Cryptographically Private Support Vector Machines

Sven Laur
Laboratory for Theoretical Computer Science
Helsinki University of Technology
slaur@tcs.hut.fi

Helger Lipmaa
Institute of Computer Science,
University of Tartu
and Cybernetica AS
lipmaa@cs.ut.ee

Taneli Mielikäinen
HIIT Basic Research Unit,
Department of Computer Science, University of Helsinki
tmielika@cs.helsinki.fi

ABSTRACT

We propose private protocols implementing the Kernel Adatron and Kernel Perceptron learning algorithms, give private classification protocols and private polynomial kernel computation protocols. The new protocols return their outputs—either the kernel value, the classifier or the classifications—in encrypted form so that they can be decrypted only by a common agreement by the protocol participants. We show how to use the encrypted classifications to privately estimate many properties of the data and the classifier. The new SVM classifiers are the first to be proven private according to the standard cryptographic definitions.

Categories and Subject Descriptors

E.3 [**DATA ENCRYPTION**]: Public key cryptosystems; H.2.8 [**DATABASE MANAGEMENT**]: Database Applications—*Data mining*; H.2.7 [**DATABASE MANAGEMENT**]: Database Administration—*Security, integrity, and protection*

General Terms

Theory, Algorithms, Security

Keywords

Privacy Preserving Data Mining, Kernel Methods

1. INTRODUCTION

Private classification like ordinary classification comprises of two subtasks: learning a classifier from data with class labels—often called a training data—and predicting the class labels for unlabeled data using the learned classifier. However, the main emphasis is on privacy, i.e., how to disclose only the minimal amount of data. There are two fundamentally different ways algorithms can disclose sensitive information: algorithms can leak some side information that is not specified by desired output or the end result itself reveals sensitive aspects of the data. As common in cryptographic literature, we address only the first question, i.e., we design algorithms that only reveal the desired output.

For simplicity, we assume that the data can be stored as vectors with fixed length, often called *feature vectors*. As an example, consider the classification task of detecting email spam. The training data comprises of emails with labels "spam" or "nospam". More precisely, classified emails are converted to word count vectors and then the classifier is learned from these vectors. The classifier itself can be, e.g., a linear threshold function on the word frequencies in the bodies of the messages. The learned classifier is used to predict which of the unlabeled emails are spam.

Private classification considers the scenario where the training data is divided between two or more parties with possibly conflicting interests, so that they are not willing to reveal their data. However, the parties are willing to train a common classifier provided that none of them can use it without others and that their data remains private. Such examples are quite common in the case of medical studies, e.g., when finding out risk groups for a diseases without leaking the identities of infected patients and the medical data. Other similar examples include military surveillance and identity-specific content providing services.

We derive private versions of the Kernel Perceptron and Kernel Adatron algorithms that extend the basic linear classification techniques. In particular, the Kernel Adatron algorithm can be used to implement both hard and soft margin Support Vector Machines. See [14, 13] for references. As SVMs have excellent statistical stability and sensitivity, they have been successful in many application areas. Hence, our work is an important extension of the research on cryptographically private classifiers [2, 8, 6, 15, 16].

Data perturbation combined with robust aggregation techniques provides also privacy-preserving methods for classification. However, there the context is completely different: the training data is owned by a single entity and data perturbation is used to protect privacy of individual records. Basic applications are various statistical questionnaires and databases that must preserve anonymity of each participant. Such techniques have several intrinsic limitations: privacy guarantees are somewhat heuristic and there is a tradeoff between privacy and accuracy.

By using classical results of secure multi-party computation [5], any protocol can be implemented without leakage of any side information, though with "polynomial" slowdown. Thus, secure multi-party computation methods can

KDD'06, August 20–23, 2006, Philadelphia, Pennsylvania, USA.
Copyright 2006 ACM 1-59593-339-5/06/0008 ...$5.00.

be applied to any protocol to avoid unnecessary disclosure. However, such generic techniques are usually too resource-consuming in practice. This is especially true in the case of data mining protocols that handle enormous amount of data, and are often themselves on the verge of being (im)practical. In the current article, we combine several well-known cryptographic techniques such as homomorphic encryption, secret sharing and secure circuit evaluation to get reasonably efficient private classification algorithms. As an important restriction, all proposed algorithms are private only in the semi-honest model where all participants follow the protocol but try to deduce extra information.

We derive protocols for the three basic steps of kernel based classifiers: *evaluation of the kernel matrix, prediction* and *training.* The complexity of the algorithms depends on how the data is divided between participants. Due to the space limitations, we cover completely only the simplest case when the feature vectors are owned by Server, and Client possesses only the classification labels. General horizontal and vertical split have slightly more complex solutions, since there Client and Server must first share the kernel matrix. In this shortened version, we outline only the main differences between the simplified and the complex case.

A few methods for privacy-preserving learning of Support Vector Machines have been proposed [18, 17] but they reveal the kernel and the Gramm matrix of the data. Since the Gramm matrix consists of scalar products between all data vectors, such leak is extremely dangerous. If more than m linearly independent vectors leak out, where m is the dimensionality of the data, then all other vectors can be restored knowing only the Gramm matrix. Hence, these methods are unusable for horizontally partitioned data, where each participant possesses many complete feature vectors. Moreover, other kernel methods like the Kernel-PCA reveal statistically relevant information about data points without any auxiliary knowledge beyond the kernel matrix.

2. CLASSIFICATION

Let $\mathcal{X}$ be the set of all possible data points and $\mathcal{Y}$ be the set of possible classes. Let $\mathcal{G}$ be a family of functions $g : \mathcal{X} \to \mathcal{Y}$ that we consider being potential classifiers, and let $\mathcal{D}$ be a multiset of data points with class labels, i.e., $\mathcal{D}$ comprises of pairs $(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$. Usually the pairs in $\mathcal{D}$ are assumed to be drawn independently from the same unknown probability distribution, often referenced as i.i.d. data. We consider only the case when vectors are real, $\mathcal{X} \subseteq \mathbb{R}^m$, and there are two classes $\mathcal{Y} = \{-1, 1\}$.

