Cryptographically Private Support Vector Machines*

Sven Laur[1], Helger Lipmaa[2,3], and Taneli Mielikäinen[4]

[1] Helsinki University of Technology
[2] Cybernetica AS
[3] University of Tartu
[4] HIIT

Abstract. We study the problem of private classification using kernel methods. More specifically, 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 also 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.
Keywords: Privacy Preserving Data Mining, Kernel Methods

1 Introduction

Currently, data mining is playing an important role as vast amounts of data are being collected in various areas of science, finance, industry and government [HMS01]. In many applications, the data repositories are becoming increasingly shared by several parties, which gives rise of need for privacy-preserving data mining (PPDM) protocols that involve two or more parties.

We are interested in private classification. 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. 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"/"nospam". More precisely, the training data is constructed by converting the classified emails to word count vectors and then the classifier is learned from this training data. The classifier could 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.

Linear classifiers have been central in classifier learning since the introduction of Rosenblatt's Perceptron algorithm [Vap00]. Important recent improvements in learning linear classifiers have been maximum margin classifier learning and kernel methods, improving the classification results in theory and practice, and extending the applicability of the linear classification techniques to virtually any kind of data [STC04]. Support

* Short version published in KDD 2006

Vector Machines (SVMs) [Vap00] are the most well-known kernelized maximum margin classifiers. The Kernel Perceptron algorithm is a kernelized version of the classical Perceptron algorithm and the Kernel Adatron is a kernelized version of a maximum margin linear classifier. In Sect. 2, we provide a short introduction to linear classifiers, Support Vector Machines and private classification.

There are two fundamentally different ways a protocol run can disclose sensitive information: (a) the protocol can leak some side information that is not specified by protocol's functionality, and (b) the specified protocol result itself can reveal sensitive aspects of the data. Thus, it should first be stated which information can be revealed. Second, a concrete PPDM protocol should be implemented so as not to reveal more information than required.

By the classical results of cryptography, by using secure multi-party computation [Gol04], any protocol can be implemented without leakage of side information (with "polynomial" slowdown). Thus, secure multi-party computation methods can be applied to any data mining protocol to solve the first mentioned issue. For many practical protocols, applying general secure multi-party computation results however in very inefficient protocols. 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.

What about the second issue, data mining result revealing sensitive information? Clearly, there is a tradeoff between revealing sensitive information and useful knowledge: in the case of data mining, one can avoid disclosing of sensitive information by not mining the data at all, but then also one discovers no useful knowledge.

Following these general principles, also a classification protocol—that consists of the learning and prediction protocols—can reveal (a) internal values, and (b) the classifier and classification results. Any of the revealed internal values can breach the privacy of the training data, and the output values can also breach the privacy of the classified data. Thus, it is important to define what additional information should the participants get after running a classification protocol. In private classification, only an authorized party should get new information, and this information should only consist of the class labels of the classified data. Moreover, if the classification results are too sensitive then the authorized party should obtain only an aggregation of multiple classifications. In Sect. 3, we describe the cryptographic tools that we need in the rest of the paper.

Our contributions In the current paper, we describe private protocols for kernel sharing, prediction and training. Although the new protocols are specified for vector data, they are also applicable for structured data (e.g. strings), provided that the kernel values can be securely computed.

All our protocols are provably private in the standard semi-honest model, where it is assumed that the participants follow the protocol but use the obtained information to gain new knowledge. Moreover, the new protocols neither leak information about the training data nor about the classifiers, since the data and the learned classifier will stay encrypted or more precisely, private but available for further processing. Privacy in the malicious model, where one is concerned about arbitrary adversaries, can be achieved by using standard cryptographic techniques; however, the resulting protocols will often be very inefficient in practice. Hence, we will consider only the semi-honest model.

On the other hand, we consider privacy in the cases when the data is divided horizontally or vertically between two parties. These cases are conceptually different and require different protocols.

In Sect. 4, we first show how to privately share the kernel matrix between Client and Server. At the end of the sharing, they have random (but dependent) values that make it later possible for them to engage in private prediction and training protocols. The corresponding kernel sharing protocol is very efficient. In Sect. 5, we then consider private prediction that is an important task by itself. For example, one can use private prediction to securely filter out relevant documents from a data-stream, instead of using private filtering based on triggering keywords [OI05] and thus get significantly more precise results. In addition, we discuss how to privately implement cross-validation and other aggregation methods over classification results. In Sect. 6, we propose private versions of the Kernel Perceptron and the Kernel Adatron protocols, that both update the weight vector by using the private prediction protocols of previous section. We prove that these protocols only leak the number of rounds (and information that can be deduced from it). In Sect. 7, we conclude the paper and discuss some open problems.

Related work The research on private (or "privacy-preserving") classification techniques has resulted protocols for learning Naïve Bayes Classifiers [VC04,WY04,YZW05], Decision Trees [AS00,BDMN05,DZ03,LP02], k-Nearest Neighbors Classifiers [CL05,KC04] and Linear Discriminant Analysis [DHC04]. A cryptographically private protocol for non-linear classification with the feed-forward neural networks was proposed in [CL01], however, this solution is too resource demanding to be practical. On the other hand, linear classifier learning using the Perceptron algorithm on randomly perturbed data has led to more practical methods [BDMN05,CL05], but their notions of privacy are more liberate than ours; namely, they guarantee only a certain amount of uncertainty about the private inputs.

A few methods for privacy-preserving learning of Support Vector Machines have been proposed [CL05,YJV06] but they reveal the kernel and the Gramm matrix of the data, that then can be easily used to expose a lot of information about the data. Recall that the Gramm matrix consists of scalar products between all data vectors. Now, 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 possess complete feature vectors. Moreover, other kernel methods like the Kernel-PCA reveal partial information data points without any auxiliary knowledge beyond the kernel matrix.

2 Classification: Preliminaries

2.1 Overview of classification methods

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 at random from the same 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 $g_* \in \mathcal{G}$. Typically the best classifier is considered to be the one with the smallest expected error. The expected error of a classifier $g \in \mathcal{G}$ is $\Pr[g(X) \neq Y]$, where X and Y are random variables with unknown joint probability distribution over $\mathcal{X} \times \mathcal{Y}$. As the probability distribution is unknown, we must relay on an empirical error estimate $\sum_{(x_i, y_i) \in \mathcal{D}} \mathbb{I}[g(x_i) \neq y_i]$ instead, where $\mathbb{I}[\cdot]$ is a zero-one indicator function.

