Machine Learning Classification over Encrypted Data

Raphael Bost
DGA MI
MIT
raphael_bost@alumni.brown.edu

Raluca Ada Popa
ETH Zürich and MIT
rpopa@inf.ethz.ch

Stephen Tu
MIT
stephentu@csail.mit.edu

Shafi Goldwasser
MIT
shafi@theory.csail.mit.edu

***Abstract*—Machine learning classification is used for numerous tasks nowadays, such as medical or genomics predictions, spam detection, face recognition, and financial predictions. Due to privacy concerns, in some of these applications, it is important that the data and the classifier remain confidential.**

In this work, we construct three major classification protocols that satisfy this privacy constraint: hyperplane decision, Naïve Bayes, and decision trees. We also enable these protocols to be combined with AdaBoost. At the basis of these constructions is a new library of building blocks, which enables constructing a wide range of privacy-preserving classifiers; we demonstrate how this library can be used to construct other classifiers than the three mentioned above, such as a multiplexer and a face detection classifier.

We implemented and evaluated our library and our classifiers. Our protocols are efficient, taking milliseconds to a few seconds to perform a classification when running on real medical datasets.

I. Introduction

Classifiers are an invaluable tool for many tasks today, such as medical or genomics predictions, spam detection, face recognition, and finance. Many of these applications handle sensitive data [1], [2], [3], so it is important that the data and the classifier remain private.

Consider the typical setup of supervised learning, depicted in Figure 1. Supervised learning algorithms consist of two phases: (i) the training phase during which the algorithm learns a model w from a data set of labeled examples, and (ii) the classification phase that runs a classifier C over a previously unseen feature vector x, using the model w to output a prediction $C(x, w)$.

In applications that handle sensitive data, it is important that the feature vector x and the model w remain secret to one or some of the parties involved. Consider the example of a medical study or a hospital having a model built out of the private medical profiles of some patients; the model is sensitive because it can leak information about the patients, and its usage has to be HIPAA[1] compliant. A client wants to use the model to make a prediction about her health (e.g., if she is likely to contract a certain disease, or if she would be treated successfully at the hospital), but does not want to reveal her sensitive medical profile. Ideally, the hospital and the client run a protocol at the end of which the client learns one bit ("yes/no"), and neither party learns anything else about the other party's input. A similar setting arises for a financial institution (*e.g.*, an insurance company) holding a sensitive model, and a customer wanting to estimate rates or quality of service based on her personal information.

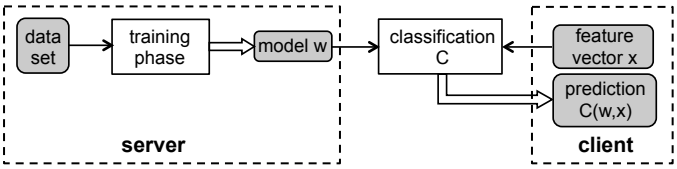


Fig. 1: Model overview. Each shaded box indicates private data that should be accessible to only one party: the dataset and the model to the server, and the input and prediction result to the client. Each straight non-dashed rectangle indicates an algorithm, single arrows indicate inputs to these algorithms, and double arrows indicate outputs.

Throughout this paper, we refer to this goal shortly as *privacy-preserving classification*. Concretely, a client has a private input represented as a feature vector x, and the server has a private input consisting of a private model w. The way the model w is obtained is independent of our protocols here. For example, the server could have computed the model w after running the training phase on plaintext data as usual. Only the classification needs to be privacy-preserving: the client should learn $C(x, w)$ but nothing else about the model w, while the server should not learn anything about the client's input or the classification result.

In this work, we construct efficient privacy-preserving protocols for three of the most common classifiers: hyperplane decision, Naïve Bayes, and decision trees, as well as a more general classifier combining these using AdaBoost. These classifiers are widely used – even though there are many machine learning algorithms, most of them end up using one of these three classifiers, as described in Table I.

While generic secure multi-party computation [4], [5], [6], [7], [8] can implement any classifier in principle, due to their generality, such schemes are not efficient for common classifiers. As described in Section X-E, on a small classification instance, such tools ([6], [8]) ran out of memory on a powerful machine with 256GB of RAM; also, on an artificially simplified classification instance, these protocols ran $\approx$ 500 times slower

[1]Health Insurance Portability and Accountability Act of 1996

NDSS '15, 8-11 February 2015, San Diego, CA, USA

http://dx.doi.org/10.14722/ndss.2015.23241

Machine learning algorithm	Classifier
Perceptron	Hyperplane decision
Least squares	Hyperplane decision
Fischer linear discriminant	Hyperplane decision
Support vector machine	Hyperplane decision
Naive Bayes	Naïve Bayes
Decision trees (ID3/C4.5)	Decision trees

TABLE I: Machine learning algorithms and their classifiers, defined in Section III-A.

than our protocols ran on the non-simplified instance.

Hence, protocols specialized to the classification problem promise better performance. However, most existing work in machine learning and privacy [9], [10], [11], [12], [13], [14], [15] focuses on preserving privacy during the *training phase*, and does not address classification. The few works on privacy-preserving classification either consider a weaker security setting in which the client learns the model [16] or focus on specific classifiers (*e.g.*, face detectors [17], [18], [19], [20]) that are useful in limited situations.

Designing efficient privacy-preserving classification faces two main challenges. The first is that the computation performed over sensitive data by some classifiers is quite complex (e.g., decision trees), making it hard to support efficiently. The second is providing a solution that is more generic than the three classifiers: constructing a separate solution for each classifier does not provide insight into how to combine these classifiers or how to construct other classifiers. Even though we contribute privacy-preserving protocols for three of the most common classifiers, various settings use other classifiers or use a combination of these three classifiers (e.g., AdaBoost). We address these challenges using two key techniques.

Our main technique is to identify *a set of core operations* over encrypted data that underlie many classification protocols. We found these operations to be comparison, argmax, and dot product. We use *efficient* protocols for each one of these, either by improving existing schemes (e.g., for comparison) or by constructing new schemes (e.g., for argmax).

Our second technique is to design these building blocks in a *composable* way, with regard to *both* functionality and security. To achieve this goal, we use a set of sub-techniques:

- The input and output of all our building blocks are data encrypted with additively homomorphic encryption. In addition, we provide a mechanism to switch from one encryption scheme to another. Intuitively, this enables a building block's output to become the input of another building block;
- The API of these building blocks is flexible: even though each building block computes a fixed function, it allows a choice of which party provides the inputs to the protocol, which party obtains the output of the computation, and whether the output is encrypted or decrypted;
- The security of these protocols composes using modular sequential composition [21].

We emphasize that the contribution of our building blocks library goes beyond the classifiers we build in this paper: a user of the library can construct other privacy-preserving classifiers in a modular fashion. To demonstrate this point, we use our building blocks to construct a multiplexer and a classifier for face detection, as well as to combine our classifiers using AdaBoost.

We then use these building blocks to construct novel privacy-preserving protocols for three common classifiers. Some of these classifiers incorporate additional techniques, such as an efficient evaluation of a decision tree with fully homomorphic encryption (FHE) based on a polynomial representation requiring only a small number of multiplications and based on SIMD FHE slots (see Section VII-B). All of our protocols are secure against passive adversaries (see Section III-B3).

We also provide an implementation and an evaluation of our building blocks and classifiers. We evaluate our classifiers on real datasets with private data about breast cancer, credit card approval, audiology, and nursery data; our algorithms are efficient, running in milliseconds up to a few seconds, and consume a modest amount of bandwidth.

The rest of the paper is organized as follows. Section II describes related work, Section III provide the necessary machine learning and cryptographic background, Section IV presents our building blocks, Sections V–VIII describe our classifiers, and Sections IX–X present our implementation and evaluation results.

II. Related work

Our work is the first to provide efficient privacy-preserving protocols for a broad class of classifiers.

Secure two-party computation protocols for generic functions exist in theory [4], [5], [22], [23], [24] and in practice [6], [7], [8]. However, these rely on heavy cryptographic machinery, and applying them directly to our problem setting would be too inefficient as exemplified in Section X-E.

Previous work focusing on privacy-preserving machine learning can be broadly divided into two categories: (i) techniques for privacy-preserving training, and (ii) techniques for privacy-preserving classification (recall the distinction from Figure 1). Most existing work falls in the first category, which we discuss in Section II-A. Our work falls in the second category, where little work has been done, as we discuss in Section II-B. We also mention work related to the building blocks we use in our protocols in Section II-C.

It is worth mentioning that our work on privacy-preserving classification is complementary to work on differential privacy in the machine learning community (see *e.g.* [25]). Our work aims to hide each user's input data to the classification phase, whereas differential privacy seeks to construct classifiers/models from sensitive user training data that leak a bounded amount of information about each individual in the training data set.

A. Privacy-preserving training

A set of techniques have been developed for privacy-preserving *training* algorithms such as Naïve Bayes [14], [11], [12], decision trees [13], [9], linear discriminant classifiers [10], and more general kernel methods [26].

Grapel *et al.* [15] show how to train several machine learning classifiers using a somewhat homomorphic encryption scheme. They focus on a few simple classifiers (*e.g.* the linear means classifier), and do not elaborate on more complex algorithms such as support vector machines. They also support private classification, but in a weaker security model where the client learns more about the model than just the final sign of the classification. Indeed, performing the final comparison with fully homomorphic encryption (FHE) alone is inefficient, a difficulty we overcome with an interactive setting.

