The VLDB Journal (2008) 17:879–898
DOI 10.1007/s00778-006-0041-y

REGULAR PAPER

Privacy-preserving Naïve Bayes classification

Jaideep Vaidya · Murat Kantarcıoğlu · Chris Clifton

Received: 30 September 2005 / Revised: 15 March 2006 / Accepted: 25 July 2006 / Published online: 3 February 2007
© Springer-Verlag 2007

Abstract Privacy-preserving data mining—developing models without seeing the data – is receiving growing attention. This paper assumes a privacy-preserving *distributed* data mining scenario: data sources collaborate to develop a global model, but must not disclose their data to others. The problem of secure distributed classification is an important one. In many situations, data is split between multiple organizations. These organizations may want to utilize all of the data to create more accurate predictive models while revealing neither their training data/databases nor the instances to be classified. Naïve Bayes is often used as a baseline classifier, consistently providing reasonable classification performance. This paper brings privacy-preservation to that baseline, presenting protocols to develop a Naïve Bayes classifier on both vertically as well as horizontally partitioned data.

Keywords Data mining · Privacy · Security · Naïve Bayes · Distributed computing

This material is based upon work supported by the National Science Foundation under Grant No. 0312357. Portions of this work were also supported by a Rutgers Research Resources committee award.

J. Vaidya (✉)
Rutgers University, Newark, NJ, USA
e-mail: jsvaidya@rbs.rutgers.edu

M. Kantarcıoğlu
University of Texas at Dallas, Dallas, TX, USA
e-mail: muratk@utdallas.edu

C. Clifton
Purdue University, West Lafayette, IN, USA
e-mail: clifton@cs.purdue.edu

1 Introduction

There has been growing concern over the privacy implications of data mining. Some of this is public perception—the "Data Mining Moratorium Act of 2003" introduced in the US Senate [14] was based on a fear of Government searches of private data for individual information, rather than what the technical community views as Data Mining. However, concerns remain. While data mining is generally aimed at producing general models rather than learning about specific individuals, the *process* of data mining creates integrated data warehouses that pose real privacy issues. Data that is of limited sensitivity by itself becomes highly sensitive when integrated, and gathering the data under a single roof greatly increases the opportunity for misuse.

Our solution to this problem is to avoid disclosing data beyond its source, while still constructing data mining models equivalent to those that would have been learned on an integrated data set. Since we prove that data is not disclosed beyond its original source, the opportunity for misuse is not increased by the process of data mining. This concept was first proposed for decision tree classification [28]; protocols have also been developed for association rule mining [24,38] and clustering [26,39].

The definition of privacy followed in this line of research is conceptually simple: no site should learn anything new from the *process* of data mining. Specifically, anything learned during the data mining process must be derivable given one's own data and the final result—nothing is learned about any other site's data that is not inherently obvious from the data mining result. The approach followed in this research has been to select a type of data mining model to be learned and develop a protocol to learn the model while meeting this definition of privacy.

Naïve Bayes is a simple but highly effective classifier. This combination of simplicity and effectiveness has lead to its use as a baseline standard by which other classifiers are measured. With various enhancements it is highly effective, and receives practical use in many applications (e.g., text classification [29]). This paper extends the portfolio of privacy-preserving distributed data mining to include this standard classifier.

In addition to the type of data mining model to be learned, the different types of data distribution result in a need for different protocols. For example, the first decision tree paper in this area proposed a solution for learning decision trees on horizontally partitioned data: each site has complete information on a distinct set of entities, an integrated dataset consists of the union of these datasets. In contrast, vertically partitioned data has different types of information at each site; each has partial information on the same set of entities. In this case an integrated dataset would be produced by *joining* the data from the sites. While [27] showed how to generate ID3 decision trees on horizontally partitioned data, a completely new method was needed for vertically partitioned data [10]. This paper proposes solutions for both vertically partitioned, as well as horizontally partitioned, data. For vertically partitioned data, our solution does not even reveal the model. It only reveals the class value of an individual instance when the classification is performed.

Privacy-preserving classification has many real-world applications. Consider a medical research study which wants to compare medical outcomes of different treatment methods of a particular disease (e.g., to answer the question "will this treatment for this patient be successful or not?"). The insurance companies must not disclose individual patient data without permission [18], and details of patient treatment plans are similarly protected data held by hospitals. Similar constraints arise in many applications; European Community legal restrictions apply to disclosure of any individual data [11]. Here, we require classification models on vertically partitioned data. On the other hand, consider the case of several banks which decide to leverage all of their transaction data to identify fraudulent credit card usage, or insurance companies which jointly try to identify high-risk customers. In both of these situations, the kinds of data collected is the same, though it is collected for a different set of entities. This is a situation which requires classification models on horizontally partitioned data.

In Sect. 2, we briefly describe the Naïve Bayes classifier. Section 3 gives references to prior work in this area as well as giving the background necessary to understand the protocols, including definitions from Secure Multiparty Computation. We then present the model, algorithm and proof of security for horizontally partitioned data in Sect. 4 and for vertically partitioned data in Sect. 5. Section 6 concludes the paper. Appendix provides a brief description of prior secure protocols utilized as subroutines.

2 The Naïve Bayes classifier

The *Naïve Bayes classifier* is a highly practical Bayesian learning method. The following description is based on the discussion in Mitchell [29]. The Naïve Bayes classifier applies to learning tasks where each instance x is described by a conjunction of attribute values and the target function $f(x)$ can take on any value from some finite set C. A set of training examples of the target function is provided, and a new instance is presented, described by the tuple of attribute values $\langle a_1, a_2, \ldots, a_n \rangle$. The learner is asked to predict the target value, or classification, for this new instance.

The Bayesian approach to classifying the new instance is to assign the most probable target value, c_{MAP}, given the attribute values $\langle a_1, a_2, \ldots, a_n \rangle$ that describe the instance.

$$c_{\text{MAP}} = \underset{c_j \in C}{\operatorname{argmax}} \left(P(c_j | a_1, a_2, \ldots, a_n) \right) \tag{1}$$

Using Bayes theorem,

$$\begin{aligned} c_{\text{MAP}} &= \underset{c_j \in C}{\operatorname{argmax}} \left(\frac{P(a_1, a_2, \ldots, a_n | c_j) P(c_j)}{P(a_1, a_2, \ldots, a_n)} \right) \\ &= \underset{c_j \in C}{\operatorname{argmax}} \left(P(a_1, a_2, \ldots, a_n | c_j) P(c_j) \right) \end{aligned} \tag{2}$$

The Naïve Bayes classifier makes the further simplifying assumption that the attribute values are conditionally independent given the target value. Therefore,

$$c_{\text{NB}} = \underset{c_j \in C}{\operatorname{argmax}} \left(P(c_j) \prod_i P(a_i | c_j) \right) \tag{3}$$

where c_{NB} denotes the target value output by the Naïve Bayes classifier.

The conditional probabilities $P(a_i | c_j)$ need to be estimated from the training set. The prior probabilities $P(c_j)$ also need to be fixed in some fashion (typically by simply counting the frequencies from the training set). The probabilities for differing hypotheses (classes) can also be computed by normalizing the values received for each hypothesis (class).

Probabilities are computed differently for nominal and numeric attributes.

2.1 Nominal attributes

For a nominal attribute X with r possible attributes values $x_1, \ldots, x_r$, the probability $P(X = x_k|c_j) = \frac{n_{kj}}{n}$ where n is the total number of training examples for which $C = c_j$, and n_{kj} is the number of those training examples that also have $X = x_k$.

2.2 Numeric attributes

In the simplest case, numeric attributes are assumed to have a "normal" or "Gaussian" probability distribution.

The probability density function for a normal distribution with mean μ and variance σ^2 is given by

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4}$$

The mean μ and variance σ^2 are calculated for each class and each numeric attribute from the training set. Now the required probability that the instance is of the class c_j, $P(X = x'|c_j)$, can be estimated by substituting $x = x'$ in Eq. (4).

3 Related work in privacy-preserving computation

Recently, there has been significant interest in the area of privacy-preserving data mining. We briefly cover some of the relevant work. Several solution approaches have been suggested. One approach is to add "noise" to the data before the data mining process, and then use reconstruction techniques that mitigate the impact of the noise from the data mining results [1,2,13,37]. However,there is some debate about the security properties of such algorithms—Kargupta et al. pointed out an important issue: arbitrary randomization is not safe [25]. Huang et al. [19] further explore this issue as well and propose two new data reconstruction methods based on data correlations.

The data distortion approach addresses a different problem from our work. The assumption with distortion is that values must be kept private from the data mining party. We instead assume that *some* parties are allowed to see *some* of the data, while no one is allowed to see *all* the data. In return, *exact* results can be obtained.

The other approach uses cryptographic techniques to protect privacy. This approach of protecting privacy of distributed sources was first used for the construction of decision trees [28]. 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 off computation and communication cost for accuracy, improving efficiency over the generic secure multiparty computation method. There has since been work to address association rule mining [24,42], clustering [21,26,39], classification [10,40,41,43]. and generalized approaches to reducing the number of "on-line" parties required for computation [22]. While some of this work makes trade-offs between efficiency and information disclosure, all maintain provable privacy of individual information and bounds on disclosure, and disclosure is limited to information that is unlikely to be of practical concern.

We have made use of several primitives from the Secure Multiparty Computation literature. Recently, there has been a renewed interest in this field, a good discussion can be found in [9]. Currently, assembling these into efficient privacy-preserving data mining algorithms, and proving them secure, is a challenging task. This paper demonstrates how these can be combined to implement a standard data mining algorithm with provable privacy and information disclosure properties.

3.1 Secure multiparty computation

To prove that our Naïve Bayes algorithm preserves privacy, we need to define privacy preservation. We use the framework defined in *Secure Multiparty Computation*.

Yao first postulated the two-party comparison problem (Yao's Millionaire Protocol) and developed a provably secure solution [44]. This was extended to multiparty computations by Goldreich et al. [17].

We start with the definitions for security in the semi-honest model. A semi-honest party follows the rules of the protocol using its correct input, but is free to later use what it sees during execution of the protocol to compromise security. A formal definition of private m-party computation in the semi-honest model is given below.

Definition 1 (privacy w.r.t. semi-honest behavior): [16]

Let $f : (\{0,1\}^*)^m \longmapsto (\{0,1\}^*)^m$ be a probabilistic, polynomial-time functionality, where $f_i(x_1, x_2, \ldots, x_m)$ denotes the ith component of $f(x_1, x_2, \ldots, x_m)$ and let Π be m-party protocol for computing f. For $I = \{i_1, i_2, \ldots, i_t\} \subseteq [m]$ where $[m]$ denotes the set $\{1, 2, \ldots, m\}$, we let $f_I(x_1, x_2, \ldots, x_m)$ denote the subsequence $f_{i_1}(x_1, x_2, \ldots, x_m), \ldots, f_{i_t}(x_1, x_2, \ldots, x_m)$. Let the view of the ith party during an execution of protocol Π on $\bar{x} = (x_1, x_2, \ldots, x_m)$, denoted $\text{view}_i^\Pi(\bar{x})$ be $(x_i, r_i, m_i^1, \ldots, m_i^t)$ where r_i represents the outcome of the ith party's internal coin tosses, and m_i^j represents the jth message received by the ith party. Also for given $I = \{i_1, i_2, \ldots, i_t\}$, we let $\text{view}_I^\Pi(\bar{x})$ denote the subsequence $\left(I, \text{view}_{i_1}^\Pi(\bar{x}) \cdots \text{view}_{i_t}^\Pi(\bar{x})\right)$.

