Privacy-Preserving Collaborative Prediction using Random Forests

Irene Giacomelli, Ph.d.[1], Somesh Jha, Ph.d.[2], Ross Kleiman, M.S.[2], David Page, Ph.d.[2], Kyonghwan Yoon, M.S.[2]
[1]ISI Foundation, Turin, Italy; [2]University of Wisconsin-Madison, Madison, WI, USA

Abstract *We study the problem of privacy-preserving machine learning (PPML) for ensemble methods, focusing our effort on random forests. In collaborative analysis, PPML attempts to solve the conflict between the need for data sharing and privacy. This is especially important in privacy sensitive applications such as learning predictive models for clinical decision support from EHR data from different clinics, where each clinic has a responsibility for its patients' privacy. We propose a new approach for ensemble methods: each entity learns a model, from its own data, and then when a client asks the prediction for a new private instance, the answers from all the locally trained models are used to compute the prediction in such a way that no extra information is revealed. We implement this approach for random forests and we demonstrate its high efficiency and potential accuracy benefit via experiments on real-world datasets, including actual EHR data.*

Introduction

Nowadays, machine learning (ML) models are deployed for prediction in many privacy sensitive scenarios (*e.g.*, personalized medicine or genome-based prediction). A classic example is disease diagnosis, where a model predicts the risk of a disease for a patient by simply looking at his/her health records. Such models are constructed by applying learning methods from the literature to specific data collected for this task (the training data,—instances for which the outcome is known—in the preceding example these are health records of patients monitored for the specific disease). Prior experience in ML model training suggests that having access to a large and diverse training dataset is a key ingredient in order to enhance the efficacy of the learned model[1]. A training dataset with these feature can be created by merging several silos of data collected locally by different entities. Therefore, sharing and merging data can result in mutual gain to the entities involved in the process and, finally, to the broader community. For example, hospitals and clinics located in different cities across a country can locally collect clinical data that is then used to run a collaborative analysis with the potential to improve the health-care system of the entire country. However, in privacy sensitive scenarios, sharing data is hindered by significant privacy concerns and legal regulations (*e.g.*, HIPAA laws in the United States and GDPR for the European Union). In the example described before, sharing clinical data directly competes with the need for healthcare providers to protect the privacy of each patient and respect current privacy policies and laws.

Based on the preceding discussion, we often face the following dilemma: share data to improve accuracy or keep data and information secret to protect privacy? Notice that de-identification cannot resolve this standoff: several works[2,3] demonstrated that sharing de-identified data is not a secure approach since in many contexts the potential for re-identification is high. More sophisticated anonymization criteria (*e.g.*, k-anonimity, l-diversity, t-closeness, etc.) were proposed by the database community. While arguably better than de-identification, all such "syntactic" approaches work only in presence of assumptions regarding the adversary's background knowledge. Conversely, cryptographic tools can guarantee perfect privacy of shared data in more general situations. For example, a number of *privacy-preserving training* algorithms have been proposed since the seminal paper of Lindell and Pinkas[4] introduced this concept in 2000. These algorithms use advanced cryptographic tools in order to allow different parties to run known learning algorithms on the merge of local datasets without revealing the actual data. This approach guarantees privacy at the price of high communication and computation overhead. Once the model is learned, we face another privacy problem: using the model to compute a prediction for new instances while both the model and the instances data are sensitive information privately held by different parties. This problem can be solved using again cryptographic tools, and an algorithm designed for this task is called *privacy-preserving scoring*. In conclusion, a solution that uses the current tools to guarantee privacy at all levels (*e.g.*, for the data providers, model providers, model users) deploys two privacy-preserving systems, a first one for training and a second one for scoring.

In this work, we notice that for *ensemble methods*, for which the learned model is formed by a set of more basic models and the prediction for a new instance is computed by blending together the basic predictions, there can be an easier and

more efficient solution that needs only one system; we refer to this solution as the *"locally learn then merge"* approach. Each entity with a local data silo (*i.e.*, providers) can train its own local model M_i, and then the prediction given by these models can be merged at the moment when the scoring for a new (eventually private) instance is computed. That is, a user with input $\mathbf{x}$ gets $y = \Phi(M_1(\mathbf{x}), \ldots, M_t(\mathbf{x}))$ for a specific merging function Φ. Here $M_i(\mathbf{x})$ indicates the prediction of the local model M_i for the instance $\mathbf{x}$. In this approach, privacy concerns coming from data sharing in the training phase are not present since, clearly, local training does not require data sharing. Moreover, there is no overhead for the training phase (this is run as in the standard ML scenario), while the final prediction can benefit from merging the local predictions via the function Φ. On the other hand, accuracy loss (with respect to a model learned from the merged data) and information leakage can happen during the merging/scoring phase. In particular, a challenge remains with this simple and elegant approach to collaborative ML: if we want to guarantee model and user's input privacy (*i.e.*, the user learns y and no other information on the models M_i, the providers learn nothing about $\mathbf{x}$), then even after the training phase each provider must maintain its own on-line server and communicate with the client and the other providers each time a new prediction is requested. Since in a real-world scenario (*i.e.*, healthcare environment), this requirement can be cumbersome to implement, we design our system in the *cloud model*, where the computation of the prediction from the local models is outsourced to a central server and providers are not required to be on-line during the scoring process (Fig. 1). Since we do not require the server to be trusted, each model M_i is sent to the server in encrypted form (*i.e.*, $[M_i]$). Once this is done, the providers (*e.g.*, clinics) can go off-line and when a user (*e.g.*, medical research institution) requires access to the models to compute predictions for new data, the server communicates with it and computes the answer from the encrypted models.