The classifier learning task is, given the function class $\mathcal{G}$ and the dataset $\mathcal{D}$, to find the best classifier $g_* \in \mathcal{G}$. Ideally, one would like to have a classifier with smallest misclassification probability $\Pr[g(X) \neq Y]$, where X and Y are random variables with joint probability distribution over $\mathcal{X} \times \mathcal{Y}$. As the actual probability distribution on $\mathcal{X} \times \mathcal{Y}$ is unknown, we must relay on a partial information revealed by $\mathcal{D}$.

We consider only linear classifiers and their extensions. Linear classifiers are described by the normals of the hyperplanes $\vec{w} \in \mathbb{R}^m$. The classification of a point $\vec{x} \in \mathbb{R}^m$ is then determined by the sign of the scalar product $f_{\vec{w}}(\vec{x}) := \langle \vec{w}, \vec{x} \rangle$ that is also known as the *discriminative function.* The most common linear classification algorithm is known as Perceptron. The idea of the Perceptron algorithm is to find a linear combination $\vec{w}$ of the points $\vec{x}_i$ such that $\operatorname{sign} \langle \vec{w}, \vec{x}_i \rangle = y_i$ for all $(\vec{x}_i, y_i) \in \mathcal{D}$. The algorithm updates the weight vector $\vec{w}$ (initially $\vec{0}$) by adding to $\vec{w}$ each data point $\vec{x}_i$ that is misclassified by the current $\vec{w}$. See [13, 14] for more details.

A major drawback of the Perceptron algorithm is that it assumes that the data is linearly separable, i.e., that there is an hyperplane $\vec{w} \in \mathbb{R}^m$ that separates the positive examples from the negative ones. Therefore, data is often mapped into a higher dimensional Hilbert space $\mathcal{H}$ to make it linearly separable using some (nonlinear) mapping $\phi : \mathcal{X} \to \mathcal{H}$. Such a mapping is often called a *feature mapping* and the Hilbert space a *feature space.* Common feature spaces have very high or even infinite dimensionality and computations in feature spaces are done implicitly using kernels. A kernel of a feature map ϕ is a function κ such that $\kappa(\vec{x}_i, \vec{x}_j) = \langle \phi(\vec{x}_i), \phi(\vec{x}_j) \rangle$ for all $\vec{x}_i, \vec{x}_j \in \mathcal{X}$. Many machine learning algorithms can be written in dual form by expressing sought feature vectors as linear combination of $\phi(\vec{x}_1), \ldots, \phi(\vec{x}_n)$. In particular, if $\vec{w} = \alpha_1 \phi(\vec{x}_1) + \cdots \alpha_n \phi(\vec{x}_n)$ for some $\vec{\alpha} \in \mathbb{Z}^n$, then

$$f_{\vec{w}}(\vec{x}_i) = \sum_{j=1}^{n} \alpha_j \cdot \langle \phi(\vec{x}_i), \phi(\vec{x}_j) \rangle = \sum_{j=1}^{n} \kappa(\vec{x}_i, \vec{x}_j) \alpha_j \enspace ,$$

i.e., it suffices to compute only the kernel values $\kappa(\vec{x}_i, \vec{x}_j)$. Furthermore, the values $k_{ij} = \kappa(\vec{x}_i, \vec{x}_j)$ have to be computed only once for a particular $\mathcal{D}$ and ϕ. Let $K = (k_{ij})_{i,j=1}^{n}$ denote the kernel matrix of $\mathcal{D}$. Then the Perceptron algorithm can be written down as Algorithm 1.

Algorithm 1 Kernel Perceptron algorithm

```
Input: A kernel matrix $K$ and class labels $\vec{y} \in \{-1, 1\}^n$.
Output: A weight vector $\vec{\alpha} \in \mathbb{Z}^n$.

Function Kernel-Perceptron($K$, $\vec{y}$)
  $\vec{\alpha} \leftarrow \vec{0}$
  repeat
      for $i = 1, \ldots, n$ do
          if $y_i \cdot \sum_{j=1}^{n} k_{ij} \alpha_j \leq 0$ then $\alpha_i \leftarrow \alpha_i + y_i$
      end for
  until convergence
end function
```

By Novikoff's Theorem [14], the number of iterations before convergence is less than R^2/γ_*^2, where R is the radius of the smallest origin-centered ball containing all data points, and γ_* is the maximal margin. Recall that the margin of a given weight vector $\vec{w}$ w.r.t. the dataset $\mathcal{D}$ is defined as

$$\gamma = \min_{(x_i, y_i) \in \mathcal{D}} \frac{y_i \cdot \langle \vec{w}, \vec{x}_i \rangle}{\|\vec{w}\|}$$

and $\gamma_* = \max \{\gamma(\vec{w}) : \vec{w} \in \mathbb{R}^m\}$. However, the output of the Perceptron algorithm is ambiguous, as it finds some separating hyperplane for data if such exists, but basically any separating hyperplane will do. It is more natural to select the separating hyperplane that maximizes the margin γ, i.e., the maximum margin hyperplane $\vec{w}_*$. Intuitively, such choice minimizes the risk of misclassification. The maximum margin hyperplane is justified also by the generalization error bounds [13, 14]. Learning algorithms that output a maximum margin separating hyperplane are called Support Vector Machines (SVM-s in short) [14]. A particularly flexible and simple Support Vector Machine is the Adatron algorithm [13].

The Adatron algorithm has several nice properties. First, it is based on iterative gradient descent and has a simple structure. Therefore, it is a perfect starting point for a privacy-preserving learning algorithm, since there are only a few operations that require complex cryptographic solutions. Second, the Adatron algorithm allows to implement both hard and soft margin Support Vector Machines with few changes. Recall that a hard margin SVM finds the maximal margin hyperplane if the dataset is linearly separable. For linearly non-separable datasets, the hard margin SVM returns a solution where outliers—points that cause non-separability—have large impact on classification results. Soft margin SVM-s bound these harmful disturbances: either $\alpha_j \in [0, C]$ is forced (ℓ_1-norm SVM) or a regularizing term $C > 0$ is added to the main diagonal of the kernel matrix (ℓ_2-norm SVM). Algorithm 2 implements the ℓ_1-norm soft margin SVM, which is the most popular SVM. We get a hard margin SVM by setting $C = \infty$, and a ℓ_2-norm SVM by adding C to the main diagonal.

