

Privacy Preserving Naïve Bayes Classifier for Vertically Partitioned Data

Jaideep Vaidya*

Chris Clifton*

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. Naïve Bayes is often used as a baseline classifier, consistently providing reasonable classification performance. This paper brings privacy-preservation to Naïve Bayes classification on vertically partitioned data.

Keywords: privacy, security, distributed classification

1 Introduction.

There is growing concern with the privacy implications of data mining. While data mining usually produces general models rather than knowledge 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. The definition of privacy 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 isn't inherently obvious from the data mining result. 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 trees[8]; protocols have also been developed for association rules[11, 6] and clustering[12, 7].

Naïve Bayes is a simple but highly effective classifier. This combination of simplicity and effectiveness has led 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[9]). This

paper extends the portfolio of privacy-preserving distributed data mining to include this standard classifier.

The method presented in this paper applies to vertically partitioned data, i.e., each site has a different set of attributes but all reference the same set of entities. For example, assume a medical research study may want to compare medical outcomes based on techniques in pharmaceutical manufacturing processes (e.g., to answer the question “are generic drugs really as effective as brand-name”, and more important, what manufacturing processes produce the best results?) The insurance companies can't disclose individual patient data without permission [4], and complete manufacturing processes are trade secrets (although individual techniques may be commonly known.)

Section 2 gives the model in which the problem is set. We present a protocol for privately learning a Naïve Bayes classifier in Section 3. The proof of the privacy preserving properties of the method is given in Section 4. We conclude with a discussion of future work.

2 Problem Statement.

This paper addresses classification over vertically partitioned data: where different sites hold different attributes. 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 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.

*Purdue University

It is important that the model learned not reveal information – the model parameters (probability distribution of classes) would reveal information about the (protected) class values. 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. 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.

Learning the predicted class of enough instances may allow reverse-engineering the classifier, but this is unavoidable given the goal of learning the classes of the test data. A party can prevent further disclosure at any time by disposing of their share of the classifier.

To prove that our Naïve Bayes algorithm preserves privacy, we need to define privacy preservation. We use the framework defined in *Secure Multiparty Computation*[3], and several primitives from the Secure Multiparty Computation literature. 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 Privacy Preserving Naive Bayes Classifier.

A complete exposition of the Naïve Bayes algorithm can be found in [9]. We assume that the basic formulae are well known. 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 must address two issues: How to compute shares of the model parameters, and how to classify a new instance. To compute shares of nominal attributes, the parameters are $P(x_i|v_l) = n_i/n$ for each class i and attribute value l . For numeric attributes, we need the mean and variance for the probability density function for the Normal distribution.

3.1 Nominal Attributes Let party P_d holds the nominal attribute D , and party P_c the class attribute C . D has r possible values, a_1, \dots, a_r . C has k possible class values v_1, \dots, v_k . The goal is to compute $r \times k$ matrices S^c, S^d where the sum of corresponding entries $s_{li}^c + s_{li}^d$ gives the probability estimate for class v_i given that the attribute has value a_l .

The key idea is that to compute a given entry s_{li}, P_d

Protocol 1 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^c, S^d where $S = S^c + S^d$ gives the probability values for each class/attribute

```

1: for  $i = 1 \dots k$  {For each class value} do
2:    $\{P_c$  generates the vector  $\vec{Y}$  from  $C\}$ 
3:   for  $j = 1 \dots n$  do
4:     if  $c_j = v_i$  then
5:        $y_j \leftarrow \lfloor Const/n_i \rfloor$  {Class value is  $v_i$ }
6:     else
7:        $y_j \leftarrow 0$ 
8:     end if
9:   end for
10:  for  $l = 1 \dots r$  {For each attribute value} do
11:     $\{P_d$  generates the vector  $\vec{X}$  from  $D\}$ 
12:    for  $j = 1 \dots n$  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:     $s_{li}^c, s_{li}^d \leftarrow \vec{X} \cdot \vec{Y}$  computed using a secure scalar
    product protocol [1, 11, 5]
20:  end for
21: end for

```

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. The scalar product of the vectors gives the appropriate probability for the entry.