The family of linear functions is particularly important in machine learning. Linear functions can be conveniently described by the normals of the hyperplanes $\boldsymbol{w} \in \mathbb{R}^m$. The classification of a point $\boldsymbol{x} \in \mathbb{R}^m$ is then determined by the sign of the scalar product $f_{\boldsymbol{w}}(\boldsymbol{x}) := \langle \boldsymbol{w}, \boldsymbol{x} \rangle$ that is also known as the *discriminative function*. The value of $f_{\boldsymbol{w}}(\boldsymbol{x})$ characterizes the classification certainty.

Linear classification has been in a central role in the theory and practice of machine learning for more than 40 years since the introduction of the Rosenblatt's Perceptron algorithm and Novikoff's convergence results for the Perceptron algorithm [Vap00]. In addition to the applications in machine learning, the Perceptron algorithm has an independent interest as a simple method for linear programming [BD02].

The idea of the Perceptron algorithm is to find a linear combination $\boldsymbol{w}$ of the points $\boldsymbol{x}_i$ such that $\operatorname{sign}\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle = y_i$ for all $i = 1, \ldots, n$. The algorithm updates the weight vector $\boldsymbol{w}$ (initially $\mathbf{0}$) by adding to $\boldsymbol{w}$ each data point $\boldsymbol{x}_i$ that is misclassified by the current $\boldsymbol{w}$. (See [STC04,Vap00] 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 $\boldsymbol{w} \in \mathbb{R}^m$ that separates the positive examples from the negative ones. A natural solution to this problem is to map the data into a suitable Hilbert space $\mathcal{H}$ (i.e., a complete inner product space; see [STC04]) 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*.

A potential problem with this approach is that the space where the data is linearly separable can have very high, even infinite dimensionality. Sometimes this problem can be avoided by the use of kernels. A kernel of a feature map ϕ is a function κ such that $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$ for all $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{X}$; it is used to obtain the implicit description $\boldsymbol{\alpha}$ of the weight vector $\boldsymbol{w}$. The use of kernels in data analysis has become a standard practice, often providing state of the art data mining methods [STC04]. The Perceptron algorithm can also be written using kernels, by observing that the weight vector $\boldsymbol{w}$ can be written as $\sum_{j=1}^n \alpha_j \phi(\boldsymbol{x}_j)$ for some $\boldsymbol{\alpha} \in \mathbb{Z}^n$. Then

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

i.e., it suffices to compute the kernel values $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for all $(\boldsymbol{x}_i, y_i), (\boldsymbol{x}_j, y_j) \in \mathcal{D}$. Furthermore, the values $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ have to be computed only once for a particular $\mathcal{D}$ and ϕ. The kernel values of data $\mathcal{D}$ are often represented as a kernel matrix $K \in \mathbb{R}^{n \times n}$, where $k_{ij} = \kappa(x_i, x_j)$. The Perceptron algorithm using a kernel matrix—the Kernel Perceptron algorithm—is given as Algorithm 1. (See [STC04].)

Algorithm 1 Kernel Perceptron algorithm

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

Function KERNEL-PERCEPTRON($K$, $\boldsymbol{y}$)
 1: $\boldsymbol{\alpha} \leftarrow \mathbf{0}$
 2: repeat
 3:     for $i = 1, \dots, n$ do
 4:         if $y_i \cdot \sum_{j=1}^{n} k_{ij}\alpha_j \leq 0$ then $\alpha_i \leftarrow \alpha_i + y_i$
 5:     end for
 6: until convergence
end function
```

By Novikoff's Theorem [Vap00], the number of misclassifications (i.e., the number of the weight updates) can be bounded above by 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 $\boldsymbol{w}$ w.r.t. the dataset $\mathcal{D}$ is defined as

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

and $\gamma_* = \max\{\gamma(\boldsymbol{w}) : \boldsymbol{w} \in \mathbb{R}^m\}$. Novikoff's Theorem emphasizes also the following shortcoming of the Perceptron algorithm: the Perceptron algorithm finds some separating hyperplane for data (if it exists), but basically any separating hyperplane will do. In particular, the Perceptron algorithm can return different separating hyperplanes for different permutations of data. A natural way to select the unique separating hyperplane is to pick the one that maximizes the margin γ, i.e., the maximum margin hyperplane $\boldsymbol{w}_*$. Intuitively, the maximum margin hyperplane aims minimizing the risk of misclassification with unknown data distribution and the maximum margin hyperplane is justified also by the generalization error bounds [STC04,Vap00].

There exist many variants of the Perceptron algorithm that maximize the margin [FS99,FCC98,STC04]. Learning algorithms finding a maximum margin separating hyperplane are called Support Vector Machines (SVM-s in short) [Vap00]. A particularly flexible and simple Support Vector Machine is the Adatron algorithm [STC04], depicted by Algorithm 2. 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 as 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.

A hard margin SVM finds the separating hyperplane that maximizes the margin γ, given that the data 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. Therefore, it is advantageous to use soft margin SVM-s that bound the disturbances caused by potential outliers. Soft margin SVM-s come in two flavors: either a constraint $\alpha_j \in [0, C]$ is added for each $i \in \{1, \ldots, n\}$, or a regularizing term $C > 0$ is added to the main diagonal of the kernel matrix. The first gives a rise to ℓ_1-norm and the second to ℓ_2-norm soft margin SVM [STC04]. Algorithm 2 implements the ℓ_1-norm soft margin SVM, which is the most popular SVM. We get the a margin SVM by setting C to be ∞.

Algorithm 2 Kernel Adatron algorithm

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

Function KERNEL-ADATRON($K$, $y$, $C$)
 1: $\boldsymbol{\alpha} \leftarrow \mathbf{0}$
 2: repeat
 3:     for $i = 1, \ldots, n$ do
 4:         $\alpha_i \leftarrow \alpha_i + y_i \cdot \left(1 - y_i \cdot \sum_{j=1}^{n} k_{ij}\alpha_j y_j\right)$
 5:         if $\alpha_i > C$ then $\alpha_i \leftarrow C$
 6:         if $\alpha_i < 0$ then $\alpha_i \leftarrow 0$
 7:     end for
 8: until convergence
end function
```

2.2 Private classification

We assume that the sample data is divided between two parties with possibly conflicting interests and in particular, that they are not willing to reveal their data. On the other hand, we assume that the parties are still 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 [OI05] and identity-specific content providing services. There is a conceptual distinction between the cases where the data is owned by two or more parties. In the multi-party scenario, it is traditional in cryptography to assume the existence of a honest majority. In this case, there are relatively efficient solutions for arbitrary private computation [BOGW88,CCD88,GRR98]. In the two-party scenario, one cannot assume the existence of ha honest majority (unless both parties are honest, which makes the problem trivial) and then one has to use more elaborate and time-consuming methods to achieve security. In this paper, we consider the conceptually more difficult two-party case.