Algorithm 2 Kernel Adatron algorithm

Input: A kernel matrix K, class labels $\vec{y} \in \{-1, 1\}^n$ and the soft margin parameter C.
Output: A weight vector $\vec{\alpha} \in \mathbb{Z}_+^n$.

```
Function Kernel-Adatron($K$, $y$, $C$)
  $\vec{\alpha} \leftarrow \vec{0}$
  repeat
      for $i = 1, \dots, n$ do
          $\alpha_i \leftarrow \alpha_i + \left(1 - y_i \cdot \sum_{j=1}^{n} k_{ij}\alpha_j y_j\right)$
          $\alpha_i \leftarrow \min\{\max\{\alpha_i, 0\}, C\}$
      end for
  until convergence
end function
```

3. CRYPTOGRAPHIC AIMS AND TOOLS

Our main assumption is that data is divided between two parties, Client and Server, that are willing to train a common classifier if nothing beyond the expected end results are revealed. In the matrix evaluation and training phase, Client and Server must learn nothing new. In the prediction phase, Client must learn only the predicted label $f_{\vec{w}}(\vec{x})$ and Server must learn nothing. In case of secure aggregation, even the individual class labels must remain secret and Client should learn only the aggregate value, e.g., the training error.

Feature vectors can be divided horizontally, vertically or in a more complex way. Essentially, there is no difference in private learning algorithms, unless the data is divided between Client and Server so that Client possesses the label vector $\vec{y}$ and Server has the corresponding feature vectors $\vec{x}_i$. We call such scenario a *restricted vertical split*. As the vectors $\vec{x}_i$ correspond to the real life objects, it is quite plausible that Client can still classify the objects although the features $\vec{x}_i$ are not known. Examples of restricted vertical split naturally emerge when Client must use a confidential database for classification, e.g., medical and genetic studies. Since Server owns all feature vectors, the kernel matrix K can be locally computed. Recall that Algorithms 1 and 2 require efficient evaluation of linear forms $f_{\vec{w}}(\vec{x}_i)$. If Server knows all entries of K, then an additively homomorphic encryption is sufficient for secure evaluation of $f_{\vec{w}}(\vec{x}_i)$.

In all other data sharing models, Client and Server must use cryptographic methods to share K, such that neither of them learns anything about K. Then, for the secure evaluation of $f_{\vec{w}}(\vec{x}_i)$, we need a two-party homomorphic cryptosystem where decryption requires collaboration between Client and Server. Due to the space constraints, we consider only restricted vertical split. Complete treatment of all data sharing models along with corresponding security proofs are given in the full version [7].

Next, we introduce the formal security model and three basic cryptographic techniques: homomorphic encryption, secret sharing and secure circuit evaluation. Since all these techniques can natively handle only integer inputs, classification algorithms must be discretized, i.e., fixed point arithmetics must be used instead of floating point calculations. This introduces some intricate questions about numerical stability that are discussed further in the later sections.

First lets establish some notation. For a finite set X, let $x \leftarrow X$ denote that x is chosen uniformly from X. For an algorithm A with inputs $x_1, \dots, x_n$, let $A(x_1, \dots, x_n)$ denote the output distribution of A. Let k be the security parameter. A function $f(k)$ is $\mathsf{poly}(k)$ if $f(k) = k^{O(1)}$, i.e., if $f(k)$ increases asymptotically not faster than k^c for some $c > 0$. A function $f(k)$ is *negligible* if $f(k) = k^{-\omega(1)}$, i.e., if $f(k)$ decreases asymptotically faster than k^{-c} for any $c > 0$.

Formal security model. Let Π^f denote a *protocol* (a well-specified distributed algorithm) between Client and Server for computing the functionality $f = (f_1, f_2)$. Let ϱ be Client's private input and σ Server's private input. Intuitively, the protocol Π^f preserves privacy if Client learns nothing but $f_1(\varrho, \sigma)$, and Server learns nothing but $f_2(\varrho, \sigma)$. This intuitive notion is formalized by using the non-uniform polynomial security model [5, p. 620–624, 626–631]. A protocol is *private* if any probabilistic polynomial-time honest-but-curious adversary (that follows the protocol) obtains additional information with a negligible probability w.r.t. the security parameter k (e.g., the key length). That means that in this case, one can choose a sufficiently small security parameter k, such that the protocol is still efficient but the adversarial success probability is reasonably small, say 2^{-80}. See the full version of the article [7] for a detailed discussion.

The next (sequential) composition property allows to simplify cryptographic security proofs and omit unnecessary details. Let $\Pi^{g|f}$ denote a *sequential* protocol for computing functionality g, where parties can access a trusted third party TTP that computes functionality f. In other words, parties can send their arguments to the incorruptible TTP that privately replies with the answers f_1 and f_2. Now, let $\Pi^{f|g} \circ \Pi^f$ denote the protocol, where parties execute $\Pi^{g|f}$ but instead of TTP use Π^f to compute f. Then the following sequential composition theorem [5, p. 637] holds.

Composition Theorem 1. *Let protocols $\Pi^{g|f}$ and Π^f be private in the semi-honest model. Then the combined protocol $\Pi^g = \Pi^{f|g} \circ \Pi^f$ is also private in the semi-honest model.*

If the protocol $\Pi^{g|f}$ contains many invocations of f, then all of them can be safely replaced by an invocation of Π^f, provided that TTP always computes a single value of f. That is, we cannot run two instances of Π^f in parallel or otherwise the composition theorem might not hold.

