TECHNISCHE
UNIVERSITÄT
DARMSTADT

Bachelor Thesis

Silently Learning your Support Vector Machines Models

Robert Reith

October 31, 2018

Engineering Cryptographic Protocols Group
Department of Computer Science
Technische Universität Darmstadt

Supervisors: M.Sc. Oleksandr Tkachenko
Prof. Dr.-Ing. Thomas Schneider

Erklärung zur Abschlussarbeit
gemäß §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Robert Reith, die vorliegende Bachelor Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Thesis Statement
pursuant to §23 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I, Robert Reith, have written the submitted Bachelor Thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

In the submitted thesis the written copies and the electronic version for archiving are identical in content.

Darmstadt, October 31, 2018

Robert Reith

Abstract

As Machine-Learning-as-a-Service (MLaaS) is a growing paradigm in the Machine Learning (ML) landscape, more and more ML models get uploaded into the cloud to be accessible from all over the world. Creating good ML models, however, can be expensive and the used data can be sensitive. New Secure Multi-Party Computation (SMPC) protocols for MLaaS have been proposed to protect sensitive user data and models while preserving the privacy of the model providers and their customers. In this thesis we show that for a subset of ML models used in the MLaaS paradigm, Support Vector Machine (SVM) and Support Vector Regression Machine (SVR), it is possible for malicious users to extract the private models, even when they are protected by an ideal implementation of an SMPC protocol, using known and newly devised model extraction attacks. We show that the attacks are not only theoretically possible, but also practically employable and cheap, making them interesting to financially motivated attackers such as competitors or customers. For one particular case, an SMPC protocol introduced by Zhang et al.(International Workshop on Security 2016), which aims at protecting privacy in an MLaaS scheme using mainly RBF kernelSVRs, we show that it is possible to extract a highly accurate model using 854 queries costing a total of $0.09, and that such an attack would take 7 minutes. Knowing the nature of the attacks, we propose a few possible countermeasures to protect ML models.

Acknowledgments

I would like to express my sincerest gratitude and appreciation to my supervisor Oleksandr Tkachenko, for his extended guidance, useful comments, and vast support throughout the thesis, and professor Thomas Schneider, for overseeing this work, and his valuable feedback.

I would also like to wholeheartedly thank my family and friends for supporting me throughout my studies.

Contents

1 Introduction

Secure Multi-Party Computation (SMPC) aims to protect users' privacy in joint computations performed with distrustful remote parties. This means that data provided for calculations by each party stays private, and only the result of the computation is revealed. Machine Learning (ML) is a term that summarizes different algorithms of deriving models from data to make predictions. Such predictions can be either categorical, such as predicting if a picture contains a cat or dog, or real-valued, e.g., predicting a stocks value. One such method are Support Vector Machines (SVMs) for categorical predictions, also called classification tasks, and Support Vector Regression Machines (SVRs) for real-value predictions, called regression tasks. As ML needs high amounts of processing power, new service providers have emerged, 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 their users to monetize their models by letting other users query them for predictions. This business model is a classic multi-party computation task, making it an area of interest for SMPC, as model providers want to keep their intellectual property private and their users might want to keep their concrete data to themselves. To provide privacy for MLaaS, SMPC protocols have been introduced (e.g., [ZCZL16] for SVRs, [YJV06; LLM06; RPV+14] for SVMs, [Çat15] for Extreme Learning Machines) that allow for remote predictions without directly revealing the model to the user, and the user's data to the model provider. However, Tramèr et al. [TZJ+16] showed for different ML algorithms, including SVMs, that it is possible for users to steal accurate models, using the only prediction APIs. We transfer their findings to the case of SVRs, as used in SMPC protocols and show that these protocols fail to protect the model provider's privacy as they allow extract the model. Our attacks target perfect implementations of MLaaS schemes — hence we show that MLaaS schemes are inherently vulnerable to extraction attacks. Concretely, we show that an attacker can extract an Radial Basis Function (RBF) kernel SVR model as described in [ZCZL16] within 7 minutes for a cost of $0.09 and 71 MByte of bandwidth querying the SMPC based protocol.

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

1.1 Motivation

In the paper by Zhang et al. [ZCZL16], 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 a building.

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 try stealing 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 intend to show that the privacy of MLaaS providers is not given, even when employing SMPC protocols such as [ZCZL16]. With our findings we show the need of improved protection mechanisms in the MLaaS paradigm.

1.2 Support Vector Machines and Support Vector Regression Machines Basics

SVMs [CV95] and SVRs [DBK+97] belong to a category of ML 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 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 and 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.

SVMs are binary linear classifiers. This means they work by splitting the input space into two categories using a hyperplane. Data found on one side of the hyperplane are classified as positive, and data found on the other side of the hyperplane are classified as negative (see Figure 1.1a). While for binary linear classifiers any hyperplane that accurately separates the data is valid, SVMs use the hyperplane which maximizes the distance to the closest input data of each category. The closest input data to the hyperplane are called Support Vectors — hence the name Support Vector Machine. The buffer between the hyperplane and the Support Vectors containing no data is called the margin. Note that training data that are not Support Vectors do not influence the hyperplane or margin.

New input data **x** are classified by a function

$$f(\mathbf{x}) = sign(\langle \mathbf{w}, \mathbf{x} \rangle + b), \tag{1.1}$$

which returns either 1 or 0, depending on which side of the hyperplane the input data are (for further details, see Section 2.2). In most cases however, the input data can not be simply separated with a hyperplane. For this case, SVMs introduce further concepts. Firstly, a soft margin can be used, which denotes a margin that can contain samples, as shown in Figure 1.1b. A soft margin penalizes samples within the margin when scoring possible solutions while calculating the optimal hyperplane. Hyperplanes with bigger margins and less samples within the margin get better scores. Secondly, kernels can be used. Kernels transform the input space into a higher dimension, where a linear separation might be possible. For instance, when using the RBF kernel which is a proximity function, samples that are close to a certain point get separated from samples that are far from that point (see Figure 1.2).

SVRs transfer the concept of Support Vectors to a regression task. In SVR, there is no hyperplane to be found. Instead, a linear function is fitted to the training data (a kernel can be used to fit nonlinear functions). Furthermore, instead of the margin, a tube with width ϵ is defined containing all training data, or most of them in the case of a soft margin (see Figure 1.3). The size of the tube is the maximum error with which the fitted function should assimilate the training data. An SVR intends to find a function with a minimal ϵ. To predict new data, a linear SVR uses the function:

$$f(\mathbf{x} = \langle \mathbf{w}, \mathbf{x} \rangle + b. \tag{1.2}$$

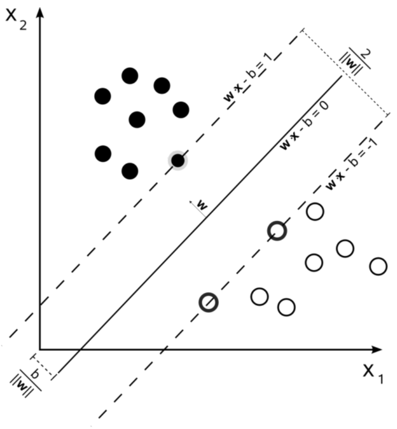
To account for errors in the training data and data that are not linear, the concepts of a soft-margin and kernels can be applied to SVR (for further details, see Section 2.3).

1.3 General Attacks on Machine Learning Algorithms

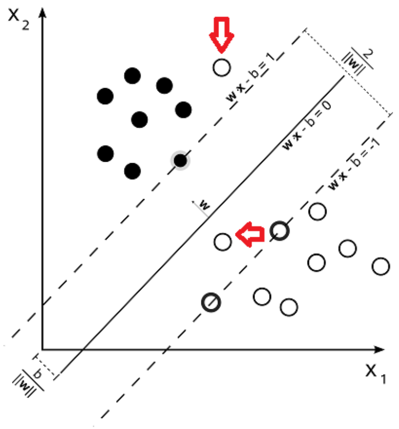
Machine Learning algorithms can be attacked and manipulated in many different ways. In [BNS+06], Barreno et al. propose a separation of attacks into two types: causative and exploratory. A relatively new type of attacks, evasion attacks, was introduced later in [SSG17] and [WG18].

