Efficiently Stealing your Machine Learning Models

Robert Nikolai Reith
robert.reith@t-online.de
TU Darmstadt

Thomas Schneider
schneider@encrypto.cs.tu-darmstadt.de
TU Darmstadt

Oleksandr Tkachenko
tkachenko@encrypto.cs.tu-darmstadt.de
TU Darmstadt

ABSTRACT

Machine Learning as a Service (MLaaS) is a growing paradigm in the Machine Learning (ML) landscape. More and more ML models are being uploaded to the cloud and made accessible from all over the world. Creating good ML models, however, can be expensive and the used data is often sensitive. Recently, Secure Multi-Party Computation (SMPC) protocols for MLaaS have been proposed, which protect sensitive user data and ML models at the expense of substantially higher computation and communication than plaintext evaluation.

In this paper, we show that for a subset of ML models used in MLaaS, namely Support Vector Machines (SVMs) and Support Vector Regression Machines (SVRs) which have found many applications to classifying multimedia data such as texts and images, it is possible for adversaries to passively extract the private models even if they are protected by SMPC, using known and newly devised model extraction attacks. We show that our attacks are not only theoretically possible but also practically feasible and cheap, which makes them lucrative to financially motivated attackers such as competitors or customers. We perform model extraction attacks on the homomorphic encryption-based protocol for privacy-preserving SVR-based indoor localization by Zhang et al. (International Workshop on Security 2016). We show that it is possible to extract a highly accurate model using *only* 854 *queries* with the estimated cost of $0.09 on the Amazon ML platform, and our attack would take *only* 7 *minutes* over the Internet. Also, we perform our model extraction attacks on SVM and SVR models trained on publicly available state-of-the-art ML datasets.

CCS CONCEPTS

• **Security and privacy → Privacy protections**; *Privacy-preserving protocols*; • **Computing methodologies → Support vector machines**.

KEYWORDS

Machine learning, model extraction, Support Vector Machine, Support Vector Regression Machine, ideal leakage

ACM Reference Format:
Robert Nikolai Reith, Thomas Schneider, and Oleksandr Tkachenko. 2019. Efficiently Stealing your Machine Learning Models. In *18th Workshop on Privacy in the Electronic Society (WPES'19), November 11, 2019, London, United Kingdom.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3338498.3358646

WPES'19, November 11, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6830-8/19/11…$15.00
https://doi.org/10.1145/3338498.3358646

1 INTRODUCTION

Machine Learning (ML) is a term that summarizes different algorithms of deriving models from data to make predictions. Such predictions can be either categorical (called classification tasks), e.g., for predicting if a picture contains a cat or a dog, or real-valued (called regression tasks), e.g., for predicting a stocks value. One such method are Support Vector Machines (SVMs) for categorical predictions, and Support Vector Regression Machines (SVRs) for real-value predictions. Since ML needs high amounts of processing power, new service providers have emerged that offering Machine-Learning-as-a-Service (MLaaS). These providers, such as Amazon ML[1], Google Cloud ML[2], BigML[3], and many others, let their users create ML models on their platforms, and offer them to monetize their models by letting other users query them for predictions. In this business model, model providers want to keep their intellectual property private and their users might want to keep their concrete data private. This is a classical multi-party computation task, making it an area of interest for Secure Multi-Party Computation (SMPC). To provide privacy for MLaaS, SMPC protocols for ML have been introduced, e.g., for SVMs [24, 36, 55], SVRs [56], and Extreme Learning Machines [57], to name just a few. These protocols allow for remote predictions without directly revealing the model to the user, or the user's data to the model provider.

However, Tramèr et al. [48] showed for different ML algorithms that it is possible for users to steal accurate models using solely the prediction API. In our paper, we extend the findings of [48] from black-box SVM model extraction attacks to the case of SVRs and introduce new white-box attacks for SVRs. We show that SMPC protocols fail to protect the provider's model privacy, since it is possible to extract the model. Our attacks target implementations of MLaaS schemes, and we show that MLaaS schemes are inherently vulnerable to extraction attacks. Concretely, we show that an attacker can extract a Radial Basis Function (RBF) kernel SVR model as described in [56] with only 854 queries within 7 minutes for a cost of $0.09 on the Amazon ML platform and 71 MB of bandwidth for running the SMPC-based protocol.

1.1 Motivation

In recent years, location privacy using SMPC got a considerable amount of attention [16, 20, 21, 53, 56]. In the paper by Zhang et al. [56], the authors proposed a privacy-preserving protocol for indoor localization using Wi-Fi fingerprints. For this, an SVR is used, which can be queried by users to find their location within

[1] https://aws.amazon.com/machine-learning/
[2] https://cloud.google.com/ml-engine/
[3] https://bigml.com/

a building. Their scheme uses homomorphic encryption, which implies a high computation overhead.

Using such a scheme, the provider of the ML model can monetize its knowledge by charging fees for each localization. Such a service is called MLaaS, a business model being embraced by big companies like Amazon or Google. These companies have an economic interest in protecting their intellectual property, i.e., the ML models. A client unwilling to pay for a big amount of predictions from these providers, or a competitor trying to copy the business model, have a financial incentive to steal the ML models. If the cost of extracting an ML model is lower than the potential financial gain from it, MLaaS will always be a target for financially motivated attackers.

With the aim of protecting ML models, we study the feasibility of model extraction attacks on SVM and SVR and propose possible countermeasures to our attacks. Indeed, we will show that the privacy of MLaaS providers for these classifiers is not given, even when employing SMPC protocols such as [56]. With our findings we show the need for additional protection mechanisms in MLaaS.

1.2 Our Contributions

We make the following contributions in our paper:

- We introduce new equation-solving attacks on linear and quadratic Support Vector Regression Machines (SVRs) and propose retraining approaches for other kernels, such as Radial Basis Function (RBF) kernel SVRs.
- We design and implement a framework for Machine Learning (ML) model extraction attacks in the Python programming language. Our framework is open-source and publicly available at https://github.com/robre/attacking-mlaas.
- We use our framework to examine the accuracy of our attacks on ML models trained on publicly available state-of-the-art datasets, as well as their computation efficiency and the number of required queries to perform attacks, and compare them to the existing extraction attacks on Support Vector Machine (SVM).
- We further investigate the feasibility of extraction attacks in realistic scenarios and show that Wi-Fi localization schemes using SVRs (e.g., [56]) are indeed vulnerable to our extraction attacks even when protected by Secure Multi-Party Computation (SMPC) techniques. In a simulated Machine-Learning-as-a-Service (MLaaS) environment using a RBF kernel SVR, our extraction attack was able to extract a near-perfect model within 7 minutes in a high-latency network using only 854 queries. Moreover, we confirm our results on further publicly available indoor localization datasets with even more complex ML models.

2 PRELIMINARIES

Notation. We denote vectors as bold latin characters, e.g., $\mathbf{x}$. Angled brackets, e.g., $\langle \mathbf{x}, \mathbf{y} \rangle$, are used to denote dot products. Sole features of a feature vector $\mathbf{x}$ are denoted by x_i.

Basics. Support Vector Machines (SVMs) [7] and Support Vector Regression Machines (SVRs) [11] belong to a category of ML algorithms known as supervised learning. In supervised learning, the learning algorithm is provided with a set of data, called training data, which consists of example inputs with their corresponding desired outputs. The learning algorithm is tasked with finding the rules that map the inputs to their corresponding outputs. The outputs for binary classifiers, such as SVMs, can be either negative or positive. For example, they can classify whether an image contains a cat or not. Multiclass classifiers can distinguish even more classes, such as if an image contains a black, white, orange, or gray cat. Regressors, such as SVRs, have real valued outputs. They can be therefore used to predict continuous values, e.g., stock prices.

2.1 Support Vector Machines

Support Vector Machines (SVM) [7] are a class of machine learning algorithms that are used for classification tasks. For this, an SVM defines a hyperplane, which linearly separates a given set of training data $\{(\mathbf{x}_i, y_i) | i = 1, \ldots, m; y_i \in \{0, 1\}\}$ with samples $\mathbf{x}_i$, their corresponding targets y_i and the number of training data m, into two classes and classifies new samples by checking what side of the hyperplane the sample would be on. The hyperplane is defined in such a manner that its distance to the closest datapoints for each class is maximized. The closest datapoints are called support vectors. The distance between the hyperplane and the support vectors is called the margin.

An SVM is defined by a function $f(\mathbf{x}) = sign(\langle \mathbf{w}, \mathbf{x} \rangle + b)$, where '*sign*' outputs 0 for negative inputs and 1 otherwise. The inputs to this function are a feature vector $\mathbf{x}$, a normalized vector $\mathbf{w}$ that points orthogonally to the defined hyperplane from the origin of the coordinate system, and bias b, a dislocation of the hyperplane. The margin is defined in such a manner that its size is the same in each direction. For a normalized dataset, its size is 1, such that if $\mathbf{w} \cdot \mathbf{x} + b \geq 1$ a sample is classified as a positive instance and for $\mathbf{w} \cdot \mathbf{x} + b < 1$ a sample is classified as a negative instance, without providing a confidence score.