Homomorphic encryption. Homomorphic cryptosystems provide an efficient way to securely evaluate linear forms when data is divided between Client and Server as it facilitates computations with ciphertexts. Formally, a public-key cryptosystem is a triple of algorithms $(\mathrm{G}, \mathrm{E}, \mathrm{D})$, where the key generation algorithm G with input 1^k returns a secret key sk and a public key pk corresponding to the security parameter k, E is the encryption algorithm, and D is the decryption algorithm. Let $\mathcal{P}$ and $\mathcal{C}$ denote the plaintext and ciphertext space. Then encryption with key pk implements a function $\mathrm{E}_{\mathsf{pk}} : \mathcal{P} \times \mathcal{R} \to \mathcal{C}$, where $\mathcal{R}$ denotes the randomness space used by the encryption algorithm. For the sake of brevity, we denote $\mathrm{E}_{\mathsf{pk}}(x) := \mathrm{E}_{\mathsf{pk}}(x; r)$ for a uniformly chosen $r \leftarrow \mathcal{R}$. It is required that always $\mathrm{D}_{\mathsf{sk}}(\mathrm{E}_{\mathsf{pk}}(x)) = x$, i.e., it is possible to decrypt cryptograms.

A cryptosystem is additively homomorphic if for any $(\mathsf{sk}, \mathsf{pk})$, (a) the plaintext space $\mathcal{P} = \mathbb{Z}_N$; (b) for $x, y \in \mathbb{Z}_N$

$$\mathrm{E}_{\mathsf{pk}}(x + y \mod N) = \mathrm{E}_{\mathsf{pk}}(x) \cdot \mathrm{E}_{\mathsf{pk}}(y) \enspace ,$$
$$\mathrm{E}_{\mathsf{pk}}(x \cdot y \mod N) = \mathrm{E}_{\mathsf{pk}}(x)^y \enspace ,$$

and $\mathrm{E}_{\mathsf{pk}}(x; r) \cdot \mathrm{E}_{\mathsf{pk}}(0)$ has the same output distribution as $\mathrm{E}_{\mathsf{pk}}(x)$. Hence, given sk and $\mathrm{E}_{\mathsf{pk}}(x) \cdot \mathrm{E}_{\mathsf{pk}}(y) \cdot \mathrm{E}_{\mathsf{pk}}(0)$, Client can deduce only $x + y \mod N$. If cryptosystem is secure then Server without sk leans nothing from $\mathrm{E}_{\mathsf{pk}}(x)$.

Security of a cryptosystem is defined as follows. Consider two experiments EXP_0 and EXP_1. In experiment EXP_i, $i \in \{0, 1\}$, $\mathrm{G}(1^k)$ is first executed to generate a new key pair $(\mathsf{sk}, \mathsf{pk})$. Then an adversary A, given pk, computes two messages $x_0, x_1 \in \mathcal{P}$. Next, A receives $\mathrm{E}_{\mathsf{pk}}(x_i)$. A cryptosystem is *IND-CPA secure*, if for any polynomial-time non-uniform algorithm A, the next difference is negligible: $\mathrm{Adv}(A) = \left|\Pr\left[A = 1 \middle| \mathsf{EXP}_0\right] - \Pr\left[A = 1 \middle| \mathsf{EXP}_1\right]\right|$. Here, the probability is taken over the random choices of G, E and A. Essentially all our security results follow from the composition theorem and from the next straightforward fact.

FACT 1. *Let Π be an IND-CPA secure cryptosystem. Assume Server is a polynomial-time non-uniform algorithm. If during a protocol execution, Server sees only* pk *and* $\mathrm{E}_{\mathsf{pk}}(x_i)_{i=1}^{\mathsf{poly}(k)}$ *then Server learns no new information.*

Several additively homomorphic cryptosystems [3, 12] are proven to be IND-CPA secure under reasonable complexity assumptions. All of them are based on modular exponentiations of large integers, say 1024 bits long, and thus quite resource consuming. Still thousands of encryption and decryption operation can be done per second, at least using dedicated hardware.

Secret sharing. Algorithms 1 and 2 above contain variables that can leak information about data points. Therefore, neither Client or Server must learn the values of these variables, however, together they must be able to manipulate with them. We use additive and multiplicative sharing for such variables. Let N be a public modulus. If (s_1, s_2) are chosen uniformly from the set $\left\{(s_1, s_2) \in \mathbb{Z}_N^2 : s_1 + s_2 = x \mod N\right\}$ then the knowledge of s_i reveals nothing about x, as s_i has uniform distribution. We call it the *additive sharing* of x. For invertible elements $\mathbb{Z}_N^* = \{a \in \mathbb{Z}_N : a \cdot b = 1 \mod N\}$, *multiplicative sharing* is defined by using the set of shares $\left\{(s_1, s_2) \in (\mathbb{Z}_N^*)^2 : s_1 \cdot s_2 = x \mod N\right\}$.

Conditional oblivious transfer. To efficiently implement private classification, we have to rely on *conditional oblivious transfer* (COT), also known as *secure circuit evaluation.* Conditional oblivious transfer protocol for a public predicate π is defined as follows. Client has an input ϱ and Server's input is a triple (σ, r_0, r_1). At the end of the protocol, Client learns r_0 if $\pi(\varrho, \sigma) = 0$. Otherwise, Client learns r_1. Server learns nothing. If Sender sets $r_0 = -s_2 \mod N$ and $r_1 = 1 - s_2 \mod N$ for random $s_2 \in \mathbb{Z}_N$, and Client stores the output of COT as s_1 then they have additively shared $s_1 + s_2 = \pi(\varrho, \sigma) \mod N$.

In *1-out-of-2 oblivious transfer* (OT), Server holds a two-element database (r_0, r_1) and Client holds an index ϱ. At the end of the protocol, Client learns r_ϱ if $\varrho \in \{0, 1\}$ and nothing otherwise. Server learns nothing. This can be seen as a special case of COT. The protocol must be secure even if Client is malicious (deviates arbitrarily from the protocol). For efficiency reasons, the OT protocol must remain secure even if a multiple instances of it are run in parallel and still have low amortized complexity, see, e.g., [1, 9].