Protocol 1 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, \dots, 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 and computational cost. To simplify presentation, we will speak of “probability” when the algorithm in fact computes $C * probability$.

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

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. 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 (multiplied by a constant, to convert to an integral value.)

Computing the variances $\sigma_1^2, \dots, \sigma_k^2$ 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. Protocol 2 gives the algorithm to compute both. In lines 15-22, P_d computes encrypted vectors of the data values and its share of the means and sends them to P_c , along with the encryption (but not decryption) key.

In the next phase (lines 23-31), P_c takes the data values and subtracts the means (both its share and the share sent by P_d) to get the distance needed to compute the variance. P_c also subtracts a random value, keeping the random value as its share of the distances. Homomorphic encryption ($E(a + b) = E(a) * E(b)$) 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 to compute shares t'_j, t''_j of the square of the sum of P_c 's randoms r_j and the decrypted distance. Square computation can be done using oblivious polynomial evaluation[10]. The scalar product of P_d 's share vector and the class vector \vec{Y} is taken, giving two shares. To its share, P_c adds the scalar product of its vector of randoms and \vec{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.)

3.3 Evaluation The class of a new instance is predicted as follows:

$$(3.1) \quad v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} \left(P(v_j) \prod_i P(a_i | v_j) \right)$$

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:

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} \left(P(v_j) \prod_i P(a_i | v_j) \right)$$

Protocol 2 Computing Mean and Variance

Require: n data items, k class values, precision/field size $Const$

Require: P_d has data vector \vec{D} , P_c has class vector \vec{C}

```

1: {Compute the mean:}
2: for  $i = 1 \dots k$  do
3:   for  $j = 1 \dots n$  do
4:     if  $c_j = v_i$  then
5:        $y_j \leftarrow \lfloor Const/n_i \rfloor$  {Class value is  $v_i$ }
6:     else
7:        $y_j \leftarrow 0$ 
8:     end if
9:   end for
10:   $\mu'_i, \mu''_i \leftarrow \vec{D} \cdot \vec{Y}$  {Computed with secure scalar
    product.}
11:  {shares  $\mu'_i + \mu''_i = Const * \mu_i$ , 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:    $d_j^e \leftarrow E_k(Const * d_j)$ 
18: end for
19: for  $i = 1 \dots k$  do
20:    $m_i^e \leftarrow E_k(\mu'_i)$ 
21: end for
22:  $P_d$  sends  $\vec{D}^e, \vec{M}^e$ , and  $E_k$  to  $P_c$ 
23:  $P_c$ : generate the vectors  $\vec{Z}$  and  $\vec{X}$ :
24: for  $j = 1 \dots n$  do
25:   Generate random  $r_j$ 
26:    $z_j \leftarrow y'_j / (m_{c_j} * E_k(\mu''_{c_j} + r_j))$ 
27:    $\{ = E_k(Const * d_j - \mu'_{c_j} - \mu''_{c_j} - r_j) \}$ 
28:    $\{ = E_k(Const * (d_j - \mu_{c_j}) - r_j) \}$ 
29: end for
30:  $P_c$  sends  $\vec{Z}$  to  $P_d$ 
31:  $P_d$  decrypts all the transactions in  $\vec{Z}$  to get  $\vec{W}$ 
    (i.e.,  $w_j \leftarrow D_k(E_k(Const * (d_j - \mu_l) + r_j)) =
    Const * (d_j - \mu_l) + r_j$ )
32:
33: for  $j = 1 \dots n$  do
34:   Shares  $t'_j, t''_j \leftarrow (r_j + w_j)^2$  using a protocol for
    square computation
35: end for
36: for  $i = 1 \dots k$  do
37:    $\{\vec{Y}$  is vector for class  $k$  as generated in steps 1-5}
38:   Compute shares  $temp, \sigma''_j$  where  $temp + \sigma''_j =
    \vec{T}'' \cdot \vec{Y}$ 
39:    $P_c : \sigma'_j \leftarrow \vec{T} \cdot \vec{Y} + temp$ 
40:   {Note  $\sigma'_j + \sigma''_j = Const^3 * (\frac{1}{n_j} * (\sum_j (d_j - \mu_j)^2))$ }
41: end for