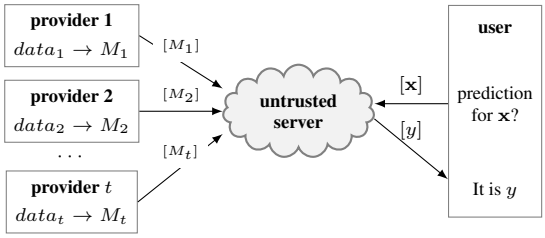


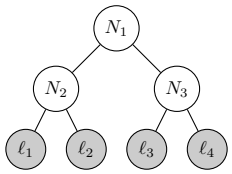
Figure 1: Overview of the new "locally learn then merge" approach in the cloud model. The providers upload the encrypted models to the server and then go off-line. The server is on-line to answer to the prediction requests of the user.

In this work, we specify and evaluate the "locally learn then merge" paradigm in the cloud model for a widely-used ensemble method: random forests. *Random forests*[5] are among the most accurate and widely-used ML ensemble models and are employed across a variety of challenging tasks, including predictive modeling from clinical data[6], that are characterized by high dimension and variable interactions, or other non-linearities in the target concept. A random forest is a collection of simple decision trees. By the use of different trees, a random forest can capture variable interactions without the need for the learner to know or guess all relevant interactions ahead of time in order to represent them with new variables (interaction terms); by their ensemble nature, random forests effectively reduce the over-fitting often observed with ordinary decision tree learning. A less-recognized advantage of random forests is that they can be learned in a distributed manner. In this circumstance, separate random forests can easily be learned locally by entities with data silos, and then the prediction for a new instance is computed as the arithmetic mean of the predictions of all the trees in the locally trained random forests (*i.e.*, the merging function Φ is the arithmetic mean). We design a system implementing this approach for random forest using standard and fast cryptographic primitives. While our scheme is efficient even for forests of many trees, not surprisingly its run-time and communication complexity grow exponentially with maximum tree depth in a forest. Therefore we also provide empirical evidence that across a variety of data sets and tasks, increasing the number of trees can effectively make up for any accuracy or AUC lost by incorporating a stringent limit on tree depth, such as 8 or even 6.

Related Work: There is an extensive research that propose *privacy-preserving training*[7] protocols and few of them focus on training decision tree. After that Lindell and Pinkas[4] in 2000 presented a system for two data-providers, Xiao et al.[8] and Samet and Miri[9] considered the case of more than two providers. While the former works consider horizontally partitioned data, another line of work[10,11] assumes data that are vertically partitioned among two or more data holders. Lastly, in 2014 Vaidya et al.[12] proposed a method to learn and score *random trees* in a privacy preserving manner. Like our approach, their approach requires encryption-based collaboration to make predictions. Unlike our approach, their approach also requires interaction and collaboration at training time. One party proposes a random tree structure, and all parties must contribute information to the distributions at the leaf nodes. In our approach, learning

is completely independent for each party, and hence training is much faster. An advantage of Vaidya et al. is the ability to also address vertically partitioned data. *Privacy-preserving scoring* protocols for decision trees have been designed using different cryptographic tools (*e.g.*, levelled homomorphic encryption[13], LHE[14], secret-sharing[15], OT-channels[16]). In 2017, Backes et al.[17] improved and extended to random forests the algorithm presented by Brickell et al.[18] Another line of research focuses on constructing *differentially private decision trees*, see for example the work of Jagannathan et al.[19] and Rana et al.[20] Our approach is orthogonal to differential privacy since we consider a different threat model.

Methods: decision trees (DTs) and random forests (RFs)



$P_{2,1}(x_1, x_2) = (x_1 - 1)(x_2 - 1)$
$P_{2,2}(x_1, x_2) = (x_1 - 1)(x_2 + 1)$
$P_{2,3}(x_1, x_3) = (x_1 + 1)(x_3 - 1)$
$P_{2,4}(x_1, x_3) = (x_1 + 1)(x_3 + 1)$

Figure 2: Polynomial representation of a depth 2 complete DT.

Decision trees (DTs) are a nonparametric ML model used for both classification and regression problems. While there are a myriad of algorithms for constructing DTs, we focus here on describing the model representation of the scoring procedure. A decision tree (DT), T, can be viewed as mapping a column vector $\mathbf{x} = (\mathbf{x}[1], \ldots, \mathbf{x}[n])^\top$ of features to a prediction value y. In practice, we assume that T is represented as a directed acyclic graph with two types of nodes: *splitting nodes* which have children, and *leaf nodes* which have no children. Moreover, T has a single root node, which is also a splitting node, that has no parents. For an input $\mathbf{x} \in \mathbb{R}^n$, we traverse the tree T starting from the root and reach a leaf. Each splitting node N_i is defined by a pair (j_i, t_i) where j_i is an index in $\{1, \ldots, n\}$ and $t_i \in \mathbb{R}$ is a threshold value. In the root-leaf path, at node i we take the right branch if $\mathbf{x}[j_i] \geq t_i$. Otherwise, we take the left one. Thus, each splitting node N_i is associated with the function $N_i(\mathbf{x}) = \mathbf{e}_{j_i}^\top \cdot \mathbf{x} - t_i$ and the value $n_i = \text{sign}(N_i(\mathbf{x}))$ (where $\cdot$ is the standard row-by-column multiplication). Here the vector $\mathbf{e}_i$ is the column vector in $\mathbb{R}^n$ with all zeros except for a 1 in position i and $\mathbf{e}_i^\top$ is its transpose. Moreover, if $x \in \mathbb{R}$, then $\text{sign}(x) = 1$ if $x \geq 0$ and $\text{sign}(x) = -1$ otherwise. In this way we traverse the tree and we reach a leaf node. The i-th leaf node is associated with the label ℓ_i, which is defined to be the prediction of the query $\mathbf{x}$ that reaches the i-th leaf (*i.e.*, $y = T(\mathbf{x}) = \ell_i$). The format of the labels $\{\ell_i\}_i$ depends on the specific ML problem (regression, multiclass classification or binary classification). In this work, we assume $\ell_i \in [0, 1]$ representing the probability of $\mathbf{x}$ being classified as $+$ in a binary classification problems with labels $\{+, -\}$. The *depth* of a tree is the maximum number of splitting nodes visited before reaching a leaf. In general, DTs need not be binary or complete. However, all DTs can be transformed into a complete binary tree by increasing the depth of the tree and introducing "dummy" splitting nodes. Without loss of generality, here we only consider complete binary DTs. A complete binary tree of depth d has 2^d leaves and $2^d - 1$ splitting nodes. *Random forests* (RFs), proposed by Leo Breiman[5], are an ensemble learning algorithm that are based on DTs. An ensemble learner incorporates the predictions of multiple models to yield a final consensus prediction. More precisely, a random forest RF consists of m trees, $T_1, \ldots, T_m$, and scoring RF on input $\mathbf{x}$ means computing $y = \frac{1}{m}\sum_{i=1}^{m} T_i(\mathbf{x})$. Let d be the maximum of the depths of the trees in RF, we refer to d and m as the *hyperparameters* of the forest.