We say that Π privately computes f if there exists a probabilistic polynomial time algorithm denoted S such that for every $I \subseteq [m]$, it holds that

$$\left\{\left(S\left(I, \left(x_{i_1}, \ldots, x_{i_t}\right), f_I(\bar{x})\right), f(\bar{x})\right)\right\}_{\bar{x} \in (\{0,1\}^*)^m} \overset{C}{\equiv} \left\{\left(\text{view}_I^\Pi(\bar{x}), \text{output}^\Pi(\bar{x})\right)\right\}_{\bar{x} \in (\{0,1\}^*)^m}$$

where $\overset{C}{\equiv}$ denotes computational indistinguishability and $\text{output}^\Pi(\bar{x})$ denotes the output sequence of all parties during the execution represented in the $\text{view}_I^\Pi(\bar{x})$.

The above definition states that a computation is secure if the view of the dishonest parties (combined view, as dishonest parties may collude) during the execution of the protocol can be effectively simulated knowing only the input and the output of the dishonest parties. This is not quite the same as saying that private information is protected. If information can be deduced from the final result, it is obviously not kept private under this definition. For example, if two entities that differ in only one attribute are classified differently, information about how that attribute affects classification is revealed. This cannot be helped, as this information can always be deduced from the result and the input.

While the semi-honest model may seem questionable for privacy (if a party can be trusted to follow the protocol, why do not we trust them with the data?), we believe that it meets several practical needs for early adoption of the technology. Consider the credit card fraud example given in Sect. 1. In many cases the parties involved already have authorization to see the data (e.g., the recent theft of credit card information from CardSystems [35] involved data that CardSystems was expected to see during processing). The problem is that *storing* the data brings with it a responsibility (and cost) of protecting that data; CardSystems was supposed to delete the information once processing was complete. If parties could develop the desired models without seeing the data, then they are saved the responsibility (and cost) of protecting it.

This provides a practical motivation for the semi-honest model: parties will follow the protocol to avoid seeing data they do not need, saving them the responsibility of protecting the data from unauthorized disclosure. Also in this paper, we consider only the protocols that are secure under no collusion assumption. In other words, we only consider the cases where the size of I (as defined in Definition 1) is one. The above argument also justifies the use of no-collusion assumption and the resulting loss of protection against collusion among the parties. Until the technology is proven, use where all parties are not authorized to see the data is unlikely (e.g., medical research will still require institutional review board approval for all parties). The simplicity and efficiency possible with semi-honest protocols will help speed adoption so that trusted parties are saved the expense of protecting data other than their own. As the technology gains acceptance, malicious protocols will become viable for uses where the parties are not mutually trusted. In any case, we do discuss the effect of collusion on our protocols at the appropriate place. This allows parties to know what is at risk and choose whether to tolerate the risk.

In our later proofs, we use a key result from the secure multiparty computation literature, the composition theorem. We state it here for the semi-honest model. A detailed discussion of this theorem, as well as the proof, can be found in [16].

Theorem 1 (Composition theorem for the semi-honest model) *Suppose that g is privately reducible to f and that there exists a protocol for privately computing f. Then there exists a protocol for privately computing g.*

Proof Refer to [16]. □

In the above definition, "g is privately reducible to f" implies that we can come up with a private protocol for evaluating function g given a black box access to function f. This allows us to use existing secure protocols as components, worrying only that their results do not reveal anything and ignoring the communications carried on by those components.

4 Privacy-preserving Naïve Bayes for horizontally partitioned data

In this section, we will focus on securely learning a Naïve Bayesian classifier on horizontally partitioned data. Records of different patients that are treated by different hospitals can be seen as an example of horizontally partitioned data. All of the information for a given patient is contained at one hospital, but different hospitals have different patients.

In order to see how a privacy-preserving Naïve Bayesian classifier is constructed, we need to address two issues: how to select the model parameters and how to classify a new instance.The following subsections provide details on both issues. The protocols presented below are very efficient. However, they compromise a little on security. At the end of the protocol, all parties learn the total number of instances. In effect, they learn the numerator and denominator for all the fractions computed. For multiple parties, this may not be a serious privacy concern. However, we also present a technical solution to this problem. Thus, in Sect. 4.4, we present methods which do not reveal anything except the final classifier.

For horizontally partitioned data model, all the attributes needed for classifying an instance are held by one site. Therefore, given the model, no collaboration is needed for classifying a new instance. At the first glance, it appears that a more secure solution can be obtained by hiding the model parameters and using a secure protocol to classify each new instance. Unfortunately, this costly option (i.e., using a secure distributed protocol for classifying a new instance) does not add much to security. Since each site has knowledge of all of the attributes, eventually by classifying sufficiently many instances, a site may learn/infer the hidden model quite easily. Therefore, we do not try to hide the model in the horizontally partitioned data case. Instead, we simply try to learn the Naive Bayes classifier without revealing anything else.

For vertically partitioned case, different parties must collaborate to find the classification result for every new instance (we stress that no party has all the attributes for a given instance). Since parties do not know all the attributes of a new instance, they will not be able to predict the full model even by classifying many instances. Therefore, hiding the model brings additional security for vertically partitioned data and is necessary.

4.1 Building the classifier model

The procedures for calculating the parameters are different for nominal attributes and numeric attributes. They are described in the subsections below.

4.1.1 Nominal attributes

For a nominal attribute, the conditional probability that an instance belongs to class c given that the instance has an attribute value $A = a$, $P(C = c|A = a)$, is given by

$$P(C = c|A = a) = \frac{P(C = c \cap A = a)}{P(A = a)} = \frac{n_{ac}}{n_a}. \tag{5}$$

n_{ac} is the number of instances in the (global) training set that have the class value c and an attribute value of a, while n_a is the (global) number of instances which simply have an attribute value of a. The necessary parameters are simply the counts of instances, n_{ac} and n_a. Due to horizontal partitioning of data, each party has partial information about every attribute. Each party can locally compute the local count of instances. The global count is given by the sum of the local counts. Securely computing a global count is straightforward. (See secure summation in Appendix A.1.) Assuming that the total number of instances is public, the required probability can be computed by dividing the appropriate global sums. Note that local number of instances will not be revealed. Protocol 1 formally defines the protocol.

For an attribute a with l different attribute values, and a total of r distinct classes, $l \cdot r$ different counts need to be computed for each combination of attribute value and class value. For each attribute value a total instance count also needs to be computed, which gives l additional counts.

Algorithm 1 Nominal Attributes

```
Require: $k$ parties, $r$ class values, $l$ attribute values
 1: {$c^x_{yz}$ represents #instances with party $P_x$ having class $y$ and
    attribute value $z$}
 2: {$n^x_y$ represents #instances with party $P_x$ having class $y$}
 3: {$p_{yz}$ represents the probability of an instance having class $y$
    and attribute value $z$}
 4: for all class values $y$ do
 5:   for $i = 1, \ldots, k$ do
 6:     $\forall z$, Party $P_i$ locally computes $c^i_{yz}$
 7:     Party $P_i$ locally computes $n^i_y$
 8:   end for
 9: end for
10: $\forall(y, z)$, All parties calculate using the secure sum
    protocol (see Appendix A.1),  $c_{yz} = \sum_{i=1}^k c^i_{yz}$
11: $\forall y$, All parties calculate using secure sum protocol,
    $n_y = \sum_{i=1}^k n^i_y$
12: All parties calculate $p_{yz} = c_{yz}/n_y$
```

4.1.2 Numeric attributes

For a numeric attribute, the necessary parameters are the mean μ and variance σ^2 for each class. Again, the necessary information is split between the parties. To compute the mean, each party needs to sum the attribute values of the appropriate instances having the same class value. These local sums are added together and divided by the total number of instances having that same class to get the mean for that class value. Once all of the means μ_y are known, it is quite easy to compute the variance σ^2_y, for all class values. Since each party knows the classification of the training instances it has, it can subtract the appropriate mean μ_y from an instance having class value y, square the value, and sum all such values together. The global sum divided by the global number of instances having the same class y gives the required variance σ^2_y. Protocol 2 formally describes the protocol.

Algorithm 2 Numeric Attributes

```
1: {$x_{iyj}$ represents the value of instance $j$ from party $i$ having class value $y$}
2: {$s^i_y$ represents the sum of instances from party $i$ having class value $y$}
3: {$n^i_y$ represents #instances with party $P_i$ having class value $y$}
4: for all class values $y$ do
5:   for $i = 1, \ldots, k$ do
6:     Party $P_i$ locally computes $s^i_y = \sum_j x_{iyj}$
7:     Party $P_i$ locally computes $n^i_y$
8:   end for
9:   All parties calculate using secure sum protocol (see Appendix A.1), $s_y = \sum_{i=1}^k s^i_y$
10:  All parties calculate using secure sum protocol, $n_y = \sum_{i=1}^k n^i_y$
11:  All parties calculate $\mu_y = s_y/n_y$
12: end for
13: {Create $\mathbf{V} = (\mathbf{X} - \mu)^2$}
14: for $i = 1, \ldots, k$ do
15:   $\forall j, v_{iyj} = x_{iyj} - \mu_y$
16:   $\forall j, v_{iy} = \sum_j (v^2_{iyj})$
17: end for
18: $\forall y$, All parties calculate using secure sum protocol,
    $v_y = \sum_{i=1}^k v_{iy}$
19: All parties calculate $\sigma^2_y = \frac{1}{n_y - 1} \cdot v_y$
```

4.2 Evaluating an instance

Since all the model parameters are completely present with all the parties, evaluation is simple. The party that wants to evaluate an instance simply uses the Naïve Bayes evaluation procedure locally to classify the instance. The other parties have no interaction in the process. Thus, there is no question of privacy being compromised.

4.3 Proof of security

To prove that the protocols are secure, we utilize the definitions given in the earlier section on secure multiparty computation.

Theorem 2 *Protocol 1 securely computes the probabilities p_{yz} without revealing anything except the probability p_{yz}, the global count c_{yz} or the global number of instances n_y.*

Proof The only communication taking place is at steps 11 and 12. During these steps, the secure sum algorithm is invoked to compute the global counts c_{yz} and n_y. We apply the composition theorem stated in Theorem 1, with g being the nominal attribute computation algorithm and f being the secure sum algorithm. □

Theorem 3 *Protocol 2 securely computes the means μ_y and variance σ^2_y without revealing anything except μ_y, σ^2_y, the global sum of instance values for each class s_y and the global number of instances n_y, as also the sum v_y.*

Proof The only communication takes place at steps 9, 10 and 18. At all three of these steps the secure sum algorithm is invoked to compute s_y, n_y and v_y.Thus, again, we simply apply the composition theorem stated in Theorem 1, with g being the numeric attribute computation algorithm and f being the secure sum algorithm. □

4.4 Enhancing security

The protocols given above are not completely secure in the sense that something more than just the model parameters are revealed. The true numerators and the denominators making up the actual parameter values are revealed. For three or more parties, this allows upper bounds on the number of instances with a party and upper bounds on the composition of those instances (i.e., upper bound on the number belonging to a particular class, etc.). Privacy of individual instances is always preserved. With an increasing number of parties, it is more difficult to get accurate estimates of the remaining parties. However, with just two parties, this does reveal quite a bit of extra information. In general, the problem is to calculate the value of the fraction without knowing the shared numerator and/or shared denominator.

