The increase of available data and computational power have lead to a more widespread usage of machine learning (ML) techniques. Yet, ML algorithms can easily become the target of an attack themselves. The most famous example are evasion attacks, also known as adversarial examples. A small perturbation is added to a test sample, which is subsequently misclassified. Examples for targeted systems include, but are not limited to Malware detectors which then classify Malware as benign [11, 41], vision for autonomous driving that mis-classifies traffic signs [40], and robot visual systems [26]. Most defenses for such evasion attacks are shown to lead to an arms race [3, 7].

Other attackers aim to steal or copy the ML model [31, 44] or hyper-parameters set for training [30], thereby harming the intellectual property of the owner of the model. Such attacks are referred to as model stealing or model extraction attacks. Another, yet different attacker retrieves the data that was used to train the classifier [19, 37], leading to either a privacy breach for the subjects contributing to the data and/or the person or company that collected the data. Such attacks are known as membership inference attacks.

The vulnerability towards these attacks can formally and empirically be studied using Gaussian Process (GP) algorithms. GP are nonlinear, as are deep neural networks, yet provide the means for a rigorous analysis: After training, a GP yields a closed form expression, where classification depends directly on both parameters learned and the data used during training. Hence, formal reasoning is more fruitful than for models like deep neural networks which depend largely on randomness. We use this property to analyze evasion attacks, model stealing, and membership inference.

At the same time, the curvature of the decision function of a GP can be predetermined before training: choosing a long lengthscale yields a GP with a flat decision surface, whereas a short lengthscale leads to a more curvy surface. This gives us the unique opportunity to study vulnerability towards evasion attacks in context of nonlinear model curvature. Such a relationship is known for linear models like support vector machines [36] and used for mitigations in deep neural networks [18, 33]. We conduct a broad empirical study that investigates decision surface curvature and vulnerability towards evasion attacks on GP.

Moreover, a steep curvature of the classification function is more likely to fit randomness in the data, and thus overfit. Membership inference attacks have been found to depend on overfitting [19]. Such overfitting further often implies high confidence on seen training data. GPs, instead, are not forced to be overly confident on the training data, and overfitting can directly controlled by setting the lengthscale. This yields yet again an ideal setting to study membership inference attacks on GP. Intriguingly, we further find that membership inference and model stealing are closely related on GP, and hence extend our study to model stealing attacks.

More in detail, our contributions are as follows:

- We prove using GP that learning and static security guarantees, such as a fixed, secure radius around each training point, are opposed. Our analysis also emphasizes the need for a rejection option in context of robustness. We further show how many queries an attacker needs to analytically recompute GP’s training data and parameters.
- To complement our formal work, we conduct a broad empirical study on four security and two computer vision data sets, this time focusing on curvature of the GP. In evasion attacks, decision function curvature only changes the kind of attack that is needed to succeed: Highly optimized attacks tend to fool a steep curvature, whereas less optimized attacks are more effective on flat curvature classifiers. In contrast, we show that the training data can not be empirically determined if the GP is properly configured. Then, however, the kernels parameters are easier to estimate.
- We conclude that attacks on ML algorithms should not be studied in isolation, but in relationship to each other: mitigating one attack might enable another, different attack vector.

ABSTRACT
The widespread usage of Machine Learning (ML) leads to direct security threats, as ML algorithms are vulnerable to a plethora of attacks themselves. Different attack vectors are known, and target for example the training phase using manipulated data. Alternatively, they take place at test time and aim for misclassification, the leakage of the training data or extraction of the model. Previous works studied different test time attacks individually. We show that using an ML model enabling formal analysis and allowing control over the decision surface curvature, interesting insights can be gained when attack vectors are not studied in isolation but in relation to each other. We show for example how we can secure Gaussian Process Classification against empirical membership inference by properly configuring the algorithm. In this configuration, however, the model’s parameters are leaked. This allows an analytic computation of the the training data, which is thus leaked, against the original intention of protecting the data. We extend our study to evasion attacks, and find that analogously, hardening the model against one attack boils down to enabling a different attacker.

1 INTRODUCTION
The increase of available data and computational power have lead to a more widespread usage of machine learning (ML) techniques. Yet, ML algorithms are vulnerable to a plethora of attacks themselves. Different attack vectors are known, and target for example the training phase using manipulated data. Alternatively, they take place at test time and aim for misclassification, the leakage of the training data or extraction of the model. Previous works studied different test time attacks individually. We show that using an ML model enabling formal analysis and allowing control over the decision surface curvature, interesting insights can be gained when attack vectors are not studied in isolation but in relation to each other. We show for example how we can secure Gaussian Process Classification against empirical membership inference by properly configuring the algorithm. In this configuration, however, the model’s parameters are leaked. This allows an analytic computation of the the training data, which is thus leaked, against the original intention of protecting the data. We extend our study to evasion attacks, and find that analogously, hardening the model against one attack boils down to enabling a different attacker.

We prove using GP that learning and static security guarantees, such as a fixed, secure radius around each training point, are opposed. Our analysis also emphasizes the need for a rejection option in context of robustness. We further show how many queries an attacker needs to analytically recompute GP’s training data and parameters.

To complement our formal work, we conduct a broad empirical study on four security and two computer vision data sets, this time focusing on curvature of the GP. In evasion attacks, decision function curvature only changes the kind of attack that is needed to succeed: Highly optimized attacks tend to fool a steep curvature, whereas less optimized attacks are more effective on flat curvature classifiers. In contrast, we show that the training data can not be empirically determined if the GP is properly configured. Then, however, the kernels parameters are easier to estimate.

We conclude that attacks on ML algorithms should not be studied in isolation, but in relationship to each other: mitigating one attack might enable another, different attack vector.

