a random bit to the other party for each input, and makes its own share the exclusive-or (xor) of its input and the random bit. The parties then run a cryptographic protocol to learn shares of each gate. At the end, the parties combine their shares to obtain the final result. This protocol has been proven to produce the desired result, and to do so without disclosing anything except that result.

The cost of circuit evaluation for large inputs has resulted in several algorithms for more efficiently computing specific functionality. We do make use of circuit evaluation as a subroutine for privately determining if $a \geq b$. We also use the definitions and proof techniques of Secure Multiparty Computation to verify the privacy and security properties of our algorithm; these will be introduced as needed. First, we give the details of the algorithm itself.

3 Secure k-nn Classification

We first formally define the problem. Let R be the domain of the attributes and C be the domain of the class values. Let D_i denote the database of instances at site S_i. Let (x, d, k) be the query originated by site O, where $x \in R$ is the instance to be classified, and $d : R \times R \rightarrow [0, 1]$ is a distance function used to determine which k items are closest to x (e.g., Euclidean distance, although any metric could be used provided each site can compute $d(x, x_j)$ for every x_j in its database.) Given the data instance x, our goal is to find the k nearest neighbors of x in the union of the databases and return the class of the majority of those neighbors as the predicted class of x:

$$C_x = \text{Maj}\left(\prod_{c}\left(\underset{(x_i,c_i)\in D_1\cup D_2\ldots\cup D_n}{\text{argmin}_k}(d(x_i, x))\right)\right)$$

where $\prod$ is the projection function and Maj is the majority function.

The security/privacy goal is to find C_x while meeting the following criteria:

- No site except O will be able to predict C_x better than looking at (x, d, k) and its own database D_i (E.g., if Site S_i has k points x_i such that $d(x, x_i) = 0$, it is likely that the majority class of the x_i will be the result); and
- No site learns anything about the source of x_i except its own.

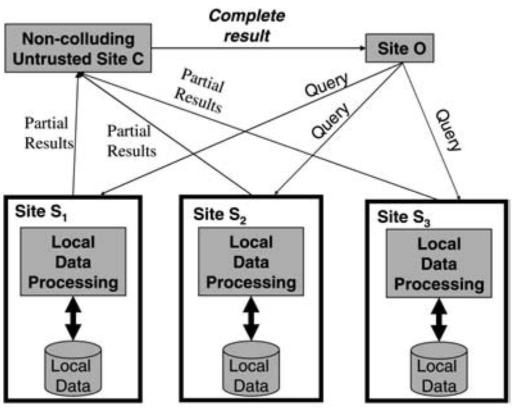

Fig. 1. Information flow in secure k-nn classification

Theorem 1 shows this by proving what *is* disclosed. Assuming the number of sites n and the query (x, d, k) are public, site O learns only the result C_x. The other sites learn only what they can infer from their own data and the query.

To achieve a secure and efficient solution, we make use of an untrusted, non-colluding site, an approach first suggested in [6]. Such a site learns nothing in isolation, however by colluding with other sites it could obtain information that should not be revealed. Therefore, the only trust placed in the site is that it not collude with any other sites to violate privacy. Although this seems like a strong assumption, it often occurs in real life. For example, bidders or sellers on e-bay assume that e-bay is not colluding with other bidders or sellers against them. We emphasize that the untrusted site learns nothing except the public values k and n. A diagram showing the information flow is given in Figure 1.

3.1 The Algorithm

Given the query, each site can find its own closest k items without exchanging information. These $n * k$ candidate items must contain the k nearest neighbors of x; what remains is to find the closest k among the candidates and return the majority class of those k instances to site O. This poses two security challenges:

1. Determining which of the $n * k$ items are the k nearest neighbors without revealing anything about the items, where they come from, or their distance to the instance that needs to be classified; and
2. Learning the class value while disclosing it to only the originating site.

At first glance, this appears simple – have each site send their local k-nn distances and classes to a third party C. C learns the result, but also learns distances and the sites that contributes to the global k-nn result. A slightly more sophisticated approach is for C to publish a candidate distance value, and have local sites return the number of items within that distance, refining the value until k items are within the distance. Then the class value can be computed.