Note that the amount of information revealed for multiple parties is not much more than what the parameters themselves reveal. However, technical solutions (even with increased cost) are more satisfying as they allow an individual decision of whether to trade off security for efficiency. In the following subsection, we now present a secure protocol based on computing the logarithm securely.

4.5 Secure-logarithm- based approach

As mentioned above, to make our algorithm fully secure (i.e., reveal nothing),we need to evaluate $(\sum_{i=1}^k c_i / \sum_{i=1}^k n_i)$ securely. Here evaluating the division becomes the main problem. In order to overcome this problem, we can rewrite the above expression as follows:

$$\exp\left[\ln\left(\sum_{i=1}^k c_i\right) - \ln\left(\sum_{i=1}^k n_i\right)\right]$$

Then evaluating $\ln(\sum_{i=1}^k c_i) - \ln(\sum_{i=1}^k n_i)$ securely is sufficient. Clearly, this requires secure evaluation of $\ln(\sum_{i=1}^k x_i)$ function. In our work, we will use the secure $\ln(x)$ evaluation method given in [28]. The one important restriction of their method is that it only works for two parties. In our case, it is easy to reduce the k party problem to the two-party case. Note that the last step in the semi-honest version of the secure summation protocol

Algorithm 3 Fully secure approach for nominal attributes

```
Require: $k$ parties, $r$ class values, $l$ attribute values
1: $\{c^x_{yz}, n^x_y, p_{yz}$ are defined as in Protocol 1$\}$
2: for all class values $y$ do
3:   for $i = 1, \ldots, k$ do
4:     $\forall z$, Party $P_i$ locally computes $c^i_{yz}$
5:     Party $P_i$ locally computes $n^i_y$
6:   end for
7: end for
8: $\forall(y,z)$, All parties, use secure sum protocol (see
   Appendix A.1) until last step for finding
   $c_{yz} = \sum_{i=1}^{k} c^i_{yz}$ and $n_y = \sum_{i=1}^{k} n^i_y$
9: Let party 1 have $R_c$ and $R_n$
10: Let party $k$ have $R_c + c_{yz} \bmod p$ and $R_n + n_y \bmod p$
11: {Note that last step of the summation has not been executed}
12: Using secure $\ln(x)$ protocol, party 1 and $k$ gets random $v_1$,$v_k$
    s.t.,
    $v_1 + v_k = C \cdot \ln(R_c + c_{yz} - R_c \bmod p) \bmod p$
13: Using secure $\ln(x)$ protocol, party 1 and $k$ gets random $u_1$,$u_k$
    s.t.,
    $u_1 + u_k = C \cdot \ln(R_n + n_y - R_n \bmod p) \bmod p$
14: Party $k$ calculates $s_k = v_k - u_k \bmod p$ and sends it to party 1
15: Party 1 calculates the $s_1 = s_k + v_1 - u_1 \bmod p$
16: All parties calculate $p_{yz} = \exp(s_1/C)$
```

has the first party subtracting the random number from the result. So just before this subtraction occurs, no party has the summation and nothing is revealed. At this point, instead of subtracting the random number, both parties can use the secure approximate $\ln(x_1+x_2)$ protocol given in [28]. Using their protocol, it is easy to get random v_1, v_2 such that $v_1 + v_2 = C \cdot \ln(x_1 + x_2) \bmod p$.

One important fact about the secure $\ln(x)$ evaluation algorithm is that there is a public constant C used to make all elements integral. The method for determining C is given in [28]. Also, operations are executed in a field with size p that is capable of containing the actual results multiplied by the constant. Our reduction requires us to slightly change the protocol. In [28], protocol $x_1 + x_2$ is directly added using a small addition circuit. In our case we use modular addition. (This does not change the asymptotic performance of the method.) After using secure logarithm, it is easy to evaluate our desired function securely. Protocol 3 describes how these ideas can be applied to our problem. Here, we only give the protocol for nominal attributes, it is straightforward to extend this to continuous attributes.

Theorem 4 *Protocol 3 securely evaluates p_{yz} in semi-honest model (assuming no collusion between parties).*

Proof To show that the above protocol is secure in semi-honest model where there is no collusion between parties, we will show that each party's view of the protocol can be simulated based on its input and its output. Again, we will use the secure composition theorem to prove the entire protocol secure since our method securely reduces to the logarithm function.

Parties $2, \ldots, k-2$ only see a summation added to some random number. Therefore, as earlier, the simulator for these parties will be a uniform number generator. Note that the probability that they will see some number x during the execution is $\frac{1}{p}$. The simulator will generate the number with the same probability.

For parties 1 and k, there is the additional step of computing the logarithm. We have to show that this does not reveal anything either. Assume that logarithm protocol returns random shares of the result.

Now let us define the simulator for the party k. Clearly, before the logarithm protocol has started, party k has $R_n + n_y \bmod p$ and $R_c + c_{yz} \bmod p$. These are indistinguishable from a random number drawn from an uniform distribution. The execution of the logarithm protocol can be simulated by using the simulator for the logarithm protocol. The details for this simulator can be found in [28]. After the protocol, party k only sees u_2, v_2 which are also indistinguishable from uniform distribution. Therefore the messages it sees during the protocol can be easily generated by an uniform random number generator.

If we look at the messages received by the party 1, one set of messages come from the execution of logarithm, then it receives random shares of u_1, v_1. Also it receives $(u_2 - v_2) \bmod p$. We can define the simulator for k as follows: first it runs the simulator of the logarithm function, then it generates three random numbers uniformly chosen between 0 and $p-1$. Note that u_2, v_2 are independent and $u_2 - v_2 \bmod p$ is also uniformly distributed, as

$$\begin{aligned}
&\Pr(u_2 - v_2 = k \bmod p)\\
&\quad = \sum_{v=0}^{p-1} \Pr(u_2 = k + v + v \bmod p | v_2 = v) \cdot \Pr(v_2 = v)\\
&\quad = \sum_{v=0}^{p-1} \Pr(u_2 = k + v \bmod p) \cdot \Pr(v_2 = v)\\
&\quad = \sum_{v=0}^{p-1} \frac{1}{p^2} = \frac{1}{p}
\end{aligned}$$

This concludes the proof. □

4.5.1 *Effect of collusion on privacy*

The solution described above assumes no collusion among parties. This assumption could be easily relaxed for the summation part by using the techniques described in Appendix A.1. Those techniques allow us to calculate c_{yz} and n_y privately even if up to $k-1$ parties collude. (If all k parties collude, they would probably share data outside the protocol and privacy protection is irrelevant.)

For the secure logarithm part, it is harder to protect against collusion. Note that before the secure logarithm evaluation, the two parties hold the shares of the summation results. Clearly, if those two parties collude, they can learn the c_{yz} and n_y values that we want to protect. At the same time, if we use a secure summation protocol that can protect against collusions that involve $k-1$ parties, then we can easily prove that colluding parties 1 and k can at most learn c_{yz} and n_y values.

To prevent even the revelation of c_{yz} and n_{yz} under collusion, it would be worth developing a practical $\ln(x)$ protocol that can handle more than two shares; this is an interesting challenge for future work.

4.5.2 Communication and computation cost

Privacy is not free. To evaluate the secure logarithm, we need to make $O(\log(p))$ oblivious transfers for a total of $O(\log(p)\cdot t)$ bits.(p is the size of the field used and depends on the total database size and the precision required in calculating the logarithm; t is the security parameter.) Based on the precision required for the logarithm protocol (let u be the number of terms used in the Taylor approximation), we need to do a secure polynomial evaluation with degree u for evaluating each p_{yz}.

Therefore, total number of bits transferred will be $O(\log(p)\cdot(t+k))$, where k is the number of parties. Since oblivious transfer and secure polynomial evaluation is much more expensive than addition, the $O(\log(p))$ oblivious transfers plus one secure polynomial evaluation with degree u will dominate the computation cost.

We can give a reasonable estimate of the computational cost by looking at the details of the protocol given in [28]. For estimation purposes, let us assume that total size of union of all the databases is T. Let $\tilde{\ln}(x)$ be the approximate result of the secure logarithm protocol. From [28], we know that the difference between the actual result and the secure approximation is, $|\ln(x)-\tilde{\ln}(x)| \le \frac{1}{2^u.(u+1)}$. Since in our case, we try to estimate $\exp\left(\ln(c_{yz})-\ln(n_y)\right)$ by calculating $\exp\left(\tilde{\ln}(c_{yz})-\tilde{\ln}(n_y)\right)$. We can make the relative error arbitrarily small by choosing an approximating polynomial with appropriate degree u. More precisely, relative error re can be written as

$$\text{re} = \left|\frac{\exp\left(\ln(c_{yz})-\ln(n_y)\right)-\exp\left(\tilde{\ln}(c_{yz})-\tilde{\ln}(n_y)\right)}{\exp\left(\ln(c_{yz})-\ln(n_y)\right)}\right|$$

$$= \left|1-\exp\left(\tilde{\ln}(c_{yz})-\ln(c_{yz})+\ln(n_y)-\tilde{\ln}(n_y)\right)\right|$$

$$\le \left|1-\exp\left(\frac{2}{2^u.(u+1)}\right)\right|$$

In the last step of the above equation, we used the fact that $|\ln(x)-\tilde{\ln}(x)| \le \frac{1}{2^u.(u+1)} \le 0.5$. Clearly, u can be adjusted to bound the re value.

Now given the database size T (i.e., number of tuples in the database), desired upper bound on re, and security parameter t, we can estimate the actual computational cost. As we mentioned before, secure summation only involves simple summations, therefore the computation cost will be dominated by the computation of the $\exp\left(\tilde{\ln}(c_{yz})-\tilde{\ln}(n_y)\right)$ assuming that $k<1{,}000$. (Note that an exponentiation for reasonable t values is at least 1,000 times more expensive than an addition.) For each evaluation of the secure $\tilde{\ln}(x)$, we need to do $2\cdot\lceil\log(T)\rceil$ one out of two oblivious transfers. Each oblivious transfer requires four exponentiations. (See the appendix for details.) Each polynomial evaluation with degree u requires u exponentiations. Therefore, for each evaluation of $\tilde{\ln}(x)$, we need to do $8\cdot\lceil\log(T)\rceil+u$ exponentiations with t bit long numbers. Therefore, we can estimate the total computation cost for calculating p_{yz} as $16\cdot\lceil\log(T)\rceil+2u$ exponentiations.

We estimated the cost of an exponentiation using the GMP library on an Intel Pentium Dual 830, 3.00 GHZ with 2 GB ram. The exponentiation time for $t=512$ was 0.0015 s, for $t=1{,}024$ it was 0.0104 s.

The following table summarizes the estimated total exponentiation computation time for the p_{yz} for two different total database sizes $(10^6, 10^8)$; two different u values 10, 20; and with respective relative error bounds $1.8\times10^{-4}, 9.1\times10^{-8}$.

Looking at Table 1, we can also easily estimate the effect of accuracy on the computation time. These results indicate that the reduction in the relative error from 1.8×10^{-4} to 9.1×10^{-8} on the average requires 5% increase in the computation time. Since for practical purposes relative error 9.1×10^{-8} is sufficient, the effect of increased accuracy is not too significant in overall computation time. Since the total computation cost for calculating p_{yz} can be estimated as $16\cdot\lceil\log(T)\rceil+2u$ exponentiations, increasing the accuracy (i.e., increasing the u) will not significantly increase the overall cost for large data sets (i.e., for large T values).