To calculate optimal parameters $\mathbf{w}$ and b for a given problem, an optimization problem has to be solved: minimize $\frac{1}{2}||\mathbf{w}||_2^2$ subject to $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$ for $1 \leq i \leq m$, where $||\mathbf{w}||_2^2$ denotes the quadratic Euclidean norm of $\mathbf{w}$. To account for outliers and to prevent overfitting, which is a common error in ML when a model corresponds too closely to a specific set of data, a soft-margin can be used, which allows misclassification in some instances by introducing a penalty parameter ξ_i. This parameter is positive for misclassifications and 0 otherwise. Another positive parameter C is introduced, which is the proportional weight of the penalties. The optimization problem is then defined by:

$$\begin{aligned} &\text{minimize } \frac{1}{2}||\mathbf{w}||_2^2 + C \sum_{i=1}^{m} \xi_i \text{ subject to} \\ &y_i \left(\langle \mathbf{w}, \mathbf{x_i} \rangle + b\right) \geq 1 - \xi_i \text{ for } 1 \leq i \leq m. \end{aligned} \tag{1}$$

The computation of $\mathbf{w}$ and b is done by transforming the optimization problem into a Lagrangian dual problem. With $\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$, the Lagrangian dual problem is defined as: maximize for α: $\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$ subject to $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^{m} \alpha_i y_i = 0$. This dual problem can be efficiently solved using techniques described in [46, 52].

When a dataset is not linearly separable, a kernel-technique can be used to map the input space to a higher dimension, where such separation might be possible. To map the input space to a higher dimension, a projection function $\phi \colon \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}, \mathbf{x} \mapsto \phi(\mathbf{x})$ is used. Such a function however can be hard to compute, especially if the

Table 1: Popular kernels.

Kernel	K(x,x')
Linear	$\mathbf{x} \cdot \mathbf{x}'$
Polynomial	$(\mathbf{x} \cdot \mathbf{x}' + 1)^d$
RBF	$\exp(\gamma \|\|\mathbf{x} - \mathbf{x}'\|\|^2)$
Sigmoid	$\tanh(\gamma \mathbf{x}^t \cdot \mathbf{x}' + r)$

projected dimension is large, or even infinite, as it is the case for the popular Radial Basis Function (RBF) kernels. In practice, for the classification, only the dot product of two transformed vectors $\phi(\mathbf{x}) \cdot \phi(\mathbf{x}')$ is used, which makes it possible to apply specialized kernel-functions $K(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x}) \cdot \phi(\mathbf{x}')$, that simplify the computation of this product. The resulting classification function is defined as $f(\mathbf{x}) = sign(\langle \mathbf{w}, \phi(\mathbf{x}) \rangle + b) = sign\left(\sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right)$.

Table 1 lists a few kernels that have seen broad use in different areas. While the linear and RBF kernels are commonly used for all kinds of tasks, such as cancer classification [34] or bankruptcy prediction [29], polynomial kernels have seen use in document classification [27]. The sigmoid kernel gained popularity coming from its use in neural networks, but hasn't seen much use in the context of SVMs, because it only becomes a Positive Semi Definite (PSD) kernel for a few combinations of its parameters, which is a mathematical condition for kernels in SVM [4].

2.2 Support Vector Regression Machines

Support Vector Regression Machines (SVRs) [11] use the concepts of SVMs to create regression models. While SVMs classify data into two classes, SVRs aim to find a function $f(\mathbf{x})$, that approximates a given training set with error at most ϵ for each y_i and is as flat as possible [45]. Similarly to SVMs, a linear SVR is defined by a function $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$, where $\mathbf{w}$ and b are calculated by solving an optimization problem:

$$\text{minimize } \frac{1}{2}||\mathbf{w}||_2^2 \text{ subject to } \begin{cases} y_i - \langle \mathbf{w}, \mathbf{x_i} \rangle - b & \leq \epsilon, \\ y_i + \langle \mathbf{w}, \mathbf{x_i} \rangle + b & \leq \epsilon. \end{cases}$$

As for SVMs, we can introduce a soft-margin to account for some errors in the training data. For each condition, we introduce penalties, ξ_i and ξ_i^*, indicating a positive and negative error respectively. Also, the parameter C to weight the errors in trade-off to the flatness of $\mathbf{w}$ is introduced. The optimization problem becomes:

$$\text{minimize } \frac{1}{2}||\mathbf{w}||_2^2 + C \sum_{i=1}^{m} (\xi_i + \xi_i^*) \text{ subject to}$$

$$\begin{cases} y_i - \langle \mathbf{w}, \mathbf{x_i} \rangle - b & \leq \epsilon + \xi_i, \\ y_i + \langle \mathbf{w}, \mathbf{x_i} \rangle + b & \leq \epsilon + \xi_i^*, \\ \xi_i, \xi_i^* & \geq 0. \end{cases}$$

Translating the optimization problem into its dual form, we have:

$$\text{maximize } \begin{cases} -\frac{1}{2} \sum_{i,j=1}^{m} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) \langle \mathbf{x_i}, \mathbf{x_j} \rangle, \\ -\epsilon \sum_{i=1}^{m} (\alpha_i + \alpha_i^*) + \sum_{i=i}^{m} y_i (\alpha_i - \alpha_i^*) \end{cases}$$

subject to $\sum_{i=i}^{m} (\alpha_i - \alpha_i^*) = 0$ and $\alpha_i, \alpha_i^* \in [0, C]$, where $\mathbf{w}$ can now be written as $\mathbf{w} = \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) \mathbf{x_i}$ and, therefore, the regression function is defined by $f(\mathbf{x}) = \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) \langle \mathbf{x_i}, \mathbf{x} \rangle + b$.

Just like for SVMs, we can use the kernel trick to perform regression on non-linear functions, by substituting the dot-product with a chosen kernel $f(\mathbf{x}) = \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) K(\mathbf{x_i}, \mathbf{x}) + b$. The kernels that can be used for SVRs are the same as for SVMs (see Tab. 1).

2.3 General Attacks on Machine Learning Algorithms

ML algorithms can be attacked and manipulated in many different ways. In [2], Barreno et al. propose a separation of attacks into two types: causative and exploratory. A relatively new type of attacks, evasion attacks, was introduced later in [43] and [51].

Causative Attacks. Causative attacks, better known as poisoning attacks, attempt to sabotage a machine learning algorithm in such a way, that it fails to perform its intended task. For instance, a poisoning attack on a spam filter would manipulate the training data to cause misclassification of certain spam mails, e.g., having malicious mails classified as harmless [3, 37]. This attack is performed while training an algorithm.

Evasion Attacks. Evasion attacks aim at finding input data to a fully trained algorithm that result in an incorrect output [23]. For instance, an evasion attack on a spam-filter could find potential spam-mails that would not be classified as such, or on a malware-scanner to find an obfuscation of the malware which does not get flagged. This type of attack can be improved and facilitated by gathering prior information about the attacked model, i.e., by performing an exploratory attack (see §2.3), specifically an extraction attack, beforehand.

Exploratory Attacks. Exploratory attacks, which our paper further explores, intend to unravel information about the algorithm's configuration, or general inner workings, in order to extract the algorithm or its parameters, to find weaknesses in the algorithm, or to gain knowledge useful for further attacks, such as evasion attacks. This type of attack is performed on a fully trained model.

The class of exploratory attacks is defined very broadly and two subclasses can be defined: model inversion and model extraction attacks. Model inversion attacks ultimately aim to compromise users' privacy by extracting information about training data, such as if a particular instance was used in the training set or not [12, 17, 41, 44]. Model extraction attacks try to extract parameters from an ML model. For instance, an extraction attack might learn the exact decision boundary used by a linear classifier such as an SVM [26]. In a broader sense, such an attack might learn the general set of rules that the algorithm follows, or other statistical or logical attributes of the underlying model. In our paper, we explore model extraction attacks on SVMs and SVRs that intend to extract or estimate the model parameters.

2.4 White-Box and Black-Box Attacks

We differentiate between white-box and black-box attacks on MLaaS. In a white-box attack, the attacker knows some details about how the MLaaS provider has implemented a model. Specifically, he knows or can make an educated guess, which kernel is used in

the model. In a black-box attack, the attacker has no knowledge of the kernel that is used in the model. Most attacks shown in this paper are white-box attacks. However, if a kernel is not explicitly known they can still be performed with the assumption of a specific kernel or, as described in §5.5, in parallel for a subset of all possible kernels. The extracted model can then be compared to the original one using a test set of training data. If it performs well enough, the actual kernel used by the original model was probably guessed correctly, or is not particularly important, when two different kernels can separate the same data.

2.5 Secure Multi-Party Computation

Secure Multi-Party Computation (SMPC) allows multiple distrustful parties to jointly compute a public function g on their private inputs, while revealing nothing but the result of the computation. This is achieved by using cryptographic techniques such as homomorphic encryption [32], Yao's garbled circuits [54], secret sharing [14], or combinations thereof [8, 18]. SMPC is used to compute the function g revealing only what the ideal functionality would reveal, i.e., the result of g and nothing else. SMPC guarantees privacy of the parties' inputs, however, if the output of the ideal functionality leaks information about the private data, the use of SMPC is in vain. This was recognized as a threat for several privacy-preserving protocols [1, 21, 40].

3 RELATED WORK

Lowd and Meek [26] worked on evasion attacks specifically targeting spam detection in emails and showed an exploratory attack that extracts parameters from a linear classifier, such as linear SVMs. Using their algorithm, which we describe in §4, they devised a way of creating spam emails that are not detected by the spam filter under attack.

Tramèr et al. [48] further explored model extraction attacks. Targeting MLaaS, they proposed attacks on Logistic Regression [19], Multilayer Perceptrons [38], Decision Trees [35], SVMs [7], Multiclass Logistic Regression [13] and Neural Networks [42]. They found that providing an attacker with confidence scores, which are values that attest how certain the algorithm is about a prediction, gives the attacker a big advantage in extracting the model. However, it is still possible to extract models without confidence scores. Especially relevant to our work is their research on SVMs. They introduced an extension to the Lowd-Meek attack [26], enabling an attacker to extract the model of an SVM using polynomial kernels which include linear kernels. For the extraction of other kernels, they propose different retraining schemes (see §4.2 for details). Papernot et al. [33] explore black-box extraction and evasion attacks on MLaaS using Deep Neural Networks (DNNs) [15]. Their attacks consist of retraining a local DNN from class labels obtained from the MLaaS, and using the local model to craft adversarial samples that get misclassified by the MLaaS. In [9], Dmitrenko further explores the model extraction attacks on DNN introduced in [33].