This still reveals the sites that contribute to the global result, the distances to the queried instance, the final classification result to C, and more. To ensure privacy concerns are met, we provide a solution that (under the non-collusion assumption) reveals **nothing** to any site that cannot be inferred by looking at its own data and the instance, except that O learns the final result.

The use of an untrusted third party, along with public-key encryption, makes it easy to solve challenge 2. Each site encrypts the class value with the public key of O before passing it to the non-colluding site C. The data source sites are then left out of the loop – since they never see the data again, they can learn nothing after passing their data to C (e.g., they cannot test to see if their own encrypted values match those selected for the result.) C and O will participate in a special protocol to reveal only the majority class (explained later.)

Meeting challenge 1 is more difficult. Sending the distance $d(x_i, x) = d_i$ to C, even with encrypted results, reveals information about the location of the points. Instead site C is sent an $n * k - 1$ length vector for each point x_i, containing the results of comparing x_i with all other points. This enables C to order the points and select the k nearest neighbors. Since the existence of a distance metric implies a total ordering exists, and the number of points is fixed, C learns nothing.

Two problems remain. The first is that we must prevent C from learning which point comes from which site, or it can learn the source of the k nearest neighbors (among other things.) This is easily addressed – all points and their associated comparison vectors are sent to one of the data sites S_s, which combines them and scrambles the order before passing them on to C. Public key encryption, using C's public key, prevents S_s from learning anything.

The second issue is more challenging: How do we build the comparison vectors at site S_i without S_i learning about values at other sites? If two sites just compare items, they both learn something about the data at the other site (e.g., if S_i's items are all closer to x than S_j's items, both learn that S_j does not have an item that contributes to the result.) For this we use the share-splitting idea from secure multiparty computation. Instead of containing the results of comparing x_i with other items, the vector contains a *random share* of the comparison result. E.g., if d_i for x_i is smaller than d_j for x_j, then the comparison $d_i < d_j$ should return 0. Either the element of the d_i vector corresponding to d_j and the element of the d_j vector corresponding to d_i both contain 0, or they both contain 1 – $0 \oplus 0 = 1 \oplus 1 = 0$. However, knowing only one share tells nothing: a share 0 could mean either $0 \oplus 0 = 0$ or $0 \oplus 1 = 1$. From d_i's view, the share has equal probability of being 0 or 1 (a random choice), so it learns nothing.

To generate random shares of the comparison, we return to secure multiparty computation. We stop the generic circuit comparison method before combining shares to learn the final result. In other words, given two integers a and b, secure comparison of a, b ($f : \{0,1\}^* \times \{0,1\}^* \longmapsto \{0,1\} \times \{0,1\}$) is defined as follows:

$$f(a,b) = \begin{cases} (1 \oplus r, r) \text{ if } a > b \\ (0 \oplus r, r) \text{ if } a < b \end{cases}$$

where each site sees only one component of the function output. This states that if $a > b$ the xor of the shares of the participating sites will be 1, otherwise the xor of the shares will be 0. Using this function, each site can compare its elements with all other elements and learn nothing about the result of the comparisons.

Two additional details. First, the identifiers used to track the d_j in the comparison share vector for d_i must not disclose anything. One option would be for the combining site S_s to assign identifiers, but this would require independent encryption of the single bits of the comparison shares, and single bit encryption is problematic. Instead, each site generates pseudo-random unique identifiers site C cannot distinguish from random. One simple and secure way to handle this problem is using a pseudo-random permutation function. With a fixed key, DES is assumed to be such a function. The sites agree on a key K, and each site S_i generates k identifiers by evaluating $E_K(ik), E_K(ik+1), \ldots, E_K(ik+k-1)$. Since encryption is assumed to be secure and a permutation, each site will get non-intersecting identifiers that appear random to any (polynomial time) adversary not knowing K (in particular, C). The identifiers are also used to determine which of a comparison pair will be the left-hand side in a comparison: The item with the smaller identifier corresponds to x in the comparison function $f(x, y)$.