Kathrin Grosse
CISPA Helmholtz Center for Information Security,
Saarland Informatics Campus
kathrin.grosse@cispa.saarland

Michael T. Smith
Department of Computer Science,
University of Sheffield

Michael Backes
CISPA Helmholtz Center for Information Security,
Saarland Informatics Campus
Table 1: Notation used throughout the paper.

| Notation | Meaning |
|----------|---------|
| $x$      | data point, equivalent notation: $X_i$ |
| $x_i$    | feature $i$ in data point $x$ |
| $y$      | label, equivalent notation: $Y_i$ |
| $X_{tr}$, $Y_{tr}$ | training data |
| $X_t$, $Y_t$ | test data, one individual test point also $x'$. |
| $k(x_i, x_j)$ | covariance function, $K$ resulting matrix |
| $\rho$ | a particular value of $k$ (threshold) |
| $\kappa$ | some value of $k$ |
| $B(x, \rho)$ | ball around $x$ with radius $\rho$, also $\rho$-ball |
| $y^*$ | GP’s prediction, predictive mean |
| $\epsilon$ | parameter to globally perturb data point |
| $l$ | lengthsacle of GP’s covariance (RBF) |

2 BACKGROUND

In this section we present all necessary background required for this paper. We start by introducing classification and Gaussian Processes (GP), then give a short summary of adversarial learning and finally summarize our threat model. We list all notations used throughout this paper in table 1.

In classification, the task is to assign labels $Y$ to some data samples $X$. In general, we separate the data into training ($X_{tr}$, $Y_{tr}$) and test data ($X_t$, $Y_t$). We then adapt the parameters or weights $\theta$ of a classifier function $f$ on the training data, e.g. $f(X_{tr}, \theta) = Y_{tr}$. Our goal is that $f(X_t, \theta) = Y_t$, or that the classifier predicts correctly on unknown test data.

2.1 Gaussian Process Classification

We use Gaussian Process Classification (GPC) [34] for two classes using the Laplace approximation. The goal is to predict the labels $Y_t$ for the test data points $X_t$ accurately. In GP, we use a covariance function, which is equivalent to a kernel or similarity metric. We will use those names interchangeably in this paper.

We first introduce Gaussian Process regression (GPR), and assume that the data is produced by a GP and can be represented using a covariance function $k$:

$$
\begin{bmatrix}
Y_{tr} \\
Y_{t}
\end{bmatrix} = \mathcal{N} \left( \begin{bmatrix}
0 \\
K_{tr}
\end{bmatrix}, K_{tr} \right),
$$

(1)

where $K_{tr}$ is the covariance of the training data, $K_{t}$ of the test data, and $K_{tt}$ between test and training data. Having represented the data, we now review how to use this representation for predictions. As we use a Gaussian model, our predictions are Gaussian too, with a predictive mean and a predictive variance which we define now. At a given test point $x'$, assuming a Gaussian likelihood function, the predictive mean $y^*$ is

$$
y^*_t = K_{x's}^{T}K_{tr}^{-1}Y_{tr},
$$

(2)

where $K_{x's}$ is the vector with the distances from $x'$ to each training point.

For brevity, we do not detail the procedure for optimizing the parameters of the covariance function $k$ that defines our Gaussian process. Instead, we outline how to alter this regression model to perform classification. Since our labels $Y_t$ are not real valued and non-Gaussian class labels,

we apply a link function $\sigma(\cdot)$ that normalizes the output to be in range $[0, 1]$. This procedure is called Laplace approximation.

In other words, GP does not learn any explicit weights $\theta$. Instead, we adapt the covariance metric $k$ to fit the training data. For a given test point, $k$ weights all training points with their labels, resulting in the output for this test point.

In the formal analysis, we focus on regression: it is similar to the latent representation that is used for the prediction in Laplace approximated classification. More concretely, when the mean of the regression tends towards one class, GPC’s classification will do so as well. As we are not interested in the strength of a response in our analysis, we thus skip the additional steps introduced by the link function $\sigma(\cdot)$. Finally, whenever we write GP, we refer to properties that both GPC and GPR share.

2.2.1 Covariance Function. We introduce the most common similarity metric in GP, the RBF kernel. This metric is defined as follows

$$
k(x, x') = \exp \left(-\frac{|x - x'|^2}{2\sigma^2} \right),
$$

(3)

where the $L_2$ distance between two points is re-scaled by length-scale $l$ and variance $\sigma^2$. These two parameters, $\sigma^2$ and $l$, form the parameters $\theta$ which are adapted during training.

In particular the lengthscale $l$ affects how local the resulting similarity metric is: a small $l$ yields for example a very local classifier with high decision function curvature. To obtain such a local classifier, $l$ is set before the training is started, and is then not adapted during optimization.

More in detail, we can influence how fast the kernel decays by fixing the lengthscale $l$. As we use the exponential function, the output similarity approaches $0$ as the (by $l$ re-scaled) distance gets larger. This property is called abation, and useful for outlier detection or open set tasks [38]. The faster the similarity abates, the more local the classifier and the steeper the decision function.

2.2 Adversarial Machine Learning

We now show how our previously described GP can be targeted by an adversary. First, we describe evasion in general and then introduce all attacks used in our evaluation. We then briefly comment on other attacks which are relevant for this paper.

Given a trained classifier $f(\cdot, \theta)$, evasion attacks compute a small perturbation $\delta$ for a sample $x$ such that

$$
\delta : f(x, \theta) \neq f(x + \delta, \theta).
$$

(4)

In a nutshell, given some metric, we aim to find a perturbation $\delta$ that changes classification. There are two basic constraints here, one is to minimize $\delta$ to be as small as possible. We might, however, also maximize the difference in classification, to be sure the resulting example is misclassified when targeting a second classifier $f'$. In general, we distinguish targeted and untargeted attacks. Since we only evaluate on binary tasks, however, this distinction is superfluous in this paper.
Before we detail on the attacks used in this work, we want to address how to measure \( \delta \). We might use the \( L_0 \) metric which counts the number of changed features. It is well suited for binary data, such as Malware features. The \( L_2 \) metric is equivalent to the euclidean or squared-root distance, and thus well suited for images. Another metric for images is the \( L_\infty \) metric that measures the largest change introduced.