Polynomial representation: We can represent a tree using polynomials. Let T be a complete binary tree of depth d, then we associate each leaf with the product of d binomials of the form $(x_i - 1)$ or $(x_i + 1)$ using the following rule: in the root-leaf path, if at the node N_i left turn is taken, choose $(x_i - 1)$ otherwise choose $(x_i + 1)$. We indicate with $P_{d,i}$ the polynomial of degree d corresponding to the i-th leaf. Notice that $P_{d,i}$ contains only d variables, out of the $2^d - 1$ total possible variables (one for each splitting node). We call $\mathcal{I}_{d,i}$ the set of indices of the variables that appears in $P_{d,i}$ and we write $P_{d,i}((x_j)_{j \in \mathcal{I}_i})$ to indicate this; in Fig. 2, $\mathcal{I}_{2,1} = \mathcal{I}_{2,2} = \{1, 2\}$ and $\mathcal{I}_{2,3} = \mathcal{I}_{2,4} = \{1, 3\}$. Now $T(\mathbf{x})$ can be computed by evaluating the polynomials $\{P_{d,i}((x_j)_{j \in \mathcal{I}_i})\}_{i=1,\ldots,2^d}$ on the values $\{n_j\}_{j=1,\ldots,2^d-1}$. Indeed, if i^* is the unique value for the index i for which $P_{d,i}((n_j)_{j \in \mathcal{I}_i}) \neq 0$, then $T(\mathbf{x}) = \ell_{i^*}$.

Methods: cryptographic tools

A *linearly-homomorphic encryption* (LHE) scheme is defined by three algorithms: The key-generation algorithm Gen takes as input a security parameter and outputs the pair of secret and public key, $(\mathbf{sk}, \mathbf{pk})$. The encryption algorithm Enc is a randomized algorithm that takes $\mathbf{pk}$ and an input $\mathbf{x}$, and outputs a ciphertext, $c \leftarrow \mathsf{Enc}_{\mathbf{pk}}(\mathbf{x})$. The decryption algorithm Dec is a deterministic function that takes $\mathbf{sk}$ and c, and recovers the original input $\mathbf{x}$ with probability 1

over Enc's random choice. The standard security property (semantic security) states that it is infeasible to gain extra information about an input when given only its ciphertext c and the public key. Moreover, we have the homomorphic property: informally, linear functions of encrypted data can be computed without decrypting (*e.g.*, from $\mathsf{Enc}_{\mathbf{pk}}(\mathbf{x}_1)$ and $\mathsf{Enc}_{\mathbf{pk}}(\mathbf{x}_2)$ we can compute $\mathsf{Enc}_{\mathbf{pk}}(\mathbf{x}_1+\mathbf{x}_2)$ without knowing $\mathbf{x}_1$ and $\mathbf{x}_2$). Efficient instantiations of this primitive are known[21]. In the design of the privacy-preserving system presented later on in this work, we will deploy the *secure comparison protocol* Π_{SC}. The latter[22] is a modification of the protocol presented by Kerschbaum and Terzidis[23] and has the following structure: party 1 has an encryption of an integer a, while party 2 knows the corresponding secret key. They run the protocol and the output is a multiplicative sharing of the sign of a and no extra information about a. In particular, party 1 receives a share $\alpha \in \{-1,+1\}$ and party 2 receives a share $\beta \in \{-1,+1\}$ such that $\alpha\beta = \text{sign}(a)$ and the knowledge of only one share gives no information on $\text{sign}(a)$.

Results: the proposed system

In this section we describe our system, where the prediction for a new instance is computed using the RFs trained by different and mutually distrustful parties on their local data silos. We start by describing the role of the parties involved and the security model.

Providers: There are t providers, the k-th one, P_k, has a forest $RF_k = \{T_1^k, \ldots, T_{m_k}^k\}$ with m_k DTs; we assume that the forest hyperparameters (m_k and maximum tree depth d_k) are public values, while the description of the trees is the secret input of P_k to the system. The providers have no output.

Server: The server has no input and no output; it is not trusted to handle private data neither proprietary models. Its function is providing reliable software and hardware to store encrypted version of the models RF_k and handling prediction request from the user in real-time.

User: Its secret input is an instance $\mathbf{x} \in \mathbb{R}^n$ and the output is the prediction for $\mathbf{x}$ according to all the trees T_j^k (n is public); more precisely, the user's output from the system is $y = \frac{1}{m}\sum_{k=1}^{t}\sum_{j=1}^{m_k} T_j^k(\mathbf{x})$ with $m = m_1 + \cdots + m_k$.