The second detail is equality. Identifying that two items are at equal distances reveals information. We must disambiguate consistently, without giving even a probabilistic estimate on the likelihood of equality. The solution is to add extra low-order bits to each distance, based on a unique mapping from the identifier that appears random to C - the same trick used to generate identifiers. The distance used for comparison is actually $d||E_u(ik+j)$, where the encryption function E is as above, but with a different key. This ensures that distances are unique, guaranteeing a total ordering of equal distances.

Protocol 1 gives the algorithm details. Note that at the end of the k-nearest neighbor selection phase, C has the class of the k-nearest neighbors encrypted with E_o. Assuming the usage of Blum-Goldwasser encryption, each class value $class_i$ will have ciphertext of the form $(\acute{r}, class_i \oplus r)$, where O has enough information to determine r given $\acute{r}$, enabling decryption to get $class_i$. Instead of sending these values to O, C will xor each of these values with a random value r_i. C then sends $(\acute{r}, class_i \oplus r \oplus r_i)$ to O. O decrypts to get $class'_i = class_i \oplus r_i$, indistinguishable (to O) from a random value. O and C now use the generic secure circuit evaluation approach to evaluate the majority function:

$$\text{Maj}(class'_1 \oplus r_1, \ldots, class'_k \oplus r_k).$$

This is a simple circuit with size complexity dependent on k and number of distinct classes. The cost is dominated by the k-nearest neighbor selection phase.

To clarify, we give an example for $k = 1$ and $n = 3$.

Example 1. Given $(x, d, 1)$, each site finds its 1-nearest neighbor. Assume that site S_1 has $(d(x, x_1), \prod_c(x_1, c_1) = (0.1, c_1)$, site S_2 has (x_2, c_2) at distance 0.2, and site S_3 has (x_3, c_3) at 0.15. After generating random identifiers, S_1 has $(3, 0.1, c_1)$, S_2 has $(1, 0.2, c_2)$, and S_3 has $(2, 0.15, c_3)$. (For simplicity we omit the low-order bit disambiguation.) In generating the comparison vector for c_1 and c_2, S_1 notes that $c_1.id = 3 > 1$, so it has the right argument of $f(a, b)$ and generates a random bit (say 0) as its share of the output. Since $0.2 \geq 0.1$, S_2

Protocol 1 Privacy-preserving k-nn classification algorithm

```
Require: $n$ sites $S_i$, $1 \leq i \leq n$, each with a database $D_i$; permuting site $S_s$ (where
  $s$ may be in $1, \ldots, n$); and untrusted non-colluding site $C$ (distinct from $S_i$ or $S_s$).
  Query $(x, d, k)$ generated by originating site $O$. Public encryption keys $E_c$ for site $C$
  and $E_o$ for site $O$, key generation function $E_K$ and $E_u$ known only to the $S_i$.
  for all sites $S_i$, in parallel do
    {Build vector of random key, distance, and result for local closest $k$}
    Select $k$ items $(d(x_i, x), \prod_{c_i}(x_i, c_i))$ with smallest $d(x_i, x)$ from $D_i$ into $N_i$
    $R_i = \emptyset$
    for $j = 0..k-1$ {Compute identifiers and "extended" local distances} do
      $R_i = R_i \cup \{(E_K(ik+j), N_i[j].d || E_u(ik+j),\ E_o(N_i[j].result))\}$ {|| is string
      concatenation}
    end for
    $ER_i = \emptyset$
    for each $(id, d, E_o(c)) \in R_i$ {Comparison phase} do
      $v = \emptyset$
      for each site $h = i \ldots n$ {If $i = h$, just generate values locally.} do
        for $j = 0 \ldots k-1$ do
          if $id < R_h[j].id$ then
            $v = v \cup\ \{(R_h[j].id, S_i\text{'s share of } f(d, R_h[j].d)\}$
            $v_{hj} = v_{hj} \cup \{(id, S_j\text{'s share of } f(d, R_h[j].d)\}$
          else if $id > R_h[j].id$ then
            $v = v \cup\ \{(R_h[j].id, S_i\text{'s share of } f(R_h[j].d, d)\}$
            $v_{hj} = v_{hj} \cup \{(id, S_j\text{'s share of } f(R_h[j].d, d)\}$
          end if
        end for
      end for
      $ER_i = ER_i \cup (id, E_c(v), E_o(c))$
    end for
    send $ER_i$ to $S_s$
  end for
  {At site $S_s$: Permutation phase}
  set $ER = \cup_{i=1}^{n}(ER_i)$
  permute $ER$ and send it to $C$
  {At site $C$: $k$ nearest neighbor selection phase}
  Decrypt the encrypted shares of the comparison results
  Use the pairwise comparisons to find the global $k$ nearest neighbors
  Let $R$ be the set of encrypted class values($E_o(c)$) of the global $k$ nearest neighbor
  for all $R.E_o(c_i)$ {encrypted as $(\acute{r}, c_i \oplus r)$} do
    $NR[i] = R.E_o(c_i) \oplus$ random $r_i$
  end for
  Site $C$ sends $NR$ to $O$
  {At site $O$:}
  for all $NR[i]$ {$= (\acute{r}, c_i \oplus r \oplus r_i)$} do
    Find $r$ using $\acute{r}$ and the private key
    $NR_d[i] = c_i \oplus r \oplus r_i \oplus r$
  end for
  Find Maj from the random shares of $C$ and $NR_d$ using secure circuit evaluation.
```

