Privacy-Preserving Boosting

Esma Aïmeur, Gilles Brassard, Sébastien Gambs and Balázs Kégl *

Université de Montréal
Département d'informatique et de recherche opérationnelle
C.P. 6128, Succ. Centre-Ville, Montréal (Québec), H3C 3J7 Canada
{aimeur,brassard,gambsseb,kegl}@iro.umontreal.ca
http://www.iro.umontreal.ca/∼{aimeur,brassard,gambsseb,kegl}

Abstract. We present an approach that allows two participants to implement an algorithm of the boosting family when the dataset is split between them. Although they are willing to collaborate for the accomplishment of this task of mutual benefit, the participants wish to preserve the privacy of their data. For this purpose, we introduce MABoost (Multiparty Abstention Boost), a distributed and privacy-preserving boosting-by-abstention algorithm.

1 Introduction

Data mining is a domain on the rise, which is at the crossroads of databases, statistics and artificial intelligence. Its main goal can be summarized as *finding useful information inside a vast amount of data*. In our Information Age, in which huge databases are increasingly common, data mining has become very popular. It is now used in a wide range of different areas such as finances, sociology and astrophysics, to name a few.

Secure multiparty computation is the branch of cryptography that deals with *the realization of distributed tasks in a secure manner*, where the definition of security can take different flavours, depending on the setting considered such as preserving the privacy of the data or protecting the computation against malicious participants. Typically, secure multiparty computation consists of computing some function $f(x, y)$, in which input x is in the hands of one participant and input y is in the hands of the other. For the computation to be considered totally secure, the two participants should learn nothing after the completion of the task, except for what can be inferred from the output of the function itself, given their own input and the *a priori* information they had concerning the other input. Yao was the first to describe a technique that enables the implementation of any probabilistic computation between two participants in a secure manner [19]. Later, his results were generalized to the setting of multiple participants [4, 6]. It must be mentioned, however, that although universal and general, these methods can be very inefficient and heavy in terms of communication complexity when the inputs are large and when the function to compute is relatively complicated to describe. When this occurs, it is often more desirable to develop an *ad hoc* solution.

* This research is supported in parts by Canada's Nserc.

In this paper, we study the encounter of data mining and secure multiparty computation, when the task is *the construction of a classifier*. In machine learning, *classification* can be defined as the task of *accurately predicting the class of an object from some observations about this object*. Consider the following scenario: Alice and Bob are directors of two competing banks. They wish to collaborate in order to perform a task of mutual interest. For example, they might want to construct a device that is able to give an advice on whether or not a person is a good candidate for a loan. Each of the two bankers owns a database that contains all the confidential data concerning their clients, upon which the classifier has to be built. Clearly, if Alice and Bob were willing to put their data together (e.g. if Bob would send his database directly to Alice), then the problem of building a classifier would be greatly simplified as no distributed or security aspect would have to be taken into account. But this is not the situation since Alice and Bob would like to protect as much as possible the sensitive information they possess concerning their clients, even if they are willing to cooperate with each other. Therefore, they wish to build a classifier based on their respective databases in a distributed but also confidential manner, meaning that they want to disclose as little information as possible concerning their respective clients. Notice that, obviously, the final classifier does leak some information about the clients, as in the example given above where we want to distinguish between a good payer and a bad payer, but for the protocol to be considered secure we only require that it does not reveal more than it is possible to learn by looking directly at the description of the final classifier. The comparison of different types of classifiers, considering how well and in which sense they preserve privacy [8], is also a relevant and interesting (and of course complex!) question, which we shall not discuss here.

We introduce a *boosting*-like algorithm to create the final classifier. We consider a security model in which the participants are *semi-honest*, also called the *passive* or *honest-but-curious* model, which means that the participants *follow the execution of their prescribed protocols, without any attempt at cheating, but they try to learn as much information as possible concerning the other participant's data by analysing the information exchanged during the protocol* [11]. This model is weaker than if we allowed participants to be malicious and to cheat actively during the protocol, but it is nevertheless useful and often considered in privacy-preserving data mining. Once a protocol has been proved secure in this model, it is natural to attempt upgrading its security towards a stronger model.

The outline of this paper is as follows. First we describe privacy-preserving data mining in Section 2 and we review the three approaches that cope with this problem. We briefly review the principles of boosting in Section 3. In Section 4, we introduce MABoost (Multiparty Abstention Boost), our novel boosting algorithm with abstention that works in a distributed and secure manner. The complexity of the algorithm (both communication and computation) and its security are studied in Sections 5 and 6, respectively. In Section 7, we provide empirical results on the behaviour of MABoost by using it on a real dataset, and we compare its performance with the standard AdaBoost algorithm. Finally, we discuss possible generalizations of our approach in Section 8.

2 Privacy-Preserving Data Mining

Today, the Internet makes it possible to reach and connect sources of information throughout the world, but at the same time it raises many questions concerning their security and privacy. *Privacy-preserving data mining* is an emerging field that studies *how data-mining algorithms can affect the privacy of data* and tries to *find and analyse new algorithms that will preserve this privacy.* A review of the issues and state-of-the-art of privacy-preserving data mining can be found in [18], where some of the algorithms are reviewed and three different trends of approaches for dealing with privacy-preserving issues in data mining are identified.

The algorithms that belong to the first approach are usually based on *heuristics that modify the values of selected attributes on individual records* in order to preserve privacy. This technique of selective data modification is called *sanitization* and has been proven to be NP-hard [3]. When making data sanitization, one always needs to find an *equilibrium between the amount of privacy and the utility loss* resulting from this sanitization process. In order to sanitize data, it is possible to make different modifications on the data such as *altering* the value of an attribute by either *perturbing* it or *replacing* it with the "unknown" value, *swapping* the values of attributes between individual records, *using a coarser granularity* by merging several possible values, using a *sampling* strategy, etc.

With the second approach, the data is modified again, but in a *global* rather than a local manner. By *adding noise drawn from a known distribution* (e.g. Gaussian or uniform), it is possible to construct a classifier and to apply a *reconstruction algorithm*, which will try to clean up the effect of the noise and construct a classifier that is as close as possible to the one constructed on the original distribution [2]. Here also, there will be a trade-off between the *level of privacy* (the noise added) and the *quality of the reconstruction*, which directly affects the performance of the classifier. This approach can be extended in a natural way to the problem of *density estimation*, in which we are trying to estimate directly the shape of the original distribution rather that building a classifier based on this distribution. A reconstruction algorithm based on the principle of Expectation-Maximization (EM), which accurately and efficiently estimates the shape of the original distribution, is described in [1]. It is also possible to tune the intensity and the type of added noise in order to adjust the level of privacy and the quality of the reconstruction.