We assume that all the parties involved are honest-but-curious (*i.e.*, they always follow the specifications of the protocol but try to learn extra information about other parties secret input from the messages received during the execution of the protocol) and non-colluding (*e.g.*, in real world applications, physical restrictions or economic incentives can be used to assure that the server has no interest in colluding with another party). Using the cryptographic tools described before and other standard tools, we design a system where only the user gets to know y and it gets no other information about the private models held by the providers. Moreover, the providers and the server gain no information about the input $\mathbf{x}$. The system we present has two phases: an off-line phase, during which each provider uploads an encrypted form of its forest to the server, and an on-line phase, during which the prediction for a specific input $\mathbf{x}$ is computed by the server and the user. Notice that the off-line phase is independent of the actual input of the user and needs to be executed only once (*i.e.*, when the providers join the system). After that, the providers can leave the system and the server will manage each prediction request. In particular, for each request, a new on-line phase is executed by the server together with the user making the request (it is possible to have more than one user requesting predictions). Each phase is described below:

Off-line Phase: The goal of this phase is to transmit all the trees to the server, but in encrypted form. That is, the server will know the public hyperparameters of each locally learned forest but have no knowledge about the specific structure of the trees in the forests (*i.e.*, it does not know the indices i_j, the thresholds t_i, or the leaf values ℓ_i). This is achieved by having each provider execute a new *model-encryption procedure* we design[22]. Using this, the P_k encrypts the thresholds and leaf values and hides the vectors $\mathbf{e}_{j_i}$ using a standard PRF (pseudorandom function); then it sends the encrypted forest to the server; after this P_k can leave the system. We indicate the encrypted forest, which is a collection of encrypted trees, with the notation $\{[\,T_j^k\,]\}_{j=1\ldots,m_k}$.

On-line Phase: For each prediction request, this phase is executed. A user with input $\mathbf{x}$ joins the system sending $\mathsf{Enc}_{\mathbf{pk}}(\mathbf{x})$ to the server. Now, the user and the server run the *tree evaluation protocol* Π_{TE} for each encrypted tree $[\,T_j^k\,]$ of the forests that were uploaded to the server. This protocol returns an additive sharing of $T_j^k(\mathbf{x})$ (*i.e.*, the server gets the share $r_j^k \in [0,1]$ and the user gets the share $s_j^k \in [0,1]$ such that $T_j^k(\mathbf{x}) = s_j^k + r_j^k$ and the knowledge of only one share does not reveal $T_j^k(\mathbf{x})$). Finally, the server sends the sum $r = \sum_{k=1}^{t}\sum_{j=1}^{m_k} r_j^k$ of its shares (one for

each tree) to the user, which computes y as $(s+r)/m$, where $s = \sum_{k=1}^{t} \sum_{j=1}^{m_k} s_j^k$ is the sum of the user's shares. The security of our system against a honest-but-curious server follows from the security of the encryption scheme: all the messages received by the server are ciphertexts. Moreover, the user does not learn any extra information about the local models since it does not see the individual predictions (*i.e.*, for each tree the user only see the share s_j^k).

High level description of protocol Π_{TE} (more details in the full version[22]): Recall that, given a tree T and an input $\mathbf{x}$, finding the index i^* such that the polynomial P_{d,i^*} evaluates 0 on the values $\{n_j\}_j$ is equivalent to compute $T(\mathbf{x})$ (*i.e.*, $T(\mathbf{x}) = \ell_{i^*}$). Therefore, finding i^* is sufficient in order to then compute an additive sharing of $T(\mathbf{x})$. In the privacy-preserving scenario, the main challenges in doing this are: 1) First of all, notice that neither the server or the user can see i^* in the clear, indeed knowing the index of the reached leaf can leak information about the inputs and the tree structure (when more than a request is made). We solve this using a simple *tree randomization* stratagem that hides i^* for the user (*i.e.*, the user gets to know i^* for an tree T' equivalent to T but with nodes randomly permuted by the server) and a standard cryptographic tool called *oblivious transfer* that hides i^* for the server (*i.e.*, once that the user gets i^* for T', the oblivious transfer protocols allows it to receive ℓ_{i^*} without revealing i^* to the server); 2) Then observe that neither the server or the user can see the $\{n_j\}_j$ in the clear, indeed also these values can leak information about $\mathbf{x}$ or T. To solve this we use the homomorphic property of the underlying LHE [a], the secure comparison protocol Π_{SC} and an algebraic property of the polynomials $\{P_{d,i}\}_i$. Since each $n_j = N_j(\mathbf{x})$ is a linear function of $\mathbf{x}$, the server can compute $\mathsf{Enc}_{\mathbf{pk}}(N_j(\mathbf{x}))$ from $\mathsf{Enc}_{\mathbf{pk}}(\mathbf{x})$ (assuming that the underlying scheme is an LHE scheme); then the server and the user run protocol Π_{SC}: the server is party 1 with $a = N_j(\mathbf{x})$ and the user is party 2; at the end they know α_j and β_j, respectively and such that $\alpha_j \beta_j = n_j$. However, the value n_j is kept secret. Finally, notice the following: for each $i = 1, \ldots, 2^d$ we have $P_{d,i}((\beta_j)_{j \in \mathcal{I}_i}) \prod_{j \in \mathcal{I}_i} \alpha_j = P_{d,i}((n_j)_{j \in \mathcal{I}_i})$ and therefore $P_{d,i}((n_j)_{j \in \mathcal{I}_i}) = 0$ if and only if $P_{d,i}((\beta_j)_{j \in \mathcal{I}_i}) = 0$. This implies that i^* can be computed locally by the user that knows $\{\beta_j\}_j$.