Many algorithms exist for creating adversarial examples. We briefly recap the the algorithms that we rely on in our evaluation. We first review the fast gradient sign method (FGSM) [14]. This method is formalized as

\[
\delta = \epsilon \times \text{sign}(\nabla_x J_c(F(x, \theta)))
\]

where \( \epsilon \) parametrizes the strength of the perturbation. Further, the gradient of the model’s loss warranted the input \( x' \) is written as \( \nabla_x J_c((F(x, \theta))) \). FGSM implicitly minimizes the \( L_\infty \) norm, as the same change is applied to all features. FGSM has been extended to \( L_0 \) norm respectively. \( L_\infty \) norm is non-differentiable, an iterative attack is proposed where the \( L_2 \) attacker is used to determine which features are changed. Analogously, the \( L_\infty \) norm is poorly differentiable and hard to optimize. The authors propose here to use an iterative \( L_2 \) attacker with a penalty taking into account the \( L_\infty \) norm. Many defenses have been introduced to alleviate the threat of adversarial examples, yet these mitigations are often insufficient [3, 7].

We now address model stealing [31, 44], model extraction [30] and then membership inference [19, 37]. The first two attacks query the model with particular queries to deduce the desired information. In our work, however, we will show that GP enable an analytic computation of the targeted parameters and/or data. Membership inference attacks further exploit differences in confidence of the model for known and unknown data. In GP, we will use both confidence (predictive mean) and the predictive variance to deduce this information. This is a slight variation of the original attacks.

For completeness, we add attacks which alter training data maliciously to change a classifier in the attackers favor [5, 25]. In this area as well, defenses have been proposed (for example [9]).

2.3 Threat Model

To conclude the background, we specify the adversary that we consider in our study. We consider three different attackers. All operate at test time, when training is completed. In general, the attacker might have full knowledge of the model (white-box), where she can access each parameter of the trained algorithm. The gray-box setting considers information about the algorithm applied, where the parameters or weights are unknown. In the black-box model, the attacker has no knowledge about the targeted model, not even which algorithm is used.

Analogously, we have to take into account the knowledge the attacker has about the data. The training data might be fully known. If access is restricted, the attacker might have learned the kind and/or the number of features. In a more difficult setting, nothing except the task is known. In the following, we summarize the attacker by setting.

Evasion. The formal analysis shows a general contradiction between learning and security against evasion attacks, which is independent from a specific attacker. Given further the results from [17] and [12] for the vulnerability in white box settings, we investigate the influence of decision function curvature towards a black-box attacker concerning the model with full knowledge concerning the data.

Model Stealing, Model extraction, and Membership Inference. In a GP, both attacks can be analytically formulated, allowing to determine the exact number of queries needed given differing knowledge of the attacker. In the empirical study, we investigate whether we can speed up the attacks when we leverage learning and partial knowledge about the data. Concerning the model, we assume a gray-box attacker who is only aware that a GP is applied. In case of model stealing, we further consider the training data to be known or not known (e.g., gray-box or black-box). For membership inference, we take a different approach: we assume a very strong attacker (in an effort to study a worst case scenario) who has some labeled training data from the model. We use this knowledge to study which factors affect the success of the attacker.

3 FORMAL ANALYSIS OF VULNERABILITY

In this section, we analyze formally the vulnerability towards different attacks using GPR. We start with evasion attacks and then continue with model stealing and membership inference.

3.1 Evasion Attacks

To start our formal analysis, we define a classifier that cannot be fooled by an adversarial example or an evasion attack. However, we first recap the goal of the attacker.

Evasion Attack. Given a classifier \( f \) and an instance \( x \) of a class \( c \), find a perturbation \( \delta \) such that \( f(\delta + x) \neq f(x) \). A labeling oracle \( o \) however still assigns \( o(x + \delta) = c \).

We will show in the following that a static security guarantee fulfilling this definition is opposed to learning. To proceed, we assume correctly labeled data, and leave the study of training time attacks for future work. We further briefly define rejection of a classifier which we need throughout this analysis. We assume a classifier can reject a sample, in the sense that it does not assign the given sample to any predefined class.
To define the secure classifier, we chose an abating similarity measure: as the distance from the training data increases, the measure approaches 0. There is a \( \rho \) such that for all training points, iff point \( x' \) is in the closed ball \( B(x, \rho) \) around a training point \( x \in x_{tr} \) with radius \( \rho \), then \( x' \) cannot be an example of another class than \( x \). In other words, all points in the \( \rho \)-ball around \( x \) are of the same class. We formalize secure classification as

\[
f(x') = \begin{cases} 
    y_i & \text{if } x' \in B(x_i, \rho) \\
    \text{reject} & \text{otherwise}
\end{cases}
\]

and obtain a secure classifier which cannot be fooled: Changing a sample enough to be classified as a different class means to alter \( x' \) so much that \( x' \in B(x_j, \rho) \) where \( y_i \neq y_j \), or according to our definition, \( o(x_j) \neq o(x_j) \). Then, by our definition, \( x' \) is a valid instance of this class and not an adversarial example.

The constant classifier, which can also not be fooled by evasion attacks, is then a special instance of the classifier described here: The number of points is one and \( \rho = \infty \). Further, this secure classifier is equivalent to a 1-nearest-neighbor classifier using a threshold \( \rho \). Finally, this secure classifier is also equivalent to a GP given the following conditions:

1. GP has a rejection option based on \( \rho \).
2. There is no point \( x' \) such that for two distinct \( x_i, x_j \in x_{tr} \), both \( k(x_i, x_i) > 0 \) and \( k(x_j, x_j) > 0 \).