Compared to the non-secure version of the p_{yz} calculation that involves only k additions and a division, the secure version is clearly much slower. On the other hand, our estimation indicates that calculating p_{yz} privately in practice is feasible.

Table 1 Estimated computation time for p_{yz}

Security parameter	Number of tuples	Degree of the polynomial	Estimated time (s)
512	10^6	10	0.51
512	10^6	20	0.54
512	10^8	10	0.68
512	10^8	20	0.71
1024	10^6	10	3.54
1024	10^6	20	3.74
1024	10^8	10	4.70
1024	10^8	20	4.91

5 Privacy-preserving Naïve Bayes for vertically partitioned data

This section addresses the issue of classification over vertically partitioned data. With vertically partitioned data, different sites hold different attributes. We have already discussed real-life situations where this is applicable. One issue of particular interest with classification is the location and security properties of the class attribute. We can divide this into two possibilities:

- All the parties hold the (common/public) class attribute, or
- Only a subset of the parties have the (secret) class attribute.

The first case is the simplest, assuming that the class attribute of the training data is known to all parties. In some cases this is reasonable—e.g., manufacturers of subcomponents collaborating to determine expected failure rates of fully assembled systems based on attributes of the subcomponents. In this case, it is easy to estimate all the required counts for nominal attributes and means and variances for numeric attributes locally, causing no privacy breaches. Prediction can be accomplished by independently estimating the probabilities, and securely multiplying and comparing to obtain the predicted class.

More interesting is the general case, where not all parties have the class attribute. We can simplify this to the basic case where one party has the class attribute and the other has the remaining attributes. Imagine, as before, manufacturers of subcomponents; but in this case a manufacturer wants to analyze failure of its subcomponent when used as part of the full system. While all parties would like to see overall failure rates drop, most would prefer not to disclose their own problems to get this to happen. Solving this enables us to solve any distribution of attributes. (Extension to more than two parties, or where the party with the class attribute has more information, is straightforward.)

It is also necessary that the model learned not reveal information—the model parameters (probability distribution of classes) would reveal information about the (protected) class values. Instead, we build a model where each party has random shares of the model, and collaborate to classify an instance. The only knowledge gained by either side is the class of each instance classified.

The obvious alternative, generating and sharing the classifier, reveals considerable information about both the attributes and the classes. The relative distribution of classes in the training data is likely to be sensitive, as is the mean/variance or distribution of the attribute values. With our approach, neither party learns anything new until a new instance is classified, and then the only thing learned is the predicted class of that instance. While learning the predicted class of enough instances may allow reverse-engineering the classifier, this is unavoidable given the goal of learning the classes of the test data. In addition, if either party feels too much is being revealed, they can simply dispose of their share of the classifier to ensure nothing further is disclosed. Also, it is possible to extend the protocols developed such that the class of each instance is learned only by the party holding the class attribute (nothing is learned by the remaining parties). In some cases, this might be preferable.

Having both parties (the data site and the class site) hold shares of all the model parameters complicates the evaluation of a new instance. Classifying a new instance is no longer a straightforward task and a joint protocol is required to classify any new instance. The method to do this is given in Sect. 5.2. At this point, it is necessary to emphasize that the protocols proposed here for the vertically partitioned case are fully secure. Thus, only the *classification* of a new instance is computed—no other information is revealed, not even about the classifier.

5.1 Building the classifier model

The basic idea behind our protocol is that each party ends up with shares of the conditionally independent probabilities that constitute the parameters of a Naïve Bayes classifier. By themselves, the shares appear random—only when added do they have meaning. This addition only occurs as part of evaluating the classifier on an instance—and the protocol that does this reveals only the class of the instance.

We need to address two issues: how to compute shares of the model parameters, and how to classify a new instance. We start with computing the shares of the parameters. For nominal attributes, the parameters are $P(x_i|c_l) = n_i/n$ for each class i and attribute value l. For numeric attributes, we need the mean and variance for the probability density function given in Eq. (4).

5.1.1 Nominal attributes

Party P_d holds the nominal attribute D, while party P_c holds the class attribute C. D has r possible values, $a_1, \ldots, a_r$. C has k possible class values $c_1, \ldots, c_k$. The goal is to compute $r \times k$ matrices S^c, S^d where the sum of corresponding entries $s^c_{li} + s^d_{li}$ gives the probability estimate for class c_i given that the attribute has value a_l.

Thus, we compute the probabilities one at a time. Note that the probability for a particular class value c_i and attribute value a_l is nothing but the number of instances having that attribute value and class value (n_{li}) divided by the total number of instances having that class value (n_i). If each entity having that attribute value and that class value contributes $1/n_i$ to the sum, the total will be equal to the required probability. Thus, the key idea is that to compute a given entry s_{li}, P_d constructs a binary vector corresponding to the entities in the training set with 1 for each item having the value a_l and 0 for other items. P_c constructs a similar vector with $1/n_i$ for the n_i entities in the class, and 0 for other entities. Now, the scalar product of the vectors gives the appropriate probability for the entry. Essentially each instance that has the required class value and attribute value contributes $1/n_i$ (as required) to the total sum. Since the scalar product protocol gives random splits of the scalar product to the two parties, each party gets a random share of the probability as required.

Protocol 4 defines the protocol to compute the shares of these renormalized ratios (probabilities) in detail. To accomplish the security proof, calculations must occur over a closed field; as a result values are premultiplied by a constant and truncated to integral values. To achieve full precision, this constant should be a multiple of the least common multiple of $n_1, \ldots, n_k$; however sharing this would reveal private information about the distribution of classes. ($n!$ would be an acceptable multiple that would not reveal class distributions, but is computationally intractable.) In practice, using n on the order of word size (e.g., 2^{64}) will give reasonable precision (around 19 decimal digits) and computational cost. To simplify presentation, we will speak of "probability" when the algorithm in fact computes $C \cdot \mathit{probability}$.

5.1.2 Numeric attributes

For numeric attributes, computing the probability requires knowing the mean μ and variance σ^2 for each class value.

Algorithm 4 Computing shares of all probabilities

```
Require: Nominal attribute $D$, Class attribute $C$
Require: $n$ transactions, $r$ attribute values, $k$ class values
Require: Const (field size/precision)
Ensure: $r \times k$ share matrices $S^\mathrm{c}, S^\mathrm{d}$ where $S = S^\mathrm{c} + S^\mathrm{d}$ gives the
   probability values for each class/attribute
 1: for $i = 1, \ldots, k$ {For each class value} do
 2:   {$P_c$ generates the vector $\mathbf{Y}$ from $C$:}
 3:   for $j = 1, \ldots, n$ do
 4:     if $C_j = c_i$ {$j$th entry of $C$ has value $c_i$} then
 5:       $y_j \leftarrow \lfloor \mathrm{Const}/n_i \rfloor$
 6:     else
 7:       $y_j \leftarrow 0$
 8:     end if
 9:   end for
10:   for $l = 1, \ldots, r$ {For each attribute value} do
11:     {$P_d$ generates the vector $\mathbf{X}$ from $D$:}
12:     for $j = 1, \ldots, n$ do
13:       if $d_j = a_l$ then
14:         $x_j \leftarrow 1$ {Attribute value is $a_l$}
15:       else
16:         $x_j \leftarrow 0$
17:       end if
18:     end for
19:     $s^c_{li}, s^d_{li} \leftarrow \mathbf{X} \cdot \mathbf{Y}$ computed using a secure scalar product
        protocol (Appendix A.2)
20:   end for
21: end for
```

Computing the mean is similar to the preceding algorithm—for each class, P_c builds a vector of $1/n_i$ and 0 depending on whether the training entity is in the class or not, and the mean for the class is the scalar product of this vector with the projection of the data onto the attribute. The scalar product gives each party a share of the result, such that the sum is the mean (actually a constant times the mean, to convert to an integral value.) The result is a length k vector of the shares of the means. Since, homomorphic encryption is used as an underlying tool in all the protocols, and homomorphic encryption only works over integral numbers, all the values are converted to integers by premultiplying them with a reasonable constant (and truncating if necessary).

Computing the variances $\sigma_1^2, \ldots, \sigma_k^2$ is more difficult, as it requires summing the square of the distances between values and the mean, without revealing values to P_c or classes to P_d, or means to either. This is accomplished with homomorphic encryption: $E(a+b) = E(a) \cdot E(b)$, as described subsequently.

Remember that P_d owns the data vector while P_c holds the class vector. Thus subtracting the correct mean from each data value can only be done by P_c though it should not know what the data values or the means are. To achieve this, P_d generates a homomorphic encryption key-pair E_k, D_k. Next, P_d encrypts the data vector as well as the shares of the means that he has. These are sent to P_c along with the encryption key (Note: the decryption key is not sent). P_c can now encrypt his shares of the means and using the homomorphic property of the encryption system, subtract the appropriate mean from each data value in encrypted form. P_c also subtracts a random value, keeping the random value as its share of the distances. Homomorphic encryption makes this possible without decrypting. It now sends the vector back to P_d, which can decrypt to get the distance minus a random value.

The parties now engage in a square computation protocol (Appendix A.4) to compute shares t'_j, t''_j of the square of the sum of P_c's randoms r_j and the decrypted distance. The scalar product of P_d's share vector and the class vector $\mathbf{Y}$ is taken, giving two shares. To its share, P_c adds the scalar product of its vector of randoms and $\mathbf{Y}$. This gives each party a share of σ^2 multiplied by the probability of an item appearing in the class (again scaled to an integral value, in this case by the cube of the chosen constant.) Protocol 5 describes this process in detail.

The scalar product and square computation subroutines are based on previous work, and are discussed in Appendix.