B. Privacy-preserving classification

Little work has been done to address the general problem of privacy-preserving classification in practice; previous work focuses on a weaker security setting (in which the client learns the model) and/or only supports specific classifiers.

In Bos *et al.* [16], a third party can compute medical prediction functions over the encrypted data of a patient using fully homomorphic encryption. In their setting, everyone (including the patient) knows the predictive model, and their algorithm hides only the input of the patient from the cloud. Our protocols, on the other hand, also hide the model from the patient. Their algorithms cannot be applied to our setting because they leak more information than just the bit of the prediction to the patient. Furthermore, our techniques are notably different; using FHE directly for our classifiers would result in significant overheads.

Barni *et al.* [27], [28] construct secure evaluation of linear branching programs, which they use to implement a secure classifier of ECG signals. Their technique is based on finely-tuned garbled circuits. By comparison, our construction is not limited to branching programs (or decision trees), and our evaluation shows that our construction is twice as fast on branching programs. In a subsequent work [29], Barni *et al.* study secure classifiers based on neural networks, which is a generalization of the perceptron classifiers, and hence also covered by our work.

Other works [17], [18], [19], [20] construct specific face recognition or detection classifiers. We focus on providing a set of *generic* classifiers and building blocks to construct more complex classifiers. In Section X-A2, we show how to construct a private face detection classifier using the modularity of our techniques.

C. Work related to our building blocks

Two of the basic components we use are private comparison and private computation of dot products. These items have been well-studied previously; see [4], [30], [31], [32], [33], [19], [34] for comparison techniques and [35], [36], [37], [19] for techniques to compute dot products. Section IV-A discusses how we build on these tools.

III. Background and Preliminaries

A. Classification in machine learning algorithms

The user's input x is a vector of d elements $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$, called a feature vector. To classify the input x means to evaluate a classification function $C_w : \mathbb{R}^d \mapsto \{c_1, ..., c_k\}$ on x. The output is $c_{k^*} = C_w(x)$, where $k^* \in \{1 \ldots k\}$; c_{k^*} is the class to which x corresponds, based on the model w. For ease of notation, we often write k^* instead of c_{k^*}, namely $k^* = C_w(x)$.

We now describe how three popular classifiers work on regular, unencrypted data. These classifiers differ in the model w and the function C_w. For more details, we refer the reader to [38].

Hyperplane decision-based classifiers. For this classifier, the model w consists of k vectors in $\mathbb{R}^d$ ($w = \{w_i\}_{i=1}^k$). The classifier is (*cf.* [38]):

$$k^* = \underset{i \in [k]}{\operatorname{argmax}} \langle w_i, x \rangle, \tag{1}$$

where $\langle w_i, x \rangle$ denotes inner product between w_i and x.

We now explain how Eq. (1) captures many common machine learning algorithms. A hyperplane based classifier typically works with a hypothesis space $\mathcal{H}$ equipped with an inner product $\langle \cdot, \cdot \rangle$. This classifier usually solves a binary classification problem ($k = 2$): given a user input x, x is classified in class c_2 if $\langle w, \phi(x) \rangle \geq 0$, otherwise it is labeled as part of class c_1. Here, $\phi : \mathbb{R}^d \mapsto \mathcal{H}$ denotes the feature mapping from $\mathbb{R}^d$ to $\mathcal{H}$ [38]. In this work, we focus on the case when $\mathcal{H} = \mathbb{R}^d$ and note that a large class of infinite dimensional spaces can be approximated with a finite dimensional space (as in [39]), including the popular gaussian kernel (RBF). In this case, $\phi(x) = x$ or $\phi(x) = Px$ for a randomized projection matrix P chosen during training. Notice that Px consists solely of inner products; we will show how to support private evaluation of inner products later, so for simplicity we drop P from the discussion. To extend such a classifier from 2 classes to k classes, we use one of the most common approaches, *one-versus-all*, where k different models $\{w_i\}_{i=1}^k$ are trained to discriminate each class from all the others. The decision rule is then given by (*cf.* [38]) to be Eq. (1). This framework is general enough to cover many common algorithms, such as support vector machines (SVMs), logistic regression, and least squares.

Naïve Bayes classifiers. For this classifier, the model w consists of various probabilities: the probability that each class c_i occurs, namely $\{p(C = c_i)\}_{i=1}^k$, and the probabilities that an element x_j of x occurs in a certain class c_i. More concretely, the latter is the probability of the j-th component x_j of x to be v when x belongs to category c_i; this is denoted by $\{\{\{p(X_j = v | C = c_i)\}_{v \in D_j}\}_{j=1}^d\}_{i=1}^k$, where D_j is X_j's domain[2]. The classification function, using a *maximum a posteriori* decision rule, works by choosing the class with the highest posterior probability:

$$\begin{aligned} k^* &= \underset{i \in [k]}{\operatorname{argmax}}\, p(C = c_i | X = x) \\ &= \underset{i \in [k]}{\operatorname{argmax}}\, p(C = c_i, X = x) \\ &= \underset{i \in [k]}{\operatorname{argmax}}\, p(C = c_i, X_1 = x_1, \ldots, X_d = x_d) \end{aligned}$$

where the second equality follows from applying Bayes' rule (we omitted the normalizing factor $p(X = x)$ because it is the same for a fixed x).

[2]Be careful to distinguish between X_j, the probabilistic random variable representing the values taken by the j-th feature of user's input, and x_j, the actual value taken by the specific vector x.

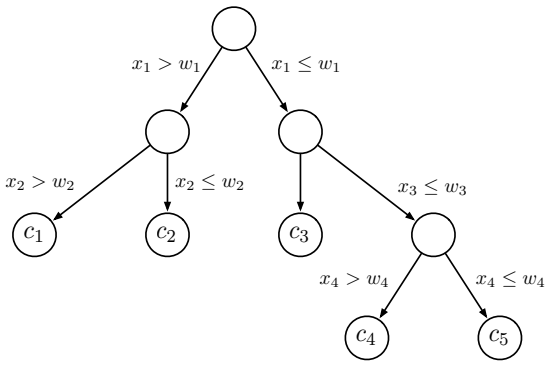


Fig. 2: Decision tree

The Naïve Bayes model assumes that $p(C = c_i, X = x)$ has the following factorization:

$$p(C = c_i, X_1 = x_1, \ldots, X_d = x_d) = p(C = c_i) \prod_{j=1}^{d} p(X_j = x_j | C = c_i),$$

namely, each of the d features are conditionally independent given the class. For simplicity, we assume that the domain of the features values (the x_i's) is discrete and finite, so the $p(X_j = x_j | C = c_i)$'s are probability masses.

Decision trees. A decision tree is a non-parametric classifier which works by partitioning the feature vector space one attribute at a time; interior nodes in the tree correspond to partitioning rules, and leaf nodes correspond to class labels. A feature vector x is classified by walking the tree starting from the root, using the partitioning rule at each node to decide which branch to take until a leaf node is encountered. The class at the leaf node is the result of the classification.

Figure 2 gives an example of a decision tree. The model consists of the structure of the tree and the decision criteria at each node (in this case the thresholds $w_1, \ldots, w_4$).

B. Cryptographic preliminaries

1) Cryptosystems: In this work, we use three additively homomorphic cryptosystems. A public-key encryption scheme HE is additively homomorphic if, given two encrypted messages $\mathsf{HE.Enc}(a)$ and $\mathsf{HE.Enc}(b)$, there exists a public-key operation $\oplus$ such that $\mathsf{HE.Enc}(a) \oplus \mathsf{HE.Enc}(b)$ is an encryption of $a+b$. We emphasize that these are homomorphic *only for addition*, which makes them efficient, unlike fully homomorphic encryption [40], which supports any function. The cryptosystems we use are:

1) the QR (Quadratic Residuosity) cryptosystem of Goldwasser-Micali [41],
2) the Paillier cryptosystem [42], and
3) a leveled fully homomorphic encryption (FHE) scheme, HELib [43]

2) Cryptographic assumptions: We prove that our protocols are secure based on the semantic security [44] of the above cryptosystems. These cryptosytems rely on standard and well-studied computational assumptions: the Quadratic Residuosity assumption, the Decisional Composite Residuosity assumption, and the Ring Learning With Error (RLWE) assumption.

3) Adversarial model: We prove security of our protocols using the secure two-party computation framework for passive adversaries (or honest-but-curious [44]) defined in our extended paper [45]. To explain what a passive adversary is, at a high level, consider that a party called party A is compromised by such an adversary. This adversary tries to learn as much private information about the input of the other party by watching all the information party A receives; nevertheless, this adversary cannot prevent party A from following the prescribed protocol faithfully (hence, it is not an active adversary).

To enable us to compose various protocols into a bigger protocol securely, we invoke modular sequential composition (see our extended paper [45]).

C. Notation

All our protocols are between two parties: parties A and B for our building blocks and parties C (client) and S (server) for our classifiers.