In other words, we require that GP is able to reject a sample and assumption two states that the similarity between any two training points is zero, independent of their class. Given the last assumption holds, however, the resulting covariance matrix is the identity matrix, as the similarity between any two points is zero. This covariance matrix does not allow any learning [27].

We implicitly assume that these assumptions hold as well for the secure classifier. We may enforce them by deleting training data, for example. We also implicitly assume that \( \rho \) is the same for all training points. In case of different \( \rho \)s, we apply the smallest. Further, in the infinite sample limit, the secure classifier might cover the whole generating distribution of the data. Our analysis however concerns a set of finite samples—for the security in the infinite sample limit, refer to [45].

We start by showing that given the previous two assumptions, GP is indeed equivalent to this secure classifier.

**GP as a Secure Classifier** An example for GPC as a secure classifier is visualized in fig. 1a. The predictive mean of GPR is composed of weighted labels, where the weight is the similarity between the training points and the distance to the queried test point, as formalized in eq. (2).

Throughout this analysis, we assume the values representing the two classes to be 1 for class 1 and -1 for class 0 respectively.

When assumption two holds, the kernel matrix is the identity matrix: the similarity between any two points is zero, and the similarity of each point with itself is 1.0. A test point \( x' \) is at most close to a single training point, here \( x_n \). We summarize:

\[
y' = k(x', x_n) * 1 + y_n ,
\]

where 1 denotes the similarity from \( x_n \) to itself; all other terms are zero as \( k(x', x_i) = 0 \) iff \( i \neq n \). This classifier is almost equivalent to section 3.1, as it classifies each point if it is in the \( \rho \)-ball of any training point.

To fulfill the equivalence, we define rejection for GPR (assumption 1). We need to reject any test point for which \( k(x', x_n) \leq \rho \). The minimum and maximum output of GPR is (due to the labels) either \( 1 = y_i \) or \( -1 = y_0 \), thus we do not classify a sample iff

\[
y' \in [-1 + \tau_0, 1 - \tau_1] ,
\]

where we chose \( \tau_0 = \tau_1 = \rho \) to reject points further away than \( \rho \). We may, however, define a different thresholds (where \( 0.0 < \tau < 1.0 \)) for the two classes or even each test point, if required.

![Figure 1: Intuition of formal Analysis of GP. Bright and Dark areas are classes, other areas (red) rejection areas. Black and white points are training data. The axis are the two features.](image)

**Generalization and Secure Classifier** Let us assume the second assumption does not hold. As the similarity between some points is not 0 anymore, they will jointly influence classification as visualized in fig. 1 and thus generalize.

**Theorem 1.** Learning in GPR involves classifying areas outside the \( \rho \)-balls (generalization), or there is no improvement over GPR with rejection and an identity matrix covariance, hence no learning takes place.
Proof. To be classified, we need a classification output \( p > \rho \) or \( p < -\rho \). We start with the first case. We thus obtain prediction

\[
p \leq \sum_{i} (\rho - \kappa_i) \ast K_i^{-1} \ast 1 ,
\]

where \( K_i^{-1} \) is the sum over the inverted covariance matrix column corresponding to point \( i \). Before inversion, this column contains the similarities between \( i \) and all other training points. So far, we have ignored that we need a test point to obtain this prediction. Without loss of generality, we pick \( x' \) which maximizes the above sum under the restriction that \( x' \) is none of the \( \rho \)-balls: hence \( \rho - \kappa_i \), the distance to \( \rho \)-balls is \( \kappa_i \).

There are three cases. In the first, \( p \geq \rho \) and we classify outside the \( \rho \)-ball. In the second case, \( p = 0 \) or \( 0 < p < \rho \). As we reasoned about the maximal \( x' \), we know that there are no other points for which \( p > \rho \). Then GPR is still secure: no area outside the \( \rho \)-ball is classified, as the output is below the defined threshold. It remains to be shown, however, that there is no contradiction for the opposite class. We proceed analogously with an \( x' \) that is chosen to minimize the sum.

Remark. We used in the proof that the minimal output of a point chosen to maximize the sum is zero. Analogously, the maximal value when minimizing the sum is zero as well. This holds due to the abating property of the kernel: As we move away from the data, eventually all distances become zero, thus the sum is zero as well.

We conclude that generalization leads to classification in areas which might enable test time attacks such as adversarial examples. More intriguingly, such a secure classifier is highly vulnerable to poisoning attacks: injection of a single point will guarantee classification of points in its \( \rho \)-ball.

3.2 Model Stealing, Model extraction, and Membership Inference

We now analyze GP’s vulnerability to model stealing and membership inference. We first define these two goals of the attacker before we carry out the formal analysis.

Model stealing. Given a trained GP with black box access (only access to input and output of the model), the attacker aims to find out which lengthscales was learned during or determined before training and which training data was used.

Model extraction. Given a trained GP with black box access (only access to input and output of the model), the attacker aims to find out which lengthscales was learned during or determined before training.

Membership inference. Given a trained GP with black box access (only access to input and output of the model), the attacker aims to learn whether one or several samples where used to train the GP.

We will now refresh how a classification is computed in a GP (introduced in eq. (2)), namely

\[
y_* = K_x^T K^{-1}_x Y_x = \sum_{i} k(x' - X_i) \ast K_i^{-1} \ast Y_i ,
\]

where we iterate over the \( n \) training data points. Consider that \( k \), as depicted in eq. (3), is further parametrized using \( l \) and \( \sigma^2 \). When we unfold this sum and add the observed output of a GP, we obtain an equation system we can solve to carry out the previously defined attacks.

To run a model extraction attack given the training data, we plug in this data, observe the output of the GP and compute the resulting linear equation system yielding the lengthscales values. We then need either 2 points or \( d + 1 \), depending on whether the GP has one adjustable lengthscales or \( d \).