There are three basic scenarios depending on how the feature vectors are divided between parties: restricted and general vertical split and horizontal split. In the case of *restricted vertical split*, the data is divided between Client and Server so that Client possesses the label vector $\boldsymbol{y}$ and Server has the corresponding feature vectors $\boldsymbol{x}_i$. As the vectors $\boldsymbol{x}_i$ correspond to the real life objects, it is quite plausible that Client can still classify the objects although the features $\boldsymbol{x}_i$ are not known. This means that the requirements are not too restrictive. In the case of *general vertical split*, Client and Server possess different coefficient of feature vectors $\boldsymbol{x}_i$. The *horizontal split* means that different parties own different feature vectors.

Data sharing model determines the complexity of different classification phases, *kernel matrix computation*, *prediction* and *training*. Restricted vertical split is the simplest case: in this case, the kernel matrix computation is trivial as Server owns all feature vectors. In other cases, Client and Server must additionally use cryptographic methods to share K so that neither of them learns nothing about K. This complicates also the private training and prediction algorithms.

In the training phase, parties privately update similarly shared weight vector $\boldsymbol{\alpha}$. Finally, parties use shared knowledge of $\boldsymbol{\alpha}$ to securely predict labels, i.e., to privately compute $\operatorname{sign} f_{\boldsymbol{w}}(\boldsymbol{x})$. Note that if the predicted labels leak and Server possesses the corresponding vectors, then he can train a new approximate classifier and privacy is compromised, however, corresponding countermeasures are out of our scope.

3 Cryptographic tools

In the current paper, we introduce 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.

Notation For a distribution X, let $x \leftarrow X$ denote the assignment of x according to X. We often identify sets with the uniform distributions on them, and algorithms with their output distributions, assuming that the algorithm that outputs this distribution is clear from the context or just straightforward to construct. Let k be the security parameter. 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 $c > 0$. A function $f(k)$ is $\mathsf{poly}(k)$ if $f(k)$ has a polynomial bound.

3.1 Formal security model

Let Π^f denote a well-specified distributed algorithm (*protocol*) 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 usually formalized by using the non-uniform polynomial security model [Gol04, p. 620–624, 626–631]. A protocol is *private* if any probabilistic polynomial-time honest-but-curious adversary (that follows the protocol) can succeed in obtaining any additional information with a negligible probability in 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 Appendix A for further references.

Cryptographic security proofs are often very technical. The next (sequential) composition property allows to simplify 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 [Gol04, p. 637] holds.

Composition Theorem 1 *Let protocols $\Pi^{g|f}$ and Π^f be a 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 since otherwise the composition theorem might not hold.

Now, we can formally specify security goals for private classification. In the matrix evaluation and training phase, Client and Server must learn nothing new, i.e., $f_i = \perp$. In the prediction phase, Client must learn only the predicted label $f_1 = f_{\boldsymbol{\alpha}}(\boldsymbol{x})$ and Server must learn nothing.

3.2 Homomorphic encryption

A public-key cryptosystem is a triple $\mathit{\Pi} = (\mathrm{G}, \mathrm{E}, \mathrm{D})$, 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. For a fixed secret key sk, 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 for all valid sk, $x \in \mathcal{P}$ and $r \in \mathcal{R}$, $\mathrm{D}_{\mathsf{sk}}(\mathrm{E}_{\mathsf{pk}}(x; r)) = x$.

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:

$$\mathrm{Adv}(A) = \left|\Pr\left[A = 1 \middle| \mathsf{EXP}_0\right] - \Pr\left[A = 1 \middle| \mathsf{EXP}_1\right]\right| \enspace .$$

Here, the probability is taken over the random choices of G, E and A. The next straightforward result allows to simplify some of the following security proofs.

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.*

A cryptosystem is (additively) homomorphic if for all valid key pairs $(\mathsf{sk}, \mathsf{pk})$, messages $x_1, x_2 \in \mathcal{P}$ and random nonces $r_1, r_2 \in \mathcal{R}$,

$$\mathrm{E}_{\mathsf{pk}}(x_1; r_1) \cdot \mathrm{E}_{\mathsf{pk}}(x_2; r_2) = \mathrm{E}_{\mathsf{pk}}(x_1 + x_2; r_1 \circ r_2) \enspace ,$$

where $+$ is a group operation in $\mathcal{P} = \mathbb{Z}_N$, for a large integer N, and $\circ$ is such that $\mathcal{R} = r \circ \mathcal{R} = \mathcal{R} \circ r$ for any r (this enables rerandomization of ciphertexts). Several homomorphic cryptosystems (e.g., [DJ01,Pai99]) are proven to be IND-CPA secure under reasonable complexity assumptions. For such cryptosystems, we can *compute on ciphertexts* by using the following three identities:

$$\begin{aligned} \mathrm{E}_{\mathsf{pk}}(x) \cdot \mathrm{E}_{\mathsf{pk}}(y) &= \mathrm{E}_{\mathsf{pk}}(x + y \mod N) \quad \forall x, y \in \mathcal{P} \enspace , \\ \mathrm{E}_{\mathsf{pk}}(x)^y &= \mathrm{E}_{\mathsf{pk}}(x \cdot y \mod N) \quad \forall x \in \mathcal{P}, \forall y \in \mathbb{Z} \enspace , \\ \mathrm{E}_{\mathsf{pk}}(x; r) \cdot \mathrm{E}_{\mathsf{pk}}(0) &= \mathrm{E}_{\mathsf{pk}}(x) \quad \forall x \in \mathcal{P}, \forall r \in \mathcal{R} \enspace . \end{aligned}$$

We need a two-party version of such cryptosystems, where the secret key is divided into two subkeys sk_1 and sk_2, such that there is a symmetric decryption algorithm D with

$$\mathrm{D}_{\mathsf{sk}_1}(\mathrm{D}_{\mathsf{sk}_2}(\mathrm{E}_{\mathsf{pk}}(m))) = \mathrm{D}_{\mathsf{sk}_2}(\mathrm{D}_{\mathsf{sk}_1}(\mathrm{E}_{\mathsf{pk}}(m))) = m \enspace ,$$

but where the knowledge of a single key sk_i does not help to distinguish between $\mathrm{E}_{\mathsf{pk}}(m_0)$ and $\mathrm{E}_{\mathsf{pk}}(m_1)$. Such *two-party* homomorphic cryptosystems were proposed by Damgård and Jurik [DJ01]. In the semi-honest case, key-generation becomes more complicated, encryption remains unaltered and decryption becomes only twice more resource-consuming compared to the original cryptosystem.