Inputs and outputs of our building blocks are either unencrypted or encrypted with an additively homomorphic encryption scheme. We use the following notation. The plaintext space of QR is $\mathbb{F}_2$ (bits), and we denote by $[b]$ a bit b encrypted under QR; the plaintext space of Paillier is $\mathbb{Z}_N$ where N is the public modulus of Paillier, and we denote by $[\![m]\!]$ an integer m encrypted under Paillier. The plaintext space of the FHE scheme is $\mathbb{F}_2$. We denote by SK_P and PK_P, a secret and a public key for Paillier, respectively. Also, we denote by SK_{QR} and PK_{QR}, a secret and a public key for QR.

For a constant b, $a \leftarrow b$ means that a is assigned the value of b. For a distribution $\mathcal{D}$, $a \leftarrow \mathcal{D}$ means that a gets a sample from $\mathcal{D}$.

IV. Building Blocks

In this section, we develop a library of building blocks, which we later use to build our classifiers. We designed this library to also enable constructing other classifiers than the ones described in our paper. The building blocks in this section combine existing techniques with either new techniques or new optimizations.

Proofs. We provide formal cryptographic proofs of the security and correctness of the protocols. This task naturally requires space beyond the page limit. Hence, in this paper, we provide only the intuition behind the proofs, and delegate all formal proofs to the extended paper [45].

A. Comparison

We now describe our comparison protocol. In order for this protocol to be used in a wide range of classifiers, its setup needs to be *flexible*: namely, it has to support a range of choices regarding which party gets the input, which party gets the output, and whether the input or output are encrypted or

Type	Input A	Input B	Output A	Output B	Implementation
1	PK_P, PK_{QR}, a	SK_P,SK_{QR}, b	$[a < b]$	–	Sec. IV-A1
2	PK_P, SK_{QR}, $[\![a]\!]$, $[\![b]\!]$	SK_P,PK_{QR}	–	$[a \leq b]$	Sec. IV-A2
3	PK_P, SK_{QR}, $[\![a]\!]$, $[\![b]\!]$	SK_P,PK_{QR}	$a \leq b$	$[a \leq b]$	Sec. IV-A2
4	PK_P, PK_{QR}, $[\![a]\!]$, $[\![b]\!]$	SK_P,SK_{QR}	$[a \leq b]$	–	Sec. IV-A3
5	PK_P, PK_{QR},$[\![a]\!]$, $[\![b]\!]$	SK_P,SK_{QR}	$[a \leq b]$	$a \leq b$	Sec. IV-A3

TABLE II: The API of our comparison protocol and its implementation. There are five types of comparisons each having a different setup.

not. Table II shows the various ways our comparison protocol can be used. In each case, each party learns *nothing else* about the other party's input other than what Table II indicates as the output.

We implemented each row of Table II by modifying existing protocols. We explain only the modifications here, and defer full protocol descriptions to our extended paper [45].

There are at least two approaches to performing comparison efficiently: using specialized homomorphic encryption [30], [31], [17], [32], or using garbled circuits [46]. We compared empirically the performance of these approaches and concluded that the former is more efficient for comparison of encrypted values, and the second is more efficient for comparison of unencrypted values.

1) Comparison with unencrypted inputs (Row 1): To compare unencrypted inputs, we use garbled circuits implemented with the state-of-the-art garbling scheme of Bellare et al. [46], the short circuit for comparison of Kolesnikov et al. [34] and a well-known oblivious transfer (OT) scheme due to Naor and Pinkas [47]. Since most of our other building blocks expect inputs encrypted with homomorphic encryption, one also needs to convert from a garbled output to homomorphic encryption to enable composition. We can implement this easily using the random shares technique in [48].

The above techniques combined give us the desired comparison protocol. Actually, we can directly combine them to build an even more efficient protocol: we use an enhanced comparison circuit that also takes as input a masking bit. Using a garbled circuit and oblivious transfer, A will compute $(a < b) \oplus c$ where c is a bit randomly chosen by B. B will also provide an encryption $[c]$ of c, enabling A to compute $[a < b]$ using the homomorphic properties of QR.

2) Comparison with encrypted inputs (Rows 2, 3): Our classifiers also require the ability to compare two encrypted inputs. More specifically, suppose that party A wants to compare two encrypted integers a and b, but party B holds the decryption key. To implement this task, we slightly modify Veugen's [32] protocol: it uses a comparison with unencrypted inputs protocol as a sub-procedure, and we replaced it with the comparison protocol we just described above. This yields a protocol for the setup in Row 2. To ensure that A receives the plaintext output as in Row 3, B sends the encrypted result to A who decrypts it. Our extended paper [45] provides the detailed protocol.

3) Reversed comparison over encrypted data (Row 4, 5): In some cases, we want the result of the comparison to be held by the party that does not hold the encrypted data. For this, we modify Veugen's protocol to reverse the outputs of party A and party B: we achieve this by exchanging the role of party A and party B in the last few steps of the protocol, after invoking the comparison protocol with unencrypted inputs. We do not present the details in the paper body because they are not insightful, and instead include them in our extended paper [45].

This results in a protocol whose specification is in Row 4. To obtain Row 5, A sends the encrypted result to B who can decrypt it.

4) Negative integers comparison and sign determination: Negative numbers are handled by the protocols above unchanged. Even though the Paillier plaintext size is "positive", a negative number simply becomes a large number in the plaintext space due to cyclicity of the space. As long as the values encrypted are within a preset interval $(-2^\ell, 2^\ell)$ for some fixed ℓ, Veugen's protocol and the above protocols work correctly.

In some cases, we need to compute the sign of an encrypted integer $[\![b]\!]$. In this case, we simply compare to the encryption of 0.

B. argmax *over encrypted data*

In this scenario, party A has k values $a_1, \ldots, a_k$ encrypted under party B's secret key and wants party B to know the argmax over these values (the index of the largest value), but neither party should learn anything else. For example, if A has values $[\![1]\!]$, $[\![100]\!]$ and $[\![2]\!]$, B should learn that the second is the largest value, but learn nothing else. In particular, B should not learn the order relations between the a_i's.

Our protocol for argmax is shown in Protocol 1. We now provide intuition into the protocol and its security.

Intuition. Let's start with a strawman. To prevent B from learning the order of the k values $\{a_i\}_{i=1}^k$, A applies a random permutation π. The i-th element becomes $[\![a'_i]\!] = [\![a_{\pi(i)}]\!]$ instead of $[\![a_i]\!]$.

Now, A and B compare the first two values $[\![a'_1]\!]$ and $[\![a'_2]\!]$ using the comparison protocol from row 4 of Table II. B learns the index, m, of the larger value, and tells A to compare $[\![a'_m]\!]$ to $[\![a'_3]\!]$ next. After iterating in this manner through all the k values, B determines the index m of the largest value. A can then compute $\pi^{-1}(m)$ which represents the argmax in the original, unpermuted order.

Since A applied a random permutation π, B does not learn the ordering of the values. The problem, though, is that A learns this ordering because, at every iteration, A knows the value of m up to that step and π. One way to fix this problem is for B to compare every pair of inputs from A, but this would result in a quadratic number of comparisons, which is too slow.

Instead, our protocol preserves the linear number of comparisons from above. The idea is that, at each iteration, once B determines which is the maximum of the two values compared, B should *randomize* the encryption of this maximum in such a way that A cannot link this value to one of the values compared. B uses the Refresh procedure for the randomization of Paillier ciphertexts. In the case where the "refresher" knows the secret key, this can be seen as a decryption followed by a re-encryption. If not, it can be seen as a multiplication by an encryption of 0.

A difficulty is that, to randomize the encryption of the maximum $[\![a'_m]\!]$, B needs to get this encryption – however, B must not receive this encryption because B has the key SK_P to decrypt it, which violates privacy. Instead, the idea is for A itself to add noise r_i and s_i to $[\![a'_m]\!]$, so decryption at B yields random values, then B refreshes the ciphertext, and then A removes the randomness r_i and s_i it added.

In the end, our protocol performs $k-1$ encrypted comparisons of l bits integers and $7(k-1)$ homomorphic operations (refreshes, multiplications and subtractions). In terms of round trips, we add $k-1$ roundtrips to the comparison protocol, one roundtrip per loop iteration.

Protocol 1 argmax over encrypted data

Input A: k encrypted integers $([\![a_1]\!], \ldots, [\![a_k]\!])$, the bit length l of the a_i, and public keys PK_{QR} and PK_P
Input B: Secret keys SK_P and SK_{QR}, the bit length l
Output A: $\text{argmax}_i\, a_i$

A: chooses a random permutation π over $\{1, \ldots, k\}$
A: $[\![\mathsf{max}]\!] \leftarrow [\![a_{\pi(1)}]\!]$
B: $m \leftarrow 1$
for $i = 2$ **to** k **do**
  Using the comparison protocol (Sec. IV-A3), B gets the bit $b_i = (\mathsf{max} \leq a_{\pi(i)})$
  A picks two random integers $r_i, s_i \leftarrow (0, 2^{\lambda+l}) \cap \mathbb{Z}$
  A: $[\![m'_i]\!] \leftarrow [\![\mathsf{max}]\!] \cdot [\![r_i]\!]$ ▷ $m'_i = \mathsf{max} + r_i$
  A: $[\![a'_i]\!] \leftarrow [\![a_{\pi(i)}]\!] \cdot [\![s_i]\!]$ ▷ $a'_i = a_{\pi(i)} + s_i$
  A sends $[\![m'_i]\!]$ and $[\![a'_i]\!]$ to B
  if b_i is true **then**
    B: $m \leftarrow i$
    B: $[\![v_i]\!] \leftarrow \mathsf{Refresh}[\![a'_i]\!]$ ▷ $v_i = a'_i$
  else
    B: $[\![v_i]\!] \leftarrow \mathsf{Refresh}[\![m'_i]\!]$ ▷ $v_i = m'_i$
  end if
  B sends to A $[\![v_i]\!]$
  B sends to A $[\![b_i]\!]$
  A: $[\![\mathsf{max}]\!] \leftarrow [\![v_i]\!] \cdot (g^{-1} \cdot [\![b_i]\!])^{r_i} \cdot [\![b_i]\!]^{-s_i}$
    ▷ $\mathsf{max} = v_i + (b_i - 1) \cdot r_i - b_i \cdot t_i$
end for
B sends m to A
A outputs $\pi^{-1}(m)$

Proposition 4.1: Protocol 1 is correct and secure in the honest-but-curious model.