Complexity of the system in terms of cryptographic operations: During the off-line phase, the providers run the model-encryption procedure to encrypt their models $RF_1, \cdots, RF_t$. Assume that RF_k has hyperparameters d_k and m_k, then for P_k the model-encryption procedure costs $\Theta(m_k\, 2^{d_k})$ encryptions. Moreover, P_k sends to the server $m_k\, 2^{d_k+1}$ ciphertexts. The complexity of the on-line phase is dominated by the m repetitions of protocol Π_{TE}. The latter requires $\Theta(n\, 2^d)$ operations ($\Theta(2^d)$ for the user and $\Theta(n\, 2^d)$ for the server) and generates $\Theta(2^d)$ ciphertexts exchanged among the server and the user to score a tree of depth d on an instance with n features. Therefore, the on-line phase has complexity proportional to $n\, m\, 2^d$, where $d = \max_k\, d_k$. Finally, notice that many steps of our system can be easily *run in parallel*. For example, the m needed instances of protocol Π_{TE} can be executed concurrently.

Discussion: random forest (RF) hyperparameters

Since the depth and number of trees (*i.e.* model hyperparameters) affect the efficiency of our system, we provide here an empirical demonstration that bounding them can be done without adversely the prediction efficacy.

Bounded depth: Typically, during the training phase a RF is grown such that each tree may split in a greedy fashion without concern for the depth of the trees. We provide here an empirical inspection of the effect of bounding the depth to a maximum value d on the efficacy (AUC value) of the learned forest. We utilize the public Kent Ridge Colon-cancer dataset from the UCI repository (reference in Table 2) and we looked at various combinations of d and the number of trees in the forest, m. Specifically, we consider values of d in $\{1, 2, \ldots, 28, 30\}$ and 25 different choices of m in $\{1, 5, 10, 25, 50, 100, 200, 300, \ldots, 1900, 2000\}$. For each pair of values, we performed 30 replicates of a RF model construction and evaluation process. For each model, the construction began with choosing a random 70% of the data to serve as training data and the remaining 30% as testing data. A model was then built with the specified hyperparamters and AUC was measured on the testing data. In Fig. 3 we present the results of this investigation as a heatmap. For this task even a maximum depth of 6 was competitive with larger depth choices if 300 trees are considered. This suggests that while the standard learning algorithm may greedily grow trees very deeply, the overall performance is not substantially impacted by bounding the maximum depth.

Tuning methodology: Common practice for training ML algorithms involves some selection method for determining

[a] A party (different form the server) runs $\mathsf{Gen}(\kappa)$, makes $\mathbf{pk}$ public and safely stores $\mathbf{sk}$. The user needs to authenticate itself with this party in order to get the secret key $\mathbf{sk}$. Notice that the role of this party can be assumed by the user itself or by one or more of the providers.

a choice for the hyperparameters. One standard selection method is a grid-based search wherein a researcher will predefine some set of choices for each hyperparameter and then compute the cross product of these sets and choose the combination that maximized the AUC of the model. For example, for RF, we pick the hyperparameters as $d^*, m^* = arg\,max_{(d,m)\in D\times M} AUC(RF(d,m))$, where $RF(d,m)$ is a RF trained with hyperparameters d and m, $AUC(\cdot)$ is the AUC of a given RF on some held aside validation data and D, M are fixed sets. However, this procedure searches all combinations of d and m, whereas we are interested in controlling the value $m\,2^d$ because the overhead of our system its directly proportional to it. Therefore, between two hyperparameters choices giving the same efficacy, we are interested to choose the one that produces smaller overhead. In other words, our approach for tuning is the following: we fix a value s and then we maximize the model efficacy constrained to choosing the hyperparameters d and m in the set $Q_s = \{(m,d) \in \mathbb{Z}^+ \times \mathbb{Z}^+ \mid m\,2^d \leq s\}$. The gray line in Fig. 3 depicts the boundary of Q_s when $s = 2^{15}$ and dictates that choices above it are too large, and choices below are of acceptable overhead. Even if the number of acceptable choices is relatively small compared to the total number of combinations, it is worth noting that we saw competitive performance as both depth and number of trees exceeded some minimum choices. This suggests that we may be able to achieve both good performance and small overhead.

Performance: efficacy

To conclude our work, we want to experimentally validate our system. First, we study the effect of the "locally learn then merge" approach on the prediction accuracy. In particular, we want to compare the accuracy of the proposed method with the one of the standard "merge then learn" approach[b]. We provide an empirical investigation of this in two fashions: across three disease prediction tasks using EHR data from the Marshfield Clinic in Marshfield, WI, and across five predictions tasks from the UCI database.

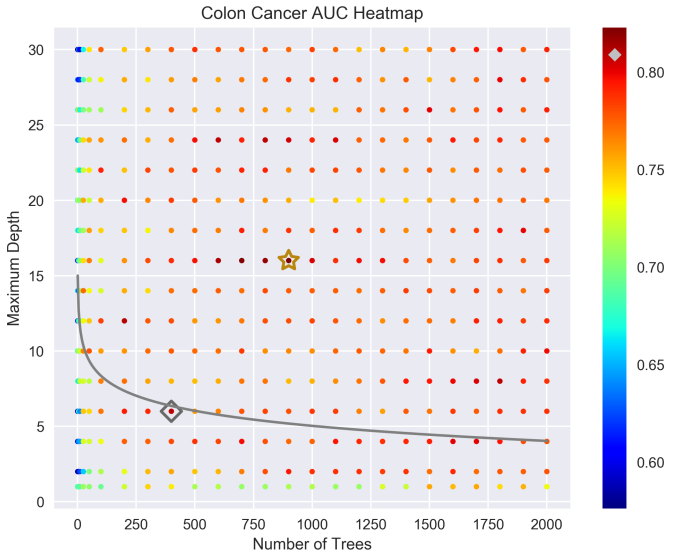


Figure 3: Heatmap of mean AUC values for various combinations of d and m. The gray line indicates $m\,2^d = 2^{15}$. The gold star indicates the best overall combination of d and m (AUC=0.823), the silver diamond indicates the best overall combination constrained by $m\,2^d \leq 2^{15}$ (AUC=0.809). The silver diamond is also on the colorbar indicating the corresponding AUC.