1.3.1 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 function. A poisoning attack directed at a spam filter, for instance, would manipulate the training data to cause misclassification of certain spam mails, e.g., having malicious mails classified as harmless [BNL12;



(a) Hard Margin SVM



(b) Soft Margin SVM

Figure 1.1: Support Vector Machine with Hard and Soft Margin (Taken from [11])

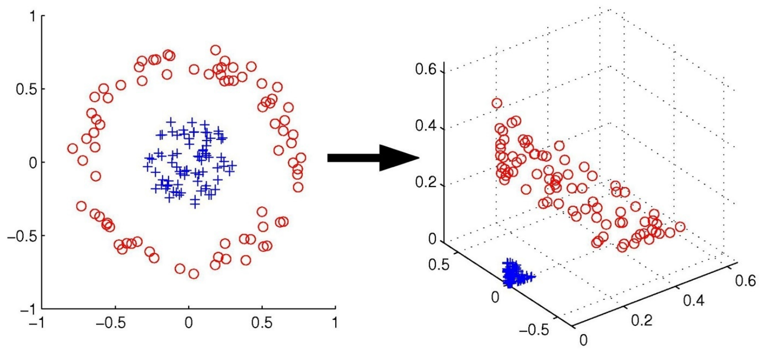


Figure 1.2: Transforming the Input Space Making Data Linearly Separable Using an RBF Kernel (Taken from [Fle09])

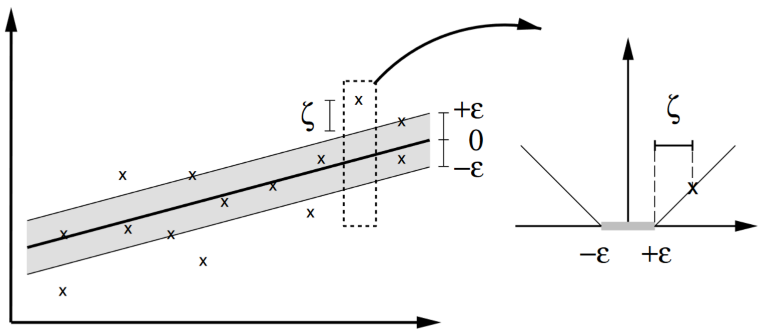


Figure 1.3: A Linear Support Vector Regression Machine with a Soft Margin (Taken from [SS04])

RNH+09]. This attack is performed while training an algorithm. In supervised learning (the subset of machine learning algorithms that uses labeled training data), a causative attack can be achieved by incorrectly labeling training data, e.g., in a scenario, when the training algorithm gets its data from an untrusted source. In reinforcement learning, incorrect rewards and punishments can be used to perform this type of attack. In unsupervised learning, causative attacks can be performed by manipulating the decision, which training data are picked in such a way that the underlying statistical distribution is misrepresented.

1.3.2 Evasion Attacks

Evasion attacks aim at finding input data to a fully trained algorithm that result in an incorrect output. For instance, an evasion attack could be performed on a spam-filter to 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 by the scanner. This type of attack can be improved and facilitated by prior information gathering about the attacked model, i.e., by performing an exploratory attack, specifically an extraction attack, beforehand. Furthermore, causative attacks might be used in an attempt to alter a model in such a manner that evasion attacks become less difficult for the attacker [BCM+13].

1.3.3 Exploratory Attacks

Exploratory attacks, which this thesis further explores, intend to unravel information about the algorithm's configuration, or general inner workings, in order to extract the algorithm itself, to find weaknesses in the algorithm, or to gain knowledge that can be leveraged 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. 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 SVMs [LM05]. 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 this thesis, we explore model extraction attacks on SVMs and SVRs that intend to extract or estimate model parameters.

1.4 White-Box and Black-Box Attacks

We differentiate between white-box and black-box type of attacks on MLaaS providers. In a white-box attack, the attacker knows some details about how the MLaaS provider has implemented a model. Specifically, she 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 theses 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 Section 4.5, 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.

1.5 Our Contributions

We introduce new equation-solving attacks on linear and quadratic SVRs and propose re-training approaches for other kernels, such as RBF kernel SVRs. We implement our attacks to examine their accuracy and their behavior, such as speed and generated queries, and compare them to existing extraction attacks on SVM. Furthermore, we study the feasibility of such attacks in realistic scenarios and show that Wi-Fi localization schemes using SVR (e.g., [ZCZL16]) are indeed vulnerable to our extractions, even when they are protected using SMPC protocols. In a simulated MLaaS environment using a RBF kernel SVR, our extraction attack was able to extract a near-perfect model within 7 minutes at the cost of $0.09. In the end of the thesis, we suggest some possible countermeasures for future research.

1.6 Outline

In Chapter 2, we give an overview over the concepts behind SVMs and SVRs. In Chapter 3, we explain previous work and show current attacks on SVMs in detail. in Chapter 4, we introduce newly devised model extraction algorithms for SVRs. In Chapter 5, we describe our implementations of an attack simulation framework for the MLaaS scenario. In Chapter 6, we test and evaluate attacks on SVMs and our attacks on SVRs using different datasets. In Chapter 7, we give our conclusion on the feasibility of our attacks and propose a few possible countermeasures that can be the topic of further research.

2 Preliminaries

In this chapter we introduce basic concepts and notation used throughout this thesis.

2.1 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.

2.2 Support Vector Machines

Support Vector Machines (SVM) [CV95] 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

$$\{(\mathbf{x}_i, y_i) | i = 1, \dots, m; y_i \in \{0, 1\}\} \tag{2.1}$$

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.
Mathematically, an SVM is defined by a function

$$f(\mathbf{x}) = sign(\langle \mathbf{w}, \mathbf{x} \rangle + b), \tag{2.2}$$

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 $\mathbf{w} \cdot \mathbf{x} + b > 1$ 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 **w** and b for a given problem, an optimization problem has to be solved:

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||_2^2 \text{ subject to } y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \text{ for } 1 \leq i \leq m, \tag{2.3}$$

where $||\mathbf{w}||_2^2$ denotes the quadratic Euclidean norm of **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

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{m}\xi_i \text{ subject to } y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b) \geq 1 - \xi_i \text{ for } 1 \leq i \leq m. \tag{2.4}$$

The computation of **w** and b is done by transforming the optimization problem into a Lagrangian dual problem. With

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i, \tag{2.5}$$

the Lagrangian dual problem is defined as:

$$\text{Maximize for } \alpha\text{:} \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{2.6}$$

subject to

$$0 \leq \alpha_i \leq C \text{ and } \sum_{i=1}^{m}\alpha_i y_i = 0. \tag{2.7}$$

This dual problem can be solved efficiently using modern quadratic programming techniques or linear approximations [SV99; WZ05].

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 $\phi(\mathbf{x})$ is used:

$$\phi : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}, \mathbf{x} \mapsto \phi(\mathbf{x}). \tag{2.8}$$

Such a function however can be hard to compute, especially if the projected dimension is large, or even infinite, as it is the case for the popular 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

$$f(\mathbf{x}) = sign(\langle \mathbf{w}, \phi(\mathbf{x}) \rangle + b) = sign\left(\sum_{i=1}^{m}\alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right). \tag{2.9}$$

Table 2.1 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 [PG07], or bankruptcy prediction [ML05], polynomial kernels have seen use in document classification [MY01]. 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 [CMRS04].