3.3 Secret sharing and share conversion

Algorithms 1 and 2 contain variables that 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 modulus. If (s_1, s_2) are chosen uniformly from the set

$$\bigl\{(s_1, s_2) \in \mathbb{Z}_N^2 : s_1 + s_2 = x \mod N\bigr\}$$

then the knowledge of s_i reveals nothing about x, as s_i has uniform distribution. We call it the *additive sharing* of x. One can similarly define *multiplicative sharing* for invertible elements $\mathbb{Z}_N^* = \{a \in \mathbb{Z}_N : a \cdot b = 1 \mod N\}$ by using the set

$$\bigl\{(s_1, s_2) \in (\mathbb{Z}_N^*)^2 : s_1 \cdot s_2 = x \mod N\bigr\} \,.$$

3.4 Conditional oblivious transfer

To efficiently implement all operations needed for private classification, we have to rely on the generic method called *secure circuit evaluation*. Secure circuit evaluation has evolved dramatically from early proposals by Yao [Yao82,Yao86] to the modern and practical approaches [BMR90,NPS99]. We next briefly describe a two-party protocol popularized by Naor, Pinkas and Sumner [NPS99], that is based on the multi-party solution of [BMR90]. The solution itself can be viewed as a generalization of the *conditional oblivious transfer (*COT*)* protocol [COR99].

We specify the conditional oblivious transfer protocol by a public predicate π, that can be say the "greater than" (GT) predicate). 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. (This generalizes the original definition of [COR99].) 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 $s_1+s_2 = \pi(\varrho, \sigma) \mod N$. As any function can be computed bit by bit, secure function evaluation can be based on COT.

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 n_1 instances of it are run in parallel and still have low amortized complexity, e.g. like the protocols based on homomorphic encryption [AIR01,Lip05,LL05].

A COT protocol, based on the circuit evaluation protocol of [NPS99], is depicted by Protocol 1. We omit precise implementation details, as these are completely described and analyzed in [NPS99]; the protocol has even a freeware Java implementation Fairplay [MNPS04a,MNPS04b]. The full COT protocol has two rounds and is private in semi-honest model, given that the used pseudorandom function and the used OT protocol are secure. Recent extensions [Lin01,KO04] to this protocol provide security against malicious participants, though at the cost of performance degradation. We emphasize that the malicious model is out of our scope.

Protocol 1 Two round COT protocol for the predicate π

Client's Input: n_1-bit string ϱ.
Server's Input: n_2-bit string σ and $r_0, r_1 \in \{0,1\}^m$.
Let C_π be a circuit for $\pi : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \to \{0,1\}$ and $f : \{0,1\}^{k-1} \times \{0,1\} \to \{0,1\}^k$ be a pseudorandom function.

1. Server sends a special garbled circuit $\mathrm{E}(\mathrm{C}_\pi)$ to Client.
2. Client makes n_1 OT calls to get inputs for $\mathrm{E}(\mathrm{C}_\pi)$.
3. Client emulates computations in $\mathrm{E}(\mathrm{C}_\pi)$ and obtains:
   a) k-bit string r_0 if $\pi(\mu, \sigma) = 0$,
   b) k-bit string r_1 if $\pi(\mu, \sigma) = 1$.

The following facts follow from the construction of [NPS99]. (Note that we do not need unary gates, since they can be combined into binary or ternary gates.)

Fact 2 *Let the circuit C_π consist of ℓ_2 binary or duplication gates and ℓ_3 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. Computational complexity is linear in the size of circuit: both parties evaluate f at most $(4\ell_2+8\ell_3+4\log_2(\frac{m}{k}))k$ times and carry out a parallel execution of n_1 1-out-of-2 OT protocols.*

Fact 3 *If the function f is chosen from a family of pseudorandom functions that are secure against non-uniform adversaries, and Π^{OT} is a secure oblivious transfer protocol then Protocol 1 is private in the semi-honest model.*

Fact 4 *Several instances of Protocol 1 can be run in parallel without loosing privacy in the semi-honest model.*

Essentially, Facts 2 and 3 assure that Protocol 1 can be used to efficiently implement any function with simple structure (i.e., small circuit complexity), since there exist efficient and reasonably secure candidates for 80-bit pseudorandom functions, and thousands of OT protocols can be executed in parallel per second (at least, using dedicated hardware). Therefore, private comparison between n-bit integers is efficient, as latter can be done with n ternary gates. The main bottleneck is the OT protocol and therefore we will consider several techniques how to decrease the bit-size of inputs of the OT 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 $\mathbb{Z}_N = \{-L, \dots, L\}$, 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 $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$,

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

Obviously, approximation errors can change classification results. On the other hand, numerical approximation errors emerge also in floating-point implementations. Moreover, it is reasonable to assume that if approximation is sufficiently precise then modeling error, made by the choice of kernel, has much larger impact on the classification errors. We assume that 64-bit precision, with $\delta \approx 2^{-64}$, is sufficient, as it corresponds roughly to float precision provided that the matrix entries are in the range $(1.52 \cdot 10^{-5}, 65536)$. Such precision is achievable with a 64 bit modulus N.

In the case of restricted vertical split, Server owns all feature vectors $\boldsymbol{x}_i$ and thus can locally compute K. In the other two cases, 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 $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$, widely used in the text classification, reduces to private evaluation of shared scalar product for which several solutions are known [GLLM04,WY04].

Higher-degree polynomial kernels $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle^d$ can be evaluated by using a share conversion algorithm, as depicted by Protocol 2. Here, we assume that $\mathbb{Z}_N$ is the message space of the underlying additively homomorphic cryptosystem and that Client has a secret key sk. Alternatively, Client and Server can use two-party cryptosystem, as described in Section 3.2, then Server must help Client to decrypt (similarly to Prot. 4 and 7).