Proof intuition. The correctness property is straightforward. Let's argue security. A does not learn intermediary results in the computation because of the security of the comparison protocol and because she gets a refreshed ciphertext from B which A cannot couple to a previously seen ciphertext. B does learn the result of each comparison – however, since A applied a random permutation before the comparison, B learns no useful information. See our extended paper [45] for a complete proof.

C. Changing the encryption scheme

To enable us to compose various building blocks, we developed a protocol for converting ciphertexts from one encryption scheme to another while maintaining the underlying plaintexts. We first present a protocol that switches between two encryption schemes with the *same* plaintext size (such as QR and FHE over bits), and then present a different protocol for switching from QR to Paillier.

Concretely, consider two additively homomorphic encryption schemes E_1 and E_2, both semantically secure with the same plaintext space M. Let $[\![.]\!]_1$ be an encryption using E_1 and $[\![.]\!]_2$ an encryption using E_2. Consider that party B has the secret keys SK_1 and SK_2 for both schemes and A has the corresponding public keys PK_1 and PK_2. Party A also has a value encrypted with PK_1, $[\![c]\!]_1$. Our protocol, protocol 2, enables A to obtain an encryption of c under E_2, $[\![c]\!]_2$ without revealing anything to B about c.

Protocol intuition. The idea is for A to add a random noise r to the ciphertext using the homomorphic property of E_1. Then B decrypts the resulting value with E_1 (obtaining $x+r \in M$) and encrypts it with E_2, sends the result to A which removes the randomness r using the homomorphic property of E_2. Even though B was able to decrypt $[\![c']\!]_1$, B obtains $x + r \in M$ which hides x in an information-theoretic way (it is a one-time pad).

Protocol 2 Changing the encryption scheme

Input A: $[\![c]\!]_1$ and public keys PK_1 and PK_2
Input B: Secret keys SK_1 and SK_2
Output A: $[\![c]\!]_2$

A uniformly picks $r \leftarrow M$
A sends $[\![c']\!]_1 \leftarrow [\![c]\!]_1 \cdot [\![r]\!]_1$ to B
B decrypts c' and re-encrypts with E_2
B sends $[\![c']\!]_2$ to A
A: $[\![c]\!]_2 = [\![c']\!]_2 \cdot [\![r]\!]_2^{-1}$
A outputs $[\![c]\!]_2$

Note that, for some schemes, the plaintext space M depends on the secret keys. In this case, we must be sure that party A can still choose uniformly elements of M without knowing it. For example, for Paillier, $M = \mathbb{Z}_N^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ where p and q are the private primes. However, in this case, A can sample noise in $\mathbb{Z}_N$ that will not be in $\mathbb{Z}_N^*$ with negligible probability $(1 - \frac{1}{p})(1 - \frac{1}{q}) \approx 1 - \frac{2}{\sqrt{N}}$ (remember N is large – 1024 bits in our instantiation).

Proposition 4.2: Protocol 2 is secure in the honest-but-curious model.

In our classifiers, we use this protocol for $M = \{0, 1\}$ and the encryption schemes are QR (for E_1) and an FHE scheme over bits (for E_2). In some cases, we might also want to switch from QR to Paillier (*e.g.* reuse the encrypted result of a comparison in a homomorphic computation), which has

a different message space. Note that we can *simulate* the homomorphic XOR operation and a message space $M = \{0, 1\}$ with Paillier: we can easily compute the encryption of $b_1 \oplus b_2$ under Paillier when at most one of the b_i is encrypted (which we explain in the next subsection). This is the case in our setting because party A has the randomness r in the clear.

1) XOR with Paillier.: Suppose a party gets the bit b_1 encrypted under Paillier's encryption scheme, and that this party only has the public key. This party knows the bit b_2 in the clear and wants to compute the encryption of $[\![b_1 \oplus b_2]\!]$.

To do so, we just have to notice that

$$b_1 \oplus b_2 = \begin{cases} b_1 & \text{if } b_2 = 0 \\ 1 - b_1 & \text{if } b_2 = 1 \end{cases}$$

Hence, it is very easy to compute an encryption of $b_1 \oplus b_2$ if we know the modulus N and the generator g (*cf.* Paillier's scheme construction):

$$[\![b_1 \oplus b_2]\!] = \begin{cases} [\![b_1]\!] & \text{if } b_2 = 0 \\ g[\![b_1]\!]^{-1} \bmod N^2 & \text{if } b_2 = 1 \end{cases}$$

If we want to unveil the result to an adversary who knows the original encryption of b_1 (but not the secret key), we have to refresh the result of the previous function to ensure semantic security.

D. Computing dot products

For completeness, we include a straightforward algorithm for computing dot products of two vectors, which relies on Paillier's homomorphic property.

Protocol 3 Private dot product

Input A: $x = (x_1, \ldots, x_d) \in \mathbb{Z}^d$, public key PK_P
Input B: $y = (y_1, \ldots, y_d) \in \mathbb{Z}^d$, secret key SK_P
Output A: $[\![\langle x, y \rangle]\!]$

1: B encrypts $y_1, \ldots, y_d$ and sends the encryptions $[\![y_i]\!]$ to A
2: A computes $[\![v]\!] = \prod_i [\![y_i]\!]^{x_i} \mod N^2$ ▷ $v = \sum y_i x_i$
3: A re-randomizes and outputs $[\![v]\!]$

Proposition 4.3: Protocol 3 is secure in the honest-but-curious model.

E. Dealing with floating point numbers

Although all our protocols manipulate integers, classifiers usually use floating point numbers. Hence, when developing classifiers with our protocol library, we must adapt our protocols accordingly.

Fortunately, most of the operations involved are either additions or multiplications. As a consequence, a simple solution is to multiply each floating point value by a constant K (*e.g.* $K = 2^{52}$ for IEEE 754 doubles) and thus support finite precision. We must also consider the bit length for the comparisons. We show an example of a full analysis in Section VI for the Naïve Bayes classifier.

V. Private hyperplane decision

Recall from Section III-A that this classifier computes

$$k^* = \underset{i \in [k]}{\operatorname{argmax}} \langle w_i, x \rangle.$$

Now that we constructed our library of building blocks, it is straightforward to implement this classifier securely: the client computes the encryption of $[\![\langle w_i, x \rangle]\!]$ for all $i \in [k]$ using the dot product protocol and then applies the argmax protocol (Protocol 1) to the encrypted dot products.

Protocol 4 Private hyperplane decision

Client's (C) Input: $x = (x_1, \ldots, x_d) \in \mathbb{Z}^d$, public keys PK_P and PK_{QR}
Server's (S) Input: $\{w_i\}_{i=1}^k$ where $\forall i \in [k], w_i \in \mathbb{Z}^n$, secret keys SK_P and SK_{QR}
Client's Output: $\underset{i \in [k]}{\operatorname{argmax}} \langle w_i, x \rangle$

1: **for** $i = 1$ **to** k **do**
2: C and S run Protocol 3 for private dot product where C is party A with input x and S is party B with input w_i.
3: C gets $[\![v_i]\!]$ the result of the protocol.
▷ $v_i \leftarrow \langle x, w_i \rangle$
4: **end for**
5: C and S run Protocol 1 for argmax where C is the A, and S the B, and $[\![v_1]\!], \ldots, [\![v_k]\!]$ the input ciphertexts. C gets the result i_0 of the protocol.
▷ $i_0 \leftarrow \underset{i \in [k]}{\operatorname{argmax}} v_i$
6: C outputs i_0

Proposition 5.1: Protocol 4 is secure in the honest-but-curious model.

VI. Secure Naïve Bayes classifier

Section III-A describes the Naïve Bayes classifier. The goal is for the client to learn k^* without learning anything about the probabilities that constitute the model, and the server should learn nothing about x. Recall that the features values domain is discrete and finite.

As is typically done for numerical stability reasons, we work with the logarithm of the probability distributions:

$$\begin{aligned} k^* &= \underset{i \in [k]}{\operatorname{argmax}} \ \log p(C = c_i | X = x) \\ &= \underset{i \in [k]}{\operatorname{argmax}} \left\{ \log p(C = c_i) + \sum_{j=1}^{d} \log p(X_j = x_j | C = c_i) \right\} \end{aligned} \tag{2}$$

A. Preparing the model

Since the Paillier encryption scheme works with integers, we convert each log of a probability from above to an integer by multiplying it with a large number K (recall that the plaintext space of Paillier is large $\approx 2^{1024}$ thus allowing for a large K), thus still maintaining high accuracy. The issues due to using integers for bayesian classification have been previously studied in [49], even though their setting was even more restricting

than ours. However, they use a similar idea to ours: *shifting* the probabilities logarithms and use fixed point representation.