The third and final approach is very different in spirit from the first two, as it attacks the problem with a *cryptography-based view* [7]. After all, any distributed and privacy-preserving computation can be considered as an instance of a secure multiparty computation, and privacy-preserving data mining is no exception. An algorithm that computes a *modified version of ID3* in a distributed and privacy-preserving manner is described in [13]. ID3 is a well established algorithm that uses an *entropy-based metric* in order to construct a decision tree [14]. The decision tree that is computed in [13] is an *approximation* to what the original ID3 would have computed. It differs from the exact version by a

factor that is a parameter of the algorithm, and has a direct impact on the level of privacy.

The algorithm described in this paper has a cryptographic flavour: it belongs to the third approach. In our case, however, the classifier we are after is not a decision tree but rather the result of applying a boosting scheme.

3 Boosting

Boosting is *a method that enables the creation of an efficient classifier by iteratively combining several weak classifiers* whose predictions have to be only slightly better than random guesses. For example, a *weak classifier* (sometimes called a *weak hypothesis*) could be a simple rule that makes a prediction of the class of an object by looking at the value of a single attribute. Boosting can be seen as a *meta-algorithm* because it combines the output of other algorithms.

The idea of boosting originated from research in Probably Approximately Correct (PAC) Learning [17], which is at the intersection of theoretical computer science and machine learning. It has recently enjoyed a resurgence as it became popular in the machine learning community with the introduction of AdaBoost [10]. This algorithm has been studied extensively both theoretically and experimentally, and it has been shown to have several interesting properties. Among them, it has been proven that AdaBoost *makes the training error decrease exponentially fast with the number of iterations*, and also that it is able to *make the test error continue to decrease even after the training error has vanished*. To illustrate the concept of boosting, we briefly review how AdaBoost works.

Formally, a boosting algorithm takes as input a *training set* and a *family of weak classifiers*. It returns as output a final classifier, which is a *linear combination of several weak classifiers*. The training set D_n is a collection of *data points* $\big\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\big\}$, where each data point $(\mathbf{x}_j, y_j)$ is a pair composed of the *object* itself $\mathbf{x}_j$ and its associated *class* y_j. Object $\mathbf{x}_j$ is usually represented as *a vector of k attributes*, in which $\mathbf{x}_j^{(i)}$ denotes the value of attribute i for this particular object j. For the rest of the paper, we shall limit ourselves to attributes whose values are *real numbers of finite precision*, but there are ways of adapting the algorithm for all kinds of data types, including *integers*, *binary values* or even *symbolic values*. For the *binary classification*, the associated class y_j belongs to $\{-1, +1\}$, where the label "-1" stands for the *negative class* and "$+1$" for *the positive class*.

One of the key features of a boosting algorithm is the ability to *allocate weights to the training points*, which typically represent *how hard the points tend to be to classify*. The higher the weight of a data point, the harder it is to classify it correctly. Let w_j denote the weight of the j^{th} data point. The weights are normalized: $\sum_{j=1}^{n} w_j = 1$. Initially, to indicate that all the data points are *a priori* equally difficult (or easy) to classify, the weight of the all the data points is set to $1/n$.

Boosting can be seen as an *iterative process that, at each step, tries to find the optimal weak classifier for the current distribution of the weights on the data points*. By optimal, we mean a weak classifier that minimizes *the weighted error of the current distribution*. Let $h^{(t)}$ be the weak classifier of the t^{th} iteration and $w_j^{(t)}$ the weight of the j^{th} point at the t^{th} iteration. Minimizing the weighted error at the t^{th} iteration consists in finding the weak classifier $h^{(t)}$ such that

$$\varepsilon^{(t)} = \sum_{j=1}^{n} w_j^{(t)} I_{\{h^{(t)}(\mathbf{x}_j) \neq y_j\}} \tag{1}$$

is minimum, where $\varepsilon^{(t)}$ is the weighted error of the t^{th} iteration and $I_{\{b\}}$ is the indicator function, which is 1 if b is true and 0 otherwise.

The coefficient of $h^{(t)}$, called $\alpha^{(t)}$, indicates how good are the predictions of this classifier. It is directly linked to $\varepsilon^{(t)}$ by the formula

$$\alpha^{(t)} = \frac{1}{2} \ln \frac{1 - \varepsilon^{(t)}}{\varepsilon^{(t)}} \,. \tag{2}$$

After each iteration, the weights of the data points are updated. If a data point was correctly classified, we *decrease* its weight by dividing it by $2(1 - \varepsilon^{(t)})$, but if it was misclassified, we *increase* its weight by multiplying it by $1/2\varepsilon^{(t)}$. After T iterations, AdaBoost outputs a discriminant function of the form

$$f^{(T)}(\cdot) = \sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(\cdot) \,. \tag{3}$$

The sign of $f^{(T)}(\mathbf{x})$ is then used as the final classification of $\mathbf{x}$.

There exist numerous versions and extensions of AdaBoost. In this paper, we shall use a variant proposed by Schapire and Singer [16], which allows the weak classifier not only to answer "-1" or "$+1$", but also to abstain by returning "0". This variant is a particular instance of a more general algorithm that allows the classifier to output a real number from the interval $[-1, +1]$. The absolute value of the output can be interpreted as the confidence that the classifier has in its prediction. If the classifier returns "0", this is considered as an abstention.

4 MABoost

In general, boosting-like algorithms implicitly assume the whole dataset to be available in the hands of one person at the beginning. In our case, we make the assumption that the *dataset is split equally between the two participants* and we call $D_n^A = \left\{(\mathbf{x}_1^A, y_1^A), \ldots, (\mathbf{x}_n^A, y_n^A)\right\}$ and $D_n^B = \left\{(\mathbf{x}_1^B, y_1^B), \ldots, (\mathbf{x}_n^B, y_n^B)\right\}$ the datasets of Alice and Bob, respectively. We wish to design a distributed and privacy-preserving boosting algorithm able to perform boosting while protecting the privacy of the data. MABoost (Multiparty Abstention Boost), which is

described in this section, falls in this category. Designing a boosting algorithm that can abstain on some data was not originally intended, but it naturally followed from our approach to obtaining a distributed and privacy-preserving algorithm. Note that this does not mean that *every* distributed and privacy-preserving boosting algorithm has to use abstention.

The weak classifiers that we use are *decision stumps*. A decision stump is a *very simple decision tree with one root and two leaves*. For example, it could be "if $\mathbf{x}^{(i)} < \theta$ then $\mathbf{x}$ belongs to class C1 else $\mathbf{x}$ belongs to class C2". This family of classifiers may seem very simple *a priori* but it has been empirically proven that one could obtain excellent results by using them as weak classifiers with a boosting algorithm. Formally, decision stumps are defined by the following formulas. (Note that $h^{(i)}_{\theta-}(\mathbf{x}) = -h^{(i)}_{\theta+}(\mathbf{x})$.)

$$h^{(i)}_{\theta+}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x}^{(i)} \geq \theta \\ -1 & \text{otherwise} \end{cases} \tag{4}$$

$$h^{(i)}_{\theta-}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x}^{(i)} < \theta \\ -1 & \text{otherwise} \end{cases} \tag{5}$$

MABoost is an iterative algorithm in which each iteration consists of five steps. See Figure 1 for a description in pseudocode.