Shi et al. [43] proposed another approach to extract models. They trained an ML model relying on deep learning and using the MLaaS provider as an oracle. Kesarwani et al. [22] proposed a mechanism which they denote as an extraction monitor, which alerts an MLaaS provider when too much information about a Decision Tree model was released, enabling an attacker to potentially extract such a model with techniques described in [48]. Wang and Gong [51] proposed techniques to extract hyperparameters from MLaaS providers. Hyperparameters are optimization parameters for ML algorithms, such as for SVMs the parameter C in Eq. 1 in §2.1. Attacks on SVRs in particular have not been researched yet. In this work, we fill this gap by extending known attacks on SVMs to SVRs, finding new equation-solving attacks on SVR, and putting our attacks into the context of SMPC for private MLaaS.

The known model extraction attacks on SVMs described previously are explained in detail below, as extraction attacks on SVMs and extending them to SVRs are the main focus of our work. Our attack scenario consists of an MLaaS provider, where a fully-trained SVM is provided by a server to classify data from a client, who acts maliciously and tries to extract the SVM parameters. In short, the attacker can poll an SVM for labels on arbitrary data.

4 THE LOWD-MEEK ATTACK

The first extraction attack on linear classifiers in general and thus also on SVMs was proposed by Lowd and Meek [26]. Their white-box attack assumes a third-party oracle, for instance, an MLaaS API, with membership queries that return the predicted class label without any confidence values. As some SVMs are linear classifiers, they are susceptible to this attack. This attack was also described in the context of privacy-preserving branching programs [1].

To initiate the Lowd-Meek attack, a sample $\mathbf{x}^+$ classified as positive and a sample $\mathbf{x}^-$ classified as negative have to be provided. Furthermore, the algorithm takes two parameters, the approximation threshold ϵ, and the minimum ratio of two non-zero weights δ. The algorithm starts by finding sign-witnesses $\mathbf{s}^+$ and $\mathbf{s}^-$, which are samples that differ only in one feature f, but $\mathbf{s}^+$ is classified as positive and $\mathbf{s}^-$ as negative. To find such a pair, the algorithm starts with $\mathbf{x}^+$, which is classified as a positive instance, and traverses through its features, in each step changing one feature value f from the initial value it had as $\mathbf{x}^+_f$ to the value of $\mathbf{x}^-_f$ and checking if the classification has changed with it, by querying the server. This is done until a negative instance is found, denouncing the instances, where the classification change occurred, as sign-witnesses. Next, the feature f of $\mathbf{s}^+$ or $\mathbf{s}^-$ is adjusted using line-search, until a negative instance $\mathbf{x}$ with a gap of less than $\epsilon/4$ is found. Now, to have a weight within 1 and $1 + \epsilon/4$ on the feature f, a one is added to or subtracted from $\mathbf{x}_f$. Having $\mathbf{w}_f \approx 1$, the other feature weights are found by adjusting their value to find their distance to the decision boundary using line search. The found distance is the according weight for that feature. But first, $1/\delta$ is added or subtracted to their weight and if their classification does not change, a weight of 0 for that feature is assumed. Having found all weights, the bias can be easily found, as we have the inclination $\mathbf{w}$ and at least one point is on or with maximum distance $\epsilon/4$ of the hyperplane.

Given d total features, the algorithm uses a maximum of $d - 2$ queries to find the sign-witnesses. However, we can add the two queries back to confirm the initial classes of $\mathbf{x}^+$ and $\mathbf{x}^-$. To find a negative instance with maximum distance $\epsilon/4$ to the hyperplane, $O(\log(1/\epsilon) + size(\mathbf{s}^+, \mathbf{s}^-))$ queries are needed. To find the relative weight of each other feature, another $O(\log(1/\epsilon) + size(c))$ queries are needed, where $size(c)$ describes the encoding length. In total,

the algorithm uses a polynomial number of queries. This attack however is limited to the setting, where the feature translation happens on the server-side. This means that features like strings or other data types that have to be translated into a numerical representation get translated by the server and not the client. This can make it impossible for the attacker to create the queries he needs for this attack.

4.1 The Lowd-Meek Attack for Nonlinear Kernels

Tramèr et al. [48] proposed an extension to the Lowd-Meek attack which enables the attacker to extract some nonlinear kernels, such as the polynomial kernel. Their attack is performed by extracting the model within the transformed feature space, where the model is effectively linear. The hyperplane in the transformed feature space is described as $\langle \mathbf{w}^F, \phi(\mathbf{x}) \rangle + b = 0$, where $\mathbf{w}^F = \sum_{i=1}^{m} \alpha_i \phi(\mathbf{x}_i)$. Therefore, we can use the Lowd-Meek attack to extract $\mathbf{w}^F$ and b if we can efficiently calculate $\phi(\mathbf{x})$ and its inverse. Unfortunately, the kernel-trick [28] was specifically introduced so that $\phi(\mathbf{x})$ and $\phi(\mathbf{x}')$ do not have to be explicitly computed when calculating their dot product, because the feature spaces can have infinite dimension or be very inefficient to compute. For some kernels, such as the polynomial kernel, it is possible to derive the concrete function $\phi(\mathbf{x})$ which makes them susceptible to this attack. The RBF-kernel however uses an infinite feature space so that this attack cannot be performed on it. The number of queries used in this attack is the same as for the classic Lowd-Meek attack performed in the transformed feature space.

Because the extraction happens in the transformed feature space, we calculate our extraction steps within that space. However, our queries themselves are not in the transformed feature space, because the transformation is done by the server. Therefore, we have to calculate the representation of the desired queries within the original feature space by calculating the inverse of $\phi(\mathbf{x})$ before issuing each query, which increases the processing costs in this attack method for each query.

4.2 Retraining Attack for SVMs

A retraining attack lets the attacked entity classify a set of samples and uses the resulting sample-classification tuples to train an SVM itself, which should result in similar model parameters. The amount of samples used in a retraining attack is capped by a query budget m. To find an effective query budget, Tramèr et al. [48] introduced the budget factor α, with $m = \alpha(d+1)$ and $0.5 \leq \alpha \leq 100$, where d is the number of features. They found that with a budget factor of $\alpha = 50$ most models could be extracted with 99% accuracy. They introduce three possible approaches to the retraining attack:

Retraining with uniform queries. The first and simplest approach works as follows: take a set of m uniformly random samples, have them classified by the prediction API and use the result for retraining.

Line-search retraining. This approach uses line search techniques as they are used in the Lowd-Meek attack (see beginning of §4), to have samples classified that are close to the decision boundary, yielding a more accurate result.

Adaptive retraining. The third approach is adaptive retraining, which utilizes active learning techniques [6] to improve the extraction. This approach splits the total query budget m into r rounds of $n = m/r$ samples each. First, n uniformly random samples are classified to train an initial model. Next, for each round until the prediction doesn't improve, n samples are chosen to be classified next. The samples chosen are those, the current extracted model is the least certain about, which are the samples that are the closest to the decision boundary. In the case of SVMs, the decision boundary is the hyperplane of the current extracted model.

4.3 Neural Network Retraining Attacks

Shi et al. [43] have shown that it is possible to perform black-box retraining attacks on Naive Bayes classifiers, Artificial Neural Networks, and SVMs using Deep Learning. They have also found, that Feedforward Neural Networks are better at inferring other classifiers, such as SVMs, than SVMs or Naive Bayes Classifiers are at inferring other classifiers. Papernot et al. [33] have also introduced a retraining attack using DNNs, attacking other black-box DNNs. Their attacks were further researched by Dmitrenko [9]. As a black-box DNN can be substituted by any other classifier with the same function, these attacks should also work on different classifiers.

5 OUR NEW ALGORITHMS FOR MODEL EXTRACTION OF SVRs

We propose the following new algorithms for model-extraction attacks on Support Vector Regression Machines (SVRs). To the best of our knowledge, no model extraction attacks for SVRs have been proposed before. The white-box algorithms described in §5.1 and §5.2 are our new equation-solving attacks that can extract exact models for linear and quadratic kernels, respectively. The algorithms described in §5.3 and §5.4 extend the retraining approach by Tramèr et al. [48] to the extraction of SVR models. The last algorithm, described in §5.5, is an approach to black-box model extraction.

5.1 Exact SVR Model Extraction using Linear Kernels

For SVRs using a linear kernel, the regression function can be described as $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$ with $\mathbf{w}, \mathbf{x} \in \mathcal{X}$ and $b \in \mathbb{R}$. Since we can query the oracle for arbitrary $\mathbf{x}$, we get the exact value of b by querying the oracle with a zero vector $\mathbf{x} = \{0, .., 0\}$. To find $\mathbf{w}$, we find the value of each dimension i in $\mathbf{w}$ one by one by querying the oracle with a vector $\mathbf{x}$ that is 1 in the corresponding position and 0 everywhere else, i.e., $w_i = f(\mathbf{x}) - b$ with $x_i = 1, x_{j \neq i} = 0$. In total, this algorithm needs $n + 1$ queries to extract the exact model parameters, where n is the dimension of the feature space.