As the only operations used in the classification step are additions and comparisons (*cf.* Equation (2)), we can just multiply the conditional probabilities $p(x_j|c_i)$ by a constant K so to get integers everywhere, while keeping the same classification result.

For example, if we are able to compute the conditional probabilities using IEEE 754 double precision floating point numbers, with 52 bits of precision, then we can represent every probability p as

$$p = m \cdot 2^e$$

where m binary representation is $(m)_2 = 1.d$ and d is a 52 bits integer. Hence we have $1 \leq m < 2$ and we can rewrite m as

$$m = \frac{m'}{2^{52}} \text{ with } m' \in \mathbb{N} \cap [2^{52}, 2^{53})$$

We are using this representation to find a constant K such that $K \cdot v_i \in \mathbb{N}$ for all i. As seen before, we can write the v_i's as

$$v_i = m'_i \cdot 2^{e_i - 52}$$

Let $e^* = \min_i e_i$, and $\delta_i = e_i - e^* \geq 0$. Then,

$$v_i = m'_i \cdot 2^{\delta_i} \cdot 2^{e^* - 52}$$

So let $K = 2^{52-e^*}$. We have $K \cdot v_i = m'_i \cdot 2^{\delta_i} \in \mathbb{N}$. An important thing to notice is that the v_i's can be very large integers (due to δ_i), and this might cause overflows errors. However, remember that we are doing all this to store logarithms of probabilities in Paillier cyphertexts, and as Paillier plaintext space is *very* large (more than 1024 bits in our setting) and δ_i's remain small[3]. Also notice that this *shifting* procedure can be done without any loss of precision as we can directly work with the bit representation of the floating points numbers.

Finally, we must also ensure that we do not overflow Paillier's message space when doing all the operations (homomorphic additions, comparisons, ...). If – as before – d is the number of features, the maximum number of bits when doing the computations will be $l_{max} = d + 1 + (52 + \delta^*)$ where $\delta^* = \max \delta_i$: we have to add the probabilities for the d features and the probability of the class label (the $d + 1$ term), and each probability is encoded using $(52 + \delta^*)$ bits. Hence, the value l used for the comparison protocols must be chosen larger than l_{max}.

Hence, we must ensure that $\log_2 N > l_{max} + 1 + \lambda$ where λ is the security parameter and N is the modulus for Paillier's cryptosystem plaintext space (*cf.* Section IV-A2). This condition is easily fulfilled as, for a good level of security, we have to take $\log_2 N \geq 1024$ and we usually take $\lambda \approx 100$.

Let D_j be the domain of possible values of x_j (the j-th attribute of the feature vector x). The server prepares $kd + 1$ tables as part of the model, where K is computed as described just before:

- One table for the priors on the classes P: $P(i) = \lceil K \log p(C = c_i) \rceil$.
- One table per feature j per class i, $T_{i,j}$: $T_{i,j}(v) \approx \lceil K \log p(X_j = v | C = c_i) \rceil$, for all $v \in D_j$.

The tables remain small: P has one entry by category *i.e.* k entries total, and T has one entry by category and feature value *i.e.* $k \cdot D$ entries where $D = \sum |D_j|$. In our examples, this represents less than 3600 entries. Moreover, this preparation step can be done once and for all at server startup, and is hence amortized.

B. Protocol

Let us begin with some intuition. The server encrypts each entry in these tables with Paillier and gives the resulting encryption (the encrypted model) to the client. For every class c_i, the client uses Paillier's additive homomorphism to compute $[\![p_i]\!] = [\![P(i)]\!] \prod_{j=1}^{d} [\![T_{i,j}(x_j)]\!]$. Finally, the client runs the argmax protocol, Protocol 1, to get $\operatorname{argmax} p_i$. For completeness, the protocol is shown in Protocol 5.

Protocol 5 Naïve Bayes Classifier

Client's (C) Input: $x = (x_1, \ldots, x_d) \in \mathbb{Z}^d$, public key PK_P, secret key SK_{QR}
Server's (S) Input: The secret key SK_P, public key PK_{QR} and probability tables $\{\log p(C = c_i)\}_{1 \leq i \leq k}$ and $\left\{ \{\log p(X_j = v | C = c_i)\}_{v \in D_j} \right\}_{1 \leq j \leq d, 1 \leq i \leq k}$
Client's Output: i_0 such that $p(x, c_{i_0})$ is maximum

1: The server prepares the tables P and $\{T_{i,j}\}_{1 \leq i \leq k, 1 \leq j \leq d}$ and encrypts their entries using Paillier.
2: The server sends $[\![P]\!]$ and $\{[\![T_{i,j}]\!]\}_{i,j}$ to the client.
3: For all $1 \leq i \leq k$, the client computes $[\![p_i]\!] = [\![P(i)]\!] \prod_{j=1}^{d} [\![T_{i,j}(x_j)]\!]$.
4: The client runs the argmax protocol (Protocol 1) with the server and gets $i_0 = \operatorname{argmax}_i p_i$
5: C outputs i_0

Proposition 6.1: Protocol 5 is secure in the honest-but-curious model.

Proof intuition. Given the security property of the argmax protocol, Protocol 1, and the semantic security of the Paillier cryptosystem, the security of this classifier follows trivially, by invoking a modular composition theorem.

Efficiency. Note that the tables P and $\{T_{i,j}\}_{1 \leq i \leq k, 1 \leq j \leq d}$ can be prepared in advance. Hence the cost of constructing the tables can be amortized over many uses. To compute the encrypted probabilities p_i's, the client runs d homomorphic operations (here multiplications) for each i, hence doing kd modular multiplications. Then the parties run a single argmax protocol *i.e.* $k - 1$ comparisons and $O(k)$ homomorphic operations. Thus, compared to non-encrypted computation, the overhead comes only from the use of homomorphic encryption operations instead of plaintext operations. Regarding the number of round trips, these are due to the argmax protocol: $k-1$ runs of the comparison protocol and $k - 1$ additional roundtrips.

VII. Private decision trees

A private decision tree classifier allows the server to traverse a binary decision tree using the client's input x such that the

[3] If the biggest δ_i is 10, the ratio between the smallest and the biggest probability is of order $2^{2^{10}} = 2^{1024}$ ...

server does not learn the input x, and the client does not learn the structure of the tree and the thresholds at each node. A challenge is that, in particular, the client should not learn the path in the tree that corresponds to x – the position of the path in the tree and the length of the path leaks information about the model. The outcome of the classification does not necessarily leak the path in the tree

The idea is to express the decision tree as a polynomial P whose output is the result of the classification, the class predicted for x. Then, the server and the client privately compute inputs to this polynomial based on x and the thresholds w_i. Finally, the server evaluates the polynomial P privately.

A. Polynomial form of a decision tree

Consider that each node of the tree has a boolean variable associated to it. The value of the boolean at a node is 1 if, on input x, one should follow the right branch, and 0 otherwise. For example, denote the boolean variable at the root of the tree by b_1. The value of b_1 is 1 if $x_1 \leq w_1$ (recall Figure 2), and 0 otherwise.

We construct a polynomial P that, on input all these boolean variables and the value of each class at a leaf node, outputs the class predicted for x. The idea is that P is a sum of terms, where each term (say t) corresponds to a path in the tree from root to a leaf node (say c). A term t evaluates to c iff x is classified along that path in T, else it evaluates to zero. Hence, the term corresponding to a path in the tree is naturally the multiplication of the boolean variables on that path and the class at the leaf node. For example, for the tree in Figure 3, P is $P(b_1, b_2, b_3, b_4, c_1, \ldots, c_5) = b_1(b_3 \cdot (b_4 \cdot c_5 + (1 - b_4) \cdot c_4) + (1 - b_3) \cdot c_3) + (1 - b_1)(b_2 \cdot c_2 + (1 - b_2) \cdot c_1)$.

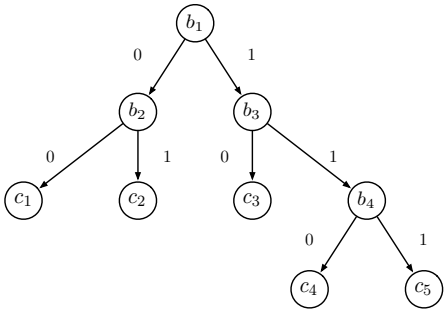
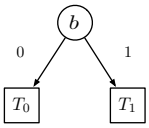


Fig. 3: Decision tree with booleans

We now present $\mathcal{F}$, a recursive procedure for constructing P given a binary decision tree T:

If T consists only of a leaf node with category index c_i, $\mathcal{F}(T) = c_i$.

If T is empty, return $\mathcal{F}(T) = 0$.



Otherwise, T has an internal node using boolean b and T_0 and T_1 are its left and right subtrees. Then $\mathcal{F}(T) = b \cdot \mathcal{F}(T_1) + (1 - b) \cdot \mathcal{F}(T_0)$.