Protocol 2 Evaluation of degree-d polynomial kernels

Client's Input: a vector $\boldsymbol{x} \in \mathbb{Z}_N^m$.
Server's Input: a vector $\boldsymbol{y} \in \mathbb{Z}_N^m$.
Output: Additive shares $t_1 + t_2 = \langle \boldsymbol{x}, \boldsymbol{y} \rangle^d \mod N$.
Client knows $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathrm{G}(1^k)$ and Server knows pk.

1. Client and Server privately share $s_1 \cdot s_2 = \langle \boldsymbol{x}, \boldsymbol{y} \rangle \mod N$:
   a) Client sends $c_i \leftarrow \mathrm{E}_{\mathsf{pk}}(x_i)$, $i \in \{1, \ldots, m\}$.
   b) Server sends $c' \leftarrow \left(\prod_{i=1}^m c_i^{y_i}\right)^{s_2^{-1}} \cdot \mathrm{E}_{\mathsf{pk}}(0)$ for $s_2 \leftarrow \mathbb{Z}_N^*$.
   c) Client sets $s_1 \leftarrow \mathrm{D}_{\mathsf{sk}}(c')$.
2. Client and Server convert shares $t_1 + t_2 = s_1^d \cdot s_2^d \mod N$:
   a) Client sends $e \leftarrow \mathrm{E}_{\mathsf{pk}}(s_1^d)$ to Server.
   b) Server replies with $e' \leftarrow e^{s_2^d} \cdot \mathrm{E}_{\mathsf{pk}}(-t_2)$, where $t_2 \leftarrow \mathbb{Z}_N$.
   c) Client sets $t_1 \leftarrow \mathrm{D}_{\mathsf{sk}}(e')$.

Theorem 1. *Assume that for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$, $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \in \mathbb{Z}_N^*$ and that the underlying homomorphic cryptosystem is IND-CPA secure. Then Protocol 2 is a correct, computationally Client-private and perfectly Server-private kernel evaluation protocol.*

Proof. Correctness: First, note since $2\,|\mathbb{Z}_N^*| \geq |\mathbb{Z}_N|$ and computing the greatest common divisor can be done efficiently then one can efficiently sample from $\mathbb{Z}_N^*$ without knowing the factorization of N. Now, the homomorphic properties of the cryptosystem assure that $c' = \mathrm{E}_{\mathsf{pk}}\left(s_2^{-1} \cdot \sum_{i=1}^m x_i y_i\right)$, and thus Client and Server first obtain uniformly random multiplicative shares (s_1, s_2), s.t. $s_1 \cdot s_2 = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$. Similarly $e' = \mathrm{E}_{\mathsf{pk}}(s_1^d \cdot s_2^d - t_2)$, and thus in Step 2, Client and Server obtain uniformly random shares (t_1, t_2) s.t. $t_1 + t_2 = \langle \boldsymbol{x}, \boldsymbol{y} \rangle^d \mod N$.

Client-privacy follows straightforwardly from Fact 1, since Server sees only $m + 1$ ciphertexts.

Server-privacy: Server's replies are encryptions of either multiplicative or additive shares. Therefore, even a computationally unbounded Client gains no additional information except $s_2 \in \mathbb{Z}_N^*$ and $t_2 \in \mathbb{Z}_N$. $\square$

The assumption that $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ and N are coprime is essential for the privacy of Client. Namely, if $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \notin \mathbb{Z}_N^*$ then also $s_1 \notin \mathbb{Z}_N^*$. As the nontrivial factors of N are at least 512-bit integers for cryptosystems [Pai99,DJ01], then $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \notin \mathbb{Z}_N^* \Rightarrow \langle \boldsymbol{x}, \boldsymbol{y} \rangle = 0$ for all "reasonable" input ranges $\mathcal{X}$. For many interesting cases, $x_1, \ldots, x_m \geq 0$ for all $\boldsymbol{x} \in \mathcal{X}$ and a different polynomial kernel $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + 1)^d$ can be used instead.

What if we cannot assume that $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \in \mathbb{Z}_N^*$? One possible solution is as follows. One first computes additive shares of $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ modulo a small modulus N, and then uses circuit evaluation to map additive shares of 0 to shares of special value ζ and then resumes to Protocol 2. Finally, one uses circuit evaluation again to map shares of ζ^d back to shares of 0. This is quite resource-consuming and hence it is more advantageous, whenever it is possible, to use spaces $\mathcal{X}$ for which $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \in \mathbb{Z}_N^*$.

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 $\mathrm{E}_{\mathsf{pk}}(\boldsymbol{\alpha}) = (\mathrm{E}_{\mathsf{pk}}(\alpha_1), \ldots, \mathrm{E}_{\mathsf{pk}}(\alpha_m))$ to Server before the protocol. For brevity, denote $\boldsymbol{\kappa} := (\kappa(\boldsymbol{x}_1, \boldsymbol{x}), \ldots, \kappa(\boldsymbol{x}_n, \boldsymbol{x}))$, where $\boldsymbol{\kappa}$ has integer coordinates. Then $f_{\boldsymbol{\alpha}}(\boldsymbol{x}) = \sum_{j=1}^{n} \alpha_j \kappa_j$.

A private prediction protocol that works in the case of restricted vertical split is depicted by Protocol 3. 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. 3 and 4 can be modified so that Client learns the predicted label.)

Protocol 3 Private prediction for restricted split

Common parameters: Π with plaintext space $\mathbb{Z}_N$.
Client's input: secret key sk.
Server's input: public key pk, feature vectors $\boldsymbol{x}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, vector $\boldsymbol{\kappa}$, and encrypted weight vector $\mathrm{E}_{\mathsf{pk}}(\boldsymbol{\alpha})$.
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 $s_1 + s_2 = \sum \alpha_j \kappa_j$.
2. Client and Server use circuit evaluation to share $t_1 + t_2 = \mathrm{sign}(s_1 + s_2) \mod N$.

Theorem 2. *Assume that Π is an IND-CPA secure additively homomorphic cryptosystem and that the circuit evaluation step is private. Then Protocol 3 is correct and private.*

Proof. Correctness is straightforward. Privacy follows from the composition theorem, as Step 1 privately implements additive sharing of $s_1 + s_2 = f_{\boldsymbol{\alpha}}(\boldsymbol{x}) \mod N$.