A COT protocol, popularized and analyzed in [11], consists of three phases. First, Server sends a garbled circuit $\mathrm{E}(\mathrm{C}_\pi)$ to Client. Second, for each input bit, Client makes OT call to get the corresponding input for $\mathrm{E}(\mathrm{C}_\pi)$. Third, Client emulates computations in $\mathrm{E}(\mathrm{C}_\pi)$, and obtains k-bit string r_0 if $\pi(\mu, \sigma) = 0$ and string r_1 otherwise. This protocol has two rounds, is private in semi-honest model, and has even a freeware Java implementation $\mathsf{Fairplay}$ [10].

The following facts follow from the construction of [11]. Let the circuit C_π consist of ℓ_2 binary or duplication gates and ℓ_3 ternary gates (Unary gates are redundant, as they can be combined into binary or ternary gates). Then the size of garbled circuit $E(\mathrm{C}_\pi)$ is $(4\ell_2 + 8\ell_3 + 4\log_2(\frac{m}{k}))k$ bits, for $k \approx 80$. The computational complexity needed to construct and emulate computations in $\mathrm{E}(\mathrm{C}_\pi)$ is linear in the size of the circuit π. The main computational workload comes from n parallel executions of 1-out-of-2 OT protocols, i.e. bit length of ϱ must be as small as possible. Several instances of COT protocol can be run in parallel without loosing privacy in the semi-honest model.

In practice, thousands of OT protocols can be executed in parallel per second. Therefore, private comparison between n-bit integers is efficient, as latter can be done with n ternary gates. Still, we will consider several techniques how to decrease the bit-size of inputs of the COT protocol.

4. PRIVATE KERNEL SHARING

Kernel methods are typically applied to continuous data, and therefore most kernels operate over the real domain, except the discrete kernels that are used for text classification. As cryptographic methods natively support discrete ranges, we have to embed kernel values in $\mathbb{Z}_N = \{-L, \ldots, L\}$, where the odd integer $N = 2L + 1$ is sufficiently large to prevent overflows in computations.

If data points contain non-integer values then we need to map data vectors into the discrete domain. Let $\mathsf{toint} : \mathbb{R}^m \to \mathbb{Z}^m$ be the corresponding embedding that, say, multiplies its arguments by some large constant and then rounds them to the nearest integer value. Let $\widehat{\kappa} : \mathbb{Z}^m \times \mathbb{Z}^m \to \mathbb{Z}_N$ be the corresponding kernel approximation. We say that kernel approximation is *δ-precise* with respect to scaling factor $c > 0$ and domain $\mathcal{X}$, if for all $\vec{x}, \vec{y} \in \mathcal{X}$,

$$|c \cdot \widehat{\kappa}(\mathsf{toint}(\vec{x}), \mathsf{toint}(\vec{y})) - \kappa(\vec{x}, \vec{y})| \leq \delta \enspace .$$

Obviously, approximation errors can change classification results. On the other hand, numerical approximation errors emerge also in floating-point implementations where the precision is usually 32 bits (float precision). Moreover, it is reasonable to assume that if approximation is sufficiently precise then the modeling error, made by the choice of kernel, has much larger impact on the classification errors. As linear classification requires only evaluation of linear forms $\langle\vec{\alpha},\vec{\kappa}\rangle$, then 64-bit relative precision $\delta \approx 2^{-64}$ is sufficient to mimic float computations, as smaller values are rounded to zero even in case of floating-point operations. Such precision is achievable with a 64 bit modulus N, provided that $\kappa(\cdot,\cdot)$ is scaled into the proper range.

If Server does not own all feature vectors $\vec{x}_i$ then Client and Server have to privately share K. We consider only polynomial kernels; private evaluation of more complex kernels is an independent research topic. Evaluation of the scalar product kernel $\kappa(\vec{x}_i,\vec{x}_j) = \langle\vec{x}_i,\vec{x}_j\rangle$, widely used in the text classification, reduces to private evaluation of shared scalar product for which several solutions are known [4, 15].

Higher-degree polynomial kernels $\kappa(\vec{x}_i,\vec{x}_j) = \langle\vec{x}_i,\vec{x}_j\rangle^d$ can be efficiently evaluated using share conversion: first the additive shares $s_1 + s_2 = \langle\vec{x},\vec{y}\rangle \mod N$ are computed, then the shares are converted to multiplicative shares $t_1 \cdot t_2 = \langle\vec{x},\vec{y}\rangle \mod N$ and finally the exponentiated shares are converted back $u_1 + u_2 = t_1^d \cdot t_2^d = \langle\vec{x}_i,\vec{x}_j\rangle^d \mod N$. These share conversions are straightforward to implement with homomorphic encryption. (See the full version [7] for further discussion.) Compared with other methods, the computational workload and communication are small, as the exponentiation is done locally.

Share manipulation requires that $\langle\vec{x}_i,\vec{x}_j\rangle$ and N are coprime, since otherwise multiplicative sharing modulo N does not exist. Because homomorphic encryption forces the use of N with nontrivial factors that are at least 512-bit integers, then it is sufficient that $\langle\vec{x}_i,\vec{x}_j\rangle \neq 0$ for all "reasonable" input ranges $\mathcal{X}$. For many interesting cases, $z_1,\ldots,z_m \geq 0$ for all $\vec{z} \in \mathcal{X}$ and a kernel $\kappa(\vec{x}_i,\vec{x}_j) = (\langle\vec{x}_i,\vec{x}_j\rangle + 1)^d$ can be used instead. Finally, if $\langle\vec{x}_i,\vec{x}_j\rangle = 0$ then one can escape the problem by remapping the shares of 0 to shares of a special symbol $\zeta \in \mathbb{Z}_N^*$, and then later mapping the shares of ζ^d back to shares of 0. This requires costly circuit evaluation and should be avoided if possible.