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	$\tanh(\gamma \mathbf{x}^t \cdot \mathbf{x}' + r)$

Table 2.1: Popular Kernels

2.3 Support Vector Regression Machines

Support Vector Regression Machines (SVRs) [DBK+97] 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 a maximum error ϵ for each y_i and is as flat as possible [SS04]. Similarly to SVMs, a linear SVR is defined by a function

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b, \tag{2.10}$$

where $\mathbf{w}$ and b are calculated by solving an optimization problem:

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||_2^2 \tag{2.11}$$

$$\text{subject to } \begin{cases} y_i - \langle \mathbf{w}, \mathbf{x_i} \rangle - b & \leq \epsilon \\ \langle \mathbf{w}, \mathbf{x_i} \rangle + b + y_i & \leq \epsilon \end{cases}. \tag{2.12}$$

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 positive and negative error respectively. Also, the parameter C to weight the errors in trade-off to the flatness of $\mathbf{w}$ is introduced. Our optimization problem now is:

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^*) \tag{2.13}$$

$$\text{subject to } \begin{cases} y_i - \langle \mathbf{w}, \mathbf{x_i} \rangle - b & \leq \epsilon + \xi_i \\ \langle \mathbf{w}, \mathbf{x_i} \rangle + b + y_i & \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* & \geq 0 \end{cases}. \tag{2.14}$$

Translating the optimization problem into its dual form, we end up with:

$$\text{Maximize} \begin{cases} -\frac{1}{2}\sum_{i,j=1}^{m}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle \mathbf{x_i}, \mathbf{x_j}\rangle \\ -\epsilon \sum_{i=1}^{m}(\alpha_i + \alpha_i^*) + \sum_{i=i}^{m} y_i(\alpha_i - \alpha_i^*) \end{cases} \tag{2.15}$$

$$\text{subject to } \sum_{i=i}^{m}(\alpha_i - \alpha_i^*) = 0 \text{ and } \alpha_i, \alpha_i^* \in [0, C], \tag{2.16}$$

where **w** can now be written as follows:

$$\mathbf{w} = \sum_{i=1}^{m}(\alpha_i - \alpha_i^*)\mathbf{x_i}, \tag{2.17}$$

and, therefore, the regression function is defined by

$$f(\mathbf{x}) = \sum_{i=1}^{m}(\alpha_i - \alpha_i^*)\langle \mathbf{x_i}, \mathbf{x}\rangle + b. \tag{2.18}$$

Just like for SVMs, we can use the kernel trick to perform regression on non-linear functions, by substituting the dot-product with a kernel of choice:

$$f(\mathbf{x}) = \sum_{i=1}^{m}(\alpha_i - \alpha_i^*)K(\mathbf{x_i}, \mathbf{x}) + b. \tag{2.19}$$

The kernels that can be used for SVRs are the same as for SVMs (see Table 2.1).

2.4 Secure Multi-Party Computation

Secure Multi-Party Computation (SMPC) is a field of study, which aims at jointly computing a public function g by distrustful parties on their private inputs and revealing nothing but the result of the computation. This is achieved by using cryptographic tools like homomorphic encryption (e.g., the Paillier cryptosystem [Pai99]), Yao's Garbled Circuits [Yao86], secret sharing (e.g., "How to Play any Mental Game" [GMW87]), and other protocols. SMPC is used to protect the ideal functionality of the algorithm guarantee privacy of the input data of the protocol participants. However, if the idea functionality leaks information about the private data, the use of SMPC does not yield benefits.

3 Related Work

In this section, we introduce related work. Lowd and Meek [LM05] worked on evasion attacks specifically targeting spam detection in emails and came up with an exploratory attack, an algorithm to extract parameters from linear classifiers, such as linear SVMs. Using their algorithm, which is described in Section 3.1, they devised a way of creating optimal spam emails, that will not get flagged by the spam filter under attack.

Tramèr et al. [TZJ+16] explored the topic of model extraction attacks further. With the target of MLaaS in mind, they proposed attacks on Logistic Regression [HLS13], Multilayer Perceptrons [RRK+90], Decision Trees [Qui86], SVMs [CV95], Multiclass Logistic Regression [FHT+00] and Neural Networks [Sch15] (for further reading on these ML algorithms, we also suggest the books "Deep Learning" by Goodfellow et al. [GBCB16], "Pattern Recognition and Machine Learning" by Bishop [Bis06] and "Artificial Intelligence: A Modern Approach" by Norvig and Russel [RN10]). 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 [LM05], enabling an attacker to extract the model of an SVM using polynomial kernels in addition to linear kernels. For the extraction of other kernels, they propose different retraining schemes (see Section 3.3 for details). Papernot et al. [PMG+17] explore black-box extraction and evasion attacks on MLaaS using Deep Neural Networks (DNNs) [GBCB16]. 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 [Dmi+18], Dmitrenko further explores the model extraction attacks on DNN introduced by Papernot et al.

Another approach to extract models was proposed by Shi et al. [SSG17], who suggested training a deep learning algorithm using the MLaaS provider as an oracle. Kesarwani et al. [KMAM17] 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 [TZJ+16]. Wang and Gong [WG18] proposed techniques to extract hyperparameters from MLaaS providers. Hyperparameters are optimization parameters for ML algorithms, such as the parameter C in Equation (2.4) in Section 2.2. Attacks on SVRs in particular have not been researched yet. In this work, we fill this gap by transferring known attacks on SVMs to SVRs, finding new equation-solving attacks on SVR, and putting our attacks into the context of SMPC.

The known model extraction attacks on SVMs described previously are explained in detail below, as extraction attacks on SVMs and transferring them to SVRs are the main focus of this 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.

3.1 The Lowd-Meek Attack

The first extraction attack on linear classifiers in general and thus also on SVMs was proposed by Lowd and Meek [LM05]. Their white-box attack assumes a third-party oracle, for instance, an MLaaS Application Programming Interface (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.

To initiate the Lowd-Meek attack, a positive sample $\mathbf{x}^+$ and a negative sample $\mathbf{x}^-$ 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 this 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 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 $\epsilon/4$ distance to the hyperplane, $O(\log(1/\epsilon)+\mathit{size}(\mathbf{s}^+,\mathbf{s}^-))$ queries are needed. To find the relative weight of each other feature, another $O(\log(1/\epsilon)+\mathit{size}(c))$ queries are needed, where $\mathit{size}(c)$ describes the encoding length of the computational environment. In total, the algorithm uses a polynomial number of queries. This attack however is limited in 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 she needs for this attack.

3.2 The Lowd-Meek Attack for Nonlinear Kernels

Tramèr et al. [TZJ+16] 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

$$\langle \mathbf{w}^F, \phi(\mathbf{x}) \rangle + b = 0, \tag{3.1}$$

where

$$\mathbf{w}^F = \sum_{i=1}^{m} \alpha_i \phi(\mathbf{x}_i). \tag{3.2}$$

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 [MRW+99] 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, increasing processing costs in this attack method for each query.

3.3 Retraining Attack

Any SVM can be extracted using a white-box retraining approach, proposed by Tramèr et al. [TZJ+16]. 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. 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.

Tramèr et al. introduce three possible approaches to the retraining attack:

1. **Retraining with uniform queries.** The first and simplest approach works as follows: take a set of m uniformly random samples, have them classified by a prediction API and use the result for retraining.
2. **Line-search retraining.** This approach uses line search techniques as they are used in the Lowd-Meek attack (see Section 3.1), to have samples classified that are close to the decision boundary, yielding a more accurate result.
3. **Adaptive retraining.** The last approach they propose is denoted adaptive retraining, which utilizes active learning techniques [CAL94] 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, which is the hyperplane in the case of SVMs, of the current extracted model.

3.4 Neural Network Retraining Attacks

Shi et al. [SSG17] has shown, that it is possible to perform black-box retraining attacks on Naive Bayes classifiers, Artificial Neural Networks, and SVMs using Deep Learning. The authors 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. [PMG+17] have also introduced a retraining attack using (Deep) Neural Networks, attacking other black-box DNNs. Their attacks were further researched by Dmitrenko [Dmi+18]. As a black-box DNN can be substituted by any other classifier with the same function, their attacks should also work on different classifiers.

4 Our New Algorithms for Model Extraction of SVRs

We propose the following new algorithms for model-extraction attacks on SVR. To the best of our knowledge, no model extraction attacks for SVR have been proposed before. The white-box algorithms described in Section 4.1 and Section 4.2 are new equation-solving attacks, that can extract exact models for the linear and quadratic kernels respectively. The algorithms described in Section 4.3 and Section 4.4 transfer the retraining approach proposed by Tramèr et al. [TZJ+16] to the extraction of SVR models. The last algorithm, described in Section 4.5, is an approach to black-box model extraction.