Real EHR Data. We consider the tasks of predicting three diseases 1 month in advance: Influenza (ICD-9 487.1), Acute Myocardial Infarction (ICD-9 410.4), and Lung Cancer (ICD-9 162.9). Each dataset was comprised of up to 10,000 randomly selected case-control matched patients on age and date of birth (within 30 days), with cases having 2 or more positive entries of the target diagnosis on their record and the control having no entries (rule of 2). Data for each case-control pair were truncated following 30-days prior to the case patient's first diagnosis entry to control for class-label leakage. Features were comprised of patient demographics, diagnoses, procedures, laboratory values and vitals. Unsupervised feature selection was performed on a per-disease basis first with a 1% frequency-based filter to remove very uncommon features and then followed up with principal component analysis to reduce the overall dimension of the data to 1,000 (this was done to improve the performance speed of our algorithm). For each of the 3 diseases, we constructed, as a performance baseline, a RF model with 500 trees, a maximum depth of 8, and 10% of features considered at each split in the tree. Models were trained on 90% of the data and tested on a held aside 10%. We compared these baseline models (*i.e.*, "merge then learn" approach) with our "locally learn then merge" approach by again constructing a forest with the same hyperparameters except the training data were partitioned between two simulated providers each with 45% of the original data that were used to train two smaller forests of 250 trees each

[b]If there are no privacy concerns, parties can simply share their data with one another and learn a single model. Otherwise a privacy-preserving training algorithm can be used to achieve the same result.

ICD-9	Disease	Samples	Features	Base AUC	LLM AUC	Prediction Time (s)
487.1	Influenza	10,000	8,211	0.8011	0.7640	105.37±14.70
410.4	Acute MI	9,284	9,136	0.6797	0.6658	121.75±9.43
162.9	Lung Cancer	10,000	9,021	0.6313	0.5786	125.94±8.19

Table 1: Efficacy testing results for 3 EHR datasets. Number of features are calculated before applying PCA (post-PCA selected the top 1,000 components). Base AUC refers to a forest learned on the whole dataset ("merge then learn" approach) and LLM AUC refers to a forest learned in our "locally learn then merge" fashion. Prediction Time refers to the mean±std time required for our system to return a prediction for a single patient query.

and then merged together. Model performance was measured using the area under the receiver operating characteristic curve (AUC), a common ML accuracy metric. We present in Table 1 both the dataset information and results of our experimentation. We find that the AUC achieved on partitioned data for these three tasks are less than the shared data. While this efficacy loss is meaningful, it is possible that with the additional data providers it may be mitigated.

UCI Datasets. We use five UCI datasets (references in Table 2) to investigate the effect of the number of providers sharing data on the performance of a RF. To simulate a dataset being shared amongst t providers, we randomly split each UCI dataset into t equal sized and unique chunks, $D_1, \ldots, D_t$, with each chunk belonging to a single provider. Each chunk was then split into a training (70% of the data) and testing set (30% of the data), *i.e.* $D_i = \text{Train}_i \cup \text{Test}_i$. We then learned models in three different ways. To simulate the effect of "zero sharing" (*i.e.*, providers with silo data do not share data or models), provider i learns a forest on Train_i and tests on Test_i achieving AUC_i with the average silo AUC taken as the mean across all t providers. Each forest was learned with 50 trees of maximum depth 8. To simulate the effect of "locally learn then merge", each provider learns a RF on their own training data, the forests are merged together, and the AUC is calculated on the merged testing data, $\cup_i \text{Test}_i$. Again, each provider learned 50 trees of maximum depth 8 and the final merged forest being of size $50\,t$ trees. To simulate the effect of a merged dataset ("merge the learn") we learn a single forest with $50\,t$ trees and maximum depth 8 from $\cup_i \text{Train}_i$ and then evaluate the AUC on $\cup_i \text{Test}_i$. This process was repeated 50 times to produce confidence intervals and performed for each of the five datasets in Table 2 across five choices of $t \in \{2, 3, 4, 5, 6\}$. We present the results of these experiments in Fig. 4. We see from it that the effect of locally learning the merging has neither a strictly positive or negative effect on the quality of the model. Indeed, our results indicate that the effect is dataset dependent. Therefore, we believe that it would be critical for a provider to investigate how the quality of their predictions are impacted by merging their learned models with another hospital system as compared to using their own data.

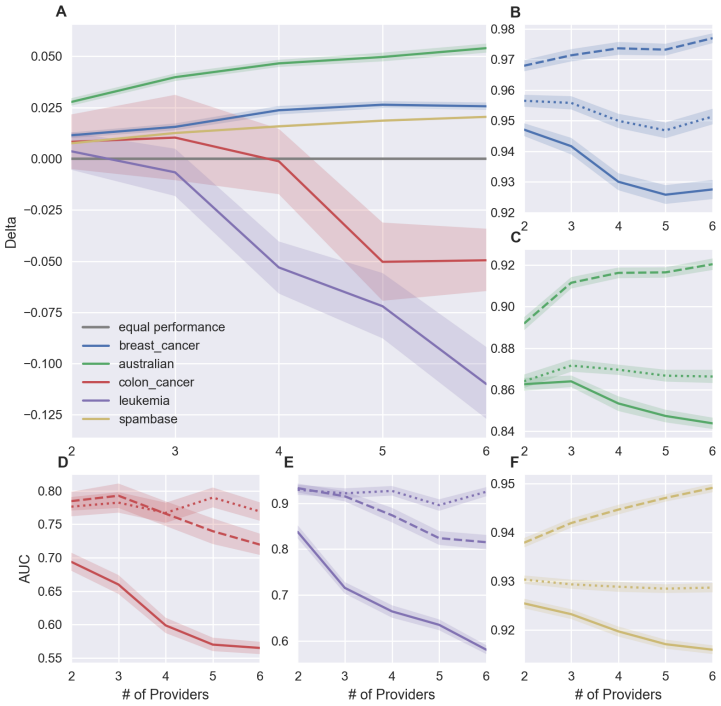


Figure 4: Effect of locally learning then merging compared to learning from a merged dataset. Subfigures **B**-**F** shows on the datasets of Table 2 how AUC is impacted by the number of providers. The dashed, solid and dotted lines shows AUC values for the locally learn then merge, the zero-sharing and the merge then learn approach, respectively. Subfigure **A** shows the AUC difference between the locally learn then merge and merge then learn (positive values indicate an improvement using our approach).