Analogously, to run a membership inference attack given the lengthscales, we plug the lengthscales in this equation set, observe the output of the GP and compute the resulting linear equation system yielding the training data. We then need either \( n + 1 \) points or \( nd + 1 \), depending on whether the GP has one adjustable lengthscales or \( d \).

Finally, we consider to running a model stealing attack on the GP. Then, however, the resulting equation system is not linear anymore, and all we know is that we need certainly more than \( nd + 1 \) or \( n2d + 1 \) points. We summarize our findings in fig. 2.

Remark. there are some additional factors that might influence the complexity of the computation. The values we determined should therefore be perceived as lower bounds.

Kernel used. We assumed implicitly that we know which kernel is used. In case this is not known, we might still set up one equation system per kernel and then observe which one is solvable.

Additional parameters. The RBF kernel is also parametrized by variance \( \sigma \) (as depicted in eq. (3)), which can be assumed to be a global constant, however. Depending on the kernel, all resulting equation system might be nonlinear and the resulting complexity higher.

Number of samples. We also assume to know the number of used samples. The solution to this is however straightforward: We may increase the number of assigned training points, depending on whether the solution was unique or not.
Unknown data. A recomputed training data set is only of use to an attacker if the features are known (for example in vision). On some Malware data set, an additional step is needed to determine how each feature relates to which property of the actual Malware sample.

We conclude that model stealing and membership inference attacks are analytically computable on a GP. Given these formal results, we now investigate vulnerability empirically.

4 EMPIRICAL STUDY OF VULNERABILITY

We have shown in our formal analysis the principal vulnerability of GP when targeted by either evasion, model stealing, model extraction, or membership inference. We have seen that for the latter three attacks, when the attacker has no knowledge, the resulting equation system was non-linear. As we will see now, the attacker can easily determine some of this knowledge empirically, thereby obtaining a much easier problem. Key ingredient to this analysis is the impact the curvature of the decision function to vulnerability. This curvature or the gradient of the RBF kernel (defined in eq. (3)) can be influenced by choosing a lengthscale before training. Hence, GP thus offers us an ideal setting to study the gradient of the classification function wrt the input in relation to vulnerability. The usefulness of this setting extends to evasion attacks, where curvature in other models than GP is often influenced by regularization or mitigations. GP allows us to study the effect of curvature of the decision function on evasion in a direct, controlled manner.

We start with describing the models used in our evaluation. Once we have described the setting, we focus on evasion attacks and then start with evasion attacks, study model stealing and model extraction and conclude the section investigating membership inference.

4.1 Experimental Setting

We describe the general setting all empirical studies share, and describe details jointly with the corresponding study. The code to reproduce all the experiments is accessible at URL blinded for submission.

4.1.1 Data. We evaluate the classifier on a range of tasks, focusing on security settings such as Malware [2, 42] and Spam detection [23]. Additionally, we use a loosely security related task: fake banknote detection by [23]. We further consider the MNIST benchmark data set [21] and the SVHN data set [29]. Finally, we employ the two moons data set in two dimensions for visualization purposes.

4.1.2 Implementation. We implement our experiments in Python using GPy for the Gaussian Process approaches [16]. To obtain adversarial examples, we use Tensorflow [1] and the Cleverhans library 1.0.0 [15] for DNN, and public implementations for all other attacks [8, 17].

4.1.3 Setting. We train our GPC using RBF kernel with a predefined lengthscale. This GPC is optimized until convergence or for 100 iterations. For each task, we chose two lengthcales, both achieving similar accuracy. We provide more details on how we determined the two lengthcales used in our experiments in the Appendix, and depict our choices with the resulting accuracies in table 2.

4.2 Evasion Attacks

Given the gradient of the classification function, we expect that a classifier (here GPC) with a long lengthscale miss-classifies fewer adversarial examples: A larger perturbation $\delta$ is needed to cause the same change in the output as compared to a steep curvature. To test our hypothesis, we craft a range of transferred adversarial examples from other methods. We use FGSM, JSMA and Carlini and Wagner’s attacks for DNN, the linear SVM attack and GPGFS and GPJM on a GPC substitute. Our intention is to study a range of attacks, including optimized, unoptimized, one-step and iterative attacks as well as different metrics ($L_0$, $L_2$, and $L_\infty$). We summarize all attacks based on the Jacobian in IBM, and plot all attacks by Carlini and Wagner according to the norm optimized ($L_2$; for example $L_2$ for optimization of the $L_2$-norm).

4.2.1 Setting. We compare the previously chosen lengthscales by comparing their accuracies in absolute percent: a value of 5 indicates that a classified 95% and $b$ 90% of adversarial examples correctly, leading to a difference of 5 percent. In general, a value above zero in the plot signifies that the shorter lengthscale classified more data correctly (e.g., recovered the correct class before changing the original, benign sample). Analogously, below zero, a longer lengthscale (flat curvature) performed better.

4.2.2 Results. We plot the results of our experiments on all datasets in fig. 3a. We observe that the correct classification is in general higher when the lengthscale is shorter. In particular, for attacks with global perturbation ($\epsilon > 0.01$), a shorter lengthscale performs better. For highly optimized attacks (for example $L_2$), a longer lengthscale is of advantage.