Algorithm 5 Computing mean and variance

```
Require: $n$ data items, $k$ class values, precision/field size Const
Require: $P_d$ has data vector $\mathbf{D}$, $P_c$ has class vector $\mathbf{C}$
 1: {Compute the mean:}
 2: for $i = 1, \dots, k$ do
 3:   for $j = 1, \dots, n$ do
 4:     if $C_j = c_i$ {$j$th entry of $C$ has value $c_i$ } then
 5:       $y_j \leftarrow \lfloor \text{Const}/n_i \rfloor$
 6:     else
 7:       $y_j \leftarrow 0$
 8:     end if
 9:   end for
10:   $\mu'_i, \mu''_i \leftarrow \mathbf{D} \cdot \mathbf{Y}$ {Computed with secure scalar product.}
11:   {shares $\mu'_i + \mu''_i = \text{Const} \cdot \mu_i$, where $\mu_i$ is the mean for class $i$}
12: end for
13:
14: {Compute the variance}
15: $P_d$: generate a homomorphic public key encryption pair $E_k, D_k$
16: for $j = 1, \dots, n$ do
17:   $d^e_j \leftarrow E_k(\text{Const} \cdot d_j)$
18: end for
19: for $i = 1, \dots, k$ do
20:   $m^e_i \leftarrow E_k(\mu'_i)$
21: end for
22: $P_d$ sends $\mathbf{D^e}, \mathbf{M^e}$, and $E_k$ to $P_c$
23: $P_c$: generate the vectors $\mathbf{Z}$:
24: for $j = 1, \dots, n$ do
25:   Generate random $r_j$
26:   $z_j \leftarrow d^e_j / (m_{c_j} \cdot E_k(\mu''_{c_j} + r_j))$
27:   $\{= E_k(\text{Const} \cdot d_j - \mu'_{c_j} - \mu''_{c_j} - r_j)\}$
28:   $\{= E_k(\text{Const} \cdot (d_j - \mu_{c_j}) - r_j)\}$
29: end for
30: $P_c$ sends $\mathbf{Z}$ to $P_d$
31: $P_d$ decrypts all the transactions in $\mathbf{Z}$ to get $\mathbf{W}$ (i.e., $w_j \leftarrow D_k(E_k(\text{Const} \cdot (d_j - \mu_l) + r_j)) = \text{Const} \cdot (d_j - \mu_l) + r_j$)
32:
33: for $j = 1, \dots, n$ do
34:   Shares $t'_j, t''_j \leftarrow (r_j + w_j)^2$ using the protocol in Appendix A.4
35: end for
36: for $i = 1, \dots, k$ do
37:   {$\mathbf{Y}$ is vector for class $k$ as generated in steps 1–5}
38:   Compute shares temp, $\sigma''_j$ where temp $+ \sigma''_j = \mathbf{T''} \cdot \mathbf{Y}$
39:   $P_c : \sigma''_j \leftarrow \mathbf{T} \cdot \mathbf{Y} + \text{temp}$
40:   {Note $\sigma'_j + \sigma''_j = \text{Const}^3 \cdot (\frac{1}{n_j} \cdot (\sum_j (d_j - \mu_j)^2))$}
41: end for
```

5.2 Evaluating an instance

A new instance is classified according to Eq. (3). Since both $y = x^2$ and $y = \ln x$ are monotonically increasing functions, squaring and taking the natural log still preserves the correctness of the argmax. Thus, the equation can be rewritten as follows:

$$
\begin{aligned}
c_{\text{NB}} &= \operatorname*{argmax}_{c_j \in C} \left(P(c_j) \prod_i P(a_i|c_j) \right) \\
&= \operatorname*{argmax}_{c_j \in C} \left(\ln \left(P(c_j) \prod_i P(a_i|c_j) \right)^2 \right) \\
&= \operatorname*{argmax}_{c_j \in C} \left((2 \cdot \ln P(c_j)) + \sum_i \ln \left(P(a_i|c_j)^2 \right) \right) \\
&= \operatorname*{argmax}_{c_j \in C} \left(\begin{array}{l} C + (2 \cdot \ln P(c_j)) + \\ \sum_i \ln \left(P(a_i|c_j)^2 \right) \end{array} \right) \qquad (6)
\end{aligned}
$$

where the constant C is determined by the number and composition of the nominal attributes. Given l nominal attributes, $C = \prod_{i=1}^{l} \text{Const}_i$, where each Const_i is the constant the ith nominal probability is multiplied by to get adequate integral precision for that attribute. taking the logarithm converts the constant multiplicative factor into a constant additive factor.

For a nominal attribute,

$$\ln \left(P(a_i|c_j)^2 \right) = \ln \left(\frac{n_j}{n} \right)^2 = 2 \ln(p' + p'')$$

We have already shown how to compute p' and p'' in Sect. 5.1.1. The parties can compute shares of the ln function securely using the secure ln method developed by Lindell and Pinkas, outlined later in Appendix A.5. Finally, they can multiply their shares by 2 to generate the necessary shares.

For a numeric attribute,

$$\ln\left(P(a_i|c_j)^2\right) = \ln\left(\frac{1}{2\pi\sigma^2}\mathrm{e}^{-\frac{(x-\mu)^2}{\sigma^2}}\right)$$
$$= -\ln(2\pi\sigma^2) - \frac{(x-\mu)^2}{\sigma^2}$$
$$= -\ln(2\pi) - \ln(\sigma^2) - \frac{(x-\mu)^2}{\sigma^2} \tag{7}$$

$\ln(2\pi)$ is publicly computable, but it does not need to be computed since it is applied equally to both sides of the comparison. Shares of σ^2 are present with both parties. Shares of $\ln(\sigma^2)$ can again be computed using the method discussed in Appendix A.5. Shares of $(x-\mu)^2$ can be computed using the square computation method given in Appendix A.4. Finally, shares of $(x-\mu^2)/\sigma^2$ can once again be obtained using the same trick of computing the natural log and exponentiating it as described in Sect. 4.5. In brief, this is done as follows: observe that

$$\frac{(x-\mu^2)}{\sigma^2} = \mathrm{e}^{(\ln((x-\mu^2)/\sigma^2))} \tag{8}$$
$$= \mathrm{e}^{((\ln(x-\mu^2))-ln(\sigma^2))} \tag{9}$$

Since, shares of $(x-\mu)^2$ and σ^2 are present with both parties, shares of $\ln(x-\mu)^2$ and $\ln(\sigma^2)$ can again be computed using the method discussed in Appendix A.5. Each party independently adds its shares and raises e to the appropriate power (computes $\mathrm{e}^{\pm\text{share}}$). Therefore, party 1 computes $\mathrm{e}^{\text{share}_1}$. Party 2 computes $\mathrm{e}^{-\text{share}_2}$. Now, they simply run a protocol to compute shares v_1, v_2 such that $v_1 + v_2 = \mathrm{e}^{\text{share}_1} \cdot \mathrm{e}^{-\text{share}_2}$. This can be easily done through polynomial evaluation.

Thus, for every class value, for each attribute, the shares of the required values are present with the party owning the attribute and the party owning the class attribute. Now, evaluating Eq. (6) reduces to a simple circuit evaluation. The required circuit adds all of the shares for each attribute for each class value and outputs the name of the class with the maximum such value. This circuit is similar to the *Secure_Add_and_Compare* circuit used in [39] except that it is extended to multiple attributes. Thus, the only result is the classification of the new instance.

The Taylor series expansion is a bounded approximation to the real value. However, the result class of the algorithm can only be wrong if the true Naïve Bayes probability estimate of the correct class and the incorrect result are within the bound δ. Again, from [28], we know that the difference between the actual result and the secure approximation is, $|\ln(x) - \tilde{\ln}(x)| \leq \frac{1}{2^u.(u+1)}$. Increasing the number of steps in the Taylor series expansions, and thus the communication cost, allows the choice of δ to be arbitrarily small—indeed, improving the relative error from the order of 10^{-4} to the order of 10^{-8} results only in a performance penalty of approximately 10%. If the correct class and the class returned are this close, then the "incorrect" result is nearly as good an answer as the best result, and likely to be adequate in practice. Similarly, the errors in calculating probability due to precision can be ignored, since precision can be improved arbitrarily by selecting a good value for Const.

5.3 Proof of security

We now give a proof of security for the protocols developed, assuming security of the prior work described in Appendix. We start with a lemma that share splitting does in fact preserve privacy.

Lemma 1 *If a function $y = f(x_1 + x_2)$ is evaluated over a finite field $\mathcal{F}$, where the inputs x_1 and x_2 are shares known to two different parties; and the output y is also split into shares, with share y_1 is chosen randomly from an uniform distribution over the field $\mathcal{F}$ and $y_2 = y - y_1$, then both parties can independently simulate their share y_i.*

Proof The proof is quite obvious. First, we need to prove that $P(y_2 = a) = \frac{1}{|\mathcal{F}|}$.

$$P(y_2 = a) = P(y - y_1 = a)$$
$$= P(y_1 = y - a)$$
$$= \frac{1}{|\mathcal{F}|}$$

This is equivalent to choosing y_2 from an uniform distribution over the field $\mathcal{F}$. Note that though the *joint* distribution of y_1, y_2 is *not* necessarily uniform, independently both y_1 and y_2 can be simulated using a uniform distribution. □

Theorem 5 *Protocol 4 privately computes the shares of all the probabilities.*

Proof The only communication occurs at line 19 with the invocation of the scalar product protocol. The results of the scalar product protocol are random shares, which can be simulated as shown in Lemma 1. Applying the composition theorem (1 on the scalar product protocol completes the simulation of Protocol 4. □

Theorem 6 *Protocol 5 privately computes the shares of the means and variances.*

Proof Communication occurs only at lines 10, 22, 30, 34 and 38. We prove the protocol secure by providing a simulator for both parties P_c and P_d. The simulator for both P_c and P_d proceeds simply by executing the actual protocol. In order to show that the view of each party can be simulated, we only need to simulate the messages received by each party.

At line 10, the results of the scalar product protocol are random shares, which can be simulated by both P_c and P_d as shown in Lemma 1.

At line 22, P_c simulates the message received by P_c generating a key pair and using the generated encryption key for E_k. It also generates a n random numbers to comprise $\mathbf{D}^e$ and k random numbers to form $\mathbf{M}^e$. Assuming security of encryption, these are computationally indistinguishable from the true vectors and encryption key.

To simulate the message received by P_d at line 30, P_d chooses n random numbers from an uniform distribution over the field $\mathcal{F}$ and encrypts these numbers with its key E_k to form the vector $\mathbf{Z}$. Note that each z_j simulates the encryption of Const $\cdot (d_j - \mu_{c_j}) - r_j$. Since the operations are over a finite field $\mathcal{F}$ and the r_j is also uniformly chosen over the finite field $\mathcal{F}$,

$$\begin{aligned} P(D_k(z_j) = x) &= P(\text{Const} \cdot (d_j - \mu_{c_j}) - r_j = x) \\ &= P(r_j = \text{Const} \cdot (d_j - \mu_{c_j}) - x) \\ &= \tfrac{1}{|\mathcal{F}|} \end{aligned}$$

Thus simulating the value is possible by choosing a random number from an uniform distribution over $\mathcal{F}$ and encrypting this random with the encryption key E_k.

At line 34,the results of the square computation are random shares, which can be simulated by both P_c and P_d as shown in Lemma 1.

At line 38, the results of the scalar product protocol are random shares, which can be simulated by both P_c and P_d as shown in Lemma 1.

Note that the scalar product in line 39 is a completely local computation by P_c and thus does not need to be simulated by P_d. Applying the composition theorem using the scalar product protocol at lines 10 and 38, and with the square computation protocol at line 34, completes the proof that Protocol 5 can be simulated. □

Theorem 7 *The evaluation protocol in Sect. 5.2 privately computes the class.*

Proof For nominal attributes, the shares of the probabilities are present with both the parties to begin with. The secure ln computation returns random shares to both parties. By Lemma 1, these shares can be independently simulated by both the parties.

Similarly, for numeric attributes, the shares of the means and variances are present with both parties. The secure ln computation returns random shares of the variance to both parties. By Lemma 1, these shares can be independently simulated. The shares of $(x-\mu)^2$ are computed by a call to the secure square computation protocol. Since this protocol also computes random shares, by Lemma 1, these shares can be independently simulated by both parties. Finally, the shares of $(x - \mu)^2/\sigma^2$ are computed using multiple invocations of the secure ln protocol, as also the polynomial evaluation protocol, both of which compute random shares, again Lemma 1 shows that these can be simulated.

The addition and comparison circuit is a generic circuit, proven secure by [17]. The result is simply the output class, and is simulated exactly as the final result is presumed known by the simulator. Applying Theorem 1 to the secure ln computation, protocols 4, 5 and square computation protocol, the evaluation protocol is also secure. □

5.4 Effect of collusion on security

The final issue requiring discussion is that of collusion. With only two parties, collusion is irrelevant in practice—parties willing to collude could simply share data outside the protocol. The entire protocol for computing model parameters only involves communication between the two parties holding the class attribute and the data attribute, so collusion with any other party will not give any additional information. The protocol for evaluating an instance uses generic circuit evaluation techniques and can easily be made resistant to collusion.