Step 1: Finding the Optimal Decision Stump for Each Attribute (lines 4–10 in Fig. 1)**.** Before trying to agree on a common attribute i, Alice and Bob first independently compute from their respective databases the optimal decision stump for each attribute. Let $h^A_{(i)}$ be the weak classifier of Alice for attribute i and let $\varepsilon^A_{(i)}$ be the weighted error of this classifier on her database; $h^B_{(i)}$ and $\varepsilon^B_{(i)}$ are defined similarly for Bob.

Step 2: Agreeing on a Common Attribute (lines 11–13 in Fig. 1; Fig. 2)**.** Alice and Bob are now searching for an attribute i such that $\varepsilon^A_{(i)} < \frac{1}{2}$ and $\varepsilon^B_{(i)} < \frac{1}{2}$ simultaneously. If such an attribute does not exist, they want to be aware of that, and if there are several such attributes, they want to find one *determined at random among them.*

To make this problem intuitively easier to understand, we rephrase it into a totally equivalent well-studied problem. Suppose Alice and Bob each have a agenda with a list of k time slots. Associated with each time slot is a binary value 0 or 1 indicating if this time slot is already filled or if it is still free. If there is one or more time slot that is free for both, they want to find one chosen at random among all the possibilities, and if there is no time slot where they are both free, they want to detect this situation. The problem of Alice and Bob is that they do not trust each other to the point where one could simply send his or her entire agenda to the other. In fact, they do not wish to disclose any unnecessary information about their agenda while they proceed. This problem calls for a solution based on a secure multiparty computation, in which the function $f(x, y)$ that

MABoost(D_n^A, D_n^B, T)

1 $\mathbf{w}^A \leftarrow (1/2n, \ldots, 1/2n)$ ▷ *weights are normalized uniformly*
2 $\mathbf{w}^B \leftarrow (1/2n, \ldots, 1/2n)$
3 **for** $t \leftarrow 1$ **to** T
4 **for** $i \leftarrow 1$ **to** k
5 $h_{(i)}^A \leftarrow$ Alice's best decision stump for attribute i
6 **if** $\varepsilon_{(i)}^A < \frac{1}{2}$ **then** $a_i \leftarrow 1$
7 **else** $a_i \leftarrow 0$
8 $h_{(i)}^B \leftarrow$ Bob's best decision stump for attribute i
9 **if** $\varepsilon_{(i)}^B < \frac{1}{2}$ **then** $b_i \leftarrow 1$
10 **else** $b_i \leftarrow 0$
11 $i \leftarrow$ RandomRendezVous$((a_1, \ldots, a_k), (b_1, \ldots, b_k))$
12 **if** $i =$ "you will never be able to meet each other" **then**
13 **return** $f^{(t-1)}(\cdot) = \sum_{j=1}^{t-1} \alpha^{(j)} h^{(j)}(\cdot)$ ▷ *previous combined classifier*
14 $h^{(t)}(\mathbf{x}) = \begin{cases} h^A(\mathbf{x}) & \text{if } h^A(\mathbf{x}) = h^B(\mathbf{x}) \\ 0 & \text{otherwise} \end{cases}$ ▷ *merged classifier*
15 $\varepsilon_+ = \sum_{j=1}^{n} w_j^A I_{\{h^{(t)}(\mathbf{x}_j^A) = y_j^A\}} + \sum_{j=1}^{n} w_j^B I_{\{h^{(t)}(\mathbf{x}_j^B) = y_j^B\}}$ ▷ *correct rate*
16 $\varepsilon_- = \sum_{j=1}^{n} w_j^A I_{\{h^{(t)}(\mathbf{x}_j^A) = -y_j^A\}} + \sum_{j=1}^{n} w_j^B I_{\{h^{(t)}(\mathbf{x}_j^B) = -y_j^B\}}$ ▷ *error rate*
17 $\varepsilon_0 = 1 - \varepsilon_+ - \varepsilon_-$ ▷ *abstention rate*
18 $\alpha^{(t)} \leftarrow \frac{1}{2} \ln {}^{\varepsilon_+}/_{\varepsilon_-}$ ▷ *coefficient of* $h^{(t)}$
19 **for** $X \leftarrow A, B$; **for** $j \leftarrow 1$ **to** n ▷ *weight update*
20 $w_j^{X\,(t+1)} = w_j^{X\,(t)} \times \begin{cases} \frac{1}{2\varepsilon_- + \varepsilon_0 \sqrt{\varepsilon_-/\varepsilon_+}} & \text{if } h^{(t)}(\mathbf{x}_j^X) = -y_j^X \quad \triangleright \textit{ errors} \\ \frac{1}{2\varepsilon_+ + \varepsilon_0 \sqrt{\varepsilon_+/\varepsilon_-}} & \text{if } h^{(t)}(\mathbf{x}_j^X) = y_j^X \quad \triangleright \textit{ good points} \\ \frac{1}{\varepsilon_0 + 2\sqrt{\varepsilon_+ \varepsilon_-}} & \text{if } h^{(t)}(\mathbf{x}_j^X) = 0 \quad \triangleright \textit{ abstentions} \end{cases}$
21 **return** $f^{(T)}(\cdot) = \sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(\cdot)$ ▷ *final combined classifier*

Fig. 1. Pseudocode for the MABoost algorithm. D_n^A, and D_n^B are Alice's and Bob's training sets, respectively, and T is the number of iterations

they want to compute takes x (the agenda of Alice) and y (the agenda of Bob) as input and returns the index i of a time slot if a common free time slot exists or "you will never be able to meet each other" if there is no such time slot. We call this task the "*random rendez-vous problem*". In the area of communication complexity, this problem was proposed and analysed in [12]: it was proved that it requires $\Theta(n)$ bits of communication on the average. That paper, however, was concerned only about the issue of minimizing communication, but it was not interested in confidentiality or security aspects. Nevertheless, *any distributed and privacy-preserving protocol has to use at least as many bits of communication as the best non privacy-preserving distributed protocol.* Therefore, any privacy-

preserving protocol for this task needs to have a communication complexity of at least $\Omega(n)$ bits, but possibly more since security usually comes with a price.