```

$$\begin{aligned}
&= \operatorname{argmax}_{v_j \in V} \left(\ln \left(P(v_j) \prod_i P(a_i|v_j) \right)^2 \right) \\
&= \operatorname{argmax}_{v_j \in V} \left((2 * \ln P(v_j)) + \sum_i \ln (P(a_i|v_j)^2) \right) \\
(3.2) \quad &= \operatorname{argmax}_{v_j \in V} \left(C + (2 * \ln P(v_j)) + \sum_i \ln (P(a_i|v_j)^2) \right)
\end{aligned}$$

where the constant C is determined by the number and composition of the nominal attributes. If there are l nominal attributes, $C = \text{Const}_1 * \dots * \text{Const}_l$ where each Const_i is the constant that the nominal probability of attribute i was multiplied by. Taking the logarithm converts the constant multiplicative factor into a constant additive factor.

For a nominal attribute,

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

We have already shown how to compute p' and p'' in Section 3.1. The parties can compute shares of the \ln function securely using the secure \ln method developed by Lindell and Pinkas [8]. Finally, they multiply their shares by 2 to generate the necessary shares.

For a numeric attribute,

$$\begin{aligned}
\ln (P(a_i|v_j)^2) &= \ln \left(\frac{1}{2\pi\sigma^2} e^{-\frac{(x-\mu)^2}{\sigma^2}} \right) \\
&= -\ln(2\pi\sigma^2) - \frac{(x-\mu)^2}{\sigma^2} \\
(3.3) \quad &= -\ln(2\pi) - \ln(\sigma^2) - \frac{(x-\mu)^2}{\sigma^2}
\end{aligned}$$

Shares of σ^2 are present with both parties. Shares of $\ln(\sigma^2)$ can again be computed using Lindell's method [8]. Shares of $(x-\mu)^2$ can be computed using the square computation method. Finally, shares of $(x-\mu^2)/\sigma^2$ can be computed using an extension of the division protocol of Du and Atallah[1]. Thus, for every class value and attribute, the shares of the required values are divided between the parties owning the attribute and the class attribute. Evaluating equation 3.2 is accomplished through secure circuit evaluation as in the *Secure_Add_and_Compare* circuit used in [12], returning only the class label of the maximal class.

4 Proof of Security

We now give a proof of security for protocols of Section 3. The composition theorem [3] allows us to use existing secure protocols as components, worrying only that their results do not reveal anything. We start with a lemma that share splitting does in fact preserve privacy.

LEMMA 4.1. *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, where the 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. We first show that $P(y_2 = a) = \frac{1}{|\mathcal{F}|}$.

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

This is equivalent to choosing y_2 from an uniform distribution over the field \mathcal{F} . Although 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 4.1. *Protocol 1 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 4.1. Protocol 1 can thus be simulated, with the composition theorem being applied to the scalar product protocol.

THEOREM 4.2. *Protocol 2 privately computes the shares of the means and variances.*

Proof. We prove the protocol secure by providing a simulator for the views of parties P_c and P_d ; the messages they receive during an execution of the protocol (only lines 10, 22, 30, 34, and 38).

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 4.1.