We additionally allow for a rejection option, as we previously forced GPC to output a classification. Instead, we now reject a sample if $y^* = 0$: in this case, the test point is far away from the training data. A short lengthscale might thus allow to reject more abnormal data. We plot our results in fig. 3b. The results are similar to the experiment without rejection option. We observe, however, that now many cases where the long lengthscale was beneficial disappeared: the correct classification was due to the default class being, by chance, in favor of the correct classification. We thus investigate rejection in general in fig. 4. Adding a rejection option when using a long length-scale has no effect at all. For a short

| Data set  | $n$ | $l$ | $Acc_r$ | $Acc$ | $l$ | $Acc_r$ | $Acc$ |
|-----------|-----|-----|---------|-------|-----|---------|-------|
| Hidost    | 500 | .5  | 98.4    | 98.4  | 1.9 | 97.7    | 99.6  |
| Drebin    | 750 | .5  | 54.4    | 94    | 1.9 | 94.8    | 94.8  |
| Spam      | 500 | .3  | 92.6    | 91.7  | 5   | 92.7    | 90.2  |
| Bank      | 500 | .3  | 100     | 100   | 2   | 100     | 100   |
| MNIST91   | 500 | 1   | 98.9    | 98.3  | 8   | 99.5    | 99.5  |
| MNIST38   | 500 | 1   | 93.4    | 93.4  | 8   | 97.4    | 97.1  |
| SVHN91    | 1500| 8   | 85.4    | 88.5  | 16  | 83.8    | 87.6  |
| SVHN10    | 1500| 8   | 88.7    | 88.7  | 16  | 88.7    | 88.7  |
lengthscale, the effect is positive or neutral, and only negative two cases. These two cases stem from the Hidost data set, which is highly imbalanced: We observe that by chance the assignment of the forced classification was in favor of the larger class.

To understand the effect of the lengthscale in more detail, we investigate a toy example. We depict GPC with two different lengthscales on the two moons data set in fig. 5. Red in the picture denotes the areas of rejection. We observe that using a long lengthscale, the density mass is pushed away from the training data, and the classifier is confident in areas where no training data was observed. We illustrate this by denoting a possible one-step attack in the picture overshooting the target class. Additionally, we add a possible multi-step attack which is misclassified as the short lengthscale yields a very unsteady decision boundary.

Our empirical results on the individual attacks confirm this intuition: highly optimized attacks such as Carlini and Wagner’s attack and some cases of other multi-step attacks based on the Jacobian are classified correctly more often with a long lengthscale (boundary areas), whereas for one-step attacks (ε) a shorter lengthscale is beneficial.

4.2.3 Conclusion. We conclude from our experiments that only classifiers with steep decision functions benefit from rejection. We further observe that only altering the classifier’s curvature does not alleviate vulnerability, but merely changes which attacks the classifier is vulnerable to.

4.3 Model extraction

In the theoretical analysis, we considered analytic computation of the lengthscale. In contrast, we now aim at approximating the lengthscale when we are only given the output of the GP on a set of points. We then study in a second experiment in how far the used kernel of the GP can be determined when similar black-box access is given to the attacker.

4.3.1 Setting l. We pick the same lengthscales as before and evaluate whether it is possible on a data set to determine the lengthscale of a victim GP by the attacker. In particular, we investigate the feasibility to run binary search to obtain l: We aim to know whether the distance between the labels shrinks as the lengthscale chosen by the attacker, l0, approaches the original lengthscale l. We evaluate three settings: Training GPC on the same data as the victim, a mix of the same and disjoint (50% same data) and disjoint data. In each setting, we train 50 GPCs, starting with a lengthscale l0 = l/2 and increasing the lengthscale in steps of (1/50)l. We then compute the absolute differences between the outputs of the two trained GPCs on hold out test data that was used by neither GPC in training.

4.3.2 Results l. We plot the results in fig. 6. In the case where we use the same training data as the victim, we observe that since the training of the GPC is deterministic, we can recover the lengthscale. As expected, the decrease for all data sets towards the original lengthscale (l) is monotonic, with small exceptions on the SVHN and Bank data sets when we use a high lengthscale. For a short lengthscale, all distances decrease monotonically towards the original l.

For a mixed data set (see middle plots of fig. 6), where 50% of the points are taken from the training set, the results are much less clear. We still observe a general decrease towards l, however it is less pronounced and does not hold for all data sets. For a long lengthscale, the distance is smallest roughly around l except for SVHN and Spam, where it remains constant. For Drebin, the distance seems shortest around l + (l/2). In case of a short lengthscale, the results vary: for some data sets (MNIST91, Bank) the distance is closest to l. For others (including SVHN and Malware), the smallest distance occurs at l/2.

In case of the disjoint data sets (bottom plots of fig. 6), the results are even less pronounced. We observe a slight decrease towards the original lengthscale, yet the average minimum is at a lengthscale that is longer than the original one. In case of a short lengthscale, there are no differences at all for the Malware data set. For the SVHN data sets, the minimum is very close to l/2. In general, we conclude from the plots that a lengthscale can be approximated using binary search.
to deduce which kernel was used. Yet, our results show that for neither short nor long lengthscales, we observe a clear trend that the distances to RBF, the original kernel, are smallest. The results are plotted in fig. 7, where we depict the mean over the three settings and the standard deviation as error bars.

In case of short lengthscales, the distance are often similar in all settings (Malware, Drebin, SVHN10). In the other cases (SVHN91, both MNIST, bank, spam), RBF exhibits the lowest distances, but the distance are close to the ones of other kernels. The only data set where the kernel is correctly determined is the bank data set.

In contrast to the short lengthscale, we now observe that the distances vary strongly between the settings. Yet, the shortest distances points often to the polynomial kernel (bank, MNIST91) or are similar across different kernels. An example are linear and RBF kernel for spam, or polynomial and linear kernel for SVHN91.

4.4 Membership Inference

We now investigate how well an attacker can empirically deduce which points were used in training. First, we study the general setting, and then investigate particular settings influencing the attackers success, such as overfitting, distribution drift, and sparse features.

In an effort to study a worst case scenario, the attacker has an oracle that labels a large fraction of the points used in training by the victim GPC as such. The attacker uses this data to train a fresh classifier that predicts for unseen data points whether they were used in training.