5.2 Exact SVR Model Extraction using Quadratic Kernels

SVRs using quadratic kernels can be described as follows: $f(\mathbf{x}) = \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) K_{quadratic}(\mathbf{x}_i, \mathbf{x}) + b$, where $\mathbf{x}$ denotes the vector to be classified, m is the amount of training data, α_i and α_i^* are dual Lagrangian coefficients describing the weight of each training sample,

$\mathbf{x}_i$ describes the training data, and b is the bias, with

$$K_{quadratic}(\mathbf{x}', \mathbf{x}) = (\langle \mathbf{x}', \mathbf{x} \rangle + c)^2 = \langle \phi_{quadratic}(\mathbf{x}'), \phi_{quadratic}(\mathbf{x}) \rangle,$$

where $K_{quadratic}$ is the quadratic kernel, c is a kernel parameter, and $\phi_{quadratic}$ is the feature transformation function for the quadratic kernel. Therefore, $\mathbf{w} = \sum_{i=1}^{m}(\alpha_i - \alpha_i^*)\phi_{quadratic}(\mathbf{x}_i)$, which is a weight vector in the transformed feature space and we end up with a linear regression function in the transformed feature space:

$$f(\mathbf{x}) = \langle \mathbf{w}, \phi_{quadratic}(\mathbf{x}) \rangle + b. \tag{2}$$

For most of the kernels, calculating $\phi(\mathbf{x})$ is infeasible, which is the reason why we normally depend on the kernel-trick. However, for some kernels such as the quadratic kernel, $\phi_{quadratic}(\mathbf{x})$ is rather simple and allows for extraction [5]. The transformation function of the quadratic kernel is defined as follows:

$$\phi_{quadratic}(\mathbf{x}) = \begin{cases} x_n^2, \ldots, x_1^2, \\ \sqrt{2}x_n x_{n-1}, \ldots, \sqrt{2}x_2 x_1, \\ \sqrt{2c}x_n, \ldots, \sqrt{2c}x_1, \\ c. \end{cases}$$

$\phi_{quadratic}(\mathbf{x})$ produces a vector of dimension $d = \binom{n}{2} + 2n + 1$. Knowing the nature of the feature transformation $\phi(\mathbf{x})$, and being able to query the oracle with arbitrary $\mathbf{x}$ and get the corresponding $f(\mathbf{x})$, we can reconstruct $f'(\mathbf{x})$ equivalent to Eq. 2. If we look closely at Eq. 2 and remember how $\phi_{quadratic}$ transforms vectors, we can see that for instance w_d, the last feature weight in $\mathbf{w}$ when calculating the function, will always be multiplied by c at the end, after which b will be added. For our reconstructed function $f'(\mathbf{x})$ we can therefore create an equivalent functionality by setting $w'_d = 0$ and $b' = w_d c + b$. To find our b', we use a zero vector $\mathbf{v}_0 = (0, \ldots, 0)^T$ to get $f(\mathbf{v}_0) = \langle \mathbf{w}, \phi_{quadratic}(\mathbf{v}_0) \rangle + b = w_d c + b = b'$.

Now, we can see that the first n weights $w_1, \ldots, w_n$, which get multiplied with $x_n^2, \ldots, x_1^2$, respectively in the dot product, can be extracted by sending two queries for each of them. Those two queries are positive and negative unit vectors $\mathbf{v}_i^+$ and $\mathbf{v}_i^-$, where $\mathbf{v}_i^+ := (v_1, \ldots, v_i, \ldots, v_n) = (0, \ldots, 1, \ldots, 0)\ \forall i \in 1, \ldots, n$ and $\mathbf{v}_i^- := (v_1, \ldots, v_i, \ldots, v_n) = (0, \ldots, -1, \ldots, 0)\ \forall i \in 1, \ldots, n$. Putting those values into Eq. 2, we get:

$$\begin{aligned} f(\mathbf{v}_i^+) &= w_{n-i+1} + w_{d-i}\sqrt{2c} + b', \\ f(\mathbf{v}_i^-) &= w_{n-i+1} - w_{d-i}\sqrt{2c} + b'. \end{aligned} \tag{3}$$

Next, we compute the following:

$$w_{n-i+1} = ((w_{n-i+1} + w_{d-i}\sqrt{2c} + b') + (w_{n-i+1} - w_{d-i}\sqrt{2c} + b') - 2b')/2.$$

As we can do this for all values $i \in 1, \ldots, n$, we can get all w_{n-i+1} which is equivalent to w_n. If we instead subtract $f(\mathbf{v}_i^+)$ and $f(\mathbf{v}_i^-)$ and divide by 2, we end up with:

$$\sqrt{2c}w_{d-i} = ((w_{n-i+1} + w_{d-i}\sqrt{2c} + b') - (w_{n-i+1} - w_{d-i}\sqrt{2c} + b'))/2.$$

Looking back at Eq. 2, we know that the weights $\forall i \in 1, \ldots, n : w_{d-i}$ always will be multiplied by $\sqrt{2c}$. Therefore, for our reconstruction of $f'(\mathbf{x})$ we can set $\forall i \in 1, \ldots, n : w'_{d-i} = \sqrt{2c}w_{d-i}$, so that we do not have to find the value of c.

Now, all we need to reconstruct Eq. 2 are w_n to w_{d-n-1}, or these values multiplied by $\sqrt{2}$. To calculate them, we have to go through each combination of two ones in $\mathbf{v}$ and fix all other values at zero. Querying for those $\mathbf{v}$ we get:

$$f(\mathbf{v}) = w_{n-i+1} + w_{n-j+1} + \sqrt{2}w_r + w_{d-i}\sqrt{2c} + w_{d-j}\sqrt{2c} + b',$$

with r being calculated as $r = \sum_{i=1}^{n-s+1}(n-i) - t + n + 1$, where $s = \max(i, j)$ and $t = \min(i, j)$. By subtracting all the known values, we can easily get the last unknown coefficients w_r. With the extracted values we can now construct an identical regression function $f'(\mathbf{x})$, which however is less efficient than using the kernel trick, because instead of the more efficient kernel function $(\langle \mathbf{x}, \mathbf{x}' \rangle + 1)^2$, the dot product has to be calculated explicitly, which takes $n + \binom{n}{2} + d$ multiplications and $d - 1$ additions. Because we do not know the parameter c explicitly, as we use it integrated in the weights $\forall i \in 1, \ldots, n : w'_{d-i}$ and in b', we have to adjust the transformation function that we are using accordingly:

$$\phi'_{quadratic}(\mathbf{x}) = \begin{cases} x_n^2, \ldots, x_1^2, \\ \sqrt{2}x_n x_{n-1}, \ldots, \sqrt{2}x_2 x_1, \\ x_n, \ldots, x_1, \\ 0 \end{cases}$$

and our extracted weights are

$$\mathbf{w}' = \begin{cases} w_n, \ldots, w_1, \\ w_{n+1}, \ldots, w_{d-n}, \\ \sqrt{2c}w_{d-n+1}, \ldots, \sqrt{2c}w_{d-1}, \\ 0. \end{cases}$$

The resulting extracted equation looks then as follows: $f'(\mathbf{x}) = \langle \mathbf{w}', \phi'_{quadratic}(\mathbf{x}) \rangle + b'$. This method uses a total of $d = \binom{n}{2} + 2n + 1 = \frac{1}{2}n^2 + \frac{3}{2}n + 1$ queries, which is in $O(n^2)$, where n is the number o features.

5.3 SVR Model Extraction with Retraining for Arbitrary Kernels

Just as SVMs, SVR model parameters can be extracted, or rather approximated using a retraining strategy. By having the model provider label a set of samples, its predictions can be used as training data for the attacker's model. The most simple approach is to have the model provider label a set of uniformly random samples. The accuracy of this approach largely depends on which samples get picked by the algorithm, as some samples contribute more to the extraction accuracy than others. A larger number of samples therefore increases the chance of extracting an accurate model.

5.4 SVR Model Extraction with Adaptive Relearning for Arbitrary Kernels

Using the adaptive retraining approach, the amount of queries used to extract an accurate model can be reduced. Instead of picking a random set of samples to get labeled by the model provider, the samples are picked carefully to achieve a maximum improvement of the extracted model. Having a total query budget m, this is done by performing r rounds on $n = m/r$ samples being labeled at once and calculating the ideal set of samples of the next round in between. The first set of n samples to be labeled is picked at random and sent to the API. An initial model is being trained on the results obtained from the API. We know that for SVR only support vectors

influence the model. We use the approach by Douak et al. [10] of picking samples by their distance from the support vectors. In other words, we can take the samples that are located the furthest from our current support vectors and rank them by the importance of the closest support vector, which is determined by the support vector coefficients values' closeness to the model parameter c.

This algorithm can be rather slow, because in every round the SVR has to be recalculated. The algorithm's runtime efficiency also depends on the number of samples n to be labeled per round. Rounds with more samples imply that calculation of a smaller amount of in-between models is necessary, however, higher amounts of samples per round also mean, that every round more samples are chosen on grounds of an inaccurate model. The algorithm's efficiency can be increased using online learning techniques for SVRs [25]. Online learning techniques enable the incremental addition of new training data to an already fitted model, removing the need to retrain the model on the whole set of training data. Another approach to increasing the algorithm's efficiency would be incrementing the round size for each round. Since with each round the SVR gets more precise, we can increase the number of samples to predict, as they rely on a more precise model. Using exponential increments, the total number of rounds, and therefore the total number of models that have to be calculated, can be reduced significantly.

5.5 Kernel Agnostic Extraction

Even though the classic and adaptive retraining approaches can be applied to any kernel, the attacker has to know which kernel he is attacking, i.e., they are white-box attacks. This can be circumvented by employing a kernel agnostic algorithm, i.e., an algorithm that makes no difference in which kernel it is attacking.