learns that its share should be $1 \oplus 0 = 1$. (Neither S_2 or S_1 learns the other's share, or the comparison result.) Secure comparisons with the other sites are performed, giving each site tuples containing its share of the comparison with all other items. These are encrypted with C's public key to give:

$$\begin{aligned}
S_1 &: (3, E_c((1,1),(2,0)), E_o(c_1))\\
S_2 &: (1, E_c((2,1),(3,0)), E_o(c_2))\\
S_3 &: (2, E_c((1,1),(3,1)), E_o(c_3))
\end{aligned}$$

The above are sent to S_3, which permutes the set, strips source information, and sends it to C. Site C decrypts the comparison share vectors to get:

$$\begin{gathered}
(2, ((1,1),(3,1)), E_o(c_3))\\
(3, ((1,1),(2,0)), E_o(c_1))\\
(1, ((2,1),(3,0)), E_o(c_2))
\end{gathered}$$

C now compares the items to find the nearest neighbor. As an example, to compare items 2 and 3, we take the pair $(3,1)$ from the first (2) row and the pair $(2,0)$ from the second (3) row. Combining the share portions of these pairs gives $1 \oplus 0 = 1$, so $d(x,x_2) \geq d(x,x_3)$. Likewise, comparing 1 and 3 gives $0 \oplus 1 = 1$, so $d(x,x_1) \geq d(x,x_3)$. Therefore, x_1 is closest to x. C sends $E_o(c_1)$ to O, which decrypts to get c_1, the correct result. (With $k > 1$, C and O would engage in a protocol to determine which c_i was in the majority, and send $E_o(c_i)$ to O.)

3.2 Security of the Protocol

We now prove that Protocol 1 is secure. We assume that sites O and C are not among the S_i. We have discussed the need for C being a separate site. O cannot be a data source, as it would be able to recognize its own $E_o(x)$ among the results, thus knowing if it was the source of some of the k nearest neighbors.

To define security we use definitions from the Secure Multiparty Computation community, specifically security in the semi-honest model. Loosely speaking, a semi-honest party follows the rules of the protocol, but is free to try to learn additional information from what it sees during the execution of the protocol.

The formal definition is given in [7]. Basically, it states that the view of each party during the execution of the protocol can be effectively simulated knowing only the input and the output of that party. Extending this definition to multiple parties is straightforward. The key idea is that of *simulation*: If we can simulate what is seen during execution of the protocol knowing only our own input and our portion of the final output, then we haven't learned anything from the information exchanged in a real execution of the protocol. Under certain assumptions, we can extend our protocols to malicious parties (those that need not follow the protocol). Due to space limitations we omit the discussion here.