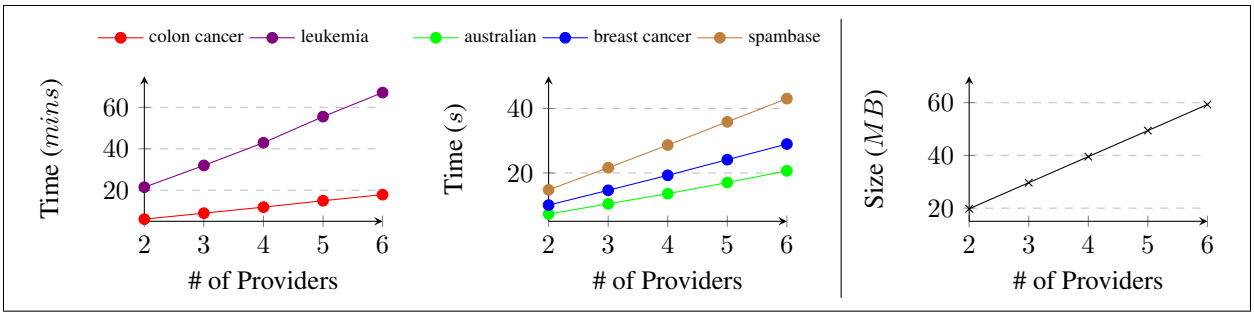


Figure 5: Performance of protocol Π_{TE} on $(50 \times \#$ of Providers) trees of depth 8 for the datasets of Table 2.

Performance: efficiency

Implementation details. To test efficiency (*i.e.*, bandwidth and running time) we implemented our proposed system in Python3.5.2. As underlying LHE we use Joye-Libert's scheme[21] with $\mathcal{M} = \mathbb{Z}_{2^{64}}$ and 100-bit security. We assume all inputs are real number with absolute value less or equal to $2 \cdot 10^3$ and at most 3 digits in the fractional part. To convert them into values in $\mathcal{M}$, we multiply each value by 10^3. This allows to represent all inputs with 21-bits values (we represent negative values using the upper half of $\mathbb{Z}_{2^{21}}$) and avoid overflow in the secure comparison protocol. The $\binom{2^d}{1}$-OT protocol for 20148-bit strings is implemented[24] using d calls to a standard $\binom{2}{1}$-OT protocol (*i.e.*, emp-toolkit) for 100-bit strings and $2d$ calls to a PRF (*i.e.*, AES_{128}). We provide an empirical investigation of the efficiency in two fashions: using a commodity machine and using the HTCondor system.

	size	n	Model-encryption	
			Time (s)	Size (MB)
Australian	609	14	0.84	7.96
Breast cancer	569	30	0.90	9.6
Spambase	4601	57	1.01	12.35
Colon cancer	62	2000	10.57	210.54
Leukemia	72	7129	41.7	733.69

Table 2: References for the UCI datasets (n = number of features). The last two columns show the overhead of the off-line phase of our system.

Commodity machine. We report the performance of our system executed on a commodity machine (60GB memory and 48core CPU, Intel Xeon CPU E5-2680 v3) for the UCI datasets of Table 2 in the setting described before (*i.e.* each provider knows a RF with 50 trees of maximum depth 8). Several tasks in the implementation were parallelized by multi-threading; all the timing values are averaged on five repetitions of the same experiment. Table 2 (last two columns on the right) reports the running time of the model-encryption procedure executed by one provider during the *off-line phase*; it also reports the size of the encrypted model obtained via this procedure. The number n of features influences both results, however even for the high dimensional cases (*i.e.*, thousands of features) the encrypted model has size less than 1 GB and is produced in less than a minute. The *on-line phase* of our system consists of three steps: first, the user submits its encrypted input to the server. Clearly, the performance of this step is influenced only by the encryption scheme used and by the dimension of the input (*i.e.*, number of features n). In our experiments, even for the largest value of n, this step takes less than a second (*e.g.*, 0.17 seconds for $n = 7129$). Then, the server and the user execute m times the protocol Π_{TE} to evaluate each tree in the merge of all the forests. Fig. 5 illustrates the performance of this part (the most expensive one in the on-line phase): The two graphs on the left depict the running time of the protocol Π_{TE} run on $50\,t$ trees as function of the parameter t, number of providers; the results are dataset dependent since the server executes $\Theta(n\,2^d)$ cryptographic operations. The graph on the right of Fig. 5 reports the size of the messages exchanged by the server and the user as function of t. This value is not influenced by n (dataset size) and it only increases linearly with the number of trees; in our experiment, even for 300 trees the bandwidth required is always less than 60 MB. In the last step of the on-line phase, the server and the user sum their shares; the overhead of this step is independent of n and influenced only by the total number of trees (*e.g.*, in our experiment this needs less than 8 ms for 300 trees).

HTCondor. The experiments for the real EHR data were executed using the HTCondor system, a high-throughput computing architecture that we utilized in a "master-worker" fashion. For each forest, one tree was learned as a separate "job" exploiting the heavy parallelization available to RFs. Thus, both training and prediction were performed

in a high-throughput manner. We report the running time of the on-line phase in this setting in the last column on the right of Table 1. We find that this parallelized version of our algorithm allows us to provide near real-time interactions as predictions are returned on average within two minutes of providing a query to the system. We believe that this would be reasonably fast enough to support the workflow of a physician who wishes to query the model for a patient.