B. Private evaluation of a polynomial

Let us first explain how to compute the values of the boolean variables securely. Let n be the number of nodes in the tree and $\mathsf{nleaves}$ be the number of leaves in the tree. These values must remain unknown to the server because they leak information about x: they are the result of the intermediate computations of the classification criterion. For each boolean variable b_i, the server and the client engage in the comparison protocol to compare w_i and the corresponding attribute of x. As a result, the server obtains $[b_i]$ for $i \in 1 \ldots n$; the server then changes the encryption of these values to FHE using Protocol 2, thus obtaining $[\![b_i]\!]$.

The server evaluates P on $([\![b_1]\!], \ldots, [\![b_n]\!])$ using the homomorphic properties of FHE. In most cases, FHE evaluation is very slow, but we succeed to make it efficient through a combination of techniques we now discuss. To understand these techniques, recall that a typical FHE evaluation happens over a circuit whose gates are modular addition and multiplication. The performance of FHE depends a lot on the depth of multiplications in this circuit.

First, we use a *leveled* FHE scheme: a scheme that supports only an a priori fixed multiplicative depth instead of an arbitrary such depth. As long as this depth is small, such a scheme is much faster than a *full* FHE scheme.

Second, we ensure that the multiplicative depth is very small using a tree-based evaluation. If h_{max} is the maximum height of the decision tree, then P has a term $a_1 \cdot \ldots \cdot a_{h_{\mathsf{max}}}$. If we evaluate this term naïvely with FHE, we multiply these values sequentially. This yields a multiplicative depth of h_{max}, which makes FHE slow for common h_{max} values. Instead, we construct a binary tree over these values and multiply them in pairs based on the structure of this tree. This results in a multiplicative depth of $\log_2 h_{\mathsf{max}}$ (e.g., 4), which makes FHE evaluation significantly more efficient.

Finally, we use $\mathbb{F}_2$ as the plaintext space and SIMD slots for parallelism. FHE schemes are significantly faster when the values encrypted are bits (namely, in $\mathbb{F}_2$); however, P contains classes (e.g., c_1) which are usually more than a bit in length. To enable computing P over F_2, we represent each class in binary. Let $l = \lceil \log_2 k \rceil$ (k is the number of classes) be the number of bits needed to represent a class. We evaluate P l times, once for each of the l bits of a class. Concretely, the j-th evaluation of P takes as input $b_1, \ldots, b_n$ and for each leaf node c_i, its j-th bit c_{ij}. The result is $P(b_1, \ldots, b_n, c_{1j}, c_{2j}, \ldots, c_{\mathsf{nleaves}j})$, which represents the j-th bit of the outcome class. Hence, we need to run the FHE evaluation l times.

To avoid this factor of l, the idea is to use a nice feature of FHE called *SIMD slots* (as described in [50]): these allow encrypting multiple bits in a single ciphertext such that any operation applied to the ciphertext gets applied in parallel to each of the bits. Hence, for each class c_j, the server creates an FHE ciphertext $[\![c_{j0}, \ldots, c_{jl-1}]\!]$. For each node b_i, it creates an FHE ciphertext $[\![b_i, \ldots, b_i]\!]$ by simply repeating the b_i value in each slot. Then, the server runs one FHE evaluation of P over all these ciphertexts and obtains $[\![c_{o0}, \ldots, c_{ol-1}]\!]$ where c_o is the outcome class. Hence, instead of l FHE evaluations, the server runs the evaluation only once. This results in a performance improvement of $\log k$, a factor of 2 and more in our experiments. We were able to apply SIMD slots parallelism

due to the fortunate fact that the same polynomial P had to be computed for each slot.

Finally, evaluating the decision tree is done using $2n$ FHE multiplications and $2n$ FHE additions where n is the number of criteria. The evaluation circuit has multiplication depth $\lceil \log_2(n) + 1 \rceil$.

C. Formal description

Protocol 6 describes the resulting protocol.

Protocol 6 Decision Tree Classifier

Client's (C) Input: $x = (x_1, \ldots, x_n) \in \mathbb{Z}^n$, secret keys $\mathsf{SK}_{QR}, \mathsf{SK}_{FHE}$
Server's (S) Input: The public keys $\mathsf{PK}_{QR}, \mathsf{PK}_{FHE}$, the model as a decision tree, including the n thresholds $\{w_i\}_{i=1}^n$.
Client's Output: The value of the leaf of the decision tree associated with the inputs $b_1, \ldots, b_n$.

1: S produces an n-variate polynomial P as described in section VII-A.
2: S and C interact in the comparison protocol, so that S obtains $[b_i]$ for $i \in [1 \ldots n]$ by comparing w_i to the corresponding attribute of x.
3: Using Protocol 2, S changes the encryption from QR to FHE and obtains $[\![b_1]\!], \ldots, [\![b_n]\!]$.
4: To evaluate P, S encrypts the bits of each category c_i using FHE and SIMD slots, obtaining $[\![c_{i1}, \ldots, c_{il}]\!]$. S uses SIMD slots to compute homomorphically $[\![P(b_1, \ldots, b_n, c_{10}, \ldots, c_{\mathsf{nleaves}0}), \ldots, P(b_1, \ldots, b_n, c_{1l-1}, \ldots, c_{\mathsf{nleaves}l-1})]\!]$. It rerandomizes the resulting ciphertext using FHE's rerandomization function, and sends the result to the client.
5: C decrypts the result as the bit vector $(v_0, \ldots, v_{l-1})$ and outputs $\sum_{i=0}^{l-1} v_i \cdot 2^i$.

Proposition 7.1: Protocol 6 is secure in the honest-but-curious model.

Proof intuition. The proof is in the extended paper [45], but we give some intuition here. During the comparison protocol, the server only learns encrypted bits, so it learns nothing about x. During FHE evaluation, it similarly learns nothing about the input due to the security of FHE. The client does not learn the structure of the tree because the server performs the evaluation of the polynomial. Similarly, the client does not learn the bits at the nodes in the tree because of the security of the comparison protocol.

The interactions between the client and the server are due to the comparisons almost exclusively: the decision tree evaluation does not need any interaction but sending the encrypted result of the evaluation.

VIII. Combining classifiers with AdaBoost

AdaBoost is a technique introduced in [51]. The idea is to combine a set of *weak* classifiers $h_i(x) : \mathbb{R}^d \mapsto \{-1, +1\}$ to obtain a better classifier. The AdaBoost algorithm chooses t scalars $\{\alpha_i\}_{i=1}^t$ and constructs a strong classifier as:

$$H(x) = sign\left(\sum_{i=1}^{t} \alpha_i h_i(x)\right)$$

If each of the $h_i(\cdot)$'s is an instance of a classifier supported by our protocols, then given the scalars α_i, we can easily and securely evaluate $H(x)$ by simply composing our building blocks. First, we run the secure protocols for each of h_i, except that the server keeps the intermediate result, the outcome of $h_i(x)$, encrypted using one of our comparison protocols (Rows 2 or 4 of Table II). Second, if necessary, we convert them to Paillier's encryption scheme with Protocol 2, and combine these intermediate results using Paillier's additive homomorphic property as in the dot product protocol Protocol 3. Finally, we run the comparison over encrypted data algorithm to compare the result so far with zero, so that the client gets the final result.

IX. Implementation

We have implemented the protocols and the classifiers in C++ using GMP[4], Boost, Google's Protocol Buffers[5], and HELib [43] for the FHE implementation.

The code is written in a modular way: all the elementary protocols defined in Section IV can be used as black boxes with minimal developer effort. Thus, writing secure classifiers comes down to invoking the right API calls to the protocols. For example, for the linear classifier, the client simply calls a key exchange protocol to setup the various keys, followed by the dot product protocol, and then the comparison of encrypted data protocol to output the result, as shown in Figure 4.

X. Evaluation

To evaluate our work, we answer the following questions: (i) can our building blocks be used to construct other classifiers in a modular way (Section X-A), (ii) what is the performance overhead of our building blocks (Section X-C), and (iii) what is the performance overhead of our classifiers (Section X-D)?

A. Using our building blocks library

Here we demonstrate that our building blocks library can be used to build other classifiers modularly and that it is a useful contribution by itself. We will construct a multiplexer and a face detector. A face detection algorithm over encrypted data already exists [19], [20], so our construction here is not the first such construction, but it serves as a proof of functionality for our library.

1) Building a multiplexer classifier: A multiplexer is the following generalized comparison function:

$$f_{\alpha,\beta}(a, b) = \begin{cases} \alpha \text{ if } a > b \\ \beta \text{ otherwise} \end{cases}$$

We can express $f_{\alpha,\beta}$ as a linear combination of the bit $d = (a \leq b)$:

$$f_{\alpha,\beta}(d) = d \cdot \beta + (1 - d) \cdot \alpha = \alpha + d \cdot (\beta - \alpha).$$

To implement this classifier privately, we compute $[\![d]\!]$ by comparing a and b, keeping the result encrypted with QR, and then changing the encryption scheme (*cf.* Section IV-C) to Paillier.