Such an algorithm can be constructed by performing a classic (randomized) retraining attack, but training multiple models at once, each with a different kernel. Then, using a test set, the algorithm compares the predictions of each model to the original one. The model with the lowest error rate is then picked and assumed as the "correct" one. This approach would not need an additional amount of queries, as it can train all models on the same set. However, the computation time increases, because multiple models have to be calculated instead of just one. In practice, there are only a handful of classic kernels that see widespread use (see Tab. 1), which can be parametrized with classic approaches to setting the model parameters. Therefore, the number of models that have to be trained and compared is limited.

Theoretically, it would be possible to also use adaptive training methods for this approach. Then, the training would be divided into multiple rounds. The first round would use a randomized set of samples to train the initial kernels. In each subsequent round, the samples that each kernel is the least certain about are selected into a subset S. Afterwards, the samples within this subset are ranked by how many models are how uncertain about them. The queries that the most models are the most uncertain about are selected to be queried and trained on. Problematic is that in every round a number of models have to be trained simultaneously. Even training a single model can be very slow and although the models can be trained in parallel, to get the next round of queries, we have to wait until all models are done.

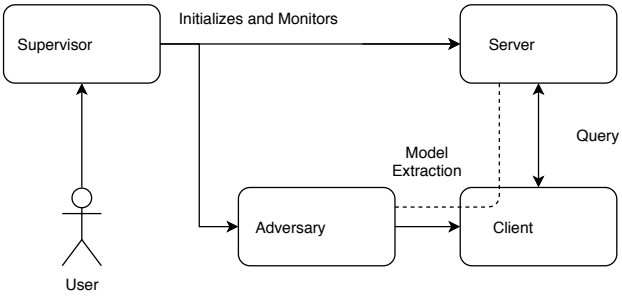

Figure 1: Architecture of our model extraction attack framework.

6 OUR FRAMEWORK FOR MODEL EXTRACTION ATTACKS

For studying different attacks and finding feasible attack strategies, we implemented a simulator for the MLaaS paradigm in Python. Our simulator allows to monitor and tweak all parameters and test different attacks in an environment, where there are no legal implications to our attacks. This simulator enables us to compare extracted models with the original ones in terms of prediction accuracy and quantify the quality of the extractions. Note that our implementation works entirely locally by simulating the required network latency, whereas a real-world attack on MLaaS would be performed over a real network. Our implementation is open-source and available at https://github.com/robre/attacking-mlaas.

Architecture

In the following, we give the architecture of our implementation and show how it can be used to simulate an attack scenario.

Our implementation consists of four classes: Server, Client, Adversary, and Supervisor (see Fig. 1). The Server class represents an MLaaS provider and stores different trained ML models, providing answers to classification or regression requests. The Client class imitates a client who interacts with such a server. The Adversary class uses a client object to interact with the server and contains all the attack algorithms to extract the model. The Supervisor class is used to initialize all the classes and functions, to keep track of data, and to compare results by calculating errors. Apart from these classes, we have a main file that features a csv reading function for parsing the inputs and is the point of initialization and interaction. The user only interacts with the Supervisor class. The classes' detailed functionalities are described below.

The Server Class. The Server class represents an MLaaS provider or a similar service that offers ML classification and regression via an Application Programming Interface (API). Its basic functionality is to store machine learning models, such as SVMs or SVRs, and to provide an interface for clients to query these models. The Server class implements methods that allow to add, delete, get, and query ML models from its database.

The Client Class. The Client class mimics a user of an MLaaS. Its only functionality is to poll a server object with data, asking for prediction on this data. It also implements a query counter, which

counts the total number of queries issued by a client. To derive a time estimate in a real-world scenario, the query count can be multiplied by the average round-trip time for a packet to reach an MLaaS provider and added to the server calculation time and the client calculation time.

The Adversary Class. The Adversary class represents an attacker who intends to extract model parameters from MLaaS providers. It provides an attack method which wraps all attacks that are implemented and returns a recreation of the attacked model derived from the results of the attacks. For the attack method, a number of variables are in place so that attack type, kernel, maximum number of queries, and attack specific variables may be tweaked to improve the results. To interact with the Server class, the Adversary controls an instance of the Client class. It features the option to attack either an SVM or an SVR. An SVM can be attacked using the Lowd-Meek attack (see §4), retraining, or adaptive retraining (see §4.2). An SVR can be attacked using a set of our newly devised algorithms (see §5). These new algorithms include an equation-solving attack which extracts the exact parameters of a linear SVR (see §5.1) and for quadratic kernels (see §5.2), and furthermore, SVRs can be attacked using retraining (see §5.3) and adaptive retraining (see §5.4).

The Supervisor Class. The Supervisor class provides utilities to run and analyze different scenarios. All interaction from the user goes through the supervisor class which features methods to generate training data, create models from random or given data, to compare predictions of the actual model with the extracted model, and calculate errors. This class also creates plots for visualization (e.g., those in Fig. 2 and Fig. 3).

7 EVALUATION

To assess the feasibility of our model extraction attacks, we need to determine what makes an attack successful and what is the cost of making an attack succeed. This is done for different kernels as they differ in extraction complexity. We define metrics for the quality of the extracted models and the costs associated with the extraction. We then perform extractions with different data, kernels, and techniques. We compare the measurements across the extractions and give a conclusion on the feasibility of our proposed model extraction attacks.

7.1 Model Approximation Accuracy

An extraction attack produces an approximation of a model. The approximation accuracy can be assessed by comparing the prediction results of the extracted model to that of the original model.

Approximation Accuracy of SVM Extraction. SVMs produce labels 0 or 1 as predictions. An extracted prediction therefore can be either correct or wrong. To assess the quality of an extracted SVM, the quota of wrong predictions in a test set can be calculated as

$$P_{wrong} = \frac{\#wrong\ predictions}{\#total\ predictions}. \tag{4}$$

We generally try to achieve a minimal percentage of wrong classifications. Extractions with $P_{wrong} \leq 1\,\%$ are considered sufficient and $P_{wrong} \leq 0.1\,\%$ very good as described by Tramèr et al. [48].

Approximation Accuracy of SVR Extraction. For SVRs, the predictions are continuous values. Therefore, the Mean Squared Error (MSE) can be calculated as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(o_i - p_i)^2. \tag{5}$$

To get a comparable error value between different models with differently normed data, we can use the Relative Mean Squared Error (RMSE) calculated as

$$RMSE = \frac{\sum_{i=1}^{n}(o_i - p_i)^2}{\sum_{i=1}^{n}(\bar{o} - o_i)^2}, \tag{6}$$

where $\bar{o}$ denotes the mean of o.

7.2 Cost Factors

A model extraction can be further quantified by considering cost factors for the attacker. If the monetary cost associated with these cost factors exceeds the financial value of the extracted model, there is no financial incentive to perform an extraction, eliminating the principal motivation for the attack. Consequently, we measure the subsequent cost factors. We consider the number of queries as the most implementation-independent cost metric.

Query Cost. A classic business model for MLaaS is to train a private model and give users the possibility to query this model for predictions, charging them a fixed amount of money or tokens per query. A financially motivated attacker could try to extract a model to circumvent the future query costs or even sell his extracted model to third parties. This type of attacker would have a query budget m of queries that they would be willing to pay in order to extract the model. Hence, low query costs increase the attack feasibility.

Time Costs. Another cost factor is runtime. The extraction can be considered infeasible if the runtime associated with it is very high. The extraction runtime has two main factors: local calculation time and query latency. The local calculation time is the amount of time needed for all the necessary computations performed by the attacker during an extraction. The query time considers all the time factors that play a role when querying the server for a prediction. The query time consists of the network time, which is the time that each query has to travel through the network, reach the server, and come back with the answer, and the server-side calculation time, which is the time, the server needs to compute a prediction on a query. The time costs significantly depend on the specific implementations — which can often be optimized — and network setting, and we consider it for only a few examples.

7.3 Datasets

For our evaluation, we use a number of datasets from different sources. We can differentiate the datasets by several factors: firstly, if the dataset is for classification (SVM), or regression (SVR). Secondly, if the dataset was generated artificially, or consists of natural data. Thirdly, the amount of features. Lastly, the size of the dataset and test set. The real datasets we used are listed in Tab. 2.

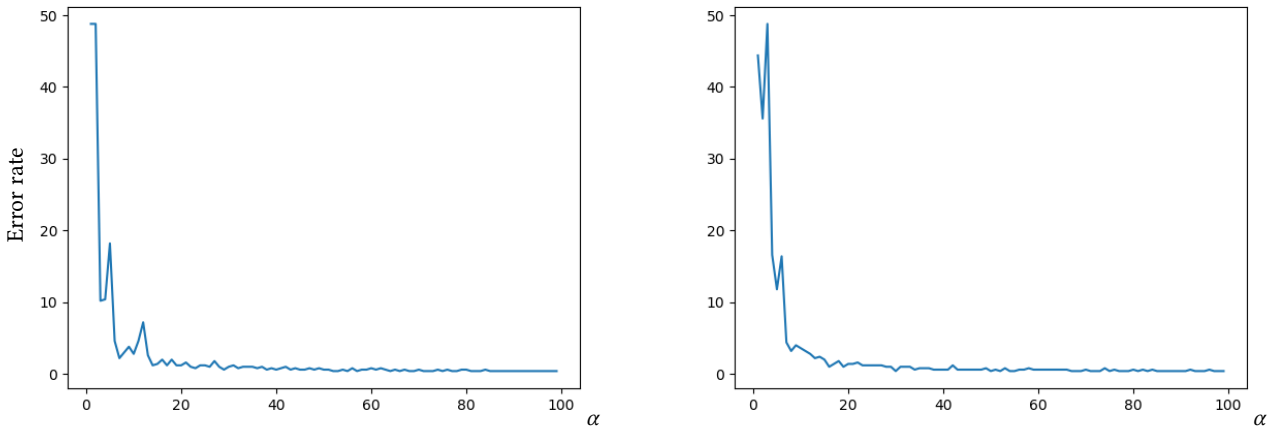

Figure 2: Error rate (see Eq. 4) for different values of budget factor α for randomized (left) and adaptive (right) retraining for Radial Basis Function Support Vector Machines (RBF SVMs).