The general case (see Prot. 4) is slightly more complicated, since Client and Server have to first share the kernel vector $\boldsymbol{\kappa}$, and then to repeat the same steps as in Prot. 3.

Protocol 4 Private prediction protocol in general case

Common inputs: two-party Π with plaintext space $\mathbb{Z}_N$, public key pk, encrypted weight vector $\mathrm{E}_{\mathsf{pk}}(\boldsymbol{\alpha})$.
Input data: Client has some coordinates of $\boldsymbol{x}, \boldsymbol{x}_1, \dots, \boldsymbol{x}_n$, Server has the remaining coordinates.
Other private inputs: Client knows a secret subkey sk_1, Server knows a secret subkey sk_2.
Output: Client and Server share a predicted class label.

Client and Server jointly do:

1. Share $\boldsymbol{a_1} + \boldsymbol{a_2} = \boldsymbol{\kappa} \mod N$.
2. Share $s_1 + s_2 = f_{\boldsymbol{\alpha}}(\boldsymbol{x}) \mod N$.
3. Use circuit evaluation to share $t_1 + t_2 = \mathrm{sign}(s_1 + s_2) \mod N$.

Step 1 of Prot. 4 can be implemented by using the techniques from Sect. 4. Importantly, one of the subkeys of a two-party cryptosystem reveals nothing about sk; for example, in the case of the two-party versions of cryptosystems [Pai99,DJ01] (see Sect. 3.2), sk_i are additive shares of sk w.r.t. a properly chosen modulus, and thus neither Client nor Server can decrypt $\mathrm{E}_{\mathsf{pk}}(\boldsymbol{\alpha})$. Therefore, Client and Server can locally compute and then exchange the encryptions c_1 and c_2 of randomized shares

$$c_i = \mathrm{E}_{\mathsf{pk}}\Big(\sum_{j=1}^{m} a_{ij}\alpha_j - r_i\Big) = \mathrm{E}_{\mathsf{pk}}(-r_i) \cdot \prod_{j=1}^{m} \mathrm{E}_{\mathsf{pk}}(\alpha_j)^{a_{ij}} \enspace ,$$

where $r_i \leftarrow \mathbb{Z}_N$. More precisely, Client sends $d_1 \leftarrow \mathrm{D}_{\mathsf{sk}_1}(c_1)$ and Server replies with $d_2 \leftarrow \mathrm{D}_{\mathsf{sk}_2}(c_2)$. Next, Client sets $s_1 \leftarrow \mathrm{D}_{\mathsf{sk}_1}(d_2) + r_1 \mod N$ and Server sets $s_2 \leftarrow \mathrm{D}_{\mathsf{sk}_2}(d_1) + r_2 \mod N$. Then, the sharing of $f_{\boldsymbol{\alpha}}(\boldsymbol{x})$ is complete since

$$s_1 + s_2 = \sum_{j=1}^{m} a_{1j}\alpha_j + \sum_{j=1}^{m} a_{2j}\alpha_j = \sum_{j=1}^{m} \kappa_j \alpha_j \mod N \enspace .$$

Since c_1 and c_2 are random encryptions of random values, then the described protocol for sharing $f_{\boldsymbol{\alpha}}(\boldsymbol{x})$ is private, provided that Π is a secure two-party IND-CPA cryptosystem.

Theorem 3. *Assume that Steps 1, 2 and 3 are computed correctly and privately and that Π is an IND-CPA secure two-party additively homomorphic cryptosystem. Then Protocol 4 is correct and private.*

Proof. Correctness is straightforward. Privacy follows from the composition theorem, since $\mathrm{E}_{\mathsf{pk}}(\boldsymbol{\alpha})$ reveals nothing to either party.

5.1 Circuit evaluation: targeted optimizations

Protocol 3 and 4 rely on circuit evaluation. We can use Protocol 1 to evaluate say the GT 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 $\boldsymbol{\kappa}$ and $\boldsymbol{\alpha}$ 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 of proper size, say $M \approx 2^{140}$.

For clarity, Protocol 5 is depicted for the representation $\mathbb{Z}_N = \{0, \dots, N-1\}$. The same result applies for the signed representation $\mathbb{Z}_N = \{-L, \dots, L\}$ 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 5 and then subtract the *public* value $2 \cdot \lfloor \frac{M}{2} \rfloor$ from the result. Similar techniques can be used for $M > N$.

Protocol 5 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 $\mathbb{Z}_N = \{0, \dots, N-1\}$, $0 \leq x < \frac{M}{2}$ 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 4. *Protocol 5 is private and correct, provided that the oblivious transfer protocol is private and correct, N is odd, $0 \leq x < \frac{M}{2}$ and $M < N$.*

Proof. Correctness: if $2x = t_1 + t_2 - N$ then the parities of the shares t_i differ and thus $r_1 + r_2 = t_1 + t_2 - N = 2x \mod M$. Otherwise, $t_1 + t_2 = 2x$ and the correctness follows similarly. Security: if the oblivious transfer protocol is ideal then Client sees a random $m_i \in \mathbb{Z}_M$ and thus Client's view is perfectly simulatable. As the oblivious transfer protocol is assumed to be secure, the 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. Therefore, Step 2 can be implemented with ℓ ternary gates for Protocol 3 and 4 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. To summarize, the communicational and computational costs decrease at least by a factor of 10.

5.2 Aggregated statistics and KKT violators

Note that if Client and Server locally add together shares for different feature vectors, they can straightforwardly count the number of positive examples. Recall that the class labels are ± 1, and that the sum of shares reveals difference between positive and negative examples. Moreover, Fact 4 allows to parallelize all substeps. The resulting protocol takes four rounds (i.e., all protocol messages can be combined into four larger ones).

One can straightforwardly modify Protocol 3 and 4 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 do privately estimate training and validation error or even do private cross-validation.

Finally, one can also extend Protocol 3 and 4 to count the number of Karush-Kuhn-Tucker violators. Recall that a feature vector $\boldsymbol{x}_i$ is a KKT violator if one of the next three conditions does not hold:

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

and one can build a similar circuit for detecting the KKT violators. Usually people use the number of the KKT violators 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 6 and 7, presented next, are private in the sense that Client and Server learn nothing except the number of iterations. 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.