[4] http://gmplib.org/
[5] https://code.google.com/p/protobuf/

```
bool Linear_Classifier_Client::run()
{
  exchange_keys();

  // values_ is a vector of integers
  // compute the dot product
  mpz_class v = compute_dot_product(values_);
  mpz_class w = 1; // encryption of 0

  // compare the dot product with 0
  return enc_comparison(v, w, bit_size_, false);
}
```

```
void Linear_Classifier_Server_session::
        run_session()
{
  exchange_keys();

  // enc_model_ is the encrypted model vector
  // compute the dot product
  help_compute_dot_product(enc_model_, true);

  // help the client to get
  // the sign of the dot product
  help_enc_comparison(bit_size_, false);
}
```

Fig. 4: Implementation example: a linear classifier

Bit size	A Computation	B Computation	Total Time	Communication	Interactions
10	14.11 ms	8.39 ms	105.5 ms	4.60 kB	3
20	18.29 ms	14.1 ms	117.5 ms	8.82 kB	3
32	22.9 ms	18.8 ms	122.6 ms	13.89 kB	3
64	34.7 ms	32.6 ms	134.5 ms	27.38 kB	3

TABLE III: Comparison with unencrypted input protocols evaluation.

Protocol	Bit size	Computation		Total Time	Communication	Interactions
		Party A	Party B			
Comparison	64	45.34 ms	43.78 ms	190.9 ms	27.91 kB	6
Reversed Comp.	64	48.78 ms	42.49 ms	195.7 ms	27.91 kB	6

TABLE IV: Comparison with encrypted input protocols evaluation.

Party A Computation	Party B Computation	Total Time	Communication	Interactions
30.80 ms	255.3 ms	360.7 ms	420.1 kB	2

TABLE V: Change encryption scheme protocol evaluation.

Then, using Paillier's homomorphism and knowledge of α and β, we can compute an encryption of $f_{\alpha,\beta}(d)$:

$$[\![f_{\alpha,\beta}(d)]\!] = [\![\alpha]\!] \cdot [\![d]\!]^{\beta-\alpha}.$$

2) Viola and Jones face detection: The Viola and Jones face detection algorithm [52] is a particular case of an AdaBoost classifier. Denote by X an image represented as an integer vector and x a particular detection window (a subset of X's coefficients). The *strong* classifier H for this particular detection window is

$$H(x) = sign\left(\sum_{i=1}^{t} \alpha_i h_i(x)\right)$$

where the h_t are weak classifiers of the form $h_i(x) = \text{sign}\left(\langle x, y_i\rangle - \theta_i\right)$.

In our setting, Alice owns the image and Bob the classifier (*e.g.* the vectors $\{y_i\}$ and the scalars $\{\theta_i\}$ and $\{\alpha_i\}$). Neither of them wants to disclose their input to the other party. Thanks to our building blocks, Alice can run Bob's classifier on her image without her learning anything about the parameters and Bob learning any information about her image.

The weak classifiers can be seen as multiplexers; with the above notation, we have $h_t(x) = f_{1,-1}(\langle x, y_t\rangle - \theta_t)$.

Using the elements of Section X-A1, we can easily compute the encrypted evaluation of every one of these weak classifiers under Paillier, and then, as described in Section VIII, compute the encryption of $H(x)$.

B. Performance evaluation setup

Our performance evaluations were run using two desktop computers each with identical configuration: two Intel Core i7 (64 bit) processors for a total 4 cores running at 2.66 GHz and 8 GB RAM. Since the machines were on the same network, we inflated the roundtrip time for a packet to be 40 ms to mimic real network latency. We used 1024-bit cryptographic keys, and chose the statistical security parameter λ to be 100. When using HELib, we use 80 bits of security, which corresponds to a 1024-bit asymmetric key.

C. Building blocks performance

We examine performance in terms of computation time at the client and server, communication bandwidth, and also number of interactions (round trips). We can see that all these protocols are efficient, with a runtime on the order of milliseconds.

Data set	Model size	Computation		Time per protocol		Total running time	Comm.	Interactions
		Client	Server	Compare	Dot product			
Breast cancer (2)	30	46.4 ms	43.8 ms	194 ms	9.67 ms	204 ms	35.84 kB	7
Credit (3)	47	55.5 ms	43.8 ms	194 ms	23.6 ms	217 ms	40.19 kB	7

(a) Linear Classifier. Time per protocol includes communication.

Data set	Specs.		Computation		Time per protocol		Total running time	Comm.	Interactions
	C	F	Client	Server	Prob. Comp.	Argmax			
Breast Cancer (1)	2	9	150 ms	104 ms	82.9 ms	396 ms	479 ms	72.47 kB	14
Nursery (5)	5	9	537 ms	368 ms	82.8 ms	1332 ms	1415 ms	150.7 kB	42
Audiology (4)	24	70	1652 ms	1664 ms	431 ms	3379 ms	3810 ms	1911 kB	166

(b) Naïve Bayes Classifier. C is the number of classes and F is the number of features. The Prob. Comp. column corresponds to the computation of the probabilities $p(c_i|x)$ (*cf.* Section VI). Time per protocol includes communication.

Data set	Tree Specs.		Computation		Time per protocol		FHE		Comm.	Interactions
	N	D	Client	Server	Compare	ES Change	Eval.	Decrypt		
Nursery (5)	4	4	1579 ms	798 ms	446 ms	1639 ms	239 ms	33.51 ms	2639 kB	30
ECG (6)	6	4	2297 ms	1723 ms	1410 ms	7406 ms	899 ms	35.1 ms	3555 kB	44

(c) Decision Tree Classifier. ES change indicates the time to run the protocol for changing encryption schemes. N is the number of nodes of the tree and D is its depth. Time per protocol includes communication.

TABLE VI: Classifiers evaluation.

1) Comparison protocols:

Comparison with unencrypted input. Table III gives the running time of the comparison protocol with unencrypted input for various input size.

Comparison with encrypted input. Table IV presents the performance of the comparison with encrypted inputs protocols.

2) argmax*:* Figure 5 presents the running times and the communication overhead of the argmax of encrypted data protocol (*cf.* Section IV-B). The input integers were 64 bit integers.

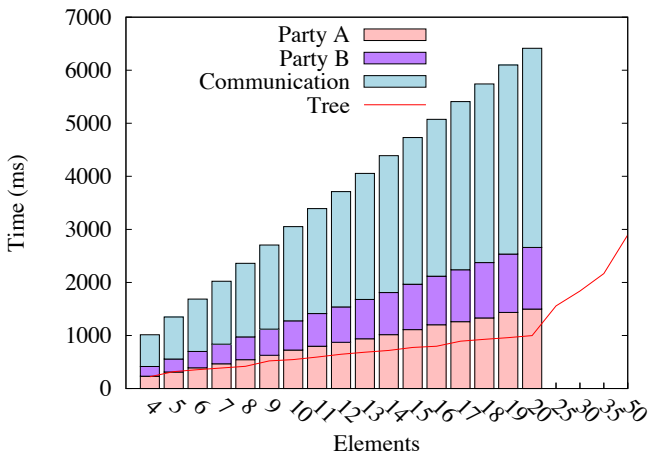


Fig. 5: Argmax of encrypted data protocol evaluation. The bars represent the execution of the protocol when the comparisons are executed one after each other, linearly. The line represents the execution when comparisons are executed in parallel, tree-wise.

3) Consequences of the latency on performances: It is worth noticing that for most blocks, most of the running time is spend communicating: the network's latency has a huge influence on the performances of the protocols (running time almost linear in the latency for some protocols). To improve the performances of a classifier implemented with our blocks, we might want to run several instances of some building blocks in parallel. This is actually what we did with the tree-based implementation of the argmax protocol, greatly improving the performances of the protocol (*cf.* Figure 5).

D. Classifier performance

Here we evaluate each of the classifiers described in Sections V–VII. The models are trained non-privately using `scikit-learn`[6]. We used the following datasets from the UCI machine learning repository [53]:

1) the Wisconsin Diagnostic Breast Cancer data set,
2) the Wisconsin Breast Cancer (Original) data set, a simplified version of the previous dataset,
3) Credit Approval data set,
4) Audiology (Standardized) data set,
5) Nursery data set, and
6) ECG (electrocardiogram) classification data from Barni *et al.* [27]

These data sets are scenarios when we want to ensure privacy of the server's model and client's input.

Based on the suitability of each classifier, we used data sets 2 and 3 to test the hyperplane decision classifier, sets 1, 4 and 5 for the Naïve Bayes classifier, and sets 5 and 6 for the decision tree classifier.

[6]http://scikit-learn.org

Table VI shows the performance results. Our classifiers run in at most a few seconds, which we believe to be practical for sensitive applications. Note that even if the datasets become very large, the size of the model stays the same – the dataset size only affects the training phase which happens on unencrypted data before one uses our classifiers. Hence, the cost of our classification will be the same even for very large data sets.

For the decision tree classifier, we compared our construction to Barni *et al.* [27] on the ECG dataset (by turning their branching program into a decision tree). Their performance is 2609 ms[7] for the client and 6260 ms for the server with communication cost of 112.2KB. Even though their evaluation does not consider the communication delays, we are still more than three times as fast for the server and faster for the client.

E. Comparison to generic two-party tools

A set of generic secure two- or multi-party computation tools have been developed, such as TASTY [6] and Fairplay [7], [8]. These support general functions, which include our classifiers.

However, they are prohibitively slow for our specific setting. Our efficiency comes from specializing to classification functionality. To demonstrate their performance, we attempted to evaluate the Naïve Bayes classifier with these. We used FairplayMP to generate the circuit for this classifier and then TASTY to run the private computation on the circuit thus obtained. We tried to run the smallest Naïve Bayes instance, the Nursery dataset from our evaluation, which has only 3 possible values for each feature, but we ran out of memory during the circuit generation phase on a powerful machine with 256GB of RAM.