Table 2: Natural datasets for regression used in this work.

id	Dataset Name	Features	Size
1	California Housing [31]	8	20 640
2	Boston House Prices [30]	13	506
3	UJIIndoorLoc [47]	520	19 937
4	IPIN 2016 Tutorial [39]	168	927

7.4 Attacker Model

In our attacker model, we assume a financially motived attacker, such as a competitor, who has the intention to ultimately be able to predict data himself without paying the MLaaS provider. Therefore, we can assume that the attacker has access to a substantial set of unlabeled training data. Particularly, we can also assume that in case of a classification task, the attacker has access to at least one positive and one negative sample, an assumption that has also been made by Lowd and Meek [26], which is crucial to the extraction attack.

We note that an attack without any knowledge of the data to be predicted, e.g., no knowledge of what each feature is and what values it may take, is theoretically possible. In low-dimensional settings, values can be guessed until a negative and positive instance is found. When one classification however is a rare case with high dimensions, say the prediction that a certain type of cancer is present, and derived from a feature vector of 30 features, a very specific alignment of values is necessary to get this prediction. Finding a vector that gets classified as cancer with random values is highly unlikely, therefore the attacker's knowledge of the data he wants to have predicted is crucial.

7.5 Benchmarking Results of our Attacks

For each setting, we first perform an attack using the best, i.e., most accurate technique we know of. Then, we perform retraining attacks with a similar and smaller number of queries, and finally the most accurate attacks, because it only makes sense to use the retraining attacks in settings where we either do not have a high enough query budget for the best attack types or where only a retraining approach is available. As our implementation is local, the runtimes for each extraction have an optimal latency of near zero. To arrive at more realistic estimates for the runtimes, we add a 100 ms network latency for each query. We denote this setting as Internet latency. Some services limit the number of consecutive queries for a user in a specific time frame. We assume that the attacker has a sufficient query budget. If the number of allowed queries is too low for performing an accurate attack, the attacker could either use multiple accounts eventually using different IP addresses or cooperates with other attackers.

Extraction of Linear SVMs. To the best of our knowledge, the best known attack on linear SVMs is the Lowd-Meek attack [26], which can extract a model with an adjustable accuracy of ϵ. We create 100 random classification problems and perform an attack on them using the Lowd-Meek attack with $\epsilon = 0.01$ and $\delta = 0.01$.

We find, that the extraction takes on average $17 \cdot n$ queries for n features. For 2 features, the extraction took on average 35 queries, for 9 features 154 queries and for 100 features 1 700 queries. The execution time was 3 ms for 2 features and stayed under 1.5 s for 1 000 features. At 1 000 features, the total amount of queries however is about 17 000, which would take an additional $17\,000 \cdot 0.1\,\text{s} = 1\,700\,\text{s}$ (29 minutes) in the Internet latency setting. The extracted models had 100% accuracy when comparing them to the original ones.

Next, we perform an extraction on the same 100 models using retraining and a budget factor $\alpha \leq 17$, so that we get a comparable maximum $17 \cdot (n + 1)$ queries per attack. We find that we can extract 100% accurate models in about 0.01 s. Lowering α did not affect the accuracy — we could extract 100% accurate models with $\alpha = 1$. For adaptive retraining, we get the same results. This can be explained because linear SVMs are Probably Approximately Correct (PAC) learnable [49] as their Vapnik-Chervonenkis dimension (VC

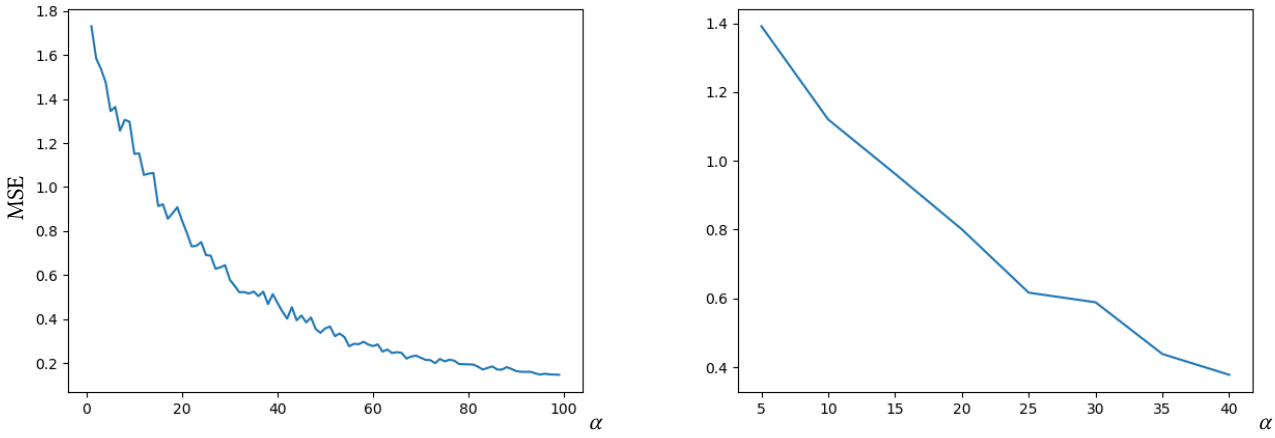

Figure 3: Mean Squared Error (MSE) (see Eq. 5) for different values of budget factor α for randomized (left) and adaptive (right) retraining strategies for Radial Basis Function Support Vector Regression Machines (RBF SVRs).

dimension) is $n + 1$ [50]. RBF kernels however have an infinite VC dimension, and are therefore not PAC learnable.

Extraction of RBF SVMs. To simulate the extraction of an SVM using the RBF kernel, we create a 20-dimensional dataset with 1 500 samples using the first 500 to train the original model, the next 500 for the attacker, and the last 500 as the test set. In Fig. 2, we plot the error percentage in dependence of the budget factor α for randomized and adaptive retraining. We can see that the accuracies are very similar, being slightly more stable for adaptive retraining. At about $\alpha = 20$ we get a sufficiently accurate extraction with 99% accuracy. Concretely, for 20 features, this means we need about $\alpha \cdot (n+1) = 20 \cdot (20+1) = 420$ queries to extract an accurate model. However, the attacks vary noticeably in speed. While the randomized retraining attack takes up to 3 s, the adaptive retraining took up to 70 s per attack. Considering 100 ms Internet latency, such an attack would take about $70\,\text{s} + 420 \cdot 0.1\,\text{s} \approx 2\,\text{min}$. Please note that our Python implementation can be optimized to improve its runtime, e.g., by using low-level programming languages such as C++.

Extraction of Linear SVRs. We use three different datasets for the extraction of linear SVRs: the California housing dataset, Boston house-prices dataset, and an artificial dataset with 100 features.

First, we employ the equation solving attack described in §5.1. Attacking a linear model trained on the California housing dataset, we achieved an exact extraction of the model parameters using a total of 9 queries. The processing for the attack took 1 ms (0.9 s with Internet latency). Attacking the Boston House-prices dataset, we got an exact extraction with 14 queries in 1 ms (1.4 s with Internet latency). Lastly, extracting the generated dataset took 101 queries and 59 ms (10.2 s with Internet latency).

For retraining strategies to make sense, we need to set $\alpha \leq 1$. However, even with $\alpha = 1$, the highest setting, we get error rates of $RMSE > 2$ when using randomized retraining. Using adaptive retraining, the error rates get better, at $RMSE > 0.02$ for $\alpha = 1$. Still, we do not consider extractions with such error rates as good.

Extraction of Quadratic SVRs. For the extraction of quadratic SVRs, we use the same three datasets as in §7.5, but trained with a quadratic kernel. We use the equation solving attack described in §5.2 first. Using this attack, the extraction of the model trained on the California housing dataset took a total of 45 queries and 7 ms (4.5 s with Internet latency). Extracting the model trained on the Boston house-prices dataset took 105 queries and 6 ms (10.5 s with Internet latency). Extracting the model trained on the generated dataset with 100 features took 5 151 queries and 0.589 s (516 s, or about 9 minutes with Internet latency).

The optimal attacks query counts are comparable to setting $\alpha \in [9, 50]$ for the retraining approach. Actually, no concrete value for α can be determined, because the equation solving attack uses a quadratic amount of queries. However, using retraining attacks, we found that the error was unproportionally high when extracting the natural datasets. Furthermore, at $\alpha = 11$, the extraction took over 20 minutes (without Internet latency) when using adaptive retraining and the generated dataset with $RMSE > 0.8$. This is due to the fact that retraining a quadratic kernel SVR is slow.