To solve the random rendez-vous problem, Alice and Bob encode their agendas as bit strings a and b, respectively, where a and $b \in \{0,1\}^k$ and k is the number of time slots in their agenda. For $1 \leq i \leq k$, $a_i = 1$ if Alice is available during the i^{th} time slot and $a_i = 0$ otherwise; b_i is defined similarly for Bob. Suppose that Alice and Bob are given Yao's protocol ORBAN (for the OR of all Bitwise ANd) [19], which can securely compute the function $\bigvee_{i=1}^{k}(a_i \wedge b_i)$, where $\bigvee$ denotes the usual "or", which is true if *one or more* of its arguments are true, and $\wedge$ denotes the "and" function, which is true only if *both* of its arguments are true. For simplicity, we shall consider "1" and "0" to be equivalent to "true" and "false", respectively. Protocol ORBAN(a,b) returns 1 if Alice and Bob have a common time slot and 0 otherwise.

```
RandomRendezVous(A, B)

 1   v ← ORBAN((a_1, ..., a_k), (b_1, ..., b_k))
 2   if v = 0 then
 3       return "you will never be able to meet each other"
 4   else
 5       Permute A and B using same random permutation
 6       i ← 1
 7       p ← k ÷ 2          ▷ binary search of a common time slot
 8       while p ≥ 1 do
 9           v ← ORBAN((a_i, ..., a_{i+p-1}), (b_i, ..., b_{i+p-1}))
10           if v = 0 then
11               i ← i + p          ▷ search in the other half of the space
12           p ← p ÷ 2
13       return i          ▷ return the index of a random common free time slot
```

Fig. 2. Pseudocode for the RandomRendezVous algorithm. A and B are encodings of Alice's and Bob's agendas respectively

Figure 2 gives a formal description of the RandomRendezVous protocol. We can assume without loss of generality that the total number of time slots k is a power of 2. (If this is not the case, it is always possible to add some virtual "I'm unavailable" time slots in order to enforce this condition.) First, Alice and Bob use ORBAN with their entire agendas to determine if they have an available time slot in common. If this is the case, they can search for this time slot by applying a procedure similar to binary search, which tries to locate an empty common slot by putting aside half of the search space at each iteration. Therefore, in $\lg k$ iterations, they find a common empty time slot. For this time slot to be chosen at random among all the possible free time slots, Alice and Bob initially *randomize* their inputs by both applying the same random permutation on the indices of their time slots. By doing so, they avoid always finding the common free time slot whose index is the lowest.

Imagine now that we use the RandomRendezVous protocol not with an agenda, but rather with a list in which a slot is set to 1 if the participant was able to find a weak classifier with weighted error below $1/2$ for the attribute associated to this slot, and to 0 otherwise. If Alice and Bob each make such a list and then apply the RandomRendezVous protocol, they are able to discover a common attribute for which they both have a weak classifier with error smaller than $1/2$, provided such an attribute exists. Otherwise, they find out that such an attribute does not exist. In the latter case, they stop the execution of MABoost and return as output the final classifier $f^{(t-1)}(\cdot)$.

Step 3: Merging the Weak Classifiers (line 14 in Fig. 1)**.** Once Alice and Bob have agreed on an attribute i, they merge the two weak classifiers they each computed for this attribute, which only described their own data, into a classifier that covers the whole dataset. Let h^A and h^B be Alice's and Bob's weak classifiers, respectively. The merged classifier h will abstain on objects that h^A and h^B classify differently, and it will take the commonly predicted class when h^A and h^B agree.

$$h(\mathbf{x}) = \begin{cases} h^A(\mathbf{x}) & \text{if } h^A(\mathbf{x}) = h^B(\mathbf{x}) \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

The training error [1] of AdaBoost converges to zero if the weighted error of every weak classifier is strictly less than $1/2$. If abstention is allowed, convergence is guaranteed provided the weighted error of each weak classifier is less than the weighted rate of correctly classified points. Said otherwise, the error must be less than $1/2$ on the subset of the points on which the weak classifier does not abstain.

We now proceed to show that if h^A and h^B are optimal on their respective datasets and if they satisfy the convergence criterion of AdaBoost, then the merged classifier satisfies the convergence criterion of AdaBoost with abstention. By optimality, we mean that

$$h^A = \arg\min_{h \in \mathcal{H}} \sum_{j=1}^{n} w_j^A I_{\{h(\mathbf{x}_j^A) = -y_j^A\}} \tag{7}$$

and

$$h^B = \arg\min_{h \in \mathcal{H}} \sum_{j=1}^{n} w_j^B I_{\{h(\mathbf{x}_j^B) = -y_j^B\}}\,, \tag{8}$$

where $\mathcal{H}$ is the set of weak classifiers on which Alice and Bob have agreed (decision stumps over the i^{th} attribute in our case). For a formal description, let

$$\varepsilon_-^A = \sum_{j=1}^{n} w_j^A I_{\{h^A(\mathbf{x}_j^A) = -y_j^A\}} \quad \text{and} \quad \varepsilon_+^A = \sum_{j=1}^{n} w_j^A I_{\{h^A(\mathbf{x}_j^A) = y_j^A\}} \tag{9}$$

[1] The number of training points $(\mathbf{x}_i, y_i)$ on which $\text{sign}(f^{(T)}(\mathbf{x}_i)) \neq y_i$.

be the weighted error and the weighted rate of correctly classified points, respectively, of Alice's classifier h^A, and let ε_-^B and ε_+^B be defined similarly for Bob. Furthermore, let

$$\omega_-^A = \sum_{j=1}^{n} w_j^A I_{\{h(\mathbf{x}_j^A) = -y_j^A\}} \quad \text{and} \quad \omega_+^A = \sum_{j=1}^{n} w_j^A I_{\{h(\mathbf{x}_j^A) = y_j^A\}} \tag{10}$$

be the weighted error and the weighted rate of correctly classified points, respectively, of the merged classifier h on Alice's data points, and let ω_-^B and ω_+^B be defined similarly for Bob. (Please note the subtle difference between the definitions of $\varepsilon_\pm^A$ and $\omega_\pm^A$. The former uses Alice's classifier h^A whereas the latter uses the merged classifier h.) Finally, let

$$\varepsilon_- = \omega_-^A + \omega_-^B\,, \quad \varepsilon_+ = \omega_+^A + \omega_+^B \tag{11}$$

and

$$\varepsilon_0 = \sum_{j=1}^{n} w_j^A I_{\{h(\mathbf{x}_j^A) = 0\}} + \sum_{j=1}^{n} w_j^B I_{\{h(\mathbf{x}_j^B) = 0\}} \tag{12}$$

be the weighted error, the weighted rate of correctly classified points, and the weighted abstention rate, respectively, of the merged classifier h. (Note that $\varepsilon_- + \varepsilon_+ + \varepsilon_0 = 1$.) Then the following holds.