4.1 Exact Extraction of SVR Models using Linear Kernels

For Support Vector Regression Machine using a linear kernel the regression function can be described as

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b \quad \text{with} \quad \mathbf{w}, \mathbf{x} \in \mathcal{X}, b \in \mathbb{R}. \tag{4.1}$$

Since we can query an oracle for values of 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.

$$w_i = f(\mathbf{x}) - b \quad \text{with} \quad x_i = 1, x_j = 0 \quad \forall j \neq i. \tag{4.2}$$

In total this algorithm needs $n + 1$ queries to extract the exact model parameters, where n is the dimension of the feature space.

4.2 Exact Extraction of SVR Models using Quadratic Kernels

SVRs using quadratic kernels can be described as

$$f(\mathbf{x}) = \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) K_{quadratic}(\mathbf{x}_i, \mathbf{x}) + b, \tag{4.3}$$

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 the bias, with

$$K_{quadratic}(\mathbf{x}', \mathbf{x}) = (\langle \mathbf{x}', \mathbf{x} \rangle + c)^2 = \langle \phi_{quadratic}(\mathbf{x}'), \phi_{quadratic}(\mathbf{x}) \rangle, \tag{4.4}$$

where $K_{quadratic}$ is the quadratic kernel, c a kernel parameter, and $\phi_{quadratic}$ the feature transformation function for the quadratic kernel. Therefore, we can summarize:

$$\mathbf{w} = \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) \phi_{quadratic}(\mathbf{x}_i), \tag{4.5}$$

which is a weight vector in the transformed feature space and we end up with a linear regression function in the transformed feature space:

$$f(\mathbf{x}) = \langle \mathbf{w}, \phi_{quadratic}(\mathbf{x}) \rangle + b. \tag{4.6}$$

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 to happen [CHC+10]. The transformation function of the quadratic kernel is defined as follows:

$$\phi_{quadratic}(\mathbf{x}) = \begin{cases} x_n^2, \dots, x_1^2, \\ \sqrt{2}x_n x_{n-1}, \dots, \sqrt{2}x_n x_1, \sqrt{2}x_{n-1}x_{n-2}, \dots, \sqrt{2}x_{n-1}x_1, \dots, \sqrt{2}x_2 x_1, \\ \sqrt{2c}x_n, \dots, \sqrt{2c}x_1, \\ c \end{cases} . \tag{4.7}$$

$\phi_{quadratic}(\mathbf{x})$ 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 an oracle (the MLaaS provider) with arbitrary $\mathbf{x}$ and get the corresponding $f(\mathbf{x})$, we can reconstruct a function $f'(\mathbf{x})$ equivalent to Equation (4.6). If we look closely at Equation (4.6) 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 $\mathbf{v}_0 = (0, \dots, 0)^T$ to get:

$$f(\mathbf{v}_0) = \langle \mathbf{w}, \phi_{quadratic}(\mathbf{v}_0) \rangle + b = w_d c + b = b'. \tag{4.8}$$

Looking back at Equation (4.6), we can see that the first n weights $w_1, \dots, w_n$, which are the ones that get multiplied with $x_n^2, \dots, x_1^2$, 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

$$\mathbf{v}_i^+ := (v_1, v_2, \dots, v_i, \dots, v_n) = (0, 0, \dots, 1, \dots, 0, 0) \quad \forall i \in 1, \dots, n \tag{4.9}$$

and

$$\mathbf{v}_i^- := (v_1, v_2, \dots, v_i, \dots, v_n) = (0, 0, \dots, -1, \dots, 0, 0) \quad \forall i \in 1, \dots, n. \tag{4.10}$$

Putting those values into Equation (4.6), we get:

$$f(\mathbf{v}_i^+) = w_{n-i+1} + w_{d-i}\sqrt{2c} + b' \tag{4.11}$$

and

$$f(\mathbf{v}_i^-) = w_{n-i+1} - w_{d-i}\sqrt{2c} + b'. \tag{4.12}$$

If we now add $f(\mathbf{v}_i^+)$ and $f(\mathbf{v}_i^-)$, subtract $2b'$, and divide by 2, we end up with:

$$w_{n-i+1} = ((w_{n-i+1} + w_{d-i}\sqrt{2c} + b') + (w_{n-i+1} - w_{d-i}\sqrt{2c} + b') - 2b')/2. \tag{4.13}$$

As we can do this for all values $i \in 1, \dots, n$, 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:

$$\sqrt{2c}w_{d-i} = ((w_{n-i+1} + w_{d-i}\sqrt{2c} + b') - (w_{n-i+1} - w_{d-i}\sqrt{2c} + b'))/2. \tag{4.14}$$

Looking back at Equation (4.6), we know that the weights $\forall i \in 1, \dots, n : w_{d-i}$ always will be multiplied by $\sqrt{2c}$. Therefore, for our reconstruction $f'(\mathbf{x})$ we can set:

$$\forall i \in 1, \dots, n : w'_{d-i} = \sqrt{2c}w_{d-i}, \tag{4.15}$$

so that we do not have to find the value of c.

Now, all we need to reconstruct Equation (4.6) are w_n to w_{d-n-1}, or them multiplied by $\sqrt{2}$. To calculate them, we have to go through each combination of two ones in $\mathbf{v}$ and all other values at zero:

$$\forall i, j \in n, i \neq j : \mathbf{v} = (v_1, v_2, \dots, v_i, \dots, v_j, \dots, v_n) = (0, 0, \dots, 1, \dots, 1, \dots, 0). \tag{4.16}$$

Querying for those $\mathbf{v}$ we get:

$$f(\mathbf{v}) = w_{n-i+1} + w_{n-j+1} + \sqrt{2}w_r + w_{d-i}\sqrt{2c} + w_{d-j}\sqrt{2c} + b', \tag{4.17}$$

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 largely less efficient then using the kernel trick, because instead of the more efficient kernel function $(\mathbf{x} \cdot \mathbf{x}' + 1)^2$, 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 $\forall i \in 1, \dots, n : w'_{d-i}$ and in b', we have to adjust the transformation function that we are using accordingly:

$$\phi'_{quadratic}(\mathbf{x}) = \begin{cases} x_n^2, \dots, x_1^2, \\ \sqrt{2}x_n x_{n-1}, \dots, \sqrt{2}x_n x_1, \sqrt{2}x_{n-1}x_{n-2}, \dots, \sqrt{2}x_{n-1}x_1, \dots, \sqrt{2}x_2 x_1, \\ x_n, \dots, x_1, \\ 0 \end{cases} \tag{4.18}$$

and our extracted weights are

$$\mathbf{w}' = \begin{cases} w_n, \dots, w_1, \\ w_{n+1}, \dots, w_{d-n} \\ \sqrt{2c}w_{d-n+1}, \dots, \sqrt{2c}w_{d-1}, \\ 0 \end{cases} . \tag{4.19}$$

The resulting extracted equation looks then as follows:

$$f'(\mathbf{x}) = \langle \mathbf{w}', \phi'_{quadratic}(\mathbf{x}) \rangle + b'. \tag{4.20}$$

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 $O(n^2)$.

4.3 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 attackers model. The most simple approach is to have the model provider label a set of uniformly random samples. This random approaches accuracy 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.

4.4 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 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 server. An initial model is being trained on the results obtained from the server. We know that for SVR only support vectors influence the model. We use the approach by Douak et al. [DMPB12] of picking samples by their distance from the support vectors. That means, 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 every round the SVR has to be recalculated. The algorithm's speed also depends on the variable n, the number of samples 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 algorithms' speed can be increased using online learning techniques for SVR [LZ16]. Online learning techniques enables 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 speed 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 reduces significantly.

4.5 Kernel Agnostic Extraction Technique

Even though the classic and adaptive retraining approaches can be applied to any kernel, the attacker has to know which kernel she is attacking, assuming the white-box attack scenario (see Section 1.4). 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. The computation time however increases, because multiple models have to be calculated instead of just one. There are only a handful of classic kernels that see widespread use, listed in Table 2.1, 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. Each next round, the samples that each kernel is the least certain about are selected into a subset S. Then, 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 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, it has to be waited until all models are done training.

5 Our Software Framework for Model Extraction Attacks on MLaaS

To study different attacks and find working attack strategies, we implement a simulator for the MLaaS paradigm in which we have the possibility 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. Our implementation is open-source and can be viewed on Github[1].