4.4.1 Setting. The victim GPCs are trained using the same lengthscales as before. We then build a dataset using the output (specified below) of the GPC and labels that indicate whether a data point was used in training or not. We split the resulting dataset randomly in training and a separate test set of 50 points which we compute the reported accuracy on. We further provide random guess accuracy as a baseline. The training data is used to train the GPC (fig. 8b). We further train on both variance and mean or the unnormalized, latent mean of GP and depict the results in fig. 8c.

4.4.2 Results. We run several experiments, each corresponding to different input the attacker uses to train on.

where we train random forests only on predictive mean or variance without (fig. 8a) or including the samples $X$ used to train the GPC (fig. 8b). We further train on both variance and mean or the unnormalized, latent mean of GP and depict the results in fig. 8c.

For the first experiment, we train random forests only on predictive mean or variance without the corresponding samples $X$. The results are depicted in (fig. 8a). Overall, using only the predictive
Figure 6: Normalized, absolute differences in output stacked for different data sets when a GP’s lengthscale approaches the length-scale of the victim ($l$ denotes the point of equivalence).
mean (dots) and a long lengthscale (larger markers in the plots), all data sets are not vulnerable, with the exception of the two Malware data sets.

In the second experiment, we provide the learner more information: it trains also on the data training samples $X$ of the GPC. These results are depicted in fig. 8b. We observe results similar to the first experiment, with the exception that the attacker is now successful in all cases on SVHN tasks. In case of the Malware and MNIST tasks, the accuracy decreases slightly.

In the third experiment, the attacker trains either on mean and variance (squares) or the unnormalized, latent mean (stars). We depict the results in fig. 8c. The attacker succeeds in both cases on all SVHN tasks or when using a small lengthscale, with the exception of non-vision tasks. The attacker is also successful on the Malware data sets with a long lengthscale.

To summarize, we observe that on the Bank and Spam data sets, the attack is never successful. Whereas in general a shorter lengthscale is more vulnerable, this trend is inverted on the Malware data sets: here, a short lengthscale benefits the defender. Before we focus

---

Figure 7: Unnormalized distances in output when training GP’s with different kernels (x-axis) on different data sets (colors). RBF is the kernel used by the victim.

Figure 8: Lengthscale and membership inference on GPC. Bigger symbols denote a long lengthscale of targeted GPC, small symbols a short lengthscale. Random Guess Accuracy is denoted by x.
on these cases, however, we investigate what enables the attacks on the SVHN data and why a short lengthscale is beneficial for the adversary.

4.4.3 Overfitting and Distribution Drift. We now investigate whether the success of the attack depends on overfitting or distribution drift. For overfitting, we compare training and test accuracies. In case of distribution drift, we measure the standard deviation over the distances between training and test data. Since the distance measure is learned and adapted in GP, we expect the test data to cause larger variance in these values if the data is distributed differently.

Except for the Bank, Spam and SVHN data sets, we observe that the training accuracy always reaches 100% and is higher than test accuracy. For the Bank data, test accuracy is also 100% and thus not lower than train accuracy. In case of the spam and SVHN data, train accuracy is below 100%. For all data sets except Bank, the difference between test and train accuracy is smaller for a longer lengthscale. We conclude that slight overfitting occurs at short lengthscales, and enables the membership inference attacks.

For all SVHN settings and MNIST8 with a small lengthscale, the variance between training and test data is two magnitudes larger than among either training or test data. We conclude that the attack was enabled as training and test data were different from the perspective of GPC. This might imply that the model is actually not expressive enough to model the data in detail.

4.4.4 Sparse Data. We have so far explained all successful attack, leaving open only the two Malware datasets, Hidost and Drebin, unexplained. Both data sets share the high dimensionality and sparsity, which might influence the success of the attack. To evaluate this claim, we train a GPC using inducing variables (GPy’s sparse GPC) to account for the sparse data. We repeat the attack as described in the previous experiments. We depict the results of the same settings as in the previous study in fig. 9. We observe that the accuracy is now on all settings close to a random guess, with the exception of a short lengthscale for Hidost on mean or variance, latent mean, or mean and variance. For Drebin, we observe a very small improvement over random guess when a short lengthscale is used and the attacker accesses mean and variance.

4.4.5 Conclusion. We observe that even assuming a very strong attacker, the attacker is not successful when the applied GPC and data are properly configured: there is no distribution drift, overfitting and sparse data is properly taken care of. This is somewhat expected: in contrast to other algorithms such as DNN, a GP is only punished during training when a prediction is wrong, in contrast to requiring (as in DNN) very high confidence on the training data.

5 RELATED WORK

We first review the relationship between GP and Deep learning in general, then adversarial learning in the Bayesian context. Afterwards, we turn to other formal methods in the context of adversarial learning. We then review regularization and mitigations in adversarial ML that are related to our empirical study of evasion attacks. To conclude, we review other works in Models Stealing and Membership inference.

Gaussian Processes and Deep Learning. Albeit superficially unconnected, the relationship between GP and deep neural networks is studied by Neal [28]. To gain more understanding, recent approaches by Matthews et al. [10] and Lee et al. [22] represent DNN with infinite layers as Gaussian Processes. Garriga-Alonso et al. [13] extend this finding to convolutional networks.

Bayesian Learning and Adversarial Learning. Other works such as Bradshaw et al. [6] and Rawat et al. [35] investigate adversarial learning in the context of the Bayesian framework. Yet, they do not focus on GP specifically. Grosse et al. [17] propose test time attacks on GP and show that Bayesian model uncertainty is not a reliable defense to evasion attacks. All of them, however, take an empirical approach and further only investigate evasion attacks.

Formal Approaches to Adversarial Machine Learning. Wang et al. [45] show the vulnerability for the k-nearest-neighbor classifier, where they define robustness given a ρ-ball, analogous to our approach. Yet, their approach works in the infinite sample limit, and they propose deletion of points as countermeasure. Fawzi et al. [12] show general vulnerability for all classifiers. Further, a radius classifier is also introduced in Tanay et al. [43], however in context of SVM: the authors focus on decision plane classification without rejection option. In contrast, we work on non-linear methods without decision planes, outline the importance of rejection, and additionally analyze model stealing and membership inference. We thereby outline relationships between robustness among different methods.