Lemma 1. *If $\mathcal{H}$ is closed under multiplication by -1 and if h^A and h^B are optimal, then $\varepsilon_- \le \varepsilon_+$. Furthermore, $\varepsilon_- = \varepsilon_+$ only if $-h^A$ is optimal on Bob's dataset D_n^B and $-h^B$ is optimal on Alice's dataset D_n^A.*

Proof. The main observation of the proof is that $\omega_-^A \le \omega_+^A$, otherwise the classifier $-h^B$ would have a lower error on Alice's dataset D_n^A than Alice's chosen classifier h^A, which would violate the optimality of h^A. To see this, first note that by the definition (6) of the merged classifier h, $-h^B(\mathbf{x})$ agrees with $h^A(\mathbf{x})$ on objects $\mathbf{x}$ on which h abstains, and they disagree otherwise. Thus, the error of $-h^B$ on D_n^A is the sum of 1) the error of h^A on the subset of D_n^A where h abstains, that is, $(\varepsilon_-^A - \omega_-^A)$, and 2) the rate of correctly classified points of h^A on the subset of D_n^A where h does not abstain, that is, ω_+^A. Hence, if $\omega_+^A < \omega_-^A$, then the error $(\varepsilon_-^A - \omega_-^A + \omega_+^A)$ of $-h^B$ is less then the error ε_-^A of h^A, which violates the optimality of h^A. It can be shown in a similar way that $\omega_-^B \le \omega_+^B$, and the first statement of the Lemma follows. The second statement is also easy to see by observing that $\omega_-^A \le \omega_+^A$, $\omega_-^B \le \omega_+^B$ and $\varepsilon_- = \varepsilon_+$ imply $\omega_-^A = \omega_+^A$ and $\omega_-^B = \omega_+^B$, so the error of $-h^B$ on D_n^A is $(\varepsilon_-^A - \omega_-^A + \omega_+^A) = \varepsilon_-^A$, and, similarly, the error of $-h^A$ on D_n^B is ε_-^B. □

Remark 1. If $\mathcal{H}$ is closed under multiplication by -1, then $\varepsilon_-^A \le \varepsilon_+^A$ (the error $\varepsilon_-^{A\prime}$ of $-h^A$ is equal to ε_+^A and hence h^A would not be optimal if $\varepsilon_-^A > \varepsilon_+^A = \varepsilon_-^{A\prime}$). Moreover, if $-h^A$ is optimal, then $\varepsilon_-^A = \varepsilon_+^A$ can happen only if we have equality for *all* weak classifiers, in which case AdaBoost must be stopped. The same thing happens in MABoost if Alice and Bob pick exactly the same classifier but

with opposite signs. If we add to the protocol that in this case other attributes should be examined, then MABoost has to be stopped only if Alice and Bob pick exactly opposite classifiers for *all* attributes.

Remark 2. The optimality of h^A and h^B is crucial for the lemma. It is easy to find an example of a non-optimal h^A for which $\varepsilon_-^A < \varepsilon_+^A$, yet $\varepsilon_- > \varepsilon_+$. Such an h^A would be admissible for AdaBoost if Alice were playing alone, but it would make the merged classifier fail in MABoost.

Remark 3. The generality of the lemma allows us to consider any kind of weak classifier class. In the extreme case, Alice and Bob could find the best classifier over the whole class. In this case they would not need the RandomRendezVous protocol to find the best attribute; they would only need to exchange their best classifiers at each iteration. In general, they could divide the weak classifier class into k (not necessarily disjoint) subclasses, as we did in the particular case of decision stumps. The only algorithmic design issue here is that they should be able to minimize the training error over each of the subclasses. This general version also gives us more flexibility for the cryptographic design.

Step 4: Computing the Coefficient of the Merged Classifier (lines 15–18 in Fig. 1)**.** The coefficient $\alpha^{(t)}$ of the merged classifier $h^{(t)}$ depends only on ε_- and ε_+. It is formally defined as

$$\alpha^{(t)} = \frac{1}{2} \ln \frac{\varepsilon_+}{\varepsilon_-} \,. \tag{13}$$

Intuitively, the smaller the error ε_- of $h^{(t)}$, the larger its coefficient $\alpha^{(t)}$ with which it participates in the final vote. In order to avoid a singularity when ε_- becomes too small, Schapire and Singer [16] suggest to use instead

$$\alpha^{(t)} = \frac{1}{2} \ln \frac{\varepsilon_+ + \delta}{\varepsilon_- + \delta} \,, \tag{14}$$

where δ is a small appropriate constant.

Step 5: Updating the Weight of the Data Points (lines 19 and 20 in Fig. 1)**.** Updating the weights of the data points can be made *independently* by Alice and Bob once they know the description of the merged classifier h, as well as its weighted error ε_-, its weighted rate of correctly classified points ε_+ and its weighted abstention rate ε_0. In general, we increase slightly the weights of points on which h abstains, increase the weight of misclassified points, and decrease the weights of correctly classified points.

Once Step 5 is completed, Alice and Bob can choose to start a new iteration or to stop the algorithm and output $f^{(T)}(\cdot)$. The number T of iterations of MABoost they choose to perform can be decided in advance or chosen adaptively depending on some criteria such as the performance of the classifier they currently have. The number of iterations also influences the description length of the final classifier, and therefore it is possible to play with this parameter in order to tune the level of privacy.

5 Complexity

When studying a distributed protocol, one can be interested in two different measures of complexity: its *communication complexity*, which is the number of bits exchanged during the execution of the protocol, and its *computational complexity*, which considers the processing time required from the participants.

5.1 Communication Complexity

To define the quantity of communication used in one iteration of MABoost, we need to *quantify the number of bits exchanged* during this iteration. First, it is easy to observe that the only steps that require communication between Alice and Bob are the second, the third and the fourth. During the first step, Alice and Bob compute independently their optimal weak classifiers for each attribute and therefore they do not need to talk to each other. The same situation occurs during the fifth step, when they do not need to engage in a dialog in order to reweigh their data points. In fact, the only step that has a significant cost in terms of communication is the second one, when they have to agree on a common attribute. Let k be the number of attributes for an object. Alice and Bob use the random rendez-vous protocol to find the common attribute upon which they build the classifier h. This protocol requires $\lg k$ calls to the protocol ORBAN in the worst case and one call to ORBAN at best if it turns out that they will not be able to agree on a common attribute. In this latter case, they stop MABoost and return directly $f^{(T)}(\cdot)$. This situation can be considered marginal as it happens at most once.

The computational complexity of the random rendez-vous protocol is in $\Theta(\textit{complexity of}\ \text{ORBAN}\ \textit{protocol} \times \log k)$. There exists more than one way of implementing the ORBAN protocol, but if we use in a straightforward manner the technique described in [19], its complexity is in $\Theta(k)$. The overall complexity of the RandomRendezVous protocol is therefore in $\Theta(k \log k)$. During the third and fourth steps, some communication also occurs, but the cost of this communication is constant because the exchange of the descriptions of h^A and h^B necessary for the creation of h and the communication of ε_+^A, ε_-^A, ε_-^B and ε_+^B can be done with a number of bits that is constant for a fixed finite precision. Thus, the cost of Steps 3 and 4 is negligible compared to Step 2, and the dominance of the latter implies that the overall cost of one iteration of MABoost is in $\Theta(k \log k)$. The cost of the whole protocol is therefore in $\Theta(Tk \log k)$, where T is the total number of iterations of MABoost.