For the ease of presentation, Protocol 6 is formulated for the restricted vertical split and Protocol 7 is formulated for the general case. It is straightforward to modify the protocols to cover the remaining cases.

Protocol 6 Private Kernel Perceptron

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

```
1. Server sets $\boldsymbol{c} = \mathrm{E}_{\mathsf{pk}}(\mathbf{0})$.
2. Client and Server execute the next cycle:
   for $i = 1$ to $n$ do
    a) They compute shares $s_1 + s_2 = f_{\boldsymbol{\alpha}}(\boldsymbol{x_i}) \mod N$.
    b) They use circuit evaluation to compute shares
       $t_1 + t_2 = \begin{cases} y_i, & \text{if } y_i(s_1+s_2) \le 0 \\ 0, & \text{if } y_i(s_1+s_2) > 0 \end{cases} \mod N$ .
    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 5. *Protocol 6 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 $|f_{\boldsymbol{\alpha}}(x_i)| < \frac{N}{2}$ and $|\alpha_i| < \frac{N}{2}$ always hold.*

Proof. Correctness is straightforward, since the substeps 2a)–2c) implement Step 4 in Algorithm 1 and values of $f_{\boldsymbol{\alpha}}(\boldsymbol{x}_i)$ and α_i are in $\mathbb{Z}_N$. Privacy follows, as either Server sees encrypted values or both parties obtain random shares.

In the Kernel Adatron algorithm, it is advantageous to define a coefficient vector $\boldsymbol{\beta} = (\alpha_i y_i)_{i=1}^n$. Then the predictor has a new form $f_{\boldsymbol{\beta}}(\boldsymbol{x}_i) = \sum_{i=1}^n k_{ij}\beta_i$ and there is no need to use labels $\boldsymbol{y}$ directly in the computation. Note that the update Step 3 has a form $\beta_i \leftarrow \beta_i + 1 - y_i f_{\boldsymbol{\beta}}(\boldsymbol{x}_i)$ and correction Steps 4-5 in Algorithm 2 imply $0 \le y_i\beta_i \le C$.

Theorem 6. *Protocol 7 is a correct and private implementation of the Kernel Adatron algorithm (Algorithm 2) provided that (1) the cryptosystem is additively homomorphic and IND-CPA secure; (2) all substeps are implemented correctly and privately; (3) constraints $|\beta_i + 1 - f_{\boldsymbol{\beta}}(\boldsymbol{x}_i)| < \frac{N}{2}$ and $|\beta_i| \le \frac{N}{2}$ always hold.*

Proof. Analogous to Theorem 5 and thus omitted.

Exact implementation of Substeps 2a) and 2b) is completely analogous to treatment given in Section 5, and thus we omit the details. Still, there is one important detail:

Protocol 7 Privacy-preserving Kernel Adatron

Common inputs: two-party Π with plaintext space $\mathbb{Z}_N$.
Input data: Client has some coordinates of $\boldsymbol{x}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, Server has the remaining coordinates.
Other private inputs: Client knows a secret subkey sk_1 and Server knows a secret subkey sk_2. Both know pk.
Server's Output: An encrypted weight vector $\boldsymbol{c} = \mathrm{E}_{\mathsf{pk}}(\boldsymbol{\beta})$.
Allowed side information: the number of iterations.