Extraction of RBF SVRs. For the extraction of Radial Basis Function (RBF) SVRs, we again use the same three datasets as for Quadratic SVRs trained with an RBF kernel. As we do not have an equation-solving or similar attack for RBF kernels, we commence with the retraining approach. In Fig. 3, the MSE is shown in dependence of the used α for the randomized retraining approaches. We can see that for the artificial dataset, the MSE decreases as we increase α. For the natural datasets however, the MSE fluctuates within a specific area relatively unaffected by increasing α. In Fig. 3, the MSE is shown in dependence of the used α for the adaptive retraining approach. In comparison to the randomized retraining, we can see that now there is a trend for all datasets, natural and synthetic, to have a lower MSE with increasing α. Note that we conducted the experiment for α in steps of 5 for all datasets, and capping at $\alpha = 45$ for the artificial dataset, because the extraction took up to 20 978 s (almost 5 hours). For the natural datasets, with lower dimensions,

the adaptive retraining took up to 800 s (14 minutes, or 17 minutes with Internet latency). The randomized retraining was significantly faster at a maximum of 0.6 s, or 3 minutes with Internet latency.

Extraction of Wi-Fi Localization Models . Here, we show a practical attack on an implementation of SMPC-protected MLaaS for indoor localization from Wi-Fi fingerprints using SVRs which is very similar to [56]. Our implementation guarantees even stronger security of the ideal functionality than [56] due to the higher number of access points.

Zhang et al. [56] described an SMPC protocol to protect the privacy of the user and model provider in an MLaaS setting, where the server provides a localization service to clients using Wi-Fi signal strengths. The server uses SVRs to predict the client's location. In their scheme, the server trains an arbitrary number of SVRs on different subsets of features and returns the prediction of a randomly chosen SVR. This makes it impossible to extract the one exact model the service uses because there is no explicit single model. However, all of their SVRs predict approximately the same values, so from an outside perspective they can be regarded as one single implicit SVR with a small rounding error, which does not effectively prevent extraction attacks (see [48]). Therefore, our extraction efforts are targeted on this implicit model. To simulate an implicit model for indoor localization using Wi-Fi fingerprints, we can simply create an explicit model with the same function as there is no difference to the outside attacker. The scheme that Zhang et al. proposed, uses as few as 4 access points, however we found that a similar dataset with such a low number of access points was unattainable. We use two real-world publicly available datasets for indoor localization instead: the "UJIIndoorLoc" dataset [47] with 520 access points, and the "IPIN 2016 Tutorial" dataset [39] with 168 access points. We train our SVRs using the RBF kernel and a set of 100 training samples and use the rest of the data for the attacks and evaluation. We attack each model using adaptive retraining with low α values of 1 and 5. Because of the high dimensions, the extractions using higher values for α would take multiple hours and the accuracy using the low α values is already very high.

For the "UJIIndoorLoc" dataset, we get an extremely low error of $4.24 \cdot 10^{-6}$ with an $\alpha = 1$ and an error of $6.36 \cdot 10^{-7}$ with an $\alpha = 5$. The attacks took up to 1 368 s for computation and 2 605 queries. This attack would run in about 1 629 s (28 minutes) with a 100 ms Internet latency. We get a similar result for the "IPIN 2016 Tutorial" dataset, with an error of $1.06 \cdot 10^{-4}$ at $\alpha = 1$ and an error of $5.16 \cdot 10^{-7}$ at $\alpha = 5$. This attack took 165 s of calculation time and a total of 845 queries, translating to a total of 250 s, or 5 minutes of runtime with a 100 ms Internet latency. Consistent among our experiments, we saw that a lower number of features results in a faster extraction. Considering that this scheme might use much fewer features than the datasets we conducted our experiments on, we can deduct that this attack is much faster in a realistic scenario.

Note that an algorithm to protect Sigmoid kernel SVRs was proposed by Zhang et al. [56]. However, we found it difficult to achieve consistent results when training models using it. This might be due to the fact that not every Sigmoid kernel is a valid kernel, and parameters have to be set precisely. Furthermore, the Sigmoid kernel behaves more or less similar to the much more common RBF kernel, which we explored instead.

7.6 Summary

Our evaluation results are summarized in Tab. 3). They show that firstly, the attacks on SVMs by Lowd and Meek [26], and Tramèr et al. [48] are very effective and allow an attacker to extract accurate models with a relatively small number of queries. Secondly, our new equation-solving attacks on SVR proved to be not only very accurate, but also up to 80 times faster than the retraining approaches. Indeed, we deduct that for linear and quadratic SVRs, our equation-solving attacks are the most effective. For RBF-kernel SVRs, our experiments show that the best results are achieved by using an adaptive retraining approach, and that the budget factor α can be set very low, as the extraction does not improve by much for higher values of α, and the extraction using adaptive retraining with a low α consistently has a lower error than using randomized retraining with a higher α.

To translate our results into a real-world scenario, we analyze the extraction of the model trained on the "UJIIndoorLoc" dataset. At $\alpha = 5$, this extraction took a total of $5 \cdot (520 + 1) = 2\,605$ queries and 36 minutes of runtime. Using a cost of \$0,000 1 per query, as is the cost at AmazonML, this extraction would have a total cost of \$0.26. The extraction of the "IPIN 2016 Tutorial" model took just 7 minutes of runtime and 854 queries, translating to a total cost of \$0.09. For bandwidth, Zhang et al. [56] show that their protocol uses $(2n + 6)L$ bits of bandwidth per query, with $L = 2\,048$ and n being the number of features. For the "IPIN 2016 Tutorial" dataset with 168 features this translates to 80 kB per query. The full extraction would have a bandwidth cost of 71 MB. At these cost factors, we can safely assume that this extraction would be very feasible to a financially motivated attacker.

8 COUNTERMEASURES

To help protect from the attacks shown in this paper, we discuss several possible countermeasures that MLaaS can employ to increase the security of their intellectual property, i.e., the ML models.

Rounding. Rounding as a countermeasure was already introduced by Tramèr et al. [48], who proposed rounding confidence values given by MLaaS providers. They found that in some cases the attack was weakened, but the attacks stayed viable in general. Instead of rounding confidence scores, the actual prediction can be rounded in regression problems. This would decrease the accuracy of the equation-solving attacks we proposed, but they would still be usable. Generally, a precise prediction is desirable, which makes this approach unsuitable in some cases.

Extraction Monitor. Kesarwani et al. [22] propose an extraction monitor, which observes the queries issued by multiple users of an MLaaS and gives a warning when the information that a user or a subset of users might deduct from their queries exceeds a certain threshold. This threshold is defined by the average number of queries needed to reconstruct a model with a definable accuracy. For many attacks, such as adaptive retraining, the number of queries can be so low that they submerge in normal traffic. Furthermore, in the case of attacks with more queries, an MLaaS provider has an incentive to keep clients with more queries, as they bring revenue. It can, therefore, be a tough decision whether to cut off a client with more queries, or not, because there might be the possibility of

Table 3: Extraction results for SVM and SVR kernels for real and artificial datasets. Accuracy is calculated as $1 - P_{wrong}$ (see Eq. 4) for Support Vector Machines (SVMs) and as $1 - RMSE$ (see Eq. 6) for Support Vector Regression Machines (SVRs). The attacker runtime is evaluated in our non-optimized simulation environment written in Python.

Type	Kernel	Method	Dataset	# Features	Accuracy	# Queries	Runtime
SVM	Linear	Lowd-Meek [26]	Artificial	100	100%	1 700	9 min
SVM	Linear	Retraining	Artificial	100	100%	1 800	9 min
SVM	Linear	Retraining	Artificial	100	100%	101	30 s
SVM	RBF	Retraining	Artificial	20	99%	420	3 min
SVM	RBF	Adaptive Retraining	Artificial	20	99%	420	4 min
SVR	Linear	Equation Solving	Artificial	100	100%	101	31 s
SVR	Linear	Equation Solving	Boston Housing [30]	13	100%	14	5 s
SVR	Linear	Equation Solving	California Housing [31]	8	100%	9	3 s
SVR	Quadratic	Equation Solving	Artificial	100	100%	5 151	26 min
SVR	Quadratic	Equation Solving	Boston Housing [30]	13	100%	105	32 s
SVR	Quadratic	Equation Solving	California Housing [31]	8	100%	45	14 s
SVR	RBF	Retraining	Artificial	100	99%	4 040	21 min
SVR	RBF	Retraining	Boston Housing [30]	13	99%	14	5 s
SVR	RBF	Retraining	California Housing [31]	8	99%	9	3 s
SVR	RBF	Adaptive Retraining	Artificial	100	99%	4 040	7 h
SVR	RBF	Adaptive Retraining	Boston Housing [30]	13	99.9%	14	6 s
SVR	RBF	Adaptive Retraining	California Housing [31]	8	99.9%	9	4 s
SVR	RBF	Adaptive Retraining	UJIIIndoorLoc [47]	520	99.9%	2 605	36 min
SVR	RBF	Adaptive Retraining	IPIN 2016 [39]	168	99.9%	845	7 min

a model extraction going on. Also, they might be an honest client using the service extensively.

Monitoring for Suspicious Queries. An MLaaS could monitor incoming queries for suspicious qualities that may occur specifically in extraction attacks. Such suspicious queries could be zero vectors and unit vectors as they are used in equation solving attacks, vectors that get too close to the decision boundary, as used in the Lowd-Meek attack [26], and vectors that have unusual values for specific features, such as −1, when the feature normally takes values between 1 000 and 10 000. In addition, it is possible to monitor for queries with very unusual statistical distribution. Yet, retraining attacks using real datasets issue queries that are indistinguishable from normal queries, leaving them undetected by such a countermeasure. Furthermore, deploying such check in privacy-preserving systems based on SMPC would incur a significant overhead which contradicts the business model of MLaaS of providing inexpensive predictions.

Server Side Feature Translation. To prevent an attacker from issuing specific queries, such as vectors arbitrarily close to the decision boundary, or zero- and unit-vectors, the server might translate features themselves instead of having the client send translated features. Feature translation means, that non-numerical features, such as strings, get translated into a numerical representation so that they can be used for calculations. This countermeasure also can not prevent retraining attacks, as they do not rely on specifically crafted queries. However, in the case of SMPC when the client does not trust the server with his data and needs to hide the cleartext values on each feature, the feature transformation has to be performed by the client or done within SMPC.