5. PRIVATE PREDICTION

Private prediction has several interesting applications even if the classifier is directly provided by Client, e.g., in finding potential patients without revealing private medical data. Then Client has to send encrypted weight vector $\mathrm{E}_{\mathsf{pk}}(\vec{\alpha}) = (\mathrm{E}_{\mathsf{pk}}(\alpha_1),\ldots,\mathrm{E}_{\mathsf{pk}}(\alpha_m))$ to Server before the protocol. For brevity, denote $\vec{\kappa} := (\kappa(\vec{x}_1,\vec{x}),\ldots,\kappa(\vec{x}_n,\vec{x}))$, where $\vec{\kappa}$ has integer coordinates. Then $f_{\vec{\alpha}}(\vec{x}) = \alpha_1\kappa_1 + \cdots + \alpha_n\kappa_n$.

A private prediction protocol that works in the case of restricted vertical split is depicted by Protocol 1. There, the parties first privately compute the additive shares of a scalar product and then use circuit evaluation to determine the shares of class label. Note that Prot. 1 can be modified so that Client learns the predicted label.

THEOREM 1. *Assume that* Π *is an IND-CPA secure additively homomorphic cryptosystem and that the circuit evaluation step is private. Then Protocol 1 is correct and private.*

Protocol 1 Private prediction for restricted split

```
Common parameters: $\Pi$ with plaintext space $\mathbb{Z}_N$.
Inputs: Client has a secret key sk. Server has the public
        key pk, feature vectors $\vec{x},\vec{x}_1,\ldots,\vec{x}_n$, vector $\vec{\kappa}$, and
        encrypted weight vector $\mathrm{E}_{\mathsf{pk}}(\vec{\alpha})$.
Output: Client and Server share a predicted class label.

1. Server sends $c \leftarrow \mathrm{E}_{\mathsf{pk}}(-s_2)\cdot\prod_{j=1}^{n}\mathrm{E}_{\mathsf{pk}}(\alpha_j)^{\kappa_j}$, for $s_2 \leftarrow \mathbb{Z}_N$.
   Client sets $s_1 \leftarrow \mathrm{D}_{\mathsf{sk}}(c)$. // I.e., they share $s_1 + s_2 = \langle\vec{\alpha},\vec{\kappa}\rangle$.
2. Client and Server use circuit evaluation to share
   $t_1 + t_2 = \mathrm{sign}(s_1 + s_2) \mod N$.
```

Recall that in the general case vector $\vec{\kappa}$ is additively shared between Client and Server, i.e., $\vec{\kappa} = \vec{\kappa}_1 + \vec{\kappa}_2 \mod N$ where $\mathbb{Z}_N$ is the plaintext space. Hence, given $\mathrm{E}_{\mathsf{pk}}(\vec{\alpha})$ both parties can compute $\mathrm{E}_{\mathsf{pk}}(\langle\vec{\kappa}_i,\vec{\alpha}\rangle)$ similarly to Protocol 1. However, neither of them can have secret key sk or otherwise $\vec{\alpha}$ or $\vec{\kappa}_i$ leaks out. Therefore, one needs a two-party version of additive homomorphic encryption scheme [3] where parties can only together decrypt values. Essentially, parties have to execute two copies of Protocol 1 with switched identities to share sign $f_{\vec{\alpha}}(\vec{x})$. The corresponding protocol along with the security proof is present in the full version of the paper [7].

Targeted optimizations.

Protocol 1 relies on circuit evaluation. We can use two-round COT protocol (described in Section 3) to evaluate say the "greater than" predicate, but additional share conversion can significantly increase the efficiency. For example, to guarantee the security of homomorphic encryption, N must usually be at least a 1024-bit integer. On the other hand, if we use 64-bit precision for $\vec{\kappa}$ and $\vec{\alpha}$ then the shared values fit roughly into 140 bits. Hence, it is advantageous to convert random shares $s_1 + s_2 = x \mod N$ to random shares $r_1 + r_2 = x \mod M$ where M is significantly smaller, say $M = 2^{140}$.

For clarity, Protocol 2 is depicted for the representation $\mathbb{Z}_N = \{0,\ldots,N-1\}$. The same result applies for the signed representation $\mathbb{Z}_N = \{-L,\ldots,L\}$ where $N = 2L+1$. If $M < N$ and in the signed representation $-\frac{M}{4} < x < \frac{M}{4}$, then $0 \leq \lfloor\frac{M}{4}\rfloor + x < \frac{M}{2}$, and the parties can directly apply Protocol 2 and then subtract the *public* value $2\cdot\lfloor\frac{M}{4}\rfloor$ from the result. Similar techniques can be used for $M > N$.

Protocol 2 Share conversion algorithm.

```
Input: Additive shares $s_1 + s_2 = x \mod N$, $N$ is odd.
Output: Additive shares $r_1 + r_2 = 2x \mod M$.
We assume $\mathbb{Z}_N = \{0,\ldots,N-1\}$, $0 \leq x < \frac{M}{2}$ and $M < N$.

1. Parties locally compute $t_i \leftarrow 2s_i \mod N$, $i = \{1,2\}$.
2. Server prepares an OT-table $(m_0,m_1)$ for $r_2 \leftarrow \mathbb{Z}_M$:
   a) If $t_2$ is even then
      $m_0 \leftarrow t_2 - r_2 \mod M$ and $m_1 \leftarrow t_2 - r_2 - N \mod M$.
   b) If $t_2$ is odd then
      $m_0 \leftarrow t_2 - r_2 - N \mod M$ and $m_1 \leftarrow t_2 - r_2 \mod M$.
3. Client uses a 1-out-of-2 OT protocol to set $r_1 \leftarrow m_b + t_1$
   where $b$ denotes the parity of $t_1$.
```

THEOREM 2. *Protocol 2 is private and correct, provided that the oblivious transfer protocol is private and correct,* N *is odd,* $0 \leq x < \frac{M}{2}$ *and* $M < N$.

The correctness of Prot. 2 is clear as $t_1 + t_2 = 2x \mod N$. Thus if $0 \leq t_1 + t_2 < N$ then both t_1 and t_2 are either odd or even. If $n \leq t_1 + t_2 < 2N$ then t_1 and t_2 have different parity. Hence, $r_1 + r_2 = 2x \mod N$. Security follows from the composition theorem.