5.5 Communication and computation cost

We now look at the cost of the devised algorithms. First, we consider the communication cost of the protocols proposed. Then, we estimate the computation cost of the protocols. As earlier, for the purpose of this analysis, the number of distinct class values is assumed to be k.

5.5.1 Communication cost

For a nominal attribute with r attribute values, the scalar product protocol is called a total of $r \cdot k$ times over n-dimensional vectors. Thus, depending on the cost of the scalar product (which is typically linear in n), the cost of protocol 4 is $O(rkn)$. For small values of r, k this is feasible, though for large values it may be *quite* inefficient. A mitigating factor is that if r, k are large relative to the size of the training set n, Naïve Bayes is probably not a good classifier to use anyway.

For numeric attributes, to compute the shares of the means requires k invocations of the scalar product protocol. To compute the variance, at line 22 P_d sends $2n$-dimensional vectors to P_c. At line 30, P_c sends one n-dimensional vector to P_d. Line 34 involves n invocations of the square computation protocol. Since the square computation protocol consists of one polynomial evaluation for a polynomial of degree 2, the communication cost of n invocations of the square computation require only linear (O(nk)) communication cost where the constant is quite small. Finally, line 38 again involves k invocations of the scalar product protocol. Thus, the total communication cost is clearly linear in n (O(nk)). The cost for numeric attributes is significantly lower than for nominal attributes.

Selecting the parameters is done off-line, while classification of a new instance can be considered "online", and is done one instance at a time. For every class, evaluation requires one call to the secure ln protocol for every nominal attribute and two calls to the secure ln protocol, one call to the square computation protocol and one call to the polynomial evaluation protocol for every numeric attribute. Finally, it also requires one call to the generic addition and comparison circuit to find the class having the maximum. Secure ln computation requires running Yao's protocol on a circuit that is linear in the size of the inputs followed by the private evaluation of a polynomial of degree k' over the field $\mathcal{F}$. The value of this k' is user decidable depending on the accuracy/cost tradeoff. The total communication cost is dominated by the circuit evaluation and is $O(k \cdot k' \log |\mathcal{F}| \cdot |S|)$ bits, where $|S|$ is the length of the key for a pseudo-random function.

The cost of the square computation protocol is insignificant (since it is a constant). The cost for a numeric attribute is dominated by the secure ln protocol.

The single generic circuit required to find the class with the maximum value requires a total of k comparison circuits built on top of q addition circuits, where q is the total number of attributes. The cost of this is linear in $q + k$. For a total of q attributes, the total cost of a single evaluation is $O(qk \cdot \log |\mathcal{F}| \cdot |S|)$ bits.

5.5.2 Computation cost

We now estimate the computation cost for the proposed protocols. For categorical attributes, Protocol 4 invokes the scalar product protocol for every combination of class value and attribute value. Thus, with k class values, r attribute values, the scalar product protocol is called $r \cdot k$ times. For a general vector of size n, the scalar product protocol requires n modular exponentiations, and n modular multiplications. However, if at least one of the vectors is boolean (all values are only one or zero), no exponentiations are required, and the cost is significantly reduced to only n multiplications. Furthermore, a multiplication is required only whenever the element is 1, not when it is 0. In Protocol 4, an element for the vector $\mathbf{Y}$ is 1 only once per class value (per different $\mathbf{X}$). Therefore the total cost of protocol 5 is $k \cdot n$ modular multiplications.

Table 2 Estimated time for computing model parameters for a nominal attribute

Security parameter	Number of tuples	Number of classes	Estimated time
512	10^6	2	7.6 s
512	10^6	3	11.4 s
512	10^8	2	12.66 min
512	10^8	3	19 min
1,024	10^6	2	24 s
1,024	10^6	3	36 s
1,024	10^8	2	40 min
1,024	10^8	3	1 h

We estimated the cost of a multiplication using the GMP library on an Intel Pentium Dual 830, 3.00 GHZ with 2 GB ram. For $t = 512$, one million multiplications take 3.8 s, for $t = 1,024$, they require 12 s.

Since the number of attribute values does not affect the computation cost, Table 2 shows the estimated cost for two different database sizes ($10^6, 10^8$), and two different class sizes $(2, 3)$. As we can see, for nominal attributes, our solution is very scalable (since the optimized scalar product protocol can be used) and large quantities of data can easily be processed.

For numeric attributes, computing the mean requires k scalar products over vectors of size n. For the variance, one needs to encrypt two vectors of length n, multiply the vectors, and decrypt it once. Then the square computation protocol is run once over the entire vector. Finally, the k variances are computed using k scalar products over vectors of length n. n square computations require $2n$ exponentiations. A scalar product also requires n exponentiations. Each encryption and decryption also requires an exponentiation. Thus the total number of exponentiations required is $kn + n + n + n + 2n + kn = (2k + 5)n$ exponentiations. As earlier, an exponentiation requires 0.0015 s for $t = 512$ bits and 0.0104 s for $t = 1,024$ bits. In Table 3 we show the total cost by looking at different values of t, n, k. As we can see, for numeric attributes, the proposed protocols are not so scalable. Inherently, the optimized version of the scalar product cannot be used, and this significantly slows the computation. If a more streamlined scalar product protocol were used, or security restrictions were relaxed, it would be possible to speed the computation time up significantly.

Now, we consider the cost of evaluating an instance. For each class, evaluation requires one call to the secure ln protocol for every nominal attribute and two calls to the secure ln protocol, one call to the square computation protocol and one call to the polynomial evaluation protocol for every numeric attribute. Finally, it also

Table 3 Estimated time for computing model parameters for a numeric attribute

Security parameter	Number of tuples	Number of classes	Estimated time(s)
512	1,000	2	13.5
512	1,000	3	16.5
512	10,000	2	135
512	10,000	3	165
1,024	10,00	2	93.6
1,024	1,000	3	114.4
1,024	10,000	2	936
1,024	10,000	3	1,144

requires one call to the generic addition and comparison circuit to find the class having the maximum.

Secure ln computation requires $2 \cdot \lceil \log(|\mathcal{F}|) \rceil$ one-out-of-two oblivious transfers, where $\mathcal{F}$ denotes the field in which all operations are carried out. Each oblivious transfer requires four exponentiations. (See the appendix for details.) Each polynomial evaluation with degree u requires u exponentiations. Therefore, for each evaluation of secure ln, we need to do $8 \cdot \lceil \log(|\mathcal{F}|) \rceil + u$ exponentiations with t bit long numbers.

A polynomial evaluation with degree w requires w exponentiations. The square computation protocol requires evaluation of a polynomial of degree 2. Therefore, the square computation protocol requires two exponentiations. The final polynomial evaluation required for numeric attributes is over a polynomial of degree 1. Therefore, this requires one exponentiation. Thus, the total cost per nominal attribute is $8 \cdot \lceil \log(|\mathcal{F}|) \rceil + u$ exponentiations with t bit long numbers, and the total cost per numeric attribute is $16 \cdot \lceil \log(|\mathcal{F}|) \rceil + 2u + 3$ exponentiations. We implemented the final add-and-compare circuit to find out the time taken. The time required by this circuit is linear in seconds to the number of total classes. Thus, for k classes, the final circuit requires $k/2$ secs.

Thus, if there are n_1 nominal attributes and n_2 numeric attributes, we can estimate the total computation cost as $(8 \cdot \lceil \log(|\mathcal{F}|) \rceil + u) \cdot n_1 + (16 \cdot \lceil \log(|\mathcal{F}|) \rceil + 2u + 3) \cdot n_2$ exponentiations + $k/2$ seconds. At the outside, a field size of 10^6 is much more than sufficient. Thus, $\lceil \log(|\mathcal{F}|) \rceil = 6$. Using the earlier estimated cost of exponentiation (for $t = 512$, 0.0015 s and for $t = 1,024$, 0.0104 s), we can now estimate the final cost. The total cost is $((48 + u)n_1 + (99 + 2u)n_2) \cdot 0.0015 + k/2$ seconds, for $t = 512$ and it is $((48 + u)n_1 + (99 + 2u)n_2) \cdot 0.0104 + k/2$ seconds, for $t = 1,024$.

Table 4 shows the overall cost for different values of u, n_1, n_2, k. As we can see, the time required to evaluate an instance is still quite reasonable (and within the web response time of 30 s). Obviously, this time grows linearly with the number of attributes—and thus very high dimensional data will need significantly more time; though, in this case Naïve Bayes is probably not the best classifier to use anyway.

6 Conclusion

When legal/commercial reasons make it impossible to share data, it may be imprudent to share even models generated from the data. We have presented a method that bypasses this restriction for mining Naïve Bayes models.

This paper has concentrated on the semi-honest model, where each party assumed to follow the protocols but may try to infer information from the messages it sees. While many of the components can be extended to the malicious model,doing so efficiently is an open research problem.

Privacy is not free. In general, privacy-preserving protocols are more expensive than non-privacy-preserving protocols for the same problem. Progress in this area will enable application of data mining to opportunities that are currently unexplored due to privacy and security concerns.

Appendix: Secure computation subroutines

The privacy-preserving algorithms developed later are based on several secure computation protocols. While most have either been previously published, or are straightforward given previously published work, we summarize them here for completeness. Most are two-party protocols; this does lead to concern with collusion between the two parties. For some uses of these protocols (e.g., in the vertical partitioning protocol), collusion would reveal information of the colluding parties; in the horizontally partitioned case the problems with collusion are specifically noted in the theorem.

Note that the primary practical application of the semi-honest model is for parties that wish to avoid seeing data they do not need; parties are presumed to have incentives not to collude.

A.1 Secure sum

One building block frequently required is a way to securely calculate the sum of values from individual sites. Assuming three or more parties and no collusion, the following method [23] securely computes such a sum.

Table 4 Estimated cost for classifying an instance

Security parameter	Degree of the polynomial	Number of nominal attributes	Number of numeric attributes	Number of classes	Estimated time(s)
512	10	5	5	2	2.328
512	10	5	5	3	2.828
512	10	10	10	2	3.655
512	10	10	10	3	4.155
512	20	5	5	2	2.553
512	20	5	5	3	3.053
512	20	10	10	2	3.955
512	20	10	10	3	4.455
1,024	10	5	5	2	10.204
1,024	10	5	5	3	10.704
1,024	10	10	10	2	19.408
1,024	10	10	10	3	19.908
1,024	20	5	5	2	11.764
1,024	20	5	5	3	12.264
1,024	20	10	10	2	22.528
1,024	20	10	10	3	23.028

Assume that the value $v = \sum_{i=1}^{k} v_i$ to be computed is known to lie in the range $[0, \ldots, n-1]$ where v_i denotes the share of the ith site.

One site is designated the *master* site, numbered 1. The remaining sites are numbered $2, \ldots, k$. Site 1 generates a random number R, uniformly chosen from $[0, \ldots, n-1]$. Site 1 adds this to its local value v_1, and sends the sum $R + v_1 \bmod n$ to site 2. Since the value R is chosen uniformly from $[0, \ldots, n-1]$, the number $R + v_1 \bmod n$ is also distributed uniformly across this region, so site 2 learns nothing about the actual value of v_1. For the remaining sites $i = 2, \ldots, k-1$, the algorithm is as follows: Site i receives

$$V = R + \sum_{j=1}^{i-1} v_j \bmod n.$$

Since this value is uniformly distributed across $[0, \ldots, n-1]$, i learns nothing. Site i then computes

$$R + \sum_{j=1}^{i} v_i \bmod n = (v_i + V) \bmod n$$

and passes it to site $i+1$.