5.2 Computational Complexity

The only moment at which Alice and Bob require some work from their respective computers is at the beginning of the iteration, during the first step, when they both have to compute the optimal weak classifier for each attribute. The naïve way to find the best weak classifier for an attribute is to first sort the objects according to this attribute and then to find the optimal threshold

by looking at the objects in order to find the threshold at which the weighted error is minimized. The most costly operation is the sorting of objects, which can be done in $\Theta(n \log n)$ if we are only concerned by one attribute, the rest of the operations being negligible compared to this cost. As the sorting has to be repeated for each attribute, the global cost for the k attributes is in $\Theta(kn \log n)$ for the first iteration. Note that it is enough to sort once the objects for each attribute during the first iteration as it is possible to store the results in order to use them for the remaining iterations. The storage of these values has a space complexity of $\Theta(kn)$. Computing the error of the classifier h and reweighing the points can be done in $\Theta(n)$ as it only requires looking at each individual data point once in order to perform both tasks. Therefore we have a computational complexity in $\Theta(kn \log n)$ for the first iteration because of the sorting process and a complexity of $\Theta(kn)$ for the remaining iterations. The overall time complexity of MABoost is a function of T, the number of iterations, and is in the order of $\Theta((T + \log n)kn)$. This complexity increases linearly with the number of attributes. If there are too many attributes, the processing time of MABoost could be prohibitive.

6 Security of the Protocol

A multiparty protocol can be considered totally secure if its execution does not reveal more information than the output of the protocol itself. In cryptography, this is formalized with the *simulation paradigm* [11]. Intuitively, this means that if the view that each participant has of the protocol can be efficiently simulated solely based on his input and the output of the protocol, then the different participants learn nothing from the execution of the protocol other than its output.

Imagine a simulator that has access to the input of Alice (her database) and to the output of the protocol (the final classifier). Would it be possible for this simulator to create a transcript of the execution of the protocol that would be indistinguishable from a real transcript of the execution of the protocol? If we can answer this question positively, then we can assert that the protocol privately computes f, in particular in the sense of the simulation paradigm. For the analysis of the security of the protocol, we only have to concentrate on the second, third and fourth steps as they are the only steps in which communication takes place. It is important to observe that in the case of MABoost, the process of creating the final classifier is iterative and that the information exchanged in the clear between Alice and Bob during Steps 3 and 4 is explicitly contained inside the description of this final classifier. This observation gives us for "free" the security aspect of these two steps. Indeed, why care to protect the privacy of some information during the execution of the protocol when this information is going to be revealed at the end of the protocol anyway? Therefore, the only step we have to be careful about for the analysis of the security of the protocol is the second step.

The second step consists of the random rendez-vous protocol, which heavily relies on the ORBAN protocol as a subroutine. For this step to be considered secure, Alice and Bob must learn nothing other than the index i of a common attribute for which they both have a weak classifier with weighted error smaller than $^1/_2$. Suppose we use a secure ORBAN protocol, as in [19]. The first call to ORBAN only reveals to Alice and Bob whether or not they can agree on a common attribute. If it is not the case, then they stop the protocol and they have learned no extra information. On the other hand, if Alice and Bob are able to continue with the protocol, they select half of the attributes and make another call to ORBAN. From this call, they will learn whether or not there is at least one attribute they can agree upon in this half of the attribute space. If ORBAN returns 1, they learn that there is at least one potential attribute in the part of the attribute space but nothing else about the other half of the space. But if ORBAN returns 0, they learn that there is no attribute they can agree upon on the considered part of the attribute space *and* that there is at least one such attribute in the other part of the space. In this latter case, the protocol has leaked one bit of additional information. As there will be $\lg k$ calls to ORBAN at the most, the protocol reveals $O(\log k)$ bits of extra information before the two parties agree on a common attribute. This cannot be considered a totally privacy-preserving protocol, and we must acknowledge that a small amount of information will leak. To which extend could someone do something useful with this kind of information is still an open question. Note that it would be possible to design a totally secure protocol by using more complicated and sophisticated protocols, which solve the set intersection problem [9], but this would come with a significant increase of the communication cost. Would this additional cost be warranted given that we cannot prevent the final classifier from leaking information anyway?

7 Empirical Results

In order to test and illustrate the MABoost algorithm in practice, we chose to observe its behaviour on the "ionosphere" dataset, which can be found among the benchmarks of the repository of the University of California at Irvine (UCI) [5]. In particular, this dataset was used in the original paper about confidence-rated boosting [16]. This dataset is concerned with the classification of radar data from the ionosphere, collected in Goose Bay, Labrador. "Good" data are those showing evidence of some type of structure in the ionosphere; "Bad" data are those that do not.

The ionosphere dataset contains 351 examples, each consisting of 34 continuous attributes and one class attribute. The class attribute has only two possible values ("Good" or "Bad"), which makes this dataset well suited for binary classification. For MABoost, we decided to split the dataset into three sets: the training set of Alice (40%), the training set of Bob (40%) and the test set (20%). We ran a simulation of MABoost on these sets in order to observe how the training error on Alice's set, the training error of Bob's set and

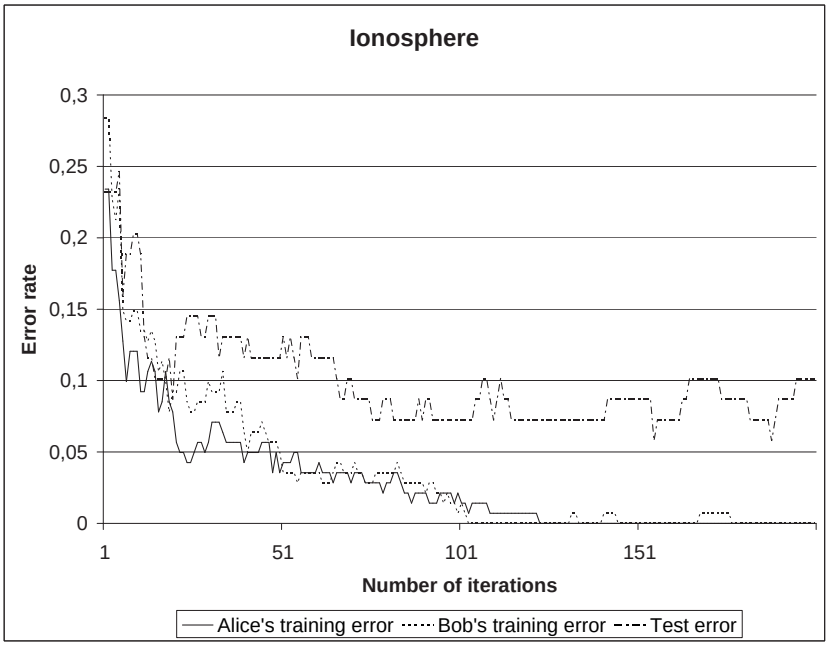

Fig. 3. Evolution of Alice's training error, Bob's training error and the test error on 200 iterations