Conclusion

We propose a new approach for computing privacy-preserving collaborative predictions using random forests. Instead of a system composed by a training algorithm, which usually has high overhead in the privacy-preserving setting, followed by a scoring algorithm, we propose a system based on locally learning and then privacy-preserving merging. To avoid the need for providers to be on-line for each prediction request, we instantiate the new approach in the cloud model. That is, an untrusted server collects the locally trained models in encrypted form and takes care of scoring them on a new private instance held by the user. Our system is secure in the honest-but-curious security model and extending it to the malicious model, especially for a corrupted server, is an interesting direction for future work. We evaluate the performance of our system on real-world datasets, the experiments we conducted show that (1) the efficacy of the new approach is dataset dependent; this opens to future works that aim to characterize this dependency in terms of the dataset parameters and distribution, (2) the efficiency is influenced by the forest hyperparameters, which we showed we can control, and by the number of features n, which is given by the specific application; avoiding the dependency on n is another interesting direction that may lead more efficient implementation of this new approach.

Acknowledgments. This work was partially supported by the Clinical and Translational Science Award (CTSA) program, through the NIH National Center for Advancing Translational Sciences (NCATS) grant UL1TR002373, by the NIH BD2K Initiative grant U54 AI117924 and by the NLM training grant 5T15LM007359.

References

1. International Warfarin Pharmacogenetics Consortium. Estimation of the warfarin dose with clinical and pharmacogenetic data. *New England Journal of Medicine*, 360(8):753–764, 2009.

2. Nils Homer, Szabolcs Szelinger, Margot Redman, David Duggan, Waibhav Tembe, Jill Muehling, John V Pearson, Dietrich A Stephan, Stanley F Nelson, and David W Craig. Resolving individuals contributing trace amounts of dna to highly complex mixtures using high-density snp genotyping microarrays. *PLoS genetics*, 4(8):e1000167, 2008.

3. Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy, 18-21 May 2008, Oakland, California, USA*, pages 111–125, 2008.

4. Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, pages 36–54, 2000.

5. Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

6. Eric Lantz. *Machine Learning for Risk Prediction and Privacy in Electronic Health Records*. PhD thesis, The University of Wisconsin-Madison, 2016.

7. Theodora S. Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch. Paschalidis, and Wei Shi. Federated learning of predictive models from federated electronic health records. *I. J. Medical Informatics*, 112:59–67, 2018.

8. Mingjun Xiao, Liusheng Huang, Yonglong Luo, and Hong Shen. Privacy preserving ID3 algorithm over horizontally partitioned data. In *Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2005), 5-8 December 2005, Dalian, China*, pages 239–243, 2005.

9. Saeed Samet and Ali Miri. Privacy preserving ID3 using gini index over horizontally partitioned data. In *The 6th ACS/IEEE International Conference on Computer Systems and Applications, AICCSA 2008, Doha, Qatar, March 31 - April 4, 2008*, pages 645–651, 2008.

10. Jaideep Vaidya, Chris Clifton, Murat Kantarcioglu, and A. Scott Patterson. Privacy-preserving decision trees over vertically partitioned data. *TKDD*, 2(3):14:1–14:27, 2008.

11. Sebastiaan de Hoogh, Berry Schoenmakers, Ping Chen, and Harm op den Akker. Practical secure decision tree learning in a teletreatment application. In *Financial Cryptography and Data Security - 18th International Conference, Barbados, March 3-7, 2014, Revised Selected Papers*, pages 179–194, 2014.

12. Jaideep Vaidya, Basit Shafiq, Wei Fan, Danish Mehmood, and David Lorenzi. A random decision tree framework for privacy-preserving data mining. *IEEE Trans. Dependable Sec. Comput.*, 11(5):399–411, 2014.

13. Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*, 2015.

14. Raymond K. H. Tai, Jack P. K. Ma, Yongjun Zhao, and Sherman S. M. Chow. Privacy-preserving decision trees evaluation via linear functions. In *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II*, pages 494–512, 2017.

15. Martine De Cock, Rafael Dowsley, Caleb Horst, Raj Katti, Anderson Nascimento, Wing-Sea Poon, and Stacey Truex. Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation. *IEEE Transactions on Dependable and Secure Computing*, 2017.

16. Marc Joye and Fariborz Salehi. Private yet efficient decision tree evaluation. In *Data and Applications Security and Privacy XXXII - 32nd Annual IFIP WG 11.3 Conference, DBSec 2018, Bergamo, Italy, July 16-18, 2018, Proceedings*, pages 243–259, 2018.

17. Michael Backes, Pascal Berrang, Matthias Bieg, Roland Eils, Carl Herrmann, Mathias Humbert, and Irina Lehmann. Identifying personal DNA methylation profiles by genotype inference. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 957–976, 2017.

18. Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 498–507, 2007.

19. Geetha Jagannathan, Krishnan Pillaipakkamnatt, and Rebecca N. Wright. A practical differentially private random decision tree classifier. *Trans. Data Privacy*, 5(1):273–295, 2012.

20. Santu Rana, Sunil Kumar Gupta, and Svetha Venkatesh. Differentially private random forest with high utility. In *2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14-17, 2015*, pages 955–960, 2015.

21. Marc Joye and Benoît Libert. Efficient cryptosystems from 2^k-th power residue symbols. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 76–92, 2013.

22. Irene Giacomelli, Somesh Jha, Ross Kleiman, David Page, and Kyonghwan Yoon. Privacy-preserving collaborative prediction using random forests (full version). *arXiv.org*.

23. Florian Kerschbaum and Orestis Terzidis. Filtering for private collaborative benchmarking. In *Emerging Trends in Information and Communication Security, International Conference, ETRICS 2006, Freiburg, Germany, June 6-9, 2006, Proceedings*, pages 409–422, 2006.

24. Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 245–254, 1999.