Site k performs the above step, and sends the result to site 1. Site 1, knowing R, can subtract R to get the actual result. Note that site 1 can also determine $\sum_{i=2}^{k} v_i$ by subtracting v_1. This is possible from the global result *regardless of how it is computed*, so site 1 has not learned anything from the computation.

This method faces an obvious problem if sites collude. Sites $i-1$ and $i+1$ can compare the values they send/receive to determine the exact value for v_i. The method can be extended to work for an honest majority. Each site divides v_i into shares. The sum for each share is computed individually. However, the path used is permuted for each share, such that no site has the same neighbor twice. To compute v_i, the neighbors of i from each iteration would have to collude. Varying the number of shares varies the number of dishonest (colluding) parties required to violate security. Detailed analysis of this method can be found in [6].

A.2 Scalar product protocol

Another key sub-protocol required is a protocol for computing the scalar product of two vectors. Many scalar product protocols have been proposed in the past [8, 15,20,38]. The method of Goethals et al. [15] is quite simple and provably secure. We now briefly describe it. The problem is defined as follows: Alice has a n-dimensional vector $\mathbf{X}$ while Bob has a n-dimensional vector $\mathbf{Y}$. At the end of the protocol, Alice should get $r_a = \mathbf{X} \cdot \mathbf{Y} + r_b$ where r_b is a random number chosen from an uniform distribution and is known only to Bob. The key idea behind the protocol is to use a homomorphic encryption system such as the Goldwasser–Micali cryptosystem [4], the Benaloh cryptosystem [3], the Naccache–Stern cryptosystem [30], the Paillier cryptosystem [34], and the Okamoto–Uchiyama cryptosystem [33]. Homomorphic Encryption is a semantically secure public-key encryption which, in addition to standard guarantees, has the additional property that given any two encryptions $E(A)$ and $E(B)$, there exists an encryption $E(A*B)$ such that $E(A) * E(B) = E(A * B)$, where $*$ is either addition or multiplication (in some abelian group). The cryptosystems mentioned above are additively homomorphic (thus the operation $*$ denotes addition). Using such a system, it is quite simple to create a scalar product protocol. The key is to note that $\sum_{i=1}^{n} x_i \cdot y_i = \sum_{i=1}^{n}(x_i + x_i + \cdots + x_i)$ (y_i times). If Alice encrypts her vector and sends in encrypted form to Bob, Bob can using the additive homomorphic property to compute the dot product. The specific details are given below:

Here is the actual protocol:

```
Require: Alice has input vector $\mathbf{X} = \{x_1, \ldots, x_n\}$
Require: Bob has input vector $\mathbf{Y} = \{y_1, \ldots, y_n\}$
Require: Alice and Bob get outputs $r_A, r_B$ respectively
    such that $r_A + r_B = \mathbf{X} \cdot \mathbf{Y}$
1: Alice generates a private and public key pair (sk, pk).
2: Alice sends pk to Bob.
3: for $i = 1, \ldots, n$ do
4:     Alice sends to Bob $c_i = E_{pk}(x_i)$.
5: end for
6: Bob computes $w = \prod_{i=1}^{n} c_i^{y_i}$
7: Bob generates a random plaintext $r_B$.
8: Alice sends to Bob $w' = w \cdot E_{pk}(-r_B)$.
9: Alice computes $r_A = D_{sk}(w') = \mathbf{X} \cdot \mathbf{Y} - r_B$.
```

A.3 One-out-of-two oblivious transfer

The one-out-of-two oblivious transfer protocol involves two parties, Alice and Bob. Alice has an input bit σ, while Bob has 2 inputs B_0 and B_1. At the end of the protocol, Alice learns only B_σ and nothing else while Bob learns nothing at all. The one-out-of-two oblivious transfer (OT_1^2) was suggested by Even, Goldreich and Lempel [12] as a generalization of Rabin's "oblivious transfer" [36]. For completeness, we now describe a method for doing oblivious transfer [32].

First, we describe two cryptographic definitions that are used in the protocol.

Computational Diffie–Hellman Assumption. Assume that p is a very large prime number and g is the generator of its multiplicative group (i.e. every number between $1, \ldots, p-1$ can be written as $g^k \bmod p$ for some k between $1 \ldots p-1$). The computational Diffie–Hellman Assumption states that given $g^a \bmod p$ and $g^b \bmod p$ (Note that a and b is not given), there is no efficient way to compute $g^{ab} \bmod p$. This assumption is the basis for the Diffie–Hellman key exchange protocol; if it does not hold many cryptographic techniques would be breakable.

Random Oracle Assumption. In the construction of the protocol, we will use a cryptographic Hash function H. We assume that this function is known to all parties (e.g., SHA) and it maps its input to what appears to be a random output. Again, this is a common cryptographic tool used in many protocols.

Now using the Diffie–Hellman Assumption and a hash function H, we can implement a one-out-of-two oblivious transfer that discloses no information even if one of the parties tries to deviate from the protocol. In the following protocol, as defined above, let σ be the input of Alice and let B_0 and B_1 be the inputs of Bob. Also note that every operation except evaluating the function H and $\oplus$ (exclusive or) is done mod p.

1. Bob publishes a random number C between $1, \ldots, p-1$ along with g and p.
2. Alice picks a random number k between $1, \ldots, p-1$, sets $P_\sigma = g^k$ and $P_{1-\sigma} = C/P_\sigma$, and sends P_0 to Bob.
3. Bob finds P_1 by evaluating C/P_0, creates $E_0 = (g^{r_0}, H((P_0)^{r_0}) \oplus B_0)$, $E_1 = (g^{r_1}, H((P_1)^{r_1}) \oplus B_1)$, by randomly choosing r_0, r_1 between $1, \ldots, p-1$, and sends E_0, E_1 to Alice.
4. Alice computes $H((P_\sigma)^{r_\sigma}) = H((g^{r_\sigma})^k))$ to find B_σ.

In the above protocol the choice of Alice (σ) is not revealed because all Bob receives is either g^k or C/g^k, where k is chosen randomly. Since operations are done in mod p, both g^k or C/g^k values are uniformly distributed between $0, \ldots, p-1$. Therefore, Bob does not see anything more than a random number. Alice learns nothing by receiving the random C, or (because of the random oracle hash function) from inspecting E_0 or E_1. While Alice can decrypt E_σ to obtain the final result, by the original Diffie–Hellman assumption it cannot determine $(g^{r_{\sigma-1}})^k$ to decrypt the other box. If Alice could decrypt both E_0 and E_1, this would contradict with the Diffie–Hellman assumption.

A.4 Square computation

The problem is defined as follows: Alice holds x_a, while Bob holds x_b. Together they wish to compute shares of the function $f = (x_a + x_b)^2$. Thus, at the end of the protocol, Alice should have y_a and Bob should have y_b such that $y_a + y_b = (x_a + x_b)^2$. An obvious way to do this is using oblivious evaluation of polynomials. Alice first generates a random value y_a. Alice then forms the polynomial $P(z) = (1)z^2 + (2x_a)z + (x_a^2 - y_a)$. An oblivious evaluation of $P(x_b)$ by Bob gives Bob, $y_b = P(x_b)$. Note that $y_b + y_a = x_b^2 + 2x_ax_b + x_a^2 - y_a + y_a = (x_a + x_b)^2$ as required.

A.4.1 Oblivious evaluation of polynomials

Alice has a polynomial P of degree k over some finite field $\mathcal{F}$. Bob has an element $x \in \mathcal{F}$ and also knows k. Alice would like to let Bob compute the value $P(x)$ in such a way that Alice does not learn x and Bob does not gain any additional information about P (except $P(x)$). This problem was first investigated by [31]. Subsequently, there have been more protocols improving the communication and computation efficiency [7] as well as extending the problem to floating point numbers [5]. For our protocols, we use the protocol given in [7] since it requires only $O(k)$ exponentiations in order to evaluate a polynomial of degree k (where the constant is very small). This works well since we only require evaluation of low-degree polynomials.

We now briefly describe the protocol used for oblivious polynomial evaluation that uses the secure dot product described before as a subroutine. Given a dot product protocol, we can easily create a protocol for polynomial evaluation as follows: Let $P(y) = \sum_{i=0}^{k} a_i y^i$ be Alice's input and x be Bob's input, using secure dot product, Bob can evaluate the $P(x)$ as follows:

Alice forms Bob forms

$$\mathbf{U} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} \quad \mathbf{V} = \begin{bmatrix} 1 \\ x \\ \vdots \\ x^k \end{bmatrix}$$

Alice and Bob engage in secure dot product so that (only) Bob gets $r = \mathbf{U}.\mathbf{V}$ Clearly $r = \sum_{i=0}^{k} a_i x^i = P(x)$. Assuming that dot product protocol is secure, above protocol is also secure.

A.5 Privately computing ln x

In classifying an instance, we need to be able to privately compute $\ln x$, where $x = x_1 + x_2$ with x_1 known to Alice and x_2 known to Bob. Thus, Alice should get y_1 and Bob should get y_2 such that $y_1 + y_2 = \ln x = \ln(x_1 + x_2)$. One of the key results presented in [28] was a cryptographic protocol for this computation. We now describe the protocol in brief: Note that $\ln x$ is *Real* while general cryptographic tools work over finite fields. We multiply the $\ln x$ with a known constant to make it integral.

The basic idea behind computing random shares of $\ln(x_1 + x_2)$ is to use the Taylor approximation for $\ln x$. Remember that the Taylor approximation gives us

$$\ln(1+\epsilon) = \sum_{i=1}^{\infty} \frac{(-1)^{i-1}\epsilon^i}{i}$$
$$= \epsilon - \frac{\epsilon^2}{2} + \frac{\epsilon^3}{3} - \frac{\epsilon^4}{4} + \cdots \quad \text{for } -1 < \epsilon < 1$$

For an input x, let $n = \lfloor \log_2 x \rfloor$. Then 2^n represents the closest power of 2 to x. Therefore, $x = x_1 + x_2 = 2^n(1+\epsilon)$ where $-1/2 \le \epsilon \le 1/2$. Consequently,

$$\begin{aligned} \ln(x) &= \ln(2^n(1+\epsilon)) \\ &= \ln 2^n + \ln(1+\epsilon) \\ &\approx \ln 2^n + \sum_{i=1\ldots k} (-1)^{i-1}\epsilon^i / i \\ &= \ln 2^n + T(\epsilon) \end{aligned}$$

where $T(\epsilon)$ is a polynomial of degree k. This error is exponentially small in k.

There are two phases to the protocol. Phase 1 finds an appropriate n and ϵ. Let N be a predetermined (public) upper-bound on the value of n. First, Yao's circuit evaluation is applied to the following small circuit which takes x_1 and x_2 as input and outputs random shares of $\epsilon 2^N$ and $2^N n \ln 2$. Note that $\epsilon 2^n = x - 2^n$, where n can be determined by simply looking at the two most significant bits of x and $\epsilon 2^N$ is obtained simply by shifting the result by $N - n$ bits to the left. Thus, the circuit outputs random α_1 and α_2 such that $\alpha_1 + \alpha_2 = \epsilon 2^N$, and also outputs random β_1 and β_2 such that $\beta_1 + \beta_2 = 2^N n \ln 2$. This circuit can be easily constructed. Random shares are obtained by having one of the parties input random values $\alpha_1, \beta_1 \in_R \mathcal{F}$ into the circuit and having the circuit output $\alpha_2 = \epsilon 2^N - \alpha_1$ and $\beta_2 = 2^N n \ln 2 - \beta_1$ to the other party.