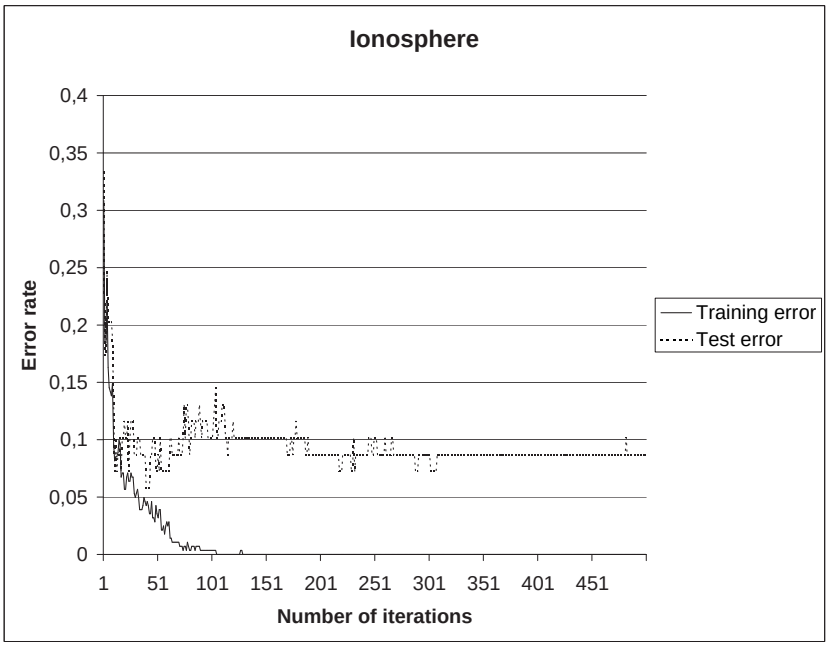

Fig. 4. Evolution of the training error and the test error on 500 iterations

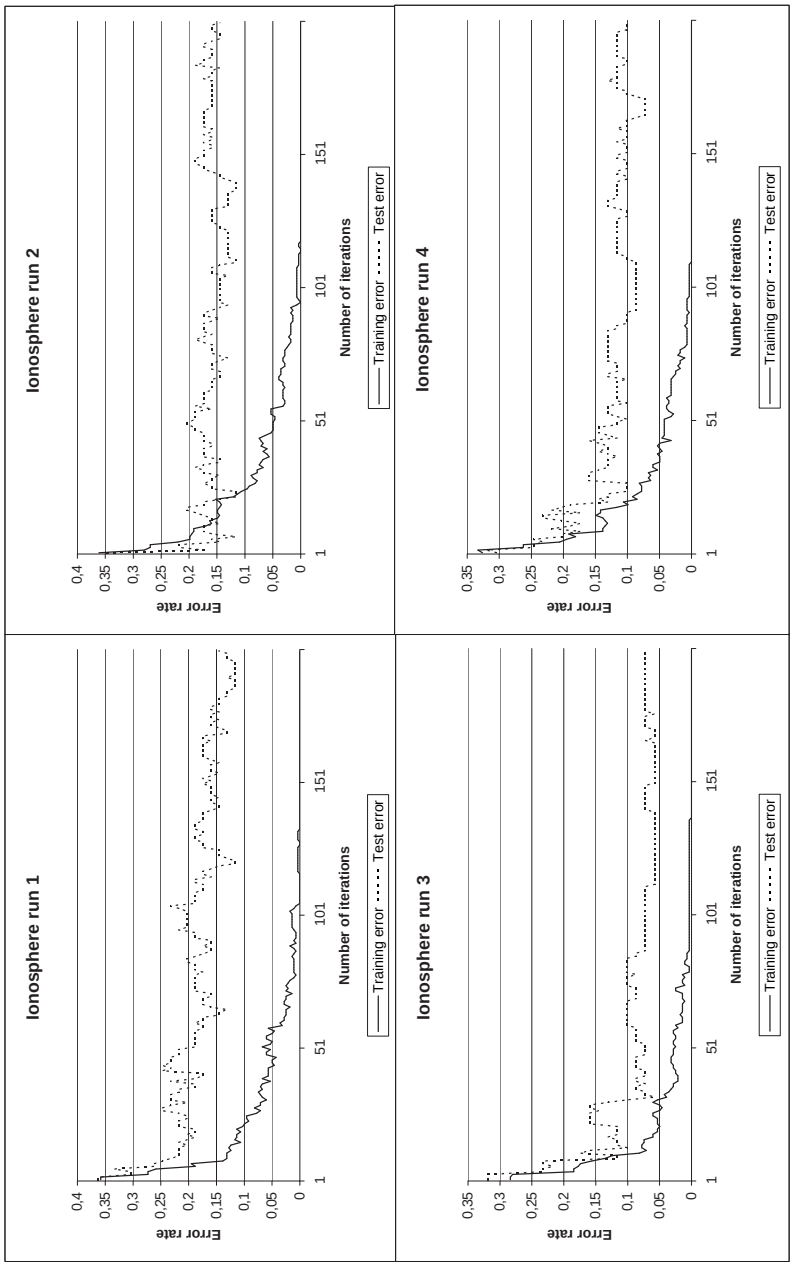

Fig. 5. Training error and test error during four different runs of MABoost on the ionosphere dataset

the test error evolve with the number of iterations. Figures 3 and 4 illustrate the results when we carry out 200 and 500 iterations, respectively. (We illustrate both Alice's and Bob's training errors separately in Figure 3, whereas we illustrate the average training error of Alice and Bob in Figure 4.) The most important observation is that MABoost inherits a crucial property from AdaBoost: *the training error decreases exponentially with the number of iterations*. Our data is not sufficient to conclude that the test error continues to decrease even after the training error has vanished (another crucial property of AdaBoost), but some of the runs shown in Figure 5 indicate that this may be the case.

One important difference between the two algorithms is that AdaBoost is *deterministic* (because it always chooses the best attribute for which the weighted error of the weak classifier is minimal), whereas MABoost is *probabilistic* (because it chooses an attribute at random among all those whose weighted error is less than $^1/_2$ both for Alice and for Bob). It follows that the behaviour of MABoost could vary even if we run it twice on the same dataset. Figure 5 illustrates four different runs of MABoost on the ionosphere dataset. Although it can be observed that the training error does not always vanish after the same number of iterations, and that the test error does not decrease with the same intensity and the same speed each time, these differences do not seem to be really significant. Moreover, if we compare the convergence speed of AdaBoost in the original paper [16] with that of MABoost, no significant differences can be observed. These preliminary results indicate the possibility that boosting can be performed in a distributed and privacy-preserving manner without incurring a severe adverse effect on the performance of the final classifier. However, more experiments are needed before a conclusive statement can be made.