Hence, we had to reduce the classification problem to only 3 classes (versus 5). Then, the circuit generation took more than 2 hours with FairplayMP, and the time to run the classification with TASTY was 413196 msec (with no network delay), which is ≈ 500 times slower than our performance (on the non-reduced classification problem with 5 classes). Thus, our specialized protocols improve performance by orders of magnitude.

XI. Conclusion

In this paper, we constructed three major privacy-preserving classifiers as well as provided a library of building blocks that enables constructing other classifiers. We demonstrated the efficiency of our classifiers and library on real datasets.

Acknowledgment

We thank Thijs Veugen, Thomas Schneider, and the anonymous reviewers for their helpful comments.

References

[1] J. Wiens, J. Guttag, and E. Horvitz, "Learning evolving patient risk processes for c. diff colonization," in *ICML*, 2012.

[2] A. Singh and J. Guttag, "Cardiovascular risk stratification using non-symmetric entropy-based classification trees," in *NIPS workshop on personalized medicine*, 2011.

[3] A. Singh and J. Guttag, "Leveraging hierarchical structure in diagnostic codes for predicting incident heart failure," in *ICML workshop on role of machine learning in transforming healthcare*, 2013.

[4] A. C. Yao, "Protocols for secure computations," in *FOCS*, 1982, pp. 160–164.

[5] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *STOC*, 1987, pp. 218–229.

[6] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, "Tasty: Tool for automating secure two-party computations," in *CCS*, 2010, pp. 451–462.

[7] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay-secure two-party computation system." in *USENIX Security Symposium*, 2004, pp. 287–302.

[8] A. Ben-David, N. Nisan, and B. Pinkas, "Fairplaymp: A system for secure multi-party computation," in *CCS*, 2008, pp. 17–21.

[9] Y. Lindell and B. Pinkas, "Privacy preserving data mining," in *Advances in Cryptology (CRYPTO)*, 2000, pp. 36–54.

[10] W. Du, Y. S. Han, and S. Chen, "Privacy-preserving multivariate statistical analysis: Linear regression and classification," in *Proceedings of the 4th SIAM International Conference on Data Mining*, vol. 233, 2004.

[11] R. Wright and Z. Yang, "Privacy-preserving bayesian network structure computation on distributed heterogeneous data," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, pp. 713–718.

[12] Z. Y. S. Zhong and R. N. Wright, "Privacy-preserving classification of customer data without loss of accuracy," in *SIAM International Conference on Data Mining (SDM)*, 2005.

[13] A. Blum, C. Dwork, F. McSherry, and K. Nissim, "Practical privacy: the sulq framework," in *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2005, pp. 128–138.

[14] J. Vaidya, M. Kantarcıoğlu, and C. Clifton, "Privacy-preserving naive bayes classification," *The International Journal on Very Large Data Bases*, vol. 17, no. 4, pp. 879–898, 2008.

[15] T. Graepel, K. Lauter, and M. Naehrig, "ML confidential: Machine learning on encrypted data," in *Information Security and Cryptology (ICISC)*, 2012, pp. 1–21.

[16] J. W. Bos, K. Lauter, and M. Naehrig, "Private predictive analysis on encrypted medical data," in *Microsoft Tech Report 200652*, 2013.

[17] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft, "Privacy-preserving face recognition," in *Privacy Enhancing Technologies*, 2009, pp. 235–253.

[18] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, "Efficient privacy-preserving face recognition," in *Information, Security and Cryptology (ICISC)*, 2009, pp. 229–244.

[19] S. Avidan and M. Butman, "Blind vision," in *Computer Vision–ECCV 2006*, 2006, pp. 1–13.

[20] S. Avidan and M. Butman, "Efficient methods for privacy preserving face detection," in *Advances in Neural Information Processing Systems*, 2007, p. 57.

[21] R. Canetti, "Security and composition of multi-party cryptographic protocols," *JOURNAL OF CRYPTOLOGY*, vol. 13, p. 2000, 1998.

[22] Y. Lindell and B. Pinkas, "An efficient protocol for secure two-party computation in the presence of malicious adversaries," in *Advances in Cryptology (EUROCRYPT)*, 2007, vol. 4515, pp. 52–78.

[23] Y. Ishai, M. Prabhakaran, and A. Sahai, "Founding cryptography on oblivious transfer – efficiently," in *Advances in Cryptology – CRYPTO 2008*, 2008, vol. 5157, pp. 572–591.

[24] Y. Lindell and B. Pinkas, "A proof of security of Yao's protocol for two-party computation," *J. Cryptol.*, vol. 22, pp. 161–188, April 2009.

[25] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate, "Differentially private empirical risk minimization," *J. Mach. Learn. Res.*, vol. 12, 2011.

[26] S. Laur, H. Lipmaa, and T. Mielikäinen, "Cryptographically private support vector machines," in *Proceedings of the 12th ACM SIGKDD*

[7] In Barni *et al.* [27], the evaluation was run over two 3GHz computers directly connected via Gigabit Ethernet. We scaled the given results by $\frac{3}{2.3}$ to get a better comparison basis.

international conference on Knowledge discovery and data mining. ACM, 2006, pp. 618–624.

[27] M. Barni, P. Failla, V. Kolesnikov, R. Lazzeretti, A.-R. Sadeghi, and T. Schneider, "Secure evaluation of private linear branching programs with medical applications," in *Computer Security (ESORICS)*, 2009, pp. 424–439.

[28] M. Barni, P. Failla, R. Lazzeretti, A. Paus, A.-R. Sadeghi, T. Schneider, and V. Kolesnikov, "Efficient privacy-preserving classification of ecg signals," in *Information Forensics and Security, 2009. WIFS 2009. First IEEE International Workshop on*, 2009, pp. 91–95.

[29] M. Barni, P. Failla, R. Lazzeretti, A.-R. Sadeghi, and T. Schneider, "Privacy-preserving ECG classification with branching programs and neural networks," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 6, no. 2, pp. 452–468, June 2011.

[30] I. Damgård, M. Geisler, and M. Krøigaard, "Efficient and secure comparison for on-line auctions," in *Information Security and Privacy*, 2007, pp. 416–430.

[31] I. Damgard, M. Geisler, and M. Kroigard, "A correction to efficient and secure comparison for on-line auctions," vol. 1, no. 4, pp. 323–324, 2009.

[32] T. Veugen, "Comparing encrypted data," http://msp.ewi.tudelft.nl/sites/default/files/Comparingencrypteddata.pdf, 2011.

[33] H.-Y. Lin and W.-G. Tzeng, "An efficient solution to the millionaires' problem based on homomorphic encryption," in *Applied Cryptography and Network Security*, 2005, pp. 456–466.

[34] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider, "How to combine homomorphic encryption and garbled circuits - improved circuits and computing the minimum distance efficiently," in *1st International Workshop on Signal Processing in the EncryptEd Domain (SPEED'09)*, 2009.

[35] M. J. Atallah and W. Du, "Secure multi-party computational geometry," in *Algorithms and Data Structures*, 2001, pp. 165–179.

[36] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikäinen, "On private scalar product computation for privacy-preserving data mining," in *Information Security and Cryptology (ICISC)*, 2004, pp. 104–120.

[37] E. Kiltz, "Unconditionally secure constant round multi-party computation for equality, comparison, bits and exponentiation." *IACR Cryptology ePrint Archive*, p. 66, 2005.

[38] C. M. Bishop and N. M. Nasrabadi, "Pattern recognition and machine learning," in *Journal of Electronic Imaging*, vol. 1, 2006.

[39] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *NIPS*, 2007.

[40] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC*, 2009, pp. 169–178.

[41] S. Goldwasser and S. Micali, "Probabilistic encryption and how to play mental poker keeping secret all partial information," in *STOC*. ACM, 1982, pp. 365–377.

[42] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*, 1999, pp. 223–238.

[43] S. Halevi, "Helib - an implementation of homomorphic encryption," https://github.com/shaih/HElib, 2013. [Online]. Available: https://github.com/shaih/HElib

[44] O. Goldreich, *Foundations of Cryptography - Basic Applications*. Cambridge University Press, 2004.

[45] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," in *Crypto ePrint Archive*, 2014, https://eprint.iacr.org/2014/331.pdf.

[46] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher," in *IEEE SP*, 2013, pp. 478–492.

[47] M. Naor and B. Pinkas, "Efficient oblivious transfer protocols," in *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, 2001, pp. 448–457.

[48] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider, "A systematic approach to practically efficient general two-party secure function evaluation protocols and their modular design," in *Journal of Computer Security*, 2013.

[49] S. Tschiatschek, P. Reinprecht, M. Mücke, and F. Pernkopf, "Bayesian network classifiers with reduced precision parameters," in *Machine Learning and Knowledge Discovery in Databases*, 2012, pp. 74–89.

[50] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," Cryptology ePrint Archive, Report 2011/133, 2011.

[51] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, no. 1, pp. 119–139, 1997.

[52] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *IEEE Computer Vision and Pattern Recognition (CVPR)*, vol. 1, 2001, pp. I–511.

[53] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

See our full paper [45] for details and proofs.