If $r_1 + r_2 = 2x \mod 2^\ell$ then the sign of x is determined by the highest bit of the sum and latter can be evaluated using ℓ ternary gates. Hence, it is advantageous to use Protocol 2 to reduce the input size of garbled circuit. As a result, Step 2 can be implemented with ℓ ternary gates for Protocol 1 and the size of the garbled circuit is roughly $\mathcal{O}(\ell)$. Moreover, we need only $\ell + 1$ invocations of OT counting also the one needed for share conversation. The communication and computation costs decrease at least by a factor of 10.

Secure Aggregation. Note that if Client and Server locally add together shares of different class labels, they can straightforwardly count the number of positive examples. Recall that the class labels are ± 1, hence the sum of shares reveals difference between positive and negative examples. Moreover, due to the properties of the COT protocol (Section 3), all shares can be computed in parallel by first running Step 1 in Prot. 1 for all feature vectors and then execute Step 2. The resulting protocol takes four rounds, i.e., all protocol messages can be combined into four larger ones.

One can straightforwardly modify Protocol 1 so that the parties obtain the shares $t_1 + t_2 = 0 \mod N$, if predicted value corresponds to the true label y, and 1, otherwise. Then the sum of the shares counts the number of misclassified data points and we can privately estimate training and validation error or even do private cross-validation.

Stopping criterion and KKT violators. Protocol 1 can be extended to count the number of Karush-Kuhn-Tucker violators. Recall that a feature vector $\vec{x}_i$ is a KKT violator if one of the next three conditions does not hold:

$$\begin{aligned} \alpha_i = 0 &\Leftrightarrow f_{\vec{\alpha}}(\vec{x}_i) y_i \geq 1 \ , \\ 0 < \alpha_i < C &\Leftrightarrow f_{\vec{\alpha}}(\vec{x}_i) y_i = 1 \ , \\ \alpha_i = C &\Leftrightarrow f_{\vec{\alpha}}(\vec{x}_i) y_i \leq 1 \ . \end{aligned}$$

Circuit for detecting the KKT violators has $\mathcal{O}(\ell)$ ternary gates. The number of the KKT violators is often used as an indicator for stopping: algorithm has converged if there are no KKT violators. Alternatively, one can stop if the number of the KKT violators is below some threshold or has not significantly changed during several iterations. However, private counting of the KKT violators or training error is resource consuming, and should be done after several iterations of the Kernel Adatron or Perceptron algorithm.

6. PRIVATE TRAINING ALGORITHMS

Private training algorithms have the same structure as private prediction algorithms. Whenever possible, we use homomorphic properties of the cryptosystem to compute shares directly. If this is not possible, we use circuit evaluation to circumvent the problem. Protocol 3, presented next, is private in the sense that Client and Server learn nothing except the number of iterations. Learning the latter is unavoidable in practice, since the amount of computations always provides an upper bound to the number of iterations. One can achieve better privacy by doing extra rounds but this would seriously affect the efficiency.

Due to the space limitations, we present explicitly only a secure analog of Algorithm 1, depicted by Protocol 3. The corresponding secure protocol for the Kernel Adatron algorithm has the same structure. We explain only how Step 2 is implemented, the rest is the same as in Prot. 3.

Protocol 3 Private Kernel Perceptron

Common parameters: Π with plaintext space $\mathbb{Z}_N$.
Inputs: Client has a secret key sk and labels $\vec{y}$. Server has the public key pk and vectors $\vec{x_1}, \ldots, \vec{x}_n$.
Server's output: An encrypted weight vector $\vec{c} = \mathrm{E}_{\mathsf{pk}}(\vec{\alpha})$.
Allowed side information: the number of iterations.

```
1. Server sets $\vec{c} = \mathrm{E}_{\mathsf{pk}}(\vec{0})$.
2. Client and Server execute the next cycle:
   for $i = 1$ to $n$ do
     a) They compute shares $s_1 + s_2 = f_{\vec{\alpha}}(\vec{x_i}) \mod N$.
     b) They use circuit evaluation to compute shares
        $t_1 + t_2 = \begin{cases} y_i, & \text{if } y_i(s_1+s_2) \leq 0 \\ 0, & \text{if } y_i(s_1+s_2) > 0 \end{cases} \mod N$ .
     c) Client sends $d = \mathrm{E}_{\mathsf{pk}}(t_1)$, Server sets $c_i \leftarrow c_i d \cdot \mathrm{E}_{\mathsf{pk}}(t_2)$.
   end for
3. If not converged then repeat Step 2.
```

Theorem 3. *Protocol 3 is a correct and private implementation of the kernel Perceptron algorithm (Algorithm 1) provided that (1) the cryptosystem is additively homomorphic and IND-CPA secure; (2) all substeps are implemented correctly and privately; (3) the constraints* $|f_{\vec{\alpha}}(x_i)| < \frac{N}{2}$ *and* $|\alpha_i| < \frac{N}{2}$ *always hold.*

Correctness follows as Substep 2b) implements incremental update $c_i = \mathrm{E}_{\mathsf{pk}}(\alpha_i + t_1 + t_2 \mod N) = \mathrm{E}_{\mathsf{pk}}(\alpha_i + y_i)$ if $\vec{x}_i$ is incorrectly classified. Since N is at least 1024 bits long, $|\alpha_i| \ll N/2$ for all iterations. Similarly, there are no overflows in computation of $f_{\vec{\alpha}}(\vec{x}_i)$ provided that kernel matrix has reasonable discretization.