8 Possible Generalizations and Conclusion

In this paper, we have presented a protocol that performs a boosting-like algorithm in a distributed and nearly privacy-preserving manner in the restricted case that we only deal with two semi-honest participants, where the weak classifiers have the form of decision stumps. There are at least two natural ways of generalizing this protocol, first by allowing the number of participants to be greater than 2, and also by looking at other families of weak classifiers such as more complex decision trees or neural networks with a few units in the hidden layer.

To extend this protocol to the case of m participants, with $m \geq 2$, we could define at each iteration a classifier h that carries out a majority vote from the individual weak classifiers h_1, h_2, ..., h_m of the different participants. The classifier could return the vote of the majority of the classifiers, for example, when this majority is above a threshold set to $\frac{1}{2} + \delta$ and abstain otherwise, δ being a well-chosen constant between 0 and $^1/_2$ whose value can be tuned in order to

increase the confidence of the classifier h in its prediction. The protocol described in this paper is a particular instance of this more general method.

It could be interesting also to test this algorithm with other types of weak classifiers and to compare the results. In particular, it would be interesting to observe empirically how the choice of the type of weak classifiers, the number of bits required to describe these classifiers (which in a sense is a measure of their complexity), and the number of iterations influence the convergence speed of MABoost. For example, it is conceivable that with a richer family of weak classifiers such as decision trees generated by the algorithm C4.5 [15], one can obtain an identical or better performance than with the decision stumps in fewer iterations, meaning that the convergence would be faster. However, this would come at the price of communicating more bits at each iteration because the decision trees inferred by C4.5 require a greater (and possibly unbounded) number of bits for their description. One could define a criterion to take into account at the same time the number of iterations and the expected size needed to describe the weak classifiers, which would be able to compare two possible choices of families of weak classifiers even in case of equal performance. Intuitively, we are searching for weak classifiers that can be described with few bits, but which at the same time yield a good performance after a "sensible" number of iterations of MABoost. This would make it possible to bound directly the number of bits exchanged during the protocol and therefore give an upper bound on the quantity of information that has been communicated.

For some families of weak classifiers, there must exist ways of merging weak classifiers without requiring the resulting classifier h to abstain on a part of the dataset. Being able to merge two or more weak classifiers into a weak classifier that does not abstain would allow us to apply the original AdaBoost algorithm directly in a distributed manner. Potentially, the convergence could be a little faster in practice in this case.

We are also investigating the privacy-preserving part of Step 2 in Section 4. In particular, we are working on a more efficient randomized version of the set intersection problem. An even more promising direction would be to find an *efficient* privacy-preserving technique to determine the weak classifier that minimizes the weighted error over the entire dataset shared between Alice and Bob. This would make MABoost as discriminating as the non-distributed AdaBoost.

The work we have described in this paper is still in progress. It is certainly not sufficient to test our algorithm on the sole ionosphere dataset. More experimentation is needed in order to compare MABoost with the original AdaBoost in terms of performance and convergence speed. In particular, it is important to study the behaviour of the test error once the training error has vanished. This will allow us to make a more convincing evaluation of how much the preservation of privacy has an adverse influence on the performance of the final classifier, if at all.

References

1. D. Agrawal and C. C. Aggarwal, "On the design and quantification of privacy preserving data mining algorithms", *Proceedings of the 20th ACM Symposium of Principles of Databases Systems*, pp. 247 – 255, 2001.
2. R. Agrawal and R. Ramakrishnan, "Privacy-preserving data mining", *Proceedings of the ACM SIGMOD on Management of Data*, pp. 439 – 450, 2000.
3. M. J. Atallah, E. Bertino, A. K. Elmagarmid, M. Ibrahim and V. S. Verykios, "Disclosure limitations of sensitive rules", *Proceedings of the IEEE Knowledge and Data Engineering Workshop*, pp. 45 – 52, 1999.
4. M. Ben–Or, S. Goldwasser and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation", *Proceedings of the 20th ACM Annual Symposium on the Theory of Computing*, pp. 1 – 10, 1988.
5. C. L. Blake and C. J. Merz, "UCI Repository of machine learning databases", Available at `http://www.ics.uci.edu/~mlearn/MLRepository.html`, Irvine, CA: University of California, Department of Information and Computer Science, 1998.
6. D. Chaum, C. Crépeau and I. Damgård, "Multiparty unconditionally secure protocols", *Proceedings of the 20th ACM Annual Symposium on the Theory of Computing*, pp. 11 – 19, 1988.
7. C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin and M. Y. Zhiu, "Tools for privacy preserving distributed data mining", *SIGKDD Explorations* **4**(2):28 – 34, 2002.
8. E. Evfimievski, J. E. Gehrke and R. Srikant, "Limiting privacy breaches in privacy preserving data mining", *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Databases Systems*, pp. 211 – 222, 2003.
9. M. Freedman, K. Nissim and B. Pinkas, "Efficient private matching and set intersection", *Proceedings of Eurocrypt'04*, pp. 1 – 19, 2004.
10. Y. Freund and R. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting", *Journal of Computer and System Sciences* **55**(1):119 – 139, 1997.
11. O. Goldreich, *Foundations of Cryptography, Volume II: Basic Applications*, Cambridge University Press, 2004.
12. B. Kalyanasundaran and G. Schnitger, "The probabilistic communication of set intersection", *Proceedings of the 2nd Annual IEEE Conference on Structure in Complexity Theory*, pp. 41 – 47, 1987.
13. Y. Lindell and B. Pinkas, "Privacy preserving data mining", *Journal of Cryptology* **15**:177 – 206, 2002.
14. J.R. Quinlan, "Induction of decision trees", *Machine Learning* **1**(1):81 – 106, 1986.
15. J.R. Quinlan, "Bagging, boosting and C4.5", *Proceedings of the 13th National Conference on Artificial Intelligence*, pp. 725 – 730, 1996.
16. R. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions", *Machine Learning* **37**(7):297 – 336, 1999.
17. L. Valiant, "A theory of the learnable", *Communications of the ACM* **27**(11):1134 – 1142, 1984.
18. V. S. Verykios, E. Bertino, I. N. Fovino, L. P. Provenza, Y. Saygin and Y. Theodoridis, "State-of-the-art in privacy preserving data mining", *SIGMOD Record* **33**(1):50 – 57, 2004.
19. A. C.-C. Yao, "How to generate and exchange secrets", *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pp. 162 – 167, 1986.