5.1 Architecture

Below, we explain how the implementation is designed in detail, and show how it can be used to simulate an attack scenario.

The implementation consists of four classes: Server, Client, Adversary, and Supervisor (see Figure 5.1). The Server class represents an MLaaS provider and stores different trained ML models, providing answers to classification and 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 and is the point of initialization and interaction. The user only interacts with the Supervisor class. The classes detailed functionalities are described below.

5.1.1 The Server Class

The Server class' purpose is to represent an MLaaS provider or a similar service that offers ML classification and regression via an API. Its basic functionality is therefore 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.

[1]`https://github.com/robre/attacking-mlaas`

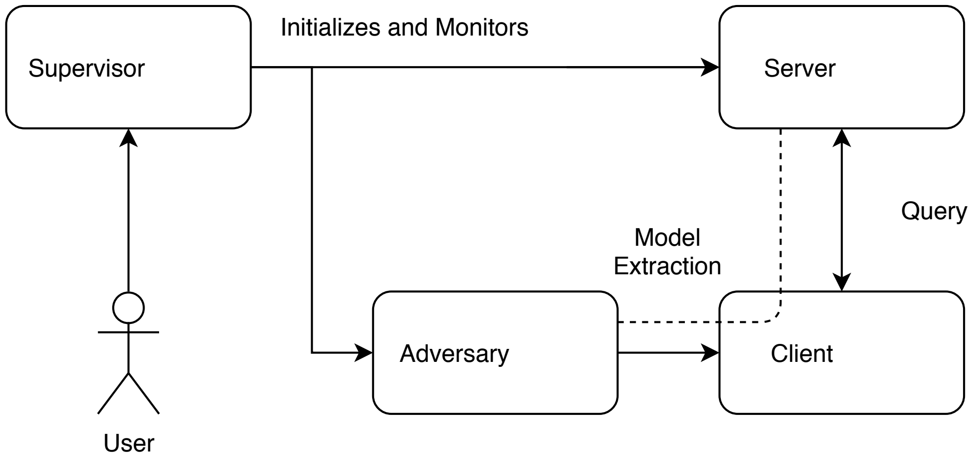


Figure 5.1: Program Structure Diagram

5.1.2 The Client Class

The Client class is a rather simple implementation that 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, so that the total number of queries used by a client can be checked. 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.

5.1.3 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 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 SVR. An SVM can be attacked using the Lowd-Meek attack (see Section 3.1), retraining, or adaptive retraining (see Section 3.3). An SVR can be attacked using a set of our newly devised algorithms (see 4). These new algorithms include an equation-solving attack which extracts the exact parameters of a linear SVR (see Section 4.1), and for quadratic kernels (see Section 4.2), and furthermore, SVR can be attacked using retraining and adaptive retraining, as described in Section 4.3 and Section 4.4, respectively.

5.1.4 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.

5.2 Implementation Formalities

We implement our system in Python 3.6. Our implementation makes use of the following Python libraries: *sklearn* for the machine learning tasks, *numpy* for handling arrays and *matplotlib* for plotting. It also uses the Python standard libraries *random*, *math*, *statistics*, *time* and *csv*. Note that the implementation works entirely locally, whereas a real world attack on MLaaS would be performed over a network.

5.2.1 Sample Usage

```python
s = Supervisor()
X = []
y = []
m = 3
b = 12
for i in range(0, 1000):
    n = random.randrange(-50, 50, 1) / 100
    y.append( m * i + b + n )
    X.append([i])
training_set_X = X[0:500]
training_set_y = y[0:500]
attacker_training = X[500:700]
test_set_X = X[700:1000]
test_set_y = y[700:1000]
name = "test-rbf-svr"
prediction_type = "SVR"
kernel_type = "rbf"
query_budget = 100
round_size = 8
attack_type = "adaptive retraining"
s.add_model(name, training_data_X, training_data_y,
    prediction_type, kernel_type)
run_time, queries, model = s.attack_with_metrics(name, kernel_type
    , attack_type, dimension, query_budget, attack_training_set,
    round_size)
s.compare_predictions(name, test_set, correct_results=test_set_y,
    verbose=True)
```

Listing 5.1: Using the implementation to simulate a model extraction attack.

The implementation is used by initializing an instance of the Supervisor class, generating the training set for the initial model, a set of unlabeled data for the attacker, and a test set with correctly labeled data. Then, a model is created and added to the server using the *Supervisor.add_model()* method. The model is attacked using the *Supervisor.attack_with_metrics()* method, which returns the calculation time, queries used, and extracted model. To evaluate the extraction, the *Supervisor.compare_predictions()* method is used, which compares the classification or regression on the test set of the original model to the extracted model.

In Listing 5.1, we show an example of a simulated model extraction attack: first, a Supervisor is created. Next, a total of 1 000 samples are generated, of which the first 500 are used to train the original model, the next 200 are samples the attacker can use, and the last 300 are used to compare the original models' predictions to the extracted ones. After creating the data, some variables, such as the kernel, query budget, prediction type and attack type are set. With the given parameters and data, a model is created and added to the server and subsequently attacked. In the last step, the extracted model is evaluated by comparing its predictions to the original models predictions.

5.2.2 Data Generation

In case no suitable training data are at hand, they can be generated algorithmically. To generate training data for SVM, the function *sklearn.datasets.makeblob()* can be used as shown in Listing 5.2, which creates clusters of points with corresponding labels.

```
X, y = sklearn.datasets.make_blobs(n_samples=60, centers=2,
    random_state=7)
```

Listing 5.2: Training Data Generation for SVM with 60 Samples

To generate training data for SVR, the mathematical function which should be regressed can be sampled on m points with addition of random noise. In Listing 5.3, the generation of training data for a linear SVR is shown. Alternatively, the function *sklearn.datasets.make_regression()* can be used.

```
w = 3
b = 12
X = []
y = []

for i in range(0, 60):
    noise = random.randrange(-50, 50, 1) / 100
    X.append([i])
    y.append(m * i + b + n)
```

Listing 5.3: Training Data Generation for SVR with 60 Samples.

Furthermore, we implement two functions *create_similar_data()* and *generate_positive_negative()* that can take a known set of data and create an arbitrary number of similar data. The first function, *create_similar_data()*, takes a set of data, calculates the mean value and standard derivation for each feature, and then creates new data with similar attributes. The second function, *generate_positive_negative()*, takes a set of classification training data and their targets, separates them by classification, and creates two sets of equal length of data using *create_similar_data()*, each derived from the positive and negative separately, and merges the two sets into one set of new data.

5.2.3 Loading Data from Files

Training data from .csv files can be used by reading them with the *load_training_data_from_csv()* function. The function can be parametrized to read training data with any number of features.

6 Evaluation

6.1 Evaluation Tasks

To assess the feasibility of model extraction attacks, it needs to be determined what makes such 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.

6.2 Approximation Quality

A model extraction attack in general produces an approximated version of a model. The approximation accuracy can be assessed by comparing the prediction results of the extracted model to the predictions of the original model.

6.2.1 Approximation Quality for SVM Extractions

SVMs produce labels of 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:

$$P_{wrong} = \frac{wrong\ predictions}{total\ predictions}. \tag{6.1}$$

We generally try to achieve a minimal percentage of wrong classifications. Extractions with $P_{wrong} \leq 0.01$ are considered sufficient and $P_{wrong} \leq 0.001$ very good as described by Tramèr et al. [TZJ+16].

6.2.2 Approximation Quality for SVR Extractions

For SVRs, the predictions are continuous values. Therefore, the Absolute Mean Approximation Error (AMAE) can be calculated as:

$$AMAE = \frac{1}{n}\sum_{i=1}^{n} |o_i - p_i|, \tag{6.2}$$

with the original prediction o and the extracted prediction p. The Mean Squared Error (MSE) can be calculated as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n} (o_i - p_i)^2. \tag{6.3}$$

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

$$RMSE = \frac{\sum_{i=1}^{n} (o_i - p_i)^2}{\sum_{i=1}^{n} (\bar{o} - o_i)^2}, \tag{6.4}$$

where $\bar{o}$ denotes the mean of o. We consider extracted models, where $RMSE \leq 0.01$ as very accurate.