The update step of the Kernel Adatron algorithm can be restated as $\beta_i \leftarrow \beta_i + y_i - f_{\vec{\beta}}(\vec{x}_i)$, where $\vec{\beta} = (\alpha_i y_i)_{i=1}^n$ and $f_{\vec{\beta}}(\vec{x}_i) = k_{i1}\beta_1 + \cdots + k_{in}\beta_n$. The corresponding correction Step 5 in Algorithm 2 implements the constraint $0 \leq y_i \beta_i \leq C$. Hence, Client and Server can still use private prediction to compute shares $s_1 + s_2 = \beta_i + y_i - f_{\vec{\beta}}(\vec{x}_i) \mod N$. Then the correction step must be done with circuit evaluation

$$t_1 + t_2 = \begin{cases} 0, & \text{if } y_i(s_1+s_2) < 0 \\ y_i C, & \text{if } y_i(s_1+s_2) > C \\ s_1 + s_2, & \text{otherwise} \end{cases} \mod N \ .$$

Finally, Server computes $\mathrm{E}_{\mathsf{pk}}(\beta_i)$ as $\mathrm{E}_{\mathsf{pk}}(t_1)\mathrm{E}_{\mathsf{pk}}(t_2)$. It can be shown that correction step can be implemented with $2\ell + 1$ ternary gates. Thus, the size of the garbled circuit is roughly $\mathcal{O}(2\ell)$ for both the Kernel Perceptron and Kernel Adatron. The parties have to do $\ell + 1$ invocations of OT counting also the one needed for share conversation.

Batch processing. Both algorithms are instances of stochastic gradient descent method, as the update changes a single coordinate of $\vec{\alpha}$. Alternatively, one can use a full gradient descent step instead, i.e., compute all values $f_{\vec{\alpha}}(\vec{x}_i)$ simultaneously and the update all coordinates of $\vec{\alpha}$ also simultaneously. Such batch updates tend to stabilize gradient descent methods but they also decrease the number of rounds, i.e., latency. Due to the properties of COT protocol, Substeps 2a) and 2b) can be executed in parallel and the number of rounds decreases from $6n$ to 6 per iteration.

7. CONCLUDING REMARKS

We have described cryptographically secure protocols for Kernel Perceptron and kernelized Support Vector Machines. We have also provided cryptographically secure protocols for evaluating polynomial kernels and also shown how to securely aggregate encrypted classification results.

An interesting open question is how to securely hide the convergence speed of the Kernel Perceptron and the Kernel Adatron algorithms. Recall that our private implementations did not leak anything but the number of rounds.

Another, more practical, question is whether there are any iterative private linear classification methods that need no costly circuit evaluation. The Widrow-Hoff classification algorithm is a good candidate, as it contains only addition and multiplication operations. Unfortunately, there one has to also round the values, so it is not clear whether one can escape circuit evaluation.

The proposed classification and classifier learning protocols are not limited to data represented as feature vectors, but can be used on any data with secure kernel evaluation. Hence, another relevant issue is private computation of encrypted kernel matrices for structured data.

Acknowledgments

We thank Matti Kääriäinen, Juho Rousu and Sandor Szedmak for valuable discussions on the nature of SVMs and the current state of the art in kernel methods. The first author was supported by Finnish Academy of Sciences, and by Estonian Doctoral School in Information and Communication Technologies. The second author was supported by the Estonian Science Foundation, grant 6848. The third author was supported by the European Union IST programme, contract no. FP6-508861, *Application of Probabilistic Inductive Logic Programming II.*

8. REFERENCES

[1] Aiello, W., Ishai, Y., and Reingold, O. Priced Oblivious Transfer: How to Sell Digital Goods. In *Advances in Cryptology — EUROCRYPT 2001*, vol. 2045 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 119–135.

[2] Chang, Y.-C., and Lu, C.-J. Oblivious Polynomial Evaluation and Oblivious Neural Learning. In *Advances on Cryptology — ASIACRYPT 2001*, vol. 2248 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 369–384.

[3] Damgård, I., and Jurik, M. A Generalisation, A Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In *Public Key Cryptography 2001*, vol. 1992 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 119–136.

[4] Goethals, B., Laur, S., Lipmaa, H., and Mielikäinen, T. On Private Scalar Product Computation for Privacy-Preserving Data Mining. In *Information Security and Cryptology - ICISC 2004*, vol. 3506 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 104–120.

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

[6] Kantarcioglu, M., and Clifton, C. Privately Computing A Distributed k-nn Classifier. In *PKDD* (2004), vol. 3202 of *LNCS*, Springer, pp. 279–290.

[7] Laur, S., Lipmaa, H., and Mielikäinen, T. Cryptographically Private Support Vector Machines. Tech. rep. 2006/198, International Association for Cryptologic Research, 2006. Available at http://eprint.iacr.org/2006/198.

[8] Lindell, Y., and Pinkas, B. Privacy Preserving Data Mining. *Journal of Cryptology 15*, 3 (2002), 177–206.

[9] Lipmaa, H. An Oblivious Transfer Protocol with Log-Squared Communication. In *The 8th Information Security Conference (ISC'05)*, vol. 3650 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 314–328.

[10] Malkhi, D., Nisan, N., Pinkas, B., and Sella, Y. Fairplay - Secure Two-Party Computation System. In *Proceedings of the 13th USENIX Security Symposium*, USENIX, pp. 287–302.

[11] Naor, M., Pinkas, B., and Sumner, R. Privacy Preserving Auctions and Mechanism Design. In *The 1st ACM Conference on Electronic Commerce*, 1999.

[12] Paillier, P. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology — EUROCRYPT '99*, vol. 1592 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 223–238.

[13] Shawe-Taylor, J., and Cristianini, N. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[14] Vapnik, V. N. *The Nature of Statistical Learning Theory*. Statistics for Engineering and Information Science. Springer, 2000.

[15] Wright, R. N., and Yang, Z. 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*, ACM, pp. 713–718.

[16] Yang, Z., Zhong, S., and Wright, R. N. Privacy-preserving classification of customer data without loss of accuracy. In *SDM* (2005).

[17] Yu, H., Jiang, X., and Vaidya, J. Privacy Preserving SVM Using Secure Set Intersection Cardinality. In *The 21st ACM Symposium on Applied Computing*, ACM 2006.

[18] Yu, H., Vaidya., J., and Jiang, X., Privacy Preserving SVM Classification on Vertically Partitioned Data, In *PAKDD 2006*, Springer-Verlag 2006.