We need one additional tool to prove the security of the protocol. The encrypted items seen by S_s and C during execution of the protocol may disclose some information. The problem is that two items corresponding to the same plaintext map to the same ciphertext. If multiple items are of the same class (as would be expected in k-nn classification), the permuting site S_s would learn the class entropy in the k-nn of each site as well as the number of identical results between sites. The comparison site C would learn this for the data as a whole. Neither learns the result, but something of the distribution is revealed.

Fortunately, the cryptography community has a solution: *probabilistic* public-key encryption. The idea is that the same plaintext may map to different ciphertexts, but these will all map back to the same plaintext when decrypted. Using probabilistic public-key encryption for E_o allows us to show Protocol 1 is secure. (Deterministic public-key encryption is acceptable for E_c, as the set of $nk-1$ identifiers in the set v are different for every item, so no two plaintexts are the same.) The Blum-Goldwasser probabilistic encryption scheme[8], with a cipher text of the form $(\acute{r}, M \oplus r)$ for message M, is one example. In this, given $\acute{r}$ and the private key, it is possible to compute r to recover the original message.

Theorem 1. *Protocol 1 privately computes the k-nn classification in the semi-honest model where there is no collusion; only site O learns the result.*

Proof. To show that Protocol 1 is secure under the semi-honest model, we must demonstrate that what each site sees during the execution of the protocol can be simulated in polynomial time using only its own input and output. Specifically, the output of the simulation and the view seen during the execution must be computationally indistinguishable. We also use the general composition theorem for semi-honest computation: if g securely reduces to f and there is a way to compute f securely, then there is a way to compute g securely. In our context, f is the secure comparison of distances, and g is Protocol 1. We show that our protocol uses comparison in a way that reveals nothing.

We first define the simulator for the view of site S_i. Before the comparison phase, S_i can compute its view from its own input. The comparison phase involves communication, so simulation is more difficult. If we look at a single comparison, S_i sees several things. First, it sees the identifier $R_h[j].id$. Since S_i knows E_K, h, and j; the simulator can generate the exact identifier, so the simulator view is identical to the actual view. It also sees a share of the comparison result. If $i = h$, S_i generates the values locally, and the simulator does the same. If not local, there are two possibilities. If $id > R_h[j].id$, it holds the second argument, and generates a random bit as its share of the comparison result. The simulator does the same. Otherwise, the secure comparison will generate S_i's share of the comparison. Assume $d < R_h[j].d$: S_i's share is $0 \oplus r$, where r is S_h's randomly chosen share. Assuming S_h is equally likely to generate a 1 or 0, the probability that S_i's share is 1 is 0.5. This is independent of the input – thus, a simulator that generates a random bit has the same likelihood of generating a 1 as S_i's view in the real protocol. The composition theorem (and prior work on secure comparison) shows the algorithm so far is privacy preserving.

We can extend this argument to the entire set of comparisons seen by S_i during execution of the protocol. The probability that the simulator will output a particular binary string x for a given sequence of comparisons is $\frac{1}{2^{nk-1}}$. Since actual shares of the comparison result are chosen randomly from a uniform distribution, the same probability holds for seeing x during actual execution:

$$\begin{aligned} Pr\left[VIEW_{S_i}^{v_j} = x\right] &= \frac{1}{2^{nk-1}} \\ &= Pr\left[Simulator_i = x\right] \end{aligned}$$

Therefore, the distribution of the simulator and the view is the same for the entire result vectors. Everything else is simulated exactly, so the views are computationally indistinguishable. Nothing is learned during the comparison phase.

The sites S_i now encrypt the result vectors; again the simulator mimics the actual protocol. Since the sources were indistinguishable, the results are as well.