6.3 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.

6.3.1 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 extracting 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 for extracting the model. Hence, low query costs increase the attack feasibility.

6.3.2 Time Costs

Another cost factor to be considered is runtime. The extraction can be considered infeasible if the runtime associated with it is unproportionally high. The extraction runtime has two main factors: local calculation time and query time. 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.

6.4 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 datasets we used, that were not generated by us, are listed in Table 6.1.

id	Dataset Name	Type	Origin	Features	Size
1	California Housing	Regression	Natural	8	20 640
2	Boston House-prices	Regression	Natural	13	506
3	UJIIndoorLoc	Regression	Natural	520	19 937
4	IPIN 2016 Tutorial	Regression	Natural	168	927

Table 6.1: Datasets Used in this Work

6.5 Attacker Model

For our attacker model, we assume a financially motived attacker, such as a competitor, who has the intention to ultimately be able to predict data herself 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 [LM05], which is crucial to their extraction attack.

It is to be noted 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-dimension 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, 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 she wants to have predicted is crucial.

6.6 Results

For each setting, we first perform an attack using the best, i.e., most accurate technique that we know of. Then, we perform retraining attacks with similar and less numbers of queries then 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 unrealistic latency of near zero. To arrive at more realistic estimates for the runtimes, the queries used in each attack have to be multiplied with the sum of the server-side calculation time and the network latencies. We multiply the query count by 0.3 s, that is 0.1 s latency times two (for request and answer), and another 0.1 s for the server-side calculation time.

6.6.1 Extraction of Linear SVMs

To the best of our knowledge, the best known attack on linear SVMs is the Lowd-Meek attack [LM05], which can extract a model with an adjustable accuracy of epsilon. We create 100 random classification problems as shown in Listing 6.1, and attack each of them using the Lowd-Meek attack with epsilon and delta both set to 0.01.

We find, that the extraction takes an average amount $17 \cdot n$ queries, with number of features n. 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 0.003 s 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 $17\,000 \cdot 0.3\,\text{s} = 5\,100\,\text{s}$ (85 minutes) in a real network. 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 an α of 1. For adaptive retraining, we get the same results. This can be explained because linear SVMs are Probably Approximately Correct (PAC) learnable [Val84] as their Vapnik-Chervonenkis dimension (VC dimension) is

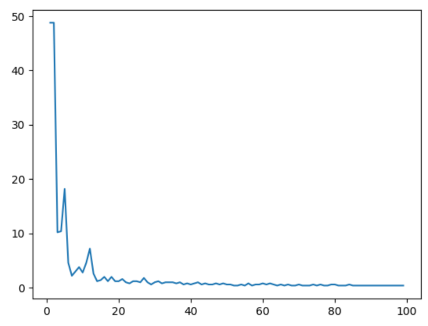
$n+1$ [VLC94]. RBF kernels however have an infinite VC dimension, and are therefore not PAC learnable.

```python
for i in range(1, 100):
    X, y = sklearn.datasets.make_blobs(n_samples=500, centers=2,
        random_state=i, n_features=2)
```

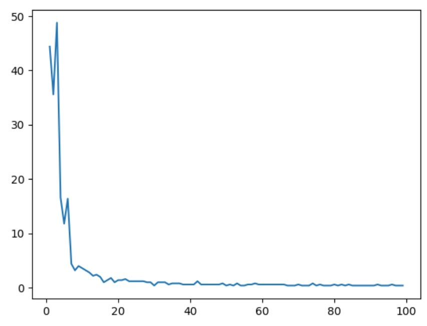
Listing 6.1: Creating 100 Random Binary Linear Classification Problems.

6.6.2 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 Figure 6.1, 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 0.3 s network time per query, such an attack would take about $70\,\text{s} + 420 \cdot 0.3\,\text{s} = 196\,\text{s}$ (3 minutes and 16 seconds).



(a) Error percentage in dependence of α for randomized retraining



(b) Error percentage in dependence of α for adaptive retraining

Figure 6.1: Retraining Strategies for RBF SVMs

6.6.3 Extraction of Linear SVRs

We use three different datasets to test the extraction of linear SVRs: the California housing dataset, the Boston house-prices dataset and a generated dataset with 100 features. Datasets for regression problems can be generated using *sklearn.datasets.make_regression()*.

First, we employ the equation solving attack described in Section 4.1. 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 0.001 s (2.7 s with Internet latency). Attacking the Boston House-prices dataset, we got an exact extraction with 14 queries in 0.001 s (4.2 s with Internet latency). Lastly, extracting the generated dataset took 101 queries and 0.059 s (30.3 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.

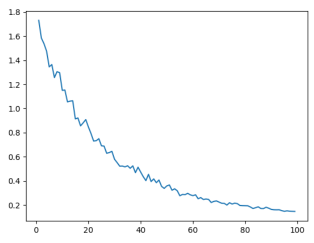
6.6.4 Extraction of Quadratic SVRs

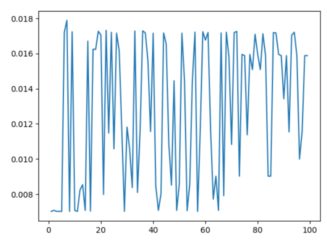
For the extraction of quadratic SVRs, we use the same three datasets as in Section 6.6.3, but trained with a quadratic kernel. We use the equation solving attack described in Section 4.2 first. Using this attack, the extraction of the model trained on the California housing dataset took a total of 45 queries and 0.007 s (13.5 s with Internet latency). Extracting the model trained on the Boston house-prices dataset took 105 queries and 0.006 s (31.5 s with Internet latency). Extracting the model trained on the generated dataset with 100 features took 5 151 queries and 0.589 s (1 545.8 s, or about 26 minutes with Internet latency).

The optimal attacks query counts are comparable to setting α between 9 and 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 an α of just 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 incredibly slow.

6.6.5 Extraction of RBF SVRs

For the extraction of RBF SVRs, we again use the same three datasets as in Section 6.6.4 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 Figure 6.2, the MSE is shown in dependence of the used α for the randomized retraining approaches. We can see that for the generated dataset, the MSE gets proportionally lower as we increase α. For the natural datasets however, the MSE fluctuates within a specific area relatively unaffected by increasing α. In Figure 6.3, the MSE is shown in dependence of the used α for the adaptive retraining approaches. In comparison to the randomized retraining, we can see that now there is a trend for all datasets, natural or synthetic, to have a lower MSE with the increase of α. Note that we conducted the experiment for α in steps of 5 for all datasets, and capping at $\alpha = 45$ for the generated dataset, because the extraction took up to 20 978 s (5 hours and 50 minutes). For the natural datasets, with lower dimensions, the adaptive retraining took up to 800 s (14 minutes, or 21 minutes with Internet latency). The randomized retraining was significantly faster at a maximum of 0.6 s, or 7 minutes with Internet latency.


(a) MSE in dependence of α for the generated dataset


(b) MSE in dependence of α for California housing

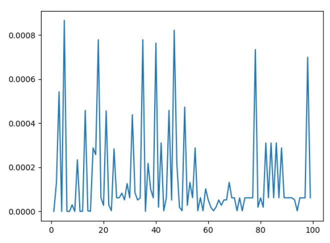
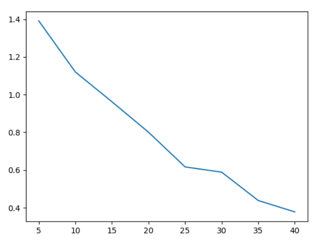
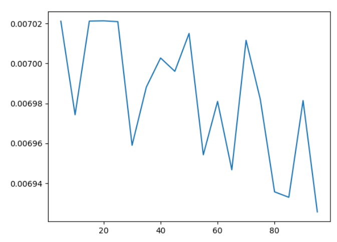

(c) MSE in dependence of α for Boston housing

Figure 6.2: MSE for Randomized Retraining Strategies for RBF SVRs


(a) MSE in dependence of α for the generated dataset


(b) MSE in dependence of α for California housing

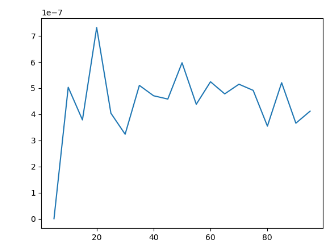

(c) MSE in dependence of α for Boston housing

Figure 6.3: MSE for Adaptive Retraining Strategies for RBF SVRs

6.6.6 Extraction of Wi-Fi Localization Models

In the following, we show a practical attack on an ideal implementation of an SMPC protected MLaaS providing indoor localization from Wi-Fi fingerprints using SVRs.