Phase 2 of the protocol involves computing shares of the Taylor series approximation, $T(\epsilon)$. This is done as follows: Alice chooses a random $w_1 \in \mathcal{F}$ and defines a polynomial $Q(x)$ such that $w_1 + Q(\alpha_2) = T(\epsilon)$. Thus $Q(\cdot)$ is defined as

$$Q(x) = \text{lcm}(2,\ldots,k) \sum_{i=1}^{k} \frac{(-1)^{i-1}}{2^{N(i-1)}} \frac{(\alpha_1 + x)^i}{i} - w_1$$

Alice and Bob then execute an oblivious polynomial evaluation with Alice inputting $Q(\cdot)$ and Bob inputting α_2, in which Bob obtains $w_2 = Q(\alpha_2)$. Alice and Bob define $u_1 = \text{lcm}(2,\ldots,k)\beta_1 + w_1$ and $u_2 = \text{lcm}(2,\ldots,k)$ $\beta_2 + w_2$. We have that $u_1 + u_2 \approx 2^N \text{lcm}(2,\ldots,k) \ln x$. Further details on the protocol, as well as the proof of security, can be found in [28].

References

1. Agrawal, D., Aggarwal, C.C.: On the design and quantification of privacy preserving data mining algorithms. In: Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 247–255. ACM, Santa Barbara, California, USA (2001). http://doi.acm.org/10.1145/375551.375602
2. Agrawal, R., Srikant, R.: Privacy-preserving data mining. In: Proceedings of the 2000 ACM SIGMOD Conference on Management of Data, pp. 439–450. ACM, Dallas, TX (2000). http://doi.acm.org/10.1145/342009.335438

3. Benaloh, J.C.: Secret sharing homomorphisms: Keeping shares of a secret secret. In: A. Odlyzko (ed.) Advances in Cryptography–CRYPTO86: Proceedings, vol. 263, pp. 251–260. Lecture Notes in Computer Science, Springer Heidelberg (1986). http://springerlink.metapress.com/openurl.asp?genre=article&issn= 0302-9743&volume=263&spage=251
4. Blum, M., Goldwasser, S.: An efficient probabilistic public-key encryption that hides all partial information. In: R. Blakely (ed.) Advances in Cryptology—Crypto 84 Proceedings. Springer, Heidelberg (1984)
5. Chang, Y.C., Lu, C.J.: Oblivious polynomial evaluation and oblivious neural learning. Lecture Notes in Computer Science, vol. 2248, pp. 369+ (2001). citeseer.nj.nec.com/531490.html
6. Chor, B., Kushilevitz, E: A communication-privacy tradeoff for modular addition. Inf. Process. Lett. **45**, 205–210 (1993)
7. Cramer, R., Gilboa, N., Naor, M., Pinkas, B., Poupard, G.: Oblivious Polynomial Evaluation. In: The Privacy Preserving Data Mining (paper by Naor and Pinkas) (2000)
8. Du, W., Atallah, M.J.: Privacy-preserving statistical analysis. In: Proceeding of the 17th Annual Computer Security Applications Conference. New Orleans, Louisiana, USA (2001). http://www.cerias.purdue.edu/homes/duw/research/paper/acsac20 01.ps
9. Du, W., Atallah, M.J.: Secure multi-party computation problems and their applications: A review and open problems. In: New Security Paradigms Workshop, pp. 11–20. Cloudcroft, New Mexico, USA (2001). http://www.cerias.purdue.edu/homes/duw/research/paper/nspw2001.ps
10. Du, W., Zhan, Z.: Building decision tree classifier on private data. In: C. Clifton, V. Estivill-Castro (eds.) IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining, vol. 14, pp. 1–8. Australian Computer Society, Maebashi City, Japan (2002). http://crpit.com/Vol14.html
11. Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. Official J. Eur. Communities **I**(281), 31–50 (1995) http://europa.eu.int/comm/internal_market/privacy
12. Even, S., Goldreich, O., Lempel, A: A randomized protocol for signing contracts. Commun. ACM **28**(6), 637–647 (1985)
13. Evfimievski, A., Srikant, R., Agrawal, R., Gehrke, J.: Privacy preserving mining of association rules. In: The 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 217–228. Edmonton, Alberta, Canada (2002). http://doi.acm.org/10.1145/775047.775080
14. Feingold, M., Corzine, M., Wyden, M., Nelson, M.: Data Mining Moratorium Act of 2003. U.S. Senate Bill (proposed) (2003). http://thomas.loc.gov/cgi-bin/query/z?c108:S.188:
15. Goethals, B., Laur, S., Lipmaa, H., Mielikäinen, T.: On secure scalar product computation for privacy-preserving data mining. In: C. Park S. Chee (eds.) The 7th Annual International Conference in Information Security and Cryptology (ICISC 2004), vol. 3506, pp. 104–120 (2004)
16. Goldreich, O.: The Foundations of Cryptography, vol. 2, chap. General Cryptographic Protocols. Cambridge University Press, Cambridge (2004). http://www.wisdom.weizmann.ac.il/∼oded/PSBookFrag/prot.ps
17. 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, pp. 218–229 (1987). http://doi.acm.org/10.1145/28395.28420
18. Standard for privacy of individually identifiable health information. Fed. Regist. **67**(157), 53,181–53,273 (2002). http://www.hhs.gov/ocr/hipaa/finalreg.html
19. Huang, Z., Du, W., Chen, B.: Deriving private information from randomized data. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, Baltimore, MD (2005)
20. Ioannidis, I., Grama, A., Atallah, M.: A secure protocol for computing dot-products in clustered and distributed environments. In: The 2002 International Conference on Parallel Processing, Vancouver, British Columbia (2002)
21. Jagannathan, G., Wright, R.N.: Privacy-preserving distributed *k*-means clustering over arbitrarily partitioned data. In: Proceedings of the 2005 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, IL, pp. 593–599 (2005)
22. Kantarcioglu, M., Vaidya, J.: An architecture for privacy-preserving mining of client information. In: C. Clifton V. Estivill-Castro (eds.) IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining, vol. 14, pp. 37–42. Australian Computer Society, Maebashi City, Japan (2002). http://crpit.com/Vol14.html
23. Kantarcıoǧlu, M., Clifton, C.: Privacy-preserving distributed mining of association rules on horizontally partitioned data. In: The ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'02), Madison, Wisconsin, pp. 24–31 (2002) http://www.bell-labs.com/user/minos/DMKD02/Papers/kantarcioglu.pdf
24. Kantarcıoǧlu, M., Clifton, C.: Privacy-preserving distributed mining of association rules on horizontally partitioned data. IEEE Trans. Knowl. Data Eng. **16**(9), 1026–1037 (2004) http://doi.ieeecomputersociety.org/10.1109/TKDE.2004.45
25. Kargupta, H., Datta, S., Wang, Q., Sivakumar, K.: On the privacy preserving properties of random data perturbation techniques. In: Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM'03). Melbourne, Florida (2003)
26. Lin, X., Clifton, C., Zhu, M.: Privacy preserving clustering with distributed EM mixture modeling. Knowl. Inf. Syst. **8**(1), 68–81 (2005) http://dx.doi.org/10.1007/s10115-004-0148-7
27. Lindell, Y., Pinkas, B.: Privacy preserving data mining. In: Advances in Cryptology — CRYPTO 2000, pp. 36–54. Springer, Heidelberg (2000)
28. Lindell, Y., Pinkas, B.: Privacy preserving data mining. J. Cryptol. **15**(3), 177–206 (2002) http://www.research.ibm.com/people/l/lindell//id3_abs.html
29. Mitchell, T.: Machine Learning, 1st edn. McGraw-Hill Science/Engineering/Math, New York (1997)
30. Naccache, D., Stern, J.: A new public key cryptosystem based on higher residues. In: Proceedings of the 5th ACM conference on Computer and communications security, pp. 59–66. ACM Press, San Francisco, California, United States (1998). doi: http://doi.acm.org/10.1145/288090.288106
31. Naor, M., Pinkas, B.: Oblivious transfer and polynomial evaluation. In: Proceedings of the 31st annual ACM symposium on Theory of computing, pp. 245–254. ACM Press, Atlanta, Georgia, United States (1999). doi: http://doi.acm.org/10.1145/301250.301312
32. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: Proceedings of SODA 2001 (SIAM Symposium on Discrete Algorithms), Washington, D.C. (2001)
33. Okamoto, T., Uchiyama, S.: A new public-key cryptosystem as secure as factoring. In: Advances in Cryptology—Eurocrypt '98, LNCS 1403, pp. 308–318. Springer, Heidelberg (1998)

34. Paillier, P.: Public key cryptosystems based on composite degree residuosity classes. In: Advances in Cryptology—Eurocrypt '99 Proceedings, LNCS 1592, pp. 223–238. Springer, Heidelberg (1999)
35. Perry, J.M.: Statement of John M. Perry, President and CEO, Cardsystems Solutions, Inc. before the United States House of Representatives Subcommittee on Oversight and Investigations of the Committee on Financial services. http://financialservices.house.gov/hearings.asp?formmode=detail&hearing =407&comm=4(2005). http://financialservices.house.gov/hearings.asp?formmode=detail&hearing=407&comm=4
36. Rabin, M.: How to exchange secrets by oblivious transfer. Tech. Rep. TR-81, Aiken Computation Laboratory, Harvard University (1981)
37. Rizvi, S.J., Haritsa, J.R.: Maintaining data privacy in association rule mining. In: Proceedings of 28th International Conference on Very Large Data Bases, pp. 682–693. VLDB, Hong Kong (2002) http://www.vldb.org/conf/2002/S19P03.pdf
38. Vaidya, J., Clifton, C.: Privacy preserving association rule mining in vertically partitioned data. In: The 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, Alberta, Canada, pp. 639–644. (2002). http://doi.acm.org/10.1145/775047.775142
39. Vaidya J., Clifton C (2003) Privacy-preserving *k*-means clustering over vertically partitioned data. In: The 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, pp. 206–215 (2003). http://doi.acm.org/10.1145/956750.956776
40. Vaidya, J., Clifton, C.: Privacy preserving naïve bayes classifier for vertically partitioned data. In: 2004 SIAM International Conference on Data Mining, Lake Buena Vista, Florida, pp. 522–526 (2004) http://www.siam.org/meetings/sdm04/proceedings/sdm04_059.pdf
41. Vaidya, J., Clifton, C.: Privacy-preserving decision trees over vertically partitioned data. In: The 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security. Springer, Storrs, Connecticut (2005) http://dx.doi.org/10.1007/11535706_11
42. Vaidya J., Clifton C.: Secure set intersection cardinality with application to association rule mining. J. Comput. Secur. 13(4) (2005).
43. Wright, R., Yang, Z.: Privacy-preserving bayesian network structure computation on distributed heterogeneous data. In: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, WA (2004)
44. Yao, A.C.:How to generate and exchange secrets. In: Proceedings of the 27th IEEE Symposium on Foundations of Computer Science, pp. 162–167. IEEE, NewYork (1986)