The next step is to show that S_s learns nothing from receiving ER_i. Site S_s can generate the identifiers $ER_i[j].id$ it will receive, as in simulating the comparison phase. By the security definitions of encryption, the $E_c(v)$ must be computationally indistinguishable from randomly generated strings of the same length as the encrypted values, provided no two v are equal (which they cannot be, as discussed above.) Likewise, the definition of probabilistic encryption ensures that the $E_o(x)$ are computationally indistinguishable from randomly generated strings of the same length. Since E_c and E_o are public, S_s knows the length of the generated strings. The simulator chooses a random string from the domain of E_c and E_o; the result is computationally indistinguishable from the view seen during execution of the protocol. (If S_s is one of the S_i, the simulator must reuse the ER_s generated during the comparison simulation instead of generating a new one.)

C receives $n * k$ tuples consisting of an identifier, an encrypted comparison set v, and encrypted class value $E_o(c)$. Since the identifiers are created with an encryption key unknown to C, the values are computationally indistinguishable from random values. The simulator for C randomly selects $k * n$ identifiers from a uniform distribution on the domain of ER_i. The outcomes $E_o(c)$ are simulated the same as by S_s above. The hardest part to simulate is the comparison set. Since the comparison produces a total ordering, C cannot simply generate random comparison results. Instead, the simulator for C picks an identifier i_1 to be the closest, and generates a comparison set consisting of all the other identifiers and randomly chosen bits corresponding to the result shares. It then inserts into the comparison set for each other identifier i_k the tuple consisting of i_1 and the appropriate bit so that the comparison of i_1 with i_k will show i_1 as closest to q. For example, if $i_1 \geq i_k$, then $f(i_k, i_1)$ should be 1. If the bit for i_1's share is chosen to be 0, the tuple $(i_1, 1)$ is placed in i_k's comparison set. By the same argument used in the comparison phase, this simulator generates comparison values that are computationally indistinguishable from the view seen by C. Since the actual identifiers are computationally indistinguishable, and the value of the comparison is independent of the identifier value, the order of identifiers generated by C is computationally indistinguishable from the order in the real execution. The simulator encrypts these sets with E_c to simulate the data received.

In the final stage, O sees the $NR[i]$. The simulator for O starts with $NR_d[i] = c_i \oplus r_i$. The one-time pad r_i (unknown to O) ensures NR_d can be simulated by random strings of the length of $NR_d[i]$. Xor-ing the $NR_d[i]$ with r simulates NR_d. The final step reveals $E_o(c)$ to O, where c is the majority class. Since O knows the result c, the simulator generates $E_o(c)$ directly. Applying the composition theorem shows that the combination of the above simulation with the secure circuit evaluation is secure.

We have shown that there is a simulator for each site whose output is computationally indistinguishable from the view seen by that site during execution of the protocol. Therefore, the protocol is secure in the semi-honest model. $\square$

The algorithm actually protects privacy in the presence of malicious parties, providing O and C do not collude. The proof is omitted due to space restrictions.

4 Communication and Computation Cost Analysis

Privacy is not free. Assume m is the size required to represent the distance, and q bits are required to represent the result. A simple insecure distributed k-nn protocol would have the S_i send their k nearest neighbor distances/results to O, for $O(nk(m+q))$ bit communication cost. The computation by O could easily be done in $O(nk \log(k))$ comparisons. (One pass through the data, inserting each item into the appropriate place in the running list of the k nearest neighbors.) Although we do not claim this is optimal, it makes an interesting reference point.

In Protocol 1, each site performs k^2 comparisons with every other site. There are $\binom{n}{2}$ site combinations, giving $O(n^2k^2)$ comparisons. An m bit secure comparison has communication cost $O(mt)$, where t is based on the key size used for encryption. Thus, the total communication cost of the comparison phase of Protocol 1 is $O(n^2k^2mt)$ bits. Assuming Blum-Goldwasser encryption, each site then sends $O(nk+t)$ bits of encrypted comparison shares for each item, plus the $O(q+t)$ result, to S_s and on to C. This gives $O(n^2k^2+nkq+nkt)$ bits. C sends $O(k(q+t))$ bits of encrypted result to O. The dominating factor is the secure comparisons, $O(n^2k^2mt)$.