At line 22, P_c simulates the message received by generating a key pair and using the generated encryption key for E_k . It generates n random numbers to comprise \vec{D}^e and k random numbers to form \vec{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 \vec{Z} . Note that each z_j simulates the encryption of $\text{Const} * (d_j - \mu_{c_j}) - r_j$. Since the operations are over a finite field \mathcal{F} and the r_j is also uniformly chosen over the finite field \mathcal{F} ,

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

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

At lines 34 and 38, the results of the square computation and scalar product are random shares, simulated by both P_c and P_d as shown in Lemma 4.1.

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 . Protocol 2 can thus be simulated, with the composition theorem being applied to the scalar product protocol at lines 10 and 38 and to the square computation protocol at line 34.

THEOREM 4.3. *The evaluation protocol in Section 3.3 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 the parties. By Lemma 4.1, these shares can be independently simulated by both the parties.

For numeric attributes, shares of the means and variances are present with both the parties. The secure ln computation returns random shares of the variance to both the parties, which can be independently simulated. The call to the secure square computation protocol also returns random shares of $(x - \mu)^2$. Finally, the division protocol computes random shares of $(x - \mu)^2/\sigma^2$. By Lemma 4.1 all these shares can be independently simulated by both the parties.

The addition and comparison circuit is a generic circuit, proven secure in [2]. The result is simply the output class, and is simulated exactly as the final result is presumed known by the simulator. Applying the composition theorem to the secure ln computation, protocol 1, protocol 2, and the square computation protocol, the evaluation protocol is also secure.

5 Conclusion.

Often, when legal/commercial reasons restrict sharing data, it may be imprudent to share models generated from the data. We have presented a method that bypasses this restriction.

Space restrictions preclude a detailed analysis of the communication cost. For nominal attributes, assuming k classes and r values for the attributes, protocol 1 is $O(rkn)$; this is reasonable for small values of r and k (where Naïve Bayes is most effective), while building the tree for numeric attributes is $O(kn)$. Evaluating the tree requires an operation for each attribute, where the operations are constant (although non-trivial, dominated by the cost of the Secure ln protocol). Future work will address the *practical* cost of this method, using tools such as hardware cryptographic accelerators.

This paper is based on the semi-honest model. While the components can be extended to the malicious model, doing so efficiently is an interesting research problem. In general, the efficiency of privacy-preserving protocols is open – most are significantly 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.

References

- [1] Wenliang Du and Mikhail J. Atallah. Privacy-preserving statistical analysis. In *Proceeding of the 17th Annual Computer Security Applications Conference*, New Orleans, Louisiana, Dec. 10-14 2001.
- [2] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game - a completeness theorem for protocols with honest majority. In *19th ACM Symposium on the Theory of Computing*, pages 218–229, 1987.
- [3] Oded Goldreich. Secure multi-party computation, Sep. 1998. (working draft).
- [4] Standard for privacy of individually identifiable health information. *Federal Register*, 66(40), Feb. 28 2001.
- [5] Ioannis Ioannidis, Ananth Grama, and Mikhail Atallah. A secure protocol for computing dot-products in clustered and distributed environments. In *The 2002 International Conference on Parallel Processing*, Vancouver, British Columbia, Aug. 18-21 2002.
- [6] Murat Kantarcioglu and Chris Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Trans. Knowledge Data Eng.*, 16(4), Jul. 2004.
- [7] Xiaodong Lin, Chris Clifton, and Michael Zhu. Privacy preserving clustering with distributed EM mixture modeling. *Knowledge and Information Systems*, to appear.
- [8] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002.
- [9] Tom Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1st edition, 1997.
- [10] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 245–254, Atlanta, Georgia, 1999. ACM Press.
- [11] Jaideep Vaidya and Chris Clifton. Privacy preserving association rule mining in vertically partitioned data. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 639–644, Edmonton, Alberta, Canada, Jul. 23-26 2002.
- [12] Jaideep Vaidya and Chris Clifton. Privacy-preserving k -means clustering over vertically partitioned data. In *The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, DC, Aug. 24-27 2003.