Zhang et al. [ZCZL16] described an SMPC protocol to protect user and model provider privacy 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 [TZJ+16]). Our extraction efforts are therefore 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 as few access points was unattainable. We use the "UJIIndoorLoc" dataset with 520 access points, and the "IPIN 2016 Tutorial" dataset with 168 access points instead. We train our SVRs using the RBF kernel and a set of 100 training data 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 α of just 1 and an error of $6.36 \cdot 10^{-7}$ with an α of 5. The attacks took up to 1 368 s for computation and 2 605 queries. Considering 0.3 s of network time, this attack would take about translating to a total sum of 2 150 s (36 minutes) of runtime. 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 419 s, or 7 minutes of runtime considering 0.3 s of network time per query. Consistent among our experiments we saw that a lower amount of features translates to a faster extraction. Considering that this scheme might use much less features than the datasets we conducted our experiments on, we can deduct that this attack may be considerably faster in a realistic scenario.

Note that an algorithm to protect Sigmoid kernel SVRs was proposed as well by Zhang et al. [ZCZL16]. However, we found it difficult to achieve consistent results 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 relatively similar to the vastly more common RBF kernel, which we explored instead.

6.7 Interpretation

Our results (see Table 6.2) show that firstly, the attacks on SVMs by Lowd and Meek, and Tramèr et al. are very effective and allow an attacker to extract accurate models with a relatively low amount of queries. Secondly, our new equation-solving attacks on SVR proved to be not only very accurate, but also up to eighty 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, or experiments show that the best results are achieved 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 from Section 6.6.6. 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 $0.0001 per query, as is the cost at AmazonML, this extraction would have cost a total 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. [ZCZL16] show that their protocol uses $6L + 2nL$ of bandwidth per query, with $L = 2\,048$ bit and n being the amount of features. For the "IPIN 2016 Tutorial" dataset with 168 features this translates to 80 kByte per query. The full extraction would have a bandwidth cost of 71 MByte.

At these cost factors, we can safely assume that this extraction would be very feasible to a financially motivated attacker.

Table 6.2: Extraction Results

Type	Kernel	Method	Dataset	Features	Accuracy	Queries	Runtime
SVM	Linear	Lowd-Meek	Generated	100	100%	1 700	510 s
SVM	Linear	Retraining	Generated	100	100%	1 800	540 s
SVM	Linear	Retraining	Generated	100	100%	101	30 s
SVM	RBF	Retraining	Generated	20	99%	420	129 s
SVM	RBF	Adap. Retraining	Generated	20	99%	420	196 s
SVR	Linear	Eq. Solving	Generated	100	100%	101	30.3 s
SVR	Linear	Eq. Solving	Boston Housing	13	100%	14	4.2 s
SVR	Linear	Eq. Solving	California Housing	8	100%	9	2.7 s
SVR	Quadratic	Eq. Solving	Generated	100	100%	5 151	1 545.8 s
SVR	Quadratic	Eq. Solving	Boston Housing	13	100%	105	31.5 s
SVR	Quadratic	Eq. Solving	California Housing	8	100%	45	13.5 s
SVR	RBF	Retraining	Generated	100	99%	4 040	1 212.5 s
SVR	RBF	Retraining	Boston Housing	13	99%	14	5 s
SVR	RBF	Retraining	California Housing	8	99%	9	3 s
SVR	RBF	Adap. Retraining	Generated	100	99%	4 040	22 190 s
SVR	RBF	Adap. Retraining	Boston Housing	13	99.9%	14	5.2 s
SVR	RBF	Adap. Retraining	California Housing	8	99.9%	9	3.7 s
SVR	RBF	Adap. Retraining	UJIIndoorLoc	520	99.9%	2 605	2 150 s
SVR	RBF	Adap. Retraining	IPIN 2016 Tutorial	168	99.9%	845	419 s

7 Conclusion

Our results show that model extraction attacks on SVRs and SVMs allow an attacker to extract highly accurate models with a relatively low amount of time and money. Therefore, at the current state of MLaaS there is nothing that would stop financially motivated attackers from extracting such models and using them for their own financial gain. At the currently extremely low cost of queries, the cost of extraction is actually so low that it is not only feasible for a well financed entity but also for individuals.

7.1 Countermeasures

To help protect from the attacks shown in this thesis, we discuss a few possible countermeasures that MLaaS can employ to increase the security of their intellectual property.

7.1.1 Rounding

Rounding as a countermeasure was already introduced by Tramèr et al. [TZJ+16], 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, it might be considered to round the actual prediction 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, making this approach unsuitable in some cases.

7.1.2 Extraction Monitor

Kesarwani et al. [KMAM17] 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 amount of queries needed to reconstruct a model with a definable accuracy. Their algorithm is tailored to extraction attacks on decision trees, but could be expanded to include extraction monitoring of other ML models. 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 higher amounts of queries, an MLaaS provider has an incentive to keep clients with high query counts, as they bring revenue. It can therefore be a tough decision whether to cut off a client with high query counts, or not, because there might be the possibility of a model extraction going on. After all, they might be an honest client using the service extensively.

7.1.3 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, 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 techniques in systems that rely on SMPC would incur a significant computation and communication overhead which goes against the business model of MLaaS of providing inexpensive predictions.

7.1.4 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 therefore needs to hide the clear text values on each feature (e.g., by encrypting them using Paillier encryption), the feature transformation has to be performed client-side.

7.2 Future Work

Testing the proposed countermeasures in a realistic environment would be an interesting topic for further research. Furthermore, improved extraction techniques for RBF-kernels in SVM and SVR might be found. In the context of SMPC, protocols will need to be found, that further protect the privacy of the MLaaS provider while also protecting the user privacy.

List of Figures

List of Tables

List of Abbreviations

ML	Machine Learning
SVM	Support Vector Machine
SVR	Support Vector Regression Machine
SMPC	Secure Multi-Party Computation
MLaaS	Machine-Learning-as-a-Service
API	Application Programming Interface
RBF	Radial Basis Function
AMAE	Absolute Mean Approximation Error
MSE	Mean Squared Error
RMSE	Relative Mean Squared Error
PAC	Probably Approximately Correct

Bibliography

[11] *SVM Max Sep Hyperplane With Margin*. 2011. URL: `https://commons.wikimedia.org/wiki/File:Svm_max_sep_hyperplane_with_margin.png` (cit. on p. 4).

[BCM+13] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, F. Roli. **"Evasion Attacks against Machine Learning at Test Time"**. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2013 (cit. on p. 5).

[Bis06] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006 (cit. on p. 11).

[BNL12] B. Biggio, B. Nelson, P. Laskov. **"Poisoning Attacks against Support Vector Machines"**. In: *29th International Coference on International Conference on Machine Learning*. Omnipress. 2012 (cit. on p. 3).

[BNS+06] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, J. D. Tygar. **"Can Machine Learning be Secure?"** In: *2006 ACM Symposium on Information, Computer and Communications Security*. ACM. 2006 (cit. on p. 3).

[CAL94] D. Cohn, L. Atlas, R. Ladner. **"Improving Generalization with Active Learning"**. In: *Machine Learning* (1994) (cit. on p. 14).

[Çat15] F. Ö. Çatak. **"Secure Multi-Party Computation Based Privacy Preserving Extreme Learning Machine Algorithm over Vertically Distributed Data"**. In: *International Conference on Neural Information Processing*. Springer. 2015 (cit. on p. 1).

[CHC+10] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, C.-J. Lin. **"Training and Testing Low-Degree Polynomial Data Mappings via Linear SVM"**. In: *Journal of Machine Learning Research* (2010) (cit. on p. 16).