The computation cost is dominated by encryption, both direct and in the oblivious transfers (the dominating cost for secure circuit evaluation). There are $O(nk)$ encryptions of the query results, each of size q, and $O(nk)$ encryptions of comparison sets of size $O(nk)$. The dominating factor is again the $O(n^2k^2)$ secure comparisons. Each requires $O(m)$ 1 out of 2 oblivious transfers. An oblivious transfer requires a constant number of encryptions, giving $O(n^2k^2m)$ encryptions as the dominating computation cost. Assuming RSA public-key encryption for the oblivious transfer, the bitwise computation cost is $O(n^2k^2mt^3)$.

The parallelism inherent in a distributed system has a strong impact on the execution time. Since the secure comparisons may proceed in parallel, the time complexity $O(nk^2mt^3)$. Batching the comparisons between each pair of sites allows all comparisons to be done in a constant number of rounds. Thus, the dominating *time* factor would appear to be decryption of the nk comparison sets, each of size $O(nk)$. Note that m must be greater than $\log(nk)$ to ensure no equality in distances, so unless n is large relative to the other values the comparisons are still likely to dominate. Once decrypted, efficient indexing of the comparison vectors allows the same $O(nk\log(k))$ cost to determine the k nearest neighbor as in the simple insecure protocol described above.

A more interesting comparison is with a fully secure k-nn algorithm based directly on secure circuit evaluation. For n parties and a circuit of size C, the generic method requires $O(n^2C)$ 1 out of 2 oblivious transfers: a communication complexity $O(n^2Ct)$. To compare with the generic method, we need a lower bound on the size of a circuit for k-nn classification on nk $(m+q)$-bit inputs. An obvious lower bound is $\Omega(nk(m+q))$: the circuit must (at least) be able to process all data. This gives a bit complexity of $O(n^2nk(m+q)t)$. Our method clearly wins if $n > k$ and is asymptotically superior for fixed k; for $n \leq k$ the question rests on the complexity of an optimal circuit for k-nn classification.

5 Conclusions

We have presented a provably secure algorithm for computing k-nn classification from distributed sources. The method we have presented is not cheap – $O(n^2k^2)$ where n is the number of sites – but when the alternative is not performing the task at all due to privacy concerns, this cost is probably acceptable. This leads us to ask about lower bounds: can we *prove* that privacy is not free. Privacy advocates often view privacy versus obtaining knowledge from data as an either/or situation. Demonstrating that knowledge can be obtained while maintaining privacy, and quantifying the associated costs, enables more reasoned debate.

References

1. Korn, F., Sidiropoulos, N., Faloutsos, C., Siegel, E., Protopapas, Z.: Fast nearest neighbor search in medical image databases. In Vijayaraman, T.M., Buchmann, A.P., Mohan, C., Sarda, N.L., eds.: Proceedings of 22th International Conference on Very Large Data Bases, Mumbai (Bombay), India, VLDB, Morgan Kaufmann (1996) 215–226
2. Fukunaga, K.: Introduction to statistical pattern recognition (2nd ed.). Academic Press Professional, Inc. (1990)
3. : Special section on privacy and security. SIGKDD Explorations **4** (2003) i–48
4. Yao, A.C.: How to generate and exchange secrets. In: Proceedings of the 27th IEEE Symposium on Foundations of Computer Science, IEEE (1986) 162–167
5. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game - a completeness theorem for protocols with honest majority. In: 19th ACM Symposium on the Theory of Computing. (1987) 218–229
6. Feige, U., Kilian, J., Naor, M.: A minimal model for secure computation. In: 26th ACM Symposium on the Theory of Computing (STOC). (1994) 554–563
7. Goldreich, O.: General Cryptographic Protocols. In: The Foundations of Cryptography. Volume 2. Cambridge University Press (2004)
8. Blum, M., Goldwasser, S.: An efficient probabilistic public-key encryption that hides all partial information. In Blakely, R., ed.: Advances in Cryptology – Crypto 84 Proceedings, Springer-Verlag (1984)