ACKNOWLEDGMENTS

This work was supported by the German Federal Ministry of Education and Research (BMBF) and the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within the National Research Center for Applied Cybersecurity CRISP, and by the DFG as part of project E4 within the CRC 1119 CROSSING and project A.1 within the RTG 2050 "Privacy and Trust for Mobile Users".

REFERENCES

[1] M. Barni, P. Failla, V. Kolesnikov, R. Lazzeretti, A.-R. Sadeghi, and T. Schneider. 2009. Secure Evaluation of Private Linear Branching Programs with Medical Applications. In *ESORICS*. Full version: https://ia.cr/2009/195.
[2] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. 2006. Can Machine Learning be Secure. In *CCS*.
[3] B. Biggio, B. Nelson, and P. Laskov. 2012. Poisoning Attacks against Support Vector Machines. *Machine Learning* (2012).
[4] G. Camps-Valls, J. D. Martín-Guerrero, J. L. Rojo-Alvarez, and E. Soria-Olivas. 2004. Fuzzy Sigmoid Kernel for Support Vector Classifiers. *Neurocomputing* (2004).
[5] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin. 2010. Training and Testing Low-Degree Polynomial Data Mappings via Linear SVM. *Machine Learning Research* (2010).
[6] D. Cohn, L. Atlas, and R. Ladner. 1994. Improving Generalization with Active Learning. *Machine Learning* (1994).
[7] C. Cortes and V. Vapnik. 1995. Support-Vector Networks. *Machine Learning* (1995).
[8] D. Demmler, T. Schneider, and M. Zohner. 2015. ABY-A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *NDSS*.
[9] A. Dmitrenko. 2018. DNN Model Extraction Attacks using Prediction Interfaces. (2018).
[10] F. Douak, F. Melgani, E. Pasolli, and N. Benoudjit. 2012. SVR Active Learning for Product Quality Control. In *Information Science, Signal Processing and their Applications (ISSPA)*.

[11] H. Drucker, C. J. C. Burges, L. Kaufman, A. J. Smola, and V. Vapnik. 1997. Support Vector Regression Machines. In *Advances in Neural Information Processing Systems*.

[12] M. Fredrikson, S. Jha, and T. Ristenpart. 2015. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In *CCS*.

[13] J. Friedman, T. Hastie, R. Tibshirani, et al. 2000. Additive Logistic Regression: A Statistical View of Boosting. *The Annals of Statistics* (2000).

[14] O. Goldreich, S. Micali, and A. Wigderson. 1987. How to Play any Mental Game. In *STOC*.

[15] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. 2016. *Deep Learning*. MIT press Cambridge.

[16] P. Hallgren, C. Orlandi, and A. Sabelfeld. 2017. PrivatePool: Privacy-Preserving Ridesharing. In *Computer Security Foundations*.

[17] J. Hayes, L. Melis, G. Danezis, and E. De Cristofaro. 2019. LOGAN: Membership Inference Attacks against Generative Models. *PETs* (2019).

[18] W. Henecka, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. 2010. TASTY: Tool for Automating Secure Two-Party Computations. In *CCS*.

[19] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant. 2013. *Applied Logistic Regression*. John Wiley & Sons.

[20] K. Järvinen, Á. Kiss, T. Schneider, O. Tkachenko, and Z. Yang. 2018. Faster Privacy-Preserving Location Proximity Schemes. In *CANS*.

[21] K. Järvinen, H. Leppäkoski, E. S. Lohan, P. Richter, T. Schneider, O. Tkachenko, and Z. Yang. 2019. PILOT: Practical Privacy-Preserving Indoor Localization using OuTsourcing. In *EuroS&P*.

[22] M. Kesarwani, B. Mukhoty, V. Arya, and S. Mehta. 2018. Model Extraction Warning in MLaaS Paradigm. *Computer Security Applications* (2018).

[23] B. Kulynych, J. Hayes, N. Samarin, and C. Troncoso. 2018. Evading Classifiers in Discrete Domains with Provable Optimality Guarantees. In *NeurIPS Workshop on Security in Machine Learning*.

[24] S. Laur, H. Lipmaa, and T. Mielikäinen. 2006. Cryptographically Private Support Vector Machines. In *Knowledge Discovery and Data Mining*.

[25] J. Liu and E. Zio. 2016. An Adaptive Online Learning Approach for Support Vector Regression: Online-SVR-FID. *Mechanical Systems and Signal Processing* (2016).

[26] D. Lowd and C. Meek. 2005. Adversarial Learning. In *Knowledge Discovery in Data Mining*.

[27] L. M. Manevitz and M. Yousef. 2001. One-Class SVMs for Document Classification. *Machine Learning Research* (2001).

[28] S. Mika, G. Ratsch, J. Weston, B. Scholkopf, and K.-R. Mullers. 1999. Fisher Discriminant Analysis with Kernels. In *Neural Networks for Signal Processing*.

[29] J. H Min and Y.-C. Lee. 2005. Bankruptcy Prediction using Support Vector Machine with Optimal Choice of Kernel Function Parameters. *Expert Systems with Applications* (2005).

[30] K. Pace. 1999. *Boston House Prices Dataset*. http://lib.stat.cmu.edu/datasets/boston_corrected.txt

[31] K. Pace. 1999. *Califonria Housing Dataset*. http://lib.stat.cmu.edu/datasets/houses.zip

[32] P. Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT*.

[33] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami. 2017. Practical Black-Box Attacks Against Machine Learning. In *ASIACCS*.

[34] K. Polat and S. Güneş. 2007. Breast Cancer Diagnosis using Least Square Support Vector Machine. *Digital Signal Processing* (2007).

[35] J. R. Quinlan. 1986. Induction of Decision Trees. *Machine learning* (1986).

[36] Y. Rahulamathavan, R. C.-W. Phan, S. Veluru, K. Cumanan, and M. Rajarajan. 2014. Privacy-Preserving Multi-Class Support Vector Machine for Outsourcing the Data Classification in Cloud. *Dependable and Secure Computing* (2014).

[37] B. I. P. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S. Lau, S. Rao, N. Taft, and J. D. Tygar. 2009. Antidote: Understanding and Defending against Poisoning of Anomaly Detectors. In *Internet Measurement*.

[38] D. W. Ruck, S. K. Rogers, M. Kabrisky, M. E. Oxley, and B. W. Suter. 1990. The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function. *Neural Networks* (1990).

[39] A. R. J. Ruiz, G. M. Mendoza-Silva, R. Montoliu, F. Seco, and J. Torres-Sospedra. 2016. *IPIN 2016 Tutorial Dataset*. http://indoorloc.uji.es/ipin2016track3/

[40] A.-R. Sadeghi and T. Schneider. 2008. Generalized Universal Circuits for Secure Evaluation of Private Functions with Application to Data Classification. In *Information Security and Cryptology*.

[41] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes. 2019. Ml-leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models. In *NDSS*.

[42] J. Schmidhuber. 2015. Deep Learning in Neural Networks: An Overview. *Neural Networks* (2015).

[43] Y. Shi, Y. Sagduyu, and A. Grushin. 2017. How to Steal a Machine Learning Classifier with Deep Learning. In *Technologies for Homeland Security*.

[44] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. 2017. Membership Inference Attacks against Machine Learning Models. In *S&P*.

[45] A. J. Smola and B. Schölkopf. 2004. A Tutorial on Support Vector Regression. *Statistics and Computing* (2004).

[46] J. A. K. Suykens and J. Vandewalle. 1999. Least Squares Support Vector Machine Classifiers. *Neural Processing Letters* (1999).

[47] J. Torres-Sospedra, R. Montoliu, A. Martínez-Usó, T. J. Arnau, J. P. Avariento, M. Benedito-Bordonau, and J. Huerta. 2014. *Multi-Building Multi-Floor Indoor Localization Database*. https://archive.ics.uci.edu/ml/datasets/ujiindoorloc

[48] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. 2016. Stealing Machine Learning Models via Prediction APIs. In *USENIX Security*.

[49] L. G. Valiant. 1984. A Theory of the Learnable. *Communications of the ACM* (1984).

[50] V. Vapnik, E. Levin, and Y. L. Cun. 1994. Measuring the VC-dimension of a Learning Machine. *Neural Computation* (1994).

[51] B. Wang and N. Z. Gong. 2018. Stealing Hyperparameters in Machine Learning. *arXiv preprint arXiv:1802.05351* (2018).

[52] Q. Wu and D.-X. Zhou. 2005. SVM Soft Margin Classifiers: Linear Programming versus Quadratic Programming. *Neural Computation* (2005).

[53] Z. Yang and K. Järvinen. 2018. The Death and Rebirth of Privacy-Preserving WiFi Fingerprint Localization with Paillier Encryption. In *INFOCOM*.

[54] A. C.-C. Yao. 1986. How to Generate and Exchange Secrets. In *FOCS*.

[55] H. Yu, X. Jiang, and J. Vaidya. 2006. Privacy-Preserving SVM using Nonlinear Kernels on Horizontally Partitioned Data. In *Applied Computing*.

[56] T. Zhang, S. S. M. Chow, Z. Zhou, and M. Li. 2016. Privacy-Preserving Wi-Fi Fingerprinting Indoor Localization. In *International Workshop on Security*.

[57] F. Ö. Çatak. 2015. Secure Multi-Party Computation Based Privacy Preserving Extreme Learning Machine Algorithm over Vertically Distributed Data. In *Neural Information Processing*.