```
1. Server sets $\boldsymbol{c} = \mathrm{E}_{\mathsf{pk}}(\mathbf{0})$ and Client sets $\boldsymbol{c} = \mathrm{E}_{\mathsf{pk}}((0))$.
2. Client and Server execute next cycle:
   for $i = 1$ to $n$ do
     a) They compute shares $s_1 + s_2 = \beta_i + 1 - f_{\boldsymbol{\beta}}(\boldsymbol{x}_i) \mod N$.
     b) They use circuit evaluation to compute shares of $\beta_i$
        $t_1 + t_2 = \begin{cases} 0, & \text{if } y_i(s_1+s_2) < 0 \\ y_i C, & \text{if } y_i(s_1+s_2) > C \\ s_1 + s_2, & \text{otherwise} \end{cases} \mod N$ .
     c) Client sends $\mathrm{E}_{\mathsf{pk}}(t_1)$, Server replies with $\mathrm{E}_{\mathsf{pk}}(t_1)$.
        Both parties set $c_i \leftarrow \mathrm{E}_{\mathsf{pk}}(t_1)\mathrm{E}_{\mathsf{pk}}(t_2)$.
   end for
3. If not converged then repeat Step 2.
```

Substep 2b) can be implemented with $2\ell + 1$ ternary gates and the size of the garbled circuit is roughly $\mathcal{O}(2\ell)$ for both algorithms and 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 $\boldsymbol{\alpha}$. Alternatively, one can use a full gradient descent step instead, i.e., compute all values $f_{\boldsymbol{\alpha}}(\boldsymbol{x}_i)$ simultaneously and the update all coordinates of $\boldsymbol{\alpha}$ also simultaneously. In general, such batch updates tend to stabilize gradient descent methods but they also decrease the number of rounds. Fact 4 allows to execute Substeps 2a) and 2b) parallel for all $i = 1, \ldots, n$ and the number of rounds decreases from $6n$ to 6 per iteration.

7 Concluding remarks

In this paper, 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, for kernelized linear classification and for aggregation of encrypted classification results. However, there are still many open problems in private SVM classification and learning.

An interesting 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 (and everything that can be computed from this number). Analyzing the convergence speed of these algorithms seems to be quite difficult. For example, although Novikoff's theorem gives a bound for the number of iterations, the computation of the bound requires private information about the data: namely, the margin of the data. To achieve complete privacy, one might instead need to implement a private polynomial-time convex programming algorithm, such as ellipsoid or interior point methods.

Another interesting and practical question is whether there are any iterative private linear classification methods that need no 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 relatively costly 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 say kernels for structured data. Real-life data analysis using such kernels is computationally demanding even in the non-private case, but the corresponding techniques are developed fast, and kernels for structured data are becoming increasingly common in various applications.

Acknowledgments

We wish to 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.*

References

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT*, volume 2045 of *LNCS*, pages 119–135. Springer, 2001.

[AS00] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *SIGMOD Conference*, pages 439–450. ACM, 2000.

[BD02] Avrim Blum and John Dunagan. Smoothed analysis of the perceptron algorithm for linear programming. In *SODA*, pages 905–914, 2002.

[BDMN05] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. Practical privacy: The SuLQ framework. In *PODS*, 2005.

[BMR90] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *STOC*, pages 503–513. ACM Press, 1990.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10. ACM Press, 1988.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, pages 11–19. ACM Press, 1988.

[CL01] Yan-Cheng Chang and Chi-Jen Lu. Oblivious polynomial evaluation and oblivious neural learning. In *ASIACRYPT*, volume 2248 of *LNCS*, pages 369–384. Springer, 2001.

[CL05] Keke Chen and Ling Liu. Privacy preserving data classification with rotation perturbation. In *ICDM*, pages 589–592. IEEE Computer Society, 2005.

[COR99] Giovanni Di Crescenzo, Rafail Ostrovsky, and Sivaramakrishnan Rajagopalan. Conditional oblivious transfer and timed-release encryption. In *EUROCRYPT*, pages 74–89, 1999.

[DHC04] Wenliang Du, Yunghsiang S. Han, and Shigang Chen. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *SDM*. SIAM, 2004.

[DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *PKC*, volume 1992 of *LNCS*, pages 119–136. Springer-Verlag, 2001.

[DZ03] Wenliang Du and Zhijun Zhan. Using randomized response techniques for privacy-preserving data mining. In *KDD*, pages 505–510. ACM, 2003.

[FCC98] Thilo-Thomas Frieß, Nello Cristianini, and Colin Campbell. The Kernel-Adatron algorithm: A fast and simple learning procedure for Support Vector Machines. In *ICML*, pages 188–196. Morgan Kaufmann, 1998.

[FS99] Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

[GLLM04] Bart Goethals, Sven Laur, Helger Lipmaa, and Taneli Mielikäinen. On private scalar product computation for privacy-preserving data mining. In *ICISC*, volume 3506 of *LNCS*, pages 104–120. Springer, 2004.

[Gol04] Oded Goldreich. *Foundations of Cryptography*, volume II Basic Applications. Cambridge University Press, 2004.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *PODC*, pages 101–111. ACM Press, 1998.

[HMS01] David Hand, Heikki Mannila, and Padraic Smyth. *Principles of Data Mining*. MIT Press, 2001.

[KC04] Murat Kantarcioglu and Chris Clifton. Privately computing a distributed k-nn classifier. In *PKDD*, volume 3202 of *LNCS*, pages 279–290. Springer, 2004.

[KO04] Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In *CRYPTO*, volume 3152 of *LNCS*, pages 335–354. Springer, 2004.

[Lin01] Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. In *CRYPTO*, volume 2139 of *LNCS*, pages 171–189. Springer, 2001.

[Lip05] Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In *ISC*, volume 3650 of *LNCS*, pages 314–328. Springer-Verlag, 2005.

[LL05] Sven Laur and Helger Lipmaa. Additive conditional disclosure of secrets and applications. Cryptology ePrint Archive, Report 2005/378, 2005. http://eprint.iacr.org/.

[LP02] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002.

[MNPS04a] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay - secure two-party computation system. In *USENIX Security Symposium*, pages 287–302. USENIX, 2004.

[MNPS04b] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. The Fairplay project. Available from http://www.cs.huji.ac.il/labs/danss/Fairplay/, 2004. Last visit 10.01.2005.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce*, pages 129–139, 1999.

[OI05] Rafail Ostrovsky and William E. Skeith III. Private searching on streaming data. In *CRYPTO*, volume 3621 of *LNCS*, pages 223–240. Springer, 2005.

[Pai99] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *LNCS*, pages 223–238. Springer-Verlag, 1999.

[STC04] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[Vap00] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Statistics for Engineering and Information Science. Springer, 2000.

[VC04] Jaideep Vaidya and Chris Clifton. Privacy preserving Naïve Bayes classifier for vertically partitioned data. In *SDM*. SIAM, 2004.

[WY04] Rebecca N. Wright and Zhiqiang Yang. Privacy-preserving Bayesian network structure computation on distributed heterogeneous data. In *KDD*, pages 713–718. ACM, 2004.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 60–164, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

[YJV06] Hwanjo Yu, Xiaoqian Jiang, and Jaideep Vaidya. Privacy-preserving SVM using nonlinear kernels on horizontally partitioned data. In *SAC*, 2006.

[YZW05] Zhiqiang Yang, Sheng Zhong, and Rebecca N. Wright. Privacy-preserving classification of customer data without loss of accuracy. In *SDM*, 2005.

A Polynomial security model

We formalize the intuitive notion in the ideal *vs* real world paradigm [Gol04]. All adversaries are assumed to be polynomial-size non-uniform algorithms (i.e., circuit families). As we consider security against honest-but-curious adversaries, the ideal world model is simple: Client learns $(\varrho, f_1(\varrho, \sigma))$ and Server learns $(\sigma, f_2(\varrho, \sigma))$. Denote by $A(\sigma)$ a probabilistic honest-but-curious Server with an input σ that interacts with honest Server. Let Sim be a non-uniform algorithm that given $(\sigma, f_2(\varrho, \sigma))$ tries to mimic output of $A(\sigma)$. Then a distinguisher B achieves advantage

$$\mathrm{Adv}(B) = \max_{\varrho,\sigma} |\Pr[B = 1|\mathsf{EXP}_1] - \Pr[B = 1|\mathsf{EXP}_0]|$$

where in experiment EXP_1 the output of $A(\sigma)$ is given as input of B and in EXP_0 the output of $\mathsf{Sim}(\sigma, f_2(\varrho, \sigma))$ is used instead. The protocol is Client-private, if for all polynomial-time non-uniform adversaries A there exist a non-uniform polynomial-time simulator Sim_A such that all polynomial-time non-uniform distinguishers B achieve negligible advantage. More precisely, there are actually a family of protocol implementations indexed by security parameter k and all working times must be polynomial in k and the advantage $\mathrm{Adv}(B) = k^{\omega(1)}$. Server-privacy is defined dually.

Note that if there is no desired output, i.e., $f_1(\varrho, \sigma) = \perp$, then Client-privacy has equivalent formulation: for any two inputs ϱ_0 and ϱ_1, the honest-but-curious Server A cannot distinguish the inputs, i.e.,

$$\max_{\varrho,\sigma_0,\sigma_1} |\Pr[A = 1|\mathsf{EXP}_0] - \Pr[A = 1|\mathsf{EXP}_1]|$$

is negligible in k, where in EXP_i Client's input is ϱ_i.