Regularization, Evasion Vulnerability, and Mitigations. We are not the first to analyze the curvature of the classifier function in context of test-time attacks. Russu et al. [36] show a direct relationship between the gradient of a linear model and its vulnerability. Also previous defenses for DNN, as for example introduced by Raghu et al. [33], can be linked to regularization. In contrast, we find evidence that in non-linear methods, regularization will change, not alleviate the problem of vulnerability to evasion attacks.
Another line of work in DNN has a strong relation to the secure classifier used in our theoretical analysis. Adversarial training, as proposed by Madry et al. [24] and Wong et al. [46] aim directly at training the network to output the same class in a ρ-ball around each training point. Yet these approaches do not consider a reject option, which has however been done empirically by Melis et al. [26]. Further, Bendale and Boult [4] investigate open set recognition. We connect both ideas, rejection and ρ-balls, in our work.

**Model Stealing, Model Extraction and Membership Inference.**

Copying the model without consent of the owner has been introduced by Papernot et al. [31] and Tramèr et al. [44]. Additionally, Oh et al. [30] where the first ones to propose particular attacks that allow to deduce hyper-parameter such as the usage of dropout in the training of a neural network. Further, Shokri et al. [39] and Salem et al. [37] deal with the extraction of the training data or properties thereof. Yet, these approaches mainly target deep neural network models. We support and confirm findings from these works, yet we show that a GP which is secure towards membership inference easily leaks the models parameters. We further show how distribution drift influences membership inference. Dealing with distribution drift is an open research question on it own [20].

### 6 CONCLUSION

In this paper, we investigated the security of GP at test time, including its vulnerability towards evasion, model stealing, model extraction, and membership inference attacks. We studied these settings both formally and empirically. Our formal analysis is two-fold: We first show that static security guarantees during test time, for example a fixed radius around training data points, are opposed to learning. Our finding emphasizes the necessity of rejection in secure learning. Additionally, we argue that an attacker can represent a GP as a system of linear equations, thereby recomputing parameters learned or the data used during training, given that she can query the model linearly in dimensionality of the data. Our analysis further outlines the close relation of model stealing, model extraction, and membership inference on GP.

We further leveraged the property of GP to fit a model with a predefined curvature, and conducted an empirical study on four security and two vision data sets. We summarize our results briefly in table 3, where we skip the fact that model stealing eases evasion. In the case of such evasion attacks, or adversarial examples, the curvature merely affects whether highly optimized or one-step attacks succeed. Concerning model stealing, we find that the lengthscale can be estimated empirically as well as determined analytically, where it is easier to estimate when the curvature is flat (or the lengthscale is long). Independent from the lengthscale, we conclude that it is not possible to empirically deduce the used kernel. For membership inference, we show that the success of the attack is influenced by measurable factors such as a steep gradient, overfitting or distribution drift. We conclude that GP can be protected against such an attack when properly configured (e.g. no overfitting or distribution drift, taking care of sparse features, choosing the lengthscale appropriately). Intriguingly, a short lengthscale leaks the data, whereas a long lengthscale leaks the parameters of the GP. Since either can be used to retrieve the other, the algorithm leaks both at a low number of queries.

Our findings emphasize that attack vectors on learning should not be seen in isolation, but also studied in relation to each other, as a mitigation towards one attack might enable or ease another attack.

### ACKNOWLEDGMENTS

This work was supported by the German Federal Ministry of Education and Research (BMBF) through funding for the Center for IT-Security, Privacy and Accountability (CISPA) (FKZ: 16KIS0753). This work has further been supported by the Engineering and Physical Research Council (EPSRC) Research Project EP/N014162/1.

### REFERENCES

[1] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kadivar, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaogang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Daniel Arp, Michael Spreitenbarth, Malte Hübner, Hugo Gascon, and Konrad Raack. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In Proceedings of the 2014 Network and Distributed System Security Symposium (NDSS), 2014.

[3] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. arXiv preprint arXiv:1802.00420, 2018.

[4] Abhijit Bendale and Terrance E. Boult. Towards open set deep networks. In 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, pages 1563–1572, 2016.

[5] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012, 2012.
Lengthscale and Accuracy

To determine the two lengthscales used throughout the paper, we trained a range of GPCs on each data set. We varied the lengthscale in small steps between 0.0 and 2.0 in steps of 0.01 and between 2.0 and 151.0 in steps of 1.0. We report the resulting accuracies and AUC with all rejection thresholds in Fig. 10. This illustrates the trade-off between model complexity and generalization on the training data. We report the resulting accuracies and AUC with all rejection thresholds in Fig. 10. This illustrates the trade-off between model complexity and generalization on the training data.

We varied the lengthscale in small steps between 0.0 and 2.0 in steps of 0.01 and between 2.0 and 151.0 in steps of 1.0. We report the resulting accuracies and AUC with all rejection thresholds in Fig. 10. This illustrates the trade-off between model complexity and generalization on the training data. We report the resulting accuracies and AUC with all rejection thresholds in Fig. 10. This illustrates the trade-off between model complexity and generalization on the training data.

We varied the lengthscale in small steps between 0.0 and 2.0 in steps of 0.01 and between 2.0 and 151.0 in steps of 1.0. We report the resulting accuracies and AUC with all rejection thresholds in Fig. 10. This illustrates the trade-off between model complexity and generalization on the training data. We report the resulting accuracies and AUC with all rejection thresholds in Fig. 10. This illustrates the trade-off between model complexity and generalization on the training data.
Figure 10: Accuracy and variations of lengthscales in GP. Accuracies are lines, dotted lines correspond to AUC (labels on the left side).