[CMRS04] G. Camps-Valls, J. D. Martín-Guerrero, J. L. Rojo-Alvarez, E. Soria-Olivas. **"Fuzzy Sigmoid Kernel for Support Vector Classifiers"**. In: *Neurocomputing* (2004) (cit. on p. 9).

[CV95] C. Cortes, V. Vapnik. **"Support-Vector Networks"**. In: *Machine Learning* (1995) (cit. on pp. 2, 7, 11).

[DBK+97] H. Drucker, C. J. Burges, L. Kaufman, A. J. Smola, V. Vapnik. **"Support Vector Regression Machines"**. In: *Advances in Neural Information Processing Systems*. 1997 (cit. on pp. 2, 9).

[Dmi+18] A. Dmitrenko. **“DNN Model Extraction Attacks Using Prediction Interfaces”**. In: (2018) (cit. on pp. 11, 14).

[DMPB12] F. Douak, F. Melgani, E. Pasolli, N. Benoudjit. **“SVR Active Learning for Product Quality Control”**. In: *11th International Conference on Information Science, Signal Processing and their Applications (ISSPA)*. IEEE. 2012 (cit. on p. 18).

[FHT+00] J. Friedman, T. Hastie, R. Tibshirani. **“Additive Logistic Regression: a Statistical View of Boosting”**. In: *The Annals of Statistics* (2000) (cit. on p. 11).

[Fle09] T. Fletcher. **“Support Vector Machines Explained”**. In: *Tutorial Paper* (2009) (cit. on p. 4).

[GBCB16] I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio. *Deep Learning*. MIT press Cambridge, 2016 (cit. on p. 11).

[GMW87] O. Goldreich, S. Micali, A. Wigderson. **“How to Play any Mental Game”**. In: *The Nineteenth Annual ACM Symposium on Theory of Computing*. ACM. 1987 (cit. on p. 10).

[HLS13] D. W. Hosmer Jr, S. Lemeshow, R. X. Sturdivant. *Applied Logistic Regression*. John Wiley & Sons, 2013 (cit. on p. 11).

[KMAM17] M. Kesarwani, B. Mukhoty, V. Arya, S. Mehta. **“Model Extraction Warning in MLaaS Paradigm”**. In: *Computer Security Applications Conference, 2018* (2017) (cit. on pp. 11, 34).

[LLM06] S. Laur, H. Lipmaa, T. Mielikäinen. **“Cryptographically Private Support Vector Machines”**. In: *12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2006 (cit. on p. 1).

[LM05] D. Lowd, C. Meek. **“Adversarial Learning”**. In: *11. ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. ACM. 2005 (cit. on pp. 5, 11 sq., 27 sq.).

[LZ16] J. Liu, E. Zio. **“An Adaptive Online Learning Approach for Support Vector Regression: Online-SVR-FID”**. In: *Mechanical Systems and Signal Processing* (2016) (cit. on p. 19).

[ML05] J. H. Min, Y.-C. Lee. **“Bankruptcy Prediction Using Support Vector Machine with Optimal Choice of Kernel Function Parameters”**. In: *Expert Systems with Applications* (2005) (cit. on p. 9).

[MRW+99] S. Mika, G. Ratsch, J. Weston, B. Scholkopf, K.-R. Mullers. **“Fisher Discriminant Analysis with Kernels”**. In: *Neural networks for Signal Processing IX, 1999*. Ieee. 1999 (cit. on p. 13).

[MY01] L. M. Manevitz, M. Yousef. **“One-Class SVMs for Document Classification”**. In: *Journal of Machine Learning Research* (2001) (cit. on p. 9).

[Pai99] P. PAILLIER. **“Public-Key Cryptosystems Based on Composite Degree Residuosity Classes”**. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1999 (cit. on p. 10).

[PG07] K. POLAT, S. GÜNEŞ. **“Breast Cancer Diagnosis Using Least Square Support Vector Machine”**. In: *Digital Signal Processing* (2007) (cit. on p. 9).

[PMG+17] N. PAPERNOT, P. MCDANIEL, I. GOODFELLOW, S. JHA, Z. B. CELIK, A. SWAMI. **“Practical Black-Box Attacks Against Machine Learning”**. In: *2017 ACM on Asia Conference on Computer and Communications Security*. ACM. 2017 (cit. on pp. 11, 14).

[Qui86] J. R. QUINLAN. **“Induction of Decision Trees”**. In: *Machine learning* (1986) (cit. on p. 11).

[RN10] S. RUSSELL, P. NORVIG. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010 (cit. on p. 11).

[RNH+09] B. I. RUBINSTEIN, B. NELSON, L. HUANG, A. D. JOSEPH, S.-h. LAU, S. RAO, N. TAFT, J. TYGAR. **“Antidote: Understanding and Defending against Poisoning of Anomaly Detectors”**. In: *9th ACM SIGCOMM Conference on Internet Measurement*. ACM. 2009 (cit. on p. 5).

[RPV+14] Y. RAHULAMATHAVAN, R. C.-W. PHAN, S. VELURU, K. CUMANAN, M. RAJARAJAN. **“Privacy-Preserving Multi-Class Support Vector Machine for Outsourcing the Data Classification in Cloud”**. In: *IEEE Transactions on Dependable and Secure Computing* (2014) (cit. on p. 1).

[RRK+90] D. W. RUCK, S. K. ROGERS, M. KABRISKY, M. E. OXLEY, B. W. SUTER. **“The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function”**. In: *IEEE Transactions on Neural Networks* (1990) (cit. on p. 11).

[Sch15] J. SCHMIDHUBER. **“Deep Learning in Neural Networks: An Overview”**. In: *Neural networks* (2015) (cit. on p. 11).

[SS04] A. J. SMOLA, B. SCHÖLKOPF. **“A Tutorial on Support Vector Regression”**. In: *Statistics and computing* (2004) (cit. on pp. 4, 9).

[SSG17] Y. SHI, Y. SAGDUYU, A. GRUSHIN. **“How to Steal a Machine Learning Classifier with Deep Learning”**. In: *Technologies for Homeland Security (HST), 2017 IEEE International Symposium on*. IEEE. 2017 (cit. on pp. 3, 11, 14).

[SV99] J. A. SUYKENS, J. VANDEWALLE. **“Least Squares Support Vector Machine Classifiers”**. In: *Neural Processing Letters* (1999) (cit. on p. 8).

[TZJ+16] F. TRAMÈR, F. ZHANG, A. JUELS, M. K. REITER, T. RISTENPART. **“Stealing Machine Learning Models via Prediction APIs”**. In: *USENIX Security Symposium*. 2016 (cit. on pp. 1, 11, 13, 15, 25, 32, 34).

[Val84] L. G. VALIANT. **“A Theory of the Learnable”**. In: *Communications of the ACM* (1984) (cit. on p. 28).

[VLC94] V. Vapnik, E. Levin, Y. L. Cun. **"Measuring the VC-dimension of a Learning Machine"**. In: *Neural Computation* (1994) (cit. on p. 29).

[WG18] B. Wang, N. Z. Gong. **"Stealing Hyperparameters in Machine Learning"**. In: *arXiv preprint arXiv:1802.05351* (2018) (cit. on pp. 3, 11).

[WZ05] Q. Wu, D.-X. Zhou. **"SVM Soft Margin Classifiers: Linear Programming versus Quadratic Programming"**. In: *Neural Computation* (2005) (cit. on p. 8).

[Yao86] A. C.-C. Yao. **"How to Generate and Exchange Secrets"**. In: *Foundations of Computer Science (FOCS'86)*. IEEE, 1986 (cit. on p. 10).

[YJV06] H. Yu, X. Jiang, J. Vaidya. **"Privacy-Preserving SVM Using Nonlinear Kernels on Horizontally Partitioned Data"**. In: *2006 ACM Symposium on Applied Computing*. ACM. 2006 (cit. on p. 1).

[ZCZL16] T. Zhang, S. S. Chow, Z. Zhou, M. Li. **"Privacy-Preserving Wi-Fi Fingerprinting Indoor Localization"**. In: *International Workshop on Security*. Springer. 2016 (cit. on pp. 1 sq., 6, 31 sqq.).